Swiss German Part-of-Speech Tagging

Christoph Schwizer – 09-918-210
Tim Strasser – 12-742-573

University of Zurich

1 Task Description

The goal of this project was to apply neural network models to a part-of-speech (PoS) tagging task on Swiss German text data. PoS tagging is a common task in the field of natural language processing, but research on Swiss German PoS tagging is scarce. While other statistical models, such as BTagger or TnT tagger [1] have been applied to said task, to our knowledge, no neural network approaches have been evaluated so far. The source code of this project is available on GitHub at `https://github.com/MethDamon/SwissTagger` .

2 Material

We trained and evaluated our models on the NOAH corpus of Swiss German Dialects[1]. The corpus contains 55 articles from five different sources. Each source is stored as an XML file: a news outlet, blog posts, an annual report of the Swatch group, extracts from criminal novels by Viktor Schobinger, and articles from the Alemannic Wikipedia. Overall, the corpus consists of 7'327 sentences, 113'565 words with corresponding PoS tags (86 different tags were used in the corpus), of those words 23'806 are unique (22'526 when ignoring upper and lower case).

Swiss German is not a standardized language, i.e., there are no rules for grammar, syntax, or orthography. Moreover, there are big variations in terms of vocabulary, spelling, and even sentence structure between regional dialects.

3 Method

3.1 Multilayer Perceptron

Initially, we used a simple Multilayer Perceptron[2] with one hidden layer (MLP_M) to try and solve the task, get acquainted with the data and to get a baseline performance measure. We defined features explicitly for this model (see Table 1). Additionally, we applied a dropout of 0.5 between the input and hidden layer, as well as between the hidden and the output layer to avoid overfitting.

[1] `https://noe-eva.github.io/NOAH-Corpus/`
[2] Our implementation is based on the tutorial by Axel Bellec (Link).

feature name	*description*
nb_terms	number of words in the sentence
term	the focus word
is_first	whether the focus word is the first word in the sequence
is_last	whether the focus word is the last word in the sequence
is_capitalized	whether the first character of the focus word is a capital letter
is_all_caps	whether the entire focus word is upper case
is_all_lower	whether the entire focus word is lower case
prefix-1	the first letter of the focus word
prefix-2	the first two letters of the focus word
prefix-3	the first three letters of the focus word
suffix-1	the last letter of the focus word
suffix-2	the last two letters of the focus word
suffix-3	the last three letters of the focus word
prev_word	word before the focus word (or empty string if focus word is first word in sequence)
next_word	word after the focus word (or empty string if focus word is last word in sequence)

Table 1. Features of the MLP_M model.

In a next step, we removed the manual features and fed the input sentences directly to the MLP. The sentences were zero-padded and the words were integer-encoded, i.e., each unique word was represented by an integer (this will be the case for all following models, too). To replace the manual features, we added an embedding layer of dimension 128, which should allow the network to learn good features for the task. We call this model MLP_E.

3.2 Recurrent Neural Network with Long Short-Term Memory Cells

Building upon the approach using a multilayer perceptron, we applied a recurrent neural network (RNN) using bidirectional long short-term memory (LSTM) cells to the task. We will refer to this approach as $BiLSTM$. The model was implemented using the Keras Deep Learning library[3] .

For training the recurrent neural network (RNN), the data mentioned in section 2 was used without taking the article information into account, i.e. all the XML files were read in and all the sentences were stored in one data structure, without storing information about which article they belonged to initially. Based on this list, the data was then split into 90% training data and 10% testing data without shuffling. The lack of shuffling is motivated by the wish to get the model to generalize as much as possible. The task was formulated as solving the problem of mapping a new word to an PoS tag. Words and tags were encoded using a simple sequential numerical mapping, including reserved numbers for padding tags and tags assigned to out-of-vocabulary words (same as in the MLP_E).

[3] keras.io

The RNN model consists of an embedding layer, reducing the input dimensionality, followed by a layer consisting of bidirectional LSTM cells (BiLSTM) followed by the final layer with a softmax activation function, transforming the output to a probability distribution over the PoS tags.

The BiLSTM model was trained using the following hyper parameters resp. additional tunings:

- **Regularization:** Regularization applied to the kernel weights matrix used for linear transformation of the recurrent state. We used a mixed regularizer using both L1 and L2 penalties, using a 0.2 to 0.2 mix.
- **Dropout:** We used a dropout and a recurrent dropout of 0.5. Dropout refers to the fraction of units to drop for the linear activation of the inputs. Recurrent dropout refers to the fraction of the units to drop for the linear transformation of the recurrent state.
- **Adding 1 to the bias of forget gate at initialization:** This adds 1 to the bias at initialization, as recommended in Jozefowicz et al. [2]. This was used in combination with setting the bias initializer to zero.
- **Size of embedding layer:** We chose 512 as the output dimensionality of the embedding layer
- **Size of the BiLSTM layer:** We chose 512 as the output dimensionality of the BiLSTM layer
- **Batch size:** We chose a batch size of 512
- **Number of epochs:** We trained the model using 30 epochs

Additionally, we also trained the model without lower casing words based on the observation that the number of unique words stays roughly the same (23'806 without lower casing, 22'526 with lower casing).

