Tagging using Fixed-Ordinally Forgetting Encodings (FOFEs) on character level

Anna Bettina Steinberg,
Matrikelnr.11742581

Course: Profilierungsmodul II: Deep Learning in NLP
WS 2018/19
Ludwig-Maximilians-Universität München
a.steinberg@campus.lmu.de

1 Introduction

The concept of word embeddings, used as input to neural networks, constitutes one of the most remarkable advancements in Natural Language Processing in the last decade. This paper examines a particular type of word embeddings, namely the FOFE method which deterministically encodes each word using its letters, and compares it to classical randomly initialised word embeddings.

1.1 FOFE Encoding

The Fixed-Sized Ordinally Forgetting Encoding Method was introduced by Zhang et al. [1] originally to encode variable-length sequences of words into a fixed-size representation for a sentence. The encoding is based on a forgetting factor which varies depending on the position of the word in the sequence. The authors test their encoding method on two language modelling benchmark tasks. Their best FOFE Feed-Forward Language Model outperforms all baseline n-gram-based Feed-Forward Language Models and even a Recurrent Language Model in terms of model perplexity. The impressive results by Zhang et al. have raised the question whether such fixed-sized encodings can be useful in other NLP tasks as for example POS-tagging.

This paper aims at testing the FOFE encodings for two different tagging tasks. The encoding is, however, not applied to sequences of words as in the original paper, but instead to sequences of characters. Consequently the forgetting factor varies with the position of the character within the word. The goal is to come up with a word representation similar to a classical word embedding. To give an example of how the character-based FOFE method works, consider the word *abcb* and an exemplary forgetting factor of $p = 0.5$. The basic idea is

[1] Zhang, S., Jiang, H., Xu, M., Hou, J. and Dai, L., 2015. The fixed-size ordinally-forgetting encoding method for neural network language models. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers) (Vol. 2, pp. 495-500)

to first transform each character of the word into a one-hot vector with as many dimensions as there are distinct characters in the word. Each one-hot vector is then weighted by a factor depending on its position in the word. This factor equals the forgetting factor to a power corresponding to the position of the character in the word starting from the end of the word. In the final step all these vectors are summed up to result in a word representation. To understand the exact calculations, consider Table 1.

Starting from the end of the word the character b is transformed into a one-hot vector with length 3 since there are 3 distinct characters in this word. It is then multiplied with a factor of $0.5^0 = 1$ because it is the last character. The next letter is equally transformed into a one-hot vector, but multiplied with a smaller factor of $0.5^1 = 0.5$. This process continues until the start of the word at which point the resulting vectors are added up to the word representation in the far right column of Table 1. In the original paper this process is realised by a recursive formula

$$z_t = \alpha \cdot z_{t1} + e_t \tag{1}$$

letter	a	b	c	b	FOFE Code
multiplying factor	0.5^3	0.5^2	0.5^1	0.5^0	
transformed vector	$\begin{pmatrix}0.125\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\0.25\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\0.5\end{pmatrix}$	$\begin{pmatrix}0\\1\\0\end{pmatrix}$	$\begin{pmatrix}0.125\\1.25\\0.5\end{pmatrix}$

Table 1. Example of applying FOFE on character level to the word *abcb*

where z_t represents the FOFE code for the sequence up to the t-th character of the word, α is again the fixed forgetting factor and e_t is the one-hot vector for the t-th character. This formula forms the basis of this project.

1.2 Task

The goal of this project is to evaluate the FOFE method for tagging word sequences. As a first step, a FOFE layer is implemented using the *Pytorch* framework which takes character separated word sequences and encodes them using the FOFE method. The encoded sequences are then passed to a recurrent neural network to predict tags. Classical randomly initialised word embeddings passed to the same network serve as a baseline. Two tagging tasks are evaluated: a BIOS and a POS tagger.

2 Implementation

2.1 Data

The BIOS tagger is evaluated on the Airline Travel Information System (ATIS[2]) dataset which consists of 5871 sentences describing flights and their departure and arrival time and place. An exemplary sentence would be

> i want to fly from boston at DIGITDIGITDIGIT am and arrive in denver at DIGITDIGITDIGITDIGIT in the morning

The corresponding tagging sequence is the following

> O O O O O $B-fromloc.city\_name$ O $B-depart\_time.time$ $I-depart\_time.time$ O O O $B-toloc.city\_name$ $OB-arrive\_time.time$ O O $B-arrive\_time.period\_of\_day$

There are in total 127 different tags of the above kind. The outside tag "O" is considerably more frequent relative to the other tags in the data set. To address this issue the F1 score is weighted using the inverse label frequencies.

