Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MASTERTHESIS

Estimating the influence of auxiliary training data for learning natural language processing tasks

vorgelegt von

Fynn Schröder

Universität Hamburg

MIN-Fakultät

Fachbereich Informatik

Studiengang: Informatik

Matrikelnummer: 7003971

Abgabedatum: 23.10.2019

Erstgutachter: Prof. Dr. Chris Biemann

Zweitgutachter: Benjamin Milde

Abstract

Today's computational natural language processing is largely based data-based machine learning approaches. Deep neural networks achieve state-of-the-art results and are thus commonly used. A downside is their need for large training corpora and the time- and resource-consuming training process. In case of insufficient amounts of training data, a solution is the usage of auxiliary training data. However, finding the most suitable auxiliary dataset requires an even more time-consuming trial-and-error approach.

Therefore, new methods are developed in this thesis to shorten the tedious search or make it entirely unnecessary. Based on the hypothesis that auxiliary data similar to the main training data results in a better performance, new methods are designed to automatically compute the similarity between any two sequence tagging datasets.

Compared to previous approaches, the new methods have two advantages. First, words and their labels are both taken into account when computing the similarity. This makes the methods more robust than previous approaches that only use either words or labels as their source of information. Second, the methods can compare raw datasets of arbitrary sequence tagging tasks without any preprocessing. Previous approaches working on labels required one of the datasets to be annotated in parallel with the labels from the other task. Thus, it was only possible to compare datasets of different tasks where at least one of the tasks had to be almost perfectly automatically taggable.

In multiple experiments, the hypothesis could be confirmed empirically. The designed similarity measurement methods conform to the intuition, which pairs of datasets should be similar to each other. In addition, they correlate with the test results obtained from neural networks that use the corresponding auxiliary datasets for multi-task learning. To a certain extent, it is possible to predict the multi-task learning results on the test sets based on the datasets' similarity and the neural network's single-task learning performance.

As part of this work, an efficient, open-source implementation[1] is provided, which can compute multiple similarity measures on sequence tagging datasets. In the course of the implementation, an effective solution was designed to identify all most similar pairs of vectors for a large amount of word vectors. The software allows automatically identifying similar datasets instead of performing the time-consuming training process for every possible auxiliary dataset to find the most suitable auxiliary training data. As a result, the time spent in a trial-and-error search for a suitable auxiliary dataset to train deep neural networks with multi-task learning can be reduced to a fraction.

[1]Source code available at `https://github.com/zoodyy/seq-tag-sim` and on the attached disk

Zusammenfassung

Die automatisierte Verarbeitung natürlicher Sprache findet heutzutage vor allem mit datenbasierten Ansätzen des maschinellen Lernens statt. Insbesondere künstliche neuronale Netzwerke werden bevorzugt eingesetzt, da sie aktuell die besten Ergebnisse erzielen. Diese benötigen jedoch viele Trainingsdaten und der Lernprozess ist zeit- und ressourcenintensiv. Bei unzureichend vielen Trainingsdaten sind passende Hilfsdatensätze eine Lösung. Den am besten geeigneten Hilfsdatensatz zu finden, erfordert jedoch zeitaufwändiges Ausprobieren.

Daher werden in dieser Arbeit Methoden entwickelt, um diese Suche abzukürzen oder gar überflüssig zu machen. Basierend auf der Hypothese, dass zu den primären Trainingsdaten ähnliche Datensätze ein besseres Ergebnis erzielen, werden Methoden zur Bestimmung der Ähnlichkeit zweier Sequenz-Tagging-Datensätze konzipiert.

Im Vergleich zu bisherigen Ansätzen haben diese zweierlei Vorteile. Einerseits werden sowohl Worte als auch deren Annotationen zur Berechnung der Ähnlichkeit einbezogen, wodurch die neu entwickelten Methoden robuster sind als bisherige Vorgehensweisen, die nur entweder Worte oder Annotationen berücksichtigen. Andererseits sind die Methoden zum Vergleich von unbearbeiteten Datensätzen beliebiger Aufgaben anwendbar, während bisherige Ansätze basierend auf Annotationen es erfordern, dass mindestens einer der Datensätze parallel mit den Annotationen aus der anderen Aufgabe versehen ist. So ist es bisher nur möglich gewesen, Datensätze verschiedener Aufgaben zu vergleichen, wobei eine dieser Aufgaben beinahe perfekt automatisch annotierbar sein musste.

Die Hypothese wird anhand mehrerer Experimente empirisch bestätigt. Die neu konzipierten Methoden zur Ähnlichkeitsmessung stimmen mit der Intuition überein, welche Datensätze sich ähnlicher sein sollten als andere, und sie korrelieren mit den Ergebnissen neuronaler Netzwerke, die mit den entsprechenden Hilfsdaten trainiert wurden. Bis zu einem gewissen Grad ist sogar eine Vorhersage der Ergebnisse auf den Testdatensätzen anhand der Ähnlichkeit der verwendeten Daten möglich.

Im Rahmen dieser Arbeit wird eine effiziente Softwareimplementierung[2] bereitgestellt, mit der die Ähnlichkeit von Sequenz-Tagging-Datensätzen anhand mehrerer Ähnlichkeitsmaße berechnet werden kann. Im Zuge der Implementierung wurde eine effektive Methode zum Finden der ähnlichsten Vekotrenpaare in einer großen Anzahl von Wortvektoren entwickelt. Mit dieser Software wird es ermöglicht, ähnliche Datensätze automatisiert und schnell zu identifizieren, anstatt zeitintensives Training für alle möglichen Hilfsdatensätze durchzuführen um geeignete zusätzliche Daten zu finden. Somit lässt sich der zeitliche Aufwand des Ausprobierens diverser Hilfsdatensätze für das Training komplexer neuronaler Netze bei Verwendung von Hilfsdaten auf einen Bruchteil reduzieren.

[2]Quellcode verfügbar unter `https://github.com/zoodyy/seq-tag-sim` und auf der CD

Danksagung

Ich möchte Chris herzlich dafür danken diese spannende Thesis schreiben zu können. Zu schätzen gelernt habe ich insbesondere die wunderbare Betreuung, die unglaubliche schnelle Beantwortung allerlei Fragen meinerseits, die Vorschläge für Verbesserungsmöglichkeiten, die vielen guten Ideen sowie die Motivation um die Arbeit auch abzuschließen.

Ich möchte auch Benjamin für seine hilfreichen Vorschläge bezüglich Inhalt und Strukturierung der Arbeit sowie Tipps zu den Experimenten samt deren Durchführung und Auswertung danken.

Meinen Freunden danke ich für ihre moralische Unterstützung und für ihr Verständnis während der anstrengenden Zeit des Schreibens.

Meinen Eltern Renate und Olaf möchte ich zutiefst dafür danken, dass sie immer an mich glauben und mich jedes Mal unterstützen, wenn nötig.

Meine umfassende Dankbarkeit und Liebe möchte ich Swaantje ausdrücken. Sie war immer für mich da, wenn ich sie brauchte. Außerdem hat sie mich immer motivert, wenn ich bei der Thesis zweifelte, mich erinnert auch Zeit für schöne Dinge neben der Thesis zu nehmen und mir so oft ein Lächeln auf die Lippen gezaubert.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Research question . . . 2
- 1.3 High-level overview . . . 2

2 Theoretical background **5**
- 2.1 Sequence tagging with statistical modeling methods . . . 5
  - 2.1.1 Hidden Markov Model (HMM) . . . 5
  - 2.1.2 Maximum Entropy Markov Models (MEMM) . . . 7
  - 2.1.3 Conditional Random Fields (CRF) . . . 8
- 2.2 Neural networks . . . 9
  - 2.2.1 Fundamentals . . . 9
  - 2.2.2 Feedforward neural networks . . . 11
  - 2.2.3 Recurrent neural networks . . . 14
  - 2.2.4 Regularization . . . 18
  - 2.2.5 Hyperparameter optimization . . . 20
  - 2.2.6 Word embeddings . . . 22
- 2.3 Multi-task learning . . . 23
  - 2.3.1 Theory and application in neural networks . . . 23
  - 2.3.2 MTL variants and differences to transfer learning . . . 25
- 2.4 Information theoretic clustering comparison measures . . . 25
  - 2.4.1 Entropy, joint entropy and conditional entropy . . . 26
  - 2.4.2 Mutual information . . . 26
  - 2.4.3 Comparing clusterings . . . 27
  - 2.4.4 Information theoretic similarity measures . . . 27

3 Related work **29**
- 3.1 Multi-task learning for sequence tagging . . . 29
- 3.2 Effect of auxiliary task similarity . . . 31

4 Dataset similarity concepts **33**
- 4.1 Hypotheses . . . 33
- 4.2 Requirements . . . 34
- 4.3 Framing label similarity as a clustering comparison problem . . . 35
- 4.4 Evaluation of clustering comparison measures . . . 37
- 4.5 Calculation of dataset similarity from labels . . . 40
  - 4.5.1 Text overlap . . . 41

4.5.2 Vector space similarity . . . 44
4.6 Calculation of dataset similarity from gradients observed during training 46

5 Neural MTL system implementation 49
5.1 Objectives . . . 49
5.2 Architecture . . . 50
5.3 Design decisions and training process . . . 52
5.4 Implementation summary . . . 55

6 Dataset similarity tool 57
6.1 Objectives . . . 57
6.2 Architecture overview . . . 57
6.3 Text overlap . . . 59
6.4 Vector space similarity . . . 61
6.5 Contingency table similarity measures . . . 64

7 Experiments 67
7.1 Preliminary evaluation of dataset similarity design decisions . . . 67
7.1.1 Evaluation of the similarity of identical datasets . . . 68
7.1.2 Comparison of the approaches to fill the contingency table . . . 69
7.1.3 Run time efficiency of word vector comparisons . . . 72
7.2 Experimental Setup . . . 74
7.3 Results and analysis . . . 77
7.3.1 Part-of-speech tagging . . . 77
7.3.2 Named entity recognition . . . 82
7.4 Multi-task learning test score prediction . . . 87

8 Summary, Conclusion & Future Work 89
8.1 Summary . . . 89
8.2 Conclusion . . . 90
8.3 Future work . . . 90

List of Figures 93

List of Tables 95

Bibliography 97

A Appendices 111
A.1 Neural MTL system implementation details . . . 111
A.2 Dataset similarity tool implementation details . . . 113
A.3 Preliminary dataset similarity evaluation results . . . 116
A.4 Experiment results POS tagging . . . 122
A.5 Experiment results NER . . . 127

1 Introduction

In the field of computational natural language processing an example for a simple, concrete task is part-of-speech tagging in which each word in a text has to be labeled with its corresponding part of speech, e.g. noun, verb or adjective. To automatically label texts with the part-of-speech labels, manually labeled texts are used to train a program for this task. This thesis shows how to find the best text to improve the training process depending on the labeled texts' similarities.

1.1 Motivation

The trend in computational natural language processing is to use data-based machine learning methods for a variety of tasks. State-of-the-art results on most tasks are mainly achieved by employing deep neural networks for the entire natural language processing pipeline. Such methods require a large amount of training data in order to learn all the parameters in their complex models.

In many cases, the dataset used for training does not contain enough samples to obtain satisfactory results. Data scarcity is a problem especially for uncommon tasks or when working with manually created datasets. Often it is not feasible to label more training data because the raw data source is limited or manual annotation is too expensive. In this case, the possible options are to reduce the model's complexity and accept a lower performance or use similar datasets as auxiliary training data. For common tasks in natural language processing, there are plenty of datasets available. In case of niche tasks, related common tasks can be leveraged. Auxiliary datasets can be incorporated into the training process by applying transfer learning or multi-task learning.

When including auxiliary datasets to improve the performance on the main dataset, the question arises which dataset(s) to use. In order to find the additional dataset or combination of datasets that provide the largest performance gain on the main task, it is necessary to try them all and perform individual hyperparameter optimization. Comparing different auxiliary datasets with the same hyperparameters would be like comparing apples to oranges. One of the following two types of errors is likely to occur: When testing on a low-capacity model, the gains of using more data are miniscule. In the other extreme, testing a high-capacity model with no or only small amounts of extra data will overfit, fail to generalize and thereby perform poorly.

Especially for complex deep neural networks with many hyperparameters, the process of comparing multiple auxiliary training datasets with individual hyperparameter optimization will take a long time and consume precious resources. For these reasons, a better way of selecting suitable additional datasets needs to be found.

1.2 Research question

It would be valuable to know whether it is worthwhile to perform transfer learning or multi-task learning for a specific task and training dataset given some auxiliary dataset. The question is how to estimate the effect of an auxiliary dataset on the main task without performing expensive training and hyperparameter optimization. The goal of this thesis is to devise a fast method to estimate the effect of auxiliary training data on the performance of the main task. More specific, the method should be able to distinguish datasets having the potential to improve results on the main task from those that will have no effect or even worsen the performance.

While there are numerous tasks in natural language processing, the scope of this thesis is limited to a subset of sequence tagging problems. Common sequence tagging tasks are for example part-of-speech (POS) tagging and named entity recognition (NER). To be easily applicable for a wide range of sequence labeling tasks and combinations of datasets, the method must work on raw training data. It should be able to compare arbitrary datasets without any constraints on the tokens or labels. Unlike the work by Bjerva (2017), the method must not be constrained to automatically taggable tasks. While the method should be able to work with datasets belonging to different tasks, only the effects between datasets of the same task will be explored in this thesis to reduce the scope to a feasible level. The subsequent problem, how to reliably evaluate the method to be designed, needs also to be resolved.

1.3 High-level overview

In order to provide a broad idea of how the research question will be tackled throughout this thesis, the high-level approach will be briefly illustrated. An example schematic of the approach is also shown in Figure 1.1. The effect of auxiliary datasets on the main task performance shall be estimated by comparing the similarity between the main and auxiliary dataset. Two sequence tagging datasets can be compared via their labels by using their shared vocabulary as a bridge. Words and labels from one dataset are joined with the words and their labels from another dataset to create a probabilistic mapping between both label sets. By assessing the quality of this label mapping, the similarity of the datasets will be measured. Finally, experiments will be performed to check whether the similarity correlates with the effect on the multi-task learning performance when using the second dataset as auxiliary training data.

Before the concepts for the outlined approach can be described in detail, the necessary theoretical background for the understanding of this thesis will be explained in following chapter. This includes the relevant aspects of sequence tagging, neural networks, multi-task learning and information theoretic measures. In Chapter 3, related work regarding multi-task learning for sequence tagging and the effect of auxiliary task similarity is summarized. The concepts of measuring dataset similarity are developed in Chapter 4. This includes a definition of the underlying hypotheses and precise descriptions of the functionality of multiple similarity calculation methods. In Chapter 5, the key aspects

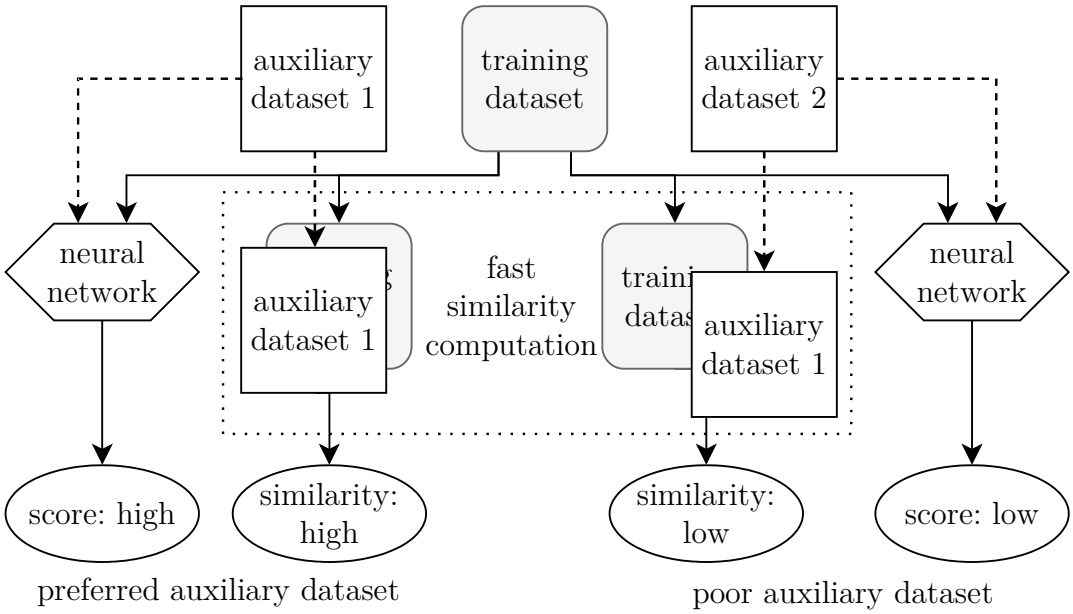


Figure 1.1: High-level approach example: Two auxiliary datasets are separately compared regarding their similarity with the training dataset and their usefulness as additional training data for a neural network.

of the neural multi-task learning system are described. The neural network is used in the evaluation of the dataset similarity concepts. The implementation details of the developed dataset similarity concepts are explained in Chapter 6. Aside from the software architecture, different methods are clarified to create a probabilistic mapping between the label sets of two datasets. In Chapter 7, the previously developed dataset similarity approaches are evaluated. In extensive experiments, their concordance with the effect of different auxiliary datasets on the multi-task learning performance is examined.

2 Theoretical background

2.1 Sequence tagging with statistical modeling methods

Sequence tagging is a class of tasks in natural language processing where each word or token is assigned a label (also called tag). For example in part-of-speech tagging, each word is tagged with a POS tag, e.g. adjective, noun, verb etc. When working with raw text data, the following steps are performed in the processing pipeline: Read the input text, split into sentences, tokenize each sentence into words, apply the desired tagging method and store the results. In case a preprocessed dataset is used, the pipeline is reduced to reading the input, tagging the words and storing the results. (Jurafsky and Martin, 2009, pp. 133–135)

The specialty of sequence tagging compared to other tasks such as classification is that the tags depend on the other tags in the same context. As an example for part-of-speech tagging, a verb is very unlikely to follow a determiner while adjective and noun are common POS tags after a determiner. These dependencies between already predicted labels and labels to be predicted in future require specific prediction methods to perform well. Three classic approaches have been used for sequence tagging in the past before the advent of neural models, which will be covered in Section 2.2.3: (1) Rule-based techniques rely on a large set of handwritten linguistic rules to tag a word. These rule sets quickly become unmanageably complex and time-consuming to maintain.(Jurafsky and Martin, 2009, pp. 137–139) (2) Transformation based learning starts with a simple solution and uses an algorithm to apply the best transformation rule in subsequent steps until no improving rules can be applied. The necessary transformation rules are extracted from an annotated training corpus.(Jurafsky and Martin, 2009, pp. 151–153; Manning and Schütze, 1999, pp. 361–365) (3) Statistical modeling methods are based on the probabilities of word occurrences for a particular tag. These probabilities are obtained from a training corpus. During inference, a probability distribution is computed over possible labels and the best label sequence is chosen. There are three different statistical modeling methods that will be described in greater detail as some of them are used as a part in state-of-the-art neural sequence taggers.

2.1.1 Hidden Markov Model (HMM)

The foundation of a Hidden Markov Model is a Markov chain, which can be represented by a special weighted finite-state automaton. It can be defined by a set of observable states, a transition probability matrix (probabilities for moving from each state to every other state) as well as a start and an end state. A first-order Markov chain adheres to the

Markov property: The probability of a particular state is dependent only on the previous state. For a Markov chain the input sequence must uniquely determine the states gone through the automaton. Thus, it cannot handle ambiguous sequences commonly found in natural languages. Further, a Markov chain can only compute a probability for an observable sequence. For sequence tagging it is necessary to obtain a probability for the labels while only observing the corresponding words. (Jurafsky and Martin, 2009, pp. 173–176; Manning and Schütze, 1999, pp. 317–320)

A HMM can be seen as an extension of a Markov chain solving its two main problems for use in natural language sequence tagging tasks. It allows to incorporate both the observed words and the labels in form of the hidden states. A HMM is a generative model, assigning a joint probability to sequences with pairs of observation and label. Compared to the Markov chain, its definition additionally includes a sequence of hidden states and a sequence of observation likelihoods called emission probabilities (probability of a particular observation being generated from a specific state). While the Markov chain has to use the observation i.e. vocabulary as its states, a HMM uses labels as its states and incorporates the observations into the model with their emission probabilities. As a consequence, HMMs are not deterministic so that generating the same observation can result in different states. The Markov property also holds true for a HMM. Emission of observations can be modeled either as a state-emission HMM or arc-emission HMM. In a state-emission HMM, a symbol is emitted for each state and the emission probability of an observation depends only on the state that produced it — not on any other states or observations. In case of an arc-emission HMM, symbols are emitted at the connecting "arc" between states. Thus, each generated symbol depends on both the previous i.e. source state and the next i.e. target state. (Jurafsky and Martin, 2009, pp. 177–179; Manning and Schütze, 1999, pp. 321–325)

Rabiner (1989) popularized three problems to be solved for a useful application on real-world tasks. (1) Efficiently compute the likelihood of an observation sequence, (2) find the state sequence that best explains the observations and (3) train the model to learn the transition probability matrix and the emission probabilities.

To compute the observation likelihood (1), a naïve algorithm is to sum all hidden state sequence probabilities. The probability of each hidden state sequence is the product of all transition probabilities. Instead of this exponential algorithm, an efficient dynamic programming solution called the forward algorithm is used. For an explanation of the forward algorithm refer to e.g. Jurafsky and Martin (2009, pp. 181–184).

Finding the best hidden sequence (2) for an observation sequence could be done by performing an exhaustive search over all possible hidden state sequences. Each sequence would require running the forward algorithm resulting in an extremely inefficient computation. Instead, the Viterbi algorithm can be used to efficiently perform the decoding, i.e. finding the best hidden state sequence. Refer to e.g. Jurafsky and Martin (2009, pp. 184–187) for a detailed description of the Viterbi algorithm.

Training a HMM (3) is typically performed with an algorithm called forward-backward or Baum-Welch (Baum, 1972) which is a specialization of the more general Expectation-Maximization algorithm (Dempster et al., 1977). It allows to iteratively learn both the transition probability matrix and the emission probabilities, which are randomly

initialized. If a HMM is trained on an annotated training dataset, no Expectation-Maximization training is necessary. The transition probabilities between the labels are directly observable from the training data by counting the occurrences of all label-pairs (Brejová et al., 2007).

Instead of only looking at the previous label to predict the current label (bi-gram tagger), a HMM can be extended to look at the two previous labels to improve its predictions. This results in a tri-gram HMM in which each state represents a label bi-gram. In theory, an arbitrary n-gram could be used, but in practice there are no gains due to data sparsity: Seeing a longer n-gram more than once is highly unlikely (Rosenfeld, 2000). Using word features such as capitalization, suffix etc. instead of the word itself to reduce data sparsity is impractical in HMMs as it results in many more states greatly increasing the number of parameters (Jurafsky and Martin, 2009, p. 208).

2.1.2 Maximum Entropy Markov Models (MEMM)

In contrast to the generative HMM, a Maximum Entropy Markov Model is a discriminative model conditioning the label on current and previous observations including arbitrary features. MEMM is based on Maximum Entropy Modeling, which is a multinomial logistic regression classifier. It allows to compute the probability of a single label given an observation, i.e. a word. Maximum Entropy models can be trained by finding the weights which maximize the log-likelihood of training samples with the help of a convex optimization algorithm. (McCallum et al., 2000; Jurafsky and Martin, 2009, pp. 201–207)

MEMMs combine Maximum Entropy together with the Viterbi algorithm used in HMMs. This enables finding the optimal label sequence for the whole sentence instead of selecting the best label for each word on its own. While HMMs include distinct probabilities for both state transition and observation, MEMMs produce only an estimate for the probability of the next tag given the observation and the previous tag. With an annotated dataset, the weights are easily trained to maximize the log-likelihood on the training corpus. (McCallum et al., 2000; Jurafsky and Martin, 2009, pp. 207–211)

Although MEMMs solve the lacking feature-support of HMMs, they are prone to the so-called label bias problem. Outgoing probabilities of the current state to the next states are normalized per state. When a state has many outgoing transitions, the probability mass has to be divided among them, resulting in lower average transition probabilities compared to states with fewer outgoing transitions. In the extreme case of a single outgoing transition, the current observation is fully ignored. In general, states with lower entropy are preferred regardless of the observations. This results in selecting a state sequence with high probability that occasionally does not fit to the observation at all. (Lafferty et al., 2001)

Figure 2.1 shows an example that illustrates the issue. Assuming the model starts in *State 1*, the path (1–1–1–1) has the highest probability $0.45 \cdot 0.45 \cdot 0.5 = 0.1013$. However, with greedy Viterbi decoding, the system will end in *State 2* because *State 1* has a higher probability to change to *State 2* whereas *State 2* prefers to stay in *State 2*. The path (1–2–2–2) only has a probability of $0.55 \cdot 0.3 \cdot 0.3 = 0.0495$. Yet it or another path leading to *State 2* will be the outcome for this example.

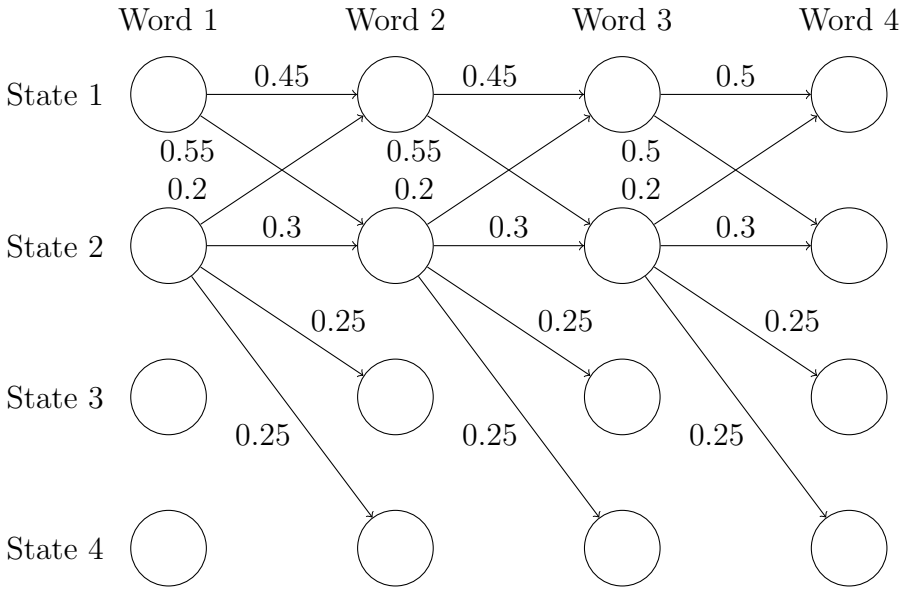
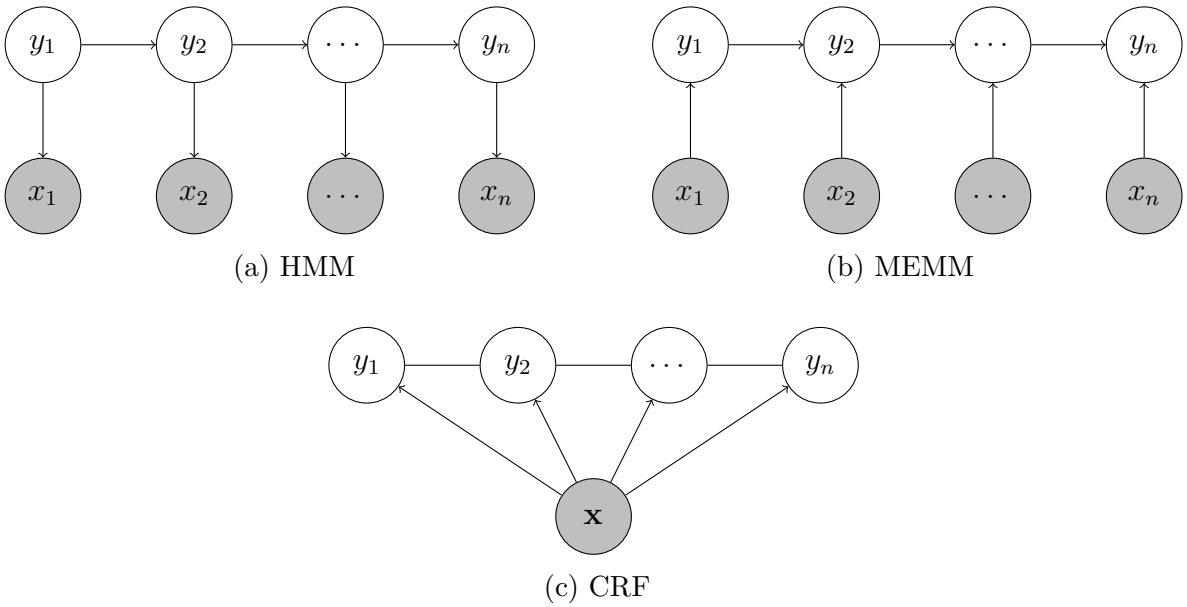


Figure 2.1: MEMM: Example illustrating the label bias problem, after Xing (2007)

2.1.3 Conditional Random Fields (CRF)

Conditional Random Fields combine the strengths of both HMMs and MEMMs without their weaknesses. A CRF is a discriminative model like MEMM with support for features on the observation sequence. By applying global normalization over the whole observation sequence, the label bias problem is solved. The idea is that transitions can have a higher probability depending on the observation. While it is possible to construct a HMM-like CRF by using an appropriate feature functions, CRFs are more powerful as arbitrary dependencies on the observation and state sequence can be modeled. In the simplest form, the linear chain CRF, each hidden state has two neighbors allowing dependencies on both previous and future states to be modeled. Figure 2.2 shows a comparison of HMMs, MEMMs and CRFs. The Markov property holds for linear chain CRFs, enabling efficient decoding using a variant of the Viterbi algorithm. (Lafferty et al., 2001)

The downside of the mutual dependence between labels and observations is a slower training process even on annotated datasets. Counting the occurrences of label-pairs for maximum likelihood estimation is no longer possible. Instead, iterative methods calling variants of the forward-backward and Viterbi algorithm can be used. The original and rather slow training algorithms are based on generalized iterative scaling and improved iterative scaling (Lafferty et al., 2001). Newer, faster approaches are based on gradient descent. This includes quasi-Newton methods such as the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm and conjugate gradient methods. (Sutton and McCallum, 2012)



(a) HMM

(b) MEMM

(c) CRF

Figure 2.2: Comparison of HMM, MEMM and CRF

2.2 Neural networks

An artificial neural network is a computational model inspired by biological neural networks such as the human brain. In this model, many simple entities, called neurons, are able to solve complex tasks through manifold connections between them.

With enough neurons, the right connections between them and suitable activation functions, a neural network can approximate any continuous function (Hornik et al., 1989). Therefore, neural networks are able to solve almost any complex task with one caveat: The correct weights of the connections need to be found. Before the general ideas to train the weights of neural networks will be outlined in Section 2.2.2, the fundamentals of neural networks will be described in the upcoming section.

2.2.1 Fundamentals

A single neuron

A single artificial neuron is the basic building block of an artificial neural network. It receives inputs from external sources or other neurons. Each of these input connections has a weight assigned allowing the values of each connection to be weighted individually. The weights are stored for each input and can be changed "learned" separately. These learnable weights allow a neural network to be adjusted to a specific task. (Schmidhuber, 2015)

A neuron sums up all of its inputs resulting in an weighted sum. It applies a function, called activation function, on this weighted sum and outputs the function's result. An

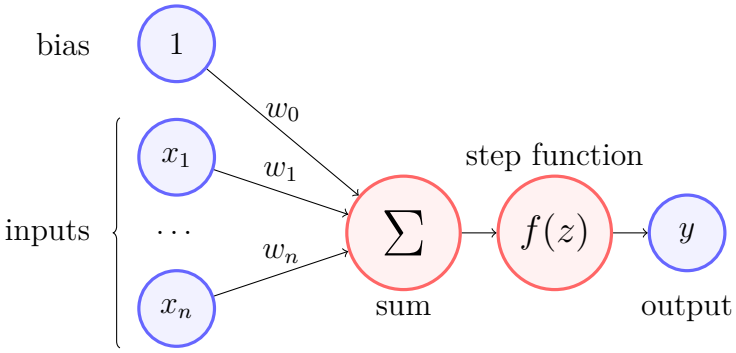


Figure 2.3: Perceptron

activation function can introduce non-linearity to the computation. This is needed for solving any non-linear problem, which is the case for most real-word problems. The output is connected to other neurons or to an external sink. This enables multiple neurons to have arbitrary connections between them allowing to form a complex neural network. (Freund and Schapire, 1999)

Perceptron

The perceptron (Rosenblatt, 1957) is a binary classifier mapping a real-valued vector as input to either false: 0 or true: 1. Figure 2.3 shows the structure of a perceptron. It is a simplistic neural "network" consisting of multiple inputs and a single neuron with a threshold function as activation function. The unit step function outputs 1 if its input (the weighted sum) is a positive value. In any other case, the output is 0. A perceptron uses an additional constant input, called bias, regardless of the number of real input signals. The bias is a constant input of 1 with a learnable weight w_0 allowing to add any value to the weighted sum of a neuron. It is used as an effective way to shift the activation function to the left or right, which is necessary because the activation function is fixed. Thus, it allows to output 1 even if the weighed sum of the inputs without the bias is negative. Otherwise, differentiating between two inputs with weighted sums of identical signs e.g. 0.5 and 1.5 would not be possible. (Freund and Schapire, 1999)

In order to solve a task with a perceptron, its weights need to be learned. Formal specification of a perceptron: $x = (x_1, x_2, \ldots, x_n)$ is a single training sample, where x_i represents the value of the ith feature. c is the correct output for the input vector $\mathbf{x}$ and y is the output of the perceptron for the same input. $x_0 = 1$ is the constant bias. $w = (w_0, w_1, w_2, \ldots, w_n)$ is the weight vector to be learned where w_i is the weight x_i is multiplied with. $f(z)$ is the activation function applied on the weighted sum. The training process additionally requires a learning rate λ typically in range $(0, 1]$. For training, the weights w can be initially set to 0 or random values. For each training sample (x, c), two steps are executed. (1) Calculate the perceptron's output $y = f(\sum w \cdot x) = f(w_0x_0 + w_1x_1 + \cdots + w_nx_n)$ (2) Update the perceptron's weights for

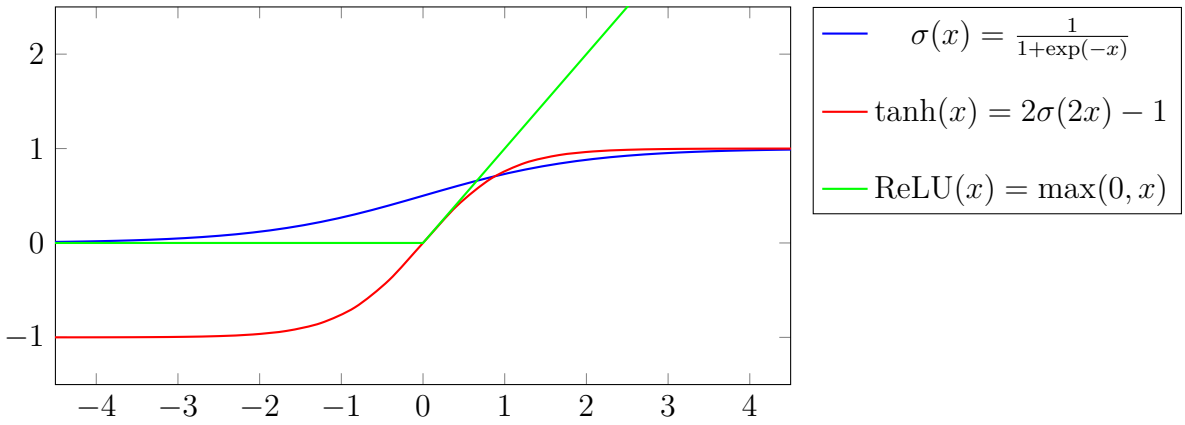


Figure 2.4: Activation functions: Sigmoid, tanh and ReLU

all features $0 \leq i \leq n$: $w_i = w_i + \lambda \cdot (c - y)x_i$. Looping over all training samples can be stopped at a predefined iteration count, when all training samples are classified correctly or when a user defined training error threshold is reached. Minsky (1987, pp. 62–68) has shown that the perceptron can only solve linearly separable problems. To overcome this limitation, multiple neurons can be stacked together as shown in Section 2.2.2.

Activation functions

Apart from the step function previously introduced with the perceptron, there are many other suitable activation functions. Figure 2.4 shows commonly used activation functions: sigmoid, tanh and ReLU. The sigmoid function $\sigma(x) = 1/(1 + \exp(-x))$ squashes a real-valued input to the range of 0 to 1. The hyperbolic tangent function $\tanh(x) = 2\sigma(2x) - 1$ can be defined to use the sigmoid function and squash the range of a real-valued input to range of -1 to 1. The tanh function has two valuable properties when working on normalized data compared to the simpler sigmoid function: It avoids bias in the gradients and provides stronger derivatives (Glorot and Bengio, 2010). This is helpful for training as explained later in Section 2.2.2. The Rectified Linear Unit (ReLU) $f(x) = \max(0, x)$ clips a real-valued input to zero or more. While it is not differentiable at $x = 0$, it does provide a constantly strong gradient of 1 for all positive inputs. This property becomes important for training when many neurons are used in succession (Krizhevsky et al., 2012).

2.2.2 Feedforward neural networks

A perceptron is the simplest feedforward neural network consisting only of one input layer and a single neuron producing the output. A natural extension is to use multiple neurons to produce more than one output. This changes the binary classifier to a multi-class classifier. The group of output neurons can also be seen as the output layer

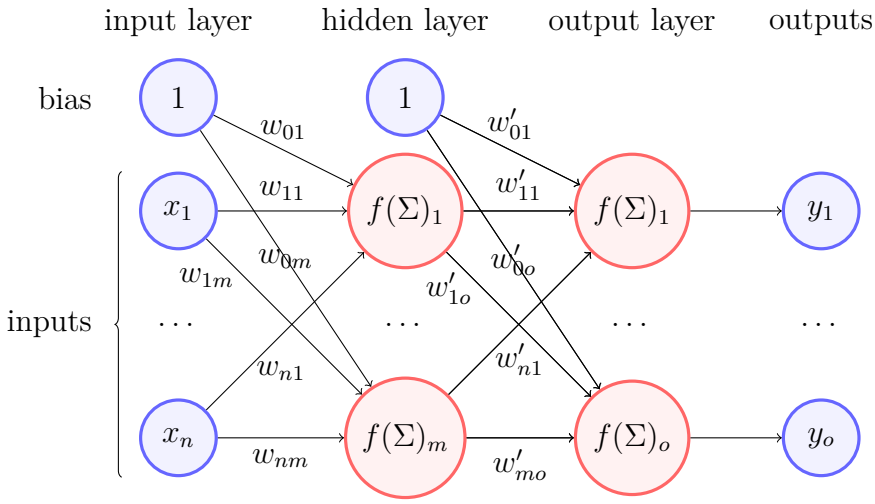


Figure 2.5: Two-layer fully-connected feedforward neural network with n input features, m hidden neurons and o output neurons

of the network. Feedforward networks in general have one input layer, any number of intermediate hidden layers and an output layer. Figure 2.5 shows a feedforward network with one hidden layer having m hidden neurons. It takes n features as input and uses o output neurons to produce o output signals. The number of inputs n, hidden neurons m and outputs o are independent. Further, it is possible to add more hidden layers, resulting in a deeper neural network.

In contrast to single-layer networks, multi-layer feedforward networks are universal approximators. Hornik et al. (1989) proved that neural networks with a single hidden layer using enough neurons can approximate any measurable function to any degree of precision.

Inference

Inference is the process of computing the neural network output for a given input. For a multi-layer feedforward architecture, it is necessary to calculate the output of each layer sequentially. Within a layer, many computations can be performed in parallel as neurons within the same layer are independent of each other.

The following explanation refers to the two-layer network shown in Figure 2.5. Every input is connected to each neuron in the hidden layer with an individual weight. The ith input for the jth hidden neuron needs to be multiplied with the weight w_{ij}. Then the weighted inputs are summed up for each hidden neuron. The weights can be stored in a two-dimensional matrix W with n columns and m rows. The input values are in an n dimensional vector x. Performing all these multiplications and summations can be done efficiently in a single operation by computing the matrix-vector product Wx resulting in an m-dimensional vector h. The jth component of h contains the weighted sum of the

inputs for the jth hidden neuron. Applying the activation function f in the hidden layer is an element-wise in-place operation on vector h.

If there were additional hidden layers, the intermediate output h would be used as an input for the next hidden layer analogous to the vector x. The weights of the output layer can be arranged in an $m \times o$ matrix W' similar to W. The intermediate result vector h is then used in the matrix-vector product $W'h$ producing an o-dimensional vector y. This computes the weighted sums of the output layer. Finally, the activation functions of the output neurons are applied on the vector o in an element-wise manner to obtain the actual output of the neuron network.

Inference in a feedforward neural network is a fast and straightforward process. There are numerous highly optimized software frameworks for matrix and vector operations capable of utilizing modern multi-core CPUs and GPUs. Due to the high amount of independent, parallel operations, large networks with thousands of neurons do not pose a problem for current computer hardware.

Training

While inference is fast and simple, training a multi-layer neural network is both more complex and significantly slower. With a perceptron having only a single layer the weights can be directly adjusted from the comparison of correct and actual outputs. When using multiple layers, this is no longer possible. The solution is to apply backpropagation together with gradient-based learning (Rumelhart et al., 1986). The idea is to minimize a loss function $L(x, y^*, \theta)$ where x is the input, y^* the correct output and θ are the network's parameters, i.e. the weights of all layers. Using mean squared error as loss function on a single training sample (x, y^*), the loss function is $L(x, y^*, \theta) = \frac{1}{4}(y^* - f(x, \theta))^2$. When plotting the actual output y against the loss $L(x, y^*, \theta)$, a multi-dimensional error surface is created. Gradient descent is an algorithm to iteratively minimize the loss by changing the network parameters in the direction of the steepest descent along the error surface. For multi-layer networks, "backpropagation" is used to efficiently calculate the steepest descent direction for every weight in the network. To compute the gradient of the loss function, the derivative of the neural network's function f is needed. It can be obtained by applying the chain-rule as f is a chain of matrix multiplications and activation functions from all layers. The term "backpropagation" describes the idea of propagating the error from the output back through the network to the first hidden layer. (Goodfellow et al., 2016, pp. 167–173)

For each iteration it is necessary to compute the actual output of each training sample, compute the error gradient and modify the weights. As non-linear activation functions are used in multi-layer networks training becomes a non-convex optimization problem without the guarantee of finding a global minimum. However, LeCun et al. (2015) argue that in practice, this does not pose a problem as gradient descent in a high-dimensional space is able to find a local optimum very close to the global one.

