Lecture Notes for

Machine Learning in Python

[👨‍🏫, 👨‍💻, 🐍, 👨‍🔬]

Professor Eric Larson

Tensorflow, Wide and Deep Networks

Lecture Agenda

- Logistics:
  - CS 8321 in Spring (renaming to 5/7325)
  - Grading and lab deadlines
- Review: Get out of the long winter…
- Introduction to TensorFlow
  - Tensors, Tf.Data
  - 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

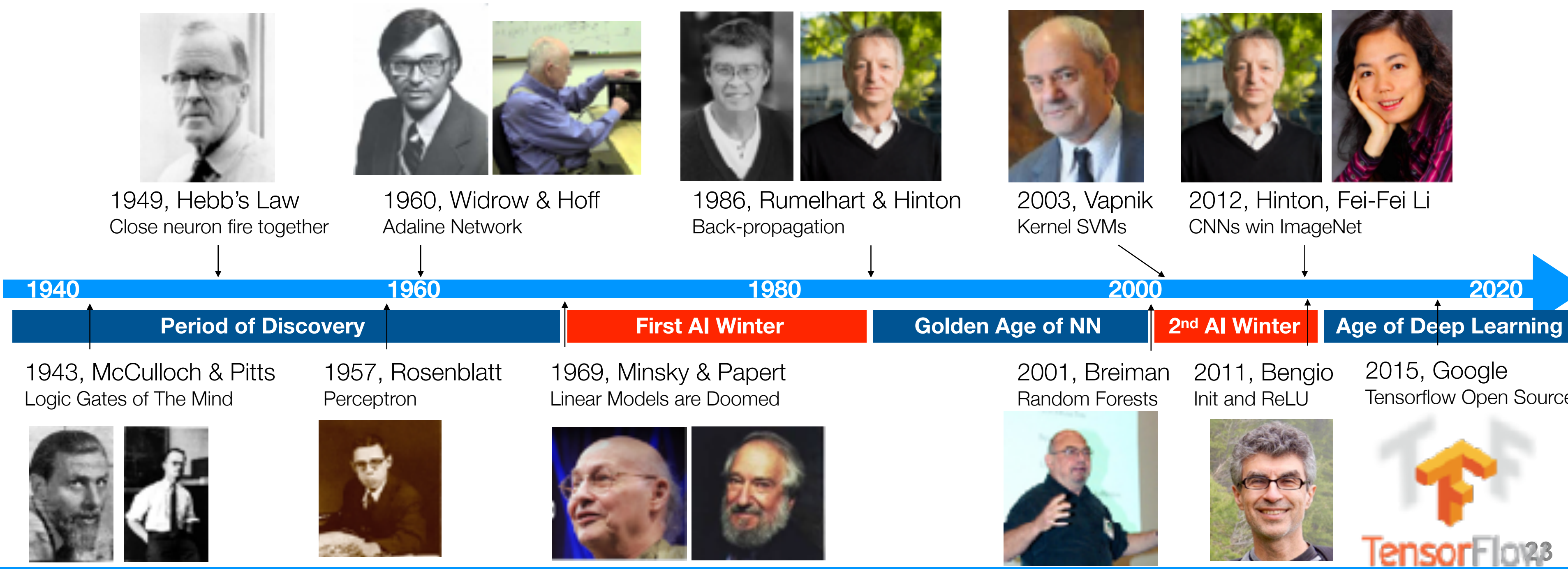
Neural Networks and Back Prop.
Numpy
Detailed mathematics for NN operations

Wide and Deep Networks | **Convolutional Networks** | **Recurrent Networks**
Keras, Tensorflow
Intuition, Detailed implement.