4 Results

We compare our results to an existing implementation of a PoS tagger using the NOAH corpus developed by Nora Hollenstein & Noëmi Aepli [1], which is based on the BTagger[4], referring to it as $BTagger$.

Table 2 shows the resulting test and training accuracies of the different models. Table 3 lists the most common mistakes that the $BiLSTM$ model makes. Figures 1, 2, 3 and 4 show the training accuracy, training loss, validation accuracy and validation loss of the $BiLSTM$ over all epochs. The figures were generated by Tensorboard[5]. As mentioned in section 3, we also trained the $BiLSTM$ model without lower casing words. As results do not differ significantly from each other, we refrain from listing them explicitly.

[4] `https://github.com/agesmundo/BTagger`
[5] `https://www.tensorflow.org/guide/summaries_and_tensorboard`

model	*test set accuracy*	*training set accuracy*	*training epochs*
$BTagger$	90.62%	-	-
MLP_M	80.85%	92.42%	5
MLP_E	66.48%	92.98%	200
$BiLSTM$	70.76%	98.51%	30

Table 2. Results of the different neural network models.

truth	*prediction*	*error counts*	*misclassified words*
ADJA	NN	294 (8.34%)	uhrmachorischi, rundä, extraflachä, subtilä, ...
NE	NN	179 (5.08%)	www.longines.com, Chile, Baker, H.H., Mohammed, ...
NN	APPR	141 (4%)	Neuiuuflage, Ahfang, Damämodäll, Edelstaahl, ...
NN	NE	130 (3.69%)	Uhrä, Brichtsjohr, Damämodäll, Segment, Rennä, ...
VVPP	NN	129 (3.66%)	fortgsetzt, verankoret, iifüegt, inspiriert, untostützt, ...
FM	NN	125 (3.55%)	MOOVE, Heritage, Edition, Watch, Type, Rising, ...
NN	ADJA	98 (2.78%)	Markä, Priis, Tennisturnior, Taläant, Prädikat, ...
FM	NE	73 (2.07%)	GROOVE, La, 180th, Avigation, Future, TENDER, ...
NN	ART	62 (1.76%)	Priis, Markä, Roland-Garros-Stadion, Quarzkalibo, ...
NN	APPRART	57 (1.62%)	Rahmä, Longines-Modell, Ahstööss, Partnorin, ...

Table 3. Ten most common mistakes made by the $BiLSTM$ model. The percentage in the *error counts* column denotes the frequency of this particular error compared to all errors.

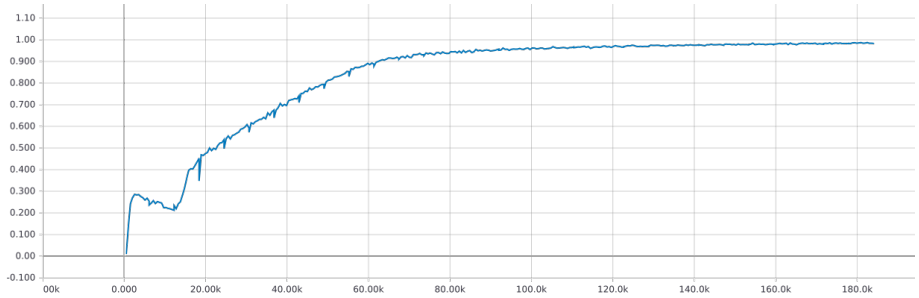

Fig. 1. Training accuracy of the $BiLSTM$ model over time.

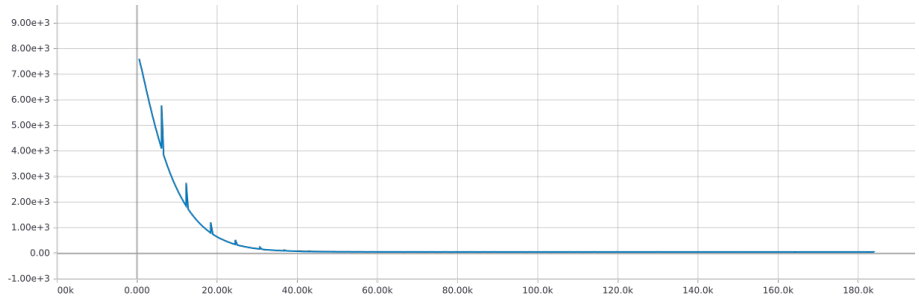

Fig. 2. Training loss of the $BiLSTM$ model over time.