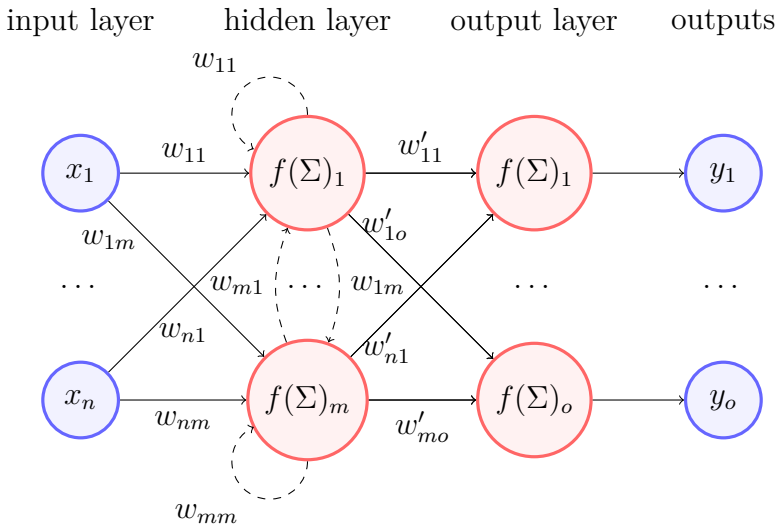
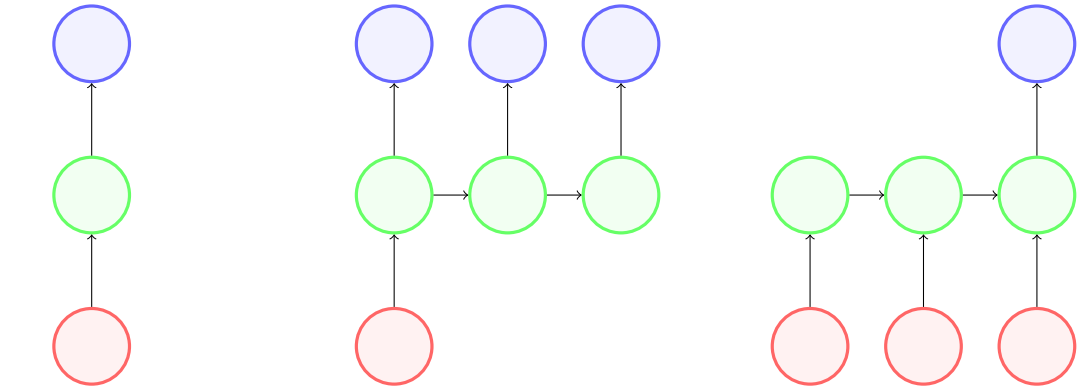


Figure 2.6: A two-layer recurrent neural network with n input features, m hidden neurons and o output neurons. The bias is omitted for clearer display. Recurrent connections are dashed.

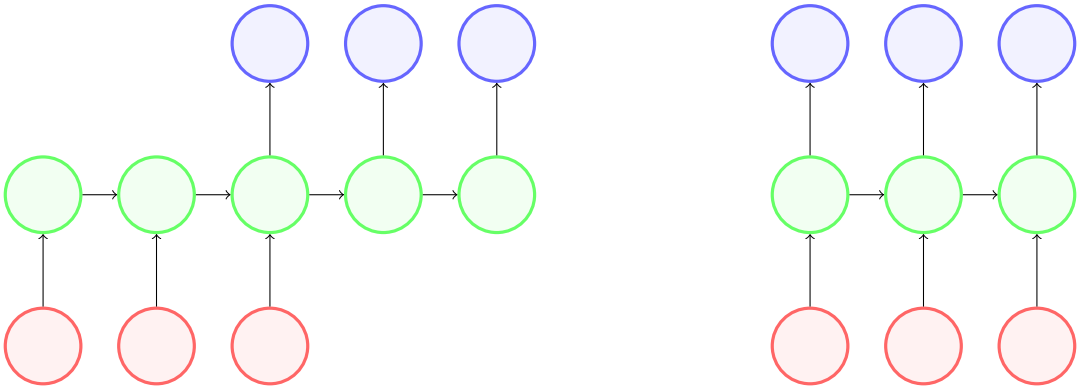
2.2.3 Recurrent neural networks

Recurrent neural networks (RNNs) are specialized for sequence processing. While feedforward networks can only work on fixed size input, RNNs can work with sequences of variable length. More importantly, they enable parameter sharing across the different positions in a sequence, reducing the amount of required training data and increasing the generalization capabilities. The output for each position is computed by a function over the current input and the previous hidden layer representation resulting in a recurrent computation. The recurrent network architecture as explained is shown in Figure 2.6 for a single hidden layer. If there were two or more hidden layers, the recurrent connections would go from the last hidden layer back to the first. (Goodfellow et al., 2016, pp. 367–368)

There are a many tasks that can be solved by working with sequences. Operating on sequences can be done in various forms. Sequences can either be the input, output or both. Figure 2.7 shows five different types of sequence processing with recurrent neural networks. The one-to-one mapping or no sequence processing (a) could also be performed by a feedforward network. The other four types use recurrent connections in the hidden layers. The one-to-many mapping (b) produces a variable length sequence as output from a single fixed size input. This is for example used when a textual description has to generated from an image. Having a sequence as input and mapping it to a single output (c) is used for classification tasks. Consuming and producing a sequence can occur in two forms: (d) Outputting a sequence of different length than the input starts after seeing the final input. Machine translation is a common task for this encoder-decoder architecture



(a) one-to-one (b) one-to-many (c) many-to-one



(d) many-to-many (e) many-to-many synced

Figure 2.7: Different types of sequence processing with recurrent neural networks. Figure adapted from Karpathy (2015). Red circles are inputs, green hidden layers and blue outputs.

proposed by Sutskever et al. (2014) and Cho et al. (2014). (e) Alternatively, each position in the input sequence is directly mapped to a position in the output sequence resulting in two sequences of equal length. This is used for sequence tagging, which is the focus in this thesis.

Inference

In order to perform the computations in a recurrent neural network, the operations have be reformulated from a recurrent definition to a linear chain of operations. A recurrent computational graph is unfolded to a directed acyclic graph with a repetitive structure. Unfolding the graph has to be done for each sequence individually as the number of repeated operations is equal to the sequence length. When the recurrence is employed in

the hidden layers, which is the most common way, Equation (2.1) can be used to define the values of the hidden neurons.

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta) \tag{2.1}$$

The hidden state $h^{(t)}$ is fed into the output layer to produce the final result for each time step t of the sequence. Unfolding Equation (2.1) results in a repetitive application of f using the same network parameters θ as shown in Equation (2.2).

$$h^{(t)} = f(f(f(\dots f(h^{(0)}, x^{(1)}; \theta) \dots, x^{(t-2)}; \theta), x^{(t-1)}; \theta), x^{(t)}; \theta) \tag{2.2}$$

$h^{(0)}$ is the initial hidden state which can be a vector of zeros or small random values. The same function f can be used with the same parameters θ for each element in a sequence allowing to share the model parameters regardless of the sequence length. The unfolded computation graph and hidden state equation make the implementation of the inference process straightforward and build the basis for the training process. (Goodfellow et al., 2016, pp. 369–372)

The forward computation, i.e. inference, for the RNN in Figure 2.6 consists of four operations per time step: (1) Transforming the input to the dimensions of the hidden layer, (2) summing up the inputs together with the recurrent connections, (3) applying the activation function to produce the new hidden state and (4) transforming the hidden state to the number of outputs. This is described in the following update equations given an initial hidden state $h^{(0)}$

$$a^{(t)} = Ux^{(t)} \tag{2.3}$$
$$b^{(t)} = Wh^{(t-1)} + a^{(t)} \tag{2.4}$$
$$h^{(t)} = f(b^{(t)}) \tag{2.5}$$
$$o^{(t)} = Vh^{(t)} \tag{2.6}$$

with input x, output o, f as activation function in the hidden layers as well as weight matrices U, V, and W for input-to-hidden, hidden-to-output and hidden-to-hidden connections. (Goodfellow et al., 2016, pp. 372–376)

Training

Gradient based learning and error backpropagation can be used on the unfolded RNN computation graph. The backpropagation through time (BPTT) algorithm is generalized from the backpropagation algorithm used for training a feedforward network (see Section 2.2.2, page 13). It accounts for the shared parameters of the recurrent connections.

To apply BPTT, the forward calculations have to be performed on the unfolded network graph. As each time step depends on the previous computations, the process is inherently sequential and cannot be parallelized. Thus, the runtime increases linearly with the sequence length. Computing gradients and propagating the errors from the loss function after the output layer back through the entire unfolded computation graph

requires all intermediate hidden states to be available. Hence, the backward pass requires both space and time linear in the sequence length. Similarly, to the feedforward network, these computations have to be performed for many training samples and iterations of the entire dataset. Learning the weights in RNNs is therefore a very expensive process. (Goodfellow et al., 2016, pp. 374–380)

The vanishing gradient and exploding gradient problems

Training a RNN is not only time-consuming but also extremely difficult with longer sequences. Bengio et al. (1994) first described the *vanishing gradient* and *exploding gradient* problems. Both originate from the exponential behavior of the hidden weight gradients. Their norm either grows exponentially (exploding) or goes to 0 exponentially (vanishing). While the product of many real-valued numbers between 0 and 1 exponentially shrinks to 0, multiplying many numbers greater than 1 exponentially goes to infinity. The matrix multiplications in a RNN have the same issues leading to the exponential gradient behavior. When gradients vanish, the weights are only changed according to the last positions in the sequence. With exploding gradients, the first few positions of a sequence greatly outweigh the later positions in their effect on the weights. In both cases, the network fails to learn temporal correlations between distant positions in the sequence. (Pascanu et al., 2013)

Gated RNNs

Today's most commonly used solution to the training problems of ordinary RNNs are gated RNNs. The *long short-term memory* (LSTM) was the first gated RNN enabling effective processing of sequences. The main idea is to create a computation graph with connections through time having constant gradients that can neither explode nor vanish. Long-term information is stored in an internal state per *cell* that is untouched by gradient-based learning. Instead, the gated network learns additional weights in its *gates* to decide when to overwrite the current state with new information. (Goodfellow et al., 2016, pp. 404–407)

Hochreiter and Schmidhuber (1997) first introduced the *long short-term memory* model with *cells* containing a self-loop and weights to control the weight change of the looped connection. Instead of using simple neurons, a LSTM network consists of *LSTM cells* having an internal recurrence (the self-loop) as well as a forget-, input- and output-gate in addition to the usual input and output of an artificial neuron. Figure 2.8 shows a single LSTM cell. These cells are still recurrently connected to each other and replace the ordinary hidden neurons in a RNN. (Goodfellow et al., 2016, pp. 404–407)

Input, output- and forget-gate use the sigmoid function σ squashing their input to a range in $(0, 1]$. The forget-gate determines how much from the previous state vector is retained based on the current input vector. A 1 keeps all information while a 0 discards everything. The operation is applied by a pointwise vector multiplication allowing some part of the hidden state to be erased and another to be kept fully intact. Input and output gate work analogously and determine how much and which parts of the current

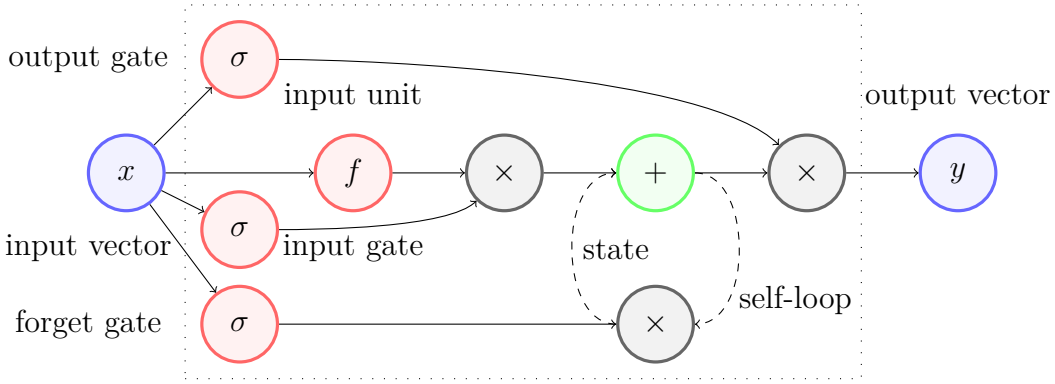


Figure 2.8: LSTM cell

input is incorporated into the hidden state respectively the hidden state is output. The input unit can have any nonlinear squashing function such as tanh. (Goodfellow et al., 2016, pp. 404–407)

LSTMs have been used successfully in many sequence processing tasks e.g. speech recognition (Li and Wu, 2015), machine translation (Sutskever et al., 2014) or parsing (Kiperwasser and Goldberg, 2016), often obtaining a new state of the art. While LSTMs overcome training issues of classic RNNs and achieve great results, their computational costs are enormous as they require four fully-connected linear layers per cell for each time step in a sequence (Culurciello, 2018). These computations cannot be parallelized and are limited by memory bandwidth so that implementations are inefficient on CPUs and GPUs. Therefore, training LSTMs or other gated RNNs takes even longer than the already time-consuming training of classic RNNs.

Apart from the original there are many LSTM variants. An important change adds *peephole connections* allowing the gate layers to look at the hidden state (Gers and Schmidhuber, 2000). This conditions the self-loop weights on the context, thereby improving performance. In another variant, forget and input gates are coupled. Thus, new information can only be stored when forgetting old information at the same time. The Gated Recurrent Unit (GRU), introduced by Cho et al. (2014), is a minimalist variant of the LSTM. It replaces the forget- and input-gate with a single update-gate. Additionally, cell and hidden state are combined making the GRU a simpler and computationally faster version of the LSTM while retaining the ability to dynamically control which information to keep and which to forget. (Goodfellow et al., 2016, pp. 407–408)

2.2.4 Regularization

Regularization is important in parameter-rich neural networks to avoid overfitting. If a network overfits, the training error is low but the test error is high. Parameter-rich networks tend to memorize the training data by learning a too complex solution that includes random noise. Many regularization techniques have been proposed to prevent

overfitting and to improve generalization, i.e. parameter norm penalties (L^1, L^2), data augmentation, parameter sharing, early stopping and dropout. (Goodfellow et al., 2016, pp. 224–225)

Parameter norm penalties Parameter norm penalties introduce an additional penalty on the weights based on their norms. This can be achieved by adding a regularization term $\Omega(\boldsymbol{\theta})$ to the loss function L used for training. The regularized loss function $\tilde{L}$ is defined by

$$\tilde{L}(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}) = L(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}) + \alpha\Omega(\boldsymbol{\theta}), \tag{2.7}$$

where α is a real-valued hyperparameter that weights the influence of the regularization term $\Omega(\boldsymbol{\theta})$ to the plain loss function L. The regularization term only depends on the parameters $\boldsymbol{\theta}$ unlike the loss function, which additionally operates on input $\mathbf{x}$, correct output $\mathbf{y}^*$ and predicted output $f(\mathbf{x}, \boldsymbol{\theta})$. Parameter norm penalties for neural networks are usually not applied on all parameters $\boldsymbol{\theta}$. The bias weights are excluded because their values are essential to shift the activation function according to the problem. Regularizing also the bias significantly reduces the performance of a model. Let $\boldsymbol{\omega}$ denote all network parameters of $\boldsymbol{\theta}$ that shall be affected by a norm penalty. The two common choices for the penalty norm are L^1 and L^2 regularization. L^2 regularization is defined as $\Omega(\boldsymbol{\theta}) = \frac{1}{2} \|\boldsymbol{\omega}\|_2^2$. It is also called weight decay because it drives the weights closer to zero. When performing a weight update with gradient descent, the weights are decreased multiplicatively with each step. L^1 regularization is defined as $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\omega}\|_1 = \sum_i |\omega_i|$. Using the sum of absolute values as a penalty term has the effect that solutions tend to be sparser compared to L^2 regularization. Because many parameters are 0, it works as a network-wide feature selection mechanism. (Goodfellow et al., 2016, pp. 226–233)

Data augmentation One of the best ways to help neural networks generalize is to use more data for training. The idea of dataset augmentation is to create fake data by making changes to the original training samples that should not affect the correct result. While this is easy for classification tasks working on image or sound data, many tasks cannot be augmented with fake data. When working on image classification tasks, new training data can be created by small changes like lightly shifting an image or changing its brightness. A similar method is to inject random noise into the network instead of the input. (Goodfellow et al., 2016, pp. 236–238)

Parameter sharing Another approach to regularize a model is to incorporate domain knowledge of the task. A common setting for parameter sharing is that some weights either within a single network or between multiple networks should be similar. Tying related parameters $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ together is made possible by applying a parameter norm penalty on their difference: $\Omega(\boldsymbol{\theta}) = \frac{1}{2} \|\boldsymbol{\omega}_1 - \boldsymbol{\omega}_2\|_2^2$. A stronger parameter sharing is achieved when the same network parameter is reused for multiple inputs like in convolutional neural networks (CNNs). CNNs are primarily used to work on image data. Instead of having unique parameters i.e. weights for every input pixel, a fix number of parameters

are reused for each pixel. This kind of parameter sharing effectively increases the number of data points used to train a single weight. (Goodfellow et al., 2016, pp. 249–251)

Multi-task learning is a specific parameter sharing technique and thus has a regularizing effect. Generalization is improved by the increased amount and diversity of training data from multiple tasks. Sharing network parameters across multiple datasets constraints the model to good values that are not specific to the random noise of a single dataset. Multi-task learning will be described in greater detail in Section 2.3.

Early stopping A pragmatic technique to prevent overfitting based on a common training process pattern is early stopping. While the training error decreases over time, the validation error increases after a certain number of training epochs. From this point on, the model overfits the training data. The validation error must be computed every epoch to use early stopping. Every time a better value is obtained, the current network parameters have to be stored. The training is stopped and the best parameters are reloaded when there have not been any validation error improvements for a number of epochs. Early stopping is simple to implement, has a low computational overhead and is not constrained to specific network types. The theoretical effectiveness is similar to L^2 regularization when the weights are initialized around zero because early stopping restricts the parameters to a small space close to their initial values. (Goodfellow et al., 2016, pp. 241–249)

Dropout Dropout (Srivastava et al., 2014) is a computationally inexpensive method to approximate an ensemble of many similar neural networks. The primary principle is to randomly remove non-output neurons from the network by multiplying their values with zero. For each minibatch, each affected neuron is independently randomly masked with a configurable probability p. Thus, many "subnetworks" are formed and trained where some neurons are missing while all the other parameters are shared because it is still a single network. All neurons are active during inference, which would lead to stronger activations compared to the training. In order to correct this, all affected weights are multiplied by p before inference. The so called "weight scaling inference rule" approximates the expected output values of the network when dropout is not used at all. Dropout is particularly effective due to the masking of hidden neurons. It randomly disables some feature detectors, so the network has to learn similar, redundant neurons or adjust the neurons in the next layer to infer the missing information. According to Srivastava et al. (2014), dropout is more effective than other inexpensive regularizes such as weight decay. Dropout works generally well for any kind of network and can be combined with other regularization techniques. (Goodfellow et al., 2016, pp. 255–265)

2.2.5 Hyperparameter optimization

Designing a neural network to solve a task requires the configuration of many hyperparameters. Hyperparameters are all adjustable network and training parameters that are no model weights, e.g. the number of hidden units or the learning rate. They can affect

both the performance of the model and the computational cost. All design decisions can be seen as selecting hyperparameters, but usually the type of the network and its architecture are considered fixed. Choosing hyperparameters can either be done manually or automatically. While manual tuning requires a deep understanding of neural networks and experience, automatic selection by an algorithm lowers the knowledge requirements but increases the computational cost by orders of magnitude. In either case, the goal is to increase the performance on unseen data, i.e. the test set, while complying to some runtime and memory limitations. (Goodfellow et al., 2016, pp. 422–431)

Manual hyperparameter selection requires a thorough understanding of the relations between training and test error, model capacity as well as the effects of hyperparameters on these. The model's capacity has to match the task complexity and amount of training data. It depends on the representational capacity of the model, the learning algorithm's ability to minimize the cost function and the regularization effect of cost function, architecture and training procedure. A higher number of neurons, i.e. more layers and hidden units per layer, increase the representational capacity allowing approximating more complex functions. However, the final model capacity is usually much lower as the learning algorithm cannot learn all possible functions. Also, the applied regularization may forbid many of these functions. (Goodfellow et al., 2016, pp. 422–426)

The effect of many hyperparameters can be explained with the model capacity. Adding more hidden units increases the model capacity but also the consumed computation time and memory. More neurons per layer and more layers behave similarly with regard to these aspects but may have other effects on the model. Setting a higher dropout rate decreases the capacity as it becomes more difficult for a group of neuron to perfectly fit the training set. Dropout works by randomly disabling the output of some hidden neurons in each step. Another way to decrease the model capacity is to increase the weight decay coefficient, which restricts the weights to smaller values. The learning rate significantly influences the effective capacity. If it is too high or too low for the specific network and training setting, minimizing the cost function will fail. Looking at the error surface the training will stay on a high plateau or shoot across areas descending to a lower error level. (Goodfellow et al., 2016, pp. 422–426)

Automatic hyperparameter selection is an algorithm to find the hyperparameters achieving the best result on a validation dataset. The difficulty of the manual tuning is significantly reduced but not completely removed. Instead of directly choosing the right value for a hyperparameter, a list or range of possible values has to be supplied. (Goodfellow et al., 2016, p. 427)

When only few hyperparameters need to be set, trying all combinations, i.e. their Cartesian product, in a brute-force manner is a simple solution. This approach is called grid search. The candidate values per hyperparameter are usually chosen in a logarithmic scale to cover a wide range. To find good results, it is necessary to repeat grid search each time widening or narrowing the range of possible values for a hyperparameter. Due to the exponential number of combinations, the cost of grid search is often too high as each combination requires a full training run, which itself is typically already a very expensive operation as pointed out before. (Goodfellow et al., 2016, pp. 427–429)

Bergstra and Bengio (2012) showed that random search is the better approach for

automatic hyperparameter optimization. Instead of trying all combinations, changing only a single hyperparameter in each iteration, random search chooses all hyperparameters randomly from a range or distribution of possible values. This allows to reduce the number of wasted iterations, when some hyperparameters do not have a measurable effect on the performance. While random search converges faster to good values for the hyperparameters, it is still necessary to make repeated runs each time refining the range of the possible hyperparameter values. (Goodfellow et al., 2016, pp. 429–430)

2.2.6 Word embeddings

Word embeddings are commonly used in natural language processing. They allow words to be represented as a numeric vector instead of text. This is a crucial component for natural language processing with neural networks because artificial neurons can only operate on numeric values but not on raw text. Each word vector points to a position in a high-dimensional vector space. Numeric representations of words can be compared using mathematical operations within the same vector space. By computing the distance of vector representations, similar words can be found, e.g. words that are somehow semantically related.

The construction of word embeddings is based on the distributional hypothesis of Harris (1954). It states that words have similar meanings if they appear in similar contexts. Bengio et al. (2003) and Collobert and Weston (2008) propose neural-network-based approaches to embed words in a vector space. The breakthrough came with an efficient embedding algorithm proposed by Mikolov et al. (2013a,b): By training the skip-gram model with the negative-sampling method and using its representations as features for various neural networks many state-of-the-art results on NLP tasks were achieved. The open source implementation *word2vec*[1] greatly helped to popularize the model. Baroni et al. (2014) compare *word2vec* embeddings with classic distributional word vectors obtained from *positive Pointwise Mutual Information* or *Local Mutual Information* and reduced in dimensionality with *Singular Value Decomposition*. According to their results, neural embedding methods significantly outperform the classic approaches on many tasks. *GloVe*[2] (Global Vectors) is another model to obtain vector representations for words. While *word2vec* is a predictive model, *GloVe* is count-based. Joulin et al. (2017) propose *fastText*[3] as a significantly faster method to train and use word embeddings. Additionally, it allows using subword information, which increases performance in downstream tasks especially for unseen words (Bojanowski et al., 2017). All these approaches have in common that they produce the same vector representation for each word regardless of its context. Peters et al. (2018) propose *ELMo* with deep contextualized word representations. The word vectors are obtained from a character-based deep bidirectional LSTM network. The exact numeric representation of a word depends on the current context. Thus, every instance of a word within a text has a different vector representation, which allows modeling polysemy, i.e. words having multiple meanings but the same surface form.

[1] `https://code.google.com/archive/p/word2vec/`
[2] `https://nlp.stanford.edu/projects/glove/`
[3] `https://fasttext.cc/`

Devlin et al. (2019) introduce *BERT* based on the Transformer model (Vaswani et al., 2017), which is an attention based model with positional encodings to represent word positions. *BERT* produces context-aware embeddings similar to *ELMo*, but it internally uses subwords like *fastText* instead of characters or words.

2.3 Multi-task learning

The standard approach to solve a task in machine learning is to optimize some task-specific metric. A single model or an ensemble of models is trained and fine-tuned on the training data of the task to be solved. The single-task learning (STL) approach ignores additional information from related tasks that might help to improve performance.

Multi-task learning (MTL), sometimes also referred to as joint learning or learning with auxiliary tasks, is a solution to this shortcoming. Performance can be further improved by sharing common representations in a model leading to better generalization abilities (Caruana, 1997). MTL often uses more than one loss function, one for each task, to train a model jointly on multiple tasks. It has been successfully applied in natural language processing tasks such as sequence tagging (see Section 3.1, page 29) as well as many other domains like computer vision (Misra et al., 2016).

2.3.1 Theory and application in neural networks

MTL requires sharing common representations, which is done by either hard or soft parameter sharing. Hard parameter sharing, introduced by Caruana (1993), is the oldest and most common approach. While the parameters of hidden layers are shared by all tasks, each task has its own output layer. Baxter (1997) showed that joint training on T tasks reduced the overfitting of shared parameters by an order of T. In soft parameter sharing, each task has its own parameters in hidden layers, but the distance between these weights is regularized by enforcing them to be similar. (Ruder, 2017)

While multi-task learning can be motivated via regularization (solving a loss function with two optimization problems at once), it provides benefits beyond those of regularization. The (theoretical) reasons why MTL works are manifold (Maurer et al., 2016). Caruana (1997) first proposed many of the following mechanisms. (1) *Inductive transfer*: An inductive bias provided by auxiliary tasks steers a model into the direction of preferring solutions that explain more than one task. This enables better generalization to unseen data and new similar tasks. (2) *Data augmentation*: Increasing number of training examples and averaging the different noise patterns of various tasks or datasets allows ignoring data-specific random noise (Goodfellow et al., 2016, p. 241). (3) *Attention focus*: It is often difficult to find relevant features in noisy, limited or high-dimensional data. Useful features may be easier to find in auxiliary tasks, enabling to focus on the important features while ignoring others (Ruder et al., 2019). (4) *Regularization*: The inductive bias lowers the risk of overfitting and decreases the random noise fitting ability of a model (Søgaard and Goldberg, 2016).

Figure 2.9 shows the structure of a hard parameter sharing neural network. While all

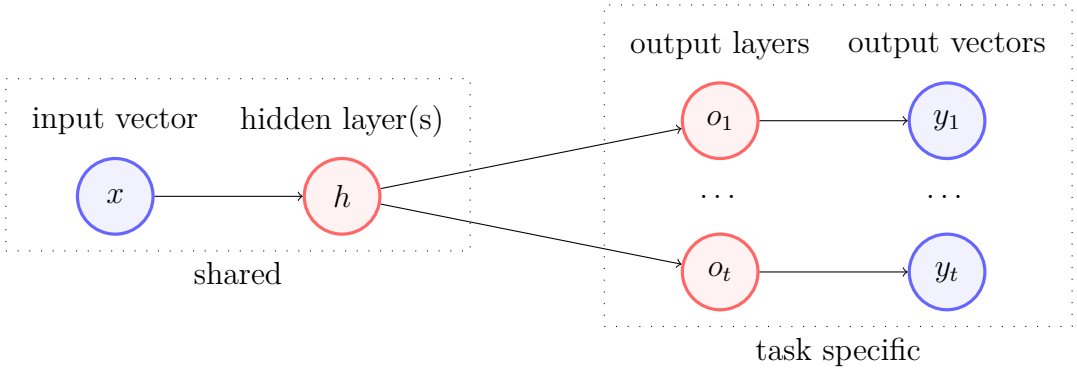


Figure 2.9: Generic hard parameter sharing multi-task learning neural network architecture for t different tasks

tasks share the inputs and hidden layer(s), each task has its own output layer producing a separate output vector. Using training data with multiple labels, e.g. POS and NER tags, the loss for each task is propagated back to the hidden layers from all output layers. If the tasks are not performed on the same input instances because each task is a different dataset, the network is only partially active. When task i is currently used for training, only the connections from hidden layer to output layer of task i are active. Other output layers are ignored and do not produce an output. Changing the weights with backpropagation is always done for hidden layer and the currently active output layer. Parameters in shared hidden layers are therefore trained with the combined error-signal of all tasks while the task-specific parameters are trained from less training examples. Thus, the hidden layers tend to learn to generalize instead of overfitting to the training data.

One difficulty of MTL is to decide what parameters to share. Hard parameter sharing with manually specifying the layers is used in most papers as it is both easy to implement and known as an effective regularizer. Apart from this approach, few other ideas have been explored in hope of better architectures. In computer vision there are *multilinear relationship networks* (Long et al., 2017), *fully-adaptive feature sharing* (Lu et al., 2016), *cross-stitch networks* (Misra et al., 2016) and *using uncertainty to weigh losses* (Kendall et al., 2018). In natural language processing, Søgaard and Goldberg (2016) proposed a hierarchy of tasks with low-level tasks supervised at lower layers. Based on this finding Hashimoto et al. (2017) propose a hierarchical architecture for multiple tasks. Ruder et al. (2019) propose *sluice networks* as a generalization of previous approaches (hard and soft parameter sharing, cross-stitch networks as well as task hierarchies). The architecture learns which layers to share and which layers provide the best output for each task.

2.3.2 MTL variants and differences to transfer learning

Multi-task learning scenarios can differ along multiple dimensions. (1) While in some cases the focus is only on the main task, in others a system has to perform well on all tasks during both training *and* inference. (2) Older approaches assume tasks are from the same distribution (Baxter, 1997). As this might only rarely be the case, more recent papers are not based on this assumption. (3) In a homogeneous setting, all tasks share an output of identical format. The more flexible heterogeneous setting allows a unique output format per task (Ruder, 2017). (4) Training multiple tasks can either be done on different datasets or on a single dataset (Søgaard and Goldberg, 2016; Schulz et al., 2018). In the latter case, the data has multiple labels allowing different supervised tasks to be trained. Alternatively, unsupervised tasks are combined with a single supervised task.

Transfer learning is a somewhat similar idea to multi-task learning. Using auxiliary data, the performance of the primary task should be improved, especially when training data is scarce. In transfer learning a model is pre-trained on a huge dataset. This pre-trained model is used to learn the specific target task. In computer vision, a common approach is to reuse a large and complex convolution neural network model trained on the large ImageNet dataset (Krizhevsky et al., 2012). The last, fully-connected classification layers of these networks are then replaced or retrained to another task. Weights in the lower layers stay fixed or are gradually "unfrozen" and fine-tuned from the output layer back to the input layers. In natural language processing, Howard and Ruder (2018) recently showed how to transfer knowledge based on large-scale language modeling. Before, only partial knowledge transfer in form of word embeddings such as *word2vec* (Mikolov et al., 2013a) or *ELMo* (Peters et al., 2018) was used extensively in NLP (see Section 2.2.6).

Transfer learning is different from multi-task learning in multiple aspects. (1) Only the results on the main task are relevant. (2) Typically, only two tasks are combined, one universal task for pre-training and the main task. (3) Tasks must not necessarily be trained in a supervised manner. (4) In MTL, all tasks are trained jointly, whereas in transfer learning they are trained in sequence one after the other.

2.4 Information theoretic clustering comparison measures

Clustering comparison measures enable the comparison of different clusterings. They are typically used to select a clustering that is most similar to a given *ground truth*, i.e. perfect clustering. While there are various types of clustering comparison measures, only measures based on information theory will be presented in this section. Before these clustering comparison measures are introduced, essential information theoretic measures are presented that build the basis for the clustering comparison measures.

2.4.1 Entropy, joint entropy and conditional entropy

Entropy is a measure for the uncertainty of a random variable. Intuitively, entropy comes close to a measure of information. Cover and Thomas (2006, pp. 13–15) define the entropy $H(X)$ of a discrete random variable X with alphabet $\mathcal{X}$ by

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x) \tag{2.8}$$

where $p(x)$ is the probability mass function $p(x) = \Pr\{X = x\}, x \in \mathcal{X}$. The log is actually $\log_2$, i.e. to the base two, which also applies to the following equations in this chapter. The unit of entropy is bits. Entropy cannot be a negative value. It is 0 when $p = 0$ or 1 because the variable X is not random and maximal when $p = \frac{1}{|\mathcal{X}|}$ (uniform distribution) corresponding to maximal uncertainty. The maximal entropy depends on the number of elements in X and has an upper bound of $H(X) \leq \log |X|$ (Cover and Thomas, 2006, p. 29).

Joint entropy extends entropy from a single to two random variables. Cover and Thomas (2006, pp. 16–18) define the joint entropy $H(X, Y)$ for a pair of discrete random variables (X, Y) with a joint probability distribution $p(x, y)$ as

$$H(X, Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(x, y). \tag{2.9}$$

The conditional entropy $H(Y|X)$ can be defined by

$$H(Y|X) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(x|y) \tag{2.10}$$

and is bound in the range $0 \leq H(Y|X) \leq H(Y)$ (Cover and Thomas, 2006, p. 29). It measures how much information is needed to describe the result of Y given knowledge of X. The joint entropy of two random variables equals the entropy of a single variable plus the conditional entropy of the other:

$$H(X, Y) = H(X) + H(Y|X) \tag{2.11}$$

2.4.2 Mutual information

Mutual information (MI) is a measure describing the amount of information one random variable contains about another. Cover and Thomas (2006, pp. 19–22) define the mutual information $I(X; Y)$ for the random variables X and Y with probability mass functions $p(x)$, $p(y)$ and a joint probability mass function $p(x, y)$ by

$$I(X; Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log \left(\frac{p(x, y)}{p(x)p(y)} \right). \tag{2.12}$$

Its value is always non-negative and not larger than the smaller entropy of X or Y resulting in a range of $[0, \min\{H(X), H(Y)\}]$ (Vinh et al., 2010). Mutual information can alternatively be defined by

$$I(X; Y) = H(X) - H(X|Y) \tag{2.13}$$

as it describes the reduction of uncertainty of X based on knowledge of Y. It is a symmetric measure and is identical to the entropy of one variable when both random variables are the same.

2.4.3 Comparing clusterings

Clustering data is a common task in many disciplines. A clustering C is a way to partition a dataset D into non-overlapping subsets $\{c_1, c_2, \dots\}$ together containing all N items of D. There are numerous algorithms to obtain a clustering from a dataset. In order to find the best clustering for a specific task, clusterings have to be compared. Another need for comparing clusterings arises when other problems are transformed to a clustering comparison to the end that aforementioned powerful algorithms can be leveraged. This can be done for many tasks, e.g. the comparison of lexical chains (Remus and Biemann, 2013). Such a comparison requires a measure determining the quality of a clustering according to a *ground truth*, i.e. the perfect solution. Thus, it should quantify the amount of information shared between both clusterings. (Vinh et al., 2010)

A contingency table, also called confusion matrix, is a convenient tool to compare the item overlap of two clusterings C and C' from the same dataset D. Table 2.1 shows the structure of a contingency table with K clusters in C, J clusters in C' and N items in D where n_{ij} is the count of items being in clusters $c_i \in C$ and $c'_j \in C'$.

	c_1	c_2	...	c_K	Σ
c'_1	n_{11}	n_{12}	...	n_{1K}	$n_{1.}$
c'_2	n_{21}	n_{22}	...	n_{2K}	$n_{2.}$
...	...	...		...	...
c'_J	n_{J1}	n_{J2}	...	n_{JK}	$n_{J.}$
Σ	$n_{.1}$	$n_{.2}$	...	$n_{.K}$	N

Table 2.1: Structure of a contingency table

Different measures to compare clusterings have been proposed, which can be categorized into a handful of classes (Vinh et al., 2010). Pair-counting and set based comparison measures form together with information-theoretic measures the fundamental classes. Another class is based on edit distances similar to the *string metric*. According to Amigó et al. (2009), B^3 measures form an additional class.

2.4.4 Information theoretic similarity measures

Information theoretic similarity measures are based on a solid mathematical foundation from information theory (Cover and Thomas, 2006) and are able to work with non-linear similarities. They have become popular by the works of Strehl and Ghosh (2003) and Meilă (2005). Meilă (2003, 2005, 2007) has shown many desirable properties for the *variation of information* measure. Mutual information has been used as clustering

comparison measure in Banerjee et al. (2005) and many normalized variants of mutual information have been proposed by Kvalseth (1987), Strehl and Ghosh (2003), Yao (2003), and Liu et al. (2008).

Generally desirable properties of clustering comparison measures include *metric property*, *normalization* and a *constant baseline property* (Vinh et al., 2010). For a measure to be a true metric, it must satisfy the *positive definiteness* and *symmetry* properties as well as the *triangle inequality*. In order to be normalized, a metric must output a value within a fixed range, e.g. $[-1, 1]$ or $[0, 1]$. The constant baseline property requires that the measure outputs a constant value, preferably zero, when comparing two independent, random clusterings. Meilă (2005) and Amigó et al. (2009) list many more properties or specific constraints and argue that a best general measure for all tasks does not exist.

Mutual information measures the information shared between two clusterings C and C'. A higher MI signals a greater help in predicting the cluster labels in C with information from C'. Several normalized variants can be derived from the mutual information:

$$NMI_{joint} = \frac{I(C, C')}{H(C, C')} \tag{2.14}$$

$$NMI_{max} = \frac{I(C, C')}{\max(H(C), H(C'))} \tag{2.15}$$

$$NMI_{sum} = \frac{2I(C, C')}{H(C) + H(C')} \tag{2.16}$$

$$NMI_{sqrt} = \frac{I(C, C')}{\sqrt{H(C)H(C')}} \tag{2.17}$$

$$NMI_{min} = \frac{I(C, C')}{\min(H(C), H(C'))} \tag{2.18}$$

They are all bounded in $[0, 1]$, equaling 0 when two clusterings share no information at all, i.e. are fully independent and 1 when two clusterings are identical. They differ in their normalization factor, i.e. the chosen upper bound of the MI. Vinh et al. (2010) prove that only two of unit-complements, i.e. the distance measures $1 - NMI$, are metrics. The normalized variation of information $NVI_{joint} = 1 - NMI_{joint}$ and the normalized information distance $NID_{max} = 1 - NMI_{max}$ are metrics. None of these conform to the *constant baseline property* and when achieving this by an adjustment for chance, they are not metrics anymore. Vinh et al. (2010) show via experimental results that an adjustment for chance of the NMI is not necessary when the number of item N is large (factor $\gtrsim 100$) compared to the number of clusters K and J. Between NMI_{max} and NMI_{joint} the former is preferable as it uses the range $[0, 1]$ better due to the tighter upper bound used for normalization.

In this chapter, the necessary theoretical background for the contributions of this thesis has been explained. In the next chapter, other work is presented that is related to the research question of this thesis. First, current approaches to multi-task learning for sequence tagging are introduced. Second, related work regarding the effect of auxiliary tasks and their similarity is presented.

3 Related work

3.1 Multi-task learning for sequence tagging

Multi-task learning (MTL) has been successfully used for various sequence tagging tasks (Søgaard and Goldberg, 2016; Bjerva et al., 2016; Plank et al., 2016; Martínez Alonso and Plank, 2017; Kaiser et al., 2017; Bingel and Søgaard, 2017; Augenstein and Søgaard, 2017; Kim et al., 2017; Yang et al., 2017; Changpinyo et al., 2018; Liu et al., 2018; Schulz et al., 2018). All of these approaches use hard parameter sharing in hidden layers. The common architecture consists of shared word plus character embeddings, bidirectional gated RNN and task-specific linear transform output layers.

Changpinyo et al. (2018) experiment with two additional architectures: They include a task embedding in either the RNN or output layer and merge the labels of all tasks allowing the output layer to be shared as well. Either bidirectional LSTMs or GRUs are used in the hidden RNN layers, which are deep, stacked RNNs of two or three layers in about half of the implementations. Yang et al. (2017) test the effect of sharing different layers in three models. Character RNNs are always shared. Additionally, word RNNs and finally the CRF is shared. Another architecture is proposed by Kim et al. (2017) who use two bidirectional LSTMs in parallel. While one is shared for all tasks, the other is task-specific. In most cases the character features are produced with another bidirectional gated RNN, only Bjerva et al. (2016) use a convolutional neural network instead. Kaiser et al. (2017) use a significantly more complex architecture as they work not only on sequence tagging tasks but also sequence generation and sequence-to-sequence tasks. Most word embeddings are initialized from pre-trained GloVe (Pennington et al., 2014) embeddings and fine-tuned during training. Plank et al. (2016), Martínez Alonso and Plank (2017), Bjerva (2017), and Ruder et al. (2019) decided against any form of pre-trained word embeddings to avoid any influence on the comparison of singe-task learning (STL) and MTL performance. In most approaches, the task specific output layers are based on a linear transformation to the tag space followed by the softmax function. Cross-entropy is used as loss function in these cases. Instead, Changpinyo et al. (2018), Liu et al. (2018), Yang et al. (2017), and Schulz et al. (2018) use a conditional random field to further improve performance on tasks with dependencies between labels. Søgaard and Goldberg (2016) do not only use the last hidden layer as input for the task-specific output layers. Instead, each RNN layer is associated with another task. Ruder et al. (2019) propose *sluice networks* that can represent hard parameter sharing, task-specific supervision of hidden layers (Søgaard and Goldberg, 2016) or cross-stitch networks (Misra et al., 2016) by learning which layers to share.

Training is in most scenarios performed by randomly choosing a task, sampling either

a single training instance or a mini batch and computing the loss for the chosen task. Changpinyo et al. (2018) sample a mini batch in a balanced manner with an equal number of training examples from all tasks. The number of training epochs is fixed in some cases while Bjerva et al. (2016), Søgaard and Goldberg (2016), Changpinyo et al. (2018), Liu et al. (2018), and Ruder et al. (2019) use early stopping based on a development set. In most approaches main and auxiliary tasks are seen as equally important, but Bjerva et al. (2016) use differently weighted losses for main resp. auxiliary task. The hyperparameters are fixed in almost all approaches, i.e. MTL uses the same parameters as STL. No hyperparameter optimization like grid or random search is performed. Schulz et al. (2018) train 50 hyperparameter combinations for STL and each MTL setting.

In the previously mentioned papers, the authors worked on over 16 different sequence tagging tasks in some multi task learning scenario. The three most common tasks are *part-of-speech tagging*, *syntactic chunking* and *named entity recognition*. The majority combined a main task with a single, supervised auxiliary task that comes from another dataset. Changpinyo et al. (2018), Kaiser et al. (2017), and Ruder et al. (2019) tried not only pairwise auxiliary tasks but also using all auxiliary tasks simultaneously. Liu et al. (2018) used language modeling as an unsupervised auxiliary task. Others (Plank et al., 2016; Martínez Alonso and Plank, 2017; Schulz et al., 2018; Kim et al., 2017; Yang et al., 2017) compared different datasets of the same task that vary in tagset or language.

Søgaard and Goldberg (2016) see improvements for both chunking and CCG tagging if additionally using the POS tags from the same dataset. Chunking also improved for auxiliary POS tagging from another dataset. Semantic tagging was used as auxiliary task for POS tagging by Bjerva et al. (2016) and improved performance. Plank et al. (2016) experimented with POS tagging across 22 languages. They found that using extra training data from other languages increases performance especially on rare words. Bingel and Søgaard (2017) found that some tasks increase performance when using any auxiliary task while others always decrease performance in an MTL setting. Martínez Alonso and Plank (2017) tried to boost performance on five higher-level tasks with four different low-level syntactic tasks. Most effects of MTL were negative.

Kim et al. (2017) performed MTL on POS tagging across 14 languages. In almost all cases, they observed significant performance improvements on the main task. They found that the similarity of the languages is important as the performance increased stronger for languages from the same family. Augenstein and Søgaard (2017) tested one main task with two different datasets combined with five auxiliary tasks. Performance of one main dataset was improved by all auxiliary tasks while the other dataset mostly showed decreasing effects in the MTL setting. Yang et al. (2017) worked on the three most common tasks with artificially reduced training datasets. MTL was able to increase the main score in all cases. The highest improvements were achieved when sharing all layers (character and word RNN as well as CRF) of the network. They attribute the degree of performance increase to three aspects: (1) label abundance for the main task, (2) task resp. dataset similarity and (3) number of shared parameters.

