Modelling Entailment with Neural Networks

Todor Davchev, University of Edinburgh 12/07/2016

1 The goal of the project

The aim of this project is to find a simple yet successful way for sentence classification and more specifically for recognising entailment relations. I compare two neural network approaches and namely RNN-based architectures as proposed by Rocktäschel et al. and convolutional neural networks.

2 Methodology and Results

Throughout the course of work I begun with a thorough study of the SNLI data set. This helped me learn that the dataset was comprised in such a way that it had more than one different hypothesis for every premise and that the dataset was large enough to enable neural network training over GPU rather than a CPU. This short study was followed by the actual implementation of the convolutional neural network I base this entire project on. That included the actual set up on the MSc students cluster which turned out not to be a trivial task. To ensure I was correctly implementing my work I used the sentence classification implementation[1] wirtten by Kim. Having set a baseline model which compraised of only one CNN followed by a feed-forward network I realised that running it against the SNLI dataset would not and did not achieve very good results - 66.55% (66.11% - 66.99%). Regardless, I used grid search to see if I can optimise the obtained results. I successfully did so to a certain level. For example, given the size of the dataset was substantially larger than SST-2 (the dataset used in Kim), reducing the dropout improved the results obtained by roughly 2% (See Figure 8). In addition, I decreased the runtime of each epoch by increasing the batch size from 50 to 200 as shown in Figure 9. Since I based my work on Kim's implementation, I am also using Adadelta. Thus, I didn't have to worry about decreasing the learning rate after increasing the batch size. Given the size of the dataset, this decreased the per-epoch runtime from 313ms to about 210ms (See Figure 9). Additionally, I used GloVe [Pennington et al., 2014] as opposed to word2vec [Mikolov et al., 2013] as embeddings. This meant implementing scripts for preprocessing both the SNLI text files and embedding the result using GloVe. Overall, the best result I achieved was 68.11% after completely removing the dropout (See Figure 8). In addition, I compared the performance between using a ReLU and Tanh as a non-linear function. As expected, the former performed better. For comparisons refer to Figure 8 and 9.

Next, I extended the model by implementing a Siamese network. I used a separate CNN for the premise and the hypothesis. However, the output of each of them was concatenated and used as an input to a third CNN which was then followed by a feed-forward neural network as per the initial model. All three CNNs were trained against the same cost function simultaneously. This substantially improved the accuracy with about 10%. I achieved 77.22% with the adopted parameters used in [Kim, 2014] and 77.99% using 0.2 dropout as shown in Figure 1. In this section onwards I am assuming a fixed dropout of 0.2 unless

[1] https://github.com/yoonkim/CNN$_sentence$

Dropout	non-linear function	Test	Train	Validation
0.5	relu	76.53%	84.92%	75.05%
0.2	**relu**	**77.99%**	**88.20%**	**76.99%**
0.2	tanh	69.7%	90.71%	77.40%

Figure 1: Alternating dropout and non-linear function. Siamese-like Model.

otherwise stated. I am also using a batch size of 200 as well as ReLU as a non-linear activation function and 0.95 as a value for the learning rate decay. Although there is a good chance this won't end up being the most optimal setting as already noted, it allows for achieving reasonably high results to the point where I find an optimal model which I will further optimise. To improve the accuracy I implemented different

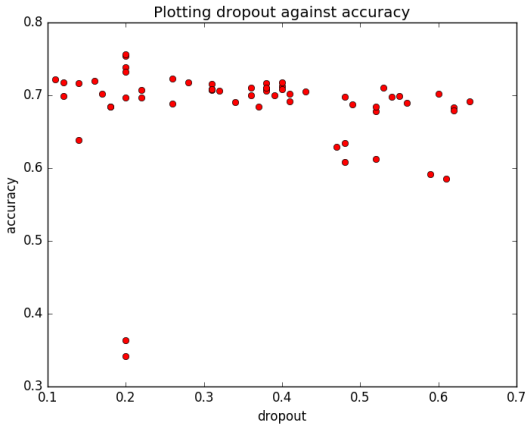

Figure 2: 2D relationship.

