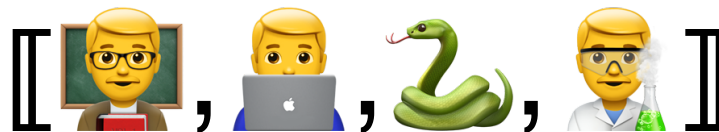


# Lecture Notes for **Machine Learning in Python**



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

# Class Overview, by topic

Table Data  
Visualization

Numpy, Pandas, Seaborn  
Overviews with some in-depth discussion

Dimension  
Reduction and  
Image Processing

Scikit-learn, Scikit Image,  
Intuition only, Some mathematics

Linear and  
Logistic  
Regression

Numpy, Recreate API for Scikit-learn  
Detailed mathematics for simple optimization  
intuition for advanced optimization

Neural Networks  
and Back Prop.

Numpy  
Detailed mathematics for NN operations

Wide and Deep  
Networks

Convolutional  
Networks

Recurrent  
Networks

Keras, Tensorflow  
Intuition, Detailed implement.

Ethics in  
Language Models

ConceptNet  
Case studies

# Last Time

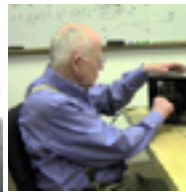
- Up to this point: back propagation saved AI winter for NN (Hinton and others!)
- 80's, 90's, 2000's: convolutional networks for image processing start to get deeper
  - but back propagation no longer does great job at training them
- SVMs and Random Forests gain traction...
  - The second AI winter 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)



1949, Hebb's Law  
Close neuron fire together



1960, Widrow & Hoff  
Adaline Network



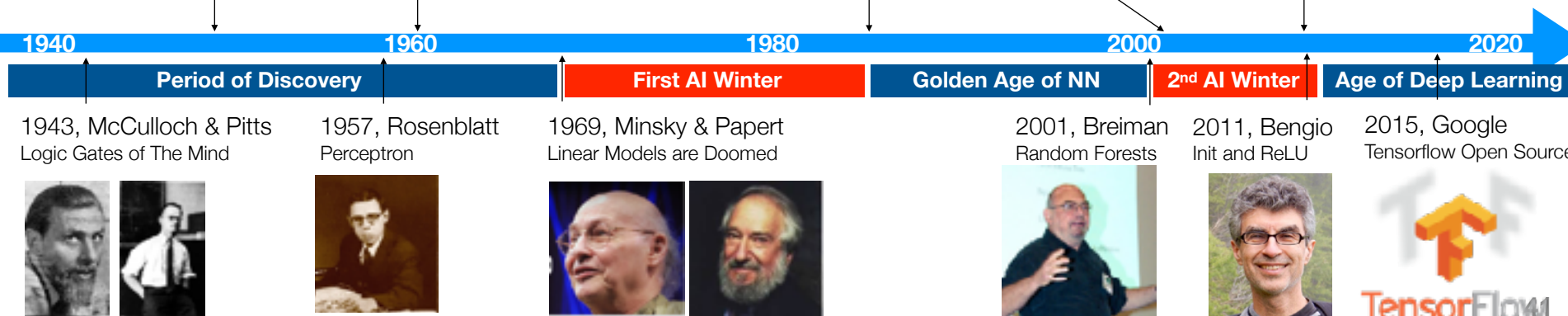
1986, Rumelhart & Hinton  
Back-propagation



