Basic Models in TensorFlow

CS 20: TensorFlow for Deep Learning Research
Lecture 3
1/19/2017

Agenda

Review

Linear regression on birth/life data

Control Flow

tf.data

Optimizers, gradients

Logistic regression on MNIST

Loss functions

Review

Computation graph

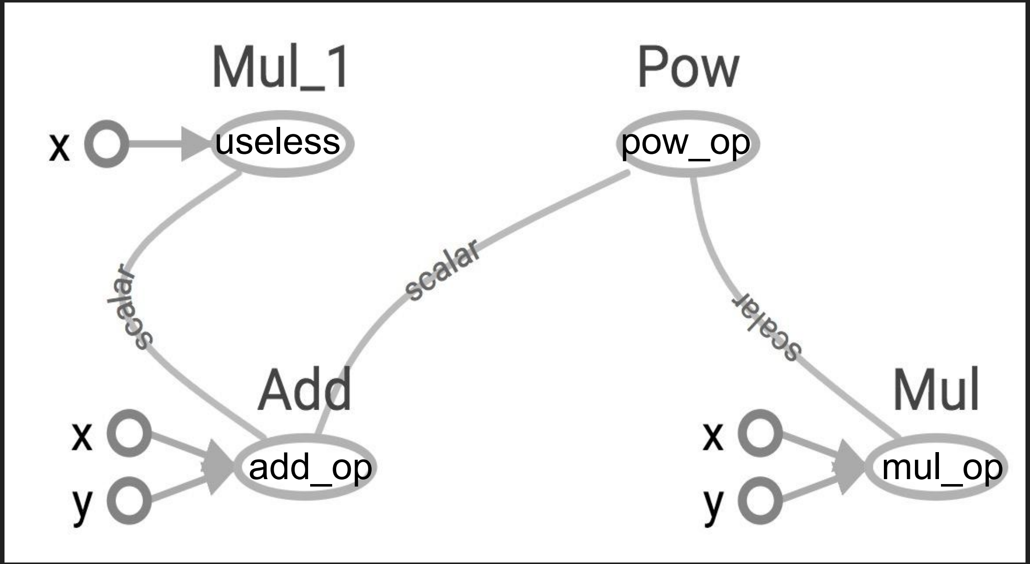
TensorFlow separates definition of computations from their execution

Phase 1: assemble a graph

Phase 2: use a session to execute operations in the graph.

TensorBoard

```
x = 2
y = 3
add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
useless = tf.multiply(x, add_op)
pow_op = tf.pow(add_op, mul_op)
with tf.Session() as sess:
      z = sess.run(pow_op)
```



Create a FileWriter object to write your graph to event files

tf.constant and tf.Variable

Constant values are stored in the graph definition

Sessions allocate memory to store variable values

tf.placeholder and feed_dict

Feed values into placeholders with a dictionary (feed_dict)

Easy to use but poor performance

Avoid lazy loading

1. Separate the assembling of graph and executing ops
2. Use Python attribute to ensure a function is only loaded the first time it's called

