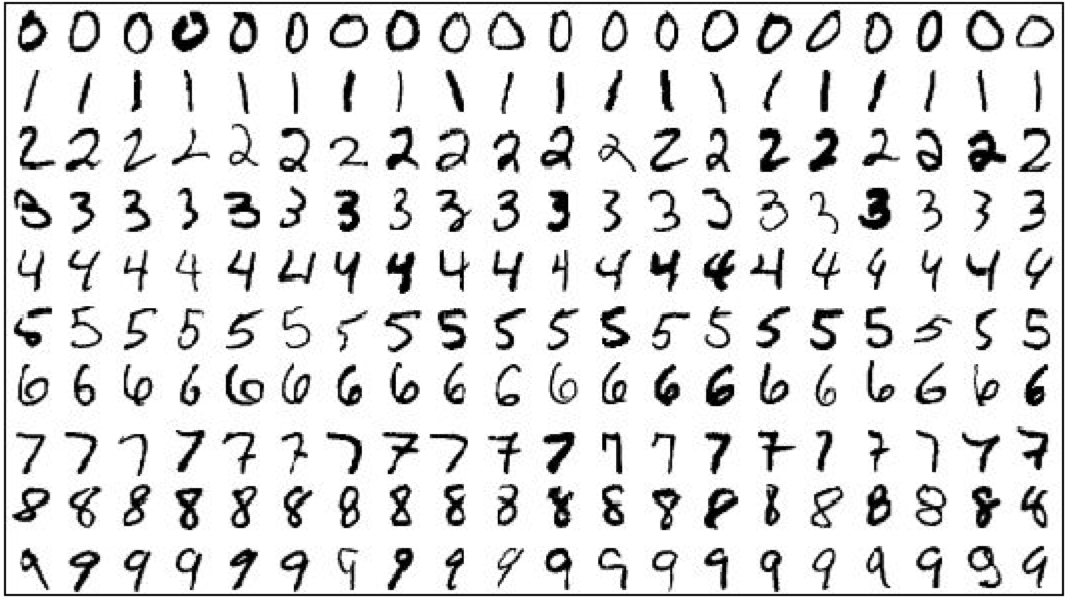
cham & chip

TensorFlow Mechanics 101

How to train a simple neural network to classify the MNIST data set

The MNIST Data Set

train-images-idx3-ubyte



tensorflow / tensorflow

Watch 3,122 | Star 33,143 | Fork 14,432

<> Code | Issues 558 | Pull requests 30 | Projects 0 | Pulse | Graphs

Branch: r0.11 | tensorflow / tensorflow / examples / tutorials / mnist /

Create new file | Upload files | Find file | History

This branch is 37 commits ahead, 298 commits behind master. #4708 Compare

vrv committed with gunan Fix mnist_softmax tutorial to define the InteractiveSession and ... Latest commit 5b18edb 7 days ago

..		
BUILD	Replace use of unstable cross entropy formulation with stable version	9 days ago
__init__.py	Update copyright for 3p/tf.	4 months ago
fully_connected_feed.py	Merge changes from github.	2 months ago
input_data.py	Update copyright for 3p/tf.	4 months ago
mnist.py	Update copyright for 3p/tf.	4 months ago
mnist_softmax.py	Fix mnist_softmax tutorial to define the InteractiveSession and	5 days ago
mnist_with_summaries.py	Replace use of unstable cross entropy formulation with stable version	9 days ago

tensorflow / tensorflow

Watch 3,122 | Star 33,143 | Fork 14,432

<> Code | Issues 558 | Pull requests 30 | Projects 0 | Pulse | Graphs

Branch: r0.11 | tensorflow / tensorflow / examples / tutorials / mnist / | Create new file | Upload files | Find file | History

This branch is 37 commits ahead, 298 commits behind master. #4708 Compare

vrv committed with gunan Fix mnist_softmax tutorial to define the InteractiveSession and ... Latest commit 5b18edb 7 days ago

..