2003, Vapnik  
Kernel SVMs



2012, Hinton, Fei-Fei Li  
CNNs win ImageNet



# 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.

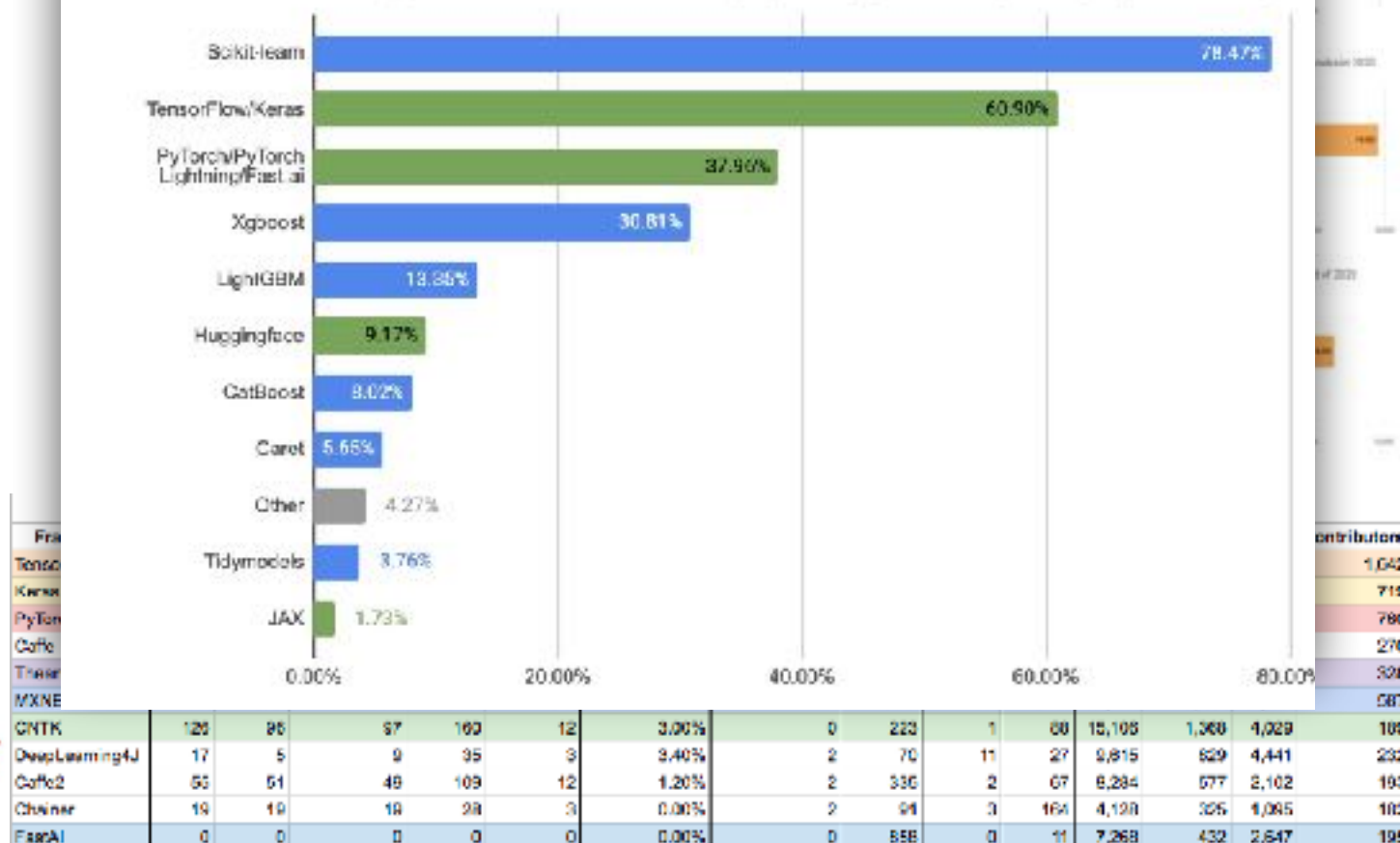


# Options for Deep Learning Toolkits

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

Overview of Deep Learning frameworks adoption metrics over 2020

2022 Machine Learning & Data Science Survey by Kaggle: library usage (N=14,531)



# 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

- Easy to define operations on tensors

```
a = tf.constant(5.0)
```

```
b = tf.constant(6.0)
```

```
c = a * b
```

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

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

# 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

```
import tensorflow as tf
```

```
a = tf.constant(5.0)
```

```
b = tf.constant(6.0)
```

```
c = a*b
```

