CS 534 Machine Learning

Project 3, due Sunday, April 15

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 Introduction

Convolutional neural network (CNN) proves to be one of the most successful training models. This time we will use CNN to train MNIST data. MNIST is a database of handwritten figures in black and white. It contains 55000 training images and 10000 test images. In this report, we split 55000 training data into 80% data for training and 20% for validation. We first construct a base model and then

After adjusted some hyper-parameters, we are going to choose the parameters that gives the best validation accuracy and lowest loss.

Then adjustment of hyper-parameters contains two stages. We first construct a base model with certain hyper-parameter.

2 Adjust hyper-parameters I

In this section we first adjust several hyper-parameters listed in the requirement 2.

2.1 Size of the filters

In basis model, size of filters is 5-by-5. According to Stanford University video courses lecture 5, common settings of filter size are listed in the Table **??**.

Table 1: Commonly adjusted filter sizes. Where F is filters' spatial extend (filter dimension), S is the stride, P is the amount of zero padding.

F	S	P
3	1	1
5	1	2
1	1	0

Here we tried three different cases of filter's size: 1*1, 3*3 and 5*5. The plot of accuracy showed in Tensorboard is listed below:

Now we know that the final accuracy after 5000 steps are shown in Table **??**.

Table 2: Accuracy of 5000 steps

Size of filters	1-by-1	3-by-3	5-by-5
Accuracy	0.7676	0.9133	0.9252

2.2 Number of filters

In basis model, filter number is 32 for conv 1 and 64 for conv 2. According to Stanford University video courses lecture 5, common settings of filter number are power of 2: 32, 64, 128 and 512, etc.

Here we tried four different cases of filter?s number (conv1, conv2): (32,32), (32,64), (64,64) and (128,128). The plot of accuracy showed in tensorboard is listed below:
(pictures)
Now we know that the final accuracy after 5000 steps are shown in Table **??**.

Table 3: Validation ccuracy at 5000 steps

Number of filter (conv1, conv2)	(32,32)	(32,64)	(64,64)	(128,128)
Accuracy at 500 step	0.3860	0.4485	0.6018	0.4367
Accuracy at 5000 step	0.9221	0.9133	0.9268	0.9253

From the table, the accuracy increases little with number, the final accuracies are very close. However, from the accuracy plots, we ca see that the initial accuracies at 500 steps have large differences:

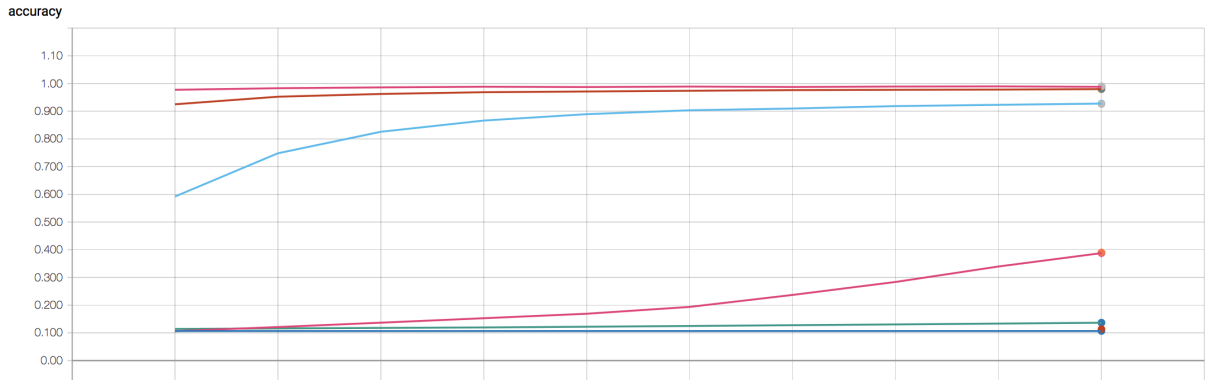

Figure 1: example caption

2.3 Number of convolution and pooling layers

The structure of base model contains 2 convolutional layers and 2 pooling layers, shown in Figure **??**.