Download from the class's GitHub

examples/`03_linreg_starter.py`

examples/`03_logreg_starter.py`

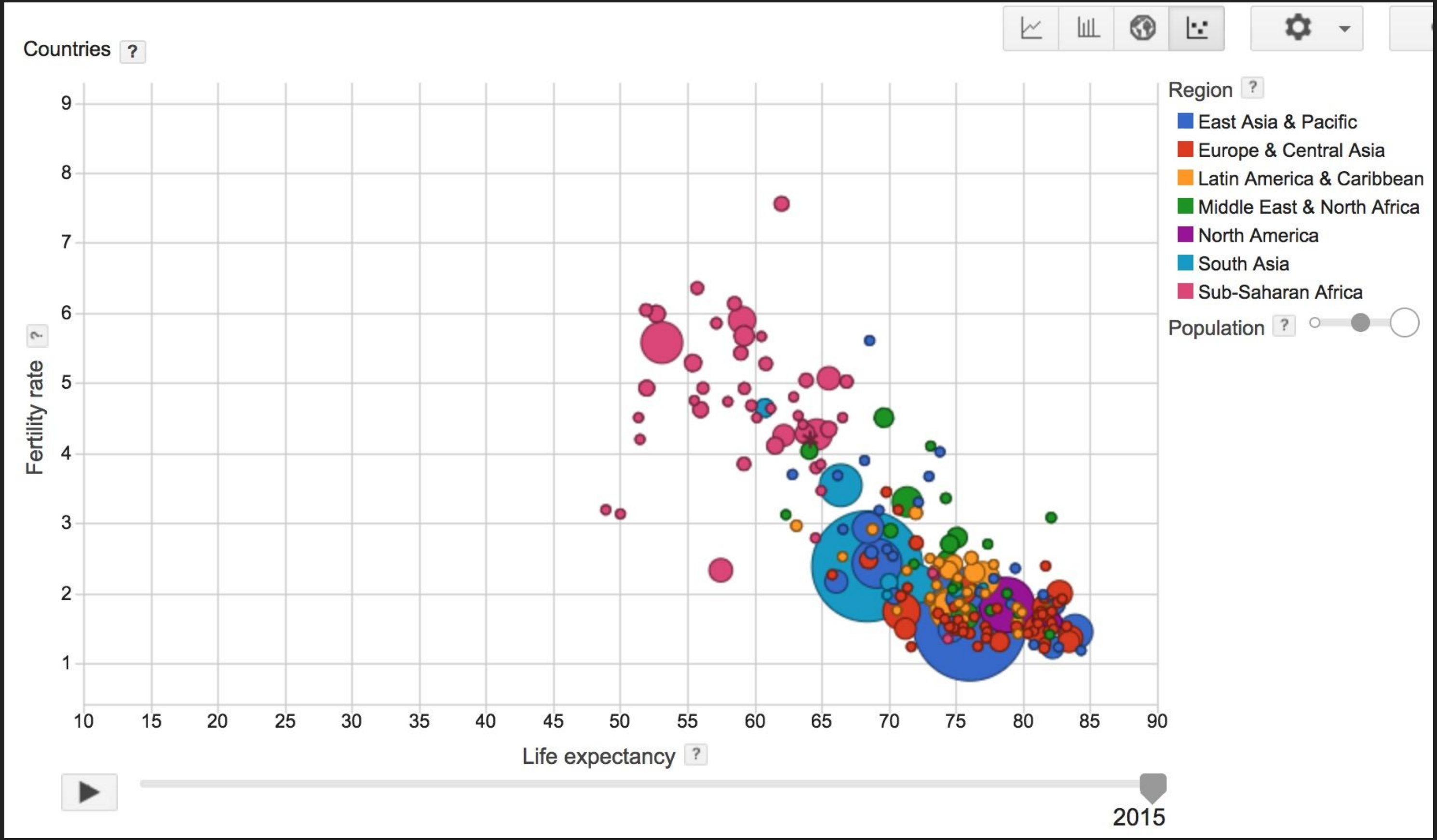
examples/`utils.py`

data/`birth_life_2010.txt`

Linear Regression in TensorFlow

Model the linear relationship between:

- dependent variable Y
- explanatory variables X



Visualization made by Google, based on data collected by World Bank

World Development Indicators dataset

X: birth rate
Y: life expectancy
190 countries

Want

Find a linear relationship between X and Y
to predict Y from X

Model

```
Inference: Y_predicted = w * X + b
```

Mean squared error: $E[(y - y\_predicted)^2]$

Interactive Coding

```
data/birth_life_2010.txt
```

Interactive Coding

`examples/03_linreg_starter.py`

Phase 1: Assemble our graph

Step 1: Read in data

I already did that for you

Step 2: Create placeholders for inputs and labels

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

Step 3: Create weight and bias

```
tf.get_variable(
    name,
    shape=None,
    dtype=None,
    initializer=None,
    ...
)
```

No need to specify shape if using constant initializer

Step 4: Inference

```
Y_predicted = w * X + b
```

Step 5: Specify loss function

```
loss = tf.square(Y - Y_predicted, name='loss')
```

Step 6: Create optimizer

```
opt = tf.train.GradientDescentOptimizer(learning_rate=0.001)

optimizer = opt.minimize(loss)
```

Phase 2: Train our model

Step 1: Initialize variables

Step 2: Run optimizer

(use a feed_dict to feed data into X and Y placeholders)

Write log files using a FileWriter

```
writer = tf.summary.FileWriter('./graphs/linear_reg', sess.graph)
```

See it on TensorBoard

```
Step 1: $ python3 03_linreg_starter.py

Step 2: $ tensorboard --logdir='./graphs'
```

TypeError?

