Kian Azadi

MLFinal

For this final I focused on each aspect of the data one bit at a time, and ultimately decided on the following preprocessing and features:

PREPROCESSING:

- Removed all capitalization. This reduced the amount of words in my word bag (described below)
- Scrubbed the input of all punctuation, to prevent any artifacts of punctuation to change the value of a word.
- Stripping excess whitespace to fully isolate each word.
- Removed all lone spaces that were present either due to scrubbing of input, or tokenization.
- Create a word bag to store all instances of words. While not perfect, the majority of words were recognized if they were unique. Some instances applied where this did not work quite as intended, however overall was successful.
- Do to the reduced amounts of positive data available, all positive data was copied. Because batching was randomized, it most likely prevented the model from seeing the same data back to back per batch. While there was originally an attempt to use a WeightedSampler, it did not work nearly as effectively for unknown reasons.

FEATURES:

- Use of the wordbag. The wordbag was used to give each sentence a score based on the key associated with that word. Afterwards, the sum of the scrubbed sentences were calculated and added together, to ultimately find the percentage contribution of each word to the sentence. Because both sentences were processed one after the other, novel words don't provide a notable difference. The values are then multiplied by two to provide a value that typically lies between 0.5 and 1, rather than 0 and 0.5.
- Levenshtein distance: The Levenshtein distance was calculated using polyleven, a library for python that is able to process English words at an impressive rate. This feature showed how closely worded each of the scrubbed sentences were.
- BLEU1, BLEU2, BLEU3, BLEU4 using the nltk library, using the smoothing function method4, which according to literature seems to be a generally accepted successful smoothing function.

MODEL

The model is a fully connected neural network using the pytorch library with 2 exposed linear layers, followed by 3 hidden linear layers. According to pytorch documentation, all layers are fully connected. While linear layers were exclusively used for the final model, other layers were attempted but there were mixed and/or inconsistent results (layers include pooling layers, padding layers, normalization layers, sparse layers, and dropout layers). Other layers either did not apply to this problem, or were prohibited from usage in this project. The hidden vary in size, with the most being in the middle, and then tapering down to one before the activation function was used.

The activation function used was Sigmoidal. While there was usage of other functions (such as ReLU, LeakyReLU, and tanh in conjunction with normalization) , none others worked quite as intended.

The number of batches used to train the model was refined to 100. While 75 worked as well for the most part, there were times when it dropped performance seemingly at random, which 100 did not experience. The batches were randomized each iteration, so no two batches were ever the exact same per epoch.

The learning rate was set to 0.02. While this appears to be high when looking at other models and literature, it worked the most optimally for this particular model.

100 epochs were run. Various amounts of epochs were run, and while there was significant dropoff after 75 epochs, there was a similar issue found with batch number that would cause the model to seemingly at random drop significantly.

The loss function used was the pytorch BCELoss function, which was widely regarded as one of the best choices for binary classification models. The optimization used was the SGD (Stochastic Gradient) function. While I originally attempted to use the Adam optimizer function, it appears my model is too shallow and I constantly encountered odd results.

The threshold was refined using the dev set F1 calculations as a way to find the threshold that didn't cause too many false positives, while ensuring positives were being selected. While observing the training data, it became clear that the negative values were dropping towards zero incredibly fast, while the positive data tended to linger and not rise as fast.

Throughout training, F1 was calculated for the training set after training took place (between 0.8 and 0.85), and for the dev set (between 0.69 and 0.72). Because of the likely bias towards the training set, the F1 for that set was removed when it became clear. The F1 score for the dev set is still persistent and was the primary source of determining the fine-tuned settings for the threshold, batch size, epochs, learning rate, loss function, and optimizer.