Deep Learning

LAB 2

Namrata Dutta | Deep learning | 16247052

1. INTRODUCTION

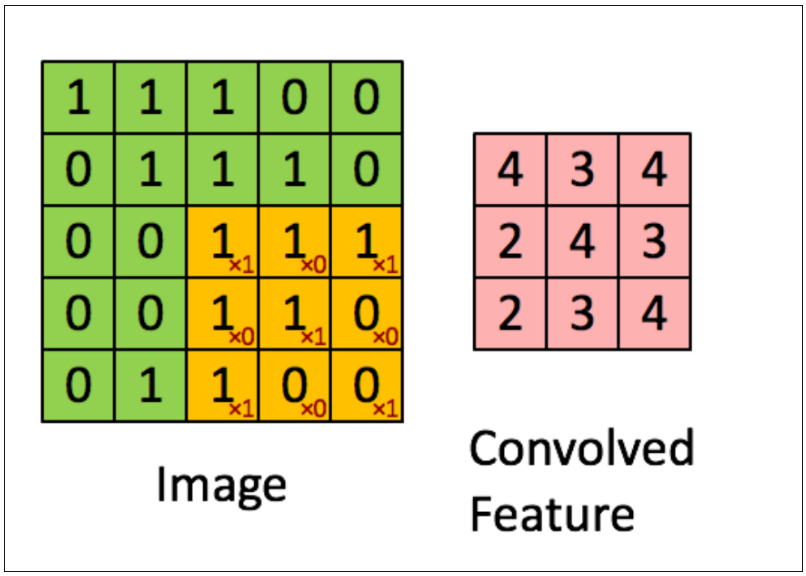
The convolutional neural networks (CNN) is a biologically inspired variants of Multilayer perceptron (MLPs) which is an artificial neural network usually availed to visualize an image. CNN can be seen as a sliding window function on a matrix. CNN involves in 4 major operations which are, Convolution, Non linearity, Sub Sampling and Classification. It has an input layer, an output layer and a multiple hidden layer. CNN's multilayer algorithm deals with supervised learning which is essential for minimal preprocessing. Here, in this task we will be doing text classification using CNN on word embedding.

2. OBJECTIVE

The objective of this assignment is to perform text classification with CNN and observe the accuracy and loss for the same. We have taken a sample text data which contains more than 10000 words and it is partial positive and partial negative sentences. The sample dataset is basically reviews from several customers about various products like mobile phones and laptops. So, here we are differentiating the positive and negative reviews by text classification.

3. APPROACH/ METHOD

The sample text dataset contains partial positive and partial negative sentences which are further divided into words. These words undergo word embedding into low- dimensional vectors for the first layer of CNN. Next, Convolutional is applied on these embedded words generated from the first layer. The multilayer perceptron uses SoftMax activation function on the output. The activation function defines the output of that particular node. For implementation of this function, the placeholders for the data is created. While executing the training data or the test data placeholders has been created by using the command **tf. placeholder**. The input for the batch size and the shape can be done by using this placeholder. Finally, the cross-entropy loss and accuracy will be calculated according to this model efficiently.



4. WORKFLOW

The steps of the process of CNN model is as follows:

- First, the data set will be read and categorized into similar groups of data.
- The labels of the words are generated, and the vocabulary is built.
- According to the model, a graph will be generated after the processing of the data.
- As mentioned before, the tf.placeholder is used for the placeholders for input and output.
- Next step is to make the data nonlinear, which is done by the activation function and hence, the output varies non- linearly with the input data.
- According to the formula, the input variables are multiplied by the corresponding weights and that data is used as the input to the activation variable.
- The accuracy and loss function is calculated according to our model.
- We pass the data into tensor board to get a visualization of it as a graph and also the accuracy and loss as a graph.

Code snippets:

trainCNN.py | textCNN.py

```
            global_step = tf.Variable(0, name="global_step", trainable=False)
            optimizer = tf.train.AdamOptimizer(1e-3)
            grads_and_vars = optimizer.compute_gradients(cnn.loss)
            train_op = optimizer.apply_gradients(grads_and_vars, global_step=global_step)

            timestamp = str(int(time.time()))
            out_dir = os.path.abspath(os.path.join(os.path.curdir, "trained_model_" + timestamp))

            checkpoint_dir = os.path.abspath(os.path.join(out_dir, "checkpoints"))
            checkpoint_prefix = os.path.join(checkpoint_dir, "model")
            if not os.path.exists(checkpoint_dir):
                os.makedirs(checkpoint_dir)
            saver = tf.train.Saver(tf.global_variables())

            # One training step: train the model with one batch
            def train_step(x_batch, y_batch):
                feed_dict = {
                    cnn.input_x: x_batch,
                    cnn.input_y: y_batch,
                    cnn.dropout_keep_prob: FLAGS.dropout_keep_prob}
                _, step, loss, acc = sess.run([train_op, global_step, cnn.loss, cnn.accuracy], feed_dict)

            # One evaluation step: evaluate the model with one batch
            def dev_step(x_batch, y_batch):
                feed_dict = {cnn.input_x: x_batch, cnn.input_y: y_batch, cnn.dropout_keep_prob: 1.0}
                step, loss, acc, num_correct = sess.run([global_step, cnn.loss, cnn.accuracy, cnn.num_correct], feed_dict)
                return num_correct

            # Save the word_to_id map since predict.py needs it
            vocab_processor.save(os.path.join(out_dir, "vocab.pickle"))
            #sess.run(tf.initialize_all_variables())
            sess.run(tf.global_variables_initializer())
```

trainCNN.py | textCNN.py

```
    """Step 1: pad each sentence to the same length and map each word to an id"""
    max_document_length = max([len(x.split(' ')) for x in x_raw])
    logging.info('The maximum length of all sentences: {}'.format(max_document_length))
    vocab_processor = learn.preprocessing.VocabularyProcessor(max_document_length)
    x = np.array(list(vocab_processor.fit_transform(x_raw)))
    y = np.array(y_raw)

    """Step 2: split the original dataset into train and test sets"""
    x_, x_test, y_, y_test = train_test_split(x, y, test_size=0.1, random_state=42)

    """Step 3: shuffle the train set and split the train set into train and dev sets"""
    shuffle_indices = np.random.permutation(np.arange(len(y_)))
    x_shuffled = x_[shuffle_indices]
    y_shuffled = y_[shuffle_indices]
    x_train, x_dev, y_train, y_dev = train_test_split(x_shuffled, y_shuffled, test_size=0.1)

    """Step 4: save the labels into labels.json since predict.py needs it"""
    #with open('./labels.json', 'w') as outfile:
    #    json.dump(labels, outfile, indent=4)

    logging.info('x_train: {}, x_dev: {}, x_test: {}'.format(len(x_train), len(x_dev), len(x_test)))
    logging.info('y_train: {}, y_dev: {}, y_test: {}'.format(len(y_train), len(y_dev), len(y_test)))

    """Step 5: build a graph and cnn object"""
    graph = tf.Graph()
    with graph.as_default():
        session_conf = tf.ConfigProto(allow_soft_placement=True, log_device_placement=False)
        sess = tf.Session(config=session_conf)
        with sess.as_default():
            cnn = TextCNN(
                sequence_length=x_train.shape[1],
                num_classes=y_train.shape[1],
                vocab_size=len(vocab_processor.vocabulary_),
                embedding_size=FLAGS.embedding_dim,
                filter_sizes=list(map(int, FLAGS.filter_sizes.split(","))),
```

trainCNN.py | textCNN.py

```python
import ...

logging.getLogger().setLevel(logging.INFO)

def train_cnn():
    """Step 0: load sentences, labels, and training parameters"""
    #train_file = sys.argv[1]
    input_file = '/Users/bhavyateja/Github_Projects/Python-DeepLearning_CS5590/DL_Lab_2/Source/TextCNN/consumer_complaints.csv.zip'
    x_raw, y_raw, df, labels = dataHelpers.load_data_and_labels(input_file)

    #parameter_file = sys.argv[0]
    #paramater_file='C:/Users/bvkka/Desktop/New folder/parameters.json'
    # Model Hyper parameters
    tf.flags.DEFINE_integer("embedding_dim", 40, "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", 32, "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 ", 1, "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
    FLAGS._parse_flags()
    print("\nParameters:")
    for attr, value in sorted(FLAGS.__flags.items()):
        print("{}={}".format(attr.upper(), value))
    print("")
```

train_cnn()

