Proceedings of the Second IN5550 Teaching Workshop on Neural Language Processing (WNNLP 2020)

Jeremy Barnes, Andrey Kutuzov, Stephan Oepen,
Vinit Ravishankar, Erik Velldal, and Lilja Øvrelid (Editors)

May 27, 2020

Oslo University (Norway)

Published by

Language Technology Group

Department of Informatics

University of Oslo

Preface

We are delighted to present the proceedings of the Second IN5550 Teaching Workshop on Neural Natural Language Processing (WNNLP 2020). Spurred by great advancements in neural approaches to NLP, this is the second in a series of successful annual workshops, each showcasing some of the best efforts made by the Oslo MSc students in Language Technology completing the IN5550 class—all tackling modern NLP research tasks. We are also thankful to the Department of Informatics for hosting the workshop, and to its Language Technology Group (LTG) for sponsorship of the Best Paper, Outstanding Reviewer, and Top Performer awards.

The workshop received seventeen submissions (by 23 authors), of which fifteen have been accepted for publication as part of the WNNLP 2020 proceedings (this volume). This programme would not have been possible without the assistance of all our reviewers, whose careful and constructive feedback has been an important element in finalizing the individual contributions. To encourage the spirit of good peer review, we have made the decision to include two Outstanding Reviewer awards in this workshop, in addition to the traditional Best Paper award.

The Programme Committee has selected the paper *Targeted Sentiment Analysis with Ensembles* by Halvor Holhjem Bugge for the WNNLP 2020 Best Paper award, reflecting a fine combination of competent engineering, in-depth experimentation, insightful analysis, and lucid visualizations. Best Paper runners-up were the articles *Model and Data Exploration with the NorNE Dataset* by Øystein Skauli and *Not Quite on Target: BIO Tag Variation and Lexicon Inclusion for Norwegian Targeted Sentiment Analysis* by Petter Mæhlum and Muhammad Asad Ali.

Among the pool of wonderful WNNLP 2020 reviewers, the Programmee Committee finds that two deserve a special mention—for providing especially detailed, insightful, and constructive feedback to their peers. The recipients of the WNNLP 2020 Outstanding Reviewer award are Vemund Justnes and Maria Singstad Paulsen. Congratulations to all award recipients (and runners-up)! And, last but not least, warmest thanks to all participants of this workshop, who spent many a sleepless nights working on the projects that are certain to make WNNLP 2020 an exciting and stimulating event!

Drøbak and Oslo; May 27, 2020

Jeremy Barnes (Sentiment Track Chair), Andrey Kutuzov (Award Chair),
Stephan Oepen (General Chair and Negation Track Chair), Vinit Ravishankar (PyTorch Master),
Erik Velldal (Named Entity Track Chair), and Lilja Øvrelid (Sentiment Track Chair)

Programme Committee

Muhammad Asad Ali
Awadelrahman Mohamedelsadig Ali Ahmed
Jeremy Barnes
Lotte Boerboom
Halvor Bugge
Maja Buljan
Arthur Dujardin
Jan Andre Fagereng
Adrian Eriksen
Diego Gorini
Silvia Nanjala Walekhwa Hertzberg
Ole Magnus Holter
Ping-Han Hsieh
Vemund Justnes
Mathias Krafft
Petter Mæhlum
Stephan Oepen
Maria Singstad Paulsen
Vinit Ravishankar
Torbjørn Ruud
Øystein Skauli
Fabian Felipe Suarez Peinado
Adrian Tysnes
Erik Velldal
Zhenying Wu
Lilja Øvrelid

Table of Contents

Model and Data Exploration with the NorNE Dataset

Øystein Skauli
UiO / Rødbergveien 5B
oysska@math.uio.no

Abstract

This paper attempts to quantify the effect of model parameters and data augmentations for neural models in named entity recognition (NER). The effects are measured on the NorNE dataset. Most of the model features tested were taken as implemented in the NCRF++ toolkit. Some additional functionality is also implemented and tested. It is shown that more labels can be both beneficial and detrimental to model performance. Model improvement was also seen when using raw text as opposed to lemmatized text for character features. Surprisingly little improvement was seen when expanding the dataset through translation.

1 Introduction

Named entity recognition (NER) is the task of locating and labeling named entities in a text. As with many other labeling tasks neural methods have become quite popular and have given good results, often outperforming alternative models (Yadav and Bethard, 2018). With neural models, one usually has a large amount of hyper parameters including, word/character embedding sizes, layer sizes and layer types to name a few. One can also augment the data in several ways to potentially improve performance. Such a large parameters space, in addition to long training times, can make it difficult to find optimal models and we hope this paper can serve as some further exploration of this large space.

First the data is presented, then an analysis of baseline results. Then an exploration of model limitations and potential solutions. Finally we test how different solutions actually affect model performance.

2 Data

The models will be fit to the NorNE dataset (Jørgensen et al., 2020). This dataset consists of 600,000 labeled tokens. Including two Norwegian written standards "bokmål" and "nynorsk", at first only the bokmål section of the dataset is considered. Though as nynorsk and bokmål are closely related, ways of improving results by including the nynorsk portion of the dataset will also be considered.

The dataset gives several different classes of named entities: person, organization, location, geopolitical location/organization, product, event and derived. As the data was doubly annotated, one could already potentially get some insight into what labels are difficult to separate. The original paper noted that annotators found it difficult to separate, locations and organizations, geopolitical locations and organizations and, product and organization.

3 Baseline Results

3.1 Model Setup

Models are fit using NCRF++ (Yang and Zhang, 2018). It enables easy implementation of many state-of-the-art features giving comparable results to recent models (Yang et al., 2018). The baseline model is similar to the optimal model given along side NorNE (Jørgensen et al., 2020). It uses pretrained word embeddings[1] (Fares et al., 2017) in conjunction with a character level CNN to extract word representations. Then these are given to a BiLSTM to output scores. The scores are then given to a CRF inference layer to predict the final label scores. The hyper parameters used are given in Table 1.

Though it is not explicitly stated it seems

[1]Model 132 and 129 in http://vectors.nlpl.eu/repository/

Parameter	Value
BiLSTM hidden size	200
LSTM layers	2
Dropout	0.5
CNN filters	50
Optimizer	SGD
Learning rate	0.015
Learning rate decay	0.05
Epochs	50

Table 1: Hyper parameters for baseline model.

Jørgensen et al. (2020) used embeddings for full-form words. This is reasonable as one would expect inflection and capitalization to be useful features in NER. However performance increased when using lemmatized embeddings. Full-form gave a strict score of 0.911 compared to 0.917 for lemmatized embeddings. While this was not looked into deeply some potential explanations are given.

The simplest reason might be variance in model performance for different random initalizations, and the initialization used for these experiments leads to worse performance in some cases. We refer the reader to section 8.2 for further discussion around the variance of results.

Another potential explanation is that, with lemmatized words, the embeddings for each word is generated based on more diverse contexts. This explanation can maybe be ruled out as when using the full-form embeddings on a lemmatized dataset it gave similar results to Table 2.

Emdeddings trained through predicting surrounding words also aren't required to encode the information we want. If the additional information in raw words that is useful for NER is not useful for prediction surronding words then this information will not be encoded in the embeddings. This would, however, result in similar performance between raw and lemmatized, but we see a decrease. To explain a decrease, it could also be that the with more possible embeddings it becomes harder for the model to fit to, and understand all words. Even if the size of the embeddings remains the same, there may be less clear borders between words. Section 7 attempts to get benefits of using raw/full-form text while still using lemmatized word embeddings.

Eval Type	Dev Score	Test Score
Strict	0.917	0.860
Type	0.919	0.862
Exact	0.982	0.966

Table 2: F1 scores on the dev and test set for the baseline model. Strict: correct placement and label, Type: only correct labels, allows partial overlap, Exact: only correct placement, allows incorrect labels

3.2 Results

Table 2 gives the prediction scores for this baseline model on the pre-split dev and test set. Scoring is done one a per entity level using a evaluation script by David S. Batista[2]. On a per entity level we consider both the label and the placement. Especially if the named entity consist of several words the model needs to correctly identify where words for a single named entity start and stop.

The baseline results are also better than what Jørgensen et al. (2020) achieved with a similar model. This could be because, by default, NCRF++ picks the optimal model based on per label F1 score, while we pick the best model based on the per named entity F1 score i.e. the score we wish to maximize. Though the paper did not mention the exact criteria used to select models in training, so this may not be correct.

3.3 Analysis

Table 2 shows that it is much harder for the model to predict the correct label type, getting scores 0.919 and 0.862 for the dev and test set respectively, compared to predicting the correct placement. Exact score, only considering placement, gives 0.982 and 0.966 for the dev and test set respectively. As a prediction needs both correct label and correct placement to be considered correct under the strict requirement, the strict score is mostly limited by the models ability to predict the correct label.

To get some idea of what labels are difficult to predict one can plot a confusion matrix of the results given in Figure 1. We can see that the usual mistakes are, PROD is labeled as ORG, LOC as GPE_LOC and GPE_ORG as GPE_LOC. While the opposite is also the case, as there are imbalanced classes, the network is incentivized to prefer the more frequent class in ambiguous cases.

[2]https://github.com/davidsbatista/NER-Evaluation

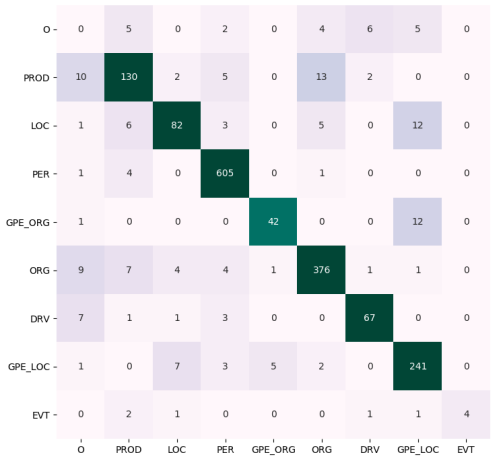


Figure 1: Confusion matrix for dev set predictions

Word	Predicted Labels
Norge	GPE_LOC GPE_ORG ORG PROD
Bergen	GPE_LOC GPE_ORG LOC ORG
Jan	GPE_LOC PER DRV PROD
i	GPE_LOC ORG PROD
Oslo	GPE_LOC LOC ORG

Table 3: Words with the most different predicted labels. Bold labels (none) are in gold labels but missing from predictions

One can get some indication of why these labels are difficult to distinguish if by looking at the words which are given the most number of different labels, given in Table 3.

Table 3 show the words that have several different labels depending on context, and thus often more difficult to predict. These words have exactly those labels that the model mostly struggled with. This is reasonable as, for the words that have a single label, we can just remember the word and the label. It is when the word can have several labels that the network needs to recognize the context and base its predictions on that, which is a more complex task. Though to make the claim of what words are most responsible for miss classifications one should also sort by the words that were miss-classified the most as Table 3 only indirectly takes into account word frequency.

We give the words assigned the most different predicted labels as opposed to most different true labels as it will depend less on rare words. There might be a word that appears 5 times with 5 different labels and in some cases it may then not be realistic to expect the model to predict all of these. By looking at the words with the most predicted labels we get a better picture of whether the model can predict different labels for the same word. Which it seems to be able to. That being said the model should be able to recognize named entities it has not seen before as we can not expect a dataset with all possible named entities in context.

We should also consider that the score we are maximizing in training is not the same as what we wish to maximize in testing. The training loss is based on individual words so we see how well the predicted label score matches the gold labels. The score we wish to maximize is a little more complex as named entities can consist of several words. Since we want high accuracy for named entities we no longer look at words individually. However as you can not take the derivative of this loss, its either match or not so not continuous, so it can not be directly used in training. Of course we do not use per label accuracy directly either we, in this case, use log-entropy loss. There might exist an equivalent for per entity loss.

These two scores have the same optimum, the model predicting all labels correctly, however the training loss may not weigh predictions optimally with respect to the testing loss. For example per label loss would see not labeling an entire 3 word entity and not labeling the first word in 3 entities similarly bad, while the per entity loss would see the former as 1 miss and the latter as 3 misses. This discrepancy can lead to worse test scores and possibly additional variance in per entity model scores. This variance might be lessened by selecting the best model with per named entity scores as opposed to per labels, however this depends on the distribution of model scores which we did not look into.

3.4 Improvements

In the following sections we will analyze how adjustments in three areas affects the performance of this baseline model. In section 4 we will consider different labels and different encodings to potentially help the model learn from the data. In section 5 we increase the amount of data through translation. In section 6 we experiment with different forms of weighting and in section 7 we try different character level features.

4 Adjusting Labels

4.1 Collapsing Classes

For some applications, some label distinctions may not be important. We can then collapse these labels to hopefully achieve better predictions. This happens as, with fewer classes, the classifier will in general have a larger chance of choosing the correct answer in ambiguous situations. It will, however, not necessarily be better. It could be that, for example, the labels represent a natural clustering of the data, and words with different labels in general behave differently. If we then combine these labels the model will have to find this clustering on its own. This could lead to a decrease in performance as the model will have to spend degrees of freedom on this clustering in addition to the task at hand.

We try two combinations of collapsing similar labels. First we combine GPE_LOC and GPE_ORG to one GPE label. Then we consider combining GPE_LOC with LOC and GPE_ORG with ORG similar to Jørgensen et al. (2020). However we provide model performance, both for model trained on the collapsed labels and models trained on the full dataset, collapsing labels after predictions.

From Figure 1 we see that the model had some difficulties distinguishing between the GPE classes. We therefore expect that these classes are somewhat similar and may not present a clustering benefit. One would therefore expect combining them to give increased model performance.

On the other hand we see that the model is rather good at distinguishing between GPE and non-GPE LOC and ORG. This grouping could the be beneficial to the model. Though maybe not more than the benefit of having fewer classes.

Table 4 Gives the F1-Score for fitting the model to these label sets. It gives performance for both training a model on the collapsed labels and training the model on the full label set then collapsing the labels after prediction. If the labels give some benefit in clustering we should see an increase in performance when training on the full set and collapsing afterwards. This is in fact what Table 4 shows. We even see that collapsing the labels which seemed similar, GPE_LOC and GPE_ORG, lead to a smaller decrease in performance than collapsing the more different labels, GPE and non-GPE LOC and ORG, comparing training on full and collapsed data set.

	Trained on Subset	
Label Set	**Dev Score**	**Test Score**
GPE	**0.927**	0.875
LOC-ORG	0.921	**0.884**

	Trained on Full	
Label Set	**Dev Score**	**Test Score**
GPE	0.928	**0.880**
LOC-ORG	**0.929**	0.876

Table 4: F1 scores on the collapsed dev and test set for models trained on the same collapsed labels and full label set. GPE is for the set where GPE_LOC and GPE_ORG are collapsed to GPE. LOC-ORG is the set where GPE and non-GPE LOC and ORG are collapsed.

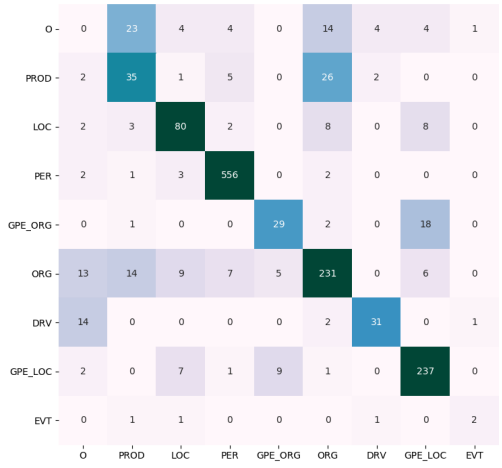


Figure 2: Confusion matrix for test set predictions of baseline model

Encoding	Strict Score	Exact Score
BIO	**0.916**	**0.982**
BIOS	0.912	0.981
BIOES	0.912	0.978

Table 5: F1 scores on the dev set for the baseline model. Strict: correct position and label, Exact: only correct position

The test score does not fit with this analysis however, seeing the inverse trend. Though the test data set seems to have some text where the model will predict many spurious ORG labels and mistake more PROD labels for ORG as seen in Figure 2. This is the case also when selecting the best model based on test set scores. This could be caused by the training set not being representative of this text in the test set or the model in general not being able learn the features important in this text. In any case it could potentially explain the discrepancy in results between dev and test set.

It should be mentioned that adding more labels wont always help. As you add more labels you get less data per label. With less data per label the model becomes worse at classifying the label. So if the decrease in performance due to a decrease in data per label is larger that the gain in performance due to some additional clustering then the model trained on the full set will perform worse. This can happen either because we have little data to begin with or the clustering is not helpful.

4.2 Label Encodings

Different label encondings give different amount of positional encodings to word labels. The BIO encoding assigns each label to be either at the beginning, B, inside, I or outside, O, a named entity. BIOS also gives single word named entities a label, S. BIOES also gives a label the the last word in a named entity. This can help with model scores as, previously mentioned, the score we are optimising is not the same as our evaluation score. The results for training with these encodings is given in Table 5

Unlike previous experiments with this dataset we do not see any improvement in using more complex encoding schemes (Jørgensen et al., 2020). It could be that with more complex labeling the number of instances of any single positional label can decrease and the overall distribution can become skewed. We can potentially see much more inside words than single and vice versa. Since the per-label score we train on does not loose a lot by ignoring some words in a named entity, schemes like BIOES can lead to the model, for example, more often ignoring ending as predicting ending can be wrong the vast majority of the time. But since the testing loss is wrong if any label is incorrect or missing this score is more sensitive to such mistakes. As we see no gain in Table 5 we might be in a situation where the decrease in performance due to the skewed amount labels is similar to the gain from adding more labels.

5 Expanding Dataset by Translation

In general model performance should improve when adding more data. With more data the amount of noise becomes relatively smaller to the amount of data. This means the model can more clearly define borders between classes and these borders will be less affected by noise, less overfitting. One would expect even the same model to perform better, but more data also allows for more complex models without a decrease in performance due to overfitting.

However this all assumes that the new data comes from the same distribution as our previous data, especially the same distribution as our test data. This is not the case for the data we have. Our extra data has text written in a different form of Norwegian. This can be solved by training embeddings jointly on nynorsk and bokmål (Jørgensen et al., 2020). We will instead try translating the nynorsk text to bokmål before training.

We translate the text using Apertium[3]. We use lemmatized text and therefore felt more comfortable translating on the word level. While it is not always the case, word order and sentence structure is often the same between nynorsk and bokmål and as we use lemmatized text we do not need to figure out the correct inflection. While this approach will not always result in a prefect translation we can be sure that the labels will match after translation. Translation at the sentence level may produce a longer or shorter sentence, or a sentence with different word order. Especially the case with just different word order can be hard to handle correctly. We would argue that miss aligned labels is more damaging to model performance than incorrect grammar and therefore opted for per word translation.

[3]https://github.com/apertium/apertium-python

Model	Dev Score	Test Score
NOB	**0.917**	0.860
NOB-NNO	0.913	**0.880**

Table 6: F1 scores on the dev and test set for the baseline model trained on model the bokmål dataset, NOB, and the combined dataset NOB-NNO.

Combining the bokmål and nynorsk datasets double the amount of words in our training set. To keep the results comparable to previous sections we still use the same dev and test set. Training the baseline model on this data set gives the results shown in Table 6

Table 6 does not show a clear improvement by using the larger dataset. This can be caused by the quality of the data not being so good and after this type of translation, as in it is not very representative of the dev and test set. However this seems strange as the sentence structure between bokmål and nynorsk is similar. The static word embeddings also only have around 1% out of vocabulary words for this dataset so words are clearly translated to bokmål.

Another explanation for seeing such a small difference could be that the model does not have the ability to exploit the larger amount of data. To investigate if a larger models give improved results we trained a grid of the LSTM hidden size, LSTM layers and character level feature size. The LSTM hidden sizes tested were 200, 250 and 300, we tested 2 and 3 LSTM Layers, and a character level feature size of 50 and 100. We acknowledge that this is a somewhat small grid and we could have gotten different results by using a larger grid, however none of the models tested surpassed the performance of the baseline model.

Overall this result is a bit surprising. Other articles have shown a slight increase in model score taking advantage of the nynorsk set (Jørgensen et al., 2020). However the extra data was included differently, doing joint training on both languages. The differences were not larger than what is seen here with them seeing a slight increase in dev set score but decrease in test score. Overall this result can indicate that we have a model limitation and further changes have to be made to the model to take advantage of additional data.

6 Weighting

We mention several times that the training loss is not the same as what we are trying to maximize. One option to potentially lessen the difference is to weigh different positional words differently. For example one could try to increase the weight of starting and ending words, and decrease the weight for inside and outside words. This could help the model give the correct importance to matching the start and end of a named entity compared to a unweighted training loss. However we saw in Table 2 that the limiting factor for our models is their ability to correctly predict the label of an entity not give the correct positions. So there might not be a large incentive to do so.

However one application for a similar type of weighting is if you are in a situation where the model missing a named entity is much less desirable than creating a spurious named entity or vice versa. In such a situation you want the model to prefer either missing or generating extra entities over the alternative. To achieve this, in training one can weigh the loss from mis-labeling an outside word differently than a word in a named entity. Weighing down the loss of classifying an outside word as a named entity will in general lead to the model making more spurious labels and have less missed labels. While weighing down the loss of classifying a word in a named entity as an outside word will have the opposite effect.

Weighted loss is not implemented for a CRF layer in NCRF++, but we can see the effect of weighting if we use normal softmax. The results of different weights for outside word loss is given in Table 7. In this case, the loss from classifying an outside word as part of a named entity was weighted down. This means the model should give more named entity labels thereby, hopefully, missing fewer named entities. Table 7 shows this trend. While a weight of 0.4 does not seem to fit this trend. We see that the model performance also had a drop for this weight, this could indicate that the increase in missed labels is due to the random variance in model performance. We also see a increase in spurious named entities which is an inevitable trade off when minimising missing words using this method. The model performance will also in general decrease as the metric we use effectively weighs misses and spurious entities equally.

We also wanted to implement weighted loss for models using a CRF layer. One of the simpler

Weight	Missing	Spurious	Score
0.1	**53**	119	0.863
0.4	78	119	0.857
0.7	71	82	0.869
0.9	82	**70**	0.879

Table 7: Missing and spurious named entities for different weighting of loss for outside label. Results from model using softmax.

Weight	Missing	Spurious	Score
0.5	**24**	26	0.914
0.6	25	35	0.914
0.7	29	**22**	0.910
0.8	**24**	23	0.911
0.9	40	27	0.909
1	30	**22**	0.917

Table 8: Missing and spurious named entities for different weighting of loss for outside label.

ways of performing weighting is by subsampling observations that should have less weight. However we have sequential data, and the model needs context to make its predictions. Therefore only giving the model a subsample of the input is not expected to give good results. Though as the full sequence is given at test time it could be seen as some form of regularisation, however this is not what we are trying to achieve. We instead gave the model the full sequence but removed the gradients from a subsample of the input thereby not letting the model update its parameters for some of the input. There are two issues with our approach. Firstly the gradients are removed after the CRF layer, so the network parameters are updated according to the user set weights however the CRF parameters are not, potentially lessening the impact of the weighting. Secondly it diverges for small weights, but we get some results given in Table 8.

The impact is less clear in this case. It does seem like the CRF layer mostly learns this weighting and compensates for it, therefore we do not see any large impact in this case.

With a weighed loss, what could be useful in this case is weighing different classes of named entities differently as there is some imbalance between the classes. In Figure 1 and 2 we see that the model to a much larger extent mistakes uncommon labels for common labels than the other way around. This is optimal if we want high accuracy but may be undesirable depending on the application. To mitigate this one can have a weight that is inversely proportional to label frequency. However as our method for weighting did not perform that well we do not have any results for such a weighting scheme.

7 Raw Text for Character Features

By default NCRF++ character level features are trained on the same data as what word embeddings use. Word embeddings can be trained on both lemmatized words and raw text. We use lemmatized word embeddings as the model seemed to perform slightly better. However there is no requirement that the character level features need to be trained on the same type of lemmatization as the embeddings use. We could therefore train the character level features on raw text. Raw text contains more information than lemmatized text, like inflection and capitalization, that could be useful when determining what is a named entity. However, with more information, raw text is also more noisy and it could be more difficult to learn something from it.

Features like capitalization and inflection are dependent on where a character is in a word. If we want the model to be able to correctly recognize features like this we can not use a CNN. With a CNN positional information is lost when we pool the filter results. While a CNN could create filters that are mostly correct, for example a filter that recognizes upper case letters could usually be activated on capitalized words, an LSTM can handle positional information. We therefore try both CNN and LSTM character filters which gives the results shown in Table 9.

Overall using character features leads to improved performance which is common and it is at least expected that this helps the model understand out of vocabulary words (Yang et al., 2018). For the test set especially we also see increased performance when using raw text for character features. The fact that the performance in increased for the test set is especially good as the model selection is independent of the test set and therefore it would seem like the increased information from raw words does not lead to overfitting.

We do, however, not see any improvement from using LSTMs over CNNs. This could be a result of RNNs having a more difficult time learning.

Layer	Size	Dev Score	Test Score
None		0.899	0.844
CNN_L	50	0.914	0.846
CNN_R	50	0.917	**0.866**
LSTM_L	50	0.897	0.845
LSTM_R	50	0.916	0.850
CNN_L	100	0.909	0.854
CNN_R	100	**0.919**	0.863
LSTM_L	100	0.902	0.849
LSTM_R	100	0.918	0.860

Table 9: Results from fitting different character feature extractors. The subscript L and R refers to whether we give lemmatized or raw words for character features. Lemmatized word embeddings was always used.

While using an LSTM cell helps with the problem of vanishing gradients, we still need to learn what information to keep stored as we move over the word. We do see that the LSTM seems to more clearly improve for larger sizes, but models with even larger sizes, we tested 150 and 200, did not perform any better. As mentioned earlier it could also be the case that the CNN can create filters that for most part capture features in the relevant position as the character combination it looks for usually only appears in a specific part of a word. Analyzing exactly what the model has learned can be difficult for neural nets, but it could be interesting to see if some pattern exists in the feature vector corresponding to specific features like capitalization and specific inflection.

8 Future Work and Comments

8.1 Label Encodings

We did not see much of a difference in model predictions when using different label encodings. While we gave some explanation as to why this could happen. We do not give any results that show this is correct. One could try looking at the rate at which the model includes words at different positions with different label encodings. Maybe one could see a significant difference in number of ending words or single words included between label encodings. To see if we are in a situation where, while the model is worse at predicting certain positions, the model overall ability is not worsened.

8.2 Reduce Variance

All of the values given here were achieved by just training a model once. This means there is definitely some noise in the results. We can for example see Table 9 where an LSTM of size 50, fit to lemmatized words, suddenly performs badly. This is probably due to random variation in model performance. However we can not re-run just that model as it would get an unfair advantage. We also cant re-run all models untill we get results we agree with. What one should to to minimize the variance is to refit all models a pre-set number of times and take the mean or max to get the approximate average or optimal performance. This was not done here due to time constraints. Just using a static seed does not fix the problem as the set seed can be beneficial for some models and detrimental for others.

8.3 Use of Test set

This paper is more meant as data exploration, if one wanted to make the best possible model and be able to report its performance one should ignore the test set up until the model has been finalized to avoid overfitting. This is because a hyper parameter that is good for some sample of the data is not necessarily optimal for the true distribution. In our case the test data is used more as a secondary validation set. The reason is, in training, model performance is measured in relation to the dev set so a model might be selected because, due to some random variation, it was very good on the dev set, overfitted. However the model is independent of the test set, as we do not select model or parameters depending on test set performance, so this is usually a better estimate of how our experiments affect performance.

8.4 Conclusion

In this paper a few different modifications to a baseline model and data were presented. We found that when using lemmatized word embeddings we can give raw text to character features for further model improvement. We hypothesize that this is because the model manages to recognize some useful features, like capitalization, but we do not show that this is the case. An analysis of how collapsing labels affects model performance found that while collapsing labels generally leads to improvement some care has to be taken when deciding whether to do it before or after training. The

tests on weighting and translation did not show much potential for model improvement.

References

Murhaf Fares, Andrei Kutuzov, Stephan Oepen, and Erik Velldal. 2017. Word vectors, reuse, and replicability: Towards a community repository of large-text resources. In *Proceedings of the 21st Nordic Conference on Computational Linguistics, NoDaLiDa, 22-24*, Gothenburg, Sweden.

Fredrik Jørgensen, Tobias Aasmoe, Anne-Stine R. Husevåg, Lilja Øvrelid, and Erik Velldal. 2020. Norne: Annotating named entities for norwegian. In *Proceedings of the 12th Edition of the Language Resources and Evaluation Conference*, Marseille, France.

Vikas Yadav and Steven Bethard. 2018. A survey on recent advances in named entity recognition from deep learning models. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2145–2158, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Jie Yang, Shuailong Liang, and Yue Zhang. 2018. Design challenges and misconceptions in neural sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3879–3889, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Jie Yang and Yue Zhang. 2018. NCRF++: An open-source neural sequence labeling toolkit. In *Proceedings of ACL 2018, System Demonstrations*, pages 74–79, Melbourne, Australia. Association for Computational Linguistics.

Sequence to Sequence Learning for Named Entity Recognition

Awadelrahman M. A. Ahmed
Department of Informatics, University of Oslo
aahmed@ifi.uio.no

Abstract

In this report we describe how we approached the named entity recognition (NER) problem as a sequence-to-sequence learning problem. We apply an encoder-decoder sequence-to-sequence model incorporated with an attention mechanism to recognize name entities in a Norwegian dataset (Bokmål and Nynorsk). Machine-translated data from one language to another is used to extend the datasets. The model showed promising performance however improvement directions were suggested at the end of this report.

1 Introduction

Named entity recognition (NER) is formally defined in (Li et al., 2020) as the task of identifying mentions of rigid designators from text belonging to predefined semantic types such as person, location, organization etc. In the literature, NER has been intuitively thought of as a form of multi-class sequence classification where the inputs are words and the outputs are classes (i.e. tags) of named entities. Those tags identify the words (inputs) as to belong to particular objects (e.g. organization or person).

NER has a wide range of applications and use cases in the field of natural language processing (NLP) and information retrieval. Some examples are extracting structured information from health records to study adverse drug reactions in patients due to chemicals in the products (Tarcar et al., 2020). Another application is classifying content for articles in social media by automatic scanning and disclosing the people names, organizations, and places mentioned in them (Lopez et al., 2017).

In this report, motivated by the fact that the named entities are mostly in the form of multiple tags (i.e. a sequence) that are to be produced from a sentence (i.e. a sequence), we are keen to approach the NER problem as a sequence-to-sequence (seq2seq) learning problem. Our choice is also based on our hypothesis about the presence of some sort of tag dependencies and positional information that can be captured if we considered the concurrence of the labels' instances. Seq2seq models are successfully used in machine translation as in (Sutskever et al., 2014). However, there are many variations of seq2seq model architectures in the literature; for this task we chose to examine applying an encoder-decoder architecture supported by a simple attention mechanism (Vaswani et al., 2017) to improve upon the model performance. The attention mechanism lets the model learn to focus over a specific range of the input sequence. In other words, the attention mechanism allows the model to better focus on relevant sections of the input, and have proven to greatly improve the performance of recurrent networks in various NLP tasks as in (Bahdanau et al., 2014). We also used machine translation as data to augment machine-translated data with the original training sets. We evaluate the approach on the NorNE dataset (Jørgensen et al., 2019) which adds named entity annotations on top of the Norwegian dependency treebank. The codes discussed in this report are available in repository[1].

The main work in this report is summarized as follows:

- We applied a seq2seq model based on an encoder-decoder architecture supported by an attention mechanism.
- We extended the dataset using machine-translated data from one language to the other.
- We evaluated the models based on different label mappings.

The rest of this report is organized as follows:

[1] https://github.uio.no/aahmed/NER

Section 2 presents NER in a seq2seq context. Section 3 illustrates the proposed model. Sections 4 gives a review of the implementation details. Section 5 discusses the results and disclose some challenges giving some suggestions about future improvements. Lastly, we draw the conclusion in Section 6.

2 NER as Sequence-to-Sequence

In this report, we perceive the NER problem as a sequence-to-sequence (seq2seq) learning problem. Let the input sequence and the name tags to be $\mathbf{X}$ and $\mathbf{Y}$ respectively. We assume both sequences have the same length (T), i.e. all words are annotated, as in (1a) and (1b).

$$\mathbf{X} = (x_1, x_2, ..., x_T) \tag{1a}$$
$$\mathbf{Y} = (y_1, y_2, ..., y_T) \tag{1b}$$

The ultimate goal of the NER seq2seq learning is to model the conditional probability $P(\mathbf{Y}|\mathbf{X})$ and generate the names sequence $\mathbf{Y}$ when the input sequence $\mathbf{X}$ is given. However, the seq2seq model does not model the probability $P(\mathbf{Y}|\mathbf{X})$ directly, instead it models $P(\mathbf{y_t}|\mathbf{Y_{:t-1}}, \mathbf{X})$; which is the probability of generating the name tag y_t given the previous tags in the sequence. This is where the recurrent neural network (RNN) comes into play to model this mapping as it is well known for modeling sequences and dependencies. A central hypothesis of perceiving NER as a seq2seq learning problem is the presence of lexical and positional information in the labels side as well as the sentences side. That can be logical when we include the labels encoding such as IOB.

3 Proposed Model

As we propose a seq2seq approach to address the NER problem. Inspired by the work in (Bahdanau et al., 2014) which successfully applied seq2seq in translation, we construct our NER model that consists of two recurrent-neural-network-based parts work together to map text sequences to their associate name tags sequences. An encoder network transforms an input sequence into a compact vector which is called a *context vector*, and a decoder network transforms the encoded vector into a label sequence. To leverage our model performance we apply an attention mechanism, which allows the decoder to focus and *pay attention* to a specific range of the input sequence. A general layout is shown in Fig.1 and a more detailed architecture is shown in Fig. 2. Next we disclose more details about each part.

3.1 Embedding layers

Both encoder and decoder parts have embedding layers. Words and tags embeddings were generated using a pre-trained models to initialize those embedding layers. These pre-trained models accessed in (Fares et al., 2017).

3.2 Encoder

The encoder network consists of an embedding layer and a recurrent unit layer. We selected our recurrent network to be a bidirectional gated recurrent unit (GRU) as it has a better performance than the standard recurrent unit and a comparable performance to the long short-term memory (LSTM) cells with less number of parameters. For every input token the encoder outputs a vector and a hidden state, and uses the hidden state for the next input word.

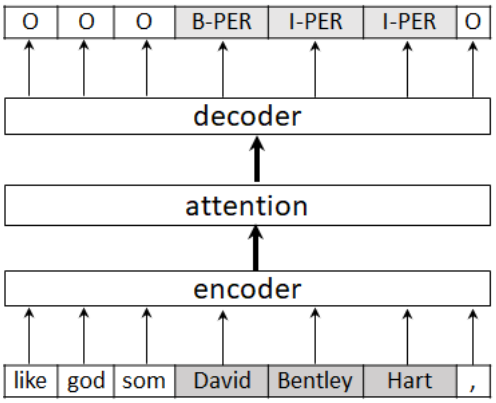


Figure 1: Sequence-to-sequence NER model layout

3.3 Attention

The essence of the attention mechanism is to allow the decoder to look at the entire input sequence at each decoding step via its hidden states. This is expected to allow the decoder network to focus on a different part of the encoder's outputs for every step of the decoder's own outputs.

Attention is performed in two steps, first calculating a set of attention weights illustrated by block *a* in Fig. 2. This step can be done by a feed-forward layer with softmax activation as the weights should be between 0 and 1 and sum to 1. The attention weights are calculated using the decoder's previous state and encoder hidden states as inputs to the

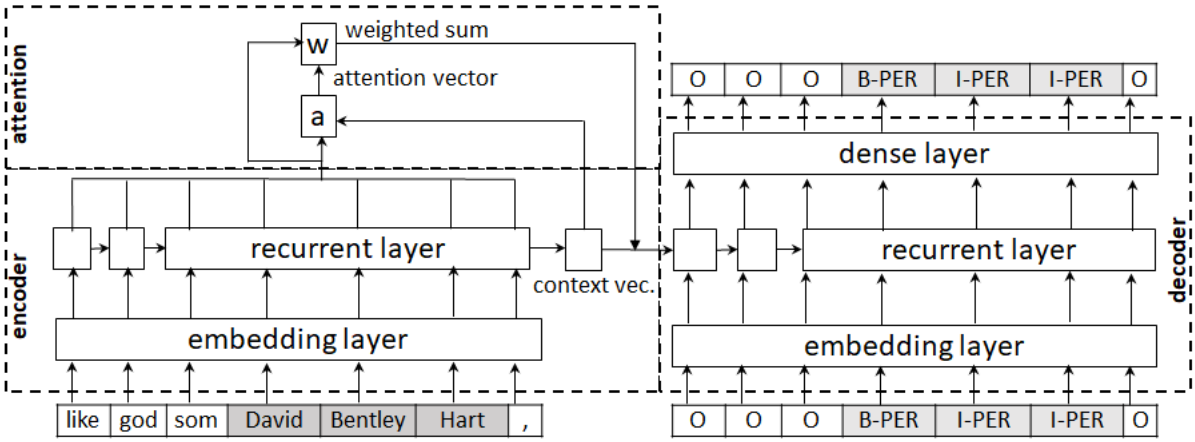


Figure 2: Sequence-to-sequence NER model structure

feed-forward network concatenated together. This attention layer size should be large enough fit all sentences in the dataset. The second step is to multiply these weights by the encoder output vectors producing a weighted combination, illustrated by block *w* in Fig. 2.

3.4 Decoder

The weighted combination resulting from the attention layer is fed to the decoder GRU with the tags embeddings. Meaning that, the decoder's hidden state is computed with a context vector, the previous output and the previous hidden state. But now we use not a single context vector, but a separate context vector for each target tag. Softmax is applied to the decoder output to calculate the tags probabilities.

4 Implementation details

This section illustrates the implementation details, here we have to reference two sources helped us to adapt some parts of the code (Robertson) and (HWeidman). We mainly used `PyTorch` framework to build the model and `TorchText`[2] package to load the datasets. The code is available in repository[3].

4.1 Data preparation and inspection

We tested the proposed model on NorNE (Jørgensen et al., 2019), a manually annotated corpus of named entities which extends the annotation of an existing Norwegian dependency treebank. We performed our experiments on both standards of Norwegian - Bokmål and Nynorsk - with two different mappings using the IBO labeling scheme. For more details about the tags explanation, reader can refer to (Jørgensen et al., 2019). Noting that we ommited the `MISC` tag as it is very rare. The two mappings are as follows:

- NorNE-7: combining instances of the geo-political subcategories GPE_ORG and GPE_LOC to the more general type GPE.
- NorNE-6: dispensing with the geo-political types entirely, merging GPE_ORG and GPE_LOC into ORG and LOC respectively.

Firstly, the `CoNLL-U` data files are transformed to two-column `tsv` files containing the text and the tags' sequences with a modified version of the on line available script [4]. We prepared separate groups of files to meet our mapping schemes mentioned above above. A sample of a data point is shown in Table 1.

Inspecting the data, evidently we can observe the extreme imbalance of the classes as we see a highly skewed distribution with more than 90% of the labels are of type "O" as shown in Fig.3. Note that we can not apply usual data balancing methods like undersampling or oversampling as these methods assume the independence of the labels which is not the case here where the labels dependency is inherited in the sentence sequences. In other words we can not delete or duplicate some labels without changing the sentences and loose their meaning. So we decided to use the dataset as it is and make more rigorous performance assessment by focusing on the per-class F1 scores rather than the average

[2]https://pytorch.org/text/
[3]https://github.uio.no/aahmed/NER
[4]https://github.uio.no/in5550/2020/blob/master/obligatories/3

'Thorbjørn'	'Jagland'	'skal'	'lede'	'utenrikskomitéen'	'.'
'B-PER'	'I-PER'	'O'	'O'	'O'	'O'

Table 1: Data point sample

values. Also, note that we did not apply text processing (e.g. we did not lower cases as the capital case is a good predicting feature for many names).

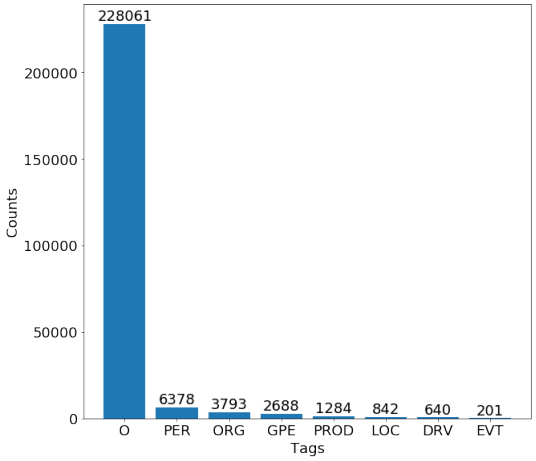


Figure 3: Class distribution of the Bokmål training set

Next step is to build vocabulary. To avoid bias, we built the vocabulary *only* using the training set. This will give us a realistic performance on the test set that is reliable if we deploy the model. This is more realistic than the practice in using NCRF++ (Yang and Zhang, 2018) toolkit, where the test set embeddings are used during training.

For the experiments we also extended the datasets for each language by using a machine-translated version from the other language and combined with the original datasets. We used an online translation tool `Apertium`[5].

4.2 Encoder

The encoder structure is shown in Table 2. It consists of an embedding layer that is initialized using pre-trained model embeddings of dimension of 100, accessed as in (Fares et al., 2017). Dropout is applied, then the embeddings are passed to a bidirectional GRU to reach a compromise between performance (over standard RNN) and computation efficiency (over LSTM). The forward and backward hidden states are concatenated and passed to a fully connected layer with dropout and `tanh` activation function.

[5]https://www.apertium.org/

4.3 Decoder with attention

The attention layer is incorporated in the decoder part, so we illustrate it here together with the decoder. The attention layer tries to estimate which words in the input sentence we should pay the most attention to in order to predict the tag of the next word. This is done by taking what we have decoded so far (i.e. decoder previous hidden state) and all of what we have encoded (i.e. encoder hidden states) and produces an attention vector. This attention vector has a length of the input sentence and element values between 0 and 1 and sums to 1. This is implemented using a feed-forward layer with a softmax activation.

This attention vector is used to create a weighted vector out of the input sentence, specifically by weighted-summing the encoder hidden states, using the activation vector as the weights. So that the input to the decoder in this case will be the weighted sentence vector, the previous decoder hidden state and the tags embeddings. Then passing the current tags sequence, the weighted vector and the current decoder state to a fully connected layer to predict the next tag.

4.4 Model training

As mentioned earlier, both encoder and decoder parts embedding layers are initialized using a pre-trained models produced by word2vec containing about 1.4 billion tokens, accessed in (Fares et al., 2017), we use a 100-length embedding. Other layers are initialized with values sampled from the normal distribution $\sim \mathcal{N}(0, 0.01)$. We use Adam optimizer with learning rate of 0.01 to minimize the cross-entropy loss. We trained for 20 epochs with recursively comparing the epoch validation loss with the previous one and save the model with less loss to avoid saving an overfitting-model.

4.5 Performance metrics

As we showed previously in Fig.3 that the datasets have an obvious imbalance with extremely skewed class distribution. This makes its not accurate to report the *accuracy* of the model as a performance metric, as it does not reflect a realistic model performance if the model predicts the class with high appearance frequency 100% of the time. Hence we

Layer	Description
Encoder	
Embedding	`sentence_vocab_size` × 100
Dropout	`probability= 0.50`
Recurrent	`GRU, bidirectional, hidden_size=256`
Dense layer	`Linear, [hidden_size= 256], tanh`
Dropout	`probability= 0.50`
Decoder	
Attention	`Linear, [hidden_size= 120], softmax`
Embedding	`tag_vocab_size` × 100
Recurrent	`GRU, unidirectional, hidden_size=256`
Dense layer	`Linear, [hidden_size= nr_of_classes], softmax`

Table 2: Model structure

will report the per-class F1 score, recall and precision; this is in addition to reporting the aggregated confusion matrices for each name entity class. This evaluation is done on the token-level, this is not the standard nor the strict method of evaluation, however it gives enough motivation for using seq2seq for NER, we expect entity-level evaluation to give lower performance values. It might be useful to mention the formulas here as we will refer to them in the next section. The formulas are shown in equations (2, 3 and 4), where TP is the number of true positives, FP is the number of false positives, TN is the number of true negative and FN is the number of false negatives.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

5 Results and Discussion

In this section we summarize our results and conduct some discussion. We start with discussing the per-class performance of the model. In Table 3 and Table 4, we show per-class precision, recall and F1 score for two selected mappings. We show the NorNE-7 for Bokmål and NorNE-6 for Nynorsk. We show the top classes and ignored the classes with low occurrence as those scored extremely low performance. Other mappings (i.e. NorNE-7 for Noynorsk and NorNE for Bokmål) showed similar performance trends so we can discuss upon these tables as representatives for the others.

The first observation is the fairly high average model performance in term of all three metrics, this is obvious because of the high values associated with the "O" type which represents more than 90% of the classes instances distribution. However, the core objective of NER is to detect the other name entities which reported low values. The second observation is that the model scores high recall values and low precision, resulting in a low F1 score which conveys the balance between the two other metrics. This high recall but low precision indicates that the results contain more false positive values than false negatives (FP > FN), this can be proofed from equations 2 and 3. In other words, for the low precision classes, our model returns many results, but most of its predicted labels are incorrect when compared to the training labels. This is a reasonable result of the imbalanced classes as many of the overabundant "O" type are assigned to some of the entities. We can also see that extending the datasets with the machine-translated versions of the other language, we have a slight better performance. We think that the imbalance problem is even aggravated when combining the two imbalanced sets as in 4, which diminishes the benefit of having more training data.

