Semantic Textual Similarity - SemEval 2017 Task 4
LSTM with Attention Mechanism Implementation

2018 OpenAI Scholars Project
Dolapo A. Martins | dolapo.martins@gmail.com

Abstract
Research and innovation within Artificial Intelligence (AI) and related (sub-)fields is on the rise. Natural language processing (NLP), in particular, is seeing an immense amount of attention. Humans, especially bi- and multi-lingual persons, are able to detect how similar two text snippets are, whether they are from the same language or not. It will be especially useful for machines to do the same; with innovations in computational speed, machines would be able to identify related texts much faster than a human. There are many different approaches to this program. This particular approach is to pair an LSTM with an attention mechanism, using ReLus as activation. This was written using the tensorflow architecture.

Discussion

At its core, the SemEval task is a multiclass classification problem built atop a simple binary classification task: are these two texts related? Answer either 'yes' or 'no' and the immediate question is, how (un-)related are they? Quantifying relatedness requires some scale. The goal is to train a model such that the machine outputs the 'relatedness' value that a human scorer would. Thus, sentences from different languages, with varying degrees of relatedness, are needed.

What then qualifies as "good" data? What qualifies as a correct similarity judgment score? The latter problem is simpler; ask many different speakers of the two languages to judge how related in content they are. Of course there is an inherent propensity for bias - the task requires a judgment call. By asking a diverse and varied group of scorers, the bias judgment of a scorer can be mitigated, along with discarding scorers who either incorrectly rate explicit positive or negative pairs incorrectly or whose scores, compared to other scorers, are abnormal.

The former requires more nuance. For a binary classification, it suffices for non-literal translations to form "negative" pairs, with exact translations forming "positive" pairs.[1]

[1]A full discussion of translations requires more nuance than presented in this paper; for the sake of brevity, this will hopefully suffice.

Most importantly, the positive data should be parallel. Negative data can be generated by shuffling the data, creating unaligned data. Once the binary classification task has been solved, the logic can be expanded to the multi-class task. The SemEval sample evaluation data has inputs with labels ranging from 1 (unrelated) to 5 (parallel). It is important that the data reflect the full spectrum of data - from nonsense, unrelated examples to perfectly aligned data. Otherwise, the model runs the risk of being biased towards predictions on one end or the other.