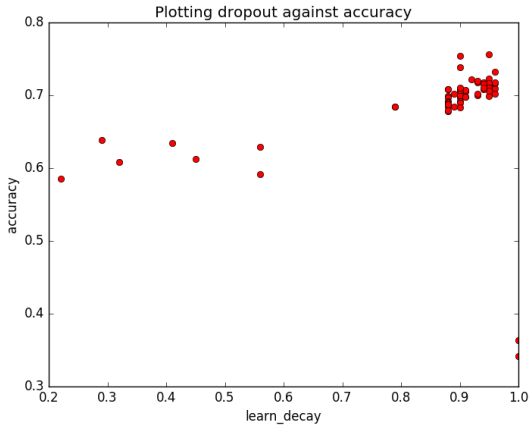

Figure 3: 2D relationship

linear operations which I used instead of concatenating the two outputs and later on in a combination with the concatenated outputs as well. I implemented the following additive and multiplicative models: addition, weighted addition, subtraction, weighted subtraction, multiplication and circular convolution. Then, I started optimising the hyperparameters associated with the models. To reduce the time required for hyperparameter search I used a worker which continuously sampled random hyperparameters and performed the optimization on a smaller network comprising of 5 epoch training and smaller batch size. Although the smaller batch size increases the time per epoch it reduces the resources required for a network training which itself reduced the number of failed jobs on the MSc cluster. During the training, the worker kept track of the training and validation performance, the per-epoch time and wrote them down to a file as advised in Stanford University's cs231n [2]. The worker was sampling the most common hyperparameters in the context of Neural Networks (namely learning rate decay and dropout). It also took into account the relatively less sensitive hyperparameters, for example in weighted addition learning methods, the setting of assigning weights for each of the two sentences. The search was performed in hyperparameter ranges as displayed in Figure 2 and 3. I used 100 separate trainings for multiplication and addition. The two plots show the results obtained from the former model. Moreover, the two figures show the dependencies between dropout and accuracy as well as between learn decay and accuracy. Note that in Figure 3 most

[2] http://cs231n.github.io/neural-networks-3/

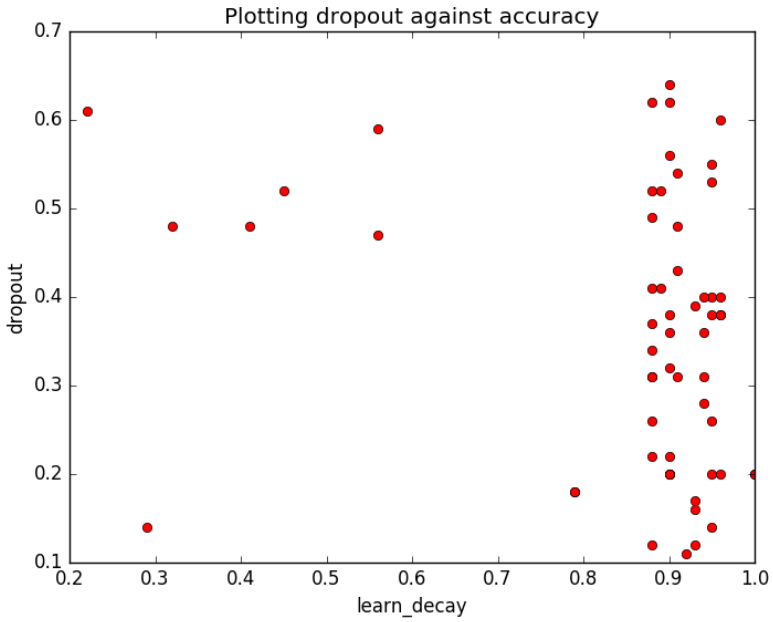

Figure 4: 2D relationship

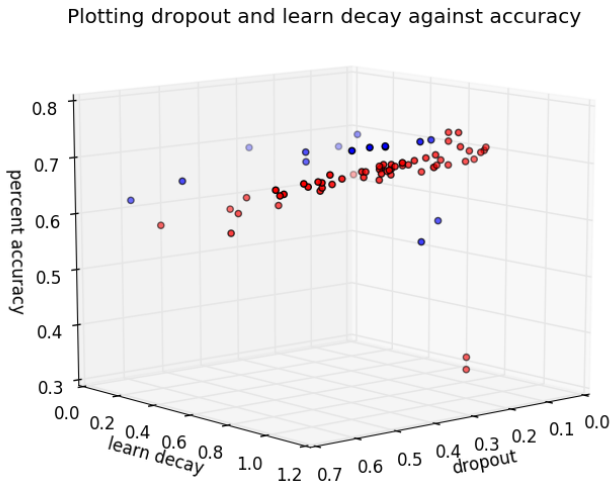

Figure 5: 3D Plot of the realtionship between dropot, learning rate decay and accuracy.

sampled points are between 0.88 and 0.96 as those seem to achieve the higher accuracy. Figure 2, however shows a rather uniform sample of dropouts. It can be noticed that the lower dropout value yields higher performance. However, both dropout and learn decay were correlated, as shown on Figure 4. One can observe that the majority of the samples were taken for dropout which is smaller than 0.4 and for learning rate decay which is higher than 0.88. Figure 5 shows the relationship between the learning rate decay, dropout and the accuracy achieved. The red dots indicate the Siamese-like model ran using multiplication between the outputs of the first two CNNs and the blue dots indicate the same however using addition instead of multiplication. It can be noticed that addition performs slightly better than multiplication in all cases. An interesting observation at that moment was that neither basic addition or multiplication could achieve results that differ by much from the basic concatenation (See Figure 6). Applying the same to a non-static model, however, yielded slightly better results - for example the model which uses subtraction results in 79.70% accuracy as opposed to 79.34%.

