Contents and examples extended from Udacity Deep Learning by Google
https://classroom.udacity.com/courses/ud730/
## OFF-THE-SHELF DEEP LEARNING TOOLS

4x slower than competitors but it’s expected to be improved.

### Table 1. Overview of existing deep learning frameworks, comparing four widely used software solutions.

<table>
<thead>
<tr>
<th></th>
<th>Caffe</th>
<th>Theano</th>
<th>Torch7</th>
<th>TensorFlow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core language</td>
<td>C++</td>
<td>Python, C++</td>
<td>LuaJIT</td>
<td>C++</td>
</tr>
<tr>
<td>Interfases</td>
<td>Python, Matlab</td>
<td>Python</td>
<td>C</td>
<td>Python</td>
</tr>
<tr>
<td>Wrappers</td>
<td>Lasagne, Keras, sklearn-theano</td>
<td></td>
<td></td>
<td>Keras, Pretty Tensor, Scikit Flow</td>
</tr>
<tr>
<td>Programming paradigm</td>
<td>Imperative</td>
<td>Declarative</td>
<td>Imperative</td>
<td>Declarative</td>
</tr>
<tr>
<td>Well suited for</td>
<td>CNNs, Reusing existing models, Computer vision</td>
<td>Custom models, RNNs</td>
<td>Custom models, CNNs, Reusing existing models</td>
<td>Custom models, Parallelization, RNNs</td>
</tr>
</tbody>
</table>

Table 1 in Angermueller et al. (2016) *Molecular Systems Biology*, (12), 878.
INSTALLING

- Install 64-bit Python 3.5 & pip (or Anaconda3-4.2.0-Windows-x86_64)
- Install virtualenv:
  - CMD: pip install virtualenv
  - CMD: pip install virtualenvwrapper-win
- Create virtual environment
  - CMD: mkvirtualenv tensorflowCPU
- Install the CPU-only version of TensorFlow in the virtual environment
  - (TENSOR~) C:\Users\Name> pip install --upgrade https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-0.12.1-cp35-cp35m-win_amd64.whl
The role of the Python code in TensorFlow is to build this external computation graph, and to dictate which parts of the computation graph should be run.

Other heavy lifting such as numerical computations are done outside Python.
MNIST DATA

- 10 labels
- 1 channel
- 28x28 images
TRYING OUT MNIST TUTORIALS IN TENSORFLOW.ORG

GOTO: https://www.tensorflow.org/tutorials/mnist/pros/

Load MNIST Data

```python
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

stores the training, validation, and testing sets

Start TensorFlow InteractiveSession

```python
import tensorflow as tf
esss = tf.InteractiveSession()
```

It allows you to interleave operations which build a computation graph with ones that run the graph.
MODEL1: Build a Softmax Regression Model

- Weight matrix $W$ is a $784 \times 10$ matrix
  - we have 784 input features fully connected to 10 outputs
- Bias vector $b$ is a 10-dimensional vector
  - we have 10 classes

$$y_{-1} = \text{softmax}(w_1^T X + b_1)$$

10 for 10 label values

784 weights for each 10 output + 1 bias

$28 \times 28 = 784$
Placeholders: create nodes for the input images and target output classes.

```python
x = tf.placeholder(tf.float32, shape=[None, 784])
y_ = tf.placeholder(tf.float32, shape=[None, 10])
```

Variables: define & initialize weights $W$ and bias $b$ variables

```python
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
sess.run(tf.global_variables_initializer())
```
Define the regression model:

\[ z = \text{tf.matmul}(x, W) + b \]

Define the loss function: one used to update \( W \) and bias

\[
\text{cross_entropy} = \frac{1}{10} \text{tf.reduce_mean}(\text{tf.nn.softmax_cross_entropy_with_logits}(z, y_))
\]

Applies the softmax on the model's unnormalized model prediction \( (z) \) and sums across all classes.

Takes average over the sums across 10 classes.

\[
\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for} \quad j = 1, \ldots, K.
\]
Train Step

\[
\text{train\_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross\_entropy)}
\]

Steeped gradient descent, with a step length of 0.5, to descend the cross entropy.

Other built-in optimization functions: https://www.tensorflow.org/api_docs/python/train/#optimizers

- TensorFlow actually added set of new operations to the computation graph.
  - Ones to compute gradients,
  - Ones to compute parameter update steps, and
  - Ones apply update steps to the parameters.
TENSORFLOW BACK-PROPAGATION APPROACH

TensorFlow take a computational graph and add additional nodes to the graph that provide a symbolic description of the desired derivatives.

symbol-to-symbol approach to computing derivatives
Training iteration