Changpinyo et al. (2018) compared eleven tasks and observed mixed results from MTL over STL. Using all auxiliary tasks instead of only one was better overall. They found some tasks increase the main task performance in almost all cases while tasks with a

small tagset usually decrease the main task performance. All the results were similar across their three tested MTL architectures. The unsupervised language modeling task of Liu et al. (2018) increased performance on POS, chunking and NER. Schulz et al. (2018) combined argumentation mining training data with datasets of the same task. Performance significantly increased over STL in four out of five cases, especially with scarce training data. Ruder et al. (2019) try their proposed *sluice networks* across several domains on chunking, NER and semantic role labeling with POS tagging as auxiliary task. MTL significantly outperforms STL in 17 of 21 cases and improves over other MTL architectures in 15 of 17 comparable cases.

3.2 Effect of auxiliary task similarity

Auxiliary tasks can have various relationships to the main task. In theory, adversarial learning, hints, focus attention, input prediction and representation learning can be used to connect an auxiliary task to the main task (Ruder, 2017). When choosing an additional task, a typical idea is to select a task that is considered helpful for a human, e.g. POS should help NER because most named entities are nouns. The assumption is that a task related to the main task helps in predicting.

In practice, the most common choice is to use a somehow related resp. similar task. When working with natural languages, task relatedness might be meant in linguistic sense. Caruana (1997) argues that tasks are similar if the same features are used for making predictions. Baxter (2000) suggest similar tasks must have the same inductive bias. Ben-David and Schuller (2003) indicate that tasks generated from the same probability distribution, with arbitrary transformations applied, are similar and perform well in a MTL setting. As of now, there is no universal measure for task similarity, but such a measure is needed to know which task should be preferred for training (Ruder, 2017).

Although multi task learning is applied frequently in recent work, few elaborate on the effect of task and dataset similarity on the main task performance. Recent work on MTL with deep neural networks found different hints indicating task similarity, but they are only applicable to each specific scenario. Martínez Alonso and Plank (2017) show results that auxiliary tasks with few labels and a uniform label distribution perform better for MTL in neural sequence tagging. Auxiliary tasks having many labels or high entropy harm the performance on the main task. While these findings are confirmed by Ruder et al. (2019), mixed results are reported by Bingel and Søgaard (2017). Bjerva (2017) found no evidence of label entropy correlating with MTL performance gains or losses. Martínez Alonso and Plank (2017) found a significant difference between two POS datasets when used as an auxiliary task. Converting an auxiliary dataset to another tagset changes the effect of MTL significantly.

Kim et al. (2015) propose a method to map labels from similar auxiliary datasets to the target tagset or vice versa. They use label embeddings induced by canonical correlation analysis to reduce a case of non-matching labels to a simple case of identical tagsets. This allows framing a MTL scenario as single task learning with an increased amount of training data. Further, they propose a method to find the nearest domain

resp. dataset based on the tagsets: Fine-grained labels are reduced to general coarse labels and these are counted in each dataset. The lowest l_2 distance between pairs of source and target multinomial distribution of the counts is chosen.

Bingel and Søgaard (2017) examine prediction of MTL gains and losses from dataset and STL learning curve features. Used dataset features include e.g. size, number of labels, label entropy and out of vocabulary rate in GloVe embeddings. From STL two features are extracted: Curve gradients at fixed percentages of the overall training instances and parameters of a fitted log-curve describing the steepness of the learning curve. Logistic regression with binary classification is run on the features normalized to the range $[0, 1]$ to predict performance increase or decrease. Because correctly predicting the relationship and magnitude failed, a binary classification was used instead of a continuous output. The classification output was correct for three out of four cases. The learning curve features were most important — dataset features alone were not enough to make good predictions. From the dataset features, the number of labels on the main task and the auxiliary label entropy showed the most predictive potential. The differences in dataset size were insufficient features in the logistic regression model. Bingel and Søgaard (2017) summarize that performance improves when the training error surface on the main task reaches a plateau while the auxiliary task does not.

Bjerva (2017) estimates the effect of an auxiliary task in MTL with information theoretic measures. He correlates label entropy, conditional entropy and mutual information with change in accuracy compared to STL. POS tagging is used as the main task and various dependency relation tasks are chosen to be auxiliary data. Computing the joint probabilities on the labels requires the datasets to be tagged with gold standard labels for each task. Therefore, tasks must be automatically taggable with almost perfect results. For POS tagging this certainly is the case and according to Bjerva (2017) the difference in the information theoretic measures between dual gold annotated data and automatic POS tags is negligible. Averaged Spearman correlations between the three information theoretic measures and the accuracy change are 0.07, 0.26 and 0.42 for entropy, conditional entropy and mutual information. Further experiments were performed on the semantic task results of Bjerva et al. (2016) and Martínez Alonso and Plank (2017) indicating that mutual information for helpful auxiliary tasks is higher than for harmful tasks.

Augenstein et al. (2018) propose an architecture that learns label embeddings for natural language classification tasks. Their analysis indicates that these label embeddings give hints regarding the gains or harms of MTL.

Ruder et al. (2019) correlate task properties with performance differences and learned meta network parameters of the sluice network. They find that (1) „multi-task learning gains, also in sluice networks, are higher when there is less training data“ and (2) „sluice networks learn to share more when there is more variance in the training data“.

In the next chapter, new methods to measure the similarity of datasets will be designed.

4 Dataset similarity concepts

Based on the related work presented in the previous chapter, new approaches to compute the similarity of sequence tagging datasets will be developed throughout this chapter.

4.1 Hypotheses

Most previous work (see Section 3.1, page 29) used related tasks and datasets to boost performance on the main task. The implicit assumption underlying the most common choice of auxiliary training data can be formulated as the following hypothesis.

Hypothesis 1: *Auxiliary data being more similar to the main training data results in a stronger main task performance compared to unrelated auxiliary data.*

Note that using related auxiliary data does not necessarily increase the performance on the main task, but it also might affect the main task performance less negatively than unrelated auxiliary data. Moreover, it certainly is not an absolute rule as outliers are always a possibility. However, the hypothesis should hold true in the majority of cases. In order to use Hypothesis 1, *dataset similarity* and *main task performance* have to be measured. The latter is measured identical to the single-task learning scenario without using any auxiliary data. The performance measurement is typically defined by the specific task and performed on a holdout test set. Common performance measurements are accuracy, precision, recall and the F_1 score, which is the weighted harmonic mean of precision and recall (Derczynski, 2016).

Dataset similarity describes the relatedness between two or more datasets. There are many dimensions in which a dataset can be related to another dataset. Two datasets could be considered related based on their content when they share the same language or refer to the same topic etc. For labeled datasets, the labels need to be taken into account as well. While two datasets might agree on the same words and labels, the actual pairing of words and labels could be e.g. inverted or entirely unrelated. For this thesis, dataset similarity shall include all these properties. As a universal measurement for dataset similarity does not exist (see Section 3.2, page 31), a suitable dataset similarity measure will be developed during this chapter. The similarity of two datasets should only depend on data-inherent properties to be generally applicable. Assuming this is possible, one can come up with the following hypothesis.

Hypothesis 2: *The similarity of two datasets D_1 and D_2 can be measured independently of a specific machine learning model.*

In the general case, the training data for any sequence tagging task consists of a sequence of tokens, possibly divided into segments or sentences, and any number of labels per token. The tokens and corresponding labels are the obvious candidates for extracting a similarity signal from the data. Labels or their distributions and embeddings seem to have some signal as many found respective hints or even correlations in their experiments (Kim et al., 2015; Martínez Alonso and Plank, 2017; Bjerva, 2017; Bingel and Søgaard, 2017; Augenstein et al., 2018; Ruder et al., 2019). Both tokens and labels on their own are not enough for measuring dataset similarity (Bjerva, 2017). Imagine comparing a dataset with a transformed version of itself. While the original is left untouched, in the other labels or tokens are randomly shuffled. Measuring token and label similarity independent of each other via entropy of token resp. label distribution results in identical values for both the original and shuffled dataset. While a machine learning model can learn from the former, it can only try to model random noise for the latter. Because learning something random is impossible, original and shuffled transformation should not be similar. Consequently, the idea further pursued in this work is to compare *pairs* of token and label in different datasets in order to measure dataset similarity. Bjerva (2017) was the first to use the joint probability of tokens and labels, which requires token-label pairs, but he obtained the pairs from a single dataset annotated with two tagsets in parallel. To my knowledge, this is the first attempt to compare token-label pairs across *different* datasets.

4.2 Requirements

The requirements for a solution to measure the similarity of datasets are derived from the goal of comparing sequence tagging datasets with each other without any further restrictions. Working on raw data of arbitrary sequence tagging tasks is necessary for this goal and to comply with Hypothesis 2. In addition, the way to measure dataset similarity must be independent of the model that will be using the datasets for training. Comparing only the same dataset with multiple labels per token would be a severe limitation. A comparison „without any further restrictions“ implies that there are no restrictions on tokens or labels in the data. This allows comparing datasets that have a different tagset with or without overlap, are written in another language or in no natural language at all.

While Bjerva (2017) limited comparison to automatically taggable tasks, the solution developed in this work is not restricted to automatically taggable tasks. Labeling a dataset for comparison has two major drawbacks. (1) The majority of tasks is not automatically taggable with sufficient results, i.e. almost indistinguishable from the gold standard annotation. (2) The similarity of the compared datasets is now dependent on a specific model. Thus, the results are not universally applicable anymore. Instead, a solution has to work with the information given in the compared datasets.

Apart from requirements regarding input or processing, the properties of the similarity measure itself have be considered. The output, i.e. the similarity score or value, should be minimal when two datasets are completely unrelated (e.g. one of them is entirely random).

Comparing identical datasets has to result in a maximal similarity score. Both minimal and maximal value have to be independent of the datasets, e.g. the similarity score is normalized to a value in the range $[0, 1]$. This normalization is crucial to compare one main dataset with a number of auxiliary datasets because it enables a direct comparison of the absolute scores. Otherwise, a similarity score of 10 for one dataset might in fact be lower than a score of 0.5 for another if the maximal scores for datasets are 100 respectively 1. While the comparison of highly related but not identical datasets should be close to the maximal similarity score, the opposite should be true for almost unrelated datasets.

Other important aspects are the time and resources it takes to obtain a similarity score for two datasets. It is only useful if it needs a fraction of the time and resources used for training the machine learning model with multi-task learning on both datasets. While this requirement is certainly vague and dependent on the applied machine learning model, the method of obtaining the similarity score has to be generally considered fast and cheap in the machine learning community.

4.3 Framing label similarity as a clustering comparison problem

Transforming the problem of token-label dataset similarity to a clustering comparison problem has a great benefit: Existing ideas and algorithms from clustering comparison with solid theoretical foundations can be reused for this new problem. The idea is quite straightforward: A clustering represents one label set and each label is a cluster within the clustering. This means that all tokens having the same label belong to one cluster.

As shown in Section 2.4.3, a contingency table is a handy tool to work on the comparison of clusterings. For now, let us assume that a dataset D is annotated with two labels in parallel from two tasks T and T' with possibly different label sets L and L'. Converting the simplified problem of comparing L with L' on D to a clustering comparison can be done in the following way. The clusters for T are the labels $l_1, l_2, \ldots, l_N$ when the label set L has N different labels in total. The clusters for T' are labeled analogously $l'_1, l'_2, \ldots, l'_M$ for the M labels in the set L'. Table 4.1 shows the resulting contingency table for the described setting. The values c_{xy} are the counts how many tokens are in the dataset that are labeled as / belong to cluster l_x in task T and simultaneously l'_y in the task T'.

To make the described approach less abstract, an example is provided below. Let the dataset D use named entity recognition (NER) as task T and part-of-speech (POS) tagging as task T' having the label sets[1]:

$$L = \{\text{organization (ORG), person (PER), location (LOC), other (OTH)}\}$$
$$L' = \{\text{noun (NN), verb (VB), determiner (DT), other (X)}\}$$

[1] A simplified variant of the typical NER tagging scheme is used for ease of demonstration: Beginning and intermediate tags are combined, e.g. B-ORG and I-ORG are reduced to ORG.

	l'_1	l'_2	...	l'_M	Σ
l_1	c_{11}	c_{12}	...	c_{1M}	$c_{1.}$
l_2	c_{21}	c_{22}	...	c_{2M}	$c_{2.}$
...	...	...		...	...
l_N	c_{N1}	c_{N2}	...	c_{NM}	$c_{N.}$
Σ	$c_{.1}$	$c_{.2}$	...	$c_{.M}$	c

Table 4.1: Contingency table for a comparison of label sets L and L' with N resp. M unique labels

	NN	VB	DT	X	Σ
ORG	3	0	0	0	3
PER	2	0	0	0	2
LOC	2	0	0	0	2
OTH	3	2	2	2	9
Σ	10	2	2	2	16

Table 4.2: Counts from example Sentences 4.1 and 4.2 for comparison of NER and POS tagsets

Let dataset D contain the following two sentences:

(4.1)

ORG	ORG	ORG	OTH	OTH	OTH	OTH	PER	PER
NN	NN	NN	VB	DT	NN	NN	NN	NN
Walt	Disney	Productions	created	the	cartoon	character	Donald	Duck

(4.2)

LOC	OTH	OTH	OTH	OTH	OTH	LOC
NN	VB	DT	X	NN	X	NN
Hamburg	is	a	large	city	in	Germany

Table 4.2 shows the contingency table filled with the counts from both example sentences. The last row resp. column shows the sum of the counts in each column resp. row. The count $c_{\text{ORG,NN}}$ is three because there are exactly three tokens (*Walt Disney Productions*) tagged both ORG and NN. Other label-pairs are derived analogously from the remaining tokens of the dataset D.

Based on the counts in the contingency table, it is straightforward to calculate information theoretic measures such as entropy, joint entropy, conditional entropy (see Section 2.4.1, page 26) or mutual information (see Section 2.4.2, page 26). For ease of reading, the definition of mutual information (MI) is provided here once more:

$$I(X;Y) = \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} p(x,y) \log_2 \left(\frac{p(x,y)}{p(x)p(y)} \right) \tag{4.3}$$

Because the (joint) probability mass functions $p(x)$, $p(y)$ and $p(x,y)$ are unknown for the label sets L and L' in dataset D, the equation needs to be rewritten to use an approximation. The probabilities are replaced by the relative frequencies of the label-pairs. With the notation from Table 4.1, the MI definition becomes

$$I(L;L') = \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{c_{ij}}{c} \log_2 \left(\frac{\frac{c_{ij}}{c}}{\frac{c_{i.}}{c}\frac{c_{.j}}{c}} \right) = \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{c_{ij}}{c} \log_2 \left(\frac{c_{ij}c}{c_{i.}c_{.j}} \right). \tag{4.4}$$

The different entropy measures can be changed analogously to use relative frequencies instead of probability mass functions. Note that the logarithm is only defined for positive values, but the counts c_{ij} are often zero. In line with Cover and Thomas (2006, p. 14), the convention $0 \log(0) = 0$ is used to mitigate this issue because $x \log(x) \to 0$ when $x \to 0$. To complete the example, calculating the mutual information for the data in Table 4.2 is shown below:

$$\begin{aligned} I(L; L') &= \frac{3}{16} \log_2 \left(\frac{3 \cdot 16}{3 \cdot 10} \right) + \cdots + \frac{2}{16} \log_2 \left(\frac{2 \cdot 16}{9 \cdot 2} \right) \\ &= 0.1875 \log_2(1.6) + \cdots + 0.125 \log_2(1.\overline{7}) \\ &= 0.1271 + \cdots + 0.1038 \\ &= 0.4379 \end{aligned}$$

So far, transforming label similarity as a clustering comparison problem has been limited in application due to the need of parallel labels for two tasks. Solutions on how to lift this restriction will be developed in Section 4.5. To answer whether $I(L; L') = 0.4379$ means that the two tasks are similar or not, the entropy of both label sets has to be taken into account. Different measure variants are compared in the following section to find a useful similarity score. This is required regardless of the current limitation as the idea of filling a contingency table to compute clustering comparison measures remains the same.

4.4 Evaluation of clustering comparison measures

As there are various information-theoretic clustering comparison measures available, it is beneficial to find that particular measure which is most suitable for the problem of dataset similarity estimation. In Section 4.3 is shown how to transform the dataset comparison to a clustering comparison problem. Multiple information-theoretic measures and some of their properties were introduced in Section 2.4. Together, both form the foundation to define a comparison measure that fulfills the requirements set in Section 4.2.

In previous work (see Section 3.2, page 31) entropy, conditional-entropy and mutual information have been applied to compute label, dataset and task similarity. To allow for a comparison, these measures are included in the upcoming evaluation process. Apart from those basic measures, Section 2.4.4 introduced more sophisticated measure candidates that are constructed from the simpler ones. Two normalized variants of the plain mutual information are to be compared against the other similarity measures as they posses the best theoretical properties. NMI_{max} and NMI_{joint} can both be converted into a proper distance metric.

The selected measurements will be evaluated both in theory and in practice on a few simple examples. Hands-on evaluation will be performed on task-agnostic contingency tables filled with counts to illustrate certain relationships of the two label sets. The measure-specific requirements are enumerated below to enable an easier reference hereafter.

(1) The minimal and maximal value of the measure should be in a fixed range independent of the dataset.

(2) Identical label sets should result in a maximal similarity score.

(3) Unrelated label sets, e.g. random, should have a minimal similarity score.

(4) Label sets that can be mapped directly to each other with only few errors should have a high similarity score close to maximal value.

(5) If label sets are not related but still not random, their similarity score should be close to the minimum.

(6) When two label sets are partly identical and random, the similarity score should be an average value in the middle of the minimum and maximum.

Entropy, conditional entropy and mutual information cannot be negative and always have 0 as their minimum value. Hence, they fulfill a part of the fixed range requirement req:fixed-range. The maximal values of both conditional entropy and mutual information depend on the entropy of the label sets. Because the maximal entropy value is dependent on the number of labels $H(L) \leq \log_2(N)$, conditional entropy and mutual information do not have a dataset-independent maximum value. Consequently, these three measures do not provide a fixed range similarity score and fail to fulfill requirement (1). As the NMI variants are normalized to the fixed range of $[0, 1]$, they satisfy requirement (1).

To examine the measures regarding the requirements (2–6), a number of scenarios, presented in Table 4.3, are used to evaluate the outputs of the different measures. The count distribution for the first three examples (a–c) is easily justified. In (a) each label l_x is mapped to exactly one label l'_y. There are no outliers. The counts for the second example (b) were drawn randomly from a uniform distribution. In (c) an identical mapping (diagonal line from l_1, l'_1 to l_4, l'_4) was used as the initial situation. A few outliers with significantly lower counts were added to transform the perfect relation to a highly related scenario. For example (d) one label-pair is an exact match between both label sets, but the other three are evenly distributed. The counts should reflect that most label information is unrelated while there is still some similarity signal. The last example (e) has different label set sizes. While labels from L are split across multiple corresponding labels in L', in the other direction there are good fits with only few outliers.

The results for each combination of information-theoretic measure and scenario are shown in Table 4.4. As entropy and conditional entropy are not symmetric, results for both directions are shown. It becomes clear immediately from the results that entropy is an unsuitable measure to compare clusterings. The values are almost identical for all except one scenario. Due to the greater number of labels $H(L')$ is 50% higher than the other entropy values. Conditional entropy is not a similarity measure, but it might work as a distance measure so that requirements regarding minimal and maximal values have to be swapped. For identical (a) label sets the value is minimal and the random (b) scenario shows the highest conditional entropy. While $H(L|L')$ value for case (d) is placed correctly among the five scenarios, the remaining two cases (c, e) are not in line

	l'_1	l'_2	l'_3	l'_4	Σ
l_1	17	0	0	0	17
l_2	0	0	0	12	12
l_3	0	19	0	0	19
l_4	0	0	12	0	12
Σ	17	19	12	12	60

(a) Identical

	l'_1	l'_2	l'_3	l'_4	Σ
l_1	7	6	3	3	19
l_2	2	4	2	3	11
l_3	4	2	3	8	17
l_4	9	4	8	5	26
Σ	22	16	16	19	73

(b) Uniform random

	l'_1	l'_2	l'_3	l'_4	Σ
l_1	10	0	0	1	11
l_2	1	11	0	0	12
l_3	0	2	12	0	14
l_4	0	0	3	13	16
Σ	11	13	15	14	53

(c) Highly related

	l'_1	l'_2	l'_3	l'_4	Σ
l_1	15	0	0	0	15
l_2	0	5	5	5	15
l_3	0	5	5	5	15
l_4	0	5	5	5	15
Σ	15	15	15	15	60

(d) Rather unrelated

	l'_1	l'_2	l'_3	l'_4	l'_5	l'_6	l'_7	l'_8	l'_9	l'_{10}	Σ
l_1	10	0	0	20	0	0	0	0	0	2	32
l_2	0	12	12	0	0	0	0	0	1	0	25
l_3	0	0	0	4	2	15	0	5	0	0	26
l_4	1	0	0	0	1	0	10	0	14	0	26
Σ	11	12	12	24	3	15	10	5	15	2	109

(e) Medium related, different label set sizes

Table 4.3: Contingency tables for different scenarios comparing label sets L and L'

Scenario	$H(L)$	$H(L')$	$H(L\|L')$	$H(L'\|L)$	$I(L;L')$	NMI_{joint}	NMI_{max}
Identical (a)	1.97	1.97	0.00	0.00	1.97	1.00	1.00
Random (b)	1.94	1.99	1.84	1.89	0.09	0.02	0.05
Related (c)	1.98	1.99	0.55	0.55	1.44	0.57	0.72
Unrelated (d)	2.00	2.00	1.19	1.19	0.81	0.25	0.41
Mixed (e)	1.99	3.07	0.26	1.34	1.73	0.52	0.56

Table 4.4: Results for the evaluated measures on each scenario from Table 4.3

with the expected outcome. Their order should have been changed. The other direction $H(L'|L)$ has a similar issue because (e) should have a lower conditional entropy than (d). A combination, e.g. average, of the values from both directions could solve the issue in the shown examples. It is unclear whether this workaround would generalize well.

The results for mutual information look promising as it is clearly a similarity measure. While the random setting (b) has by far the lowest value, the score in the identical setting (a) is maximal. The values for scenarios (c, d) are close to the expected outcome. Similar to the result for conditional entropy, the order of settings (c) and (e) is contrary to the expected order.

Both evaluated NMI variants perform great as a similarity measure. Apart from being normalized to range of $[0, 1]$, they manage to produce scores for all five scenarios agreeing to the expected order. Starting with the lowest value of almost zero in the random case (b), both put the score of scenario (d) in the lower half. The mixed example (e) is placed correctly in the middle and followed by case (c). Both score the identical label sets (a) with the maximum value of 1.

The theoretical and experimental results of this section can be summarized as follows. Entropy is unsuitable and fails to fulfill any of the requirements. Conditional entropy (as a distance measure) and mutual information are usable, but both only satisfy four out of six requirements. Only the two NMI similarity measures are able to fulfill all six requirements and produce scores as expected for each example scenario. Which of these NMI measures is best suited in a real application cannot be answered from this basic evaluation. Both need to be compared on a larger scale.

4.5 Calculation of dataset similarity from labels

Having identified suitable similarity measures in the previous section, the question of how to obtain the necessary label-pair counts to fill a contingency table remains. The counts for each pair of labels allow to approximate the joint probabilities, which are essential for the computation of the similarity measures. Until now, obtaining such counts from labeled text has been restricted to a single dataset with parallel annotations for two label sets as shown in Section 4.3. This restriction has to be lifted to compare arbitrary sequence tagging datasets and tasks.

Bjerva (2017) applies automatic tagging to mitigate the issue. When two tasks from two different datasets have to be compared, one of the datasets has to be tagged automatically with the other task's labels. This effectively reduces the problem to the restricted case mentioned above. However, this approach has multiple issues. Most notably, a comparison is only possible if at least one of the tasks can be automatically tagged with almost perfect accuracy. Otherwise, the counts and thus the similarity score does not resemble the actual data at all. While the necessary performance-level (inter-annotator agreement) has been reached for a few simple tasks, the performance on the majority of tasks is insufficient. Another downside is that the similarity score is not only data-inherent anymore because it implicitly depends on the model used to perform the automatic labeling. Finally, a comparison of two datasets for the same task cannot be meaningfully performed because automatic tagging would remove the ability to compare the subtle differences between both datasets.

In this work, other approaches are proposed to lift those restrictions on the datasets and tasks. In the following subsections two solutions are developed that enable a comparison of arbitrary task and dataset combinations.

4.5.1 Text overlap

An idea to compare different datasets is to use a label mapping function. If a manually defined one-to-one mapping from labels of one dataset to another one exists, datasets can be compared to each other because the problem can once again be reduced to the easy case of a single dataset with two parallel label sets. The obvious issue with this idea is that in many cases there is no bijective label mapping function. While mapping a fine-grained label set to a coarse label set is certainly possible, the reverse is not easily done. It is unclear to which of the finer sub-labels a coarse label should be mapped. Moreover, the predefined label mapping function causes the similarity score to be dependent on the specific mapping function used.

The basic idea of the *text overlap* approach is to generate an implicit label mapping from the token-label pairs of both datasets. This has the advantage of being independent of external knowledge and enabling a probabilistic mapping from coarse to fine-grained label sets specific for both datasets. The method uses only the labels for words that are contained in both datasets.

Instead of comparing the label of each token in dataset D_1 with D_2, tokens occurring multiple times in a dataset are aggregated. Thus, the dataset in form of a stream of token-label pairs is transformed to a set of unique words. Each word is associated with the number of times it is contained in a dataset and the counts for each label that this word has been tagged with. Below are two example datasets annotated with the extremely reduced POS tagset introduced in Section 4.3.

(4.5)

VB	DT	NN	X	VB	DT	NN	VB	DT	X	NN	X
Creating	an	example	to	explain	the	process	is	an	impossible	task	.

X	VB	DT	NN	X	NN	X	NN	X	NN	VB	VB	X
To	process	the	data	,	counts	of	words	and	labels	are	needed	.

(4.6)

X	VB	DT	NN	X	DT	X	NN	X	DT	NN	X	VB	DT
This	is	the	data	for	the	second	dataset	.	The	process	to	find	the

X	NN	X	X	NN	VB	DT	NN	X
right	words	for	this	example	took	a	second	.

Table 4.5 shows the two datasets (4.5, 4.6) after the transformation. Only words occurring in both datasets can be used to fill in the counts of a contingency table in order to compute a similarity score for the datasets. Filling in the counts of the contingency table is not straightforward. There are multiple options to consider how exactly the counts from both datasets are fused together. It becomes even more difficult if a word does not have the same label within one dataset. In the examples above, this is the case for the words *process* and *second* because they are ambiguous without context. Table 4.6 shows two possible ways to combine the counts from both datasets into a single contingency table.

The additive (a) approach is based on the assumption that the problem can once more be reduced to the simple case of a single dataset with parallel labels. By looking only at the intersection of words contained in both datasets, a new virtual dataset is created. In

Word	#	DT	NN	VB	X
Creating	1	0	0	1	0
an	2	2	0	0	0
example	1	0	1	0	0
to	1	0	0	0	1
explain	1	0	0	1	0
the	2	2	0	0	0
process	2	0	1	1	0
is	1	0	0	1	0
impossible	1	0	0	0	1
task	1	0	1	0	0
.	2	0	0	0	2
To	1	0	0	0	1
data	1	0	1	0	0
,	1	0	0	0	1
counts	1	0	1	0	0
of	1	0	0	0	1
words	1	0	1	0	0
and	1	0	0	0	1
labels	1	0	1	0	0
are	1	0	0	1	0
needed	1	0	0	1	0

(a) Counts for words and their labels in Dataset 4.5

Word	#	DT	NN	VB	X
This	1	0	0	0	1
is	1	0	0	1	0
the	3	3	0	0	0
data	1	0	1	0	0
for	2	0	0	0	2
second	2	0	1	0	1
dataset	1	0	1	0	0
.	2	0	0	0	2
The	1	1	0	0	0
process	1	0	1	0	0
to	1	0	0	0	1
find	1	0	0	1	0
right	1	0	0	0	1
words	1	0	1	0	0
this	1	0	0	0	1
example	1	0	1	0	0
took	1	0	0	1	0
a	1	1	0	0	0

(b) Counts for words and their labels in Dataset 4.6

Table 4.5: Transformation of word-label pairs to an associated count-based representation. The grayed out words do not occur in both datasets.

this virtual dataset each word is tagged from two different label sets. Label counts for words contained in both datasets are summed up because they are viewed as multiple instances from a single dataset. The word *example* occurs once in each dataset and is both times tagged as *NN*. In the contingency table (a) the count for *NN–NN*, i.e. row 2 column 2, is increased by two. The word *the* occurs two resp. three times in the datasets and is always labeled *DT*. Consequently, the count in the contingency table at *DT–DT*, i.e. row 1 column 1, is increased by five. For *process* the situation is not that simple because it has multiple labels in the first dataset: *NN* and *VB*. In the second dataset, there is only a single occurrence of *process* with label *NN*. The counts in the contingency table are increased by two for the positions *NN-NN* and *VB-NN*. The problem is that the single occurrence is now used twice. Alternatively, the count could be split by the number of labels in the other dataset, so that the two affected positions are not increased

	DT	NN	VB	X	Σ
DT	5	0	0	0	5
NN	0	8	0	0	8
VB	0	2	2	0	4
X	0	0	0	6	6
Σ	5	10	2	6	23

(a) Additive

	DT	NN	VB	X	Σ
DT	6	0	0	0	6
NN	0	4	0	0	4
VB	0	1	1	0	2
X	0	0	0	5	5
Σ	6	5	1	5	17

(b) Multiplicative

Table 4.6: Contingency table derived from the counts of words in Table 4.5 that are contained in both datasets.

by two but by one and a half.

Another idea is to use a multiplicative (b) procedure by combining the counts for matching words via multiplication instead of addition. The words *example* and *the* increase the counts in the contingency table (b) at the positions *NN–NN* resp. *DT–DT* by $1 \cdot 1 = 1$ resp. $2 \cdot 3 = 6$. For words with multiple labels like *process*, the counts for each label-combination are multiplied and added at the corresponding position in the contingency table. An effect of this approach is that words being frequent in both datasets contribute more to be counts. Whether this effect is desirable or not has to be evaluated in a large scale experiment on real data. From a theoretical point of view, it has the benefit that matching words, which are often seen in both datasets, are very unlikely outliers and can aid in a robust comparison. However, overweighting frequent words is usually not helpful for data understanding because common words are almost identical in each dataset of the same language.

There are more possible weighting schemes how to combine the raw counts from the different datasets into a mutual contingency table. In any case, all contingency tables obtained from these methods can be used to compute similarity measures such as the NMI as shown in Section 4.4. To decide which approach yields the best results with regard to the use case, an extensive end-to-end experiment including similarity score calculation should be performed.

The general advantage of the approaches proposed here is that they are fast because they only involve text processing and a few counts. The downside is that these methods can only identify an identical dataset with 100% similarity if each word always has the same label. In real datasets many words occur with different labels multiple times both erroneously and because they are ambiguous. This will very likely affect the similarity comparison negatively. Another issue is that only a fraction of each dataset is used for the actual comparison. Information from words occurring only in one of the datasets cannot be leveraged. Whether the comparison of real-world datasets is affected (negatively) by these shortcomings, will be evaluated experimentally in Chapter 7.

4.5.2 Vector space similarity

In Section 2.2.6, word embeddings were presented. They allow representing words in form of dense vectors within a vector space instead of a specific character sequence in the language's vocabulary. Thus, it is possible to perform mathematical operations on these vectors and compute e.g. the semantic similarity of two words by computing their cosine similarity within the vector space (Elekes et al., 2017). These word vector techniques can be used to tackle the problems of the previously shown text overlap approach.

A first extension allows incorporating words that do not occur in both datasets into the comparison process. Identical to the text overlap approach, each dataset is transformed into an associative array of words with the occurrence counts for each label. The unique words from each dataset are embedded to obtain vector representations. Instead of ignoring words only contained in one dataset, the most similar word from the other dataset is chosen for the pairwise label comparison. An issue is that some words may be used multiple times because each word can have one direct text match and multiple similar matches via embeddings. This further complicates the necessary weighting and combination schemes presented in the previous section. The remaining process and similarity measure computation stays the same.

Applying this extension to the two example datasets shown in Table 4.5 would make use of the grayed out words. *Creating* from Dataset 4.5 might have the closest match with *process* from Dataset 4.6 depending on the word embeddings used. Thus, the count for *VB–NN* would be increased, which clearly is a mismatch. The word *an* might have the lowest vector space distance to *a* from the other dataset. This accurate match would increase the count for *DT–DT*. The remaining, so far unused, words from Dataset 4.5 have to be matched with their semantically most similar counterparts from Dataset 4.6. For each pair of words, the count for the corresponding label-pair needs to be increased in the contingency table. While most vector representation matches between those two example datasets are arguably inadequate, the quality of these matches is considerably higher with proper, large datasets.

A more radical approach allows differentiating between the various labels of the same word depending on its context. Instead of working on the aggregated list of unique words, the raw datasets are used. For each token, a specific vector representation is obtained via contextual embeddings such as *ELMo* (Peters et al., 2018) or *BERT* (Devlin et al., 2019). The counts in the contingency table for the label-pair can either be incremented by 1 or by the vector space similarity of the two tokens. A similarity measure like NMI can be calculated from these counts as before. Identical datasets can be scored with 100% similarity when the contextual embeddings are able to produce unique vector representations for each token deterministically. In general, this method handles ambiguity in a language much better compared to the plain text approach, which should help to improve the similarity comparison between various datasets.

The application of this approach on the two example Datasets 4.5 and 4.6 would work in the following way. All tokens in the two datasets are augmented with their corresponding contextual vector representations, thereby creating an associative array from a numeric vector to a label. Tables 4.7a and 4.7b show the result for the first five

Token	Embedding	Label
Creating	$[0.01, 0.14, 0.03, \dots]$	VB
an	$[0.23, 0.07, 0.01, \dots]$	DT
example	$[0.01, 0.02, 0.22, \dots]$	NN
to	$[0.08, 0.19, 0.05, \dots]$	X
explain	$[0.02, 0.13, 0.03, \dots]$	VB

(a) First five tokens from Dataset 4.5 with their vector representations and labels

Token	Embedding	Label
This	$[0.01, 0.14, 0.03, \dots]$	X
is	$[0.23, 0.07, 0.01, \dots]$	VB
the	$[0.01, 0.02, 0.22, \dots]$	DT
data	$[0.08, 0.19, 0.05, \dots]$	NN
for	$[0.02, 0.13, 0.03, \dots]$	X

(b) First five tokens from Dataset 4.6 with their vector representations and labels

	This	is	the	data	for
Creating	0.87	**0.23**	0.76	0.50	0.61
an	0.40	0.45	**0.32**	0.68	0.72
example	0.53	0.86	0.59	**0.37**	0.60
to	0.77	0.43	0.63	0.84	**0.40**
explain	0.89	**0.33**	0.45	0.64	0.72

(c) Pairwise distances between the vector representations of tokens from (a) and (b). Closest distances from rows to columns are shown in bold.

Table 4.7: Using vector space similarity to match tokens between two datasets to obtain counts for a similarity calculation based on a contingency table. Note that both embeddings and distance numbers are artificially chosen to be intuitively comprehensible and help to explain the various cases.

tokens of Dataset 4.5 resp. 4.6 using artificially constructed embeddings. For each word embedding in the first dataset, the vector representation with the closest distance from the other dataset has to be found. This is shown in Table 4.7c. The five matches are *Creating–is*, *an–the*, *example–data*, *to–for* and *explain–is*. Consequently, the counts in a contingency table have to be increased for the label-pairs *VB–VB*, *DT–DT*, *NN-NN*, *X–X* and *VB–VB*. Because the process of selecting the closest vector representation either from rows to columns or vice versa can result in different combinations, the counts in the contingency table will be different depending on the direction. Thus, for a symmetric similarity measure like NMI, two possibly different scores will be produced. When using a similarity measure that is not symmetric such as conditional entropy, this would result in four scores per dataset comparison.

The first extension to use words not occurring in both datasets solves the smaller problem of the two issues with the text overlap method. Compared to the raw text processing without any word embeddings, it requires a lot more computational resources to work with the vector representations. The upside is that obtaining context-independent vector representations from words is fast. A fast and simple lookup table mapping a

word to its vector is enough. More importantly, the number of unique words within any dataset is small compared to the number of tokens according to Zipf's law (Zipf, 1935). Therefore, increasing dataset sizes has almost no effect on the amount of expensive vector representation computations. The situation completely changes when contextual embeddings are used for each token. Matching single tokens instead of using aggregated counts requires many expensive word vector computations because every token from one dataset has to be compared to every token in the other dataset to find the closest match. The benefits of this contextual approach are achieved by a severe increase in computational costs.

4.6 Calculation of dataset similarity from gradients observed during training

Alternatives to the methods described previously in this chapter will be outlined in this section. Instead of calculating dataset similarity from raw text tokens and labels, it could be approximated from gradients during training. This is an entirely different approach compared to the previous work in this chapter because it is not based on the datasets alone but uses the training process of a neural network as information source. Note that this approach will not be further pursued in this thesis as it does not meet the requirements set in Section 4.2. The ideas are still described here in case of having a similar problem but different requirements than assumed in this work.

The basis for this approach is the assumption that related tasks or datasets are solved with similar solutions. In case of neural networks, similar solutions are described by parameters i.e. weights being close to those of another network. To arrive at a similar parameter solution, the training gradients used to update the parameters have to be very similar as well. Note that actually the reverse is true, i.e. the same gradients must always lead to the same solution (assuming other factors such as the learning rate are constant). In practice, it is highly unlikely to arrive at the same solutions within a high-dimensional parameter space when following different gradients. In any case, highly similar gradients will result in a similar solution, which in turn indicates that two datasets are related.

A first idea working under the assumption above is to train two identical models separately for both datasets. In order to enable a fair comparison of the training gradients, the models have to be identical with respect to both hyperparameters and initial network weights. During the training process, gradient vectors obtained on the main dataset can be compared to those from the auxiliary dataset. It is possible to compare the gradient vectors of every training iteration, their running average or fixed points similar to Bingel and Søgaard (2017), who selected the gradients at 10, 20, ... percent of the total training process. Gradient vectors can be compared by computing e.g. their cosine similarity, which will only compare the direction but not the magnitudes of both vectors. Additionally, other features such as the steepness may be used for a comparison.

Another idea is to use a single model and train it in an alternating manner with both

auxiliary and main data. Training on single sentences usually results in inconsistent gradients. Thus, (mini-)batches of each dataset should be used to obtain more consistent gradients. These gradient vectors can again be compared with cosine similarity. The intuition is that gradients pointing in roughly the same direction indicate similar datasets. If gradient vectors point in orthogonal or opposite directions, the training will likely fail to converge, which should be a good indicator that the datasets are dissimilar. Instead of alternating training batches from the beginning, it might be beneficial to mix in auxiliary data when a plateau on the main task is reached. The findings from Bingel and Søgaard (2017) indicate that if the main task reaches a training or validation error plateau, using auxiliary data can help to decrease the validation error on the main task. For similar datasets the validation error should be decreased while an unrated auxiliary dataset will probably increase the validation error.

An approach based on pure trial and error could test varying amounts of auxiliary data and compare the performance on the main task. The underlying assumption is the more related two datasets are, the more from one dataset can be mixed in the training without decreasing the performance on the other dataset. If the main task performance decreases (compared to the single task learning) when only a small fraction of auxiliary data is included in the training, this is another indicator for unrelated datasets.

All the ideas outlined above share a common problem. They (partly) rely on trial and error, which can be a time-consuming process. Only the strategy of alternating batches of main and auxiliary data from the beginning provides results faster than a full training. If the goal is to learn more about data similarity, such expensive experiments can be justified. In case the best performing auxiliary dataset has to be predicted before the actual multi-task learning, the gradient-based approaches are of little help.

In this chapter, new concepts have been developed to calculate the similarity of sequence tagging datasets. To evaluate these methods, the similarity between datasets has to be compared with the effect of using the datasets in a neural network. A suitable neural network implementation will be developed in the following chapter.

5 Neural MTL system implementation

Verifying any hypothesis experimentally needs data from appropriate experiments to perform a statistical evaluation. Hypothesis 1 states that more similar auxiliary training data achieves a better performance on the main task. To check the performance effect of auxiliary data for training a neural network, a neural network implementation is required that is capable of multi-task learning. In the first section of this chapter, the specific goals and the scope of the neural system implementation are defined. They form the basis to explain and justify the network architecture, design decisions and implementation details in subsequent sections.

5.1 Objectives

The primary objective of the neural network implementation is to show a difference in performance when using various (auxiliary) datasets for training. Further, the neural model must not prefer any of the datasets to enable a fair comparison of the results. A problem is that these are quite unspecific goals, i.e. it is unclear how the network implementation can achieve those goals. To mitigate this issue, a number of smaller, more specific objectives will act as proxy for the broader goals.

The network architecture has to support both single-task and multi-task learning. Multiple additional datasets with possibly different tagsets have to be usable as auxiliary tasks in the multi-task implementation without giving an advantage in case of identical main and auxiliary tagsets. These points shall ensure that any sequence tagging datasets can be used in the experiments and that the comparison between them is as fair as possible.

The following objectives should allow differentiating the performance of various auxiliary datasets. Any amount of additional data shall be usable for training regardless of the size of the main dataset. Otherwise, only equal amounts of auxiliary and main training data can be used, which is a common setting to limit negative effects of auxiliary data to main task. The network hyperparameters and architecture should be configurable. Thus, networks with a large variety in model capacity can be constructed and trained consistently. It is especially critical that the degree of regularization can be adjusted from none to strong in order to account for the regularizing effect of more data. Otherwise, either a larger dataset might not provide any performance gains or a small dataset achieves an unnecessarily poor performance.

Apart from result-focused objectives, there are few general aspects. A simple, minimalist architecture is easier to understand, reproduce and compare with other approaches. The architecture should be based on hard parameter sharing because it is the most

commonly used method in related work. This allows a better comparison and results obtained from such an architecture are more likely to apply to other experiments, which are also performed with the common hard parameter sharing architecture. Besides, the neural network implementation has to be fast and efficient on GPUs. Otherwise, a large number of experiments would both be wasteful with resources and take too long to complete.

Finally, it is explicitly out of scope in this work to create the best-performing system for a specific task. Thus, no new state-of-the-art results have to be obtained in the experiments. Instead, the results should be helpful in understanding the connection between data similarity and effective main task performance in a multi-task learning approach.

5.2 Architecture

As mentioned before, the neural network architecture should be in line with most related work. If the similarity evaluation turns out to be model dependent, a widely used architecture is preferable. Thus, the most common hard parameter sharing architecture for sequence tagging should be resembled. As summarized in Section 3.1, the commonly used neural network consists of word embeddings, character embeddings and features, a bidirectional gated RNN, a linear transformation to the label space and a final classifier. While the last two layers are task-specific, the other layers are shared among all tasks during multi-task learning. The abstract network architecture is shown in Figure 5.1.

