Lecture Notes for

Machine Learning in Python

Professor Eric Larson

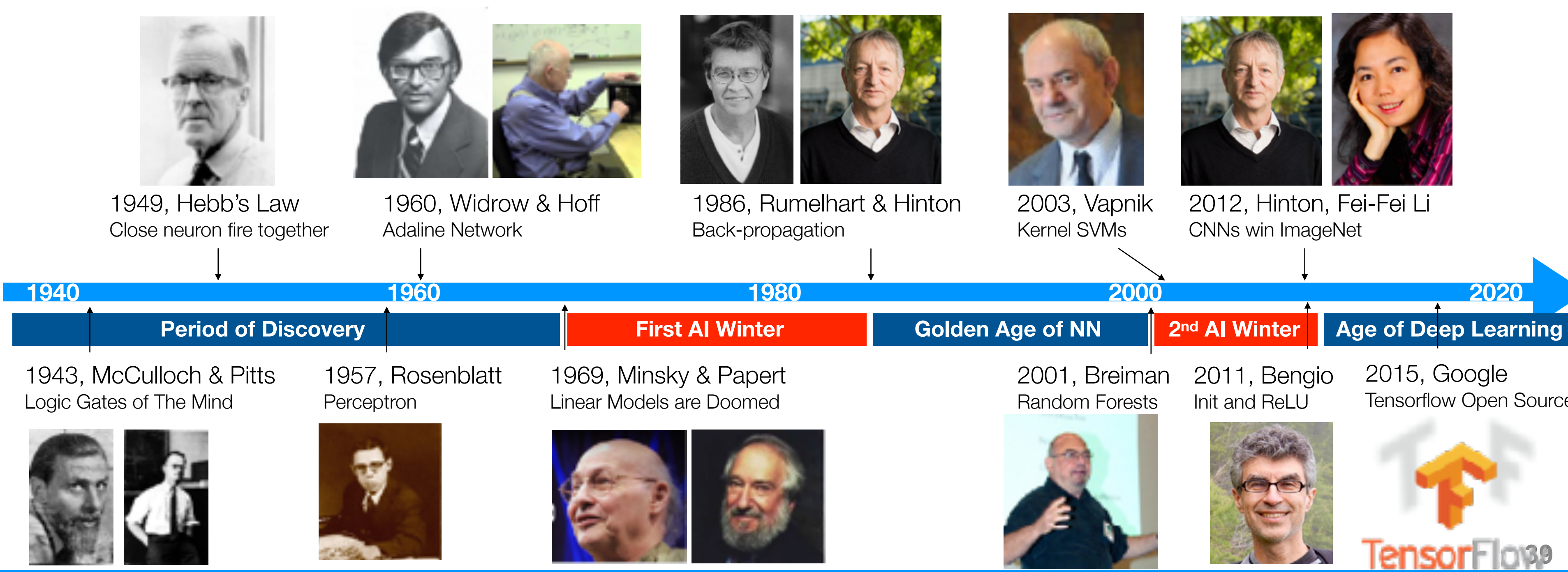
Keras: Wide and Deep Networks

Lecture Agenda

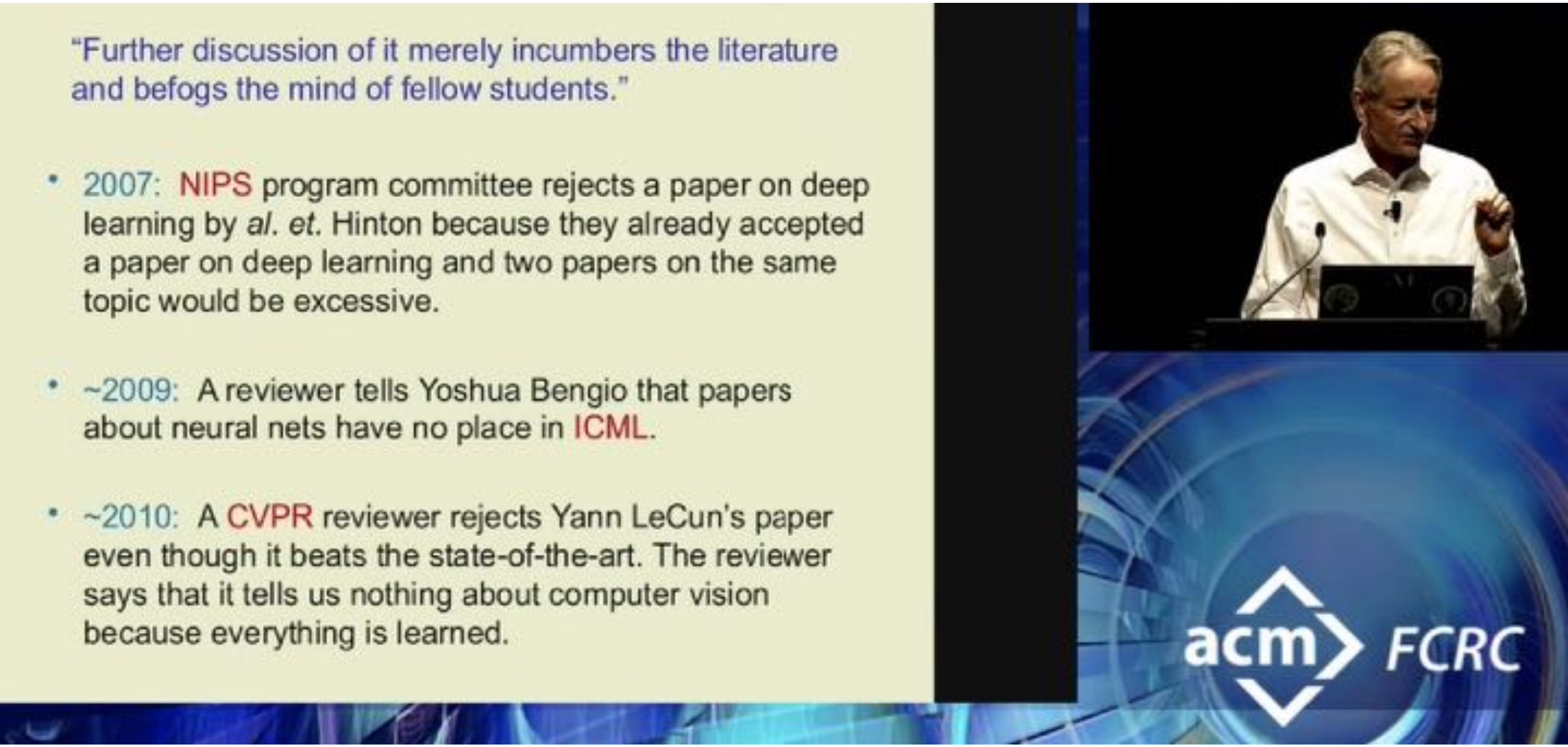
- Logistics: CS 8321 in Spring
  - Grading and lab deadlines
- Get out of the long winter…
- Introduction to TensorFlow
  - Tensors, Namespaces, Numerical methods
  - Deep APIs
- Wide and Deep Networks

Last Time

- Up to this point: back propagation saved AI winter for NN (Hinton and others!)
- 80's, 90's, 2000's: convolutional networks for image processing start to get deeper
    - but back propagation no longer does great job at training them
- SVMs and Random Forests gain traction…
    - The second AI winter begins, research in NN plummets
- 2004: Hinton secures funding from CIFAR in 2004 Hinton rebrands: Deep Learning
- 2006: Auto-encoding and Restricted Boltzmann Machines
- 2007: Deep networks are more efficient when pre-trained
- 2009: GPUs decrease training time by 70 fold…
- 2010: Hinton's students go to internships with Microsoft, Google, and IBM, making their speech recognition systems faster, more accurate and deployed in only 3 months…
- 2012: Hinton Lab, Google, IBM, and Microsoft jointly publish paper, popularity sky-rockets for deep learning methods
- 2011-2013: Ng and Google run unsupervised feature creation on YouTube videos (becomes computer vision benchmark)
- 2012+: Pre-training is not actually needed, just solutions for vanishing gradients (like ReLU, SiLU, initializations, more data, GPUs)

