Algorithm Pipeline
Our team used the baseline Keras Long-Short Term Memory (LSTM) code made public and shared by Tong Hui Kang and worked from there.

We first started off with splitting the training dataset into 2 parts: validation and training set. We decided to use 1% of the training set for validation rather than follow the original code which used only 1000 data points for validation. The validation set allows us to test accuracy of our model during the training process:

```
train_df = pd.read_csv("../input/ndsc-beginner/train.csv")
train_df = train_df.sample(frac=1.)
val_df = train_df[:int(0.01*len(train_df))]
train_df = train_df[int(0.01*len(train_df)):]
val_df.head()
```

Next, the code made use of word embedding, to capture the semantic relationships between words. This will help us in predicting the category of the test data with the title information. We decided to use pre-trained word vectors (glove.840B.300d) from https://nlp.stanford.edu/projects/glove/ to generate embeddings.

```
# Embdedding setup, save it in a dictionary for easier queries
embeddings_index = {}
f = open('../input/glove840b300dtxt/glove.840B.300d.txt')
for line in tqdm(f):
    values = line.split(" ")
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print(coefs.shape)

print('Found %s word vectors.' % len(embeddings_index))
```

```
2196018it [02:33, 14283.61it/s]


(300,)
Found 2196017 word vectors.
```

We then generated a number of batches of training data, each of size 128 using the training dataset. Shuffling of data is done to ensure that the change done by each individual data point is independent and was not biased by the same points before them.

To train the model faster, CuDNNLSTM is used instead of LSTM. Dropout rate for the first layer is set to 0.05 and activation function used is “softmax”. We stuck with the original author’s activation function as “softmax” is meant for multi-class logistic regression while the typical sigmoid is normally for 2 class logistic regression. Since our targeted prediction output is not one-hot encoded but instead are integers, where each integer represents a different category, we also stuck with the original loss function which is “sparse_categorical_crossentropy”.

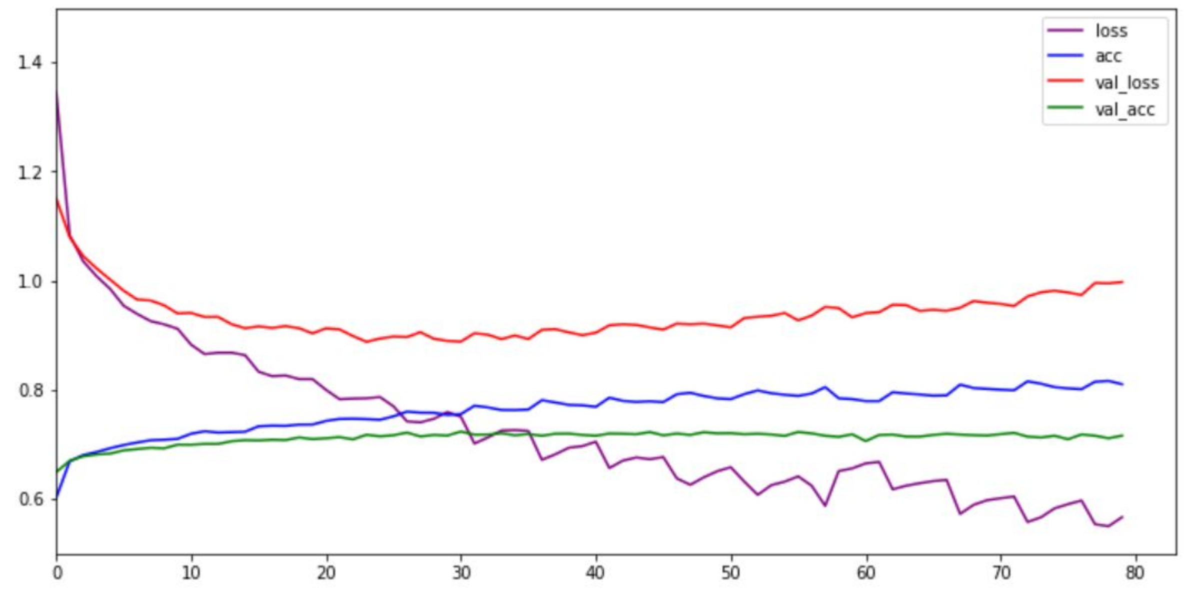
```
model = Sequential()
model.add(Bidirectional(CuDNNLSTM(128, return_sequences=True),
                        input_shape=(300,300)))
model.add(Dropout(0.05))
model.add(Bidirectional(CuDNNLSTM(128)))
model.add(Dense(58, kernel_initializer='normal'))
model.add(Activation('softmax'))

model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Under the dense layer that was added, normal distribution will be used to initialise weights.

Model is then trained on data generated batch by batch.

Result Analysis



The accuracy of our model is about 0.7, and loss, which is used to measure of inconsistency between actual and predicted value, is about 0.6.

Summary

Keras LSTM is a good model as it is able to learn long term dependencies. The baseline model, although with considerably high accuracy, has still room for improvement. However, due to time constraints, we only made minor changes to the baseline model provided. We would like to thank and credit the original author for providing the baseline code and helping our group learn more about LSTM.