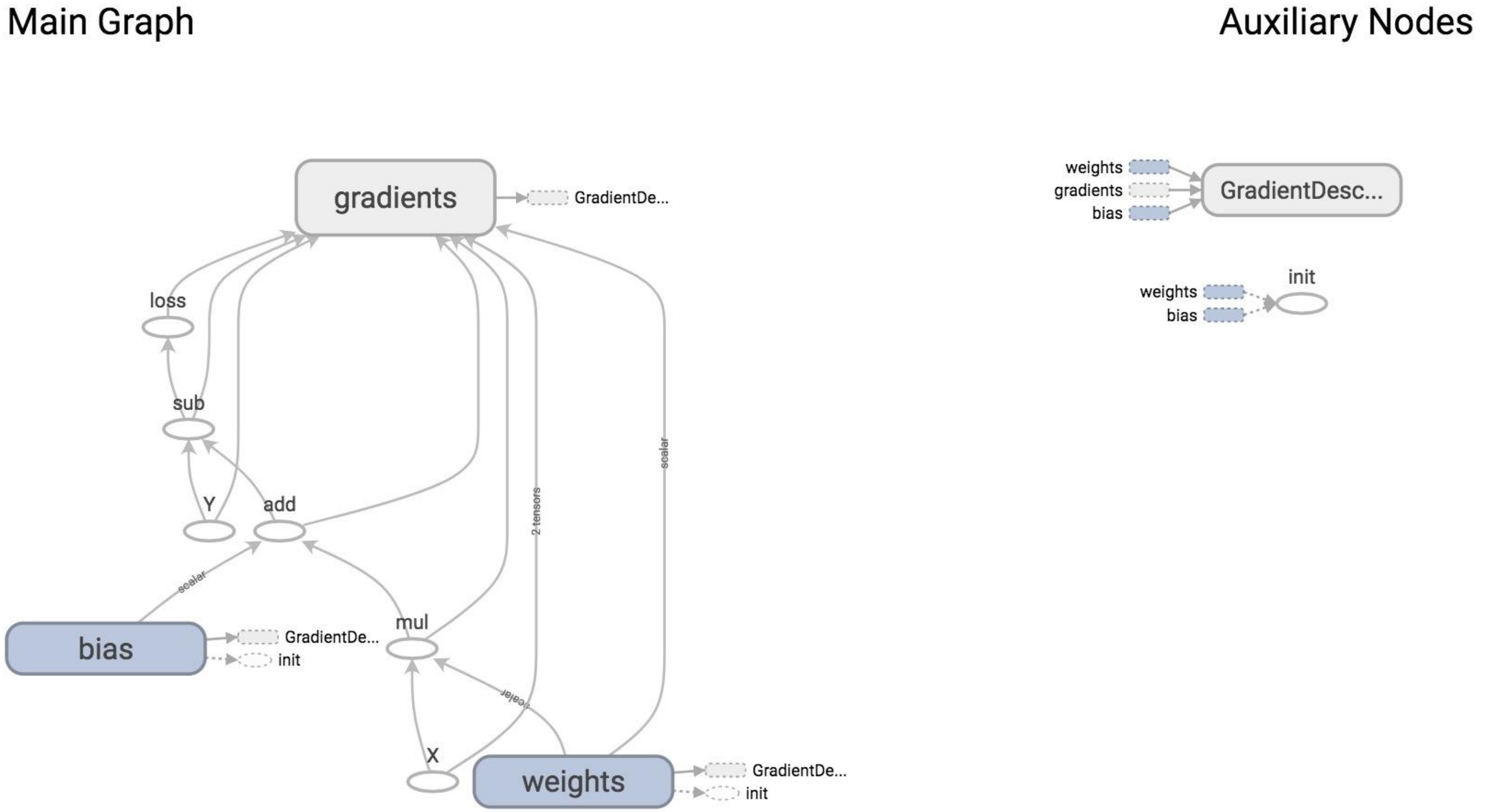
```
TypeError: Fetch argument 841.0 has invalid type <class 'numpy.float32'>,
must be a string or Tensor.

(Can not convert a float32 into a Tensor or Operation.)
```

TypeError

```
for i in range(50): # train the model 100 epochs
    total_loss = 0
    for x, y in data:
        _, loss = sess.run([optimizer, loss], feed_dict={X: x, Y:y}) # Can’t fetch a numpy array
        total_loss += loss
```

TypeError

```
for i in range(50): # train the model 100 epochs
      total_loss = 0
      for x, y in data:
            _, loss_ = sess.run([optimizer, loss], feed_dict={X: x, Y:y})
            total_loss += loss_
```