trainCNN.py | textCNN.py

```python
            # Training starts here
            train_batches = dataHelpers.batch_iter(list(zip(x_train, y_train)),FLAGS.batch_size,FLAGS.num_epochs)
            best_accuracy, best_at_step = 0, 0

            """Step 6: train the cnn model with x_train and y_train (batch by batch)"""
            for train_batch in train_batches:
                x_train_batch, y_train_batch = zip(*train_batch)
                train_step(x_train_batch, y_train_batch)
                current_step = tf.train.global_step(sess, global_step)

                """Step 6.1: evaluate the model with x_dev and y_dev (batch by batch)"""
                if current_step % FLAGS.evaluate_every == 0:
                    dev_batches = dataHelpers.batch_iter(list(zip(x_dev, y_dev)), FLAGS.batch_size, 1)
                    total_dev_correct = 0
                    for dev_batch in dev_batches:
                        x_dev_batch, y_dev_batch = zip(*dev_batch)
                        num_dev_correct = dev_step(x_dev_batch, y_dev_batch)
                        total_dev_correct += num_dev_correct

                    dev_accuracy = float(total_dev_correct) / len(y_dev)
                    logging.critical('Accuracy on dev set: {}'.format(dev_accuracy))

                    """Step 6.2: save the model if it is the best based on accuracy on dev set"""
                    if dev_accuracy >= best_accuracy:
                        best_accuracy, best_at_step = dev_accuracy, current_step
                        path = saver.save(sess, checkpoint_prefix, global_step=current_step)
                        logging.critical('Saved model at {} at step {}'.format(path, best_at_step))
                        logging.critical('Best accuracy is {} at step {}'.format(best_accuracy, best_at_step))

            """Step 7: predict x_test (batch by batch)"""
            test_batches = dataHelpers.batch_iter(list(zip(x_test, y_test)),FLAGS.batch_size, 1)
            total_test_correct = 0
            for test_batch in test_batches:
                x_test_batch, y_test_batch = zip(*test_batch)
                num_test_correct = dev_step(x_test_batch, y_test_batch)
```

5. DATASETS

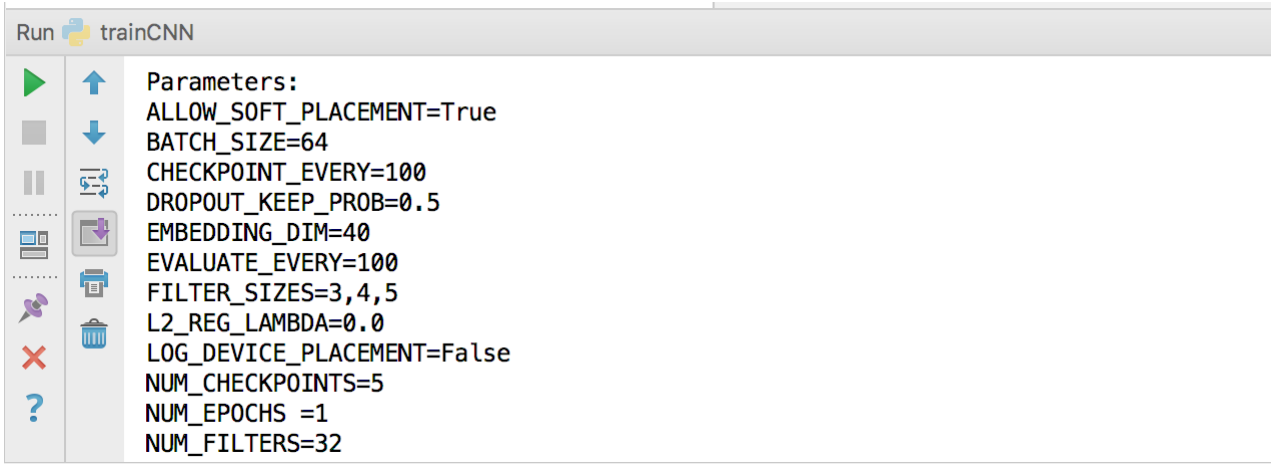
The data set used by us is the review of various customers from amazon for a span of 18 years regarding assorted products. So, we have found out the total number of positive and negative reviews.

