Data-X: Introduction to TensorFlow

Computation Graphs, Simple One Layer ANN, Vanilla DNN

Author: Alexander Fred Ojala

Sources: Sebastian Raschka, Aurélien Géron, etc.

General notebook setup

Make notebook compatible with Python 2 and 3, plus import standard packages.

In [2]:
```
# Pyton 2 and 3 support
from __future__ import division, print_function, unicode_literals

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

# Hide warnings
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# Canonical way of importing TensorFlow
import tensorflow as tf

# If this doesn't work TensorFlow is not installed correctly
```

In [4]:
```
# Check tf version, oftentimes tensorflow is not backwards compatible
tf.__version__
```

Out[4]:
```
'1.11.0'
```

Tip: When using Jupyter notebook make sure to call tf.reset_default_graph() at the beginning to clear the symbolic graph before defining new nodes.

In [5]:
```
tf.reset_default_graph()
```

TensorBoard setup

Tip2: Setup TensorBoard to monitor graph etc

In [6]:
```
from datetime import datetime
import os
import pathlib

t = datetime.utcnow().strftime("%Y%m%d%H%M%S")
log_dir = "tf_logs"
logd = "/tmp/{}/r{}/".format(log_dir, t)

# Then every time you have specified a graph run:
# file_writer = tf.summary.FileWriter(logdir, tf.get_default_graph())

# Make directory if it doesn't exist

from pathlib import Path
home = str(Path.home())

logdir = os.path.join(os.sep,home,logd)

if not os.path.exists(logdir):
    os.makedirs(logdir)
```

Please confirm that this `Tensorboard` setup works on Windows!

In [7]:
```
# TensorBoard Graph visualizer in notebook
import numpy as np
from IPython.display import clear_output, Image, display, HTML

def strip_consts(graph_def, max_const_size=32):
    """Strip large constant values from graph_def."""
    strip_def = tf.GraphDef()
    for n0 in graph_def.node:
        n = strip_def.node.add()
        n.MergeFrom(n0)
        if n.op == 'Const':
            tensor = n.attr['value'].tensor
            size = len(tensor.tensor_content)
            if size > max_const_size:
                tensor.tensor_content = "<stripped %d bytes>"%size
    return strip_def

def show_graph(graph_def, max_const_size=32):
    """Visualize TensorFlow graph."""
    if hasattr(graph_def, 'as_graph_def'):
        graph_def = graph_def.as_graph_def()
    strip_def = strip_consts(graph_def, max_const_size=max_const_size)
    code = """
        <script src="//cdnjs.cloudflare.com/ajax/libs/polymer/0.3.3/platform.js"></script>
        <script>
          function load() {{
            document.getElementById("{id}").pbtxt = {data};
          }}
        </script>
        <link rel="import" href="https://tensorboard.appspot.com/tf-graph-basic.build.html" onload=load()
        <div style="height:600px">
          <tf-graph-basic id="{id}"></tf-graph-basic>
        </div>
    """.format(data=repr(str(strip_def)), id='graph'+str(np.random.rand()))

    iframe = """
        <iframe seamless style="width:1200px;height:620px;border:0" srcdoc="{}"></iframe>
    """.format(code.replace('"', '&quot;'))
    display(HTML(iframe))
```

MNIST: Intro to NN in TensorFlow

Example taken from Google Docs

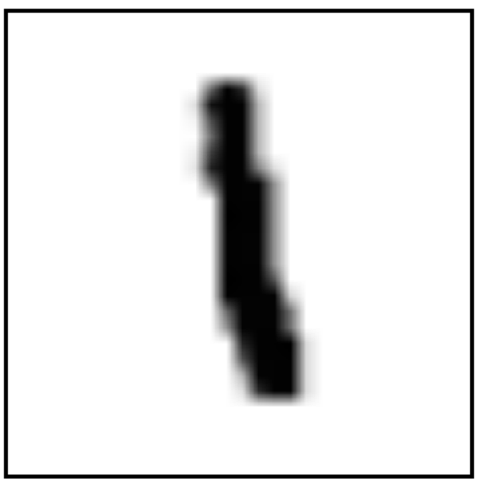
We are now going to recognize hand-written digits.

About the most classic NN dataset

The MNIST data is split into three parts: 55,000 data points of training data (mnist.train), 10,000 points of test data (mnist.test), and 5,000 points of validation data (mnist.validation). This split is very important: it's essential in machine learning that we have separate data which we don't learn from so that we can make sure that what we've learned actually generalizes!

Every MNIST data point has two parts: an image of a handwritten digit and a corresponding label. We'll call the images "x" and the labels "y". Both the training set and test set contain images and their corresponding labels; for example the training images are mnist.train.images and the training labels are mnist.train.labels.

Each image is 28 pixels by 28 pixels. We can interpret this as a big array of numbers:

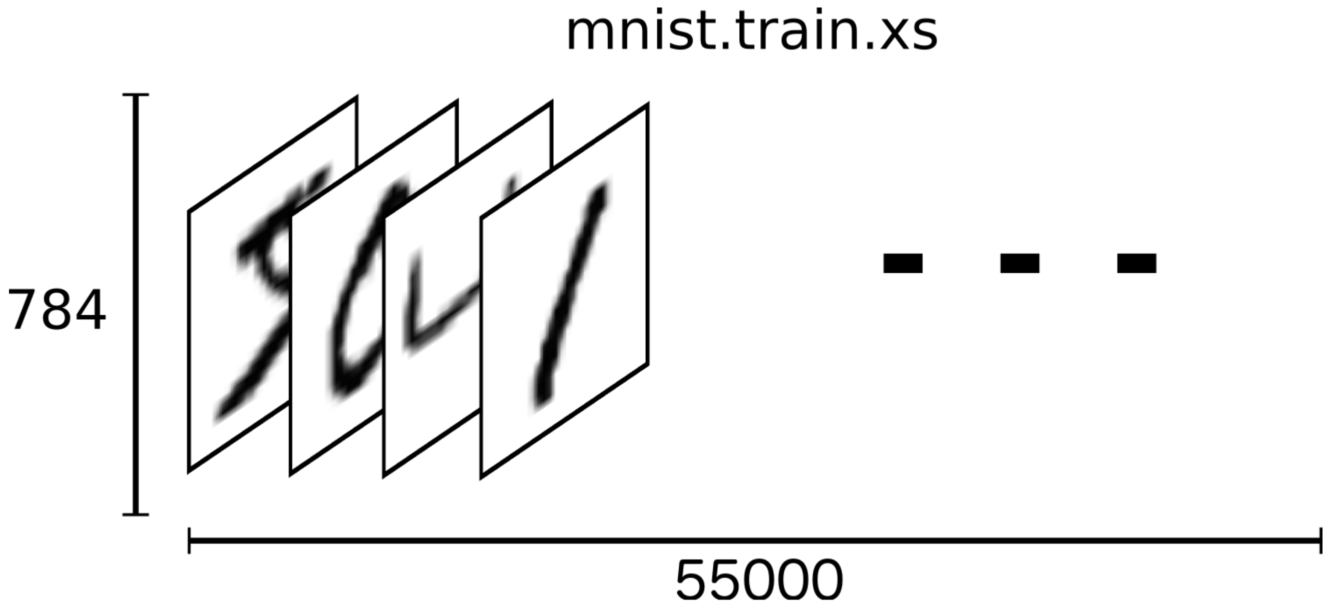


$$\simeq \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&.6&.8&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.5&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.7&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&.9&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&.3&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

We can flatten this array into a vector of 28x28 = 784 numbers. It doesn't matter how we flatten the array, as long as we're consistent between images. From this perspective, the MNIST images are just a bunch of points in a 784-dimensional vector space, with a very rich structure (warning: computationally intensive visualizations).

Flattening the data throws away information about the 2D structure of the image. Isn't that bad? Well, the best computer vision methods do exploit this structure, and we will in later tutorials. But the simple method we will be using here, a softmax regression (defined below), won't.

The result is that mnist.train.images is a tensor (an n-dimensional array) with a shape of [55000, 784]. The first dimension is an index into the list of images and the second dimension is the index for each pixel in each image. Each entry in the tensor is a pixel intensity between 0 and 1, for a particular pixel in a particular image.



Each image in MNIST has a corresponding label, a number between 0 and 9 representing the digit drawn in the image.

For the purposes of this tutorial, we're going to want our labels as "one-hot vectors". A one-hot vector is a vector which is 0 in most dimensions, and 1 in a single dimension. In this case, the nth digit will be represented as a vector which is 1 in the nth dimension. For example, 3 would be $[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$. Consequently, mnist.train.labels is a [55000, 10] array of floats.

Softmax regression

We know that every image in MNIST is of a handwritten digit between zero and nine. So there are only ten possible things that a given image can be. We want to be able to look at an image and give the probabilities for it being each digit. For example, our model might look at a picture of a nine and be 80% sure it's a nine, but give a 5% chance to it being an eight (because of the top loop) and a bit of probability to all the others because it isn't 100% sure.

This is a classic case where a softmax regression is a natural, simple model. If you want to assign probabilities to an object being one of several different things, softmax is the thing to do, because softmax gives us a list of values between 0 and 1 that add up to 1. Even later on, when we train more sophisticated models, the final step will be a layer of softmax.

