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
- 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.

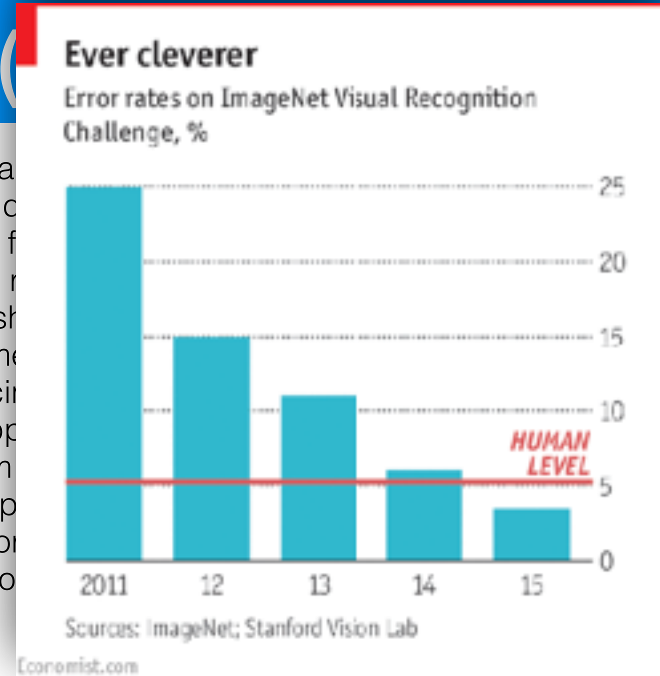
Ethics in Language Models

ConceptNet
Case studies

Machine Learning Timeline (

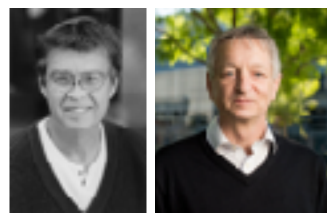
- **ImageNet competition occurs**
- **Second place**: 26.2% error rate
- **First place**:
  - From Hinton's lab, uses convolutional network with ReLU and dropout
  - 15.2% error rate
- Computer vision adopts deep learning with convolutional neural networks en mass

"I ha … first
hand … e
last f …),
so I r … ef
as sh … on
come … s
pacir … ust
happ
from
skep … t as
a cor … er
Visio



1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network



1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs

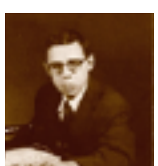
2012, Hinton, Fei-Fei Li
CNNs win ImageNet

1940 | 1960 | 1980 | 2000 | 2020

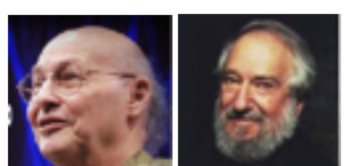
Period of Discovery | First AI Winter | Golden Age of NN | 2nd AI Winter | Age of Deep Learning

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron



1969, Minsky & Papert
Linear Models are Doomed



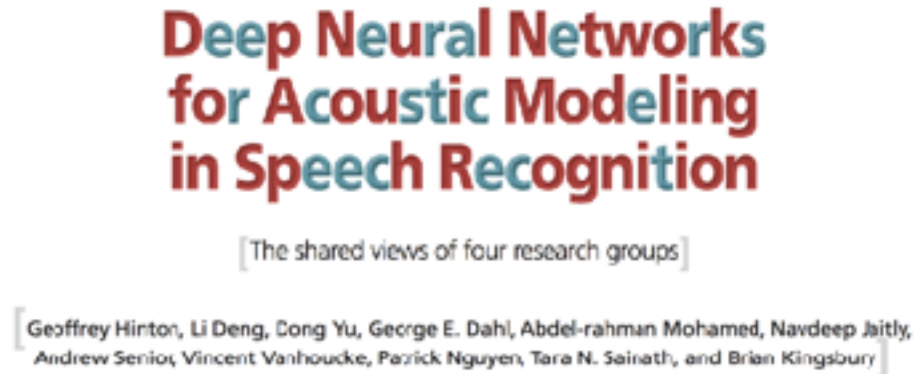
2001, Breiman
Random Forests

2011, Bengio
Init and ReLU

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

Machine Learning Timeline (Neural Nets)

