FACULTY OF ENGINEERING AND ARCHITECTURE

Design and analysis of domain adaptation systems for joint multilingual coreference resolution

Karel D'Oosterlinck
Student number: 01609137

Supervisors: Prof. dr. ir. Chris Develder, Prof. dr. ir. Thomas Demeester
Counsellors: Semere Kiros Bitew, Ir. Johannes Deleu

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2020-2021

GHENT UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

Design and analysis of domain adaptation systems for joint multilingual coreference resolution

Karel D'Oosterlinck
Student number: 01609137

Supervisors: Prof. dr. ir. Chris Develder, Prof. dr. ir. Thomas Demeester
Counsellors: Semere Kiros Bitew, Ir. Johannes Deleu

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2020-2021

GHENT UNIVERSITY

Permission of use on loan

Karel D'Oosterlinck

May 2021

Preface

The field of Natural Language Processing has seen tremendous and continuous progress over the last years. As a computer science engineering student, witnessing these progressions from afar peaked my interest. I am grateful to be given the opportunity to further explore this interest via the pursuit of a master's Thesis.

I would like to thank Prof. dr. ir. Chris Develder and Prof. dr. ir. Thomas Demeester for this opportunity and for giving me the freedom throughout the year to explore my research interests. During our weekly meetings, I received many insightful comments, vital to the success of this work. The NLP course, hosted by both promotors, served as a sublime introduction to the field during my summer vacation. Additionally, I would like to sincerely thank both promoters for their help and support with several scholarship applications, and to give me the opportunity to further explore my interest in this field in the form of a PhD.

Subsequently, I would like to extend my gratitude towards my counsellors Semere Kirow Bitew and Johannes Deleu for guiding this research and helping shape the direction of the dissertation. Semere's assistance with setting up the software infrastructure for this research and insights into neural coreference resolution was of great value. Johannes posed many thought-provoking questions which helped me further define my core hypotheses and pushed me to gain a thorough understanding of the techniques I aimed to develop.

Finally, I would like to thank those nearest to me. This final year in my engineering education, marked by a global pandemic, was in stark contrast to the previous four years. I am tremendously fortunate to have good friends who were always one zoom-call away and who helped me find meaning and structure in this rather different year. Specifically, I want to thank Marija for her continuous support and proofreading of earlier drafts, my brothers Willem and Hendrik for their camaraderie, and my parents for providing me the luxury to obtain a higher education.

Karel D'Oosterlinck

May 2021

Design and analysis of domain adaptation systems for joint multilingual coreference resolution

Karel D'Oosterlinck

Master's dissertation submitted in order to obtain the academic degree of
MASTER OF SCIENCE IN COMPUTER SCIENCE ENGINEERING

Academic year 2020 - 2021

Supervisors: Prof. dr. ir. Chris Develder, Prof. dr. ir. Thomas Demeester
Counsellors: Semere Kiros Bitew, Ir. Johannes Deleu

Faculty of Engineering and Architecture
Ghent University

Abstract

Coreference resolution is a vital subtask for many natural language processing and understanding applications. The aim of this task is to cluster mentions in a text based on the entity to which they refer. There has been an increasing gap in performance between coreference resolution for high-resource languages (English, Spanish, Chinese, etc.) and low-resource languages (Dutch, Urdu, Swahili, etc.). This is due to most novel techniques being primarily developed for high-resource languages and low-resource languages lacking the amounts of data and engineering effort required to develop sufficiently large models, which have become the driving force behind most recent increases in performance.

In this work, we aim to mitigate this gap in performance between the English and Dutch languages for the task of coreference resolution. By fine-tuning a large pre-trained multilingual model on both English and Dutch coreference resolution at the same time, we are able to gain a 7.61% increase in performance compared to our Dutch coreference resolution baseline. Additionally, we design a domain adaptation system to help our model better generalize information between the English and Dutch languages. However, this system was not able to further increase the performance of our multilingual approach.

Keywords

coreference resolution, cross-lingual learning, multilingual learning, domain adaptation, domain adversarial neural network

Design and analysis of domain adaptation systems for joint multilingual coreference resolution

Karel D'Oosterlinck

Supervisors: Prof. dr. ir. Chris Develder, Prof. dr. ir. Thomas Demeester
Councillors: Semere Kiros Bitew, ir. Johannes Deleu

Abstract - Coreference resolution is a vital subtask for many natural language processing and understanding applications. There has been an increasing gap in performance between coreference resolution for high-resource languages (English, Spanish, Chinese, etc.) and low-resource languages (Dutch, Urdu, Swahili, etc.). This is due to most novel techniques being primarily developed for high-resource languages and low-resource languages lacking the amounts of data and engineering effort required to develop sufficiently large models, which have become the driving force behind most recent increases in performance.

In this work, we aim to mitigate this gap in performance between the English and Dutch languages for the task of coreference resolution. By fine-tuning a large pre-trained multilingual model on both English and Dutch coreference resolution at the same time, we are able to gain a 7.61% increase in performance compared to our Dutch coreference resolution baseline. Additionally, we design a domain adaptation system to help our model better generalize information between the English and Dutch languages. However, this system was not able to further increase the performance of our multilingual approach.

Keywords - coreference resolution, cross-lingual learning, multilingual learning, domain adaptation, domain adversarial neural network

Introduction

Coreference resolution is the task of clustering mentions in a text based on the entity to which they refer. While coreference resolution is not solved (yet), good understanding of this task is essential to maximally capture all information expressed in a text.

Recent coreference resolution systems are all based on end-to-end deep learning. While deep learning mitigates the need for expensive feature engineering, the resulting models are heavily dependent on the availability of large annotated datasets and pre-trained Transformer models.

These large pre-trained models and the annotated datasets needed to fine-tune them are available for high-resource languages, however this is often not the case for resource-constrained languages. This has caused a growing performance gap between these languages, causing a major part of the global population to miss out on state-of-the-art natural language processing applications.

Investing in new and improved resources for low-resource languages is a possible solution. However, this approach is exceedingly expensive and unscalable with regard to the total number of languages and dialects spoken in the world today.

A compelling case can be made for the reuse of existing data from a high-resource language to increase the performance on a low-resource language. Intuitively, both language-understanding and world-knowledge is needed to achieve high coreference resolution performance. While a document in one language and its translation in another can differ in terms of how a certain piece of information is presented, we can expect the entities represented within the document to remain similar. Based on this intuition, we argue it is possible to leverage the world-knowledge contained in documents of one language for the benefit of coreference resolution on another language, given that we can make abstraction of the difference in how the information is presented using both languages.

Our goal is to improve the state-of-the-art coreference resolution results on low-resource languages by leveraging the existing high-resource data. We propose a technique based on multilingual learning and domain adaptation to achieve this. We limit our work by only evaluate our technique on the Dutch language.

We start by building a neural end-to-end coreference resolution benchmark for the Dutch language, using a publicly available annotated Dutch dataset. We further investigate how to incorporate an additional English dataset during training. Finally, we design a domain adaptation technique, based on domain adversarial neural networks, to help our model better leverage the English dataset for Dutch coreference resolution.

Related work

Neural coreference resolution - Previously, state-of-the-art results on the task of coreference resolution were attained via rule-based systems [1]. Recently, end-to-end deep learning models have been consistently achieving better results. The first successful end-to-end coreference resolution architecture was introduced by [2]. For each span of words in a text, the architecture assigns a previous span that refers to the same entity. Span pruning is used to filter out spans that do not refer to an entity. Based on these predicted links, a final coreference clustering is extracted.

The architecture requires an encoder to transform a text into a sequence of embeddings, which are subsequently used to form span-level embeddings. Originally, [2] used a bidirectional LSTM [3] to produce contextualised word-embeddings of the input sequence. Evidently, [4] found better results when using BERT [5] to produce subword-embeddings.

Additionally, [6] introduced SpanBERT to achieve even better performance. SpanBERT is specifically pre-trained on large amounts of unsupervised data to produce high-quality span embeddings.

To evaluate the performance of our coreference resolution system, we will use the MELA F1 score [7].

Multilingual NLP - Lately, a plethora of low-resource pre-trained Transformer models have been developed [8, 9, 10]. While these models are a good first step in getting low-resource NLP up to speed, they require (i) a large corpus to pre-train and (ii) language-specific annotated datasets for fine-tuning, both of which are not always readily available.

In [11], XLM is introduced. The authors pre-train a 12-layer Transformer encoder on multiple languages at the same time and achieve state-of-the-art results on cross-lingual natural language interference and (un)supervised machine translation. Interestingly, their model achieves better perplexity on some low-resource languages compared to a Transformer directly trained on these languages. This finding indicates how a multilingual model might perform better on low-resource languages when not enough data is available to fully pre-train a language-specific model.

Building on top of XLM, XLMR [12] achieves even better results by scaling the model capacity and increasing the amount of data used during pre-training. The result is a large model, containing 270M parameters, which performs better on multiple low-resource languages compared to their best monolingual models. Additionally, the authors show that for the first time, a large multilingual model also achieves on par with monolingual models of high-resource languages.

Domain adaptation - One of the key problems in Deep Learning is the lack of generalization capability across tasks and data. The subfield of Domain Adaptation tries to partially alleviate this problem by designing techniques that are more robust to changes in the input data distribution.

Where typically domain adaptation is applied to NLP in the context of a distribution shift where source and target domain are defined over the same language [13], domain adaptation can also be framed from a multilingual point of view. In this case, the source and target domain correspond to a source and target language.

A Domain Adversarial Neural Network (DANN) [14] is one of the most popular domain adaptation models [13]. The key hypothesis is that domain adaptation can be achieved via the use of features that are both *task-discriminative* and *domain-invariant*.

Given a typical model consisting of a feature extractor and classification head, domain adaptation is achieved by adding a *domain classifier* network to the feature extractor. This domain classifier is trained to optimally discriminate between the extracted features from the source and target domain. When the resulting model is optimized for the task at hand, the feature extractor is pushed to form task-discriminative features. Additionally, the feature extracted is trained to maximize the loss produced by the domain classifier. This pushes the feature extractor to form domain-invariant features. This combination of maximizing the loss of the domain classifier with domain-invariant features while simultaneously training the domain classifier to optimally distinguish between the domains gives rise to an adversarial setting between the feature extractor and domain classifier. Much like how the adversarial setting in GANs [15] helps minimize the discrepancy between synthetic and real data, the adversarial setting introduced here minimizes the discrepancy between both domains.

Data

For English and Dutch coreference resolution, we will use the OntoNotes [16] and SoNaR [17] annotated corpora respectively. While OntoNotes contains roughly 1.6M English tokens annotated with coreference information, SoNaR contains only 1M.

One key difference between the OntoNotes and SoNaR dataset is the annotation of singletons. Singleton entities are entities which are only mentioned once during a text. Where SoNaR annotates singletons, OntoNotes does not.

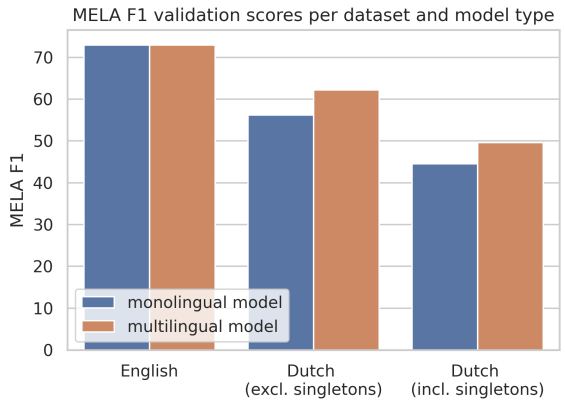


Figure 1: Validation MELA F1 score for monolingual coreference resolution per dataset and model type.

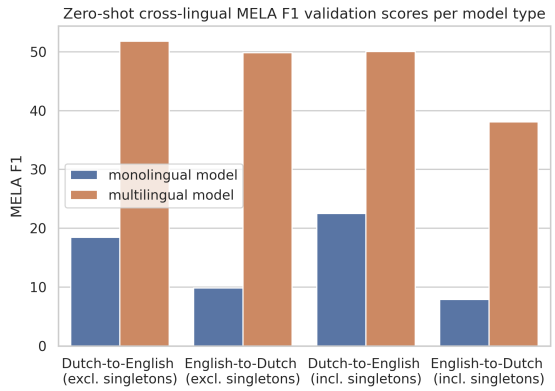


Figure 2: Validation MELA F1 score for zero-shot cross-lingual transfer per model type.

Monolingual coreference resolution

Experiments - First, we aim to construct a neural baseline for Dutch coreference resolution. The previously described coreference resolution architecture [4] serves as the basis for our system.

The original architecture uses BERT to create English subword embeddings. To adapt this architecture towards the Dutch language, we replace BERT with RobBERT [9], a Transformer which has been pretrained on Dutch data. Additionally, we experiment with using XLMR [12] to produce the contextualized embeddings.

We evaluate the monolingual coreference resolution performance of BERT, RobBERT, and XLMR by training and evaluating all three of these models on the English and Dutch data separately. Moreover, we measure the zero-shot English-to-Dutch and Dutch-to-English transfer performance to gauge the out-of-the-box cross-lingual generalization capabilities of these models.

In all experiments considered in this work, we use largely the same hyperparameters as proposed by [4]. We scaled the feed-forward neural-network size ($ffnn\_size$) down to 1000 from 3000. This slightly decreases the end performance but achieves faster training. We increased the max number of sentences the model can process per batch ($max\_training\_sentences$) from 11 to 30. This allowed the model to better fit the Dutch data, since the Dutch data features on average more sentences per document.

Results - Figure 1 shows the results of the monolingual experiments. For the Dutch language, we run the experiment twice: including and excluding the singletons. For the Dutch and English language, RobBERT and BERT are used as monolingual model respectively. XLMR is used as multilingual model.

Clearly, the Dutch performance is lagging behind. Additionally, excluding singletons from the Dutch dataset is favorable. This is unsurprising, the architecture used is not adapted towards singletons since it was developed with the English dataset in mind.

Interestingly, the multilingual model achieves on par with its monolingual counterpart on the English language but performs better on the Dutch language (+5.99%). This is in line with the claims made by [12].

Figure 2 outlines the result for the zero-shot cross-lingual experiment. Again we consider both the performance of a monolingual and a multilingual model. XLMR is used as multilingual model. As monolingual model, we consider both BERT and RobBERT in each experiment and report the best performing model.

We find that the zero-shot cross-lingual performance of the multilingual model is much stronger compared to that of the monolingual models. This result indicates that, without any additional bells and whistles, multilingual models already have a strong cross-lingual generalization capability for the task of coreference resolution.

Multilingual coreference resolution

Experiments - We now look to incorporate additional English training data in the training procedure. Because of the multilingual nature of this problem, we will solely consider XLMR in these experiments.

Three possible ways of merging the English and Dutch datasets are investigated. This results in a *full*, *document-balanced*, and *token-balanced* dataset. These respectively contain all the available Dutch and English data, an equal amount of Dutch and English batches, and an equal amount of Dutch and English tokens. While the document- and token-balanced

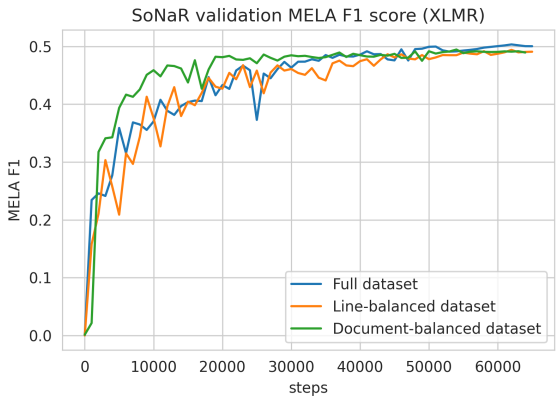


Figure 3: Validation MELA F1 score on SoNaR (incl. singletons) per dataset during multilingual training .

datasets require us to discard English data to achieve the required balance, we are interested to see how the model responds to these varying amounts of high-resource data.

We train XLMR on all three datasets proposed above. Each trained model is evaluated on both the SoNaR and OntoNotes dataset. We perform these sets of experiments twice, including and excluding singletons in the SoNaR dataset both during training and evaluation.

Results - Figure 3 shows the convergence of the validation MELA F1 score on the Dutch language per dataset during multilingual training. Regardless of the dataset used, the Dutch performance converges smoothly. This indicates that, for the amount of data we have at our disposal, the Dutch language is not *fighting* over model capacity with the English language. Therefore, we can safely consider the full dataset in subsequent experiments.

Table 1 compares the results achieved by XLMR during joint multilingual training with the best results using monolingual training. Multilingual training achieves a +1.62% increase in performance on the Dutch language compared to monolingual training.

As the zero-shot cross-lingual experiment suggested, XLMR in able to leverage the multilingual training to further increase the performance for Dutch coreference resolution. This increase is most notable when no singletons for the Dutch language are included. We hypothesise this is due to the English and Dutch tasks being more harmonized when both do not contain singletons, making it easier for XLMR to generalize between them.

Domain Adversarial Neural Networks for multilingual coreference resolution

Experiments - A domain classifier is added to the Transformer encoder in the architecture. Given a token embedding, this classifier predicts whether it originated from an English or Dutch sentence. An adversarial training objective is introduced: the domain classifier is trained to optimally classify these embeddings while the Transformer encoder is tasked with maximizing the domain classifier loss. To maximize this loss, the Transformer learns to minimize the discrepancy between both language embeddings, making them more similar. More similar embeddings will result in better cross-lingual transfer and thus better multilingual training.

The resulting objective function can be formalized as follows:

$$\hat{\theta}, \hat{\phi}, \hat{\psi} = argmin_{\phi,\psi}\ argmax_{\theta} \\ Loss_{coref}(\phi, \psi) - Loss_{domain}(\theta, \phi)$$

where θ, ϕ, and ψ represent the parameters of the domain classifier, the Transformer, and the coreference head respectively. $Loss_{coref}$ and $Loss_{domain}$ represent the losses on the coreference resolution and domain classification tasks.

Standard deep learning optimization algorithms typically do not allow to both minimize and maximize an objective with regard to different parameters. However, we can leverage the standard backpropagation algorithm to achieve the desired objective function by reversing the gradients when they are backpropagated from the domain classifier into the Transformer encoder [14]. If we now minimize the sum of both losses with regard to the three sets of parameters, we achieve the correct behaviour.

We train and evaluate our proposed domain adaptation system on the full multilingual dataset. Based on previous insights, we use XLMR as Transformer encoder and exclude singletons from the Dutch dataset.

Additionally, we focus on the problem of adversarial collapse. When the domain classifier is able to correctly discriminate between both domains, the Transformer encoder suffers from vanishing gradients [18]. In this situation, the additional of the domain classifier does not give rise to domain-invariant features.

Results - Figure 4 shows the predictions made by the domain classifier for the Dutch tokens seen during training. The model is always predicting one of both languages, indicating a state of adversarial collapse.

We found that to stop adversarial collapse, the loss produced by the domain classifier for the Dutch

Model	Trained on	Evaluated on	MELA F1
Dutch performance			
XLMR	SoNaR, train	SoNaR, dev	62.20
XLMR	OntoNotes, train + SoNaR, train	SoNaR, dev	**63.82 (+1.62)**
English performance			
XLMR	OntoNotes, train	OntoNotes, dev	72.90
XLMR	OntoNotes, train + SoNaR, train	OntoNotes, dev	**73.68 (+0.78)**

Table 1: Comparing monolingual and multilingual training. Using full dataset, excluding singletons from the Dutch dataset.

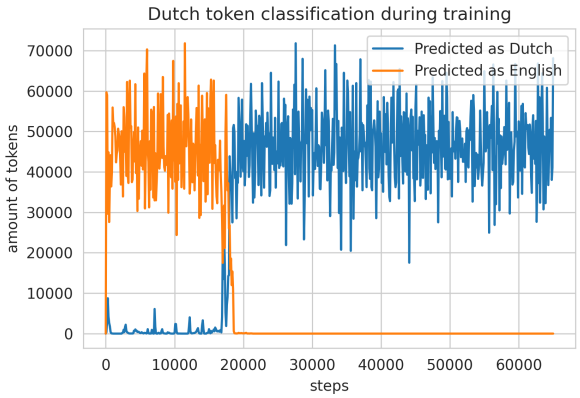


Figure 4: Dutch token classification during training when adversarial collapse occurs.

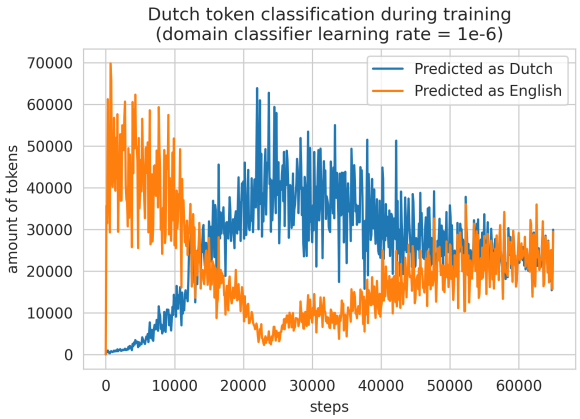


Figure 5: Dutch token classification during training, no adversarial collapse. The learning rate of the domain classifier is equal to $1\mathrm{e}{-6}$.

and English tokens needs to be equal in magnitude. This is achieved by supplying the model with an equal amount of Dutch and English batches as well as averaging the loss per batch on the amount of tokens within that batch.