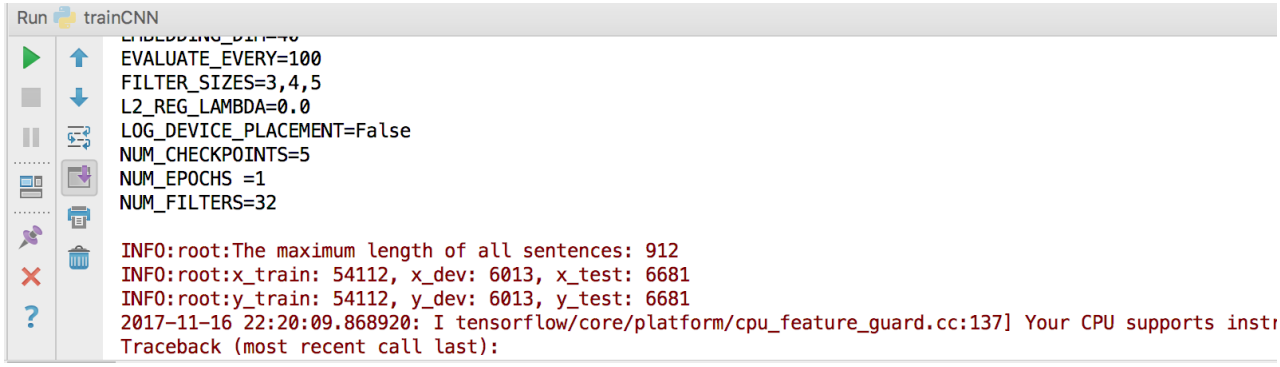
6. PARAMETERS

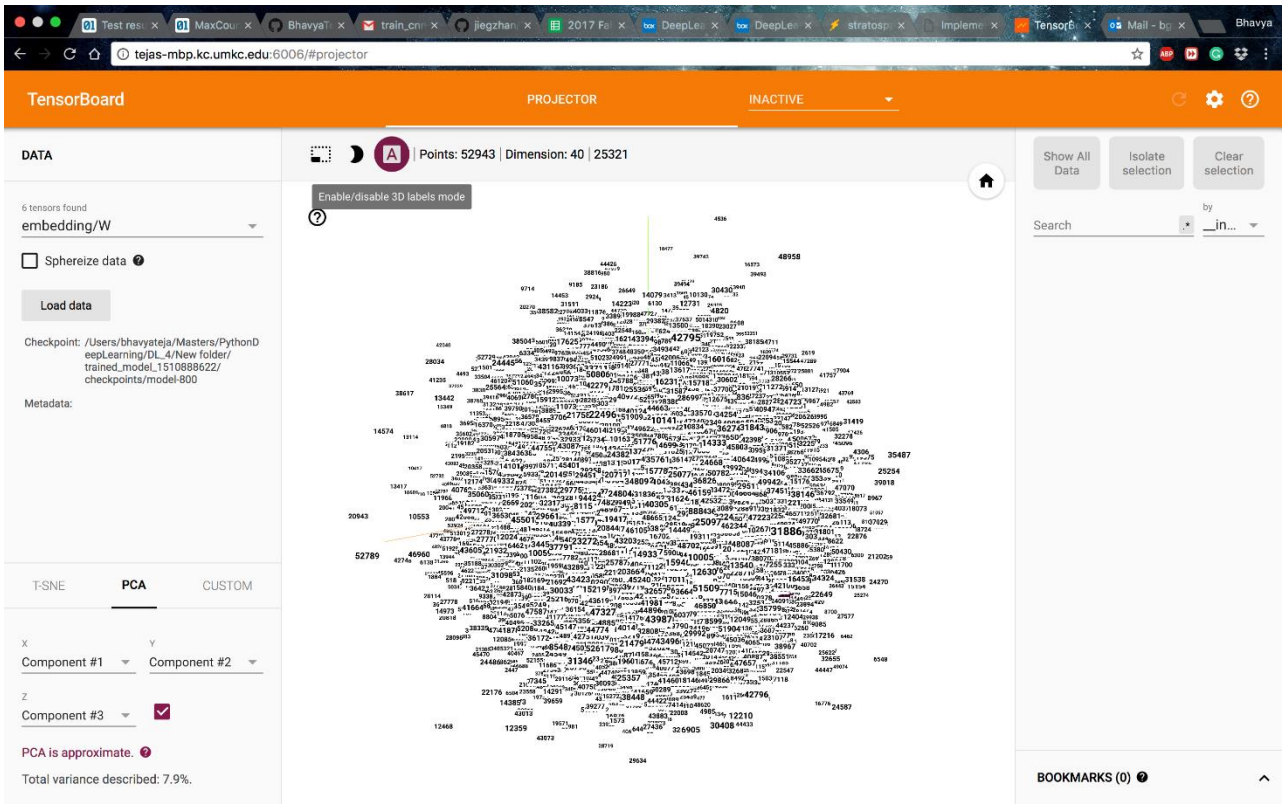
For a CNN text classification, the parameters taken into account are the batch size of the input and output layers, size of the hidden layers, size of the vocabulary and filters and finally the number of nodes in a layer.