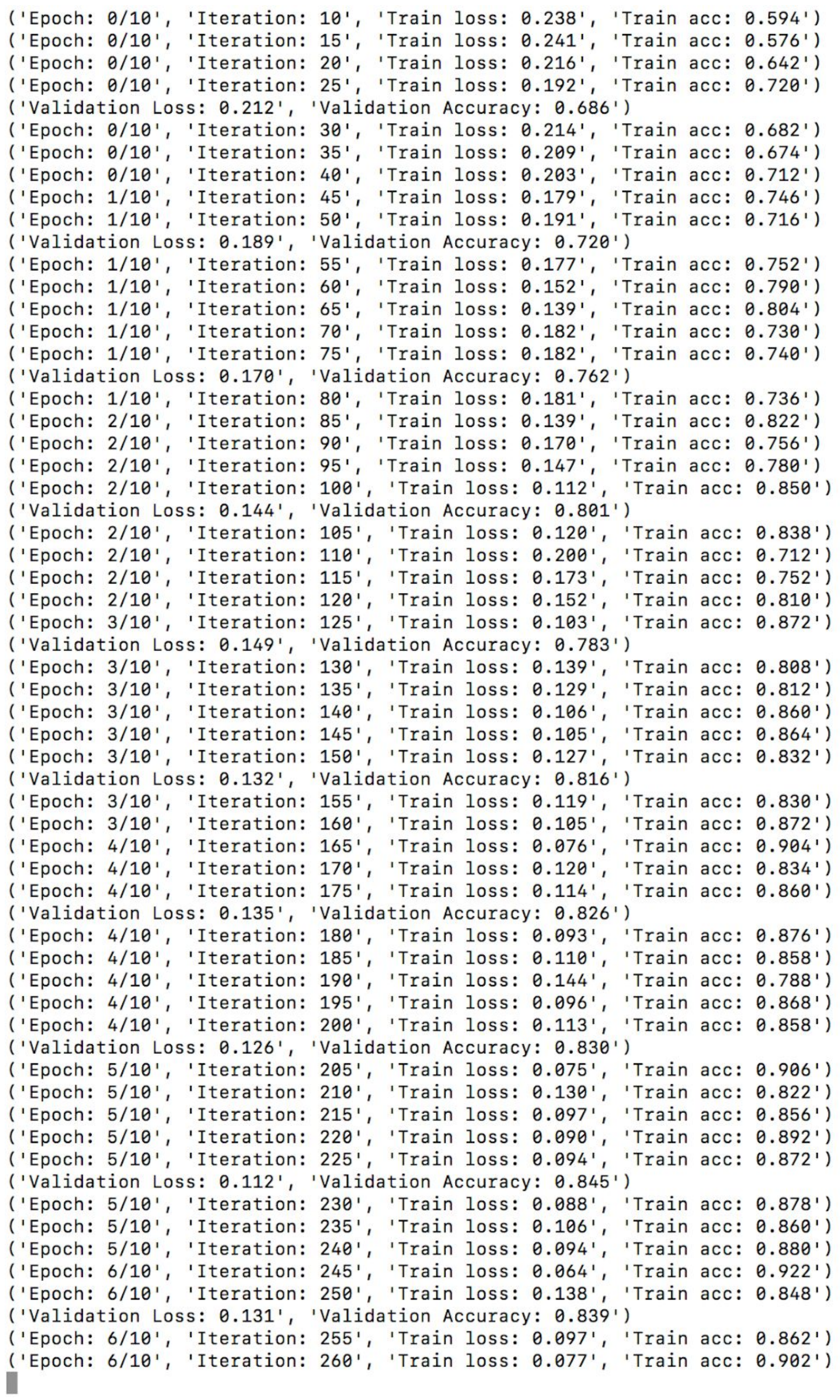
Method

Prior to approaching the SemEval task, learning the foundations of different NLP models and then building up to a fuller understand of more robust architectures like LSTMs and Google's Transformer's architecture. The initial goal was to use Google's transformer's framework, along with RNNs and LSTMs. However, that framework proved harder to integrate with the model. Instead of using the encoder and decoder, the goals was to use the learned embeddings from the LSTMs as input into the decoder for the transformer.

First, a bag of words model was implemented for email spam classification to help learn about binary classification. This was then 'upgraded' to using a neural network for english movie review classification. This helped learn about classification with one language. Next, the classification task itself was approached but using binary labels. First, 3 layers of Tensorflow's BasicLSTMCell were used; then the architecture was improved to use LSTMBlockCell and AttentionCellWrapper. Finally, the multiclass task was broached. Logits were produced from the output of the RNN using LSTM cells; the argmax of the logits were then used to produce the predictions. The logits and labels were then fed into the cost function; an AdamOptimizer was used minimize the cost.

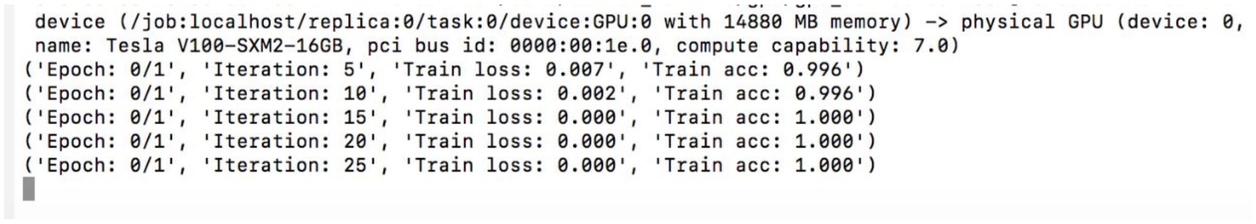
Results

Below are results on the movie review data.