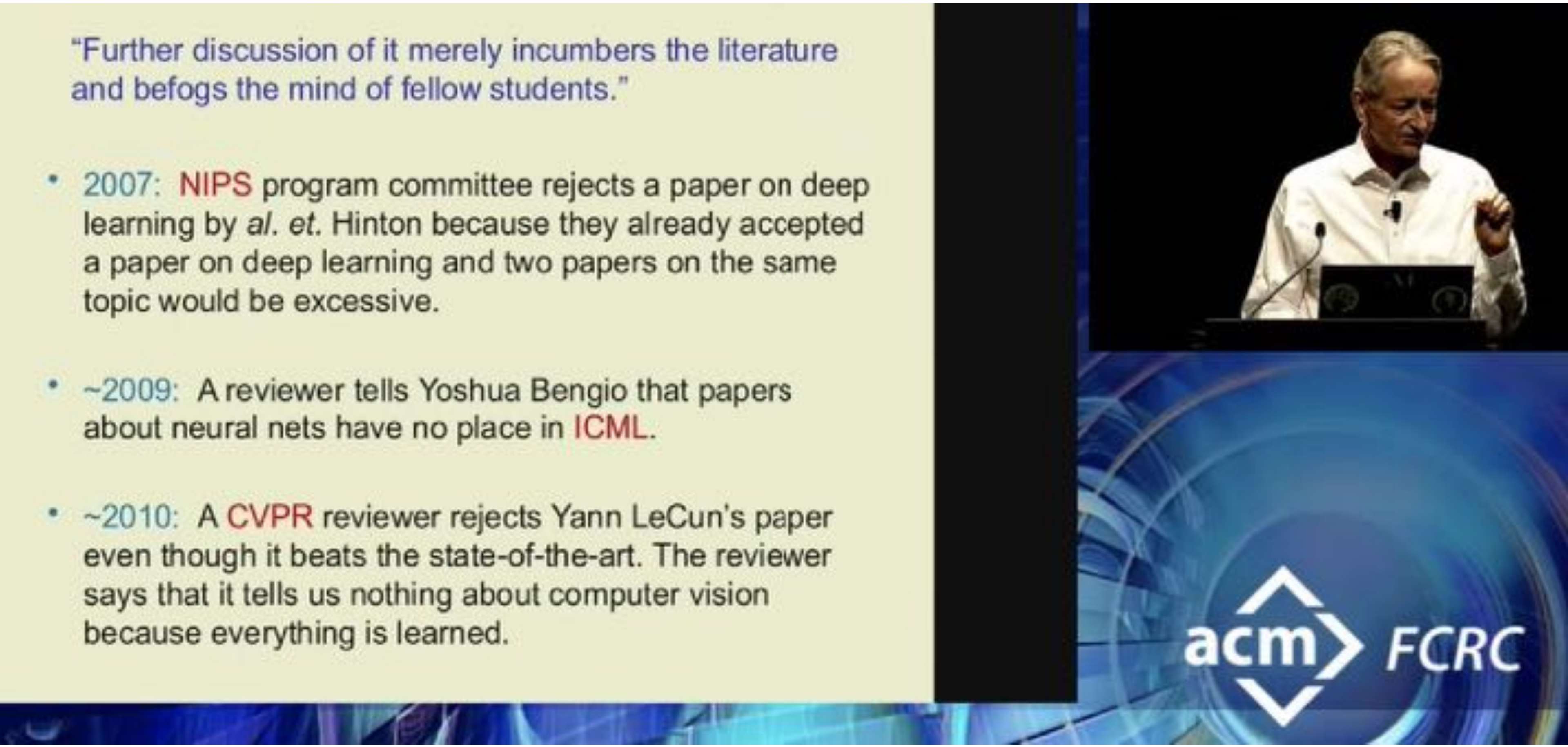
Ethics in Language Models
ConceptNet
Case studies

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)