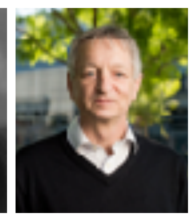
- 2012: Hinton Lab, Google, IBM, and Microsoft jointly publish paper, popularity for deep learning methods increases



https://www.cs.toronto.edu/~gdahl/papers/deepSpeechReviewSPM2012.pdf

1949, Hebb's Law
Close neuron fire together


1960, Widrow & Hoff
Adaline Network

1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs

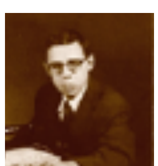

2012, Hinton, Fei-Fei Li
CNNs win ImageNet

1940 | 1960 | 1980 | 2000 | 2020

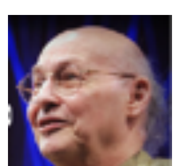
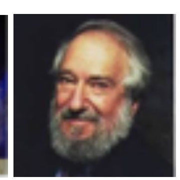
Period of Discovery | First AI Winter | Golden Age of NN | 2nd AI Winter | Age of Deep Learning

1943, McCulloch & Pitts
Logic Gates of The Mind


1957, Rosenblatt
Perceptron



1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

2011, Bengio
Init and ReLU

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

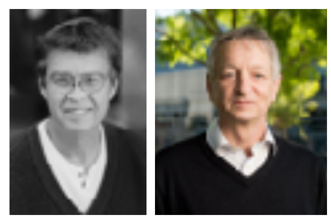
Machine Learning Timeline (Neural Nets)

- 2013: Andrew Ng and Google (BrainTeam)
  - run unsupervised feature creation on YouTube videos (becomes computer vision benchmark)

The work resulted in unsupervised neural net learning of an unprecedented scale - 16,000 CPU cores powering the learning of a whopping 1 billion weights. The neural net was trained on Youtube videos, entirely without labels, and learned to recognize the most common objects in those videos.

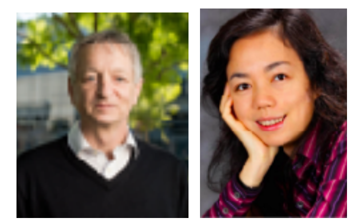
1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network



1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs



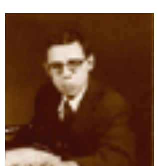
2012, Hinton, Fei-Fei Li
CNNs win ImageNet

1940 1960 1980 2000 2020

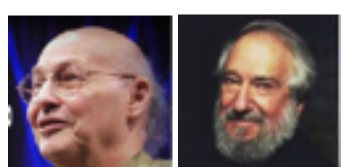
Period of Discovery | First AI Winter | Golden Age of NN | 2nd AI Winter | Age of Deep Learning

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron



1969, Minsky & Papert
Linear Models are Doomed



2001, Breiman
Random Forests

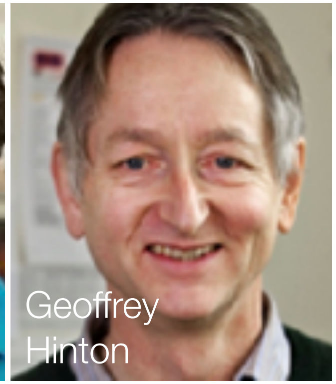
2011, Bengio
Init and ReLU

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

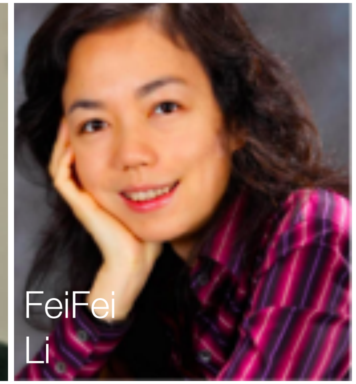
A summary of the Deep Learning people:

Stayed at Univ. Montreal
Advises IBM

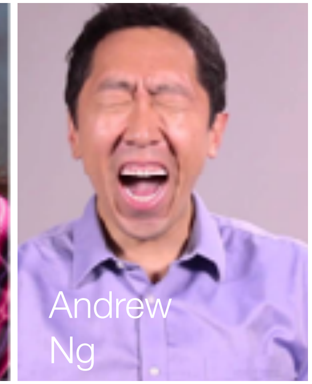
Heads Facebook
AI Team



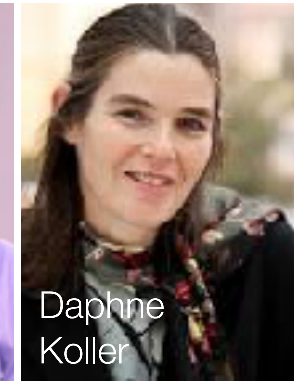
Univ. Toronto
Google



Stanford (HAI)
Former Chief Scien.,
AI/MLGoogle Cloud



Coursera
Baidu
Google



Stanford
Founded Coursera
MacArthur Genius

Made Deep Learning Instruction Accessible

deep learning


doi:10.1038/nature14539

ning

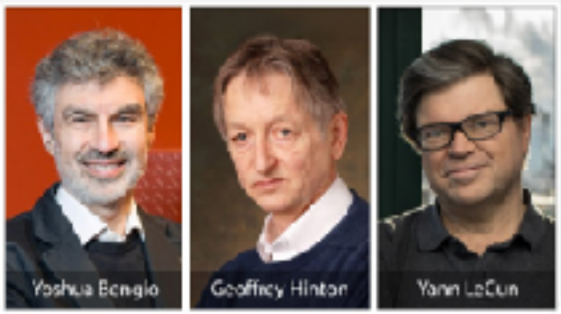
Geoffrey Hinton[4,5]

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and

Credit for Deep Learning

Official ACM @TheOfficialACM

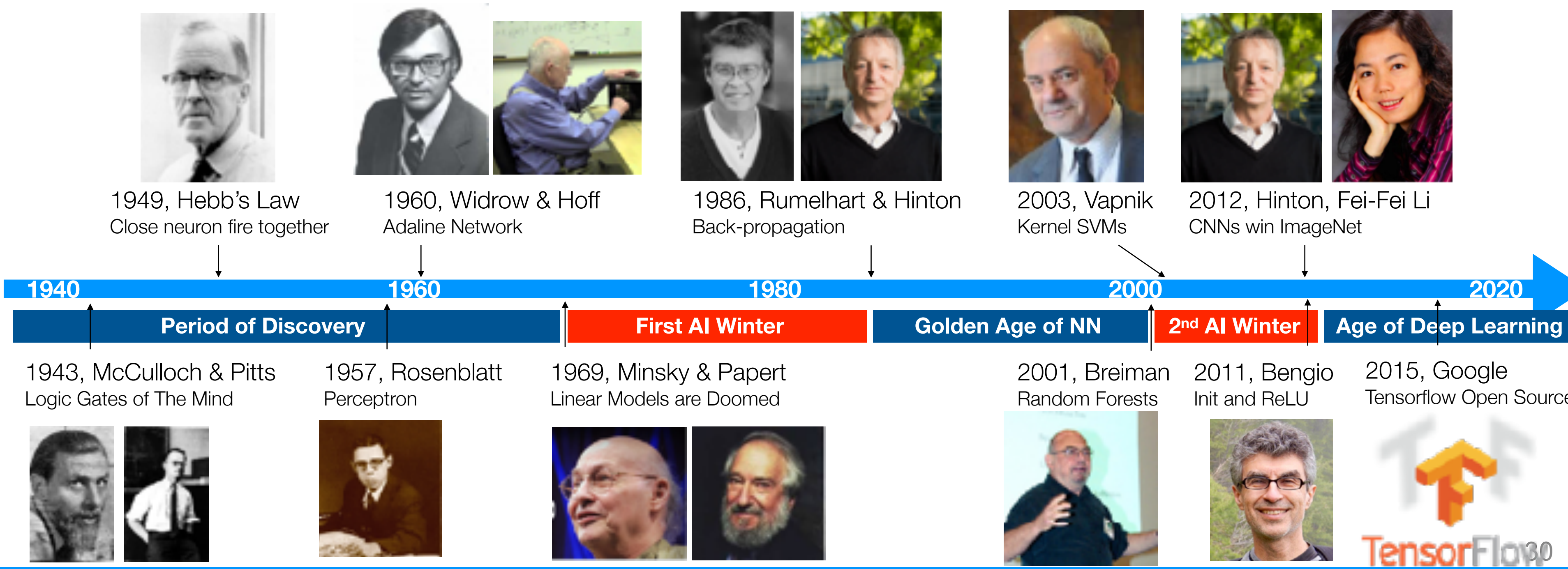
Yoshua Bengio, Geoffrey Hinton and Yann LeCun, the fathers of #DeepLearning, receive the 2018 #ACMTuringAward for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing today. bit.ly/2HVJtdV



Machine learning is the science of credit assignment. The machine learning community itself profits from proper credit assignment to its members. The inventor of an important method should get credit for inventing it. She may not always be the one who popularizes it. Then the popularizer should get credit for popularizing it (but not for inventing it). Relatively young research areas such

