Assignment 3

Group: Anette Fredriksen (anettfre), Anthi Papadopoulou (anthip), Nick Walker (nichow)

April 2021

Code: https://github.uio.no/anthip/IN9550-IN5550/tree/master/obligatory3

1 Sequence tagging with NorBERT

We started by using the parse function from the conllu tool to read the data, and then we used indexing to get the ['form'] and ['misc']['name'] for our Conllu-Dataset class, from each TokenList object the function returned. For the initial model we used Huggingface's Bert.from_pretrained to load NorBert, and a basic linear layer on top of that. We also made sure to pad the sequences with -1 to the right so that they all had the same length, based on the length of the longest sequence for each batch, and also ignore that -1 in our predictions by setting *ignore_index=-1* in our CrossEntopyLoss criterion.

One major problem we encountered was the high imbalance of classes in the dataset as one can see in Figures 1, 2, and 3, which show the distribution of classes in the entire dataset, and then in the split training and validation, both as a percentage and as logged values for smoother visualization.

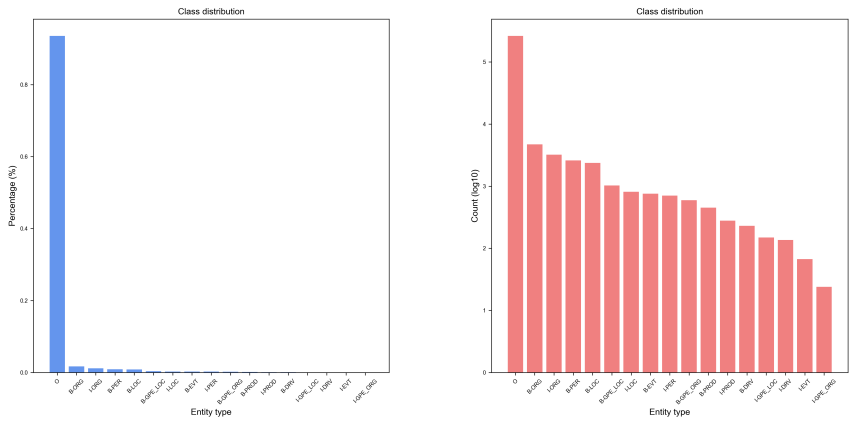

Figure 1: Class distribution in the entire dataset

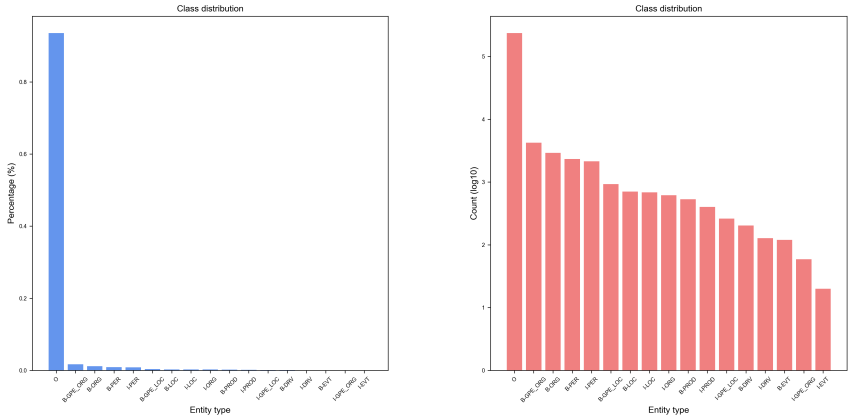

Figure 2: Class distribution in the split training dataset

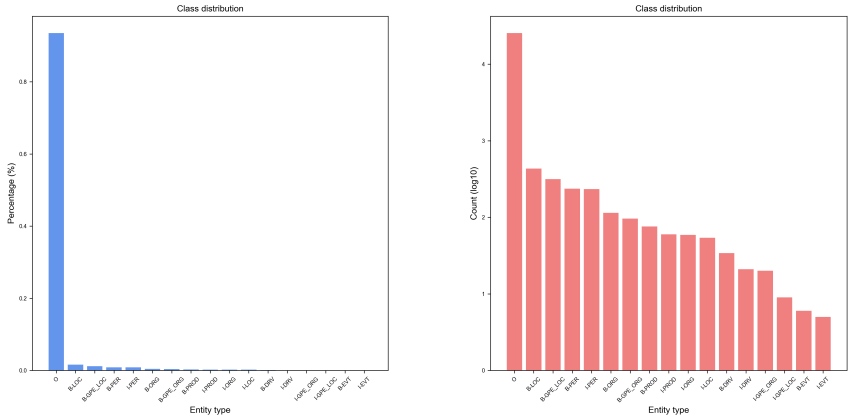

Figure 3: Class distribution in the split validation dataset

As we can see in all three figures the distribution of classes remains the same before and after splitting, with the 'O' class being the dominant class in each case. To counteract this we tried calculating the weights for each class, we then logged these values to smooth the big difference between them and finally multiplied them by a constant of 0.15. We then lowered the weight of the 'O' class to 0.1 and passed all these weights as a tensor to CrossEntopyLoss as *weight=weights*.