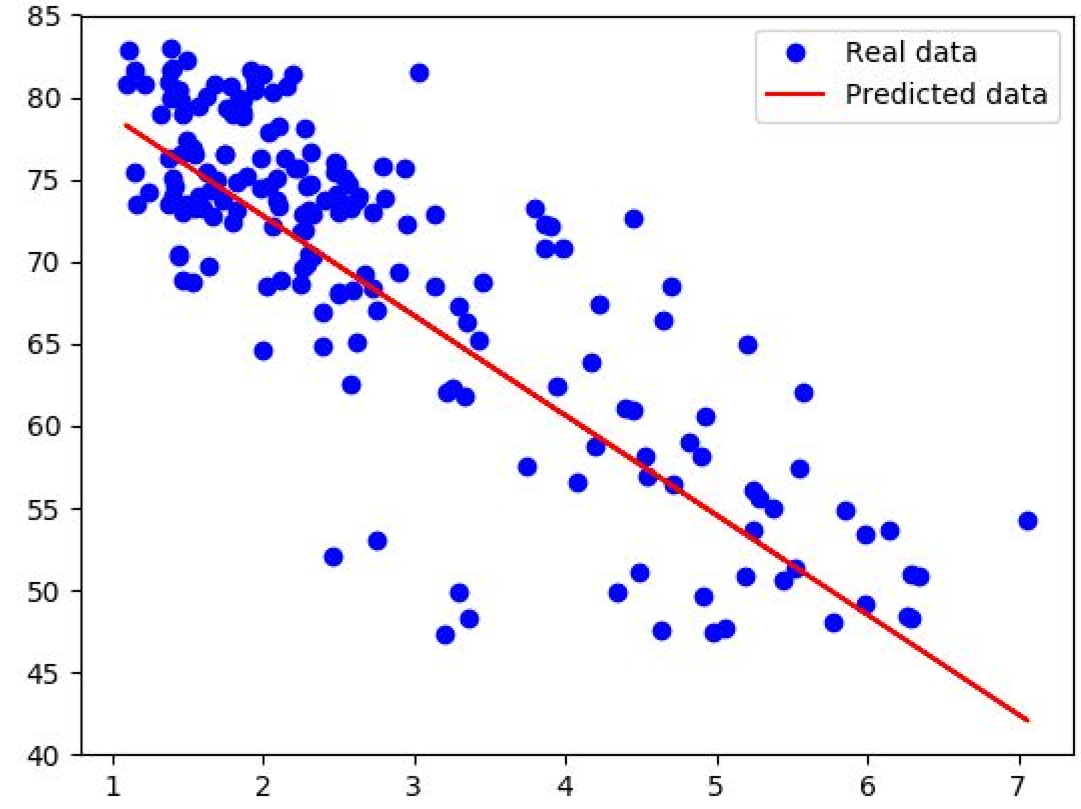
Main Graph
Auxiliary Nodes
gradients
GradientDe...
loss
sub
Y
add
scalar
bias
GradientDe...
init
mul
scalar
X
weights
GradientDe...
init
2 tensors
scalar
weights
gradients
bias
GradientDesc...
init
weights
bias

Plot the results with matplotlib

Step 1: Uncomment the plotting code at the end of your program

Step 2: Run it again

If run into problem of matplotlib in virtual environment, go to GitHub/setup and see the file possible setup problems



Real data
Predicted data
85
80
75
70
65
60
55
50
45
40
1
2
3
4
5
6
7

Huber loss

Robust to outliers

If the difference between the predicted value and the real value is small, square it

If it's large, take its absolute value

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Implementing Huber loss

Can’t write:

```
if y - y_predicted < delta:
```

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

You can write it if eager mode were enabled. Stay tuned for the next lecture!

Implementing Huber loss

```
tf.cond(pred, fn1, fn2, name=None)
```

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\, |y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Implementing Huber loss

```
tf.cond(pred, fn1, fn2, name=None)
```

```
def huber_loss(labels, predictions, delta=14.0):
    residual = tf.abs(labels - predictions)
    def f1(): return 0.5 * tf.square(residual)
    def f2(): return delta * residual - 0.5 * tf.square(delta)
    return tf.cond(residual < delta, f1, f2)
```

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

TF Control Flow

Control Flow Ops	tf.group, tf.count_up_to, tf.cond, tf.case, tf.while_loop, ...
Comparison Ops	tf.equal, tf.not_equal, tf.less, tf.greater, tf.where, ...
Logical Ops	tf.logical_and, tf.logical_not, tf.logical_or, tf.logical_xor
Debugging Ops	tf.is_finite, tf.is_inf, tf.is_nan, tf.Assert, tf.Print, ...

