Tensor Flow

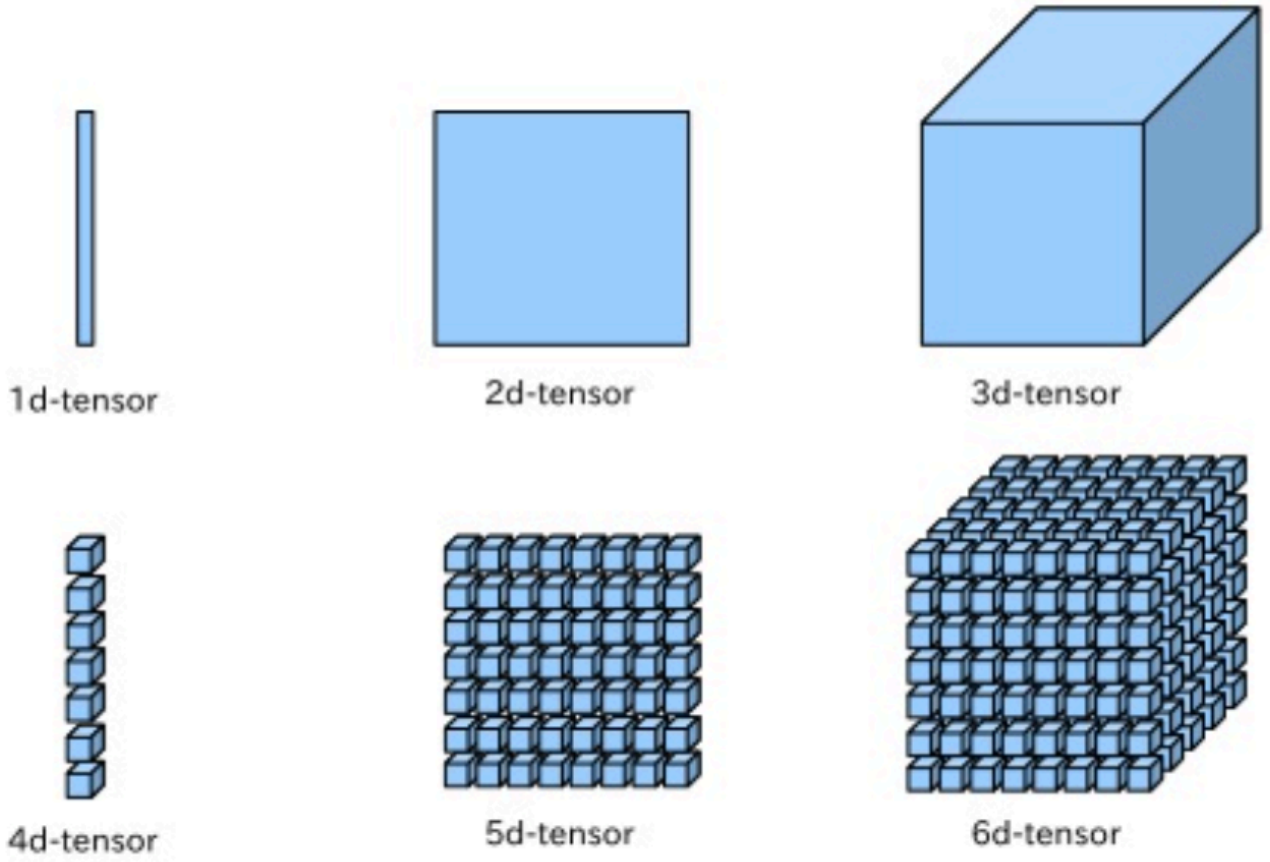
https://www.tensorflow.org

Yasmine Badr

1/19/2016

What is a Tensor?