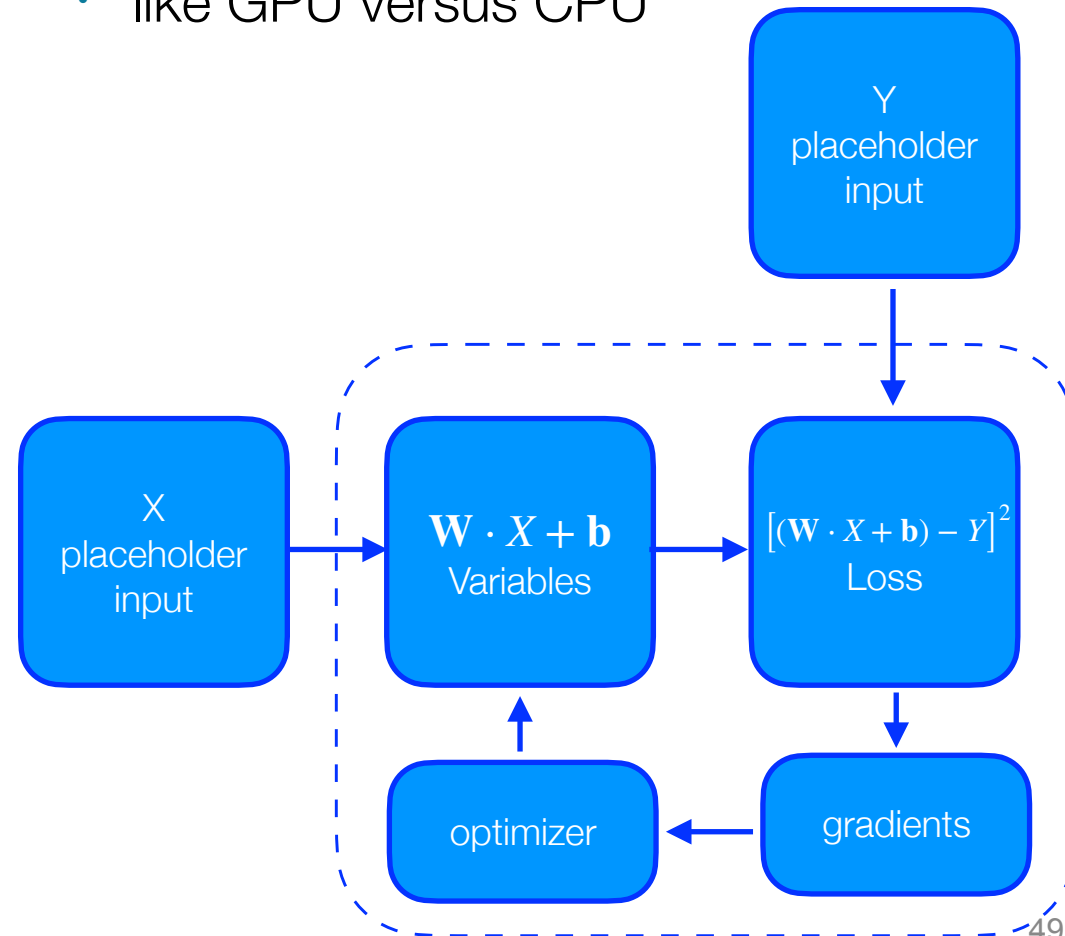
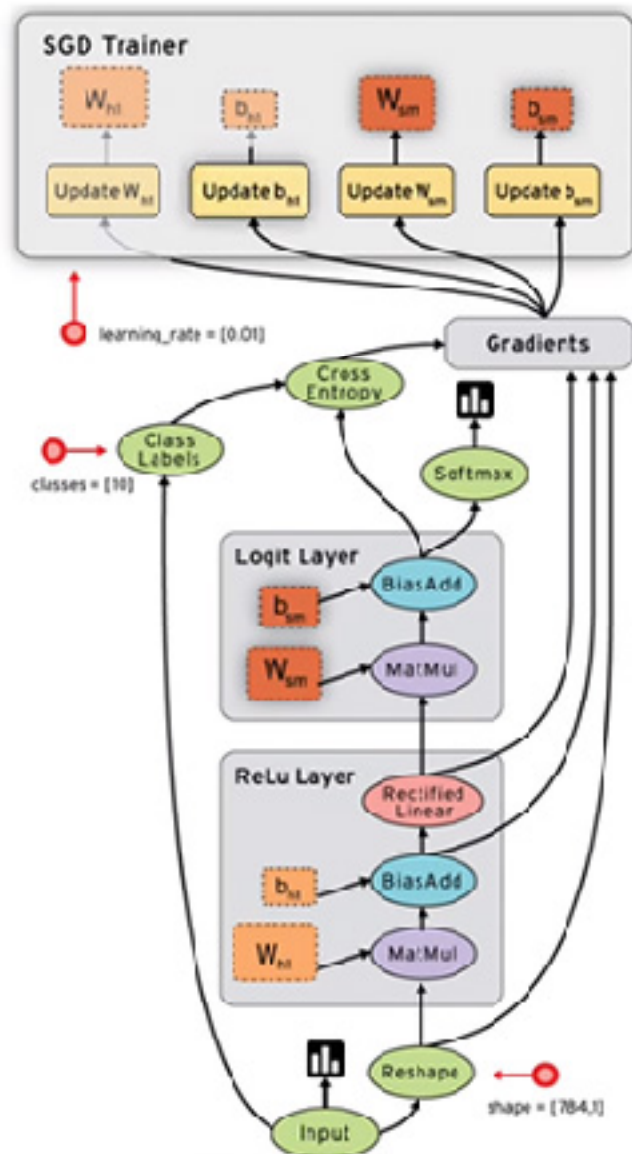
```
with tf.Session() as sess:  
    print(sess.run(c))  
    print(c.eval())
```