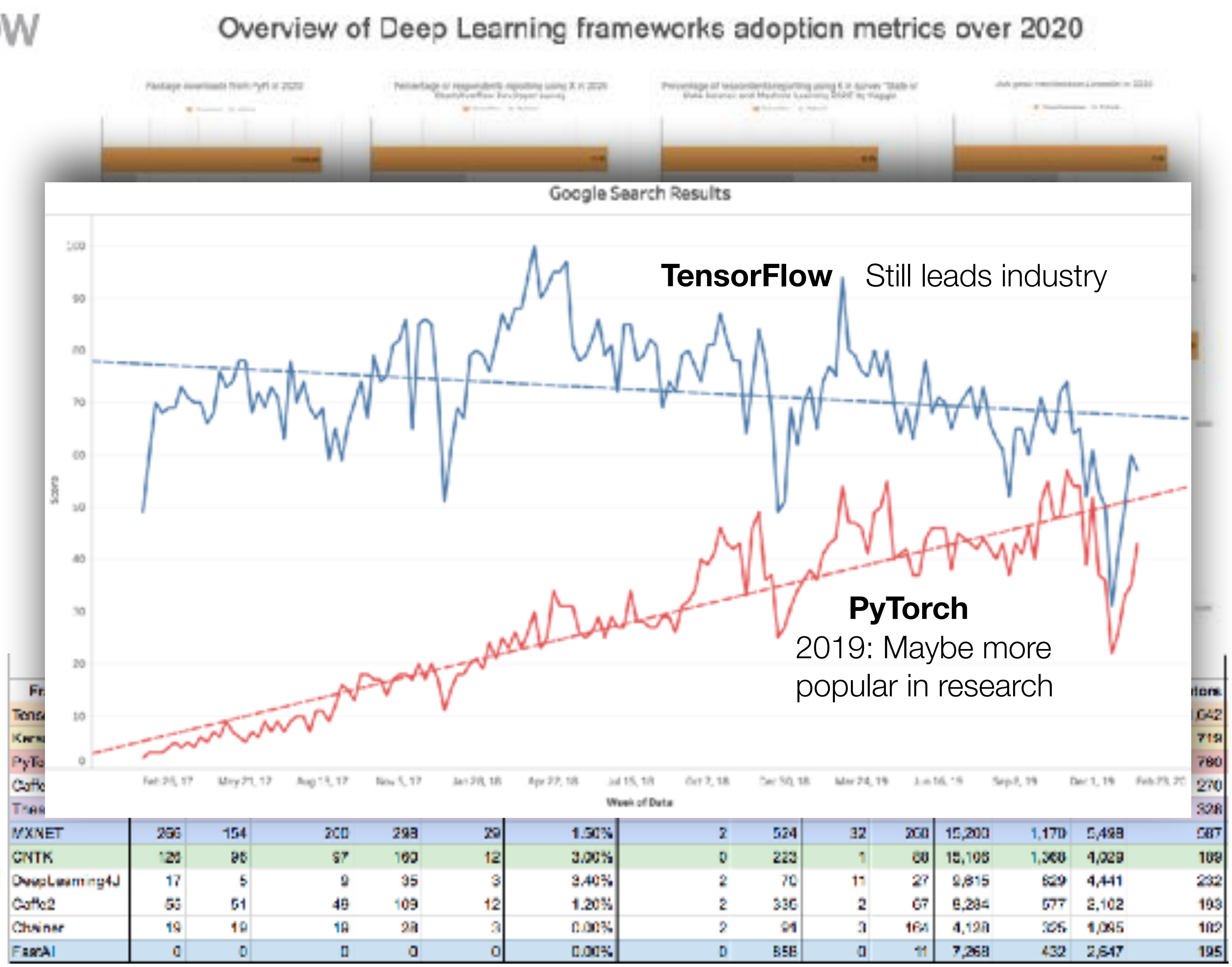


TensorFlow


"Further discussion of it merely incumbers the literature and befogs the mind of fellow students."
• 2007: NIPS program committee rejects a paper on deep learning by al. et. Hinton because they already accepted a paper on deep learning and two papers on the same topic would be excessive.
• ~2009: A reviewer tells Yoshua Bengio that papers about neural nets have no place in ICML.
• ~2010: A CVPR reviewer rejects Yann LeCun's paper even though it beats the state-of-the-art. The reviewer says that it tells us nothing about computer vision because everything is learned.
acm FCRC

Options for Deep Learning Toolkits

1. TensorFlow
2. Keras
3. PyTorch
4. Caffe
5. theano
6. Apache mxnet
7. Microsoft CNTK
8. DL4J
9. Caffe2
10. Chainer
11. fast.ai



Tensorflow

- Open sourced library from Google
- Second generation release from Google Brain
  - supported for Linux, Unix, Windows
  - Also works on Android/iOS
- Released November 9th, 2015
    - (this class first offered January 2016)

Programmatic creation

- Most toolkits use python to build a **computation graph** of operations
  - Build up computations
  - Execute computations

- **Most Toolkits Support:**
  - tensor creation
  - functions on tensors
  - automatic differentiation

- Tensors are just multidimensional arrays
  - like in Numpy
    - scalars (biases and constants)
    - vectors (e.g., input arrays)
