Sravani Konujula
16230172
ID :- 23

Deep Learning
LAB Assignment-2

Introduction
The main aim of this task is to implement text classification using the CNN with different data

Objectives

- To Display the graph in the Tensor Board using tensor flow
- Switch the hyperparameter from one value to other value and note the reports
- Compare the reports and examine the difference

Approaches/Methods

Convolutional Neural Networks are identical to Neural networks which are inclusive of neurons with measurable weight and bias.

This consists of layers

- Convolutional Layer
- Pooling Layer
- Fully-Connected Layer

Convolutional Layer
This layer is important building block of CNN which performs computational huge lifting
Pooling Layer
- This layer is fitted between ConvNet architecture.
- This layer works very independent on depth of slice with MAX operation.
- Works on volume size having W×H×D

Fully-Connected Layer
This have full of connections with complete activations with prior layers and that is done with matrix multiplications

Workflow

- Required libraries are to be imported such as Numpy, Pandas, text CNN
- Parameters are considered for Data, Model and Training
- Data to be used for pre-processing
- Load the wanted data and build vocabulary then shuffle randomly
- Splitting is done as the test and train data
- Cross validation must be done after training the data
- Train the procedure
- Note the reports for gradient and sparsity
- Summarize the loss and accuracy
- Note the checkpoints
- Train the loop for every set
- Evaluate parameters
- Epoch number of operations
- Tensor Boards graphs are displayed

Datasets

- Consumer complaints dataset with total number of 11 classes.

Parameters

```
Parameters:
ALLOW_SOFT_PLACEMENT=True
BATCH_SIZE=64
CHECKPOINT_EVERY=100
DEV_SAMPLE_PERCENTAGE=0.01
DROPOUT_KEEP_PROB=0.5
EMBEDDING_DIM=128
EVALUATE_EVERY=100
FILTER_SIZES=3,4,5
L2_REG_LAMBDA=0.0
LOG_DEVICE_PLACEMENT=False
```

Configuration

Pycharm
Python: 2.7.13
Tensor Flow

Evaluation & Discussion

- Code snippet


```
# Data loading params
tf.flags.DEFINE_float("dev_sample_percentage", .1, "Percentage of the training data to use for validation")
tf.flags.DEFINE_string("positive_data_file", "./data/rt-polaritydata/rt-polarity.pos", "Data source for the p
tf.flags.DEFINE_string("negative_data_file", "./data/rt-polaritydata/rt-polarity.neg", "Data source for the n

# 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", 64, "Batch Size (default: 64)")
tf.flags.DEFINE_integer("num_epochs", 200, "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")

FLAGS = tf.flags.FLAGS
```

```
# Build vocabulary
max_document_length = max([len(x.split(" ")) for x in x_text])
vocab_processor = learn.preprocessing.VocabularyProcessor(max_document_length)
x = np.array(list(vocab_processor.fit_transform(x_text)))

# Randomly shuffle data
np.random.seed(10)
shuffle_indices = np.random.permutation(np.arange(len(y)))
x_shuffled = x[shuffle_indices]
y_shuffled = y[shuffle_indices]

# Split train/test set
# TODO: This is very crude, should use cross-validation
dev_sample_index = -1 * int(FLAGS.dev_sample_percentage * float(len(y)))
x_train, x_dev = x_shuffled[:dev_sample_index], x_shuffled[dev_sample_index:]
y_train, y_dev = y_shuffled[:dev_sample_index], y_shuffled[dev_sample_index:]

del x, y, x_shuffled, y_shuffled

print("Vocabulary Size: {:d}".format(len(vocab_processor.vocabulary_)))
print("Train/Dev split: {:d}/{:d}".format(len(y_train), len(y_dev)))


# Training
# ==================================================

with tf.Graph().as_default():
    session_conf = tf.ConfigProto(
```

```
def train_step(x_batch, y_batch):
    """
    A single training step
    """
    feed_dict = {
      cnn.input_x: x_batch,
      cnn.input_y: y_batch,
      cnn.dropout_keep_prob: FLAGS.dropout_keep_prob
    }
    _, step, summaries, loss, accuracy = sess.run(
        [train_op, global_step, train_summary_op, cnn.loss, cnn.accuracy],
        feed_dict)
    time_str = datetime.datetime.now().isoformat()
    print("{}: step {}, loss {:g}, acc {:g}".format(time_str, step, loss, accuracy))
    train_summary_writer.add_summary(summaries, step)

def dev_step(x_batch, y_batch, writer=None):
    """
    Evaluates model on a dev set
    """
    feed_dict = {
      cnn.input_x: x_batch,
      cnn.input_y: y_batch,
      cnn.dropout_keep_prob: 1.0
    }
    step, summaries, loss, accuracy = sess.run(
        [global_step, dev_summary_op, cnn.loss, cnn.accuracy],
```