We also tried to implement a basic type of under-sampling of the data by removing a number of sentences that consisted **only** of non-entities ('O'). This lowered the number of instances in the entire dataset from *262.603* to *177.187*,

but did not seem to influence the result further , so we did not continue experimentation with it.

The performance of our basic NorBERT model with one linear layer on top is shown in Table 1 without the use of weights while Table 2 shows the same model's performance with the use of weights. We can also see the training and validation loss of the former plotted in Figure 4.

	precision	**recall**	**f1 score**
B-DRV	0.00	0.00	0.00
B-EVT	0.00	0.00	0.00
B-GPE_LOC	0.00	0.00	0.00
B-GPE_ORG	0.00	0.00	0.00
B-LOC	0.00	0.00	0.00
B-ORG	0.00	0.00	0.00
B-PER	0.77	0.11	0.20
B-PROD	0.00	0.00	0.00
I-DRV	0.00	0.00	0.00
I-EVT	0.00	0.00	0.00
I-GPE_LOC	0.00	0.00	0.00
I-GPE_ORG	0.00	0.00	0.00
I-LOC	0.00	0.00	0.00
I-ORG	0.00	0.00	0.00
I-PER	1.00	0.01	0.02
I-PROD	0.00	0.00	0.00
O	0.94	1.00	0.97
micro avg			0.93
macro avg	0.16	0.07	0.07
weighted avg	0.89	0.93	0.91

Table 1: Basic NorBERT model, no weights

	precision	**recall**	**f1 score**
B-DRV	0.63	0.15	0.24
B-EVT	0.00	0.00	0.00
B-GPE_LOC	0.43	0.91	0.58
B-GPE_ORG	0.72	0.24	0.36
B-LOC	0.61	0.46	0.52
B-ORG	0.48	0.79	0.59
B-PER	0.64	0.94	0.76
B-PROD	0.43	0.32	0.37
I-DRV	0.00	0.00	0.00
I-EVT	0.00	0.00	0.00
I-GPE_LOC	0.92	0.37	0.52
I-GPE_ORG	0.00	0.00	0.00
I-LOC	1.00	0.03	0.06
I-ORG	0.37	0.62	0.46
I-PER	0.76	0.91	0.83
I-PROD	0.48	0.34	0.40
O	0.99	0.97	0.98
micro avg			0.96
macro avg	0.50	0.41	0.39
weighted avg	0.97	0.96	0.96

Table 2: Basic NorBERT model, weights

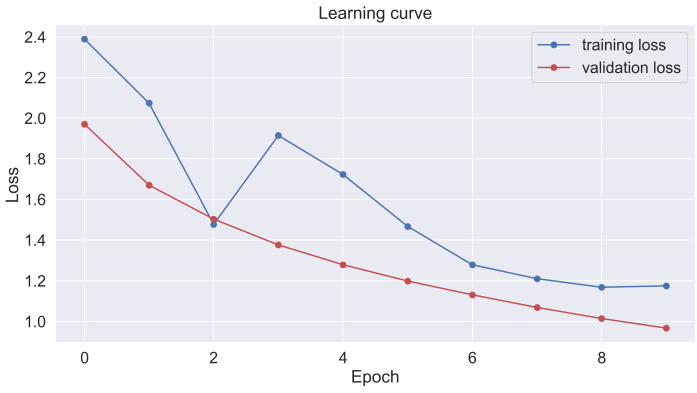

Figure 4: Training and validation loss plot

Since the use of weights had a clear benefit in the performance of the model, the experimentation onwards was done with that as a default.

