CSE143 Assignment2

Kit Lao

February 19^{th}, 2020

Text Classification with RNN

Epochs

Accuracy	10	5	20
Training	91.86%	89.96%	94.81%
Validation	89.47%	87.65%	91.22%

Epochs is number of times the models pass through the entire data set. The initial value was 10 epochs, and I decided to try values that were either half or double the initial value, because it was a common thing to do in practice. Both validation and training accuracy did better as the number of epochs increased, as a result from the model being able to fit the data better.

Training batch size

Accuracy	32	64	16
Training	85.53%	89.38%	99.18%
Validation	83.17%	79.08%	91.36%

The batch size is amount of instances used per gradient update. The initial value was 781, and I decided to use numbers that were either half or double the initial value. As the batch size increases, the training and validation accuracy increased as well, which shows that the model is learning a lot better when updating using a larger batch at once.

Choice of non-linearity

Accuracy	tanh	sigmoid	relu
Training	94.94	94.82	98.17
Validation	85.92	90.18	92.95

The tanh activation function is monotonic, non-linear, and differentiable. Because each derivative gets steeper, tanh will cause vanishing gradients. Tanh ensures that the output will be between -1 and 1. Larger more positive inputs will have a slight increase in output, and vice versa for most negative inputs.

The advantages of tanh is that it is more efficient due to its wide range of ouputs, which results in faster training.
Sigmoid is a non-linear, monotonic and differentiable activation function. The output values are between 0 and 1, where there are small changes in output for large or small inputs, which is an advantage for being a good classifier. Since the rate of change gets smaller for larger or smaller inputs, this results in vanishing gradients.
Out of the 3 different activation functions, relu is the fastest because there is less calculation load due to the 0 value region. Relu also doesn't have the vanishing gradient problems because the derivative of it is either 0 or 1, so the gradients cannot vanish. The problem with relu is that negative inputs get mapped to 0, which results to the data not fitting properly.

Choice of optimization method

Accuracy	adam	sgd	rmsprop
Training	96.00%	74.76%	99.13%
Validation	84.96%	69.52%	84.40%

The initial optimizer adam is computationally efficient, well suited for large amounts of data or hyper-parameter, and can handle noise well. It uses a combination of stochastic gradient descent and RMSprop, which gives it the advantage of being very flexible while being able to do well. Because it is generally used to large amounts of data, adam does not outperform RMSprop, due to the small training size.
Stochastic Gradient Descent (SGD) is good for shallow networks, which describes the model.
The accuracy on training and validation was very low because SGD is underfitting on the small training size.
RMSprop is similar to SGD but with momentum. The optimizer restricts the oscillation in the verticle direction, therefore, allowing us to increase the learning rate which results in taking larger steps into the horizontal direction to converge faster.

Dropout rate

Accuracy	0.0	0.2	0.4
Training	96.38%	95.62%	92.99%
Validation	88.53%	83.42%	83.38%

The dropout rate is the percentage of neurons getting ignored for each layer. The idea is that some neurons are being dropped out during training to force other neurons to step in and handle the representation required to make predictions for missing neurons. The effect of this is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data. The default value is 0.0. Due to how simple the model is and how little instances there are in the training data, increasing the dropout rate affected the training and validation accuracy.

The best performing model was the one that uses relu as it's non-linearity because it had the highest validation accuracy. The test accuracy for it was 66.82%.

Recurrent Units

With LSTM, the vanishing gradients isn't a problem because only a certain amount of previous inputs are remembered. This makes training much easier, resulting in higher training accuracy. In all of the results, the training, validation, and testing accuracy's are all higher except for the SGD optimizer, mainly because it is over fitting on an even more complex model. The experimental procedure is same as part 1.

Accuracy	10	5	20
Training	99.01%	96.00%	99.80%
Validation	93.42%	90.38%	95.44%
Testing	71.41%	74.23%	72.01%

Training batch size

Accuracy	32	16	64
Training	99.11%	96.67%	99.86%
Validation	93.97%	89.10%	95.20%
Testing	71.84%	73.66%	70.78%

Choice of non-linearity

Accuracy	tanh	sigmoid	relu
Training	82.93%	88.29%	96.70%
Validation	81.78%	88.17%	81.50%
Testing	65.13%	67.09%	52.32%

Choice of optimization method

Accuracy	adam	sgd	rmsprop
Training	98.83%	53.87%	98.70%
Validation	92.70%	50.35%	93.70%
Testing	70.94%	50.55%	69.49%

Dropout rate

Accuracy	0.0	0.2	0.4
Training	99.05%	98.64%	98.64%
Validation	92.29%	92.67%	93.54%
Testing	70.15%	70.42%	71.30%

Pretrained Word Embeddings

Accuracy	10	5	20
Training	85.75%	73.51%	98.05%
Validation	79.26%	73.08%	79.48%
Testing	78.49%	73.13%	79.37%

Training batch size

Accuracy	250	125	500
Training	85.75%	81.38%	94.21%
Validation	79.26%	78.48%	81.60%
Testing	78.49%	77.93%	81.77%

Choice of non-linearity

Accuracy	tanh	sigmoid	relu
Training	85.75%	73.37%	50.23%
Validation	79.26%	73.56%	49.08%
Testing	78.49%	73.47%	50.00%

Choice of optimization method

Accuracy	adam	sgd	rmsprop
Training	85.75%	56.38%	80.02%
Validation	79.26%	56.40%	73.74%
Testing	78.49%	56.12%	73.38%

Dropout rate

Accuracy	0.0	0.2	0.4
Training	85.75%	98.64%	98.64%
Validation	79.26%	92.67%	93.54%
Testing	78.49%	70.42%	71.30%

1. The embeddings for this part are from the glove 100 dimension data set. This dataset contains a 100 dimension vector representation for each word, and is used as the weights for the embedding layer. As for part 1 and 2, the embeddings were learned throughout the network, while in part 3, the embeddings came from a dataset.

2. Using pretrained embeddings, the model has a harder time learning during train time. Most of the training accurcys are below 90%, but on the other hand, the testing accuracy's shows an improvement, compared to learning the embeddings with other networks. The model is able to achieve the same training accuracy as part 1 and 2 when dropout is increased and epochs are increased, but their testing accuracy is still low, which may be the result of overfitting on such a small sample size. Overall, this model is terrible because it is training on a very small sample size, and uses very simple networks.

3. The way word is represented as a word embedding is as a vector. When words have opposite meaning semantically, then their vector representation should point to different directions, and vice versa. For the embeddings to be similar, the vectors should point as away from one another as possible. This explains why antonyms have similar word embeddings. The cosine similarities for 10 pairs of antonyms are the distance between the vector representation of the words, in a vector space.

Antonym	Cosine Similarity
happy, sad	0.6801
good, bad	0.7703
inferior, superior	0.7126
agree, disagree	0.7048
boy, girl	0.9176
true, false	0.5501
push, pull	0.7453
offense, defense	0.5031
wife, husband	0.9219
sit, stand	0.7370

4. The sentences I used was "this film was the greatest film i have ever seen", which was s_1 and "this film was the worst film i have ever seen", which was s_2. The model I used was the standard model for part 3. The sentence that is supposed to how a lower sentiment score ended up having a higher sentiment score because the model for part 3 was terrible.

Sentence	Sentiment
s_1	0.5809
s_2	0.6277