- Generalization of scalar, vector, matrix,…

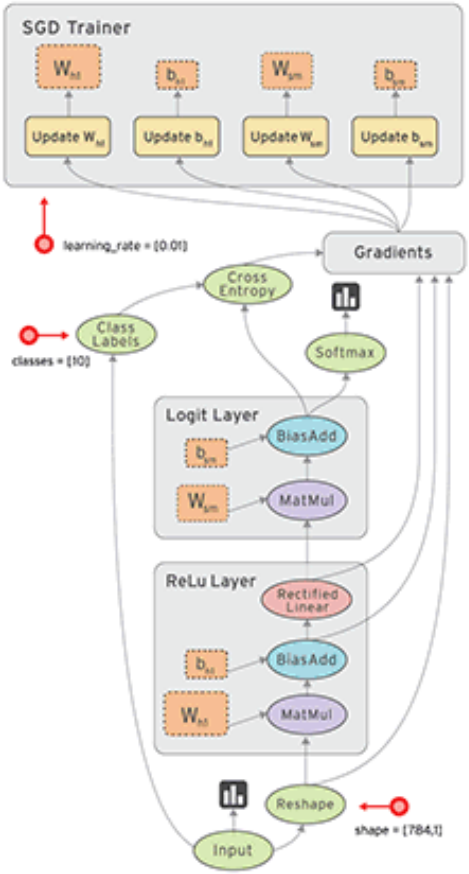


What is a Data Flow Graph?

- Directed graph
- Describes mathematical computation
- Node: mathematical operation
- Edge: input/output relationship between nodes
- data edges carry tensors

Tensor Flow

- By **Google** Brain Team
- Open Source library for numeric computation using data flow graphs
- Flow of **tensors** through a data **flow** graph
- Developed to conduct ML and DNN research
  - BUT general enough to be applicable to wide variety of other domains as well



TensorFlow

- Python API over a C/C++ engine that makes it run fast.
- Why did Google open source it?
  - Hoping to create **open standard** for exchanging ML research ideas and putting ML in products
  - Google is actually using it in its products/services

Tensor Flow Features

- Auto-differentiation
  - Good for Gradient-based ML algorithms
  - User defines **computational graph** of predictive model and **objective function** and **data** ➔ **TensorFlow** computes the derivatives
- Flexibility
  - Common subgraphs in NN are provided
  - Add your low-level operators if you wish
  - Or build higher level library on top of tensorflow
- Portable
  - CPUs or GPUs
- Python and C++ interface

Simple Example: Fitting a line

```
# Create 100 phony x, y data points in NumPy, y = x * 0.1 + 0.3
x_data = np.random.rand(100).astype("float32")
y_data = x_data * 0.1 + 0.3

# Try to find values for W and b that compute y_data = W * x_data + b
# (We know that W should be 0.1 and b 0.3, but Tensorflow will
# figure that out for us.)
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b

# Minimize the mean squared errors.
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# Before starting, initialize the variables.  We will 'run' this first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in xrange(201):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(W), sess.run(b))

# Learns best fit is W: [0.1], b: [0.3]
```

Generate data

Define the variables

Notice that we did not provide the gradient

Build the flow graph.

Nothing is running yet!

Run:

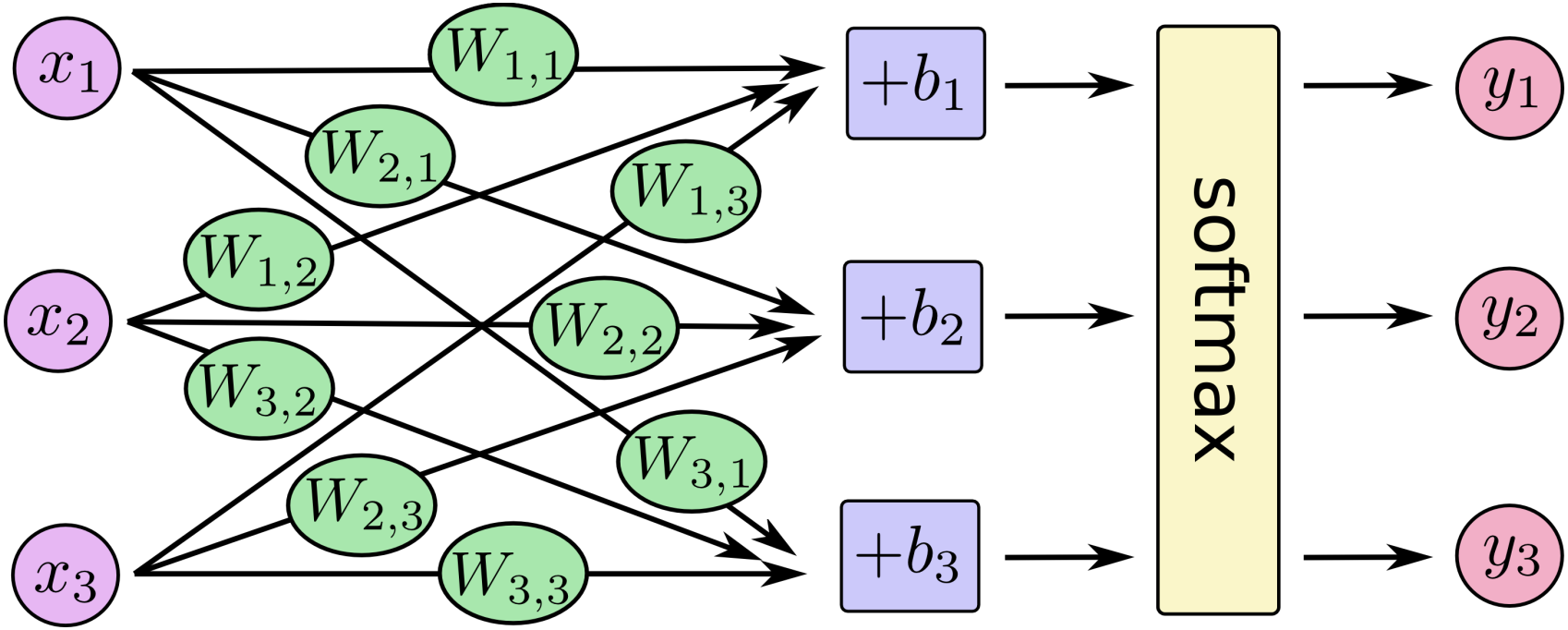
- Initialization
- Training

SoftMax regression on MNIST dataset

- MNIST dataset
  - is the “hello world” of ML
  - handwritten digits

- To get probability of an image being each of the 10 digits➔ softmax regression
  - Generalization of logistic regression to multiple classes

Softmax Regression [1]



Softmax Regression [3]

- Cost Function:

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\left\{y^{(i)} = k\right\} \log \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})}\right]$$

$$P(y^{(i)} = k | x^{(i)}; \theta) = \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})}$$

Normalized Exponential

- Gradient:

Find theta that minimizes the cost function

$$\nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^{m} \left[x^{(i)} \left(1\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)}; \theta)\right)\right]$$

SoftMax Regression using Tensor Flow: 91% on MNIST

This implementation uses a bias (b) .

```
import tensorflow as tf
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
for i in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

CNN for MNIST

- Few lines can program the multi-layer CNN:
  - Layers: Convolution, max pooling, convolution, max pooling, fully connected layer, softmax
- If interested:

https://www.tensorflow.org/versions/master/tutorials/mnist/pros/index.html

References

1. https://www.tensorflow.org
2. http://www.slideshare.net/yokotatsuya/principal-component-analysis-for-tensor-analysis-and-eeg-classification
3. http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/
4. http://deeplearning4j.org/compare-dl4j-torch7-pylearn.html

Logistic Regression [3]

$$P(y = 1|x) = h_\theta(x) = \frac{1}{1 + \exp(-\theta^\top x)} \equiv \sigma(\theta^\top x),$$

$$P(y = 0|x) = 1 - P(y = 1|x) = 1 - h_\theta(x).$$

Sigmoid to force it to 0 or 1

$$J(\theta) = -\sum_i \left(y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right).$$

$P(y=1|x^i)$ $P(y=0|x^i)$

$$\nabla_\theta J(\theta) = \sum_i x^{(i)}(h_\theta(x^{(i)}) - y^{(i)})$$

CNN on Wikipedia