Figure 5 shows the language predictions made by the domain classifier for Dutch tokens seen during training when balancing the loss. During the training steps, the model is consistently confused about the origin of the Dutch tokens, no collapse occurs.

The learning rate of the domain classifier can be used to control the balance of the adversarial regime. Figure 5 and 6 show the Dutch classification results when the learning rate is respectively lower and higher compared to that of the Transformer encoder.

As can be seen from these figures, a lower learning rate in the domain classifier leads to more wrong token classifications. We argue that a higher learning rate in the domain classifier is favorable. Indeed, we want the domain classifier to always be *one step ahead* of the Transformer encoder, such that we are sure the domain classifier has learned discriminative features before these are removed by the adversarial training objective.

While we succeed in achieving non-collapsing training and are able to further control the balance of the adversarial regime, this does not lead to gains in performance. The resulting model performance shows no significant difference compared to multilingual training without domain adaptation.

One possible explanation is that the XLMR outputs over different languages already have a small discrepancy. Indeed, XLMR is pre-trained using a Translation Language Modeling objective, specifically aimed at minimizing the discrepancy between parallel inputs. This explains the strong zero-shot cross-lingual transfer observed. The additional signal produced by the adversarial setting, aimed at further minimizing the language discrepancy, could be negligible. More research is needed to confirm this.

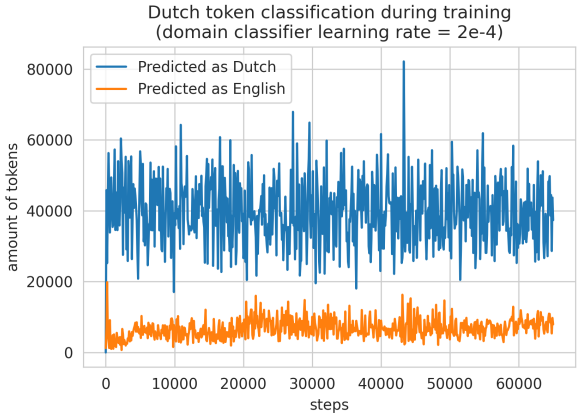


Figure 6: Dutch token classification during training, no adversarial collapse. The learning rate of the domain classifier is equal to $2\mathrm{e}{-4}$.

Conclusions

In this work we designed and analyzed a domain adaptation system for joint multilingual coreference resolution.

We built a neural baseline using a monolingual Dutch Transformer for the task of Dutch coreference resolution. We found that a multilingual Transformer trained on Dutch coreference resolution outperforms this baseline by 5.99%, indicating the strength of multilingual pre-training.

A multilingual learning approach was designed that achieves an additional 1.62% increasing in performance on Dutch coreference resolution, by incorporating English data during training. This results in a total increase of 7.61% compared to the Dutch baseline.

Finally, we studied the applicability of a domain adversarial neural network to further aid the joint multilingual training. We discovered insights into how to avoid adversarial collapse and demonstrate fine-tuned control over the adversarial training procedure.

Unfortunately, the addition of the domain adversarial neural network was unable to further increase the performance for Dutch coreference resolution.

Future work

As a first step towards more advanced multilingual coreference resolution, additional sources of data over different languages could be included. However, it is important that these datasets are similarly annotated to achieve the best possible transfer.

Parallel, the success of SpanBERT could be replicated in a multilingual setting. Where it was not cost-effective to train a SpanBERT-type model for every possible low-resource language, our findings suggest that a multilingual SpanBERT – SpanXLMR – will be able to further increase the coreference performance on many low-resource languages.

Additionally, further work is needed to evaluate our proposed techniques on languages other than English and Dutch.

References

[1] H. Lee et al. “Stanford’s multi-pass sieve coreference resolution system at the conll-2011 shared task”. 2011.

[2] K. Lee et al. “End-to-end neural coreference resolution”. 2017.

[3] M. Schuster and K. K. Paliwal. “Bidirectional recurrent neural networks”. 1997.

[4] M. Joshi et al. “BERT for coreference resolution: Baselines and analysis”. 2019.

[5] J. Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. 2018.

[6] M. Joshi et al. “Spanbert: Improving pre-training by representing and predicting spans”. 2020.

[7] P. Denis and J. Baldridge. “Global joint models for coreference resolution and named entity classification”. 2009.

[8] W. de Vries et al. “Bertje: A dutch bert model”. 2019.

[9] P. Delobelle et al. “RobBERT: a dutch RoBERTa-based language model”. 2020.

[10] D. Nozza et al. “What the [mask]? making sense of language-specific BERT models”. 2020.

[11] G. Lample and A. Conneau. “Cross-lingual language model pretraining”. 2019.

[12] A. Conneau et al. “Unsupervised cross-lingual representation learning at scale”. 2019.

[13] A. Ramponi and B. Plank. “Neural Unsupervised Domain Adaptation in NLP—A Survey”. 2020.

[14] Y. Ganin and V. Lempitsky. “Unsupervised domain adaptation by backpropagation”. 2015.

[15] I. J. Goodfellow et al. “Generative adversarial networks”. 2014.

[16] R. Weischedel et al. “Ontonotes release 4.0”. 2011.

[17] I. Schuurman et al. “Interacting semantic layers of annotation in SoNaR, a reference corpus of contemporary written Dutch”. 2010.

[18] J. Shen et al. “Wasserstein distance guided representation learning for domain adaptation”. 2018.

Table of Contents

List of Figures

List of Tables

1 Introduction

The aim of this Master's Thesis is to design and analyse domain adaptation techniques applicable to multilingual transformer models, in order to build better natural language processing systems for low-resource languages. We evaluate our technique on the difficult problem of coreference resolution, which aims to cluster mentions in a text based on the entity to which they refer.

We limit the scope of this work by only considering coreference resolution for the English and Dutch languages.

1.1 Recent trends in the field of NLP

The field of Natural Language Processing (NLP) has seen tremendous and continuous progress over the last years, most recently fueled by a new kind of neural model called the Transformer [1]. Transformer models are highly parallelizable, and are easy to reuse as pre-trained models for the benefit of many different NLP tasks. This is truly a "game changer": in terms of significance, the recent improvements in NLP have been compared to the breakthroughs in computer vision during 2012-2013.

Access to state-of-the-art NLP models has been more democratized than ever, primarily caused by the growth of pre-trained model-sharing platforms [2, 3, 4]. These platforms have made it possible for researchers to easily publish, download, and experiment with novel state-of-the-art models. Additionally, a strong community has formed around these model-sharing platforms, further developing tools that improve the quality of life of NLP researchers and developers, which also lowers the barrier to entry for new researchers and developers.

Because Transformers are highly parallelizable, scale well to enormous amounts of data, and are easy to share, researchers have been continuously pushing the size of these models – both in terms of memory, compute, and data.

1.2 Problem Statement

These ever-growing models have been requiring ever larger datasets and resources to pre-train and fine-tune. Unfortunately, the existing resources have in the past been primarily focusing on the English language. For many of the other languages in the world, either not enough data exists, or not enough resources to perform research are allocated to these problems, or both. This has caused a **growing performance gap** between NLP for English (and other high-resource languages like Spanish and Chinese) and NLP for languages with less data available (further referred to low-resource languages; e.g., Dutch, Swahili, and Urdu). This gap causes a major part of the global population to miss out on state-of-the-art NLP applications, such as: text summarization, question answering, and smart digital assistants.

In light of the – still growing – resources for high-resource NLP, a solution needs to be found to ensure low-resource languages can stay up to speed. Investing in new and better resources for less common language is a possibility, however it is exceedingly expensive and unscalable with regard to the total number of languages and dialects spoken in the world today. Thus, there is a need for novel techniques that can **efficiently leverage the existing resources to provide state-of-the-art results for a wide variety of languages**, without requiring much additional work per low-resource language.

One promising research direction investigates the use of **multilingual models** to better generalize NLP tasks across the language barrier. These multilingual models have been pre-trained using a mixture of several different input languages – sometimes up to one hundred, including both high-resource and low-resource languages. Initial work has shown that these multilingual models are able to compete with the best monolingual models for high-resource languages, while possible surpassing the best monolingual performance on low-resource languages [5]. However, it remains unclear for which downstream tasks and which low-resource languages the application of multilingual models is favorable.

A different research track explores the application of **domain adaptation** to the field of NLP. Domain adaptation in general aims to develop techniques which allow deep learning algorithms trained on a source domain

to better generalize towards a given target domain. In the context of NLP, most domain adaptation work is typically focused towards fine-tuning a model to a different textual genre [6], within the same language (e.g., adapting a pre-trained Transformer trained on a mixture of textual genres to medical texts).

To the best of our knowledge, work exploring the application of domain adaptation techniques to multilingual NLP – in order to help a model better generalize across the language barrier – is scarce.

However, a compelling case can be made for the reuse of existing data from a high-resource language to increase the performance on a low-resource language. Intuitively, both language-understanding and world-knowledge is needed to achieve high coreference resolution performance. While a document in one language and its translation in another can differ in terms of how a certain piece of information is presented, we can expect the entities represented within the document to remain similar. Based on this intuition, we argue it is possible to leverage the world-knowledge contained in documents of one language for the benefit of coreference resolution on another language, given that we can make abstraction of the difference in how the information is presented using both languages.

1.3 Main Contribution

In this work, we aim to **design and analyse** models for the task of **coreference resolution** on a **low-resource language**, using **multilingual Transformers** and **domain adaptation systems**. We evaluate our models using Dutch as a low-resource language and English as a high-resource language.

The main contributions of this work are the following:

- we set a baseline for neural Dutch coreference resolution,
- we show the applicability of multilingual models and multilingual training to Dutch coreference resolution – thereby surpassing our previous baseline – and,
- we propose and analyze a domain adaptation system for multilingual coreference resolution.

1.4 Outline

This work is outlined as follows. Section 2 describes the task of coreference resolution and its importance with regard to the overall field of NLP. The available datasets are discussed as well as the evaluation metrics used to benchmark coreference performance. Section 3 gives an overview of the Transformer model, which has recently become the dominant architecture in the field of NLP. A high-level explanation of the architecture is given. Furthermore, the application of the Transformer model to the problem of coreference resolution is discussed. Finally, efforts using Transformers to tackle the low-resource language problem, such as multilingual transformers, are explored. In section 4, we give a brief introduction to the problem of domain adaptation and outline the domain adaptation technique we will be using in this work.

The first experiment, discussed in section 5, serves as a baseline for the rest of our work: we validate existing state-of-the-art results on English coreference resolution and set a Dutch coreference resolution baseline. Additionally, we investigate the cross-lingual transfer between performance on English and Dutch coreference resolution. In the second experiment, outlined in section 6, we show that multilingual models can effectively perform coreference resolution on multiple languages at the same time and can greatly improve the cross-lingual transfer between English and Dutch coreference resolution. We show that our multilingual model increases performance for Dutch coreference resolution. Section 7 discusses our final experiment, where we study the use of domain adaptation to further increase the effectiveness of multilingual coreference resolution.

Finally, section 8 gives an overview of our main takeaways and we provide an outline for future work in section 9.

2 Coreference Resolution

Coreference resolution is the task of clustering mentions in a document based on the entity to which they refer. For example, the sentence:

[[My]$_1$ laptop]$_2$ does not fit in [[my]$_1$ bag]$_3$ because [it]$_3$ is too small.

contains the following entities: the person saying the sentence, that person's laptop, and that person's bag. The sentence contains several mentions – indicated by the brackets – that refer to these entities – indicated by the subscripts. As we can see, the possibility exists for these mentions to be nested.

Although coreference resolution is not solved (yet), good understanding of the problem is required for many important downstream NLP applications. One very important area is that of information extraction [7], which aims to convert unstructured documents to structured data, in simplified layman's terms: transform text into database entries. Here, coreference resolution is essential to maximally capture all information expressed in the text.

To indicate how difficult this problem is, consider the slightly adapted sentence

[[My]$_1$ laptop]$_2$ does not fit in [[my]$_1$ bag]$_3$ because [it]$_2$ is too big.

While the mentions and entities of the sentence remain unchanged, the clustering of the mentions is now different. Clearly, a good understanding of the context and semantics of the text is needed for proper coreference resolution. Sometimes, very large sections of text need to be considered to properly understand the context of a sentence.

Additionally, the world knowledge needed to perform coreference resolution should remain up to date. For example, *the prime minister of Canada* can refer to several entities, based on the time at which the sentence was uttered.

The situations described above are often trivial for humans, however they are notoriously difficult for machines.

In what follows, we give a slightly more in-depth explanation into the possible types of coreference resolution in subsection 2.1. The available datatasets we will use throughout this work are highlighted in subsection 2.2. Finally, the standard evaluation metrics for coreference resolution are explained in subsection 2.3.

2.1 Types of coreference resolution

Having described the goal of coreference resolution and several difficulties, we now give an overview into the different types of coreference resolution. For this overview, we will not go in-depth, since a detailed understanding of coreference resolution is not required to understand this work. The aim of this section is to gather enough knowledge about coreference resolution to compare the different datasets in subsection 2.2. The reader who is solely interested in the deep learning aspect of this work can safely skip this subsection.

The provided overview is based on that of [8].

2.1.1 Identity coreference

This is one of the most frequent forms of coreference. In identity coreference, two mentions refer to the exact same object.

> In college, [Bert]$_1$ is top of his class. [The ambitious student]$_1$ has a bright future.
>
> [Emma]$_2$ is currently in high-school. [She]$_2$ still has a long way to go.

2.1.2 Part/whole coreference

A partial coreference relationship is established when a mention refers to a part of a previously mentioned entity. This is typically a component of a previously mentioned entity or a member in a set.

> [The airplane]$_1$ is ready for take-off, [the engines]$_2$ have been checked.

Although a partial coreference relationship occurs, the mentions strictly do not refer to the same entity.

2.1.3 Type/token coreference

A relationship is established between two mentions that refer to a similar type of entity, even though the entities are not exactly identical.

> I prefer to live in [the small house]$_1$ while my partner prefers [the big house]$_2$.

> I can't decide between [the red iPhone]$_3$ or [the purple iPhone]$_4$.

2.1.4 Time-indexed coreference

Time-indexed coreference links refer to the same entity only at a certain point in time. Consider the following situation.

> Up until his death, [prince Philip]$_1$ was [the husband of Queen Elizabeth II]$_2$.

The link between *prince Philip* and *the husband of Queen Elizabeth II* is time-dependent, as is made clear by the time indication *up until his death*.

2.1.5 Metonym coreference

A figure of speech is used to refer to a previously mentioned entity. The link is established because the figure of speech closely relates to the mentioned entity.

> [Queen Elizabeth II]$_1$ issued a statement concerning the departure of Harry and Meghan. [The crown]$_1$ ensured that Harry and Meghan remain beloved members of the royal family.

2.1.6 Posessive coreference

A mention is used to describe a posessive relationship.

> [I]$_1$ don't like [my]$_1$ bag, it is too big.

2.1.7 Bound anaphora coreference

Bound anaphora are used to link properties of general categories, instead of individual entities.

> [All dogs]$_1$ like playing with [their]$_1$ toys.

2.1.8 Appositions

During an apposition, two elements are placed side-by-side and one identifies the other and gives extra information.

> [Mr. Smith]$_1$, [the English teacher]$_1$, was known to give very difficult exams.

2.2 Available data

English data: The MUC[1] and ACE[2] corpora were for a long time the standard corpora for English supervised coreference resolution learning. However, these corpora suffered either from having a limited size or not considering all possible entities for annotation [9]. Additionally, these corpora showed low Inter-Annotator Agreement [10]. As a partial aid to these problems, the OntoNotes corpus was introduced [11, 12]. The OntoNotes corpus consists roughly of 1.6M English words, 950K Chinese words, and 300K Arabic words. The entire corpus has been annotated with a layer of coreference resolution information. Additionally, a large part of the corpus has been annotated with additional layers of syntax, propositions, word sense, and named entities information. Indicating the quality of this corpus, these annotations achieve a 90% inter-annotator agreement [11]. Because of its size, high-quality annotations and additional layers of information, the OntoNotes corpus has become the de facto corpus for English coreference resolution. For the rest of this work, we will only consider the English part of this corpus.

The use of the OntoNotes corpus was further popularized by the CoNLL-2011 and CoNLL-2012 shared task [13, 9]. These tasks also paved the way for a more standardized evaluation scenario for coreference resolution systems [14] (which we will describe in the next subsection).

Dutch data: The SoNaR-1 corpus [15], consisting of 1M Dutch words, was created in light of the STEVIN-program between the Dutch and Flemish government to promote the creating of a large written corpus spanning several variants of the Dutch language. The corpus is annotated with layers of coreference information as well as part-of-speech tagging, lemmatization information, and named entity recognition information [15]. The corpus was annotated using already existing guidelines for the annotation of Dutch coreference [16, 17].

Comparison: Since our aim is to build a multilingual coreference resolution system for Dutch and English, it is crucial to understand the differences between the English (OntoNotes) and Dutch (SoNaR-1) data we will be using. In what follows, we describe the different statistics characterizing these datasets. Additionally, we also provide a brief comparison with regard to the exact types of coreference resolution annotated in both datasets.

For the SoNaR dataset, all the types of coreference as described in subsection 2.1 are annotated. The OntoNotes corpus is only annotated with identity coreference links or appositions.

The OntoNotes dataset contains roughly 1.6M English tokens over 3.493 different documents. SoNaR consists of 1M Dutch tokens over 615 documents. The average document in the OntoNotes dataset is 458 tokens long while the average SoNaR document contains 1.626 tokens.

One key difference between the OntoNotes and SoNaR dataset is the annotation of singletons. Singleton entities are entities which are only mentioned once during a text. Where SoNaR annotates singletons, OntoNotes does not.

For a more detailed overview of both corpora, we refer to the official OntoNotes and SoNaR annotation guides: [9] and [17] respectively.

train/dev/test split: For the OntoNotes dataset, we use the standard train/dev/test split proposed by [9]. For SoNaR, the train/dev/test split from [18] is taken.

2.3 Evaluation metrics

Earlier works in coreference resolution suffer from a discrepancy between the evaluation metrics used and evaluation scenarios considered [19]. Designing a satisfying coreference resolution metric is a challenging task. One of the first problems that arises is how to deal with spans of mentions. Indeed, a coreference system might correctly link a span of mentions to an entity, but the span might be slightly too large or slightly too small. To what extent is this link correct? A second problem considers how to define accuracy and recall on the predicted coreference links. Many different metrics exist [20, 21, 22, 23, 24] and researchers

[1] https://catalog.ldc.upenn.edu/LDC2003T13
[2] https://catalog.ldc.upenn.edu/LDC2006T06

have agreed that each metric has its pros and cons, depending on the considered scenario and downstream application [9].

While introducing the CoNLL-2012 Shared Task, Pradhan et al. [9] proposed to standardize the metrics and scenarios used to evaluate coreference resolution systems. Only exact spans of mentions are considered correct, too large or too small spans are completely false. The authors propose to use the MUC [20], B-CUBED [21], and CAEF_e [22] metrics. To be able to unambiguously rank coreference resolution systems, the authors propose to use the average of the previous three mentions, called the MELA metric [24]. This standardization effort was well received. In a follow-up work [14], the authors tackle some remaining ambiguity in the implementation of the standardized metrics and release a well-tested open-source reference implementation for these metrics. We will use this open-source implementation to validate our models throughout this work.

In what follows we give a high-level overview into how these metrics are calculated. For an in-depth explanation tackling the edge-cases, we refer to Pradhan et al. [14]. The coreference links in a document can be viewed as a set of sets: a document contains a set of unique entities mentioned in that document, and each entity is a set of mentions that all referring to that entity. Even though a coreference resolution system needs to only predict $x - 1$ coreferent links to add x mentions to one entity set, this clustering eventually produces $\frac{(x-1)^2}{2}$ coreferent links belonging to that entity (one for every pair of mentions of the entity).

Consider $\mathcal{G}$, the set of sets representing the ground-truth coreference links contained within the document and $\mathcal{P}$, the set of sets representing the predicted coreference links.

The MUC metric is a link-based metric, the precision is calculated by dividing the total amount of common links in $\mathcal{G}$ and $\mathcal{P}$ with the amount of links in $\mathcal{P}$ while the recall is defined by dividing the total amount of common links with the amount of links in $\mathcal{G}$. Because the MUC metric is link-based, it has some drawbacks: entities consisting only of one mention aren't accounted for (since they produce no links) and a prediction grouping all mentions under a single entity achieves 100% recall (since all possible links are present).

The B-CUBED metric calculates a precision and recall score per mention and averages those scores to achieve a document-level metric. For each mention m, the precision is calculated by dividing the size of the intersection of the set containing m in $\mathcal{G}$ and $\mathcal{P}$ with the size of the set containing m in $\mathcal{P}$. The recall for mention m is defined by dividing the size of the intersection with the size of the set containing m in $\mathcal{G}$.