    - 2D matrices (e.g., images)
    - 3D matrices (e.g., color images)
    - 4D matrices (e.g., batches of color images)

Tensor basic functions

```
a = tf.constant(5.0)

b = tf.constant(6.0)

c = a * b
```

- Easy to define operations on tensors

Numpy	**TensorFlow**
`a = np.zeros((2,2)); b = np.ones((2,2))`	`a = tf.zeros((2,2)), b = tf.ones((2,2))`
`np.sum(b, axis=1)`	`tf.reduce_sum(a,reduction_indices=[1])`
`a.shape`	`a.get_shape()`
`np.reshape(a, (1,4))`	`tf.reshape(a, (1,4))`
`b * 5 + 1`	`b * 5 + 1`
`np.dot(a,b)`	`tf.matmul(a, b)`
`a[0,0], a[:,0], a[0,:]`	`a[0,0], a[:,0], a[0,:]`

Also supports convolution: `tf.nn.conv2d, tf.nn.conv3D`

https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Tensor neural network functions

- Easy to define operations on layers of networks

```
• relu(features, name=None)
• bias_add(value, bias, data_format=None, name=None)
• sigmoid(x, name=None)
• tanh(x, name=None)
• conv2d(input, filter, strides, padding)
• conv1d(value, filters, stride, padding)
• conv3d(input, filter, strides, padding)
• conv3d_transpose(value, filter, output_shape, strides)
• sigmoid_cross_entropy_with_logits(logits, targets)
• softmax(logits, dim=-1)
• log_softmax(logits, dim=-1)
• softmax_cross_entropy_with_logits(logits, labels, dim=-1)
```

- Each function created *knows its gradient*
- **Automatic Differentiation** is just **chain rule**
- But… lets start simple…