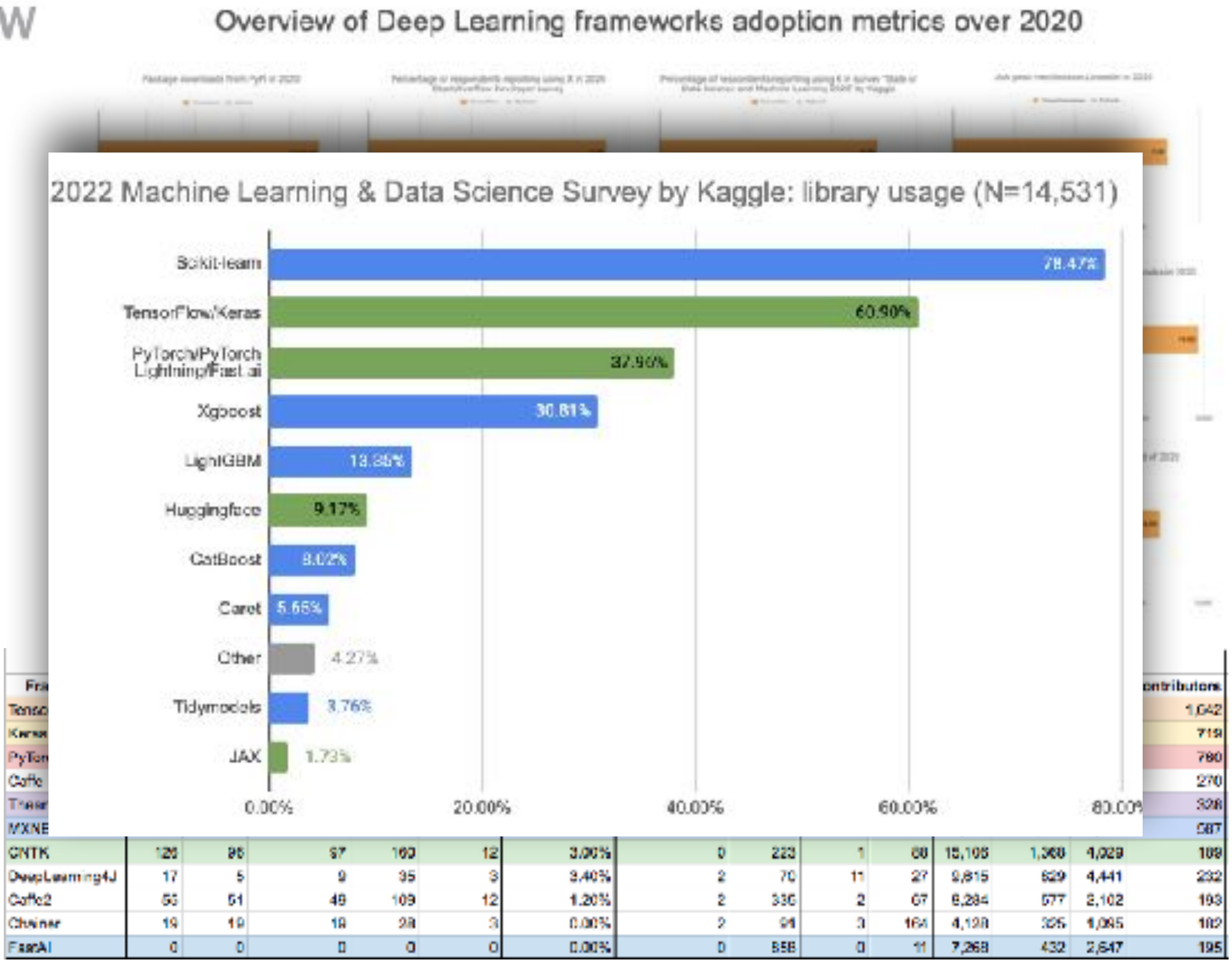


TensorFlow


"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

- Easy to define operations on tensors

```
a = tf.constant(5.0)

b = tf.constant(6.0)

c = a * b
```

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

```
import tensorflow as tf

a = tf.constant(5.0)
b = tf.constant(6.0)

c = a*b

// pre tensorflow 2
with tf.Session() as sess:
    print(sess.run(c))
    print(c.eval())

// post tensorflow 2
print(c)
```

output = 30

- Easy to define operations on tensors
  - constant
  - variables
  - placeholders
- Nothing evaluated until the output is needed
- Tensorflow now run in default eager execution