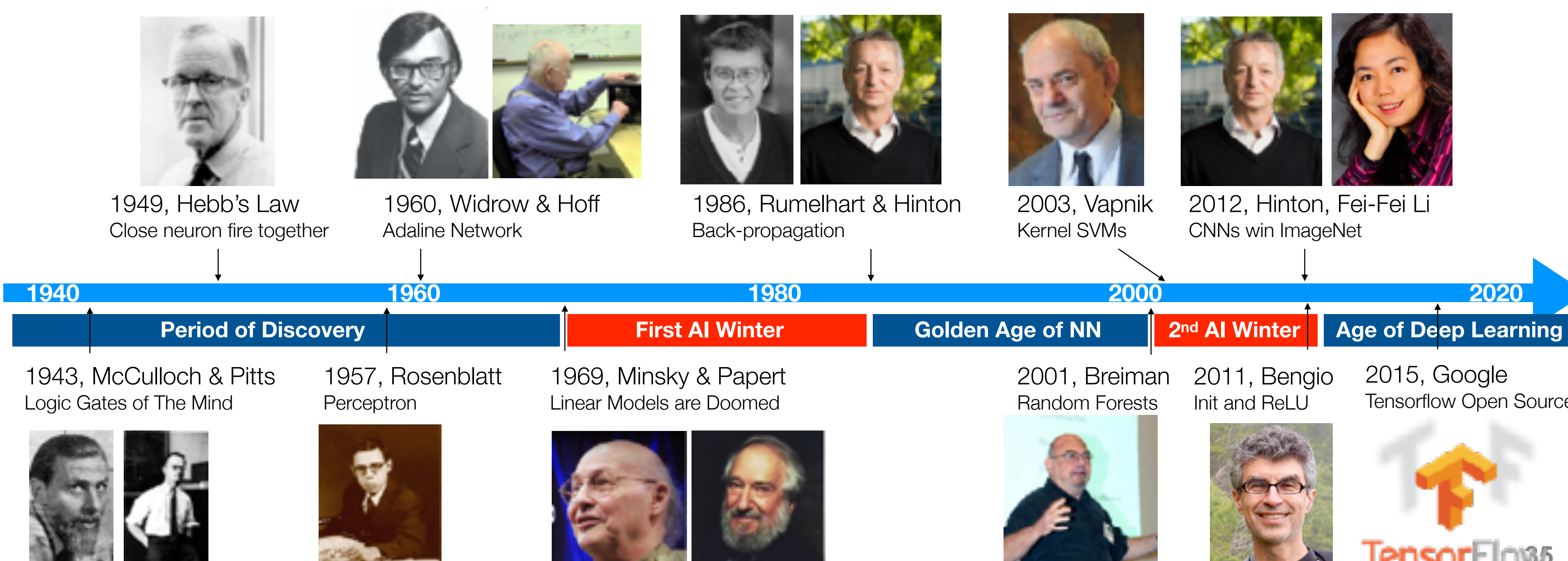
Review of Deep Learning History

- 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)