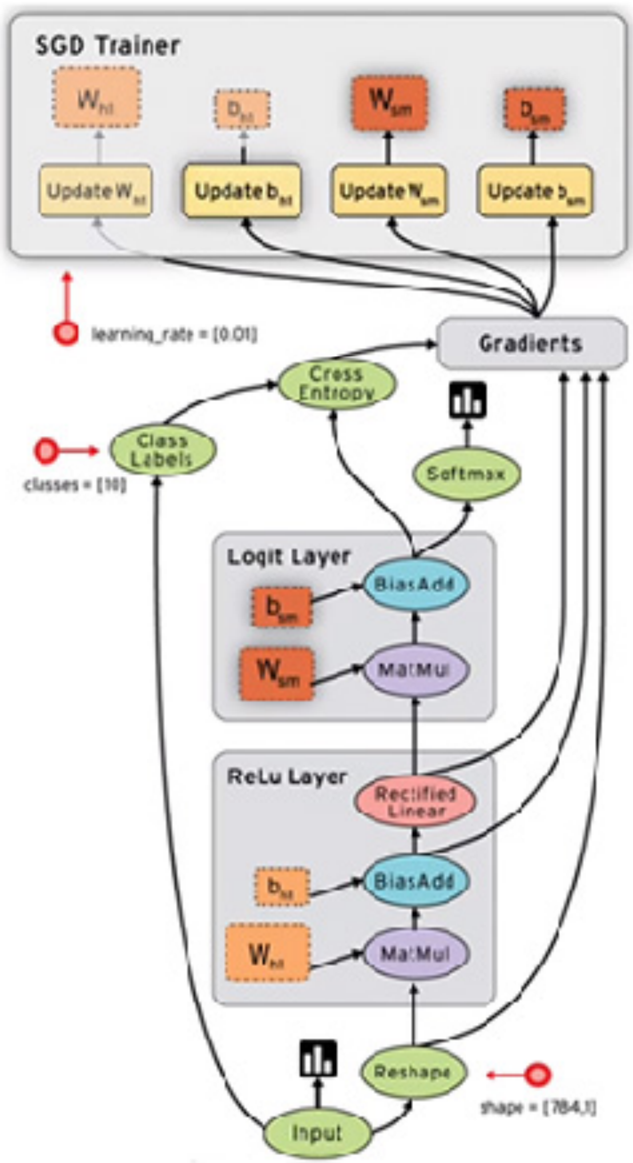
Tensor function evaluation

```
a = tf.constant(5.0)

b = tf.constant(6.0)

c = a * b

with tf.Session() as sess:
    print(sess.run(c))
    print(c.eval())
```

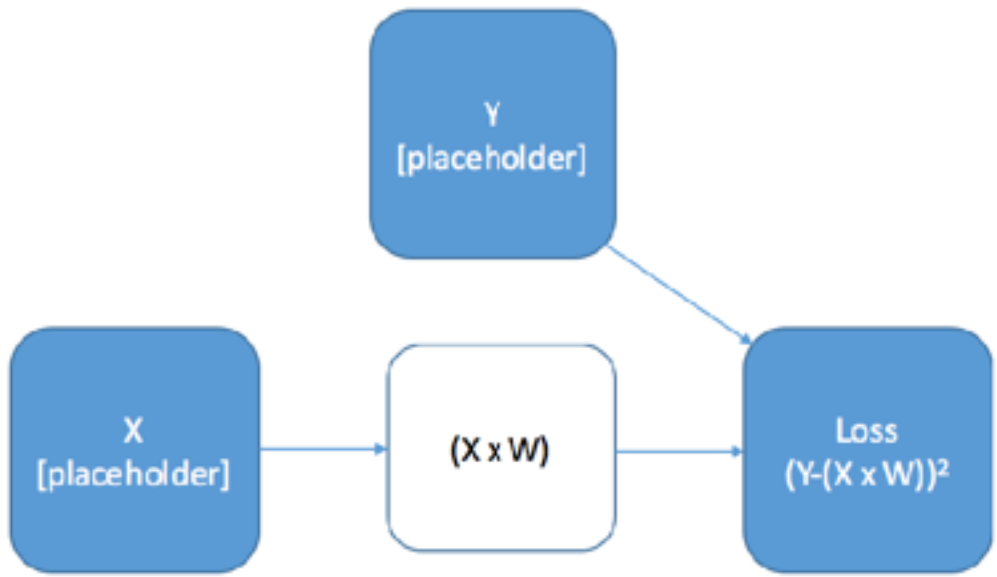
output = 30

- Easy to define operations on tensors
- Nothing evaluated until you define a session and tell it to evaluate it
- Session defines configuration of execution
  - like GPU versus CPU

