CSCI 544 NLP Assignment
(Deep Learning – Sequence Modeling)

Alagappan Maruthappan maruthap@usc.edu USC ID: 4216160750

Problem Statement:

Design a sequence model to predict the Parts of Speech (POS) tags of a sentence given pre-tagged training sequences.

Model:

Two models were designed. Both of them performed almost similarly. Both the models were built using Keras. First 2 layers were common for both the models:

i. Embedding Layer – A vector representation for every word in the document corpus is learned in the first layer.
ii. Bi-directional LSTM – LSTMs are special kind of RNN capable of learning long term dependencies. Adding Bi-directional wrapper makes the input run in 2 ways (past to future and future to past).
iii. Variant I:
    a. Time Distributed Dense Layer (Softmax): Dense or fully connected layer performs a linear operation in which every input is connected to every output with a weight. Time Distributed wrapper was used so that layer is applied to every temporal time step. Softmax activation function was used.
    Note: Using batch normalization with a momentum of 0.85 increased speed and accuracy. Since batch normalization is used, the bias term in dense layer was removed, batch normalization has an bias term.
iv. Variant II:
    a. Time Distributed Dense Layer (ReLU) – Using the softmax function as the previous variant seemed to decrease the performance when coupled with CRF layer. Among ReLU, sigmoid and tanh, ReLU was fast and had higher accuracy. Using LeakyReLU further slightly increased the accuracy.
    b. CRF Layer – It is a discriminative undirected probabilistic model. CRF layer is not available in standard Keras package. Keras-contrib package was used.

Loss Function:

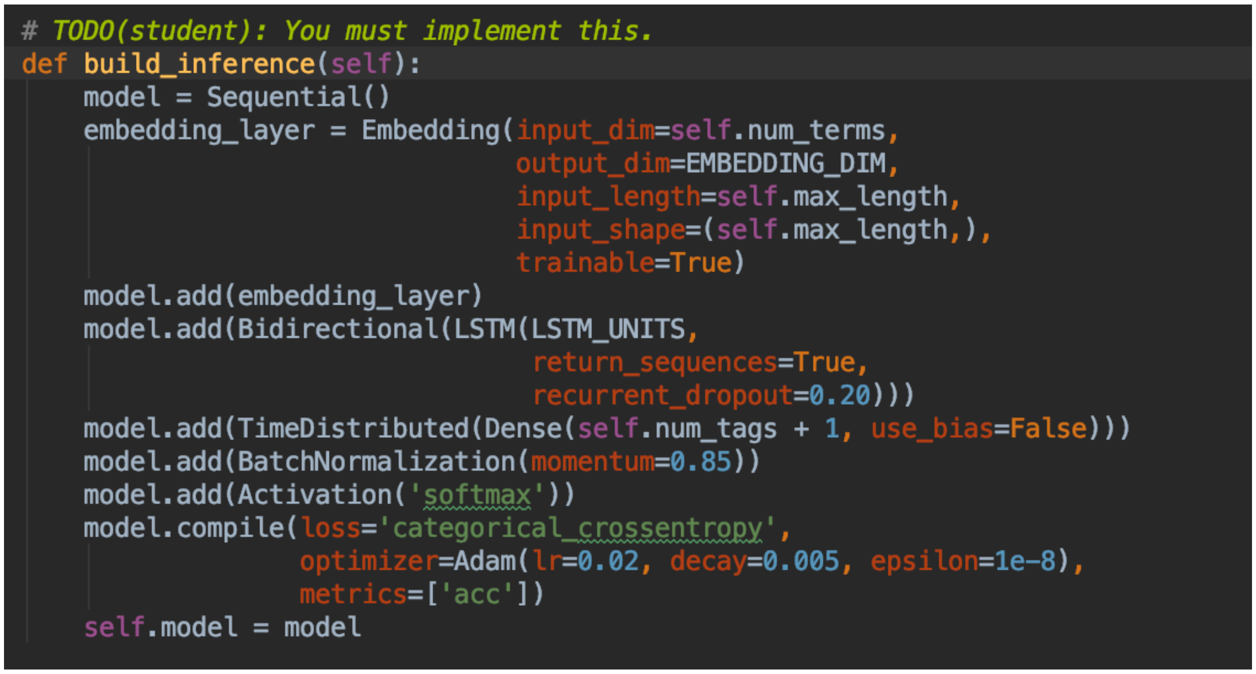
i. Variant I: Categorical cross entropy loss – Also called as Softmax loss. It is the cross-entropy loss with Softmax activation.
ii. Variant II: CRF Loss – It's a general crf loss function both for join and marginal mode. If the CRF is in join mode, the negative log-likelihood for linear chain Conditional Random Field (CRF) is considered as loss. Else in marginal mode categorical cross entropy is again used as the loss measure.

