

# Deep Learning Lab Assignment-3

Name: Saketh Garuda

ID: 16

## Introduction:

This lab assignment helps in learning about the working of tensor flow and implementation of text classification with CNN, RNN and LSTM model using python. Visualization charts and graphs are created and displayed in tensor board.

## Objective:

The main objective for this lab assignment is to know the

- Text classification and analysis using CNN, RNN and LSTM methods
- Visualizing graphs in tensorboard
- Providing justification for accuracy analysis between the models.

## Configuration:

- PyCharm IDE
- Python 3.6.4
- TensorFlow

## Approaches/Methods:

The CNN, RNN and LSTM models are implemented using python libraries 3.6. Tensorflow and visualization graphs are created using tensor board.

## Workflow:

The workflow for the entire model is as follows,

- Loading dataset using import library modules
- Processing data
- Performing CNN operations on data
- Calculating loss and predicting accuracy
- Using session graph for creating visualization graphs in tensorboard.

## Dataset:

Consumer complaints dataset is used in this exercise, which consists of 11 classes. LSTM, CNN and RNN models are used to preprocess and train the dataset.

## Parameters:

The parameters to be considered while constructing the model are,

- Number of epochs
- Batch Size
- Embedding dimensions
- Size of filter
- Probability check
- Dropout probability

## Evaluation and Discussion:

### Text Classification using CNN:

```
# Training parameters
tf.flags.DEFINE_integer("batch_size", 128, "Batch Size (default: 64)")
tf.flags.DEFINE_integer("num_epochs", 10, "Number of training epochs (default: 200)")
tf.flags.DEFINE_integer("evaluate_every", 100, "Evaluate model on dev set after this many steps (default: 100)")
tf.flags.DEFINE_integer("checkpoint_every", 100, "Save model after this many steps (default: 100)")
tf.flags.DEFINE_integer("num_checkpoints", 5, "Number of checkpoints to store (default: 5)")
```

```
# Training parameters
tf.flags.DEFINE_integer("batch_size", 64, "Batch Size (default: 64)")
tf.flags.DEFINE_integer("num_epochs", 20, "Number of training epochs (default: 200)")
tf.flags.DEFINE_integer("evaluate_every", 100, "Evaluate model on dev set after this many steps (default: 100)")
tf.flags.DEFINE_integer("checkpoint_every", 100, "Save model after this many steps (default: 100)")
tf.flags.DEFINE_integer("num_checkpoints", 5, "Number of checkpoints to store (default: 5)")
```

```

# Data Loading params
tf.flags.DEFINE_float("dev_sample_percentage", .2, "Percentage of the training data to use for validation")
tf.flags.DEFINE_string("traindata", "./data/traindata", "Data source for the training data.")
tf.flags.DEFINE_string("testdata", "./data/testdata", "Data source for the testing data.")

# Model Hyperparameters
tf.flags.DEFINE_integer("embedding_dim", 128, "Dimensionality of character embedding (default: 128)")
tf.flags.DEFINE_string("filter_sizes", "3,4,5", "Comma-separated filter sizes (default: '3,4,5')")
tf.flags.DEFINE_integer("num_filters", 128, "Number of filters per filter size (default: 128)")
tf.flags.DEFINE_float("dropout_keep_prob", 0.5, "Dropout keep probability (default: 0.5)")
tf.flags.DEFINE_float("l2_reg_lambda", 0.0, "L2 regularization lambda (default: 0.0)")

# Training parameters
tf.flags.DEFINE_integer("batch_size", 32, "Batch Size (default: 64)")
tf.flags.DEFINE_integer("num_epochs", 40, "Number of training epochs (default: 200)")
tf.flags.DEFINE_integer("evaluate_every", 100, "Evaluate model on dev set after this many steps (default: 100)")
tf.flags.DEFINE_integer("checkpoint_every", 100, "Save model after this many steps (default: 100)")
tf.flags.DEFINE_integer("num_checkpoints", 5, "Number of checkpoints to store (default: 5)")

# Misc Parameters
tf.flags.DEFINE_boolean("allow_soft_placement", True, "Allow device soft device placement")
tf.flags.DEFINE_boolean("log_device_placement", False, "Log placement of ops on devices")