https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf



- Nothing evaluated until you define a session and tell it to evaluate it
- Session defines configuration of execution
    - like GPU versus CPU



http://www.kdnuggets.com/2016/07/multi-task-learning-tensorflow-part-1.html

http://www.datasciencecentral.com/profiles/blogs/google-open-source-tensorflow

Tensorflow with Linear Regression

- Simple Computation Graph

```
import tensorflow as tf
X = tf.Placeholder()
y = tf.Placeholder()
```

$$J(W,b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (Wx_i + b))^2$$

```
W = tf.Variable("weights", (1,1), initializer=tf.random_normal_initializer())
b = tf.Variable("bias", (1,), initializer=tf.constant_initializer(0.0))

y_pred = tf.matmul(X,W) + b
loss = tf.reduce_sum((y-ypred)**2/n_samples)
```

1. **Setup** Variables and computations

```
opt = tf.train.AdamOptimizer()
opt_operation = opt.minimize(loss)
```

2. Add **optimization** operation to computation graph
   Adjusts variables to minimize loss with automatic differentiation

```
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())
    sess.run([opt_operation], feed_dict={X: X_data, y: y_data})
```

3. **Run graph operation** once, -> one optimization update on all variables

https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

TensorFlow Mini-batching

```
opt = tf.train.AdamOptimizer()
opt_operation = opt.minimize(loss)

with tf.Session() as sess:
  # Initialize Variables in graph
  sess.run(tf.initialize_all_variables())
  # Gradient descent loop for 500 steps
  for _ in range(500):
    # Select random minibatch
    indices = np.random.choice(n_samples, batch_size)
    X_batch, y_batch = X_data[indices], y_data[indices]
    # Do gradient descent step
    _, loss_val = sess.run([opt_operation, loss], feed_dict={X: X_batch, y: y_batch})
```

- Example shown is **graph execution**
  - Build up computations and Execute computations when instructed
  - Makes it sometimes **hard to debug** but its **fast**
- Alternative: **eager execution** (we won't cover this)