A softmax regression has two steps: first we add up the evidence of our input being in certain classes, and then we convert that evidence into probabilities.

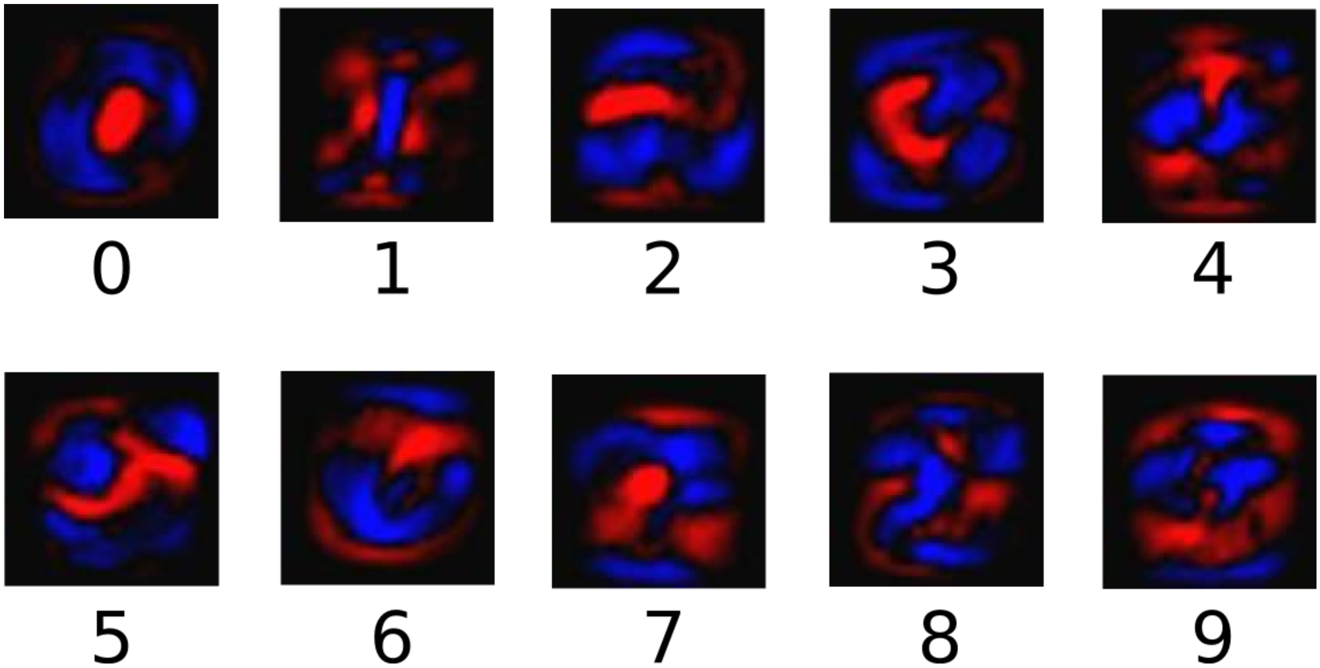
Softmax regression equation

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \, for \, j = 1, \ldots, K$$

where z represents the values from the output layer and K represents the number of output classes.

To tally up the evidence that a given image is in a particular class, we do a weighted sum of the pixel intensities. The weight is negative if that pixel having a high intensity is evidence against the image being in that class, and positive if it is evidence in favor.

The following diagram shows the weights one model learned for each of these classes. Red represents negative weights, while blue represents positive weights.



(Vanilla) ANN Network structure

- Any Neural Network with one hidden layer can be a Universal Function Approximator. Source: [https://en.wikipedia.org/wiki/Universal_approximation_theorem (https://en.wikipedia.org/wiki/Universal_approximation_theorem)](https://en.wikipedia.org/wiki/Universal_approximation_theorem)
- The number of input nodes are equal to the number of features
- The number of output nodes are equal to the number of classes (for classification tasks)
- A bias term is added to every layer that only feeds in a 1, that adds an extra degree of freedom for every functional input value to the next function

How deep should we go?

- We can overfit Neural Nets, one way to combat that is by using dropout and regularization
- Predictions will usually be better when we increase depth of network and widen it (increase the number of neurons in every layer)

Activation Functions

- Classically the sigmoid function was used in the hidden layers (simplest function between 0 - 1). Logit function.
- Nowadays it is more common to use the ReLU (Rectified Linear Unit). Much quicker! For deep networks sigmoid might not want to converge at all. Much better to handle exploding and vanishing gradients (Leaky Relu). Can also combat that with *Batch Normalization*.
- For the input layer we send in the (standardized) values.
- For the output layer we often use a softmax function (multi-class classification) or a sigmoid function (binary classification). Softmax only works if the classes are mutually exclusive, i.e. we only try to label one pattern in every training example.

Training algorithm steps

- Train a model to make a prediction
- Compute distance between predictions and true values
- Modify weights and biases to lower error

Overfitting

- Mostly because our network has too many degrees of freedom (neurons in the network)
- Can use L1 and L2 regularization on the cost function
- Drop out (used to mitigate the effects of too many degrees of freedom)

ANNs are not great at classifying images

- We don't make use of the image shapes and curves. Shape info is lost when we flatten arrays.

ANN One Layer Softmax Classification

What we will accomplish in this section:

- Create a softmax regression function that is a model for recognizing MNIST digits, based on looking at every pixel in the image
- Use Tensorflow to train the model to recognize digits by having it "look" at thousands of examples (and run our first Tensorflow session to do so)
- Check the model's accuracy with our test data

In [8]:

```
# Read in input data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# contains info
import tensorflow.examples.tutorials.mnist.mnist as mnist_info
```

```
WARNING:tensorflow:From <ipython-input-8-a55af32b7aeb>:3: read_data_sets (from tensorflow.contrib.lear
n.python.learn.datasets.mnist) is deprecated and will be removed in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/d
atasets/mnist.py:260: maybe_download (from tensorflow.contrib.learn.python.learn.datasets.base) is dep
recated and will be removed in a future version.
Instructions for updating:
Please write your own downloading logic.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/d
atasets/mnist.py:262: extract_images (from tensorflow.contrib.learn.python.learn.datasets.mnist) is de
precated and will be removed in a future version.
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting MNIST_data/train-images-idx3-ubyte.gz
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/d
atasets/mnist.py:267: extract_labels (from tensorflow.contrib.learn.python.learn.datasets.mnist) is de
precated and will be removed in a future version.
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting MNIST_data/train-labels-idx1-ubyte.gz
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/d
atasets/mnist.py:110: dense_to_one_hot (from tensorflow.contrib.learn.python.learn.datasets.mnist) is
deprecated and will be removed in a future version.
Instructions for updating:
Please use tf.one_hot on tensors.
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/d
atasets/mnist.py:290: DataSet.__init__ (from tensorflow.contrib.learn.python.learn.datasets.mnist) is
deprecated and will be removed in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
```

Here `mnist` is a lightweight class which stores the training, validation, and testing sets as NumPy arrays. It also provides a function for iterating through data minibatches, which we will use below.

In [9]:

```
!ls MNIST_data/
```

```
t10k-images-idx3-ubyte.gz  train-images-idx3-ubyte.gz
t10k-labels-idx1-ubyte.gz  train-labels-idx1-ubyte.gz
```

In [10]:

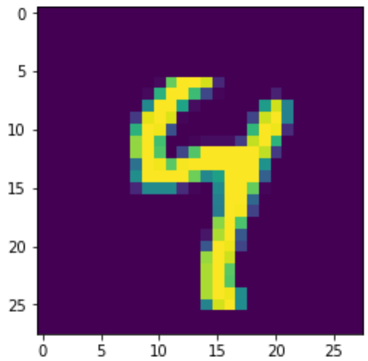
```
mnist_info.IMAGE_PIXELS
```

Out[10]:

```
784
```

In [11]:
```
mnist.train.images.shape
```
Out[11]:
```
(55000, 784)
```

In [12]:
```
plt.imshow(mnist.train.images[2,:].reshape(28,28))
```
Out[12]:
```
<matplotlib.image.AxesImage at 0x1c242a5cc0>
```



CONSTRUCTION PHASE

In [13]:
```
tf.reset_default_graph()
```

In [14]:
```
# Define input
x = tf.placeholder(tf.float32,shape = [None,784])
# None, because we don't specify how many examples we'll look at

W = tf.Variable(tf.zeros([784, 10])) # number of weights
b = tf.Variable(tf.zeros([10])) # number of bias terms
```

In [15]:
```
y_hat = tf.nn.softmax(tf.matmul(x, W) + b)
# define what we'll take the softmax activation on

# Notice order on x and W (dimensions must match)
```

In [16]:
```
# correct answers

y = tf.placeholder(tf.float32, [None, 10])
```

In [17]:
```
# define loss function
# Cross entropy

ce = tf.reduce_mean(-tf.reduce_sum( y* tf.log(y_hat),axis=1))
```

In [18]:
```
ce # sum over the columns to get cost for every training example
```
Out[18]:
```
<tf.Tensor 'Mean:0' shape=() dtype=float32>
```