Optimizer:

Various optimizers including SGD, RMSprop, Adagrad were tested. Adam optimizer seemed to perform well and was used in the model with the following parameters:

a) Learning rate: 0.01
b) Decay: 0.005
c) Beta_1: 0.9

d) Beta_2: 0.999

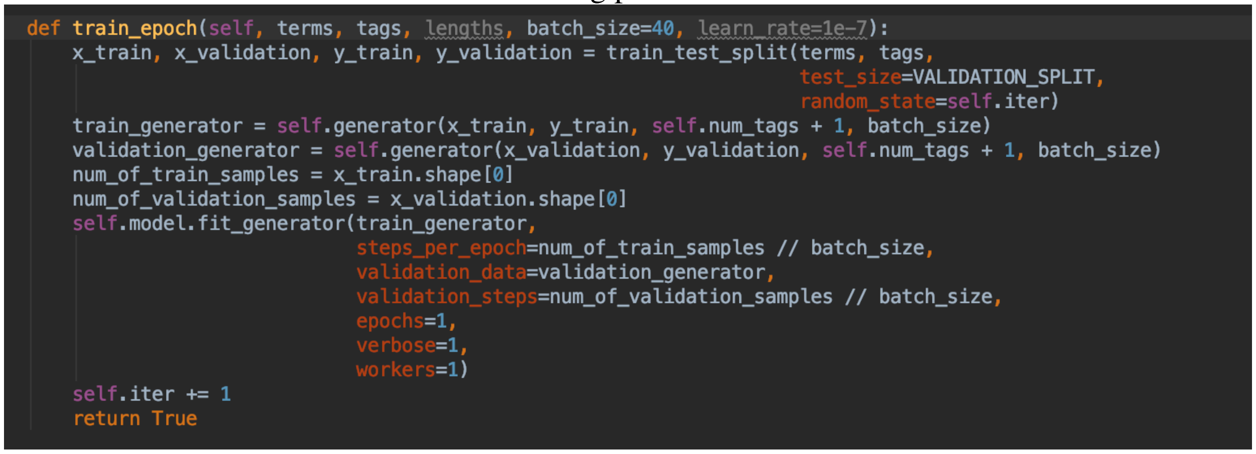


```
# TODO(student): You must implement this.
def build_inference(self):
    model = Sequential()
    embedding_layer = Embedding(input_dim=self.num_terms,
                                output_dim=EMBEDDING_DIM,
                                input_length=self.max_length,
                                input_shape=(self.max_length,),
                                trainable=True)
    model.add(embedding_layer)
    model.add(Bidirectional(LSTM(LSTM_UNITS,
                                 return_sequences=True,
                                 recurrent_dropout=0.20)))
    model.add(TimeDistributed(Dense(self.num_tags + 1, use_bias=False)))
    model.add(BatchNormalization(momentum=0.85))
    model.add(Activation('softmax'))
    model.compile(loss='categorical_crossentropy',
                  optimizer=Adam(lr=0.02, decay=0.005, epsilon=1e-8),
                  metrics=['acc'])
    self.model = model
```

<u>Training:</u>

The following practices were used during training:

1. The data was split as training (80%) and testing (20%) and based on the accuracy on validation data, tested if the model is over fitting the training data and tuned the hyper parameters accordingly.
2. Shuffling the test data on each epoch helped the model converge faster.
3. Batch size was set to 40. Increasing the batch size reduced the accuracy and decreasing the batch size slowed down the training phase.



```
def train_epoch(self, terms, tags, lengths, batch_size=40, learn_rate=1e-7):
    x_train, x_validation, y_train, y_validation = train_test_split(terms, tags,
                                                                    test_size=VALIDATION_SPLIT,
                                                                    random_state=self.iter)
    train_generator = self.generator(x_train, y_train, self.num_tags + 1, batch_size)
    validation_generator = self.generator(x_validation, y_validation, self.num_tags + 1, batch_size)
    num_of_train_samples = x_train.shape[0]
    num_of_validation_samples = x_validation.shape[0]
    self.model.fit_generator(train_generator,
                             steps_per_epoch=num_of_train_samples // batch_size,
                             validation_data=validation_generator,
                             validation_steps=num_of_validation_samples // batch_size,
                             epochs=1,
                             verbose=1,
                             workers=1)
    self.iter += 1
    return True
```

<u>Accuracy Achieved:</u>

95+ Accuracy in Japanese & Italian test set and >90 percentile in surprise test set