1.1 Fine-tuning

To fine-tune all of Bert's hyperparameters we set *param.requires_grad = True* for all of them in our VecModel class. This had a positive effect in the performance as shown in Table 3. Higher training time was also needed as expected (approx. 10 hours).

	precision	**recall**	**f1 score**
B-DRV	0.61	0.92	0.73
B-EVT	0.74	0.74	0.74
B-GPE_LOC	0.91	0.89	0.90
B-GPE_ORG	0.63	0.86	0.73
B-LOC	0.86	0.90	0.88
B-ORG	0.90	0.94	0.92
B-PER	0.95	0.99	0.97
B-PROD	0.62	0.82	0.71
I-DRV	0.67	0.77	0.71
I-EVT	1.00	0.56	0.71
I-GPE_LOC	0.78	0.90	0.84
I-GPE_ORG	1.00	0.56	0.71
I-LOC	0.78	0.90	0.84
I-ORG	0.78	0.90	0.84
I-PER	0.95	0.99	0.97
I-PROD	0.77	0.77	0.77
O	1.00	0.99	1.00
micro avg			0.99
macro avg	0.83	0.84	0.82
weighted avg	0.99	0.99	0.99

Table 3: Fine-tuned basic NorBERT model

Secondly, we tried fine-tuning only the embedding layers, by first getting the model's *named_children()*, locating the embedding layer by name, and allowing this layer's children to fine-tune only, by setting *param.requires_grad = True* for that, and *param.requires_grad = False* for the rest we did not want to fine-tune. These then were passed these to AdamW as *filter(lambda p: p.requires_grad, model.parameters())*. The results are shown in Table 4.

Finally, we also tried changing the regularization of our AdamW optimizer by choosing to apply weight decay to all parameters but bias and layer normalization, by first listing the model's parameters, adding the ones we do not want

to apply weight decay (*weight_decay_rate = 0.0*) to in a list and then passing those grouped parameters to AdamW. The results are shown in Table 5.

	precision	recall	f1 score
B-DRV	0.00	0.00	0.00
B-EVT	0.00	0.00	0.00
B-GPE_LOC	0.62	0.88	0.72
B-GPE_ORG	0.00	0.00	0.00
B-LOC	1.00	0.05	0.09
B-ORG	0.54	0.63	0.58
B-PER	0.59	0.91	0.72
B-PROD	0.80	0.05	0.09
I-DRV	0.00	0.00	0.00
I-EVT	0.00	0.00	0.00
I-GPE_LOC	0.60	0.36	0.45
I-GPE_ORG	0.00	0.00	0.00
I-LOC	0.00	0.00	0.00
I-ORG	1.00	0.09	0.16
I-PER	0.77	0.62	0.69
I-PROD	1.00	0.05	0.09
O	0.99	0.99	0.99
micro avg			0.97
macro avg	0.46	0.27	0.27
weighted avg	0.96	0.97	0.96

Table 4: Fine-tuned embeddings only

	precision	recall	f1 score
B-DRV	0.00	0.00	0.00
B-EVT	0.00	0.00	0.00
B-GPE_LOC	0.50	0.92	0.65
B-GPE_ORG	0.00	0.00	0.00
B-LOC	0.68	0.24	0.36
B-ORG	0.58	0.66	0.62
B-PER	0.56	0.91	0.69
B-PROD	0.58	0.09	0.15
I-DRV	0.00	0.00	0.00
I-EVT	0.00	0.00	0.00
I-GPE_LOC	0.80	0.40	0.53
I-GPE_ORG	0.00	0.00	0.00
I-LOC	0.00	0.00	0.00
I-ORG	0.71	0.20	0.32
I-PER	0.73	0.75	0.74
I-PROD	0.77	0.26	0.38
O	0.99	0.99	0.99
micro avg			0.96
macro avg	0.41	0.32	0.32
weighted avg	0.96	0.96	0.96