Since the ATIS data set is rather small and only includes English sentences, additionally a part of the German TIGER corpus [3] was used to explore the FOFE method with a traditional Part-of-Speech Tagger. The complete TIGER corpus consists of 50000 sentences from newspaper text. For the purpose of this project only 16000 sentences were used to aid computation time. An overview of the two data sets is given in Table 2.

It should be noted that for the implementation of the FOFE method on character level the words in the data sets have been split into individual characters and the characters then transformed to IDs in a preprocessing step.

Dataset	Samples	Train	Dev	Test	Tags
ATIS	5871	3982	996	893	127
Tiger	16000	10000	5000	1000	53

Table 2. Descriptive statistics of both data sets

2.2 Architecture

Apart from the embedding layer, the FOFE and the classic model share the same architecture. The FOFE method is implemented as a separate neural layer in *Pytorch* with the forgetting factor set as a trainable parameter. For the classic model a standard randomly-initialised embedding layer is used. A drop-out layer is implemented after the FOFE and the traditional embedding layer respectively.

[2] downloadable form https://www.kaggle.com/siddhadev/ms-cntk-atis
[3] https://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/tiger.html

The core of the architecture is a bi-directional Gated Recurrent Unit (GRU). The output of the GRU is linearly projected and a softmax activation is applied (it is included in the loss calculations).

2.3 Difficulties

While I expected the implementation of a custom neural layer to be the bottleneck of the project this task turned out to be straight-forward. As in any neural network one needs to first define the variables in the `__init__` function of the layer. Here the class variable `forgetting_factor` needs to be defined as *Pytorch* `Parameter`. The `forward` function of the layer performs the encoding by looping over the words of a sentence and using the characters of each word to calculate a FOFE representation for each word.

Major problems arose when the data set was split into batches and the *PyTorch* `pack_padded_sequence()` function was used to pass batches to the neural network. Firstly, the function can only be used with RNN layers, so the packing has to be done in the `forward` function of the main neural network and not in the preprocessing. On the other hand the padding has to be performed before passing the input to the network which accepts only tensors. For reasons that I still haven't figured out using batches and `pack_padded_sequence()` results in incorrect scores for loss and accuracy although the padding and packing is implemented correctly. This issue has already been raised in *PyTorch* online discussion forums [4], however, no fix is available for this issue to the best of my knowledge. Unfortunately it is rather difficult to provide a reproducible example for this problem. To circumvent this problem single sentences were passed to the network during training.

3 Results

For the neural network the following parameters are set

- size of embedding layer = 50 (only for classic model)
- drop-out rate = 0.5
- size of hidden layer in GRU = 50
- optimiser = *Adam* with default learning rate ($lr = 0.001$) and no weight decay
- loss function = cross entropy loss

and its results are shown in Figure 1. The Classic model clearly outperforms the FOFE model with a maximal accuracy of 0.98 in epoch 20 versus a maximal accuracy of 0.91 in epoch 30. Since the Classic model reaches such a high accuracy after epoch 20 the training is stopped afterwards while it continues for the FOFE model. However, the FOFE model's training loss does not decrease

[4] https://discuss.pytorch.org/t/padding-for-rnn-with-pack-padded-sequence/12481/4
https://discuss.pytorch.org/t/correct-way-of-using-packed-sequence/2675

noticably after 25 epochs, so the training is stopped at epoch 30. Since the distribution of tags in the ATIS data set is unbalanced with many words tagged with the outside token, a more meaningful metric is the F1 score weighted with inverse label frequencies. Here the Classic model reaches its maximal value of 0.74 in epoch 15 while the FOFE model only attains a value of 0.48 in epoch 30 with still rising tendency. Exact values can be found in Table 3. The optimal forgetting factor is estimated at 0.98 which is surprisingly high.

Due to its greater size training on the Tiger corpus subset was extended to a maximum of 50 epochs for the FOFE model and 40 epochs for the Classic Model. Results are shown in Figure 2. Again, the FOFE model stays behind the Classic model, not even undercutting a training loss of 1.0. Additionally, the FOFE model seems to stop learning after epoch 20. While the Classic model reaches a maximal accuracy of 0.91 and a maximal weighted F1 score of 0.78, the FOFE model does not surpass an accuracy of 0.71 and a F1 score of 0.5. The forgetting factor that produced these results is 1.27 which raises the question. The best values for all five metrics are summarised in Table 3.