The CAEF_e metric calculates precision and recall at the entity level. An entity-similarity metric $ES(a, b)$ is defined between two entities a and b. A mapping

$$m\colon \mathcal{G} \mapsto \mathcal{P}$$

is defined between elements of $\mathcal{G}$ and $\mathcal{P}$. The mapping is chosen to maximize the resulting summed entity-similarity over the linked entities:

$$m = argmax_m \sum_{g \in \mathcal{G}} EM(g, m(g))$$

The precision p and recall r are defined as:

$$p = \frac{\sum_{g \in \mathcal{G}} EM(g, m(g))}{\sum_{p \in \mathcal{P}} EM(p, p)}$$

$$r = \frac{\sum_{g \in \mathcal{G}} EM(g, m(g))}{\sum_{g \in \mathcal{G}} EM(g, g)}$$

In the rest of this work, we will report only the MELA metric, which consists of a uniformly weighted sum of the MUC, B-CUBED, and CAEF_e metric. More specifically, we will report on the MELA F1 score, which is the harmonic mean of the MELA precision and recall scores.

3 Transformers

Recently, the Transformer [1] model has become the dominant architecture for many NLP tasks, including (but not limited to): machine translation, document classification, and coreference resolution.

In subsection 3.1, we give an overview of the Transformer architecture. Since this architecture has become well-known throughout the field of deep learning, we will limit the discussion to a high-level overview (the reader who is familiar with Transformers can safely skip this subsection). We explain how the Transformer architecture can be leveraged to perform end-to-end coreference resolution in subsection 3.2. Finally, subsection 3.3 addresses the application of Transformers to low-resource languages and multilingual Transformers.

3.1 Transformer Architecture

Introduced by [1], the Transformer model first served as a sequence-to-sequence architecture. Sequence-to-sequence architectures are applied to problems such as machine translation and text summarization, where both the input and output is a sequence. These models consist of an encoder and decoder, which are respectively used to map the input sequence to a latent representation and map this latent representation to the output sequence. Whereas previous state-of-the-art sequence-to-sequence architectures rely on recurrent or convolutional operations to encode and decode sequences, the Transformer solely relies on feed forward operations and an attention mechanism to achieve state-of-the-art sequence-to-sequence results. Without this reliance on recurrence, the Transformer architecture is highly parallelizable.

Figure 1 gives a schematic overview of the Transformer architecture. Subsection 3.1.1 starts by describing the attention mechanism. How a sentence is pre-processed before being passed to a Transformer is addressed in subsection 3.1.2. Subsection 3.1.3 and 3.1.4 respectively describe the encoder and decoder module of the Transformer. Finally, subsection 3.1.5 explains how the original Transformer architecture is adapted to create Transformer architectures such as BERT [25] and GPT [26].

3.1.1 (Self-)Attention mechanism

The (self-)attention mechanism is the bread and butter of the Transformer architecture. It allows the model to create contextualized embeddings. For example, in the sentences *The Titanic will not sink* and *Put the dishes in the sink*, the semantics of the word *sink* are clearly context dependent. When forming an embedding for the work *sink*, attention allows the Transformer to *attend to* the different words in the sentence, leading to a better understanding of the context.

In this subsection, we describe the (self-)attention mechanism. In subsequent subsection, we will position this mechanism with regard to the rest of the Transformer architecture.

Consider two sequences of embeddings, $(x_i)_{i=1}^{i=N}$ and $(y_j)_{j=1}^{j=L}$, which will serve as input to the attention mechanism. The attention mechanism will transform the sequence of embeddings $(y_j)_{j=1}^{j=L}$ while attending to $(x_i)_{i=1}^{i=N}$. In the case of self-attention, both inputs are the same sequence: $(x_i)_{i=1}^{i=N} = (y_j)_{j=1}^{j=L}$. We can denote these sequences as the matrices X and Y respectively, with dimensions (N, h) and (L, h) where h represents the dimension of one token embedding.

First, a *query*, *key*, and *value* embedding are created per token embedding. Both the query and key embeddings have dimension k, while the value embeddings has dimension v. The query sequence $(q_j)_{j=1}^{j=L}$ is formed by applying a learned linear transformation to each y_j. This linear transformation can be denoted via the matrix W^q, having dimensions (h, k). The resulting matrix operation can be denoted as $Q = YW^q$, where Q has dimensions (L, k). Similar, the key and value are formed by applying a linear transformation to each x_i. This results in the following matrix operations: $K = XW^k$ and $V = XW^v$. The dimensions of K equals (N, k) and the dimension of V equal (N, v).

Now that the query, key, and value embeddings are available, we seek to create new, contextualized embeddings for y_j. We calculate an attention score $s_{i,j}$ between x_i and y_j by taking the dot product of q_j and k_i.

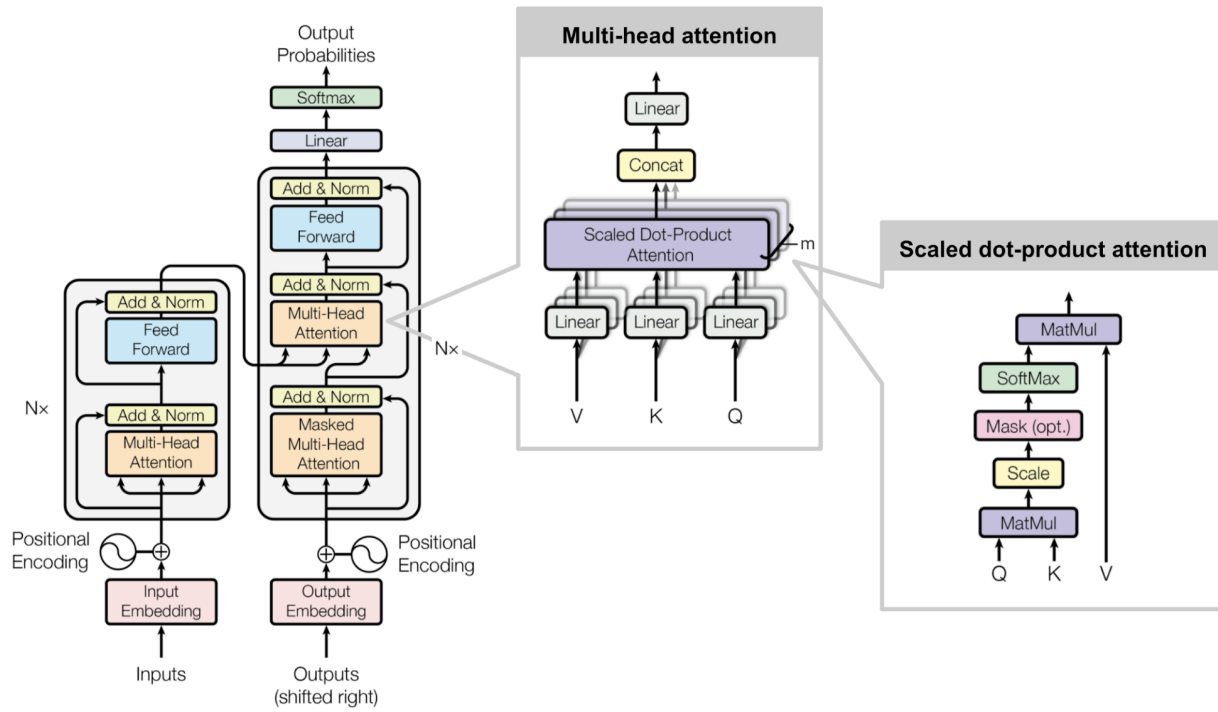


Figure 1: **The original Transformer architecture**. (Left) Encoder-decoder architecture consisting of multi-headed attention blocks, layer normalization and feed forward neural networks. (Middle) Multi-headed attention mechanism. (Right) Scaled-dot product attention. Images from Vaswani et al. (2017) [1].

This is denoted in matrix form as follows: $S = QK^T$, where $S_{j,i} = s_{j,i}$. Every entry in S is scaled by a factor $\frac{1}{\sqrt{k}}$, which leads to more stable gradients (a detail on which we will not elaborate). Finally, every row of S is passed through the *softmax* function, such that the attention scores associated with embedding y_j are normalized. The resulting operation becomes $S = softmax_{row}(\frac{QK^T}{\sqrt{k}})$.

These normalized attention scores S are now used to construct a linear combination of v_i for every y_j, which we will denote as z_j. This is given via the matrix operation: $Z = SV$, where the dimension of Z is equal to (L, v).

Z now contains for every element in the input sequence $(y_j)_{j=1}^{j=L}$ a new embedding that was formed by attending between the embeddings of $(x_i)_{i=1}^{i=N}$ and $(y_j)_{j=1}^{j=L}$. In a multi-headed attention mechanism, multiple Z matrices are constructed, denoted Z_0, Z_1, ..., Z_{m-1} where m represents the number of heads. For each of these matrices Z, the linear transformations to create Q, K, and V differ.

All these Z matrices are concatenated along the second dimension, creating a final matrix with dimension $(N, m \cdot v)$. The output of the attention mechanism, X', is formed via a final linear mapping, W_{final}, which transforms this matrix to dimensions (N, h).

The following equations summarize the multi-headed attention mechanism and correspond to the middle and right part of figure 1:

$$Q_0 = YW_0^q \tag{1}$$

$$K_0 = XW_0^k \tag{2}$$

$$V_0 = XW_0^v \tag{3}$$

$$Z_0 = softmax_{row}(\frac{Q_0 K_0^T}{\sqrt{k}})V_0 \tag{4}$$

$$Z = concat(Z_0, Z_1, ...Z_{m-1}) \tag{5}$$

$$X' = ZW^{final} \tag{6}$$

3.1.2 Pre-processing

Before we hand a piece of text to a Transformer, we first need to apply some pre-processing steps. First, we need to tokenize the text. Tokenization aims to split an input text into a limited amount of distinct (sub)words. While the actual splitting of the text into (sub)words is trivial, the decision which distinct (sub)words are used is not. The most famous tokenization schemes include Byte-Pair Encoding (BPE) [27], WordPiece [28], and SentencePiece [29]. Each of these techniques tries to learn an optimal set of (sub)words, in which to tokenize the input text. An important hyperparameter here is the vocabulary size, the total amount of distinct (sub)words the tokenizer considers. A low vocabulary size reduces the complexity of the model, since less unique (sub)words are considered. However, each (sub)word is less expressive.

Once the input text is tokenized, every (sub)word gets embedded. Additionally, a positional embedding is added to each (sub)word in the input text. This positional embedding consists of a signal the model can learn to leverage to estimate which position a (sub)word has in its input sentence.

3.1.3 Encoder

The encoder is used to form a contextualized latent representations of the input sequence. These contextualized representations are then used by the decoder to generate text in the target domain.

As previously mentioned, the input sequence is first pre-processed. The resulting sequence of embeddings is handed to the encoder.

The encoder consists of N stacked encoder layers, each identical in architecture – as can be seen in figure 1. One encoder layer consists of two stacked sublayers: one multi-headed self-attention layer and one feed-forward layer. After each sublayer, the output of the sublayer is summed with its input (via a skip-connection) and normalized [30]. This normalized result is passed to the following (sub)layer.

While the multi-headed self-attention layers transforms a sequence as a whole, the feed-forward network is applied in parallel to each embedding in the sequence.

3.1.4 Decoder

The decoder consists of a stack of N decoder layers. The decoder only differs from the encoder in two ways: (i) a decoder layer features an additional sublayer between the self-attention sublayer and the feed-forward sublayer, called an encoder-decoder attention sublayer, and (ii) the decoder is used in an autoregressive manner.

The encoder-decoder attention sublayer is very similar to the self-attention sublayer. Instead of calculating an attention score between the input of the layer and itself, the encoder-decoder attention mechanism calculates attention between the input of the layer and the embeddings outputted by the encoder. This allows the decoder to efficiently reference the contextualized embeddings generated by the encoder model.

While the encoder model takes as input an entire sequence and calculates attention between every pair of tokens in this sequence, the decoder model takes as input a partially generated sequence and autoregressively predicts the next word based on the contextualized embeddings generated from this partially generated sequence. In each step of this iterative algorithm, the self-attention mechanism is only applied between the pairs of tokens representing already generated words.

3.1.5 BERT- and GPT-like models

The success of the original Transformer model prompted many researchers to develop their own adaptations of this architecture. Specifically, researchers aimed to apply the Transformer architecture to non-sequence-to-sequence problems. We will discuss two of the best-known architectures, BERT [25] and GPT [26] and

highlight their differences. The most popular Transformer models today all somewhat resemble either BERT or GPT, although a lot of research focuses toward creating novel Transformer architectures. Additionally, we explain how these models can be pre-trained on large unlabeled datasets and subsequently fine-tuned on a smaller downstream dataset. It is partially due to this successful pre-training and easy fine-tuning that BERT and GPT have become so popular.

BERT-like models: The BERT model [25] consists of only a stack of encoder layers, the decoder from the original Transformer architecture is omitted. Thus, given an input sequence, BERT outputs a sequence of contextualized embeddings which can be used for further downstream tasks, named the *sequence output*. Additionally, BERT produces one extra embedding, named the *pooled embedding*, which serves to summarize the entire input sequence. This pooled embedding is a simple learned linear transformation of the first embedding of the sequence output.

The BERT model is pre-trained on two objectives: (i) the Masked Language Modelling (MLM) objective and (ii) the Next Sentence Prediction (NSP) objective. Both of these objectives are *self-supervised*, meaning that they require no labeled data to be trained. This, together with the high grade of parallelism, allows the BERT model to easily scale to very large amounts of data during pre-training.

During pre-training, BERT is given a concatenation of two sentences from a training corpus. These sentences are either adjacent sentences from the corpus or sentences picked randomly from the corpus. In both sentences, some words are randomly masked. BERT processes these input sentences and is tasked with (i) predicting the masked words (MLM objective) and (ii) predicting whether both sentences were originally adjacent (NSP objective), based on the contextualized embeddings formed. Performing well on these objectives requires excellent language skills. Interestingly, work has shown that by pre-training these objectives, BERT develops attention heads which are specialized towards mainstream NLP tasks like: POS tagging, parsing, named entity recognition, semantic role labeling, and coreference resolution [31].

Once the BERT model is pre-trained, it can be used for many downstream tasks. Typically, a head architecture is introduced which leverages either the sequence output or pooled output for world-level or sentence-level tasks respectively. This head architecture is trained from scratch on a smaller downstream dataset. During this downstream training, the possibility exists to either freeze or continue updating BERT's parameters.

Different variations of the BERT model differ in total number of layers used, as well as the size of the feed-forward network and the width of the token embeddings. Additionally, these models can be pre-trained on different dataset or even using different objectives.

GPT-like models: Whereas BERT is said to consist of the Transformer encoder, GPT is based on the Transformer decoder. GPT is a stack of decoder layers, where the encoder-decoder attention sublayer has been omitted. The main difference between BERT and GPT concerns how the self-attention is calculated. In BERT, the self-attention layer computes attention between all the pairs of input tokens. In GPT, self-attention is exclusively applied between a token and all its token to the left. This is sometimes referred to as *left-to-right* attention.

GPT aims to generate sequences. This is done in an autoregressive fashion. An input sequence is fed into the model. The contextualized output embeddings (using left-to-right attention) are used to predict the next word in the sequence. This next word gets concatenated with the original input sequence. The resulting sequence is again used to predict the new next word.

Pre-training GPT happens via a standard Language Modelling (LM) objective. Given a training corpus, GPT is fed parts of sentences and asked to predict the next word. Using the original corpus, the next word prediction of GPT can be evaluated and optimized.

3.2 End-to-end coreference resolution using Transformers

Where previously state-of-the-art results on the task of coreference resolution were attained via rule-based systems (such as [32]), end-to-end deep learning models have lately been consistently achieving new state-of-the-art results. Like with many NLP tasks, these deep learning systems have greatly benefited from the arrival of Transformer-based models.

The first successful end-to-end coreference resolution architecture that does not rely on syntactic parser or hand engineered features was introduced by [33]. The authors propose an architecture that, for each span of words in a text, assigns a previous span that refers to the same entity. Span pruning is used to filter out spans that do not refer to an entity. Based on these predictions, a final coreference clustering can be extracted.

Figure 2 shows the architecture for end-to-end coreference resolution proposed by Lee et al. [33]. This architecture consists of two consecutive steps: a mention scoring step and a coreference scoring step. In what follows, we will untangle this architecture in more detail.

The mention scoring step is described in subfigure 2a. Given embeddings of an input sequence, the architecture computes span representations for each span of words in the input sequence. Only spans shorter than a predefined number of words are considered (as too limit the total possible amount of spans). Each span representation is formed by concatenating (i) the word embedding of the span's first word, (ii) the word embedding of the span's last word, (iii) a learned weighted average over all the word embeddings contained in the span, and (iv) a feature vector encoding the length of the span.

For each produced span representation, the model computes a mention score via a fully-connected feed-forward neural network. The spans are ranked by mention score and a top amount of spans are kept while the rest are discarded. This pruning greatly reduces the computational needs of the subsequent steps. The amount of spans kept is defined as a fraction of the input sequence length. Additionally, top scoring spans that (partially) overlap with a better scoring span are also pruned. While aggressively pruning spans has a positive impact on the model complexity, it should be noted that a low mention recall (due to over aggressive pruning) will have a detrimental effect on the resulting performance of the model.

The coreference scoring step is described in subfigure 2b. For each pair consisting of a span and one of its first K antecedent spans, an antecedent score is calculated by passing the span representations through a fully-connected feed-forward neural network. Only K antecedents are considered to reduce the computational complexity. For every span-antecedent pair, a coreference score is calculated by summing the mention scores for the span and the antecedent together with the antecedent score of this pair.

The final coreference prediction of the model is produced by the most likely clustering of the spans based on the calculated coreference scores. Indeed, simply taking the most likely antecedent for each span does not produce the overall most likely clustering.

For a given mention, multiple antecedents are valid as long as they give rise to the same final coreference clustering. Therefore, the architecture is trained to optimize the log-likelihood of all the possible antecedent links implied by the gold truth coreference clustering.

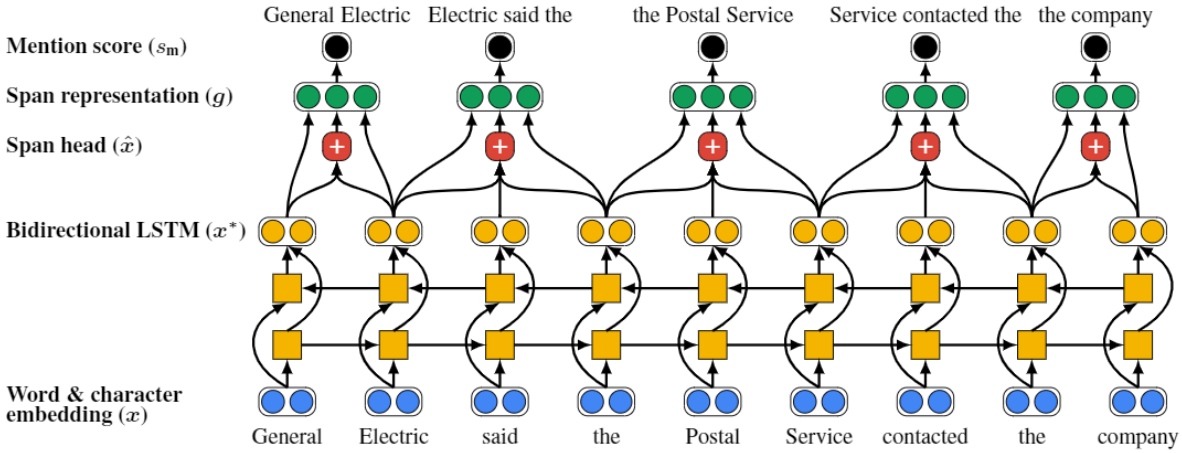
Like with many NLP systems, the final performance is strongly impacted by the quality of the initial world-level embeddings. Originally, [33] used a bidirectional LSTM [34] to produce contextualised word-embeddings of the input sequence. Both [35] and [36] found better results when using BERT to produce (sub)word-embeddings.

Additionally, [37] introduced SpanBERT to achieve even better performance. SpanBERT is specifically pre-trained to produce high-quality span embeddings. To this end, the authors pre-train SpanBERT by replacing the Next Sentence Prediction (NSP) objective with a Span-Boundary objective: given a sentence containing a masked span, the model is required to predict a masked word in this span using the boundary words of the span.

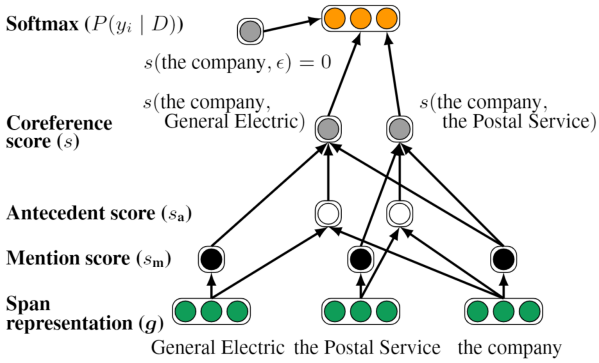
3.3 Transformers for low-resource languages & Multilingual Transformers

Lately, a plethora of low-resource language-specific pre-trained models have been developed [38, 39, 40, 41]. While these models are a good first step in getting low-resource NLP up to speed with English NLP, they still require (i) a large corpus to pre-train and (ii) language-specific annotated datasets for fine-tuning, both of which are not always readily available for low-resource languages.