Table 5: Selective regularization

1.2 Architecture

Apart from the basic NorBERT model with a simple Linear layer we tried adding a multi layer FFNN on top of BERT. It consisted of two linear layers of size 768, 256 respectively and a dropout rate of 0.1, as well as a ReLU activation function to remove negative values. Table 6 and Table 7 show the performance without fine-tuning and with full fine-tuning. Again, training time was affected by fine tuning all of Bert's parameters, as expected.

	precision	recall	f1 score
B-DRV	0.42	0.17	0.24
B-EVT	0.00	0.00	0.00
B-GPE_LOC	0.48	0.91	0.63
B-GPE_ORG	0.33	0.41	0.37
B-LOC	0.49	0.68	0.57
B-ORG	0.49	0.85	0.62
B-PER	0.70	0.95	0.80
B-PROD	0.38	0.39	0.38
I-DRV	0.00	0.00	0.00
I-EVT	0.00	0.00	0.00
I-GPE_LOC	0.90	0.32	0.47
I-GPE_ORG	0.00	0.00	0.00
I-LOC	0.60	0.13	0.21
I-ORG	0.41	0.78	0.54
I-PER	0.85	0.96	0.90
I-PROD	0.50	0.43	0.46
O	1.00	0.97	0.99
micro avg			0.96
macro avg	0.44	0.47	0.42
weighted avg	0.97	0.96	0.96

Table 6: Basic NorBERT-FFNN model

	precision	recall	f1 score
B-DRV	0.58	0.79	0.67
B-EVT	0.54	0.76	0.63
B-GPE_LOC	0.91	0.92	0.92
B-GPE_ORG	0.68	0.90	0.77
B-LOC	0.76	0.86	0.80
B-ORG	0.88	0.92	0.90
B-PER	0.97	0.98	0.97
B-PROD	0.50	0.85	0.63
I-DRV	0.78	0.70	0.74
I-EVT	0.58	0.78	0.67
I-GPE_LOC	0.79	0.71	0.75
I-GPE_ORG	0.78	0.68	0.71
I-LOC	0.57	1.00	0.72
I-ORG	0.88	0.91	0.89
I-PER	0.98	1.00	0.99
I-PROD	0.62	0.88	0.73
O	1.00	0.99	1.00
micro avg			0.99
macro avg	0.75	0.87	0.80
weighted avg	0.99	0.99	0.99

Table 7: Fine-tuned NorBERT-FFNN model

2 Discussion

Regarding the architecture build on top of NorBERT, we see no great improvement between the model having one linear layer and the model having a FFNN of two linear layers with dropout of *p=0.1*. In both cases, the classes whose number of instances is the smallest (namely *B-EVT, I-DRV, I-EVT, I-GPE-ORG*) do not get predicted, and for the rest of the classes the improvement is minimal, with the overall micro avg f1 score remaining the same. We believe that more sophisticated architectures like an LSTM or a GRU would have yielded better results.

As far as regularization is concerned, in AdamW weight decay is calculated as *weight_decay (float, optional) – weight decay coefficient (default: 1e-2)* [1] and is used as a penalty for the loss function, for regularization purposes. We chose not to use weight decay for the bias terms and the layer normalization. The bias describes how well a model matches the training set, and layer normalization is used to calculate the normalization statistics through summing the inputs to the neurons of a hidden layer. There is a slight drop in performance doing this, which we could attribute to weight decay not being used for the layer normalization since bias terms are not usually regularized due to the fact that they are usually fixed.

[1] https://pytorch.org/docs/stable/optim.html

Regarding fine-tuning we can see a clear improvement in the case when all of the model's parameters were allowed to be fine-tuned to the task of NER, since the f1 score for each class got higher, and also the before mentioned classes who were not predicted at all, were predicted now. In both the basic linear NorBERT and the FFNN NorBERT the micro avg f1 score was the same, with the first having a higher f1 score for more classes than the latter. Again, the lack of difference in performance here could be attributed to the fact that our FFNN is essentially two linear layers after one another, which is not highly more complex than one linear layer.

Finally, by only fine-tuning the embedding layers of the model we see that there is a clear improvement, greater than the basic model's performance but slightly less so when compared to the full-fined tuned model's performance, which was to be expected.

3 Note

Things we did not attempt for Obligatory 3: More principled fine-tuning strategies, and the elective experiments on sequence tagging with NorELMo.