To have a close look at the results, we show the aggregated confusion matrices of 8 different cases based on language and granularity. The cases are applying the model on Bokmål and Nynorsk mapped on NorNE-7 and NorNE-6 in a case where the original datasets for each language are used and a case where the dataset is extended using translated dataset. These matrices are shown in Fig.5(a-h). We represented the "O" type raw to `nan` (not a number) for visibility purpose, tables 3

	Original dataset			Combined with translated data		
Type	Precision	Recall	F1	Precision	Recall	F1
O	0.99	0.94	0.97	1.00	0.95	0.97
B-PER	0.05	0.62	0.10	0.09	0.77	0.16
B-ORG	0.02	0.25	0.03	0.07	0.48	0.12
B-LOC	0.02	0.46	0.03	0.04	0.38	0.08
I-PER	0.19	0.36	0.25	0.26	0.67	0.38
I-ORG	0.15	0.16	0.15	0.10	0.38	0.16
B-PROD	0.01	1.00	0.01	0.06	0.32	0.10
I-PROD	0.05	0.50	0.08	0.02	0.18	0.03
average	0.99	0.93	0.96	0.99	0.94	0.97

Table 3: Bokmål NorNE-6 on the original dataset and combined with the translated Nynorsk

	Original dataset			Combined with translated data		
Type	Precision	Recall	F1	Precision	Recall	F1
O	0.99	0.94	0.97	0.99	0.95	0.97
B-PER	0.09	0.78	0.16	0.10	0.80	0.17
B-ORG	0.05	0.64	0.09	0.07	0.60	0.12
B-GPE	0.02	0.70	0.05	0.06	0.59	0.12
I-PER	0.17	0.55	0.26	0.26	0.78	0.39
I-ORG	0.12	0.31	0.17	0.05	0.91	0.10
B-PROD	0.03	0.15	0.04	0.01	0.07	0.02
I-PROD	0.18	0.25	0.20	0.07	0.07	0.07
average	0.98	0.94	0.96	0.98	0.95	0.96

Table 4: Nynorsk NorNE-7 on the original dataset and combined with the translated Bokmål

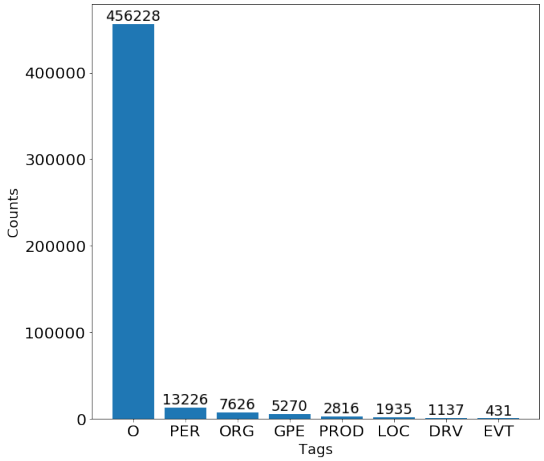
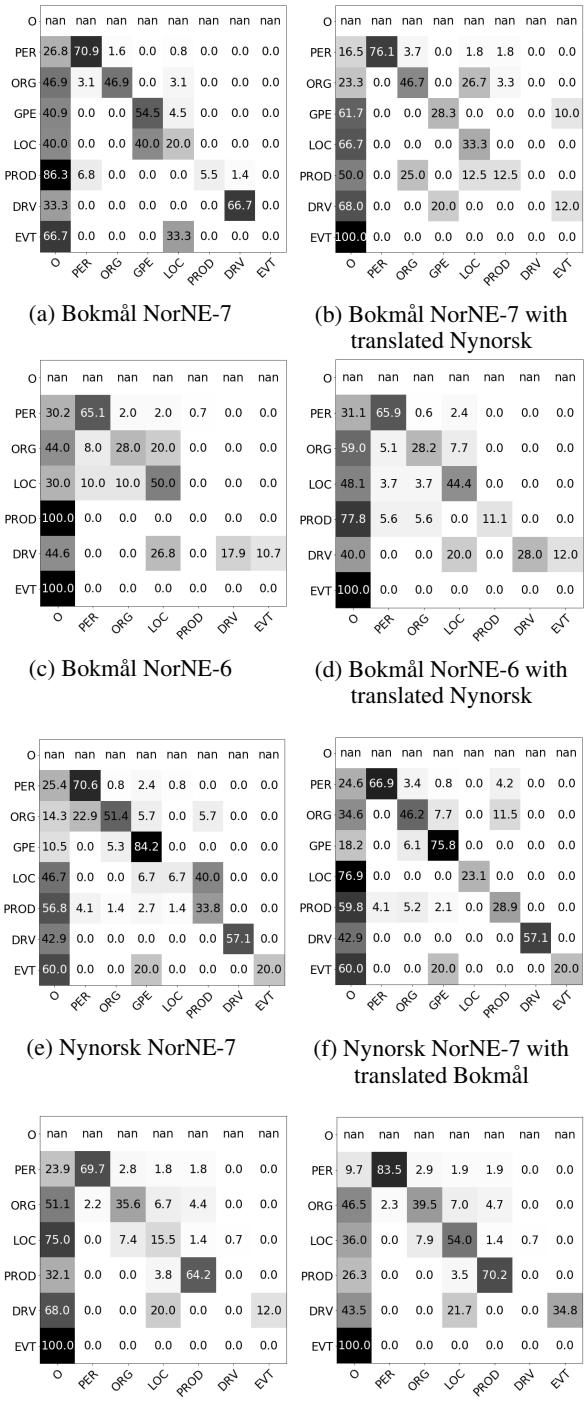


Figure 4: Class distribution of the combined training set

and 4 gives ”O” performance. Generally, we can observe high values in the diagonal indicating good recognition values, showing that most of the labels are either assigned to the correct class or to the ”O” class. However we also observe high values in the ”O” type column, which indicates that many labels were incorrectly assigned to the ”O” class. We can also observe that classes with high presence in the dataset are more likely to be assigned to the correct class, e.g. `PER`, `ORG` and `GPE`, for instance `PER` is the second frequent class after ”O” and has been classified fairly well (65-83%). On the other hand, entities with low presence in the training set has low classification values (e.g. `EVT` is classifies 20% of the time in only one case and has never been classified correctly in other cases). We can also see the improvement when extending the data with the translated datasets.

Despite the results we illustrated so far and the optimism we had about having good performance applying seq2seq supported by attention, the model still has a large room for improvement to show desirable results. This is a main reason that discouraged us to compare to state of the art models as we focused more on applying sequence to sequence learning to the NER. So far, we tried to adapt seq2seq model in the hope to learn from label (name tags) dependencies and treat the tags as sequences which we assumed to have lexical and positional information in their encoded representations. We even supported the model by attention mechanism which is always claimed to be useful



(a) Bokmål NorNE-7

(b) Bokmål NorNE-7 with translated Nynorsk

(c) Bokmål NorNE-6

(d) Bokmål NorNE-6 with translated Nynorsk

(e) Nynorsk NorNE-7

(f) Nynorsk NorNE-7 with translated Bokmål

(g) Nynorsk NorNE-6

(h) Nynorsk NorNE-6 with translated Bokmål

Figure 5: Confusion matrices for each mapping. Matrices on the left are for the original datasets and matrices to the right are when the original dataset are combined with translated versions from other language.

for long sentences and letting the model to focus over a specific range of the input sequence. We supported all of that by extending the datasets with machine-translated data. However, there are many issues with this model that should be investigated further. The biggest issue is the imbalance in the dataset, of course this problem faces all other NER models as it is inherited in the dataset, however that might be reduced by synthesizing datasets with more name entities, a recent work on this is in (Ger and Klabjan, 2019). Even though we tried many hyper-parameters' values (e.g. the number of nodes in hidden layers, dropout values), but more systematic investigation can be carried out on this, for instance increasing the number of layers (we tried that but led to overfitting, so it might be optimized in-line with other parameters), hidden units and adjusting the dropout values. Also, using conditional random field (CRF) instead of the softmax in the decoder output can be investigated. Moreover, other sophisticated seq2seq architectures can be tested to, such as Transformers (Vaswani et al., 2017).

6 Conclusion

In this report we summarized implementing seq2seq architecture for the NER problem. We applied an encoder-decoder with attention architecture. We tested the model in NorNE dataset for both Bokmål and Nynorsk, and extended each datasets by a machine-translated version of the other. The model reported relatively high recall values but low precision values which degraded the per-class F1 scores, mainly due to the imbalanced datasets. We also highlighted some issues and proposed some improvements such as doing more hyper-parameter optimization and using more sophisticated seq2seq models.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Murhaf Fares, Andrey Kutuzov, Stephan Oepen, and Erik Velldal. 2017. Word vectors, reuse, and replicability: Towards a community repository of large-text resources. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 271–276, Gothenburg, Sweden. Association for Computational Linguistics.

Stephanie Ger and Diego Klabjan. 2019. Autoencoders and generative adversarial networks for imbalanced sequence classification. *arXiv preprint arXiv:1901.02514*.

Seth HWeidman. pytorch-seq2seq. https://github.com/SethHWeidman/

pytorch-seq2seq. [Online; accessed 11-May-2020].

Fredrik Jørgensen, Tobias Aasmoe, Anne-Stine Ruud Husevåg, Lilja Øvrelid, and Erik Velldal. 2019. Norne: Annotating named entities for norwegian.

Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. 2020. A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering*.

Cédric Lopez, Ioannis Partalas, Georgios Balikas, Nadia Derbas, Amélie Martin, Coralie Reutenauer, Frédérique Segond, and Massih-Reza Amini. 2017. Cap 2017 challenge: Twitter named entity recognition. *arXiv preprint arXiv:1707.07568*.

Sean Robertson. NLP from scratch: Translation with a sequence to sequence network and attention. https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html. [Online; accessed 11-May-2020].

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Amogh Kamat Tarcar, Aashis Tiwari, Dattaraj Rao, Vineet Naique Dhaimodker, Penjo Rebelo, and Rahul Desai. 2020. Healthcare ner models using language model pretraining. In *HSDM 2020 Workshop on Health Search and Data Mining*, volume 1.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Jie Yang and Yue Zhang. 2018. NCRF++: An open-source neural sequence labeling toolkit. *arXiv preprint arXiv:1806.05626*.

Using Machine Translation to Combine the two Corpora in NorNE

Torbjørn A. Ruud
Department of Informatics,
University of Oslo
torbjoar@ifi.uio.no

Vemund Justnes
Department of Informatics,
University of Oslo
vemundju@ifi.uio.no

Abstract

There exist a very limited amount of high-quality annotated named entities for Norwegian, and as there are two official written languages (Nynorsk and Bokmål), it is more time-consuming to annotate than most other languages. With this in mind, we set out to translate the Nynorsk part of the NorNE corpus to Bokmål in order to increase the amount of available training data for Bokmål. We found out that, although Nynorsk and Bokmål are different languages, the effect on performance caused by the translation is very limited and makes it difficult to determine whether to translate or not – solely based on performance.

1 Motivation

The goal of this experiment was to evaluate the effect of combining the Nynorsk (NN) and Bokmål (BM) datasets from the NorNE corpus (Jørgensen et al., 2020) for named entity recognition (NER) task by translating the Nynorsk dataset to Bokmål. This was motivated as a larger dataset for training a neural network for NER on Norwegian could potentially increase its accuracy of labelling the correct named entities.

2 Introduction

Our task was to experiment with the corpus for Norwegian named entities (NorNE) created by Jørgensen et al. (2020). They have created a corpus consisting of approximately 600,000 tokens, roughly 50/50 split between Nynorsk (NN) and Bokmål (BM) – the two official written languages in Norwegian, and contains more than 28,000 manually annotated named entities.

Considering that applying supervised machine learning methods on named entity recognition (NER) requires a large amount of training data, we set out to explore the possibility of translating the NN part of the corpus to BM, nearly doubling the amount of training data for NER-related tasks in BM. Although NN and BM are two seperate languages, they have a similar sentence structure. Therefore, we wanted to use *Apertium*[1] for automating the translation part of the experiment, and consider the NN sentences that had the same token count after the translation process, to have been successfully translated to BM.

In Section 3, we give an overview of how the data in NorNE is structured, and the results of Jørgensen et al. (2020) experiments, and briefly mention the experiment of using *Apertium* for machine translation between NN and BM done by Velldal et al. (2017). Then, in Section 4, we detail the architecture and parameters used by Jørgensen et al. (2020) so that we can replicate their experiment to have a stronger baseline for our translated training set. Furthermore, we show some numbers related to the translation of NN to BM in Section 5, before we report the results in Section 6 and provide an analysis in Section 7.

3 Background

NER is the task of extracting named entities (e.g., persons, organisations, locations, etc.) from unstructured text written in a natural language (i.e., English, Norwegian, etc.). This has been a research field for years, and with new architectures for neural networks emerging more rapidly in recent years, it is interesting to experiment with applying these architectures to tasks related to NER.

However, training neural networks requires a large amount of data to achieve a confident result. With regards to Norwegian, it is even more challenging and time consuming to obtain a good amount of training data for NER, as Norwegian

[1] https://www.apertium.org/index.eng.html?dir=nno-nob#translation

Standard	Sentences	Tokens	Entities
Bokmål	16,309	301,897	14,369
Nynorsk	14,878	292,315	13,912

Table 1: Distribution of the NorNE corpus (Jørgensen et al., 2020).

Type	Train	Dev	Test	Total	%
PER	4033	607	560	5200	36.18
ORG	2828	400	283	3511	24.43
GPE_LOC	2132	258	257	2647	18.42
PROD	671	162	71	904	6.29
LOC	613	109	103	825	5.74
GPE_ORG	388	55	50	493	3.43
DRV	519	76	48	644	4.48
EVT	131	9	5	145	1.00

Table 2: Entity distribution of BM in NorNE (Jørgensen et al., 2020).

has two official written languages (Nynorsk and Bokmål). In a worldwide context the language is very small which also affects the demand for annotations.

Jørgensen et al. (2020) has created a corpus (NorNE) by manually annotating named entities in Norwegian texts – both NN and BM, using the IOB2-scheme and eight (8) different named entity types. As Table 1 shows, the corpus consists of approximately 600,000 tokens – around 300,000 tokens for each NN and BM.

As Krishnan and Ganapathy (2005) denotes, the IOB2-scheme chunks named entities by tagging 1) the token at the beginning of the chunk as *B*, 2) tokens inside of a chunk as *I*, and 3) tokens that are not part of a named entity as *O*.

As mentioned above, Jørgensen et al. (2020) used eight (8) different entity types, which are defined as; 1) person (PER) – named real or fictional character, 2) organization (ORG) – a collection of people, 3) location (LOC) – geographical places or facilities, 4) geo-political entity a) with a locative sense (GPE_LOC) and b) with an organization sense (GPE_ORG), 5) product (PROD) – artificially produced entities, 6) event (EVT) – such as festivals, weather phenomena, etc., and 7) derived (DRV) – nominals derived from names that are not named entities themselves. Table 2 shows the distribution of named entity types in the BM sets and Table 3 shows the distribution in the NN sets.

Type	Train	Dev	Test	Total	%
PER	4250	481	397	5128	36.86
ORG	2752	284	236	3272	23.51
GPE_LOC	2086	195	171	2452	17.62
PROD	728	86	60	874	6.28
LOC	893	85	82	1060	7.61
GPE_ORG	367	66	11	444	3.19
DRV	445	50	30	525	3.77
EVT	141	7	9	157	1.12

Table 3: Entity distribution of NN in NorNE (Jørgensen et al., 2020).

	Development		Heldout	
Training	BM	NN	BM	NN
BM	89.47	82.34	**83.89**	81.59
NN	84.01	86.53	76.88	83.89
BM+NN	**90.92**	**88.03**	83.48	**85.32**

Table 4: F1-scores of training and testing joint and cross-standard NN and BM on NER models (Jørgensen et al., 2020).

Looking at Table 2 and Table 3, the distribution of entity types between BM and NN looks fairly similiar. However, to some extent, the distribution differs (i.e., 5.74% of annotated entities in BM are locations, while 7.61% in NN), which may affect training. I.e., a trained model for NN could be more accurate at tagging locations compared to a model trained on BM (or vice versa). The reason for the variation between BM and NN is that the sentences that have been annotated are different, and not a translation between those two written languages (Jørgensen et al., 2020).

As the sentences are unique, Jørgensen et al. (2020) experimented with training a NER model on each language and on a combined dataset of both languages. Table 4 shows that the model trained on both BM and NN achieves the highest F1-score on the development set and a high F1-score on the test set. They concluded in the paper that training a joint model (i.e., use both BM and NN data for a single model) eliminates the need for maintaining two separate models, while still achieving a high score.

Although the combined model trained in the NorNE paper achieved a high F1-score, BM and NN are still two different languages. However, the sentence structure is very similar between BM and

NN, and may sometimes be regarded as written dialects – where only certain words differs (Velldal et al., 2017; Jørgensen et al., 2020).

Velldal et al. (2017) used *Apertium* to translate BM and NN. As the structure of sentences in BM and NN are fairly similar, they used an approach where they assumed that if the token count was different between the original and translated sentence, the sentence was not correctly translated and the original was kept – otherwise, it was successfully translated to the other language. Translating from BM to NN, roughly 13% of the sentences had different token count, while roughly 4% from NN to BM (Velldal et al., 2017).

4 Setup

In this section, we present how we setup our experiment to replicate the experiment done by Jørgensen et al. (2020) as accurately as possible, so that our results are comparable to past results.

As explained in Section 3, the best performing model was the one trained on a joint training set consisting of both NN and BM sentences. This is due to the fact that NN and BM have a similar sentence structure, therefore, nearly doubling the amount of high-quality training data (w.r.t. named entity recognition). To achieve this result, Jørgensen et al. (2020) used the NCRF++ toolkit (Yang and Zhang, 2018) for the implementation of the model. Therefore, we also used the same toolkit and setup the configuration to be the same as Jørgensen et al. (2020) did in their experiment. We also used the same pre-trained 600 dimension embeddings which yielded the best result.

The model specification is similar to Chiu and Nichols (2016) and Lample et al. (2016), which follows the architecture of having three layers in the model. The first layer is a character-level convolutional neural network (CNN) that extracts features from the characters. The second layer consists of a word-level bi-directional long-short term memory (BiLSTM) layer to induce context information from words. Finally, the third and last layer is a conditional random field (CRF) that jointly decode labels for the whole sentence (Ma and Hovy, 2016).

Furthermore, the parameters of the NCRF++ toolkit configuration was set 1) to train for 20 epochs[2], 2) using a learning rate of 0.015 with a decay of 0.05, 3) using stochastic gradient descent (SGD) optimizer, 4) with a batch size of 10, 5) using a dropout of 0.5, 6) having L2-norm with a value of 1^{-8}, and 7) with random seed set to 42 (Jørgensen et al., 2020).

[2]Although it was suggested to use 50 epochs, we saw in early test experiments that epochs beyond 20 did not yield any enhancements.

Dataset	Unique entities
NN translated	5244
BM training	4436
Overlap	573

Table 5: Unique entity count for BM training and the complete NN.

To prepare the data, we combined the corpora by shuffling the sentences from the complete corpora (consisting of training, development, and test sets) of NN and the training set of BM. The sentences were parsed from the CoNLL-U format (Jørgensen et al., 2020; Tjong and Sang, 2002; Tjong et al., 2003) to a file that, for each line, contained a token (word) with a corresponding IOB2-tag – as this format is supported by the NCRF++ toolkit. The output received from labelling the data with the model, was a sequence of IOB2-tags in combination with named entity types defined by Jørgensen et al. (2020) – previously mentioned in Section 3.

5 Translation of Nynorsk to Bokmål

In this section we are covering the translation process for translating the NN corpora to BM.

We used *Apertium* to translate using the same approach as Velldal et al. (2017). Each sentence was separately translated and then each word was translated with a space separation to make sure the sentence correspond to the CoNLL-U format. This was to ensure that the tag transfer from NN to BM could be done with ease. One thing to emphasize is that *Apertium* strips away the period at the end of a sentence. Since periods and commas are considered as words with CoNLL-U format that was something that needed explicit handling.

The complete corpora for NN (17,575 number of sentences) was translated to BM, where only 986 sentences had a different token count after the automated translation process. That corresponds to 5.61% of the NN sentences, which is relatively close to what Velldal et al. (2017) experienced in their translation process (roughly 4%). The translated sentences with equivalent length to its NN sentence inherited the tagset directly. However, we

0	B-PER	I-PER	0	0	0	0	0	0	0	0	0	0	0	0	0	B-ORG	I-ORG	0
Milliardæren	Alisher	Osmanov	er	ny	eigar	,	ikkje	berre	av	avisa	,	men	av	heile	Kommersant	forlag	.	
Milliardæren	Alisher	Osmanov	er	ny	eier	,	ikke	bare	av	avisen	,	men	av	hele	Kommersant	forlag	.	

Figure 1: Comparison of a NN sentence and its translation underneath. The BIO tagging is based on the NN sentence.

B-PER	0	0	0	B-PER	0	0	0	0	0	0
Eivind	får	hjelp	av	Håkon	slik	at	utseende	blir	teke vare på	.
Eivind	får	hjelp	av	Håkon	slik	at	utseende	blir	ivaretatt	.

Figure 2: Comparison of a NN sentence and its translation underneath. The BIO tagging is based on the NN sentence.

did not discard the other 986 sentences, as those still could improve the learning. These sentences was included in its original form.

Figure 1 shows a random sample of a NN sentence and the translated result. The translated sentence contains identical word count and sentence structure, and the BIO-tags can be converted directly. In Figure 2, the translated sentence has different length so the original NN sentence is included instead. For this example the sentence is still useful because of the similarity with BM.

By including the complete NN corpora in the training set for BM, we have effectively increased the training set with 17,575 sentences to a total of 33,271, which corresponds to an increase of 111.97%. The joint model experiment also combined training data from both languages so the differences in regards to training amount is not considerably different except that we do include the development and test set from NN. This is because our experiment is based on finding the effect of evaluation against BM.

In Figures 3 and 4, we compare the frequency of the entities for the complete NN corpus and BM training. This is to get a sense of the difference in the most common entities in the two sets. There are some tags like ”Norge”, ”USA” and ”Oslo” that is present in both of the models. Table 5 lists the frequency for the unique entities in the training material. Whats interesting is that there is only 573 entities that overlap, so by combining the translated NN corpora with the BM training set, the amount of entities is increased by 4,671. The low amount of overlap is probably caused by the amount of entities that occur with low frequency. Around 87% of all the entities have a frequency of lower than 4.

Figure 3: Word cloud with entities in NN translated corpus. The most common entity ”Norge” is removed.

In our experiment we have focused on translating all the NN corpus to BM. We have a hypothesis that it will not yield any substantial improvement in performance to do the experiment in reverse because the datasets for each language has comparable distribution. One might try to experiment with a different translator, e.g., Nyno.

6 Experimental Results

In this section we discuss the results of the replication study, as well as training the model on BM and NN translated. Note that, like Jørgensen et al. (2020), we evaluated the models by using strict evaluation – meaning that the reported F1-scores are based on entity-level scores. Strict evaluation has a criterion for the prediction to classify the entity chunks and correct label.

Table 6 shows the results of our replicated model trained on the joint training set compared to the set where NN was translated to BM. Our replication experiment using a joint training set is similar to the score that Jørgensen et al. (2020) achieved in

Figure 4: Word cloud with entities in BM training set. The most common entity ”Norge” is removed.

	Development	Heldout
Training	BM	BM
Joint (BM+NN)	90.62	**84.78**
Translated (NN→BM)	**91.27**	84.37

Table 6: Comparison of F1-scores of our model trained on the joint set of NN and BM vs. the training set where NN was translated to BM. All evaluation was done on the BM development and test sets. Best scores are highlighted in **bold**.

their experiment (see Table 4). Note that Jørgensen et al. (2020) only trained on the joint training sets, while we trained using the complete NN corpora, in addition to the BM training set. However, because these sentences are in a different language, we expected that the impact of including all sentences (when recognising named entities in BM) would be limited.

It is difficult to pinpoint exactly what caused the marginal difference between our joint model and the joint model of Jørgensen et al. (2020). A reason, but to some degree debatable, is that we included all the data in the NN part of the NorNE corpus. However, the reason we think that the amount of NN sentences had a small impact on the outcome, is that the complete corpus increased the total amount of training data, but mainly benefit NN language. Therefore, as we did not test on NN (reasoned by the fact that we used up all NN sentences for training and only focused on translating NN to BM) we might not have seen all the consequences regarding that experiment.

The variation between the joint models can have been affected by other non-deterministic factors as well. As we setup the model exactly like Jørgensen et al. (2020) (see Section 4), we assumed that our results would be the same if we used the same data. However, achieving non-deterministic learning is a highly advanced task where one would also have to work on the hardware level in addition to the implementation level.

On the other hand, under the assumption that we would be affected by the non-determinism – accompanied by the marginal difference between our joint model and the joint model of Jørgensen et al. (2020), made us able to also assume that our joint model was the same as the best-performing model in Jørgensen et al. (2020) (proving that the model is reproducible). This assumption is important as it provides us a basis for how the translated NN corpora performs compared to simply joining NN and BM.

As seen in Table 6, the training set containing translated NN performs almost equally compared to the training set containing both NN and BM. The translated model achieves a F1-score that is 0.65 higher than the joint for the development set and a F1-score that is 0.41 lower than the joint for the heldout set, making it extremely difficult to say with certainty that the translated model is better than a simple joint model of NN and BM. Therefore, given the uncertainty of the performance between the models, the best model should be based on ease-of-use, where simply concatenating sentences to create a larger corpora triumphs the extra step of translating from one language to another – when there is no clear gain (w.r.t. performance) from the extra process.

7 Reflections

This section will cover the analysis and interpretation of the results from the experiment.

As covered in Section 6 the experiment did not yield a noticeably better result than previous experiments. Its hard to pinpoint the exact reason for why, but a reason that might seem obvious is that named entities are rarely translated between languages – as Figure 1 and Figure 2 illustrates. An example of when named entities might be translated is, for instance, geo-political entities such as countries. I.e., Norway is spelled ”Noreg” in NN, while ”Norge” in BM. Considering the low effect of translating named entities, it would be interesting to experiment further by combining other languages that also have a similar structure (i.e., combine NorNE with corpus for Danish and/or Swedish), and see

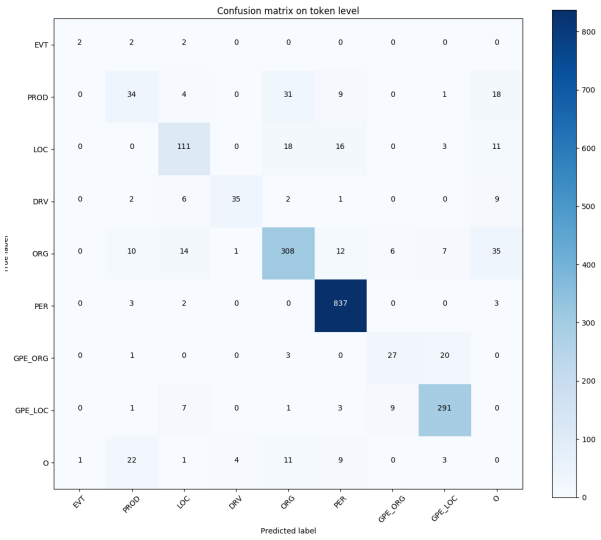


Figure 5: Confusion matrix showing the true labels on y-axis and predicted label on x-axis. The numbers are computed from the test set

how this affect the performance – as joining NN and BM yielded better results than training separately (Jørgensen et al., 2020).

It is useful to consider which labels the model predicted wrongly. Figure 5 is a diagram showing a confusion matrix between the predicted labels and the true labels for our model. The data column *O* for predicted and true label is zeroed since this is the most populated cell, and would not make the color scheme useful. The diagram is based on token level which means that the BIO notation is not taken into consideration. An entity (e.g., an organization) that has three tokens and the model labels one of them to an organization but the others to not an entity (*O*), will be able to count that as one correct. Strict evaluation would require the entity borders to be correct, but that is not considered. This essentially means that an entity spanning over multiple tokens will be considered as multiple unique entities.

The confusion matrix visualizes that the model does a fairly good job of labelling named entities correct. The desired outcome is to have a diagonally line (from top left to bottom right). That line is present but the color varies since the label distribution is skewed per entity type.

The wrong labeling can be categorized into two groups. The first group includes miss-labeling of non-entities, including labeling true non-entities as entities, but also labeling true entities as non-entities. In Section 9 we are proposing an experiment that would aim to reduce the amount of error related to this group. The second group is miss-labeling of actual entities. In this group the percentwise error between the categories are highest for label groups with low amount of instances. The probable reason for this outcome is that the model has reduced instances to train on. The biggest category of miss-labeling in this group is the separation of organization from production and location. One example of an organization that can also be a product is *Skype*.

We also see that geo-political entity with organizational sense (GPE-ORG) is miss-labelled as geo-political entity with a locational sense (GPE-LOC). According to Table 2 and Table 3 in Section 3, roughly 18% of the entities in the complete corpus (NN and BM) are GPE-LOCs, while under 3.5% of the entities are GPE-ORGs, which leads to a highly un-even distribution between the two GPE types. Jørgensen et al. (2020) provided an experiment where they grouped both GPE-ORG and GPE-LOC to a single joint geo-political entity, GPE, which limited the number of miss-labels regarding GPE. However, this removes the additional information of whether it is related to an organization or a location, which might be useful in some NER tasks. Therefore, a possible experiment in the future would be to adjust the distribution of GPE-ORG and GPE-LOC in the training set so that it is more even than now.

8 Conclusion

In conclusion, we have found that under the assumptions 1) including the development and test sentences of NN does not largely affect the score on evaluating BM and 2) affected by non-determinism without highly advanced precautions, then the best-performing model from Jørgensen et al. (2020) is reproducible. Considering that, we have seen that translating from Nynorsk to Bokmål (using *Apertium*) does not necessarily make the model perform better than simply concatenating the two languages. Therefore, we would, in future tasks related to named entity recognition in Norwegian, avoid the extra step of translating NN to BM and just use a joint set consisting of both.

We also discovered some new potential experiments that would be interesting to investigate further:

- Join other similar languages (i.e., Norwegian, Danish and/or Swedish) and see if those can

boost the performance like joining Nynorsk and Bokmål did for NorNE (Section 7).

- Adjust the distribution of geo-political entity tags in order to experience if a more even distribution would increase the performance (Section 7).

- Split into two separate models; 1) a model that handles named entity chunking and 2) a model that handles named entity recognition (Section 9).

9 Future Work

In this section we are suggesting an experiment to reduce the amount of error in non-entity estimation – as referred to as the first group of errors in Section 7.

The outline for the experiment is to separate the two tasks of NER into two different models. The first model would be tasked to only recognize entity boundaries, deciding the token span of an entity (i.e., identify entity chunks). It would only be trained on BIO-tags, thereby, leaving out the details of entity type. The second model would specialize in deciding entity type and only needs training from sentences containing labeled entities. The complete architecture would be more complex compared to having one model doing all the decisions.

A suggestion to the implementation is to train the two different models separately and only considering re-implementation of the evaluation phase. In the evaluation, each sentence would first be passed through the first network to decide if the sentence contains any entities. For each observed entity the second model would be used to pick the entity with the highest probability.

The hypothesis is that this architecture would yield good predictions. The main argument for having this experiment to work is that each of the two models would specialize in the two sub-tasks of NER. Since specialization is a very key aspect of neural networks, the splitting into sub-tasks of NER could have an impact on the performance.

We actually started on the implementation of the experiment and performed some training of the entity-boundary model. It managed to produce an F1-score higher than 95.61%, which indicates that the experiment can work in practice. We did not manage to finish on the combination of the evaluation for the models, but that is something that can be explored further in the future.

Acknowledgments

We would like to thank all the instructors and teacher assistants in the course IN5550 at the University of Oslo. They have provided us with deep knowledge of neural methods for natural language processing that was useful for this paper and will help us with research in the future – mainly our master theses. Also, we would like to thank the reviewers of this paper for providing useful comments to improve the paper.

References

Jason P.C. Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, 4:357–370.

Fredrik Jørgensen, Tobias Aasmoe, Anne-Stine Ruud Husevåg, Lilja Øvrelid, and Erik Velldal. 2020. Norne: Annotating named entities for norwegian. In *Proceedings of the 12th Edition of the Language Resources and Evaluation Conference*.

Vijay Krishnan and Vignesh Ganapathy. 2005. Named entity recognition.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *CoRR*, abs/1603.01360.

Xuezhe Ma and Eduard H. Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *CoRR*, abs/1603.01354.

Erik F. Tjong and Kim Sang. 2002. Introduction to the conll-2002 shared task:language-independent named entity recognition. In *Proceedings of the 6th Conference on Computational Natural Language Learning*.

Erik F. Tjong, Kim Sang, and Fien De Meulder. 2003. Introduction to the conll-2003 shared task:language-independent named entity recognition. In *Proceedings of the 7th Conference on Computational Natural Language Learning*.

Erik Velldal, Lilja Øvrelid, and Petter Hohle. 2017. Joint ud parsing of norwegian bokmål and nynorsk. In *Proceedings of the 21st Nordic Conference on Computational Linguistics (NoDaLiDa)*.

Jie Yang and Yue Zhang. 2018. Ncrf++: An open-source neural sequence labeling toolkit. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Named Entity Recognition on NorNE Dataset: CNNs, BiLSTMs, or Transformers?

Fabian Felipe Suarez **Diego Gorini**
University of Oslo (UiO)
fabianfs@student.matnat.uio.no diegog@student.hf.uio.no

Abstract

This paper discusses two types of neural network architectures for the named-entity recognition track, namely softmax models (CNNs/BiLSTMs) and transformers. For this, we use a newly released norwegian dataset (NorNE). We consider some state-of-the-art approaches in the literature, and compare our results and observations with them. All the code used in our experiments was built from scratch with PyTorch, and it is available on GitHub.

1 Introduction

The purpose of this paper is to report and compare our results from applying what we call softmax architectures (i.e., models with character- and/or word-level CNNs-BiLSTMs, and a softmax inference layer), and Transformers models to the Named Entity Recognition (NER) task. In practice, this is a sequence labeling task. The dataset we used for training and testing is the newly released NorNE dataset (Jørgensen, et al., 2020).

Our main contribution consists in applying model architectures from some top-cited papers in sequence labeling tasks to a norwegian corpus, and trying to reproduce (or at least to reflect) their results. Furthermore, we consider good practice trying to apply consolidated model architectures on new and different datasets, and to compare the respective conclusions. Nonetheless, during experimentation we expanded the hyperparameter tuning in some directions, specially for the transformers models.

2 Dataset

The dataset we used comprises ~300,000 entity-annotated tokens of the Bokmål variety (we completely omitted the Nynorsk variety). Following the advice in Jørgensen (Jørgensen, et al., 2020), we used Nor-NE-7, with the following entity types: Person (PER), Organization (ORG), Location (LOC), Geo-political entity (GPE), Product (PROD), Event (EVT), and Derived (DRV), for a total of 7 categories and 15 classes (including the *outside-of-entity* class 'O'). The split for training, development and test set is 80-10-10 respectively. Since the dataset follows the CONLL-U format, and we used Python3.7 throughout, we used the CoNLL-U Parser (https://github.com/EmilStenstrom/conllu/) to parse the data. Also, we mantained the IOB2 encoding scheme. All the evaluations were performed at the token-level.

3 Softmax Models

Great results in the NER task are achieved by the standard CNNs-BiLSTM-CRF model detailed by Ma & Hovy (2016). It is an end-to-end sequence labeling architecture, which uses both word- and character-level representations, and Conditional Random Fields as the prediction layer. This is the baseline choice of Jørgensen (2020), achieving results comparable to models trained on english corpora. The CNNs-BiLSTM-CRF model improves and builds on "simpler" neural models, such as word-level CNNs-CRF (Collobert, et al., 2011), character-level CNNs-CRF (Santos, et al., 2015), word-level BiLSTM (Ling, et al., 2015), and hybrid word- and character-level (Bi)LSTM-CNNs (Chiue & Nichols, 2016). Recent studies in hyperparameter optimization for sequence labeling (Reimers & Gurevych, 2017a, 2017b) seem to confirm the superior performance of the BiLSTM-CRF over the softmax models, even without character level representations.

In our experiments, we focused on the Softmax models described by Chiue & Nichols (2016) and by Yang (Yang, et al., 2018). From the literature is quite clear that CRF models are superior to softmax models, however the intent of our research is to explore the possibilities of less popular, yet still powerful architectures. Unlike Chiue & Nichols, and following Yang, we tried to optimize end-to-end models, with no hand-crafted features whatsoever. Regarding hyperparameter configuration, we followed the results in the literature (Chiu & Nichols, 2016; Reimers & Gurevych, 2017; Yang, et al., 2018).

3.1 "Lego" Architectures

A softmax model is a typical two- or three-layered framework. It contains a «character sequence representation layer, a word sequence representation layer and an inference layer.». (Yang, et al., 2018, p. 3381). We built and trained 8 architectures (or "lego" combinations). From the simplest to the more complex: (1) word-level 4-layered CNN; (2) word-level 2-layered BiLSTM; (3) word-level 4-layered CNN fed into word-level 2-layered BiLSTM; (4) word-level 2-layered BiLSTM fed into word-level 4-layered CNN; (5) character-level single-layered CNN fed into word-level 4-layered CNN; (6) character-level single-layered CNN fed into word-level 2-layered BiLSTM; (7) character-level 2-layered BiLSTM fed into word-level 2-layered BiLSTM; (8) character-level 2-layered BiLSTM fed into word-level 4-layered CNN.

3.2 Hyperparameters

The following parameters apply to all softmax models. The dimension of the character embeddings is equal to 50 (both before and after the CNN/BiLSTM transformations); the dimension of the word embeddings is equal to 100 (both before and after the CNN/BiLSTM transformations); the dimension of the softmax layer is equal to 200. Max-pooling was applied to all convolutions. Character embeddings were randomly intialized; for the word embeddings, we used the pre-trained word2vec (skip-gram) norwegian vectors trained on the Norwegian-Bokmaal CoNLL17 corpus (www.vectors.nlpl.eu/repository), with no further fine-tuning. In all cases, char- and word-embeddings were concatenated before being fed to the next layer. For the stochastic optimization, we used the Adam algorithm. The learning rate was fixed for all epochs at 1e-3.

3.3 Results and observations

Figure (1) shows the quality of the models in terms of (macro) F1 scores against number of epochs (on the development set).

- All models seem to reach a maximum score around epoch nº 30, before start oscillating up and down. This may mean that some fine tuning (perhaps using a decaying learning rate) would improve their overall performance.

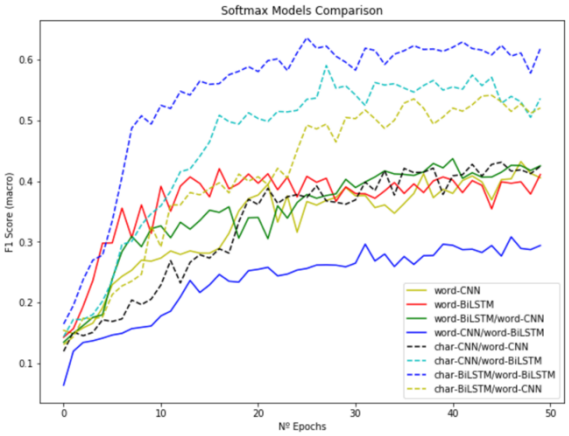


Figure 1: F1 (macro) score vs. number of epochs for the different softmax based architectures.

- Word-level "compound" models (solid blue and green lines in the plot) do not provide any noticeably improvement to performance over word-level "simple" models (solid yellow and red lines). In fact, the worst performing model (solid blue line) is a compound model.
- Models with character-level representations (dotted lines) perform noticeably better than models without them (solid lines). This concurs with the results of recent research. Furthermore, it has an intuitive justification: most entities, for example, start with an uppercase letter.
- Sequential information is crucial. Models with character-level representations (either by CNNs or BiLSTMs) and word-level BiLSTMs representations perform the best. In fact, the model with character- and word-level BiLSTM representations (blue dotted line) is the best performing model in absolute, while the model with character- and word-level CNNs representations (black dotted line) is the worst performing model relative to the three-layered models.

Contrary to what expected from the literature, our best softmax model was not the char-CNNs/word-BiLSTM, but the char-BiLSTM/word- BiLSTM, with 35 and 39 F1 (macro) points on the test set, respectively. This discrepancy with the literature may be due to hyperparameter optimization. Training for the latter model, however, took 3.2 more times than the former, in accordance with the literature.

For better comparison, below we show the confusion matrix results of both models on the test set:

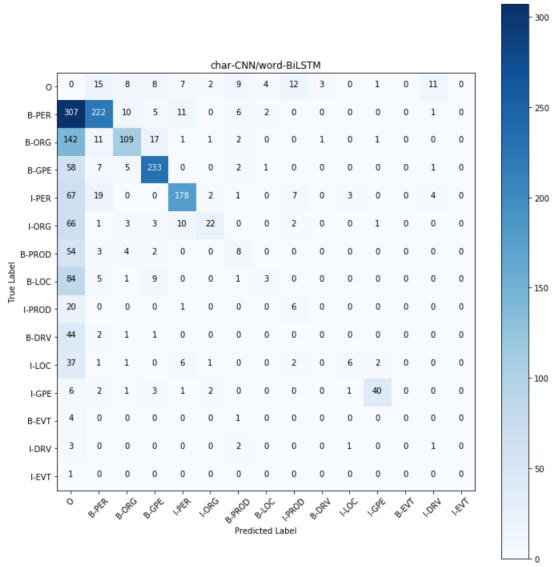


Figure 2. Confusion matrix for the character-level CNNs / word-level BiLSTM model.

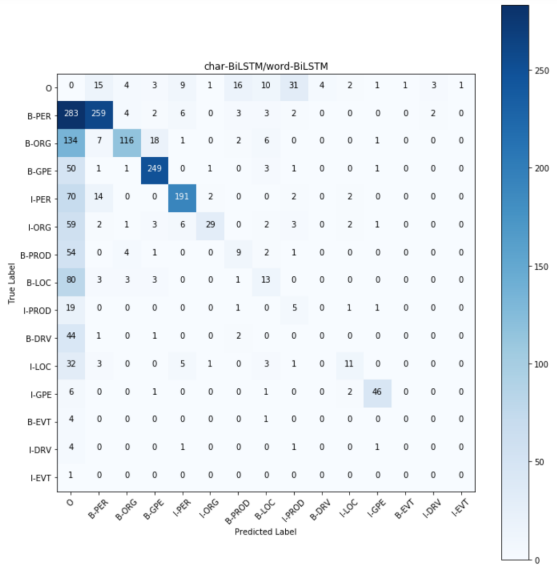


Figure 3. Confusion matrix for the character-level BiLSTM / word-level BiLSTM model.

In general, we observe that our models are not "aggressive" enough, and they miss many entity-tokens (hence the blue-colored "O" column). This is of course due to the nature of the datasets for the task. The supervision signal is weak. We can easily imagine, for example, that many sentences in the batches have no entities at all.

4 Transformers for NER classification

4.1 Evaluation metric

In many papers on NER classification (Hang Yan, et al., 2019) it is popular to use the F1 score, as an evaluation method for the different classification algorithms. However using this metric in the NER task is not straightforward, as this is a multiclass classification problem, which usually has a very uneven distribution of the different tagging features.

On one hand if one uses the micro or weighted F1 score, the first tag will always dominate, thus very high scores of the order of 0.92 - 0.95, will always be obtained. On the other hand if one uses the macro F1 measurement, the scores for the tags that have very few appearances, will weight the same as the other tags, thus highly reducing the score.

Therefore our solution to this problem was to use the F1 weighted score, without the results of the O tags. In this way we would make a metric which takes into account the number of appearances of each tag, but doesn't get overruned by the popularity of the O tag. Therefore from here on, when we mention the F1 score without further specification, we refer to this metric.

4.2 The model

The next part of our research was to investigate the applicability of a transformer in the NER classification task, as opposed to the above mentioned CNNs/BiLSTM/Softmax based architectures. The motivation for using transformer is that this is an architecture known to be much faster to train and which has been quite successful in other NLP tasks.

A transformer is a Neural Network architecture based on the self attention mechanism. It is used to treat sequential data, with the advantage that many of the training calculations can be performed in parallel, unlike other sequential architectures like recurrent neural networks.

The main idea of the self attention mechanism, is to multiply the representation of each element in a sequence with the representations of the other elements, in order to

obtain a new representation which carries information about its surroundings. For example, in the case of a sentence, the embedding representation of each word, gets multiplied with the embeddings from the other words. The most common way of multiplying is to take the dot product. This dot products will then provide a set of weights, which will be used to obtain the new representation of a given word. This is done by carrying out a weighted sum of the embeddings, where the weights are the ones obtained from the dot products.

A transformer can be composed of an encoder and a decoder or just an encoder, depending on the task. Usually the encoder-decoder architectures are used in seq-to-seq tasks like translation or text generation. In the case of NER classification only the encoder is needed. Therefore in order to obtain the predicted tags for each word, the final step after obtaining the new representations for each word, is passing this vectors through one or various neural layers.

It is important to mention that since the product computations between the words are performed in parallel, the self attention mechanism doesn't include per se any information about the order of the sequence, for this reason it is necessary to include a positional encoding that account for the relative position of the elements on the sequence. There a various methods for constructing the positional encoding. It can for example be learned or fixed. In this work we used the sine cosine positional encoding proposed by the authors of the original transformer paper (Vaswani, et al., 2017).

In this research we used Transformers for word embeddings analisis and for character embeddings analisis.

4.3 Code Implementation

For the transformer code the torch text library was used in order to build the vocabulary and constructing the batches. The transformer was obtained from the PyTorch nn library, which from version 1.4 includes a transformer module. In our case we used the TransformerEncoder and the TransformerEncoderLayer classes. Here it must be mentioned that there exist an already optimized library that uses transformers (huggingface/transformers) for different NLP tasks including NER classification.

However for didactic reasons and in order to be able to directly compare transformers with other CNNs/BiLSTM/Softmax architectures we decided to use the more basic PyTorch library.

Perhaps the most cumbersome part of the code was constructing the mask functions to cover the padding in the sentence batches.

In the case of the character embeddings the dynamic was similar. First for each sentence a batch was created. This batch had dimension [NW, MWL, CE] where NW is the number of words in a giving sentence, MWL is the length of the longest word and CE is the character embedding. As it is expected in order to create the PyTorch tensor a padding function for the shorter words had to be constructed. As is the case with the word embeddings, in order for the transformer to carry some positional information of the letters, some positional function had to be implemented. Here we used again the sinus-cosinus function. Then the sentence based character batch would be feeded into the transformer, which would return a tensor of equal dimensions as the input batch, that is a tensor with the new representations for each letter. Therefore in order to obtain one character vector representation for each word, so to later concatenate it with the word embedding, we tried a Max pooling strategy, for each of the components of the character embeddings in each word. Then, as mentioned this new character word representation would be concatenated with the word embeddings before passing it into the sentence level transformer.

4.4 The experiments

In the first experiment the transformer with only word level embeddings was trained. In figure (4) it is shown the F1 score for different epochs and different embeddings sizes on the validation set. In this figure it can be observed that in general the best result was provided by the embedding of size 200 for 5 epochs. With more training epochs the network's score decreases. The training time per epoch for this architecture was between 24-26 seconds.

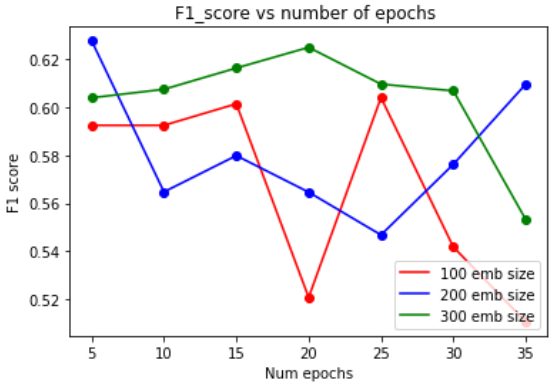


Figure 4: F1 score vs number of epochs for Transformer architecture for various embedding size.

In the next experiments a character level embedding was concatenated to each word. In figure (5) it is shown the F1 score results for various character embedding sizes and different epochs. It can be seen that the best results were obtained for a character level embedding of 60 at 5 epochs. In general it can be observed that the character level embedding increased the F1 score in about one point. The training time for this architecture was about 3 minutes and 40 seconds per epoch.

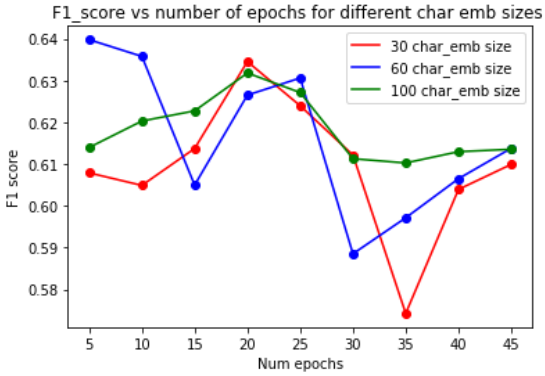


Figure 5: F1 score vs number of epochs for Transformer architecture for various chararacter embedding size.

As it can be observed from figures 1, 2 and 3 Transformer require much less training to reach its peak performance, and also the training times are much faster. It was observed that in general after 20 epochs the scores start to highly oscillate, which again means that using a decaying learning rate.

One of the reasons why our Transformer models didn't perform that well, could be the wrong choice of the positional function. This is so, as the sinus-encoding is fixed and doesn't adapt itself to learn extra features on how to encode the position, which in NER tasks is critical. Dai et al. (2019), propose a new positional encoding which better adapts the positional features.

In general for the transformers including the char embedding information increased only in a small percentage the score of the model. This could be because the Transformer based character level representation didn't manage to catch well the suffix and prefix information, mainly due to the use of a poor positional encoding.

5 Conclusions

A study of different architectures for NER classification task on a Norwegian corpora was carried out.

In general, NER is a difficult task as the biggest part of most corpora don't contain any entity name, which creates a very big unbalance on the distribution of the tags. This has the effects that the algorithm doesn't receive enough examples of some tags, thus doesn't learn properly to identify them. This is observed in the confusion matrix figure (3), where it is seen that the most common mistake that the network make is to take the low appearance tags to be a non entity word, which is the most popular tag.

It was shown that in general using Transformers for NER is possible, and will substantially reduce the training time. However, models based purely on transformer architectures perform lower than CNNs-BiLSTM models. This results are in line with some other literature where it is reported that the use of Transformer models without pre training doesn't yield a very high performance (Hang Yan et al., 2019). However some papers report that using pretraining techniques as BERT or ELMO drastically increases the performance of the model (see, for example, Devlin, et al., 2019) .

Finally, character-level representations are crucial when using CNNs-BiLSTM-Softmax models. However, for Transformer architectures the character-level didn't substantially increase the performance of the model.

References

Jason Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional LSTM-CNNs. In *TACL*.

Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel

Kuksa. 2011. Natural Language Processing (Almost) from Scratch. In *JMLR*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc Viet Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. In ACL, pages 2978–2988.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pp. 4171–4186

Fredrik Jørgensen, Tobias Aasmoe, Anne-Stine Ruud Husevåg, Lilja Øvrelid, Erik Velldal. 2020. NorNE: Annotating Named Entities for Norwegian. ArXiv: 1911.12146.

Xuezhe Ma and Eduard H. Hovy. 2016. End-to-end sequence labeling via bi-directional LSTM-CNNsCRF. In *ACL*.

Wang Ling, Chris Dyer, Alan W. Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luís Marujo, and Tiago Luís. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *EMNLP*.

Nils Reimers and Iryna Gurevych. 2017. Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 338–348.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, et al. 2017. Attention Is All You Need.

Hang Yan, Bocao Deng, Xiaonan Li, and Xipeng Qiu. 2019. Tener: Adapting transformer encoder for named entity recognition.

Jie Yang, Shuailong Liang, and Yue Zhang. 2018. Design challenges and misconceptions in neural sequence labeling. In COLING.

Toward multilingual Named Entity Recognition for Norwegian and English

Ole Magnus Holter
University of Oslo / Problemveien 7 0315 Oslo
olemholt@ifi.uio.no

Abstract

Identifying named entities is an essential subtask in many natural language processing pipelines, such as information extraction and relation extraction. Named entity recognition models have traditionally been trained for only one language and depend on the existence of a large labeled training corpus in that language. The first publicly available labeled corpus for Norwegian, NorNE, was recently released with a version for each of the two official languages (Bokmål and Nynorsk). We do preliminary work in using the pre-trained multilingual BERT model to make taggers for both Bokmål and Nynorsk. We observe that the drop in performance for cross-standard training and testing is not significant, but that the model improves for both languages when trained jointly. We also demonstrate that adding another language to the model not only achieves the best general average performance, but results that are the best or comparable to the best of all evaluations. We also show that the greatest improvement of the performance of the model is seen during the first four epochs and that more training can improve performance, but can lead to reduced cross-lingual performance.

1 Introduction

A Named Entity is a word or a span of words corresponding to a real-world object. Named Entity Recognition (NER) is the task to identify such Named entities and (possibly) assign them to one of several predefined classes. NER is an important subtask in common NLP tasks such as Information Extraction (IE), relation extraction (RE), interlinking documents with structured data (Mendes et al., 2011), and is a useful feature in many other tasks.

More formally, NER is the task, where, given an input sequence $s = \langle w_1, w_2, \cdots, w_n \rangle$ it produces a list of tuples $\langle I_S, I_e, t \rangle$ where I_S is the start index, I_e the end index and t the entity type from a predefined set of entity categories (Li et al., 2020). The entity categories vary from application to application. However, typical categories include *people*, *organizations*, *locations*, *geopolitical entities*, *products*, and often include categories that are usually not regarded as named entities such as *numerical values* and *times*. NER can also be used in particular domains and use a very different set of categories. In the biomedical domain, relevant categories can, for instance, be *proteins* and *genes*.

NER is commonly approached as a sequence labeling task where we would like to find the most probable sequence of labels. It was first solved by using manually created rules before statistical machine learning methods were employed for NER. Currently, state-of-art-systems are Neural Network models. All of the systems can make use of external information.

Pre-trained representations of words (word embeddings) are important components in many NLP tasks. The skipgram and BOW-models introduced by Mikolov et al. (2013) train word representations by predicting either the target word given the words in a context window or by predicting the context words given the target word. The success of these models is to a large extent because the simple architecture of skipgram and BOW enable them to train on very large unlabeled corpora. Peters et al. (2018) introduced deep contextualized word representations, where the concatenation of the vectors from bidirectional LSTMs trained with a two-way language model on a large unlabeled corpus are used as word representations. The Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019) trains a unidirectional language model on an unlabeled text by using a "masked language" model (trying to predict randomly masked words in a sentence) and predicting the next sentence using a deep bidirectional transformer.

To be able to build state-of-the-art systems based

on end-to-end neural approaches, it is necessary first to create large labeled datasets. Thus, in many languages and particular domains, such approaches are difficult to use. Recently, a large labeled NER dataset was released for Norwegian, extending the Norwegian dependency treebank (Jørgensen et al., 2020). This dataset makes it possible to train neural NER models for the Norwegian languages Bokmål and Nynorsk.

Traditionally, a natural language model is trained on one corpus and is used only for one language. The NorNE paper (Jørgensen et al., 2020) contains preliminary experiments on creating a joint model with both Nynorsk and Bokmål using the NCRF++-toolkit (Yang and Zhang, 2018). Here, they trained new word embeddings on a joint corpus and used these embeddings as the embeddings for the joint training on the two NorNE-corpora. Moon et al. (2019) suggests that we can create a "language-independent" model for NER by using pre-trained multilingual word representations such as the multilingual BERT (Google-Research). Hakimov et al. (2018), on the contrary, create a multilingual model for Question Answering by first learning how to map the universal syntax dependency representation of the natural language text into a logical form.

We investigate to what extent we can leverage the multilingual BERT to train a combined NER model for the NorNE Nynorsk and Bokmål corpora. In addition, we train on the English CoNLL 2003 NER corpus (Sang and De Meulder, 2003) and see if this further improves the performance of the model and, at the same time, is able to make predictions for English.

The evaluation of NER can be challenging, especially on neural models (Yang et al., 2018). The use of different datasets with different complexity will give different results and tuning hyperparameters for a model can strongly affect the performance and make them hard to compare. To make it even more challenging, the methods authors use for evaluating NER performance differ when dealing with partial matches (i.e., does one consider a partly matching entity correct or incorrect?).

2 Related work

Moon et al. (2019) investigate the use of the multilingual BERT (Google-Research) model for NER and show that the same model trained on multiple languages perform better than the model trained on only one language. They also show that a BERT-embeddings model trained on multiple languages can be fine-tuned to NER for one language and make zero-shot predictions for another language. The authors also demonstrate that a multilingual BERT model that is fine-tuned on one language corpus can "lose" zero-shot performance on other languages when trained for many epochs.

NorNE (Jørgensen et al., 2020), the first publicly available annotated corpus for named entities in Norwegian, is based on the existing Norwegian Dependency Treebank. There are two different corpora, one for each of the two official languages in Norway (i.e., Nynorsk and Bokmål). The paper includes some preliminary results of training a classifier on these two corpora. Most notably, Jørgensen et al. (2020) report that a model trained jointly on both the corpora outperforms the models trained on either one of them in isolation. This result may come as a surprise given that the two languages are, lexically, quite different. The authors also explore the impact of combining some of the categories (i.e., reducing the size of the tagset) and find that combining the *GPE_ORG* and *GPE_LOC* into one more general category *GPE* (the NorNE 7 tagset) substantially improves the performance of the trained model. Removing the category *GPE* entirely and classifying the *GPE_ORG* and *GPE_LOC* as *ORG* and *LOC* (the NorNE 6 tagset), however, give only a slight improvement in performance.

Joint training of the two Norwegian languages has been shown to be beneficial also in other applications. Velldal (2017) demonstrated that while cross-language performance is poor, combining the training data from Bokmål and Nynorsk improve performance for both languages on multiple taggers and parsers.

3 Methods

3.1 Datasets

The NorNE corpus[1] consists of two separate corpora, one for Bokmål and one for Nynorsk. The two corpora do not contain the same texts but are the result of two separate data collection efforts. The distribution of the named entity categories also differs from one corpus to another. The text corpora are the same as the Norwegian Dependency Treebank (NDT), containing about 300.000 tokens from different non-fictional genres. Both corpora come with train, development, and test splits (80-10-10) and are available in the CONLL-U format.

[1] https://github.com/ltgoslo/norne

NorNE uses the IOB2 labeling scheme and uses six different main categories of named entities PER, ORG, LOC, GPE, PROD, and EVT. In addition, it has a special category for nominals that are derived from proper names (DRV), and the GPE category is further divided into GPE_LOC and GPE_ORG, where any GPE entity must be one of the two. The training corpus also includes some occurrences of the MISC. This label is not considered part of the NorNE corpus since it has very few occurrences.

The English CoNLL 2003 corpus (Sang and De Meulder, 2003) comes, as the NorNE corpus, with train, development, and test splits. The text consists of Reuters news stories from between 1996 and 1997. The version used here[2] also uses the IOB2 labeling scheme. CoNLL 2003 does not use the same categories as NorNE. The corpus has four categories of named entities PER, ORG, LOC, and MISC. The CoNLL 2003 corpus comes on a format with one word per line followed by its POS tag, chunk tag, and NER tag.

To use the NorNE corpus, we converted it from the original CONLL-U format to a tab-separated document containing the complete sentence in one column and the complete sequence of NER-labels corresponding to that sentence. This conversion was done by a custom script. The training data from NorNE is then used as-is without any further preprocessing. Similarly, the English CoNLL 2003 corpus was converted from the distributed format into the same tab-separated format suitable for loading the entire sentence together with the entire sequence of NER tags. This was also done by a custom script. No further preprocessing was done on the CoNLL 2003 corpus either. Having all the documents in the same format, to combine the training data from multiple corpora, it suffices to concatenate the documents. All datasets, use the same IOB2-scheme and can be combined without difficulty.

To do joint training with the NorNE and the CoNLL 2003 corpora, we have to consider the differences in the labeling schemes. For the purpose of the experiments with the full NorNE tagset, we keep the complete set of NorNE categories and ignore the MISC category for all datasets (as this category is not supported in NorNE). This categorization means that the categories that are not found in CoNLL 2003 corpus are expected to have fewer examples overall. Since the CoNLL 2003-corpus has a smaller tagset than the NorNE-corpus, we hypothesize that a reduction in the number of tags of the NorNE-corpus, thus creating more overlap with the tags of the CoNLL 2003-tagset, can improve performance. Consequently, we also did experiments on a reduced version of the NorNE corpora (NorNE 6). The GPE_ORG GPE_LOC tags of the NorNE-corpora were reduced to the ORG and LOC respectively by regex substitution on the original corpora.

To say something about the similarity of the training and test data, we evaluate the "lexical overlap" of the test-sets with the respective training-sets. We extract the sentences from each corpus. The sentences are split into individual words, which are case normalized. Punctuations and words containing non-alphabetic characters are not considered. We then take the (set-) intersection between the two corpora and divide the cardinality of the intersection-set with the cardinality of the test-set. More formally, we calculate the lexical overlap $\mathcal{O}$ between the set of words in a training set T and the set of words in the test set E as:

$$\mathcal{O} = \frac{|T \cap E|}{|E|} \tag{1}$$

3.2 The model

For the implementation of the NER-tagger, we used PyTorch and the Hugging Face implementation of BERT (Wolf et al., 2020). The `bert-base-multilingual-cased` ($\text{Bert}_{\text{MULTILINGUAL}}$) pre-trained model was used in all experiments. This BERT model is the recommended multilingual model by Google-Research, and it is trained on the 104 most common languages. The model uses 12 layers (Transformer Blocks), 768 hidden units and 12 attention heads. The architecture is the same as for the English BERT model, but it is trained on a dump of the Wikipedia for each language. In addition, it applies some weighting of the training data to account for the huge difference in the size of different language's Wikipedia dumps. The set of languages include both the Norwegian languages (i.e., Bokmål Nynorsk) and English (Google-Research).

The extracted sentences are passed to the BERT model, which returns contextualized word piece embeddings. In order to preserve the alignment between word piece embeddings and NER labels,

[2] https://github.com/davidsbatista/NER-datasets/tree/master/CONLL2003

extra labels, equal to that of the original word, were added to each word piece. A linear layer for the token classification is placed on top of the hidden-states output of the BERT layer. This layer has a number of outputs equal to the number of tags we consider. The overall structure of fine-tuning BERT on named entity recognition is shown in Figure 1 where E_i is an input embedding and T_i a representation of the NER token.

4 Experiments

All the parameters were kept close to the values suggested in (Devlin et al., 2019). For training, we used the AdamW optimizer in the Hugging Face transformers library with a L2 weight decay of 0.1 (except for the bias, gamma and beta which should not have weight decay), a learning rate of 3e-5 and an epsilon value of 1e-8. The β_1 was set to 0.1 and β_2 to 0.999. Also, a scheduler is used to reduce learning rate linearly, and experiments were done using a batch size of 32.

The evaluation was done using the SemEval 2013 task 9.1 reimplemented by David S. Batista[3]. For all experiments, the F1 score is presented for the exact type and boundary match of the entities (strict evaluation according to Segura Bedmar et al. (2013)).

4.1 Single language and cross-language validation

All the single language experiments were trained on the train-fraction of the NorNE and CoNLL 2003 corpora. We trained one model on each of the three corpora using the same settings, and the test-split of each corpus was used to test each of the three models. We tested first with the full range of NorNE labels, and the experiments were repeated with the NorNE 6 tagset. The training examples were shuffled before each epoch. All experiments were done with 4 epochs and with 40 epochs.

4.2 Multi-language training

Where the training includes multiple corpora, the training corpus of the different corpora were concatenated into a single corpus. All the training examples were shuffled between epochs and all experiments were done both with 4 and 40 epochs. All corpora are approximately the same size, so there should be no need to do any normalization. As for the single language and cross-language validation experiments, all experiments were repeated using the NorNE 6 tag-set.

4.3 F1 and loss over epochs

We also monitored the tag-based F1-value and loss for the number of epochs both on Bokmål and Nynorsk. For this experiment, we let the training continue for 40 (20) epochs and evaluate the performance on the development test sets. The F1 score that is reported is, as for the other experiments, on strict matches.

5 Results

5.1 The full NorNE tagset

In Table 1, we present the results of the experiments done with the full NorNE tagset (without the `MISC`) with 4 epochs. The same experiments with 40 epochs are shown in 2. All results are presented as F1-scores on strict matches. We denote the training sets as follows Bokmål (bm), Nynorsk (nn), English CoNLL 2003 (en). The combined training partition of Bokmål and Nynorsk is denoted (nn-bm) and the training set consisting of all the three corpora (en-nn-bm). We observe that the cross-language performance between Nynorsk and Bokmål, while not perfect, is quite good when trained with 4 epochs. The performance-drop when testing with Nynorsk on a Bokmål model is about 7.5 % while the other way, the performance dropped only 0.13 %. Jørgensen et al. (2020) report 8.0 % and 2.9 % performance drop, respectively. When trained on 40 epochs, the performance drop between the two Norwegian languages increases to 3.7 % and 8.7 %.

When trained for 4 epochs, we observe a general trend of improved performance for all test-languages for each additional language that is used to train the model. This effect is most notable when testing Bokmål and Nynorsk on the model of the two languages combined. The performance is improved for CoNLL 2003 by adding the Norwegian languages. The exception is Bokmål that benefits from being trained together with Nynorsk, but show a decrease in performance after the inclusion of the CoNLL 2003 corpus. When trained on 40 epochs, however, the benefit of adding the other languages corpora is less pronounced. The best performance for Nynorsk and CoNLL 2003 increase, but very little. The best performance for the Bokmål actually decreases when trained together

[3] https://github.com/davidsbatista/NER-Evaluation

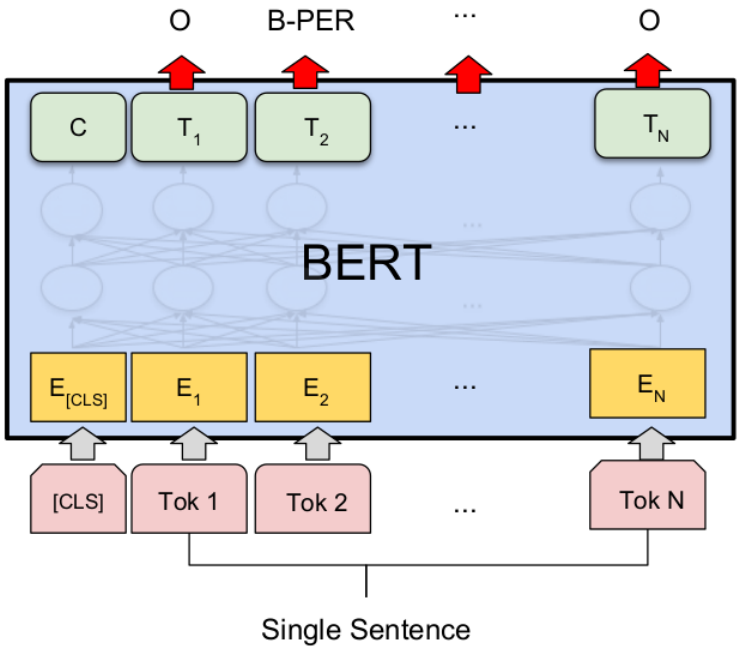


Figure 1: Finetuning BERT for NER (from Devlin et al. (2019))

with Nynorsk and the inclusion of the CoNLL 2003 corpus to the training degrades the performance for both Nynorsk and Bokmål.

training	BM	NN	CoNLL 2003
en-nn-bm	0.810	**0.851**	**0.917**
nn-bm	**0.818**	0.842	0.447
bm	0.784	0.783	0.442
nn	0.762	0.822	0.415
conll	0.569	0.563	0.902

Table 1: F1-scores using the full NorNE tagset with 4 epochs

training	BM	NN	CoNLL 2003
en-nn-bm	0.787	0.845	**0.921**
nn-bm	0.802	**0.859**	0.427
bm	**0.811**	0.781	0.424
nn	0.776	0.850	0.433
conll	0.557	0.580	0.910

Table 2: F1-scores using the full NorNE tagset with 40 epochs

5.2 The NorNE 6 tagset

The results of the experiments with the NorNE 6 tagset using 4 epochs are found in Table 3. To avoid confusion with the experiments with the full tagset, we denote the training data used in these experiments bm6, nn6, nn-bm6 and en-nn-bm6.

When trained only on the CoNLL 2003 corpus, we already achieve a F1-score of 0.726 on the Nynorsk test and 0.688 on the Bokmål test. The performance of the CoNLL 2003 test on the Bokmål and Nynorsk also improve. We see that both Bokmål and Nynorsk benefit from each other as on the full NorNE tagset. The inclusion of CoNLL 2003, improve Nynorsk performance slightly, but has no effect on the performance on the Bokmål test set. The performance does not, however, drop on the Bokmål by including CoNLL 2003 as it does when training on the full tagset. Overall, all languages benefit from the inclusion of all the other languages, thus the corpus with all three languages show the best performance for all three. A quite surprising finding is that we get better performance on the Nynorsk test corpus when trained with Bokmål than what we get with the Bokmål test set. The performance of the Bokmål on a model trained with Nynorsk, on the other hand, drops 2.77 %.

With 40 epochs, both Bokmål and CoNLL 2003 show the best performance when trained alone. Nynorsk, however, benefits from joint training both with Bokmål and CoNLL 2003.

training	BM	NN	CoNLL 2003
en-nn-bm6	**0.818**	**0.874**	**0.912**
nn-bm6	**0.818**	0.873	0.571
bm6	0.816	*0.823*	0.578
nn6	0.806	0.829	0.554
CoNLL 2003	0.688	0.726	0.908

Table 3: F1-scores using the NorNE 6 tagset with 4 epochs

training	BM	NN	CoNLL 2003
en-nn-bm6	0.799	**0.871**	0.914
nn-bm6	0.812	0.865	0.579
bm6	**0.825**	0.823	0.533
nn6	0.788	0.849	0.524
CoNLL 2003	0.668	0.724	**0.918**

Table 4: F1-scores using the NorNE 6 tagset with 40 epochs

5.3 F1 and loss with epochs

We measured the effect of the number of epochs on the F1-value of the validation dataset. The results of testing for 40 epochs both on the Bokmål and the Nynorsk datasets are shown in Figure 2. We observe that already after the first epoch (where the value is about 0.8), most of the fine-tuning is already done, the increase in performance is, however, substantial up to about 4 epochs, then the improvement is almost linear, but with quite some variation. It is, however, possible to improve the F1-score on the validation set all the way up to 40 epochs. Figure 3 show how the F1 score varies for the CoNLL 2003 tested on the Bokmål corpus. We observe that the training does influence performance on a quite different task. Both different languages and different tagset. However, after about 15 epochs, it has no effect to keep training.

The effect on the loss on the set of training for additional epochs is shown in Figure 4. The y-axis is logarithmic, so we observe an approximately exponential decrease in loss over epochs. We observe that even after 20 epochs, the loss is still decreasing, but the decrease is very small compared to the first epochs.

5.4 Lexical overlap

On evaluating the lexical overlap between a train-set and a test-set, we observe that there is more overlap between the Bokmål test-set and the Nynorsk

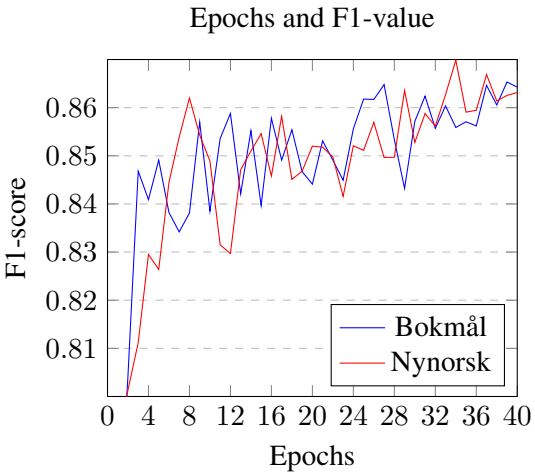


Figure 2: Effect of increased epoch count on F1-value

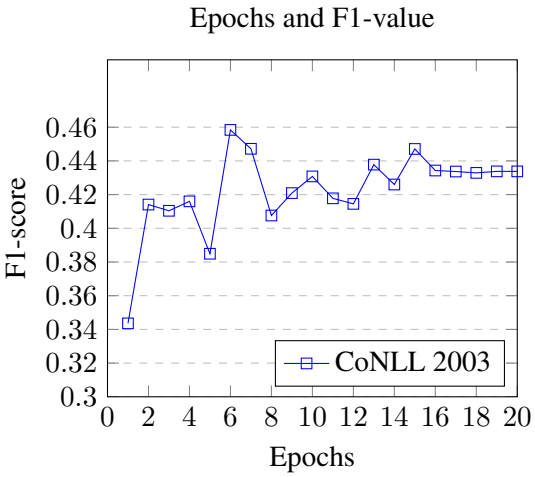


Figure 3: Effect of increased epoch count on F1-value when testing wth the CoNLL 2003 on a model trained on Bokmål corpus

train-set than the other way around. We also observe that the Nynorsk test-set has more in common with the CoNLL 2003 train-set, and the CoNLL 2003 test-set has most in common with the Bokmål train-set. The lexical overlaps are reported in Table 5.

5.5 Comparison with benchmarks

In Table 7, we compare our results with some benchmarks. The $\text{Bert}_{\text{LARGE}}$ (Devlin et al., 2019), which is trained on the English Wikipedian and the Book corpus by Zhu et al. (2015), obtain the best score for the CoNLL 2003 corpus. Our model is close, but to give a better impression of the performance of the model, we replicate the study using the $\text{Bert}_{\text{LARGE}}$-model on the CoNLL 2003-corpus with our implementation obtaining a slightly lower

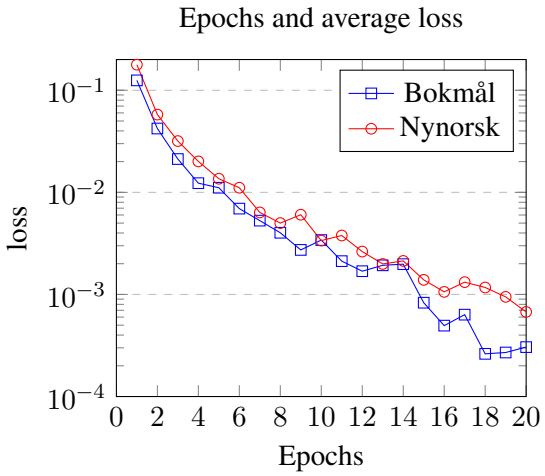


Figure 4: Effect of increased epoch count on loss (semi-log scale)

Trainset	BM	NN	CoNLL 2003
bm	*0.655*	0.395	0.139
nn	0.415	*0.661*	0.103
CoNLL 2003	0.056	0.063	*0.681*

Table 5: The degree of lexical overlap between test-sets and train-sets accross languages

F1 score than Devlin et al. (2019). Interestingly, the score obtained using the $\text{BERT}_{\text{LARGE}}$ fine-tuned on the CoNLL 2003-corpus is equal to what we observe using the $\text{BERT}_{\text{MULTILINGUAL}}$ trained on all three corpora.

We replicate some of our own experiments using the $\text{Bert}_{\text{LARGE}}$, and get 30.1 as a zero-shot on the NorNE Bokmål test set. Fine-tuning the $\text{Bert}_{\text{LARGE}}$-model on the en-nn-bm dataset for four epochs, we get 72.4. This is substantially lower than the results presented in Sections 5.1 and 5.2. Another interesting observation is that we get better performance than the results by Jørgensen et al. (2020) on the Nynorsk corpus, but not so on the Bokmål corpus.

Furhter, we include experiments where we only do training on the linear layer of the NER classifier (i.e., we do not do fine-tuning of the Bert-embeddings). The complete results of these experiments are found in Table 6. After four epochs, the model only got a F1-score of 0.025 on the Bokmål test-set. And after 50 epochs, we got a F-score of 0.580.

It is, however, challenging to compare directly with literature because the purpose of our study is not to obtain the best possible fit for one individual language, but rather investigate if good performance can be obtained for multiple languages by a single model. We ignore, for instance, the *MISC*-category on the CoNLL 2003 corpus to adapt to the NorNE tagset.

Epochs	BM	NN	CoNLL 2003
4	0.025	0.027	0.035
50	0.580	0.628	0.591

Table 6: F1-socres without fine-tuining the BERT-embeddings. Trained on the en-nn-bm6 corpus.

6 Discussion

We observe the same effect as Moon et al. (2019), that training on multiple languages can improve overall performance for the languages involved. In our experiments, training with three corpora (i.e., Nynorsk, Bokmål and CoNLL 2003), not only achieves the best average performance on the training set of all the languages, but also achieve a performance that is better than, or at least comparable to, the best performing model for each individual language. The benefit seems most expressed among similar languages such as Bokmål and Nynorsk. Using this architecture, we can train a joint NER tagger on Nynorsk and Bokmål that has about the same performance for both languages as a model specifically trained on one of them. The joint training improves the Nynorsk model in all cases, but the effect is not so pronounced for the Bokmål model. Adding another, different, language can also improve the model's performance, and it is quite possible to train a model to make predictions on all three languages with a performance comparable to the best performance on each individual language.

While the first 4 epochs give the bulk of the improvement in the performance of the system, there is still some performance to gain by keep training. The catch is that the model becomes less general. It may be that the model with the three languages would be able to perform better with even less training.

Using the reduced tagset, NorNE 6, improves cross-train performance. It improves upon the best performance for Bokmål and Nynorsk, but the performance on the CoNLL 2003 drops slightly. The zero-shot performance on Nynorsk and Bokmål using a model trained only on the CoNLL 2003 corpus is quite impressive and shows that the

Model	Train	Test	Test F1
$\text{Bert}_{\text{LARGE}}$ [1]	CoNLL 2003	CoNLL 2003	0.928
$\text{Bert}_{\text{LARGE}}$ [2]	CoNLL 2003	CoNLL 2003	0.912
$\text{Bert}_{\text{LARGE}}$ [2]	CoNLL 2003	$\text{NorNE}_{\text{Bokmål}}$	0.301
$\text{Bert}_{\text{LARGE}}$ [2]	en-nn-bm	$\text{NorNE}_{\text{Bokmål}}$	0.724
CharCNN + LSTM + CRF [3]	$\text{NorNE}_{\text{Bokmål}}$	$\text{NorNE}_{\text{Bokmål}}$	0.839
CharCNN + LSTM + CRF [3]	$\text{NorNE}_{\text{Nynorsk}}$	$\text{NorNE}_{\text{Nynorsk}}$	0.839
CharCNN + LSTM + CRF [3]	$\text{NorNE}_{\text{Bokmål + Nynorsk}}$	$\text{NorNE}_{\text{Bokmål}}$	0.835
CharCNN + LSTM + CRF [3]	$\text{NorNE}_{\text{Bokmål + Nynorsk}}$	$\text{NorNE}_{\text{Nynorsk}}$	0.853
$\text{BERT}_{\text{MULTILINGUAL}}$ no fine-tuning of BERT (4 epohcs)	en-nn-bm6	$\text{NorNE 6}_{\text{Bokmål}}$	0.025
$\text{BERT}_{\text{MULTILINGUAL}}$ no fine-tuning of BERT (50 epohcs)	en-nn-bm6	$\text{NorNE 6}_{\text{Bokmål}}$	0.580
$\text{BERT}_{\text{MULTILINGUAL}}$ [4]	en-nn-bm6	$\text{NorNE 6}_{\text{Bokmål}}$	0.818
$\text{BERT}_{\text{MULTILINGUAL}}$ [4]	en-nn-bm6	$\text{NorNE 6}_{\text{Nynorsk}}$	0.874
$\text{BERT}_{\text{MULTILINGUAL}}$ [4]	en-nn-bm6	CoNLL 2003	0.912

[1] Devlin et al. (2019)
[2] Replicated on our implementation with $\text{BERT}_{\text{LARGE}}$, trained for 4 epochs, ignoring MISC labels
[3] Jørgensen et al. (2020) on the NorNE Heldout (test)
[4] Our model trained for 4 epochs

Table 7: Comparing our results with relevant baselines

$\text{BERT}_{\text{MULTILINGUAL}}$ model is able to learn about the task independent of the language used.

The Nynorsk test-set shows less overlap with the Bokmål train-set than the Bokmål test-set on the Nynorsk train-set. Based on this finding, we would expect the training on Nynorsk to help Bokmål more than is observed in this study. The differences in lexical overlap are not significant between the Bokmål and Nynorsk against the CoNLL 2003 training data. Thus, it is somewhat surprising that the Nynorsk benefits much more from the CoNLL 2003 corpus than does Bokmål. We can therefore not conclude that the benefits of joint training is due to lexical overlap of the test-set in one language and the test-set in another language. There must be other reasons for the unequal benefits of using Bokmål training data on the Nynorsk and using CoNLL 2003 training data on the two Norwegian languages. The BERT-model does not consider words as the atomic, but rather as word parts, so further evaluation about the similarity of the copora using word-parts instead of words could give insight into why the bert model perform better accross some languages than others.

For future work on this line, we would like to tweak the hyperparameters to improve the performance of the BERT-model fine-tuned to both the Norwegian languages (and possibly other languages). Another idea is to investigate the performance of a Norwgegian BERT-model that is trained on a more comprehensive Norwegian corpus rather than just the Wikipedia. It would also be interesting to train with a third language that has a classification scheme more similar to the NorNE corpus. Training with a Swedish and Danish corpus could also prove beneficial. We would also like to see furhter investigations on how language similarities affect cross-training with $\text{Bert}_{\text{MULTILINGUAL}}$.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *eprint arXiv:1810.04805*.

Google-Research. Google-research/bert. `https://github.com/google-research/bert`, visited on 2020-05-20.

Sherzod Hakimov, Soufian Jebbara, and Philipp Cimiano. 2018. AMUSE: Multilingual Semantic Parsing for Question Answering over Linked Data. *eprint arXiv:1802.09296*.

Fredrik Jørgensen, Tobias Aasmoe, Anne-Stine Ruud Husevåg, Lilja Øvrelid, and Erik Velldal. 2020. NorNE: Annotating Named Entities for Norwegian. *eprint arXiv:1911.12146*.

Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. 2020. A Survey on Deep Learning for Named Entity Recognition. *eprint arXiv:1812.09449*.

Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. 2011. DBpedia spotlight: Shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems - I-Semantics '11*, pages 1–8. ACM Press.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *eprint arXiv:1301.3781*.

Taesun Moon, Parul Awasthy, Jian Ni, and Radu Florian. 2019. Towards Lingua Franca Named Entity Recognition with BERT. *eprint arXiv:1912.01389*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. *eprint arXiv:0305050*.

Isabel Segura Bedmar, Paloma Martínez, and María Herrero Zazo. 2013. SemEval-2013 Task 9 : Extraction of Drug-Drug Interactions from Biomedical Texts (DDIExtraction 2013). Association for Computational Linguistics.

Erik Velldal. 2017. Joint UD Parsing of Norwegian Bokmal and Nynorsk. In *Proceedings of the 21st Nordic Conference on Computational Linguistics, NoDaLiDa, 22-24 May 2017, Gothenburg, Sweden*, 131, pages 1–10.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2020. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *eprint arXiv:1910.03771*.

Jie Yang, Shuailong Liang, and Yue Zhang. 2018. Design Challenges and Misconceptions in Neural Sequence Labeling. *eprint arXiv:1806.04470*.

Jie Yang and Yue Zhang. 2018. NCRF++: An Open-source Neural Sequence Labeling Toolkit. *eprint arXiv:1806.05626*.

Yukun Zhu, Ryan Kiros, et al. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27.

How Model Architecture and Language Variation Affect NER Model Performance

Zhenying Wu
Department of Informatics, UiO / Gaustadalléen 23B, 0373 Oslo
zhenyinw@ifi.uio.no

Abstract

This paper documents the experiments with altering Neural Sequence Labelling model architecture and training with different corpora (including machine translation) and discusses the observations. The model used in this paper is NCRF++, which is a state-of-the-art neural sequence labeling toolkit built on PyTorch. The corpora used are from Universal Dependency (UD) version of NorNE, in two forms of Norwegian written languages: Bokmål and Nynorsk. Several experiments are carried out to see how word embedding, word sequence model and character sequence model influences model performance. It also includes another experiment of training Named Entity Recognition (NER) model on different corpus in Bokmål, Nynorsk and machine translation, as well as an analysis of their impact.

1 Introduction

Named Entity Recognition (NER) is the task of locating named entities in text and classifying them into pre-defined categories such as names of persons, locations, products, expressions of times, events, etc. NER can be used to answer many real-world questions, such as:

- Which event were mentioned in the news article?
- Which city does the event occur? Who organized?
- Does the google review contains product name? Is the reviewer's name given?

In the previous work by Jørgensen et al. (2020), the first publicly available Norwegian named entities (NorNE) corpus has been created. This paper used NorNE dataset for both Bokmål (BM) and Nynorsk (NN) -two variations of written Norwegian- combined with NCRF++ toolkit from Yang and Zhang (2018) to explore how model architecture and training corpus variation affect model performance.

In this paper, we first give a brief introduction of the corpus that we used in this paper in Section 2. And in Section 3, we highlight the structure of the Neural Sequence Labeling model that we are going to use. Machine translation which has been used in this paper is briefly explained in Section 4. Section 5 and 6 sums up the experimental set up and obtained results for each set of experiments.

2 NorNE Dataset

NorNE [1] extends the annotation of the existing Norwegian Dependency Treebank has been manually annotated by two annotators. The corpus contains both Bokmål and Nynorsk and has a rich set of entity types such as persons, organizations, locations, events and products. In the paper by Jørgensen et al. (2020), they have presented their guidelines of annotation, annotation effort and also an analysis of the corpus applying a neural sequence labeling architecture.

Corpora for both Bokmål (BM) and Nynorsk (NN) consist of train, development, and test dataset. Corpora of the two different languages contain different contents, instead of translations. According to Velldal et al. (2017), the extent of lexical overlapping between the two training sets is estimated to be around 61.6%.

It should be noted that the entity distributions (as shown in Table 1 and 2) are slightly different from what has been documented in (Jørgensen et al., 2020), which can be caused by addition to the corpora after the publish of the paper. Besides, additional MISC type has been included in the comparison. In general, corpora sizes and entity distributions for BM and NN corpus are very

[1] https://github.com/ltgoslo/norne/tree/master/ud

Types	Train	Dev	Test	Total	%
PER	5223	837	840	6900	40.11
ORG	3202	475	388	4065	23.63
GPE_LOC	1983	258	312	2553	14.84
PROD	1068	302	89	1459	8.48
LOC	633	120	159	912	5.31
GPE_ORG	321	55	51	427	2.48
DRV	525	110	50	685	3.98
EVT	176	13	6	195	1.14
MISC	7	0	0	7	0.01

Table 1: Entity distributions for Bokmål corpus (UD) in NorNE

Types	Train	Dev	Test	Total	%
PER	6599	721	626	7946	40.90
ORG	3889	381	376	4646	23.92
GPE_LOC	2122	205	195	2522	12.99
PROD	1371	186	159	1716	8.84
LOC	1060	107	111	1278	6.58
GPE_ORG	394	75	17	486	2.50
DRV	473	61	36	570	2.93
EVT	221	10	12	243	1.25
MISC	6	0	14	20	0.01

Table 2: Entity distributions for Nynorsk corpus (UD) in NorNE

similar.

3 NCRF++

We used a toolkit NCRF++ from Yang and Zhang (2018) for modeling. It is a configurable sequence labeling tool built on PyTorch. For familiarization, the framework of NCRF++ is shown in Figure 1.

3.1 NCRF++ architecture

Three layers are incorporated and they are: a character-level sequence layer; a word-level sequence layer and inference layer. Character representations can be taken in as an input feature to join word representation as a part of input to word sequence layer. Sentence level features are then extracted and fed into the inference layer to generate a label for each word. In this paper, several sets of experiments were carried out to see the influence of architecture on model performance. These include word embedding dimensions, the model we use for character-level representations, as well as numbers of layers of word level model. Besides, a comparison of model performance of models trained on different variation of corpus has been also explored in the next section. It is worth mentioning that there has already been similar exploration of the influence of network settings, from the paper *Design Challenges and Misconceptions in Neural Sequence Labelling* by Yang et al. (2018).

3.2 Data format conversion

The required format of data input is .bmes, while the annotations in the UD version of NorNE is .CoNLL-U. To convert .CoNLL-U files to .bmes, we have used a CoNLL-U Parser[2].

4 Machine Translation

Among the experiments carried out by Jørgensen et al. (2020), joint modeling of Bokmål and Nynorsk has been explored and discussed. They have investigated effects of applying parsing models across Bokmål and Nynorsk dataset and combined dataset. For example, training a model on Nynorsk and applying to Bokmål, and vice versa.

It is important to be aware that Bokmål and Nynorsk are both used in Norway as two written languages, whereas Bokmål is used by majority and Nynorsk is used by 15% of the Norwegian population. The two languages are very closely related: the word order is identical and many words are clearly related or even identical.

Despite similarity between Bokmål and Nynorsk, it is interesting to train a model on machine-translated Nynorsk (denoted as OVS) and applying to evaluation dataset.

Machine translation used in this study is provided by Apertium (Forcada et al., 2011) [3]. Here is one example of machine translation with Apertium:

- NN: Etter reaksjonane hennar på Sogn (B-GPE_ORG) og (IGPE_ORG) Fjordanes (I-GPE_ORG) bunadsbidrag til kongens bursdag i 2007, fekk Viken (BPER) mykje mediemerksemd.

- OVS: Etter reaksjonene hennes på Sogn (B-GPE_ORG) og (IGPE_ORG) Fjordanes (I-GPE_ORG) bunadsbidrag til kongens bursdag i 2007, fikk Viken (BPER) mye medieoppmerksomhet.

[2]https://pypi.org/project/conllu/
[3]https://www.apertium.org/index.nob.html

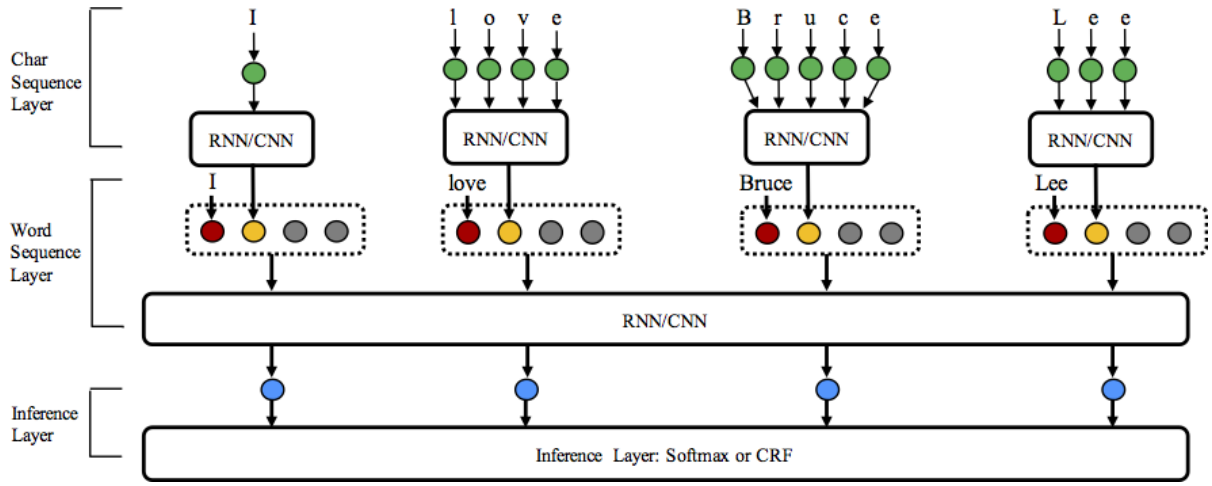


Figure 1: An illustration of NCRF++ architecture. Source: Yang and Zhang (2018)

- English: Viken got lots of media attention after her reactions on Sogn og Fjordanes bunad contribution to the king's birthday in 2017.

Browsing through the machine translation, one can find it is quite accurate. Therefore, we have translated the Nynorsk tree banks with Apertium and use it as a third tree bank besides Bokmål and Nynorsk tree banks from NorNE.

4.1 Suggestion for usage of Apertium

In general, Apertium is a rather efficient machine translation tool. However, in our experience, it is better to limit the data file size to be under 3 MB. Otherwise, Apertium is not going to generate machine translation, neither giving any detailed failure message.

Therefore, in some cases (especially for training dataset), it is necessary to split them in chunks and feed them to Apertium.

5 Experiment Setup

The model used is based on NCRF++ (Yang and Zhang, 2018), a modular sequence labelling toolkit built on PyTorch. Similar to the setup used in (Jørgensen et al., 2020), the benchmark model combines a character-level CNN and a word-level BiLSTM, and at the end feeding into a CRF inference layer. Concatenation of the character sequence representations from the CNN using max-pooling and random (untrained) word embeddings are the input to the word-level BiLSTM. The label mapping used in our experiments is the full set of 8 entity types as below. You can find more details about annotation rules and guidelines followed by annotators in (Jørgensen et al., 2020).

- PER - Person names
- ORG - Organization
- LOC - Location
- GPE_LOC - Geo-political entity: Location
- GPE_ORG - Geo-political entity: Organization
- PROD - Product
- EVT - Event
- DRV - Derived

The annotations of NorNE used the standard IOB2 scheme. The B-label is used at the beginning of every named entity, no matter of its span and the following entity type. It should be noted that same random seed for initializing the models has been used in all experimental runs, in order to reduce the effect of non-determinism. As for evaluation, a implementation provided by David S. Batista[4] has been used. The reported F1 score is based on strict match on the entity level (Segura-Bedmar et al., 2011), i.e., both the boundary and entity label are considered.

6 Experimental Results and Discussion

In this section, experimental results for named entity recognition using NorNE dataset are presented. We investigate the effects using different embedding dimensionalities, different inference layers, different numbers of model layers, as well as including or not including a

[4]https://github.com/davidsbatista/NER-Evaluation

Pre-trained	Dim.	F1
None(Jørgensen et al., 2020)	100	76.54
None	100	79.31
CBOW(Jørgensen et al., 2020)	100	84.36
None	300	78.67
None	600	79.10
SG(Jørgensen et al., 2020)	600	90.75

Table 3: Impact of word embedding dimensionality on the Bokmål development set of NorNE.

character-level model. Code used for this paper can be found at https://github.uio.no/zhenyinw/IN5550_exam.

6.1 Word embedding dimension

In this part, we evaluated the impact of word embedding dimensions on the model performance. The word embeddings used in our experiments are not pre-trained. The F1 score values are compared against different word embedding dimensionalities, as well as similar configurations from other papers.

First, we replicated the experiment from the Jørgensen et al. (2020), where there is no pre-trained word embedding plus word embedding dimension 100. Interestingly, we got a slightly higher F1 score 79.31 (weighted average) against theirs 76.54. This can be caused by some unknown potential differences in hyper parameters or evaluation settings. It is clear from their result that pre-trained embeddings improve model performance much. In this paper, we are not looking at pre-trained word presentation, but it is worthy investigation in future work.

On the other hand, increasing word embedding dimension does not lead to improvement of model performance. The F1 scores resulted from when word embedding dimensions are 100, 300 and 600 are very similar, with the best score achieved when embedding dimension is 100. This is potentially led by the limitation of the native corpus that the word embedding has been trained on.

6.2 Character embedding

NCRF++ provided by Yang and Zhang (2018) supports different types of neural encoders for character sequence information, such as LSTM, GRU and CNN. The RNN encoders are bidirectional and concatenates the final states of two RNNs as the encoder of the input character sequence. On the other hand, the CNN encoder takes a sliding window and then a max-pooling for aggregated encoding.

Character-level model	F1
None	69.40
CNN	79.10
LSTM	78.40
GRU	77.35

Table 4: Impact of character level model on the Bokmål development set of NorNE.

No. LSTM layers	F1
1	69.15
3	68.72
5	65.86

Table 5: Impact of numbers of LSTM layers on the Bokmål development set of NorNE.

The results from Table 4 reveal that enabling character level model helps a lot to improve model performance. This is mainly because for those words which are out of vocabulary (OOV), the character level model helps to predict their syntactic roles. Among all character-level model, CNN scores the highest, with slight advantage over LSTM and GRU. CNN has generally better performance in aggregating short-length spatial information than RNN models. This result coincides to the conclusion given in *Design Challenges and Misconceptions in Neural Sequence Labeling* by Yang et al. (2018) that character LSTM and CNN gives comparable results.

6.3 LSTM layers

Summing up the result in Table 5, one can tell that larger number of LSTM layers does not boost model performance and on the contrary, it degrades model performance. A sensible explanation is that model turns easily overfitted with stacking LSTM layers vertically.

6.4 Bokmål, Nynorsk and machine translation

Pre-trained embeddings are not considered in our study. Thus, embeddings in models trained with different tree banks are different from each other.

Experiments are carried in the way such an individual model is trained on Bokmål UD, Nynorsk UD, machine translation to Bokmål from Nynorsk

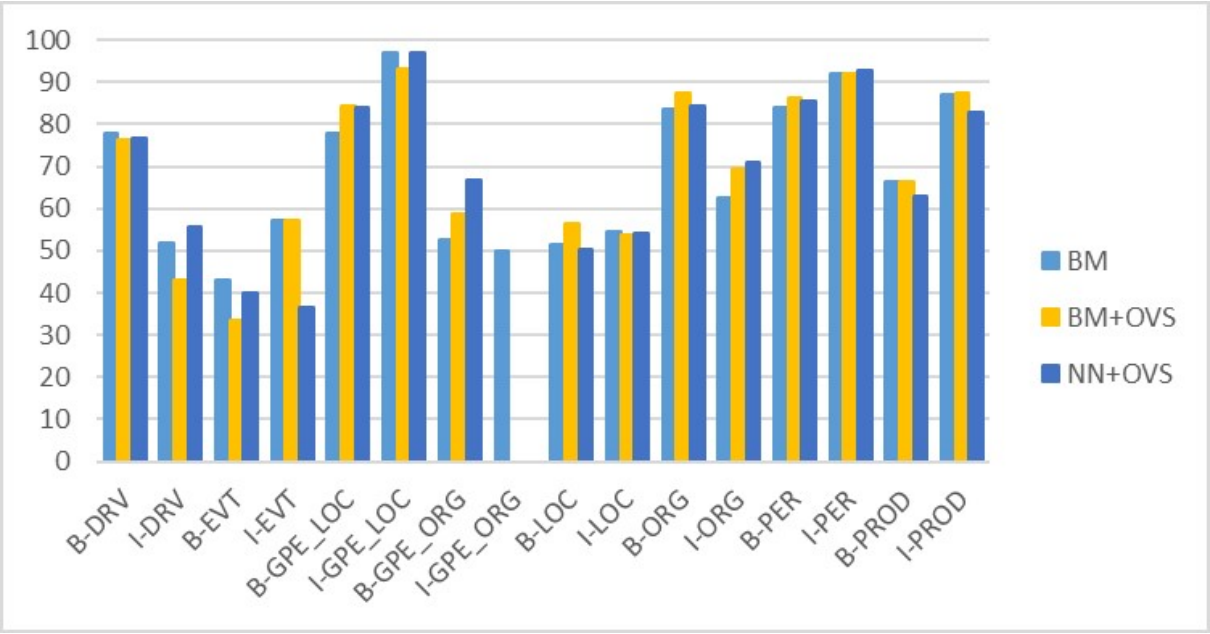


Figure 2: Comparison of F1 score at entity level, evaluated on Bokmål development set.