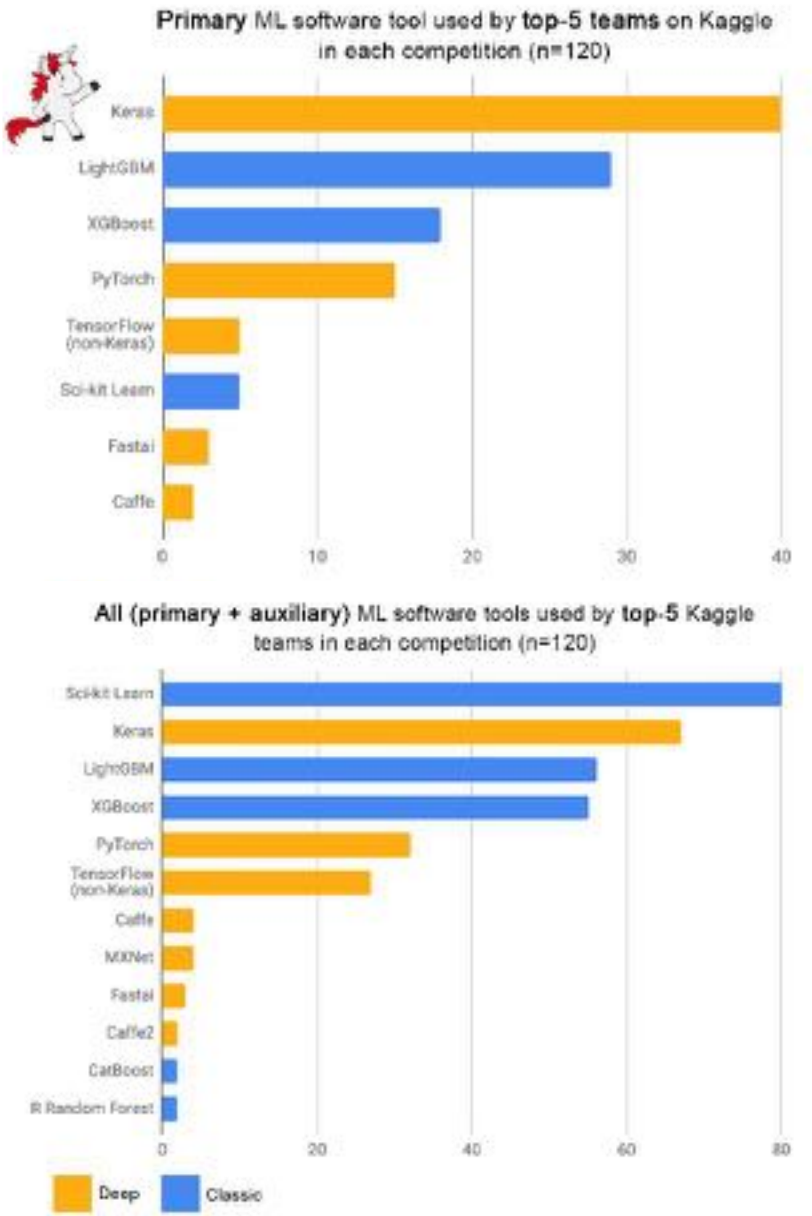
https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Tensor-flow Simplification

- **Self Test**: Can the syntax be simplified?
  - (A) **Yes**, we could write a generic mini-batch optimization computation graph, then use it for arbitrary inputs
  - (B) **Yes**, but we lose control over the optimization procedures
  - (C) **Yes**, but we lose control over the NN models that we can create via Tensorflow
  - (D) **None of the above**