Since TF builds graph before computation, we have to specify all possible subgraphs beforehand. PyTorch's dynamic graphs and TF's eager execution help overcome this

tf.data

Placeholder

Pro: put the data processing outside TensorFlow, making it easy to do in Python

Cons: users often end up processing their data in a single thread and creating data bottleneck that slows execution down.

Placeholder

```
data, n_samples = utils.read_birth_life_data(DATA_FILE)

X = tf.placeholder(tf.float32, name='X')
Y = tf.placeholder(tf.float32, name='Y')
…
with tf.Session() as sess:
       …
     # Step 8: train the model
     for i in range(100): # run 100 epochs
          for x, y in data:
               # Session runs train_op to minimize loss
               sess.run(optimizer, feed_dict={X: x, Y:y})
```

tf.data

Instead of doing inference with placeholders and feeding in data later, do inference directly with data

tf.data

`tf.data.Dataset`

`tf.data.Iterator`

Store data in tf.data.Dataset

- `tf.data.Dataset.from_tensor_slices((features, labels))`
- `tf.data.Dataset.from_generator(gen, output_types, output_shapes)`

Store data in tf.data.Dataset

```
tf.data.Dataset.from_tensor_slices((features, labels))

dataset = tf.data.Dataset.from_tensor_slices((data[:,0], data[:,1]))
```

Store data in tf.data.Dataset

```
tf.data.Dataset.from_tensor_slices((features, labels))

dataset = tf.data.Dataset.from_tensor_slices((data[:,0], data[:,1]))

print(dataset.output_types)      # >> (tf.float32, tf.float32)

print(dataset.output_shapes)     # >> (TensorShape([]), TensorShape([]))
```

Can also create Dataset from files

- `tf.data.TextLineDataset(filenames)`
- `tf.data.FixedLengthRecordDataset(filenames)`
- `tf.data.TFRecordDataset(filenames)`

tf.data.Iterator

Create an iterator to iterate through samples in Dataset

tf.data.Iterator

- `iterator = dataset.make_one_shot_iterator()`
- `iterator = dataset.make_initializable_iterator()`

tf.data.Iterator

- `iterator = dataset.make_one_shot_iterator()`
  Iterates through the dataset exactly once. No need to initialization.
- `iterator = dataset.make_initializable_iterator()`
  Iterates through the dataset as many times as we want. Need to initialize with each epoch.

tf.data.Iterator

```
iterator = dataset.make_one_shot_iterator()
X, Y = iterator.get_next()         # X is the birth rate, Y is the life expectancy

with tf.Session() as sess:
      print(sess.run([X, Y]))          # >> [1.822, 74.82825]
      print(sess.run([X, Y]))          # >> [3.869, 70.81949]
      print(sess.run([X, Y]))          # >> [3.911, 72.15066]
```

tf.data.Iterator

```
iterator = dataset.make_initializable_iterator()

...

for i in range(100):
        sess.run(iterator.initializer)
        total_loss = 0
        try:
            while True:
                sess.run([optimizer])
        except tf.errors.OutOfRangeError:
            pass
```

Handling data in TensorFlow

```
dataset = dataset.shuffle(1000)

dataset = dataset.repeat(100)

dataset = dataset.batch(128)

dataset = dataset.map(lambda x: tf.one_hot(x, 10))
# convert each elem of dataset to one_hot vector
```

Does tf.data really perform better?

Does tf.data really perform better?

With placeholder: 9.05271519 seconds

With tf.data: 6.12285947 seconds

Should we always use tf.data?

- For prototyping, feed dict can be faster and easier to write (pythonic)
- tf.data is tricky to use when you have complicated preprocessing or multiple data sources
- NLP data is normally just a sequence of integers. In this case, transferring the data over to GPU is pretty quick, so the speedup of tf.data isn't that large

How does TensorFlow know what variables to update?

Optimizers

Optimizer

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(loss)