In [19]:
```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(ce)
```

In [20]:
```
# monitor accuracy
def acc():
    correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_hat,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    print(sess.run(accuracy, feed_dict={x: mnist.test.images, y: mnist.test.labels}))
```

EXECUTION PHASE

In [21]:
```
sess = tf.Session()
```

In [22]:
```
sess.run(tf.global_variables_initializer())
```

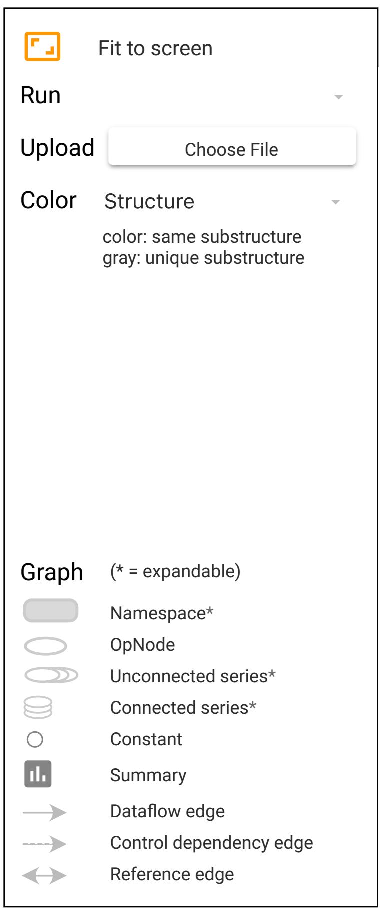
In [23]:
```
for i in range(1000):
    # get batches of training data
    # we don't show everything to the network at once
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y: batch_ys})
    if i%100==0:
        acc()
```

```
0.2073
0.8749
0.9081
0.9116
0.9118
0.9131
0.9177
0.9137
0.9201
0.9192
```

In [24]:
```
acc()
```

```
0.9164
```

In [25]:
```
show_graph(tf.get_default_graph()) #not that great, we should probably use scopes
```



Deep MINST

Example from the book `Hands-On Machine Learning with Scikit-Learn and TensorFlow`.

Improve the one layer model above, by adding extra layers with ReLU activation functions

In [26]:
```
#from tensorflow.examples.tutorials.mnist import input_data
#mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

tf.learn (high level API)

Predefined models, for convenience. This is for fast model building when you have a standard problem.

In [27]:
```
# Read input_data (not as one_hot)
from tensorflow.examples.tutorials.mnist import input_data

# new folder
mnist = input_data.read_data_sets("/tmp/data/")

# Assign them to values
X_train = mnist.train.images
X_test = mnist.test.images
y_train = mnist.train.labels.astype("int")
y_test = mnist.test.labels.astype("int")
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

CONSTRUCTION PHASE

In [28]:
```
# define features
feature_cols = tf.contrib.learn.infer_real_valued_columns_from_input(X_train)

# dense neural network classifier
# three layers 300, 200 and 100
# 10 classes
dnn_clf = tf.contrib.learn.DNNClassifier(hidden_units=[300,200,100], n_classes=10,
                                         feature_columns=feature_cols)

# if TensorFlow >= 1.1, make compatible with sklearn
dnn_clf = tf.contrib.learn.SKCompat(dnn_clf)
```

```
WARNING:tensorflow:From <ipython-input-28-123803bdab74>:2: infer_real_valued_columns_from_input (from
tensorflow.contrib.learn.python.learn.estimators.estimator) is deprecated and will be removed in a fut
ure version.
Instructions for updating:
Please specify feature columns explicitly.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/e
stimators/estimator.py:143: setup_train_data_feeder (from tensorflow.contrib.learn.python.learn.learn_
io.data_feeder) is deprecated and will be removed in a future version.
Instructions for updating:
Please use tensorflow/transform or tf.data.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/l
earn_io/data_feeder.py:96: extract_dask_data (from tensorflow.contrib.learn.python.learn.learn_io.dask
_io) is deprecated and will be removed in a future version.
Instructions for updating:
Please feed input to tf.data to support dask.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/l
earn_io/data_feeder.py:100: extract_pandas_data (from tensorflow.contrib.learn.python.learn.learn_io.p
andas_io) is deprecated and will be removed in a future version.
Instructions for updating:
Please access pandas data directly.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/l
earn_io/data_feeder.py:159: DataFeeder.__init__ (from tensorflow.contrib.learn.python.learn.learn_io.d
ata_feeder) is deprecated and will be removed in a future version.
Instructions for updating:
Please use tensorflow/transform or tf.data.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/l
earn_io/data_feeder.py:340: check_array (from tensorflow.contrib.learn.python.learn.learn_io.data_feed
er) is deprecated and will be removed in a future version.
Instructions for updating:
Please convert numpy dtypes explicitly.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/e
stimators/estimator.py:183: infer_real_valued_columns_from_input_fn (from tensorflow.contrib.learn.pyt
hon.learn.estimators.estimator) is deprecated and will be removed in a future version.
Instructions for updating:
Please specify feature columns explicitly.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/e
stimators/dnn.py:378: multi_class_head (from tensorflow.contrib.learn.python.learn.estimators.head) is
deprecated and will be removed in a future version.
Instructions for updating:
Please switch to tf.contrib.estimator.*_head.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/e
stimators/estimator.py:1180: BaseEstimator.__init__ (from tensorflow.contrib.learn.python.learn.estima
tors.estimator) is deprecated and will be removed in a future version.
Instructions for updating:
Please replace uses of any Estimator from tf.contrib.learn with an Estimator from tf.estimator.*
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/e
stimators/estimator.py:428: RunConfig.__init__ (from tensorflow.contrib.learn.python.learn.estimators.
run_config) is deprecated and will be removed in a future version.
Instructions for updating:
When switching to tf.estimator.Estimator, use tf.estimator.RunConfig instead.
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: /var/folders/wj/97_zhf897sn1kf3xf99c5rvw
0000gn/T/tmpbrfkibb7
INFO:tensorflow:Using config: {'_task_type': None, '_task_id': 0, '_cluster_spec': <tensorflow.python.
training.server_lib.ClusterSpec object at 0xb1614c240>, '_master': '', '_num_ps_replicas': 0, '_num_wo
rker_replicas': 0, '_environment': 'local', '_is_chief': True, '_evaluation_master': '', '_train_distr
ibute': None, '_eval_distribute': None, '_device_fn': None, '_tf_config': gpu_options {
  per_process_gpu_memory_fraction: 1.0
}
, '_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_secs': 600, '_log_step_count
```

```
_steps': 100, '_protocol': None, '_session_config': None, '_save_checkpoints_steps': None, '_keep_chec
kpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_model_dir': '/var/folders/wj/97_zhf897sn1kf
3xf99c5rvw0000gn/T/tmpbrfkibb7'}
WARNING:tensorflow:From <ipython-input-28-123803bdab74>:11: SKCompat.__init__ (from tensorflow.contri
b.learn.python.learn.estimators.estimator) is deprecated and will be removed in a future version.
Instructions for updating:
Please switch to the Estimator interface.
```

EVALUATION PHASE

In [29]:
```
# fit the model, 4000 iterations
dnn_clf.fit(X_train, y_train, batch_size=50, steps=1000)
```

```
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/l
earn_io/data_feeder.py:98: extract_dask_labels (from tensorflow.contrib.learn.python.learn.learn_io.da
sk_io) is deprecated and will be removed in a future version.
Instructions for updating:
Please feed input to tf.data to support dask.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/l
earn_io/data_feeder.py:102: extract_pandas_labels (from tensorflow.contrib.learn.python.learn.learn_i
o.pandas_io) is deprecated and will be removed in a future version.
Instructions for updating:
Please access pandas data directly.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/e
stimators/head.py:678: ModelFnOps.__new__ (from tensorflow.contrib.learn.python.learn.estimators.model
_fn) is deprecated and will be removed in a future version.
Instructions for updating:
When switching to tf.estimator.Estimator, use tf.estimator.EstimatorSpec. You can use the `estimator_s
pec` method to create an equivalent one.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 0 into /var/folders/wj/97_zhf897sn1kf3xf99c5rvw0000gn/T/tmpbrfk
ibb7/model.ckpt.
INFO:tensorflow:loss = 2.3555002, step = 1
INFO:tensorflow:global_step/sec: 262.305
INFO:tensorflow:loss = 0.33504337, step = 101 (0.383 sec)
INFO:tensorflow:global_step/sec: 303.202
INFO:tensorflow:loss = 0.25071108, step = 201 (0.331 sec)
INFO:tensorflow:global_step/sec: 289.398
INFO:tensorflow:loss = 0.38195786, step = 301 (0.343 sec)
INFO:tensorflow:global_step/sec: 303.412
INFO:tensorflow:loss = 0.19546433, step = 401 (0.332 sec)
INFO:tensorflow:global_step/sec: 314.673
INFO:tensorflow:loss = 0.21923284, step = 501 (0.315 sec)
INFO:tensorflow:global_step/sec: 302.729
INFO:tensorflow:loss = 0.03860847, step = 601 (0.330 sec)
INFO:tensorflow:global_step/sec: 304.989
INFO:tensorflow:loss = 0.14427614, step = 701 (0.328 sec)
INFO:tensorflow:global_step/sec: 266.178
INFO:tensorflow:loss = 0.1559586, step = 801 (0.377 sec)
INFO:tensorflow:global_step/sec: 236.208
INFO:tensorflow:loss = 0.07918204, step = 901 (0.422 sec)
INFO:tensorflow:Saving checkpoints for 1000 into /var/folders/wj/97_zhf897sn1kf3xf99c5rvw0000gn/T/tmpb
rfkibb7/model.ckpt.
INFO:tensorflow:Loss for final step: 0.13439712.
```

Out[29]:
```
SKCompat()
```

In [30]:
```
# Calculate accuracies
from sklearn.metrics import accuracy_score

y_pred = dnn_clf.predict(X_test)
print()
print('Accuracy',accuracy_score(y_test, y_pred['classes']))
```

```
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from /var/folders/wj/97_zhf897sn1kf3xf99c5rvw0000gn/T/tmpbrfkibb
7/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.

Accuracy 0.959
```

Let's build a DNN with 2 hidden layers from scratch

This is great for understanding!