BUILD	Replace use of unstable cross entropy formulation with stable version	9 days ago
__init__.py	Update copyright for 3p/tf.	4 months ago
fully_connected_feed.py	Merge changes from github.	2 months ago
input_data.py	Update copyright for 3p/tf.	4 months ago
mnist.py	Update copyright for 3p/tf.	4 months ago
mnist_softmax.py	Fix mnist_softmax tutorial to define the InteractiveSession and	5 days ago
mnist_with_summaries.py	Replace use of unstable cross entropy formulation with stable version	9 days ago

definitions

tensor - a typed multi-dimensional array that represents the data.
For example, you can represent a mini-batch of images as a 4-D array of floating point numbers with dimensions [batch, height, width, channels].

graph - represents the computations.
To compute anything, a graph must be launched in a Session.

session - executes the graph
A Session places the graph ops onto Devices, such as CPUs or GPUs, and provides methods to execute them.

variable - maintains the state during computation
You typically represent the parameters of a statistical model as a set of variables that you constantly update over the execution period.

preparation

preparation

Declare constants based on the data set

mnist.py

```
# The MNIST dataset has 10 classes, representing the digits 0 through 9.
NUM_CLASSES = 10

# The MNIST images are always 28x28 pixels.
IMAGE_SIZE = 28
IMAGE_PIXELS = IMAGE_SIZE * IMAGE_SIZE
```

prepa-ration

user-defined parameters for the network

fully_connected_feed.py

```
# Basic model parameters as external flags.
flags = tf.app.flags
FLAGS = flags.FLAGS
flags.DEFINE_float('learning_rate', 0.01, 'Initial learning rate.')
flags.DEFINE_integer('max_steps', 2000, 'Number of steps to run trainer.')
flags.DEFINE_integer('hidden1', 128, 'Number of units in hidden layer 1.')
flags.DEFINE_integer('hidden2', 32, 'Number of units in hidden layer 2.')
flags.DEFINE_integer('batch_size', 100, 'Batch size.  '
                     'Must divide evenly into the dataset sizes.')
flags.DEFINE_string('train_dir', 'data', 'Directory to put the training data.')
flags.DEFINE_boolean('fake_data', False, 'If true, uses fake data '
                     'for unit testing.')
```

prepa- ration

a function to create a placeholder for the inputs per batch

tf.placeholder(dtype, shape=None, name=None)

returns a tensor that may be used as a handle for feeding a value.

fully_connected_feed.py

```
def placeholder_inputs(batch_size):
  """Generate placeholder variables to represent the input tensors.

  These placeholders are used as inputs by the rest of the model building
  code and will be fed from the downloaded data in the .run() loop, below.

  Args:
    batch_size: The batch size will be baked into both placeholders.

  Returns:
    images_placeholder: Images placeholder.
    labels_placeholder: Labels placeholder.
  """
  # Note that the shapes of the placeholders match the shapes of the full
  # image and label tensors, except the first dimension is now batch_size
  # rather than the full size of the train or test data sets.
  images_placeholder = tf.placeholder(tf.float32, shape=(batch_size,
                                                          mnist.IMAGE_PIXELS))
  labels_placeholder = tf.placeholder(tf.int32, shape=(batch_size))
  return images_placeholder, labels_placeholder
```

building the model

mnist.py

building the model

inference( ): to create an output of predictions

mnist.py

```
def inference(images, hidden1_units, hidden2_units):
  """Build the MNIST model up to where it may be used for inference.

  Args:
    images: Images placeholder, from inputs().
    hidden1_units: Size of the first hidden layer.
    hidden2_units: Size of the second hidden layer.

  Returns:
    softmax_linear: Output tensor with the computed logits.
  """
```

building the model

inference( ):
2 hidden layers, 1 classification layer

tf.name_scope()
a “context manager” for use in defining the operation

tf.truncated_normal()
outputs random values from a truncated normal distribution.