_, l = sess.run([optimizer, loss], feed_dict={X: x, Y:y})
```

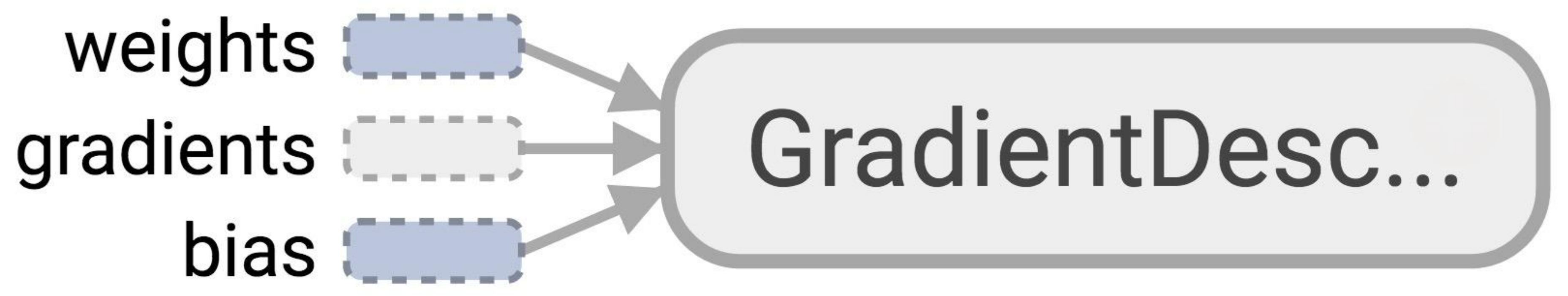
Optimizer

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001).minimize(loss)

_, l = sess.run([optimizer, loss], feed_dict={X: x, Y:y})
```

Session looks at all trainable variables that loss depends on and update them

Optimizer

Session looks at all trainable variables that optimizer depends on and update them



Trainable variables

```
tf.Variable(initial_value=None, trainable=True,...)
```

Specify if a variable should be trained or not
By default, all variables are trainable

List of optimizers in TF

```
tf.train.GradientDescentOptimizer

tf.train.AdagradOptimizer

tf.train.MomentumOptimizer

tf.train.AdamOptimizer

tf.train.FtrlOptimizer

tf.train.RMSPropOptimizer

...
```

“Advanced” optimizers work better when tuned, but are generally harder to tune

Discussion question

1. How to know that our model is correct?
2. How to improve our model?

Assignment 1

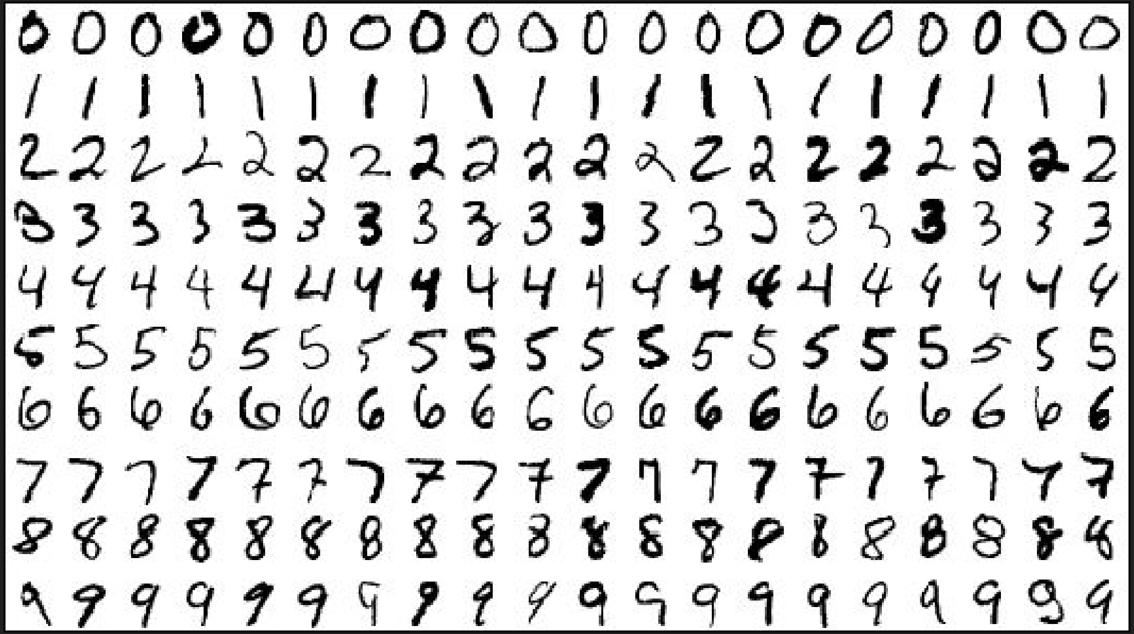
Out tomorrow
Due 1/31
Optional Interactive Grading

Logistic Regression in TensorFlow


Then he separated the light from the darkness
The first logistic regression model

MNIST Database

Each image is a 28x28 array, flattened out to be a 1-d tensor of size 784



MNIST

X: image of a handwritten digit
Y: the digit value
Recognize the digit in the image