There are several options for word embeddings: They can be randomly initialized and trained like the other network parameters via backpropagation. Alternatively, fixed, pre-trained word vectors are used, which are not updated during training. It is also possible to use pre-trained vectors for initialization and continue to train the parameters. The last two options are the far more common choices in recent single- and multi-task learning approaches that are focused on achieving state-of-the-art performance. In extensive tests by Reimers and Gurevych (2017) and Yang et al. (2018), pre-trained word vectors provided a significant performance increase. In case the focus is on the comparison of STL versus MTL performance, an understanding of multi-task characteristics or task similarity, pre-trained embeddings were avoided in related work (Plank et al., 2016; Martínez Alonso and Plank, 2017; Bjerva, 2017; Ruder et al., 2019). As the goal of this work is clearly the latter and not the former, no pre-trained word vectors will be used in this neural network implementation. This way, any influence on the performance of various auxiliary datasets should be avoided in order to ensure a fair comparison.

Automatically learned character features are a common component in state-of-the-art neural sequence tagging architectures. Surprisingly, Reimers and Gurevych (2017) did not see significant performance improvements in their sequence tagging experiments when testing both convolutional (Ma and Hovy, 2016) and recurrent (Lample et al., 2016) neural networks for character representations. However, similar tests of the same two approaches by Yang et al. (2018) showed significant performance increases for many tasks, which is consistent with the broad usage of character representations in related

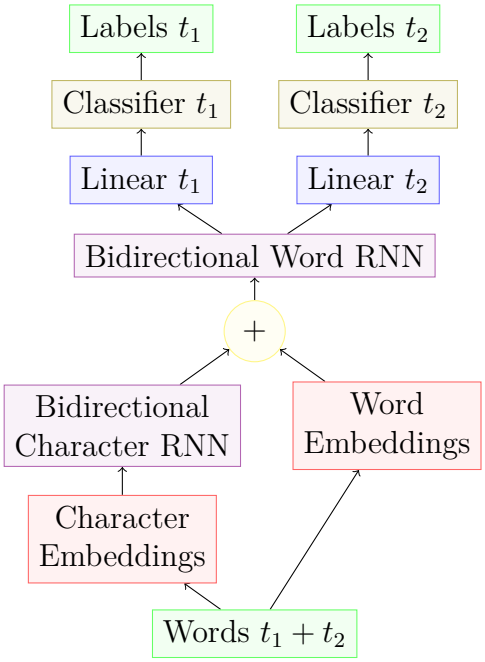


Figure 5.1: Generic MTL-capable neural network architecture for sequence tagging

work. Reimers and Gurevych (2017) and Yang et al. (2018) agree that the differences between convolutional and recurrent neural networks for extracting character features are negligible and neither approach decreases performance over a baseline without character representations. Thus, following the majority of related work, character RNNs will also be used in this neural network architecture. In contrast to word vectors, there is no need for pre-trained character embeddings as they can be easily learned from scratch due to their small vocabulary (language alphabet plus punctuation etc.). In most related work, a character RNN is implemented as a single-layer, bidirectional LSTM or GRU (see Section 2.2.3, page 17). Both types and their hyperparameters, e.g. number of hidden units, are made configurable for experiments.

The core component of most recent neural networks for natural language processing is a recurrent neural network operating on a sequence of word vector representations and their character features. For such a word RNN, a multi-layer, bidirectional LSTM or GRU is chosen in most related work. In this neural network implementation, the type of word RNN and hyperparameters, e.g. number of layers and hidden units, are configurable. The word RNN output is the last part that is shared across tasks in case of multi-task learning. For each task, this fixed-size representation has to be mapped to a vector with its size equal to the number of labels. This operation is performed by task-specific linear transformations, which are fully-connected feedforward neural networks. Consequently, any number of auxiliary datasets can be incorporated into the network as there is an output layer for the main dataset and for each individual auxiliary dataset.

The final component of the architecture is a classifier that produces label probabilities for each word in the sequence. For sequence tagging, two options are widely used. The Softmax function normalizes a real-valued vector into a probability distribution, so that each value is in range $(0, 1)$ and the resulting vector has a sum of 1. The Softmax function has no internal state or parameters and is directly applied onto the outputs from the task-specific linear transformations. A potentially stronger performing alternative is a conditional random field (see Section 2.1.3, page 8). A CRF (Lafferty et al., 2001) considers dependencies between labels and optimizes the probability of the whole sequence. Especially for tasks with high dependency between tags, e.g. named entity recognition, a CRF outperforms the simpler Softmax according to Reimers and Gurevych (2017) and Yang et al. (2018). However, their results show that for a few tasks, a CRF performs slightly worse than the Softmax. Because a CRF has trainable parameters, each task will need its own instance analogous to the task-specific linear transformations. In order to let each task or dataset perform best using either Softmax or CRF, both will be implemented and made configurable in this network architecture.

In summary, the architecture is highly configurable and applicable to any sequence labeling task in a single- or multi-task learning setup. It consists of a shared bidirectional LSTM or GRU that uses character representations from another LSTM or GRU and word embeddings that are learned from scratch. For MTL, the main and auxiliary tasks have individual linear transformation and either Softmax or CRF layers. The model should be able to extract and combine morphological, structural and possibly semantic features from any sequence tagging dataset to predict tags for each word.

5.3 Design decisions and training process

Regularization is a key aspect in parameter-rich neural networks to avoid overfitting. So far, the above described network architecture is completely unregularized, which is especially harmful for single-task learning performance. In case of multi-task learning the shared parameters are implicitly regularized due to the larger and more diverse training data. More explicit regularization is required to make STL more competitive. Otherwise, the comparison of STL and MTL performance would not be fair because the STL is more likely to overfit and perform poorly on an unseen test set.

Many regularization techniques have been proposed for neural networks, e.g. parameter norm penalties (L^1, L^2), data augmentation, parameter sharing, early stopping and dropout. Refer to Section 2.2.4 on page 18 for a description of these techniques. Data augmentation and parameter sharing are not generally applicable for recurrent neural networks processing natural language data. Convolutional neural networks working on image data can share the convolution kernel weights across all pixels of an image. Although RNNs already share parameters by using the same weights for multiple time steps in a sequence, the degree of parameter sharing is lower than in CNNs by an order of magnitude. Additional parameter sharing can be achieved with multi-task learning, but that does not make the STL setting more competitive. Data augmentation requires modifying the input data without impacting the results a human would be capable to

obtain. For natural languages, even slight modifications, e.g. swapping words in a sentence or characters in words, typically result in invalid training instances. Consequently, only parameter norm penalties, dropout and early stopping remain as regularization methods for the developed neural network.

Parameter norm penalties are applied by adding a term to the loss function used for training. Thus, it is automatically applied to all weights that are updated during backpropagation. The weight decay i.e. L^2 regularization factor can be globally configured as a hyperparameter. Early stopping affects the training procedure by limiting the number of training epochs. To apply it, the network implementation has to support checkpoints, i.e. the network parameters have to be saved and restored. As soon as the validation score has not increased for n epochs, the checkpoint with the best validation score is reloaded and the training is stopped. The number of epochs n, also called patience level, is a configurable hyperparameter in this implementation.

Dropout operations can be placed at multiple locations within the overall network architecture. In this implementation, dropout is performed on the output vectors of character and word embeddings, character representations and after each layer of the word RNN. This is in accordance with the sequence tagging implementations of *flair*[1], a state-of-the-art NLP framework (Akbik et al., 2018, 2019), and *AllenNLP*[2], a NLP research library (Gardner et al., 2018; Peters et al., 2018). In contrast to those sophisticated networks, the dropout rate is the same at each position in this implementation. Thus, it is a single configurable, real-valued hyperparameter in the interval $[0, 1)$.

Another important aspect is the runtime speed of the implementation. All building blocks of the network operate on batches of data in order to optimally utilize GPUs. Graphic and specialized tensor processing units can only achieve a high throughput when working on problems with lots of parallel arithmetic operations. Because the character RNN and the CRF Viterbi decoding significantly slowdown the computation (Yang et al., 2018), these implementations were carefully tuned to achieve a high degree of parallelism to be fast on GPUs. Processing data in mini batches is the commonly used strategy between the two extremes of processing either a single sentence or the whole dataset in one large batch. Smaller batch sizes tend to increase both the convergence rate of the training and the achieved test performance (Li et al., 2014; Masters and Luschi, 2018). Thus, the optimal batch size should be as small as possible while still almost fully utilizing the parallel processing capabilities of the hardware. Applying batch normalization (Ioffe and Szegedy, 2015) and using large datasets allows to increase the batch size without sacrificing convergence or performance according to Masters and Luschi (2018). However, for recurrent neural networks batch normalization does either not perform well (Laurent et al., 2016) or greatly increases computational cost (Cooijmans et al., 2016). Layer normalization (Ba et al., 2016) is a related method that is applicable to recurrent neural networks, but it is unknown whether layer normalization enables an increase in batch size similar to batch normalization. Hence, in this network implementation neither layer nor batch normalization is applied. Instead, the batch size

[1] `https://github.com/zalandoresearch/flair`
[2] `https://allennlp.org/`

is made a configurable hyperparameter that should be set in accordance with the above findings for experiments.

Operating on batches of data affects the training process in case of multi-task learning. When training on multiple datasets, one mini batch has to be filled with data from a single dataset. This allows computing the whole chain of operations including the task-specific output and calculating the loss with a task-specific loss function. During error backpropagation, the parameters specific to other tasks remain untouched as they were not involved in the forward computation. Mixing batches of the main dataset with possibly multiple auxiliary datasets can be done in many ways. The most common approach is to randomly select a task and sample the mini batch from its data (see Section 3.1, page 29). This process is repeated for either a fixed number of epochs or until the early stopping criterion is fulfilled. Another approach is to alternate the tasks and create mini batches from a dataset, which is shuffled per iteration. This ensures that every training instance is used exactly once per iteration if the dataset sizes are equal. In case the dataset sizes for the tasks differ significantly, there are multiple options of when to stop an iteration. It is possible to end the current iteration as soon as any dataset — alternatively, the main dataset — has been fully used. Consequently, the amount of data used from larger datasets is limited to the size of the smallest resp. the main dataset. To use the remaining training samples of the larger dataset, training is continued even after other datasets are exhausted. If one dataset is e.g. twice as large as the other, in the second half of the iteration, the network weights are only updated to improve on the larger dataset, which might drive the network parameters to suboptimal values for the smaller dataset. This especially becomes an issue if the main task is defined by the smallest dataset, which is a common scenario in multi-task learning as there are the highest possible gains from auxiliary training data. A solution is to weight the parameter updates by the inverse of the dataset size. This reduces the effect of the larger (auxiliary) data to be equal to the smaller (main) data. Thus, the effective training size can at most be doubled without favoring the task with the larger dataset during training.

An alternative to mitigate the above issue could be to interleave batches from all datasets according to their number of samples. Batches from the smaller dataset are evenly distributed across the whole iteration. This combines the advantages of randomly choosing a task and alternating tasks. The network is no longer trained solely on the larger dataset for the majority of an iteration and each training sample is used exactly once per iteration. To achieve a similar effect while randomly selecting the tasks, a task must be chosen from a distribution weighted by the dataset size. Thereby, tasks with larger datasets will be chosen more often compared to a uniform distribution. Table 5.1 shows a comparison for the different strategies. From the example sequence it becomes clear that four out of six strategies have significant downsides. Selecting tasks randomly from a uniform distribution uses batches from smaller datasets multiple times. The alternating approaches either do not fully use the available training samples or repeatedly sample batches from the largest dataset at the end. The interleaving strategy uses each batch exactly once and distributes the batches from different datasets evenly across the whole sequence. The random selection weighted by dataset size approximates the interleaving strategy.

batch combination strategy	possible example sequence
random (uniform task distribution)	$B_2, A_2, C_1, A_1, B_4, C_1, A_2, B_1, C_1, A_1, B_3, B_5$
alternating (stop when any empty)	B_3, A_2, C_1
alternating (stop when main empty)	B_2, A_2, C_1, B_5, A_1
alternating (use all remaining)	$B_3, A_1, C_1, B_4, A_2, B_1, B_2, B_5$
interleaving	$B_4, B_1, A_2, B_3, C_1, A_1, B_2, B_5$
random (weighted by dataset size)	$B_4, B_1, A_2, B_3, C_1, A_1, B_2, A_2, B_5$

Table 5.1: Batch combination strategies for three datasets A, B, C consisting of $2, 5, 1$ batches each. A is the main task.

5.4 Implementation summary

The neural network is implemented in Python[3] on top of the PyTorch[4] framework. PyTorch features a dynamic computation model, i.e. functions used in the forward path are automatically differentiated for backpropagation (Paszke et al., 2017). This is a perfect fit for the processing of variable-length sequences and dynamic architecture changes as required by the multi-task learning approach. The network implementation for this work is divided into separate functionalities for data loading, configuration, training, hyperparameter search and the actual neural network model. The source code can be found on the UHH Informatics Git server at `https://git.informatik.uni-hamburg.de/7schroed/mtl-nn`. A copy is also contained on the attached disc. For details, please refer to the Git repository or to the Appendix A.1 starting on page 111.

In this chapter, the neural multi-task learning system used to analyze the effect of auxiliary training data was described. To examine whether these effects coincide with the similarity concepts from the previous chapter, a program implementing the computation of those similarity measures is needed. This will be presented in next chapter.

[3] `https://www.python.org`
[4] `https://pytorch.org`

6 Dataset similarity tool

In this chapter, an implementation of the dataset similarity concepts shown in Chapter 4 is described. After an initial outline of the objectives for the implementation, an overview of the software architecture is given. While the implementation's core contributions — filling the contingency tables, fast word embedding comparison and computing similarity measures — will be explained in greater detail in subsequent sections, common functionalities are treated briefly within the architecture overview. The source code can be found on GitHub at `https://github.com/zoodyy/seq-tag-sim` and on the attached disk.

6.1 Objectives

The primary objective of this implementation is to apply and verify Hypothesis 2 — dataset similarity can be measured from data-inherent features — on real datasets, which in turn is needed to verify Hypothesis 1 experimentally later on. Thus, the program has to compare two datasets and output their similarity score as a result. To accomplish these objectives, the implementation has to read commonly used sequence tagging data formats, work on any tagsets and languages and provide a command line interface for easy automation in experiments.

Further, the different techniques developed in Section 4.5 for filling in the counts of a contingency table need to be implemented. This includes text overlap, vector space distance for unknown words and a fast contextual embedding comparison for each token. For the text overlap approach, different strategies to combine the counts from two datasets have to be developed. Finally, multiple similarity resp. distance measures, e.g. (normalized) mutual information and conditional-entropy, have to be calculated based on the contingency table counts.

In order to enable an easy usage for other researchers, the program should be instantly usable without a complex installation. It would be preferable to download a single binary file and run it. Optionally, a feature for printing or plotting (intermediate) results would be a convenient addition to see which tags are matched between two datasets.

6.2 Architecture overview

The software architecture of the implementation closely resembles a data-processing pipeline or buffered stream model. The pipeline model enables straightforward unit tests, delivers a high performance, allows reusing components and is easily extensible. In this implementation, multiple building blocks with well-defined APIs can be combined into

a pipeline This allows using various similarity computation techniques with any data formats without excessive code duplication. Figure 6.1 shows the architecture and data flow. The various parts depicted in the chart are referenced in the paragraphs below.

The entry point for the program resp. pipeline is a command line argument parser, which transforms the arguments into a tuple of configuration options such as input data formats, filenames etc. The actual pipeline composition is chosen based on the selected configuration options. This includes the type of input data parser, the approach to obtain label-pair counts to fill in the contingency table and optionally the type of word embedding to use.

The data readers share a common interface, i.e. they all produce a stream of sentences with token-tag pairs from a filename and file format. Multiple readers are grouped by their common data format, e.g. XML, CSV or TSV, to allow reusing code that is always needed to process a certain file format. The input files are assumed to be encoded as ASCII or UTF-8, which is able to represent any Unicode character and thus most written languages.

Regardless of the specific reader used to parse the input, a nested stream of sentences with word-tag pairs is supplied as input for one of the implementations that create a count-based contingency table from it. In the pipeline, the actual implementation is chosen at run time depending on the configuration. Thus, this step in the pipeline is a function mapping a stream of sentences to a real-valued, two-dimensional matrix. The details of the different implementations are described in Sections 6.3 and 6.4.

Another building block is the functionality to create (contextual) word embeddings from sentences. It is not part of the main pipeline, but included as a sub functionality in

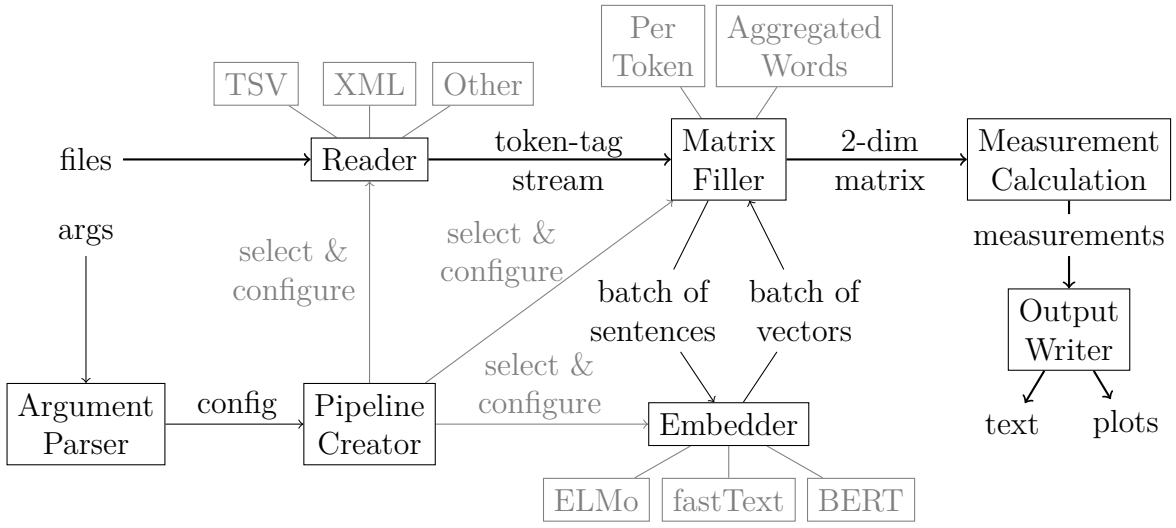


Figure 6.1: Software architecture for the dataset similarity tool. While black elements are part of the data flow, gray elements show the configuration possibilities. The path indicated by thicker arrows is the primary data-processing pipeline.

some implementations of the previous step. The supported word embedding libraries are fastText (Joulin et al., 2017), ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019). Independently of the chosen embedding type, the interface for this functionality is the same. A batch of sentences, consisting only of words, is transformed into a batch of lists containing word vectors.

The contingency table in form of a two-dimensional matrix is the input for the functionality that computes various information-theoretic clustering comparison measures. Details of this pipeline step are shown in Section 6.5. The output is a named tuple of single, real-valued measurements. The final step in the processing pipeline is a functionality to print the computed similarity scores to the console and optionally plot the contingency table to a separate window or file.

The entire system is implemented in the D programming language[1], which is a general purpose, multi-paradigm, statically-typed language with a C-like syntax. It compiles to native code, which allows creating a standalone runnable binary and provides great performance at run time including multi-threading. High performance and productivity combined with easy integration of existing C/C++ and Python libraries make D a perfect fit for the implementation of the dataset similarity tool. For a description of the program usage as well as implementation details going beyond the explanations in the following sections, please refer to Appendix A.2 starting on page 113.

6.3 Text overlap

The implementation for the text overlap approach closely follows the concept described in Section 4.5.1. The first step is to reduce all token-label pairs from a dataset to a set of unique words together with the corresponding label counts. This is achieved with a hash table that maps a word to the counts for each label. The input stream of token-label pairs is processed one pair at a time. If a word does not exist in the hash table, it is added with the initial label count 1. Otherwise, the label count for the existing entry in the hash table is increased by 1. Both datasets are stored in separate hash tables and are fully processed before the next step. To compare only the words contained in both datasets, the hash table for the first dataset is iterated. Only if a word is contained in the second dataset, the label-pair counts in the contingency table are increased according to the specific label count combination method. The implementation is highly efficient as hash table lookups have a constant $\mathcal{O}(1)$ time complexity. Processing two datasets of over 100 million tokens each takes only about two seconds.

Label count combination methods

For the text overlap approach, the label counts of a word contained in both datasets can be combined either via multiplication or addition per label. There are two optional weighting schemes for both multiplication and addition that may be applied simultaneously. First, the counts of each label per word can be divided by the total number of occurrences for

[1] `https://dlang.org`

that word in one dataset. This has the effect that label counts are split, i.e. they are no longer used multiple times. Second, the counts of each label per word can be divided by the number of times the specific label was seen in the entire dataset. When using the inverse label frequency weighting, all labels — both frequent and rare — contribute equally to the resulting contingency table. Table 6.1 summarizes the eight possible methods to combine the per word aggregated label counts. These methods transform the information contained in the aggregated label counts per word into a contingency table. An evaluation of the different methods will be performed in Section 7.1.

	Inverse frequency		
Method	Word	Label	Formula
Multiplicative methods			
`mul`			$\mathbf{C}_{ij} = l_i^1 \cdot l_j^2$
`mulIwf`	✓		$\mathbf{C}_{ij} = (l_i^1 / \sum l^1) \cdot (l_j^2 / \sum l^2)$
`mulIlf`		✓	$\mathbf{C}_{ij} = (l_i^1 / L_i^1) \cdot (l_j^2 / L_j^2)$
`mulIwfIlf`	✓	✓	$\mathbf{C}_{ij} = (l_i^1 / (L_i^1 \sum l^1)) \cdot (l_j^2 / (L_j^2 \sum l^2))$
Additive methods			
`add`			$\mathbf{C}_{ij} = l_i^1 + l_j^2$
`addIwf`	✓		$\mathbf{C}_{ij} = (l_i^1 / \sum l^2) + (l_j^2 / (\sum l^1)$
`addIlf`		✓	$\mathbf{C}_{ij} = l_i^1 / L_i^1 + l_j^2 / L_j^2$
`addIwfIlf`	✓	✓	$\mathbf{C}_{ij} = (l_i^1 / (L_i^1 \sum l^2)) + (l_j^2 / (L_j^2 \sum l^1))$

Table 6.1: Overview of methods to combine per word aggregated label counts into a contingency table $\mathbf{C}$. The counts per label for a single word in dataset 1 resp. 2 are denoted as l_i^1 resp. l_j^2 whereas L_i^1 resp. L_j^2 denotes the total number of occurrences of a label in a dataset. The method name parts `Iwf` / `Ilf` are abbreviations for inverse word / label frequency.

Incorporate non-shared words via vector representations

To use words, which are not contained in both datasets, word embeddings are required. Vector representations are obtained from all words of the second dataset. Because the words are aggregated from the whole dataset, they are no longer seen in context. Thus, using contextual embeddings does not provide any benefit over classic word embeddings, which are simply obtained from a lookup table mapping word to vector. The downside of these simplistic word embeddings is that they can only produce vector representations for words seen during creation of the embeddings. A solution is to use embeddings that include sub-word information like characters or morphemes. Therefore, a custom C++ wrapper for the fastText library is called for each word to be embedded. This enables fastText to use sub-word information to create vector representations even for unknown words.

To incorporate non-shared words into the comparison, the text overlap process is augmented. When iterating the first dataset and finding a word that is not found in the second dataset, the following steps are performed. The cosine similarities of the corresponding word vector to all word embeddings from the second dataset are calculated. Next, the word from the second dataset with the highest cosine similarity is chosen. The process described for the text overlap scenario is continued with the chosen word as if it was a direct match. However, the values added to the contingency table from both words' label counts are weighted by the previously computed cosine similarity. No strategy is implemented to mitigate the potential double use or other issues mentioned in Section 4.5.2.

The run time performance of this approach is acceptable because it operates on unique words instead of all tokens in the dataset. Additionally, non-contextual embeddings are rather fast to obtain. Moreover, comparing vector representations from the first dataset is only necessary for words that are not contained in the other dataset. Combined with an efficient implementation, these factors allow two datasets of one million tokens each to be compared in a minute or less on a modern multi-core computer.

6.4 Vector space similarity

Using vector space similarity exclusively requires to embed each token in both datasets. It is necessary to have different vector representations for two identical tokens occurring in different contexts. Thus, only contextual embeddings such as ELMo (Peters et al., 2018) or BERT (Devlin et al., 2019) are meaningful to use. ELMo is included into the system by calling the appropriate AllenNLP (Gardner et al., 2018) Python functions with help of the PyD library[2]. Because the raw ELMo vector representations consist of multiple layers, those are averaged to a single-layer vector. The integration of BERT embeddings is achieved by connecting a custom client to a bert-as-service (Xiao, 2018) server. The client sends a batch of sentences via network to the service and receives a batch of vector representations in return.

Regardless of which contextual word embedding library is used, batches of sentences are transformed to batches of lists with vectors that are inserted into a huge continuous tensor. For each dataset, this tensor contains the embeddings of every single token in the order of appearance within the dataset. An array of the same length holds the labels for each token in the same order. Thus, finding the label for a specific token vector can be done by looking at the same index in the label array.

To fill in the contingency table, the vector representations $\mathbf{v}_i$ and labels l_i of the first dataset are iterated in lockstep. For each vector $\mathbf{v}_i$ the cosine similarity needs to be computed against all token vectors from the second dataset. After finding the vector $\mathbf{v}'_j$ with maximum similarity and the corresponding label l'_j via its index, the count in the contingency table at the position $[l_i, l'_j]$ is increased. Two different matrices are filled in parallel. While the count is incremented by the fix value 1 in the unweighted matrix, it is increased by the cosine similarity value in the weighted matrix.

[2]PyD (`https://code.dlang.org/packages/pyd`) allows calling Python code from D and vice versa.

The implementation as described above should certainly outperform the text overlap approach with respect to the comparison quality. It does not have any inaccuracies compared to the probabilistic combination of aggregated label counts. It does, however, have a serious run time issue. Only tiny datasets containing a few thousand tokens can be compared in a short time with such a naïve implementation. Let the first dataset contain n tokens and the second m tokens. Every vector $\mathbf{v}_i$ needs to be compared via cosine similarity against all vectors $\mathbf{v}'_{1\dots m}$. Thus, in total $n \cdot m$ comparisons are needed, resulting in a run time complexity of $\mathcal{O}(n \cdot m)$. Computing the cosine similarity of two vectors $\mathbf{a}$ and $\mathbf{b}$ with d dimensions is defined as

$$\cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|_2 \|\mathbf{b}\|_2} \tag{6.1}$$

where $\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{d} a_i b_i$ is the dot or scalar product and $\|\mathbf{a}\|_2 = \sqrt{\sum_{i=1}^{d} a_i^2}$ is the Euclidean norm.

For a single d-dimensional vector, computing the Euclidean norm requires d multiplications and additions. The dot product of two d-dimensional vectors needs d multiplications and additions. Computing the cosine similarity of two vectors consists of at least $6d$ operations. Consequently, filling the count matrix for two datasets with n resp. m tokens using d-dimensional vectors requires at least $6nmd$ operations. Common word embeddings use hundreds and some even thousands of dimensions for their vector representations. In case of BERT or ELMo, official, pre-trained embeddings have 512, 768 or 1024 dimensions. Commonly used datasets such as the dataset for the CoNLL-2003 task on named entity recognition (Tjong Kim Sang and De Meulder, 2003) contain more than 300 000 tokens.

Assuming $d = 1000$ and $n, m = 300\,000$ the total number of operations is $6 \cdot 1000 \cdot 300\,000 \cdot 300\,000 = 5.4 \times 10^{14}$. A single core processor with a clock frequency of 3 GHz can perform 3×10^9 operations per second. It would need 180 000 seconds or 50 hours *iff* it does not need to wait for data to be loaded from or written to memory. Certainly, the processor must wait for data from memory as it exceeds the processor's cache by far. While vectors from the first dataset have to be read only once during the comparison, every vector from the second dataset must be read n-times from memory. This significantly slows down the computation because the processor cannot calculate at full speed. However, as all the vector operations are easily parallelizable, using a processor with c cores could theoretically reduce the time by a factor of c. The problem is that even if additional cores provide a speedup, processing is still limited by memory bandwidth and latency. Experiments to quantify this phenomenon by comparing the run time with an optimized implementation will be shown in Section 7.1.3.

The problem of finding the closest vector as described above is an instance of the nearest neighbor search (NNS) optimization problem. Word embeddings in form of vectors create a d-dimensional vector space, in which the closest neighbor of a certain vector is searched with cosine similarity as the distance measure. The method described above to find the closest neighbor is called linear search. Other methods also providing exact results are space partitioning approaches, where the vector space is recursively split

into two parts until the closest vector is found. While space partitioning algorithms can have a logarithmic time complexity for some distance measures and spaces, the linear search outperforms space partitioning approaches as the dimensionality of the search space increases. According to Weber et al. (1998), the threshold $\hat{d}$, when a linear search becomes in practice faster than any partitioning algorithm, is far below 610 dimensions. As the contextual embeddings have 512, 768 or 1024 dimensions, a linear search is preferable to space partitioning algorithms.

An alternative are approximation methods. While there are many approximate nearest neighbor (ANN) approaches, neighborhood-graph-based methods such as HNSW (Malkov and Yashunin, 2018) are the current state of the art when measuring speed versus recall according to Aumüller et al. (2017) and Li et al. (2019). They independently compared various ANN libraries on different datasets including word embeddings like GloVe (Pennington et al., 2014) using Euclidean distance or cosine similarity. The problem with any approximation method is that no exact results are obtained. Whether and how severely this would affect the dataset similarity calculation, has to be tested and thus to be compared with an exact method. Another downside is that the fast ANN approaches are designed for a different purpose. They require an index that is expensive to create, but can be used to answer any number of queries efficiently. For these reasons, the vector space similarity is implemented as a linear search. However, in case the run time of the sequential search cannot be reduced and remains impracticably high, trying an ANN library might be a possible solution.

A first step to reduce the run time of the linear search is to decrease the number of operations. Instead of computing the cosine similarity from scratch for every pair of vectors, the vector norms can be computed before the actual comparison. Dividing each vector by its norm produces normalized unit vectors. Calculating the cosine similarity of two unit vectors requires only the computation of the dot product, which reduces the number of operations from $6d$ to $2d$. Applying this simplification reduces the total operation count to $2nmd$ resp. 1.8×10^{14}. With the single-core processor from above, the theoretical run time is now 60 000 seconds or ≈17 hours.

The next step is to ensure that the real run time is not orders of magnitudes higher than the theoretical numbers due to memory bandwidth limitations. For both datasets, all vector representations are stored in one huge, continuous tensor. It is possible to exploit this fact. Instead of calculating n matrix-vector products, i.e. linear searches, which requires reading all m vectors n times, to compute all pairwise dot products, the problem can be reformulated as a single matrix-matrix product. The resulting matrix of size $n \times m$ contains the cosine similarity for every pair of tokens from both datasets. This has following advantage over the previous approach: Matrix-matrix multiplications can be both implementationally and algorithmically more efficient.

The matrix-matrix product is supported by the general matrix multiplication (gemm) procedure of the Basic Linear Algebra Subprograms (BLAS) (Dongarra et al., 1990). Advanced BLAS implementations such as ATLAS (Whaley and Dongarra, 1998), BLIS (Van Zee and van de Geijn, 2015), OpenBLAS (Xianyi et al., 2012), Intel MKL (Intel Corporation, 2009) or NVIDIA cuBLAS (NVIDIA Corporation, 2007) perform the gemm operations highly efficient on modern hardware. They use a block-wise computation that

is optimized to read the input data less often from memory than the naïve algorithm. Because this mitigates the memory latency and bandwidth issues, the implementations can further increase the processing speed *multiple times* by means of multithreading and SIMD (single instruction, multiple data) operations.

Assuming both datasets and vector length are of equal size l, it is possible to improve over the naïve $\mathcal{O}(l^3)$ time matrix multiplication algorithm. Strassen (1969) was the first to show matrix multiplication can be faster for large, square matrices and proved an $\mathcal{O}(l^{2.807})$ time algorithm (Strassen's algorithm). An asymptotically faster $\mathcal{O}(l^{2.2375477})$ time algorithm (Coppersmith–Winograd algorithm) was shown by Coppersmith and Winograd (1987) with subsequent, small improvements by Williams (2012), Davie and Stothers (2013), and Le Gall (2014). While the Coppersmith–Winograd algorithm and improved versions of the algorithm are not applicable to practical problems due to large constant factors and implementation difficulties on current hardware, Strassen's algorithm can be applied in practice even for small, non-square matrices on both CPUs and GPUs yielding a performance increase of 10–20 percent over the previously mentioned BLAS libraries (Huang et al., 2016, 2018).

Unfortunately, the fast implementation of Strassen's algorithm is not publicly available. Thus, classic BLAS libraries are used to compute the vector space similarity. In particular, NVIDIA's cuBLAS library is used because it builds on CUDA (Compute Unified Device Architecture) (Nickolls et al., 2008) to run on GPUs, which achieve a significantly higher throughput than CPUs on massively parallelizable workloads such as matrix multiplication. The approach described so far requires to multiply an $n \times d$ matrix $\mathbf{M}_a$ with an $d \times m$ matrix $\mathbf{M}_b$, which produces an enormous $n \times m$ matrix $\mathbf{M}_c$. Because the matrix $\mathbf{M}_c$, storing the pairwise similarities for all tokens, exceeds the memory of most GPUs for larger datasets, the operation is performed in batches. While the matrix $\mathbf{M}_b$ is fully copied onto the GPU, only a part of the matrix $\mathbf{M}_a$ is copied and processed at a time. Instead of copying $\mathbf{M}_c$ back to main memory, the indices with the highest similarity are computed and saved on the GPU with a custom CUDA kernel. This procedure is repeated until the matrix $\mathbf{M}_a$ is completely processed. For this approach to work, the only two requirements are that the smaller of both input matrices completely fits into the GPU memory and that some megabytes are still free for storing a part of the other matrix as well as the result. The batch size is automatically inferred from the amount of available GPU memory. The fewer batches are needed, the more efficient is the entire computation because $\mathbf{M}_b$ needs to be read less often. Preliminary tests indicate that filling the contingency table counts from contextual word embeddings with this custom implementation takes only a few minutes for datasets with one million tokens on a GPU with 11 GB of memory. See Section 7.1.3 for a more detailed analysis.

6.5 Contingency table similarity measures

Once the difficult work of filling the contingency table from the datasets is done, calculating their similarity is straightforward. The method to frame label similarity as a clustering comparison problem proposed in Section 4.3 shows how to compute the

mutual information as a similarity measure from a count-based contingency table in Equation (4.4).

While in the theoretical considerations division by zero or the convention $0 \log(0) = 0$ do not pose problems, an actual implementation requires a workaround for these cases. Although it is possible to clutter the calculation with conditional statements checking whether a count is zero, a simpler, faster and more robust approach is to use smoothing. The value of every cell in the contingency table is initially not set to zero. Instead, the smallest possible, positive number ϵ is used that can be correctly represented in IEEE 754 floating-point format. For double precision this value is $\epsilon \approx 4.94 \times 10^{-324}$, which should not have a measurable effect greater than the inevitable rounding errors inherent to floating-point math.

To easily compute the various information-theoretic measures outlined in Section 4.4, the probability estimates, i.e. frequencies, are pre-computed for all subsequent operations. Summing the contingency table separately by both dimensions returns a vector with the total counts per label for each dataset. Summing one of these vectors again yields the total count, which is then used to divide the original counts in the contingency table to produce relative frequencies i.e. joint probabilities.

Entropy is calculated twice, from both row and column probability vectors. Joint entropy is computed directly from the joint probability matrix. Mutual information is calculated according to Equation (2.12) from the joint probability estimates as well as row and column probability estimates. The measures compared in Section 4.4 are all derived from the previously computed entropy, joint entropy and mutual information values. Conditional entropy is obtained twice by rearranging and applying Equation (2.11) on both row and column entropy. The different variants of normalized mutual information are calculated by dividing the mutual information value by the joint entropy for NMI_{joint} resp. the maximum of row and column entropy for NMI_{max}.

In this chapter, the implementation of the various dataset similarity measures and their exact mechanics was explained. The similarity measures can now be quickly computed for the datasets that are used for training the neural network designed in the previous chapter. With the final piece of the puzzle in place, experiments can be performed in the next chapter to evaluate the similarity measures and assess their correlation with the effects of different auxiliary training datasets.

7 Experiments

The primary goal of this chapter is to experimentally verify the two hypotheses declared in Section 4.1. Hypothesis 1 states that „auxiliary data being more similar to the main training data results in a stronger main task performance compared to unrelated auxiliary data“. To test Hypothesis 1 an independent measure of dataset similarity is required. According to Hypothesis 2, „the similarity of two datasets D_1 and D_2 can be measured independently of a specific machine learning model“. Based on Hypothesis 2 different similarity measures were developed in Chapter 4 and implemented in Chapter 6. Before these measures can be used to test Hypothesis 1, preliminary experiments will be performed in Section 7.1 to evaluate implementation decisions and finalize the similarity measures. In subsequent sections, the final similarity measures will be used to experimentally verify Hypothesis 1.

The basic idea to test Hypothesis 1 is to verify it experimentally on different sequence tagging tasks with multiple datasets each. The scores from all multi-task learning combinations of datasets will be correlated against the corresponding dataset similarities. In Section 7.2, the experimental setup for performing meaningful experiments to test the hypothesis is outlined. The experiment results are shown and analyzed in Section 7.3. Finally, in Section 7.4 it is evaluated whether the multi-task performance can be predicted based on single-task performance and dataset similarity.

7.1 Preliminary evaluation of dataset similarity design decisions

While different variants of the similarity measures were applied to tiny, artificial datasets in Sections 4.4 and 4.5, they have yet to be evaluated on real-world datasets. In this section, preliminary experiments will be performed to select and finalize the similarity measure algorithms. Thereby, Hypothesis 2 is tested independently before the similarity measures are used in the full-sized end-to-end experiments that will test Hypotheses 1 and 2 simultaneously. The preliminary experiments are used to evaluate procedures for combining the counts in a contingency table and possible weighting schemes. Further, the theoretical shortcomings of the purely textual comparison will be checked experimentally. Likewise, the assumed benefits of embeddings for unused words and completely using contextual embeddings are to be tested. Last, the run time efficiency of embedding comparison algorithms is verified to ensure these techniques can be applied in the full-size experiments.

	identity NMI_{max} scores for datasets				
Method	WSJ	UD EWT	CoNLL'03	OntoNotes	H
`mul`	0.967	0.875	0.324	0.040	0.132
`mulIwf`	0.892	0.880	0.876	0.798	0.860
`mulIlf`	0.950	0.813	0.521	0.311	0.539
`mulIwfIlf`	0.887	0.970	0.843	0.929	0.905
`add`	0.005	0.009	0.026	0.049	0.011
`addIwf`	0.895	0.818	0.785	0.486	0.705
`addIfl`	0.003	0.007	0.011	0.003	0.004
`addIwlIfl`	0.858	0.776	0.617	0.387	0.600
`tokCtx`	0.999	0.993	0.955	0.996	0.985

Table 7.1: Comparison of label count combination methods for the text overlap approach. The similarity scores are obtained by comparing a dataset with itself. Methods should produce NMI_{max} similarity scores close to 1.0. H is the harmonic mean across all datasets: WSJ (English Penn Treebank Wall Street Journal release 3 (LDC99T42), POS), UD EWT (Universal Dependencies English Web Treebank (LDC2012T13), POS) ConLL'03 (English CoNLL-2003 Shared Task, NER), OntoNotes (English OntoNotes release 5.0 (LDC2013T19), NER). For more information about the datasets, refer to Table 7.3.

7.1.1 Evaluation of the similarity of identical datasets

For the text overlap approach explained in Section 4.5.1, the label counts of a word contained in both datasets can be combined by different methods described in Section 6.3. These methods transform the information contained in the aggregated label counts per word into a contingency table. The multiplicative and additive methods with optional weighting by inverse word frequency and/or inverse label frequency are referred to as `mul`, `mulIwf`, `mulIfl` and `mulIwfIfl` resp. `add`, `addIwf`, `addIlf` and `addIwfIlf` as in Table 6.1. A simple means to assess their quality is to measure how much information is lost during the transform. When comparing two identical datasets, the similarity score should be exactly 1.0. Because normalized mutual information measures (e.g. NMI_{joint} or NMI_{max}) can produce a score of 1.0 for contingency tables representing identical datasets, the root cause for lower scores is in earlier processing steps. As the aggregation of token-label pairs (into a list of unique words with multiple label counts) is fixed, the only remaining variable is the label count combination method. Thus, producing a NMI score close to 1.0 for identical datasets is a beneficial property. The `tokCtx` method uses BERT word embeddings to compare each individual token as in Section 4.5.2. It is included as a reference in the comparison.

Table 7.1 shows the NMI_{max} similarity score when comparing a dataset with itself.

In all but one cases, the multiplicative methods are far superior to the corresponding additive approaches regarding the identity similarity. Only on the WSJ dataset, `addIwf` minimally exceeds the value of `mulIwf`, which may easily originate from floating-point inaccuracies. Within the additive methods, both `add` and `addIfl` produce scores close to zero for each tested dataset and thus completely fail the identity test. The simplest multiplicative method `mul` without any weighting shows highly different scores across the dataset. While its score on the POS tagging datasets is high with 0.967 resp. 0.875, the scores for the NER datasets are very low at 0.324 resp. 0.040. The additional weighting by the inverse word frequency (`mulIwf`) is especially helpful in case of the NER datasets. Weighting by the inverse label frequency is not enough to obtain high values for `mulIlf`, but it still increases the NER scores significantly. The highest scores over all datasets are achieved by the multiplicative method `mulIwfIlf` with both weighting schemes applied. Because the identity comparison test is not a perfect criterion to select good methods, it will only be used to sort out those methods that failed the identity test. The four worst methods `mul`, `mulIlf`, `add` and `addIlf` are discarded from the main experiments.

The data from Table 7.1 may also be used to partly check the theoretical shortcomings of the purely textual comparison. Words occurring with different labels multiple times are not only a theoretical problem but are also visible in the experimental data. If there were no ambiguous words with different labels, the NMI_{max} similarity scores would be exactly 1.0 for all count combination methods in Table 7.1 when comparing identical datasets. The absolute counts in the contingency table do not matter as long as each label from the first dataset is mapped to exactly one label from the other. Consequently, the methods to combine the label counts are crucial to reduce the negative effect of ambiguity that is inherent to the aggregated label counts. The similarity scores from Table 7.1 show that some methods are able to handle this shortcoming of the text overlap approach as the harmonic mean of the NMI across four datasets can be 0.905, which is close to the optimal value of 1.0. Nevertheless, there is still room for improvement. By using contextual embeddings to circumvent the ambiguity problem, the `tokCtx` method achieves a harmonic mean value of 0.985.