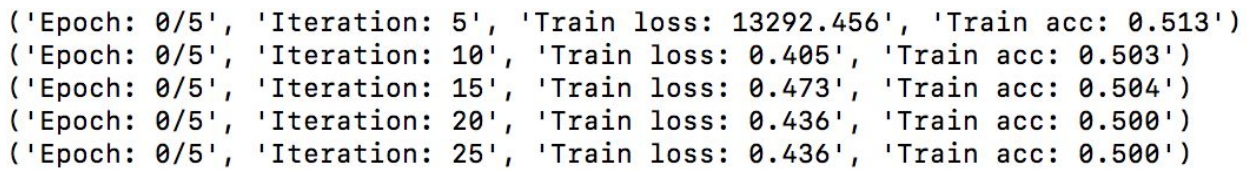
```
('Epoch: 0/10', 'Iteration: 10', 'Train loss: 0.238', 'Train acc: 0.594')
('Epoch: 0/10', 'Iteration: 15', 'Train loss: 0.241', 'Train acc: 0.576')
('Epoch: 0/10', 'Iteration: 20', 'Train loss: 0.216', 'Train acc: 0.642')
('Epoch: 0/10', 'Iteration: 25', 'Train loss: 0.192', 'Train acc: 0.720')
('Validation Loss: 0.212', 'Validation Accuracy: 0.686')
('Epoch: 0/10', 'Iteration: 30', 'Train loss: 0.214', 'Train acc: 0.682')
('Epoch: 0/10', 'Iteration: 35', 'Train loss: 0.209', 'Train acc: 0.674')
('Epoch: 0/10', 'Iteration: 40', 'Train loss: 0.203', 'Train acc: 0.712')
('Epoch: 1/10', 'Iteration: 45', 'Train loss: 0.179', 'Train acc: 0.746')
('Epoch: 1/10', 'Iteration: 50', 'Train loss: 0.191', 'Train acc: 0.716')
('Validation Loss: 0.189', 'Validation Accuracy: 0.720')
('Epoch: 1/10', 'Iteration: 55', 'Train loss: 0.177', 'Train acc: 0.752')
('Epoch: 1/10', 'Iteration: 60', 'Train loss: 0.152', 'Train acc: 0.790')
('Epoch: 1/10', 'Iteration: 65', 'Train loss: 0.139', 'Train acc: 0.804')
('Epoch: 1/10', 'Iteration: 70', 'Train loss: 0.182', 'Train acc: 0.730')
('Epoch: 1/10', 'Iteration: 75', 'Train loss: 0.182', 'Train acc: 0.740')
('Validation Loss: 0.170', 'Validation Accuracy: 0.762')
('Epoch: 1/10', 'Iteration: 80', 'Train loss: 0.181', 'Train acc: 0.736')
('Epoch: 2/10', 'Iteration: 85', 'Train loss: 0.139', 'Train acc: 0.822')
('Epoch: 2/10', 'Iteration: 90', 'Train loss: 0.170', 'Train acc: 0.756')
('Epoch: 2/10', 'Iteration: 95', 'Train loss: 0.147', 'Train acc: 0.780')
('Epoch: 2/10', 'Iteration: 100', 'Train loss: 0.112', 'Train acc: 0.850')
('Validation Loss: 0.144', 'Validation Accuracy: 0.801')
('Epoch: 2/10', 'Iteration: 105', 'Train loss: 0.120', 'Train acc: 0.838')
('Epoch: 2/10', 'Iteration: 110', 'Train loss: 0.200', 'Train acc: 0.712')
('Epoch: 2/10', 'Iteration: 115', 'Train loss: 0.173', 'Train acc: 0.752')
('Epoch: 2/10', 'Iteration: 120', 'Train loss: 0.152', 'Train acc: 0.810')
('Epoch: 3/10', 'Iteration: 125', 'Train loss: 0.103', 'Train acc: 0.872')
('Validation Loss: 0.149', 'Validation Accuracy: 0.783')
('Epoch: 3/10', 'Iteration: 130', 'Train loss: 0.139', 'Train acc: 0.808')
('Epoch: 3/10', 'Iteration: 135', 'Train loss: 0.129', 'Train acc: 0.812')
('Epoch: 3/10', 'Iteration: 140', 'Train loss: 0.106', 'Train acc: 0.860')
('Epoch: 3/10', 'Iteration: 145', 'Train loss: 0.105', 'Train acc: 0.864')
('Epoch: 3/10', 'Iteration: 150', 'Train loss: 0.127', 'Train acc: 0.832')
('Validation Loss: 0.132', 'Validation Accuracy: 0.816')
('Epoch: 3/10', 'Iteration: 155', 'Train loss: 0.119', 'Train acc: 0.830')
('Epoch: 3/10', 'Iteration: 160', 'Train loss: 0.105', 'Train acc: 0.872')
('Epoch: 4/10', 'Iteration: 165', 'Train loss: 0.076', 'Train acc: 0.904')
('Epoch: 4/10', 'Iteration: 170', 'Train loss: 0.120', 'Train acc: 0.834')
('Epoch: 4/10', 'Iteration: 175', 'Train loss: 0.114', 'Train acc: 0.860')
('Validation Loss: 0.135', 'Validation Accuracy: 0.826')
('Epoch: 4/10', 'Iteration: 180', 'Train loss: 0.093', 'Train acc: 0.876')
('Epoch: 4/10', 'Iteration: 185', 'Train loss: 0.110', 'Train acc: 0.858')
('Epoch: 4/10', 'Iteration: 190', 'Train loss: 0.144', 'Train acc: 0.788')
('Epoch: 4/10', 'Iteration: 195', 'Train loss: 0.096', 'Train acc: 0.868')
('Epoch: 4/10', 'Iteration: 200', 'Train loss: 0.113', 'Train acc: 0.858')
('Validation Loss: 0.126', 'Validation Accuracy: 0.830')
('Epoch: 5/10', 'Iteration: 205', 'Train loss: 0.075', 'Train acc: 0.906')
('Epoch: 5/10', 'Iteration: 210', 'Train loss: 0.130', 'Train acc: 0.822')
('Epoch: 5/10', 'Iteration: 215', 'Train loss: 0.097', 'Train acc: 0.856')
('Epoch: 5/10', 'Iteration: 220', 'Train loss: 0.090', 'Train acc: 0.892')
('Epoch: 5/10', 'Iteration: 225', 'Train loss: 0.094', 'Train acc: 0.872')
('Validation Loss: 0.112', 'Validation Accuracy: 0.845')
('Epoch: 5/10', 'Iteration: 230', 'Train loss: 0.088', 'Train acc: 0.878')
('Epoch: 5/10', 'Iteration: 235', 'Train loss: 0.106', 'Train acc: 0.860')
('Epoch: 5/10', 'Iteration: 240', 'Train loss: 0.094', 'Train acc: 0.880')
('Epoch: 6/10', 'Iteration: 245', 'Train loss: 0.064', 'Train acc: 0.922')
('Epoch: 6/10', 'Iteration: 250', 'Train loss: 0.138', 'Train acc: 0.848')
('Validation Loss: 0.131', 'Validation Accuracy: 0.839')
('Epoch: 6/10', 'Iteration: 255', 'Train loss: 0.097', 'Train acc: 0.862')
('Epoch: 6/10', 'Iteration: 260', 'Train loss: 0.077', 'Train acc: 0.902')
```