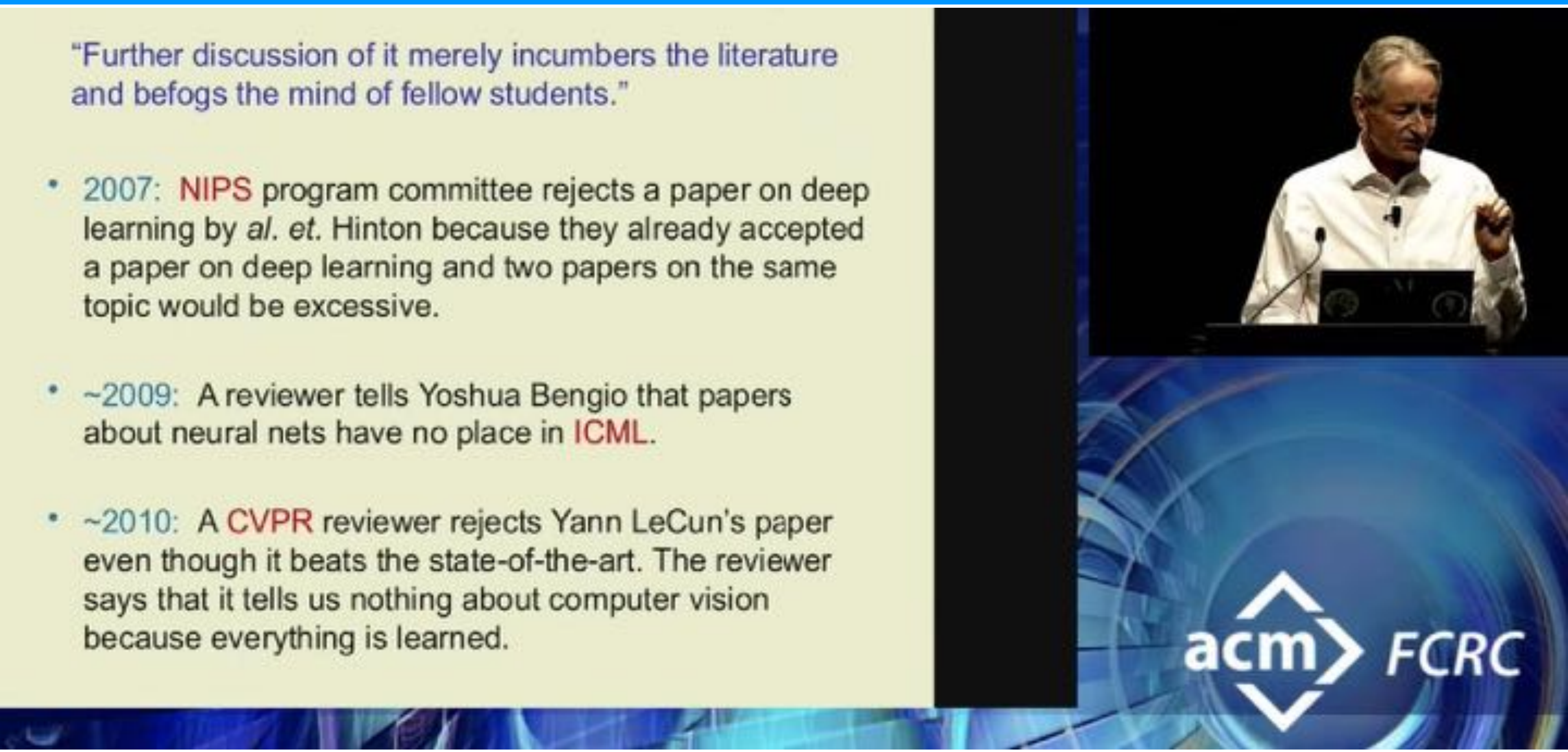


Last Time

- 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)