output = 30

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

# Computation Graph

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



# Tensorflow with Linear Regression

- Simple Computation Graph

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

$$J(\mathbf{W}) = \frac{1}{N} \sum_i^N (y^{(i)} - \underbrace{(\mathbf{W} \cdot \mathbf{x}^{(i)} + \mathbf{b})}_{y_{pred}})^2$$

```
W = tf.Variable("weights", (1,num_features),
               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-y_pred)**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 (W, b) 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_numpy, y: y_numpy})
```

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:
    sess.run(tf.initialize_all_variables())
    sess.run([opt_operation], feed_dict={X:X_numpy, y: y_numpy})

    for _ in range(500):
        indices = np.random.choice(n_samples, batch_size)
        X_batch, y_batch = X_numpy[indices], y_numpy[indices]

        _, 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)

# Tensorflow Simplification

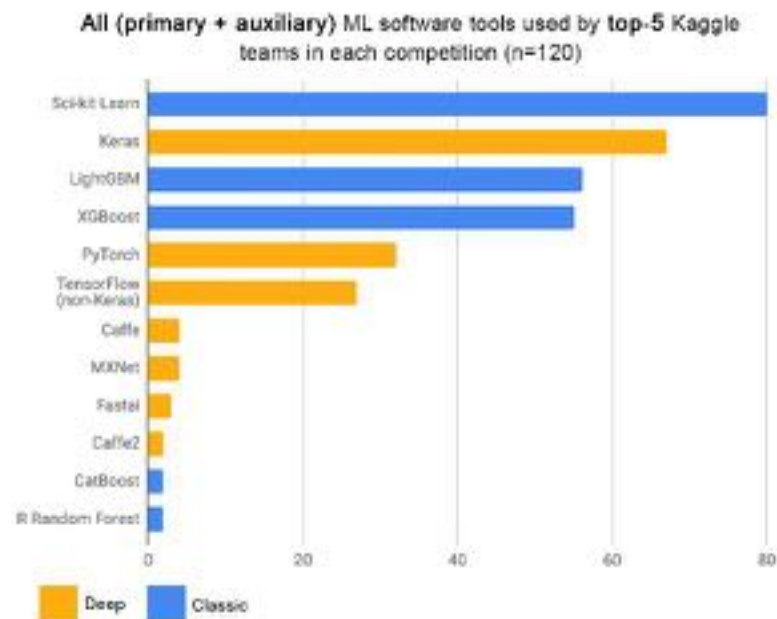
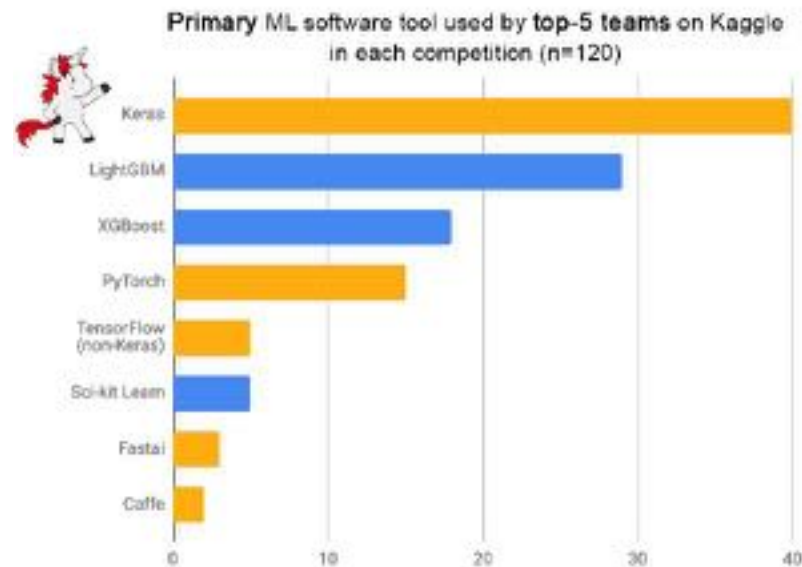
- **Self Test:** Can the syntax be simplified?
  - (A) **Yes**, we could write a generic mini-batch optimization computation graph, then use it for arbitrary inputs
  - (B) **Yes**, but we lose control over the optimization procedures
  - (C) **Yes**, but we lose control over the NN models that we can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