Other efforts have been directed towards building multilingual Transformers [42, 5]. These models are pre-trained on a multitude of languages at the same time and aim to generalize several language understanding



(a) **The first mention scoring step**. For each span up to a predefined length, a mention score s_m is calculated. This score is calculated via a fully-connected neural network which takes a span representation g as input. Each span representation consists of two embeddings representing the first and last word in the span (x^*), and a learned linear combination of all the embeddings in the span ($\hat{x}$). The contextualized word embeddings $\hat{x}$ are formed via a Bidirectional LSTM. Additionally, a feature vector used to encode the length of the span is considered when calculating s_m (not shown on the figure). Based on the calculated mention scores, the top-scoring non-overlapping spans are considered as potential mentions and serve as input to the second step of the architecture (not shown on the figure).



(b) **The second coreference scoring step**. Given the top-scoring mentions from the previous step, an antecedent score s_a is calculated between each mention and its K antecedents. The antecedent score s_a is calculated via a fully-connected neural network using the span representations g of both the mention and its antecedent. The resulting antecedent score is summed with the mention scores s_m of both the mention and the antecedent to form the coreference score s. For a given mention, all the coreference scores for the antecedents are normalized. Given all mentions and resulting coreference scores, the most likely coreference clusters are extracted (not shown on the figure).

Figure 2: **End-to-end coreference resolution architecture**. Subfigure (a) shows the first mention scoring step. Subfigure (b) shows the second coreference scoring step. Images from Lee et al. (2017) [33].

tasks (e.g., the recognition of named entities in a text) across the different languages.

In [42], XLM is introduced. the authors pre-train a 12-layer Transformer encoder on multiple languages and achieve state-of-the-art results on cross-lingual natural language interference and (un)supervised machine translation. Interestingly, their model achieves better perplexity on some low-resource languages compared to a Transformer directly trained on these languages. This finding indicates how a multilingual model might perform better on low-resource languages when not enough data is available to fully pre-train a model specifically for these languages.

Similar to how BERT was pre-trained, the authors pre-train XLM using a Masked Language Modelling (MLM) objective. When no parallel data is available for the languages on which XLM is being pre-trained, an additional Causal Language Modelling (CLM) objective is added. This CLM objective aims to predict the word following the given sequence. When parallel data is available, the authors propose Translation Language Modeling (TLM), a novel objective. During TLM, two parallel sentences over different languages are concatenated and fed to the model. In both sentences, words are randomly masked. Similar to the MLM objective, the model is asked to predict these words. Via this objective, the model can learn to leverage context in the parallel sentence. The authors reason that this additional objective helps align the embeddings of similar words and contexts across the language barrier. Additionally, the authors show that this objective has a high impact on the performance of the model on downstream cross-lingual tasks.

Building on top of XLM, XLM-R [5] achieves even better results by scaling the model capacity and increasing the amount of data used during pre-training. The result is a large model, containing 270M parameters, which performs better on several low-resource languages compared to their language-specific models. Additionally, the authors show that, for the first time, a large multilingual model can also achieve on par with monolingual models on high-resource languages.

4 Domain Adaptation

One of the key problems in deep learning is the lack of generalization capability across tasks and data. A deep learning model which achieves great results on the data distribution on which it was trained can perform much worse when evaluated on a different distribution. This is typically the case when the data presented to the model for evaluation was collected in a different way, describes a slightly different task or when the data distribution naturally drifted over time between the moment of training and the moment of evaluation. The subfield of domain adaptation tries to partially alleviate these problems by designing techniques that are more robust to changes in the input data distribution.

In section 4.0.1, we formally describe the goals of domain adaptation and consider the link between domain adaptation and cross-lingual learning. Domain Adversarial Neural Networks (DANN), one of the most popular domain adaptation techniques, is outlined in section 4.0.2. We explain the mechanism of DANN and highlight why this technique in particular has received much attention in the field of domain adaptation.

4.0.1 Domain Adaptation

A domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$ is defined by a feature space $\mathcal{X}$ and a probability distribution over this feature space $P(X)$. A task $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$ consists of a label space $\mathcal{Y}$ and a likelihood function over the label space given the feature space $P(Y|X)$. The objective of a typical supervised learning algorithm is to model this likelihood $P(Y|X)$, given a set of samples $(x_i, y_i)_{i=1}^{i=N}$ from the feature space with corresponding labels [6].

Consider a source domain $\mathcal{D}_s = \{\mathcal{X}, P_s(X)\}$ and a target domain $\mathcal{D}_t = \{\mathcal{X}, P_t(X)\}$. While the feature space $\mathcal{X}$ of both source and target domain are identical, this is not the case for the probability distribution $P_s(X)$ and $P_t(X)$. In the case of domain adaptation, we aim to model the likelihood $P(Y|X)$ of a task $\mathcal{T}$ given a source domain $\mathcal{D}_s$ such that the modelled likelihood is optimally applicable to the same task $\mathcal{T}$ applied to the target domain $\mathcal{D}_t$. In this case, the model is said to generalize from the source domain to the target domain.

The situation described above is sometimes also referred to as *transductive transfer learning* [43]. This is not to be confused with *inductive transfer learning* (sometimes just referred to as *transfer learning*) where the aim is to learn a task $\mathcal{T}_t$ using a related task $\mathcal{T}_s$. The key difference is that transductive transfer learning considers the same task over two domains. While inductive transfer learning is the mechanism by which we are able to leverage a pre-trained Transformer for a downstream task, we will not explicitly describe inductive transfer learning in this work.

Typically, the problem of domain adaptation is only considered when more data from $\mathcal{D}_s$ is available compared to data from $\mathcal{D}_t$. If this were not the case, it would be more efficient to directly model $P(Y|X)$ using samples from $\mathcal{D}_t$. A distinction should be made between supervised and unsupervised domain adaptation. In the supervised case, labeled samples from $\mathcal{D}_t$ are available. In the unsupervised case, no labeled samples from $\mathcal{D}_t$ are considered. However, unlabeled data from $\mathcal{D}_t$ might still be abundant.

Where typically domain adaptation is applied to NLP in the context of a distribution shift $P_s(X) \neq P_t(X)$ where source and target domain are defined over the same language [6], domain adaptation can also be framed from a multilingual point of view. In this case, the source and target domain correspond to a source and target language.

4.0.2 Domain Adversarial Neural Networks

Many different domain adaptation techniques exist. In this work we will only consider the technique of Domain Adversarial Neural Networks (DANN). For a more detailed overview into domain adaptation for NLP, we refer to Ruder's comprehensive thesis on neural transfer learning for natural language processing [43] or a recent survey paper by Ramponi et al. [6].

DANNs were first introduced by Ganin et al. [44] in the context of computer vision models. The authors argue that domain adaptation can be achieved via the use of features that are both *task-discriminative* and *domain-invariant*.

Given a typical model consisting of a feature extractor and classification head, domain adaptation is achieved by adding a *domain classifier* network to the feature extractor. This domain classifier is trained to optimally discriminate between the extracted features from the source and target domain. When the resulting model is optimized for the task at hand, the feature extractor is pushed to form task-discriminative features. Additionally, the feature extracted is trained to maximize the loss produced by the domain classifier. This pushes the feature extractor to form domain-invariant features. This combination of maximizing the loss of the domain classifier with domain-invariant features while simultaneously training the domain classifier to optimally distinguish between the domains gives rise to an adversarial setting between the feature extractor and domain classifier. Much like how the adversarial setting in GANs [45] helps minimize the discrepancy between synthetic and real data, the adversarial setting introduced here minimizes the discrepancy between both domains.

DANNs are a widely applicable technique. Virtually any deep learning model can be extended with a domain classifier and augmented with the adversarial loss. One downside is that the resulting model will only be able to model features that are shared over both domains. Features that are only available in source or target domain get removed during the adversarial process.

In section 7.1 we discuss the training of a DANN directly applied to our proposed model, in more detail.

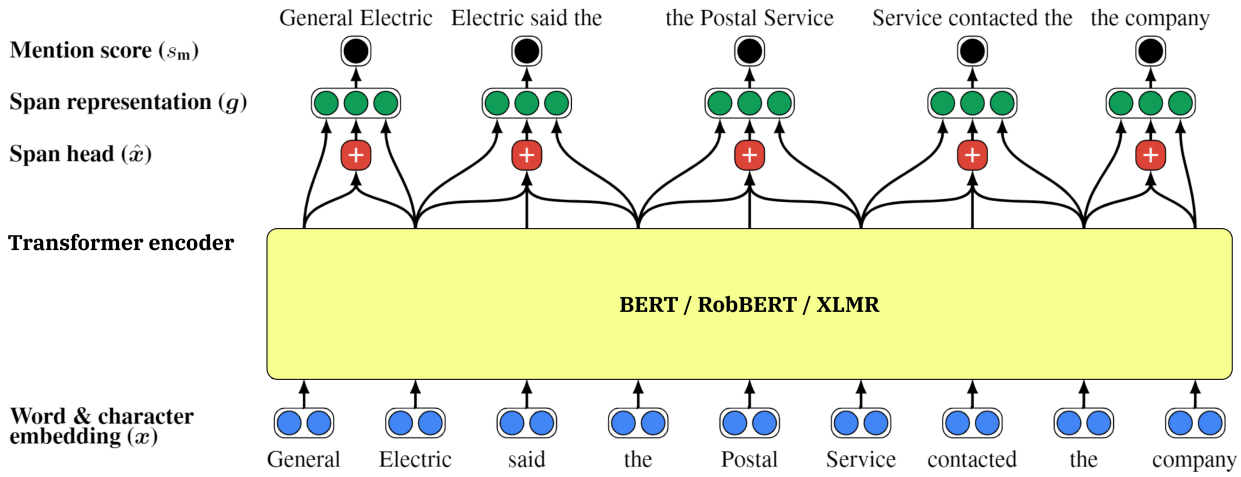


Figure 3: **End-to-end coreference resolution mention scoring using a Transformer encoder**. Image adapted from Lee et al. (2017) [33]. See section 3.2 for a detailed explanation.

5 Monolingual training

First, we aim to construct a baseline for Dutch coreference resolution using deep learning systems. We experiment with different Transformer models to build this baseline. Additionally, we compare our results with existing state-of-the-art English and Dutch coreference resolution baselines.

In subsection 5.1, we discuss the models used for this experiment as well as any relevant hyperparameters. The exact experiment setup is given in subsection 5.2. Subsection 5.3 describes the attained results and positions them with regard to the existing state-of-the-art. Finally, subsection 5.4 concludes the monolingual training experiment.

5.1 Models

The previously described coreference resolution architecture [35] (see section 3.2 for a detailed overview) serves as the basis for our model. This architecture relies on a Transformer encoder to generate contextualized embeddings of the input sequence. We can change this encoder in function of the languages we train on. For the experiments on English data, we keep the original BERT model [25] in the end-to-end architecture. For the Dutch experiments, we replace BERT with RobBERT [39], a variant of BERT which has been pre-trained on Dutch data. Additionally, we experiment with using XLMR [5] – a multilingual model – to produce the contextualized embeddings, both for the Dutch and the English experiments.

Figure 3 shows how the Transformer encoder is used in the mention scoring step of the end-to-end architecture. The subsequent coreference scoring step is unaffected by the introduction of the Transformer encoder and is thus not shown.

The hyperparameters used in this experiment are largely the same as those proposed by the original architecture [35], as described in subsection 3.2. We scaled the feed-forward neural-network size ($ffnn\_size$) down to 1000 from 3000. This slightly decreases the end performance but achieves faster training. We increased the max number of sentences the model can process per batch ($max\_training\_sentences$) from 11 to 30. This allowed the model to better fit the Dutch data, since the Dutch data features on average more sentences per document.

Model	Trained on	Evaluated on	MELA F1 (%)
Published results			
Joshi et al. (2019a) [35]	OntoNotes, train	OntoNotes, dev	**73.9**
Our results			
BERT	OntoNotes, train	OntoNotes, dev	**72.91**
RobBERT	OntoNotes, train	OntoNotes, dev	63.02
XLMR	OntoNotes, train	OntoNotes, dev	**72.90**

Table 1: **English monolingual performance**, comparing our best achieved results with the most comparable results from literature. We achieve one percent worse performance compared to literature, due to our models using a smaller feed-forward neural-network size. Interestingly, we found that XLMR (a multilingual model) performs as good as BERT (an English monolingual model). Unsurprisingly, RobBERT (a Dutch monolingual model), performs worst.

5.2 Experiment setup

First, we run experiments to study the monolingual coreference resolution performance of BERT, RobBERT, and XLMR. To measure the English performance, we train and evaluate all three models on the OntoNotes dataset. Measuring the Dutch performance is done by training and evaluating all three models on the SoNaR dataset. Note that, in these experiments, we are also training and evaluating BERT (an English model) on the Dutch dataset, and RobBERT (a Dutch model) on the English dataset. This gives us information about the importance of the specific language of the Transformer used in the architecture.

Even though XLMR is a multilingual model (i.e., pre-trained on multiple languages) we are exclusively running monolingual experiments in this step (i.e., only fine-tuning on coreference resolution for one language). Experimenting with XLMR in a monolingual setting allow us to build a fairer comparison with regard to the second step of this work (see section 6): we will investigate the use of XLMR in a multilingual setting (i.e., fine-tuning the model on multiple languages) and discuss the benefit of migrating from a monolingual training setup to a multilingual one.

Finally, we investigate the zero-shot cross-lingual performance of these three modules. To measure the English-to-Dutch transfer, we train each model on the English data and evaluate on the Dutch data. This transfer is zero-shot, the model is never fine-tuned on Dutch data before evaluation. We do a similar experiment to measure the Dutch-to-English transfer. These experiments are particularly interesting since a good cross-lingual transfer might indicate the use of multilingual training, which we will further investigate in section 6.

5.3 Results

In section 5.3.1 and section 5.3.2, the attained results for the monolingual English and Dutch experiments respectively are interpreted. Section 5.3.3 describes the zero-shot cross-lingual results.

5.3.1 English performance

Table 1 shows the performance for the English monolingual experiments. Comparing the published results of Joshi et al. [35] with our most comparable result (BERT), we see a drop in performance of 1%. This is due to the lowered $ffnn\_size$ we use to train faster. As expected, the Dutch model RobBERT performs noticeably worse (-10%) compared to BERT and XLMR. This is due to RobBERT not being pre-trained on English data, indicating the importance of using a Transformer compatible with the downstream task. Interestingly, the multilingual XLMR model achieves results on par with the BERT model on the monolingual task. This latter result strengthens the claim made by [5]: XLMR is competitive with high-resource monolingual models such as BERT[3].

[3] XLMR is the first multilingual model where this is shown to be the case.

Model	Trained on	Evaluated on	MELA F1 (%)
Published results			
Poot et al. (2020) [18]	SoNaR, train	SoNaR, dev	**71.53**
Van Cranenburgh (2019) [46]	SoNaR, train	SoNaR, dev	55.45
Our results			
BERT	SoNaR, train	SoNaR, dev	39.67
RobBERT	SoNaR, train	SoNaR, dev	44.50
XLMR	SoNaR, train	SoNaR, dev	**49.63**

Table 2: **Dutch monolingual performance, including singletons** for the SoNaR dataset. Our best result severely lags behind the existing state-of-the-art, due to our model not being able to predict the singleton entities present in the Dutch dataset. Interestingly, XLMR (a multilingual model) performed better compared to RobBERT (a Dutch monolingual model).

Model	Trained on	Evaluated on	MELA F1 (%)
Published results			
Poot et al. (2020) [18]	SoNaR, train	SoNaR, dev	**52.76**
Van Cranenburgh (2019) [46]	SoNaR, train	SoNaR, dev	46.96
Our results			
BERT	SoNaR, train	SoNaR, dev	50.02
RobBERT	SoNaR, train	SoNaR, dev	**56.21**
XLMR	SoNaR, train	SoNaR, dev	**62.20**

Table 3: **Dutch monolingual performance, excluding singletons** for the SoNaR dataset. Our best result achieves new state-of-the-art result on the SoNaR dataset when excluding singletons. Comparing the result using RobBERT with the previous state-of-the-art, we notice an increase in performance due to our approach being slightly different. Interestingly, the biggest gain in performance is due to using XLMR (a multilingual model) compared to RobBERT (a Dutch monolingual model).

5.3.2 Dutch performance

In order to build a fair comparison between the Dutch and English performance, we need to consider the slight task difference over both languages. As mentioned in section 2.2, singletons (i.e., entities mentioned only once in a document) are not annotated in the OntoNotes dataset. They are, however, annotated in the SoNaR-1 dataset. This poses a problem, since the neural architecture for coreference resolution was designed for the OntoNotes dataset, and is thus incapable of outputting singletons. To achieve competitive results on the SoNaR-1 dataset, we should either adapt the original architecture to allow for singletons or remove singletons from the SoNaR-1 dataset altogether. We chose the latter approach, since removing singletons from the SoNaR-1 dataset allows us to use one single architecture on both languages, which will make our next step (multilingual training) much easier. We give insights into the effect of removing singletons by also training and evaluating on SoNaR-1 before removing the singletons.

We compare our results with those obtained by Poot et al. [18], a concurrent work also aimed at creating a neural baseline for Dutch coreference resolution. Both our work and the work by Poot et al. is based on the end-to-end coreference resolution architecture introduced by Lee et al. [33, 47] and further developed by Joshi et al. [35]. Contrary to our approach, Poot et al. dealt with the singleton issue by allowing their architecture to predict singletons. The authors also report on results where they exclude singletons from their system output and evaluation data (however, they always train on data containing singletons). For the rest, their approach is very similar to ours except for two details: (i) they used BERTje [38] as their Dutch encoder while we use RobBERT and (ii) they freeze their encoder while fine-tuning the head architecture for coreference resolution while we fine-tune the head architecture as well as the encoder. We also compare

our results with those of Van Cranenburgh [46], a rule-based Dutch coreference resolution system based on Stanford's multi-pass sieve coreference resolution system [32]. This rule-based system held the state-of-the-art result for Dutch coreference resolution prior to our work and the concurrent work of Poot et al.

Table 2 shows the obtained results when not excluding singletons from the SoNaR dataset. Notice the large gap in performance between the best published result by Poot et al. (71.53% F1) and our best model (49.63% F1). This gap can be largely contributed to our architecture not predicting singletons. As can be expected, the BERT model applied to the Dutch data performs worst, again indicating the importance of matching the Transformer to the downstream task. While the RobBERT model achieves a gain of about 5% with regard to BERT, it is interesting to note that XLMR again achieves a 5% gain on RobBERT. The multilingual model performed better than the Dutch model, suggesting that Conneau et al. [5] were right in stating that for low-resource languages, multilingual models can outperform monolingual models.

Table 3 shows the results when singletons are excluded. Remember, in our models this means we are excluding singletons from the training and evaluation data. The models by Poot et al. and Van Cranenburgh never exclude singletons from their training data, they exclude the singletons from their system output and ground-truth data during evaluation. We immediately notice a large drop in performance (-18.77%) when comparing Poot et al.'s architecture on the dataset with and without singletons. This is due to singletons being relatively easy to predict, and thus inflating the achieved accuracy. Looking at our results, we again see the same ranking between the models: the English model performs worst and the multilingual model performs best.

Comparing our best result on SoNaR without singletons, we notice a 9.44% performance gain over Poot et al. Because our architectures are in essence very similar, this gap is most likely due to (i) our use of a multilingual model – which consistently outperforms a monolingual Dutch model in all our experiments, (ii) the fact that we update our Transformer during training, and (iii) Poot et al. having an adapted architecture that is able to predict singletons – even though these predictions are removed from the system output upon evaluation. From our experiments, we deduce that more than half of this gain (5.99%) is due to our use of a multilingual model. The remaining 3.45% is due to the other reasons mentioned above.

5.3.3 Zero-shot cross-lingual performance

Other than the monolingual results described above, we are interested in studying the zero-shot cross-lingual transfer of our model. In these experiments, we train a model on one language and evaluate it on the other, without first fine-tuning it on the evaluation language (hence zero-shot). We have already learned that the multilingual model outperforms the Dutch model in a monolingual setting, now we are interested in learning whether the multilingual model can also aid cross-lingual transfer.

Table 4 shows the zero-shot cross-lingual results when including singletons for the Dutch dataset. The results when evaluating on the Dutch dataset are noticeably lower compared to evaluating on the English dataset, due to the model not being able to output singletons.

As can be seen in table 4, a very strong cross-lingual transfer is achieved using the multilingual XLMR model. XLMR has been pre-trained on many languages, including English and Dutch. We hypothesize this strong transfer is due to XLMR producing similar output embeddings for parallel input sentences (i.e., a sentence and its translation in another languages) [5]. Indeed, XLMR was pre-trained with a Translation Language Modeling objective, which aims to achieve just that (see subsection 3.3). Thus, we can view fine-tuning XLMR on coreference resolution for one language as essentially initializing the head architecture for the other language: because the embeddings produces by XLMR over different language are similar, the initialized head architecture is partially applicable to a language for which it has not been trained.

The cross-lingual results for XLMR are strong. However, there is still room for improvement: Dutch-to-English transfer for XLMR is lagging behind 22,88% compared to training XLMR on English. Interestingly, English-to-Dutch transfer falls only 11.57% short. This could be due to the OntoNotes dataset containing roughly 60% more data than SoNaR, thus being a richer source to transfer from.