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

Computation Graph with Code

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

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

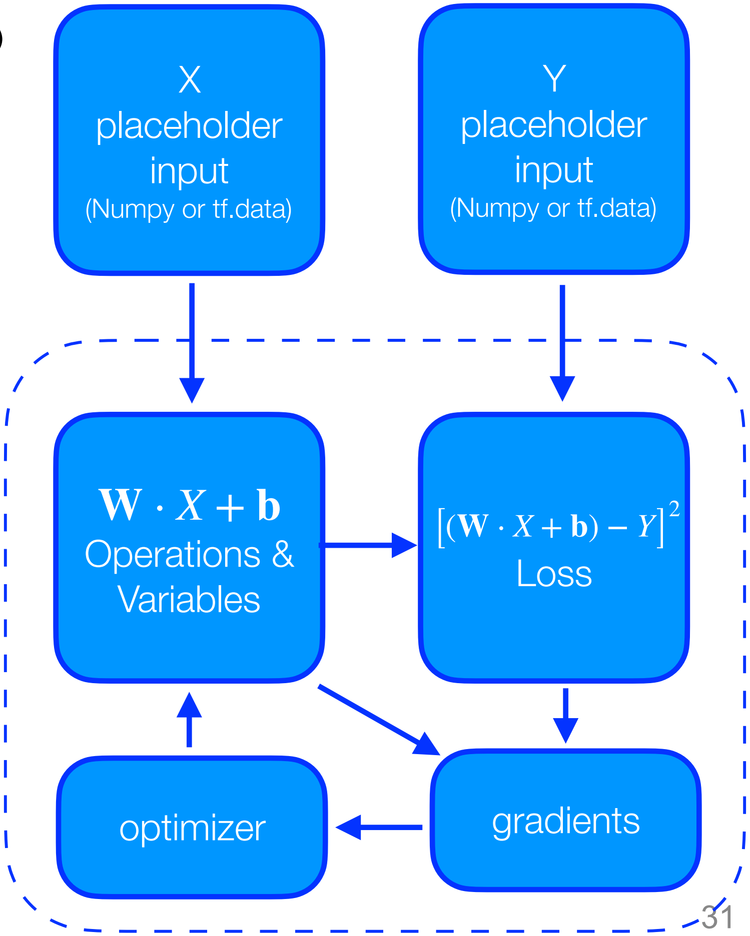
1. **Setup** Variables and computations

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

def feedforward(X,y):
     y_pred = tf.matmul(X,W) + b
     loss = tf.reduce_sum((y-y_pred)**2)
     return loss
```

2. Add **optimization** operation to computation graph
   Adjusts variables (W, b) to minimize loss with auto differentiation

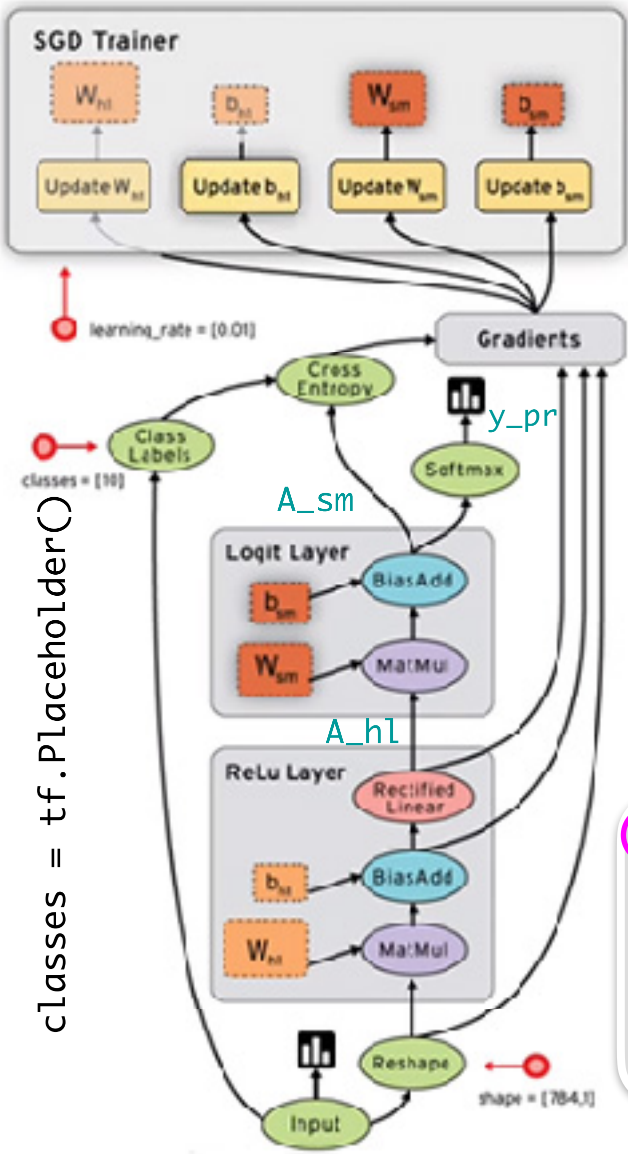
```
opt = tf.train.AdamOptimizer()

tf.initialize_all_variables()
feedforward(X_numpy, y_numpy)
      … track gradients and variables …
opt.apply_gradients(zip(grads, train_vars))
```

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



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

Computation Graph, Two Layer Network




```
Input = tf.Placeholder() # size is 28x28
Input = tf.Reshape(Input, [784,1])
classes = tf.Placeholder()

W_sm = tf.Variable(…)
b_sm = tf.Variable(…)
W_hl = tf.Variable(…)
b_hl = tf.Variable(…)
```

```
trainable_variables =
        [W_sm,b_sm,W_hl,b_hl]
```

```
def model_forward(Input):
  A_hl = tf.relu( tf.matmul(Input,W_hl) + b_hl )
  A_sm = tf.matmul(A_hl,W_sm) + b_sm
  return A_sm
```

```
y_pr = tf.softmax(A_sm)
loss = tf.sparse_softmax_cross_entropy_with_logits
```