MNIST

X: image of a handwritten digit
Y: the digit value

Model

```
Inference: Y_predicted = softmax(X * w + b)

Cross entropy loss: -log(Y_predicted)
```

Process data

```
from tensorflow.examples.tutorials.mnist import input_data
MNIST = input_data.read_data_sets('data/mnist', one_hot=True)
```

Process data

```
from tensorflow.examples.tutorials.mnist import input_data
MNIST = input_data.read_data_sets('data/mnist', one_hot=True)

MNIST.train: 55,000 examples
MNIST.validation: 5,000 examples
MNIST.test: 10,000 examples
```

Process data

```
from tensorflow.examples.tutorials.mnist import input_data
MNIST = input_data.read_data_sets('data/mnist', one_hot=True)

MNIST.train: 55,000 examples
MNIST.validation: 5,000 examples
MNIST.test: 10,000 examples
```

No immediate way to convert Python generators to tf.data.Dataset

Process data

```
mnist_folder = 'data/mnist'
utils.download_mnist(mnist_folder)
train, val, test = utils.read_mnist(mnist_folder, flatten=True)
```

Create datasets

```
mnist_folder = 'data/mnist'
utils.download_mnist(mnist_folder)
train, val, test = utils.read_mnist(mnist_folder, flatten=True)

train_data = tf.data.Dataset.from_tensor_slices(train)
train_data = train_data.shuffle(10000) # optional
test_data = tf.data.Dataset.from_tensor_slices(test)
```

Create iterator

```
mnist_folder = 'data/mnist'
utils.download_mnist(mnist_folder)
train, val, test = utils.read_mnist(mnist_folder, flatten=True)

train_data = tf.data.Dataset.from_tensor_slices(train)
train_data = train_data.shuffle(10000) # optional
test_data = tf.data.Dataset.from_tensor_slices(test)

iterator = train_data.make_initializable_iterator()
```

Create iterator

```
mnist_folder = 'data/mnist'
utils.download_mnist(mnist_folder)
train, val, test = utils.read_mnist(mnist_folder, flatten=True)

train_data = tf.data.Dataset.from_tensor_slices(train)
train_data = train_data.shuffle(10000) # optional
test_data = tf.data.Dataset.from_tensor_slices(test)

iterator = train_data.make_initializable_iterator()
img, label = iterator.get_next()
…
```

Problem?

Create iterator

```
mnist_folder = 'data/mnist'
utils.download_mnist(mnist_folder)
train, val, test = utils.read_mnist(mnist_folder, flatten=True)

train_data = tf.data.Dataset.from_tensor_slices(train)
train_data = train_data.shuffle(10000) # optional
test_data = tf.data.Dataset.from_tensor_slices(test)

iterator = train_data.make_initializable_iterator()
img, label = iterator.get_next()
…

>> Can only do inference with train_data.
>> Need to build another subgraph with another iterator for test_data!!!
```

Create iterator

```
mnist_folder = 'data/mnist'
utils.download_mnist(mnist_folder)
train, val, test = utils.read_mnist(mnist_folder, flatten=True)

train_data = tf.data.Dataset.from_tensor_slices(train)
train_data = train_data.shuffle(10000) # optional
test_data = tf.data.Dataset.from_tensor_slices(test)

iterator = tf.data.Iterator.from_structure(train_data.output_types,
                                           train_data.output_shapes)
img, label = iterator.get_next()

train_init = iterator.make_initializer(train_data)  # initializer for train_data
test_init = iterator.make_initializer(test_data)    # initializer for train_data
```

Initialize iterator with the dataset you want

```
with tf.Session() as sess:
    ...
    for i in range(n_epochs):
        sess.run(train_init)                # use train_init during training loop
        try:
            while True:
                _, l = sess.run([optimizer, loss])
        except tf.errors.OutOfRangeError:
            pass
```

Initialize iterator with the dataset you want

```
with tf.Session() as sess:
    ...
    for i in range(n_epochs):
        sess.run(train_init)
        try:
            while True:
                _, l = sess.run([optimizer, loss])
        except tf.errors.OutOfRangeError:
            pass

    # test the model
    sess.run(test_init)                    # use test_init during testing
    try:
        while True:
            sess.run(accuracy)
    except tf.errors.OutOfRangeError:
        pass
```

Phase 1: Assemble our graph

Step 1: Read in data

```
I already did that for you
```

Step 2: Create datasets and iterator

```
train_data = tf.data.Dataset.from_tensor_slices(train)
train_data = train_data.shuffle(10000) # optional
train_data = train_data.batch(batch_size)

test_data = tf.data.Dataset.from_tensor_slices(test)
test_data = test_data.batch(batch_size)
```