This remaining gap between the monolingual result and the cross-lingual transfer result described above could be due to two reasons: (i) the embeddings produced by XLMR are not exactly similar for semantically

Model	Trained on	Evaluated on	MELA F1 (%)
Dutch-to-English transfer			
BERT	SoNaR, train	OntoNotes, dev	22.53
RobBERT	SoNaR, train	OntoNotes, dev	19.72
XLMR	SoNaR, train	OntoNotes, dev	**50.02**
English-to-Dutch transfer			
BERT	OntoNotes, train	SoNaR, dev	7.89
RobBERT	OntoNotes, train	SoNaR, dev	5.44
XLMR	OntoNotes, train	SoNaR, dev	**38.06**

Table 4: **Zero-shot cross-lingual transfer results, including singletons** for SoNaR dataset. In both Dutch-to-English and English-to-Dutch transfer, XLMR strongly outperforms the monolingual models.

Model	Trained on	Evaluated on	MELA F1 (%)
Dutch-to-English transfer			
BERT	SoNaR, train	OntoNotes, dev	18.48
RobBERT	SoNaR, train	OntoNotes, dev	12.00
XLMR	SoNaR, train	OntoNotes, dev	**51.81**
English-to-Dutch transfer			
BERT	OntoNotes, train	SoNaR, dev	9.88
RobBERT	OntoNotes, train	SoNaR, dev	6.78
XLMR	OntoNotes, train	SoNaR, dev	**49.85**

Table 5: **Zero-shot cross-lingual transfer results, excluding singletons** for SoNaR dataset. In both Dutch-to-English and English-to-Dutch transfer, XLMR strongly outperforms the monolingual models.

close sentences over different input languages or (ii) there is a task difference over the input languages. We know both of these reasons to be true to some extent: (i) the embeddings produced by XLMR are not perfectly aligned [5] and (ii) SoNaR and OntoNotes differ in their exact annotation guidelines (see subsection 2.2).

5.4 Conclusions

With these experiments we have indicated the viability of applying a multilingual transformer like XLMR to a low-resource language in a monolingual setting.

Our results show that the multilingual model performs consistently better compared to the Dutch monolingual model for the task of coreference resolution: a 5.99% increase in performance was achieved. This finding is particularly interesting for situations where a downstream dataset for a low-resource language exists, but no pre-trained monolingual model is available.

Additionally, we find a much stronger zero-shot cross-lingual transfer when using a multilingual model for coreference resolution.

These results indicate that using multilingual models for coreference resolution is a viable alternative to using monolingual models – sometimes even outperforming them – and that multilingual models are able to better transfer learned coreference resolution from a source language to a target language.

In the following experiment, we aim to exploit these findings to further push the performance achieved with multilingual models.

6 Multilingual training

In this step, we leverage the findings of our monolingual experiments to train a better multilingual coreference resolution system. Based on our core hypothesis, we aim to incorporate additional English coreference resolution data during training to increase performance on the Dutch task. We investigate different ways of creating a multilingual training dataset, the impact of singletons in multilingual training, and multilingual warm-up training. Additionally, we study the convergence of the individual languages during multilingual training.

In subsection 6.1 and 6.2, an overview of the models and data used is given. The experiment setup is described in subsection 6.3. We discuss our results in subsection 6.4. Finally, subsection 6.5 gives an overview of our findings and proposes a conclusion.

6.1 Models

We will exclusively consider XLMR in these experiments, as we have shown XLMR is able to attain the best result achieved with BERT and surpasses the results achieved by RobBERT.

No changes concerning the architecture, hyperparameters or optimization procedure is made compared to previous experiments.

6.2 Data

We are interested in learning the effect of jointly training XLMR on Dutch and English data. To this extent, we need to merge the train splits of OntoNotes and SoNaR. It should be noted that, while SoNaR has fewer documents and less total amount of tokens, SoNaR has more tokens per document on average.

Several merging techniques are investigated. First, we consider the *full dataset*: all documents of SoNaR and OntoNotes are merged and shuffled, to produce one final dataset containing all the English and Dutch data. Second, we explore a *document-balanced dataset*: only the first N English documents in the dataset are kept, where N is the total number of Dutch documents. These resulting documents are merged and shuffled, producing a dataset with equal amount of documents for both languages. Because the model essentially treats one document as one batch, this results in an equal amount of Dutch and English batches during training. Finally, a *token-balanced dataset* is created: the first K English documents are kept so that the sum of all tokens in these K documents is approximately equal to the sum of all tokens in the Dutch dataset. This produces a dataset containing an equal amount of Dutch and English tokens. Given that we are dealing with low-resource languages, forming a token-balanced dataset will disregard a large portion of the available high-resource languages. While the aim of this work is to leverage as much high-resource data is possible, we are interested to see how the training procedure responds to different balances of data and therefore consider this token-balanced dataset in our experiments.

Table 6 shows several statistics of these proposed datasets. We will only use the train-split of the merged datasets, since we will evaluate our performance directly on OntoNotes or SoNaR. Briefly summarized, the full dataset train-split contains almost 2M tokens. Of these tokens, 69% are from the English data. Only 12.7% of the 3,211 documents in the merged train-split belong to SoNaR, indicating that the model will see roughly 6.9 times more English batches than Dutch batches. This batch imbalance could produce an issue during training. Therefor, we construct a document-balanced dataset to tackle this problem. However, in constructing this dataset, 85% of the original English documents need to be discarded. The train-split contains only 800,495 tokens (less than half of the amount of tokens in the full dataset), of which 77% are Dutch. While this dataset is balanced at the batch level, it introduces imbalance at the token level (since a Dutch batch contains more data on avarage). Alternatively, we can create a token-balanced dataset. We obtain a dataset containing 1,232,989 tokens in total over 2,019 documents, of which only 20.3% documents are Dutch (which, as is the case for the full dataset, leads to a batch-imbalance). An advantage of the token-balanced dataset is that we discard only 45% of the original English documents, compared to discarding 85% in the document-balanced dataset.

Split	Total documents	OntoNotes documents (percentage total)	SoNaR documents (percentage total)	Total tokens	OntoNotes tokens (percentage total)	SoNaR tokens (percentage total)
Full dataset						
Train	3,211	2,802 (87.3%)	409 (12.7%)	1,996,689	1,380,104 (69.1%)	616,585 (30.9%)
Dev	441	343 (77.8%)	98 (22.2%)	335,940	173,394 (51.6%)	162,546 (48.4%)
Test	456	348 (76.3%)	108 (23.7%)	363.325	179,755 (49.5%)	183,570 (50.5%)
Document-balanced dataset						
Train	818	409 (50.0%)	409 (50.0%)	800,495	183,910 (23.0%)	616,585 (77.0%)
Dev	196	98 (50.0%)	98 (50.0%)	200,392	37,846 (18.9%)	162,546 (81.1%)
Test	216	108 (50.0%)	108 (50.0%)	231,644	48,074 (20.8%)	183,570 (79.2%)
token-balanced dataset						
Train	2,019	1,610 (79.7%)	409 (20.3%)	1,232,989	616,404 (50.0 %)	616,585 (50.0%)
Dev	429	331 (77.2%)	98 (22.8%)	324,454	161,908 (49.9%)	162,546 (50.1%)
Test	456	348 (76.6%)	108 (23.7%)	363,325	179,755 (49.5%)	183,570 (50.5%)

Table 6: **Number of documents and tokens in the merged datasets**

6.3 Experiment setup

We train XLMR on all three datasets proposed above. Each trained model is evaluated on both the SoNaR and OntoNotes dataset. We perform these sets of experiments twice, including and excluding singletons in the SoNaR dataset.

Additionally, we consider a multilingual warm-up technique: we first pre-train XLMR on English coreference resolution before fine-tuning XLMR on Dutch coreference resolution without singletons. In this setting, the model is sequentially trained on coreference resolution for multiple languages. This contrasts with the joint multilingual experiments, where the model is trained on multiple languages at the same time.

6.4 Results

Section 6.4.1 summarizes the findings of the joint multilingual training experiment. The result of the warm-up experiment is discussed in section 6.4.2.

6.4.1 Joint multilingual training

We train XLMR on the full, token-balanced, and document-balanced datasets and evaluate this jointly trained model on OntoNotes and SoNaR. Table 7 shows the results of these experiments when the singletons in SoNaR are included, table 10 describes the same experiment when the singletons are removed from the SoNaR dataset. Immediately we notice that when evaluating on SoNaR, the different ways to merge the dataset has little impact on the end result. This is not very suprising: since SoNaR is the minority dataset both in terms of total amount of tokens and amount of documents, all three datasets contain the same amount of Dutch data. When evaluating on OntoNotes, the choice of dataset does have an impact. The full dataset achieves best results, followed by the token-balanced as second best and document-balanced as worst. This is also not surprising: the ranking directly corresponds with the total amount of English data in each dataset. We conclude that, even though the full dataset contains almost 7 times more English batches compared to Dutch batches, it still gives the best overall performance because it holds the most data. Thus, training the architecture is robust against a batch-imbalance[4].

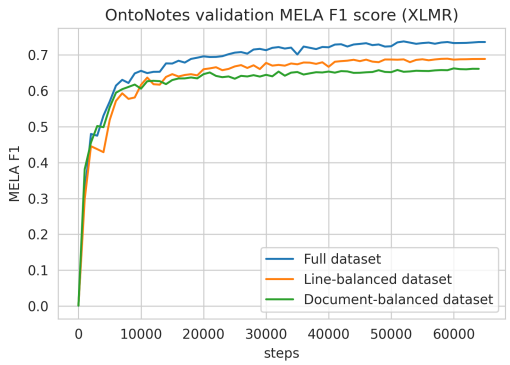
Figure 4 shows the evaluation accuracy during the joint training process per language on all three datasets. We notice very little difference in convergence over the different datasets. Only the end result for the English language is impacted. One small benefit of using a document-balanced dataset can be seen in subfigure (b): the Dutch performance converges slightly faster.

Now we compare our best models from the monolingual experiments with our best jointly trained models. Table 9 shows this comparison when singletons are included in the SoNaR dataset, table 10 shows the comparison when singletons are excluded. Interestingly, the jointly trained multilingual model performs best in all cases. However, most increases due to jointly training are below 1%, making it hard for us to determine their significance. The increase of +1.62% when jointly training without singletons and evaluating on SoNaR, however, is notable. The Dutch coreference resolution performance is modestly increased when incorporating English training data, suggesting that some positive cross-lingual transfer is benefiting the low-resource task (as our zero-cross cross-lingual result from section 5.3.3 suggested might happen). It should be noted that we only see this positive transfer when singletons are removed from the SoNaR dataset. Again, this does not surprise us: when removing the singletons from the Dutch task, the Dutch and English task descriptions are more aligned – providing additional opportunity for positive transfer between both tasks.

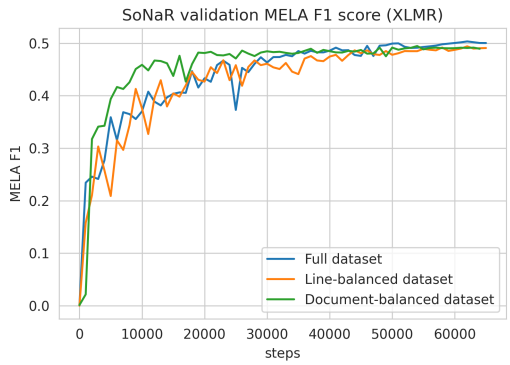
Even though we only notice a significant increase in performance when jointly training without singletons, it should be noted that in both cases the multilingual model was able to match the results previously obtained by two distinct monolingual models. This itself is impressive.

One additional question that arises is how the performance on the individual languages converges during multilingual training. It might be the case that both English and Dutch converge at a different rate during multilingual training. Considering the batch-imbalance in the full dataset, Dutch might converge slower compared

[4]For the next step of this work – described in section 7 – this will not remain true. Therefore, we will end up revisiting the use of a document-balanced dataset.



(a) OntoNotes validation MELA F1 during training of XLMR.



(b) SoNaR (including singletons) validation MELA F1 during training of XLMR.

Figure 4: **MELA F1 validation score during training of XLMR** on (a) OntoNotes and (b) SoNaR (including singletons) for the full, token-balanced, and document-balanced datasets. The final performance for (a) OntoNotes is dependent on the dataset used. The final performance of (b) SoNaR is invariant of the considered dataset.

Model	Trained on	Evaluated on	MELA F1
Full dataset			
XLMR	OntoNotes, train + SoNaR, train	OntoNotes, dev	**73.57**
XLMR	OntoNotes, train + SoNaR, train	SoNaR, dev	**49.93**
token-balanced dataset			
XLMR	OntoNotes, train + SoNaR, train	OntoNotes, dev	68.95
XLMR	OntoNotes, train + SoNaR, train	SoNaR, dev	49.41
Document-balanced dataset			
XLMR	OntoNotes, train + SoNaR, train	OntoNotes, dev	66.25
XLMR	OntoNotes, train + SoNaR, train	SoNaR, dev	49.47

Table 7: **Training XLMR on merged datasets, including singletons** for the SoNaR dataset. The best English and Dutch performance is achieved when training XLMR on the full dataset.

to English. Additionally, both languages could be *fighting* over the available model capacity: an increase in performance on one language might correlate with a decrease on the other.

Figure 5 gives us an insight into this question, by showing the convergence of the evaluation accuracy for both languages during the joint training process (considering the full dataset without singletons). Our worries are put to rest: both languages converge in a similar manner (even when a high batch-imbalance is present) and the curves do not suggest that both languages are *fighting* over model capacity.

6.4.2 Warm-up training

While the results described above indicate that jointly training coreference resolution can be a viable option to further increase performance on low-resource languages, the techniques comes at a cost. Indeed, fine-tuning a model on a merged dataset consisting of high-resource and low-resource data is more computationally expensive compared to just fine-tuning on the low-resource data.

Therefore, instead of joint training, we investigate the scenario of warm-up training. Before the model is fine-tuned on Dutch coreference resolution, we introduce an additional pre-training step where the model is first pre-trained on English coreference resolution.

Model	Trained on	Evaluated on	MELA F1
Full dataset			
XLMR	OntoNotes, train + SoNaR, train	OntoNotes, dev	**73.68**
XLMR	OntoNotes, train + SoNaR, train	SoNaR, dev	**63.82**
token-balanced dataset			
XLMR	OntoNotes, train + SoNaR, train	OntoNotes, dev	68.98
XLMR	OntoNotes, train + SoNaR, train	SoNaR, dev	61.63
Document-balanced dataset			
XLMR	OntoNotes, train + SoNaR, train	OntoNotes, dev	64.27
XLMR	OntoNotes, train + SoNaR, train	SoNaR, dev	62.56

Table 8: **Training XLMR on merged datasets, excluding singletons** for the SoNaR dataset. The best English and Dutch performance is achieved when training XLMR on the full dataset.

Model	Trained on	Evaluated on	MELA F1
Dutch performance			
XLMR	SoNaR, train	SoNaR, dev	49.63
XLMR	OntoNotes, train + SoNaR, train	SoNaR, dev	**49.93 (+0.30)**
English performance			
XLMR	OntoNotes, train	OntoNotes, dev	72.90
XLMR	OntoNotes, train + SoNaR, train	OntoNotes, dev	**73.57 (+0.67)**

Table 9: **Comparing monolingual training with multilingual training on the full dataset, including singletons** for the SoNaR dataset. Both for the Dutch and English performance, multilingual training performs marginally better.

Model	Trained on	Evaluated on	MELA F1
Dutch performance			
XLMR	SoNaR, train	SoNaR, dev	62.20
XLMR	OntoNotes, train + SoNaR, train	SoNaR, dev	**63.82 (+1.62)**
English performance			
XLMR	OntoNotes, train	OntoNotes, dev	72.90
XLMR	OntoNotes, train + SoNaR, train	OntoNotes, dev	**73.68 (+0.78)**

Table 10: **Comparing monolingual training with multilingual training using the full dataset, excluding singletons** for the SoNaR dataset. The English performance is marginally increased by the multilingual training. The Dutch performance is significantly increased by the multilingual training.

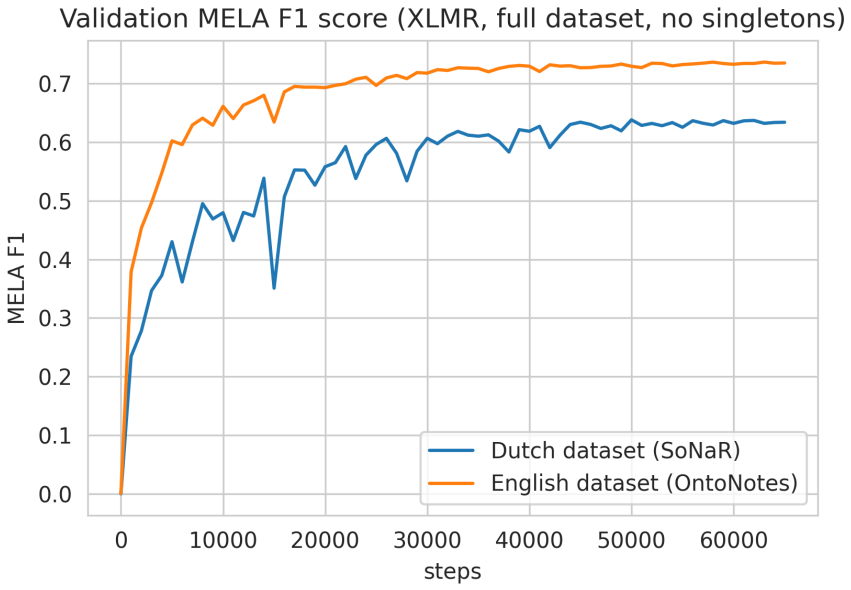


Figure 5: **Convergence of the English (OntoNotes) and Dutch (SoNaR) validation MELA F1 score** during training of XLMR on the full dataset without singletons. Although the final performance on the Dutch and English dataset differs, both curves show a similar convergence.

Figure 6 shows that this technique is able to achieve similar results for the low-resource language compared to jointly training. Warm-up training has two advantages compared to joint training: (i) the generic warm-up step can effectively be reused for different downstream low-resource languages and (ii) the fine-tuning step converges faster because the model is already partially initialized. Regardless of these benefits, we will only consider joint multilingual training in the next step of our work.

6.5 Conclusion

With these experiments we indicated the power of joint multilingual training. We were able to construct one model that achieves results on par with state-of-the-art performance for two languages at once.

Interestingly, when the tasks across the languages are harmonized by excluding singletons from the Dutch data, the joint training procedure was able to surpass the best monolingual performance. We hypothesize this is due to more aligned tasks providing more opportunities for positive cross-lingual transfer. A gain of 1.62% was achieved by training XLMR on both English and Dutch data, resulting in a total gain of 7.61% of using XLMR compared to the Dutch baseline.

Additionally, we provided insights into how to merge high-resource and low-resource datasets, how the jointly trained model converges, and we explore warm-up training as a more efficient alternative compared to joint training.

In the following experiment, we hope to leverage our newfound understanding of joint multilingual training to design systems that respond even better to joint training.

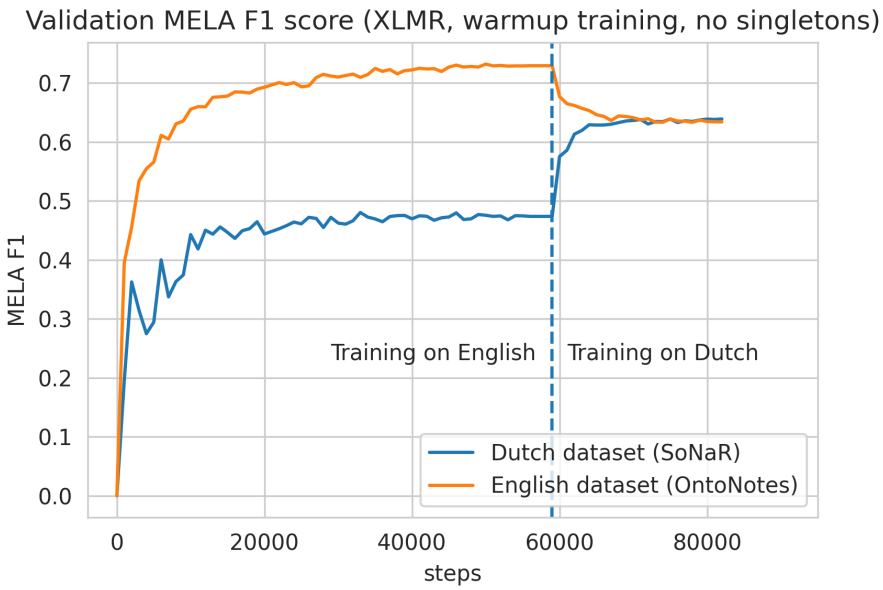


Figure 6: **Convergence of the English (OntoNotes) and Dutch (SoNaR) validation MELA F1 score during warmup training**, excluding singletons for the SoNaR dataset. XLMR is first trained on English data exclusively. Finally, XLMR is fine-tuned on Dutch data. While this warmup setting achieves similar performance for the Dutch language, the resulting English performance is lower.