tf.nn.relu()
computes rectified linear: max(features, 0).

tf.matmul()
matrix multiplication

mnist.py

```
# Hidden 1
with tf.name_scope('hidden1'):
  weights = tf.Variable(
      tf.truncated_normal([IMAGE_PIXELS, hidden1_units],
                          stddev=1.0 / math.sqrt(float(IMAGE_PIXELS))),
      name='weights')
  biases = tf.Variable(tf.zeros([hidden1_units]),
                       name='biases')
  hidden1 = tf.nn.relu(tf.matmul(images, weights) + biases)
# Hidden 2
with tf.name_scope('hidden2'):
  weights = tf.Variable(
      tf.truncated_normal([hidden1_units, hidden2_units],
                          stddev=1.0 / math.sqrt(float(hidden1_units))),
      name='weights')
  biases = tf.Variable(tf.zeros([hidden2_units]),
                       name='biases')
  hidden2 = tf.nn.relu(tf.matmul(hidden1, weights) + biases)
# Linear
with tf.name_scope('softmax_linear'):
  weights = tf.Variable(
      tf.truncated_normal([hidden2_units, NUM_CLASSES],
                          stddev=1.0 / math.sqrt(float(hidden2_units))),
      name='weights')
  biases = tf.Variable(tf.zeros([NUM_CLASSES]),
                       name='biases')
  logits = tf.matmul(hidden2, weights) + biases
return logits
```

building the model

loss( ): adds loss function to the model

tf.nn.sparse_softmax_cross_entropy_with_logits()
measures the probability error in discrete classification tasks in which the classes are mutually exclusive

tf.nn.reduce_mean()
computes the mean of elements across dimensions of a tensor.

mnist.py

```
def loss(logits, labels):
  """Calculates the loss from the logits and the labels.

  Args:
    logits: Logits tensor, float - [batch_size, NUM_CLASSES].
    labels: Labels tensor, int32 - [batch_size].

  Returns:
    loss: Loss tensor of type float.
  """
  labels = tf.to_int64(labels)
  cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(
      logits, labels, name='xentropy')
  loss = tf.reduce_mean(cross_entropy, name='xentropy_mean')
  return loss
```

building the model

training( ): minimizes the loss via gradient descent

tf.scalar_summary()
Returns a string tensor given the tag and the values.

tf.train.GradientDescentOptimizer()
creates an optimizer given the learning rate.

optimizer.minimize()
minimizes the loss/cost by updating a list of variables

mnist.py

```python
def training(loss, learning_rate):
  """Sets up the training Ops.

  Creates a summarizer to track the loss over time in TensorBoard.

  Creates an optimizer and applies the gradients to all trainable variables.

  The Op returned by this function is what must be passed to the
  `sess.run()` call to cause the model to train.

  Args:
    loss: Loss tensor, from loss().
    learning_rate: The learning rate to use for gradient descent.

  Returns:
    train_op: The Op for training.
  """
  # Add a scalar summary for the snapshot loss.
  tf.scalar_summary(loss.op.name, loss)
  # Create the gradient descent optimizer with the given learning rate.
  optimizer = tf.train.GradientDescentOptimizer(learning_rate)
  # Create a variable to track the global step.
  global_step = tf.Variable(0, name='global_step', trainable=False)
  # Use the optimizer to apply the gradients that minimize the loss
  # (and also increment the global step counter) as a single training step.
  train_op = optimizer.minimize(loss, global_step=global_step)
  return train_op
```

building the model

evaluation( ): returns correct predictions

tf.nn.in_top_k()
Returns a boolean tensor, true if the target is in the top k predictions.