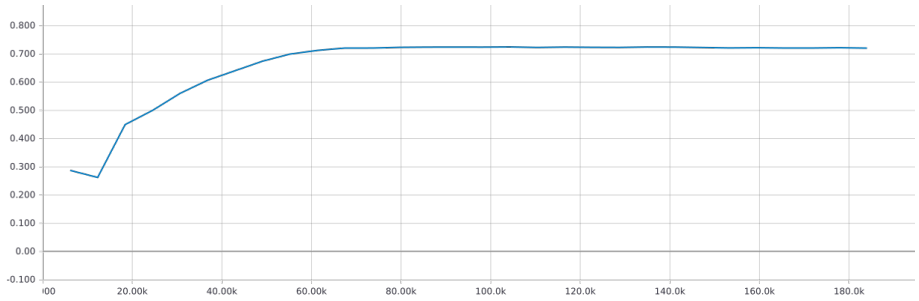

Fig. 3. Validation accuracy of the $BiLSTM$ model over time.

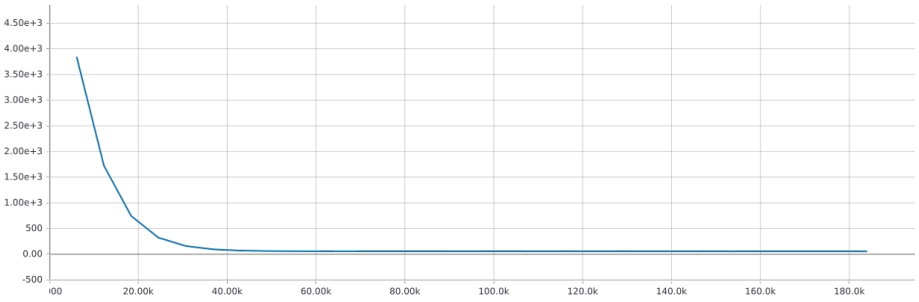

Fig. 4. Validation loss of the $BiLSTM$ model over time.

5 Discussion

Unfortunately, all our approaches to the problem, MLP_M, MLP_E and $BiLSTM$ were not able to reproduce or outperform the baseline results presented in section 4. We attribute this to several things:

Out of vocabulary words. Because there is no clearly defined orthography in Swiss German and a lot of different dialects, words can be written slightly differently. It would therefore be preferable to use character level embeddings instead of a word level embeddings.

Amount of data. As mentioned in section 2, the corpus consists of 7'327 sentences resp. 113'565 words. Deep learning techniques usually require more data to be able to generalize to a sufficient level.

Overfitting. All of the models suffered from severe overfitting issues (see discrepancy between test set accuracy and training set accuracy in Table 2), despite regularization and dropout. This was particularly problematic with the $BiLSTM$, where training accuracy reached 95% after 15 epochs while validation accuracy was still around 72%. We assume that the model complexity is too high for the small amount of data. Potentially, the model memorizes the training words and their most common tag and then simply predicts those tags for each word. The fact that the simplest model, the MLP_M, performed the best supports this theory.

6 Conclusion

As mentioned in section 4, we were not able to achieve the same results as or outperform the baseline approaches. We attribute this to some of the points mentioned in 5. Additionally, we are confident that future work on this topic could profit from looking more into following approaches:

Subword Information. The model could profit from leveraging subword information, i.e., using the output from a LSTM calculating a word representation based on character information as the input to an LSTM calculating the PoS tag scores. An example for this approach is explained in the documentation of the PyTorch machine learning library[6].

Cross-validation. In the case of the NOAH corpus, which consists of five XML files, it would make sense to train a model on, for example, four out of five files and test it on the last file. One could also extend this idea to the level of articles included in the XML files.

[6] https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html

Hyperparameter Optimization. To further optimize the model, future approaches should use an approach like grid search to find the best combination of hyperparameters.

References

1. Hollenstein, N., Aepli, N.: Compilation of a swiss german dialect corpus and its application to pos tagging. In: Proceedings of the First Workshop on Applying NLP Tools to Similar Languages, Varieties and Dialects. pp. 85–94 (2014)
2. Jozefowicz, R., Zaremba, W., Sutskever, I.: An empirical exploration of recurrent network architectures. In: International Conference on Machine Learning. pp. 2342–2350 (2015)