CS 534 Machine Learning

Project 3, due Monday, April 16

Chenxi Cai
Yifei Ren
Qingfeng (Kee) Wang

(Notice, all links (table of contents, figures references, table references etc.) are clickable! You can download this PDF and open it using Preview or Adobe Reader to enable this convenient feature.)

Contents

1 This is first section

I said nothing.

1.1 This is a subsection section.

Table 1: This is a table.

Iteration	3	5	10	20
Error rate(%)	16.67	10.0	10.0	13.3

2 Source codes

Finally, codes are enclosed here.

```
import tensorflow as tf
import numpy as np


#This is to list all INFO
tf.logging.set_verbosity(tf.logging.INFO)

def cnn_model_fn(features, labels, mode):
  """Model function for CNN. This function is used to be called
     later in main function.
    features     :: the feature in array, size nnumber-by-784.
       Here 784 = 28*28 is the flattened representation of a hand
        written figure.
                    Features is a dict structure, {'x': <tf.
                       Tensor 'fifo_queue_DequeueUpTo:1' shape=(
                       nnumber, 784) dtype=float32>}
                    features['x'] corresponds to the acutal
                       tensor object
    labels       :: values from 0 to 9. Size nnumber-by-1. A
       tensor object
    mode         :: One of three modes: TRAIN, EVAL, PREDICT
  """

  # It is a function that used to train data.

  # Input Layer
  input_layer = tf.reshape(features["x"], [-1, 28, 28, 1])

  # Convolutional Layer #1
  conv1 = tf.layers.conv2d(
      inputs=input_layer,
      filters=32,
      kernel_size=[5, 5],
      padding="same",
      activation=tf.nn.relu)

  # Pooling Layer #1
  pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2],
      strides=2)
```

```
# Convolutional Layer #2 and Pooling Layer #2
conv2 = tf.layers.conv2d(
    inputs=pool1,
    filters=64,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu)
pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2],
    strides=2)

# Dense Layer
pool2_flat = tf.reshape(pool2, [-1, 7 * 7 * 64])
dense = tf.layers.dense(inputs=pool2_flat, units=1024,
   activation=tf.nn.relu)
dropout = tf.layers.dropout(
    inputs=dense, rate=0.4, training=mode == tf.estimator.
       ModeKeys.TRAIN)

# Logits Layer
logits = tf.layers.dense(inputs=dropout, units=10)



predictions = {
    # Generate predictions (for PREDICT and EVAL mode)
    "classes": tf.argmax(input=logits, axis=1),#Calculate class
        on the fly
    # Add `softmax_tensor` to the graph. It is used for PREDICT
        and by the
    # `logging_hook`.
    "probabilities": tf.nn.softmax(logits, name="softmax_tensor
       ")
}

if mode == tf.estimator.ModeKeys.PREDICT:
  return tf.estimator.EstimatorSpec(mode=mode, predictions=
     predictions)

# Calculate Loss (for both TRAIN and EVAL modes)
loss = tf.losses.sparse_softmax_cross_entropy(labels=labels,
   logits=logits)
```

```
  # Configure the Training Op (for TRAIN mode)
  if mode == tf.estimator.ModeKeys.TRAIN:
    optimizer = tf.train.GradientDescentOptimizer(learning_rate
       =0.001)#potimized and learning rate can change
    train_op = optimizer.minimize(
        loss=loss,
        global_step=tf.train.get_global_step())
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss,
       train_op=train_op)

  # Add evaluation metrics (for EVAL mode)
  eval_metric_ops = {"accuracy": tf.metrics.accuracy(labels=
     labels, predictions=predictions["classes"])}#predictions is
     a dict and calculate classes on the fly
  return tf.estimator.EstimatorSpec(mode=mode, loss=loss,
     eval_metric_ops=eval_metric_ops)


def main(aa):


    # Load training and eval data
    mnist = tf.contrib.learn.datasets.load_dataset("mnist")

    p = int(len(mnist.train.images)*0.8) #Probably need to
       randomize

    train_data = mnist.train.images[0:p] # Returns np.array
    train_labels = np.asarray(mnist.train.labels, dtype=np.int32)
       [0:p]

    valid_data = mnist.train.images[p:]  # Returns np.array
    valid_labels = np.asarray(mnist.train.labels, dtype=np.int32)
       [p:]


    eval_data = mnist.test.images # Returns np.array
    eval_labels = np.asarray(mnist.test.labels, dtype=np.int32)
```

```
# Create the Estimator
mnist_classifier = tf.estimator.Estimator(model_fn=
   cnn_model_fn, model_dir="/tmp/mnist_convnet_model")
# Set up logging for predictions
tensors_to_log = {"probabilities": "softmax_tensor"}
logging_hook = tf.train.LoggingTensorHook(tensors=
   tensors_to_log, every_n_iter=1)

# Train the model
train_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": train_data},#First input, convert the numpy array
        data into a dict structure
    y=train_labels,
    batch_size=200,
    num_epochs=None,
    shuffle=True)

valid_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": valid_data},
    y=valid_labels,
    num_epochs=1,
    shuffle=False)
experiment = tf.contrib.learn.Experiment(
    mnist_classifier,
    train_input_fn,
    valid_input_fn,
    train_steps = 5000,
    eval_steps = None,
    train_steps_per_iteration = 500)


#experiment.continuous_train_and_eval()

#The rest come from tutorial
# mnist_classifier.train(
#      input_fn=train_input_fn,
#      steps=1,
#      hooks=[logging_hook])
#
#
# # Evaluate the model and print results
# eval_input_fn = tf.estimator.inputs.numpy_input_fn(
```

```
    #       x={"x": eval_data},
    #       y=eval_labels,
    #       num_epochs=1,
    #       shuffle=False)
    # eval_results = mnist_classifier.evaluate(input_fn=
        eval_input_fn)
    # print(eval_results)


if __name__ == '__main__':
    """Runs whole fitting program automatically"""
    import time

    start_time = time.time()
    tf.app.run()
    print("--- %s seconds ---" % (time.time() - start_time))
```