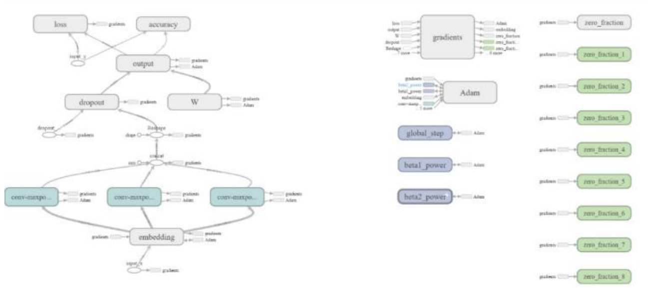
```python
"""
A CNN for text classification.
Uses an embedding layer, followed by a convolutional, max-pooling and softmax layer.
"""
def __init__(
  self, sequence_length, num_classes, vocab_size,
  embedding_size, filter_sizes, num_filters, l2_reg_lambda=0.0):

    # Placeholders for input, output and dropout
    self.input_x = tf.placeholder(tf.int32, [None, sequence_length], name="input_x")
    self.input_y = tf.placeholder(tf.float32, [None, num_classes], name="input_y")
    self.dropout_keep_prob = tf.placeholder(tf.float32, name="dropout_keep_prob")

    # Keeping track of l2 regularization loss (optional)
    l2_loss = tf.constant(0.0)

    # Embedding layer
    with tf.device('/cpu:0'), tf.name_scope("embedding"):
        self.W = tf.Variable(
            tf.random_uniform([vocab_size, embedding_size], -1.0, 1.0),
            name="W")
        self.embedded_chars = tf.nn.embedding_lookup(self.W, self.input_x)
        self.embedded_chars_expanded = tf.expand_dims(self.embedded_chars, -1)

    # Create a convolution + maxpool layer for each filter size
    pooled_outputs = []
    for i, filter_size in enumerate(filter_sizes):
        with tf.name_scope("conv-maxpool-%s" % filter_size):
```

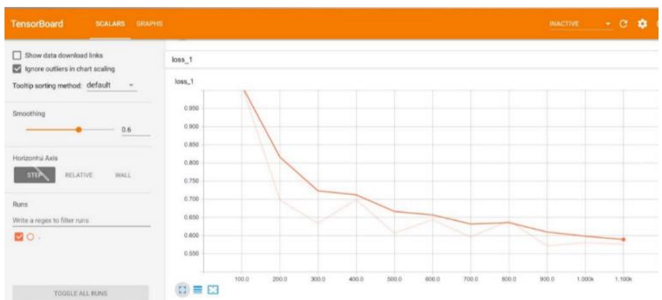
```python
    for i, filter_size in enumerate(filter_sizes):
        with tf.name_scope("conv-maxpool-%s" % filter_size):
            # Convolution Layer
            filter_shape = [filter_size, embedding_size, 1, num_filters]
            W = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.1), name="W")
            b = tf.Variable(tf.constant(0.1, shape=[num_filters]), name="b")
            conv = tf.nn.conv2d(
                self.embedded_chars_expanded,
                W,
                strides=[1, 1, 1, 1],
                padding="VALID",
                name="conv")
            # Apply nonlinearity
            h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
            # Maxpooling over the outputs
            pooled = tf.nn.max_pool(
                h,
                ksize=[1, sequence_length - filter_size + 1, 1, 1],
                strides=[1, 1, 1, 1],
                padding='VALID',
                name="pool")
            pooled_outputs.append(pooled)

    # Combine all the pooled features
    num_filters_total = num_filters * len(filter_sizes)
    self.h_pool = tf.concat(pooled_outputs, 3)
    self.h_pool_flat = tf.reshape(self.h_pool, [-1, num_filters_total])
```

- <u>Output</u>

```
train
2018-04-23T17:12:27.838248: step 982, loss 0.500326, acc 0.8125
2018-04-23T17:12:28.311586: step 983, loss 0.379159, acc 0.84375
2018-04-23T17:12:28.775909: step 984, loss 0.473596, acc 0.765625
2018-04-23T17:12:29.248970: step 985, loss 0.307726, acc 0.859375
2018-04-23T17:12:29.726602: step 986, loss 0.356266, acc 0.859375
2018-04-23T17:12:30.211947: step 987, loss 0.423783, acc 0.78125
2018-04-23T17:12:30.692788: step 988, loss 0.539414, acc 0.71875
2018-04-23T17:12:31.157760: step 989, loss 0.380394, acc 0.859375
2018-04-23T17:12:31.602589: step 990, loss 0.411435, acc 0.833333
2018-04-23T17:12:32.075926: step 991, loss 0.349379, acc 0.84375
2018-04-23T17:12:32.561771: step 992, loss 0.392038, acc 0.828125
2018-04-23T17:12:33.020096: step 993, loss 0.287705, acc 0.890625
2018-04-23T17:12:33.507943: step 994, loss 0.381357, acc 0.8125
2018-04-23T17:12:33.980279: step 995, loss 0.399409, acc 0.8125
2018-04-23T17:12:34.438605: step 996, loss 0.271033, acc 0.921875
2018-04-23T17:12:34.903121: step 997, loss 0.386839, acc 0.875
2018-04-23T17:12:35.367450: step 998, loss 0.288989, acc 0.875
2018-04-23T17:12:35.842789: step 999, loss 0.37101, acc 0.859375
2018-04-23T17:12:36.316695: step 1000, loss 0.423611, acc 0.796875
```

- <u>Tensor flow graph</u>





Conclusion

By using tensor flow the code both deploy and debug has done.
By CNN increment of accuracy got satisfied.
There is a chance of increment in both accuracy and complexity if we use RNN's and LSTM's

References

http://cs231n.github.io/convolutional-networks/