Train Config.	BM DEV	NN DEV
BM	79.10	66.31
NN	67.40	75.04
OVS	69.09	70.15
BM+OVS	**81.50**	72.55
BM+NN	80.61	**77.53**

Table 6: Impact of Bokmål vs. Nynorsk vs. machine translation and their combinations on the Bokmål and Nynorsk development set of NorNE.

Train Config.	BM Test	NN Test
BM	67.87	59.72
NN	62.41	67.44
OVS	62.54	62.70
BM+OVS	**72.16**	67.27
BM+NN	70.25	**71.30**

Table 7: Impact of Bokmål vs. Nynorsk vs. machine translation and their combinations on the Bokmål and Nynorsk held-out test set of NorNE.

UD, their combinations. Afterwards, the models are evaluated on Bokmål development/test and Nynorsk development/test dataset, respectively.

It can be clearly seen from Table 6 and Table 7 that model performance drops in the cross-language settings, i.e., train data on Bokmål dataset and evaluate on Nynorsk dataset, vice versa. Specifically, while the Bokmål NER model achieves an F1 score of 79.10 on Bokmål development data, the Nynorsk only got F1 score below 70.

Not surprisingly, 'Bokmål' (after machine translation from Nynorsk) NER model ends up with better performance than OVS and Nynorsk NER model when evaluating on Bokmål development dataset. On the one hand, this is sensible considering the fact that model trained on the same language should give better result as they should have learnt more about the word representation and word relation. On the other hand, the machine translated corpus does not show significant advantage over NN corpus.

The top three (3) models are the one trained on BM, BM+OVS and NN+OVS. Taking a closer look at the F1 score at entity-level, as shown in Figure 2. It is notable that BM+OVS has generally better or similar performance comparing with the other two models at most entities, but not I-DRV and B-EVT.

The difference of F1 scores is enlarged when we move on to evaluation on test dataset. Model trained on combination of BM and OVS outperforms others at Bokmål evaluation dataset, and the similar result applies to combination of BM and NN on Nynorsk evaluation set. One can see the benefit to train on combinations with machine translation or corpus with similar language traits. This sheds light on how machine translation could be used in NER and enhance model performance.

We would like to include machine translation

of NN and other language which is more different from BM or NN in future work, so that we can observe the effect of machine translation from another aspect and hopefully come to a more consolidated conclusion on the advantage of machine translation.

7 Summary

This paper has documented some experiments and experimental results with NCRF++ on NorNE. We have explored three model architecture design variations: word embedding dimension, character sequence representations, and numbers of LSTM layers for word sequence representations. Results show that enabling character sequence labeling helps to improve model performance by having access to character information, especially for words which are out of vocabulary. Without pre-training, word embedding dimension seems to not affect much on model performance. When it comes to the number of LSTM layers, it should be set to a reasonable value so that effective information are not lost. Stacking more layers vertically does not improve model performance but lead to certain performance degradation.

Besides, we trained models on different corpus with different language variations. Even though the cross-language performance is not optimal, the experiments indicate training on combination of similar language corpus and machine translation corpus give quite good performances across different evaluation set with different languages. This is a good indication about using machine translation to boost model training when the existing training corpus for certain language is scarce.

Acknowledgments

We thank the anonymous referees for their suggestions. This work has been part of the outcome of the course IN5550, and has been inspired and empowered by the devoted lecturers and teaching assistants. We would like to thank them for their good teaching and prompt help at most of times.

References

Mikel L. Forcada, Mireia Ginestí-Rosell, Jacob Nordfalk, Jim O'Regan, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Gema Ramírez-Sánchez, and Francis M. Tyers. 2011. Apertium: a free/open-source platform for rule-based machine translation. *Machine Translation*, 25(2):127–144.

Fredik Jørgensen, Tobias Aasmoe, Anne-Stine R. Husevåg, Lilja Øvrelid, and Velldal Erik. 2020. Norne: Annotating named entities for norwegian. In *Proceedings of the 12th Edition of the Language Resources and Evaluation Conference*.

I. Segura-Bedmar, P. Martínez, and D. Sánchez-Cisneros. 2011. The 1st ddiextraction-2011 challenge task: Extraction of drug-drug interactions from biomedical texts. *CEUR Workshop Proceedings Vol.761, pp.1-9*, 761:1–9.

Erik Velldal, Lilja Øvrelid, and Petter Hohle. 2017. Joint ud parsing of norwegian bokmål and nynorsk. In *Proceedings of the 58th Conference on Simulation and Modelling*.

Jie Yang, Shuailong Liang, and Yue Zhang. 2018. Design challenges and misconceptions in neural sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*.

Jie Yang and Yue Zhang. 2018. Ncrf++: An open-source neural sequence labeling toolkit. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics; System Demonstrations*.

A Replication Study in Negation Scope Resolution

Maja Buljan
Department of Informatics, University of Oslo
majabu@ifi.uio.no

Abstract

We conduct a replication study of the first successful implementation of bi-directional Recurrent Neural Networks on the task of negation scope resolution (NR). We perform variations on the basic network architecture and input data, and summarise empirical results, demonstrating a slight improvement over the original model. Additionally, we present a small-scale quantitative and qualitative error analysis, along with tentative conclusions about various challenging aspects of the NR task, and identify directions for future exploration.

1 Introduction and Background

In many downstream natural language processing (NLP) tasks, semantic representation at different levels has proven beneficial for building more successful models (Carreras and Màrquez, 2005; Shen and Lapata, 2007), and has seen ever-increasing interest over the past decades. Negation, being the aspect of language that reverses the truth value of utterances, is both an important element of natural language understanding, and a challenging linguistic phenomenon due to the multitude of ways in which it may be expressed (Morante and Blanco, 2012).

The focus of this submission (Oepen, 2020) is the task of negation scope resolution, *scope* being the part of the utterance that is negated (Pullum and Huddleston, 2002). It was one of the challenges at the *SEM 2012 shared task on negation scope and focus resolution (Morante and Blanco, 2012), which aimed to increase interest in this, at that point, somewhat neglected or superficially covered aspect of meaning.

In this paper, we look to the seminal work of Fancellu et al. (2016), who presented the first successful application of bi-directional Recurrent Neural Networks on this task, using data from the *SEM 2012 shared task. The authors argued that previous work on negation resolution, while overall high-performing on the aforementioned task and dataset, was highly feature-engineered and oftentimes relying on language-specific tools and heuristics (Read et al., 2012). Arguably, the RNN-based solution would be more portable across languages, and less dependant on domain-specific properties.

Our aim is to replicate their BiLSTM implementation of a scope-detection system, and build upon this basic architecture by exploring variations in internal representations and input formats. Furthermore, we introduce additional information sources and experiment with various flavours of input data. Partly inspired by (Lapponi et al., 2017) and (Fares et al., 2018), we attempt to augment the starting BiLSTM setup with additional layers of dependency parse information.

Ultimately, we find a combination that slightly exceeds the performance of the original BiLSTM system. Taking this improved model, we then perform an error analysis of the best model's output, considering properties of individual tokens and whether they correlate with negation resolution error rates, aiming to identify strengths and weaknesses in the model, as well as particular challenges in the task of scope resolution.

Section 2 frames the task, presents the data used for training and evaluation, and describes the original Fancellu et al. (2016) BiLSTM model. In Section 3, the results of our initial exploratory experiments are given, while Section 4 discusses variations and their effect on performance. Finally, in Section 5 we conduct a short quantitative and qualitative analysis of output errors, and conclude with Section 6. The most salient experimental results are highlighted in the body of the paper; however, in the interest of completeness, but also spatial efficiency, full experimental results are provided in Appendix A.

2 Replication Study

This section begins with a brief review of the *SEM negation scope resolution task, and the provided training and evaluation data (Morante and Blanco, 2012). In 2.2, we introduce the Fancellu et al. (2016) BiLSTM model.

2.1 Task definition and *SEM data

Of the several tasks proposed by the *SEM 2012 shared task on negation resolution (Morante and Blanco, 2012), we focus on the subtask of *negation scope resolution*. Given information about negation existing in a sentence, the task is to determine which tokens are affected by this negation. This information is annotated with two markers: (1) negation *cues*, which indicate a reversal of truth-value, and (2) the corresponding negation *scopes*. Example 1 below, taken from (Oepen, 2020) and sourced from the *SEM dataset, illustrates this annotation scheme:

(1) Mr. Sherlock Holmes, {who was}^{2} usually {very late in the mornings,}^{2} $_{2}\langle$save$\rangle$ {upon {{those}^{0}}^{1} $_{1}\langle$not$\rangle$ {$_{0}\langle$in$\rangle${frequent occasions when he was up all night}^{0}}^{1}}^{2}, was seated at the breakfast table.

The sentence above contains three instances of negation, marked by the cues *save*, *not*, and *in* (in *infrequent*), marked by angular brackets. Each of these cues affects a corresponding section of the sentence—its scope—marked by curly brackets and the matching sub-/superscript number. This example is particularly illustrative because of (1) several instances of overlapping scopes, and (2) a negation cue that does not correspond to the entire token, but only its prefix.

In the case of (1), such sentences in the dataset are "distributed" during pre-processing—meaning that as many instances of the sentence as there are different negation cues are generated, each containing one distinct cue + scope annotation. An obvious drawback of this approach is giving "misleading" signals to the model during training: what was an instance of negation in one copy of the sentence is no longer such in another. Fortunately, this issue is somewhat mitigated by the fact that the task is focussed only on negation scope resolution, and not the detection of negation in itself—meaning that information on the existence of negation, in the form of a cue, is provided to the model upfront, as an input feature.

	CDT	CDD	CDE
Number of sentences	3643	787	1089
Negation instances	886	168	249
Average scope length	7.85	8.14	7.33

Table 1: Core statistics for the Conan Doyle negation corpus, broken down according to the training (CDT), development (CDD), and evaluation (CDE) splits.

In the case of (2), affixes denoting negation are annotated as cues, while the remainder of the token is assigned to the scope of the negation. Fancellu et al. (2016) handle such instances by splitting tokens into distinct word-parts. Our implementation deviates from this approach by treating affix cues as a distinct type of cue, in what would otherwise be a binary true/false representation of cue status.

Table 1 gives basic information about the *SEM dataset, comprising the public-domain literary works of Sir Arthur Conan Doyle (CD) in the *Sherlock Holmes* canon.

2.2 BiLSTM model

Fancellu et al. (2016) present a bi-directional LSTM model that outperforms non-neural state-of-the-art models on the *SEM NR task. The authors initially experiment with a simple one-layer feed-forward network, comparing the performance of word-embedding features only to that of top-performing non-neural models, and then move towards a BiLSTM architecture. They argue that the use of LSTM cells is a better fit for the NR task than RNN, "given that their inner composition is able to better retain useful information when backpropagating the error". An overview of the BiLSTM model architecture (taken from (Fancellu et al., 2016)), with the sequential word-embedding and cue-embedding inputs, is given in Figure 1.

Table 2 shows a subset of experimental results reported in (Fancellu et al., 2016), focussing on the basic BiLSTM model, and its best-performing variation.

We begin by replicating[1] this model, according to the paper description, and the publicly available GitHub[2] implementation. We follow their original

[1]Models and data available at https://github.uio.no/majabu/in5550_exam

[2]https://github.com/ffancellu/NegNN

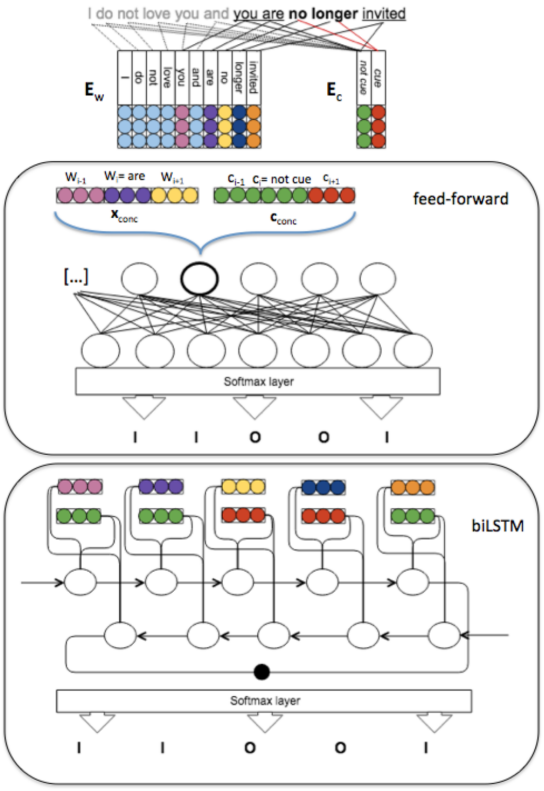


Figure 1: Negation scope detection using the BiLSTM model (bottom frame) for the example "I do not love you and {you are} ⟨no longer⟩ {invited}".

Model	P (%)	R (%)	F1 (%)
Basic	90.04	86.50	88.23
Emb + UPoS	92.62	85.13	88.72

Table 2: (Fancellu et al., 2016) results for the scope detection task (basic BiLSTM with word- and cue-embeddings, and best model variant).

Hyperparameter	Original value	Updated value
embedding vector size	50	100
hidden layer size	200	=
training epochs	50	=
learning rate	10^{-5}	=
optimization algorithm	Adam	=
early stopping	T	F

Table 3: Hyperparameters of the BiLSTM model, as used by Fancellu et al. (2016), and necessary updates used in the best-performing model reported here.

model architecture and provided hyperparameters, as shown in Table 3.

A key aspect of this model—refered to as the "basic" model from this point on—is the handling of input information. As previously mentioned, apart from the sentence tokens in their inflected form, embedded per-token cue information is also made use of, being passed to the model in a separate input channel. Both inputs are passed in the form of 50-dimensional vector representations. This approach is the starting point of our experiments.

3 Experimental Results

In this section, we present the results of our experiments with the basic model described in 2.2, and extensions in line with and inspired by experiments in (Fancellu et al., 2016).

3.1 Representation size and handling

We searched for the best representation size for the input tokens and cue information. Additionally, we considered an alternative approach to combining the representations—using the mean of the two vectors as input, rather than treating them individually as sequential input—but since this led to no marked improvement in performance, that line of probing was abandoned.

Our initial intuition was that using a 50-dimensional representation both for token and cue information was superfluous, given the scarcity of labels that need to be encoded to represent cue information. We experimented with smaller cue signal encodings, but these proved to harm performance. (In the interest of space, the full results of these and other experiments are available in Appendix A.) Therefore, we experimented with expanding the representation sizes in equal measure.

Table 4 gives an overview of the best-performing models depending on their representation sizes. The table shows the "scope tokens" score given by the official *SEM scorer, and used in (Fancellu et al., 2016). Note that, while the various hyperparameter searches were carried out and compared using an internal scorer on the CDD (development) portion of the data, all experimental results reported in Sections 3 and 4 show the *SEM scorer's score on the fully trained model evaluated on the CDE (evaluation) portion of the data.

Considering that all three models show near-identical performance, it was not clear which setup was the best to proceed with. Ultimately, we chose the model with 100-point representations for both

tokens and cue information, given that (1) this model had the lowest total number of "hard errors" (negation sentences with incorrectly detected scopes) according to the scorer, (2) the equal vector sizes would make it easier to continue conducting experiments with different approaches to representation combination, and (3) from a practical perspective, experiments conducted in Section 4 would inevitably lead to very large concatenated input vectors, and it intuitively seemed a better idea to perform some semblance of dimensionality reduction at the embedding stage, rather at the first layer. Future experiments could test whether this is actually the case.

Another deviation from the original implementation was the early stopping criterion used by Fancellu et al. (2016). Our experiments showed that training the model for the full 50 epochs defined in the description paper, but with no variation on early stopping, leads to an increase in performance—e.g., compare the 100/100 (initial Basic with early stopping) and Basic (full training cycle) models in Tables 4 and 5. Therefore, we continued further experiments with the 100/100 setup and full training.

4 Introducing Additional Information

In this section, we present continued expansions of "our" best variant of the basic model. Subsection 4.1 shows work following Fancellu et al. (2016) experiments with lemma and PoS-tag information, while 4.2 demonstrates the effect of introducing grammatical depencency information.

4.1 Lemmas and PoS-tags

We expanded the two input channels of the basic model by experimenting with different combinations of additional information sources. Apart from the surface wordforms and cue information, the CD dataset also contains lemma forms of the tokens, as well as PoS-tags provided by the GENIA tagger (Tsuruoka and Tsujii, 2005) (denoted as *XPoS* in our report). Additionally, we experimented with mapping the XPoS tags to the smaller set of Universal Dependencies (Nivre et al., 2016) PoS-tags (denoted as *UPoS*).

In line with (Fancellu et al., 2016), we also experimented with using pretrained word embeddings for the surface tokens, using the Gensim Continuous Skipgram (Řehůřek and Sojka, 2010) embeddings trained on the Gigaword corpus (Parker et al.).

Table 5 reports experimental results for the top-performing model combinations. Using pretrained embeddings led to a 1- to 2-point improvement in performance, compared to ad-hoc built word representations. However, there is surprisingly little variation between the two approaches once additional sources are introduced. Our experiments have also shown that UPoS tags work better than XPoS tags—possibly because, considering the multitude of input signals in this approach to model expansions, less complex PoS-tag schemas convey more salient information. This is in line with the findings of (Fancellu et al., 2016), whose experiments also show (with very minor deviations) that UPoS tags lead to better model performance than the extended XPoS schema. The authors also highlight the importance of the fact that language-independent features (Universal PoS-tags) are used in the ultimately best-performing model.

On the other hand, consdering that the introduction of lemmas led to no dramatic difference in the results, we abandoned further experiments with that input channel. An unexplored avenue, filed for future work, would consider introducing pretrained embeddings for the lemmas as well.

For continued experiments, we decided on using the top-performing Basic model, and the best variation of the pretrained-embeddings-based model: Emb + UPoS.

4.2 Dependency Parsers

For our final set of experiments, we expanded the input fields with the addition of dependency parse information provided with the task description (Oepen, 2020). Two annotation layers were added to the *SEM CD dataset: PoS-tags and dependencies generated using (De Marneffe and Manning, 2008) (*Stanford* parser), and those generated by the (Ji et al., 2017) parser (*ECNU* parser).

Apart from custom tokenisation and PoS-tagging, which differs to a small degree from the format of the original *SEM dataset, the parses introduce information on syntactic relations between sentence constituents, in the form of node relations in a tree-structure representation of the target sentence. For our initial introduction of dependency information to the NR system, we picked one qualitative property (top-node status in the dependency structure), and one quantitative property (number of incoming and outgoing edges for each constituent node/token).

Same as with the other additional features intro-

Model	gold	system	tp	fp	fn	P (%)	R (%)	F1 (%)	% correct	# errors
w:100 c:100	1805	1655	1460	195	345	88.22	**80.89**	84.40	**81.22**	**210**
w:200 c:150	1805	1498	1395	103	410	93.12	77.29	**84.47**	80.41	219
w:250 c:150	1805	1481	1383	98	422	**93.38**	76.62	84.17	80.86	214

Table 4: Results for the scope detection task for best-performing combinations of representation sizes.

Model	gold	system	tp	fp	fn	P (%)	R (%)	F1 (%)
Basic	1805	1827	1570	257	235	85.93	**86.98**	**86.45**
Basic + UPoS + Lemmas	1805	1662	1497	165	308	90.07	82.94	86.36
Emb + UPoS	1805	1637	1484	153	321	**90.65**	82.22	86.23
Emb + Lemmas	1805	1530	1420	110	385	92.81	78.67	85.16

Table 5: Results for the scope detection task for best-performing variations, trained without early stopping.

duced to the network, we built 100-dimensional encodings for each token's top-node status, and number of connected edges. The results of these experiments are given in Table 6.

The Basic model (wordform + cue information) suffered a marked decrease in performance (in particular, recall) with the addition of dependency data. A tentative explanation for this occurrence is the possibility that the most salient input channel—word embeddings—is overpowered by three channels of arbitrarily encoded additional data (trinary cue information, binary top-node status, and per-token edge counts), which may easily lead to precision-based overtraining.

Interestingly, the embeddings-based model saw a roughly 3-point improvement with the addition of parsing data. The PoS-tag input channel was retained, but the UPoS tags from the original *SEM dataset were replaced with the parser-provided PoS-tagging output. Any difference between the two (apart from that arising from tokenisation) would likely be down to noise, which makes it unlikely that an updated tagset alone is responsible for the increase in performance. This experiment hints, to a certain extent, the contribution of structural information to the NR task.

We retain these four models in focus and analyse their outputs in the final stage of our research project—the error analysis in Section 5.

Finally, in Table 7, we compare the performance of our basic replication and best model against the basic and best model results reported in (Fancellu et al., 2016). Comparing rows 1 and 3, it is visible that our implementation isn't able to fully replicate the results of the original paper—whether this is due to oversights in the model replication, or small variations in the evaluation data, we have yet to explore. However, as visible from rows 2 and 4, introducing dependency parse information elevates even this suboptimal replication model above the best-performing variation of the original implementation. Again, this leaves open for future exploration the question of ultimate performance scores when introducing dependency parse information to the original implementation.

5 Error analysis

In this section, we present a small-scale error analysis of the output given by the depencency-enhanced models. Although in the case of the Basic model(s) these are not the most accurate results achieved, additional layers of token properties arguably provide more aspects for analysis. We consider error rates of the four models depending on one qualitative token property (PoS-tag) and one quantitative (number of incoming and outgoing edges in the dependency tree structure).

Figure 2 plots a histogram of token/node distribution depending on the number of incoming and outgoing edges, according to each respective parser. Tokens are grouped and counted by their edges both for the entire corpus, and for in-scope tokens only (constituents of a negation scope). Unsurprisingly, for both parsers, most words are to be found at the left end of the spectrum (fewer incoming and outgoing edges, less complex dependency relations), with a sharp drop-off towards the tail-end of higher edge counts.

Figure 3 plots the error counts for the four setups, depending on edge counts. Bearing in mind the fact that the majority of nodes fall on the left end of the spectrum (fewer edges), these histograms nevertheless show some interesting properties of the models. Considering the performance of any

Model	gold	system	tp	fp	fn	P (%)	R (%)	F1 (%)
Basic + ECNU	1801	1309	1207	102	594	92.21	67.02	77.62
Emb + UPoS + ECNU	1801	1659	1546	113	255	93.19	**85.84**	**89.36**
Basic + Stanford	1802	952	898	54	904	**94.33**	49.83	65.21
Emb + UPoS + Stanford	1802	1610	1519	91	283	**94.35**	84.30	**89.04**

Table 6: Results for the scope detection task for best-performing model variations, using additional dependency parse information.

	Model	gold	tp	fp	fn	P (%)	R (%)	F1 (%)
Fancellu et al.	**Basic**	1830	1583	175	247	*90.04*	**86.50**	*88.23*
	Emb + UPoS	1830	1552	124	272	**92.62**	85.13	**88.72**
Our implementation	**Basic**	1805	1570	257	235	85.93	**86.98**	86.45
	Emb + UPoS + ECNU	1801	1546	113	255	**93.19**	85.84	**89.36**

Table 7: Basic and best model variation, as reported by Fancellu et al. (2016), and the experimental results of this replication study.

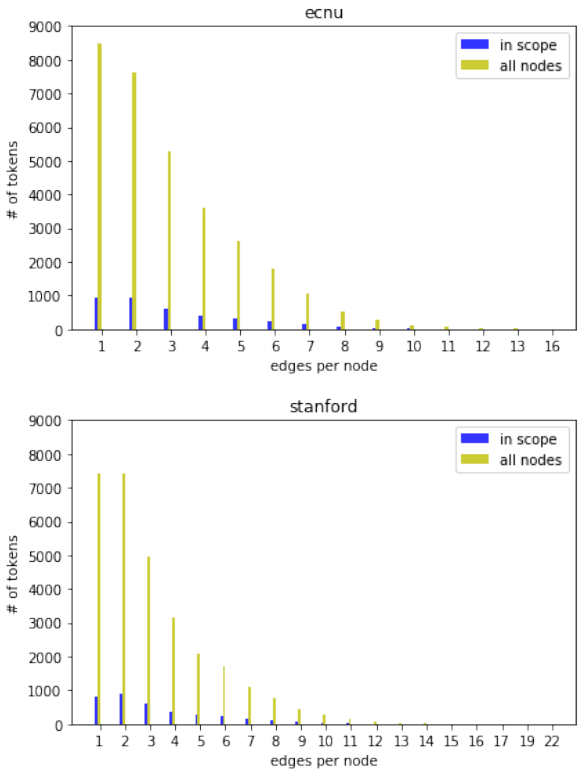


Figure 2: Incoming + outgoing edges per node, for ECNU- and Stanford-parsed data, broken down by scope-tokens (blue) and all tokens (yellow).

one setup individually (e.g. the top frame), it is visible that misclassifications (false positives, tokens incorrectly flagged as belonging to a negation scope; and false negatives, undetected scope tokens) make up a considerably smaller proportion of the total classification per egde-count group as we move towards the right end of the spectrum. Higher edge counts signify words with more dependency relations—i.e., words that are more important in the semantic frame of a sentence. Passing information to the parser about the structural significance of this particular token, in the form of edgewise connectedness, likely makes it easier to discern the word as one likely to comprise a negated unit. In other words, knowing that a token—such as the adjectival *those* in Example 1 above—is the carrier of coreference in a nested sentence (owing to its many incoming and outgoing dependency relation edges) possibly makes it more likely to be classified as the target of a negation. This is also one possible reason why any setup including PoS-tag information outperforms those without—there being a correlation between a word's part of speech and its role in a dependency structure.

Furthermore, these histograms unsurprisingly echo the overall performance of the different model variations: comparing the (mis)classifications between the Basic and Emb+UPoS setups for either parser, there is a clear reduction of errors across the edge-count spectrum for the latter model.

Table 8 gives a breakdown of (mis)classifications per model, depending on a token's top node status. Similarly as seen in the edgewise histograms, false positive classifications are a more frequent error for

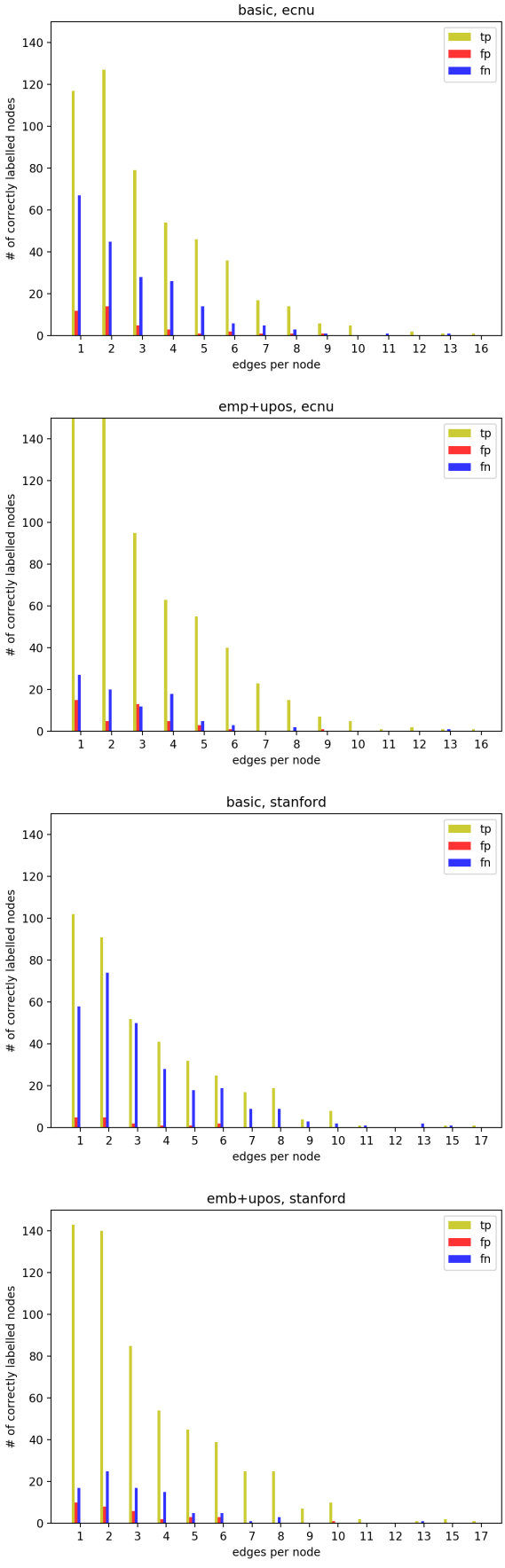


Figure 3: Token-level classification, by incoming + outgoing edges per node: correct in-scope classifications (yellow, tp), incorrectly classified scope tokens (red, fp), and undetected scope tokens (blue, fn). True negative counts were omitted, as they make up an overwhelming majority of the data.

Top node:	yes		no		total
Basic, ECNU	tp: 25.80 fn: 8.87	fp: 01.61 tn: 63.71	tp: 22.29 fn: 11.16	fp: 1.67 tn: 64.86	248 / 5131
Emb + UPoS, ECNU	tp: 32.40 fn: 2.80	fp: 02.80 tn: 62.00	tp: 28.33 fn: 51.60	fp: 02.08 tn: 63.89	248 / 5131
Basic, Stanford	tp: 22.00 fn: 18.40	fp: 00.08 tn: 58.80	tp: 16.59 fn: 17.30	fp: 01.14 tn: 64.96	250 / 5080
Emb + UPoS, Stanford	tp: 36.80 fn: 03.60	fp: 01.16 tn: 58.00	tp: 28.11 fn: 05.78	fp: 01.71 tn: 64.38	248 / 5131

Table 8: Token-level classification, by top node status (percentages).

POS	tp (%)	fp (%)	fn (%)	total
aux	**52.70**	01.54	06.54	260
adj	40.23	00.78	01.17	256
pron	39.97	00.36	**00.61**	813
verb	39.47	00.27	00.83	717
noun	37.16	00.31	01.11	627
adp	29.86	00.44	01.99	452
det	30.40	**00.25**	02.01	398
adv	24.06	00.67	02.03	295

Table 9: Classification by part of speech; (Emb + UPoS, Stanford).

the models using ECNU-parsed information, even though these models have a slight edge over the Stanford setups overall.

Finally, Table 9 gives a breakdown of (mis)classifications depending on a token's UPoS-tag. (For the sake of spatial efficiency, some parts of speech (e.g. numerals and punctuation) were omitted from this table; for full details, see Appendix A.) Even though this limited breakdown doesn't allow for inter-model comparison (e.g. whether verbs are easier to classify with or without dependency information), it still allows for educated guesswork on the difficulties of identifying scope constituents based on their part of speech. For example, auxiliary verbs have the highest rate of correct classifications, even though they are less frequent in the data than verbs or pronouns, likely because of their frequent proximity to negation cues in the simplest forms of negation (e.g. *I said that I* {*had*} ⟨*not*⟩. (Oepen, 2020)) By comparison, adverbs seem particularly difficult to correctly label as belonging to a negation scope—as conveniently illustrated by the out-of-scope-but-interjected *usually* in Example 1 above.

6 Conclusions and Outlook

We presented a best-attempt reimplementation of the Fancellu et al. (2016) BiLSTM model for negation scope resolution, and built upon it by varying details in the architecture and introducing additional layers of information about the input data. Using the "kitchen sink" approach to model expansion, we narrowed down a combination of additions that improved upon the performance of the original model on the same training and evaluation dataset.

Using this new model, we looked at several linguistic properties of the data and reflected on how they affect the model's performance. This gave us some small insight into which aspects of the data correlate with a higher challenge in detecting negation scope.

This work suggests other potentially productive directions for future research: for example, experimenting with different pretrained word embeddings, as well as introducing embeddings for token lemmas, or a combination of wordforms and PoS-tags. One of the research questions in this research track (Oepen, 2020), which did not fall within the scope of this paper, is the use of pre-trained deep contextualised embeddings (e.g. ELMo, BERT). Furthermore, from a linguistic perspective, the use of dependency parsing information in this paper focussed on structural properties of tokens as nodes in a dependency graph—leaving room for future studies utilising other qualitative properties of nodes provided by dependency parsing, and looking into this issue in greater depth.

Acknowledgments

We thank the programme chairs and their assistants for their advice and support in carrying out this study. We also thank the reviewers for their insightful comments and helpful suggestions.

References

Xavier Carreras and Lluís Màrquez. 2005. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of the ninth conference on computational natural language learning (CoNLL-2005)*, pages 152–164.

Marie-Catherine De Marneffe and Christopher D Manning. 2008. The Stanford typed dependencies representation. In *Coling 2008: proceedings of the workshop on cross-framework and cross-domain parser evaluation*, pages 1–8.

Federico Fancellu, Adam Lopez, and Bonnie Webber. 2016. Neural networks for negation scope detection. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 495–504, Berlin, Germany. Association for Computational Linguistics.

Murhaf Fares, Stephan Oepen, Lilja Øvrelid, Jari Björne, and Richard Johansson. 2018. The 2018 shared task on extrinsic parser evaluation: on the downstream utility of English Universal Dependency Parsers. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 22–33.

Tao Ji, Yuekun Yao, Qi Zheng, Yuanbin Wu, and Man Lan. 2017. ECNU at EPE 2017: Universal Dependencies Representations Parser. *EPE 2017*, page 40.

Emanuele Lapponi, Stephan Oepen, and Lilja Øvrelid. 2017. Epe 2017: The Sherlock negation resolution downstream application. *Proceedings of the 2017 Shared Task on Extrinsic Parser Evaluation*, pages 21–26.

Roser Morante and Eduardo Blanco. 2012. *SEM 2012 shared task: Resolving the scope and focus of negation. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 265–274, Montréal, Canada. Association for Computational Linguistics.

Joakim Nivre, Marie-Catherine De Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. 2016. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666.

Stephan Oepen. 2020. UiO IN5550, Spring 2020, Home Exam Task 2: Negation Scope Resolution.

Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. English Gigaword fifth edition, 2011. *URL https://catalog. ldc. upenn. edu/LDC2011T07.[Online*.

Geoffrey K Pullum and Rodney Huddleston. 2002. Adjectives and adverbs. *The Cambridge grammar of the English language*, pages 525–595.

Jonathon Read, Erik Velldal, Lilja Øvrelid, and Stephan Oepen. 2012. UiO 1: Constituent-based discriminative ranking for negation resolution. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 310–318. Association for Computational Linguistics.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. http://is.muni.cz/publication/884893/en.

Dan Shen and Mirella Lapata. 2007. Using semantic roles to improve question answering. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, pages 12–21.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 467–474. Association for Computational Linguistics.

A Appendix: Extended experimental results

This section contains the full result report of hyperparameter-search and other auxiliary experiments.

Unless stated otherwise, the scores reported here were achieved during development, using the CDT (train) portion of the dataset for training, and CDD (dev) for evaluation. Unlike the official *SEM scorer, during development we used an internal scorer that reported three values: scope label accuracy; total count of example sentences with existing negation scopes that were labelled as false negatives in their entirety ("undetected scope"); and total count of example sentences without any negation cue or related scope, where the system labelled certain tokens as belonging to a negation scope nevertheless ("phantom scope").

As mentioned in Section 3.1, one of the avenues of exploration was altering the model architecture to combine all embedding vectors by calculating the mean, and using that vector as input, rather than taking each embedding vector separately in sequence. This variation showed no improvement in performance, and was therefore abandoned. The results of these experiments are given for comparison in tables 11 and 12 (sequential vs. averaged); and 11 and 15 (sequential vs. averaged).

As further mentioned in Section 3.1, we also experimented with smaller cue signal encodings, but these proved to harm performance, and were therefore also abandoned. The results of these experiments are shown in tables 10 and 13 (*SEM scorer on CDE data).

All together, tables 10, 11, 13 (*SEM scorer on CDE), and 14 show the complete results of the representation-size grid search.

Table 16 (*SEM scorer on CDE data) reports experimental results for all model variations as described in Section 4.

Finally, Table 17 (*SEM scorer on CDE data) gives the full results for per-POS-tag error anaysis, as described in Section 5.

w/c	50	4
50	.960 (53/0)	.908 (264/0)
100	.957 (66/0)	.931 (122/5)
200	**.969 (36/0)**	.954 (58/4)
300	.937 (47/16)	.936 (108/3)

Table 10: Fancellu-based BiLSTM; dev-Accuracy (undetected/phantom scope count)

w/c	50	100	150
50	.945 (11/5)	.964 (33/0)	.945 (18/0)
100	.958 (68/0)	**.970 (20/0)**	.960 (15/0)
150	.966 (41/0)	.965 (40/0)	**.969** (39/0)
200	**.970** (40/0)	.966 (36/0)	**.972** (34/0)
250	**.969** (28/0)	.964 (44/0)	**.971** (28/0)

Table 11: Fancellu-based BiLSTM (concatenated v.); dev-Accuracy (undetected/phantom scope count)

w/c	50	100	150	200	250
50	.952 (16/5)				
100		.963 (21/1)			
150			.952 (17/0)		
200				**.970 (27/0)**	
250					**.965 (12/0)**

Table 12: Fancellu-based Bi-LSTM (averaged v.); dev-Accuracy (undetected/phantom scope count)

w/c	gold	system	tp	fp	fn	P (%)	R (%)	F1 (%)
50/50	1805	1269	1159	110	646	91.33	64.21	75.41
50/4	1805	0	0	0	1805	0.00	0.00	0.00
300/50	1805	2021	1383	638	422	68.43	76.62	72.29
300/4	1805	1036	802	234	1003	77.41	44.43	56.46
200/50	1805	1592	1398	194	407	87.81	77.45	**82.31**
200/4	1805	1465	1198	267	607	81.77	66.37	73.27
100/50	1805	1109	1037	72	768	93.51	57.45	71.17
100/4	1805	1133	808	325	997	71.32	44.76	55.00

Table 13: Fancellu-based BiLSTM; scope tokens score

w/c	50	100	150
50	76.27	76.95	79.66
100	71.57	**84.40**	81.78
150	80.02	79.49	82.06
200	82.88	79.71	**84.47**
250	81.54	77.07	**84.17**

Table 14: Fancellu-based Bi-LSTM (concatenated v.); scope tokens F1 score

w/c	50	100	150	200	250
50	79.28				
100		**82.88**			
150			79.48		
200				**82.97**	
250					**83.92**

Table 15: Fancellu-based BiLSTM (averaged v.): scope tokens F1 score

Model	gold	system	tp	fp	fn	P (%)	R (%)	F1 (%)
basic + XPOS	1805	1324	1232	92	573	93.05	68.25	78.74
basic + UPOS	1805	2224	1617	607	188	72.71	89.58	80.27
basic + lemmas (cat)	1805	1520	1344	176	461	88.42	74.46	80.84
basic + lemmas(avg)	1805	3216	1677	1539	128	52.15	92.91	66.80
basic + UPOS + lemmas (cat)	1805	1636	1473	163	332	90.04	81.61	**85.62**
basic + UPOS + lemmas (avg)	1805	1579	1416	163	389	89.68	78.45	83.69
embeddings	1805	1376	1224	152	581	88.95	67.81	76.95
emb + XPOS	1805	1556	1419	137	386	91.20	78.61	84.44
emb + UPOS	1805	1661	1479	182	326	89.04	81.94	**85.34**
emb + lemmas (cat)	1805	1927	1568	359	237	81.37	86.87	**84.03**
emb + lemmas (avg)	1805	1444	1301	143	504	90.10	72.08	80.09
emb + UPOS + lemmas (cat)	1805	1564	1411	153	394	90.22	78.17	83.76
emb + UPOS + lemmas (avg)	1805	1598	1265	333	540	79.16	70.08	74.34

Table 16: Architecture variations on Fancellu-based BiLSTM (early stopping)

POS	tp	fp	fn	tn	total
pron	325	25	42	421	813
verb	283	13	44	377	717
noun	233	15	42	337	627
adp	135	11	42	264	452
det	121	4	31	242	398
adv	71	5	17	202	295
aux	137	4	17	102	260
adj	103	5	9	139	256
part	31	1	4	160	196
conj	6	2	10	167	185
sconj	33	2	12	119	166
propn	28	4	10	82	124
intj	0	0	0	35	35
num	13	0	3	17	33
punct	1	0	20	752	773

Table 17: Emb + UPoS, Stanford; Classification by POS

Targeted sentiment analysis for Norwegian

Jan André Fagereng
janafag@ifi.uio.no
University of Oslo

Abstract

Aspect-based or target-based sentiment analysis is regarded as a relation extraction problem where the goal is to extract aspects or entities in natural language text and assign a sentiment polarity to the entity. In this regard, it can be easier to think of this specific problem as two separate subtasks. The first task is to identify and extract the entities in the text, thus has some inherent similarities with named entity recognition (NER) where the goal is to detect entities of independent domains (location, company etc.). The latter task is to assign a polarity, either positive, negative or a neutral sentiment for the extracted entities. In this paper, I examine some distinct approaches in order to find improvements in this field of targeted sentiment analysis for a particular Norwegian dataset. Experiments of three architectures, collapsed-, joint- and pipeline architectures were performed on the recently created dataset, NoReC fine. In addition, two unique deep recurrent units were applied. The dataset is provided by the University of Oslo (Øvrelid Lilja, 2019) and consists of short newspaper reviews, tagged with an entity tag (BIO-format) and a respective sentiment polarity(O, N, P).

1 Introduction

Regular sentiment analysis attempts to extract the polarity of a document or sentence using a classifier on a document-level. The outcome of this type of classifier is either positive, negative or neutral and defines the polarity towards the specific text. Targeted sentiment analysis is a sub-task of sentiment analysis which attempts to locate one or more entities within certain documents or sentences, in addition to pinpoint the polarity directed towards the entity/entities. Therefore this type of task is more fine-grained and make predictions on the text in a word-level rather than a sentence- or document-level.

Compared with document-level sentiment analysis, targeted sentiment analysis is more complicated as it requires a model to first find possible several entities and detect relational words which expresses the polarity of the entities.

In this paper, I describe three different type of neural architectures modelled by the newly created NoReC fine dataset (Øvrelid Lilja, 2019), which according to Øverlid et al. is the first Norwegian dataset of its kind (Øvrelid Lilja, 2019). The goal of these experiments is not necessarily to obtain the highest metrics achievable for this specific dataset, but to observe the differences of performance applying distinct model architectures and comparing their ability to tackle the problem of targeted sentiment analysis. Therefore, I propose three methods for obtaining entities and respective polarities explained, namely collapsed-, joint- and pipeline model architectures. The dataset is then modelled using gated recurrent units, and long-short-term-memory recurrent units (Hochreiter and Schmidhuber, 1997), which has historically shown great potential within the domain of natural language processing. In order to accentuate the differences of the methods in terms of metrics, the sizes and depths of the neural models are kept in similar shapes.

2 Related Work

Zhang et al. (2015) experimented with pipeline-, collapsed- and joint-architectures in order to detect entities and their respective polarity. They found that a joint approach yielded the best F1-scores on both a English and Spanish targeted sentiment dataset. They also concluded that a pipeline architecture is better suited for tasks which provide additional external resources. Zhang et al. (2015) proposed a novel combination of a con-

ditional random field (CRF) and a neural approach, which outperformed both the CRF and neural as standalone models. Jebbara (2016) work on aspect-based relational sentiment analysis proposes a stacked neural network approach, utilizing the local feature extraction properties of a convolutional network along with the more global feature extraction properties of the recurrent units. Based on the results, they concluded a joint-architecture to be superior and they also reported a benefit for a domain-specific word embedding compared to a domain-independent embedding.

”Munken	Bistro	mangler	litt	på	detaljene.
B	I	O	O	O	O
Neg	Neg	O	O	O	O

Translated
”Munken Bisto lacks a bit of details.”

Figure 1: Showing an example of a tagged sentence with BIO-format tags and polarity of the expressed entity.

3 Dataset

The dataset provided in this work, consists of newspaper reviews from the Norwegian corpus, NoReC(Øvrelid Lilja, 2019). These texts are reviews from multiple domains, such as literature, video games, TV-series etc. These sentences have been annotated using the BIO-schema, along with a polarity target for the expressed entity. The BIO-schema represents a set of three distinct target values, B, I, O. These characters refer to ”beginning”, ”inside” and ”outside”, of an entity in that order. An example in figure 1 follows the text and its respective targets for each word.

For the explicit example in figure 1, the review refers to a specific restaurant, ”Munken Bistro”. ”Munken Bistro” is therefore tagged ”B” following an ”I” to indicate the beginning and inside of an entity. In addition, both of these targets have an additional polarity expression which indeed refers to the directed polarity towards the entity in the sentence. The remaining words in the sentence are tagged ”O” to imply neither an entity or polarity. Notice there is no placeholder target to which words/tokens in the sentence gives the polarity towards the entity. The dataset is split in to three distinct sets for the training, validation and testing processes, respectively. All the models are trained using the train-set, subsequently validated using the validation-set and ultimately tested with the remaining test-set. All metrics following this paper are therefore a result of evaluating the test-set.

3.1 Statistics

In this sub section, I outline some statistical properties of the train-, dev- and test-dataset. Table 1 shows the distribution of tagged words a long with the size of the split dataset. Shown in table 2, the polarity of the entities indicates a higher percentage of positive tagged entities. The amount of tagged entities lies between 6.3% and 6.8% for each respective split. There are no targets which presents the holder of polarity towards a entity. Additional information of the underlying dataset can be found in Øverlid et.al paper (Øvrelid Lilja, 2019).

	Train	Dev	Test
Sents	5915	1151	895
B	3339	629	511
I	3453	643	465
O	91691	18336	14425
Pos	4584	869	714
Neg	2208	403	262

Table 1: Shows the counts of BIO-targets as well the polarity targets for each word in the datasets.

	Train	Dev	Test
Targets	0.068%	0.064%	0.063%
Positive	67%	68%	73%
Negative	33%	32%	26%

Table 2: First row shows the percentage of word with targets in all three datasets. Second and third row showing the polarity percentage of tagged words in all datasets.

4 Models

In this work, I experiment with three different approaches for neural based architectures in targeted sentiment analysis for the NoReC fine dataset.

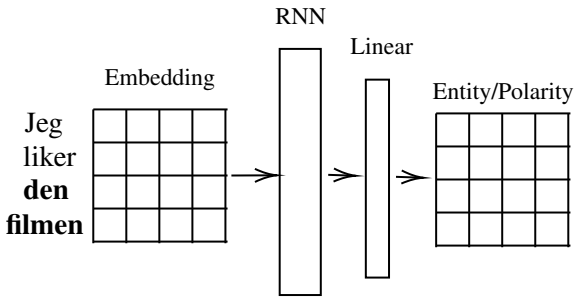


Figure 2: Figure showing data flow with a collapsed target architecture, representing the baseline.

The idea is to compare these architectures with collapsed-, joint- and pipeline targets. The resulting models are explained in more detail in each respective sub section. All experiments of these models make use of both the LSTM- and GRU-recurrent units. The hyperparameters are kept static for each recurrent model. All models are programmed using the PyTorch neural network framework, which is a framework developed and maintained by Facebook (Paszke et al., 2017).

4.1 Collapsed Model

The idea of the collapsed model is to incorporate the two tasks (entity and polarity) within targeted sentiment analysis in one. Subsequently, the targets for this model can be seen as T = {0, 1, 2, 3, 4}, for outside, beginning-positive, inside-positive, beginning-negative, inside-negative in that order. The collapsed model takes the whole sequence as input, and outputs both the entity- and polarity predictions in a collapsed manner. This task operates as the baseline architecture and therefore a baseline model is added to the experiment to provide a set of metrics in which I can compare the remaining architectures. This model consists of a 1-layered Bi-LSTM network and is modelled to give collapsed target outputs. A visualization of the architecture is provided in figure 2, where the model predicts both the entity and sentiment at each time step of the sequence.