7 Domain Adversarial Neural Networks for joint multilingual training

This experiment focuses on extending the joint training approach, developed in section 6, to allow a multilingual model to better benefit from joint training for the task of coreference resolution. We explore adding a Domain Adversarial Neural Network to our model (see section 4), which provides us with an additional training objective. This training objective aims to align the embeddings outputted by the Transformer across different languages, resulting in better positive cross-lingual transfer for the downstream task.

In subsection 7.1, we provide a detailed explanation into how we extended our existing model with a domain adversarial neural network. The experiment setup is given in subsection 7.2 and the findings are discussed in subsection 7.3. Finally, a conclusion is formed.

7.1 Models

Figure 7 shows an overview of the proposed architecture. We added a domain adversarial neural network to our architecture, further referred to as the *domain classifier*. Given an embedding, the domain classifier tries to predict whether this embedding originated from an English or a Dutch sentence. This information will be used to remove the discrepancy between English and Dutch embeddings, as to push the model towards language-independent features.

We describe the additional loss function the domain classifier introduces, what optimization objective is used, and how this is implemented.

Loss functions: Given an input sentence, the Transformer encoder outputs the sequence $(x_i)_{i=1}^{i=N}$, where x_i is the produced embedding for the ith input token and N is equal to the total amount of input tokens. Using this output sequence, the domain classifier makes a language prediction y_i per embedding as follows:

$$y_i = ffnn(x_i; \theta) \tag{7}$$

where θ represents the parameters of the domain classifier. Given the sequence of language predictions

$(y_i)_{i=1}^{i=N}$, the loss produced by these predictions is calculated as follows:

$$Loss_{domain} = \sum_{i=1}^{i=N} BCE(sigmoid(y_i), d) \tag{8}$$

where d represents the domain of the input sequence, BCE represents the binary cross-entropy loss and $sigmoid$ represents the sigmoid function.

$Loss_{domain}$ is calculated by making a language prediction on each embedding x_i of the Transformer output sequence. Alternatively, we can calculate a slightly different loss – $Loss_{domain,pooled}$ – based on only one prediction per input sequence. For sentence-level classification tasks, typically only the first produced embedding x_0 is considered. This embedding is passed through a pooling layer, forming the pooled output p. This pooling layer has been pre-trained with the rest of the Transformer to form good sentence-level embeddings. It should be noted that p and x_i have the same dimensions. $Loss_{domain-pooled}$ is calculated as follows:

$$y = ffnn(p; \theta) \tag{9}$$

$$Loss_{domain-pooled} = BCE(sigmoid(y), d) \tag{10}$$

Whether or not we actually use $Loss_{domain}$ or $Loss_{domain-pooled}$ is irrelevant for the remainder of this section.

The loss produced by the coreference head, $Loss_{coreference}$, is not affected by the introduction of the domain classifier and is therefore calculated as described in subsection 3.2.

Optimization objectives: Our goal is to promote the emergence of task-discriminative features that are language-independent. We can achieve this by solving the following optimization problem:

$$\hat{\theta} = argmin_{\theta}\ Loss_{domain}(\theta, \phi) \tag{11}$$

$$\hat{\phi} = argmin_{\phi}\ Loss_{coreference}(\phi, \psi) - Loss_{domain}(\theta, \phi) \tag{12}$$

$$\hat{\psi} = argmin_{\psi}\ Loss_{coreference}(\phi, \psi) \tag{13}$$

Indeed, the equations above give rise to a setting where the domain classifier (θ) minimizes $Loss_{domain}$ and the coreference head (ψ) minimizes $Loss_{coreference}$. Interestingly, the Transformer encoder (ϕ) aims to minimize $Loss_{coreference}$ while maximizing $Loss_{domain}$. Therefore, the transformer encoder is pushed to find a trade-off between task-discriminative features (low $Loss_{coreference}$) and language-independent features (high $Loss_{domain}$).

This contrasting optimization goal between the Transformer encoder and the domain classifier produces an adversarial setting: when the domain classifier learns to discriminate between tokens of both languages, the Transformer encoder is pushed to manipulate its output such that the domain classifier is no longer able to properly discriminate between the tokens.

Implementation: While the optimization objectives above are in essence very simple, they are not straightforward to achieve with standard optimizers often used for deep learning.

Luckily, we can frame the optimization objectives in a more typical setting, allowing us to use conventional optimizers to achieve our goal of task-discriminative language-independent features.

We use a *Gradient Reversal Layer* (GRL) to connect the output of the Transformer encoder with the input of the domain classifier. The GRL acts as an identity operation during the forward pass, whereas it reverses and scales the gradients with a scalar λ during the backward pass. We can describe this behaviour using the pseudo-function $GRL_{\lambda}(x)$ [44]. The forward and backward behaviour is summarized by:

$$GRL_{\lambda}(x) = x \tag{14}$$

$$\frac{\partial GRL_{\lambda}}{\partial x} = -\lambda \mathbf{I} \tag{15}$$

where **I** is the identity matrix.

Now consider the backward pass of the model. The optimizer aims to optimize ϕ (the Transformer parameters), θ (the domain classifier parameters), and ψ (the coreference head parameters) with regard to the loss

$$Loss(\theta, \phi, \psi) = Loss_{coreference}(\phi, \psi) + \alpha Loss_{domain}(\theta, \phi) \tag{16}$$

The hyperparameter α is used to scale $Loss_{domain}$, allowing us to specify how much emphasis we want the optimization procedure to place on the domain classification task.

The new optimization objective becomes:

$$\hat{\theta}, \hat{\phi}, \hat{\psi} = argmin_{\hat{\theta}, \hat{\phi}, \hat{\psi}} \; Loss(\theta, \phi, \psi) \tag{17}$$

We can directly optimize this objective via the use of a popular optimizer such as SGD or Adam.

To verify that the new objective function together with the GRL is equivalent to the previous objective functions, we will consider the produced gradients.

Optimizing the coreference head with regard to $Loss(\theta, \phi, \psi)$ is trivial. Since $Loss_{domain}(\theta, \phi)$ is independent of ψ, the gradients reduce to:

$$\frac{\partial Loss(\theta, \phi, \psi)}{\partial \psi} = \frac{\partial Loss_{coreference}(\phi, \psi)}{\partial \psi} \tag{18}$$

A similar situation arises when optimizing the domain classifier:

$$\frac{\partial Loss(\theta, \phi, \psi)}{\partial \theta} = \alpha \frac{\partial Loss_{domain}(\theta, \phi)}{\partial \theta} \tag{19}$$

When optimizing for the Transformer, we notice the impact of the GRL. The gradients used to optimize the domain classifier are scaled with $-\lambda$ when they are backpropagated from the domain classifier into the Transformer (remember the backward behavior of the GRL, described in equation 15). This results in an update using gradients:

$$\frac{\partial Loss(\theta, \phi, \psi)}{\partial \phi} = \frac{\partial Loss_{coreference}(\phi, \psi)}{\partial \phi} - \lambda \alpha \frac{\partial Loss_{domain}(\theta, \phi)}{\partial \phi} \tag{20}$$

As we can see from equations 19 and 20, the domain classifier is optimized to minimize $Loss_{domain}(\theta, \phi)$ while the Transformer aims to maximize $Loss_{domain}(\theta, \phi)$. This gives rise to the adversarial setting as described by the previous objective functions.

Notice the addition of α and λ, two new parameters. A scenario where α or λ is equal to zero leads to the same situation with regard to ϕ: the Transformer network is not optimized for $Loss_{domain}(\theta, \phi)$. However, these scenarios give rise to a different situation with regard to θ, the domain classifier. When α is equal to zero and λ is not, the domain classifier is not updated. Conversely, when λ is equal to zero and α is not, the domain classifier is updated.

The domain classifier is implemented using one hidden layer consisting of 512 neurons.

7.2 Experiment setup

Before discussing our experiments and results, we take a moment to first illustrate the results we want to achieve.

Training adversarial networks introduces some new difficulties. Sometimes, adversarial networks suffer from collapse. During a collapse, the equilibrium between both competing networks is broken, resulting in worsened performance.

Consider, for example, a possible collapse scenario for the domain classifier. If the domain classifier predicts all input tokens to be English, whether they originate from a Dutch or English sentence, the following situation arises. The BCE loss calculated on English input sentences will be very small, since all predictions will be

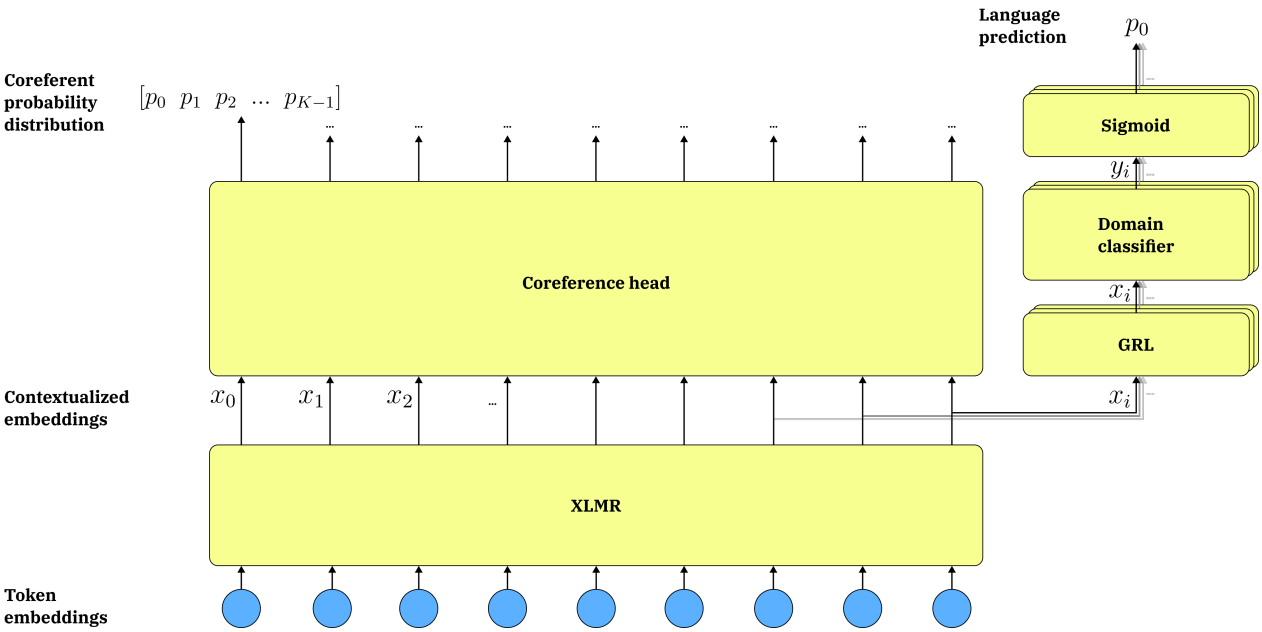


Figure 7: **Proposed architecture with Gradient Reversal Layer.** Given the token embeddings, XLMR produces a sequence of contextualized embeddings. These contextualized embeddings are used by the coreference head to predict the coreference clusters, as described in subsection 3.2. Additionally, the contextualized embeddings are processed in parallel by a Gradient Reversal Layer (GRL) and a Domain classifier, to produce a language prediction.

correct. The BCE loss for the Dutch input sentences will be very large. This leads to a situation where the English embeddings of the Transformer rarely get updated while the Dutch embeddings get updated a lot. Instead of better aligning the Dutch and English embeddings, the Transformer is holding the English embeddings fixed and trying to map the Dutch on the English embeddings. A scenario like this can result in badly aligned embeddings. Additionally, when the domain classifier is able to perfectly discriminate between source and target domain, domain adversarial training suffers from vanishing gradients [48].

In the ideal scenario, the adversarial setting should causes the domain classifier to mislabel an input embedding sometimes, but not always. This indicates that the domain classifier is struggling to consistently predict the language of its input tokens but is not collapsing.

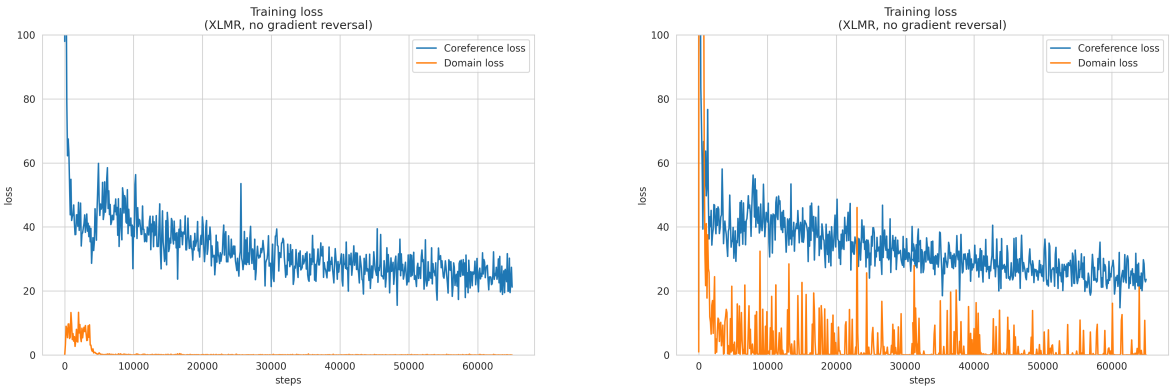
In our first experiments, we focus on how to avoid this adversarial collapse. We investigate whether it is more favorable to apply the domain classifier to the pooled output or the sequence outputs of the Transformer, as described in the previous subsection. The impact of a document-balanced dataset is re-evaluated in the context of the domain classifier. Going one step further, we show how averaging the cross-entropy loss produced by the language predictions (instead of summing) helps combat adversarial collapse.

Having gained a better understanding of adversarial collapse, we further focus on gaining fine-grained control of the adversarial training procedure. We investigate how tuning the α and λ parameters can result in better convergence. Finally, we show the impact of the relative difference between the learning rate of the Transformer and the learning rate of the domain classifier.

7.3 Results

Subsections 7.3.1, 7.3.2, and 7.3.3 focus on mitigating the adversarial collapse and respectively investigate: adding the domain classifier to the pooled output or the sequence outputs, balancing the training data, and balancing the training signal.

Subsections 7.3.4 and 7.3.5 discuss how to gain fine-grained control over the adversarial training and respec-



(a) Pooled outputs

(b) Sequence outputs

Figure 8: **Domain and coreference loss during training** when (a) adding the domain classifier to the pooled outputs and (b) adding the domain classifier to the sequence outputs. No Gradient Reversal Layer is used for this experiment. Subfigure (b) indicates that predicting the languages per sequence output is a more difficult task.

tively describe the impact of tuning the new hyperparameters and how differences in learning rates can steer the adversarial training.

7.3.1 Pooled output vs token outputs

As described above, we can either apply the domain classifier to each embedding in the sequence output of the transformer in parallel or once to a pooled representation of the sequence outputs.

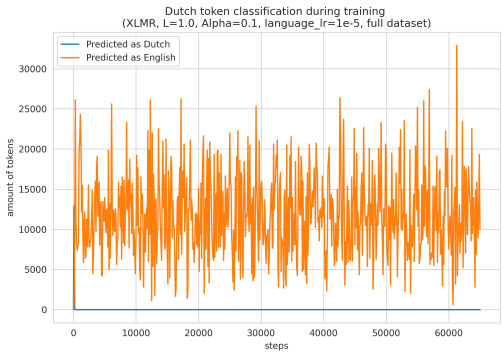
We experiment with both settings and find that applying the domain classifier to each output embedding and summing the individual losses provides better results. Figure 8 shows the evolution of $Loss_{coreference}$ and $Loss_{domain}$ during training when applying the domain classifier to (a) the pooled output and (b) the sequence outputs. In this experiment, we did not include a GRL layer, the entire model is tasked with minimizing $Loss_{domain}$. This allows us to study the difference between using sequence outputs and pooled outputs without the added complexity of a gradient reversal layer. We notice that the loss of the domain classifier quickly approaches zero when applied to the pooled output, while this does not happen when the domain classfifier is applied to the sequence outputs. This is not surprising: given a good representation of a sentence, it is relatively easy to distinguish between input languages. However, this problem becomes harder when considered at the token-level: some tokens are similar or even shared across languages, making a confident prediction much more difficult for the domain classifier. We prefer this more difficult task: the easy task gets solved by the domain classifier before the transformer can align the pooled outputs, thus resulting in a collapse of the adversarial system.

Therefore, we will use $Loss_{domain}$ as described in equation 8 for the next experiments.

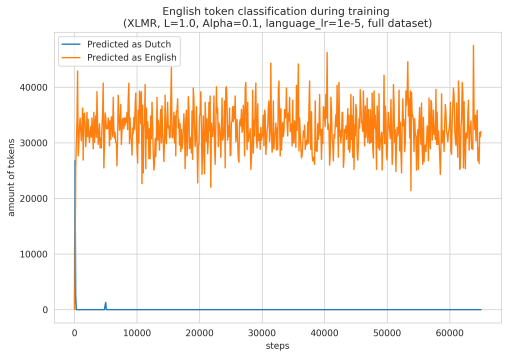
Figure 9 shows the language predictions made for the Dutch and English tokens seen during training. Even though applying the domain classifier to the sequence outputs provides a better training loss, we still suffer from adversarial collapse. The domain classifier defaults to predicting all tokens to be of English origin. This collapse already happens during the first few training steps of the model.

7.3.2 Balancing the training data

As a next method to combat adversarial collapse, we investigate balancing the training data. We already investigated balancing the training data in the case of joint multilingual learning (section 6). We found that, when using the fully merged dataset – containing roughly 7 times more English batches than Dutch batches,

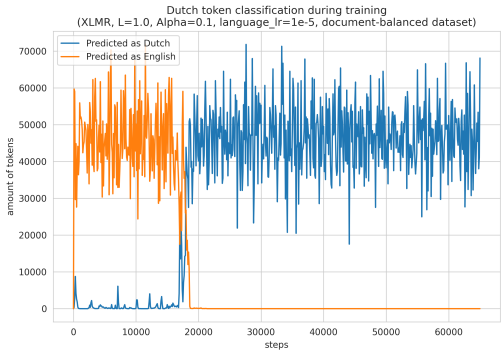


(a) Predictions for Dutch tokens

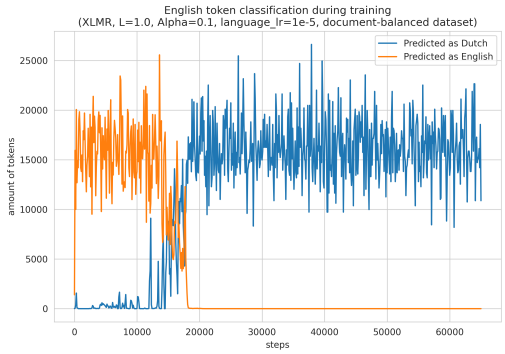


(b) Predictions for English tokens

Figure 9: **Domain classifier output when training on unbalanced data** for (a) Dutch tokens and (b) English tokens. The domain classifier immediately suffers from collapse: all tokens are predicted as being of English origin.



(a) Predictions for Dutch tokens

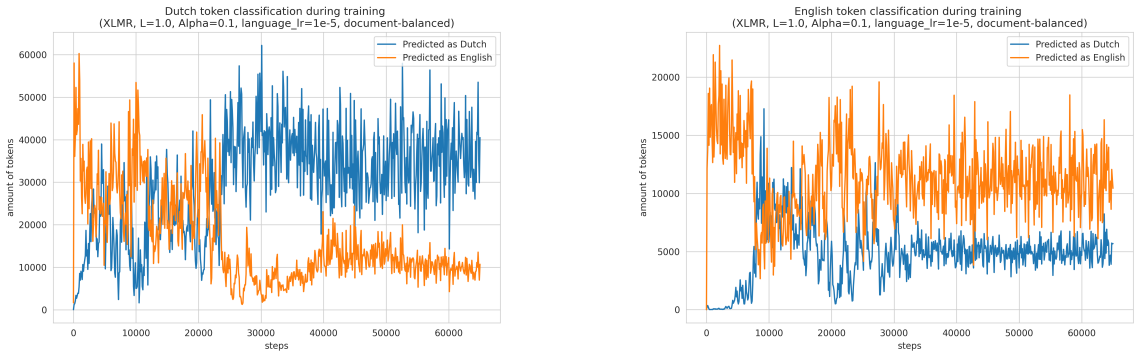


(b) Predictions for English tokens

Figure 10: **Domain classifier output when training on balanced data** for (a) Dutch tokens and (b) English tokens. The domain classifier is suffering from collapse: at a given point in the training procedure, the domain classifier only assigns one label to all tokens. Between 10,000 and 20,000 steps, the domain classifier was able to overcome this collapse for a brief moment but ended up in an inverted collapse.

the Transformer was robust against this training imbalance. However, the same might not hold for the domain classifier.

We consider training the new model again with the document-balanced dataset. Now the domain classifier is updated an equal amount of times for Dutch sentences and English sentences. Figure 9 shows the predictions made by the domain classifier for the (a) Dutch tokens and (b) English tokens during training on unbalanced data. Figure 10 shows the same metrics when training on balanced data. Both experiments result in a quasi immediate adversarial collapse. The model is always predicting one language, regardless of the actual input. To confirm this was not due to an unfortunate initialization of the domain classifier, we ran several experiments with different random initialization, all giving rise to adversarial collapse. Interestingly, we notice the situation for the balanced dataset is marginally better: the model at one point manages to break the adversarial collapse, only to end up in an inversed collapse. While these results are slightly better, we are not yet in the ideal regime described above.