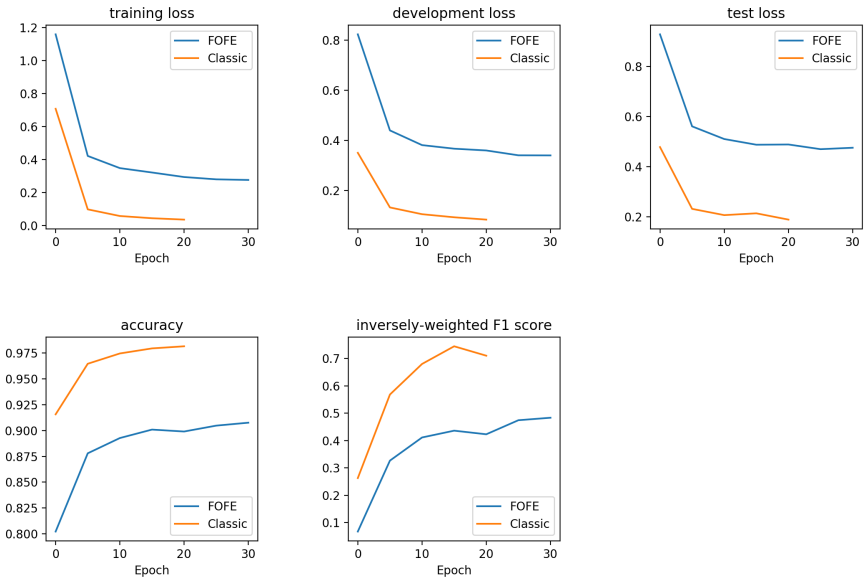

Fig. 1. Results for the ATIS data set

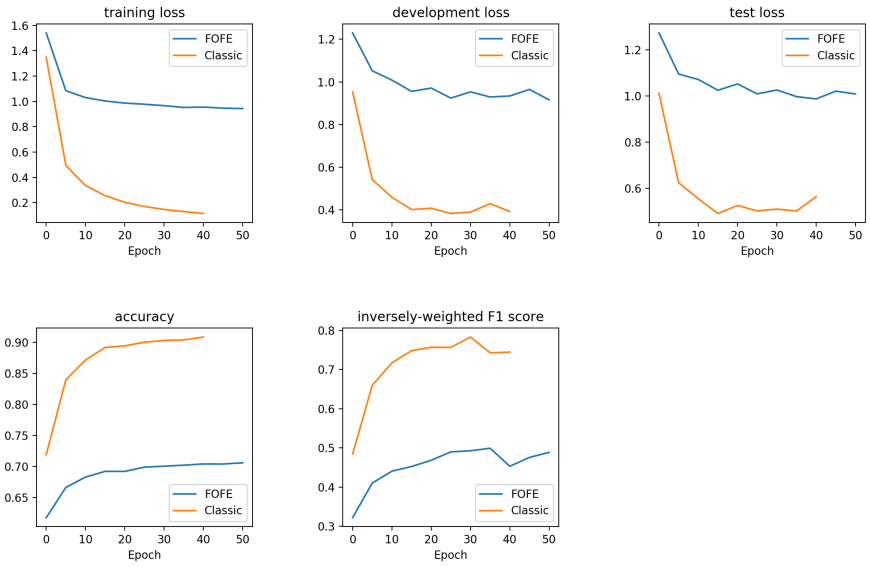

Fig. 2. Results for the Tiger data set

Data set	train loss		dev loss		test loss		accuracy		weighted F1	
	FOFE	Classic	FOFE	Classic	FOFE	Classic	FOFE	Classic	FOFE	Classic
Atis	0.28	0.04	0.34	0.08	0.47	0.19	0.91	0.98	0.48	0.74
Tiger	0.94	0.11	0.92	0.38	0.99	0.49	0.71	0.91	0.5	0.78

Table 3. Results summarised for ATIS and Tiger data sets

4 Conclusion

From the results obtained from two different data sets, using the FOFE method as an alternative embedding layer for tagging tasks does not lead to an increase in performance. It might be that different parameter settings produce better results. This could be efficiently tested using hyper parameter optimisation [5].

[5] It turns out that using hyper parameter optimisation the performance for the FOFE method can be slightly boosted using a more complex network (hidden size of 200 instead of 50) leading to a forgetting factor of 0.91