4.2 Joint Model

As the name suggests, this architecture predicts entity- and polarity-targets in a joint manner. In similarity with the collapsed model, the idea to is apply the same parameters to the input-data. Although for this type of architecture the output

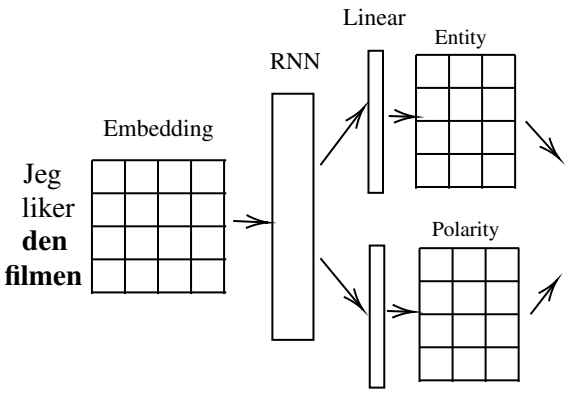


Figure 3: Figure showing the flow of a joint model architecture

consists of two fully-connected layers, which predicts the entity and polarity separately, though in parallel. The targets for each sequence are represented as two sets which can be seen as T1 = {0, 1, 2}, T2 = {0, 1, 2} for outside, beginning and inside and neutral, positive, negative, respectively. The recurrent layer of this model represents the data in a joint manner, and the output layers returns two separate predictions. A visualization of the architecture is provided in figure 3. Both linear layers is given the same encoded input from the recurrent units, and predicts the entity and polarity separately. The loss of each respective linear layer are added up in to a total loss, and the gradients are backpropagated with this total loss. The formula is provided below as the sum of losses for each fully-connected layer.

$$L(\theta) = l(x, y, \theta_1) + l(x, y, \theta_2)$$

4.3 Pipeline Model

The previously described model detects both entity and polarity in a joint manner, however, the pipeline model attempts to split the task in a set of two tasks in order to make predictions, sequentially. This way, the separate models can focus on solving completely different tasks. Subsequently, two models are instantiated to solve each task. The first model has the responsibility to detect entities in a sequence of text, and output a prediction for each word/token in the sequence. The targets can be seen as T = {0, 1, 2} as in outside, beginning and inside respectively. The second model is sequentially applied after a possible entity has been

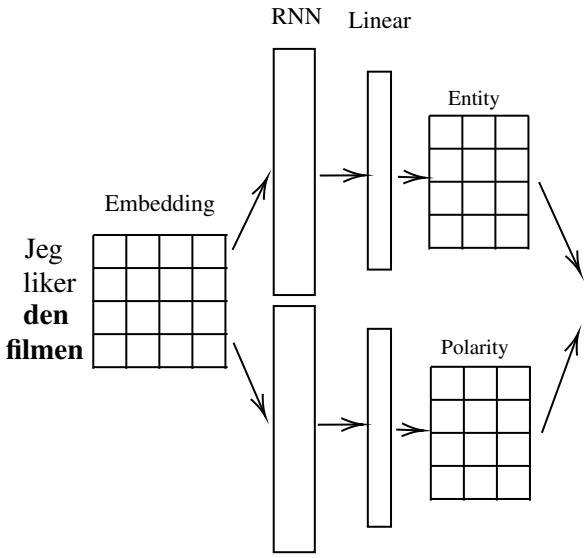


Figure 4: Figure showing the flow of a pipeline architecture. The Polarity part of the architecture only runs when a entity or entities are found within a sequence

located by the entity model. The targets can be seen as T = {0, 1} for positive and negative. In short, this model has the responsibility to detect the polarity if the first model detects any entity. For a sequence where the entity model does not detect any entities, the sentiment model is not applied to any data. As a visualization aspect, one could think of the entity model as a miner, and the sentiment model as a post-processor, hence, the sentiment model is not applicable, if the entity model did not locate an entity. In figure 4, I show a visualization of the architecture. As the visualization can be misunderstood, I highlight the fact that the sentiment model is only given the embedding data if the entity model has predicted an entity/entities. In short, the visualization showing models working in parallel, but in reality, they predict sequentially. The figure is visualized as it is, due to spacing purposes.

Both models in this approach is trained separately, and joined after training to attain the evaluation results. The targets of the dataset are split in order to be trained for the two distinct tasks.

5 Experiments and Results

5.1 Training

This section consists of information regarding the training procedure. Training is done using CPU as the size of the data, and time spent training, was trivial. I trained the models using batches of sequences of size 50, and backpropagated the gradients after each batch. The optimization of all models is performed with the Adam optimizer and the learning rate is set to 0.005. The number of epochs is set to 20, as the models converged relatively fast. Each of the models is applied with the same word embedding to attain consistency of learned word representations. The loss function applied in all experiments is the cross entropy loss, mostly applied for multi-class classification.

5.2 Word Embeddings

The word embedding applied in these experiments is the Norwegian-Bokmaal CoNLL17 corpus which was trained on Norwegian words without lemmatization. The algorithm used is ”Word2Vec Continuous Skipgram” with a window size of 10. The embedding dimension is 100 with a vocabulary size of 1182371. Tokens which are not present in the embedding is added with the same embedding representations as ¡unk¿ (unknown). Therefore new tokens are not learned.

5.3 Evaluation

The evaluation metrics applied in these experiments are binary and proportional precision, recall and F1-score. The proportional metrics are calculated with functions from sklearn, and utilizes the ”micro” average which sums up the global false positives, false negatives and true positives. The binary metrics are calculated in a binary manner, i.e. for each token in the predictions, if any token overlaps with any token in the targets, the number is increased by 1. This generally results in higher metrics for the binary calculation.

5.3.1 Collapsed vs. Joint vs. Pipeline

As the goal of this research is to not necessarily obtain the highest possible metrics for this exact dataset but to observe the metrics as the overall architecture of the models are changed, the sizes of recurrent units and linear layers are kept similar in all the models.

The collapsed model behaves as the baseline in this experiment. In table 4, observing the metrics for the baseline I can see the highest binary F1-score using the GRU-recurrent unit of 0.40, in

addition to an proportional F1-score of 0.24 with the same recurrent unit. For the LSTM-unit the binary and proportional metrics are 0.38 and 0.23, respectively. Although the metrics for each of the distinct architectures is somewhat similar, the baseline achieves the lowest in terms of both binary and proportional metrics. Both the joint- and pipeline architecture is beating the baseline model by 0.04 and 0.05 considering the proportional F1-score. For the binary F1-score, the numbers are 0.04 and 0.02. One might assume that a pipeline architecture would yield much better results than a joint architecture, based on the fact that I applied two different recurrent models on the data, solving two different tasks. But observing table 4, the two architectures shows quite similar results. In fact, the joint architecture beats the pipeline in precision, and F1-score for the binary metric, and precision for proportional. The pipeline using the LSTM recurrent unit achieves the highest proportional F1-score of 0.29. In addition, the latter is also yielding the most uniform distributed numbers for all the metrics, which could signal a more robust architecture.

6 Conclusion

In this work, I present research towards different kind of neural architectures applied on a targeted sentiment dataset for Norwegian. The idea is to provide work on which types of neural architecture might yield better metrics for a targeted sentiment task. I addressed the problem by experimenting with three types of architectures, one for a collapsed seq2seq prediction, another for addressing the task in a joint manner, and a third pipeline architecture which separates the entity and polarity task into two sub tasks. All of the models are tested using both the LSTM- and GRU-recurrent units to observe the impact the different units had on this type of task. The models are kept somewhat similar in sizes to detect the actual differences in architectures, and not necessarily achieve the highest metrics possible for this type of task. Both the joint- and pipeline-architectures performs similar in terms of metrics, and outperforms the baseline model. This comes to show that the altering the prediction flow of the data could yield a better model, than a simple architecture. In addition, the joint-architecture utilizing the GRU-units achieves the highest precision metrics for all the experiments. Ultimately, the pipeline-architecture with LSTM-units achieves the highest proportional F1-score of 0.29. As the word embedding are not updated during training, the models are not able to learn unseen word representations. This could be an exercise for later, but for this experiment, I wanted to observe the overall differences in architectures for targeted sentiment analysis. As a final conclusion for this experiment, the pipeline model obtains the highest scores and is therefore my provisional recommendation. It would be interesting to complete further experimentation of the joint- and pipeline-architectures applying more advanced methods and models.

7 Future work

To further enrich the applied architectures with external information, a placeholder for the polarity towards an entity should be provided. The models can therefore be supervised towards the word/token which expresses the polarity towards an entity. In terms of models, other types of neural architectures, such as transformers, would be highly interesting to make use of. Words in sentences are highly related to each other, and it is not easily modelled by basic neural techniques. Therefore, other techniques such as attention, which has shown great results in other experiments, could be incorporated in similar future experiments in order to increase the metrics of targeted sentiment analysis on this dataset. In addition, introducing regularizing techniques as well as training the word embedding might also yield better results for this type of task. As the size of the NoReC fine dataset is quite trivial, it would also be interesting feeding the models even more data by covering the whole of NoReC dataset Øvrelid Lilja (2019) and incorporating the polarity placeholder targets which it carries.

References

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Cimiano Philipp Jebbara, Soufian. 2016. Aspect-based relational sentiment analysis using a stacked neural network architecture. In *Proceedings of the Twenty-Second European Con-*

		Entity			Sentiment		
Model		P	R	F1	P	R	F1
Pipeline	GRU	**0.46**	0.46	0.47	0.61	0.61	0.61
	LSTM	0.45	**0.55**	**0.50**	**0.67**	**0.67**	**0.67**

Table 3: Showing the metrics for the pipeline models, which are trained as seperate models. LSTM achieves the highest metrics for both the entity- and sentiment-models. The targets of the data are split in to two set of targets, and each model trained on each task respectively.

		Binary			Prop		
Model		P	R	F1	P	R	F1
Baseline	GRU	0.42	0.37	0.40	0.29	0.21	0.24
	LSTM	0.34	0.43	0.38	0.23	0.24	0.23
Joint	GRU	**0.44**	0.39	0.37	**0.34**	0.24	0.28
	LSTM	**0.44**	0.43	**0.44**	0.26	0.21	0.23
Pipeline	GRU	0.41	0.34	0.37	0.27	0.24	0.25
	LSTM	0.40	**0.44**	0.42	0.31	**0.27**	**0.29**

Table 4: Showing the overall metrics for all experiments in this research, for collapsed-, joint- and pipeline-models. The metrics are read as P, precision, R, recall, F1, F1-score for both the binary and proportional evaluation.

ference on Artificial Intelligence, ECAI'16, page 1123–1131, NLD. IOS Press.

Petter Barnes Jeremy Velldal Erik Øvrelid Lilja, Mæhlum. 2019. A fine-grained sentiment dataset for norwegian.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Meishan Zhang, Yue Zhang, and Duy-Tin Vo. 2015. Neural networks for open domain targeted sentiment. In *EMNLP*.

Targeted Sentiment Analysis with Ensembles

Halvor Holhjem Bugge
University of Oslo
Department of Informatics
halvorbu@ifi.uio.no

Abstract

I conduct a series of experiments aimed at improving performance on a targeted sentiment analysis task, using collapsed sentiment and target labels. After getting poor results with a bidirectional GRU model, I attempt to improve the hyperparameters through a grid search. With the success of the grid search, the next experiments focus on the creation of an ensemble, which results in several systems of models with better performance than any of their constituent members.
Finally, a short error analysis is performed on the completed ensemble, and the system is evaluated on the held-out test set.

1 Introduction

Targeted sentiment analysis is a variant of traditional sentiment analysis, where the task is not only to recognize polarity (positive, negative, or otherwise), but also other aspects such as the holder and target of an opinion. Traditional sentiment analysis research often focuses on determining the polarity on a document or sentence-level, which can result in unfavorable results in cases where a text contains more than one sentiment. The more fine-grained approach of targeted sentiment analysis enables us to look at multiple targets and holders and see how these relate.
For the experiments detailed in this paper I utilize a collapsed and simplified version of the NoReC$_{fine}$ dataset [1] made for fine-grained sentiment analysis in Norwegian.
The paper is built around four experiments, and has a structure as follows: Section 4 looks at implementing a bidirectional GRU using the baseline hyperparameters, and gauges its performance. Section 5 looks at optimizing hyperparameters using grid search, and analyzes the results. Section 6 looks at utilizing the combined knowledge of the model population and building a model ensemble. Section 7 builds on the results of experiment 2 and 3, and constructs a further improved ensemble.
Apart from the experiments, in Section 2 the dataset is briefly discussed, in Section 3 the baseline system is set up and tested, and in Section 8 a simple error analysis is performed on the predictions of the final system evaluated on the dev set. In Section 9 I look at the results of the system evaluated on the held out test-set, in Section 10 possible future work is discussed, and finally Section 11 sums up the work of the paper.

Token	Label
Pølse	B-targ-Positive
og	I-targ-Positive
potetstappe	I-targ-Positive
hjemme	O

Table 1: Example sentence in training dataset. *"Sausage and mashed potatoes at home"*

2 On the Dataset

While the original dataset by Øvrelid et al. (2019) is annotated for targets, holders, polarity and intensity (of sentiment), the simplified version employed in this paper contains only labels for target and polarity (Inside, Outside, Beginning + Polarity). While these were originally separate labels, they are here collapsed into five tags by default: (I-targ-Positive, I-targ-Negative, O, B-targ-Positive, B-targ-Negative). Although there exists interesting research on collapsed vs. pipeline and joint model strategies (Mitchell et al. (2013), Zhang et al. (2015)), in this paper I choose to focus on the impact of hyperparameter tuning and ensemble strategies, and as such keep the default label encoding. The data is a subset of the Norwegian

[1] See the paper by Øvrelid et al. (2019)

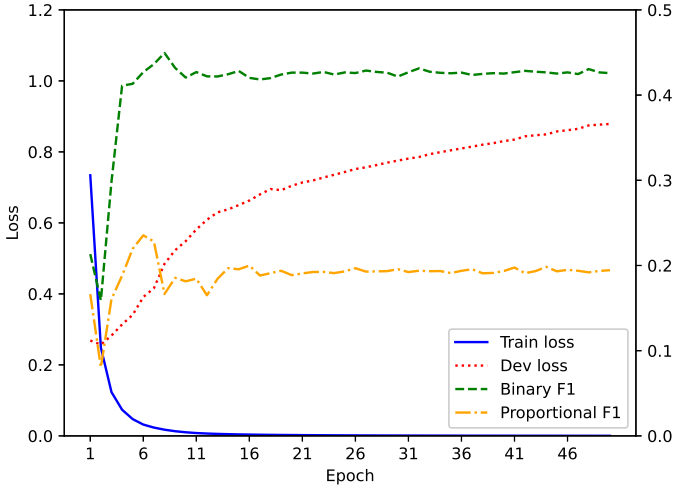


Figure 1: BiLSTM 50 epochs

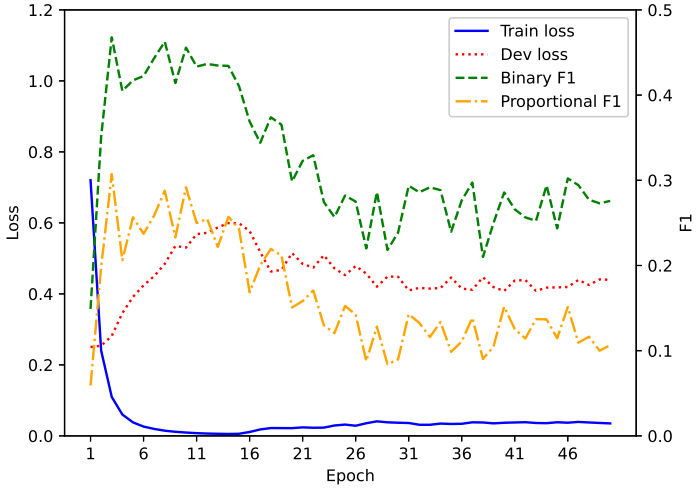


Figure 2: BiGRU 50 epochs

Review Corpus by Velldal et al. (2018), and consists of newspaper reviews ranging in topic from videogames to literature, movies, music and more (See Table 1 for an annotated sample).

3 Establishing a Baseline

Using the provided code, I establish a baseline by setting up a Bidirectional LSTM with the following hyperparameters (see Table 2).

Parameters	Values
BiLSTM layers	1
Hidden dim.	100
Learning rate	0.01
RNN dropout	0
Word dropout	0.01
Batch size	50
Epochs	50

Table 2: Hyperparameters for baseline BiLSTM

For word embeddings I use the **Norwegian-Bokmaal CoNLL17** model (model 58) from the NLPL word embeddings repository.
After training the model for 50 epochs (see Figure 1), I calculate the results using the included scripts (see Table 3).

From Figure 1 we see that the model's F1 scores seem to plateau relatively early on while the train loss deflates towards zero, and the dev loss keeps increasing at a steady incline.

Metric	Binary	Proportional
Target Precision	0.421	0.233
Target Recall	0.431	0.167
Target F1	0.426	0.194

Table 3: Performance of baseline BiLSTM

It seems likely the model quickly starts to overfit, leaving it at its most flexible at around the 6-epoch mark. It is worth mentioning that while the dev loss keeps increasing, the F1 scores seem relatively unaffected. It is possible that as the model overfits, it gets more unsure of the results it produces on the dev set, while still producing correct results.

4 Experiment 1: Bidirectional GRU

In their experiments, Ma et al. (2018) see good performance on targeted sentiment analysis using bidirectional GRUs. In order to see if this different architecture will affect performance, I implement a bidirectional GRU and train it on the same data as the baseline model. I also utilize the same word embeddings and hyperparameters as the baseline (see Table 4).

In a similar fashion to the baseline model, we immediately see that the model reaches an early peak before deteriorating (see Figure 2). In contrast to the baseline however, the BiGRU sees a much more precipitous decline in F1 performance, ending with considerably worse F1 scores than the baseline BiLSTM (see Table 5). In addition we can see that the dev loss levels off as the F1 scores

Parameters	Values
BiGRU layers	1
Hidden dim.	100
Learning rate	0.01
RNN dropout	0
Word dropout	0.01
Batch size	50
Epochs	50

Table 4: Hyperparameters for baseline BiGRU

decrease. It appears the model is more sure of its predictions, all the while performing worse. Due to the apparent instability of this metric, I elect not to use the dev loss as a guideline in my further experiments, and will instead rely on the F1 scores. In order to improve this result, it seems promising to use a form of early stopping to save the model at its peak. I explore this in the next section.

Metric	Binary	Proportional
Target Precision	0.312	0.159
Target Recall	0.247	0.080
Target F1	0.276	0.107

Table 5: Performance of baseline BiGRU

5 Experiment 2: Hyperparameter Grid Search

Variable	Range
Architecture	LSTM, GRU
RNN layers	1, 2, 3
Hidden state dim.	100, 150, 200
Train embeddings	True, False
Bidirectionality	True, False
RNN dropout	0.2, 0.4, 0.6

Table 6: Grid search variables

Since both GRUs and LSTMs are powerful architectures used to good effect in previous sentiment analysis research [2], I elect to try and find improvements with these two architectures. To explore this tuning further, I attempt to perform a grid search over a number of hyperparameters which seem likely to produce interesting and varied results (See Table 6). As constraining the number of RNN layers and hidden state size too much is likely to make a model unable to properly generalize, and too much of an increase could make convergence difficult, tuning these variables is likely to be rewarding. Further, training the embeddings might be beneficial in case unknown words are encountered, and given that the baseline is bidirectional, it is worth looking at how unidirectional models perform. Lastly, dropout is a powerful regularization technique which could prevent the model from overfitting.

To avoid the issues encountered with the baseline BiLSTM and in the experimentation with the BiGRU, I employ early stopping. Since the binary and proportional F1 scores represent slightly different aspects, I use both F1 scores as my metric when saving the best model of each parameter combination.

A model with a higher binary F1 score could be said to be good at getting many predictions partially correct, while not necessarily being completely accurate in all of these. A model with a higher proportional score on the other hand is more accurate (as we will see, while high F1 scores are in general quite correlated, there are many models sporting a high binary along with a low proportional F1 score, and vice versa. See Figure 3).

Since we are dealing with two scores for this metric, I avoid just adding them together, as this could produce a skewed result. For instance, this has the potential to allow models with unusually high binary F1 to dominate those with a high but balanced F1 combination.
To find the best model, I store the model's highest observed binary and proportional F1 score separately, and compare them to the new scores with the simple method shown in Table 8.

I employ a 30% leeway in comparing the model's F1 scores to the current best. Since the best scores are separate, this prevents a downward creep of scores, but lets a potentially more balanced model be saved. The early stopping is done with a patience of 5.

With the completed parameter search, I end up with a population of 216 models (see Figure 3). Analyzing the results, I find a wide spread of F1 scores (see Table 7). The best performing model has a higher Binary and Proportional F1 score than the baseline BiLSTM, so the grid search has thus

[2] An example being the BiLSTM used by Øvrelid et al. (2019).

Model	Layers		Train-embd.		Dropout	F1		Epoch
	Architecture	Hidden	Bidirectional			Binary	Propor.	
Max Binary F1	GRU	2	100	True	True	0.4	**0.482**	0.280	3
Min Binary F1	GRU	3	200	True	True	0.6	0.279	0.197	5
Max Proportional F1	GRU	2	100	False	True	0.4	0.465	**0.308**	2
Min Proportional F1	GRU	3	200	False	False	0.4	0.303	0.166	6
Max B+P F1	GRU	2	100	False	True	0.4	0.465	**0.308**	2
Min B+P F1	GRU	3	200	False	False	0.4	0.303	0.166	6
Median B+P F1 1/2	GRU	3	100	False	False	0.4	0.381	0.230	17
Median B+P F1 2/2	GRU	3	200	True	True	0.4	0.387	0.223	5

Table 7: An overview of notable members of the initial parameter search model population. Best F1 scores in bold.

already produced a tangible improvement.
On the other end of the scale we see that the worst model still performs better than the BiGRU model from experiment 1, which indicates that the early stopping had an effect. We also see that all of these models (apart from the outlier at 17) found their best configuration quite early, between the 2nd and 6th epoch. This fits with the early peaks seen when training the BiLSTM baseline and the BiGRU.

```
lw = 1-0.3 #leeway
if  (BF1>Best_BF1 and PF1>Best_PF1*lw) or
    (BF1>Best_BF1*lw and PF1>Best_PF1):
    save_model()
```

Table 8: Pseudocode for saving best model.
BF1 = binary F1, PF1 = proportional F1.

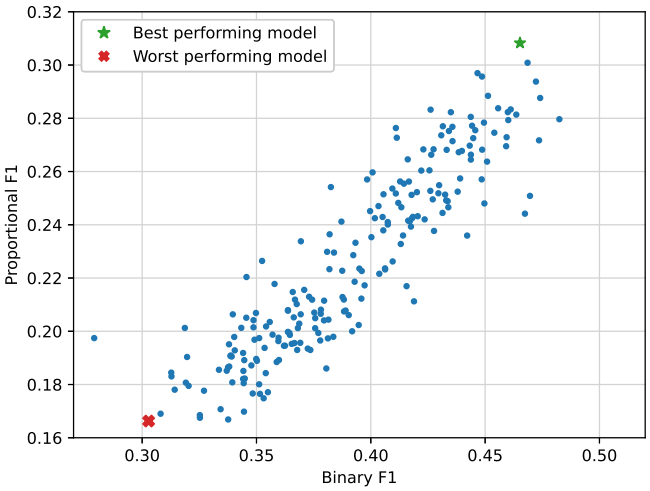


Figure 3: F1 scores of model population from the parameter search, with best and worst marked.

In Figure 3 we can clearly see a an upwards trend, indicative of some improvement factor among the searched over hyperparameters. In order to gain some further insights into the data, I plot the population against their F1 scores, with a number of hyperparameters visualized (See Figures 4, 5).

For many hyperparameters, the results look like Figure 5, with no immediately obvious trend. In Figure 4 however, there appears to be strong correlation between performance and the model's bidirectionality. I use a boxplot to get a more accurate idea of the score distribution (See Figure 6). Here we see very clearly how bidirectionality is contributing to the F1 scores of the population. Bar a few outliers, bidirectionality seems to be a deciding factor in model performance. I constrain the population to only bidirectional models, and analyze the data again (See Figure 7, 8). Looking at the constrained population, I can draw a few conclusions on what parameters might work better for a model suited to our task:

Bidirectionality It is very evident that a bidirectional model is much more capable of handling the targeted sentiment analysis task. It makes sense that having the history of both directions of the input available is advantageous when deciding on polarity and target, since these can be defined and affected by elements that appear both before and after the target.

Architecture Both GRU and LSTM appear pretty evenly matched. However, despite the distribution of GRU model scores being a lot wider than that of LSTM, the GRU population also has the highest median, and produces the highest scores.

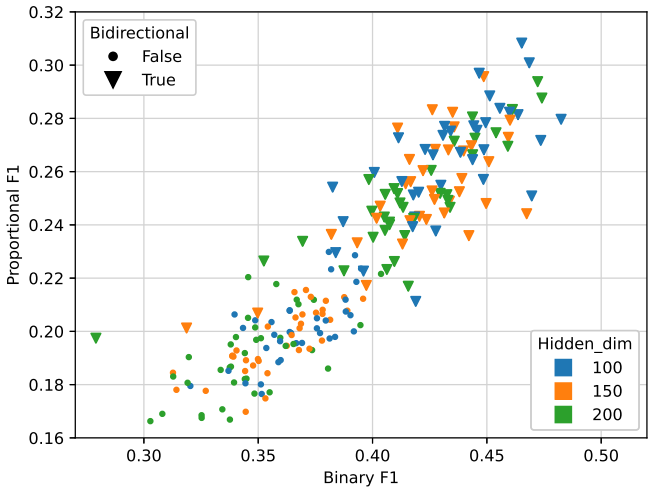


Figure 4: F1 scores with bidirectionality and hidden state dimensions visualized. The bidirectional population is clearly distinct.

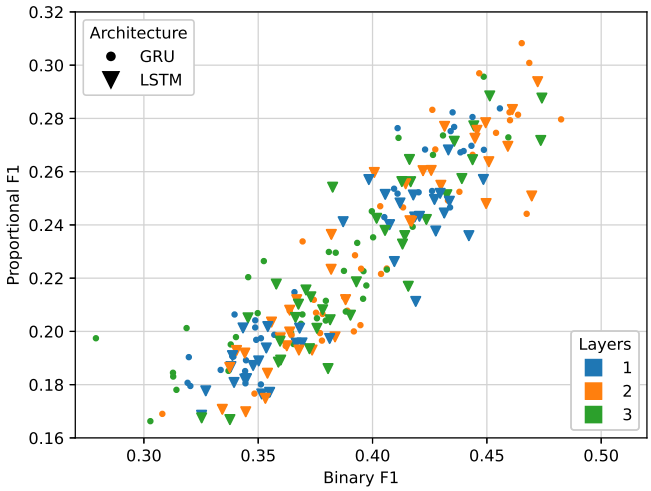


Figure 5: F1 scores with architecture and number of rnn layers visualized.

RNN dropout 0.4 seems to be a sweet spot where the potential of the model is more flexible. Like with the GRU it is wider than the other two, but the population also contains the highest scores.

Hidden state dim. The graph shows that models with a dim of 100 performed the best. This is perhaps indicative of the model being forced to abstract information to a greater degree with a more constrained hidden state, leading to better general performance.

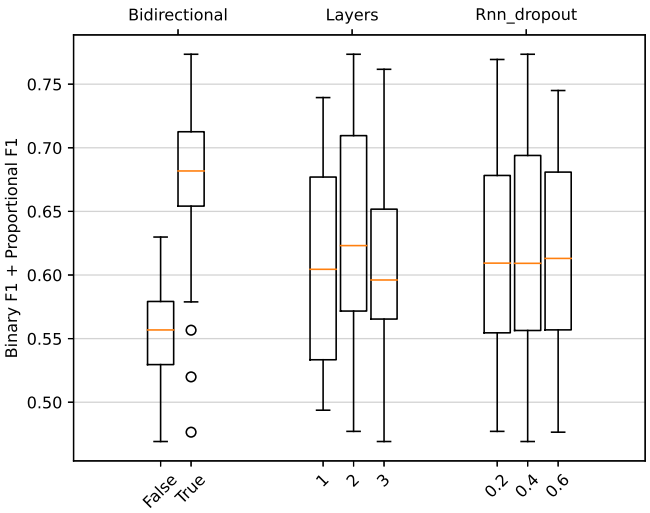


Figure 6: The distribution of F1 scores for bidirectionality, number of layers and rnn dropout for the *whole* model population.

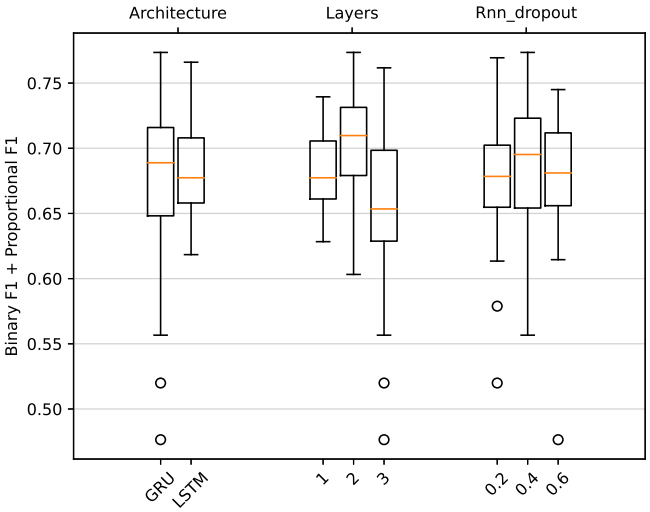


Figure 7: The distribution of F1 scores for architecture, number of layers and rnn dropout for the *bidirectional* model population.

Train embeddings It appears further training the embeddings results in a slight improvement of scores, perhaps due to word arrangements not encountered in the original training set.

RNN Layers The ideal number of layers appears to be 2. It seems logical to think that the model is not able to capture enough complexity with only 1 layer, but on the other hand might struggle to converge with 3.

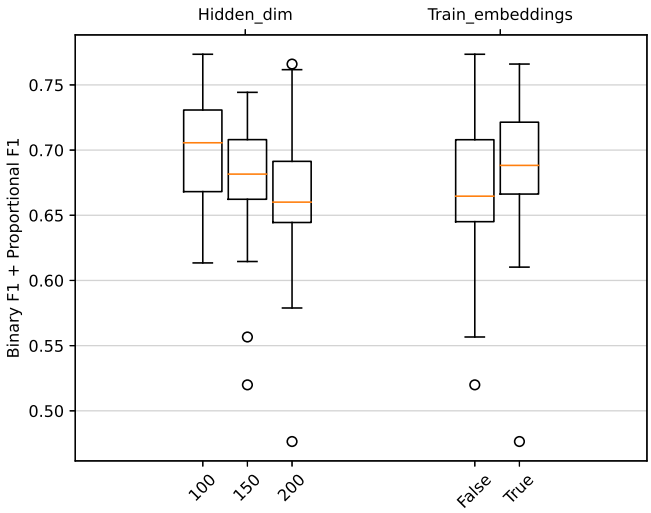


Figure 8: The distribution of F1 scores for hidden dim and train embeddings for the *bidirectional* model population.

Taking the results of this grid search into account, I believe the current best configuration would be a 2-layer Bidirectional GRU with RNN dropout of 0.4, hidden dim of 100, and train embeddings set to True. This fits with the results in Table 7, where both best-performing models have this configuration (apart from train embeddings, which is only active in one model).

6 Experiment 3: Ensemble

Inspired by the paper on Knowledge Distillation by Hinton et al. (2015), I wanted to see if an ensemble of models could outperform the best performing models of my population. I construct two methods of model cooperation:

Majority Vote This is a straightforward majority vote. All members of the ensemble evaluate the input and give their predictions, which are tallied. If a prediction ends in a tie, the first prediction of the tie wins. I take advantage of the default functionality of torch.argmax(), which returns the first of multiple max values.

Weighted Vote In this mode each individual member is assigned a weight to its vote based on its relative performance among the ensemble. The weight is calculated by taking the member's combined F1-score minus the ensemble's lowest combined F1-score, divided by the ensemble's combined F1-score span. This is then multiplied by a scale of 0.3 and added to the vote. The scale is designed to avoid too heavy an influence by higher-ranking members, and is the result of some hand-tuning.

$$weight = \frac{F1_{member} - F1_{ensembleMin}}{F1_{ensembleMax} - F1_{ensembleMin}} * 0.3$$

I run the entire 216-model population as well as the top 10 models as both majority and weighted ensembles, and report the results (See Table 9).

Metric	Binary	Proportional
216-Member Ensemble Majority		
Target Precision	0.529	0.380
Target Recall	0.334	0.201
Target F1	0.409	0.263
216-Member Ensemble Weighted		
Target Precision	0.539	0.386
Target Recall	0.340	0.206
Target F1	0.417	0.269
Top-ten Ensemble Majority		
Target Precision	0.517	0.357
Target Recall	0.504	0.314
Target F1	**0.510**	**0.334**
Top-ten Ensemble Weighted		
Target Precision	0.524	0.362
Target Recall	0.481	0.302
Target F1	0.502	0.329

Table 9: Performance of the initial ensembles, with best F1 scores in bold.

As we see in Figure 9, the ensemble scores worse than its best members if composed of the entire 216-model population. In this case the majority vote is the weakest, with the weighted vote being helped by its best performing members. However, if the top ten models are run as an ensemble, the results are quite dramatic (See Figure 10). The ensemble is stronger than any of its constituent members, with the ensemble majority vote seeing a 9.19% score increase over the best performing model in the whole population (as calculated by Binary + Proportional F1 score). In this case the weighted vote performs worse, indicating that it is not solely the high performance of a few top members that enable the predictive power of the ensemble, but likely their more diverse set of combined knowledge.

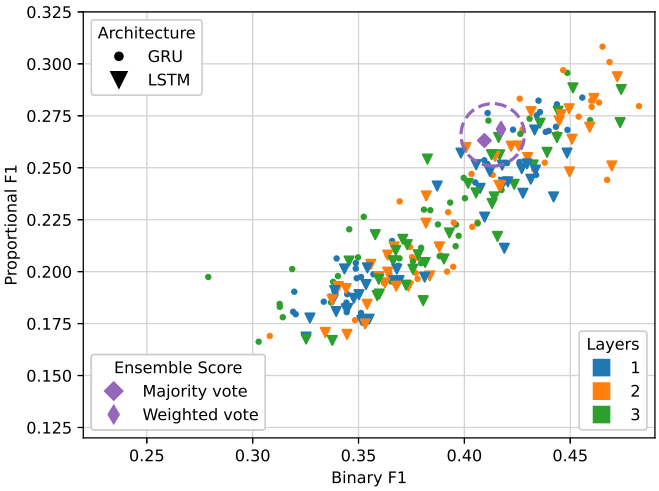


Figure 9: 216-model ensemble scores vs. individual member scores.

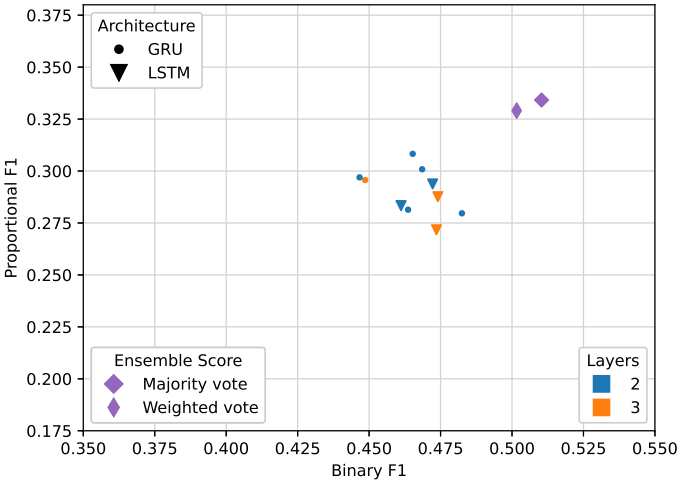


Figure 10: Top-ten model ensemble scores vs. individual member scores.

7 Experiment 4: Improving the Ensemble

Variable	Range
Optimizer	Adam, Adamax, Adamw
Learning rate	5e-4, 1e-3, 1.46e-3, 1e-2
Word dropout	0.01, 0.1, 0.3, 0.5

Table 10: Second grid search variables.

Considering the score-increase gained by both the ensemble and the grid search, I look into the possibility of further improving on these results.

For this experiment I essentially do ensemble incorporation in parallel with a hyperparameter grid search.

7.1 Grid Search

I first set the "best" hyperparameters found in experiment 2 as a baseline (2-layer Bidirectional GRU with RNN dropout of 0.4, hidden dim of 100, and train embeddings set to True), and for this second search sweep over the variables defined in Table 10. Due to the more constrained hyperparameters I anticipate the results to be closer together, and so I reduce the leeway (See 8) for saving of best model to 10%. With a lower learning rate I also increase the patience to 10 for this experiment.
Looking at the members of this new population of models, I find a smaller spread of scores (See Table 11) than with the previous search. The worst performing members are close in performance to the best of the first search, while the best clearly outperform the previous best. For Baseline + Proportional F1 score, we see a 10.6% score increase when comparing the best of both populations.

I analyze the distribution of population scores (See Figure 11) to draw some conclusions about the chosen hyperparameters:

Optimizer I try out a couple of variants of the standard Adam optimizer, available through pytorch. Adamax is *"a variant of Adam based on the infinity norm"* from the original Adam paper by Kingma and Ba (2014), and AdamW is from the paper *"Decoupled Weight Decay Regularization"* by Loshchilov and Hutter (2017).
Exploring the reasons for the performance difference seen here is beyond the scope of this paper, but it is clear that in this case Adamax performs better than standard Adam.

Learning rate When choosing learning rates, I was inspired by an article [3] in which the rate 1.43e-3 performed well with the Adam optimizer. Otherwise I chose the default rate for Adam in pytorch (1e-3), as well as half the default (5e-4), in addition to the baseline of 1e-2. The lower learning rates are certainly responsible for the much higher

[3] Medium.com article

Model	Optimizer	Learning Rate	Word Dropout	F1		Epoch
				Binary	Propor.	
Max Binary F1	Adamax	1e-2	0.5	**0.530**	0.325	42
Min Binary F1	Adamax	5e-4	0.1	0.436	0.280	27
Max Proportional F1	Adam	1e-3	0.5	0.511	**0.338**	10
Min Proportional F1	Adam	5e-4	0.01	0.444	0.236	24
Max B+P F1	Adamax	1e-2	0.5	**0.530**	0.325	42
Min B+P F1	Adam	5e-4	0.01	0.444	0.236	24
Median B+P F1 1/2	Adam	1e-3	0.1	0.466	0.298	12
Median B+P F1 2/2	AdamW	1.46e-3	0.01	0.475	0.289	4

Table 11: An overview of notable members of the second grid search model population. Best F1 scores in bold.

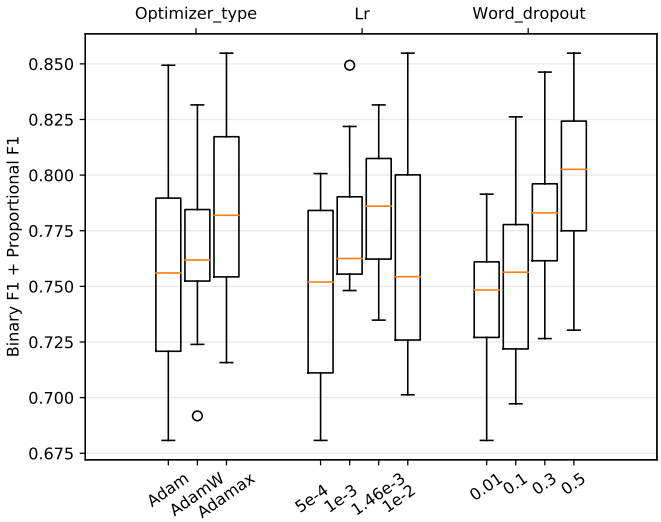


Figure 11: Second grid search F1-score distribution.

completion epochs seen in this population (See Table 11). It is also apparent that the rate 1.43e-3 results in high overall performance, although the absolute highest scores are found using the baseline rate.

Word dropout It appears the higher the dropout, the better the results. According to Goldberg (2017), the higher dropout might be allowing the model to more easily adapt to unknown words, as well as improving robustness and preventing overfitting.

7.2 Model Incorporation

The best result of the ensemble so far has been with a top-ten majority vote, so I keep this configuration going forward. For each new model that is produced in the grid search, the ensemble will iterate over itself, swapping out each member for the newcomer in turn and doing an evaluation. If the best of these configurations beats the scores of the highest performing ensemble so far, the newcomer is incorporated into the ensemble, discarding the old member.

Using the same method as when saving a best model for early stopping (See 8), the comparison of scores employs a strict leeway of 1.5%.

Surprisingly, the ensemble does not necessarily incorporate the models with the highest standalone performance. As we can see in Figure 12, sometimes a new member is recruited from a peak, while others come from relative valleys. It is likely the ensemble requires more breadth of knowledge to

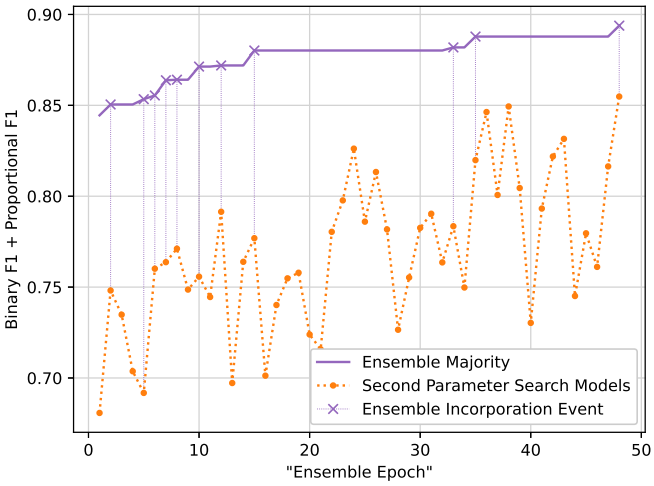


Figure 12: F1 scores improving as the ensemble incorporates new models into itself.

see improvement, and as such values variety over max performance.

Over a total of 50 new models, the ensemble incorporated a new member 10 times. Some of these newcomers were in turn replaced, as in the final tally the finished ensemble consist of 6 new members, with 4 old members still remaining. Visualiz-

ing the scores, we can see that the ensemble's performance is again greater than any of its constituent members (See Figure 13). The new ensemble sees a 5.8% score increase (as calculated by Binary + Proportional F1 score) over the previous top-ten ensemble. While the weighted vote was not used for selection of new ensembles, we can again see that it performs worse than the majority vote (See Table 12).

Metric	Binary	Proportional
Final Ensemble Majority		
Target Precision	0.542	0.410
Target Recall	0.516	0.329
Target F1	**0.528**	**0.365**
Final Ensemble Weighted		
Target Precision	0.549	0.414
Target Recall	0.483	0.308
Target F1	0.514	0.353

Table 12: Performance of the final ensembles, with best F1 scores in bold.

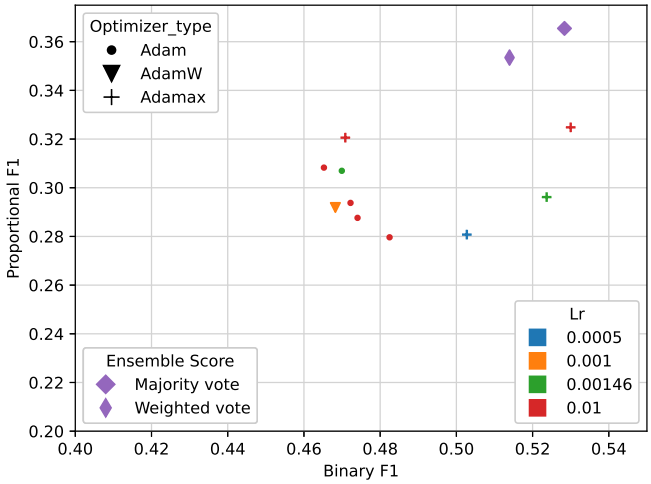


Figure 13: Final ensemble vs. individual member score.

8 Error Analysis

I evaluate the final ensemble on the dev set, and create a confusion matrix (See Figure 14).
The ensemble is good at identifying the Beginning and Inside of a target, but has a tendency to classify every target with a positive polarity (See Figure 15, first and second sentence). The system very rarely predicts a true Outside as anything else (although it can get confused by certain nouns and titles), but is more likely to classify true targets as Outside. An error often seen is the system stopping tagging halfway through a target, before picking up and finishing a word or two later (See Figure 15, first sentence).

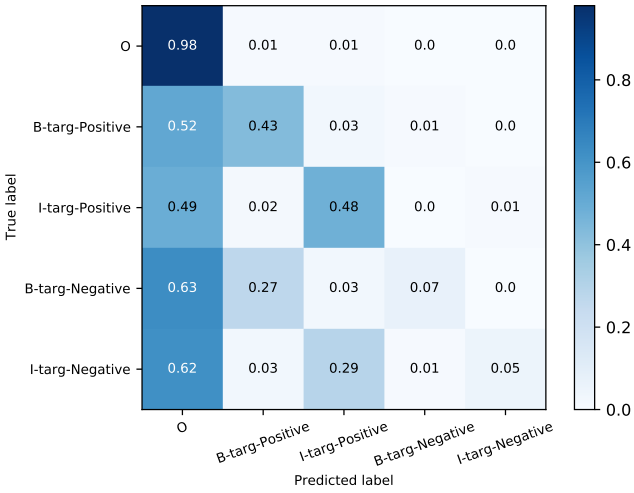


Figure 14: Simple confusion matrix for the final ensemble evaluated on the dev set. Normalized over x-axis.

It seems to have an understanding of the scope of a target, but is not always good at keeping the target internally consistent. In the same vein the system is often observed missing a Beginning, while completing the rest of the target with only I-tags.
The ensemble can get confused by words like "good" in the vicinity of a target, (classifying it as a positive), when in fact "good" was used in a negative context, such as "a good deal worse". In Figure 15 we see a representation of the internal voting of the ensemble when evaluating sentences. Although these were chosen for their interesting and representative errors, we can observe how the ensemble is very unsure for many of the targets, but votes unanimously for most of the O labels.

9 Evaluation

I evaluate the final ensemble on the held-out test set (See Table 13). The results are not quite as good as on the dev set, but the system still seems to have generalized fairly well. Looking at the predictions, it appears to display the same strengths and weaknesses as discussed in the error analysis section.

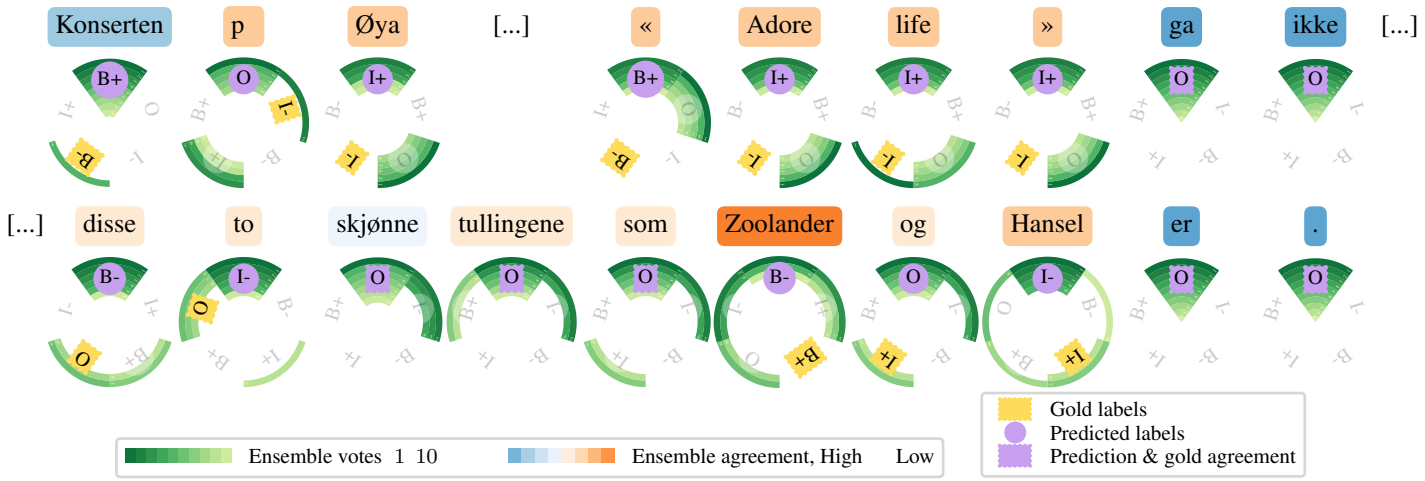


Figure 15: Three sentence predictions by final ensemble. The votes of the ensemble along with the predicted and gold labels are visualized in circle plot, rotated to emphasize predicted labels. Level of ensemble agreement is visualized with text highlighting. Labels are IOB with +/- indicating polarity. Further details such as member id on votes are visible if image is magnified.
The concert on Øya [. . .] *"Adore life" did not give* [. . .]
[. . .] *these two adorable loonies that Zoolander and Hansel are.*

Metric	Binary	Proportional
Target Precision	0.460	0.355
Target Recall	0.410	0.289
Target F1	0.434	0.318

Table 13: Evaluation of final ensemble on held-out test set.