CONSTRUCTION PHASE

In [31]:
```
# Define hyperparameters and input size

n_inputs = 28*28  # MNIST
n_hidden1 = 300
n_hidden2 = 200
n_hidden3 = 100
n_outputs = 10
```

In [32]:
```
# Reset graph
tf.reset_default_graph()
```

In [45]:
```
# Placeholders for data (inputs and targets)
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int64, shape=(None), name="y")
keep_prob=tf.placeholder(tf.float32)
```

In [46]:
```
# Define neuron layers (ReLU in hidden layers)
# We'll take care of Softmax for output with loss function

def neuron_layer(X, n_neurons, name, activation=None):
    # X input to neuron
    # number of neurons for the layer
    # name of layer
    # pass in eventual activation function

    with tf.name_scope(name):
        n_inputs = int(X.get_shape()[1])

        # initialize weights to prevent vanishing / exploding gradients
        stddev = 2 / np.sqrt(n_inputs)
        init = tf.truncated_normal((n_inputs, n_neurons), stddev=stddev)

        # Initialize weights for the layer
        W = tf.Variable(init, name="weights")
        # biases
        b = tf.Variable(tf.zeros([n_neurons]), name="bias")

        # Output from every neuron
        Z = tf.matmul(X, W) + b
        if activation is not None:
            return activation(Z)
        else:
            return Z
```

In [47]:
```
# Define the hidden layers
with tf.name_scope("dnn"):
    hidden1 = neuron_layer(X, n_hidden1, name="hidden1",
                           activation=tf.nn.relu)
    hidden2 = neuron_layer(hidden1, n_hidden2, name="hidden2",
                           activation=tf.nn.relu)
    hidden3 = neuron_layer(hidden2, n_hidden3, name="hidden3",
                          activation=tf.nn.relu)
    logits = neuron_layer(hidden3, n_outputs, name="outputs")
```

In [48]:
```
# Define loss function (that also optimizes Softmax for output):

with tf.name_scope("loss"):
    # logits are from the last output of the dnn
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,
                                                              logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
```

In [49]:
```
# Training step with Gradient Descent

learning_rate = 0.01

with tf.name_scope("train"):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    training_op = optimizer.minimize(loss)
```

In [50]:
```
tf.nn.dropout(hidden1,0.9,noise_shape=None,seed=None,name=None)
tf.nn.dropout(hidden2,0.9,noise_shape=None,seed=None,name=None)
tf.nn.dropout(hidden3,0.9,noise_shape=None,seed=None,name=None)
```

Out[50]:
```
<tf.Tensor 'dropout_5/mul:0' shape=(?, 100) dtype=float32>
```

In [51]:
```
# Evaluation to see accuracy

with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
```

Show graph

In [ ]:

EVALUATION PHASE

Train steps

In [52]:
```
init = tf.global_variables_initializer()
saver = tf.train.Saver()

n_epochs = 10
batch_size = 50

with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        for iteration in range(mnist.train.num_examples // batch_size):
            X_batch, y_batch = mnist.train.next_batch(batch_size)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch,keep_prob:0.9})
        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_val = accuracy.eval(feed_dict={X: mnist.validation.images,
                                            y: mnist.validation.labels})
        print(epoch, "Train accuracy:", acc_train, "Val accuracy:", acc_val)

    save_path = saver.save(sess, "./my_model_final.ckpt") # save model
```

```
0 Train accuracy: 0.96 Val accuracy: 0.9256
1 Train accuracy: 0.96 Val accuracy: 0.9464
2 Train accuracy: 0.98 Val accuracy: 0.953
3 Train accuracy: 0.96 Val accuracy: 0.9558
4 Train accuracy: 0.96 Val accuracy: 0.9632
5 Train accuracy: 0.98 Val accuracy: 0.9662
6 Train accuracy: 0.96 Val accuracy: 0.9684
7 Train accuracy: 0.98 Val accuracy: 0.9686
8 Train accuracy: 1.0 Val accuracy: 0.9706
9 Train accuracy: 1.0 Val accuracy: 0.973
```

Evaluate accuracy

In [53]:
```
with tf.Session() as sess:
    saver.restore(sess, "./my_model_final.ckpt") # or better, use save_path
    X_new_scaled = mnist.test.images[:40]
    Z = logits.eval(feed_dict={X: X_new_scaled})
    y_pred = np.argmax(Z, axis=1)

print("Predicted classes:", y_pred)
print("Actual classes:   ", mnist.test.labels[:40])
```

```
INFO:tensorflow:Restoring parameters from ./my_model_final.ckpt
Predicted classes: [7 2 1 0 4 1 4 9 6 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 4 7 2 7
 1 2 1]
Actual classes:    [7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 4 7 2 7
 1 2 1]
```

In [ ]:

Data-X: Introduction to TensorFlow

Computation Graphs, Simple One Layer ANN, Vanilla DNN

Author: Alexander Fred Ojala

Sources: Sebastian Raschka, Aurélien Géron, etc.

General notebook setup

Make notebook compatible with Python 2 and 3, plus import standard packages.

In [27]:
```
# Pyton 2 and 3 support
from __future__ import division, print_function, unicode_literals

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

# Hide warnings
import warnings
warnings.filterwarnings('ignore')
```

In [28]:
```
# Canonical way of importing TensorFlow
import tensorflow as tf

# If this doesn't work TensorFlow is not installed correctly
```

In [29]:
```
# Check tf version, oftentimes tensorflow is not backwards compatible
tf.__version__
```

Out[29]: '1.11.0'

Tip: When using Jupyter notebook make sure to call tf.reset_default_graph() at the beginning to clear the symbolic graph before defining new nodes.

In [30]:
```
tf.reset_default_graph()
```

TensorBoard setup

Tip2: Setup TensorBoard to monitor graph etc

In [31]:
```
from datetime import datetime
import os
import pathlib

t = datetime.utcnow().strftime("%Y%m%d%H%M%S")
log_dir = "tf_logs"
logd = "/tmp/{}/r{}/".format(log_dir, t)

# Then every time you have specified a graph run:
# file_writer = tf.summary.FileWriter(logdir, tf.get_default_graph())

# Make directory if it doesn't exist

from pathlib import Path
home = str(Path.home())

logdir = os.path.join(os.sep,home,logd)

if not os.path.exists(logdir):
    os.makedirs(logdir)
```

Please confirm that this `Tensorboard` setup works on Windows!

In [32]:
```
# TensorBoard Graph visualizer in notebook
import numpy as np
from IPython.display import clear_output, Image, display, HTML

def strip_consts(graph_def, max_const_size=32):
    """Strip large constant values from graph_def."""
    strip_def = tf.GraphDef()
    for n0 in graph_def.node:
        n = strip_def.node.add()
        n.MergeFrom(n0)
        if n.op == 'Const':
            tensor = n.attr['value'].tensor
            size = len(tensor.tensor_content)
            if size > max_const_size:
                tensor.tensor_content = "<stripped %d bytes>"%size
    return strip_def

def show_graph(graph_def, max_const_size=32):
    """Visualize TensorFlow graph."""
    if hasattr(graph_def, 'as_graph_def'):
        graph_def = graph_def.as_graph_def()
    strip_def = strip_consts(graph_def, max_const_size=max_const_size)
    code = """
        <script src="//cdnjs.cloudflare.com/ajax/libs/polymer/0.3.3/platform.js"></script>
        <script>
          function load() {{
            document.getElementById("{id}").pbtxt = {data};
          }}
        </script>
        <link rel="import" href="https://tensorboard.appspot.com/tf-graph-basic.build.html" onload=load()
        <div style="height:600px">
          <tf-graph-basic id="{id}"></tf-graph-basic>
        </div>
    """.format(data=repr(str(strip_def)), id='graph'+str(np.random.rand()))

    iframe = """
        <iframe seamless style="width:1200px;height:620px;border:0" srcdoc="{}"></iframe>
    """.format(code.replace('"', '&quot;'))
    display(HTML(iframe))
```

MNIST: Intro to NN in TensorFlow

Example taken from Google Docs

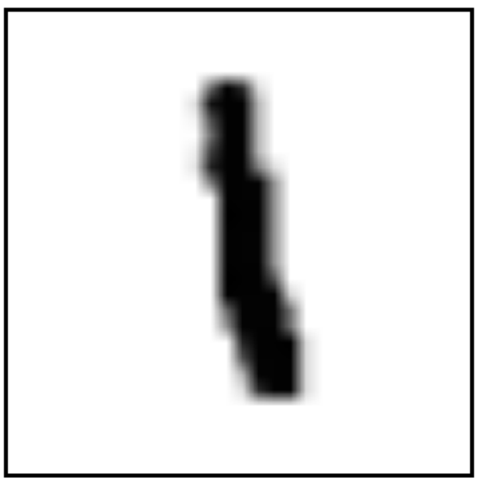
We are now going to recognize hand-written digits.