tf.cast()
typecasts a tensor to a new type

tf.reduce_sum()
reduces the dimensions of a tensor

mnist.py

```
def evaluation(logits, labels):
  """Evaluate the quality of the logits at predicting the label.

  Args:
    logits: Logits tensor, float - [batch_size, NUM_CLASSES].
    labels: Labels tensor, int32 - [batch_size], with values in the
      range [0, NUM_CLASSES).

  Returns:
    A scalar int32 tensor with the number of examples (out of batch_size)
    that were predicted correctly.
  """
  # For a classifier model, we can use the in_top_k Op.
  # It returns a bool tensor with shape [batch_size] that is true for
  # the examples where the label is in the top k (here k=1)
  # of all logits for that example.
  correct = tf.nn.in_top_k(logits, labels, 1)
  # Return the number of true entries.
  return tf.reduce_sum(tf.cast(correct, tf.int32))
```

building the model

fill_feed_dict( ): creates a dictionary to feed images and labels for every step

```python
def fill_feed_dict(data_set, images_pl, labels_pl):
  """Fills the feed_dict for training the given step.

  A feed_dict takes the form of:
  feed_dict = {
      <placeholder>: <tensor of values to be passed for placeholder>,
      ....
  }

  Args:
    data_set: The set of images and labels, from input_data.read_data_sets()
    images_pl: The images placeholder, from placeholder_inputs().
    labels_pl: The labels placeholder, from placeholder_inputs().

  Returns:
    feed_dict: The feed dictionary mapping from placeholders to values.
  """
  # Create the feed_dict for the placeholders filled with the next
  # `batch size` examples.
  images_feed, labels_feed = data_set.next_batch(FLAGS.batch_size,
                                                 FLAGS.fake_data)
  feed_dict = {
      images_pl: images_feed,
      labels_pl: labels_feed,
  }
  return feed_dict
```

fully_connectd_feed.py

training the model

fully_connected_feed.py

training the model

run_training( ): train the model using the MNIST data set

fully_connected_feed.py

```
def run_training():
  """Train MNIST for a number of steps."""
  # Get the sets of images and labels for training, validation, and
  # test on MNIST.
  data_sets = input_data.read_data_sets(FLAGS.train_dir, FLAGS.fake_data)
  # Tell TensorFlow that the model will be built into the default Graph.
  with tf.Graph().as_default():
    # Generate placeholders for the images and labels.
    images_placeholder, labels_placeholder = placeholder_inputs(
        FLAGS.batch_size)

    # Build a Graph that computes predictions from the inference model.
    logits = mnist.inference(images_placeholder,
                             FLAGS.hidden1,
                             FLAGS.hidden2)

    # Add to the Graph the Ops for loss calculation.
    loss = mnist.loss(logits, labels_placeholder)

    # Add to the Graph the Ops that calculate and apply gradients.
    train_op = mnist.training(loss, FLAGS.learning_rate)
```

training the model

run_training( ): declare variables for the result of training, summary, checkpoint, and the session of running the model over the data.

tf.train.Saver()
saves and restores the variables

tf.train.SummaryWriter()
creates an event file in a given directory and add summaries to it

fully_connected_feed.py

```
    # Add the Op to compare the logits to the labels during evaluation.
    eval_correct = mnist.evaluation(logits, labels_placeholder)

    # Build the summary Tensor based on the TF collection of Summaries.
    summary = tf.merge_all_summaries()

    # Add the variable initializer Op.
    init = tf.initialize_all_variables()

    # Create a saver for writing training checkpoints.
    saver = tf.train.Saver()

    # Create a session for running Ops on the Graph.
    sess = tf.Session()

    # Instantiate a SummaryWriter to output summaries and the Graph.
    summary_writer = tf.train.SummaryWriter(FLAGS.train_dir, sess.graph)
```

training the model

run_training( ): the loop

sess.run()
executes given operation and returns the output

fully_connected_feed.py

```
# Run the Op to initialize the variables.
sess.run(init)

# Start the training loop.
for step in xrange(FLAGS.max_steps):
  start_time = time.time()

  # Fill a feed dictionary with the actual set of images and labels
  # for this particular training step.
  feed_dict = fill_feed_dict(data_sets.train,
                             images_placeholder,
                             labels_placeholder)

  # Run one step of the model.  The return values are the activations
  # from the `train_op` (which is discarded) and the `loss` Op.  To
  # inspect the values of your Ops or variables, you may include them
  # in the list passed to sess.run() and the value tensors will be
  # returned in the tuple from the call.
  _, loss_value = sess.run([train_op, loss],
                           feed_dict=feed_dict)

  duration = time.time() - start_time
```

