1. Building a Classifier
   1. Design a neural classifier

We based our model off of the Academic Paper Classifier in the AllenNLP tutorial. Firstly, we changed the variable names to align up with the returned dictionary of the provided dataset reader in the starter code. Then in order to give the predictor what it wants, we put the original unencoded sentences into the output dictionary as metadata.

We initialized our model with a text field embedder, the number of classes, encoders for the origin language and the target language, and a classifier feed forward that uses 2 fully connected layers that uses a tanh activation function on all but the last layer. Both the encoders used stacked bidirectional LSTMs and utilized Google News embeddings and an ELMo token embedding to give us an embedded representation of tokens.

In our forward function, we take in a dictionary mapping strings to tensors for source and candidate languages, the labels (if we have them), and the metadata that contains the actual strings. We return how probable we believe each translation was for either class (machine vs human translation) and the loss if we have the labels. With the encoders for both the source and candidate translations, we get the text field mask and pass the masks and embedded translations through their respective encoders, concatenate the two outputs and send the output to the classifier feedforward to get the logits. We then pass the logits to the cross entropy loss function to determine how well our model was performing through its loss.

We classify the sentences as machine translated or human translated in decode by taking the argmax over the class probabilities and choosing the one with the highest probability.

* 1. Implement your neural classifier

We implemented our neural classifier as an AllenNLP model in mt\_classifier/mt\_classifier\_model.py.

* 1. Create a config file

We created a config file and named it mt\_classifier\_config.jsonnet.

* 1. Accuracy

We decide to use the 4th experiment as our hyperparameter of choice because offers the highest dev\_set accuracy.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Learning Rate | Encoder Type | Hidden Size | Classifier Layers | Elmo Embeddings? | Google News Embeddings? | Classifier dimensions? | Training Accuracy | Dev Accuracy |
| 0.01 | lstm | 256 | 3 | yes | yes | 1024, 512, 128, 2 | 81.57% | 73.60% |
| 0.01 | Stacked Bidirectional LSTM | 256 | 3 | yes | yes | 1024, 512, 128, 2 | 78.15% | 74.00% |
| 0.01 | Stacked Bidirectional LSTM | 256 | 2 | yes | yes | 1024, 128, 2 | 79.00% | 73.90% |
| 0.01 | Stacked Bidirectional LSTM | 128 | 2 | yes | yes | 512, 128, 2 | 79.40% | 75.10% |
| 0.01 | Stacked Bidirectional LSTM | 128 | 2 | yes | no | 512, 128, 2 | 78.50% | 74.80% |

1. Kaggle

We end up with 72.666% on Kaggle.