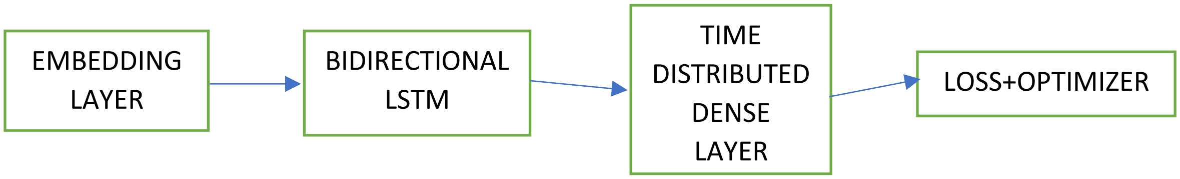
Name: Bharath Chintamani
Email: chintama@usc.edu

Model used (Attached a link at the end of document for model given by tensorboard):

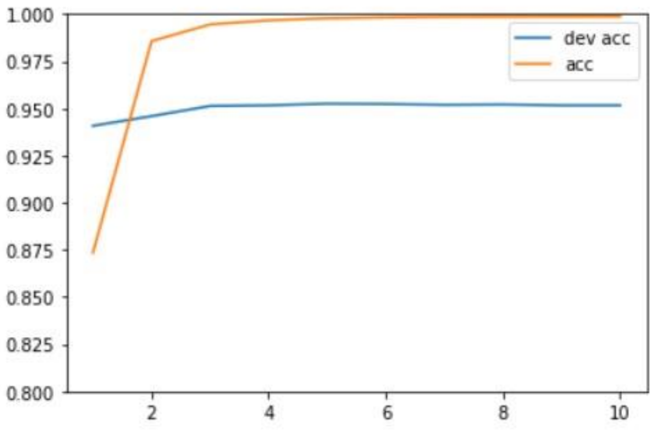

EMBEDDING LAYER
BIDIRECTIONAL LSTM
TIME DISTRIBUTED DENSE LAYER
LOSS+OPTIMIZER

```
embeddings = tf.get_variable('embeddings', [self.num_terms, 30])
model_output = tf.nn.embedding_lookup(embeddings, self.x)
cell = tf.keras.layers.LSTMCell(48)
layer2 = tf.keras.layers.Bidirectional(tf.keras.layers.RNN(cell, return_sequences=True))
model_output = layer2(model_output)
layer3 = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(self.num_tags))
self.logits = layer3(model_output)
```

Stuff that improved my model:

Hyper parameter tuning:

Learning rate (LR) is one of the most import hyper parameter that effects any model as the time for each language data set is quite small. Intuitively, I started with a high LR and decreased LR after each iteration as follows:
Initial learn_rate = 4x10^2
learn_rate*=((0.8)**(self.iter)) #new learn_rate = (initial learn rate)*(0.8)$^{\text{iteration number - 1}}$
learn_rate/=(self.iter+1) #new learn rate = new_learn_rate/(iteration number)
This is done because we want to decrease the loss fast early and slower in later iterations.
Embedding dimension size: I tried with many sizes but 30 worked best for both languages
Number of units for LSTM Cell: 48 units gave the best output for both languages
Early stopping: I used max iteration as 4 -> So it will run at most 4 iterations for any language. I observed after this, we are over fitting the model/dev accuracy doesn't change much after 4 iterations (as shown below).
Batch size: I used a batch size of 64.
Dropout: I tried different dropout and recurrent dropout, but it didn't have much effect.



Training vs Testing Accuracies for Japanese language

Loss used:

I experimented with different losses.

1) seq2seq.sequence_loss + Adam Optimizer

```
losses = tf.contrib.seq2seq.sequence_loss(self.logits,self.target,self.weights)
opt = tf.train.AdamOptimizer(learning_rate=self.learning_rate)
self.train_op = opt.minimize(losses)

#predict tags as or decoding as:
numpy.argmax(logits, axis=2)
```

2) CRF - https://en.wikipedia.org/wiki/Conditional_random_field + Adam Optimizer

```
crf_params = tf.get_variable("CRF", [self.num_tags, self.num_tags], dtype=tf.float32)
log_likelihood, self.transition_params = tf.contrib.crf.crf_log_likelihood(
        self.logits,self.target,self.lengths,crf_params)
losses = tf.reduce_mean(-log_likelihood)
opt = tf.train.AdamOptimizer(learning_rate=self.learning_rate)
self.train_op = opt.minimize(losses)
```

and predict tags as (Viterbi Decoding):

```
            logits,transitions = self.sess.run([self.logits,self.transition_params], {self.x: terms, self.lengths: lengths})
            #get logits and transition matrix
            pred_tags = numpy.zeros(terms.shape) #create a matrix for predicted tags
            for seq_len, logits_data in zip(lengths,logits):
                score = numpy.array(logits_data[:seq_len]).astype(numpy.float32)
                viterbi_sequence, viterbi_score = tf.contrib.crf.viterbi_decode(
                              score,  transitions) #for each sentence, get the corresponding score from logits and use
        the transition for Viterbi decoding to get the sequence of tags
                pred_tags[i,:seq_len] = viterbi_sequence
                i+=1
             return pred_tags
```

I tried with both 1 and 2, they gave almost the same results but went with CRF because thought it will fit better with surprise language.

Doing the above, I got an accuracy of **~95.7%** on ***Italian***, **~95.2%** on ***Japanese*** and >**95th** Percentile on the ***Surprise Language***.

Model as given by tensorboard:

CRF Model: https://drive.google.com/file/d/1SYaqOQoJW_Nf_9pJ46YPJreLOQENF9oQ/view -> looks complicated but TensorFlow does most of the work.

Normal model: https://drive.google.com/open?id=1KF1ljXdvzoW_qgSag-BVw8vronouD11F