```

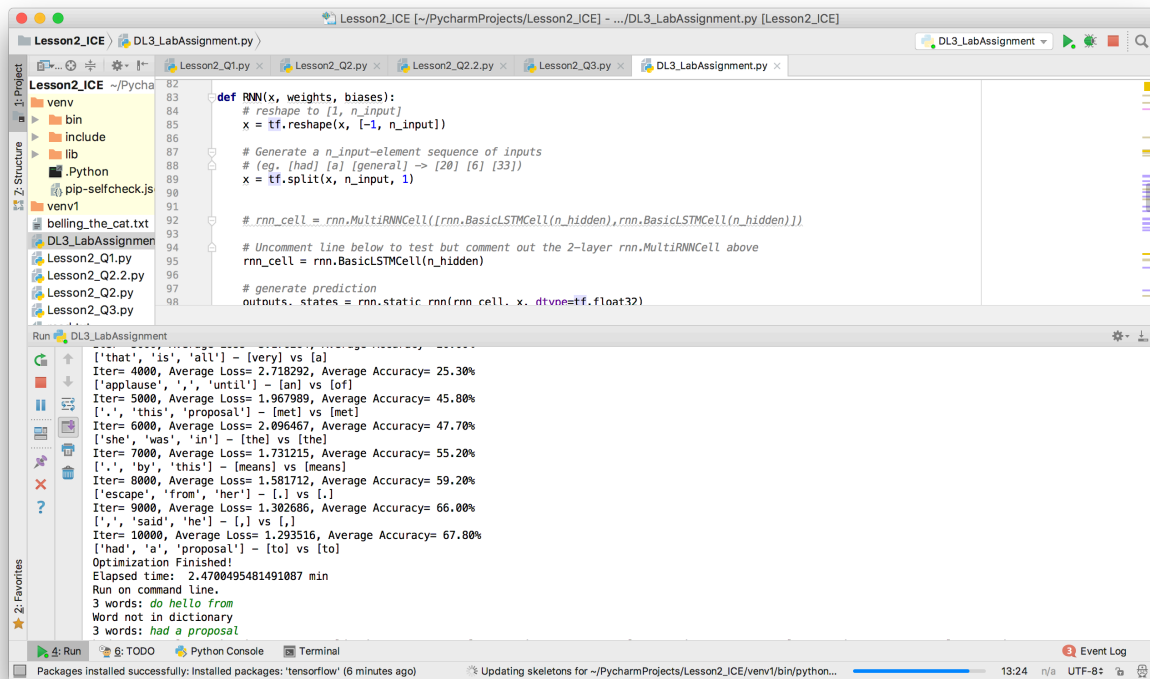
C:\WINDOWS\system32\cmd.exe

```

2018-04-23T19:52:57.264350: step 726, loss 0.359526, acc 0.835938
2018-04-23T19:52:58.147991: step 727, loss 0.248865, acc 0.898438
2018-04-23T19:52:59.109674: step 728, loss 0.315458, acc 0.851562
2018-04-23T19:53:00.039832: step 729, loss 0.302336, acc 0.890625
2018-04-23T19:53:01.040043: step 730, loss 0.304672, acc 0.890625
2018-04-23T19:53:02.037272: step 731, loss 0.296508, acc 0.882812
2018-04-23T19:53:03.089019: step 732, loss 0.280337, acc 0.875
2018-04-23T19:53:04.043194: step 733, loss 0.329338, acc 0.867188
2018-04-23T19:53:04.910310: step 734, loss 0.37443, acc 0.820312
2018-04-23T19:53:05.848476: step 735, loss 0.416879, acc 0.8125
2018-04-23T19:53:06.755118: step 736, loss 0.386182, acc 0.828125
2018-04-23T19:53:07.576701: step 737, loss 0.318781, acc 0.851562
2018-04-23T19:53:08.395204: step 738, loss 0.35367, acc 0.84375
2018-04-23T19:53:09.224306: step 739, loss 0.305283, acc 0.859375
2018-04-23T19:53:10.026387: step 740, loss 0.366216, acc 0.84375
2018-04-23T19:53:10.857480: step 741, loss 0.306961, acc 0.890625
2018-04-23T19:53:11.786636: step 742, loss 0.309526, acc 0.898438
2018-04-23T19:53:12.647746: step 743, loss 0.335762, acc 0.84375
2018-04-23T19:53:13.480337: step 744, loss 0.395916, acc 0.804688
2018-04-23T19:53:14.291912: step 745, loss 0.305199, acc 0.882812
2018-04-23T19:53:15.103596: step 746, loss 0.359615, acc 0.84375
2018-04-23T19:53:15.925205: step 747, loss 0.297534, acc 0.898438
2018-04-23T19:53:16.753310: step 748, loss 0.348543, acc 0.867188
2018-04-23T19:53:17.657951: step 749, loss 0.302975, acc 0.859375
2018-04-23T19:53:18.566594: step 750, loss 0.361681, acc 0.870968

```

## Text Classification using RNN:



```
def RNN(x, weights, biases):
    # reshape to [1, n_input]
    x = tf.reshape(x, [-1, n_input])

    # Generate a n_input-element sequence of inputs
    # (eg. [had] [a] [general] -> [20] [6] [33])
    x = tf.split(x, n_input, 1)

    # rnn_cell = rnn.MultiRNNCell([rnn.BasicLSTMCell(n_hidden), rnn.BasicLSTMCell(n_hidden)])

    # Uncomment line below to test but comment out the 2-layer rnn.MultiRNNCell above
    rnn_cell = rnn.BasicLSTMCell(n_hidden)

    # generate prediction
    outputs, states = rnn.static_rnn(rnn_cell, x, dtype=tf.float32)
```