# Keras Programming Interfaces

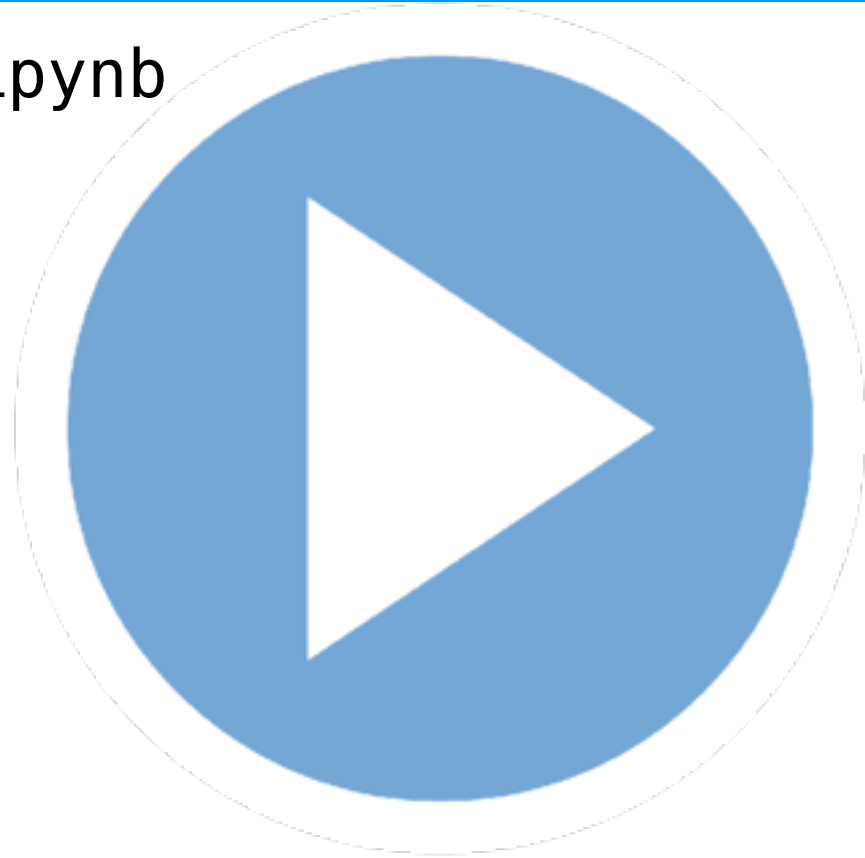
- **Keras Sequential API**
  - great for simple, feed forward models
- **Keras Functional API**
  - build models through series of nested functions
  - each “function” represents an operation in the NN
- **Keras Classes (Inheritance)**
  - good for more advanced functionality

```
from tensorflow import keras
```