10 Future Work

For a follow-up paper I would like to explore distillation of the ensembles into stand-alone models, as discussed in the paper by Hinton et al. (2015). This could drastically reduce the required processing power to run these systems, and would likely enable further experimentation.

11 Summary

In this paper we have seen a number of experimental changes producing systems increasingly more capable at the task of targeted sentiment analysis. The jump in F1 score from the initial BiLSTM baseline to the final ensemble is 23.9% when comparing the binary F1 scores, or an 88.1% increase when looking at the proportional F1 scores.

It is clear that good hyperparameter optimization is crucial to producing viable models, and while single models may be powerful, it appears there is collective wisdom to be gained when letting multiple models cooperate. This seems to consistently produce better results than any single member of an ensemble, as long as the performance gap between the members is not too large. The downside is the trading of efficiency for performance, with a 10-fold or more increase in processing power required to achieve a modest 5-10% score boost.
For a visual overview of the improvements seen throughout these experiments, see Figure 16.

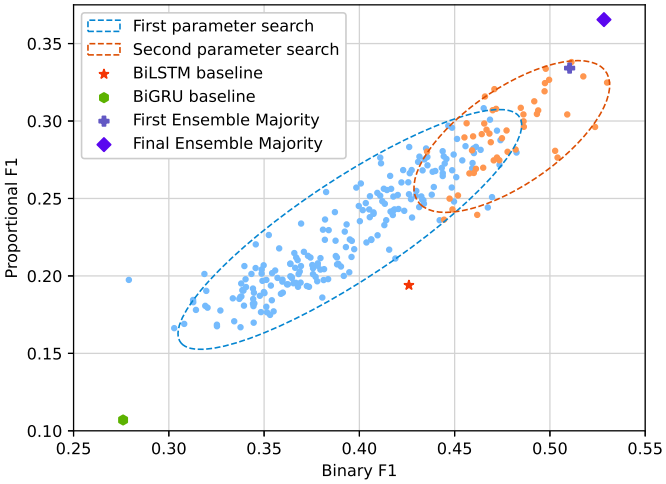


Figure 16: Overview of results from experiments.

References

Yoav Goldberg. 2017. *Neural Network Methods For Natural Language Processing*. Morgan & Claypool Publishers, San Rafael, CA, United States.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. In *NIPS 2014 Deep Learning Workshop*, Montreal, Canada.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *ICLR 2015*, San Diego, USA.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. In *ICLR 2019*, New Orleans, USA.

Dehong Ma, Sujian Li, and Houfeng Wang. 2018. Joint learning for targeted sentiment analysis. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4737–4742, Brussels, Belgium. Association for Computational Linguistics.

Margaret Mitchell, Jacqui Aguilar, Theresa Wilson, and Benjamin Van Durme. 2013. Open domain targeted sentiment. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1654, Seattle, Washington, USA. Association for Computational Linguistics.

Erik Velldal, Lilja Øvrelid, Eivind Alexander Bergem, Cathrine Stadsnes, Samia Touileb, and Fredrik Jørgensen. 2018. Norec: The norwegian review corpus. In *LREC 2018*, pages 4186–4191, Miyazaki, Japan.

Meishan Zhang, Yue Zhang, and Duy-Tin Vo. 2015. Neural networks for open domain targeted sentiment. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 612–621, Lisbon, Portugal. Association for Computational Linguistics.

Lilja Øvrelid, Petter Mæhlum, Jeremy Barnes, and Erik Velldal. 2019. A fine-grained sentiment dataset for norwegian.

Targeted Sentiment Analysis for Norwegian Language

Ping-Han Hsieh
Center of Molecular Medicine, University of Oslo
pinghanh@ncmm.uio.no

Abstract

In this study, we conducted several experiments to explore the potential to combine recurrent neural networks with different modelling strategies to improve the targeted sentiment analysis on recently released Norwegian $NoReC_{fine}$ dataset. Our experimental results showed that there is a marginal improvement between the baseline and our fine-tuned networks with collapsed modelling strategy. However, combining gated recurrent unit network with joint modelling yield noticeable improvement over the baseline model for targeted entities with negative sentiment.

1 Introduction

The internet has become the richest source of natural language data in recent years. Millions of text data from customer feedback, news comment, blog posts or movie reviews have been created every second. Therefore, it has become of interest to extract relevant information from these human-generated text sentences. One of the most important applications to exploit such dataset is sentiment analysis (SA), which aims at finding the sentimental expression for a given sentence. The task is usually a classification problem of two categories (positive and negative) or sometimes three (positive, negative and neutral) for better resolutions (Socher et al., 2013). However, the sentiment analysis on the coarse level of a sentence sometimes is not suitable for real-world applications, the simple assumption that the sentiments directed to the text aggregate to a single polar expression of the entire sentence might yield biased or unreliable conclusion from such analysis. Therefore, finer-grained sentiment analysis has become a more popular task to extract more specific and accurate emotions hidden in the sentences. There are multiple different ways to address finer-grained sentiment analysis problems. For instance, having detailed and specific subcategories (e.g. anger, sadness, depression) (Wang et al., 2016) or conducting sentiment analysis on textual segments (Zirn et al., 2011). In this study, we conducted several experiments to explore the potential of using recurrent neural networks to extract polar opinions in each individual targeted entity based on the recently published $NoReC_{fine}$ dataset (Øvrelid et al., 2019). The designed experiments evaluated the performance of different architectures with different hyperparameter settings, namely, long short-term memory networks (Hochreiter and Schmidhuber, 1997), gated recurrent units network (Cho et al., 2014) and transformers (Vaswani et al., 2017). We then compared the performance of different modelling strategies, including collapsed, joint and pipeline modelling (Mitchell et al., 2013; Zhang et al., 2015). We hope this study could bring insight into the targeted sentiment analysis (TSA) for the Norwegian language.

2 Related Works

2.1 NoReC (Fine)

In this work, Øvrelid et al. (2019) collected the text data from the Norweigan Review Corpus (NoRec) (Velldal et al., 2017), which consists of reviews written in Norwegian from multiple domains, such as games, movies, products, restaurants, etc. The authors then provided several guidelines for the annotation of a fine-grained sentiment dataset, including the span and the intensity for the polar expression, different categories of the modified target, and the subjects or holders for expressing the emotions. The annotation for the dataset was performed by native Norwegian speakers with linguistics background. Every sentence was annotated by two independent annotators and resolved by the third one if there's a con-

flict. The resulting dataset consists of 7,961 sentences in 298 documents, which provides the first valuable fine-grained sentiment dataset for Norwegian. The authors also provided a BiLSTM model which was later used as the baseline for targeted sentiment analysis in this study.

2.2 Open Domain Targeted Sentiment

In this study, Mitchell et al. (2013) applied three novel approaches to the targeted sentiment analysis based on Spanish and English Twitter Collection. The models were trained and tested using annotations derived from Amazon's Mechanical Turk Annotators. The three proposed models are (1) *pipeline*: build a model to predict the entity labels; thereafter applied a second model to predict the sentiment labels, (2) *joint*: the targeted entity labels and the sentiment labels are predicted jointly and (3) *collapsed*: the model predict one label sequence which combine entity labels and sentiment labels. Unlike previous studies that predict the sentiment and entities independently, the authors introduced the task of open domain targeted sentiment, which predict the targeted sentiment corresponding to different entities.

2.3 Neural Networks for Open Domain Targeted Sentiment

Following the study of Mitchell et al. (2013), Zhang et al. (2015) further extended the conditional random fields baseline with neural network architectures. Apart from this, the authors also compare discrete and continuous embeddings for the word in targeted sentiment analysis. Lastly, the authors proposed a novel method which integrates both kinds of embeddings features to improve the performance. The authors showed that using neural networks could slightly improve the model. In addition, the integrated model significantly outperforms other models. This study demonstrated the strength of applying neural network in the task of targeted sentiment analysis.

	Train	Dev	Test
O	91690	18335	14424
B-targ-Pos	2244	432	365
B-targ-Neg	1093	195	144
I-targ-Pos	2338	435	347
I-targ-Neg	1093	206	116

Table 1: Numbers of the collapsed label in each dataset

3 Methods

3.1 Dataset

We performed targeted sentiment analysis based on the recently released $NoReC_{fine}$ dataset. The texts in the original dataset have been annotated with targets, holders and polar expressions. However, in this particular study, we focused only on the targeted entity and the corresponding polarity of expression (*O, B-targ-Pos, B-targ-Neg, I-targ-Pos* and *I-targ-Neg*), where labels *O*, *B-targ* and *I-targ* represent tokens outside the target entity, the beginning of the target entity and part of the target, respectively. *Pos* and *Neg* represent the sentiments target towards the token. The numbers of each label in the dataset are shown in Table 1. The table shows that our predicted label is extremely unbalanced with most of the tokens are outside the targeted entities. Moreover, we can also observe more positive sentiments than negative ones.

We further explored our training dataset by analyzing the words being categorized to more than one collapsed label in different sentences. The probability of these words being assigned to each label is shown in Figure 1. Among 17,883 words in the training data, there are 1,843 words being categorized to more than one collapsed label. 1,779 of them being categorized as tokens outside of target (*O*) in at least one sentence. Therefore, it is essential to distinguish labels *O* from the others in order to build a reliable targeted sentiment analysis system in this particular dataset.

3.2 Baseline and Neural Network Architectures

In this study, we conducted multiple experiments of neural network-based methods in targeted sentiment analysis on $NoReC_{fine}$ dataset. We compared the performance of different models with the provided baseline model mainly using the *proportional F1*. The proportional F1 assigns precision and recall as the ratio of overlap with the predicted and gold span respectively, reducing the bias of occurrence of each token presented in the dataset (Øvrelid et al., 2019). We found this measure is a better choice to evaluate our models since the distribution of the tokens is heavily unbalanced (with 64.1% of the words only occur once while 0.1% of the words occur over 760 times in the training data) in the dataset. For the feature representation of each word, we used the pretrained FastText (Bojanowski et al., 2016) word embeddings

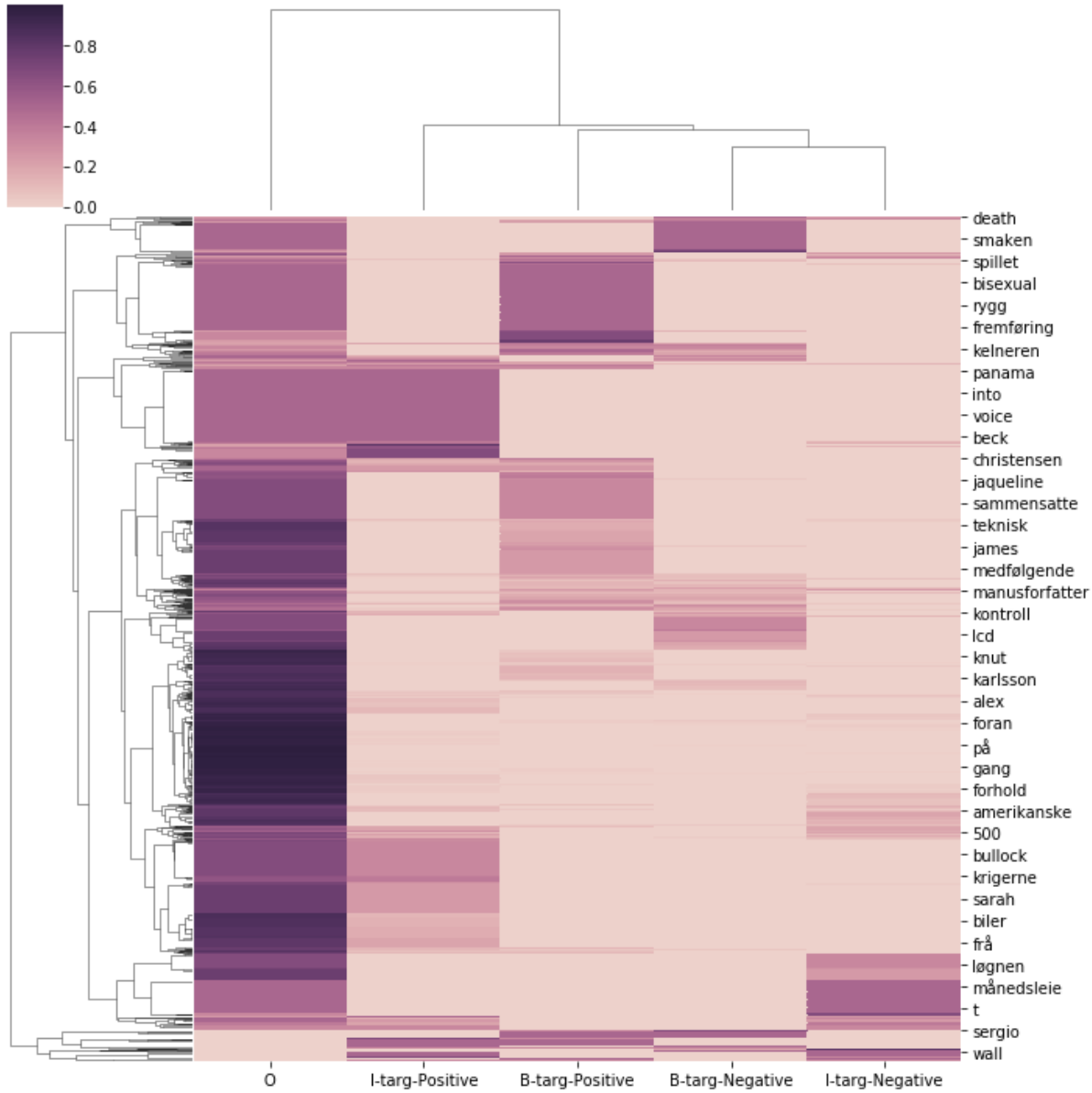


Figure 1: Probabilities of the words (which were categorized to more than one label in different sentences) being categorized to each collapsed label

which consist of 515,788 Norwegian words from Wikipedia. Each word is represented with a 100-dimensional real number vector. We experimented three neural network architectures, namely bidirectional long short-term memory (LSTM) network (Hochreiter and Schmidhuber, 1997), bidirectional gated recurrent unit (GRU) network (Cho et al., 2014) and transformer (Vaswani et al., 2017) (with fixed 10 multi-head attentions). In terms of hyperparameter tuning, we particularly evaluated how the number of hidden layers (1, 2, 3 and 4) and the number of neurons (50, 100 and 200) in each hidden layer influence the performance of our models. All of the networks were trained with $dropout = 0.01$ in the tunable word embedding layer. The parameters for each model is optimized using Adam (Kingma and Ba, 2014) with *learning rate*$= 0.005$ and *batch size*$= 128$. *Weighted cross entropy* were used as the loss function in the training process with class weights being assigned inversely proportional to the frequencies of the labels (Table 1) in order to reduce the impact of unbalance label distribution shown in the data.

3.3 Collapsed, Joint and Pipeline Modeling

Apart from different architectures of the neural networks, we also trained our networks with three different modelling strategies: (1) collapsed, (2)

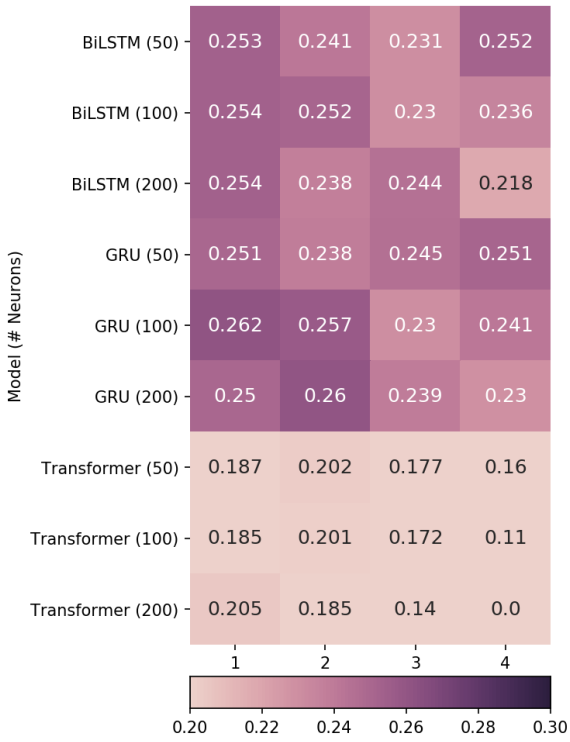


Figure 2: Proportional F1 score evaluated on the development dataset using collapsed modelling strategy.

joint and (3) pipeline modelling using the similar concept from Zhang et al. (2015). In collapsed modelling, we predicted the labels of each token which combine the meanings of target entities and the sentiments (*O, B-targ-Pos, B-targ-Neg, I-targ-Pos* and *I-targ-Neg*). In joint modelling, the target entities (*O, B-targ, I-targ*) and the sentiments *NULL, Pos, Neg* are separated into two different kinds of labels but the models predict them jointly. Similarly, pipeline modelling separates the target entities and the sentiments into different labels but separate models are used to predict the two different kind of labels. Note that the NULL sentiment is assigned to O entities automatically in joint and pipeline modelling.

4 Experiments and Results

4.1 Collapse Modeling with Different Network Architectures

In our first experiment, we evaluated the performance on different neural network architectures (see Methods 3.2 for more details) with different hyperparameter settings (number of layers and number of neurons) using the collapsed modelling strategy. The result of our experiment is shown in Figure 2. For the BiLSTM network, the performance is similar with different numbers of neurons when there is only one hidden layer. The performance fluctuates when the number of hidden layers increases, and there's no significant improvement when using more hidden layers. Similar result can be observed in the GRU network. We observed slight improvement over the baseline model (with proportional F1 score 0.254) using the GRU network with 100 neurons in one hidden layer. We then looked into the precision and recall for each label predicted using this setting (Table 2). Among the five labels, *O* show the highest precision and recall, while the four other classes performed comparatively poor (F1 score ranging from 0.14 to 0.31), showing that it is still relatively difficult for the networks to predict the target entities of tokens correctly.

	Precision	Recall	Count
O	0.97	0.92	18335
B-targ-Pos	0.22	0.49	432
B-targ-Neg	0.11	0.18	195
I-targ-Pos	0.27	0.37	435
I-targ-Neg	0.13	0.16	206

Table 2: Precision and recall for each label evaluated on the development dataset using collapsed modelling strategy with best hyperparameter setting for GRU network.

Transformer network performed the worst among the three different network architectures, with the best proportional F1 score saturated around 0.20. When the number of hidden layers increases, the model failed to distinguish the difference between labels and predicted *O* for nearly all the tokens. We suspect the drop of performance for transformer networks might result from our implementation from the original study (Vaswani et al., 2017), where the attention mask was included in order to prevent the network from peeking into the words present in the latter part of the sentence when computing self-attention. Although it is the standard operations in building the seq2seq model, this might lead to losses in information when applying on targeted sentiment analysis.

4.2 Joint and Pipeline Modeling using GRU

Because of the superior performance of the GRU networks showed in the previous section, we decided to use it as the neural network architecture to continue the evaluation on the performance with

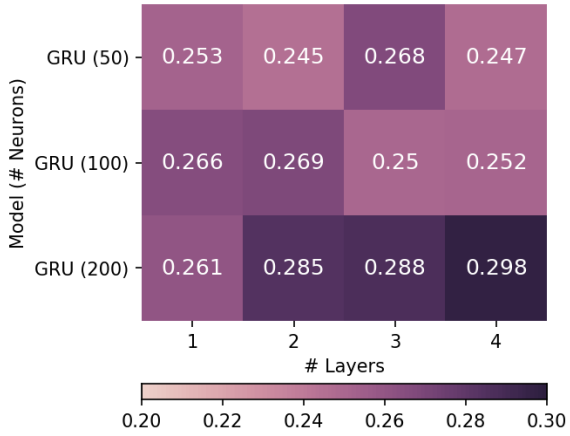


Figure 3: Proportional F1 score evaluated on the development dataset using joint modelling strategy.

different modelling strategy. Similar to the previous section, we trained our networks with different numbers of hidden layers and neurons. The experimental results for joint modelling can be found in Figure 3. Unlike collapsed modeling, we can observe that when the number of neuron in the hidden layer increases, the models generally make better predictions. The best hyperparmeter setting for GRU network (200 neurons with 4 hidden layers) using joint modeling strategy outperformed the ones using collapsed modeling with proportional F1 score close to 0.3. The precision/recall analysis for the best GRU network (4 hidden layers with 200 neurons) showed that there is a noticeable improvement (42.8% and 33.3% improvement of F1 score for B-targ-Neg and I-targ-Neg, respectively) for predicting targeted entities with negative sentiments (Table 3).

For pipeline modelling, we trained two independent networks to predict the labels of sentiments and targeted entities. The predictions for two different kinds of labels are viewed as independent tasks and the two networks are fine-tuned using different hyperparameters. The proportional F1

	Precision	Recall	Count
O	0.97	0.91	18335
B-targ-Pos	0.23	0.52	432
B-targ-Neg	0.18	0.24	195
I-targ-Pos	0.23	0.56	435
I-targ-Neg	0.21	0.19	206

Table 3: Precision and recall for each label evaluated on the development dataset using joint modelling strategy with best hyperparameter setting for GRU network.

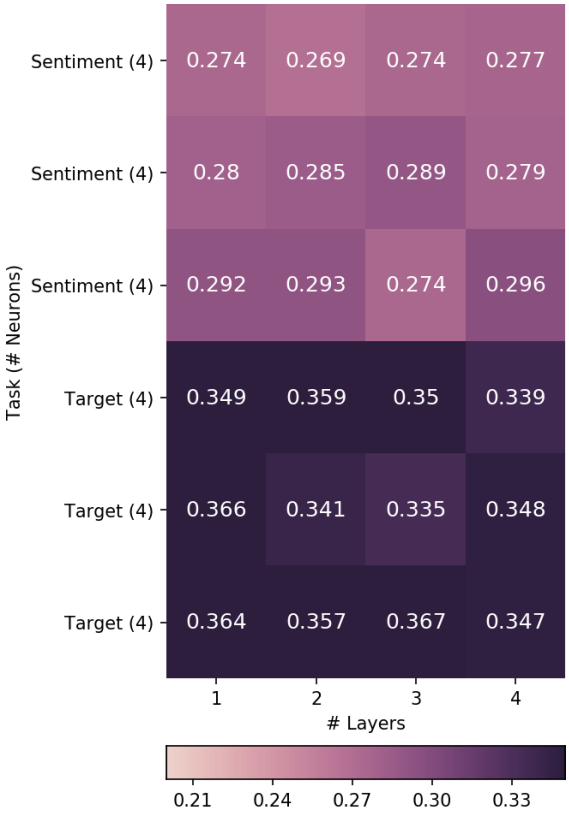


Figure 4: Proportional F1 score evaluated on the development dataset for sentiments and target entities prediction used in pipeline modelling strategy.

scores evaluated on the development dataset for the two prediction tasks are shown in Figure 4. The experimental results show that it is comparatively difficult to predict correct sentiment target to the tokens than the targeted entities. For both prediction tasks, there are only marginal impacts on the number of hidden layers. In contrast, using more number of neurons in the hidden layers seems to have positive influences on the performance of the models. Finally, we combined the GRU networks with the best hyperparameter settings from both tasks (200 neurons in 4 hidden layers for the sentiment prediction and 200 neurons in 3 hidden layers for the targeted entity prediction) to make our final prediction. Our final pipeline model slightly outperforms the baseline model with a 0.280 proportional F1 score. However, the precision/recall analysis show quite limited improvement over the baseline model.

5 Future Work

Although we have evaluated the impact of the number of hidden layers and neurons on differ-

	Precision	Recall	Count
O	0.96	0.94	18335
B-targ-Pos	0.26	0.45	432
B-targ-Neg	0.18	0.17	195
I-targ-Pos	0.28	0.37	435
I-targ-Neg	0.13	0.09	206

Table 4: Precision and recall for each label evaluated on the development dataset using pipeline modelling strategy with best hyperparameter settings for GRU networks from the two prediction tasks.

ent neural network architectures along with three modelling strategies, there are still improvements we can make in the future, such as trying gradient clipping in the optimization process, applying different initialization methods, evaluating the influence of batch size and optimization methods, etc. These are potential improvement we can make in the future. In addition, we have failed to train the transformer network properly for this particular problem, we suspected the issue might result from our code implementation. However, using different attention mechanism and different numbers of multi-head attention might also be essential to train the transformer networks. Finally, using state-of-the-art pre-trained deep contextualized word representation model such as BERT (Devlin et al., 2018) and ELMo (Peters et al., 2018) are also potential ways to further improve the targeted entity analysis on the $NoReC_{fine}$ dataset.

6 Conclusion

In this study, we have performed multiple experiments to explore the possibilities of combined recurrent neural networks with different modelling strategies to improve targeted sentiment analysis on $NoRec_{fine}$ dataset. We have shown noticeable improvement using GRU and joint modelling strategy over the baseline collapsed model. In addition, there are some advantages to using joint and pipeline modelling strategies. For instance, researchers can gain more reliable predictions on the targeted entities for the tokens even when the model failed to predict the correct sentiments target to that token. Despite showing better performance over the baseline model, we, however, have to admit that the improvement is somewhat limited and it still remains challenging to correctly predict negative targeted sentiments for all of our proposed models. We anticipate this study can give relevant insights into the related domain for the Norwegian language.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Margaret Mitchell, Jacqui Aguilar, Theresa Wilson, and Benjamin Van Durme. 2013. Open domain targeted sentiment. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1654.

Lilja Øvrelid, Petter Mæhlum, Jeremy Barnes, and Erik Velldal. 2019. A fine-grained sentiment dataset for norwegian. *arXiv preprint arXiv:1911.12722*.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Erik Velldal, Lilja Øvrelid, Eivind Alexander Bergem, Cathrine Stadsnes, Samia Touileb, and Fredrik Jørgensen. 2017. Norec: The norwegian review corpus. *arXiv preprint arXiv:1710.05370*.

Zhaoxia Wang, Chee Seng Chong, Landy Lan, Yinping Yang, Seng Beng Ho, and Joo Chuan Tong. 2016. Fine-grained sentiment analysis of social media with emotion sensing. In *2016 Future Technologies Conference (FTC)*, pages 1361–1364. IEEE.

Meishan Zhang, Yue Zhang, and Duy-Tin Vo. 2015. Neural networks for open domain targeted sentiment. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 612–621.

Cäcilia Zirn, Mathias Niepert, Heiner Stuckenschmidt, and Michael Strube. 2011. Fine-grained sentiment analysis with structural features. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 336–344.

Exploring Pre-Trained Model Strategies for Open Domain Targeted Sentiment Analysis

Adrian Tysnes
Department of Informatics, University of Oslo
adrianty@ifi.uio.no

Abstract

Few tasks have as much unlocked potential as open domain targeted sentiment analysis (ODTSA), a somewhat less complex version of Aspect-Based Sentiment Analysis (ABSA) due to the classification being binary single-category. While relatively good results can be achieved in both sentiment analysis on sentence- and document level, and named entity recognition (NER), ODTSA represents such a complex problem that even state-of-the-art results are struggling to produce results anywhere near more basic implementations of the aforementioned tasks (Sun et al., 2019). While this paper will focus on ODTSA as a sequence tagging problem, it must be mentioned that this is far from the only way of approaching the problem (Hu et al., 2019). I will follow this tradition today, experiment with the different strategies, and discuss the potential effect of introducing Bidirectional Encoder Representations from Transformers (BERT)-models to these experiments.

1 Motivation and Goals

ODTSA has many areas of use across the NLP-spectrum. It is very useful for analysing sentiment on platforms that are inherently informal, like online forums, comment sections, and twitter, to name a few (Mai and Le, 2020; Devi and Bhaskarn, 2012; Hu et al., 2019). It is common to approach the problem with three different model strategies: PIPE, JOINT, and COLLAPSED. (Mitchell et al., 2013; Hu et al., 2019; Zhang et al., 2015). Whereas the COLLAPSED version treats it as a pure sequential tagging task, attributing tags with both a value for the entity recognized and the semantic polarization, the JOINT and PIPE methods separates the tasks. The main difference between these two is that while a JOINT approach will typically employ a RNN-layer for the sentiment polarization, a PIPE approach will split the process into two separate models. For a more precise definition of the different strategies, I refer to Mitchell et al. (2013). I have visualized the different models in tabels 1, 2, 4, and 3. While many studies on the topic of ODTSA shows that the PIPE model strategy will produce the best results on average, there is definitely not a clear consensus and results vary on the task and the dataset available (Zhang et al., 2015; Hu et al., 2019). For example, Mitchell et al. (2013) found that when including little data for the training process, the COLLAPSED model would outperform the other two. Another interesting opportunity that will be discussed in this paper is the use of BERT-models. Devlin et al. (2018) showed that adding bidirectionality to pre-trained models opened for many opportunities in comparison to the established state-of-the-art models that had necessary unidirectional restrictions. Research has shown that utilizing BERT for these tasks can be useful if applied correctly, but has little effect if not attentive to details and for the best results data should be preprocessed before being used as input to a BERT model for ODTSA (Sun et al., 2019). While it might be challenging, BERT has a lot of promise, and exploring it in new settings should provide interesting results.

2 Introduction: Task Definition and Data

The dataset that I am going to use in this paper is one presented in Øvrelid et al. (2019). It is a fine grained dataset that is annotated with entity recognition and sentiment polarity. As stated previously, the task of ODTSA can be approached in different ways. The most straight forward way it to treat it as a sequence tagging task, identifying each token in the sequence as either a non-entity, the beginning of an entity, or a non-beginning part of an entity, and then applying sentiment polarization by either classifying it as expressing positive or nega-

Tag	O	O	O	O	O	O	O	B-targ-Positive	I-targ-Positive	I-targ-Positive
Input	Alt	dette	kommer	fiks	ferdig	i	dette	settet	fra	Kingston

Table 1: The structure of a collapsed model.

Sentiment								Positive	Positive	Positive
Entity	O	O	O	O	O	O	O	B-targ	I-targ	I-targ
Input	Alt	dette	kommer	fiks	ferdig	i	dette	settet	fra	Kingston

Table 2: The structure of a joint model. The blank cells in the sentiment row are not used for sentiment polarization for the entities recognized in the former layer.

tive sentiment towards the entity. Hu et al. (2019) point out that this suffers from huge space search and sentiment inconsistency, and use a heuristic multi-span decoding technique to improve performance. This could be interesting to explore in the future. There were some issues with inconsistencies in the dataset, that I will get back to in section 4. The dataset is divided into three parts, a train set that contains roughly 75% of the examples, a development set that contains roughly 15%, and a test set that contains roughly 10%. An inherent problem with sequence tagging such a dataset is obviously that the occurrence rate of the potential tags is extremely inconsistent. There are many more non-entity tags than all the entity tags combined, in the training set the number is roughly thirteen times, with similar ratios for the other sets. Future research on the dataset should explore the possibilities of utilizing sampling techniques to improve the tagging process, as discussed in Pelayo and Dick (2007).

An great example of the structure of the dataset can be found at table 5. It portrays a sentence from the dataset regarding some product from the computer hardware producer Kingston. This product, is recognized as the entity formed of the words ”settet fra Kingston”, because all the three last words directly describes the product in question. It is a good example of the inherent difficulties in the task, as none of the words in themselves have anything to do with the product from Kingston, except perhaps the brand name. Then again, Kingston is both the capital of Jamaica and a fairly common name in some countries. The entity is recognized without the name of the specific product, or even the type of product. As for sentiment polarity, the word ”fiks” can easily be found in both positive and negative settings towards a product. It isn't hard to imagine a sentence meant to fix something because it is broken, thus carrying negative sentiment. In our sentence however it is clearly used in a positive manner.

3 Background: Related Work

I have already mentioned several different works related to ODTSA, and in this section I will try to structure the study of the topic chronologically. Lafferty et al. (2001) introduced Conditional Random Fields (CRFs) as a way to improve upon the at-the-time most used Maximum Entropy Markov Models. The advantage of CRFs is that they allow for producing an exhaustive probability distribution for all potential tags in a sequence. While the original implementation was introduced as a way to deal with sequence labeling at a more general level, Mitchell et al. (2013) has shown that it can be applied to ODTSA to perform the entity extraction portion of the process. Applying CRF to a model used on this dataset is an intriguing opportunity, but it is not one that will be discussed further in this paper. More recently convolutional neural networks (Xu et al., 2018), and RNNs (He et al., 2018), have been used for sequential tagging problems.

As for sentiment analysis, there are a wide array of options. The research methods differ based on the granularity of the data, whether it is document-level (Dou, 2017), sentence-level (Chen et al., 2016), or as in our case, token-level (Zhang et al., 2015). It is important to note that while ODTSA is, as its name suggests, open domain, a lot of studies use a rather narrow dataset, often times consisting of posts from internet forums that will inherently be of similar topics (Devi and Bhaskarn, 2012), or platforms like Twitter where the data will contain similar sentence structure due to restrictions of length (Hu et al., 2019). Our dataset on the other hand contains many reviews, but of many completely different topics, ranging from concerts to computer hardware. That does mean that the results we can expect from our model will likely

Sentiment	Positive	Positive	Positive
Entities	settet	fra	Kingston

Table 3: The structure of a pipe model (2).

Entity	O	O	O	O	O	O	O	B-targ	I-targ	I-targ
Input	Alt	dette	kommer	fiks	ferdig	i	dette	settet	fra	Kingston

Table 4: The structure of a pipe model (1).

Word	Label
Alt	O
dette	O
kommer	O
fiks	O
ferdig	O
i	O
dette	O
settet	B-targ-Positive
fra	I-targ-Positive
Kingston	I-targ-Positive

Table 5: Exanple of a sentence in the dataset.

be lower than some of the other state-of-the-art research shows on such narrow datasets.

BERT has been proven to provide good results for various NLP tasks (Devlin et al., 2018) (Adhikari et al., 2019) (Hoang et al., 2019). However, as Sun et al. (2019) points out, the complexity of ODTSA means that BERT models generally will not produce state-of-the-art results without some processing. As will be laid out in the next couple of sections, I was unfortunately not able to get that far in my research, but it is something that should be focused on in future research.

4 Setup: A Replication Study

Most of the problems that I have encountered with this task were implementary. The first thing I sought to do was to test out the Norwegian BERT model we were introduced for in the assignment description [1]. There were a few initial problems that had to be dealt with. First of all, the downloadable Norwegian model was in a TensorFlow-format, and required use of a script found deep within the Transformers library to convert to a useable pytorch model, required to load their models[2]. The second problem I faced with the Norwegian BERT was that it generated the vocab slightly incorrectly, causing all indices generated by the tokenizer to appear as unknown. Luckily it was relatively easily fixable through removing the text using the Python regex module. The next issue I faced was that the dataset in use is somewhat inconsistent in its use of symbolic characters, especially hyphens. It took a lot of time to tend to all the edge cases, but it generally took more time than effort. After having done this, and tested that the tokenizer would produce the correct result, and having ensured that everything was processed correctly, it turns out that the model is probably flawed and does not work at all. Some other person in the Github issues page having the same issues enforces my belief that at this point, the problem is not on my side. There were also some difficulties with using pretrained models from the Transformers library on the Saga-server we have access to, as it would not download the model for me, which is required to run the code. I ended up downloading the TensorFlow-model from their website, using the script [2] mentioned previously to turn it into a pytorch model. In sections 6 and 7 I will discuss the potential effects this model could have had, as well as experiments with other Transformers models.

The unfortunate conclusion of my struggles with the BERT implementations is that a lot of my code is relatively half-complete and suffer from having gone through fundamental changes in trying to salvage some of it for a working implementation. I will include all my code in the GitHub repository, but it is unfortunately somewhat disconnected and only semi-functional. I really have been in such a hurry to concoct some working code that it will be in a very poor state in terms of structure. I do not have the time to properly go through and assess which files reproduce exactly which experiments, but I do for certain have the different baselines for the COLLAPSED, JOINT, and PIPE models in three separate files, named hereafter. Some of

[1] https://github.com/botxo/nordic_bert

[2] https://github.com/huggingface/transformers/blob/master/src/transformers/convert_bert_original_tf_checkpoint_to_pytorch.py

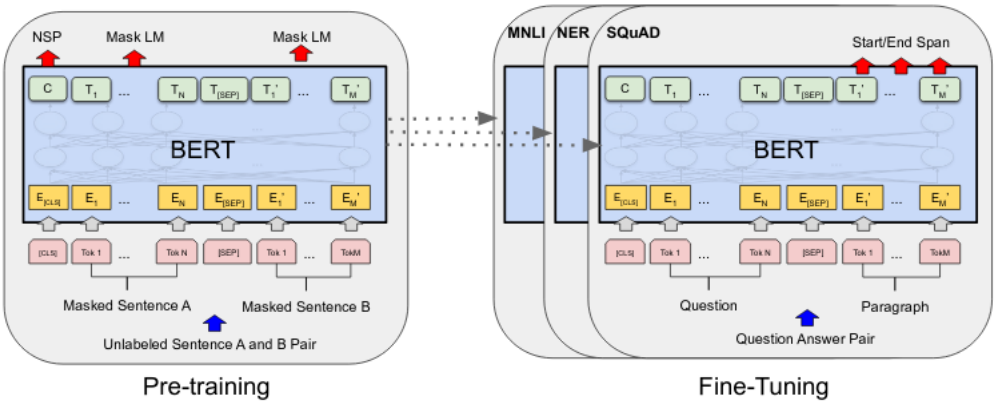


Figure 1: Bert works in two steps. First a model is pre-trained, and then it is employed in other models, and fine-tuned with the labels of the overlying model for the task and domain. Taken from Devlin et al. (2018).

my BERT-code will be a mix between what I tried to get to work for my Nordic BERT experiments, and for my multilingual BERT experiments. This is less than ideal, but not disastrous as I have not properly presented my findings from these scripts due to their volatility of the models that I could produce, and I have rather focused this paper on what I believe BERT-models could achieve. I did also include some of the files that SAGA produced, to show some of the data produced. For my experimentation I used the Norwegian Bokmål CoNLL17 word vectors as pre-trained embeddings. Due to restrictions in both computational power and time, I did not attempt to train these embeddings, or from scratch for that matter, on our dataset. Training embeddings on the dataset could certainly be interesting, and is left out for future research.

5 Experimental Results

Rather than experimenting with hyperparameters for the baseline model, I did a lot of experimentation with the representation of the data during the implementation of the JOINT and PIPE models. For the JOINT model, I attempted to use the embeddings only, the logits from the entity recognition model, and finally, the concatenation of the two. There was not too much difference in the results, but after having tried the same thing with the PIPE model, it looked to provide the best binary results. It is somewhat unstable, as the results would vary more than the COLLAPSED model, and could probably be stabilized and improved with some further research. For example, trying to utilize some data from a base BERT models 12 layers could be very interesting. There is probably more useful information than what is portrayed in the final, three dimensional output of the entity recognition process.

6 Architecture Variations

The original baseline we were given with the assignment was a COLLAPSED bidirectional Long Short Term Memory (BiLSTM). In our setting, COLLAPSED means that the prediction is performed by a single model, predicting both the the entity type, whether it's the beginning of an entity or a non-beginning part of one, and the sentiment polarization in a single step. I then implemented a JOINT architecture, where I would have one BiLSTM-layer for entity recognition, and one BiLSTM-layer for sentiment polarization, only providing the second layer with the entities the first layer would recognize. I then created a new class representing a very simple entity recognizer, and used it in a PIPE model, where my BiLSTM would provide sentiment polarization for the found entities. The results are shown in tables 6 and 7.

	COLL	JOINT	PIPE
Precision	**0.356**	0.313	0.318
Recall	0.377	0.412	**0.420**
F1	**0.367**	0.356	0.362

Table 6: Binary results on the test set for the different strategies. Best results in bold

After I had finally made the decision to go away

	COLL	JOINT	PIPE
Precision	**0.239**	0.216	0.122
Recall	0.213	**0.246**	0.128
F1	0.225	**0.230**	0.125

Table 7: Proportional results on the test set for the different strategies. Best results in bold

from the Nordic BERT-model, I basically had to scrap my entire implementation. There were too many edge cases that I had to implement fixes for in the Nordic one, that were not applicable for any other. I eventually employed the multilingual base BERT, as mentioned in section 4. At this point I was unfortunately pressed for time, and ran into computational issues. The use of GPUs are absolutely necessary when training models with BERT, potentially reducing the time required by tens, if not hundreds of times. The only real access to GPUs that I had was through the Saga cluster, which was at times very busy. There was also an issue with fine-tuning the models, as it would ask for more memory from the GPU than it could allocate. I tried to do some research on the matter myself, but the problem persisted. The models that I have been able to generate are practically useless, as they cannot produce any results even remotely close to the baseline strategies, to a point where I believe that there is an issue with the implementation that I am not able to fix. The best result I was able to produce managed a binary F1 of 0.233, and a proportional F1 of 0.034. It therefore makes more sense discussing likely results of a functioning multilingual BERT model for our dataset.

There is reason to believe that one of the main things that a functioning BERT-model could have improved in my results in tables 6 and 7, is the performance of the PIPE model, and in particular the proportional results. Studies have shown that BERT models are very good at sequence labeling tasks (Tsai et al., 2019), and at least with the implementation of some sampling techniques as briefly mentioned in section 2, sequence labeling should have been significantly improved. As for sentiment analysis, quite a lot of fine-tuning would probably be required to see a significant increase in quality. Given BERT models strength in sequence labeling, a possible improvement could come from part-of-speech-tagging the words as a pre-processing step. Because of the large potential in improvements from the baselines in tables 6 and 7, part-of-speech-tagging could provide a nice boost due to the vagueness removed, even if it would get some part-of-speech-tags wrong.

7 Reflections: Error Analysis

There are two things that stand out from the results from section 6, the first is how the PIPE model performed extremely poorly when evaluating it proportionally as done in table 7. That makes sense, as I decided to not spend too much time trying to create an optimalized NER-model for this task. Given that the PIPE results in table 6 when evaluating binary are fairly similar to the others, it must be assumed that it would perform more similarly to them proportionally. The second thing that is very noticable, is how in contrast to the COLLAPSED model that performs very stable in terms of precision and recall, the JOINT and PIPE architectures have large discrepancies between them. This could suggest that there are issues related to my implementation, or merely that the hyperparameters chosen are much more suited for a collapsed model than a PIPE or JOINT one. What is interesting about the collapsed model is that it might have been somewhat lucky in its fit towards the test set, at least in terms of F1 proportional results. It holds a significantly higher recall, and thus F1 score, on the test set, as compared with the dev set during training. While the same tendencies can be observed for the JOINT model, the difference between the last dozen epochs of the dev set and the test set was less significant than what can be observed for the COLLAPSED model.

8 Conclusions and Outlook

It is difficult to conclude anything without numbers that I have faith in. I have probably learned more from experimenting on this dataset than what I can show, which is what I have tried to demonstrate throughout the paper. I think that the dataset has a lot of interesting attributes, and that there are a lot of potential experiments that I believe could produce good results. I believe that I have been able to pinpoint some of the most prosperous possibilities from what has been done in state-of-the-art research, and I believe that I have been able to contextualize said possibilities to the dataset. I have shown that different modeling strategies have different advantages, and discussed how they could be utilized to further improve the results.

References

Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. 2019. Docbert: Bert for document classification.

Tao Chen, Ruifeng Xu, and Xuan Wang. 2016. Improving sentiment analysis via sentence type classification using bilstm-crf and cnn. *Expert Systems with Applications*.

K. Nirmala Devi and V. Murali Bhaskarn. 2012. Online forums hotspot prediction based on sentiment analysis. *Journal of Computer Science*, 8:1219–1224.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding.

Zi-Yi Dou. 2017. Capturing user and product information for document level sentiment analysis with deep memory network. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 521–526, Copenhagen, Denmark. Association for Computational Linguistics.

Luheng He, Kenton Lee, Omer Levy, and Luke Zettlemoyer. 2018. Jointly predicting predicates and arguments in neural semantic role labeling.

Mickel Hoang, Oskar Alija Bihorac, and Jacobo Rouces. 2019. Aspect-based sentiment analysis using BERT. In *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, pages 187–196, Turku, Finland. Linköping University Electronic Press.

Minghao Hu, Yuxing Peng, Zhen Huang, Dongsheng Li, and Yiwei Lv. 2019. Open-domain targeted sentiment analysis via span-based extraction and classification. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 537–546, Florence, Italy. Association for Computational Linguistics.

John Lafferty, Andrew Mccallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. pages 282–289.

Long Mai and Bac Le. 2020. Joint sentence and aspect-level sentiment analysis of product comments. *Annals of Operations Research*.

Margaret Mitchell, Jacqui Aguilar, Theresa Wilson, and Benjamin Van Durme. 2013. Open domain targeted sentiment. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1654, Seattle, Washington, USA. Association for Computational Linguistics.

L. Pelayo and S. Dick. 2007. Applying novel resampling strategies to software defect prediction. In *NAFIPS 2007 - 2007 Annual Meeting of the North American Fuzzy Information Processing Society*, pages 69–72.

Chi Sun, Luyao Huang, and Xipeng Qiu. 2019. Utilizing bert for aspect-based sentiment analysis via constructing auxiliary sentence.

Henry Tsai, Jason Riesa, Melvin Johnson, Naveen Arivazhagan, Xin Li, and Amelia Archer. 2019. Small and practical bert models for sequence labeling.

Hu Xu, Bing Liu, Lei Shu, and Philip S. Yu. 2018. Double embeddings and CNN-based sequence labeling for aspect extraction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 592–598, Melbourne, Australia. Association for Computational Linguistics.

Meishan Zhang, Yue Zhang, and Duy-Tin Vo. 2015. Neural networks for open domain targeted sentiment. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 612–621, Lisbon, Portugal. Association for Computational Linguistics.

Lilja Øvrelid, Petter Mæhlum, Jeremy Barnes, and Erik Velldal. 2019. A fine-grained sentiment dataset for norwegian.

Transfer Learning for Targeted Sentiment Analysis on NoReC$_{fine}$

Mathias Krafft
mathiabk@uio.no

Adrian Eriksen
adriane@uio.no

Abstract

In this paper, we will present experiments on a sequence labeling task, namely targeted sentiment analysis with pretrained ELMo and BERT models, as well as different model architectures, on the newly created NoReC$_{fine}$ dataset. We perform a sensitivity analysis on hyperparameters for all models and do performance comparisons to the baseline BiLSTM. While we had high hopes for both BERT and Nordic BERT to perform well, ELMo outperformed them by a lot.

1 Introduction

The task of this paper is to do experiments on a targeted sentiment analysis task using the NoReC$_{fine}$ dataset (Øvrelid et al., 2019). The task of targeted sentiment analysis is to identify the targets of opinions in a sentence and determine their polarity, meaning whether the opinion on the target is positive or negative. As opposed to most sentiment analysis tasks that classifies polarity on entire sentences, fine-grained tasks such as the one presented in this paper tries to identify the entity, or entities, towards which the opinion is directed towards, as well as polarity. The NoReC$_{fine}$ is a dataset for fine grained sentiment analysis in Norwegian. It provides a preliminary benchmark for further experimentation, which we will use as a measure for improving the binary and proportional overlap F1 scores. We will use multilingual and Norwegian langauage models. Specifically, ELMo Representations for Many Languages (Che et al., 2018) and BERT-base-multilingual-cased (Wolf et al., 2019). These models are in turn heavily based on BERT (Devlin et al., 2018) and ELMo (Peters et al., 2018b) The NoReC$_{fine}$ dataset consists of roughly 8000 sentences taken from almost 300 reviews taken from over 10 different thematic categories, such as literature and product reviews. It originally consists of sentences with opinions tagged with many different features, such as polar expression, intensity, polarity, among others. However, only the sentences and their polarity tags from the dataset will be considered in this paper. These tags uses the inside-outside-beginning-scheme (IOB) for each word in the sentence. Words can be tagged as either positive or negative for a total of five tags as shown below.