7.1.2 Comparison of the approaches to fill the contingency table

The issue of using only a fraction of each dataset does not apply to the case of comparing identical datasets. However, in every other case, this will have a negative impact on the similarity score quality. Quantifying this effect on its own is rather difficult because it is not known upfront how similar two non-identical datasets should be. In order to estimate this effect and the possible benefits of using word embeddings, the following experiments will be performed. Non-overlapping datasets will be sampled randomly from all sentences contained in the original datasets. The pairwise NMI scores, which are obtained from the text overlap approach with and without word embeddings for unused words and the token-based approach using contextual embeddings, are compared with each other. Under the assumption that the similarity within samples from the same dataset is higher than the similarity between samples from different datasets, the pairwise NMI scores may not be compared on the basis of the absolute values but their relative order. Each of

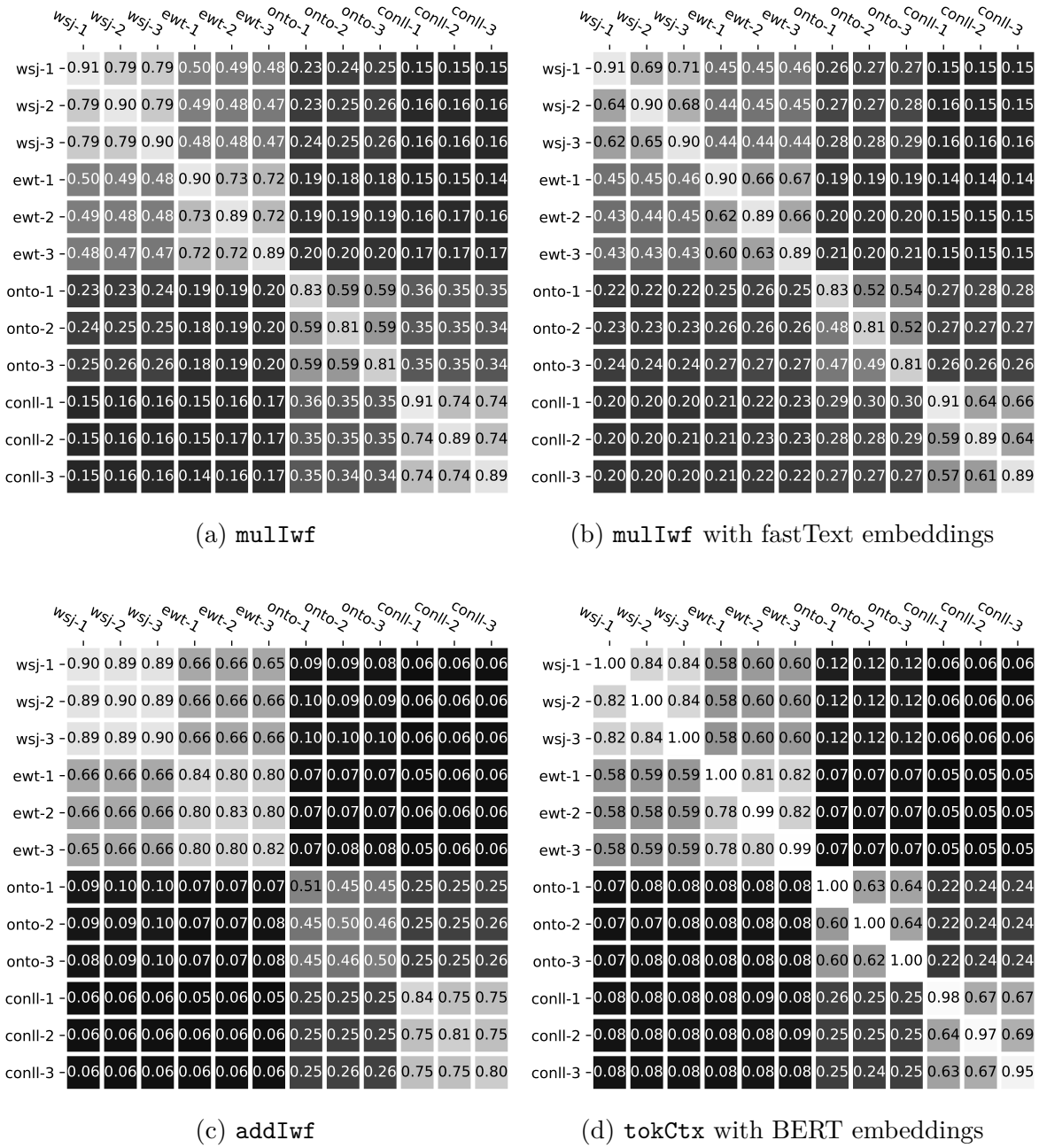
the datasets shown in Table 7.1 will be used to sample three new datasets. The samples have sizes equal to ⅙, ⅓ resp. ½ of the original number of tokens to account for effects when comparing datasets of different sizes. They are named e.g. WSJ-1, WSJ-2 and WSJ-3 because they have ⅙, ⅔ resp. ⅚ of the original number of tokens.

Figure 7.1 shows a few selected results of the pairwise comparison experiments. The full results for every label count combination method as well as with and without word embeddings can be found in Appendix A.3. The `mulIwf` label count combination method has the most promising results of all text overlap methods. Figures 7.2a and 7.2b show a clearly visible diagonal line due to the high identity similarity score. The 3×3 blocks along the diagonal are aligned with comparisons of samples within the same original dataset. Every value within these four blocks is higher than any other value outside. It is the only technique where the pairwise similarity scores within the same original dataset are always higher than between samples of different datasets. Another interesting result is that the similarity between the two POS tagging dataset samples (WSJ, EWT) is higher than the similarity between any POS–NER pair. The same is true for the NER dataset samples (CoNLL, OntoNotes). The overall results for `mulIwfIlf` (Figure A.3d) are similar to `mulIwf`, but not as good because the method seems to give higher scores to the POS samples than the NER samples. While the `mulIwf` scores shown in Figure 7.2a are symmetric, this is no longer true when using word embeddings for unused words as shown in Figure 7.2b. The similarity scores when using additional embeddings are generally lower or very similar. From these experiments, there does not seem to be an immediate benefit of using word embeddings for the multiplicative text overlap approach.

Figure 7.2c shows results for `addIwf`. Although the identity diagonal is not clearly visible, the identity scores are still higher than in the surrounding 3×3 block. However, the similarity within the OntoNotes samples is significantly lower than the scores between samples of the two POS datasets. The method `addIwfIlf` (Figure A.4d) has overall similar but lower scores. Using fastText word embeddings further reduces the similarity scores of all additive methods but does not help to correct the relative order of the scores between samples from the same resp. different original dataset.

While the text overlap approach does not seem to benefit from word embeddings to utilize non-overlapping vocabulary, the use of contextual embeddings for every token removes the necessity of complex label count combination methods. The results for `tokCtx` are shown in Figure 7.2d. Its performance is similar to `mulIwf` regarding the relative ordering of the scores. However, the similarity scores of samples from different datasets are lower except for the comparison of two POS datasets. In contrast to all text overlap methods, the token-based approach uses every token of the first dataset for the similarity calculation. Whether this property or the ability to handle ambiguously labeled words provides benefits, will have to be tested during the main experiments. Overall, the relative ordering of the similarity scores presented in Figure 7.1 indicate that Hypothesis 2 is valid, i.e. the similarity of two datasets can be measured independently of a specific machine learning model.

Figure 7.1: Selected pairwise NMI_{max} similarity scores for different methods to fill in the counts of the contingency table. The heatmaps encode the values from 0.0 in black to 1.0 in white.



(a) `mulIwf`

(b) `mulIwf` with fastText embeddings

(c) `addIwf`

(d) `tokCtx` with BERT embeddings

7.1.3 Run time efficiency of word vector comparisons

While the text overlap comparison of two 2-million-token-sized datasets takes less than 0.5 seconds, the vector similarity approach requires computationally expensive operations. In Section 6.4 two methods are described to compute the pairwise similarity of all word vectors from two datasets. For each vector from the first dataset, the *naïve* method performs a linear search through all vectors of the second dataset. The *matMul* method instead performs matrix-matrix multiplications in batches followed by operations to find the maximum similarity per row and column. Table 7.2 shows the run time of the two methods for different sizes of datasets and varying number of threads. While the naïve method is only implemented on the CPU, the matrix multiplication method is tested on both CPU and GPU.

The naïve algorithm scales very well with the number of cores resp. threads. Doubling the number of threads reduces the run time by 45 % on average. However, it is ≈ 7 times slower than the matrix multiplication approach. Although the algorithmic complexity is identical and both methods produce approximately the same results, the optimized algorithm utilizes the computation capabilities of the hardware much better — probably due to optimized memory and cache usage. Doubling the number of threads for the matrix multiplication method does not result in a consistent performance increase. While there is a speedup of 17–25 % for the smaller datasets, larger datasets do not show any gains using more threads than physical cores. Increasing the number of threads reduces the amount of available cache per thread, which in turn reduces the computational efficiency of each thread.

The GPU implementation of the optimized method is significantly faster than the CPU variant. On the smallest dataset, the GPU only needs ¼ of the processing time. This factor changes to 1/11, 1/14 resp. 1/10 for the next larger datasets. Using multiple GPUs does not provide any benefit on the two smallest datasets. For larger datasets, the run time is reduced linearly with the number of GPUs.

With increasing dataset sizes, the run times of all methods roughly increase quadratically. This is expected as comparing every word vector with all other word vectors has a quadratic complexity. The GPU implementation manages to hide this behavior when only looking at the three smallest datasets because the hardware might not be fully utilized on the smaller datasets and constant time operations such as data copying take up a lot of the total run time.

Summarizing the run time experiments, the optimized matrix multiplication method is far superior to the naïve method. The former compares two datasets of 300 000 tokens in less than five minutes while the latter requires about 40 minutes for the same task on identical hardware. Using GPUs, it becomes feasible to compare huge datasets consisting of two million tokens each. The comparison process takes less than five minutes when using four GPUs. Computing the pairwise similarities of all datasets used in the main experiments is easily possible with a single GPU as each comparison takes only a few seconds because one of the two datasets is always small. Most importantly, the run time of any similarity computation is only a minuscule fraction of time it takes to perform the actual training of the neural network.

Dataset	Tokens	Algorithm	Threads	GPUs	Batches	Time	Memory
conll-1	48 760	naïve	10		-	165 s	0.4 GB
conll-1	48 760	naïve	20		-	106 s	0.4 GB
conll-1	48 760	naïve	40		-	49.4 s	0.4 GB
conll-1	48 760	matMul	10		1	12.2 s	9.5 GB
conll-1	48 760	matMul	20		1	9.1 s	9.5 GB
conll-1	48 760	matMul	40		1	6.5 s	9.5 GB
conll-1	48 760	matMul		1	1	1.6 s	9.5 GB
conll-1	48 760	matMul		2	2×1	2.0 s	2×5 GB
conll-2	97 524	naïve	10		-	635 s	0.6 GB
conll-2	97 524	naïve	20		-	397 s	0.6 GB
conll-2	97 524	naïve	40		-	205 s	0.6 GB
conll-2	97 524	matMul	10		1	45.4 s	36 GB
conll-2	97 524	matMul	20		1	37.1 s	36 GB
conll-2	97 524	matMul	40		1	32.2 s	36 GB
conll-2	97 524	matMul		1	4	2.8 s	11 GB
conll-2	97 524	matMul		2	2×2	2.8 s	2×11 GB
conll	292 563	naïve	10		-	6741 s	1.8 GB
conll	292 563	naïve	20		-	3857 s	1.8 GB
conll	292 563	naïve	40		-	2245 s	1.8 GB
conll	292 563	matMul	10		4	399 s	105 GB
conll	292 563	matMul	20		4	271 s	105 GB
conll	292 563	matMul	40		4	268 s	105 GB
conll	292 563	matMul		1	36	18.2 s	11 GB
conll	292 563	matMul		2	2×17	10.7 s	2×11 GB
wsj	1 282 931	matMul	20		66	3490 s	105 GB
wsj	1 282 931	matMul	40		66	3576 s	105 GB
wsj	1 282 931	matMul		1	1910	348 s	11 GB
wsj	1 282 931	matMul		2	2×608	174 s	2×11 GB
wsj	1 282 931	matMul		4	4×836	85 s	4×6 GB
onto	2 001 102	matMul		4	4×3127	281 s	4×9 GB

Table 7.2: Run time and memory usage of the naïve and matrix multiplication methods to compute the pairwise closest vectors when comparing a dataset with itself. The computation time is measured for the two-way comparison including any necessary additional operations such as copying data from/to the GPU. Times to read the datasets from disk and to obtain the 768-dimensional BERT embeddings for each token are not included. The tests were performed on systems with two Intel(R) Xeon(R) Silver 4114 (10 cores / 20 hyper threads @ 2.2–3.0 Ghz), 128 GB RAM and up to four Nvidia GeForce GTX 1080 Ti (3584 CUDA cores @ 1.5 Ghz, 11 GB memory).

7.2 Experimental Setup

The plan to experimentally verify Hypotheses 1 and 2 will be described in this section. The hypotheses state that more similar auxiliary training data results in a stronger increase of the main task performance and that this similarity can be measured independently of a specific machine learning model. Both hypotheses will be tested jointly by correlating the similarities between pairs of datasets with the multi-task learning scores using one dataset from the same pair as training and the other as auxiliary data. The idea is that a positive correlation between dataset similarity and main task performance verifies both hypotheses jointly as it is unclear how to test each hypothesis on its own.

The experiments to correlate dataset similarity and the network's multi-task learning performance will be performed a) using two neural network models with different classifiers, b) for the tasks of part-of-speech tagging and named entity recognition and c) on multiple datasets per task. From each original dataset, a new training dataset will be sampled as a smaller subset to show a larger influence of auxiliary data. This is done for reasons similar as given by Yang et al. (2017). Recent neural network models are able to achieve almost perfect performance for easy tasks such as POS tagging when supplied with large datasets. There would be no room for improvement using auxiliary data. In addition, the artificially shrunk training sets are of equal size, which allows a fair comparison of the performance effect when using the same auxiliary datasets. Similar to the training data sets, multiple subsets of different sizes will be sampled from the auxiliary datasets. This makes analyzing the effect of various proportions between training and auxiliary data comparably over all original datasets.

The primary concern of the experiments is to enable significant differences in the neural network classification results when using different auxiliary datasets. If the performances using various auxiliary datasets were always insignificantly different, it would be inaccurate to correlate the values with the dataset similarity scores. To allow every training and auxiliary dataset combination to use their full potential, all relevant hyperparameters have to be tested for each pair of training and auxiliary dataset. Beside the number of hidden units, regularization has to be tuned individually per pair of datasets to obtain a fully regularized model, where the training, development and test loss resp. score become very similar. It enables a fair comparison between the multi-task and the single-task performance per training dataset. This is necessary as the difference between MTL and STL performance of combinations of training and auxiliary data has to be correlated with the dataset similarity of each pair. Comparing the relative change instead of the absolute performance values should be more generalizable when correlating with the dataset similarity.

Apart from selecting the best hyperparameters per combination of training and auxiliary dataset, it is necessary to share many parameters in the neural network to show the effect of different auxiliary datasets during multi-task learning. Sharing as many networks parameters as possible should show the strongest effect of different datasets — both positive and negative. The neural network architecture described in Chapter 5 builds on the commonly used bidirectional recurrent neural networks. Apart from self-learned word embeddings, character features are included. The neural network is designed to

ID	Dataset	Reference	Tokens	Tags
		POS Tagging Datasets		
BNC	British National Corpus	BNC Consortium, 2007	111 973 625	91
WSJ	Penn Treebank Wall Street Journal	Marcus et al., 1999	1 286 980	45
BC	Penn Treebank Brown Corpus	Marcus et al., 1999	1 162 358	45
EWT	UD English Web Treebank	Silveira et al., 2014	254 854	17
GSD	UD German GSD	McDonald et al., 2013	297 836	17
		NER Datasets		
ONT	English OntoNotes Release 5.0	Weischedel et al., 2013	2 001 102	37
CNLE	CoNLL'03 Shared Task (English)	Tjong Kim Sang and	301 418	9
CNLG	CoNLL'03 Shared Task (German)	De Meulder, 2003	310 318	9
EPG	Part of EUROPARL (German)	Faruqui and Padó, 2010	110 405	9
GEN	GermEval 2014 NER Shared Task	Benikova et al., 2014	591 005	24
GMB	Groningen Meaning Bank 2.2.0	Bos et al., 2017	1 354 149	17
SEC	SEC filings	Salinas Alvarado et al., 2015	54 256	8
WIKI	Wikigold	Balasuriya et al., 2009	39 152	8
WNUT	W-NUT'17 Shared Task	Derczynski et al., 2017	101 736	13

Table 7.3: Original POS tagging and NER datasets used to sample new training or auxiliary datasets. The number of tags is a generic count of the surface forms, where e.g. `B-PER` and `I-PER` are considered to be different tags.

share all parameters except those in the last two layers. These transform the RNN's hidden state to the task-specific labels and apply either a Softmax or conditional random field to obtain the most probable label prediction.

To perform reliable experiments, each training instance is run with multiple random seeds to mitigate performance fluctuations due to the random initialization of the network weights. Instead of choosing the result of the best random initialization, the results are averaged for all used random seeds. This is done for both the development scores used during hyperparameter search and the test score. The variance due to random weight initialization should be lower than the effects of using different auxiliary training datasets for the same main dataset. Averaging the scores for multiple random seeds and showing their standard deviation in the results helps to gauge the reliability of the experiments. The random seeds will be fixed so that any experiment is reproducible.

The exact experiment configurations are now made more concrete. From the POS tagging datasets shown in Table 7.3, a new training dataset of 25 000 tokens is sampled for WSJ, Brown and EWT while the BNC and GSD datasets are only used to sample auxiliary data. From all five POS tagging datasets, auxiliary datasets of increasing size are sampled containing $25, 50, 100, 250, 500, 1000 \times 1000$ tokens limited by the size of the original dataset.

For the NER datasets, the approach is similar to the POS tagging datasets. Training sets are sampled from all datasets except GMB because it is automatically annotated. As NER is a more difficult task, the sampled training datasets contain 50 000 tokens.

Auxiliary datasets containing $50, 100, 250 \times 1000$ tokens are created whenever possible.

For both POS tagging and NER, the canonical development and test sets of the original datasets are used for the newly sampled training sets if available. For datasets that have a test set but no development set, a new development set is sampled from the original training set. Otherwise, randomly sampled development and test sets, which are obtained from the entire corpus, do not overlap with any training or auxiliary dataset.

The range of the hyperparameter search for training the neural network model (see Chapter 5, page 49) was narrowed by manual testing. Training for both POS tagging and NER performs at most 100 epochs with an early-stopping patience of 10 using a batch size of 256. As the dimensions of the character embeddings and hidden units did not show a large effect, they are fixed at 32 and 64 for embeddings resp. hidden units. 128 and 256 dimensions are tested for the word embeddings and the hidden units of the word RNN that can have either one or two layers. Each combination of hyperparameters is run with three random seeds. For POS tagging, the learning rate is fixed at 0.002. The best dropout value is chosen from the values 0, 0.25, 0.5, 0.75. Additional regularization via weight decay is selected from the values 0, 0.1, 0.01, 0.001. For NER, the learning rate is set to 0.005 and weight decay uses a fixed value of 0.05. The range for dropout is narrowed to the values $0.3, 0.4, 0.5, 0.6$. While the POS tagging experiments only used a Softmax classifier, both Softmax and conditional random field are tested for each hyperparameter combination.

Hyperparameter search and thus training of the neural network is performed once without any additional data and multiple times for each auxiliary dataset and size. The main and auxiliary training datasets are combined deterministically via interleaved batches from both datasets (see Section 5.3). Auxiliary data is only used for the same task, i.e. no POS tagging dataset is used as auxiliary training data for NER and vice versa. For POS tagging, there are 81 pairs of training and auxiliary datasets. For each of these, 64 hyperparameter combinations are tested with three random seeds. This results in a total of 15 552 training runs for the POS tagging experiments. In case of NER, 168 pairs of training and auxiliary datasets exist. Each pair is tested with two neural network models, 16 hyperparameter combinations and three random seeds. In total, 16 128 training runs are performed for the NER datasets.

The dataset similarity tool developed in the previous chapter is used to compute the similarities for pairs of training and auxiliary datasets in three ways. The text overlap approach is used with and without word embeddings. For the latter, 300-dimensional fastText embeddings[1] with sub-word information are used that consist of 2 million word vectors trained on the Common Crawl (Mikolov et al., 2018). Contextual BERT embeddings (Devlin et al., 2019) are used for the third, token-based approach. More specific, the embedding model is „BERT-Base Multilingual Cased“[2], which has 12 layers, 768 hidden units, 12 heads and 110 million parameters.

[1] The pre-trained word vectors `crawl-300d-2M-subword.zip` were obtained from `https://fasttext.cc`

[2] `multi_cased_L-12_H-768_A-12.zip` obtained from `https://github.com/google-research/bert`

7.3 Results and analysis

In this section, the results are shown and analyzed that were obtained by performing the experiments defined in the previous section. The evaluation will be performed separately for the part-of-speech tagging and named entity recognition results.

7.3.1 Part-of-speech tagging

Table 7.4 shows the test scores achieved by the neural network with the best hyperparameter combinations for each pair of training and auxiliary dataset. Due to the small amounts of training data, the effect of different random initializations is larger than usual as can be seen from the standard deviation. Nevertheless, the mean accuracy across

Aux. data	BC-25	EWT-25	WSJ-25
none	85.61 ± 0.35	88.35 ± 0.42	86.35 ± 0.26
BC-25	87.19 ± 0.15	89.89 ± 0.39	89.29 ± 0.99
BC-50	89.32 ± 0.18	90.22 ± 0.68	90.35 ± 0.57
BC-100	91.09 ± 0.43	91.66 ± 0.39	91.35 ± 1.31
BC-250	92.19 ± 0.58	91.59 ± 0.04	93.20 ± 0.30
BC-500	92.33 ± 0.29	92.42 ± 0.58	93.57 ± 0.75
BC-1000	93.79 ± 1.00	92.62 ± 0.56	93.73 ± 0.25
EWT-25	86.58 ± 0.61	91.84 ± 0.46	87.99 ± 0.62
EWT-50	86.93 ± 0.46	94.00 ± 0.38	88.72 ± 1.22
EWT-100	87.56 ± 0.19	96.75 ± 0.17	88.92 ± 0.11
EWT-250	88.55 ± 0.52	95.87 ± 0.09	87.46 ± 0.54
WSJ-25	87.21 ± 0.47	89.73 ± 0.39	89.79 ± 0.09
WSJ-50	87.73 ± 0.27	90.87 ± 0.35	91.80 ± 0.14
WSJ-100	89.90 ± 0.54	91.41 ± 0.41	92.67 ± 0.80
WSJ-250	91.22 ± 0.77	92.53 ± 0.33	94.15 ± 0.65
WSJ-500	91.92 ± 0.69	93.18 ± 0.40	95.07 ± 0.48
WSJ-1000	92.73 ± 0.53	93.35 ± 0.32	95.70 ± 0.29
BNC-25	87.89 ± 0.78	89.92 ± 0.35	89.65 ± 0.42
BNC-50	88.92 ± 0.33	90.83 ± 0.30	89.55 ± 0.60
BNC-100	90.30 ± 1.00	91.84 ± 0.15	90.82 ± 0.46
BNC-250	90.46 ± 0.81	91.14 ± 0.98	92.01 ± 0.27
BNC-500	92.22 ± 0.53	92.34 ± 0.73	89.47 ± 0.62
BNC-1000	92.81 ± 0.16	93.06 ± 0.27	89.47 ± 0.61
GSD-25	86.32 ± 0.69	88.16 ± 1.31	86.94 ± 1.01
GSD-50	85.60 ± 0.65	88.75 ± 1.05	86.98 ± 0.46
GSD-100	86.65 ± 0.48	89.07 ± 0.83	87.58 ± 0.51
GSD-250	86.75 ± 0.53	89.29 ± 0.69	86.31 ± 0.14

Table 7.4: POS tagging MTL performance on three training datasets when combined with different auxiliary datasets. The mean accuracy and its standard deviation are obtained from three runs with different random seeds.

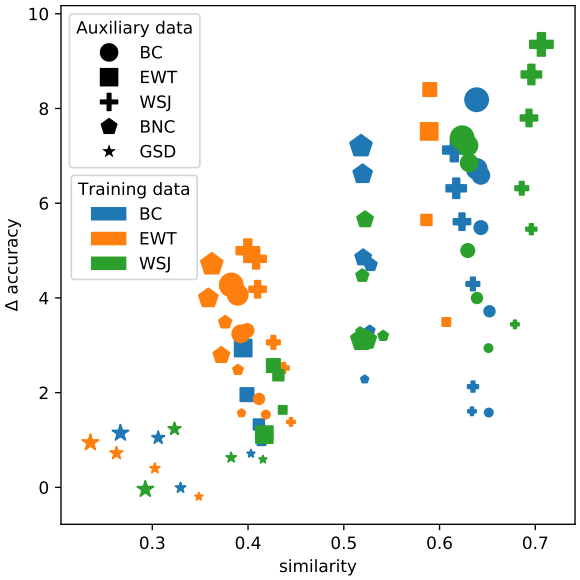
three runs generally shows a strong increase when using auxiliary data compared to the baseline without additional data. As expected, the performance increases more with larger amounts of auxiliary data. The strongest increase per training dataset is obtained when auxiliary data is used from the same original dataset, e.g. EWT-25 is paired with additional EWT data. All three training datasets (BC-25, EWT-25, WSJ-25) show only small gains or even losses in accuracy when the German GSD dataset is used as auxiliary training data. Within the English datasets, there are significant differences in accuracy depending on the combination of training and auxiliary dataset. For example, the BC-25 training data profits significantly more from WSJ or BNC than EWT as auxiliary dataset when comparing the same sizes of extra data. The POS tagging scores show clearly distinguishable performances regarding combinations of training and auxiliary datasets, which are perfect conditions to compare and possibly correlate the test scores with the similarity of the datasets.

The accuracy values are now compared with various similarity measures to find a measure that correlates with the accuracy. In Figure 7.4a, the difference in accuracy over the baseline is plotted against the NMI_{joint} similarity measure that was obtained via the text overlap approach using the `mulIwf` method to combine the label counts. The figure shows all 78 data points differentiated in three dimensions. First, the color denotes the main training dataset (BC, EWT or WSJ). Second, the form of the marker symbol indicates which auxiliary dataset (BC, BNC, EWT, GSD, WSJ) is used. Third, the size of the marker symbol relates to the size of the auxiliary dataset (25, 50, 100, 250, 500 or 1000 $\times$1000 token). This notation is also used in the other scatter plots.

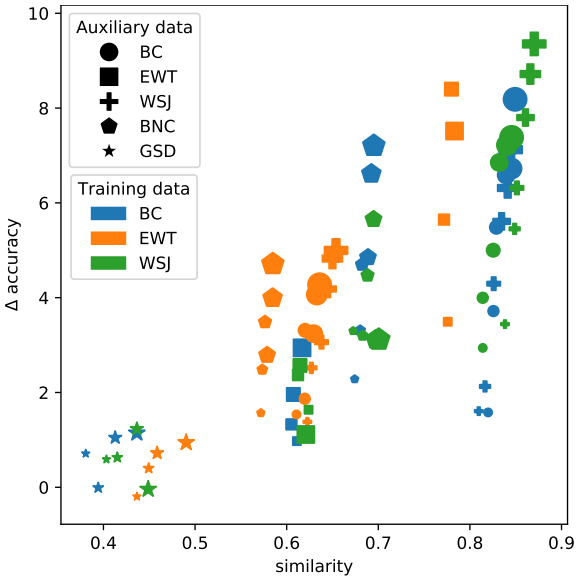
In Figure 7.4a, multiple patterns can be identified. Overall, the data points are scattered from bottom left to top right, i.e. there are no cases of low similarity coinciding with high accuracy increase and vice versa. Data points obtained from the same training and auxiliary dataset mostly form a straight line. This is the effect of the varying auxiliary dataset sizes. In case of higher similarity values ($\gtrapprox 0.5$), these lines are vertical, i.e. the similarity is independent of dataset size. Another interesting observation can be made from the data points with auxiliary data from the German GSD dataset. They are all clustered close to the bottom left i.e. low similarity and almost no gain in accuracy. This correlation concurs with the intuition that using a German auxiliary dataset for an English training dataset should not lead to a significant performance increase because the datasets are dissimilar.

In order to compare the similarity with the difference in accuracy, only data points belonging to the same main training dataset and the same size of auxiliary data may be considered. For a perfect correlation between accuracy and similarity within those data points, each point having a higher similarity than another point must also have a higher accuracy. When looking at all data points obtained with WSJ as the main training dataset, this monotone relative ordering is true for most auxiliary datasets. By starting at the bottom left, it is easy to see that the next group of green, same shape data points is always to the upper right, i.e. has both higher similarity and increase of accuracy. When performing this comparison for each set of comparable data points, the ordering is fully correct in the majority of cases. Moreover, the wrongly placed data points are no remote outliers but close to a correct position.

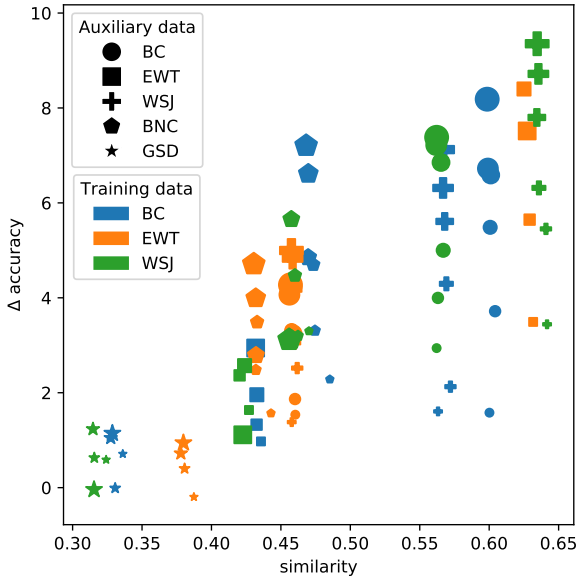
Figure 7.3: Scatter plots comparing different similarity measures with the same differences in accuracy of the POS tagging multi-task learning over the single-task learning score. Plots for more methods can be found in Appendix A.4.



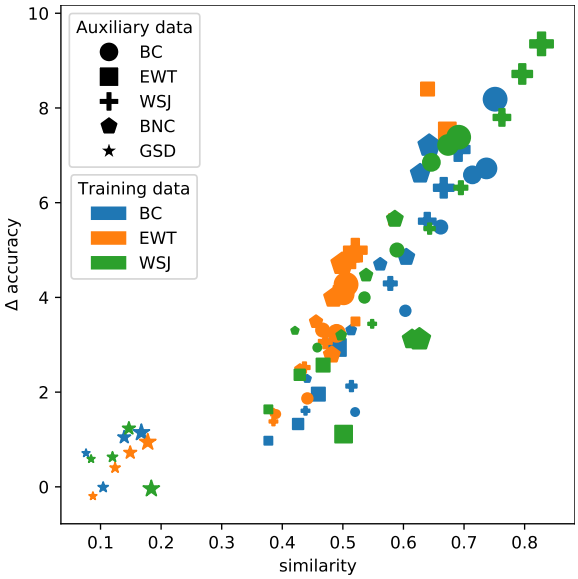
(a) plain `mulIwf` NMI_{joint}: Text overlap approach without word embeddings using the `mulIwf` method and NMI_{joint} measure



(b) mean fastText emb. `addIwf` NMI_{max}: Harmonic mean of both directions of the text overlap approach with fastText word embeddings using the `addIwf` method and NMI_{max} measure



(c) bwd. BERT emb. `tokCtx` NMI_{joint}: Backward direction (comparing auxiliary with training data) of the token-based approach with contextual BERT embeddings and NMI_{joint} measure



(d) SV & `mulIwf` NMI_{joint}: Harmonic mean of shared vocabulary and the text overlap approach using the `mulIfw` method and NMI_{joint} measure

Figure 7.4b shows the data obtained from the text overlap approach with fastText embeddings using the `addIwf` method and NMI_{max} as similarity measure. The usage of embeddings results in different similarity scores depending on the direction (comparing training with auxiliary or vice versa). Therefore, the harmonic mean is used to combine these two values into a single similarity score. While this figure looks similar to the previous figure, there is a notable difference. The data points originating from the same training and auxiliary dataset now form a line or curve that slightly leans to the right when seen from the bottom. Thus, the usage of word embeddings for otherwise unmatched words indirectly encodes the size of the auxiliary dataset into the similarity measure. This is the case because the chance of finding a highly similar word vector in the auxiliary dataset is higher with increasing size of the second dataset. More similar words tend to have a matching label, which finally increases the similarity score of the dataset comparison. Whether this effect is desirable or not, depends on the specific use case of the dataset similarity measure. The effect is stronger when the dataset comparison is performed with the token-based approach using vector similarity as shown in Figure A.14a. Again, the harmonic mean is used to combine the similarity scores of both directions. When only considering the direction from auxiliary to training dataset, the similarity measure is independent of the auxiliary data size as shown in Figure 7.4c. Regarding the quality of the embedding-based approach, the relative ordering of auxiliary datasets for the same training set is overall very similar to the text overlap approach. However, similarity and increase in accuracy correspond better for the EWT training dataset.

One possible downside of the text overlap approach without word embeddings is that the degree of shared vocabulary is not included in the measure. It is therefore easily possible to have a "false positive", i.e. a high similarity is reported for two datasets although they share only a single word. An effective solution to mitigate this potential issue is to combine the measure with the fraction of shared vocabulary. Figure 7.4d shows the harmonic mean of the text overlap approach from Figure 7.4a and the fraction of shared vocabulary. Using the harmonic mean instead of the arithmetic mean ensures that the combined similarity measure can only be a high value if both components are high as well. Including the vocabulary overlap into the similarity measure changes the look of the scatter plot significantly as the dataset size now has a dominant effect on both accuracy increase and similarity.

In order to quantify the findings from the figures and find the best similarity computation method, the correlation between accuracy increase and similarity will be calculated. However, computing the correlation across all data points has two issues. First, correlating across different auxiliary dataset sizes defeats the purpose as data size can have a stronger effect than the actual similarity. Methods implicitly including the auxiliary dataset size would perform exceptionally well while the actual data similarity is mostly ignored. Second, the correlation across different training datasets has the problem that both absolute and relative increase or decrease ignore the fact that it is usually easier to increase the accuracy or F1 score from e.g. 40 to 45 % than from 90 to 95 %. An idea could be to look at the relative error reduction, e.g. increasing the score from 90 to 95 % is a error reduction of 50 % because the error is reduced from 10 to 5 %. However, it is

	Kendall			Pearson		
Method	$\tilde{\tau}$	$\tilde{p}$-value	$\bar{\tau}$	$\tilde{\rho}$	$\tilde{p}$-value	$\bar{\rho}$
bwd. BERT emb. `tokCtx` NMI_{joint}	0.76	0.0004	0.74 ± 0.07	0.88	0.0006	0.87 ± 0.06
fwd. fastText emb. `mulIwf` NMI_{max}	0.74	0.0000	0.74 ± 0.07	0.87	0.0000	0.87 ± 0.05
mean fastText emb. `mulIwf` NMI_{max}	0.74	0.0000	0.70 ± 0.11	0.87	0.0001	0.87 ± 0.04
SV & `addIwf` NMI_{joint}	0.74	0.0003	0.74 ± 0.08	0.85	0.0005	0.84 ± 0.04
SV & `addIwf` NMI_{max}	0.74	0.0003	0.74 ± 0.08	0.84	0.0005	0.84 ± 0.04
SV & `mulIwf` NMI_{joint}	0.74	0.0001	0.74 ± 0.04	0.85	0.0002	0.85 ± 0.04
SV & `mulIwf` NMI_{max}	0.74	0.0003	0.72 ± 0.06	0.85	0.0002	0.85 ± 0.04
SV & `mulIwfIlf` NMI_{joint}	0.73	0.0004	0.70 ± 0.06	0.84	0.0004	0.84 ± 0.04
fwd. fastText emb. `mulIwf` NMI_{joint}	0.72	0.0001	0.72 ± 0.05	0.87	0.0001	0.87 ± 0.04
mean fastText emb. `mulIwf` NMI_{joint}	0.72	0.0001	0.68 ± 0.10	0.86	0.0001	0.86 ± 0.04
mean BERT emb. `tokCtx` NMI_{joint}	0.72	0.0005	0.72 ± 0.09	0.89	0.0005	0.88 ± 0.06
mean BERT emb. `tokCtx` NMI_{max}	0.72	0.0007	0.71 ± 0.07	0.86	0.0013	0.85 ± 0.07
mean BERT emb. `tokCtx` NMI_{max}	0.72	0.0007	0.71 ± 0.07	0.86	0.0014	0.85 ± 0.07
SV & `mulIwfIlf` NMI_{max}	0.72	0.0005	0.71 ± 0.06	0.83	0.0005	0.83 ± 0.04
fwd. BERT emb. `tokCtx` NMI_{max}	0.72	0.0005	0.70 ± 0.05	0.87	0.0012	0.85 ± 0.07
fwd. BERT emb. `tokCtx` NMI_{joint}	0.71	0.0002	0.70 ± 0.08	0.89	0.0005	0.87 ± 0.06
bwd. fastText emb. `mulIwf` NMI_{max}	0.70	0.0001	0.67 ± 0.08	0.85	0.0001	0.84 ± 0.05
mean BERT emb. `tokCtx` NMI_{joint}	0.70	0.0007	0.71 ± 0.10	0.89	0.0005	0.88 ± 0.06
plain `mulIwf` NMI_{joint}	0.69	0.0015	0.69 ± 0.07	0.82	0.0010	0.83 ± 0.07
bwd. fastText emb. `addIwf` NMI_{max}	0.69	0.0010	0.68 ± 0.08	0.84	0.0006	0.84 ± 0.06
shared vocabulary (SV)	0.60	0.0031	0.59 ± 0.14	0.77	0.0008	0.77 ± 0.06

Table 7.5: Correlation between various similarity measures and the change in POS tagging accuracy using multi-task learning. The entries show the median and mean of Kendall's and Pearson's correlation coefficients. They are sorted in descending order by the median of Kendall's rank correlation coefficient.

non-trivial to perfectly model this characteristic because the maximum does not need to be 100 % for every dataset as datasets resp. test sets might differ in their inter-annotator agreement. As the relative error reduction did not provide benefits with regard to the correlation, the absolute increase resp. decrease in accuracy or F1 score will be used in the further analysis.

Another idea is to calculate the correlation within groups of comparable data points, i.e. same training dataset and same auxiliary data size. The downside of this approach is that the number of data points to calculate a correlation coefficient is as small as five samples. A middle ground between both extremes is to perform the correlation calculation for larger groups, where data points belong to different training datasets but still are obtained with auxiliary datasets of identical size. As the three POS tagging training datasets have similar accuracy values without any additional data, the effect of comparing across auxiliary dataset sizes is by far the predominant issue.

Table 7.5 shows the median and mean correlation of similarity with change in accuracy

for the best 20 methods averaged over fix groups of data points obtained with the same auxiliary data size. As a baseline, the correlation with the ratio of shared vocabulary is added. Overall, the correlation between the similarity measures and change in accuracy is strong according to both Kendall's rank correlation and Pearson's linear correlation coefficients, which is in line with the previously shown scatter plots. The median p-values for most methods are below 0.001, which makes it *very* unlikely that similarity and accuracy are not correlated. The strongest correlation according to Kendall's τ is achieved with the `tokCtx` method comparing contextual BERT-embeddings from auxiliary to training data and applying the NMI_{joint} as similarity measure, which is depicted in Figure 7.4c. According to Pearson's ρ, the strongest linear correlation is again using contextual embeddings with NMI_{joint} but it combines the values of both directions via the harmonic mean. Regarding the correlation performance of the text overlap approach, there is an interesting pattern. The correlation coefficients are consistently higher when using word embeddings or combining with the ratio of shared vocabulary than the correlation coefficients of the pure text overlap approach. This is an indicator that the theoretical downside of only working on the shared vocabulary also has practical implications. Fortunately, combining the raw measure with the fraction of shared vocabulary via the harmonic mean is an effective fix as the correlation performance is similar to the text overlap approach with word embeddings. Interestingly, the shared vocabulary on its own has a much lower correlation. Its theoretical issue of being oblivious to the labels is very likely also a problem for real-world datasets.

7.3.2 Named entity recognition

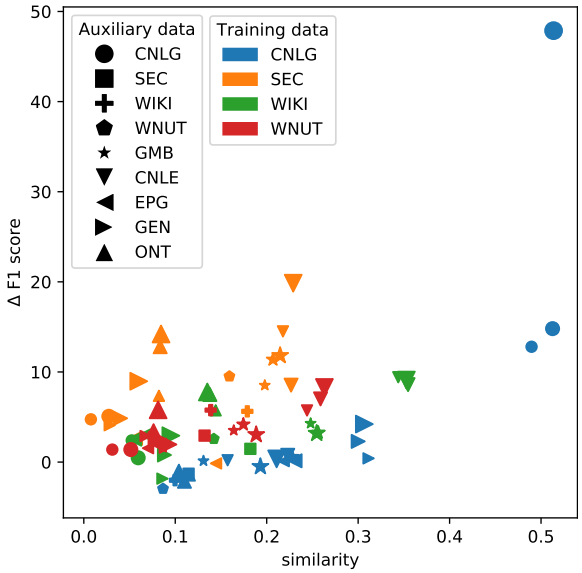
The neural network results for named entity recognition are shown in Table 7.6. These results were obtained with the neural network using a conditional random field as classifier. Similar to the POS tagging results, using auxiliary data from the same original dataset improves the performance the most. Unfortunately, the difference in F1 score between different auxiliary datasets is often smaller than the standard deviation of the values. The problem is especially dire for the training datasets EPG and ONT, where there is almost no change in the F1 score except for the case of using EPG resp. ONT as auxiliary dataset. When the data is correlated against the different similarity measures as in POS tagging, the correlation would be affected more by the randomness than anything else. For the training datasets CNLG, SEC, WIKI and WNUT, the F1 score differences between various auxiliary datasets have an acceptable level compared with the standard deviation. In order to avoid conclusions based on randomness, the datasets CNLE, EPG, GEN and ONT will be removed as training datasets from the further analysis. However, they will remain as auxiliary data for the four remaining datasets to use as many data points as possible. Because the numbers are very similar and have the same issues when using the Softmax classifier instead of a CRF classifier, those results will not be analyzed separately. For reference, the F1 scores obtained with the Softmax classifier are shown in Table A.1.