```python
for i in range(1000):
    batch = mnist.train.next_batch(100)
    train_step.run(feed_dict={x: batch[0], y_: batch[1]})
```

Evaluate model

```python
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

evaluate our accuracy on the test data

```python
print(accuracy.eval(feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```
Get 92% accuracy => very bad for MNIST
Since we're using ReLU neurons, we should initialize them with a slightly positive initial bias to avoid "dead neurons".
Define Convolution and Pooling function

Model:
- Convolution stride of 1 and are zero padded so that the output is the same size as the input (same padding).
- Pooling: max pooling over 2x2 blocks.

```python
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
```
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

*Computes a 2-D convolution given 4-D input and filter tensors.
  tf.nn.conv2d(input, filter, strides, padding,
               use_cudnn_on_gpu=None, data_format=None, name=None)

1. Flattens the filter to a 2-D matrix with shape [filter_height *
   filter_width * in_channels, output_channels].
2. Extracts image patches from the input tensor to form a virtual tensor
   of shape [batch, out_height, out_width, filter_height * filter_width *
   in_channels].
3. For each patch, right-multiplies the filter matrix and the image patch
   vector.

https://www.tensorflow.org/api_docs/python/nn/convolution#conv2d
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                         strides=[1, 2, 2, 1], padding='SAME')

tf.nn.max_pool(value, ksize, strides, padding,
                data_format='NHWC', name=None)

ARGUMENTS:
• **value**: A 4-D Tensor with shape [batch, height, width, channels] and type tf.float32.
• **ksize**: A list of ints that has length >= 4. The size of the window for each dimension of the input tensor.
• **strides**: A list of ints that has length >= 4. The stride of the sliding window for each dimension of the input tensor.
• **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm.
• **data_format**: A string. 'NHWC' and 'NCHW' are supported.
• **name**: Optional name for the operation.
1st Convolutional Layer

- Reshape $x$ to a 4D tensor:
  \[ x_{\text{image}} = \text{tf.reshape}(x, [-1, 28, 28, 1]) \]

- Convolution with 32 features for each 5x5 patch:
  \[ W_{\text{conv1}} = \text{weight_variable}([5, 5, 1, 32]) \]
  \[ b_{\text{conv1}} = \text{bias_variable}([32]) \]

- Apply ReLU and max pooling:
  \[ h_{\text{conv1}} = \text{tf.nn.relu}(\text{conv2d}(x_{\text{image}}, W_{\text{conv1}}) + b_{\text{conv1}}) \]
  \[ h_{\text{pool1}} = \text{max_pool_2x2}(h_{\text{conv1}}) \]

2nd Convolutional Layer

- Convolution with 64 features for each 5x5 patch:
  \[ W_{\text{conv2}} = \text{weight_variable}([5, 5, 32, 64]) \]
  \[ b_{\text{conv2}} = \text{bias_variable}([64]) \]

- Apply ReLU and max pooling:
  \[ h_{\text{conv2}} = \text{tf.nn.relu}(\text{conv2d}(h_{\text{pool1}}, W_{\text{conv2}}) + b_{\text{conv2}}) \]
  \[ h_{\text{pool2}} = \text{max_pool_2x2}(h_{\text{conv2}}) \]
Densely Connected Layer

\[
W_{fc1} = \text{weight\_variable}([7 \times 7 \times 64, 1024])
\]
\[
b_{fc1} = \text{bias\_variable}([1024])
\]

\[
h_{pool2\_flat} = \text{tf.\_reshape}(h_{pool2}, [-1, 7\times7\times64])
\]
\[
h_{fc1} = \text{tf.\_nn.\_relu}(\text{tf.\_matmul}(h_{pool2\_flat}, W_{fc1}) + b_{fc1})
\]

fully-connected layer with 1024 neurons to allow processing on the entire image.
Add Dropout

To reduce overfitting, apply **dropout** before the readout layer.

```python
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

Create placeholder for probability that a neuron's output is kept during dropout.