Initially, the binary classification language initially achieved 100% accuracy within 5 iterations on binary parallel data found online (EuroParl).



```
 device (/job:localhost/replica:0/task:0/device:GPU:0 with 14880 MB memory) -> physical GPU (device: 0,
 name: Tesla V100-SXM2-16GB, pci bus id: 0000:00:1e.0, compute capability: 7.0)
('Epoch: 0/1', 'Iteration: 5', 'Train loss: 0.007', 'Train acc: 0.996')
('Epoch: 0/1', 'Iteration: 10', 'Train loss: 0.002', 'Train acc: 0.996')
('Epoch: 0/1', 'Iteration: 15', 'Train loss: 0.000', 'Train acc: 1.000')
('Epoch: 0/1', 'Iteration: 20', 'Train loss: 0.000', 'Train acc: 1.000')
('Epoch: 0/1', 'Iteration: 25', 'Train loss: 0.000', 'Train acc: 1.000')
```

However, this was discovered to be a bug in the generation of the labels. The labels were all defaulting to zero, resulting in the model learning to predict zero very quickly. After the architecture was updated, the model achieved 50% accuracy on the multiclass STS data.



```
('Epoch: 0/5', 'Iteration: 5', 'Train loss: 13292.456', 'Train acc: 0.513')
('Epoch: 0/5', 'Iteration: 10', 'Train loss: 0.405', 'Train acc: 0.503')
('Epoch: 0/5', 'Iteration: 15', 'Train loss: 0.473', 'Train acc: 0.504')
('Epoch: 0/5', 'Iteration: 20', 'Train loss: 0.436', 'Train acc: 0.500')
('Epoch: 0/5', 'Iteration: 25', 'Train loss: 0.436', 'Train acc: 0.500')
```