Figure 7.6a shows the scatter plot for the text overlap approach. Form, color and size of each point are used analogously to the POS tagging plots. Overall, the data points

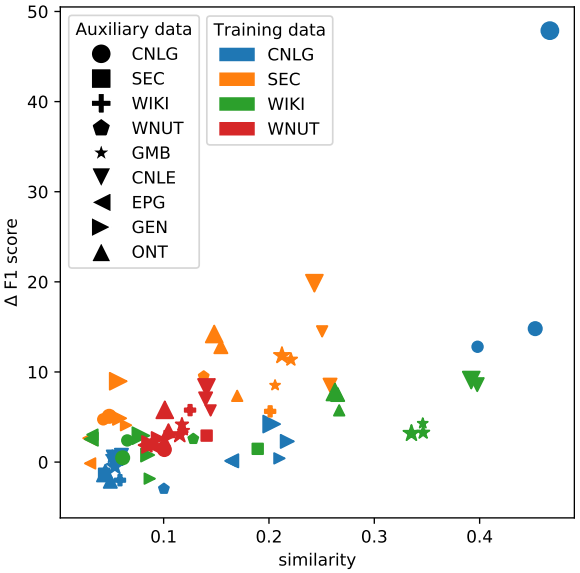
Aux. data	CNLE-50	CNLG-50	EPG-50	GEN-50	ONT-50	SEC-50	WIKI-50	WNUT-50
none	70.30 ± 2.50	41.62 ± 0.27	86.99 ± 0.42	26.97 ± 1.16	47.53 ± 0.83	43.86 ± 0.21	67.19 ± 1.38	12.67 ± 0.85
CNLE-50	83.68 ± 0.89	41.82 ± 1.03	86.53 ± 1.23	27.92 ± 0.34	47.14 ± 0.37	58.36 ± 4.74	76.61 ± 0.67	18.39 ± 1.14
CNLE-100	89.87 ± 0.12	42.38 ± 0.73	85.08 ± 0.53	29.02 ± 1.09	47.76 ± 0.33	52.39 ± 3.97	75.78 ± 1.26	19.67 ± 2.30
CNLE-250	95.51 ± 1.13	41.98 ± 0.48	86.79 ± 0.74	30.27 ± 0.97	46.89 ± 1.99	63.70 ± 2.29	76.29 ± 0.77	20.94 ± 1.34
CNLG-50	71.52 ± 1.05	54.42 ± 3.19	86.52 ± 1.05	28.24 ± 0.87	46.86 ± 0.30	48.60 ± 5.02	69.60 ± 1.28	14.05 ± 0.46
CNLG-100	71.72 ± 1.27	56.43 ± 0.79	86.97 ± 0.80	28.58 ± 1.12	46.50 ± 0.97	48.94 ± 1.00	67.67 ± 3.64	14.06 ± 2.11
EPG-50	71.44 ± 2.12	41.73 ± 2.17	94.43 ± 0.30	30.97 ± 0.75	46.84 ± 1.52	43.71 ± 0.82	70.24 ± 1.53	14.21 ± 0.58
EPG-100	70.80 ± 1.59	41.77 ± 0.47	98.64 ± 0.90	31.23 ± 0.29	47.40 ± 0.39	46.51 ± 0.74	69.74 ± 2.11	14.55 ± 0.15
GEN-50	70.93 ± 3.06	42.03 ± 1.99	86.46 ± 1.55	38.71 ± 1.80	47.14 ± 0.44	47.95 ± 2.01	65.36 ± 1.02	15.56 ± 1.15
GEN-100	72.31 ± 0.97	43.92 ± 0.37	88.14 ± 1.04	43.03 ± 2.27	46.63 ± 1.58	48.71 ± 2.53	67.98 ± 1.54	15.27 ± 0.49
GEN-250	71.75 ± 1.45	45.84 ± 2.07	88.23 ± 2.20	66.75 ± 2.88	47.11 ± 2.01	52.83 ± 1.51	70.12 ± 1.76	14.62 ± 0.91
ONT-50	73.55 ± 0.98	40.14 ± 2.13	88.54 ± 1.40	28.38 ± 2.63	52.13 ± 1.29	51.21 ± 5.01	72.94 ± 0.62	15.09 ± 1.44
ONT-100	72.18 ± 2.08	39.50 ± 2.24	85.37 ± 3.10	28.47 ± 1.38	57.47 ± 1.45	56.66 ± 2.53	74.69 ± 2.33	16.15 ± 1.10
ONT-250	75.55 ± 1.41	40.41 ± 0.93	86.69 ± 1.02	30.80 ± 1.56	61.16 ± 0.73	58.07 ± 2.61	74.98 ± 1.12	18.47 ± 1.09
SEC-50	71.50 ± 1.73	40.33 ± 2.04	85.32 ± 1.73	27.77 ± 2.19	47.23 ± 1.12	N/A	68.66 ± 0.59	15.60 ± 1.05
WIKI-50	72.66 ± 2.46	39.61 ± 1.71	88.06 ± 0.92	28.09 ± 2.43	48.65 ± 0.45	49.49 ± 2.59	N/A	18.43 ± 0.98
WNUT-50	71.90 ± 1.62	38.67 ± 1.41	84.79 ± 2.19	26.57 ± 2.25	46.80 ± 1.18	53.37 ± 2.89	69.77 ± 0.99	N/A
GMB-50	73.47 ± 1.68	41.75 ± 1.19	85.88 ± 1.78	26.97 ± 0.65	47.35 ± 0.74	52.38 ± 3.56	71.49 ± 0.60	16.18 ± 0.55
GMB-100	72.27 ± 2.65	42.21 ± 0.95	87.14 ± 1.52	27.34 ± 0.65	46.90 ± 1.56	55.24 ± 3.20	70.47 ± 2.39	16.84 ± 1.47
GMB-250	72.17 ± 1.87	41.12 ± 1.23	85.31 ± 1.40	27.81 ± 2.75	44.16 ± 0.56	55.68 ± 6.60	70.38 ± 1.89	15.71 ± 0.75

Table 7.6: MTL performance on three NER training datasets when combined with different auxiliary datasets. The mean F1 score and its standard deviation are obtained from three runs with different random seeds using the neural network model with the CRF classifier.

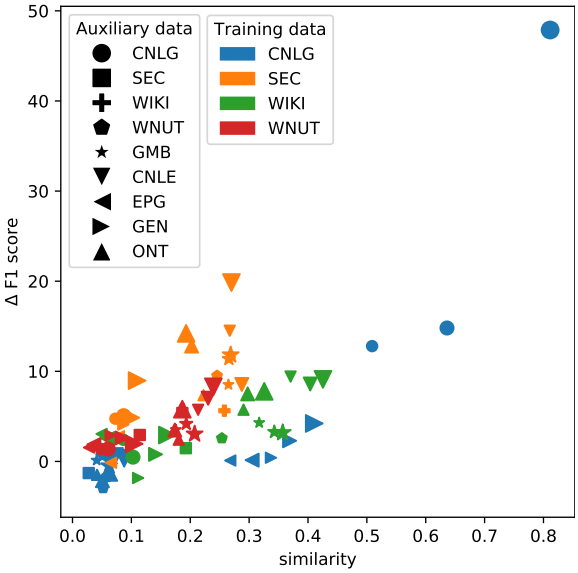
Figure 7.5: Scatter plots comparing multiple similarity measures with the difference in multi-task learning F1 score over the baseline on NER datasets. Plots for more methods can be found in Appendix A.5.



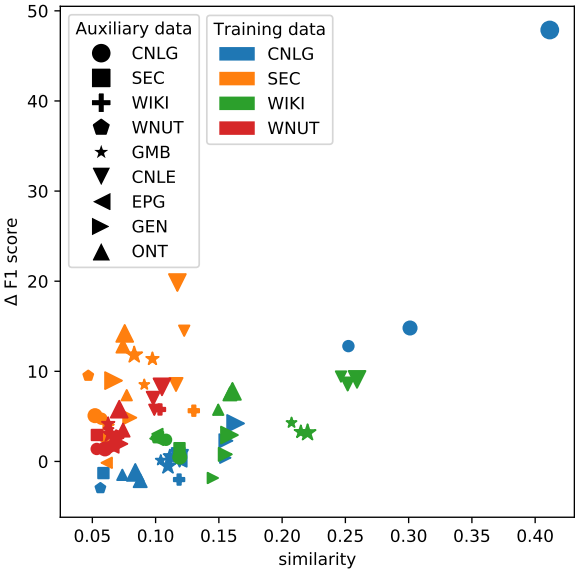
(a) plain `addIwf` NMI_{joint}: Text overlap approach without word embeddings using `addIfw` method and NMI_{joint} measure



(b) mean fastText emb. `mulIwf` NMI_{max}: Harmonic mean of both directions of the text overlap approach with fastText embeddings using `mulIwf` method and NMI_{max} measure



(c) SV & `mulIwf` NMI_{joint}: Harmonic mean of shared vocabulary and text overlap approach using `mulIwf` method and NMI_{joint} measure



(d) mean BERT emb. `tokCtx` NMI_{joint}: Harmonic mean of both directions of the token-based approach using contextual BERT embeddings and NMI_{joint} measure

show a slight upward trend with increasing similarity. The three data points originating from CNLG as training and auxiliary data stand out because it is the only combination where data from the same original dataset is used. The datasets for SEC, WIKI and WNUT are so small that no distinct auxiliary dataset could be sampled. When looking at the data points for each training set in isolation, an increase in similarity often coincides with the change in F1 score. While most data points for CNLG, WIKI and WNUT are placed on a line, a stronger spread is visible for the SEC dataset. The scatter plot obtained by additionally using word embeddings for the text overlap approach is shown in Figure 7.6b. An interesting effect is that all data points with the WNUT training dataset are now cramped together. However, the majority of those data points is still ordered as expected. For the other training datasets, the relative order within data points of the same training set is very similar to the comparison without embeddings. Combining the text overlap approach with the ratio of shared vocabulary is depicted in Figure 7.6c. It has the effect that more data points with low or no change in F1 score now have a similarity very close to zero. Figure 7.6d shows a scatter plot for the token-based approach using contextual embeddings. The placement of data points originating from the same training dataset seems to be adequate in most cases. Comparisons between data points of different training sets are probably not feasible because there are many data points with almost identical similarity that point to a largely different change in F1 score. Admittedly, this is also true for the other three scatter plots, but only to a lower degree.

To quantify the results and compare the different approaches and methods, the similarity can be correlated with the change in F1 score. In contrast to the POS results, the NER results have highly spread baseline scores ranging from $F1 = 12.67$ for WNUT to $F1 = 67.19$ for WIKI without auxiliary data. As explained in the previous section, correlating similarity and different levels of F1 scores does not make sense. A partial solution is to only correlate data points originating from the same training dataset, but this reduces the sample size for the correlation coefficient calculation to 8 resp. 9, which in turn reduces the confidence and statistical significance. The median and mean of Kendall's τ and Pearson's ρ correlation coefficients are calculated over twelve groups of data points. Within each group, both training dataset and the size of the auxiliary data are same for every data point. For the best 20 methods and shared vocabulary, median and mean of those correlation coefficients are shown in Table 7.7. The ranking of the various methods is overall quite similar to the POS correlation results. However, the previously highest ranked method using contextual embeddings for a unidirectional comparison is now slightly worse than the baseline of shared vocabulary. The top five methods according to Kendall's τ from the NER results are placed on position two to seven in the POS results. Again, combining shared vocabulary with the plain text overlap approach yields a higher correlation than either of the single values. The strongest linear correlation is obtained by the two-directional `tokCtx` method, which is in line with the POS results. Overall, the correlation coefficients are lower for NER than POS dataset comparisons. This is also visible in the scatter plots. The p-values are orders of magnitude higher because of both reduced sample size and lesser correlation. However, according to the p-values for Pearson's ρ, most similarity methods show a statistically

	Kendall			Pearson		
Method	$\tilde{\tau}$	$\tilde{p}$-value	$\bar{\tau}$	$\tilde{\rho}$	$\tilde{p}$-value	$\bar{\rho}$
mean fastText emb. `mulIwf` NMI_{max}	0.67	0.1575	0.63 ± 0.18	0.79	0.0326	0.81 ± 0.10
SV & `mulIwf` NMI_{joint}	0.65	0.1278	0.65 ± 0.16	0.79	0.0345	0.80 ± 0.08
SV & `mulIwf` NMI_{max}	0.65	0.1278	0.64 ± 0.13	0.80	0.0403	0.80 ± 0.07
SV & `addIwf` NMI_{joint}	0.65	0.1141	0.64 ± 0.18	0.76	0.0362	0.77 ± 0.13
SV & `addIwf` NMI_{max}	0.65	0.1224	0.64 ± 0.18	0.76	0.0393	0.77 ± 0.13
mean fastText emb. `mulIwf` NMI_{joint}	0.65	0.1575	0.62 ± 0.16	0.82	0.0270	0.83 ± 0.10
bwd. fastText emb. `addIwf` NMI_{max}	0.65	0.1575	0.62 ± 0.09	0.82	0.0559	0.78 ± 0.12
bwd. fastText emb. `mulIwf` NMI_{joint}	0.64	0.1361	0.63 ± 0.11	0.80	0.0438	0.78 ± 0.10
bwd. fastText emb. `addIwf` NMI_{joint}	0.64	0.1575	0.63 ± 0.11	0.79	0.0588	0.78 ± 0.12
bwd. fastText emb. `mulIwf` NMI_{max}	0.64	0.1575	0.62 ± 0.12	0.79	0.0475	0.79 ± 0.11
mean fastText emb. `addIwf` NMI_{joint}	0.63	0.1361	0.61 ± 0.19	0.76	0.0373	0.79 ± 0.14
fwd. fastText emb. `addIwf` NMI_{joint}	0.63	0.1575	0.60 ± 0.22	0.73	0.0525	0.75 ± 0.14
mean BERT emb. `tokCtx` NMI_{joint}	0.63	0.2347	0.57 ± 0.24	0.86	0.0754	0.76 ± 0.21
plain `mulIwf` NMI_{joint}	0.63	0.2347	0.55 ± 0.33	0.67	0.1218	0.64 ± 0.34
shared vocabulary (SV)	0.60	0.1260	0.61 ± 0.11	0.77	0.0446	0.79 ± 0.09
mean fastText emb. `addIwf` NMI_{max}	0.60	0.1575	0.60 ± 0.18	0.78	0.0318	0.78 ± 0.14
plain `addIwf` NMI_{joint}	0.60	0.1361	0.60 ± 0.21	0.72	0.0768	0.73 ± 0.18
plain `addIwf` NMI_{max}	0.60	0.2056	0.56 ± 0.23	0.69	0.0958	0.71 ± 0.19
bwd. BERT emb. `tokCtx` NMI_{joint}	0.60	0.3028	0.56 ± 0.20	0.67	0.1074	0.68 ± 0.23
bwd. BERT emb. `tokCtx` NMI_{max}	0.58	0.2347	0.54 ± 0.23	0.66	0.1421	0.67 ± 0.22
plain `mulIwf` NMI_{max}	0.55	0.3333	0.47 ± 0.34	0.61	0.3036	0.53 ± 0.41

Table 7.7: Correlation between various similarity measures and the change in NER F1 score using multi-task learning. The entries show the median and mean of Kendall's and Pearson's correlation coefficients. They are sorted in descending order by the median of Kendall's rank correlation coefficient.

significant linear correlation with the change in F1 score.

Summarizing the findings of the experiment analysis is a challenging task. Overall, there is a strong correlation between multi-task learning scores and dataset similarity computed by the methods developed in Chapters 4 and 6. In case of POS tagging, the correlation is impressive and statistically significant — it is obviously visible in the scatter plots and accompanied by high-confidence correlation coefficients. The results for NER are less clear as the scores obtained from the various training datasets are not easy to compare. However, there is still a strong indication that similarity and test set performance are correlated. Many ways to compute dataset similarity have been compared with respect to their correlation with the neural network test scores. While there is some consistency regarding which combinations work better than others, inaccuracies in the raw data make it difficult to select a single best method.

A computationally cheap yet high-quality measure is the harmonic mean of shared vocabulary and normalized mutual information calculated from the contingency table filled

by the text overlap approach. Possibly slightly better is the usage of word embeddings to match words not contained in the other dataset, which intrinsically adds the fraction of shared words to the similarity measure. The strongest linear correlation is achieved by using contextual embeddings to compare every token in the datasets.

In short, the similarity measures allow distinguishing good from bad candidates for usage as auxiliary data. This is an *immensely* valuable information because the number of expensive neural network training runs can be reduced to a fraction while still finding the best auxiliary dataset(s) to increase performance on the main task. Furthermore, both Hypotheses 1 and 2 can be considered valid as more similar data correlates significantly with a stronger increase of the main task performance in the majority of all examined cases. Thus, the similarity between datasets can be measured independently of a machine learning model and this similarity can be used to estimate the effect a specific auxiliary dataset will have on a neural network's test set performance.

In the next section will be explored whether the dataset sizes and multiple similarity measure methods can be combined to predict the multi-task learning performance based on the single-task learning test score of a neural model.

7.4 Multi-task learning test score prediction

Estimating the multi-task learning score from the single-task learning score and some data-specific features would be a highly convenient functionality. As the number of data points is very small, only simple models with few parameters can be reasonably fitted. The classic choice is linear regression, which is unsuitable in its basic form because it does not work well with multicollinearity. Multicollinearity describes a situation where the features used for prediction are correlated with each other. Since this is certainly true for the variations of the similarity measures, another model is needed. Ridge regression (Hoerl and Kennard, 1970) is a regularized linear regression model with L^2 regularization. It allows working on correlated features and helps to prevent overfitting. The ridge regression model shall predict the absolute multi-task learning score given the single-task learning score and features such as the size of training and auxiliary data, the fraction of shared vocabulary and numerous similarity measures.

In order to use the few collected data points most efficiently, nested cross validation (Stone, 1974) is applied. While the inner cross validation is necessary to select the regularization parameter, the outer cross validation is used to obtain an averaged error score on unseen data. The mean absolute error is selected for scoring because it allows an easy interpretation and is less sensitive to outliers than mean squared error. As the data points are not independent and identically distributed, the grouped k-fold cross validation is used. The data points are grouped by their single-task learning training dataset. This ensures that the test fold in each iteration contains data points belonging to an unseen training dataset. The data points obtained in the NER experiments using the Softmax neural network model were used to perform semi-automatic feature selection. The final evaluation uses the nested cross validation on the data points obtained from POS tagging and NER with the neural network CRF model.

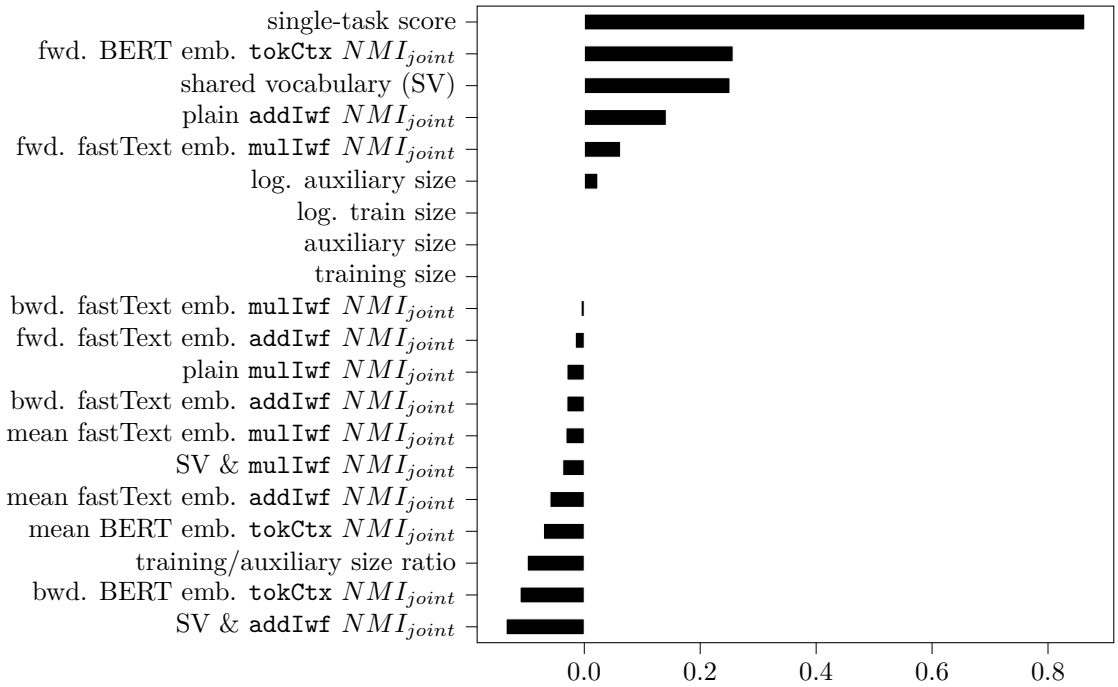


Figure 7.7: Ridge regression coefficients obtained from predicting the MTL score from dataset features and various similarity measures

Figure 7.7 shows the coefficients of the ridge regression for selected features. As expected, the single-task learning score is by far the most important feature. Manual tests without this information resulted in a very low prediction performance. Because the analysis in the previous section showed a large effect of the auxiliary dataset size on the MTL score, the five dataset size features were manually added. Interestingly, four of these are unused with coefficients near zero. It seems, the necessary information is encoded more easily in other features such as the fraction of shared vocabulary. Averaged over 100 runs of the above described nested cross validation method, a mean absolute error of 3.44 is obtained. If the deviations in the original F1 score resp. accuracy values are taken into account, this error is acceptable. When the same setup is used only on the POS tagging data, a mean absolute error of 1.01 is obtained, which is actually quite precise. As a comparison, all similarity measures are removed from the features. This results in a mean absolute error of 1.89 for POS tagging and 4.58 across all data points. Admittedly, these results should be taken with a grain of salt as the mean errors are of similar scale as the deviations in the STL and MTL scores due to random initialization of the neural network weights. Nevertheless, it is plausible that an accurate estimation of the multi-task learning scores based on dataset similarity and single-task learning score is possible given less erratic scores.

8 Summary, Conclusion & Future Work

8.1 Summary

Throughout this thesis, the research question — how the effect of auxiliary training data can be estimated in a multi-task learning scenario without performing the actual training — has been thoroughly examined. On the basis of the two hypotheses that a) auxiliary datasets of higher similarity with the main training data achieve a better performance on the main task and that b) the similarity between datasets can be measured solely from data-inherent features, theoretical concepts to measure dataset similarity based on the contained words and labels have been developed. The new methods to measure the similarity of sequence tagging datasets were developed with the goal to improve over existing approaches with regard to their applicability. The methods can be applied to any sequence tagging datasets without having any constraints on the label sets or requiring the tasks to be automatically taggable. The general idea is a two-step process. First, a probabilistic mapping between both label sets is obtained from pairs of words and labels either via aggregated counts per unique word or for each individual token. This information in form of a contingency table is used by multiple clustering comparison measures to calculate the mapping quality and thus the dataset similarity.

For the approach of working on aggregated label counts per word in the first step, various methods have been developed to combine these counts from both datasets into the single contingency table. Additionally, word embeddings have been included to match the words, which are not contained in both datasets, and compare their labels. The other approach uses contextual word embeddings to directly match the label from each token of one dataset to the most similar token of the other dataset. To make this pairwise comparison of large numbers of word vectors feasible, a sophisticated vector comparison has been developed that is both fast and efficient. Due to the modularity and existence of various alternatives for each part of the dataset similarity computation, there are many possible combinations for concrete similarity measures, which were subsequently implemented and tested experimentally both individually and together with the multi-task learning scores of a neural network.

Results of preliminary experiments on part-of-speech tagging and named entity recognition datasets are confirming to the intuition that pairs of datasets sampled from the same source have higher similarity scores than pairs of datasets from different sources. In large-scale experiments on real-world datasets, the correlation between the similarity measures and the change in the multi-task learning performance was analyzed. For

part-of-speech tagging, a significant and strong correlation of the neural network's test set accuracy and the similarity between training auxiliary dataset was found. Because the test scores obtained on the named entity recognition datasets are largely affected by random effects, the correlation to the auxiliary dataset similarity is only significant for some methods. When predicting the multi-task learning score of a specific auxiliary dataset from the single-task learning score, using various similarity measures as features improved the prediction performance over the baseline.

8.2 Conclusion

It is possible to estimate the effect of auxiliary training data on the main task performance in a multi-task learning scenario — without performing any expensive MTL training runs of the neural network! Overall, the experiments show that similarity measures allow ordering the effects of auxiliary datasets by direction and intensity for an individual training dataset. This is a strong indication for the validity of both hypotheses. As the performance of predicting the multi-task learning score for a specific auxiliary dataset could be increased by using various similarity measures as features, it seems plausible that an estimation of a neural network's multi-task learning score is possible.

The experimental findings are also supported from a theoretical point of view. The developed methods working on both words and their labels have a substantial advantage over approaches that are based only on word overlap between two datasets or the label distributions. While the similarity score might not be used as an absolute value, it provides a way to find a good auxiliary dataset for a given training dataset without actually performing the training. Consequently, the number of datasets that are actually used for training and expensive hyperparameter search can be reduced by only choosing the n auxiliary datasets most similar to the training data.

Depending on the exact similarity method, computing the similarity between two sizable datasets takes less than second for the approach without any word embeddings and a few seconds to a few minutes when using contextual embeddings depending on the exact dataset size and hardware. Thus, the dataset similarity can be leveraged to find the best auxiliary dataset(s) and improve the main task performance in a fraction of the time required by the full search through all possible datasets. The quick similarity calculation can even help to improve the main task performance further since better datasets might be tried as auxiliary data that would never have made it through the otherwise necessary, purely manual preselection process.

8.3 Future work

The possibilities for future work are vast. Based on the existing experiment results, it could be analyzed whether dataset similarity allows making estimates about the right amount of additional data. An intuitive hypothesis is that with a low similarity score, only small amounts of auxiliary data increase the performance, whereas larger

quantities will result in a reduced performance. For highly similar datasets, even a multiple amount of auxiliary data compared to the amount of training data could still improve the performance as it is the case for some part-of-speech datasets analyzed in the previous chapter. A similar direction is to investigate whether the dataset similarity could act as an indicator for the right amount of shared parameters in the neural network. For example, sharing all layers could be beneficial in case of a high similarity while for less similar datasets sharing only the lower layers would achieve a better performance.

Another possibility could be to compare the similarity measures developed in this work with related work, e.g. the results obtained by Bjerva (2017). The similarity measures would have to be computed on the same datasets used in the related work. Further, the values showing the change in test scores from single- to multi-task learning need to be available for each auxiliary dataset. This would allow correlating the similarity measures designed in this work with the results produced in related work. The other direction might not be possible because the approaches in related work compute task similarity and cannot compare the similarity of same-task datasets as used in the experiments of this thesis.

Improving the similarity measures is another possibility. By combining the newly developed similarity measuring methods via an oracle or majority voting, the estimation of the effect on the multi-task learning performance might be improved. Besides, the similarity measures could be easily extended to better support sequence tagging tasks where labels span multiple tokens or even whole sentences. Instead of using contextual embeddings per token, multi-token or sentence embeddings could be integrated while the remaining parts of the similarity calculation can stay untouched. Going further, sentence or paragraph embeddings could be used to compute the similarity for classification task datasets.

Finally, the prediction of multi-task learning scores based on single-task learning scores and dataset similarity measures could be improved and generalized. Existing results from other multi-task learning or transfer learning experiments could be used to obtain more data points of dataset similarity and test score. Additionally, new experiments could be run with other sequence taggers that are MTL-capable to gather more data points. Having a substantially larger set of these data points should improve the prediction substantially and might allow gaining new insights into the connection of data similarity and the effect of auxiliary training data.

List of Figures

1.1 High-level approach example 3

2.1 MEMM: Example illustrating the label bias problem 8
2.2 Comparison of HMM, MEMM and CRF 9
2.3 Perceptron 10
2.4 Activation functions: Sigmoid, tanh and ReLU 11
2.5 Two-layer fully-connected feedforward neural network with n input features, m hidden neurons and o output neurons 12
2.6 A two-layer recurrent neural network with n input features, m hidden neurons and o output neurons 14
2.7 Different types of sequence processing with recurrent neural networks . . 15
2.8 LSTM cell 18
2.9 Generic hard parameter sharing multi-task learning neural network architecture for t different tasks 24

5.1 Generic MTL-capable neural network architecture for sequence tagging . 51

6.1 Software architecture for the dataset similarity tool 58

7.1 Selected pairwise NMI_{max} similarity scores for different methods to fill in the counts of the contingency table 71
7.3 Scatter plots comparing different similarity measures with the differences in accuracy of the POS tagging multi-task learning results 79
7.5 Scatter plots comparing similarity measures with the difference in multi-task learning F1 score on NER datasets 84
7.7 Ridge regression coefficients obtained from predicting the MTL score from dataset features and various similarity measures 88

A.1 Pairwise relative vocabulary resp. token overlap 117
A.2 Pairwise NMI_{max} scores for the token-based methods using contextual BERT embeddings 117
A.3 Pairwise NMI_{max} scores for the multiplicative label count combination methods in the plain text overlap approach 118
A.4 Pairwise NMI_{max} scores for the additive label count combination methods in the plain text overlap approach 119
A.5 Pairwise NMI_{max} scores for the multiplicative label count combination methods in the text overlap approach with fastText embeddings for non-overlapping words 120

A.6 Pairwise NMI_{max} scores for the additive label count combination methods in the text overlap approach with fastText embeddings for non-overlapping words . . . 121
A.7 Scatter plots comparing the plain text overlap similarity measures with the differences in accuracy of the POS tagging multi-task learning results 123
A.9 Scatter plots comparing the harmonic mean of shared vocabulary (SV) and text overlap similarity measures with the differences in accuracy of the POS tagging multi-task learning results . . . 124
A.11 Scatter plots comparing similarity measures of the text overlap approach plus embeddings with the differences in accuracy of the POS tagging multi-task learning results . . . 125
A.13 Scatter plots comparing token-based similarity measures with the differences in accuracy of the POS tagging multi-task learning results . . . 126
A.15 Scatter plots comparing the plain text overlap similarity measures with the differences in accuracy of the NER multi-task learning results . . . 129
A.17 Scatter plots comparing the harmonic mean of shared vocabulary (SV) and text overlap similarity measures with the differences in accuracy of the NER multi-task learning results . . . 130
A.19 Scatter plots comparing similarity measures of the text overlap approach plus embeddings with the differences in accuracy of the NER multi-task learning results . . . 131
A.21 Scatter plots comparing token-based similarity measures with the differences in accuracy of the NER multi-task learning results . . . 132

List of Tables

2.1 Structure of a contingency table 27

4.1 Contingency table for a comparison of label sets L and L' with N resp. M unique labels 36
4.2 Counts from example Sentences 4.1 and 4.2 for comparison of NER and POS tagsets 36
4.3 Contingency tables for different scenarios comparing label sets L and L' . 39
4.4 Results for the evaluated measures on each scenario from Table 4.3 . . . 39
4.5 Transformation of word-label pairs to an associated count-based representation 42
4.6 Contingency table derived from the counts of words in Table 4.5 that are contained in both datasets. 43
4.7 Using vector space similarity to match tokens between two datasets to obtain counts for a similarity calculation based on a contingency table . . 45

5.1 Batch combination strategies for three datasets 55

6.1 Overview of methods to combine per word aggregated label counts into a contingency table 60

7.1 Comparison of label count combination methods for the text overlap approach 68
7.2 Run time and memory usage of the naïve and matrix multiplication methods to compute the pairwise closest vectors when comparing a dataset with itself 73
7.3 Original POS tagging and NER datasets used to sample new training or auxiliary datasets 75
7.4 POS tagging MTL performance on three training datasets when combined with different auxiliary datasets 77
7.5 Correlation between various similarity measures and the change in POS tagging accuracy using multi-task learning 81
7.6 MTL performance on three NER training datasets when combined with different auxiliary datasets 83
7.7 Correlation between various similarity measures and the change in NER F1 score using multi-task learning 86

A.1 NER F1 score mean and standard deviation across three runs with different random seeds using the neural network model with a Softmax classifier. . 128

Bibliography

Akbik, Alan, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf (2019). „FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP“. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 54–59. DOI: 10.18653/v1/N19-4010. URL: https://www.aclweb.org/anthology/N19-4010.

Akbik, Alan, Duncan Blythe, and Roland Vollgraf (2018). „Contextual String Embeddings for Sequence Labeling“. In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 1638–1649. URL: https://www.aclweb.org/anthology/C18-1139.

Amigó, Enrique, Julio Gonzalo, Javier Artiles, and Felisa Verdejo (2009). „A Comparison of Extrinsic Clustering Evaluation Metrics Based on Formal Constraints“. In: *Information Retrieval* 12.4, pp. 461–486. ISSN: 1386-4564. DOI: 10.1007/s10791-008-9066-8. URL: http://dx.doi.org/10.1007/s10791-008-9066-8.

Augenstein, Isabelle, Sebastian Ruder, and Anders Søgaard (2018). „Multi-Task Learning of Pairwise Sequence Classification Tasks over Disparate Label Spaces“. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 1896–1906. DOI: 10.18653/v1/N18-1172. URL: https://www.aclweb.org/anthology/N18-1172.

Augenstein, Isabelle and Anders Søgaard (2017). „Multi-Task Learning of Keyphrase Boundary Classification“. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 341–346. DOI: 10.18653/v1/P17-2054. URL: https://www.aclweb.org/anthology/P17-2054.

Aumüller, Martin, Erik Bernhardsson, and Alexander Faithfull (2017). „ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms“. In: *10th International Conference on Similarity Search and Applications (SISAP 2017)*. Ed. by Christian Beecks, Felix Borutta, Peer Kröger, and Thomas Seidl. Munich, Germany: Springer International Publishing, pp. 34–49. ISBN: 978-3-319-68474-1.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). „Layer normalization“. In: *arXiv:1607.06450*.

Balasuriya, Dominic, Nicky Ringland, Joel Nothman, Tara Murphy, and James R. Curran (2009). „Named Entity Recognition in Wikipedia“. In: *Proceedings of the 2009 Workshop on The People's Web Meets NLP: Collaboratively Constructed Semantic Resources (People's Web)*. Suntec, Singapore: Association for Computational Linguistics, pp. 10–18.

Banerjee, Arindam, Inderjit S. Dhillon, Joydeep Ghosh, and Suvrit Sra (2005). „Clustering on the Unit Hypersphere Using Von Mises-Fisher Distributions“. In: *Journal of Machine Learning Research (JMLR)* 6, pp. 1345–1382. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1046920.1088718.

Baroni, Marco, Georgiana Dinu, and Germán Kruszewski (2014). „Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors“. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 238–247. DOI: 10.3115/v1/P14-1023. URL: https://www.aclweb.org/anthology/P14-1023.

Baum, Leonard (1972). „An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process“. In: *Inequalities* 3, pp. 1–8.

Baxter, Jonathan (1997). „A Bayesian/Information Theoretic Model of Learning to Learn via Multiple Task Sampling“. In: *Machine Learning* 28.1, pp. 7–39. ISSN: 1573-0565. DOI: 10.1023/A:1007327622663. URL: https://doi.org/10.1023/A:1007327622663.

– (2000). „A Model of Inductive Bias Learning“. In: *Journal of Artificial Intelligence Research (JAIR)* 12.1, pp. 149–198. ISSN: 1076-9757. URL: http://dl.acm.org/citation.cfm?id=1622248.1622254.

Ben-David, Shai and Reba Schuller (2003). „Exploiting Task Relatedness for Multiple Task Learning“. In: *Learning Theory and Kernel Machines*. Ed. by Bernhard Schölkopf and Manfred K. Warmuth. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 567–580. ISBN: 978-3-540-45167-9.

Bengio, Y., P. Simard, and P. Frasconi (1994). „Learning Long-term Dependencies with Gradient Descent is Difficult“. In: *IEEE Transactions on Neural Networks* 5.2, pp. 157–166. ISSN: 1045-9227. DOI: 10.1109/72.279181. URL: http://dx.doi.org/10.1109/72.279181.

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Jauvin (2003). „A neural probabilistic language model“. In: *Journal of Machine Learning Research (JMLR)* 3.Feb, pp. 1137–1155.

Benikova, Darina, Chris Biemann, and Marc Reznicek (2014). „NoSta-D Named Entity Annotation for German: Guidelines and Dataset“. In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. Reykjavik, Iceland: European Language Resources Association (ELRA), pp. 2524–2531. URL: http://www.lrec-conf.org/proceedings/lrec2014/pdf/276_Paper.pdf.

Bergstra, James and Yoshua Bengio (2012). „Random Search for Hyper-parameter Optimization“. In: *Journal of Machine Learning Research (JMLR)* 13, pp. 281–305. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2188385.2188395.

Bingel, Joachim and Anders Søgaard (2017). „Identifying beneficial task relations for multi-task learning in deep neural networks“. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, pp. 164–169. URL: http://aclweb.org/anthology/E17-2026.

Bjerva, Johannes (2017). „Will my auxiliary tagging task help? Estimating Auxiliary Tasks Effectivity in Multi-Task Learning“. In: *Proceedings of the 21st Nordic Conference on Computational Linguistics*. Gothenburg, Sweden: Association for Computational Linguistics, pp. 216–220. URL: https://www.aclweb.org/anthology/W17-0225.

Bjerva, Johannes, Barbara Plank, and Johan Bos (2016). „Semantic Tagging with Deep Residual Networks“. In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, pp. 3531–3541. URL: https://www.aclweb.org/anthology/C16-1333.

BNC Consortium (2007). *The British National Corpus, version 3 (BNC XML Edition)*. URL: http://www.natcorp.ox.ac.uk/.

Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov (2017). „Enriching Word Vectors with Subword Information“. In: *Transactions of the Association for Computational Linguistics* 5, pp. 135–146. DOI: 10.1162/tacl_a_00051. URL: https://www.aclweb.org/anthology/Q17-1010.

Bos, Johan, Valerio Basile, Kilian Evang, Noortje Venhuizen, and Johannes Bjerva (2017). „The Groningen Meaning Bank“. In: *Handbook of Linguistic Annotation*. Ed. by Nancy Ide and James Pustejovsky. Vol. 2. Springer, pp. 463–496.

Brejová, Broňa, Daniel G. Brown, and Tomáš Vinař (2007). „Advances in Hidden Markov Models for Sequence Annotation“. In: *Bioinformatics Algorithms*. John Wiley & Sons, Ltd. Chap. 4, pp. 55–91. ISBN: 9780470253441. DOI: 10.1002/9780470253441.ch4.

Caruana, Rich (1993). „Multitask Learning: A Knowledge-Based Source of Inductive Bias“. In: *Proceedings of the Tenth International Conference on Machine Learning*. Amherst, Massachusetts, USA, pp. 41–48.

– (1997). „Multitask Learning“. In: *Machine Learning* 28.1, pp. 41–75. ISSN: 0885-6125. DOI: 10.1023/A:1007379606734. URL: https://doi.org/10.1023/A:1007379606734.

Changpinyo, Soravit, Hexiang Hu, and Fei Sha (2018). „Multi-Task Learning for Sequence Tagging: An Empirical Study“. In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 2965–2977. URL: http://aclweb.org/anthology/C18-1251.

Cho, Kyunghyun, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). „Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation“. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. URL: https://www.aclweb.org/anthology/D14-1179.

Collobert, Ronan and Jason Weston (2008). „A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning“. In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: ACM, pp. 160–167. ISBN: 978-1-60558-205-4. DOI: 10.1145/1390156.1390177. URL: http://doi.acm.org/10.1145/1390156.1390177.

Cooijmans, Tim, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville (2016). „Recurrent Batch Normalization“. In: *arXiv:1603.09025*.

Coppersmith, D. and S. Winograd (1987). „Matrix Multiplication via Arithmetic Progressions“. In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: ACM, pp. 1–6. ISBN: 0-89791-221-7. DOI: 10.1145/28395.28396. URL: http://doi.acm.org/10.1145/28395.28396.

Cover, Thomas M. and Joy A. Thomas (2006). *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. New York, New York, USA: Wiley-Interscience. ISBN: 0471241954.

Culurciello, Eugenio (Apr. 13, 2018). *The fall of RNN / LSTM*. URL: https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0 (visited on 10/22/2019).

Davie, Alexander M. and Andrew J. Stothers (2013). „Improved bound for complexity of matrix multiplication“. In: *Proceedings of the Royal Society of Edinburgh: Section A Mathematics* 143.2, pp. 351–369. DOI: 10.1017/S0308210511001648.

Dempster, Arthur P., Nan M. Laird, and Donald B. Rubin (1977). „Maximum Likelihood from Incomplete Data via the EM Algorithm“. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1, pp. 1–38. ISSN: 00359246.

Derczynski, Leon (2016). „Complementarity, F-score, and NLP Evaluation“. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. Portorož, Slovenia: European Language Resources Association (ELRA), pp. 261–266. URL: https://www.aclweb.org/anthology/L16-1040.

Derczynski, Leon, Eric Nichols, Marieke van Erp, and Nut Limsopatham (2017). „Results of the WNUT2017 Shared Task on Novel and Emerging Entity Recognition“. In: *Proceedings of the 3rd Workshop on Noisy User-generated Text*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 140–147. DOI: 10.18653/v1/W17-4418. URL: https://www.aclweb.org/anthology/W17-4418.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding“. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. URL: https://www.aclweb.org/anthology/N19-1423.

Dongarra, Jack, Jeremy Du Croz, Sven Hammarling, and Iain S. Duff (1990). „A Set of Level 3 Basic Linear Algebra Subprograms“. In: *ACM Transactions on Mathematical Software* 16.1, pp. 1–17. ISSN: 0098-3500. DOI: 10.1145/77626.79170. URL: http://doi.acm.org/10.1145/77626.79170.

Elekes, Ábel, Martin Schäler, and Klemens Böhm (2017). „On the Various Semantics of Similarity in Word Embedding Models“. In: *Proceedings of the 17th ACM/IEEE Joint Conference on Digital Libraries*. JCDL '17. Toronto, Ontario, Canada: IEEE Press, pp. 139–148. ISBN: 978-1-5386-3861-3. URL: http://dl.acm.org/citation.cfm?id=3200334.3200350.

Faruqui, Manaal and Sebastian Padó (2010). „Training and Evaluating a German Named Entity Recognizer with Semantic Generalization“. In: *Semantic Approaches in Natural Language Processing: Proceedings of the 10th Conference on Natural Language Processing, KONVENS 2010*. Saarbrücken, Germany, pp. 129–133.

Freund, Yoav and Robert E Schapire (1999). „Large margin classification using the perceptron algorithm“. In: *Machine learning* 37.3, pp. 277–296.

Gardner, Matt, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer (2018). „AllenNLP: A Deep Semantic Natural Language Processing Platform“. In: *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*. Melbourne, Australia: Association for Computational Linguistics, pp. 1–6. DOI: 10.18653/v1/W18-2501. URL: https://www.aclweb.org/anthology/W18-2501.

Gers, Felix A. and Jürgen Schmidhuber (2000). „Recurrent nets that time and count“. In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*. Vol. 3. Como, Italy: IEEE, pp. 189–194. ISBN: 0-7695-0619-4. DOI: 10.1109/ijcnn.2000.861302. URL: http://dx.doi.org/10.1109/ijcnn.2000.861302.

Glorot, Xavier and Yoshua Bengio (2010). „Understanding the difficulty of training deep feedforward neural networks“. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press. ISBN: 9780262035613. URL: http://www.deeplearningbook.org.

Harris, Zellig S (1954). „Distributional structure“. In: *Word* 10.2-3, pp. 146–162.

Hashimoto, Kazuma, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher (2017). „A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks“. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 1923–1933. DOI: 10.18653/v1/D17-1206. URL: https://www.aclweb.org/anthology/D17-1206.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). „Long Short-Term Memory“. In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Hoerl, Arthur E and Robert W Kennard (1970). „Ridge regression: Biased estimation for nonorthogonal problems“. In: *Technometrics* 12.1, pp. 55–67.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). „Multilayer feedforward networks are universal approximators“. In: *Neural Networks* 2.5, pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8.

Howard, Jeremy and Sebastian Ruder (2018). „Universal Language Model Fine-tuning for Text Classification“. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 328–339. URL: https://www.aclweb.org/anthology/P18-1031.

Huang, Jianyu, Tyler M. Smith, Greg M. Henry, and Robert A. van de Geijn (2016). „Strassen's Algorithm Reloaded“. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '16. Salt Lake

City, Utah: IEEE Press, 59:1–59:12. ISBN: 978-1-4673-8815-3. URL: http://dl.acm.org/citation.cfm?id=3014904.3014983.

Huang, Jianyu, Chenhan D. Yu, and Robert A. van de Geijn (2018). „Implementing Strassen's Algorithm with CUTLASS on NVIDIA Volta GPUs". In: *arXiv:1808.07984*.

Intel Corporation (2009). *Intel Math Kernel Library. Reference Manual*. Santa Clara, California, USA: Intel Corporation. ISBN: 630813-054US. URL: https://software.intel.com/en-us/mkl.

Ioffe, Sergey and Christian Szegedy (2015). „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, pp. 448–456. URL: http://dl.acm.org/citation.cfm?id=3045118.3045167.

Joulin, Armand, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov (2017). „Bag of Tricks for Efficient Text Classification". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, pp. 427–431. URL: https://www.aclweb.org/anthology/E17-2068.

Jurafsky, Daniel and James H. Martin (2009). *Speech and Language Processing (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. ISBN: 0131873210.

Kaiser, Lukasz, Aidan N. Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit (2017). „One Model To Learn Them All". In: *arXiv:1706.05137*. URL: http://arxiv.org/abs/1706.05137.

Karpathy, Andrej (May 21, 2015). *The Unreasonable Effectiveness of Recurrent Neural Networks*. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness (visited on 10/22/2019).

Kendall, Alex, Yarin Gal, and Roberto Cipolla (2018). „Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7482–7491. DOI: 10.1109/CVPR.2018.00781.