**tf.nn.dropout** op automatically handles scaling neuron outputs in addition to masking them.
Readout Layer

\[
W_{fc2} = \text{weight\_variable}([1024, 10])
\]
\[
b_{fc2} = \text{bias\_variable}([10])
\]
\[
y_{\text{conv}} = \text{tf.matmul}(h_{fc1\_drop}, W_{fc2}) + b_{fc2}
\]
Train and Evaluate the Model

Almost similar the SoftMax example with the following differences:

• Replace the steepest gradient descent optimizer with the more sophisticated ADAM optimizer.
• Include the additional parameter `keep_prob` in `feed_dict` to control the dropout rate.
• Add logging to every 100th iteration in the training process.

WARNING but it does 20,000 training iterations and may take a while (possibly up to half an hour), depending on your processor.
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y_conv, y_))

train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))

accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

sess.run(tf.global_variables_initializer())

for i in range(20000):
    batch = mnist.train.next_batch(50)

    if i%100 == 0:
        train_accuracy = accuracy.eval(feed_dict={
            x:batch[0], y_: batch[1], keep_prob: 1.0})

        print("step %d, training accuracy %g"%(i, train_accuracy))

        train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})

print("test accuracy %g"%accuracy.eval(feed_dict={
            x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0])))
step 12600, training accuracy 1
step 12700, training accuracy 1
step 12800, training accuracy 0.98
step 12900, training accuracy 1
step 13000, training accuracy 1
step 13100, training accuracy 1
step 13200, training accuracy 1
step 13300, training accuracy 1
step 13400, training accuracy 1
step 13500, training accuracy 1
step 13600, training accuracy 1
step 13700, training accuracy 1
step 13800, training accuracy 1
step 13900, training accuracy 1
step 14000, training accuracy 0.98
step 14100, training accuracy 1
step 14200, training accuracy 1
step 14300, training accuracy 1
step 14400, training accuracy 1
step 14500, training accuracy 1
step 14600, training accuracy 0.98
step 14700, training accuracy 1
step 14800, training accuracy 1
step 14900, training accuracy 1
step 15000, training accuracy 1
step 15100, training accuracy 1
step 15200, training accuracy 1
step 15300, training accuracy 0.98
step 15400, training accuracy 1
step 15500, training accuracy 0.98
step 15600, training accuracy 1
step 15700, training accuracy 1
step 15800, training accuracy 1
step 15900, training accuracy 1
step 16000, training accuracy 1
step 16100, training accuracy 1
step 16200, training accuracy 1
step 16300, training accuracy 1
step 16400, training accuracy 1
step 16500, training accuracy 1
step 16600, training accuracy 1
step 16700, training accuracy 1
step 16800, training accuracy 1
step 16900, training accuracy 1
step 17000, training accuracy 1
step 17100, training accuracy 1
step 17200, training accuracy 1
step 17300, training accuracy 1
step 17400, training accuracy 1
step 17500, training accuracy 1
step 17600, training accuracy 1
step 17700, training accuracy 0.98
step 17800, training accuracy 1
step 17900, training accuracy 1
step 18000, training accuracy 1
step 18100, training accuracy 1
step 18200, training accuracy 1
step 18300, training accuracy 1
step 18400, training accuracy 1

>>> print("test accuracy %s\naccuracy.eval(feed_dict={
    x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0})")
test accuracy 0.9924

>>>
NOTMNIST DATASET

examples of letter "A" in the notMNIST dataset


Multimodal classification problem (10 labels)
Single channel (gray image)
harder task than MNIST dataset
NOTMNIST DATA SET

- Download data and script at
- Store the data and script under
  - C:\Users\NAME\Envs\tensorflowCPU\myscripts
- Open command prompt by typing “cmd” on Windows search
- Assuming pip, virtualenv, python, tensorflow is installed type
  > `mkvirtualenv tensorflowCPU` to create new virtual environment
  or
  > `workon tensorflowCPU` to resume working on project ‘tensorflowCPU’
SO WHAT WOULD YOU NEED TO GET STARTED?

- GPU cluster?
  - Still need high computing power

- Good modeling of DNN
  - Input / Output design
  - Selection of Model Architecture (Deep Feedforward/ Convolution NN/ Autoencoder/ etc.)
  - Selecting Model Training Choices
  - Model Selection - # of neurons in each layer; # of layers
Data preparation

+ Sufficient number of data
  ✗ ( < # of model parameters )

+ Processing raw data
  ✗ Categorical data need to change to numerical
    ✗ One-hot code

✗ Numerical features are typically normalization
   ✗ z-score; log transformations;