(a) Predictions for Dutch tokens

(b) Predictions for English tokens

Figure 11: **Domain classifier output when training on balanced data and balancing the training signal per batch** for (a) Dutch tokens and (b) English tokens. The domain classifier does not collapse, during the entire training process the domain classifier is struggling to consistently predict the origin of a token. After 25,000 steps of the training procedure, an equilibrium is reached where the domain classifier manages to correctly predict the majority of the language labels.

7.3.3 Balancing the training signal

We further investigate how to achieve a stable, non-collapsing adversarial training setup. Previously, we described how balancing the training data – such that the model sees an equal amount of Dutch and English batches – gives slightly better results. When further investigating possible causes for this collapse, we notice that the loss produces by each batch is not equal in magnitude.

As described in equation 8, we apply the domain classifier in parallel to all the sequence outputs and sum the resulting losses. However, not all sequences are equally long, causing this sum to vary in size. Additionally, the mean sequence length per batch is much larger for the Dutch corpus, resulting in a loss produced by the Dutch batches that is consistently larger compared to that of the English batches.

We solve this issue by averaging the individual losses produced by the domain classifier on the sequence embeddings, instead of summing them:

$$Loss_{domain-averaged} = \frac{1}{N} \sum_{i=1}^{i=N} BCE(sigmoid(y_i), d) \tag{21}$$

This results in a loss per batch which is invariant with regard to the sequence length per batch. The order of magnitude of the loss is now consistent over the English and Dutch batches.

Figure 11 shows the language predictions made when training the domain classifier with $Loss_{domain-averaged}$. Note that we are also training on the document-balanced dataset: the amount of batches is balanced and each batch now produces a loss of similar magnitude. We can see that adversarial collapse is avoided. Interestingly, the adversarial training reaches a balance in the second half of the training procedure, with a consistent number of tokens misclassified each step.

Alternatively, instead of balancing the amount of batches across the language and averaging the loss per batch, we could also use the full dataset and multiply the loss for the minority batches with a ratio to achieve an overall balanced training signal. This would allow us to use more English data, since we now omitted several English documents to form a document-balanced dataset. We tried this, but were unable to produce a setting that did not result in adversarial collapse[5].

[5]This is most likely due to a quirk of the optimization procedure used. Before each update to the model parameters, the gradients get clipped if their global norm exceeds a certain threshold, as described in [49]. Assume the majority class has k times more batches compared to the minority class. Scaling the loss produced by the minority batches with a factor k does not necessarily balance the

Based on these insights, we only consider applying the domain classifier to the sequence embeddings, averaging the loss of the domain classifier per batch, and training on a document-balanced dataset in the following experiments.

7.3.4 Scheduling α and λ

In the above sections, we described how we were able to achieve non-collapsing adversarial training. In the subsequent sections, we shift our focus towards gaining more fine-grained control over the adversarial training procedure.

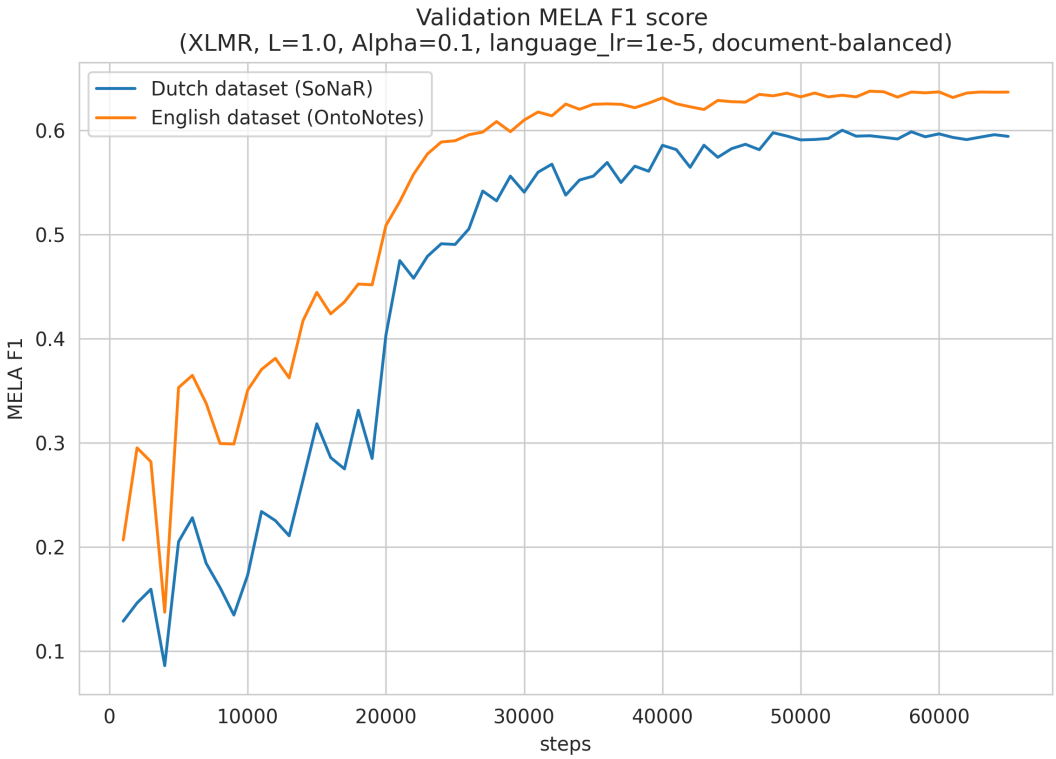
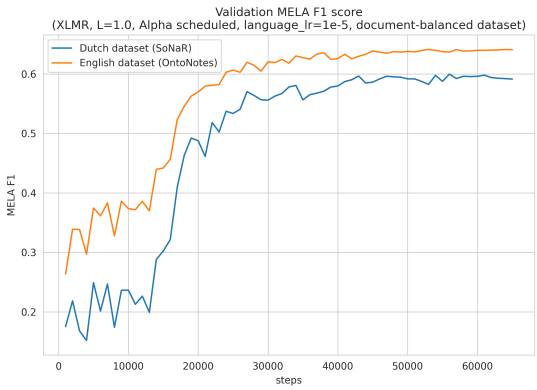


Figure 12: **MELA F1 validation score** when training XLMR on the document-balanced dataset and balancing the domain classifier training signal per batch. The training procedure suffers from bad convergence during the first 20,000 steps.

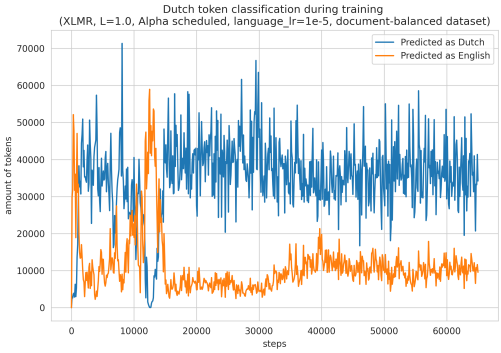
From the evaluation curves of the first result without collapse (figure 12), we notice a worse convergence for the task of coreference resolution compared to not including a domain classifier. This results in a lower final performance compared to our best multilingual model. This is not surprising, since the model is now being optimized for an additional task. Our first goal in fine-tuning the adversarial training procedure will focus on improving this convergence.

We consider scheduling both the α and λ parameters to achieve better convergence. Remember subsection 7.1: α determines how much weight in general is placed upon the domain classification task (by scaling the

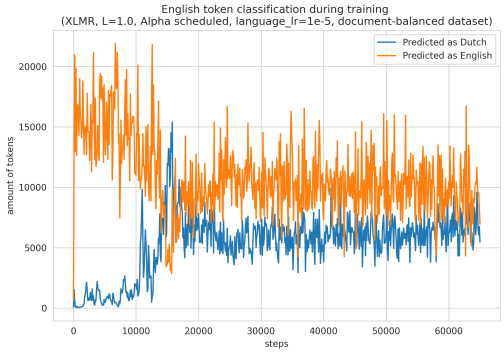
overall training procedure. Assume the loss on each batch, before scaling, has value n. Additionally, assume one minority batch and k majority batches have been seen by the model. The model gets updated once in favor of the minority batch, with a scaled loss of $k \cdot n$. The model gets updated k times in favor of the majority batches, with a loss of n per update. If the norm of the gradients produces by a loss of value $k \cdot n$ exceed the clipping threshold, but the norm of the gradients produces by a loss of value n does not, scaling the minority loss with a factor k will not result in an overall balanced training signal. While this explanation is highly simplified and does not cover all possible scenarios, it serves to highlight the difficulties with balancing the training loss when the underlying amount of batches are not balanced.



(a) MELA F1 validation score



(b) Predictions for Dutch tokens



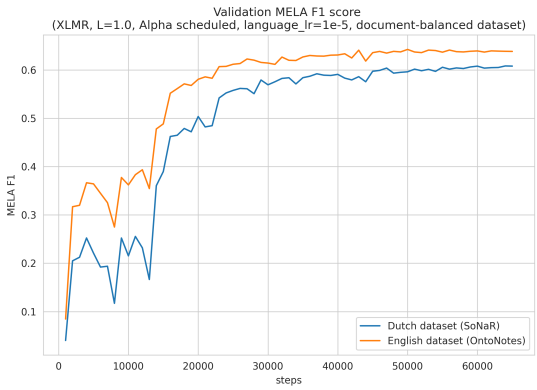
(c) Predictions for Dutch tokens

Figure 13: **Scheduling lambda during training**. Subfigure (a) shows the validation MELA F1 score during training, subfigures (b) and (c) show the language predictions made by the domain classifier for Dutch and English tokens respectively. Scheduling lambda did not improve convergence.

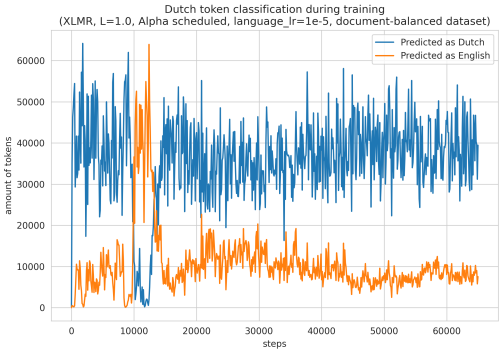
domain classification loss, see equation 16) and λ determines how much the Transformer gets updated with regard to the domain classification task (by scaling the gradients backpropagated from the domain classifier into the Transformer, see equation 20). Because low values of these parameters reduce the effect of the classification task on the Transformer, they will allow the Transformer to focus more on the coreference task, thus improving the convergence. Once the Transformer has (partially) converged, we can increase the value of α or λ to gradually place more weight on the domain classification task.

Figure 13 shows the results obtained when α is fixed at 1 and λ is scheduled. Similarly, figure 14 was obtained by fixing λ to 1 and scheduling α. In both experiments the variables are scheduled in the same manner: the variable is initialized at 0 and is linearly increased to 1 over 45,000 steps, after which it remains 1 for the rest of the experiment. Counterintuitively, neither experiments led to better convergence.

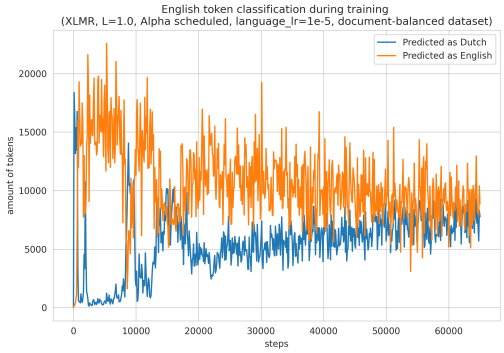
Before running additional experiments, we explain why we think scheduling α is more favorable compared to scheduling λ. When investigating figure 13 and figure 14, we notice no real difference between scheduling α and λ. When gradually increasing α, we are gradually placing more emphasis on the domain classification task. When gradually increasing λ, we are always optimizing the domain classifier and only gradually placing more emphasis on aligning the embeddings in the Transformer. Take the extreme situation where we hold λ to be 0 for an amount of initial steps before discontinuously increasing λ to 1. During the time that λ was 0, the domain classifier will have learned how to distinguish between the languages (since the Transformer is not yet introducing confusion). When λ gets increased to 1, the Transformer will first focus on manipulating the features most used by the domain classifier, since this is the optimal way for the Transformer to maximize



(a) MELA F1 validation score



(b) Predictions for Dutch tokens



(c) Predictions for Dutch tokens

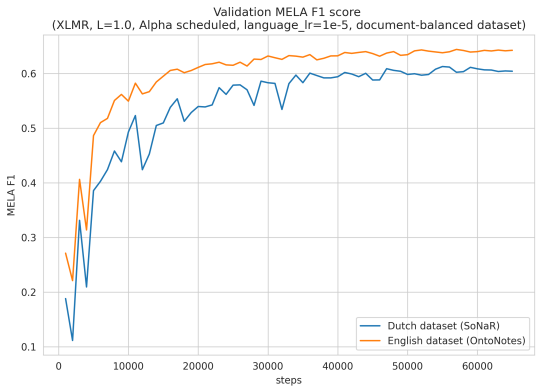
Figure 14: **Scheduling alpha during training**. Subfigure (a) shows the validation MELA F1 score during training, subfigures (b) and (c) show the language predictions made by the domain classifier for Dutch and English tokens respectively. Scheduling alpha did not improve convergence.

$Loss_{domain}$. Once these features are manipulated, the domain classifier needs to find new discriminative features and the Transformer is focused at removing these features. Thus, after the Transformer has quickly removed the features the domain classifier learned to focus on when λ was equal to zero, we arrive in the exact same setting compared to when we schedule α. Therefore, we argue that scheduling both α and λ is redundant and further only focus on studying the effect of α on the adversarial training.

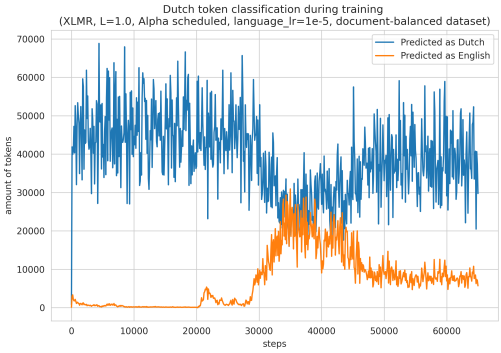
In a subsequent experiment (shown in figure 15), we schedule α more aggressively: the variable is fixed at 0 for the first 20,000 steps, before being linearly increased to 1 over the next 25,000 steps, after which the value remains at 1. This experiment is shown in figure 15. This new approach succeeds in improving the model convergence.

7.3.5 Investigating the impact of the domain classifier learning rate

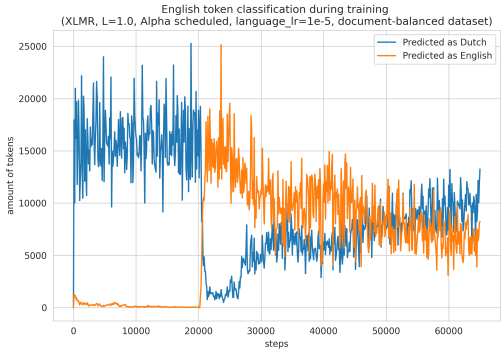
In our previous experiments, adversarial training resulted in a balance. The learning rates of the Transformer and domain classifier can be used to control the characteristics of this balance. Indeed, if the learning rate of the domain classifier is much higher, it should be able to continuously *outsmart* the Transformer – by finding newer and more complex methods to distinguish the languages of the embeddings before the Transformer is able to further align them. A similar situation arises when the domain classifier has a lower learning rate, shifting the balance of the adversarial training in favor of the Transformer.



(a) MELA F1 validation score



(b) Predictions for Dutch tokens



(c) Predictions for Dutch tokens

Figure 15: **Scheduling alpha (2nd attempt) during training**. Subfigure (a) shows the validation MELA F1 score during training, subfigures (b) and (c) show the language predictions made by the domain classifier for Dutch and English tokens respectively. Scheduling alpha for the second time did improve the convergence of the model. The final validation performance did not improve however. We can clearly see the impact of scheduling alpha when looking at the domain classifier output.

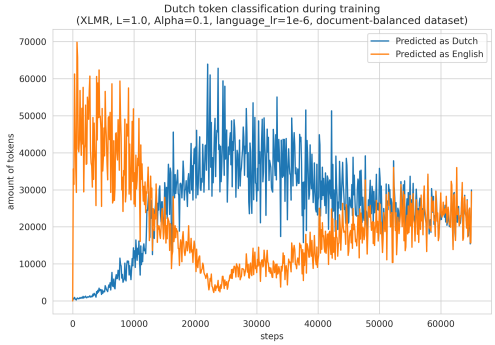
.

We consider three experiments with three different initial learning rates[6] for the domain classifier: $1\mathrm{e}{-6}$, $1\mathrm{e}{-5}$, and $2\mathrm{e}{-4}$. These learning rates are respectively: 10 times smaller than that of the Transformer, equal to that of the Transformer, and 20 timers higher than that of the Transformer (which is equal to the learning rate of the coreference head architecture). In these experiments, we fix α at 0.1 and λ at 1. We do not consider scheduling α during these experiments, such that we can directly study the effect of the different learning rates.

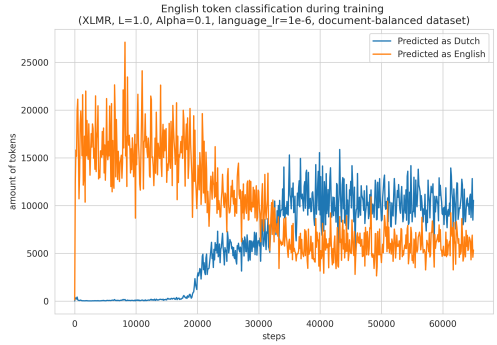
Figure 16 shows the result of these three experiments. We are successful in shifting around the balance of the adversarial learning by varying the learning rates of the domain classifier. For the smallest learning rate, the domain classifier is the most confused. For the highest learning rate, the domain classifier is able to make accurate predictions.

We propose running subsequent experiments with a learning rate for the domain classifier equal to $2\mathrm{e}{-4}$, our highest learning rate from the previous experiment. The reasoning is as follows. The gradient reversal layer pushes the Transformer to remove the features that are most useful for the domain classifier to predict the language. If the domain classifier is lagging behind the Transformer in terms of learning rate, we cannot be sure that the features on which the domain classifier focuses are actually the best features for the language

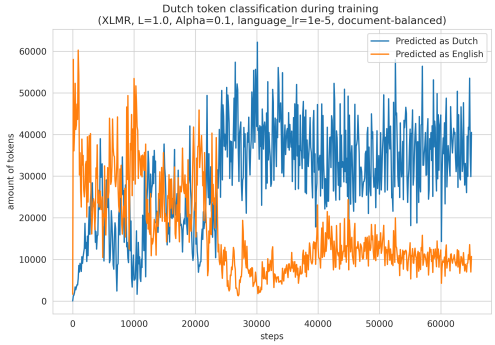
[6] The learning rates are exponentially decayed during training, as is custom when fine-tuning the model.



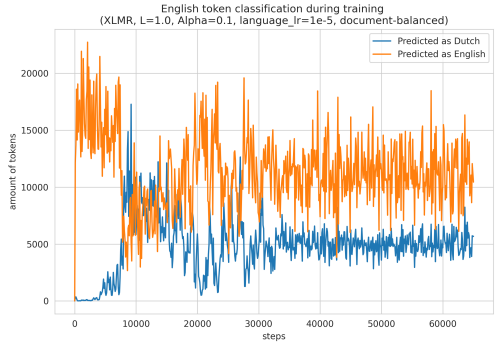
(a) Predictions for Dutch tokens, lr = $1e{-}6$



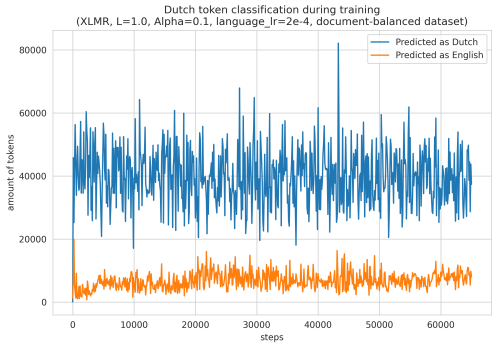
(b) Predictions for English tokens, lr = $1e{-}6$



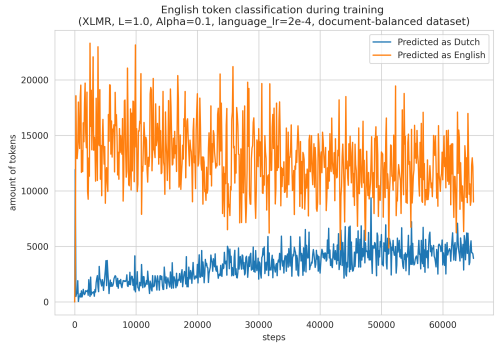
(c) Predictions for Dutch tokens, lr = $1e{-}5$



(d) Predictions for English tokens, lr = $1e{-}5$



(e) Predictions for Dutch tokens, lr = $2e{-}4$



(f) Predictions for English tokens, lr = $2e{-}4$

Figure 16: **Comparing the domain classifier outputs for different learning rates**. In the left column the Dutch predictions are shown, the right column shows the English predictions. The rows correspond respectively to a learning rate of the domain classifier equal to $1e{-}6$, $1e{-}5$, and $2e{-}4$. Once the domain classifier reaches an equilibrium position, the learning rate determines the fraction of wrongly classified tokens. Higher learning rates lead to fewer wrong predictions.

discrimination task. If the domain classifier has a higher learning rate and is *winning* with regard to the Transformer, we are sure that the training signal produced by the adversarial setup is helping to remove features from the Transformer output which are language-dependent, pushing the Transformer to output language-independent token embeddings. A much higher learning rate than $2\mathrm{e}{-4}$ is not favorable, since this might lead to adversarial collapse.