## 10. Keras Wide and Deep.ipynb

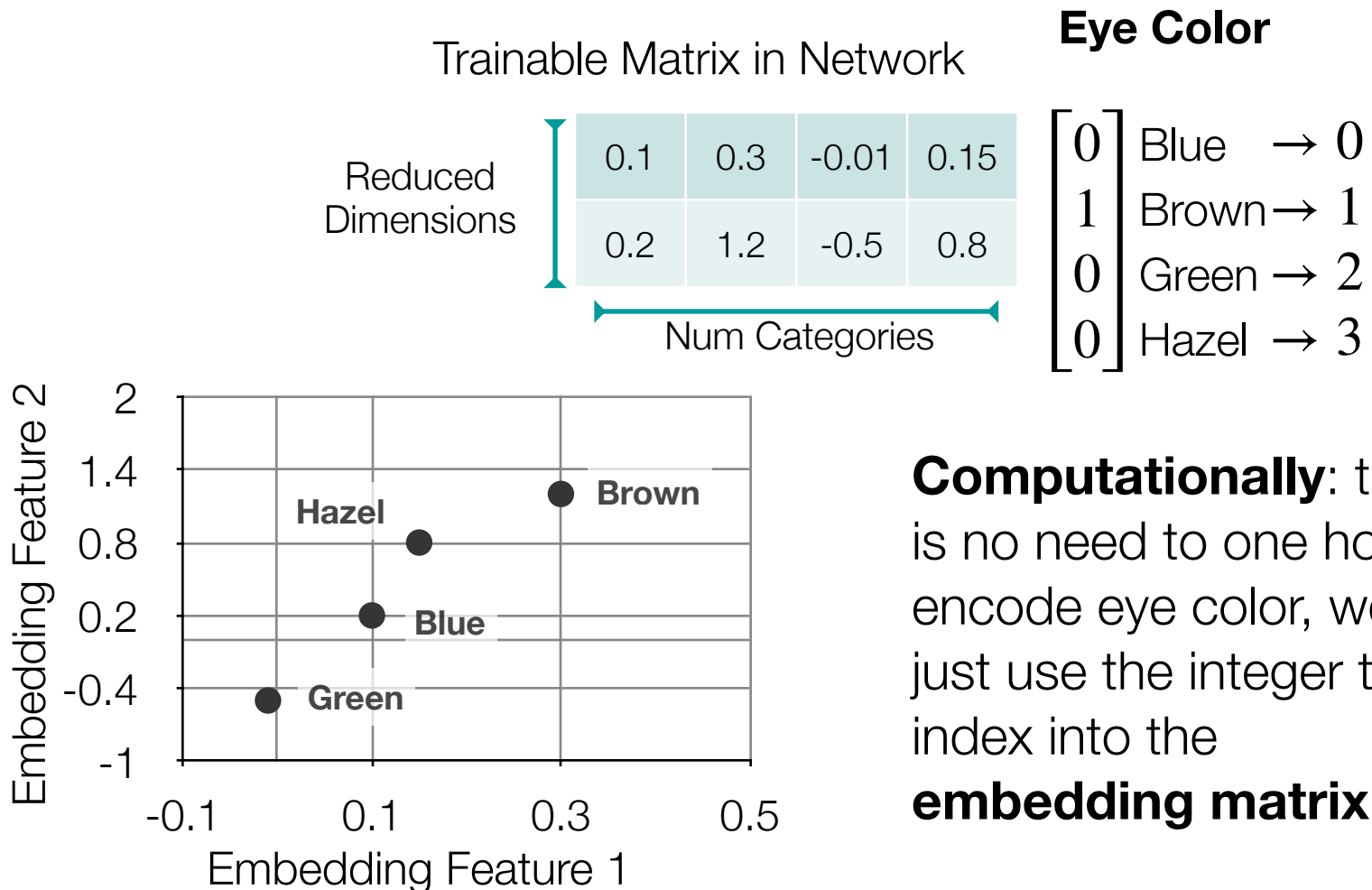
Reinventing the MLP  
Wheel



Make me slow down if I go too fast!!

