NLP Coursework

Alexandre Allani
aa4719@ic.ac.uk

Daniel Cahn
dbc19@ic.ac.uk

Guilherme Freire
gdf19@ic.ac.uk

All code is available on Github[1] and a sample notebook can be found on Google Colab[2]

1 Dataset analysis

The dataset used in this project consists of pairs of English sentences and corresponding machine translations into German. To begin, we take a look at the dataset, analyse the distribution of scores, and make some observations.

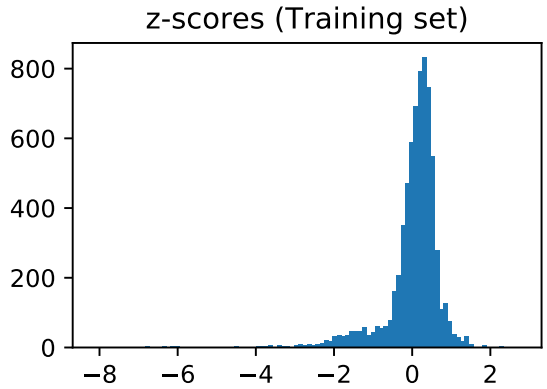

Figure 1: Histogram of scores in the training set.

We notice that the training set is skewed with a heavy left tail, the bulk of the dataset concentrated is around the zero, and are some severe outliers (mean: 0.004, median: 0.164867, std:0.83, IQR: 0.55). While the positive reviews at most reach a score of 2.76, low fidelity translations have incredibly negative scores, with a minimum at -8.14. ,

1.1 Mistranslations

To get a sense of what these outlier translations looked like, we looked at individual examples in the extremities of the distribution. We found that there were several examples where the translated text was identical to the original text, and we noticed that many of these pairs involved titles. The score variance for untranslated text is extremely high (std=2.39), with some untranslated titles receiving very negative scores and others high scores, while one would expect that all non-translations should receive similar scores, or at least all non-translated titles. We therefore attribute this behaviour to mis-labeling and treated these cases as noise, removing all exact-match pairs from the training set. Further, looking into the same translation cases by searching online, we found that some were the product of erroneous sentence segmentation, likely because the creation of this dataset did not properly handle HTML tags and Wikipedia article formatting, and included references.

1.2 Validation set

We found that the histogram of scores from the validation set is reasonably similar to that of the training set, which might indicate that it is a stratified sample. However, we found that the validation set does not represent the test set well. After training several models with different parameters, watching out for overfitting, we observed that training and validation sets achieved similar error metrics, while our test scores were disproportionately worse. To mitigate this, we repeatedly merge the training and validation sets and randomly split them again, combining metric estimations from each run in a way similar to K-Fold. We also expect that, due to the seemingly stratified nature of the validation set, it might contain valuable data points for training.

[1] https://github.com/noncuro/BERThoven

[2] https://colab.research.google.com/github/noncuro/BERThoven/blob/master/Sample%20Notebook%20using%20BERT.ipynb

2 Model

2.1 Bi-LSTM

Our first approach to solve this task was to build a relatively small model to get a baseline result, and therefore initially experiment with the use of RNNs. Since this is an NLP task, our dataset is made of strings, and thus which require pre-processing before applying any machine learning techniques. For our initial RNN experiment, we use a word-piece tokenizer and learn token embeddings. We start by passing the English vectorized sentence to a Bi-LSTM and store the outputs produced at every time step. Next, a unidirectional LSTM with attention receives the translated text as input on every time step and outputs a scalar, corresponding with the score. We consider the last output to be the network prediction and ignore all other values.

The idea is that the first bidirectional network would generate features representing each token from the source sentence and those would be used by the second network to generate a score. The second network uses the attention mechanism to select which source tokens are relevant to consider at a given time step. This would allow a simple token by token comparison between translation and source, keeping track of the score as the steps progress. The final score is the one that takes into account the entire sentence. We obtained RMSE of 0.843 on cross-validation using this model. From there, we experimented with slight modifications over this architecture, by changing the recurrent unit to GRU, trying unidirectional versions of the first network and different hyper-parameters to train. None of these showed significant improvement.

2.2 BERT

After experimenting with RNNs, we understood that this task requires a sophisticated language model; we cannot rely on only the dataset itself, with only 7000 examples, to learn the intricacies of two languages and the complexities of how words fit together to form sentences. We therefore fine-tune a pre-trained BERT, due to BERT's extensive pre-training on a vast and diverse corpus (1). Because of the multi-lingual nature of our dataset, and the use of titles and names throughout the dataset, we use the "bert-base-multilingual-cased" model from HuggingFace (2). We draw significant inspiration from (3) and (4) in our methodology for fine-tuning BERT.

Since we are using a pre-trained BERT, we encode our input sentences using the model's word-piece tokenizer and one-hot encoding of the word pieces. As for the content of our inputs, we concatenate source and translation sentences, making sure to set appropriate delimiters, as follows: `[CLS] SRC [SEP] MT [SEP]`. We then pad all encoded sentences to the same length, so that batched inputs will be of the same length. We additionally pass attention mask to indicate which tokens BERT should ignore in its attention layers. We pass these inputs through BERT, extract the output corresponding with the `[CLS]` tag and pass that to a fully connected layer with sigmoid activation. We compare this to the normalised score for the translation and backpropagate using our loss function.

During experimentation, we also tried several other, related, architectures. We tried using second forward pass with the opposite direction: `[CLS] MT [SEP] SRC`, and then either concatenating the outputs or summing them, before applying the fully connected layer. We found that both approaches led to worse or no better results and took far longer to train. We hypothesise that the lack of improvement is because the model benefits from being able to expect the first input to be a source sentence and the second to be a translation, rather than the other way around. It is also possible that concatenation or summing of two forward passes makes backpropagation more chaotic, making overall training harder.

Another method we experimented with involved using BERT only for sentence embeddings, and not fine-tuning the model. To do this, we simply freeze the model's BERT layers and only train the final fully connected neural network. Results were significantly worse using this method.

We also tried using the sum or mean of all BERT outputs as the input to the fully connected part, instead of only the output from `[CLS]` but once again found worse results. We believe this is because in pre-training, the `[CLS]` output is trained on the task of predicting if the sentence after `[SEP]` follows the former sentence, $P(\texttt{Next}|\texttt{Current})$. Given that this task is quite similar, $f(\texttt{MT}|\texttt{SRC})$, it's probably easier and quicker for BERT to learn the correct function using the `[CLS]` output.

3 Over-fitting

Using only the above model, we found that we were quickly and significantly over-fitting to our training dataset. We found that after even three epochs, our training accuracy would increase significantly ($R^2 \leq 0.5$), but with far lower validation accuracy ($R^2 \sim 0.2$). We therefore turn our attention to methods that reduce overfitting. In addition to the various methods discussed below, we experimented using a dropout layer before our final fully connected layer to no apparent improvement. We did find improvement from using weight decay, and ultimately incorporated a weight decay of 0.1 into the training of all models. We expect that weight decay forces the model to learn simpler, rather than more complex, functions.

3.1 Resampling

As a first strategy, we consider resampling the training set by upsampling certain datapoints during training. From Figure 1, we see that there are severe outliers with z-scores much higher or lower than the mean, while the majority of the z-scores lie close to the mean. We had two opposite hypotheses as to how to handle outliers to improve results.

First, we consider that outliers might serve as a hindrance to model's learning process. When using Mean Squared Error as the loss function, outliers that are badly labeled can carry very large gradients. These points might hijack the model's learning, reducing its ability to learn the true trend. To mitigate this effect, we downsample the outliers, which we define to be the points more than 2 times the interquartile range below the first quartile or above the third one.

Next, we experimented with the competing hypothesis: that outliers *define* the trend. While the elements near the mean have some information, the variance of their scores is fairly small. Moreover, we attribute some of this variance to noise, such as that from expert disagreement, as discussed in section 1.1. However, we can be more confident that very good translations receive high scores and very bad translations receive very low scores. Our model can therefore learn the general trend from the outliers, and apply that trend elsewhere. To achieve this, we took the the 1st quartile and last quartile and doubled their proportion inside the dataset. We found that this form of upsampling actually improved our results.

As highlighted in the data analysis part, sentences that are not translated can also be a problem in training the model. We expect that BERT is likely to map perfectly translated sentences very close to each other in its latent embedding space, and hence will likely map exact-match sentences close to each other as well. BERT's learning might therefore be negatively affected by these outliers, especially if these sentences have very low scores. Scores with very high magnitude could potentially saturate the gradients when passing through the sigmoid layer. By default, remove these sentences during training. Since they represent 0.81% of the dataset, it is unlikely to significantly bias the model. We also experimented with expanding the scope of these removals, defining similar translations in terms of Levenshtein distance. However, we found that increasing the distance threshold led to too many removals from an already-small dataset, and instead only remove the very small subset with identical translations.

3.2 Data Augmentation

Next, we experiment with augmenting our dataset to allow for an increased training time with less overfitting. Due to the NLP nature of the task, synthetic data augmentation appears to be hard. Substituting words in training examples with synonyms is unlikely to add significant noise, as BERT is likely to map these synonyms closely in latent space. Instead, we augment the dataset by replacing random words in the training set with `[MASK]`, as is done in pre-training.

We expect that BERT will attempt to predict the masked words and effectively replace them with the correct or good enough word given the context. This shouldn't significantly affect the quality of the translation, but makes the dataset harder to memorize. We also experimented with varied probabilities of random masking to control the degree of noise. This method may introduce bias into training, for instance in the case BERT "improves" a translation by substituting in a better fitting word. For that reason we always use the original dataset on the final training epoch.

Finally, we considered augmenting our dataset with organic data from another source, for example usi a public dataset of expert translations and assign them arbitrary high scores. This approach would introduce a considerable amount of bias, and should be used only during early stages of

training, if at all. We, similarly, considered using the Chinese to English dataset, and training BERT simultaneously on both tasks. Due to our use of a multi-lingual BERT model, we expected that this could serve as a form of data augmentation, improving the breadth of examples.

3.3 Loss function

We chose Smooth-L1 as our loss function. The intuition behind this choice is that we don't want to overfit to outliers, which may to be significantly misclassified. Applying L1 on those points equates to applying an equal gradient regardless of how badly these points are misclassified. The MSE part of Smooth L1, however, helps with reducing gradient instability that can occur when points are nearly, but not perfectly, correctly classified. It also helps simultaneously optimizing for Mean Absolute Error and Mean Squared Error.

3.4 Transformations on scores

We also consider transformation on the space of scores. As discussed above, we use a sigmoid activation at the end of our model to restrict its output range. We therefore scale all scores by subtracting the minimum and then dividing by the maximum. We also experimented with other ways of transforming the scores.

Due to various observations made above, we felt that the scores provided seemed to be more ordinal in nature than truly continuous. Translations rated as twice as high did not seem to be twice as good, and vice versa for scores in the negative range. We also felt it necessary to exaggerate score difference in the middle range, where most translations fell, in order to learn meaningful patterns. With that in mind, we experiment with converting the distribution of scores into a uniform distribution in the range [0,1] based on quantiles. This method reduces the model's tendency to output values very close to the mean, and helps balances batches. We found that using a quantile transformation to the uniform distribution improved Spearman correlation, but did not help with the metrics associated with the competition (MSE, MAE, Pearson). We consider the quantile transformation to be an improvement, but leave it aside due to it's inability to help us with the relevant metrics.

Model	Pearson r	MAE	RMSE
Best Peasron r	**0.1543**	0.4925	0.9378
Best MAE	0.0831	**0.4804**	0.9530
Best RMSE	0.0405	0.5123	**0.9407**

Table 1: Metrics of our best performing models on hidden test set.

4 Hyperparameters

In order to find the best combination of hyperparameters, we began with manual tuning. This helped us quickly eliminate bad combinations and find which parameters impacted training the most. After the initial tuning it became increasingly hard to find combinations by hand selection. To automate this procedure we built a script that would train and evaluate different models based on a generated list of potentially good candidates.

For our final, most successful, model we use smooth L1 for the loss, a cosine learning rate scheduler with 10% warm-up, and an initial learning rate of 2e-5. For two epochs, we train on an upsampled dataset with masking, and for a third epoch, we train on the original training set. We use weight decay of 0.1.

5 Conclusion

In the above sections, we discuss various experiments and variations on a basic formulation of fine-tuning BERT. Ultimately, we found that even our best model did not perform to our expectations on the hidden test set. When re-splitting the training and validation sets several times, we found significant variation in performance, so we expect that, despite our best efforts, our model might have bias towards our training data. We found that our best results on the hidden test set came from training several networks on different, random training/validation splits. We treat these models as one ensemble and have each predict scores on the test set, finally taking the mean of all scores for each sentence. We found that this approach yielded better results on the hidden test set than any individual experiment. Upon submission, we found that we performed in first place for MAE (0.4816), sixth for Pearson (0.1405) and achieved an RMSE of 0.9459. We expect that we performed far better in MAE than RMSE due to the non-linear nature of our predictions, especially due to our use of Smooth L1 as a loss function.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[2] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *ArXiv*, vol. abs/1910.03771, 2019.

[3] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune BERT for text classification?," *CoRR*, vol. abs/1905.05583, 2019.

[4] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.