Figure 17 shows the evaluation accuracy of our best performing experiment. In this experiment we applied the domain classifier to the sequence outputs, using a document-balanced dataset, and optimized for $Loss_{domain-averaged}$, as described in subsections 7.3.1, 7.3.2, and 7.3.3 respectively. A learning rate equal to $2\mathrm{e}{-4}$ is used for the domain classifier. With this learning rate, the model converges nicely without scheduling the α or λ variables: we take λ to be 1 and α to be 0.1 during the entire training procedure.

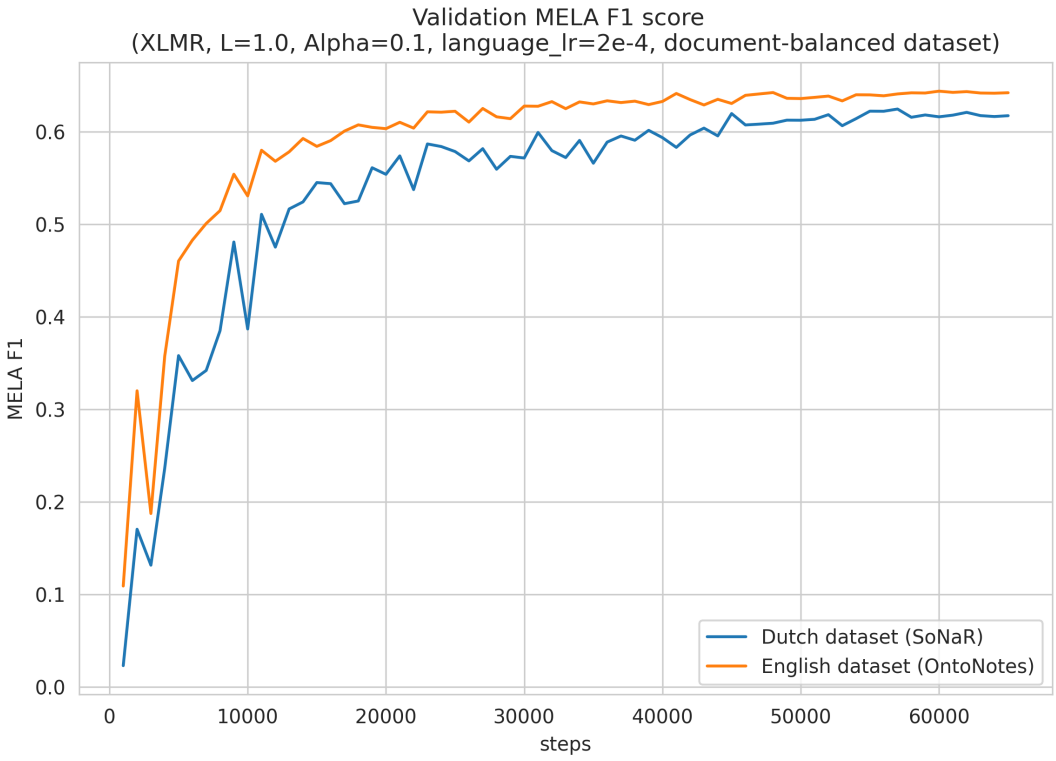


Figure 17: **Validation MELA F1 score** when training XLMR on the balanced dataset and balancing the training signal per batch. No alpha or lambda scheduling is used. The domain classifier has a learning rate equal to $2\mathrm{e}{-4}$. The model converges nicely and performs best of all models which use the adversarial domain classifier.

7.4 Conclusion

In this section we explored the use of domain adversarial neural networks to further increase the benefit of joint multilingual training for coreference resolution.

We showed how adversarial collapse can be avoided by balancing the training signal for the domain classifier. Additionally, we showed fine-grained control over the training procedure by manipulating the new α and λ hyperparameters, as well as the learning rate of the domain classifier.

Table 11 shows a comparison between the best result using a domain classifier and the best multilingual results, achieved in section 6. The addition of the domain classifier was unable to achieve a performance

Model	Trained on	Evaluated on	MELA F1
Best multilingual performance, full-dataset			
XLMR	OntoNotes, train + SoNaR, train	OntoNotes, dev	**73.68**
XLMR	OntoNotes, train + SoNaR, train	SoNaR, dev	**63.82**
Best multilingual performance, document-balanced dataset			
XLMR	OntoNotes, train + SoNaR, train	OntoNotes, dev	64.27
XLMR	OntoNotes, train + SoNaR, train	SoNaR, dev	62.56
Best domain adaptation performance, document-balanced dataset			
XLMR + domain classifier	OntoNotes, train + SoNaR, train	OntoNotes, dev	64.36
XLMR + domain classifier	OntoNotes, train + SoNaR, train	SoNaR, dev	62.43

Table 11: **Comparing best multilingual results with and without domain adaptation.** Comparing the best multilingual result with and without domain adaptation on the document-balanced dataset, we conclude that the domain adaptation technique has no added value. Therefore, the best technique thus far consists of multilingual training on the full dataset.

gain over the previously attained results. Additionally, since the domain adaptation technique requires a document-balanced dataset without producing any gain in performance, the resulting model performs significantly worse compared to multilingual training on the full dataset.

Initially, we explored the use of domain adaptation for multilingual coreference resolution under the assumption that minimizing the discrepancy between the embeddings of different languages will allow for more performance gain when jointly training on multiple languages. However, the domain adaptation technique did not succeed in increasing the performance. One possible explanation is that the XLMR outputs over different languages already have minimal discrepancy. Indeed, XLMR is pre-trained using a Translation Language Modeling (TLM) objective, specifically aimed at minimizing the discrepancy between parallel inputs (see section 3.3). This explains the strong zero-shot cross-lingual transfer observed. The additional signal produced by the adversarial setting, aimed at further minimizing the language discrepancy, could be negligible. If XLMR's TLM objective already reaches a very good alignment over languages, the domain adversarial technique will not have any additional effect. More research is needed to confirm this.

8 Conclusions

In this work we designed and analyzed a domain adaptation system for joint multilingual coreference resolution.

We started by building a neural baseline using a monolingual Dutch Transformer for the task of Dutch coreference resolution. We found that a multilingual Transformer trained on Dutch coreference resolution consistently outperformed the baseline, achieving an increase of 5.99% on the MELA F1 score. Additionally, the multilingual Transformer showed a large increase in zero-shot cross-lingual performance between both languages for the task of coreference resolution.

Building upon these promising results, we designed a multilingual learning approach that achieves a further increasing in performance on Dutch coreference resolution by incorporating English data during training. Multilingual learning achieved an additional 1.62% increase compared to our previous best result. This results in a total increase of 7.61% compared to the original Dutch baseline.

We further study the applicability of a domain adversarial neural network to further aid the joint multilingual training. We give insights into how to avoid adversarial collapse and demonstrate fine-tuned control over the adversarial training procedure. Unfortunately, the addition of the domain adversarial neural network was unable to further increase the performance for Dutch coreference resolution.

9 Future work

Since research is rarely ever finished, we highlight several options for future work. Our ideas are divided in two subsections. Subsection 9.1 addresses possible direct next steps to further develop the technique of domain adaptation for joint multilingual coreference resolution. In subsection 9.2, we discuss more indirect options for further work concerning multilingual models and low-resource languages.

9.1 Direct future work

9.1.1 Include additional languages for multilingual training

A first step to achieve better performance with relative ease would be to include additional coreference resolution datasets over different languages, both high- and low-resource. Hopefully, the multilingual model is able to leverage this increased amount of data over multiple languages to achieve a better transfer to low-resource languages. Additionally, this increased richness in data might make the model more robust against differences in the annotation schemes used for coreference resolution.

To build a proof of concept for this approach, we could vary the amount of English data used in our multilingual experiments. This would help us estimate how much the performance is impacted by the amount of high-resource data and if there is reason to believe multilingual training will benefit from more source data.

To achieve this, a study needs to be conducted to group (most of) all available coreference resolution datasets. Additionally, it is important to fully understand the differences between these datasets with regard to the task definition and annotation scheme employed. With a high probability, some harmonization effort will need to be conducted over the different languages to allow a single architecture to train on all these datasets. Based on the insights of this study, the option exists to annotate additional coreference resolution datasets, focusing on languages or language groups where the study found resources to be most lacking.

9.1.2 Unsupervised domain adaptation

Thus far we have only considered supervised domain adaptation. When using a domain adaptation setup to jointly train English and Dutch coreference resolution, we used annotated coreference resolution data for both languages.

However, annotated coreference resolution data might not always be available for a low-resource language. Therefore, we could consider an unsupervised domain adaptation setup. In this setting, the coreference head is only optimized for the high-resource language. Unsupervised samples from the low-resource language are used by the domain adaptation network to minimize the discrepancy between the Transformer output for the high- and low-resource languages.

In subsection 5.3.3 we described the impressive zero-shot cross-lingual transfer capability of XLMR. Unsupervised domain adaptation would be the natural extension of this finding: instead of solely training on labeled high-resource data we can now train on a mix of labeled high-resource and unlabeled low-resource data.

This would be especially interesting when considering a low-resource language on which XLMR is not pre-trained. Indeed, the strong English-to-Dutch transfer observed is due to XLMR being pre-trained on both these languages and already outputting similar representations for these languages. This might not be the case for low-resource languages on which XLMR has not been pre-trained. Thus, the added effect of the domain adaptation system, which helps XLMR align its outputs over languages, could be relatively greater in this setting.

We did some initial exploration concerning unsupervised domain adaptation. However, our initial experiments were suffering from adversarial collapse. Due to time constraints, we were not able to run subsequent experiments aimed at combatting this collapse for unsupervised domain adaptation.

9.1.3 Additional tuning

While we did not manage to gain better performance using a domain adversarial neural network, this could in part be due to a worsened model convergence. For example, we could explore the effect of training our proposed architecture longer with slower learning rate schedulers. However, tuning a model this big becomes really expensive and time consuming. The cost effectiveness of tuning our model is much lower compared to, for example, the cost effectiveness of incorporating other languages with readily available annotated data into the training procedure.

9.2 Indirect future work

9.2.1 Multilingual SpanBERT

As mentioned in section 3.2, the state-of-the-art English architectures for coreference resolution use SpanBERT as Transformer encoder. To achieve good coreference resolution results, it is vital to have access to good span representations. SpanBERT is specifically pre-trained to produce high-quality span embeddings, which translates into state-of-the-art coreference resolution performance.

Replicating SpanBERT's success on low-resource languages is difficult. Pre-training a model like SpanBERT for a low-resource language is quite expensive and does not scale, considering the amount of low-resource languages.

Having shown that XLMR is able to achieve competitive results for a high- and low-resource language, it can be cost-effective to train a version of XLMR with span embeddings in mind. This model – SpanXLMR – can be pre-trained via a combination of the Span Boundary Objective introduced by [37] and the Translation Language Model Objective introduces by [5]. SpanXLMR could then be used to increase the state-of-the-art for coreference resolution on many low-resource languages, using the multilingual learning approach described in this work.

9.2.2 Cross-lingual model distillation

As an alternative to training a SpanXLMR-like model, cross-lingual model distillation might prove useful. Hinton et al. introduced knowledge distillation [50] to compress the knowledge learned by a larger model (or an ensemble of models) during training in a smaller model, which leads to more efficient inference. This is achieved by jointly training the smaller model on two objectives.

Consider a model trained for a classification task. The first objective of the smaller model is to optimize the cross-entropy loss of its predictions on the training dataset – the standard objective function used to train classifiers. As the second objective, the model is optimized to output softmax logits which resemble the softmax logits of the larger model for the same inputs. These softmax logits are considered at a specific temperature, which is a hyperparameter in the distillation process. This additional objective, called the distillation loss, allows the smaller model to efficiently learn the knowledge the larger model developed during training.

A cross-lingual model distillation setup could be considered, where a multilingual model is trained with two objectives functions: one directly optimizing the coreference resolution predictions and one distilling the knowledge between SpanBERT and the multilingual model for English training samples. The aim would be that the additional training signal pushing the multilingual model towards better predictions on the English language and that this transfers into better performance on low-resource languages.

9.2.3 Task-adaptation

We found our multilingual approach to perform better when the datasets were consolidated over the different languages. However, it may not always be favorable to reduce each dataset to its *lowest common denominator* version.

Therefore, a multilingual coreference resolution system should be developed that can handle task and annotation differences over the different languages, while still allowing positive transfer between languages.

One possible way to tackle this is via the field of Multi-Task Learning (MTL) [51]. MLT allows models to build a shared representation for related task, to allow for task-specific architectures while still generalizing between tasks.

To deal with the singleton problem concerning coreference resolution, we could build an additional model with a modified architecture that allows the prediction of singletons. To still allow the core of our model – the Transformer – to benefit from multilingual training, we can softly share the weights between the cores of these two models.

References

[1] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[2] Thomas Wolf et al. "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[3] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[4] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[5] Alexis Conneau et al. "Unsupervised cross-lingual representation learning at scale". In: *arXiv preprint arXiv:1911.02116* (2019).

[6] Alan Ramponi and Barbara Plank. "Neural Unsupervised Domain Adaptation in NLP—A Survey". In: *arXiv preprint arXiv:2006.00632* (2020).

[7] Ralph Grishman. "Information extraction". In: *IEEE Intelligent Systems* 30.5 (2015), pp. 8–15.

[8] Véronique Hoste. "Optimization issues in machine learning of coreference resolution". PhD thesis. Universiteit Antwerpen. Faculteit Letteren en Wijsbegeerte., 2005.

[9] Sameer Pradhan et al. "CoNLL-2012 shared task: Modeling multilingual unrestricted coreference in OntoNotes". In: *Joint Conference on EMNLP and CoNLL-Shared Task*. 2012, pp. 1–40.

[10] Lynette Hirschman. "The evolution of evaluation: Lessons from the message understanding conferences". In: *Computer Speech & Language* 12.4 (1998), pp. 281–305.

[11] Eduard Hovy et al. "OntoNotes: the 90% solution". In: *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*. 2006, pp. 57–60.

[12] Ralph Weischedel et al. "Ontonotes release 4.0". In: *LDC2011T03, Philadelphia, Penn.: Linguistic Data Consortium* (2011).

[13] Sameer Pradhan et al. "Conll-2011 shared task: Modeling unrestricted coreference in ontonotes". In: *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*. 2011, pp. 1–27.

[14] Sameer Pradhan et al. "Scoring coreference partitions of predicted mentions: A reference implementation". In: *Proceedings of the conference. Association for Computational Linguistics. Meeting*. Vol. 2014. NIH Public Access. 2014, p. 30.

[15] Ineke Schuurman, Véronique Hoste, and Paola Monachesi. "Interacting semantic layers of annotation in SoNaR, a reference corpus of contemporary written Dutch". In: *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC)*. ELRA; Malta. 2010, pp. 2471–2477.

[16] Véronique Hoste and Walter Daelemans. "Learning Dutch coreference resolution". In: *LOT Occasional Series* 4 (2005), pp. 133–148.

[17] Gosse Bouma et al. "The COREA-project, manual for the annotation of coreference in Dutch texts". In: *University Groningen* (2007).

[18] Corbèn Poot and Andreas van Cranenburgh. "A Benchmark of Rule-Based and Neural Coreference Resolution in Dutch Novels and News". In: *arXiv preprint arXiv:2011.01615* (2020).

[19] Veselin Stoyanov et al. "Conundrums in noun phrase coreference resolution: Making sense of the state-of-the-art". In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. 2009, pp. 656–664.

[20] Marc Vilain et al. "A model-theoretic coreference scoring scheme". In: *Sixth Message Understanding Conference (MUC-6): Proceedings of a Conference Held in Columbia, Maryland, November 6-8, 1995*. 1995.

[21] Amit Bagga and Breck Baldwin. "Algorithms for scoring coreference chains". In: *The first international conference on language resources and evaluation workshop on linguistics coreference*. Vol. 1. Citeseer. 1998, pp. 563–566.

[22] Xiaoqiang Luo. “On coreference resolution performance metrics”. In: *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*. 2005, pp. 25–32.
[23] Marta Recasens and Eduard Hovy. “BLANC: Implementing the Rand index for coreference evaluation”. In: *Natural Language Engineering* 17.4 (2011), p. 485.
[24] Pascal Denis and Jason Baldridge. “Global joint models for coreference resolution and named entity classification”. In: *Procesamiento del lenguaje natural* 42 (2009).
[25] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).
[26] Alec Radford et al. “Improving language understanding by generative pre-training”. In: (2018).
[27] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Neural machine translation of rare words with subword units”. In: *arXiv preprint arXiv:1508.07909* (2015).
[28] Mike Schuster and Kaisuke Nakajima. “Japanese and korean voice search”. In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2012, pp. 5149–5152.
[29] Taku Kudo and John Richardson. “Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing”. In: *arXiv preprint arXiv:1808.06226* (2018).
[30] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: *arXiv preprint arXiv:1607.06450* (2016).
[31] Ian Tenney, Dipanjan Das, and Ellie Pavlick. “BERT rediscovers the classical NLP pipeline”. In: *arXiv preprint arXiv:1905.05950* (2019).
[32] Heeyoung Lee et al. “Stanford’s multi-pass sieve coreference resolution system at the conll-2011 shared task”. In: *Proceedings of the 15th conference on computational natural language learning: Shared task*. Association for Computational Linguistics. 2011, pp. 28–34.
[33] Kenton Lee et al. “End-to-end neural coreference resolution”. In: *arXiv preprint arXiv:1707.07045* (2017).
[34] Mike Schuster and Kuldip K Paliwal. “Bidirectional recurrent neural networks”. In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.
[35] Mandar Joshi et al. “BERT for coreference resolution: Baselines and analysis”. In: *arXiv preprint arXiv:1908.09091* (2019).
[36] Ben Kantor and Amir Globerson. “Coreference resolution with entity equalization”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 673–677.
[37] Mandar Joshi et al. “Spanbert: Improving pre-training by representing and predicting spans”. In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 64–77.
[38] Wietse de Vries et al. “Bertje: A dutch bert model”. In: *arXiv preprint arXiv:1912.09582* (2019).
[39] Pieter Delobelle, Thomas Winters, and Bettina Berendt. “RobBERT: a dutch RoBERTa-based language model”. In: *arXiv preprint arXiv:2001.06286* (2020).
[40] Louis Martin et al. “Camembert: a tasty french language model”. In: *arXiv preprint arXiv:1911.03894* (2019).
[41] Debora Nozza, Federico Bianchi, and Dirk Hovy. “What the [mask]? making sense of language-specific BERT models”. In: *arXiv preprint arXiv:2003.02912* (2020).
[42] Guillaume Lample and Alexis Conneau. “Cross-lingual language model pretraining”. In: *arXiv preprint arXiv:1901.07291* (2019).
[43] Sebastian Ruder. “Neural transfer learning for natural language processing”. PhD thesis. NUI Galway, 2019.
[44] Yaroslav Ganin and Victor Lempitsky. “Unsupervised domain adaptation by backpropagation”. In: *International conference on machine learning*. PMLR. 2015, pp. 1180–1189.
[45] Ian J Goodfellow et al. “Generative adversarial networks”. In: *arXiv preprint arXiv:1406.2661* (2014).
[46] Andreas van Cranenburgh. “A Dutch coreference resolution system with an evaluation on literary fiction”. In: *Computational Linguistics in the Netherlands Journal* 9 (2019), pp. 27–54.
[47] Kenton Lee, Luheng He, and Luke Zettlemoyer. “Higher-order coreference resolution with coarse-to-fine inference”. In: *arXiv preprint arXiv:1804.05392* (2018).
[48] Jian Shen et al. “Wasserstein distance guided representation learning for domain adaptation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.
[49] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks”. In: *International conference on machine learning*. PMLR. 2013, pp. 1310–1318.

[50] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the knowledge in a neural network”. In: *arXiv preprint arXiv:1503.02531* (2015).
[51] Sebastian Ruder. “An overview of multi-task learning in deep neural networks”. In: *arXiv preprint arXiv:1706.05098* (2017).

FACULTY OF ENGINEERING AND ARCHITECTURE

Design and analysis of domain adaptation systems for joint multilingual coreference resolution

Karel D'Oosterlinck
Student number: 01609137

Supervisors: Prof. dr. ir. Chris Develder, Prof. dr. ir. Thomas Demeester
Counsellors: Semere Kiros Bitew, Ir. Johannes Deleu

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2020-2021

GHENT UNIVERSITY