About the most classic NN dataset

The MNIST data is split into three parts: 55,000 data points of training data (mnist.train), 10,000 points of test data (mnist.test), and 5,000 points of validation data (mnist.validation). This split is very important: it's essential in machine learning that we have separate data which we don't learn from so that we can make sure that what we've learned actually generalizes!

Every MNIST data point has two parts: an image of a handwritten digit and a corresponding label. We'll call the images "x" and the labels "y". Both the training set and test set contain images and their corresponding labels; for example the training images are mnist.train.images and the training labels are mnist.train.labels.

Each image is 28 pixels by 28 pixels. We can interpret this as a big array of numbers:

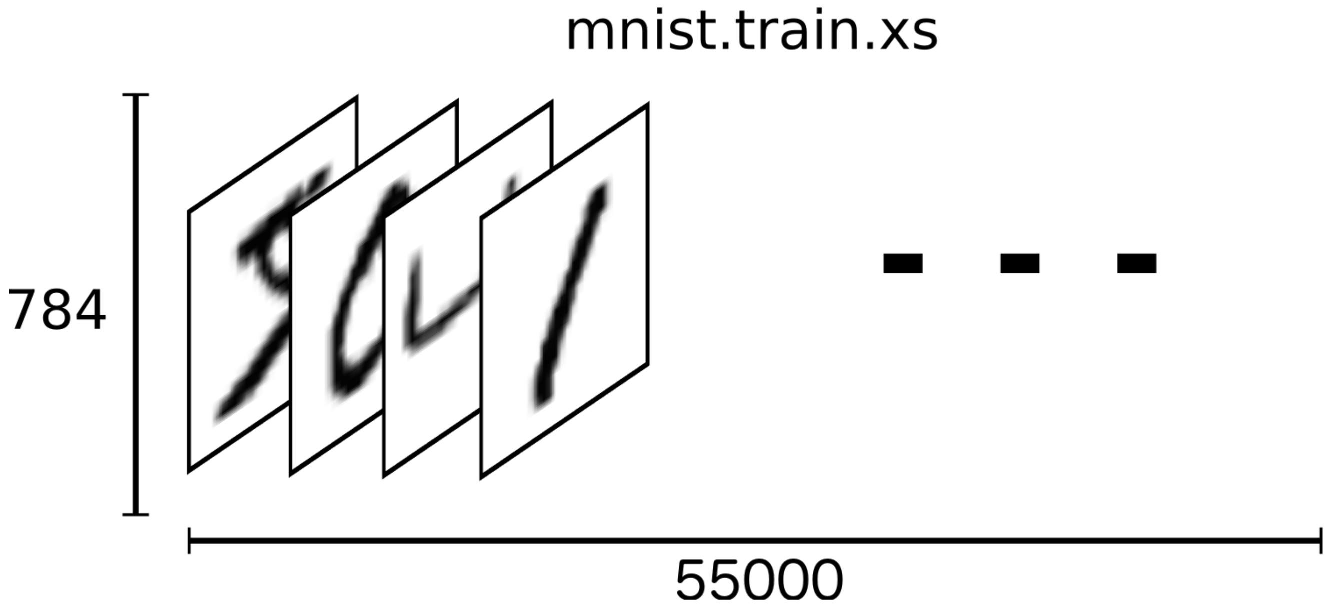


$$\simeq \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&.6&.8&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.5&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.7&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&.9&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&.3&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

We can flatten this array into a vector of 28x28 = 784 numbers. It doesn't matter how we flatten the array, as long as we're consistent between images. From this perspective, the MNIST images are just a bunch of points in a 784-dimensional vector space, with a very rich structure (warning: computationally intensive visualizations).

Flattening the data throws away information about the 2D structure of the image. Isn't that bad? Well, the best computer vision methods do exploit this structure, and we will in later tutorials. But the simple method we will be using here, a softmax regression (defined below), won't.

The result is that mnist.train.images is a tensor (an n-dimensional array) with a shape of [55000, 784]. The first dimension is an index into the list of images and the second dimension is the index for each pixel in each image. Each entry in the tensor is a pixel intensity between 0 and 1, for a particular pixel in a particular image.



Each image in MNIST has a corresponding label, a number between 0 and 9 representing the digit drawn in the image.

For the purposes of this tutorial, we're going to want our labels as "one-hot vectors". A one-hot vector is a vector which is 0 in most dimensions, and 1 in a single dimension. In this case, the nth digit will be represented as a vector which is 1 in the nth dimension. For example, 3 would be $[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$. Consequently, mnist.train.labels is a [55000, 10] array of floats.

Softmax regression

We know that every image in MNIST is of a handwritten digit between zero and nine. So there are only ten possible things that a given image can be. We want to be able to look at an image and give the probabilities for it being each digit. For example, our model might look at a picture of a nine and be 80% sure it's a nine, but give a 5% chance to it being an eight (because of the top loop) and a bit of probability to all the others because it isn't 100% sure.

This is a classic case where a softmax regression is a natural, simple model. If you want to assign probabilities to an object being one of several different things, softmax is the thing to do, because softmax gives us a list of values between 0 and 1 that add up to 1. Even later on, when we train more sophisticated models, the final step will be a layer of softmax.

A softmax regression has two steps: first we add up the evidence of our input being in certain classes, and then we convert that evidence into probabilities.

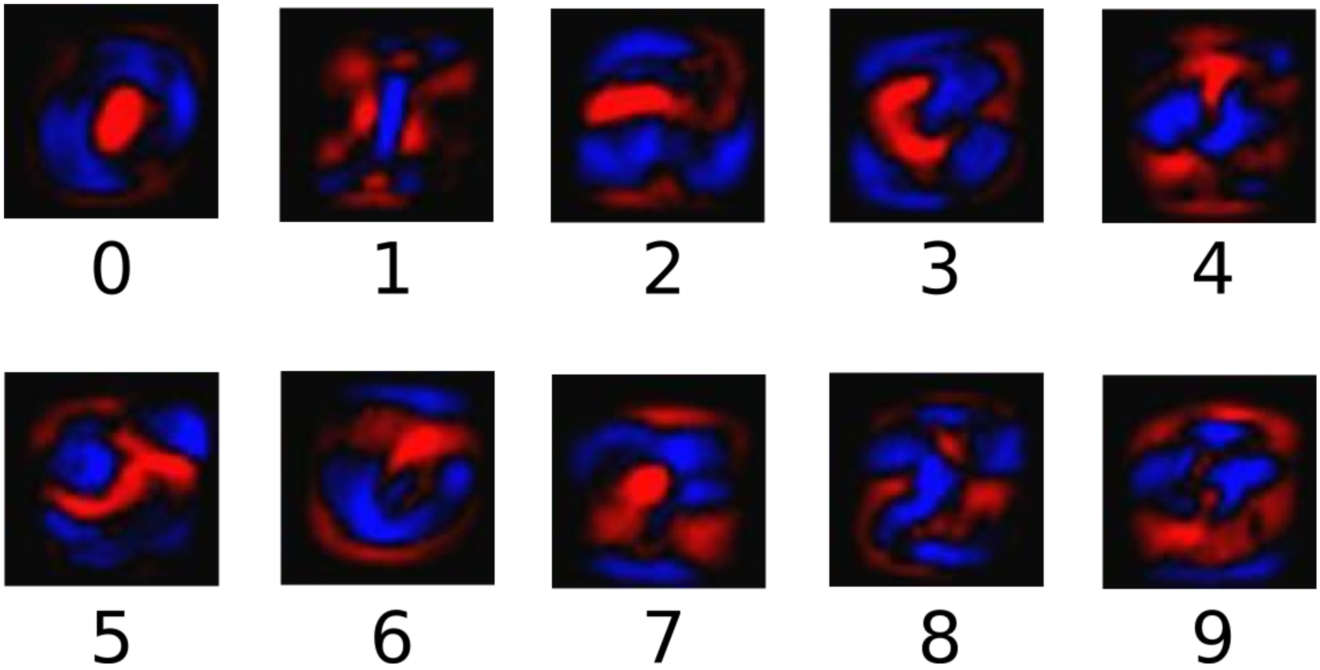
Softmax regression equation

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \; for\, j = 1, \ldots, K$$

where z represents the values from the output layer and K represents the number of output classes.

To tally up the evidence that a given image is in a particular class, we do a weighted sum of the pixel intensities. The weight is negative if that pixel having a high intensity is evidence against the image being in that class, and positive if it is evidence in favor.