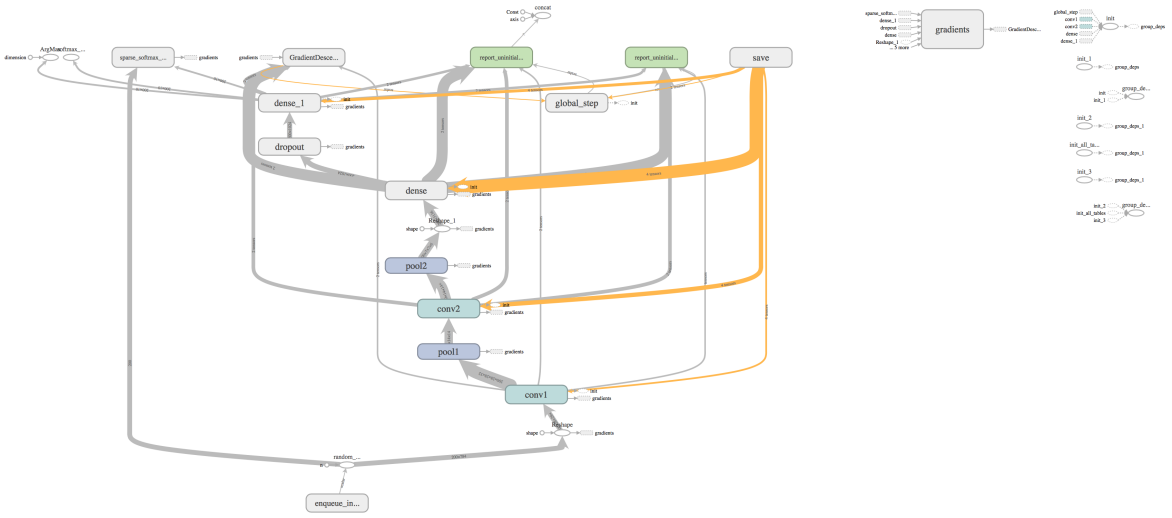

Figure 2: Zoom in to see details.

3 Finally model and summary

4 Source codes

Finally, codes are enclosed here.

```
import tensorflow as tf
import numpy as np


# Parameters
# Default: list all the default variables that need to tune


# Adjust: put the parameter that need to be adjusted
learning_rate = 1E-3

# Save dir:
save_dir = 'tensorboard/learning_rate/lr=1E-3'




#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, name='conv1')

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

# 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, name = 'conv2')

pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2],
    strides=2, name='pool2')

# 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)
    tf.summary.scalar('loss', loss) #Write the loss into
       tensorboard


    # Configure the Training Op (for TRAIN mode)
    if mode == tf.estimator.ModeKeys.TRAIN:
      optimizer = tf.train.GradientDescentOptimizer(learning_rate=
        learning_rate)#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
with tf.name_scope('train_data'):
    train_data = mnist.train.images[0:p] # Returns np.array
    train_labels = np.asarray(mnist.train.labels, dtype=np.
       int32)[0:p]

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

with tf.name_scope('test_data'):
    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=save_dir)
# Set up logging for predictions
tensors_to_log = {"probabilities": "softmax_tensor"}
logging_hook = tf.train.LoggingTensorHook(tensors=
    tensors_to_log, every_n_iter=50)

# # Merge all summaries
# merged_summary = tf.summary.merge_all()
# writer = tf.summary.FileWriter("../../tensorboard/project3/
    pj3-1")
# writer.add_graph(sess.graph)


# 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, #Will run forever
    shuffle=True)

valid_input_fn = tf.estimator.inputs.numpy_input_fn(
```

```
        x={"x": valid_data},
        y=valid_labels,
        num_epochs=1,
        shuffle=False) # Boolean, if True shuffles the queue.
           Avoid shuffle at prediction time.

    experiment = tf.contrib.learn.Experiment(
        mnist_classifier,
        train_input_fn,
        valid_input_fn,
        train_steps = 5000, #This is the step for gradient?
        eval_steps = None,
        train_steps_per_iteration = 500) #Every this step, save
           to ckpt, and evaluate accuracy
    experiment.continuous_train_and_eval()
    # The result of this step is a trained mnist_classifer

    #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))
```