Kim, Joo-Kyung, Young-Bum Kim, Ruhi Sarikaya, and Eric Fosler-Lussier (2017). „Cross-Lingual Transfer Learning for POS Tagging without Cross-Lingual Resources". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 2832–2838. DOI: 10.18653/v1/D17-1302. URL: https://www.aclweb.org/anthology/D17-1302.

Kim, Young-Bum, Karl Stratos, Ruhi Sarikaya, and Minwoo Jeong (2015). „New Transfer Learning Techniques for Disparate Label Sets". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 473–482. DOI: 10.3115/v1/P15-1046. URL: http://aclweb.org/anthology/P15-1046.

Kiperwasser, Eliyahu and Yoav Goldberg (2016). „Simple and accurate dependency parsing using bidirectional LSTM feature representations". In: *Transactions of the Association for Computational Linguistics* 4, pp. 313–327.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). „Imagenet classification with deep convolutional neural networks“. In: *Advances in neural information processing systems*. Lake Tahoe, Nevada, USA, pp. 1097–1105.

Kvalseth, T. O. (1987). „Entropy and Correlation: Some Comments“. In: *IEEE Transactions on Systems, Man, and Cybernetics* 17.3, pp. 517–519. ISSN: 0018-9472. DOI: 10.1109/TSMC.1987.4309069.

Lafferty, John D., Andrew McCallum, and Fernando C. N. Pereira (2001). „Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data“. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. San Francisco, California, USA: Morgan Kaufmann Publishers Inc., pp. 282–289. ISBN: 1-55860-778-1. URL: http://dl.acm.org/citation.cfm?id=645530.655813.

Lample, Guillaume, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer (2016). „Neural Architectures for Named Entity Recognition“. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 260–270. DOI: 10.18653/v1/N16-1030. URL: https://www.aclweb.org/anthology/N16-1030.

Laurent, César, Gabriel Pereyra, Philémon Brakel, Ying Zhang, and Yoshua Bengio (2016). „Batch normalized recurrent neural networks“. In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Shanghai, China, pp. 2657–2661. DOI: 10.1109/ICASSP.2016.7472159.

Le Gall, François (2014). „Powers of Tensors and Fast Matrix Multiplication“. In: *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. ISSAC '14. Kobe, Japan: ACM, pp. 296–303. ISBN: 978-1-4503-2501-1. DOI: 10.1145/2608628.2608664. URL: http://doi.acm.org/10.1145/2608628.2608664.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). „Deep learning“. In: *Nature* 521, pp. 436–444. DOI: 10.1038/nature14539.

Li, Mu, Tong Zhang, Yuqiang Chen, and Alexander J. Smola (2014). „Efficient Mini-batch Training for Stochastic Optimization“. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: ACM, pp. 661–670. ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623612. URL: http://doi.acm.org/10.1145/2623330.2623612.

Li, Wen, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin (2019). „Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement“. In: *IEEE Transactions on Knowledge and Data Engineering* (Early Access), pp. 1–1. DOI: 10.1109/TKDE.2019.2909204.

Li, Xiangang and Xihong Wu (2015). „Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition“. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. Brisbane, Australia, pp. 4520–4524.

Liu, Liyuan, Jingbo Shang, Frank F. Xu, Xiang Ren, Huan Gui, Jian Peng, and Jiawei Han (2018). „Empower Sequence Labeling with Task-Aware Neural Language Model“. In: *AAAI Conference on Artificial Intelligence*. New Orleans, Louisiana USA.

Liu, Zhenqiu, Zhongmin Guo, and Ming Tan (2008). „Constructing Tumor Progression Pathways and Biomarker Discovery with Fuzzy Kernel Kmeans and DNA Methylation Data“. In: *Cancer informatics* 6, pp. 1–7. DOI: 10.1177/117693510800600007.

Long, Mingsheng, Zhangjie Cao, Jianmin Wang, and Philip S. Yu (2017). „Learning Multiple Tasks with Multilinear Relationship Networks“. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 1593–1602. URL: http://papers.nips.cc/paper/6757-learning-multiple-tasks-with-multilinear-relationship-networks.pdf.

Lu, Yongxi, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, Tara Javidi, and Rogério Schmidt Feris (2016). „Fully-Adaptive Feature Sharing in Multi-Task Networks with Applications in Person Attribute Classification“. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1131–1140.

Ma, Xuezhe and Eduard Hovy (2016). „End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF“. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 1064–1074. DOI: 10.18653/v1/P16-1101. URL: https://www.aclweb.org/anthology/P16-1101.

Malkov, Y. A. and D. A. Yashunin (2018). „Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2018.2889473.

Manning, Christopher D. and Hinrich Schütze (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts, USA: MIT Press. ISBN: 0-262-13360-1.

Marcus, Mitchell P., Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor (1999). *Penn Treebank 3*. Philadelphia. URL: https://catalog.ldc.upenn.edu/LDC99T42.

Martínez Alonso, Héctor and Barbara Plank (2017). „When is multitask learning effective? Semantic sequence prediction under varying data conditions“. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, pp. 44–53. URL: https://www.aclweb.org/anthology/E17-1005.

Masters, Dominic and Carlo Luschi (2018). „Revisiting small batch training for deep neural networks“. In: *arXiv:1804.07612*.

Maurer, Andreas, Massimiliano Pontil, and Bernardino Romera-Paredes (2016). „The Benefit of Multitask Representation Learning“. In: *Journal of Machine Learning Research (JMLR)* 17.1, pp. 2853–2884. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2946645.3007034.

McCallum, Andrew, Dayne Freitag, and Fernando C. N. Pereira (2000). „Maximum Entropy Markov Models for Information Extraction and Segmentation“. In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML '00. San Francisco, California, USA: Morgan Kaufmann Publishers Inc., pp. 591–598. ISBN: 1-55860-707-2.

McDonald, Ryan, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia

Bedini, Núria Bertomeu Castelló, and Jungmee Lee (2013). „Universal Dependency Annotation for Multilingual Parsing“. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 92–97. URL: https://www.aclweb.org/anthology/P13-2017.

Meilă, Marina (2003). „Comparing Clusterings by the Variation of Information“. In: *Learning Theory and Kernel Machines*. Ed. by Bernhard Schölkopf and Manfred K. Warmuth. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 173–187. ISBN: 978-3-540-45167-9.

– (2005). „Comparing Clusterings: An Axiomatic View“. In: *Proceedings of the 22Nd International Conference on Machine Learning*. ICML '05. Bonn, Germany: ACM, pp. 577–584. ISBN: 1-59593-180-5. DOI: 10.1145/1102351.1102424. URL: http://doi.acm.org/10.1145/1102351.1102424.

– (2007). „Comparing clusterings—an information based distance“. In: *Journal of Multivariate Analysis* 98.5, pp. 873–895. ISSN: 0047-259X. DOI: https://doi.org/10.1016/j.jmva.2006.11.013.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean (2013a). „Efficient Estimation of Word Representations in Vector Space“. In: *1st International Conference on Learning Representations (ICLR), Workshop Track Proceedings*. Scottsdale, Arizona, USA. URL: http://arxiv.org/abs/1301.3781.

Mikolov, Tomas, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin (2018). „Advances in Pre-Training Distributed Word Representations“. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA). URL: https://www.aclweb.org/anthology/L18-1008.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean (2013b). „Distributed Representations of Words and Phrases and Their Compositionality“. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., pp. 3111–3119.

Minsky, Marvin (1987). *Perceptrons: An Introduction to Computational Geometry, Expanded Edition*. The MIT Press. ISBN: 0262631113. URL: https://www.xarg.org/ref/a/0262631113/.

Misra, Ishan, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert (2016). „Cross-Stitch Networks for Multi-task Learning“. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3994–4003.

Nickolls, John, Ian Buck, Michael Garland, and Kevin Skadron (2008). „Scalable Parallel Programming with CUDA“. In: *Queue* 6.2, pp. 40–53. ISSN: 1542-7730. DOI: 10.1145/1365490.1365500. URL: http://doi.acm.org/10.1145/1365490.1365500.

NVIDIA Corporation (2007). *NVIDIA cuBLAS library*. URL: https://developer.nvidia.com/cublas (visited on 08/30/2019).

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). „On the Difficulty of Training Recurrent Neural Networks“. In: *Proceedings of the 30th International Conference*

on International Conference on Machine Learning. Vol. 28. ICML'13. Atlanta, Georgia, USA: JMLR.org, pp. III-1310–III-1318.

Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer (2017). „Automatic Differentiation in PyTorch". In: *31st Conference on Neural Information Processing Systems (NIPS 2017), Autodiff Workshop: The future of gradient-based machine learning software and techniques*. Long Beach, California, USA, pp. 1–4.

Pennington, Jeffrey, Richard Socher, and Christopher Manning (2014). „Glove: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://www.aclweb.org/anthology/D14-1162.

Peters, Matthew, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer (2018). „Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: https://www.aclweb.org/anthology/N18-1202.

Plank, Barbara, Anders Søgaard, and Yoav Goldberg (2016). „Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 412–418. DOI: 10.18653/v1/P16-2067. URL: https://www.aclweb.org/anthology/P16-2067.

Rabiner, Lawrence R. (1989). „A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2, pp. 257–286. ISSN: 0018-9219. DOI: 10.1109/5.18626.

Reimers, Nils and Iryna Gurevych (2017). „Optimal Hyperparameters for Deep LSTM-Networks for Sequence Labeling Tasks". In: *arXiv:1707.06799*. URL: https://arxiv.org/abs/1707.06799.

Remus, Steffen and Chris Biemann (2013). „Three Knowledge-Free Methods for Automatic Lexical Chain Extraction". In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia, USA: Association for Computational Linguistics, pp. 989–999. URL: https://www.inf.uni-hamburg.de/en/inst/ab/lt/publications/2013-remusetal-naacl.pdf.

Rosenblatt, Frank (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.

Rosenfeld, Ronald (2000). „Two decades of statistical language modeling: Where do we go from here?" In: *Proceedings of the IEEE* 88.8, pp. 1270–1278.

Ruder, Sebastian (2017). „An Overview of Multi-Task Learning in Deep Neural Networks". In: *arXiv:1706.05098*. URL: http://arxiv.org/abs/1706.05098.

Ruder, Sebastian, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard (2019). „Latent Multi-task Architecture Learning". In: *Proceedings of the Thirty-Third Conference*

on Artificial Intelligence (AAAI-2019). Honolulu, Hawaii, USA: Association for the Advancement of Artificial Intelligence, pp. 4822–4829. DOI: https://doi.org/10.1609/aaai.v33i01.33014822.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). „Learning representations by back-propagating errors". In: *Nature* 323, pp. 533–536. DOI: 10.1038/323533a0.

Salinas Alvarado, Julio Cesar, Karin Verspoor, and Timothy Baldwin (2015). „Domain Adaption of Named Entity Recognition to Support Credit Risk Assessment". In: *Proceedings of the Australasian Language Technology Association Workshop 2015*. Parramatta, Australia, pp. 84–90.

Schmidhuber, Jürgen (2015). „Deep learning in neural networks: An overview". In: *Neural networks* 61, pp. 85–117.

Schulz, Claudia, Steffen Eger, Johannes Daxenberger, Tobias Kahse, and Iryna Gurevych (2018). „Multi-Task Learning for Argumentation Mining in Low-Resource Settings". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 35–41. DOI: 10.18653/v1/N18-2006. URL: http://aclweb.org/anthology/N18-2006.

Silveira, Natalia, Timothy Dozat, Marie-Catherine de Marneffe, Samuel Bowman, Miriam Connor, John Bauer, and Chris Manning (2014). „A Gold Standard Dependency Corpus for English". In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. Reykjavik, Iceland: European Language Resources Association (ELRA), pp. 2897–2904. URL: http://www.lrec-conf.org/proceedings/lrec2014/pdf/1089_Paper.pdf.

Søgaard, Anders and Yoav Goldberg (2016). „Deep multi-task learning with low level tasks supervised at lower layers". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 231–235. DOI: 10.18653/v1/P16-2038. URL: http://aclweb.org/anthology/P16-2038.

Srivastava, Nitish, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). „Dropout: a simple way to prevent neural networks from overfitting". In: *Journal of Machine Learning Research (JMLR)* 15.1, pp. 1929–1958.

Stone, Mervyn (1974). „Cross-validatory choice and assessment of statistical predictions". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 36.2, pp. 111–133.

Strassen, Volker (1969). „Gaussian Elimination is Not Optimal". In: *Numerische Mathematik* 13.4, pp. 354–356. ISSN: 0029-599X. DOI: 10.1007/BF02165411. URL: http://dx.doi.org/10.1007/BF02165411.

Strehl, Alexander and Joydeep Ghosh (2003). „Cluster Ensembles — a Knowledge Reuse Framework for Combining Multiple Partitions". In: *Journal of Machine Learning Research (JMLR)* 3, pp. 583–617. ISSN: 1532-4435. DOI: 10.1162/153244303321897735. URL: https://doi.org/10.1162/153244303321897735.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (2014). „Sequence to Sequence Learning with Neural Networks". In: *Proceedings of the 27th International Conference on Neural*

Information Processing Systems - Volume 2. NIPS'14. Montreal, Canada: MIT Press, pp. 3104–3112.

Sutton, Charles and Andrew McCallum (2012). „An Introduction to Conditional Random Fields". In: *Foundations and Trends in Machine Learning* 4.4, pp. 267–373. ISSN: 1935-8237. DOI: 10.1561/2200000013. URL: http://dx.doi.org/10.1561/2200000013.

Tjong Kim Sang, Erik F. and Fien De Meulder (2003). „Introduction to the CoNLL-2003 Shared Task: Language-independent Named Entity Recognition". In: *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*. CONLL '03. Edmonton, Canada: Association for Computational Linguistics, pp. 142–147. DOI: 10.3115/1119176.1119195. URL: https://doi.org/10.3115/1119176.1119195.

Van Zee, Field G. and Robert A. van de Geijn (2015). „BLIS: A Framework for Rapidly Instantiating BLAS Functionality". In: *ACM Transactions on Mathematical Software* 41.3, 14:1–14:33. URL: http://doi.acm.org/10.1145/2764454.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). „Attention is all you need". In: *Advances in neural information processing systems*. Long Beach, California, USA, pp. 5998–6008.

Vinh, Nguyen Xuan, Julien Epps, and James Bailey (2010). „Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance". In: *Journal of Machine Learning Research (JMLR)* 11, pp. 2837–2854. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1756006.1953024.

Weber, Roger, Hans-Jörg Schek, and Stephen Blott (1998). „A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces". In: *Proceedings of the 24rd International Conference on Very Large Data Bases*. VLDB '98. San Francisco, California, USA: Morgan Kaufmann Publishers Inc., pp. 194–205. ISBN: 1-55860-566-5. URL: http://dl.acm.org/citation.cfm?id=645924.671192.

Weischedel, Ralph, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, Mohammed El-Bachouti, Robert Belvin, and Ann Houston (2013). *OntoNotes Release 5.0 LDC2013T19*. Ed. by Linguistic Data Consortium. URL: https://catalog.ldc.upenn.edu/LDC2013T19.

Whaley, R. Clint and Jack J. Dongarra (1998). „Automatically tuned linear algebra software". In: *SC'98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing*. IEEE. Orlando, Florida, USA, pp. 38–38.

Williams, Virginia Vassilevska (2012). „Multiplying Matrices Faster Than Coppersmith-winograd". In: *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*. STOC '12. New York, New York, USA: ACM, pp. 887–898. ISBN: 978-1-4503-1245-5. DOI: 10.1145/2213977.2214056. URL: http://doi.acm.org/10.1145/2213977.2214056.

Xianyi, Zhang, Wang Qian, and Zhang Yunquan (2012). „Model-driven Level 3 BLAS Performance Optimization on Loongson 3A Processor". In: *Proceedings of the 2012 IEEE 18th International Conference on Parallel and Distributed Systems*. ICPADS '12. Washington, DC, USA: IEEE Computer Society, pp. 684–691. ISBN: 978-0-7695-4903-3.

DOI: 10.1109/ICPADS.2012.97. URL: http://dx.doi.org/10.1109/ICPADS.2012.97.

Xiao, Han (2018). *bert-as-service*. https://github.com/hanxiao/bert-as-service.

Xing, Eric (Oct. 29, 2007). *Probabilistic Graphical Models. Hidden Markov Model and Conditional Random Fields*. Carnegie Mellon School of Computer Science. URL: http://www.cs.cmu.edu/~epxing/Class/10708-07/Slides/lecture12-CRF-HMM.pdf (visited on 10/21/2019).

Yang, Jie, Shuailong Liang, and Yue Zhang (2018). „Design Challenges and Misconceptions in Neural Sequence Labeling". In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 3879–3889. URL: https://www.aclweb.org/anthology/C18-1327.

Yang, Zhilin, Ruslan Salakhutdinov, and William W. Cohen (2017). „Transfer Learning for Sequence Tagging with Hierarchical Recurrent Networks". In: *5th International Conference on Learning Representations, ICLR 2017, Conference Track Proceedings*. Toulon, France. URL: https://openreview.net/forum?id=ByxpMd9lx.

Yao, Yiyu (2003). „Information-Theoretic Measures for Knowledge Discovery and Data Mining". In: *Entropy Measures, Maximum Entropy Principle and Emerging Applications*. Ed. by Karmeshu. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 115–136. ISBN: 978-3-540-36212-8. DOI: 10.1007/978-3-540-36212-8_6. URL: https://doi.org/10.1007/978-3-540-36212-8_6.

Zipf, George (1935). *The Psychobiology of Language*. Oxford, England: Houghton-Mifflin.

A Appendices

A.1 Neural MTL system implementation details

This neural network implementation is designed for multi-task learning on sequence tagging problems. It is implemented in Python on top of the PyTorch[1] (currently stable version 1.3) and optimized for training speed to perform large numbers of experiments. There are separate functionalities for data loading, configuration, training, hyperparameter search apart from the actual neural network model.

Usage

Run `python main.py -h` to print the commandline help. An example configuration file and some dummy data can be found in the `example` folder. To run the example configuration on the dummy data, use `python main.py -c example/example_config.yaml`. This will perform training with hyperparameter search via validation on the development set and finally evaluate the best model on the test set.

Model architecture

The model follows the common sequence tagging network architecture with a multi-layered, bidirectional Recurrent Neural Network (RNN) as its core. Instead of the classic LSTM, a Gated Recurrent Unit (GRU) is used in the implementation due to its faster run time speed. The input for the GRU comes from a concetenation of word embeddings and character features, which are learned by another bidirectional GRU. The hidden state of the word-level GRU is transformed via a linear layer to the label space. To obtain label probabilities as the last layer either a Softmax or a Conditional Random Field (CRF) can be used.

Multi-task learning

Multiple datasets with possibly different tag sets can be used as auxiliary data. In order to enable this, each dataset has its own linear transform layer and optionally CRF depending on the configuration. All other parameters in the network are shared between the various datasets. During the training process, each batch contains only data from one dataset, which make the computation efficient as the data must not be split at the

[1] `https://pytorch.org`

last layers. Batches from different datasets are processed interleaved, so that every each dataset is fully used.

Implementation overview

The separate functionalities are organized in the following files:

Data loading `data.py` features a class `Data` that reads input data separated by tabs and newlines. It creates index tensors for to build character and word embeddings via two simple dictionaries. It supports multiple tag sets and differentiates between training and development / test data. To add support for another input file format, you would have to change the `__iter_file` function or overwrite it in a sub class.

Configuration `config.py` contains a single class `Config` to parse commandline arguments and configuration files in `YAML` or `JSON` format. Commandline arguments overwrite the settings from the chosen configuration file to quickly try changes and enable easy automation. Configurable are general settings (logging, threads, configuration file, multi-task learning yes/no, random seed etc.), input files (training, development, test, model storage path, etc.) and typical hyperparameters.

Utilities `util.py` is the home of some utility functions such as the different strategies to combine batches of training and auxiliary data. These are implemented with efficient Python generators and iterators.

Trainer `trainer.py` has a single class `Trainer` that instantiates the model, makes the batch of tensors and runs the training loop. It further contains functions for prediction, scoring, validation, test, check-pointing and early stopping.

Hyperparameter `hyperparameter.py` features a single class `HyperParameter` that performs grid search across all combination of hyperparameters. The values to be tried are read from a separate configuration file.

Model `model.py` features reusable neural network building blocks. Every operation is fully batched and tuned for run time performance to enable fast training on a GPU. On the top-level, there is the class `SequenceTagger` that combines the other blocks into a functional neural sequence tagger. It uses the default PyTorch embedding module to learn word embeddings. The `WordRNN` contains the primary multi-layered bidirectional GRU. Dropout is applied between the layers and on the final output. Packed processing is used for efficiency. The `CharRNN` uses self-learning character embeddings to convert the index tensors to a dense representation. A single-layer bidirectional GRU extracts character features from the embeddings. Dropout is used for regularization. The operations are performed in a packed manner to increase efficiency. The `CRF` module features a

linear-chain CRF implementation. It automatically chooses between the computation of Viterbi loss for training and Viterbi decoding for prediction.

Main `main.py` contains the logic to run the `Trainer` class once or multiple times (when performing hyperparameter search). If an existing model is found at the configured location, training is skipped to directly perform a validation on the test set.

External libraries

Only three external libraries are used apart from Python's standard library:

- NumPy `https://numpy.org`
- PyTorch `https://pytorch.org`
- PyYAML `https://pyyaml.org`

A.2 Dataset similarity tool implementation details

This program computes the similarity of two annotated sequence tagging datasets based on the contained words and their labels. The designated use case is to ease and speed up the tedious process of selecting suitable auxiliary training data for neural networks using multi-task learning to augment the primary training with auxiliary data. Knowing the similarity between the training dataset and different auxiliary datasets quickly allows selecting the most similar dataset, which should also provide the most improvement of the neural network's performance on the main task.

The program computes multiple similarity measures at once. There are no restrictions on the tagsets used in the datasets. Arbitrary sequence tagging task / datasets can be compared. As of now, the similarity computation is only working well for tasks where each token is tagged individually, e.g. part-of-speech (POS) tagging, or the grouped tokens are short, e.g. named entity recognition (NER).

Installation

Portable, stand-alone binary builds are available for download on the GitHub release page. Extract the archive and copy the `seq-tag-sim` file into a directory on your `PATH`. Alternatively, call the program via its absolute or relative path.

Usage

Run `seq-tag-sim -h` to print the commandline help. The general usage is straightforward. Run `seq-tag-sim path/to/dataset1 path/to/dataset2` to compare dataset 1 with dataset 2 and compute various similarity measures, which are written to the standard output stream. In case the automatic data format selection (based on file types) fails,

use the `-f` option once or twice to manually select the input format. If your datasets are split across multiple files, use shell glob operations to select the files. It is now necessary to distinguish both datasets by placing an `--` in between the to datasets. The example `seq-tag-sim -f bncPOS -f ptbPOS dataset1/*.xml -- dataset2/*.pos` shows how to compare multiple XML files from the British National Corpus with some files in the Penn Treebank POS tagging format. Windows users can use the `--pattern` option to select files with glob-like selectors.

Advanced installation and usage

Optional, advanced features are to use word embeddings to improve the quality of the similarity calculation. To use advanced features, additional software and data may be required. Depending on the type of embedding to be used

- download a fastText (Joulin et al., 2017) model from `https://fasttext.cc`
- install AllenNLP (`https://github.com/allenai/allennlp`) in your active Python environment to use contextual ELMo (Peters et al., 2018) embeddings
- install bert-as-a-service (Xiao, 2018) from `https://github.com/hanxiao/bert-as-service` in your active Python environment, download a suitable model from `https://github.com/google-research/bert` and start the service to use contextual BERT (Devlin et al., 2019) embeddings.

To use non-contextual word embeddings, i.e. fastText, supply the `-e path/to/emb.bin` option when running the program. As the fastText library takes some time to load the model, this may add considerable run time overhead when comparing small datasets. The preferred option, is to use BERT embeddings. To do so, run `seq-tag-sim -c bert`. If the bert-as-a-serice server is not running on the same computer, use the `-e` option to set the embedding server's network address.

Functioning principle

The overlapping vocabulary between the two datasets builds the bridge to compare the corresponding labels of these words. Without contextual embeddings, the general workflow is the following:

1. Read a dataset and count for each unique word, how often it is tagged with each label
2. Match and compare words of both datasets
   a) If a word from the fist dataset is not contained in the second dataset and fastText embeddings are used, the most similar word in second dataset according to the word vectors' cosine similarity is chosen.

b) The counts how often a word has a certain label are combined from both datasets by increasing the counts at the label-pair's position in a global contingency table. In total, there are eight slightly different methods to combine the label counts.

3. Once all words are processed, the contingency table with the label counts acts as a probabilistic mapping between both tagsets. For example, the counts for the tag `NOUN` from dataset 1 may correspond to 85% to `NN` from dataset 2. The remaining 15% could be distributed in roughly equal parts over other labels from dataset 2. Based on this label count contingency table, multiple information theoretic measures are calculated.

The information theoretic measures include e.g. entropy, cross-entropy, mutual information, variation of information and multiple variants of normalized mutual information. They represent the similarity of the two input datasets. When contextual embeddings (BERT or ELMo) are active, individual tokens are matched and their the counts at their labels' position is increased. The matching of tokens works by computing all most similar vector pairs.

Implementation overview

The source code is structured into the main application and independently usable subpackages. The main functionality is in the `source` folder with `app.d` defining the entry point. In subfolders are the implementations of the vocabulary overlap approach (`word.d`), the token-based approach using contextual embeddings (`token.d`) and the information theoretic measures (`measures.d`).

The top-level folder `subpackages` contains various additional functionalities. Of these subpackages, only `reader` and `util` are essential. File readers for various common sequence tagging file formats can be found in the `reader` subpackage. As its name suggests, the `util` subpackage contains utility functions and structures. The remaining subpackages are all related to the option word embeddings.

The `blas` subpackage contains an efficient functionality to compute the most similar vector pairs between two huge arrays of vectors. It uses a batched matrix multiplication implementation, which can efficiently multiply matrices that do not fit into memory. Along with the computation of these batches, the maximal similar vectors are found. An API-wise identical implementation for CUDA exists in the `cuda` subpackage. It can automatically divide the computation up across multiple GPUs, which decreases the run time for large datasets of 200 000 tokens or more. The `embedding` subpackage contains structures and functions to use the three different embeddings libraries resp. services with a uniform API. The `fasttext` subpackage is home to the external fastText source code and some custom wrapper code to make the usage as a library instead of commandline program possible.

Building from source

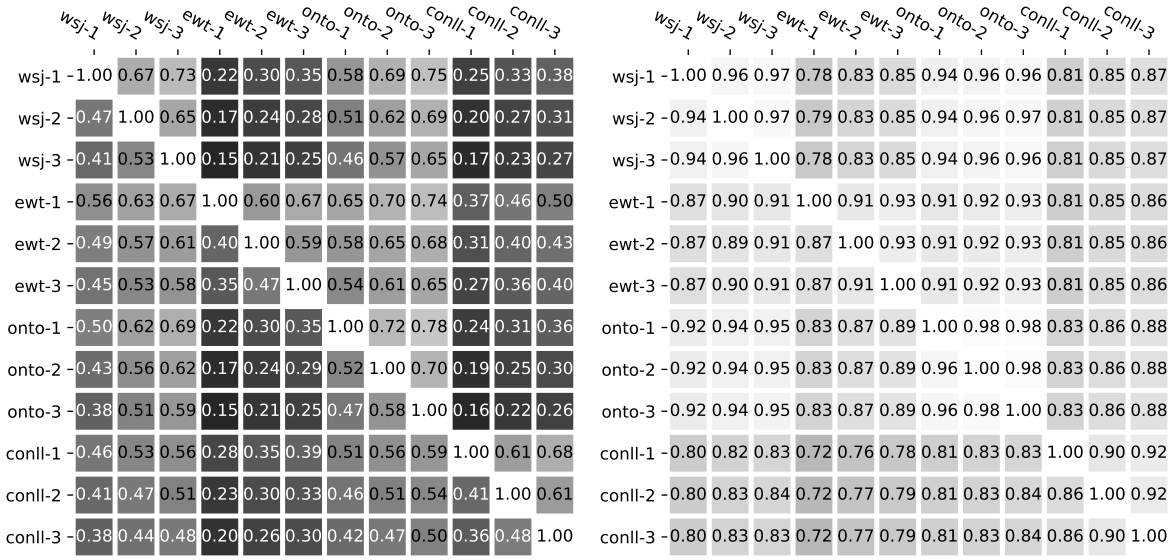
Builder the program from source should be possible on any most current POSIX-like systems (e.g. Linux, FreeBSD, MacOS) and Windows. To build the software from source, first clone this repository. A recent D language (`https://dlang.org`) compiler needs to be installed, e.g. DMD (`https://dlang.org/download.html#dmd`) (version 2.086.1 or higher) or LDC (`https://github.com/ldc-developers/ldc#installation`) (tested with version 1.16.0 and higher). If the D compiler installation does not include DUB (`https://dub.pm/getting_started`) (the D package manager), downloading and installing DUB separately is necessary. Further, the system's default compiler C/C++ compiler (e.g. gcc or clang) and linker has to be installed. Building the basic version of the program without support for word embeddings is straightforward: Run `dub build -b release` to produce the `seq-tag-sim` binary.

To build with all word embeddings, additional steps are required. Run `git submodule update -init -recursive` to get the referenced fastText library sources. In addition, Python and the development version of the ZeroMQ (`https://zeromq.org`) library `libzmq` needs to be installed on the build system. Next, run `dub build -c embedding -b release` to the produce the runnable binary. Note that the use of contextual embeddings greatly increases the run time as a naïve approach of word vector comparison is used. To mitigate this problem, additional libraries are required.

If the system has a CUDA-capable GPU, it can be leveraged to speed up the similarity computation process by an order of magnitude. This requires the NVIDAI CUDA Toolkit (in version 10.1, available at `https://developer.nvidia.com/cuda-toolkit`) to be installed and configured correctly. Run `dub build -c cuda -b release` to build an optimized version using CUDA for word vector operation acceleration. If CUDA cannot be used, installation of the Intel Math Kernel Library (MKL) is recommended, which can be found at `https://software.intel.com/en-us/mkl`. After sourcing the environment variables with e.g. `source ~/intel/bin/compilervars.sh intel64`, compiling the software with MKL can be done with `dub build -c blas -b release`.

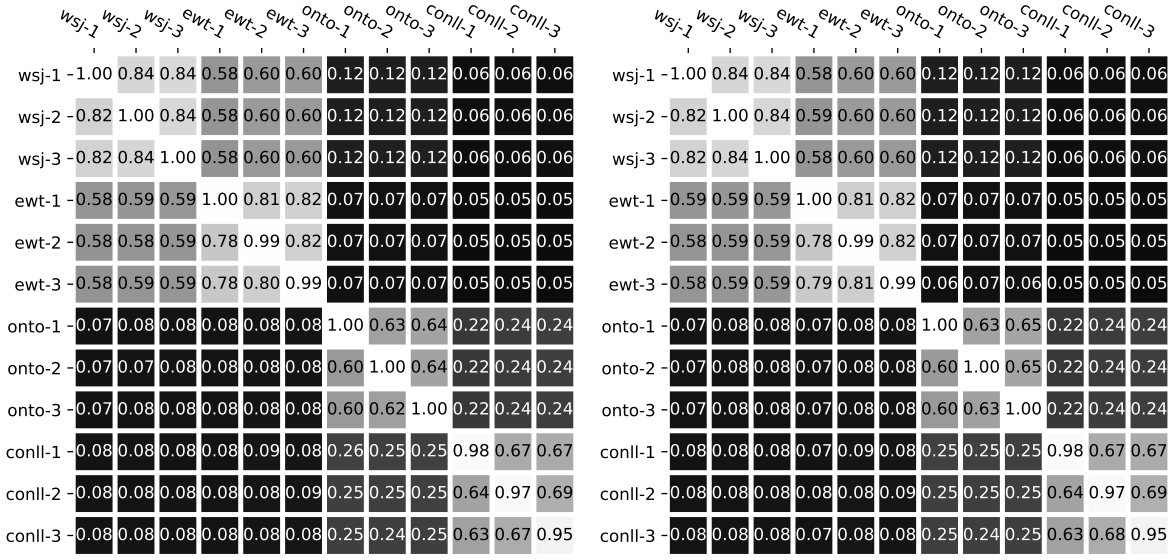
A.3 Preliminary dataset similarity evaluation results

This section includes additional heatmaps showing the datasets' self similarity for all label count combination methods tested in Section 7.1.2.



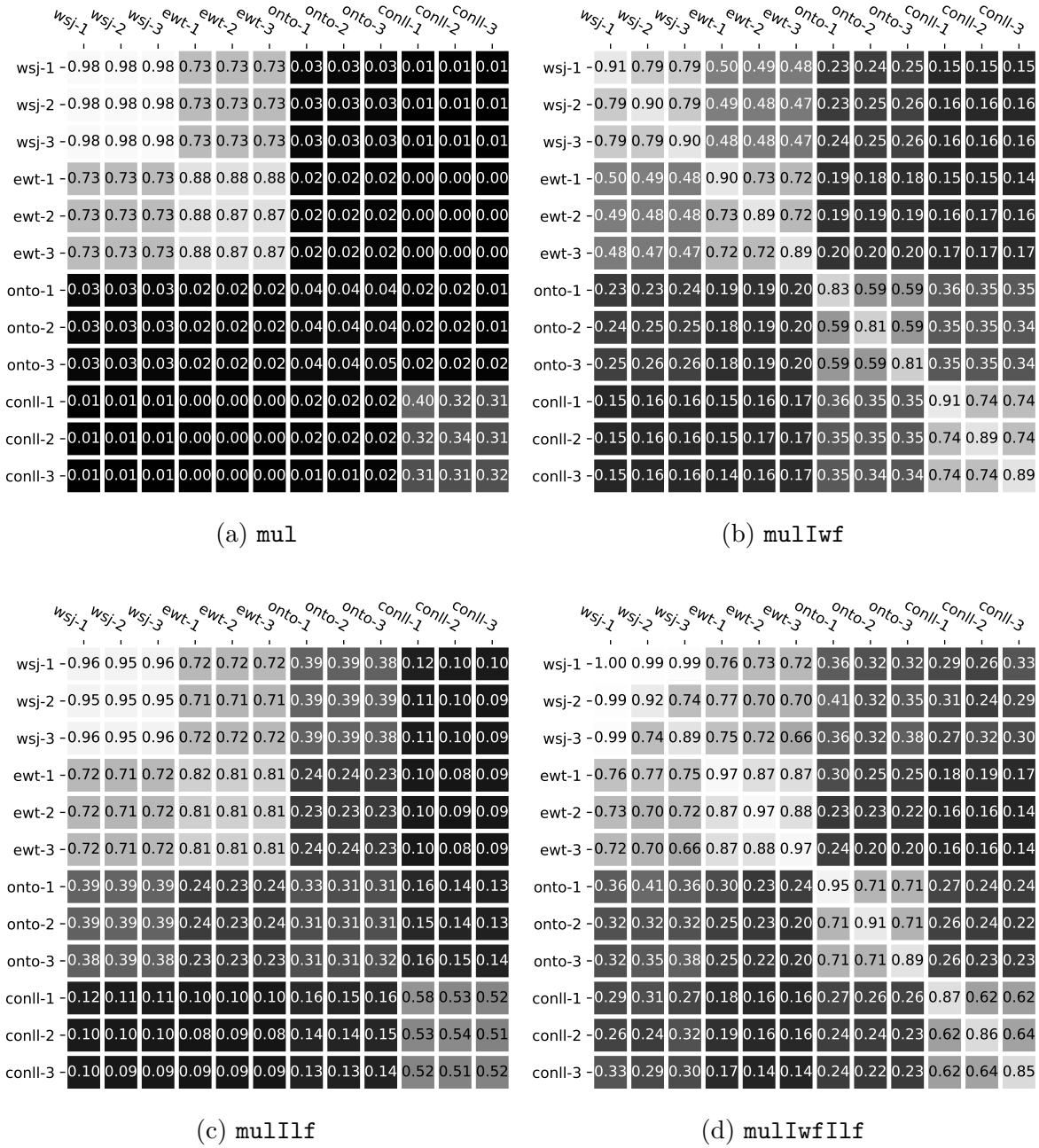
(a) Shared vocabulary (b) Shared tokens

Figure A.1: Pairwise relative vocabulary resp. token overlap



(a) `tokCtx` (b) `tokCtx` with weighting by vector similarity

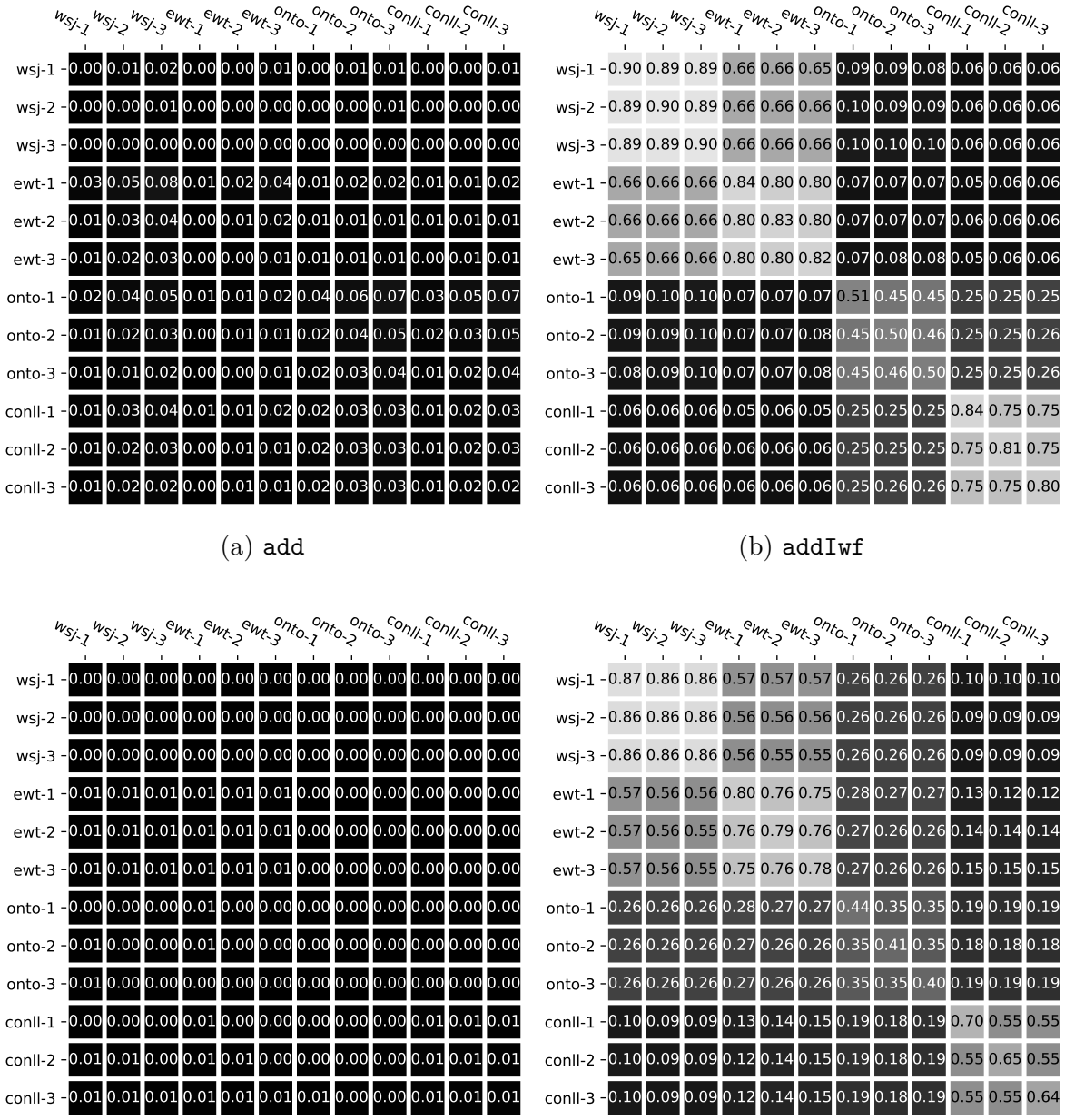
Figure A.2: Pairwise NMI_{max} scores for the token-based methods using contextual BERT embeddings



(a) `mul` (b) `mulIwf`

(c) `mulIlf` (d) `mulIwfIlf`

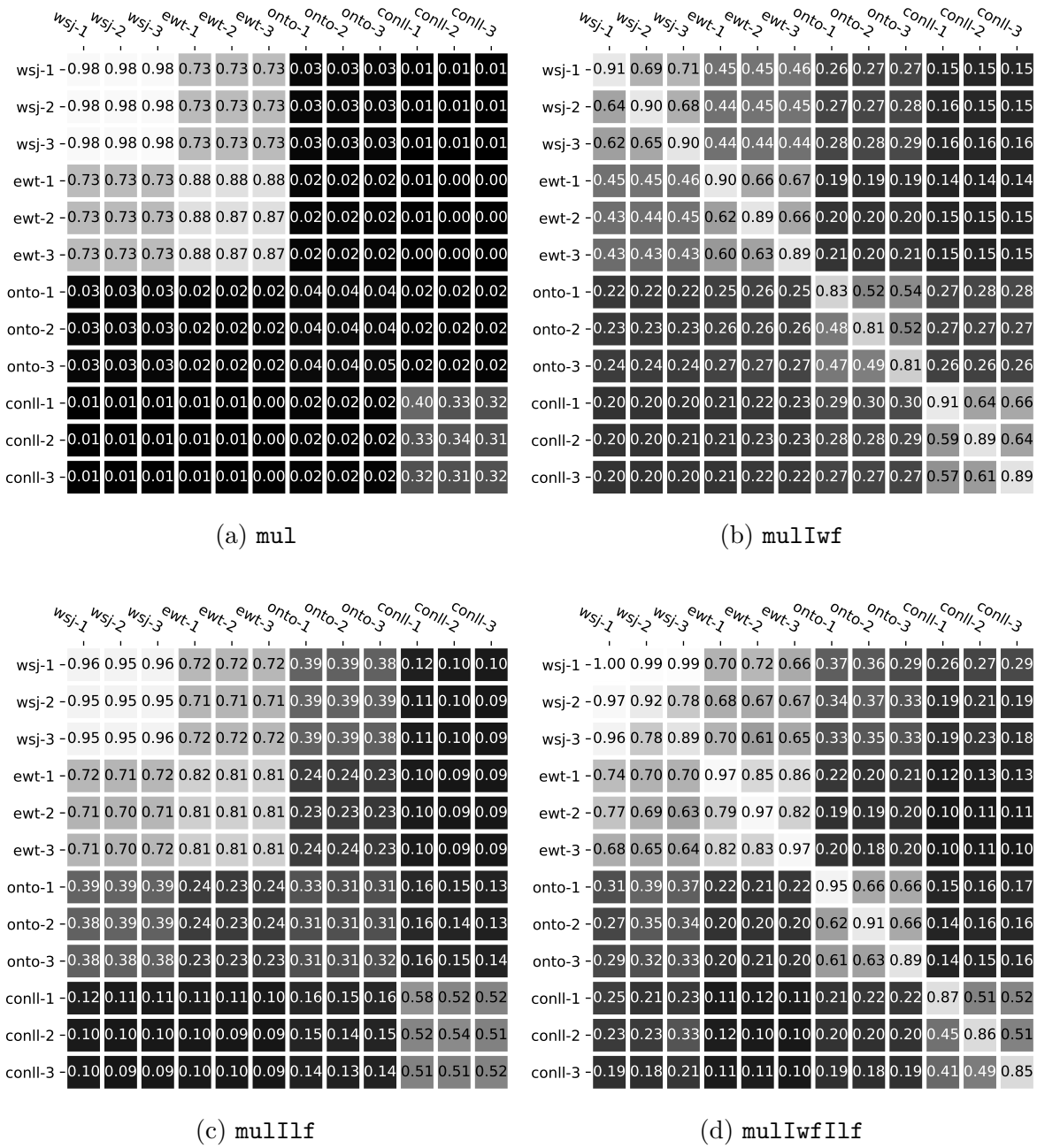
Figure A.3: Pairwise NMI_{max} scores for the multiplicative label count combination methods in the plain text overlap approach



(a) `add` (b) `addIwf`

(c) `addIlf` (d) `addIwfIlf`

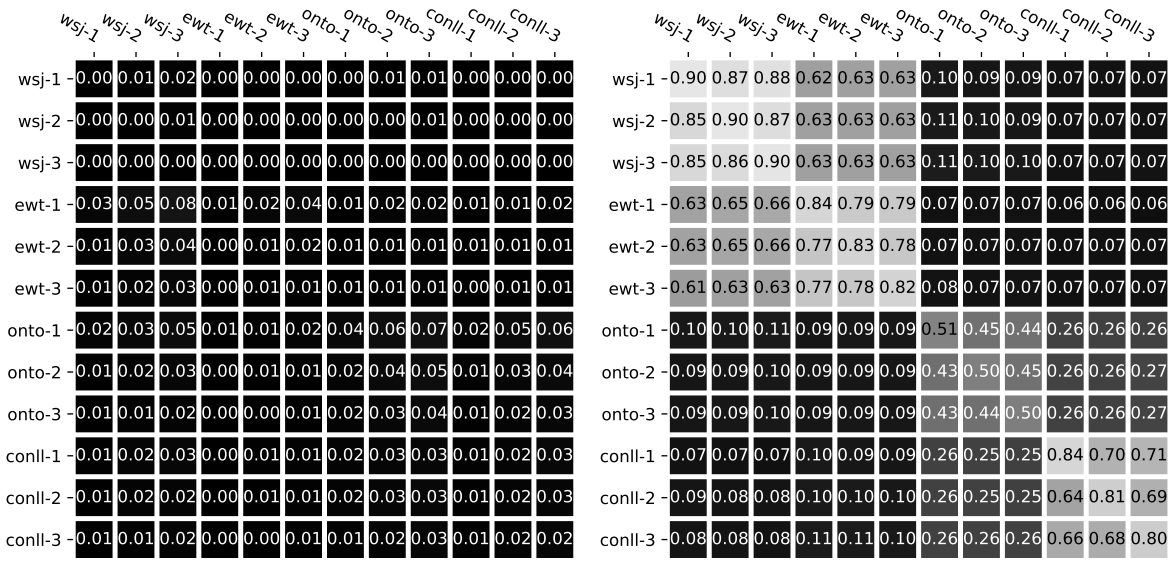
Figure A.4: Pairwise NMI_{max} scores for the additive label count combination methods in the plain text overlap approach



(a) `mul` (b) `mulIwf`

(c) `mulIlf` (d) `mulIwfIlf`

Figure A.5: Pairwise NMI_{max} scores for the multiplicative label count combination methods in the text overlap approach with fastText embeddings for non-overlapping words



(a) `add` (b) `addIwf`

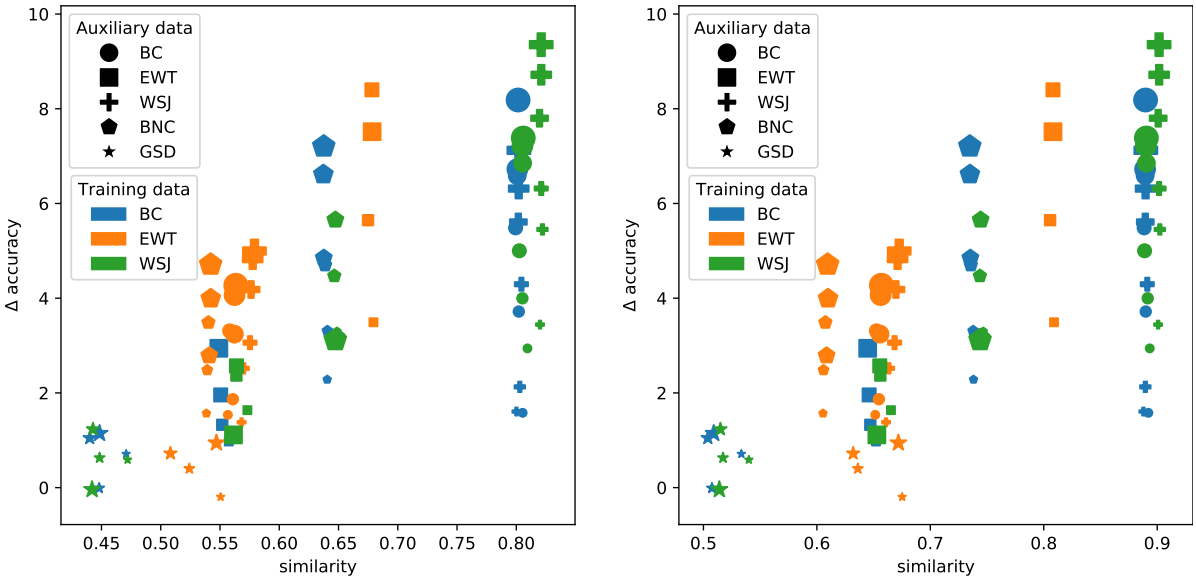
(c) `addIlf` (d) `addIwfIlf`

Figure A.6: Pairwise NMI_{max} scores for the additive label count combination methods in the text overlap approach with fastText embeddings for non-overlapping words

A.4 Experiment results POS tagging

This section contains additional scatter plots of various dataset similarity measures and the difference in accuracy of the part-of-speech tagging multi-task learning results over the single-task learning results. A description of the experiments and notation used in the figures can be found in Section 7.3.1.