I	targ-positive	targ-negative
O		
B	targ-positive	targ-negative

Our goal for this paper is to see how different architectures and uses of pretrained models compare to the binary and proportional F1 scores of a baseline model. In order to reach this goal, we will modify the baseline model with different architectures, use pretrained models to extract word representations and do hyper-parameter tuning on the different configurations. We will present a strategy for experimentation and show how changes in the aforementioned categories affect the performance of the models.

2 Related Work

Sentiment analysis is most often viewed as a sentence classification problem (Pontiki et al., 2014), classifying the entire sequences. However, more fine-grained approaches that target entities and the opinions expressed are seen as a sequence tagging task (Yang and Cardie, 2013; Vo and Zhang, 2015). While there are many approaches for model architectures and pipelines for such tasks, one common aspect is that they almost always contain transfer learning aspects (Chen and Qian, 2019), using either pre-trained embeddings or language models to extract word representations. (Peters et al., 2018a; Devlin et al., 2018). Other approaches make use of

the attention mechanism (Bahdanau et al., 2014) to learn representations for sentences that include the target entities for sentiment.

3 Data Exploration

Before starting the experimentation, we wanted to get some insight into the provided data. The data comes in the conll format, where each token in a sentence has an associated tag, separated with a tab. Each sentence in the dataset is separated with a newline. Lets have a quick look at an example at Figure 1.

```
1
2   Jeg O
3   liker   O
4   Aune    B-targ-Positive
5   Sand    I-targ-Positive
6   .   O
7
```

Figure 1

We take a closer look at how the words in the training data is distributed. As shown in Figure 2, almost every sentence has a length shorter than 50. Regarding the vocabulary, there is a significant decrease in the amount of words included after the first 8000. Because we have high computational power in our experiments, we have the luxury of being able to include all sentence lengths and the entire vocabulary. Should you not have the means to train on the entire vocabulary and with all sentences, excluding the last 5-10% is worth considering.

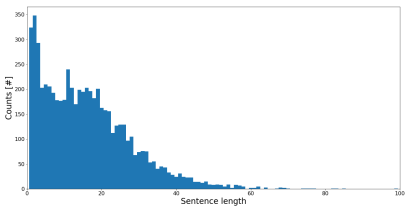


Figure 2

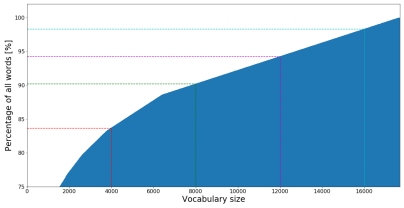


Figure 3

4 Strategy

Our overall goal with this paper was mainly to see if we could utilize transfer learning and use pretrained models to achieve proper performance on the task at hand. In order to do so, we wanted to use pretrained models and embeddings that would give good word representations to classify on, as well as look at how they would compare to the baseline. We tried Pretrained ELMo Representations for Many Languages, Nordic BERT and BERT-base-multilingual-cased. We wanted to create a working model for each of these pretrained models, and continued by improving them with fine-tuning hyperparameters and doing sensitivity analysis. However, it turned out that Nordic BERT had some serious issues that we did not have the time to go into, so we had to let that one be and rather focus on the other models. To subsidize for this, we created a bidirectional model using the GRU-architecture, using pretrained embeddings to have a more similar comparison to the baseline bidirectional LSTM model.

4.1 Baseline Model

The baseline model looks up word embeddings from the pretrained model to use as input to its Long Short-Term Memory, LSTM, block. By default the baseline uses two layers in its LSTM. The final hidden state of the LSTM is fed to a fully connected layer to produce logits for each possible label per token. The label with the highest score is picked when predicting labels for tokens.

4.2 GRU Model

The gated recurrent unit, GRU, model uses the exact same setup as the baseline, but switches out the LSTM block with a gated recurrent unit (Cho et al., 2014). The main difference between LSTM's and GRU's is that LSTM's have three gates (input, output, forget), while GRU's have two (reset and update). Because of this GRU's use less training parameters which leads to shorter training times and less memory usage.

4.3 BERT Model

The BERT model is a very large model with a stack of transformer-encoder blocks. In our sentiment analysis model, we use a BERT model that's been trained for a language modelling task on 102 languages (Wolf et al., 2019), including Norwegian, to extract word representations. These word represen-

tations are sent to a fully connected layer that takes logits for each possible label per token. BERT, however, often splits word into word pieces to not have to deal with unknown words. During training, the label for a word is therefore extended over all the created word pieces.

4.4 ELMo Model

Our sentiment analysis model that uses the pretrained ELMo does much the same as the baseline model. The main difference is that instead of using a vocabulary of pretrained word embeddings, it extracts contextualized word representations from the pretrained ELMo language model. The pretrained ELMo language model that we use has been trained on 20 million Norwegian words. These word representations are then sent to a LSTM and then a fully connected layer (Che et al., 2018).

5 Data Preparation

The provided NoReC$_{fine}$ dataset is already divided into splits for training, development and testing. Making use of frameworks such as torchtext and pytorch, we are able to create dataloaders that provide all the sentences and tags in batches. The dataloaders will then provide the tokenized raw text and the true labels for each token for every sentence in the dataset. The different models we're testing takes different types of inputs that can be gathered from this information. Both the baseline and the GRU implementation use pretrained context-free Gensim embeddings gathered from the NoWaC dataset (Guevara, 2010) with Continuous Bag-of-Words (Fares et al., 2017). The input to the model is made by looking up each individual token in the sentence and using the provided embedding.

As opposed to the context-free embeddings used in the other models, both ELMo and BERT create so-called contextual embeddings. The ELMo embedder we are using takes the raw text in the sentence and creates embeddings for each token, taking into account the surrounding tokens. BERT requires a bit more preprocessing. BERT tokenizes words using word pieces, meaning each original token can be split into multiple parts. BERT also adds '[CLS]' and '[SEP]' tokens to either end of the input sentence. An example input of 'Jeg liker Aune Sand.' is preprocesed with the BERT tokenizer to ['[CLS]', 'Jeg', 'like', '##r', 'Au', '##ne', 'Sand', '.', '[SEP]']. '[PAD]' tokens are added to the end of shorter sentences in a batch as well. Because of these additions, we also need to transform the labels to reflect the changes in the input. We opted to extend the original label of a word to each newly created word piece and label the added special tokens with the 'O' tag in the IOB-scheme. During evaluation and testing, these extended labels are averaged to get the actual predicted label.

6 Experiments

Following our strategy, we split up the experiment section in three parts. The first experiment was to set up the baseline and starting points for all the other models, the second to perform a sensitivity analysis on the hyperparameters and lastly to retrain all the models on the best configurations from the sensitivity analysis.

6.1 Starting point

Following our strategy, our first experiment was to set up the baseline and starting points for all the other models. This was done by training the baseline bidirectional LSTM model with the following configuration: for 20 epochs, using two LSTM layers, hidden dimension = 100, batch size = 64, learning rate = 0.01. We fine tune the 100 dimensional pretrained word embeddings and use a word dropout of 0.01. To get a preliminary sense for how all three (ELMo, BERT, GRU) models behave, we did training using the exact same configuration. The plots in Figure 4, 5 and 6 are evaluated on the development set during training and will give a sense of how the three other models performed with the same configuration.

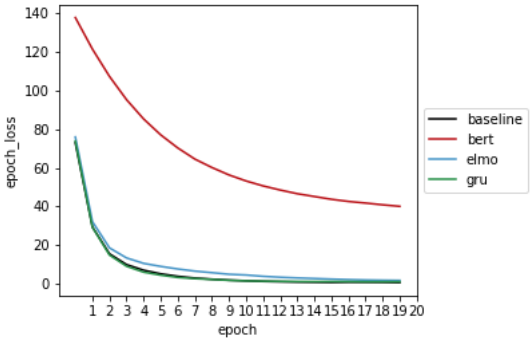


Figure 4

We can see from the onset that using the ELMo pretrained model to extract word representations

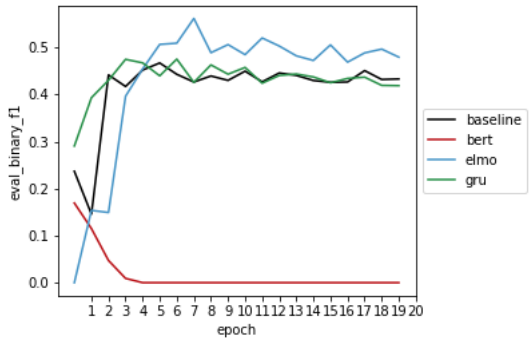


Figure 5

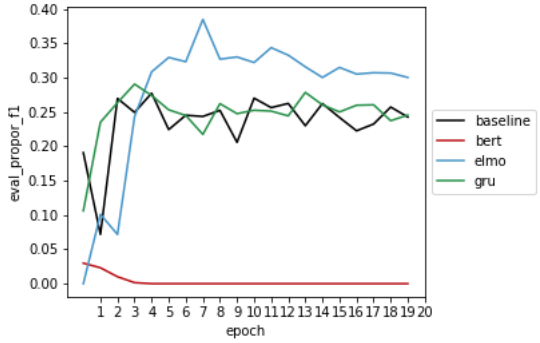


Figure 6

makes the model perform better. Comparing the GRU architecture to the LSTM baseline, it seems to only be some small differences in performance. We also note that the pretrained multilingual BERT model does not learn at all with this configuration, classifying every token with the same 'O' label.

6.2 Sensitivity Analysis

From there, we stared conducting the sensitivity analysis, changing the following parameters for the models where applicable.

We trained on the training dataset and evaluated the models on binary and proportional F1 scores on the development dataset.

As part of our experimentation, we did a sensitivity analysis of BERT and ELMo on the provided dataset. We only include the binary and proportional F1 scores in this paper, as that is what's most relevant to what we want to achieve. We experimented with batch size, learning rate, the size of the hidden dimensions and the number of layers. For the BERT model we are not able to fine tune the layers and hidden dimensions, so we only experimented on ELMo with these.

6.2.1 Batch size

Binary F1

Batch size	8	16	32	64	128
Baseline	0.46	0.45	0.46	**0.47**	0.45
GRU	0.44	0.46	**0.48**	0.47	0.47
ELMo	0.5	0.54	0.51	**0.56**	0.54
BERT	0.003	0.06	0.14	0.16	**0.25**

Proportional F1

Batch size	8	16	32	64	128
Baseline	0.26	**0.28**	0.27	0.27	0.27
GRU	0.26	0.27	**0.30**	0.29	0.28
ELMo	0.311	0.36	0.35	**0.38**	0.35
BERT	0.001	0.02	0.02	**0.03**	0.022

Across the board, it seems that having a big enough batch size increase performance. We theorize that this might be due the sheer amount of outside-'O' tags. The probability of all examples in a batch containing only such tags is bigger when the batch size is small. Backpropagating losses from such a batch early on, might lead the model towards labelling all tokens with this tag. This minimum might be difficult to get out from when encountering other labels. When using bigger batch sizes, beginning-'B' and inside-'I' labels are encountered more evenly throughout the epoch. BERT however, having all the other hyperparameters, never learned anything with the change of batch size. As we'll see later, it's because our BERT model is really sensitive to learning rate changes.

6.2.2 Learning rate

Binary F1

Learning Rate	0.1	0.01	0.001	0.0001
Baseline	0.31	0.46	**0.48**	0.31
GRU	0.28	**0.47**	0.46	0.35
ELMo	0.0	0.52	**0.54**	0.37
BERT	0.43	**0.5**	0.35	0.05

Proportional F1

Learning Rate	0.1	0.01	0.001	0.0001
Baseline	0.17	0.28	**0.29**	0.19
GRU	0.13	0.27	**0.29**	0.23
ELMo	0.0	0.33	**0.36**	0.19
BERT	0.22	**0.28**	0.19	0.03

The pretrained transfer learning models had the biggest sensitivity to change in learning rate. As we can see from the tables, these models sometimes didn't learn anything at all because of

the change in learning rate. For big learning rates, this might be because of the direction the model take in the early stages can be wrong and it's hard to recover from big steps in the wrong direction. On the other hand, too small of a learning rate will make the model not train at all. Interestingly, when the learning rate is tuned just right, these pretrained models perform better than the other recurrent model. The recurrent models however, have more even and better average performance across the board.

6.2.3 Number of recurrent layers

Binary F1

Layers	2	4	8	16
Baseline	**0.46**	0.44	0.0	0.0
GRU	**0.47**	0.44	0.0	0.0
ELMo	0.52	**0.55**	0.0	0.0

Proportional F1

Layers	2	4	8	16
Baseline	**0.28**	0.26	0.0	0.0
GRU	**0.27**	0.26	0.0	0.0
ELMo	0.344	**0.346**	0.0	0.0

Since the pretrained BERT model came with a classifier head attached, we did not feed the word representations into a recurrent layer before the final fully connected layer, but let the model perform predictions directly from this classifier head. Because of this, number of layers in the recurrent layer is not applicable for the BERT model. For the other models, however, it's interesting to see that small number of layers performed better. For all the models, increasing the number of layers above 4, made them all only predict outside-'O' tokens, and thus receive no scores on either category.

6.2.4 Hidden dimensions

Binary F1

hidden_dim	32	64	128	256
Baseline	0.44	0.44	**0.48**	0.47
GRU	0.47	0.48	**0.49**	0.46
ELMo	0.48	0.49	0.52	**0.54**

Proportional F1

hidden_dim	32	64	128	256
Baseline	0.28	0.26	**0.28**	0.27
GRU	0.28	**0.29**	0.28	0.28
ELMo	0.31	0.32	0.34	**0.36**

The trend in all applicable models for this category is that they perform better on more hidden dimensions. For the transfer learning model with ELMo, this meant that it got the best performance on the largest value, but for the GRU and LSTM models, the performance seem to steady itself after enough hidden dimensions is used.

6.2.5 Combined Overview

In addition to the results from the sensitivity analysis, we want to present plots of the metrics from all runs made, to be able to compare how the different model architectures behave with different hyperparameters. We have created plots to show all experiment runs for the sensitivity analysis in one graph, where the black line represents baseline, red represents BERT, green represents GRU and blue represents ELMo. This is to be able to compare how the different colorcoded architectures perform across multiple types of runs at a glance. We see that ELMo performs better overall, while BERT is really sensitive to parameter changes, GRU and LSTM has even and comparable results.

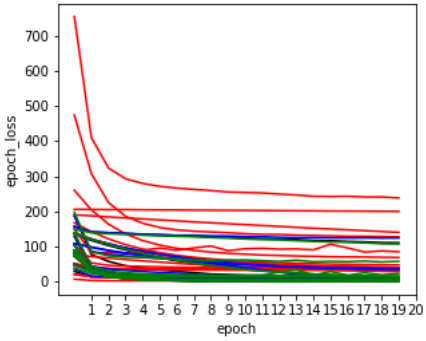


Figure 7

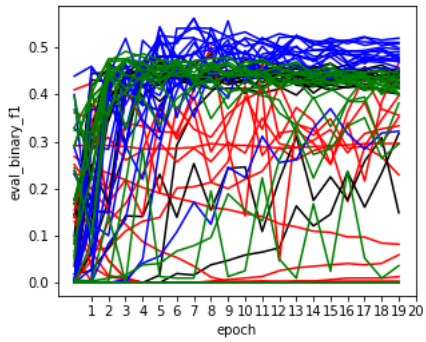


Figure 8

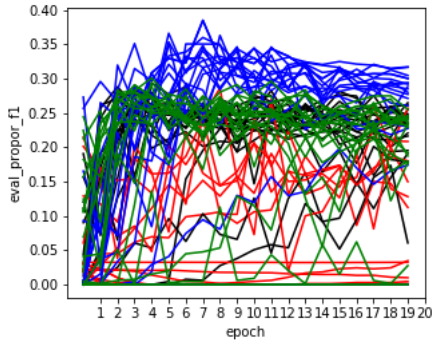


Figure 9

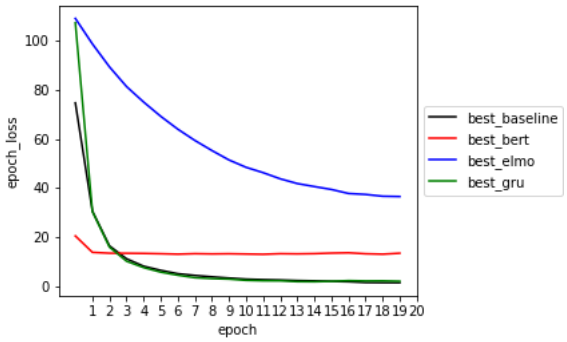


Figure 10

6.3 Best configurations

Finally, retraining all models using both the training and development data, on the best respective configurations from the sensitivity analysis, let us determine how well the models generalizes and performs on unseen data. We'll provide the results of this experiment under the Results section. All metrics under that section will be from analysis done on the held out test dataset.

7 Results

While not completely satisfied with the performance of the different models, we've conducted structured experiments and achieved results that slightly outperform the baseline. We can see that ELMo is clearly superior to the other models in both our experimentation and in the final results. However, we do suspect that our BERT model is under performing and that with some more work it might yield better results.

Binary F1	
Baseline	0.455
GRU	0.482
BERT	0.467
ELMo	**0.542**

Proportional F1	
Baseline	0.264
GRU	0.266
BERT	0.256
ELMo	**0.352**

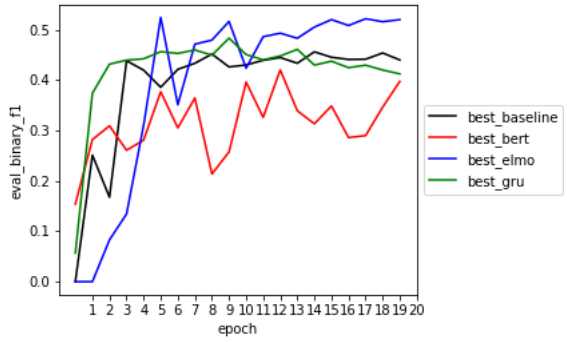


Figure 11

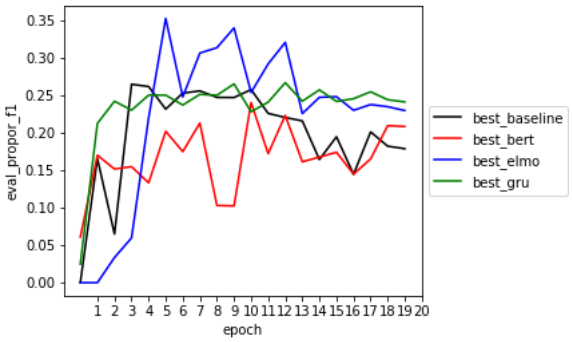


Figure 12

8 Future Work

To further improve on our results, we would first of all take a closer look at how the models trained to see if we could get them to perform better. Especially on the BERT, which under-performed throughout our experimentation. We spent some time trying to get our BERT-model to perform better, but had to leave it due to the deadline. Secondly, we believe that better results can be achieved by weighting the labels such that it would gain incentive to not choose the 'OUT'-label, which is over

represented. Using a classification matrix to see how the models are labelling the words would be useful for this. We also noticed similarities between sentiment analysis on this data and Named Entity Recognition (NER). In future work, we would consider using transfer learning between this task and NER tasks. Finally, we spent some time trying to get a Norwegian BERT-model to train properly, but we had to disregard this completely due to what we believe is a faulty model. For future work it would be interesting to see how a Norwegian model would perform on this dataset.

9 Conclusion

For the task of identifying and labeling the target entities of opinions using the NoReC$_{fine}$ dataset, training the transfer learning ELMo language model to extract contextualized word representations performed best. While the results are far from good enough yet to be used in any production ready capacity, the results show that using such word representations were superior to regular context-free pretrained embeddings on this task, using any of the model architectures experimented on. Difficulties producing proper results using the multilingual pretrained BERT model, could either be to pure instability in the model itself, or that the multilingual nature of the model made it hard to learn on a small, fine-grained Norwegian dataset.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. 2018. Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium. Association for Computational Linguistics.

Zhuang Chen and Tieyun Qian. 2019. Transfer capsule network for aspect level sentiment classification. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 547–556, Florence, Italy. Association for Computational Linguistics.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding.

Murhaf Fares, Andrey Kutuzov, Stephan Oepen, and Erik Velldal. 2017. Word vectors, reuse, and replicability: Towards a community repository of large-text resources. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 271–276, Gothenburg, Sweden. Association for Computational Linguistics.

Emiliano Raul Guevara. 2010. NoWaC: a large web-based corpus for Norwegian. In *Proceedings of the NAACL HLT 2010 Sixth Web as Corpus Workshop*, pages 1–7, NAACL-HLT, Los Angeles. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018a. Deep contextualized word representations.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018b. Deep contextualized word representations. In *Proc. of NAACL*.

Maria Pontiki, Dimitris Galanis, John Pavlopoulos, Harris Papageorgiou, Ion Androutsopoulos, and Suresh Manandhar. 2014. SemEval-2014 task 4: Aspect based sentiment analysis. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 27–35, Dublin, Ireland. Association for Computational Linguistics.

Duy-Tin Vo and Yue Zhang. 2015. Target-dependent twitter sentiment classification with rich automatic features. In *IJCAI*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Bishan Yang and Claire Cardie. 2013. Joint inference for fine-grained opinion extraction. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1640–1649, Sofia, Bulgaria. Association for Computational Linguistics.

Lilja Øvrelid, Petter Mæhlum, Jeremy Barnes, and Erik Velldal. 2019. A fine-grained sentiment dataset for norwegian.

Targeted Sentiment Analysis Using a Fine-Grained Dataset: NoReCfine

Arthur Dujardin
Department of Informatics,
University of Oslo

Lotte Boerboom
Department of Informatics,
University of Oslo

Silvia N.W. Hertzberg
Department of Informatics,
University of Oslo

Abstract

Targeted sentiment analysis (TSA) is a variant of normal sentiment analysis, with which you can analyse the sentiment towards a specific target within a text. The goal of this project was to train different neural systems to perform TSA for Norwegian text. Different design methods were examined to find out what their effects were on the recently released data set $NoReC_{fine}$. The performance of bidirectional LSTMs, GRUs and a BERT model were compared to each other and a baseline. In addition a small error analysis was performed. The results show that BERT does not outperform GRUs or LSTMs. The code has been implemented in *PyTorch* and attention was paid to the creation of online documentation on *readthedocs* [1].

1 Introduction

Sentiment analysis (SA) is a well-researched topic in Natural Language Processing (NLP). Conventionally the analysis obtains and classifies emotions or sentiments in sentences or documents as positive or negative. In this regard, there has been a parallel growth of SA identified with the growth of social media. This influx of information and with the use of a rich computational study as SA provides valuable information to marketers, managers or even manufactures on what consumers or employees perceive or demand (Xu et al., 2019). Thus, opinions derived from the study can change the policy of an organization as well as strategies to meet consumer demands. Nevertheless, SA is limited to deriving the overall polarity other than the reference aspect resulting in the development of SA branches such as targeted sentiment analysis (TSA) (Barnes and Klinger, 2019). Errors that may result from SA are harboured using TSA as the sentiment polarity identification (e.g. negative, neutral, or positive) is attached to a target in their context sentence (Pang and Lee, 2008; Jiang et al., 2011; Saeidi et al., 2016). For example, *"I hate his late coming behaviour, but he is an exceptional performer"* here the reviewer has a negative sentiment toward an employee's attendance behaviour but a positive sentiment on the performance. An aggregated sentiment drawn from the targets can allow the users to understand employee capability. This is achieved by the use of a neural network to automatically learn dimensional representation for targets in the sentence context. In this paper, we examine the effect of various design methods on $NoReC_{fine}$ dataset (Øvrelid et al., 2019) by identifying the target opinion and their associated polarity. With Bidirectional Long Short Term Memory (BiLSTM) as base model, we conduct experiments with other gated mechanisms like Gated Recurrent Unit (GRU) accompanied with deep neural layers.

2 Related Work

Several machine learning methods have been used in TSA and despite its initial design with support vector machine (SVM), there has been an achieved progress by utilizing neural networks to encode base features as continuous and low dimensional vectors. Some studies employed Recursive Neural Network in conducting semantic composition for prediction representation while LSTM implemented in modelling the left and right context of a target and concatenate them for prediction results which were essentially not target specific (Jiang et al., 2011; Socher et al., 2011; Dong et al., 2014; Ma et al., 2018; Li and Lu, 2017; Tang et al., 2016).

The target influence on prediction representations were achieved by an attention mechanism approach leading to bidirectional models that interactively learn attention weights on contexts and targets to separately generate target and context (Wang et al., 2016; Chen et al., 2017). In addition, more refined models are being studied to reduce

[1] https://pages.github.uio.no/arthurd/in5550-exam/ (20/05/2020)

information loss in case of multiple words. For instance, to capture word-level interaction within a target and context, a mix of fine and course-grained attention mechanism was employed which could also depict the same context target interaction (Fan et al., 2018).

Transformer architecture modification introduces a lightweight model in TSA, which adopts attention-based encoders in modelling target words and context differently (Song et al., 2019). For sentences with more than one target word, Zhou et al. (2019), utilizes Convolutional Neural Network (CNN) to model explicit dependency between the opinion words; the architecture obtains significant results on multiple target words in a sentence.

Regardless of the advantages of various designs, some studies show high state of the art results on certain dataset than others with regards to its composition (Chen et al., 2017). In addition, most of these architectures have been done on English datasets except a few with an example of a recent study that used fine-grained sentiment analysis on novel Norwegian dataset *NoReC*$_{fine}$ (Øvrelid et al., 2019). Our study will employ the same dataset to estimate the effect of computational TSA design as used on English dataset on the polarity and target of the data.

3 Data

3.1 NoReC dataset

The *NoReC*$_{fine}$ dataset is composed of a training, validation and testing sub-datasets. The data contains labeled sentiment analysis of professionally authored reviews across a wide variety of domains. The labels follow the Beginning - Inside - Outside (BIO) format, which are used to mark the start and end of a target within a sentence - positive or negative (Øvrelid et al., 2019).

Words	Labels
Men	O
den	B-targ-Negative
er	O
svakere	O
enn	O
”	B-targ-Positive
Sammen	I-targ-Positive
for	I-targ-Positive
livet	I-targ-Positive
”	I-targ-Positive
.	O

Table 1: Example of a sentence in *NoReC*$_{fine}$ dataset.

(a) Words inside negative targets.

(b) Words inside positive targets.

Figure 1: Word cloud from *NoReC*$_{fine}$ dataset, without stop-words. Positive and negative targets share similar words like *filmen* or *albumet*, however some patterns seems to differentiate these sentiments, like *Love* and *Fire* keywords. Note that English words are part of the dataset, like the sentence 351: “På ”California” (There Is No End To Love)” parafraseres finske The Rasmus’ hit ”In The Shadows”, av alle ting, og det lånes generelt raust fra nær pop historie.”

In addition, pre-trained word embeddings on a Norwegian-Bokmaal CoNLL17 corpus were used, obtained using a Word2Vec Continuous Skipgram model [2].

3.2 Distribution

The dataset is composed of a total of 8.259 examples, split into training, validation and testing subdatasets.

	Train	Valid	Test	Total	Avg. Length
Sentences	6145	1184	930	8259	16.8
Targets	4458	832	709	5999	2

Table 2: Number of sentences and targets per datasets.

The labels are not equally distributed over the sentences. In fact, the dataset is imbalanced and 90% of the labels are the label *Outside* (O) of a target.

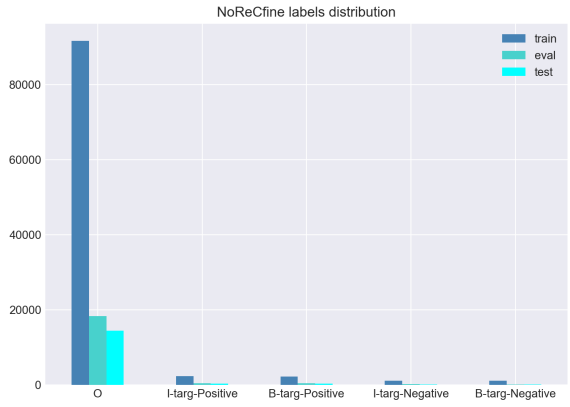


Figure 2: Labels distribution over the train, validation and test *NoReCfine* datasets.

	Train	Valid	Test	Total
O	91691	18336	14425	124452
I-targ-Pos	2339	436	348	3123
B-targ-Pos	2245	433	366	3044
I-targ-Neg	1114	207	117	1438
B-targ-Neg	1094	196	145	1435

Table 3: Imbalanced labels distribution.

Weights can be added to balance the data distribution, and this part will be discussed later.

[2] http://vectors.nlpl.eu/repository/20/58.zip (20/05/2020)

In addition, more than one sentiment can be retrieved from a sentence.

	Train	Valid	Test	Total
O	15.5	15.9	16.1	15.6
I-targ-Pos	3.3	3.0	3.0	3.2
B-targ-Pos	1.4	1.3	1.4	1.4
I-targ-Neg	3.1	3.0	2.3	3.0
B-targ-Neg	1.2	1.1	1.2	1.2

Table 4: Number of labels per sentence (mean).

4 Models

4.1 Approach

Our models are based on a provided baseline model (BiLSTM) for the TSA to analyse the polarity of the target words. As the script given did not conform precisely with (Paszke et al., 2019) standards, and the model was struggling to classify the task words, we further developed a compatible *PyTorch* version that had better results. The BiLSTM model was trained on Norwegian Bokmål word embeddings by utilizing random initialization with a one step forward pass and used cross-entropy loss function on an un-padded index to calculate the loss. In general LSTM models are capable of flexibly capturing the semantic relationship that exists between a target and the context words and avoids gradient vanishing (Tang et al., 2016).

To compare the results, a variant of an LSTM; Gated Recurrent Unit (GRU) model was developed. GRU introduces the combination of the input and forget gates resulting into a simplified gate. While among other changes the model also combines the hidden state to the cell state. As there exist mixed findings between LSTMs and GRU performance, the application of GRU on the $NoReC_{fine}$ data could establish an explainable result. The parameters remained virtually the same in both models.

Then, we wanted to explore the possibilities and results with a BERT model template. We preprocessed the $NoReC_{fine}$ datasets and generated masked arrays, but we faced the same problem of over-fitting.

4.2 BiLSTM

Our baseline model is a LSTM with a word embedding layer. We used the *58.zip* pre-trained word embeddings on Norwegian Bokmål, downloaded from NLPL vectors. We created a skeleton (table 5) that we tuned during the grid-search optimization.

Modules	Dimensions	Parameter
Embedding	(23,574, 100)	-
LSTM	(100, H)	-
Dropout	-	p
Linear	($2*H$, C)	-

Table 5: BiLSTM skeleton used. Note that the bidirectional and recurrent layers parameters were also tested during the grid-search.

4.3 BiGRU

After implementing the BiLSTM model, we quickly implemented the BiGRU base model, as only the core recurrent cell changed. Its skeleton can be described similarly in table 6.

Modules	Dimensions	Parameter
Embedding	(23,574, 100)	-
GRU	(100, H)	-
Dropout	-	p
Linear	($2*H$, C)	-

Table 6: BiGRU skeleton used. Note that the bidirectional and recurrent layers parameters were also tested during the grid-search.

4.4 BERT

Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018) models are widely used in NLP, from targeted sentiments to question answering. It can work by paired-sentences: from one sentence, it learns to predict the second. However, we did not used this functionality as sentences are independent.

BERT models use a Masked Language Model (MLM) with attention masked arrays on top of the labels (in our case, BIO labels). In addition, special tokens *[CLS]* and *[SEP]* are used to delimit a sentence, a feature mainly used with paired-sentences training.

Modules	Dimensions	Parameter
Word Embeddings	(119,547, 768)	-
Position Embeddings	(512, 768))	-
Token Embeddings	(2, 768)	-
Dropout	-	p
Encoders	[12, 24]	-
Linear	(768, 768)	-
Dropout	-	p
Linear	(768, C)	-

Table 7: BERT skeleton used.

For this short project, we used the $BERT_{BASE}$ (made of 12 encoders) multi-lingual pre-trained model on 104 language (latest model). We also used $BERT_{LARGE}$ (made of 24 encoders) for comparison, but as it is time-consuming to train, we did not experiment with a lot of hyper-parameters tuning.

A special data processing was implemented in *PyTorch*, so we could use pre-trained transformers from *HuggingFace* repository (Wolf et al., 2019).

4.5 Loss Function

We first used a Cross Entropy Loss function, but a majority of models were over-fitting due to the imbalanced data. Thus, we tried a weighted loss which improved the results but did not solve over-fitting issues on baseline models.

The weights were determined from the distribution (table 3).

Target	Weight
O	0.0677
I-targ-Positive	0.9766
B-targ-Positive	0.9772
I-targ-Negative	0.9892
B-targ-Negative	0.9893

Table 8: Weights used for the Cross Entropy Loss function.

5 Results

5.1 Scores

We used normalized confusion matrices (eq. 1) and macro F_1 scores to measure models' errors.

$$\mathcal{N}(M) = (m'_{i,j})_{(i,j)\in[\![0,n-1]\!]^2} \tag{1}$$

$$\textit{where},$$

$$\forall (i,j) \in [\![0,n-1]\!]^2, \quad m'_{i,j} = \frac{m_{i,j}}{\sum_{k=1}^{n-1} m_{k,j}}$$

Normalization makes classes proportional, so that each row of the matrix is summed to 1. When working with imbalanced datasets, normalization enables a better comprehension of the predictions, in terms of percentage. This highlights the mismatched classes, even if they are under-estimated in the data.

5.2 Grid Search

To visualize the impact of a hyper-parameter variation, we fixed all other parameters. By default, we used a learning rate $l_r = 0.1$, a batch size $B_{size} = 64$, a hidden dimension $H = 300$, a bidirectional state $B_{direction} = True$ and $n_{layers} = 2$ layers for the recurrent cell and finally the weights w_L defined in (table 8) for the loss function, trained on 100 epochs. For this section, we implemented an independent package to tune any *PyTorch* model. We only ran the grid-search on four model types, but this package can be used for further projects. The grid-search results are resumed in (Appendix, table 10).

For BiLSTMs and BiGRUs, results are not really satisfying as loss diverges at epoch 20 maximum (Appendix, Figure 4). Even with weights, the model struggles to correctly classify underrepresented targets (*B-targ-Negative* for example). However, as the class *O* is over-represented, global accuracy converges, no matter the parameters combination that was used. The confusion matrices (Appendix, figure 5) highlight these issues - baseline models over-fit the data even with a weighted loss function.

The grid search resulted in a variety of models, and we kept the best ones for each structure (table 9). The BiGRU is our top F_1 score model with the lowest loss.

5.3 Over-fitting

The major issue we faced was over-fitting and over-prediction of *O* labels as highlighted by the confusion matrix in figure 3. Baseline models are not efficient when it comes to classifying imbalanced data, and even though global accuracy converges, the loss (and F_1 score) diverges.

5.4 Error analysis

Figure 5 (Appendix) shows that the most common errors for our different labels are the over-prediction of *Outside (O)* labels. A significant proportion of meaningful labels (*I-targ, B-targ*) are predicted as *O* label.

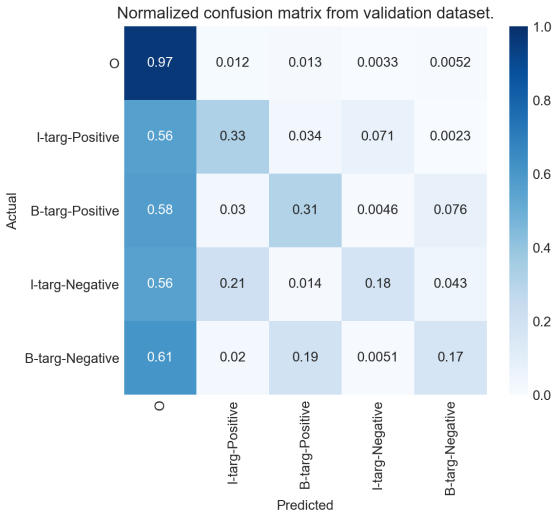


Figure 3: Normalized confusion matrix of the best BiGRU structure, trained over 100 epochs. The label *O* is over-predicted, even with a weighted loss. Meaningful labels like *B-targ, I-targ* are classified in majority as *O*.

On the one hand, our models classify correctly *O* label in more than 96% cases. On the other hand, other classes are classified as *O*.

With non-weighted loss, the over-prediction of *O* drastically increase, as they compose the majority of labels. Attributing weights reduces this phenomenon, but does not solve it fully.

The BiGRU seems to predict the label *O* for the majority of the time. The model seemed to struggle more with correctly predicting negative labels (*I-targ-negative* and *B-targ-negative*) compared to the prediction of positive labels (*I-targ-positive and B-targ-positive*). In addition, our BERT template did not achieve more than a 0.12 F_1 score with a respective 0.97 accuracy.

We did not have enough time to change or try different BERT structures, like RoBERTa with different folds that seems to work well for sentiment extraction - our main issue.

Last but not least, the dataset is more complex to handle as more than one sentiment can be extracted from a sentence. As there is only one *B-targ-Sentiment* label for a sentiment, if the begin-

	Loss	Accuracy	F_1 score	Recall
BiLSTM	0.4582	0.8889	0.2884	0.3527
BiGRU	**0.2415**	**0.9371**	**0.3501**	**0.4351**
BERT-base-multilingual-cased	0.03106	0.9717	0.1157	0.1657
BERT-large-multilingual-cased	0.03320	0.9602	0.0354	0.0152

Table 9: Results of best models from different structures, evaluated on the validation dataset.

ning is mismatched the F_1 score will drop significantly. We tried to add different weights to tackle this issue, with a significant improve (+0.1 F_1 score approximately), but remained under the 0.5 score threshold.

5.5 Conclusion

Targeted Sentiment Analysis is a variant of Sentiment Analysis. In TSA the sentiment towards a specific target in a text is analysed. Starting with a simple baseline model, a grid-search was performed with a BiLSTM and BiGRU to find out optimal parameters. The results showed that the models are suffering with overfitting. The global accuracy remained high (around 0.9), the loss diverged. A possible explanation for this might be that the model predicts the majority class ('label *O*') and therefore correctly classifies a majority, but also miss-classifies a lot of data therefore increasing the loss.

Targeted Sentiment Analysis is a similar topic to Aspect Based Sentiment Analysis, but with a more complex sentiment extraction. Even if opinion mining is a solved research subject, the uncertainty of a starting and ending target makes TSA predictions much more difficult to solve. We figured that simple models (RNN-like) perform better than more complex ones (BERT).

BERT models are well used in Named Entity Recognition areas, and we were convinced that it could outperforms our baseline model. However, because starting tokens depend mainly on the context (and not necessarily on the words) BERT models predict more *O* than reality, leading to a decrease in F_1 score. As also found by Xu et al. (2019), BERT weights may work well with some NLP tasks but fail with review-based tasks.

We did not have enough time to try different tuning of BERT, like a RoBERTa or DistilBERT, which are recently used and may perform better with $NoReC_{fine}$ dataset. For the future we still believe a BERT model might elicit a good performance. Implementing a BERT model is something that could be explored in further research.

Acknowledgements

The models were trained on resources provided by UNINETT Sigma2 - the National Infrastructure for High Performance Computing and Data Storage in Norway. In addition, we used GPU's offered by Google Colab to train and explore other model's architecture.

We would also like to thank the teachers from IFI department at the University of Oslo, who were proactive and supervised both teaching and examination despite the COVID-19 outbreak.

References

Jeremy Barnes and Roman Klinger. 2019. Embedding projection for targeted cross-lingual sentiment: Model comparisons and a real-world study. *CoRR*, abs/1906.10519.

Peng Chen, Zhongqian Sun, Lidong Bing, and Wei Yang. 2017. Recurrent attention network on memory for aspect sentiment analysis. pages 452–461.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding.

Li Dong, Furu Wei, Chuanqi Tan, Duyu Tang, Ming Zhou, and Ke Xu. 2014. Adaptive recursive neural network for target-dependent twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 49–54, Baltimore, Maryland. Association for Computational Linguistics.

Feifan Fan, Yansong Feng, and Dongyan Zhao. 2018. Multi-grained attention network for aspect-level sentiment classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3433–3442, Brussels, Belgium. Association for Computational Linguistics.

Long Jiang, Mo Yu, Ming Zhou, Xiaohua Liu, and Tiejun Zhao. 2011. Target-dependent twitter sentiment classification. pages 151–160.

Hao Li and Wei Lu. 2017. Learning latent sentiment scopes for entity-level sentiment analysis. In *AAAI*.

Dehong Ma, Sujian Li, and Houfeng Wang. 2018. Joint learning for targeted sentiment analysis. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Bo Pang and Lillian Lee. 2008. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2:1–135.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle,

A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Marzieh Saeidi, Guillaume Bouchard, Maria Liakata, and Sebastian Riedel. 2016. Sentihood: Targeted aspect based sentiment analysis dataset for urban neighbourhoods.

Richard Socher, Cliff Lin, Andrew Ng, and Christoper Manning. 2011. Parsing natural scenes and natural language with recursive neural networks. pages 129–136.

Youwei Song, Jiahai Wang, Tao Jiang, Zhiyue Liu, and Yanghui Rao. 2019. Targeted sentiment classification with attentional encoder network. *Lecture Notes in Computer Science*, page 93–103.

Duyu Tang, Bing Qin, Xiaocheng Feng, and Ting Liu. 2016. Effective LSTMs for target-dependent sentiment classification. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3298–3307, Osaka, Japan. The COLING 2016 Organizing Committee.

Junhui Wang, Xiaotong Shen, Yiwen Sun, and Annie Qu. 2016. Classification with unstructured predictors and an application to sentiment analysis. *Journal of the American Statistical Association*, 111(515):1242–1253.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Hu Xu, Bing Liu, Lei Shu, and Philip Yu. 2019. *BERT Post-Training for Review Reading Comprehension and Aspect-based Sentiment Analysis*.

Xuhui Zhou, Yue Zhang, Leyang Cui, and Dandan Huang. 2019. Evaluating commonsense in pre-trained language models.

Lilja Øvrelid, Petter Mæhlum, Jeremy Barnes, and Erik Velldal. 2019. A fine-grained sentiment dataset for norwegian.

Appendix

	Batch sizes			
	32	*64*	*128*	*256*
BiLSTM	**0.3252**	0.3068	0.2988	0.2813
BiGRU	**0.3260**	0.3203	0.3116	0.2963
BERT-base-multilingual-cased	0.1012	**0.1035**	0.0981	0.0516
BERT-large-multilingual-cased	**0.0230**	0.0094	0.0008	0.0000

	Learning rates			
	0.01	*0.05*	*0.1*	*0.2*
BiLSTM	0.3158	**0.3173**	0.2992	0.2991
BiGRU	0.2988	0.3098	**0.3255**	0.3130
BERT-base-multilingual-cased	0.0878	0.0965	**0.1129**	0.1013
BERT-large-multilingual-cased	0.0075	**0.0103**	0.0056	0.0012

	Dropouts			
	0.1	*0.2*	*0.3*	*0.4*
BiLSTM	0.3059	0.2964	0.3101	**0.3264**
BiGRU	0.3069	0.3170	0.3255	**0.3422**
BERT-base-multilingual-cased	0.1095	0.0899	0.1053	**0.1136**
BERT-large-multilingual-cased	0.0102	**0.0124**	0.1053	0.0114

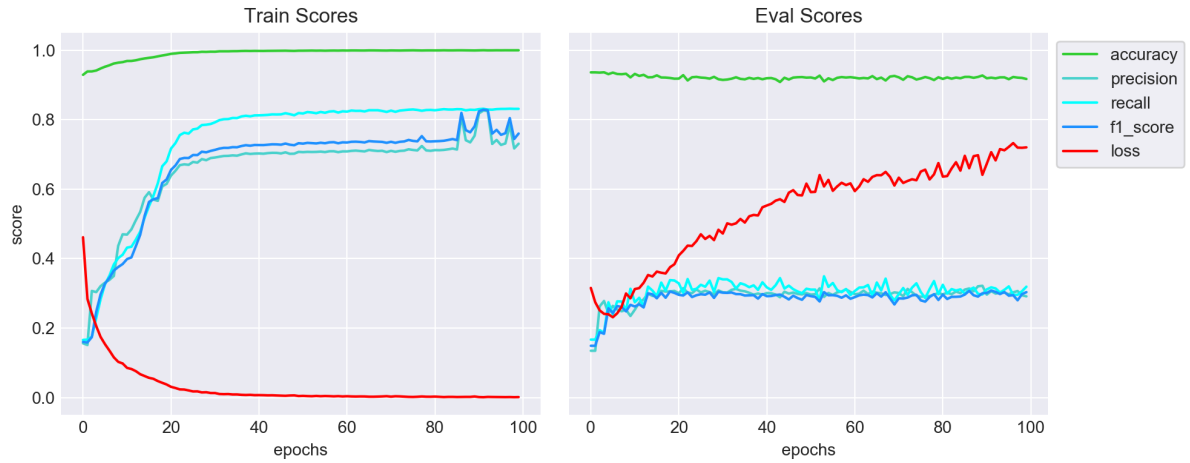
	Weighted Loss Function	
	No weights	*Weights*
BiLSTM	0.3175	**0.3183**
BiGRU	0.3307	**0.3510**
BERT-base-multilingual-cased	**0.1103**	0.0156
BERT-large-multilingual-cased	0.0026	**0.0051**

	Hidden dimensions			
	100	*200*	*300*	*400*
BiLSTM	0.2962	**0.3210**	0.3074	0.3191
BiGRU	0.2775	**0.3165**	0.3071	0.3059

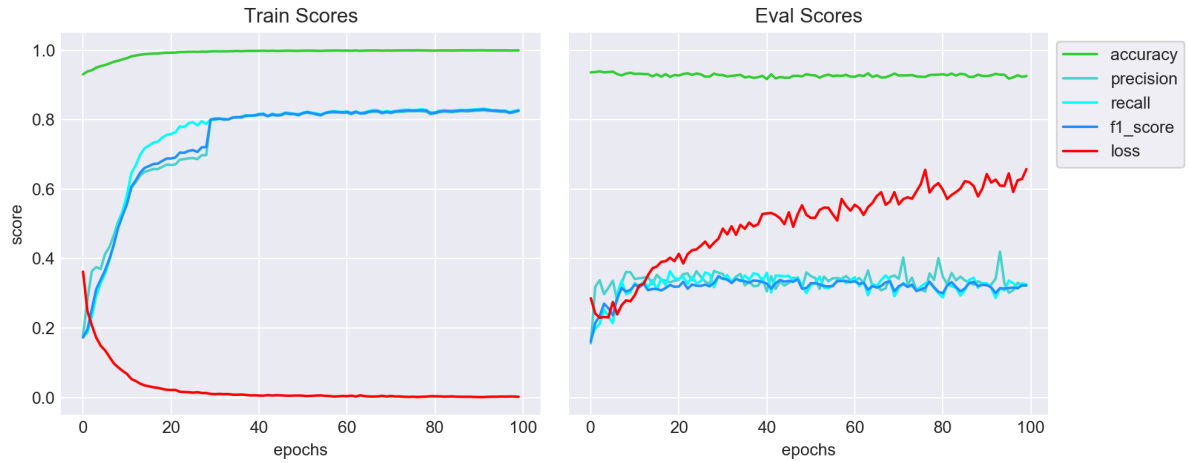
	Bidirectional	
	True	*False*
BiLSTM	**0.3183**	0.2698
BiGRU	**0.3173**	0.2577

	Number of layers			
	1	*2*	*3*	*4*
BiLSTM	0.3002	0.3031	**0.3066**	0.2986
BiGRU	**0.3206**	0.3162	0.3161	0.3109

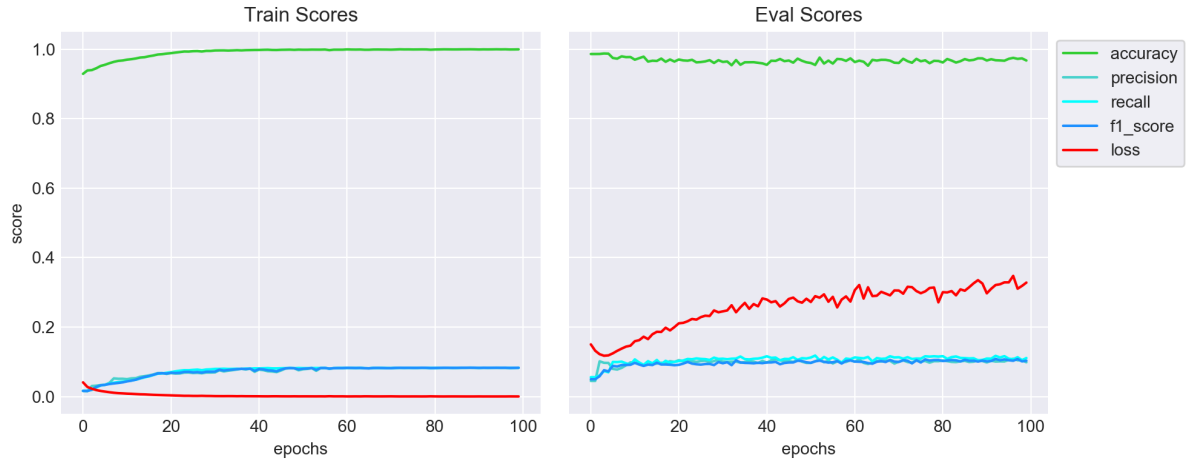
Table 10: Results of the grid-search. The table shows F_1 score regarding a variation of one hyper-parameter. All scores were obtained on the validation dataset.