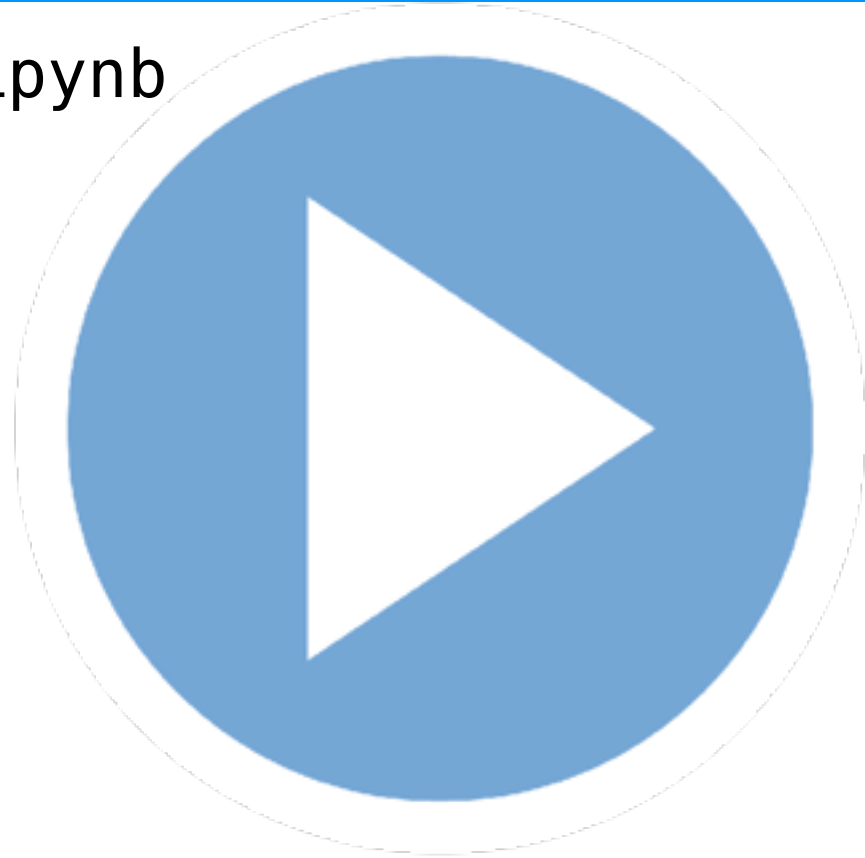
# Categorical Feature Embeddings

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



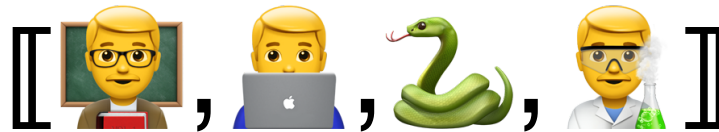
## 10. Keras Wide and Deep.ipynb

Reinventing the MLP  
Wheel



Make me slow down if I go too fast!!

# Lecture Notes for **Machine Learning in Python**



Professor Eric Larson  
**Wide and Deep Networks**

# Lecture Agenda

- Finish Keras Demo
- Wide and Deep Networks
- Wide and Deep Town Hall (if time)

# Class Overview, by topic

Table Data  
Visualization

Numpy, Pandas, Seaborn  
Overviews with some in-depth discussion

Dimension  
Reduction and  
Image Processing

Scikit-learn, Scikit Image,  
Intuition only, Some mathematics

Linear and  
Logistic  
Regression

Numpy, Recreate API for Scikit-learn  
Detailed mathematics for simple optimization  
intuition for advanced optimization

Neural Networks  
and Back Prop.

Numpy  
Detailed mathematics for NN operations

Wide and Deep  
Networks

Convolutional  
Networks

Recurrent  
Networks

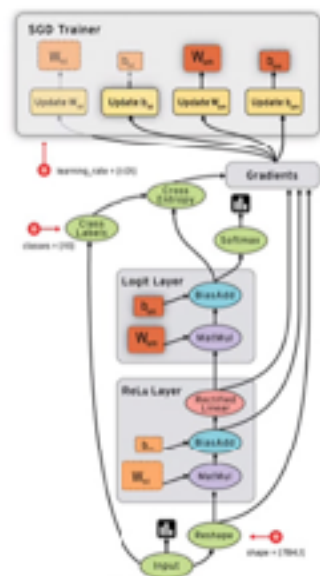
Keras, Tensorflow  
Intuition, Detailed implement.

