Neural Network for Shallow Discourse Parsing

By Cherubin Manokaran

Introduction:

Neural Networks are particularly useful for identifying relationships and similarity between words in a data set for different tasks compared to other linear models. Shallow discourse parsing is the task of determining the semantic relations between connectives and their arguments. Explicit sense classification may be performed using traditional linear models because key information such as the connectives is provided. Essentially, neural networks are more valuable for implicit sense classification. Here a convolutional, feed-forward neural network that provides a framework for identifying implicit and explicit relations, in addition to the lack of a relation, in free text is implemented.

Methods:

Neural Network models are generally computationally expensive; therefore, measures were taken to address this. A couple feature extraction techniques were implemented beginning with a simple bag of words approach. The connective and its argument text may be used. Training and validation data were used to develop this vocabulary or bag of words after preprocessing by tokenization. Implicit relations having no connective were indicated as such. For this reason, connectives were expected to provide information about explicit relations. Frequent, semantically significant words may be identified in implicit relation arguments. Combining arguments and connectives most likely provides the most robust model.

Firstly one hot vectors were constructed and concatenated into matrices for each sentence in the training and validation data provided. The total count of the words, determined based on the vocabulary, was then computed for each sentence. After encoding labels into a one hot matrix and splitting all data into batches, each batch was fed into the network, implemented in tensorflow, containing embedding, convolution, and maxpool layers. The embedding layer provides a distributed representation of the features, which in this case are the words having integer id's. Since in most neural networks, the vanishing gradient problem is caused by the activation function, particularly the popular sigmoid function, a simpler Rectified Linear Unit (ReLu) was implemented in tensorflow. After pooling, unnormalized scores and predictions were found in order to determine accuracy and loss using a softmax layer to ultimately minimize loss and improve prediction accuracy. L2 regularization was also implemented to limit overfitting.

For optimization, or loss minimization, the Adagrad optimizer in tensorflow was used. Adaptive learning rates have been shown to improve model performance. The learning rate decreases from iteration to iteration for this optimizer which showed a more gradual loss minimization. Agrad is also optimal for text classification, in general.

Feature extractions using word2vec mapping data from google was also attempted for a continuous bag of words architecture. For efficiency purpose gensim was imported to load this data. Words were mapped according to this data, while unseen data was mapped to random vectors. Data processing and then training required a great deal of time; therefore, results were unattainable.

Results:

Connectives, being a fairly small feature set, required no restriction in terms of word count or frequency. Selection of most frequent words in the training and validation sets failed to greatly improve results by a great deal. Results were very poor with accuracy, as computed by the model, ranging between 4 and 8% on the test set. Many times, discrimination between sense was lacking. Since validation data consistently had an accuracy above 20%, including at the first checkpoint or evaluation step, the vocabulary must not be sufficiently general. Either a comprehensive list of connectives must be used or words must be added dynamically from new data which is a simple addition to the program.

Training with the arguments proved to be a far more computationally expensive task. Based on observation many significant words had counts between 50 and 100. This range was therefore used to filter words. Nevertheless, the program could not be run to completing when combining the argument data. The individual argument data provides the least amount of information; therefore, the results were understandably lower. Accuracy was generally lower or just above 4-5%. Again, the validation data accuracy was also generally above 10-15%. This raises questions about whether test data is being accurately processed and propagated through the network. When combining arguments, the initial results were close to or higher than that of the connectives alone.

Thresholds for word counts may be set in the extractors python file. Otherwise, all hyperparameters and training parameters must be set in the main file. The defaults were determined to be optimal. The default filter sizes have been set to 3, 4, and 5, but to improve program efficiency many experiments were run using filter sizes of 1 and 2.