(a) BiLSTM results.

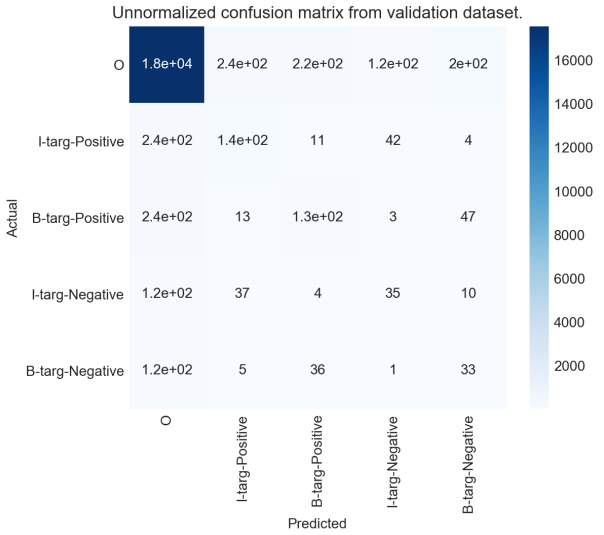


(b) BiGRU results.

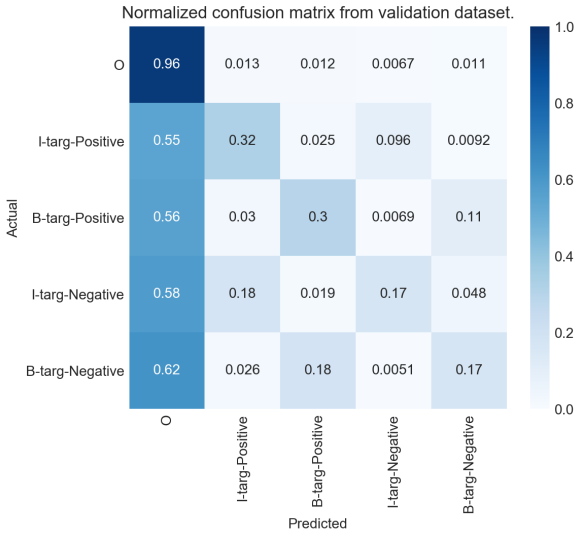


(c) BERT-base-multilingual results.

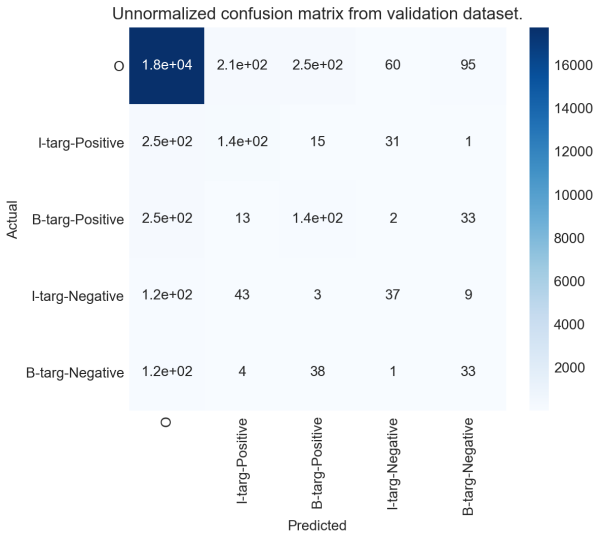
Figure 4: Results of best models from different structures, evaluated on the validation dataset. Even though the accuracy converge in all cases, loss increases when models are training for too long. The shift occurs relatively soon (5-20 epochs), and we experienced that it happens regardless of our hyper-parameters tuning.



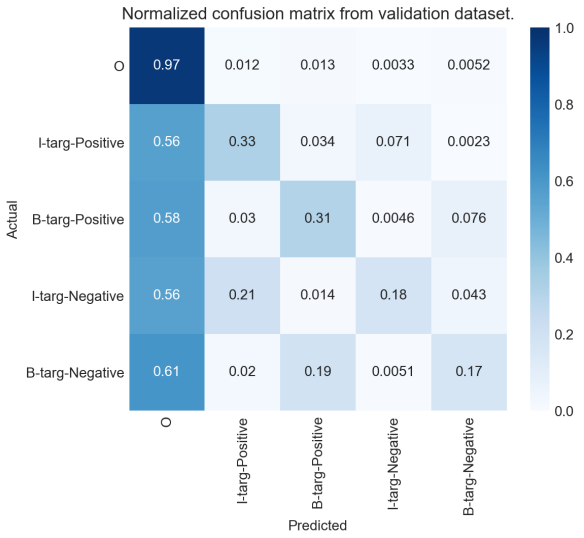
(a) Unnormalized BiLSTM confusion matrix.



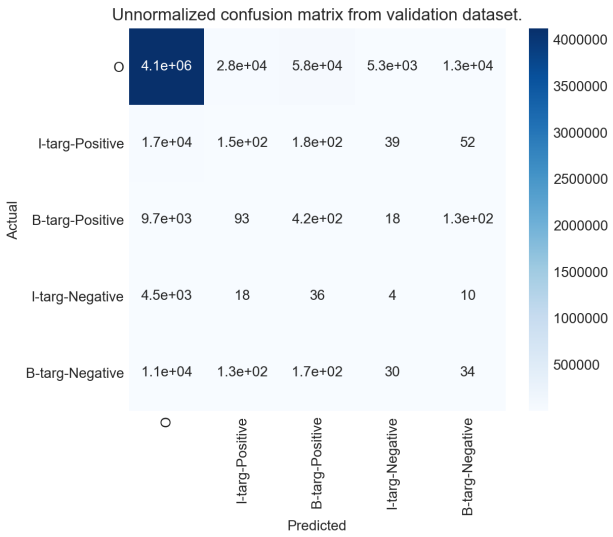
(b) Normalized BiLSTM confusion matrix.



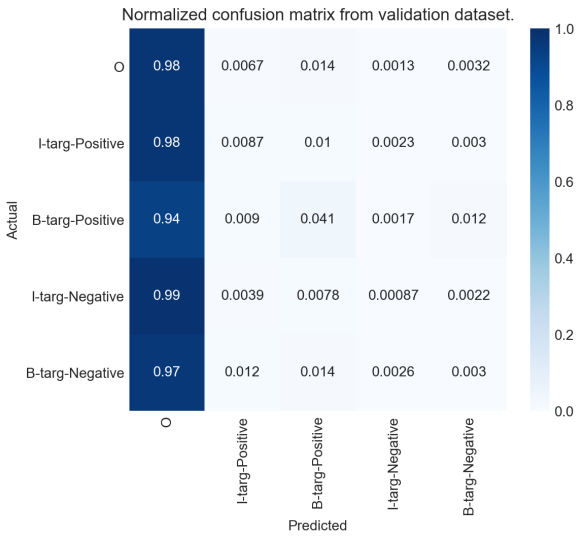
(c) Unnormalized BiGRU confusion matrix.



(d) Normalized BiGRU confusion matrix.



(e) Unnormalized BERT confusion matrix.



(f) Normalized BERT confusion matrix.

Figure 5: Unnormalized (left) and normalized (right) confusion matrices of the best model taken from each structures, evaluated on the validation dataset.

Not Quite on Target: BIO Tag Variation and Lexicon Inclusion for Norwegian Targeted Sentiment Analysis

Petter Mæhlum
University of Oslo
Department of Informatics
pettemae@ifi.uio.no

Muhammad Asad Ali
University of Oslo
Department of Informatics
muhaaa@student.matnat.uio.no

Abstract

This paper explores the effect of variation in BIO tags and the use of sentiment lexicon information on multi-genre targeted sentiment analysis. The paper hopes to shed some new light on a transformed version of a newly released dataset for Norwegian sentiment analysis. Changes in BIO schemes allows us to explore the simplifications that have been done. Incorporating a sentiment lexicon, also recently published for Norwegian, allows us to explore if this might have an effect on targeted SA in this case. Finally, an error analysis explores what types of errors are done.

1 Introduction

The following paper explores the effect of various BIO-tag schemes and sentiment lexicon on the task of targeted sentiment analysis. Sentiment analysis looks at how a person's opinions, or rather evaluations, which can be seen as a representations of a 'private state' Quirk et al. (1985) (see also (Wiebe et al., 2005)), are reflected in writing. This paper tries to employ a newly published dataset by Øvrelid et al. (2019), which contains about 8000 annotated sentences for Norwegian, a sub-corpus of the multi-domain Norwegian Review Corpus (NoReC) (Velldal et al., 2018). As defined in the original corpus, an evaluation contains a polar expression, a target, and a holder. The polar expression is the expression that represents the actual evaluation, either positive or negative, that the holder has towards a target. The target is the entity towards which the polar expression is directed. The holder (also known as source) is the owner of the expression, the one whose inner state is represented.

(1) *Jeg* *elsker* *denne* *filmen.*
I love this the.movie
'I love this movie'

An example can be seen in example 1, where *Jeg* is the holder, *elsker* is the polar expression, which in this case is positive, and *filmen* is the target. The paper aims to utilize variations on the targeted sentiment labels in the dataset itself, giving three differently labeled sets, following the BIO, BILO and BILOU schemes. In addition to this, we try to include a lexicon introduced in (Barnes et al., 2019) as part of a multi task learning architecture.

This task is focused on a simplified version of the dataset, which has been published in relation to the WNNLP 2020 conference task. In this dataset, the polar expressions are conflated with the targets to create a targeted sentiment dataset. The intensity of the polarity is also changed from a three point scale of slight, standard, strong to only one for each of the two polarities: positive and negative. Other information such as opinion holders and information about whether the expressions are on topic or not is not included. The original dataset also has discontinuous spans, but this distinction is not kept in the simplified dataset. The dataset is provided in a CoNLL format with BIO tags. Norwegian is a low resource language, and although research discussed in chapter 2 has shown that a collapsed model might yield lower scores for this type of task, our goal is two-fold: to explore to which degree the model attains lexical information, and to try and measure the effect of a sentiment lexicon and variations in BIO tags on the model. [1]

In chapter 2 we discuss earlier research related to SA, the datasets, BIO tag schemes and lexicon inclusion. Chapter 3 discusses relevant statistics, giving a brief overview of some relevant data. Chapter 4 discusses the modelling, looking at hyperparameter optimization and the MTL architec-

[1]The code is available here: https://github.uio.no/pettemae/WNNLP2020-submission

ture. Chapter 5 then presents the resulting error analysis, where we look at errors in terms of both targets and polarity. We conclude the paper with chapter 6.

2 Related Work

2.1 Dataset

The dataset published for this task has not been subjected to research yet, but the original fine grained dataset is published along with preliminary experiments which might say something about what to expect in terms of metrics. In these experiments, the binary $F_1$was 39.1 and 61.5 for targets and polar expressions respectively, and proportional $F_1$was 31.3 for both (Øvrelid et al., 2019). Some work has been done on the NoReC dataset on less fine-grained levels, namely sentence level (Mæhlum et al., 2019) and document level (Bergem, 2018).

2.2 Tag schemes

The BIO tag set is a popular tagging scheme. The tags stand for **B**eginning, **I**nside, **O**utside, and is sometimes referred to as IOB. An example can be seen in figure 1. Although the basic scheme is popular, variations on the BIO scheme have been tested with promising results. Ratinov and Roth (2009) make use of the BILOU scheme, where L and U stand for **L**ast and **U**nique, respectively. Dai et al. (2015) also look at variations in the scheme, using the IOB term order together with L and U, but calling them E and S, which stand for **E**nd and **S**ingle, respectively. They experiment with three variations of the schemes: IOBE, IOBES and a scheme they call IOB_12E, where the B tag is split into two. We will not look into the latter. They report that their IOB scheme gave the best precision, while the IOBES scheme gave the best recall. Lample et al. (2016) and Ratinov and Roth (2009) also discuss performance variations on the BIO family of schemes. Ratinov and Roth (2009) find that the BILOU scheme performs better than the BIO scheme on their data. However, Lample et al. (2016) using the terms IOB and IOBES, do not find any significant changes in either direction. In this paper we follow Dai et al. (2015) and transform the original data to match the BILO(IOBE) and BILOU(IOBES) scheme and run these three on the best model after initial parameter optimization. All tag schemes map effortlessly to each other, and so the data are automatically transformed beforehand. When running the model, the choice of which tag scheme to use is represented by a hyperparameter in the model.

Jeg	O
likte	O
den	B-targ-Positive
første	I-targ-Positive
sesongen	I-targ-Positive
av	I-targ-Positive
Frikjent	I-targ-Positive
ganske	O
godt	O
.	O

Figure 1: Example of multi-token target expression with positive polarity using the BIO tag scheme

2.3 Modelling

Basic structure Mitchell et al. (2013) uses the same format as for our dataset, modelling the polar expression and target as one, with the surrounding words as context. They use the term 'open domain' on their dataset, which is similar to multi-genre, but might also imply more variation in the *types* of texts included. As our dataset only contains reviews, we will use the term multi-genre. . Mitchell et al. (2013) use the term opinion expression, but following Liu (2015) as in Øvrelid et al. (2019), we use the term polar expression, as not all reflections of inner state in the dataset are opinions, necessarily. Zhang et al. (2015) also look at open domain targeted sentiment, basing on the model presented in Mitchell et al. (2013). They compare a pipeline, joint and collapsed task setup, and conclude that the pipeline and joint models outperform the collapsed one.

Sentiment lexicon A sentiment lexicon is a lexicon in which lexemes or word-forms are listed along with assigned sentiment information, such as 'positive' or 'negative (Bai et al., 2014). A sentiment lexicon can be single-domain or multi-domain (Hu and Liu, 2004). Barnes et al. (2019) mention that for Norwegian there is also a smaller lexicon created by Bai et al. (2014). Mitchell et al. (2013) also employ a sentiment lexicon. In this work, we chose to follow their tracks, and employ a sentiment lexicon for Norwegian by Barnes et al. (2019). Their sentiment lexicon is provided

in an expanded (full-form) and unexpanded form, but based on their results we chose to only employ the unexpanded lexicon, which does therefore not contain all word-forms for each lemma. As our baseline model is more similar to this work, we try to follow their method and extend the baseline model using Multi Task learning (MTL) with sentiment lexicon prediction as our auxiliary task. Barnes et al. (2019) write that their lexicon MTL had greater effects for English than for Norwegian, but note that this could be due to differences in the main task, where the English main task was sentence level sentiment, while for Norwegian it was whether or not sentences were evaluative. As our task involves identifying the right sentiment, we believe there might be a larger effect from including the lexicon.

Our embeddings for Norwegian are taken from the online repository of embeddings from Fares et al. (2017)[2]. We chose the Norwegian embeddings made from the Norwegian Newspaper Corpus (Norsk aviskorpus), using Gensim, with an embedding size of 300. The corpus has a vocabulary size of 1487994. These embeddings were used for both the main task and the auxiliary task in the runs using MTL with the lexicon.

3 Statistics

In order to understand the dataset and some of the problems it poses, it can be useful to look at some initial statistics. Being a multi-genre corpus it follows that there is potentially minor overlap between target tokens across the different genres. As expected, the dataset exhibits high lexical diversity. The 3339 targets found in the train set are comprised of 2366 unique items. Of the 628 targets in the development set, there are 491 unique items, and only 72 of these are found as they are in the train set. The most common targets are generally pronouns, such as *den* 'that; it', but some genre-specific targets are among the most frequent ones, such as *filmen* ' the movie', *spillet* 'the game' and *albumet* 'the album'. Some statistics are reported in table 1.

4 Modelling

Baseline A Bidirectional Long-short term memory (BiLSTM) was used as a baseline. The baseline model was provided by the WNNLP2020 chairs and then built upon. It was first run as is, in order to set the baseline for the task. The model parameters are as follows: learning rate=0.01, layers=1, hidden dimension=100. The baseline model was tested with the embeddings described above. The results are reported in table 2. The scores are somewhat lower than the numbers reported in Øvrelid et al. (2019). We compare our later results to these two results.

[2]Available at http://vectors.nlpl.eu/repository/#

	train	dev	test
sents	6145	1184	930
targets	3339	629	511
positive	2245	433	366
negative	1094	196	145

Table 1: Corpus statistics

metric type	F_1-score	σ
binary	36.0	0.018
proportional	22.1	0.011

Table 2: F_1-score and standard deviation for the provided baseline

4.1 Hyperparameter optimization

Only a simple hyperparameter grid search was done for this task. These were meant as initial testing to have some exploration of the hyperparameter space, as this task is new. The first hyperparameter testing was done with the simple, original BIO tags. The raw baseline model had F_1-scores of 36.0(0.018) and 22.1(0.011) for binary and proportional overlap, respectively. A small parameter space was explored, looking at the effect of number of epochs, the hidden dimension size and number of layers. Difference in epochs were explored as initial testing showed that some of the models took up to 20 hours to finish, and the authors wanted to see if similarly performing models could be achieved with fewer iterations. The results are reported in table 3. Following Barnes et al. (2019), all models are run with five different random seeds, and reported as the average of these with standard deviation. The best model based on binary overlap turned out to be with two layers, 200 hidden dimensions and 50 epochs. Although the same model with 32 epochs

l.	hd.	epoch	binary		prop.	
			F_1	σ	F_1	σ
1	100	16	38.0	0.014	23.8	0.011
		32	37.1	0.008	21.2	0.038
		50	36.0	0.018	22.1	0.011
	200	16	38.3	0.017	24.4	0.010
		32	37.7	0.026	24.4	0.019
		50	38.5	0.014	23.4	0.015
2	100	16	39.5	0.010	23.3	0.023
		32	38.9	0.012	23.0	0.023
		50	39.5	0.013	23.9	0.019
	200	16	39.7	0.012	24.0	0.018
		32	39.8	0.009	**24.8**	0.015
		50	**39.9**	0.008	24.7	0.020

Table 3: F_1-score and standard deviation for the baseline model with limited hyperparameter optimization for hidden dimensions(hd.), layers (l.) and epochs. Models using the lexicon auxiliary task are marked with "+"

had a slightly higher proportional overlap of 24.8, the former model was chosen as the basis for looking at the bio scheme variations.

4.2 BIO-scheme optimization

After running the hyperparameter testing, the best model was trained with three different schemes: BIO, BILO and BILOU. The results are reported in table 4. Somewhat surprisingly it seems like the BILO tags performs very slightly better than the two other schemes. All numbers are from averaging after five runs, with standard deviation, following Barnes et al. (2019).

4.3 Lexicon inclusion

Following the BIO-scheme optimization, we tried running with the sentiment lexicon information. The sentiment lexicon prediction task code is based on Barnes et al. (2019), using a feed-forward network with a ReLU nonlinearity. Our MTL set up closely follows Barnes et al. (2019), but with some simplifications. We do not compare the performance with an English dataset. Like in their setup, we include one linear layer that is shared between the main targeted sentiment task using the BLSTM, and a simple feed-forward network with softmax for the lexicon classification task. As the lexicon task is very similar to the part

	binary		prop.	
model	F_1-score	σ	F_1-score	σ
BIO	39.9	0.008	24.8	0.020
BILO	**40.1**	0.012	24.0	0.010
BILOU	39.9	0.006	24.8	0.026
BIO+	35.3	0.013	18.0	0.017
BILO+	33.8	0.020	17.4	0.022
BILOU+	35.4	0.029	16.0	0.058

Table 4: F_1-score and standard deviation for models.

of the main task that includes deciding the polarity of each target that is identified.

4.4 Metrics

We use a binary overlap and proportional overlap scores to evaluate the models. The binary metric counts at least one overlapping label as a match, while the proportional overlap counts the total number of overlapping tags, and the latter is therefore a stricter metric. For the initial hyperparameter testing these were done on the basis of the labels as they are, but for more fine-grained testing when looking at the lexicon information and BIO-schemes, we also looked at scores separately for matching polarity, to better understand how the techniques might influence this part differently. These were done using proportional overlap.

5 Error Analysis

The results of the best model were looked into in order to get insight into what could be made better, before running on the test set. We separate the errors into two groups: errors related to target spans, and errors related to the polarity of those spans.

It is interesting to see if the model can identify the correct target, even without complete overlap, regardless of the polarity. Polarity errors on the other hand are tied to identifying the polar expressions in the sentence. These are not explicitly marked, so it is up to the model to be able to identify which parts of a sentence contribute to the polarity of a sentence.

5.1 Baseline overlap

As noted there is not much overlap between the target expressions in the dev set and the train set.

	correct		incorrect	
model	absolute	%	absolute	%
BIO	379	35.4	1156	27.9
BILO	380	35.8	1098	28.8
BILOU	365	36.4	923	31.1
BIO+	383	31.6	1374	24.2
BILO+	334	33.2	1092	26.2
BILOU+	394	29.7	1515	20.2

Table 5: Number of correctly and incorrectly classified tokens, and percent-wise overlap with train vocabulary.

	prop.	
model	F_1-score	σ
BIO	71.0	0.047
BILO	69.9	0.033
BILOU	69.4	0.015
BIO+	67.6	0.050
BILO+	60.0	0.016
BILOU+	56.1	0.047

Table 6: F_1-score and standard deviation for proportional polarity scores

We took a closer look at the misclassfied sentences for the baseline model in order to have a baseline for these errors.

5.2 Best model errors

The best model after hyperparameter testing was examined, along with the same model run with the BILO and BILOU tags. One noticeable thing is that there is much variation across the random seeds, and although the models agree on several points, there are still differences. These are reported. We look at both what the models managed to get right, as well as some of the most common errors.

Negation issues The corpus is not annotated with negation information. This means that it might be difficult for the model to be able to correctly identify these cases. This is also a source of errors. This should not affect the overall ability for the model to identify targets correctly.

5.3 Target errors

There are many problems with target identification in the models. It is difficult to generalize all errors, but some patterns seem to be apparent, and will be discussed here. As noted, there is not much lexical overlap between the training set and the development set. Therefore it is to be expected that this might influence how well the model is able to deal with new words. In table 5 we have summarized some findings related to lexical overlap. It seems that for all models, there is a smaller overlap between sentences that were mislabeled, compared to those that were correctly labeled. This could be an indication of how seeing words that are familiar can be helpful. At the same time, the percentage is not very high, and this means that the models are indeed able to pick up on other signals than purely those based on tokens it has seen during training.

Entity recognition Most of the models seem to have gotten quite good at recognizing actual entities, regardless of whether they are actually targets or not.

Capital letters One problem that seems to be happening is that several cases of capitalized words are treated as entities even in cases where they are not. This can be seen as the models often treat the first, capitalized token of a sentence as a target.

Punctuation Several targets are annotated with enclosing '«' and '»'. These also seem to be picked up by several of the models. These are actually very frequent in the training set, with 261 and 264 occurrences respectively.

BIO syntax and mixed polarity In some cases the models chose tag sequences that are not possible within the bio-scheme, such as I-tags without a leading B-tag, and examples of mixed polarity, such as a B-tag with a positive tag and an I-tag with a negative tag, or a mix of the two, as seen in sentence 2, where in one randomization *valgfri figur* is erroneously identified as I-targ-Positive and I-targ-Negative, respectively.

This can be mitigated by penalizing illegal sequences more heavily in the backward propagation. In our experiments, there seems to be a difference between the three tag schemes. The average number of multitoken targets tagged with mixed polarity are: BIO=21.6(σ=7.23), BILO=18.8(σ=1.30), BILOU=17.4(σ=5.13)

(2) *[...] de fleste utfordringer kan løses*
[...] the most challenges can be.solved
med en valgfri figur .
with a elective figure .

'[...] most challenges can be solved with a figure of your choice'

5.4 Polarity errors

In addition to problems regarding the targets themselves, we wanted to look into problems with polarity. Polarity is measured using a type of proportional overlap. Given a token tagged as targ by both the gold and the predictions, we check whether the polarity matches. The polarity results are reported in table 6, and confusion matrices are shown for all three tag schemes (2). As can be seen, the results from BIO and BILO are similar, with a similar distribution, while the BILOU has a slightly higher number of true positives in relation to false negatives, and a overall lower number of overlaps with the gold sets. Polarity classification can vary between runs with different random number initializations. For example, in sentence 3, the target *designjobb* is correctly identified as a target by all randomizations of all three models, but it is tagged as 'positive' by some, and 'negative' by others. Originally the authors thought that the lexicon would improve the polarity score for the task, but from the data in table 6 it can be seen that if anything, the models did somewhat worse when including the lexicon auxiliary task. It is believed that this does not necessarily mean that lexicon inclusion has to be detrimental to modal F_1-scores, but rather that more fine-tuning is necessary to properly benefit from the sub-task, such as finer control of loss such as early stopping, and number of epochs, as well as separate learning rates for the auxiliary task. In the code for these models, both the main task and the auxiliary task run the same number of epochs, a choise the authors realize is suboptimal. These hyperparameters were not explored fully in this paper.

(3) *En middels god designjobb [...].*
A medium good design.job [...]
'A mediocre design job [...]'

5.5 Lexical learning

A possibility when it comes to ML networks is that they can focus on frequencies, and have problems learning useful representations. As noted earlier, there is not much lexical overlap between the train and dev set, and therefore it should be possible to explore to which degree the model has been able to predict targets on nouns not encountered in the train set, or at least that were not marked as such in the train set.

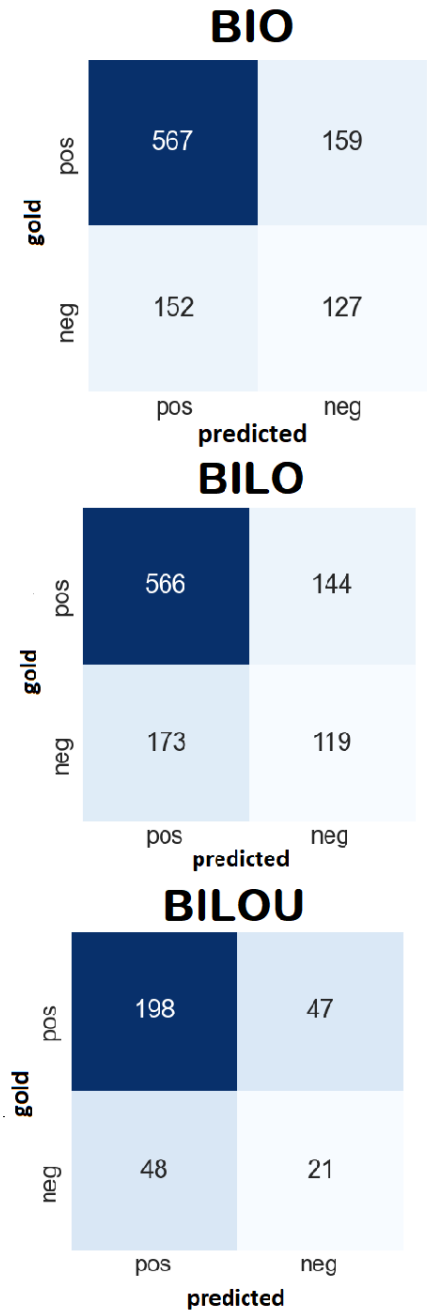


Figure 2: Polarity confusion matrix for the best BIO, BILO and BILOU models

6 Conclusion

Although our experiments have only shown meager improvements in terms of F_1-score compared to the baseline, it seems like it might be worth looking into variations in BIO tag schemes. There is a slight improvement when using BILO tags, and there seems to also be an improvement in terms of learning the tags correctly associated with the more complicated schemes. Using a lexicon as an auxiliary task gave no improvement in terms of F_1-score in our case. This does not mean that this is always the case, and we would urge others to look further into this. Options that can be explored are 1) stricter evaluation of BIO tags to avoid erroneous learning 2) more detailed integration of lexicon information, with more controlled runs for each. There are other cases where inclusion of external data did not lead to an improvement of the overall ensemble, as in Bjerva et al. (2017), but we believe that our experiments show that there might nevertheless be something to gain from experimenting with variations of the BIO schemes. Using the lexicon only with the polarity part of a pipeline task might be more beneficial.

Acknowledgments

We were able to write this paper because of the WNLLP conference, and thank all those involved in giving us this opportunity. We would also like to thank the track chairs Lilja Øvrelid and Jeremy Claude Barnes for their invaluable feedback during the writing process.

References

Aleksander Bai, Hugo Hammer, Anis Yazidi, and Paal E. Engelstad. 2014. Constructing sentiment lexicons in norwegian from a large text corpus. pages 231–237.

Jeremy Barnes, Samia Touileb, Lilja Øvrelid, and Erik Velldal. 2019. Lexicon information in neural sentiment analysis: a multi-task learning approach. In *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, Turku, Finland. Association for Computational Linguistics.

Eivind Alexander Bergem. 2018. Document-level sentiment analysis for norwegian. Master's thesis, University of Oslo.

Johannes Bjerva, Gintare Grigonyte, Robert Östling, and Barbara Plank. 2017. Neural networks and spelling features for native language identification. pages 235–239.

Hongjie Dai, Po-Ting Lai, Yung-Chun Chang, and Richard Tzong-Han Tsai. 2015. Enhancing of chemical compound and drug name recognition using representative tag scheme and fine-grained tokenization. *Journal of cheminformatics*, 7:S14.

Murhaf Fares, Andrey Kutuzov, Stephan Oepen, and Erik Velldal. 2017. Word vectors, reuse, and replicability: Towards a community repository of large-text resources. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 271–276, Gothenburg, Sweden. Association for Computational Linguistics.

Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *KDD '04*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *CoRR*, abs/1603.01360.

Bing Liu. 2015. *Sentiment analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge University Press, Cambridge, United Kingdom.

Petter Mæhlum, Jeremy Barnes, Lilja Øvrelid, and Erik Velldal. 2019. Annotating evaluative sentences for sentiment analysis: a dataset for Norwegian. In *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, pages 121–130, Turku, Finland. Linköping University Electronic Press.

Margaret Mitchell, Jacqui Aguilar, Theresa Wilson, and Benjamin Van Durme. 2013. Open domain targeted sentiment. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1654, Seattle, Washington, USA. Association for Computational Linguistics.

Randolph Quirk, Sidney Greenbaum, Geoffrey Leech, and Jan Svartvik. 1985. *A Comprehensive Grammar of the English Language*. Longman, London.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, pages 147–155, Boulder, Colorado. Association for Computational Linguistics.

Erik Velldal, Lilja Øvrelid, Eivind Alexander Bergem, Cathrine Stadsnes, Samia Touileb, and Fredrik Jørgensen. 2018. NoReC: The Norwegian Review Corpus. In *Proceedings of the 11th edition of the Language Resources and Evaluation Conference*, pages 4186–4191, Miyazaki, Japan.

Janyce Wiebe, Theresa Wilson, and Claire Cardie. 2005. Annotating expressions of opinions and emotions in language. *Language Resources and Evaluation (formerly Computers and the Humanities)*, 39:164–210.

Meishan Zhang, Yue Zhang, and Duy Vo. 2015. Neural networks for open domain targeted sentiment. pages 612–621.

Lilja Øvrelid, Petter Mæhlum, Jeremy Barnes, and Erik Velldal. 2019. A fine-grained sentiment dataset for norwegian.

Word Embeddings for Targeted Sentiment Analysis

Maria Singstad Paulsen
Department of Informatics – University of Oslo
marispau@ifi.uio.no

Abstract

This paper describes a series of experiments on targeted sentiment analysis for Norwegian. Experiments have been conducted using a one-layer bidirectional long short-term memory recurrent neural network, and effects of using different word embeddings, from generalised to domain-specific, are presented in a comparative analysis. Moreover, it includes a description of code for the neural network, written in TensorFlow, as well as the implementation of a word embeddings learner.

1 Introduction

In this paper, there are two overarching topics that will be presented to the reader. Though they differ in many ways, they are not entirely unrelated. The experiments cover analysis and evaluation of using different pretrained word embeddings, from generalised to domain-specific, for fine-grained sentiment analysis, The other main topic has partially come as a by-product of the first, and concerns the choices made with regards to the underlying codebase. Code that can be reused and distributed freely is a valuable tool and contribution to the open-source community. I have worked with this in mind, and focused equally much on reusability as reproducibility.

1.1 Different flavours of sentiment classification

Within the field of sentiment classification, there are several flavours — so to speak — to pick and choose from. At the core of any sentiment analysis task, is the sentiment itself. In its most simplistic form, it often boils down to a binary classification task, where the goal then becomes to predict one of two polar opposites. This can be expanded to not only cover the sentiment itself, but also the target thereof, and further to include a scale rather than polar opposites. A sentiment classification task also including targets is thus considered to be targeted sentiment analysis, where one task concerns named entity recognition (NER), and the other classifying sentiments related to these entities. There are several ways to go about this. One method is to jointly identify and predict entities and sentiments, where the targets are members of two disjoint sets $\mathtt{E} = \{\mathtt{B}, \mathtt{I}, \mathtt{O}\}$ and $\mathtt{S} = \{\mathtt{pos}, \mathtt{neg}\}$. Another method is collapsing the entities and sentiments, and work with a single set that is the union of entities $\mathtt{E}$ and sentiments $\mathtt{S}$ just mentioned. The latter is used in the experiments covered in this paper.

1.2 The NoReC$_{fine}$ dataset

NoReC$_{fine}$ is a fairly recent addition to the collection of datasets for Norwegian, and the first one annotated for fine-grained sentiment analysis. The dataset and annotation efforts are described in detail by Øvrelid et al. (2019) in *A Fine-Grained Sentiment Dataset for Norwegian*, while the underlying corpus is described in *NoReC: The Norwegian Review Corpus* (Velldal et al., 2018).

In this work, a simplified version (NoReC$_{fine^{\dagger}}$) is used. Instead of being placed on a three-point intensity scale, seniments are polar, and while it is annotated with targets, it is not fine-grained with respect to entity relations. That is to say, the annotations concerning details of the target holders' relations are excluded, as is the intensity of the sentiments. An example of a sentence from the train partition (Table 1), shows a two-word entity (*denne bilen*) and the sentiment assigned to it. In the train partition, there are 5915 sentences, of which 2363 contain targets and

Word	Tag
Komfortabel	O
er	O
denne	B-targ-Positive
bilen	I-targ-Positive
fremfor	O
alt	O
.	O

Table 1: An example sentence from the NoReC$_{fine}$† train partition.

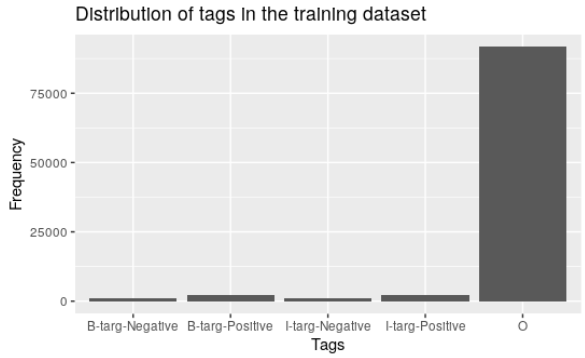


Figure 1: Distribution of tags for NoReC$_{fine}$ (train).

3552 do not.

A definition of this simplified dataset is given in the section below.

Figure 1 shows the distribution of tags in the training partition of the

1.2.1 Formal definitions pertinent to this paper

Let the set of tags be defined as $T = \{B_p, B_n, I_p, I_n, O\}$, where B, I and O, represent named entities, defined as per the IOB format (Ramshaw and Marcus, 1995), and the subscripted letters $_p$ and $_n$ the sentiment polarity of either positive or negative.

Let $S = \{W_1, ..., W_m\}$ be the set of input sentences, and let each element in S be a sequence $W = \langle w_1, ..., w_n \rangle$ of n words. The task can then formally be defined: given an input S, produce an output O such that $|S| = |O|$ and $\forall\, SxOy(|Sx| = |Oy|)$:

$$O = \bigcup_{i=1}^{|S|} \langle t \in T \mid w \in S_i \rangle \qquad (1)$$

The sentences in this dataset do not contain tokens marking the start and end of a sentence, and neither do the sentences fed to the word embeddings learner described in section 5.1.

2 Related work

Over the last decade, there have been several advances in the field of research relating to sentiment analysis. Mitchell et al. (2013) tackled the joint detection of named enteties and the sentiment expressed towards them leveraging linguistically- informed features within conditional random fields (CRFs), and Zhang et al. (2015) expanded on this research a couple of years later, using the CRF as the baseline to which their neural network models were compared.

3 Code today only what you can decode tomorrow

The codebase written for this work originally consisted of two separate packages. The one of which the BiLSTM belongs, is described in this section, and the other in the section that follows. The neural network was written using Google's TensorFlow 2.0 API. This in and of itself amassed a fair amount of work and, given that the provided pre-code was written using Facebook's PyTorch API, will be described in sufficient detail in this section. The neural network consists of modules from the Keras functional API. The necessary building blocks required for a neural network bears some semblance to those of the `tf.keras.Sequential` API. However, the functional API is more flexible, and can handle models with non-linear topology, models with shared layers, and models with multiple inputs or outputs (TensorFlow). For an in-depth presentation of the TensorFlow API and its open-source codebase,(Abadi et al., 2015), refer to *TensorFlow: A system for large-scale machine learning*.

For the purpose of learning how to string together the various pieces required for a neural network, the sequential API, available both in TensorFlow and PyTorch, is both sufficient and well-suited. However, as the architectures become more complex, the advanced user might start to feel bounded by constraints and the lack of flexibility.

User-friendliness and *best practices* are key-terms in fields more closely related to software

engineering than data science and and similar concepts more often discussed in the field of software engineering, I would like to address it for the purpose of motivating the paths taken to conclude the work presented in this paper.

One could hardly argue with is a natural progression from the sequential way of building neural networks, through the modular way, and, if there is a need for it, on to a fully object-oriented way, where most modules are tailor-made for the job at hand.

3.1 Reproducibility

An important aspect of conducting any form of research, is that of reproducibility. With respect to this project in particular, one of the challenges that arose was how to prepare the data in such a way that it was suited as input to a recurrent neural network, yet also maintaining the requirement for reproducible results. Accompanying this project's proposal, was code for a baseline model and various evaluation metrics. However, as this was written using the PyTorch API, most all the code forming the basis for this paper has been written from scratch. As a direct result of this, one major difference between the PyTorch and TensorFlow models, lies in the lack of padding on batch level in the latter. One option explored was that of using the Dataset pipeline, another was using custom generators. In the case of the former, there would be a need to pad both the training and development datasets to the same length. As for the latter, it is not suitable if the model is to be serialised and restored in a different setting at a later time. That is, to say the least, sub-optimal when reproducibility matters.

On the topic of serialising, one other significant issue was that of not being able to deserialise all of the word embeddings trained to use in this experiment. The embeddings were produced using the Gensim library, and tests were done for embeddings created in modules that was part of the same package as the neural network and the rest of the code, as well as a stand-alone package. It was in the case of the latter that deserialising with `pickle.load()` became rather troublesome. Given the scope and allotted time, a satisfactory solution did not seem feasible.

Whenever a seed was either suggested or required, the same seed was used troughout. In addition, a global seed was set through the TensorFlow API rather than using that of the Numpy API. The latter offers a well-known and much used module, `numpy.random`, though it is recommended that the new BitGenerator [1] be used instead. So as to not inadvertedly cause any incompatibility issues, whenever possible, the seeds have been set through the TensorFlow API.

4 Establishing a baseline

This work is in its entirety based on but a single architecture. As was mentioned briefly in section 1, one of the goals has been to not only conduct experiments and report the findings, but also to be able to contribute towards the collective body of work that is open-source.

The architecture of the baseline model is a bi-directional long short-term memory recurrent neural network, BiLSTM for short. Much like the RNN relaxes the Markov assumption and allows looking arbitrarily back into the past, the bi-directional RNN relaxes the fixed window size assumption, allowing to look arbitrarily far at both the past and the future within the sequence (Goldberg, 2017).

Though the architecture of the model is the same as the model distributed with the precode, some of the hyperparameters were adjusted (Table 2) and, as discussed in section 3.1, the length of the padded sequences was set to 50.

Parameter	Value
hidden dim	100
batch size	100
epochs	50
learning rate	0.01
dropout	0.1
recurrent dropout	0.5
sequence length	50

Table 2: Hyperparameters for the baseline model.

In addition, early stopping was implemented on several metrics (Table 3), and in the vast majority of cases, training finished after 6 epochs.

[1] https://numpy.org/doc/1.18/reference/random/index.html

Metric	Patience	Min Delta
False Positives	4	
False Negatives	4	
Validation Accuracy	5	0.1
Validation Loss	5	0.1

Table 3: Callbacks monitored for early stopping.

I chose to operate with a joint baseline, using both the Common Crawl and Wikipedia embeddings (see section 5) downsized from dimensionality 300 to 100. As seen in Table 4, the Wikipedia embeddings achieved the highest scores, both proportional and binary. Admittedly, this came as a bit of a surprise, as I had expected the opposite. I had hypothesised that the underlying Wikipedia dataset would differ the most from that of NoReC$_{fine^{\dagger}}$, thus expecting the Common Crawl embeddings to come out on top.

Metric	Proportional	Binary
Common Crawl (4000000, 100)		
Precision	0.3725	0.5196
Recall	0.0299	0.0417
F1	0.0554	0.0772
Wikipedia (2515788, 100)		
Precision	0.4198	0.6173
Recall	0.0535	0.0787
F1	0.0949	0.1396

Table 4: Baseline scores for the dev partition with native (albeit downsized) fastText embeddings.

5 From generalised to domain-specific embeddings

There is a plethora of different word embeddings to choose from, including embeddings trained for Norwegian. In my experiements, I chose to focus on embeddings carrying subword information, due to the nature of the task at hand, and did only include embeddings where the full word form has been kept intact. The baseline embeddings models were trained on Common Crawl and Wikipedia using fastText (Grave et al., 2018). The models were trained using CBoW with position-weights, in dimensions 300, with character n-grams of length 5, a window of size 5 and 10 negatives. As mentioned in section 1.2, NoReC$_{fine^{\dagger}}$ is a dataset based on a subset of the larger NoReC corpus. There is a decent amount of pre-trained word embeddings available for Norwegian distributed through the NLPL website [2], but none of which has been trained using NoReC.

I decided to use the native fastText Common Crawl embeddings as the baseline(s) for two reasons. One came down to the selection of embeddings in the NLPL repository, and the other because I wanted to train domain-specific ones. In order to be able to compare and contrast, it seemed reasonable to start with something that I suspected might differ the most from the rest. As it turned out, this was perhaps not really the case after all.

In her thesis, *Evaluating Semantic Vectors for Norwegian*, Stadsnes (2018) trains, evaluates and presents results for several of the word embeddings now found in the NLPL repository. Based on her findings, I decided to focus my attention mainly on embeddings trained on either Norwegian Newspaper Corpus (NNC), The Norwegian Web as Corpus (NoWaC), or NBDigital Corpus (NBDigital), a combination of NNC and NoWaC, and a combination of all three. I did not include embeddings trained on NBDigital paired with NNC or NoWaC. For good measure, I did include the one model trained on the Norwegian Bokmål CoNNL17 corpus. As it turns out, it fared the worst of them all.

5.1 Training on the NoReC corpus

All of the embeddings trained for this project used either fastText skipgram or fastText continuous bag-of-words. I used windows of size 5 and 10, a minimum word count of between 4 and 8 and dimensionalities 100 and 300. I trained embeddings on the NoReC train partition, a combination of the train and development partition [3], the Norwegian Dependency Treebank (NDT), and a combination of NoReC and NDT.

Sentences were parsed from CoNNL-U to a single sentence per line so as to comply with

[2] http://vectors.nlpl.eu/

[3] These were not actually used, and so the results are not included in the findings.

the `PathLineSentences` class in the Gensim library, and was not pre-processed any further.

6 Analysis

In total, the baseline model was trained with 32 different word embeddings configurations, of which 27 embeddings were of dimensionality 100, and 5 of dimensionality 300. Due to the fact that most available pre-trained word embeddings from the NLPL repository belong to the former group, I chose to downsize the Common Crawl and Wikipedia fastText embeddings to match this size. In this section, only embeddings of dimensionality 100 will be discussed.

Corpora	Precision	Recall	F1
	fastText Skipgram		
NoReC	0.4763	0.1267	0.2001
	fastText CBoW		
NoReC	0.5614	0.1007	0.1708
	Gensim CBoW		
NNC + NoWaC	0.5154	0.5154	0.1562
	Gensim Continuous Skipgram		
NNC + NoWaC	0.4640	0.0913	0.1525
	fastText CBoW		
NNC + NoWaC + NBDigital	0.4375	0.0826	0.0826

Table 5: Results from evaluating on the dev partition, ranked by proportional F1.

7 Conclusions and Outlook

The embeddings trained on the NoReC corpus showed potential, producing the best results from all embeddings used. However, being that they are domain-specific, it is not unlikely that it would be more beneficial to combine them with embeddings trained for other domains, either by training jointly with a larger set of data, or learn and update alongside the neural network training. Further fine-tuning the model would also be necessary in order

Corpora	Precision	Recall	F1
	fastText Skipgram		
NoReC	0.6331	0.1684	0.2660
	fastText CBoW		
NoReC	0.7237	0.1298	0.2201
	Gensim Continuous Skipgram		
NNC + NoWaC	0.6120	0.1204	0.2012
	Gensim CBoW		
NNC + NoWaC	0.6476	0.1157	0.1963
	fastText CBoW		
NNC + NoWaC + NBDigital	0.5958	0.1125	0.1893

Table 6: Results from evaluating on the dev partition, ranked by binary F1.

Metric	Proportional	Binary
Precision	0.5309	0.6914
Recall	0.1322	0.1721
F1	0.2116	0.2756

Table 7: Evaluation scores for the test partition.

to draw any conclusions, or better yet, replace the collapsed model with either a joint– or a pipeline model, as briefly touched on in section 1.1. The risk of overfitting should also be taken into consideration.

Having said that, perhaps the greatest benefit comes in terms of saving on computing resources. The embeddings trained on NoReC sport a relatively small vocabulary (81345), whereas other embeddings in the top tier had vocabularies of sizes 2551820 (NNC + NoWaC), 2515788 (Wikipedia) and 4428648 (NNC + NoWaC + NBDigital). The NoReC embeddings' vocabulary size is only 1.84–3.23 percent the size of these. A considerable difference.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S.

Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Yoav Goldberg. 2017. *Neural Network Methods in Natural Language Processing.* Number Vol. 37 in Synthesis Lectures on Human Language Technologies. Morgan Claypool Publishers.

Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. Learning word vectors for 157 languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*, Miyazaki, Japan. European Languages Resources Association (ELRA).

Margaret Mitchell, Jacqui Aguilar, Theresa Wilson, and Benjamin Van Durme. 2013. Open domain targeted sentiment. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1654, Seattle, Washington, USA. Association for Computational Linguistics.

Lance Ramshaw and Mitch Marcus. 1995. Text chunking using transformation-based learning. In *Third Workshop on Very Large Corpora*.

Cathrine Stadsnes. 2018. Evaluating semantic vectors for norwegian.

TensorFlow. The keras functional api.

Erik Velldal, Lilja Øvrelid, Eivind Alexander Bergem, Cathrine Stadsnes, Samia Touileb, and Fredrik Jørgensen. 2018. NoReC: The Norwegian review corpus. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*, Miyazaki, Japan. European Languages Resources Association (ELRA).

Meishan Zhang, Yue Zhang, and Duy-Tin Vo. 2015. Neural networks for open domain targeted sentiment. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 612–621, Lisbon, Portugal. Association for Computational Linguistics.

Lilja Øvrelid, Petter Mæhlum, Jeremy Barnes, and Erik Velldal. 2019. A fine-grained sentiment dataset for norwegian.

A Appendices