Ethics in  
Language Models

ConceptNet  
Case studies



# Last Time: Tensorflow and Keras



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

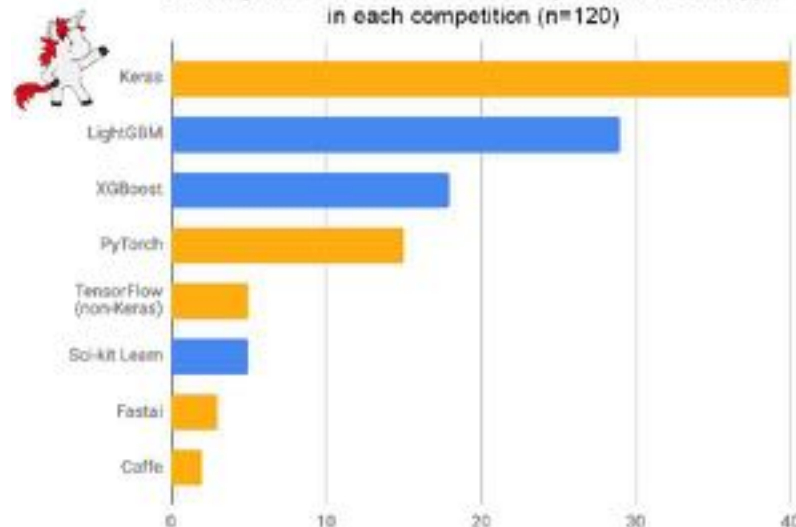


<http://www.kdnuggets.com/2016/07/multi-task-learning-tensorflow-part-1.html>

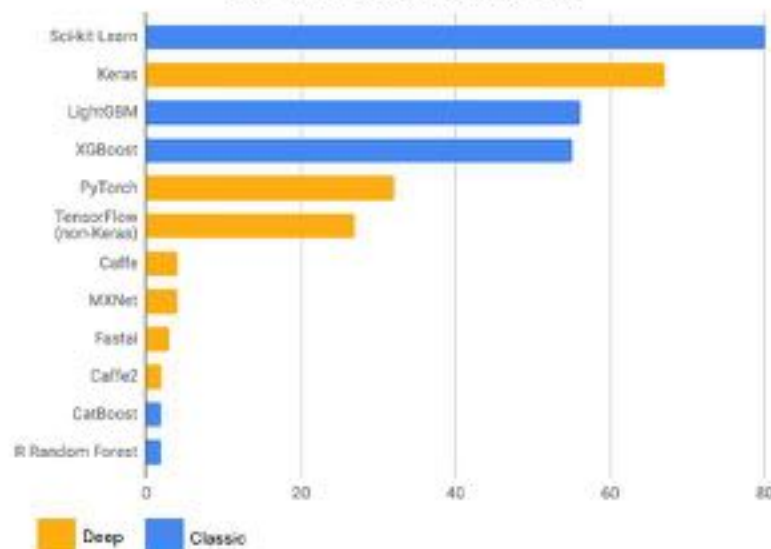
<http://www.datasciencecentral.com/profiles/blogs/google-open-source-tensorflow>

- 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

Primary ML software tool used by top-5 teams on Kaggle in each competition (n=120)



All (primary + auxiliary) ML software tools used by top-5 Kaggle teams in each competition (n=120)



# Categorical Feature Embeddings

- One hot encoded data can be “embedded” through a matrix multiplication (column select)

Trainable Matrix in Network

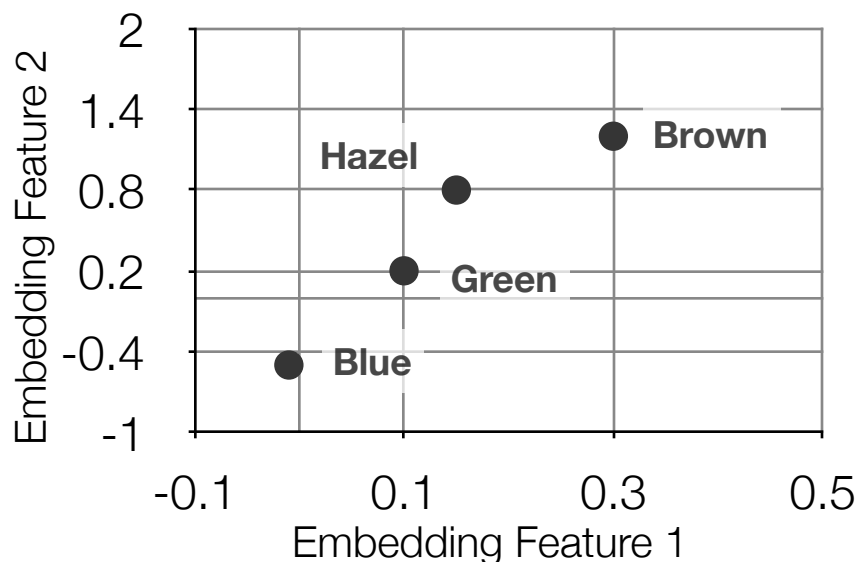
Reduced Dimensions

0.1	0.3	-0.01	0.15
0.2	1.2	-0.5	0.8

Num Categories

Eye Color

0	Green	→ 0
1	Brown	→ 1
0	Blue	→ 2
0	Hazel	→ 3



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