Step 2: Create datasets and iterator

```
iterator =
tf.data.Iterator.from_structure(train_data.output_types,
                                  train_data.output_shapes)
img, label = iterator.get_next()

train_init = iterator.make_initializer(train_data)
test_init = iterator.make_initializer(test_data)
```

Step 3: Create weights and biases

use `tf.get_variable()`

Step 4: Build model to predict Y

```
logits = tf.matmul(img, w) + b
```

We don’t do softmax here, as we’ll do softmax together with cross_entropy loss.
It’s more efficient to compute gradients w.r.t. logits than w.r.t. softmax

Step 5: Specify loss function

```
entropy = tf.nn.softmax_cross_entropy_with_logits(labels=label,
                                    logits=logits)

loss = tf.reduce_mean(entropy)
```

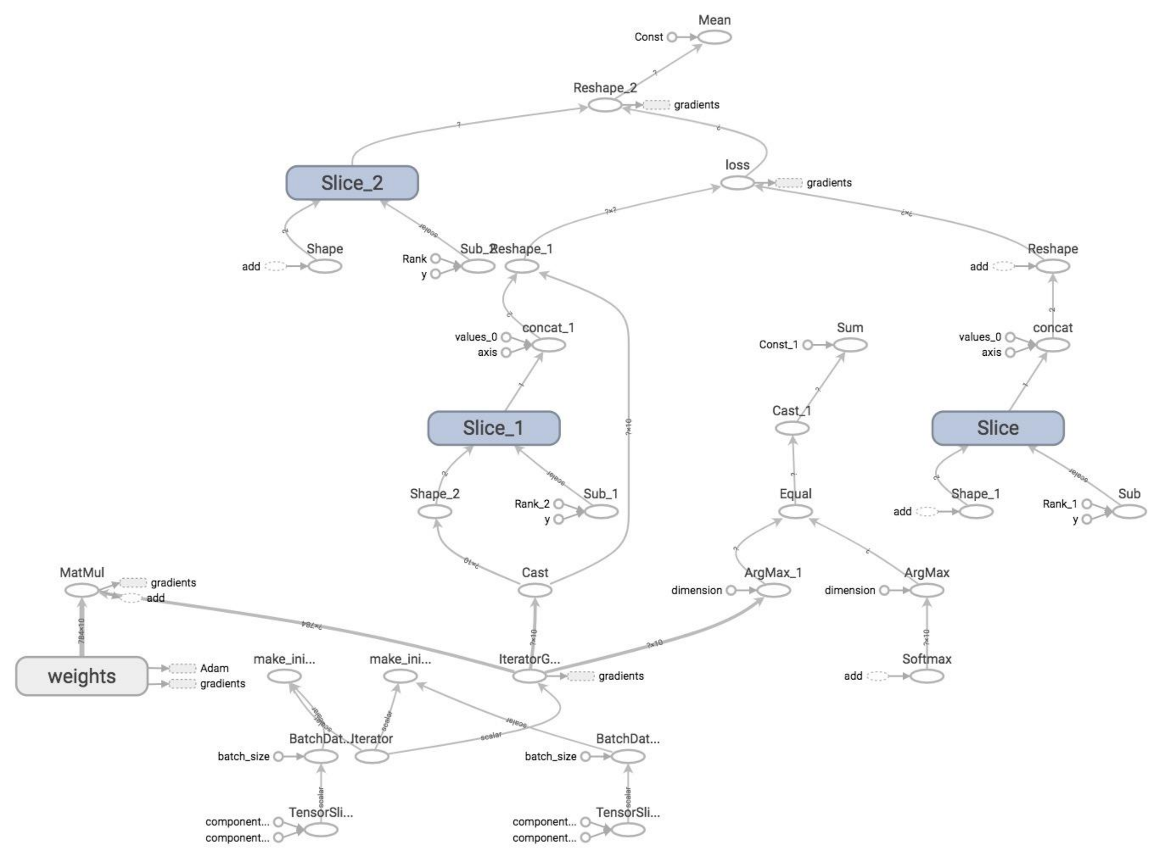
Step 6: Create optimizer

```
tf.train.AdamOptimizer(learning_rate=0.01).minimize(loss)
```

Phase 2: Train our model

```
Step 1: Initialize variables

Step 2: Run optimizer op
```

TensorBoard it



Next class

Structure your model in TensorFlow

Example: word2vec

Eager execution

Feedback: huyenn@stanford.edu

Thanks!