The following diagram shows the weights one model learned for each of these classes. Red represents negative weights, while blue represents positive weights.



(Vanilla) ANN Network structure

- Any Neural Network with one hidden layer can be a Universal Function Approximator. Source: https://en.wikipedia.org/wiki/Universal_approximation_theorem (https://en.wikipedia.org/wiki/Universal_approximation_theorem)
- The number of input nodes are equal to the number of features
- The number of output nodes are equal to the number of classes (for classification tasks)
- A bias term is added to every layer that only feeds in a 1, that adds an extra degree of freedom for every functional input value to the next function

How deep should we go?

- We can overfit Neural Nets, one way to combat that is by using dropout and regularization
- Predictions will usually be better when we increase depth of network and widen it (increase the number of neurons in every layer)

Activation Functions

- Classically the sigmoid function was used in the hidden layers (simplest function between 0 - 1). Logit function.
- Nowadays it is more common to use the ReLU (Rectified Linear Unit). Much quicker! For deep networks sigmoid might not want to converge at all. Much better to handle exploding and vanishing gradients (Leaky Relu). Can also combat that with *Batch Normalization*.
- For the input layer we send in the (standardized) values.
- For the output layer we often use a softmax function (multi-class classification) or a sigmoid function (binary classification). Softmax only works if the classes are mutually exclusive, i.e. we only try to label one pattern in every training example.

Training algorithm steps

- Train a model to make a prediction
- Compute distance between predictions and true values
- Modify weights and biases to lower error

Overfitting

- Mostly because our network has too many degrees of freedom (neurons in the network)
- Can use L1 and L2 regularization on the cost function
- Drop out (used to mitigate the effects of too many degrees of freedom)

ANNs are not great at classifying images

- We don't make use of the image shapes and curves. Shape info is lost when we flatten arrays.

ANN One Layer Softmax Classification

What we will accomplish in this section:

- Create a softmax regression function that is a model for recognizing MNIST digits, based on looking at every pixel in the image
- Use Tensorflow to train the model to recognize digits by having it "look" at thousands of examples (and run our first Tensorflow session to do so)
- Check the model's accuracy with our test data

In [33]:
```
# Read in input data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# contains info
import tensorflow.examples.tutorials.mnist.mnist as mnist_info
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

Here `mnist` is a lightweight class which stores the training, validation, and testing sets as NumPy arrays. It also provides a function for iterating through data minibatches, which we will use below.

In [34]:
```
!ls MNIST_data/
```

```
t10k-images-idx3-ubyte.gz  train-images-idx3-ubyte.gz
t10k-labels-idx1-ubyte.gz  train-labels-idx1-ubyte.gz
```

In [35]:
```
mnist_info.IMAGE_PIXELS
```

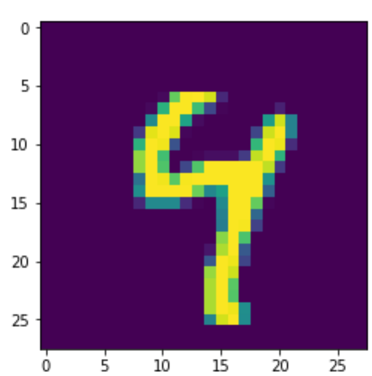
Out[35]:
```
784
```

In [36]:
```
mnist.train.images.shape
```

Out[36]:
```
(55000, 784)
```

In [37]:
```
plt.imshow(mnist.train.images[2,:].reshape(28,28))
```

Out[37]:
```
<matplotlib.image.AxesImage at 0x1c32ac4e48>
```



CONSTRUCTION PHASE

In [38]:
```
tf.reset_default_graph()
```

In [39]:
```
# Define input
x = tf.placeholder(tf.float32,shape = [None,784])
# None, because we don't specify how many examples we'll look at

W = tf.Variable(tf.zeros([784, 10])) # number of weights
b = tf.Variable(tf.zeros([10])) # number of bias terms
```

In [40]:
```
y_hat = tf.nn.softmax(tf.matmul(x, W) + b)
# define what we'll take the softmax activation on

# Notice order on x and W (dimensions must match)
```

In [46]:
```
# correct answers

y = tf.placeholder(tf.float32, [None, 10])
```

In [47]:
```
# define loss function
# Cross entropy

ce = tf.reduce_mean(-tf.reduce_sum( y* tf.log(y_hat),axis=1))
```

In [48]:
```
ce # sum over the columns to get cost for every training example
```

Out[48]:
```
<tf.Tensor 'Mean:0' shape=() dtype=float32>
```

In [49]:
```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(ce)
```

In [50]:
```
# monitor accuracy
def acc():
    correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_hat,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    print(sess.run(accuracy, feed_dict={x: mnist.test.images, y: mnist.test.labels}))
```

EXECUTION PHASE

In [ ]:

In [51]:
```
sess = tf.Session()
```

In [52]:
```
sess.run(tf.global_variables_initializer())
```

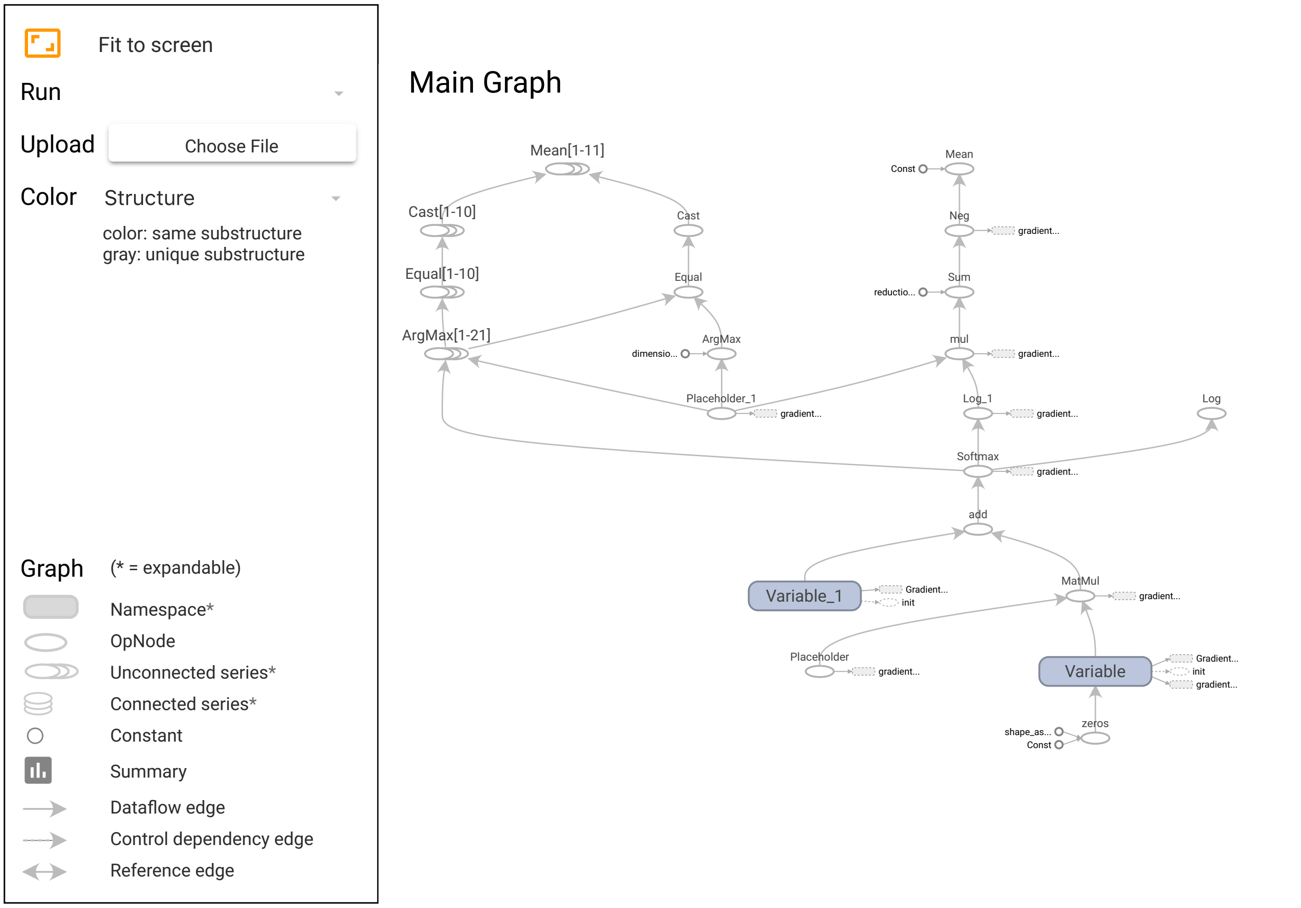
In [53]:
```
for i in range(1000):
    # get batches of training data
    # we don't show everything to the network at once
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y: batch_ys})
    if i%100==0:
        acc()
```

```
0.554
0.8939
0.9033
0.9105
0.9132
0.9077
0.9193
0.9147
0.9156
0.9191
```

In [54]:
```
acc()
```

```
0.92
```

In [55]:
```python
show_graph(tf.get_default_graph()) #not that great, we should probably use scopes
```



Deep MINST

Example from the book `Hands-On Machine Learning with Scikit-Learn and TensorFlow`.

Improve the one layer model above, by adding extra layers with ReLU activation functions

In [56]:
```python
#from tensorflow.examples.tutorials.mnist import input_data
#mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

tf.learn (high level API)

Predefined models, for convenience. This is for fast model building when you have a standard problem.