The circular convolution, however was achieving around 34% accuracy which could potentially mean it is not working. I am however still unsure of that and this is yet to be studied. It is also important to bare in mind that tests performed on models that use circular convolution had to be split to minibatches as opposed to the others that use the full batch at a time. It is also important to note that I used Zhang and

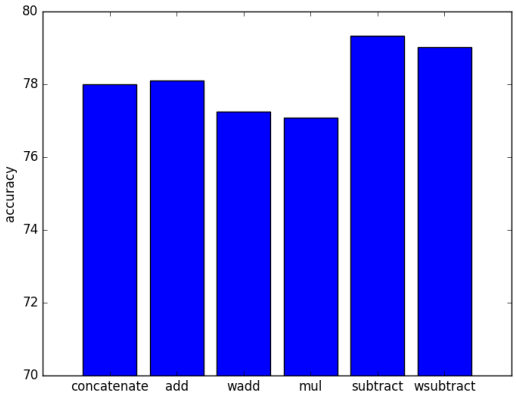

Figure 6: Achieved results for different operators.

Batch Size	Time per epoch
50	313ms
100	260ms
150	239ms
200	**210ms**

Figure 7: Runtime per epoch with respect to batch size.

Dropout	Test	Train	Validation
0.5	66.55%	69.24%	66.09%
0.2	67.32%	72.69%	66.91%
0.1	66.55%	73.71%	65.92%
0.0	**68.13%**	**75.25%**	**67.57%**
0.7	64.64%	66.44%	64.62%

Figure 8: Alternating dropout.

Dropout	Test	Train	Validation
0.5	63.76%	70.96%	68.05%
0.2	58.48%	74.00%	66.91%
0.1	61.96%	79.99%	67.56%
0.0	62.04%	75.86%	67.44%
0.7	**64.60%**	**70.84%**	**67.35%**

Figure 9: Tanh as a non-linear function.

Wallace [2015] as a guideline while fine-tuning the variety of hyperparameters.

The results achieved so far can be considered as competitive to the most simple RNN-based models used

Mix	Test	Train	Validation
1	76.53%	84.92%	75.05%
2	78.07%	90.93%	79.22%
3	**79.78%**	**90.48%**	**78.86%**

Figure 10: Alternating dropout and non-linear function. Siamese-like Model.

against the SNLI dataset. However, they were still slightly bit lower than those described in Rocktäschel et al. and far from being close to the currently known human error of 89%. Thus, I decided to increase the training parameters. I did this by introducing a list of transformed version of the outputs of the first two CNNs. Thus, introducing a new, larger input for the third CNN during training. I used different mixtures between the seven operations used so far. The results obtained for those are still in the process of being generated but there are a few promising outcomes. Currently the mixtures are split as follows:

1. Mix 1: Comprised of the concatenated inputs, weighted subtraction of the two and weighted addition;
2. Mix 2: Comprised of the concatenated inputs, multiplication between the two and weighted addition;
3. Mix 3: Comprised of the concatenated inputs, subtraction of the two and weighted addition;
4. Mix 4: Comprised of the concatenated inputs, subtraction of the two and a circular convolution;

The results from the mixed models seem to perform a bit better than the previously introduced ones. After running the same tests on a non-static version of mix1 however using word2vec instead of GloVe, non-weighted addition and a dropout of 0.15 I achieved **80.05%**. This however could further improve if I used lower dropout (which so far seems to be able to achieve the best results), smaller learn decay and use a different activation function as opposed to the currently used Iden, chosen by Kim. Furthermore, this result could be improved by increasing the number of feature maps or the size and the number of filters.

3 To do

In addition to the work described above, I would like to

1. Step 1: Gather a few more tests for the different mixtures and observe how some of the not as extensively studied so far parameters affect the network.;
2. Step 2: Increase the number of filters and their sizes and observe how this will affect the accuracy; Increase the number of parameters;
3. Step 3: Fix the existing reshape error related with the former part of 2 and further investigate the implementation of circular convolution and the associated testing of the models steps; Make sure they are working properly;
4. Step 4: Write the report associated with my project and properly document my implementation;

References

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–43, 2014.

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, and Phil Blunsom. Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664*, 2015.

Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.