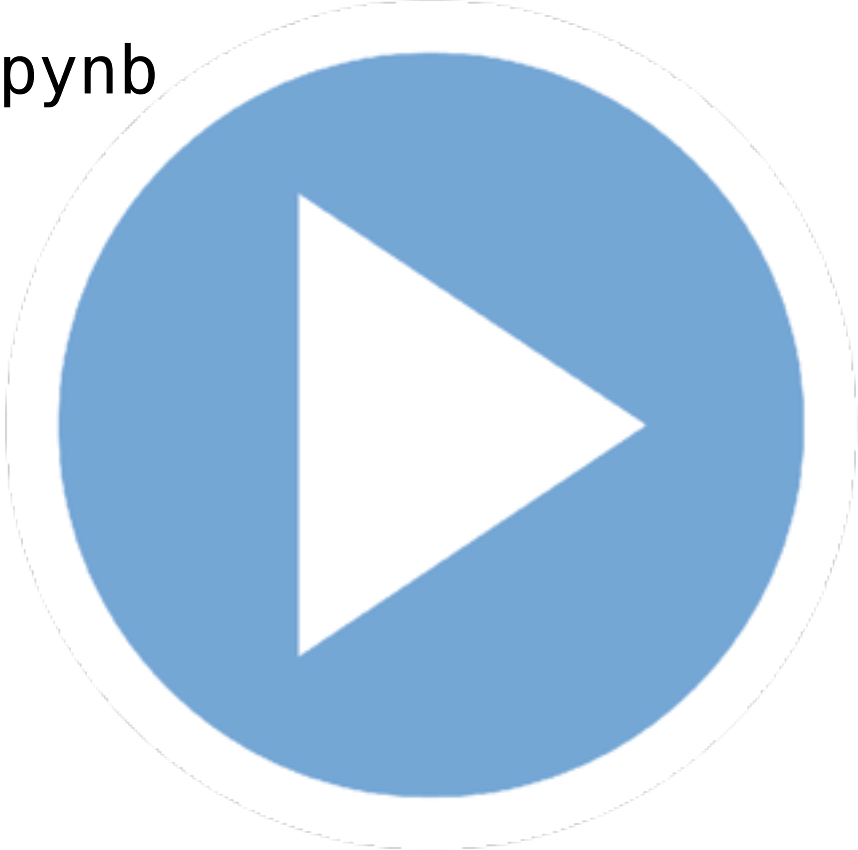
Keras Programming Interfaces

- Keras Sequential API
  - great for simple, feed forward models
- Keras Functional API
  - build models through series of nested functions
  - each “function” represents an operation in the NN
- Keras Classes (Inheritance)
  - good for more advanced functionality



Demo

`10. Keras Wide and Deep.ipynb`



Reinventing the MLP
Wheel

Other tutorials:

https://github.com/jtoy/awesome-tensorflow

https://elitedatascience.com/keras-tutorial-deep-learning-in-python

Or do a Google search!!! They are everywhere!!!

Make me slow down if I go too fast!!

Categorical Feature Embeddings

- One hot encoded data can be made dense through a matrix multiplication

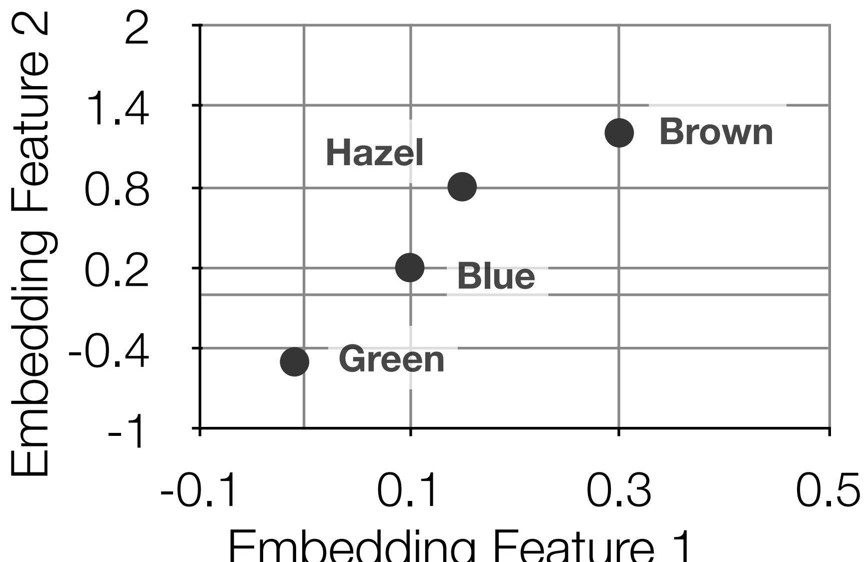
Trainable Matrix in Network

Reduced Dimensions (rows) × Num Categories (columns)

0.1	0.3	-0.01	0.15
0.2	1.2	-0.5	0.8

Eye Color

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{array}{l} \text{Blue} \rightarrow 0 \\ \text{Brown} \rightarrow 1 \\ \text{Green} \rightarrow 2 \\ \text{Hazel} \rightarrow 3 \end{array}$$



Computationally: there is no need to one hot encode eye color, we can just use the integer to index into the **embedding matrix**

“Finish” Demo

`10. Keras Wide and Deep.ipynb`

Reinventing the MLP Wheel

Other tutorials:

https://github.com/jtoy/awesome-tensorflow

https://elitedatascience.com/keras-tutorial-deep-learning-in-python

Or do a Google search!!! They are everywhere!!!

Make me slow down if I go too fast!!