In [57]:
```
# Read input_data (not as one_hot)
from tensorflow.examples.tutorials.mnist import input_data

# new folder
mnist = input_data.read_data_sets("/tmp/data/")

# Assign them to values
X_train = mnist.train.images
X_test = mnist.test.images
y_train = mnist.train.labels.astype("int")
y_test = mnist.test.labels.astype("int")
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

CONSTRUCTION PHASE

In [58]:
```
# define features
feature_cols = tf.contrib.learn.infer_real_valued_columns_from_input(X_train)

# dense neural network classifier
# three layers 300, 200 and 100
# 10 classes
dnn_clf = tf.contrib.learn.DNNClassifier(hidden_units=[300,200,100], n_classes=10,
                                         feature_columns=feature_cols)

# if TensorFlow >= 1.1, make compatible with sklearn
dnn_clf = tf.contrib.learn.SKCompat(dnn_clf)
```

```
WARNING:tensorflow:From <ipython-input-58-123803bdab74>:2: infer_real_valued_columns_from_input (from
tensorflow.contrib.learn.python.learn.estimators.estimator) is deprecated and will be removed in a fut
ure version.
Instructions for updating:
Please specify feature columns explicitly.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/e
stimators/estimator.py:143: setup_train_data_feeder (from tensorflow.contrib.learn.python.learn.learn_
io.data_feeder) is deprecated and will be removed in a future version.
Instructions for updating:
Please use tensorflow/transform or tf.data.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/l
earn_io/data_feeder.py:96: extract_dask_data (from tensorflow.contrib.learn.python.learn.learn_io.dask
_io) is deprecated and will be removed in a future version.
Instructions for updating:
Please feed input to tf.data to support dask.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/l
earn_io/data_feeder.py:100: extract_pandas_data (from tensorflow.contrib.learn.python.learn.learn_io.p
andas_io) is deprecated and will be removed in a future version.
Instructions for updating:
Please access pandas data directly.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/l
earn_io/data_feeder.py:159: DataFeeder.__init__ (from tensorflow.contrib.learn.python.learn.learn_io.d
ata_feeder) is deprecated and will be removed in a future version.
Instructions for updating:
Please use tensorflow/transform or tf.data.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/l
earn_io/data_feeder.py:340: check_array (from tensorflow.contrib.learn.python.learn.learn_io.data_feed
er) is deprecated and will be removed in a future version.
Instructions for updating:
Please convert numpy dtypes explicitly.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/e
stimators/estimator.py:183: infer_real_valued_columns_from_input_fn (from tensorflow.contrib.learn.pyt
hon.learn.estimators.estimator) is deprecated and will be removed in a future version.
Instructions for updating:
Please specify feature columns explicitly.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/e
stimators/dnn.py:378: multi_class_head (from tensorflow.contrib.learn.python.learn.estimators.head) is
deprecated and will be removed in a future version.
Instructions for updating:
Please switch to tf.contrib.estimator.*_head.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/e
stimators/estimator.py:1180: BaseEstimator.__init__ (from tensorflow.contrib.learn.python.learn.estima
tors.estimator) is deprecated and will be removed in a future version.
Instructions for updating:
Please replace uses of any Estimator from tf.contrib.learn with an Estimator from tf.estimator.*
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/e
stimators/estimator.py:428: RunConfig.__init__ (from tensorflow.contrib.learn.python.learn.estimators.
run_config) is deprecated and will be removed in a future version.
Instructions for updating:
When switching to tf.estimator.Estimator, use tf.estimator.RunConfig instead.
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: /var/folders/wj/97_zhf897sn1kf3xf99c5rvw
0000gn/T/tmpkh5lqmqv
INFO:tensorflow:Using config: {'_task_type': None, '_task_id': 0, '_cluster_spec': <tensorflow.python.
training.server_lib.ClusterSpec object at 0x1c3d29fd68>, '_master': '', '_num_ps_replicas': 0, '_num_w
orker_replicas': 0, '_environment': 'local', '_is_chief': True, '_evaluation_master': '', '_train_dist
ribute': None, '_eval_distribute': None, '_device_fn': None, '_tf_config': gpu_options {
  per_process_gpu_memory_fraction: 1.0
}
, '_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_secs': 600, '_log_step_count
```

```
_steps': 100, '_protocol': None, '_session_config': None, '_save_checkpoints_steps': None, '_keep_chec
kpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_model_dir': '/var/folders/wj/97_zhf897sn1kf
3xf99c5rvw0000gn/T/tmpkh5lqmqv'}
WARNING:tensorflow:From <ipython-input-58-123803bdab74>:11: SKCompat.__init__ (from tensorflow.contri
b.learn.python.learn.estimators.estimator) is deprecated and will be removed in a future version.
Instructions for updating:
Please switch to the Estimator interface.
```

EVALUATION PHASE

In [59]:
```
# fit the model, 4000 iterations
dnn_clf.fit(X_train, y_train, batch_size=50, steps=1000)
```

```
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/l
earn_io/data_feeder.py:98: extract_dask_labels (from tensorflow.contrib.learn.python.learn.learn_io.da
sk_io) is deprecated and will be removed in a future version.
Instructions for updating:
Please feed input to tf.data to support dask.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/l
earn_io/data_feeder.py:102: extract_pandas_labels (from tensorflow.contrib.learn.python.learn.learn_i
o.pandas_io) is deprecated and will be removed in a future version.
Instructions for updating:
Please access pandas data directly.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/e
stimators/head.py:678: ModelFnOps.__new__ (from tensorflow.contrib.learn.python.learn.estimators.model
_fn) is deprecated and will be removed in a future version.
Instructions for updating:
When switching to tf.estimator.Estimator, use tf.estimator.EstimatorSpec. You can use the `estimator_s
pec` method to create an equivalent one.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 0 into /var/folders/wj/97_zhf897sn1kf3xf99c5rvw0000gn/T/tmpkh5l
qmqv/model.ckpt.
INFO:tensorflow:loss = 2.386386, step = 1
INFO:tensorflow:global_step/sec: 197.083
INFO:tensorflow:loss = 0.26754695, step = 101 (0.509 sec)
INFO:tensorflow:global_step/sec: 178.103
INFO:tensorflow:loss = 0.32374504, step = 201 (0.562 sec)
INFO:tensorflow:global_step/sec: 275.775
INFO:tensorflow:loss = 0.3881931, step = 301 (0.362 sec)
INFO:tensorflow:global_step/sec: 250.544
INFO:tensorflow:loss = 0.19167796, step = 401 (0.399 sec)
INFO:tensorflow:global_step/sec: 278.894
INFO:tensorflow:loss = 0.20388363, step = 501 (0.359 sec)
INFO:tensorflow:global_step/sec: 160.76
INFO:tensorflow:loss = 0.0645571, step = 601 (0.622 sec)
INFO:tensorflow:global_step/sec: 304.689
INFO:tensorflow:loss = 0.117545165, step = 701 (0.327 sec)
INFO:tensorflow:global_step/sec: 285.719
INFO:tensorflow:loss = 0.14955162, step = 801 (0.351 sec)
INFO:tensorflow:global_step/sec: 282.216
INFO:tensorflow:loss = 0.094014876, step = 901 (0.355 sec)
INFO:tensorflow:Saving checkpoints for 1000 into /var/folders/wj/97_zhf897sn1kf3xf99c5rvw0000gn/T/tmpk
h5lqmqv/model.ckpt.
INFO:tensorflow:Loss for final step: 0.18117064.
```

Out[59]:
```
SKCompat()
```

In [60]:
```
# Calculate accuracies
from sklearn.metrics import accuracy_score

y_pred = dnn_clf.predict(X_test)
print()
print('Accuracy',accuracy_score(y_test, y_pred['classes']))
```

```
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from /var/folders/wj/97_zhf897sn1kf3xf99c5rvw0000gn/T/tmpkh5lqmq
v/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.