Figure A.7: Scatter plots comparing the plain text overlap similarity measures with the differences in accuracy of the POS tagging multi-task learning results



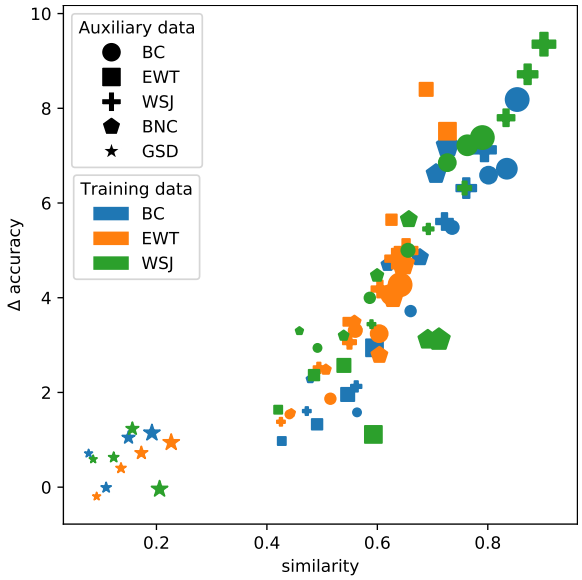
(a) plain `addIwf` NMI_{joint}

(b) plain `addIwf` NMI_{max}

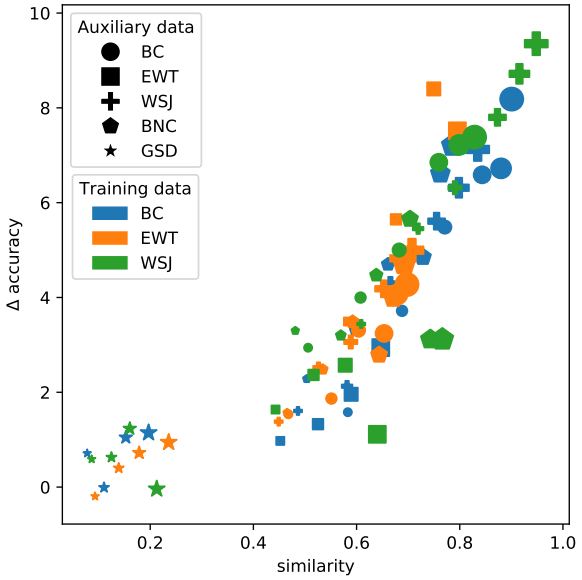
(c) plain `mulIwf` NMI_{joint}

(d) plain `mulIwf` NMI_{max}

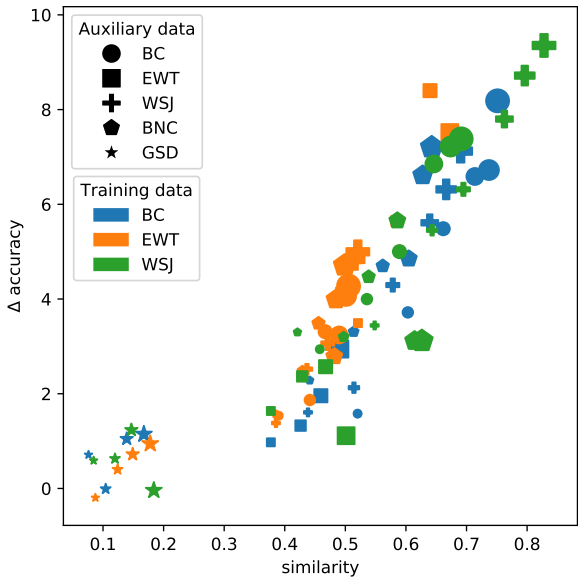
Figure A.9: Scatter plots comparing the harmonic mean of shared vocabulary (SV) and text overlap similarity measures with the differences in accuracy of the POS tagging multi-task learning results



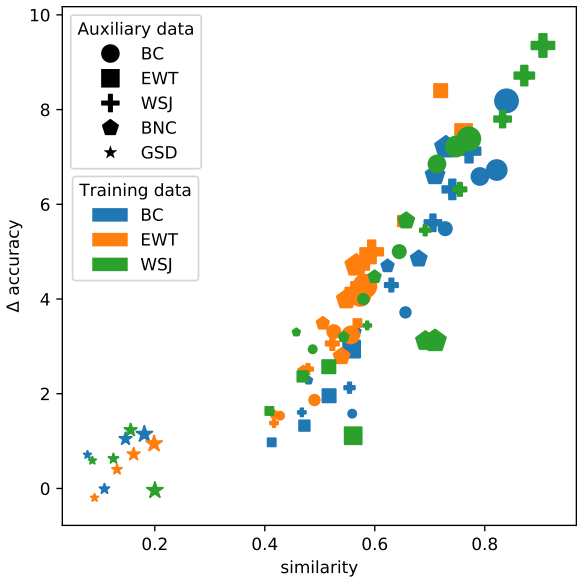
(a) SV & `addlwf` NMI_{joint}



(b) SV & `addlwf` NMI_{max}

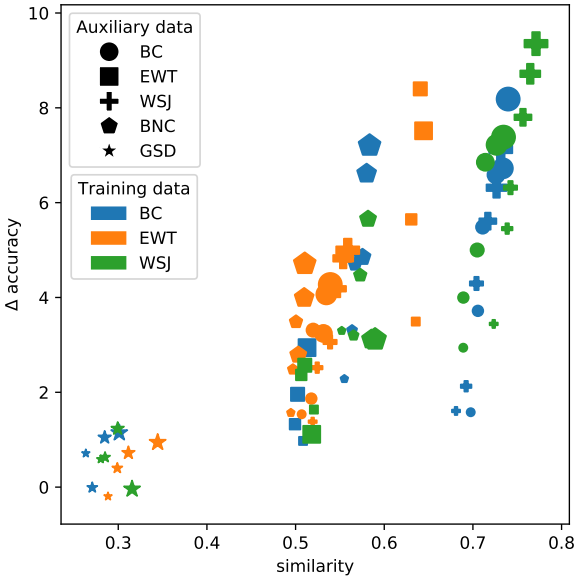


(c) SV & `mullwf` NMI_{joint}

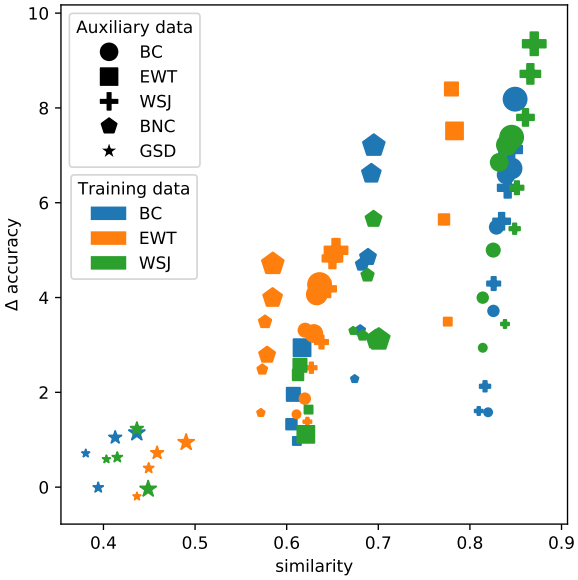


(d) SV & `mullwf` NMI_{max}

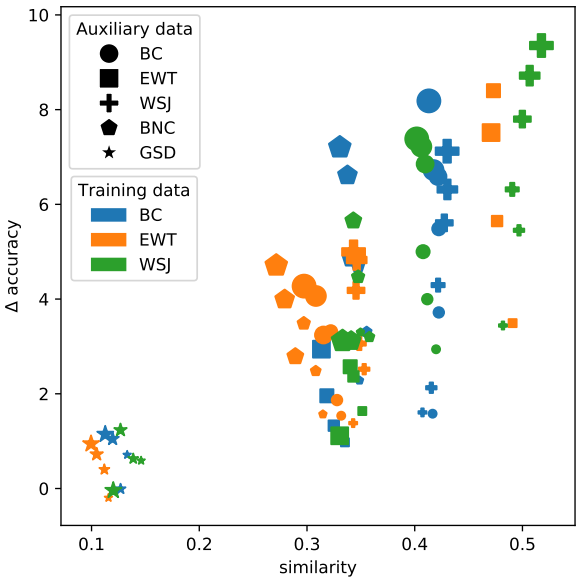
Figure A.11: Scatter plots comparing similarity measures of the text overlap approach plus embeddings with the differences in accuracy of the POS tagging multi-task learning results



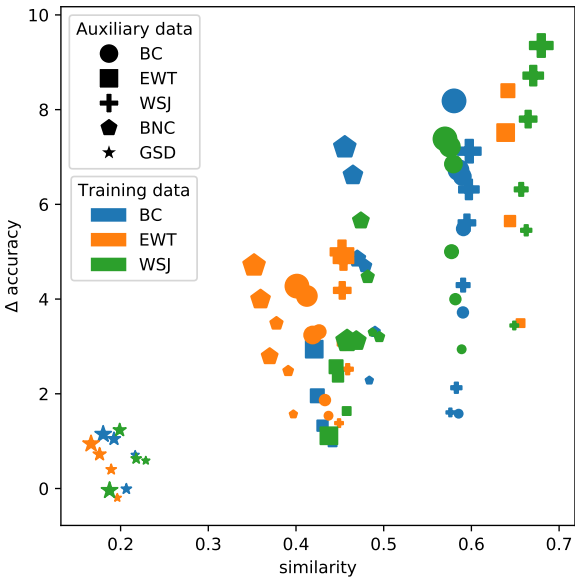
(a) mean fastText emb. `addlwf` NMI_{joint}



(b) mean fastText emb. `addlwf` NMI_{max}

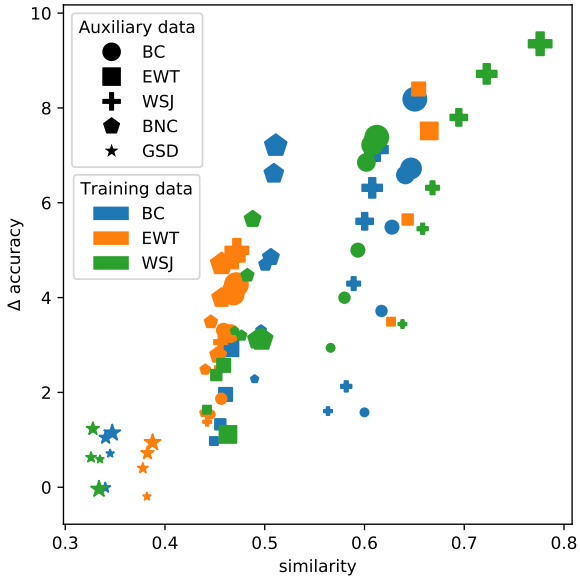


(c) mean fastText emb. `mullwf` NMI_{joint}

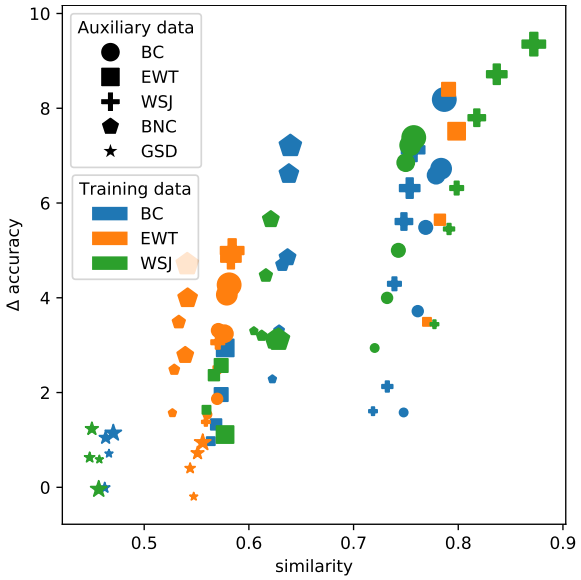


(d) mean fastText emb. `mullwf` NMI_{max}

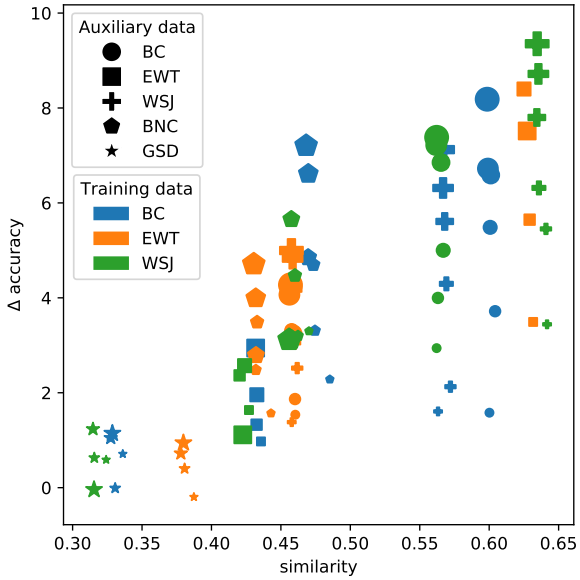
Figure A.13: Scatter plots comparing token-based similarity measures with the differences in accuracy of the POS tagging multi-task learning results



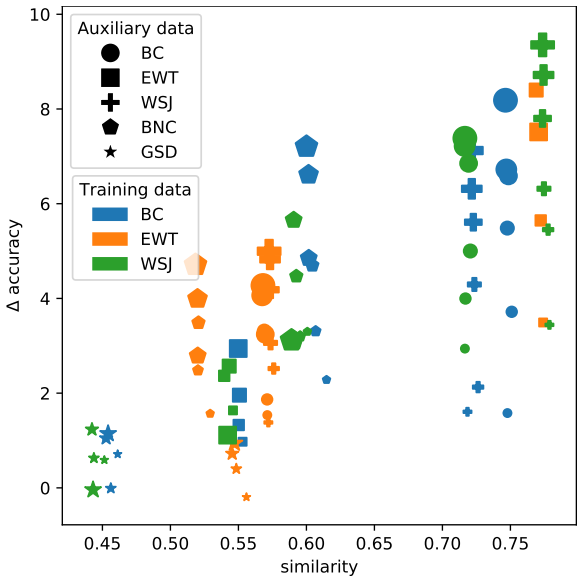
(a) mean BERT emb. `tokCtx` NMI_{joint}



(b) mean BERT emb. `tokCtx` NMI_{max}



(c) backward BERT emb. `tokCtx` NMI_{joint}



(d) backward BERT emb. `tokCtx` NMI_{max}

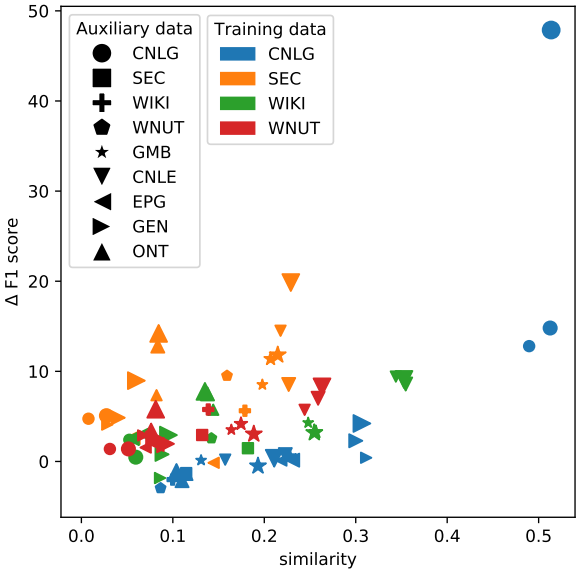
A.5 Experiment results NER

This section contains the raw F1 scores on the NER datasets using the neural network with the Softmax classifier. The corresponding experiments are described in Section 7.3.2. Further, scatter plots are contained that show additional dataset similarity measures and the difference in accuracy of the named entity recognition multi-task learning results over the single-task learning results.

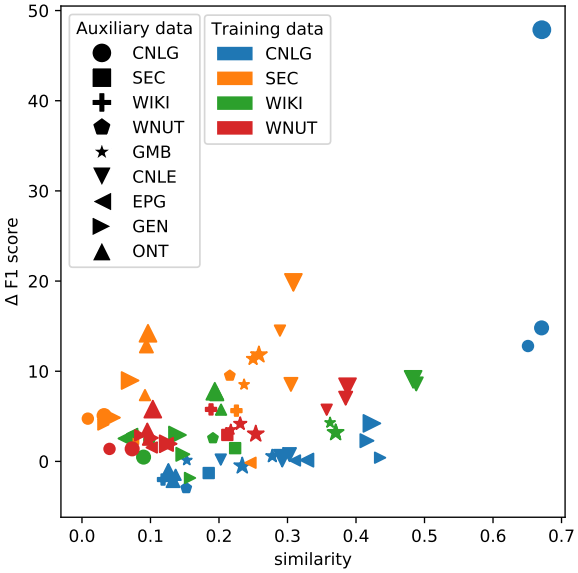
Aux. data	CNLE-50	CNLG-50	EPG-50	GEN-50	ONT-50	SEC-50	WIKI-50	WNUT-50
none	70.30 ± 0.69	41.62 ± 0.37	86.99 ± 3.70	26.97 ± 0.86	47.53 ± 1.13	43.86 ± 1.02	67.19 ± 2.62	12.67 ± 0.46
CNLE-50	83.86 ± 2.10	41.91 ± 1.50	84.40 ± 0.32	28.98 ± 3.96	47.84 ± 0.89	51.03 ± 1.85	74.66 ± 0.80	18.68 ± 1.28
CNLE-100	90.21 ± 0.43	42.32 ± 0.52	85.82 ± 1.41	30.42 ± 1.16	47.32 ± 0.65	56.97 ± 2.87	73.32 ± 1.07	18.86 ± 0.73
CNLE-250	96.77 ± 0.27	42.86 ± 0.74	85.54 ± 0.99	29.05 ± 0.97	46.82 ± 0.59	64.67 ± 3.21	72.86 ± 1.98	21.51 ± 1.96
CNLG-50	71.47 ± 1.80	53.73 ± 1.75	86.12 ± 2.08	29.19 ± 1.06	46.32 ± 2.04	47.32 ± 3.89	68.86 ± 0.40	14.06 ± 0.81
CNLG-100	71.30 ± 2.30	59.03 ± 2.51	86.85 ± 0.87	30.87 ± 2.18	45.93 ± 0.76	47.17 ± 3.05	69.97 ± 0.25	13.24 ± 1.43
EPG-50	70.47 ± 2.80	41.12 ± 1.51	93.89 ± 0.97	31.44 ± 0.57	46.79 ± 1.12	47.40 ± 4.82	71.31 ± 1.27	14.70 ± 0.65
EPG-100	72.26 ± 1.20	42.35 ± 2.11	99.14 ± 0.27	28.24 ± 1.39	45.87 ± 2.05	44.85 ± 1.80	68.81 ± 0.12	13.11 ± 1.63
GEN-50	70.49 ± 2.77	42.28 ± 0.69	86.58 ± 1.85	39.45 ± 2.50	46.88 ± 1.31	47.57 ± 0.73	69.37 ± 0.40	15.57 ± 1.03
GEN-100	71.76 ± 1.13	42.70 ± 1.60	87.77 ± 0.59	43.61 ± 1.98	46.37 ± 1.87	49.06 ± 3.95	68.65 ± 1.80	13.83 ± 1.05
GEN-250	71.85 ± 1.34	47.75 ± 0.92	86.65 ± 1.70	69.31 ± 4.17	45.74 ± 1.14	50.13 ± 4.10	69.63 ± 1.75	14.55 ± 1.03
ONT-50	73.37 ± 2.58	40.05 ± 1.24	87.01 ± 1.18	29.50 ± 0.76	51.76 ± 1.32	51.90 ± 3.94	70.24 ± 1.15	15.64 ± 1.11
ONT-100	72.91 ± 0.64	40.35 ± 1.95	84.55 ± 2.05	28.51 ± 3.50	58.11 ± 0.72	52.18 ± 2.68	72.74 ± 1.41	16.32 ± 0.72
ONT-250	75.42 ± 0.93	39.96 ± 2.31	85.72 ± 0.45	30.44 ± 2.48	63.41 ± 0.94	59.99 ± 2.58	74.86 ± 1.51	18.89 ± 0.94
SEC-50	73.40 ± 0.84	40.57 ± 2.80	86.62 ± 1.05	28.06 ± 1.25	47.24 ± 0.57	N/A	68.63 ± 2.87	14.80 ± 1.40
WIKI-50	73.07 ± 2.33	40.89 ± 1.99	87.99 ± 0.48	29.33 ± 2.41	47.30 ± 0.25	53.45 ± 1.03	N/A	18.84 ± 0.75
WNUT-50	70.28 ± 1.67	38.44 ± 0.23	83.44 ± 0.81	27.91 ± 2.19	45.13 ± 0.67	50.17 ± 4.95	69.88 ± 2.38	N/A
GMB-50	73.10 ± 1.06	41.30 ± 1.10	87.85 ± 0.79	28.50 ± 1.77	48.10 ± 0.59	53.45 ± 1.51	71.32 ± 0.51	15.62 ± 2.42
GMB-100	73.34 ± 1.28	42.26 ± 0.47	85.77 ± 0.34	28.89 ± 1.42	45.63 ± 2.38	56.38 ± 5.46	69.46 ± 2.56	16.13 ± 0.94
GMB-250	73.12 ± 0.83	41.29 ± 1.56	85.07 ± 1.76	27.86 ± 0.47	44.79 ± 0.91	56.44 ± 9.17	69.94 ± 0.81	15.85 ± 0.59

Table A.1: NER F1 score mean and standard deviation across three runs with different random seeds using the neural network model with a Softmax classifier.

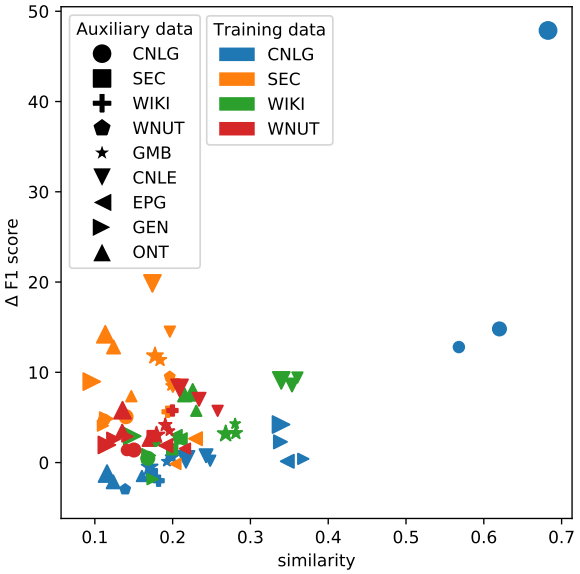
Figure A.15: Scatter plots comparing the plain text overlap similarity measures with the differences in accuracy of the NER multi-task learning results



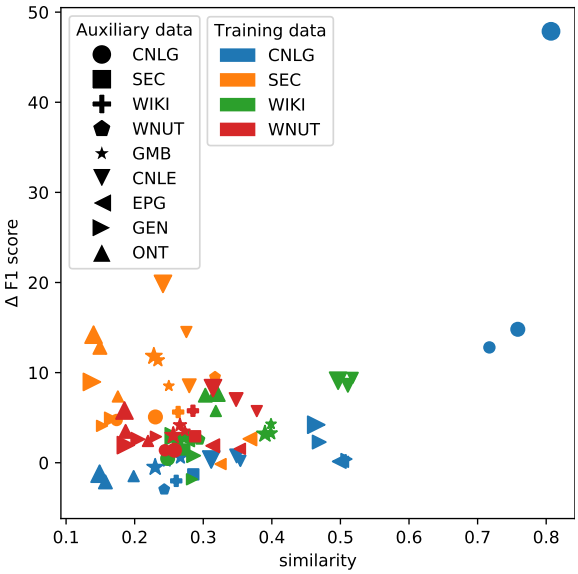
(a) plain `addIwf` NMI_{joint}



(b) plain `addIwf` NMI_{max}

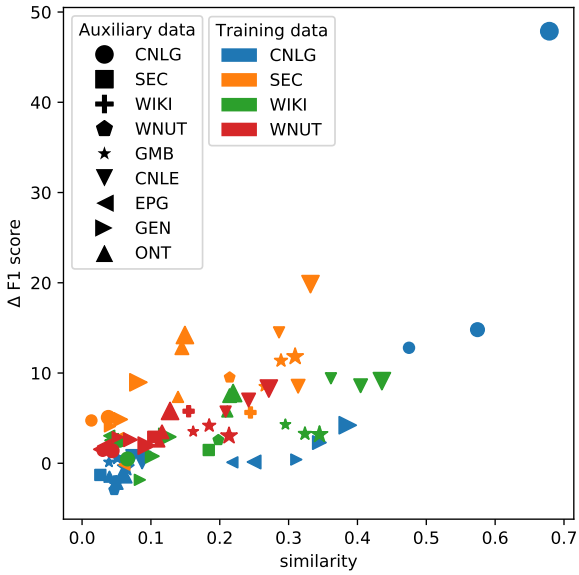


(c) plain `mulIwf` NMI_{joint}

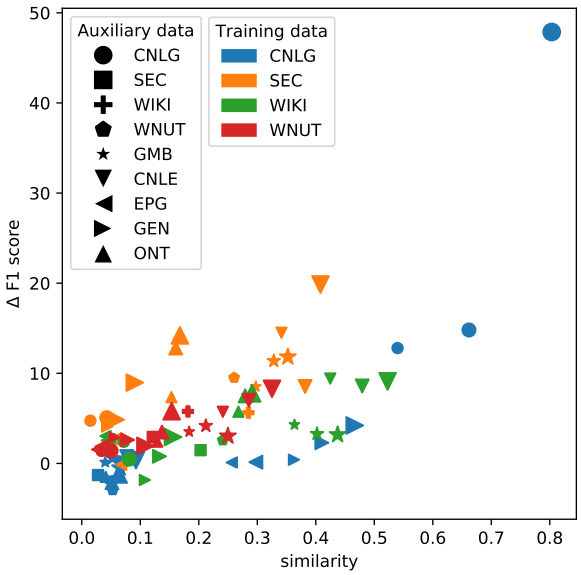


(d) plain `mulIwf` NMI_{max}

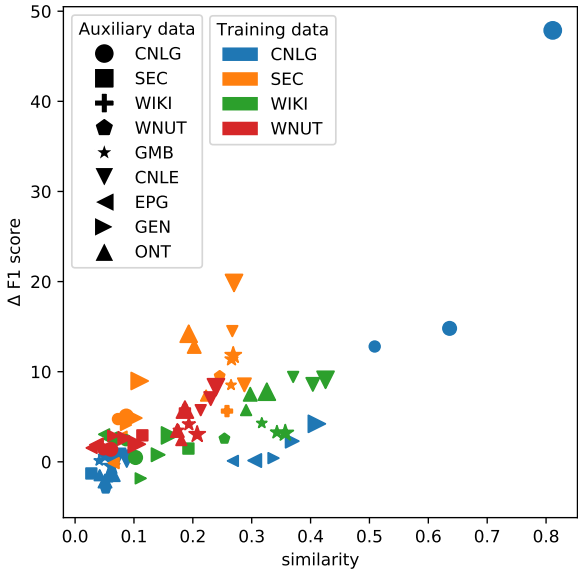
Figure A.17: Scatter plots comparing the harmonic mean of shared vocabulary (SV) and text overlap similarity measures with the differences in accuracy of the NER multi-task learning results



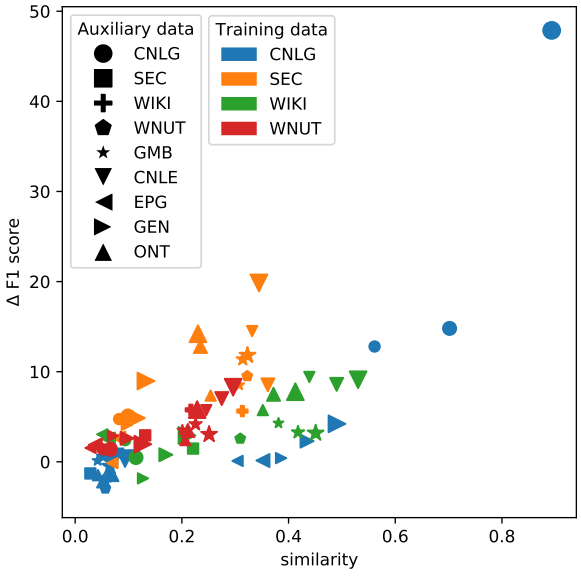
(a) SV & `addIwf` NMI_{joint}



(b) SV & `addIwf` NMI_{max}

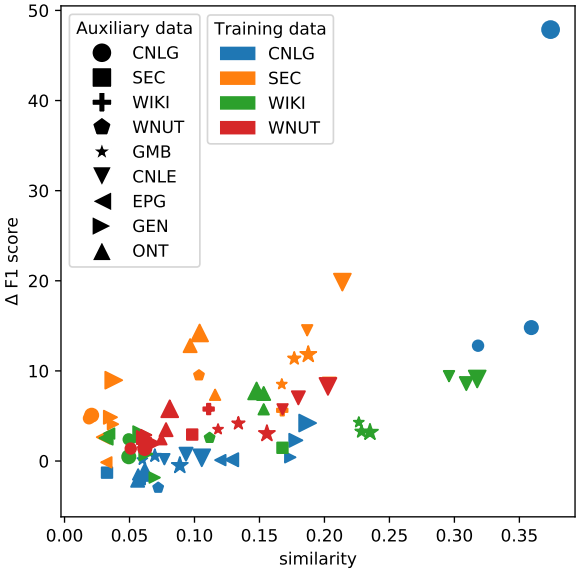


(c) SV & `mulIwf` NMI_{joint}

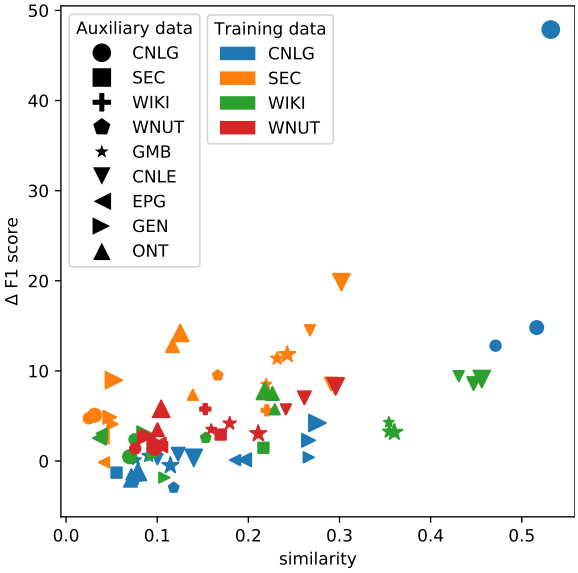


(d) SV & `mulIwf` NMI_{max}

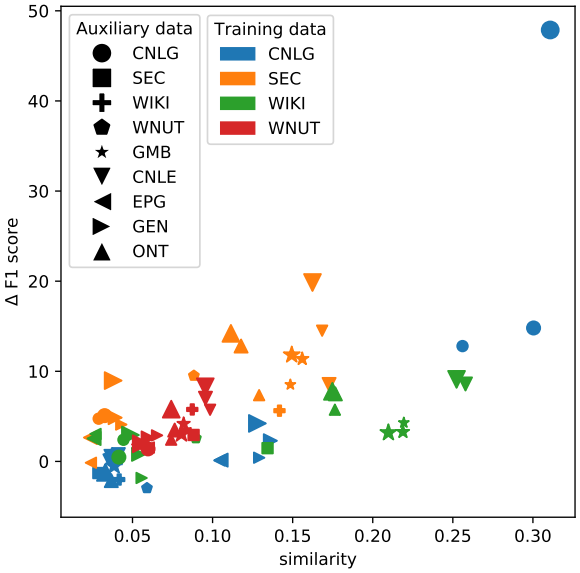
Figure A.19: Scatter plots comparing similarity measures of the text overlap approach plus embeddings with the differences in accuracy of the NER multi-task learning results



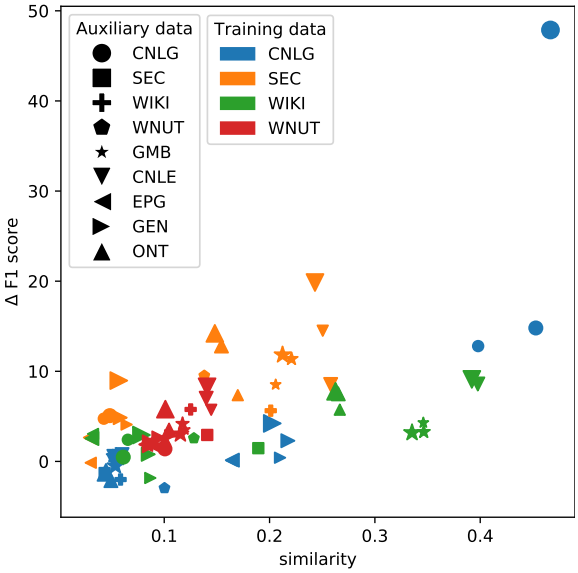
(a) mean fastText emb. `addIwf` NMI_{joint}



(b) mean fastText emb. `addIwf` NMI_{max}

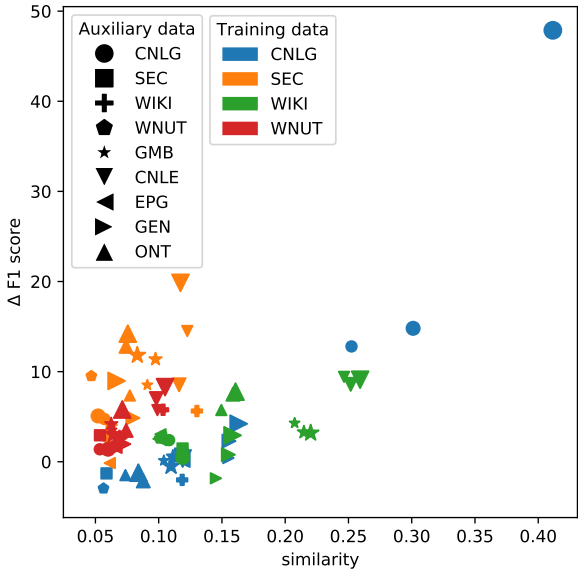


(c) mean fastText emb. `mulIwf` NMI_{joint}

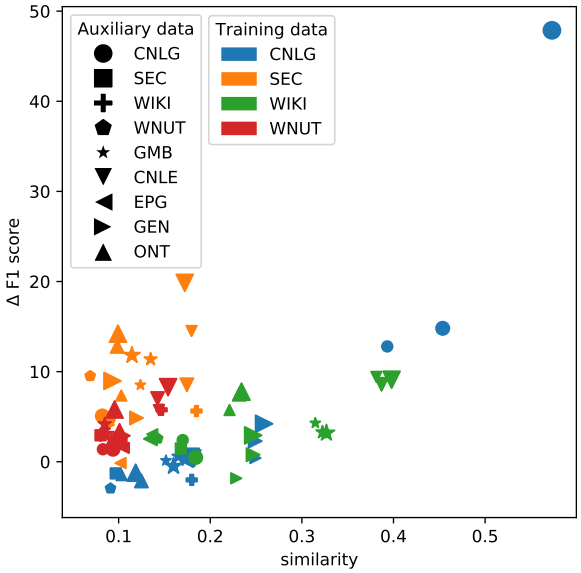


(d) mean fastText emb. `mulIwf` NMI_{max}

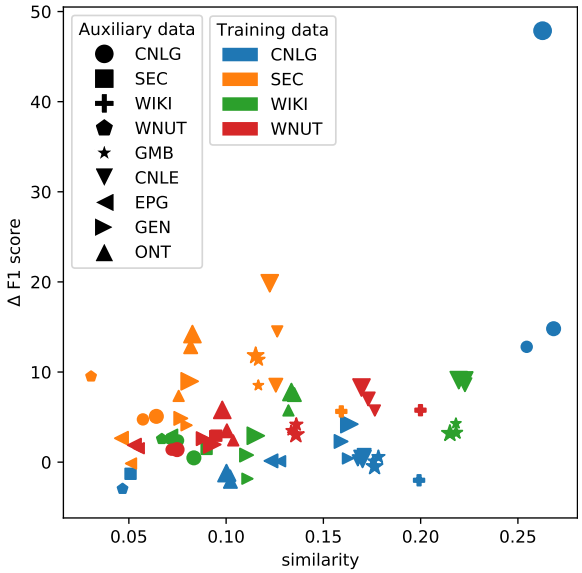
Figure A.21: Scatter plots comparing token-based similarity measures with the differences in accuracy of the NER multi-task learning results



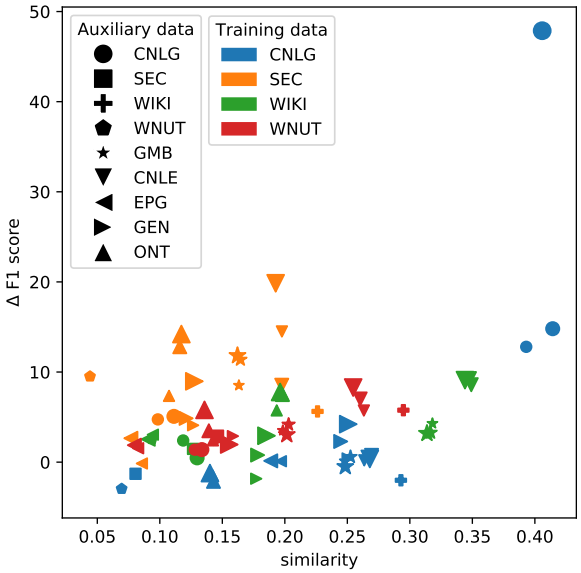
(a) mean BERT emb. `tokCtx` NMI_{joint}



(b) mean BERT emb. `tokCtx` NMI_{max}



(c) backward BERT emb. `tokCtx` NMI_{joint}



(d) backward BERT emb. `tokCtx` NMI_{max}

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum	Vorname Nachname

Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Ort, Datum	Vorname Nachname