Run DL3\_LabAssignment

```
[ 'that', 'is', 'all' ] - [ very ] vs [ a ]
Iter= 4000, Average Loss= 2.718292, Average Accuracy= 25.30%
[ 'applause', ' ', 'until' ] - [ an ] vs [ of ]
Iter= 5000, Average Loss= 1.967989, Average Accuracy= 45.80%
[ '.', 'this', 'proposal' ] - [ met ] vs [ met ]
Iter= 6000, Average Loss= 2.096467, Average Accuracy= 47.70%
[ 'she', 'was', 'in' ] - [ the ] vs [ the ]
Iter= 7000, Average Loss= 1.731215, Average Accuracy= 55.20%
[ '.', 'by', 'this' ] - [ means ] vs [ means ]
Iter= 8000, Average Loss= 1.581712, Average Accuracy= 59.20%
[ 'escape', 'from', 'her' ] - [ ] vs [ ]
Iter= 9000, Average Loss= 1.302686, Average Accuracy= 66.00%
[ '.', 'said', 'he' ] - [ ] vs [ ]
Iter= 10000, Average Loss= 1.293516, Average Accuracy= 67.80%
[ 'had', 'a', 'proposal' ] - [ to ] vs [ to ]
Optimization Finished!
Elapsed time: 2.4700495481491087 min
Run on command line.
3 words: do hello from
Word not in dictionary
3 words: had a proposal
```

Event Log

13:24 n/a UTF-8

## Text Classification using LSTM:

```
def LSTM(x, weights, biases):
    # reshape to [1, n_input]
    x = tf.reshape(x, [-1, n_input])

    # Generate a n_input-element sequence of inputs
    # (eg. [had] [a] [general] -> [20] [6] [33])
    x = tf.split(x, n_input, 1)

    rnn_cell = rnn.MultiRNNCell([rnn.BasicLSTMCell(n_hidden), rnn.BasicLSTMCell(n_hidden)])

    # Uncomment line below to test but comment out the 2-layer rnn.MultiRNNCell above
    # rnn_cell = rnn.BasicLSTMCell(n_hidden)

    # generate prediction
    outputs, states = rnn.static_rnn(rnn_cell, x, dtype=tf.float32)

    # there are n_input outputs but
    # we only want the last output
    return tf.matmul(outputs[-1], weights['out']) + biases['out']

pred = LSTM(x, weights, biases)

# Loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
optimizer = tf.train.RMSPropOptimizer(learning_rate=learning_rate).minimize(cost)

# Model evaluation
correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initializing the variables
init = tf.global_variables_initializer()
```

```
Loaded training data...
Iter= 1000, Average Loss= 4.576476, Average Accuracy= 4.50%
['spoke', '.', 'then'] - [the] vs [?]
Iter= 2000, Average Loss= 2.819081, Average Accuracy= 23.90%
['to', 'bell', 'the'] - [cat] vs [cat]
Iter= 3000, Average Loss= 2.426100, Average Accuracy= 37.70%
['applause', ',', 'until'] - [an] vs [an]
Iter= 4000, Average Loss= 2.037940, Average Accuracy= 48.40%
['in', 'the', 'neighbourhood'] - [.] vs [.]
Iter= 5000, Average Loss= 1.880303, Average Accuracy= 52.10%
['by', 'this', 'means'] - [we] vs [we]
Iter= 6000, Average Loss= 1.448044, Average Accuracy= 63.90%
['easily', 'escape', 'from'] - [her] vs [her]
Iter= 7000, Average Loss= 1.339940, Average Accuracy= 66.50%
['us', '.', 'now'] - [,] vs [,]
Iter= 8000, Average Loss= 1.218868, Average Accuracy= 68.90%
['manner', 'in', 'which'] - [the] vs [the]
Iter= 9000, Average Loss= 0.858300, Average Accuracy= 78.40%
['', 'said', 'he'] - [,] vs [,]
Iter= 10000, Average Loss= 0.691095, Average Accuracy= 83.20%
['he', ',', 'that'] - [our] vs [our]
Optimization Finished!
Elapsed time: 12.375051414966583 min
```

The results between the model i.e. CNN, RNN and LSTM are compared and analyzed on the given data set. RNN and LSTM model are comparatively better in producing accurate results than CNN model for the dataset. LSTM is better over RNN since it LSTM cell can maintain memory for longer periods of time than RNN. Longer term dependencies is an advantage of LSTM which has a good chance of increasing accuracy of data.

## References:

- <https://www.tensorflow.org/tutorials/wide>
- <https://www.kaggle.com/cfpb/us-consumer-finance-complaints>
- <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>