Natural Language Understanding with Distributed Representations - Assignment 1

Matthew Dunn (mtd368) 9/28/16

Activation

RELU v. Tanh Non-Linearity

To better understand how different activation functions impact the performance of a single layer Convolutional Neural Network (CNN), the initial non-linear Hyperbolic Tangent (Tanh) activation function was replaced with the Rectified Linear Unit (RELU) activation function. As shown in Table 1, there was a measureable improvement in the model's performance when the RELU was implemented without dropout.

		Accuracy	
Activation Function	Dropout?	Average Validation	Test
Tanh	No	~0.735	~0.723
	Yes	~0.730	~0.725
RELU	No	**~0.763**	**~0.735**
	Yes	~0.735	~0.720

Table 1: Results of experiments with Tanh and RELU activation functions in relation to dropout.

Initialization

To understand if different embeddings impact the performance of the model, the embedding dimensions were changed from 16, 64, to 128. Interesting, it didn't impact the performance of the model dramatically. Another area of exploration, would be to use word2vec or some other embedding technique to see if there are more accurate representations.

Learning Rate

An issue identified early on was that the loss increased even as our accuracy increased during training and validation. After some research, I came across a resource explaining that as training progresses and we execute gradient descent, we need to adjust our learning rate to become smaller. The learning rate needs to be adjusted to become smaller as learning progresses to mitigate the possibility that a gradient step taken later in the training process doesn't step across a local minimum, or bounce back and forth across a convex hull because the step size is too large. To account for this issue, the starter_learning_rate was initialized at 1e-2, passed as a parameter into the Tensor Flow (tf) exponential_decay functional, which was set to

*decayed_learning_rate = learning_rate * decay_rate ^ (global_step / decay_steps)*

And, was initialized at:

*decayed_learning_rate = 1e-2 * 0.96 ^ (0/200)*

Regularization

Dropout

To help reduce overfitting of the model, Dropout was implemented with a 0.5 probability during training and 1.0 during validation and testing. By implementing this technique, the likelihood of neurons learning off one another was reduced and this was confirmed by the performance stability observed between the accuracy achieved during validation compared to that found during test evaluation.

Train, Validation, Test Split

In hopes of improving the model's performance and validate the employed techniques further, I choose to setup Train, Validation and Test data sets, with 8635, 960 and 1067 respectively. This allowed me to implement early stopping, because I was able to train the model on the Train data set, determine whether or not performance was deteriorating using the Validation data set, and once performance deteriorated over 600 steps, checked every 200 steps, I evaluated the model against the Test data. By evaluating the model's performance not simply against Train and Test data sets, I was able determine how well the model generalized after training and validation against a data set it previously hadn't been exposed to.

Early Stopping

As described above, I implemented early stopping by evaluating the performance of a train model every 200 steps against the Validation data set to determine if the performance was improving, not changing, or deteriorating. When the model's performance on the Validation data set deteriorated by more than 0.01 for more than 600 steps, i.e., three rounds of validation, I stopped training and evaluated the model's performance against the Test data set. This approach allowed me to save numerous hours evaluating various neural net architectures and hyper-parameter settings, because often the model's performance plateaued or deteriorated well short of the ~15k training steps it would take without early stopping.

L2 Regularization

One issue that was partially address by adjusting the learning rate as discussed above was the model's validation loss. When the code was initially run without L2 regularization, but with a decaying learning rate, the validation rate would continue to increase from ~0.5 to somewhere around ~5.0. After implementing L2 regularization, the validation loss stayed well below 1.0 throughout training once the l2_reg_lambda value was set greater than 2.0.

Depth

When a second layer was added to the architecture, we saw the training accuracy quickly go to almost 1.0 after only ~600 steps, while the validation accuracy remained plateaued around ~0.70 after about ~500 steps. This seems to indicate that the additional layer resulted in the CNN learning the relatively small Train data set almost perfectly and consequently overfitting. Nonetheless, it seems the Dropout and L2 regularization forced the model to generalize somewhat, as the accuracy on the Test data set was ~0.69, indicating even though our model over fit the Train data set perfectly, the regularization parameters mitigated this issue as much as possible.

After implementing a second CNN, I played with changing the activation function from a RELU back to a Tanh. I saw no noticeable change in the model's performance, which seems to indicate the overall model architecture, i.e., number of layers, embedding of word vector, etc., have more impact on a model's performance than an activation function. Further, it seems an individual parameter, i.e.,

Summary

When we look at what architectures seemed to work best in predicting the sentiment of a sentence in this data set, it is clear that a single layer CNN outperforms a CNN with two layers. This seems to be caused by the fact that the CNN with two layers over fits during training and as a result doesn't generalize well to the Test data set because the accuracy dropped