Accuracy 0.9575
```

Let's build a DNN with 2 hidden layers from scratch

This is great for understanding!

CONSTRUCTION PHASE

In [61]:
```
# Define hyperparameters and input size

n_inputs = 28*28  # MNIST
n_hidden1 = 1200
n_hidden2 = 600
n_hidden3 = 240
n_hidden4 = 60
n_hidden5 = 24
n_outputs = 10
```

In [62]:
```
# Reset graph
tf.reset_default_graph()
```

In [63]:
```
# Placeholders for data (inputs and targets)
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int64, shape=(None), name="y")
keep_prob=tf.placeholder(tf.float32)
lr=tf.Variable(0.001,dtype=tf.float32)
```

In [64]:
```
# Define neuron layers (ReLU in hidden layers)
# We'll take care of Softmax for output with loss function

def neuron_layer(X, n_neurons, name, activation=None):
    # X input to neuron
    # number of neurons for the layer
    # name of layer
    # pass in eventual activation function

    with tf.name_scope(name):
        n_inputs = int(X.get_shape()[1])

        # initialize weights to prevent vanishing / exploding gradients
        stddev = 2 / np.sqrt(n_inputs)
        init = tf.truncated_normal((n_inputs, n_neurons), stddev=stddev)

        # Initialize weights for the layer
        W = tf.Variable(init, name="weights")
        # biases
        b = tf.Variable(tf.zeros([n_neurons]), name="bias")

        # Output from every neuron
        Z = tf.matmul(X, W) + b
        if activation is not None:
            return activation(Z)
        else:
            return Z
```

In [65]:
```
# Define the hidden layers
with tf.name_scope("dnn"):
    hidden1 = neuron_layer(X, n_hidden1, name="hidden1",
                           activation=tf.nn.relu)
    hidden2 = neuron_layer(hidden1, n_hidden2, name="hidden2",
                           activation=tf.nn.relu)
    hidden3 = neuron_layer(hidden2, n_hidden3, name="hidden3",
                          activation=tf.nn.relu)
    hidden4 = neuron_layer(hidden3, n_hidden4, name="hidden3",
                          activation=tf.nn.relu)
    hidden5 = neuron_layer(hidden4, n_hidden5, name="hidden3",
                          activation=tf.nn.relu)
    logits = neuron_layer(hidden5, n_outputs, name="outputs")
```

In [66]:
```
# Define loss function (that also optimizes Softmax for output):

with tf.name_scope("loss"):
    # logits are from the last output of the dnn
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,
                                                              logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
```

In [67]:
```
# Training step with Gradient Descent

learning_rate = 0.01

with tf.name_scope("train"):
   # optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    training_op = tf.train.AdamOptimizer(lr).minimize(loss)
```

In [68]:
```
tf.nn.dropout(hidden1,0.9,noise_shape=None,seed=None,name=None)
tf.nn.dropout(hidden2,0.9,noise_shape=None,seed=None,name=None)
tf.nn.dropout(hidden3,0.9,noise_shape=None,seed=None,name=None)
tf.nn.dropout(hidden4,0.9,noise_shape=None,seed=None,name=None)
tf.nn.dropout(hidden5,0.9,noise_shape=None,seed=None,name=None)
```

Out[68]:
```
<tf.Tensor 'dropout_4/mul:0' shape=(?, 24) dtype=float32>
```

In [69]:
```
# Evaluation to see accuracy

with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
```

Show graph

In [ ]:
```
```

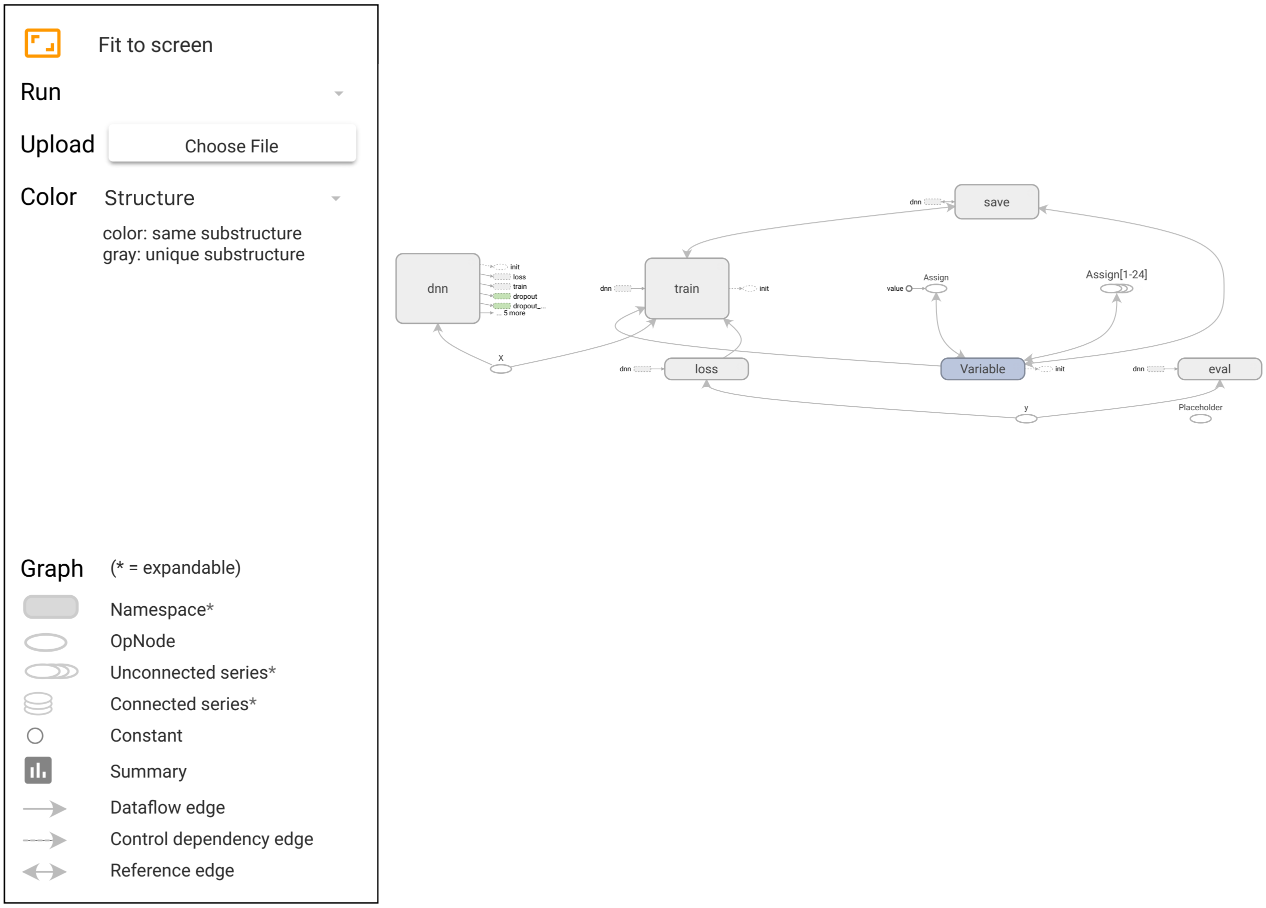
EVALUATION PHASE

Train steps

In [70]:
```
init = tf.global_variables_initializer()
saver = tf.train.Saver()

n_epochs = 25
batch_size = 100

with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        sess.run(tf.assign(lr,0.001*(0.95**epoch)))
        for iteration in range(mnist.train.num_examples // batch_size):
            X_batch, y_batch = mnist.train.next_batch(batch_size)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch,keep_prob:0.9})
        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_val = accuracy.eval(feed_dict={X: mnist.validation.images,
                                            y: mnist.validation.labels})
        print(epoch, "Train accuracy:", acc_train, "Val accuracy:", acc_val)

    save_path = saver.save(sess, "./my_model_final.ckpt") # save model
```

```
0 Train accuracy: 0.98 Val accuracy: 0.9668
1 Train accuracy: 0.99 Val accuracy: 0.974
2 Train accuracy: 0.98 Val accuracy: 0.9746
3 Train accuracy: 1.0 Val accuracy: 0.9782
4 Train accuracy: 1.0 Val accuracy: 0.9772
5 Train accuracy: 1.0 Val accuracy: 0.9782
6 Train accuracy: 1.0 Val accuracy: 0.9798
7 Train accuracy: 0.99 Val accuracy: 0.9784
8 Train accuracy: 1.0 Val accuracy: 0.9804
9 Train accuracy: 1.0 Val accuracy: 0.9796
10 Train accuracy: 1.0 Val accuracy: 0.9832
11 Train accuracy: 1.0 Val accuracy: 0.9828
12 Train accuracy: 1.0 Val accuracy: 0.9824
13 Train accuracy: 1.0 Val accuracy: 0.9834
14 Train accuracy: 1.0 Val accuracy: 0.9844
15 Train accuracy: 1.0 Val accuracy: 0.9852
16 Train accuracy: 1.0 Val accuracy: 0.9824
17 Train accuracy: 1.0 Val accuracy: 0.9846
18 Train accuracy: 1.0 Val accuracy: 0.9852
19 Train accuracy: 1.0 Val accuracy: 0.9864
20 Train accuracy: 1.0 Val accuracy: 0.985
21 Train accuracy: 1.0 Val accuracy: 0.9848
22 Train accuracy: 1.0 Val accuracy: 0.9864
23 Train accuracy: 1.0 Val accuracy: 0.9878
24 Train accuracy: 1.0 Val accuracy: 0.9872
```

In [71]:
```
show_graph(tf.get_default_graph())
```



Evaluate accuracy

In [72]:
```
with tf.Session() as sess:
    saver.restore(sess, "./my_model_final.ckpt") # or better, use save_path
    X_new_scaled = mnist.test.images[:40]
    Z = logits.eval(feed_dict={X: X_new_scaled})
    y_pred = np.argmax(Z, axis=1)

print("Predicted classes:", y_pred)
print("Actual classes:   ", mnist.test.labels[:40])
```

```
INFO:tensorflow:Restoring parameters from ./my_model_final.ckpt
Predicted classes: [7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 4 7 2 7
 1 2 1]
Actual classes:    [7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 4 7 2 7
 1 2 1]
```

In [ ]: