Text Project

Wet Llamas
Arttu Pönni, Nikolai Denissoff, Tomi Sariola

January 6, 2019

Introduction

We are tasked with classification of news items from Reuters to corresponding topics. Each news item consists only of textual data and can correspond to none, one or multiple topics, so this is a multi-label classification problem. We solved this problem with a deep neural network architecture we call CRNN, a hybrid model of a convolutional and a recurrent network.

Data and preprocessing

Dataset consisted of 300k articles from Reuters from which each instance had a headline, a title, a body of text and a list of topic codes. For each instance we separated special characters as their own words and lower cased everything, for example "*(EC)*" was replaced with "*(ec)*". This was done because the former string is not present in the *GloVe*-vocabulary but the latter string is a combination of three strings, after splitting at white space, all of which are present in the vocabulary. This way we minimize the number of out-of-vocabulary words while retaining maximal information present in the input sequence. Lastly we concatenated the headline and text body together and used this as the input sequence for our model. Topic codes were encoded as binary vectors where ones and zeros represent the presence and absence of a topic code, respectively.

We separated the data to training, validation and test sets with 80/10/10 split. We also used pre-trained *GloVe*-embeddings with 400k words.

The training instances were assembled in batches, where each instance was padded to the same length. This length was determined either by the longest sequence in the batch or a set maximum sequence length, depending on which of these was shorter. This was done in order to speed up processing and to avoid large amounts of padding within batches, because some input sequences were much longer than others. Further, we speculated that the beginning of a news article should contain sufficient information for label prediction.

Model exploration

All models we tried had the same first and last processing steps. First, we split input sequences to a set of words, which were tokenized and represented as vectors by using *GloVe*. In the end, we always had a linear layer which produced vectors in $\mathbb{R}^L$. Lastly we applied a sigmoid function elementwise to constrain network output to $[0, 1]^L$ which we interpret as probabilities for the L different labels[1]. Even though we tried models with multiple different embedding dimensions, we held the embedding static for all models. All models were trained with the default ADAM optimizer. We used binary cross entropy as a loss function for quantifying the distance between our prediction vector and the multi-hot vector of true labels. During prediction, we consider post-sigmoid activations greater than 0.5 to indicate that the corresponding label should be assigned to the input sequence.

In our work we tried for three different different approaches: a CNN, a RNN (implemented with Gated Recurrent Units or GRUs) and a CRNN which ran a CNN and RNN parallel. For each of the approaches we optimized hyperparameters with a Python library called *hyperopt*. For the CNN we optimized stride, filter sizes, no of filters and batch normalization(true/false). For the RNN we opti-

[1] In our case, we have $L = 126$ different labels.

mized bidirectional RNN(true/false), RNN hidden size, number of RNN-layers. For the fully connected layers we optimized: bottleneck dimension, dropout percentage, embedding dimension and text length. In total we trained about 500 different models.

We quickly noticed that a large and deep model is required to tackle this challenge. Further, the dataset is large enough to justify training huge model. We utilized a NVIDIA Quadro P4000 GPU for our trainings which has 8GB of memory. The 8GB of memory quickly became a bottleneck and therefore we started using a batch size of only 64. We also noticed that more layers with less neurons seemed to give more bang for the buck than having a few layers with a lot of neurons.

Effects of parameters

Similarity between all of the best models was that they used batch normalization, bottleneck dimension was between $500 - 600$, maximum input sequence length between $700 - 800$ and embedding dimension was 300.

For the RNNs, bidirectional GRUs seemed to outperform onedirectional models. A minimum viable number of hidden layers in RNNs seemed to be at around 100.

Regarding CNNs, a stride of 1 outperformed larger strides. Multiple different filter sizes seemed to work well, a sweet spot around $2 - 4$.

As for the other parameters, we could not draw any conclusions. For example the three best models performed similarly despite having quite different structures. This can be verified by inspecting the training logs of our best performing models.

Results

Figure 1 shows training and validation losses of the three best models, each of them being CRNN. From the plot we see that each of them begins to overfit just after two epochs. The best model achieved with test set loss of 0.01583 and micro-averaged F1-score of 0.89. Even the ten best models with different structures had a test set loss at around 0.016.

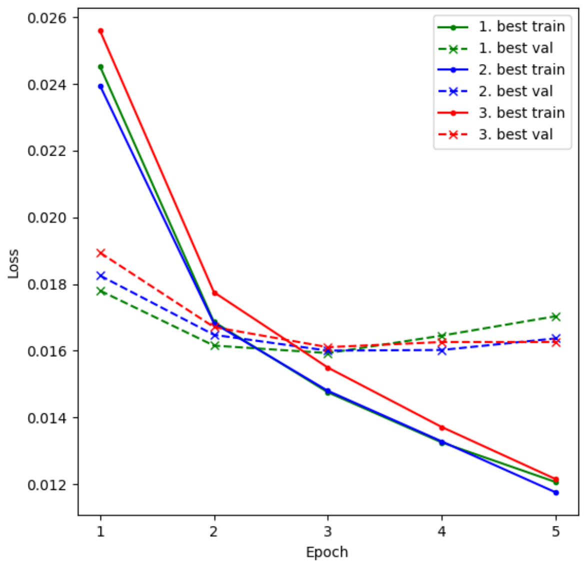

Figure 1: Training and validation losses of the three best models.

Final model

We achieved the best model by using both CNN and GRU architectures in parallel in order to leverage the strengths of each approach. That is, the word embedded input sentences were fed into both CNN and RNN, and then the final representations of each were concatenated and fed into a final fully connected layer to produce probabilities for the output classes. We call this architecture a *convolutional recurrent neural network (CRNN)*. With this CRNN we achieved a test set F1-score of 0.89. Test set precision for one label prediction was reliably over 97% and for three labels per input we still had over 84% correct predictions.

Discussion

Even though we found multiple models that performed well, there still are many potential ways for improvement. These include, for example, unfreezing the word embeddings, more stacked convolutional layers, different optimizers and learning rates/schedules and weight initialization strategies.

Also one could have tried more architectures. For example, in our CRNN model the CNN and RNN were working in parallel: both were given the same inputs and the results where concatenated before the fully connected layers. We did not explore giving the output of the CNN to the RNN. In addition we could have stacking convolutional layers for achieving deeper convolutions. Also we could have tried out something not based on convolutions or recurrence, for example, hierarchical attention networks or RNNs augmented with attention mechanisms.

Further, there was a strong change element in our hyperparameter optimization as they were stochastically chosen by *hyperopt*. Since our search space for optimal hyperparameters was so huge, we did not converge to a narrow set of optimal hyperparameters. This could have been mitigated with a more powerful GPU.

Deliverables

Click here to download the test set output for our best performing model. All code used for this project can be found from this Github repository.