training the model

run_training( ): printing and creating a summary

fully_connected_feed.py

```
      # Write the summaries and print an overview fairly often.
      if step % 100 == 0:
        # Print status to stdout.
        print('Step %d: loss = %.2f (%.3f sec)' % (step, loss_value, duration))
        # Update the events file.
        summary_str = sess.run(summary, feed_dict=feed_dict)
        summary_writer.add_summary(summary_str, step)
        summary_writer.flush()
```

training the model

run_training( ): creating a checkpoint and evaluating on training, validation, and test data sets.

```
# Save a checkpoint and evaluate the model periodically.
if (step + 1) % 1000 == 0 or (step + 1) == FLAGS.max_steps:
  checkpoint_file = os.path.join(FLAGS.train_dir, 'checkpoint')
  saver.save(sess, checkpoint_file, global_step=step)
  # Evaluate against the training set.
  print('Training Data Eval:')
  do_eval(sess,
          eval_correct,
          images_placeholder,
          labels_placeholder,
          data_sets.train)
  # Evaluate against the validation set.
  print('Validation Data Eval:')
  do_eval(sess,
          eval_correct,
          images_placeholder,
          labels_placeholder,
          data_sets.validation)
  # Evaluate against the test set.
  print('Test Data Eval:')
  do_eval(sess,
          eval_correct,
          images_placeholder,
          labels_placeholder,
          data_sets.test)
```

fully_connected_feed.py

```
(tensorflow) Chips-MacBook-Pro:mnist Chippy$ python fully_connected_feed.py
```

```
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting data/train-images-idx3-ubyte.gz
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting data/train-labels-idx1-ubyte.gz
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting data/t10k-labels-idx1-ubyte.gz
```

```
Step 0: loss = 2.30 (0.324 sec)
Step 100: loss = 2.20 (0.004 sec)
Step 200: loss = 1.97 (0.005 sec)
Step 300: loss = 1.70 (0.004 sec)
Step 400: loss = 1.42 (0.004 sec)
Step 500: loss = 1.01 (0.004 sec)
Step 600: loss = 0.83 (0.003 sec)
Step 700: loss = 0.70 (0.005 sec)
Step 800: loss = 0.78 (0.005 sec)
Step 900: loss = 0.47 (0.004 sec)
Training Data Eval:
  Num examples: 55000  Num correct: 46915  Precision @ 1: 0.8530
Validation Data Eval:
  Num examples: 5000  Num correct: 4315  Precision @ 1: 0.8630
Test Data Eval:
  Num examples: 10000  Num correct: 8605  Precision @ 1: 0.8605
```

```
Step 1000: loss = 0.72 (0.014 sec)
Step 1100: loss = 0.51 (0.113 sec)
Step 1200: loss = 0.48 (0.003 sec)
Step 1300: loss = 0.44 (0.003 sec)
Step 1400: loss = 0.37 (0.003 sec)
Step 1500: loss = 0.40 (0.004 sec)
Step 1600: loss = 0.31 (0.003 sec)
Step 1700: loss = 0.26 (0.004 sec)
Step 1800: loss = 0.33 (0.004 sec)
Step 1900: loss = 0.48 (0.003 sec)
Training Data Eval:
  Num examples: 55000  Num correct: 49178  Precision @ 1: 0.8941
Validation Data Eval:
  Num examples: 5000  Num correct: 4500  Precision @ 1: 0.9000
Test Data Eval:
  Num examples: 10000  Num correct: 8975  Precision @ 1: 0.8975
```