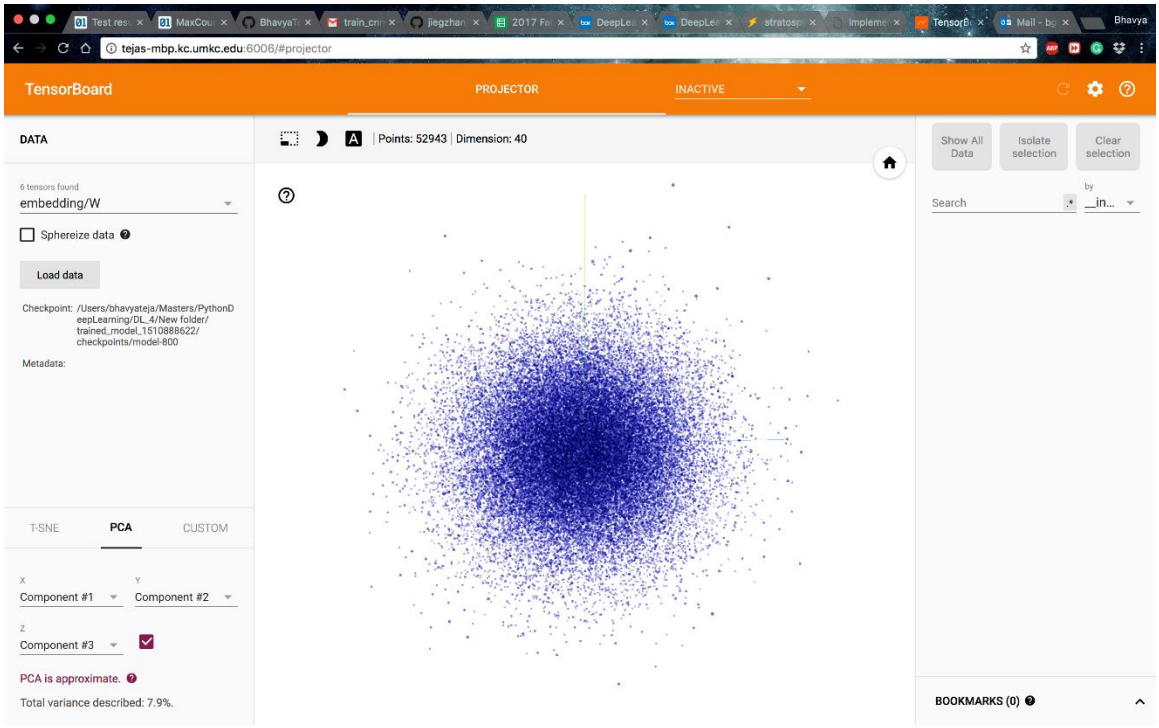
7. EVALUATION & DISCUSSION

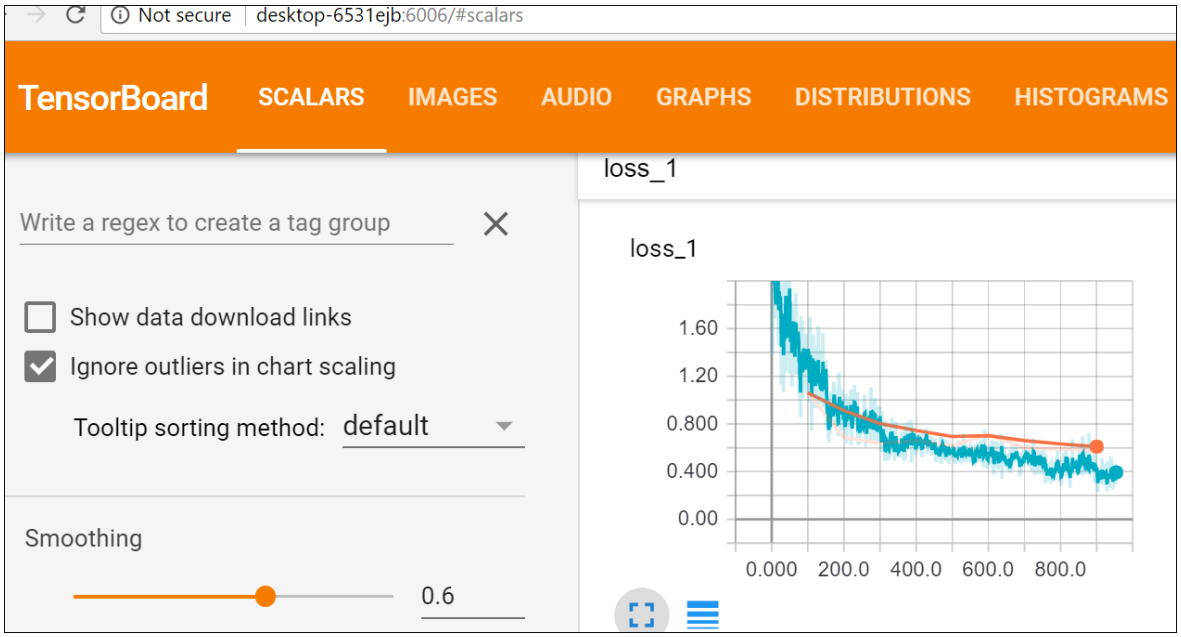
The model was implemented as CNN text classification and evaluated and can be seen in tensor board as a graph.

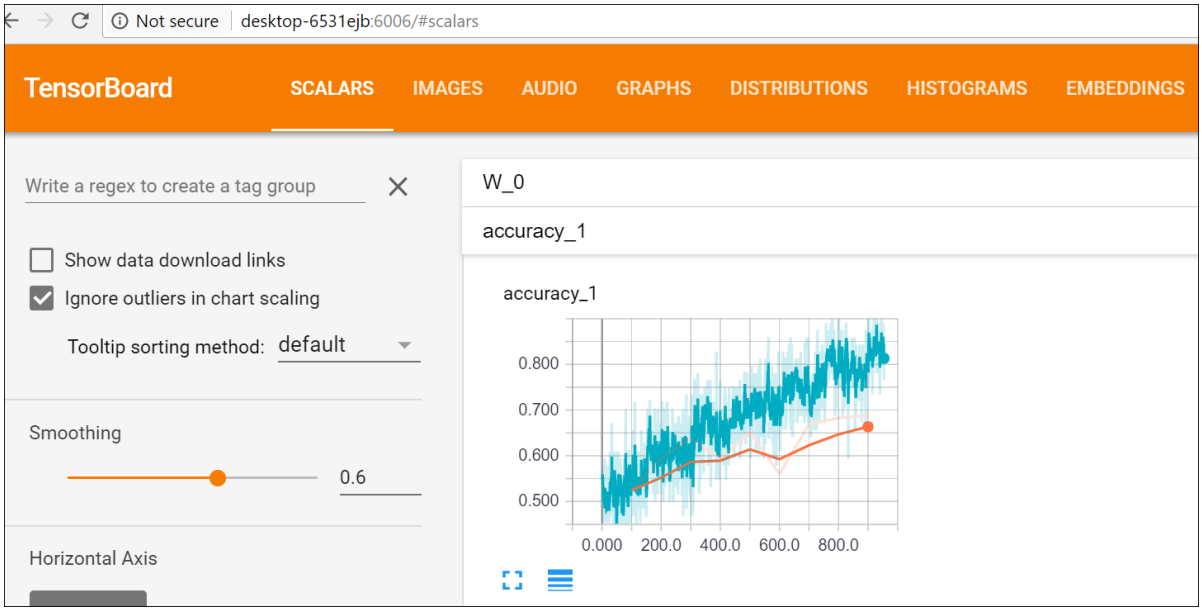





TensorBoard
PROJECTOR
INACTIVE
DATA
6 tensors found
embedding/W
Sphereize data
Load data
Checkpoint: /Users/bhavyateja/Masters/PythonDeepLearning/DL_4/New folder/trained_model_1510888622/checkpoints/model-800
Metadata:
Points: 52943 | Dimension: 40 | 25321
Enable/disable 3D labels mode
Show All Data
Isolate selection
Clear selection
Search
T-SNE
PCA
CUSTOM
X
Component #1
Y
Component #2
Z
Component #3
PCA is approximate.
Total variance described: 7.9%.
BOOKMARKS (0)


TensorBoard
PROJECTOR
INACTIVE
DATA
6 tensors found
embedding/W
Sphereize data
Load data
Checkpoint: /Users/bhavyateja/Masters/PythonDeepLearning/DL_4/New folder/trained_model_1510888622/checkpoints/model-800
Metadata:
Points: 52943 | Dimension: 40
Show All Data
Isolate selection
Clear selection
Search
T-SNE
PCA
CUSTOM
X
Component #1
Y
Component #2
Z
Component #3
PCA is approximate.
Total variance described: 7.9%.
BOOKMARKS (0)





8. Conclusion

The cross validation is to be used for building the best model. The single layer CNN convolution is better fit than multi-layer.