```
opt = tf.train.SGDOptimizer(learning_rate=0.01)
```

tf.data

```
for features, labels in train_data:
   with tf.GradientTape( ) as tape:
      yhat = model_forward(features)
      loss_val = loss(labels, yhat)
   grads = tape.gradient(loss_val, trainable_variables)
   opt.apply_gradients(zip(grads, trainable_variables))
```

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

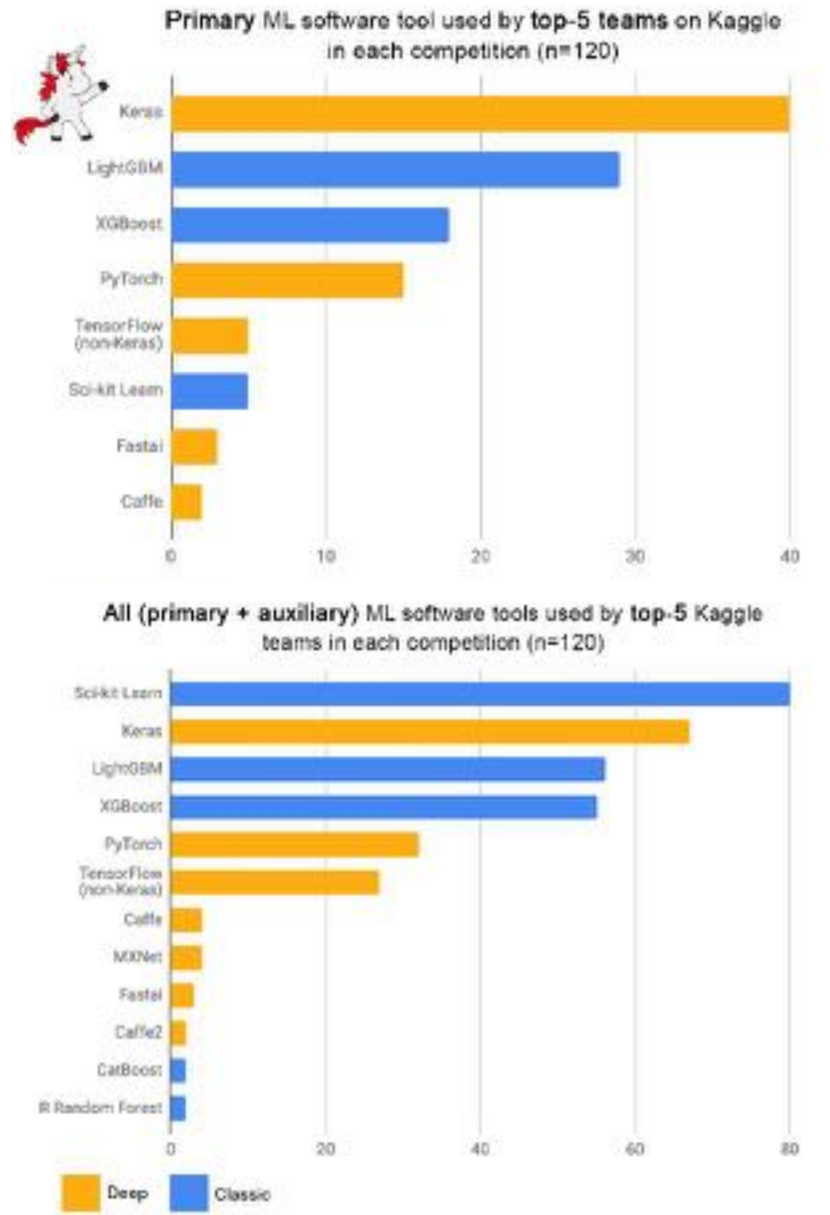
Tensorflow Simplification

- **Self Test**: Can the syntax be simplified?
  - (A) **Yes**, we could write a generic mini-batch optimization computation graph, then use it for arbitrary graph instructions
  - (B) **Yes**, but we need to learn the Keras API, which can be mixed with tensorflow operations
  - (C) **Yes**, but we need to understand how to access the gradients to apply them, a lot like PyTorch
  - (D) **All 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
```



Demo

Reinventing the MLP Wheel

`10. Keras Wide and Deep.ipynb`

`10a. Keras Wide and Deep as TFData.ipynb`

Make me slow down if I go too fast!!

Lecture Notes for

Machine Learning in Python

[👨‍🏫, 👨‍💻, 🐍, 👨‍🔬]

Professor Eric Larson

Feature Spaces + Wide and Deep Networks