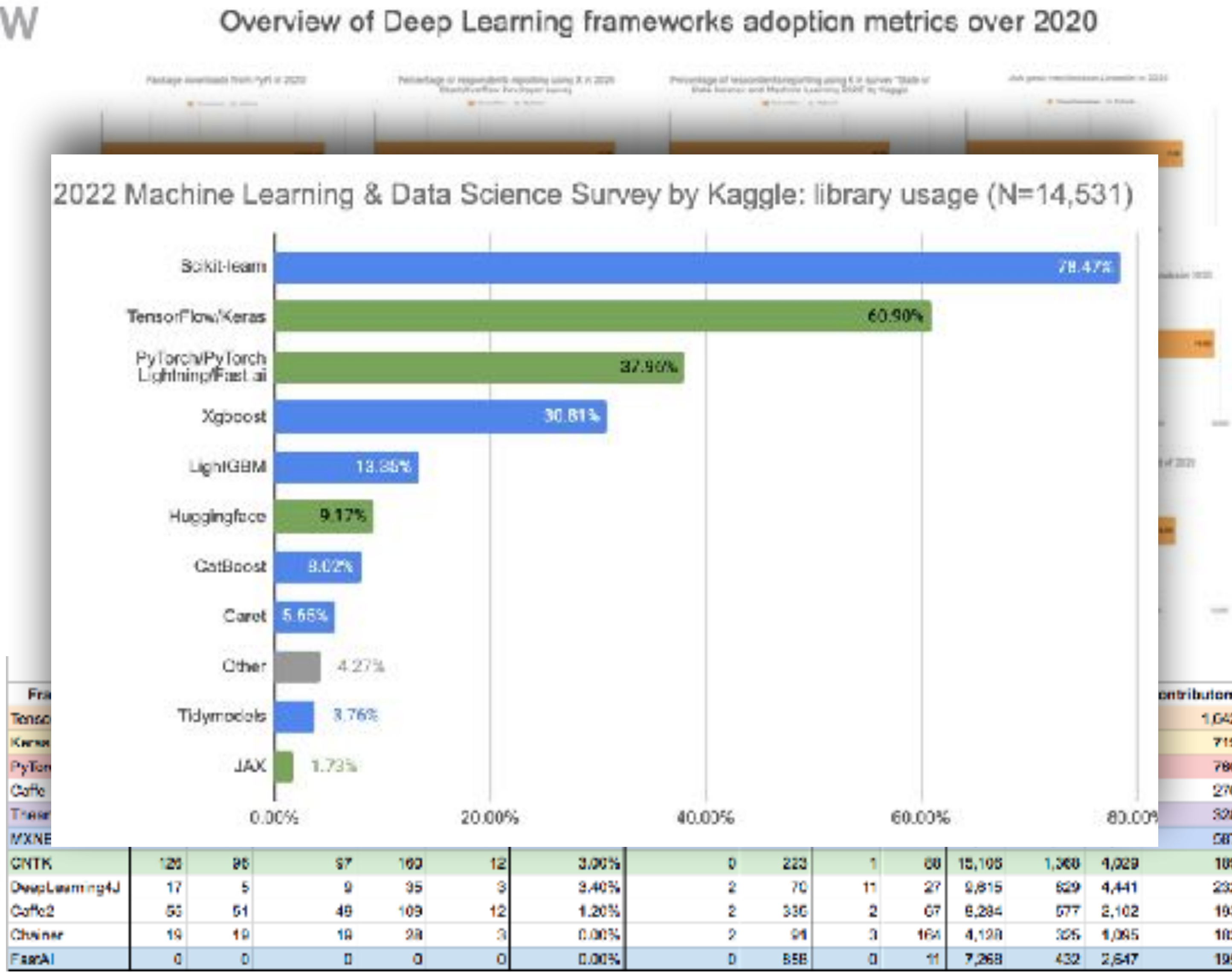


TensorFlow


"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

```
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

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

Computation Graph with Code

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

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

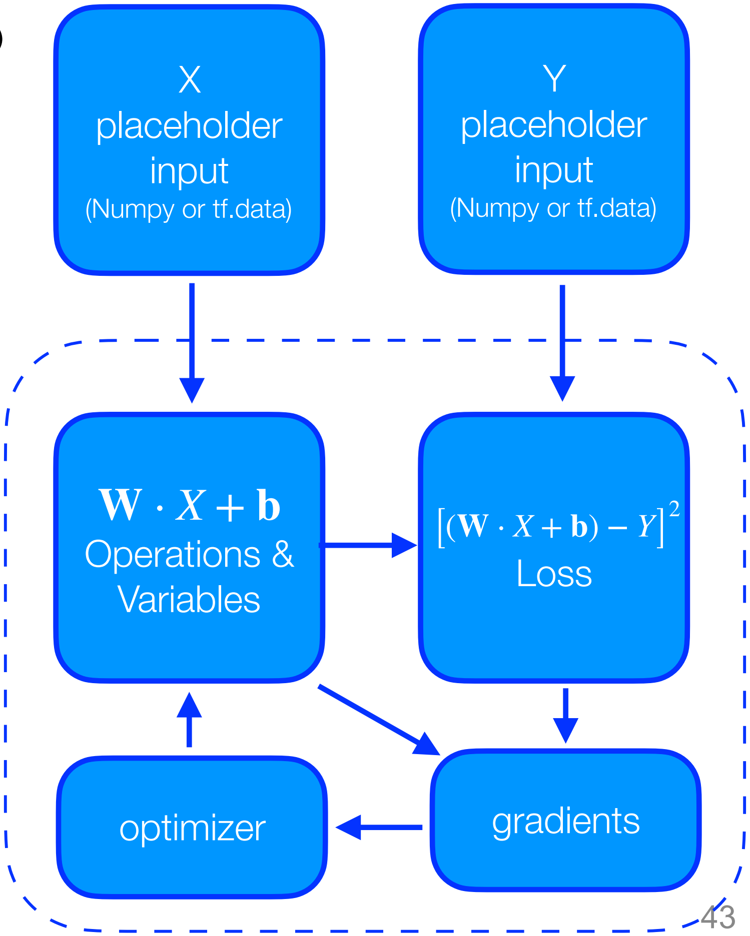
1. **Setup** Variables and computations

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

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

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

```
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})
```

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



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

Tensorflow Mini-batching

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

Define reusable operation, → one optimization update

```
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())

    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})
```

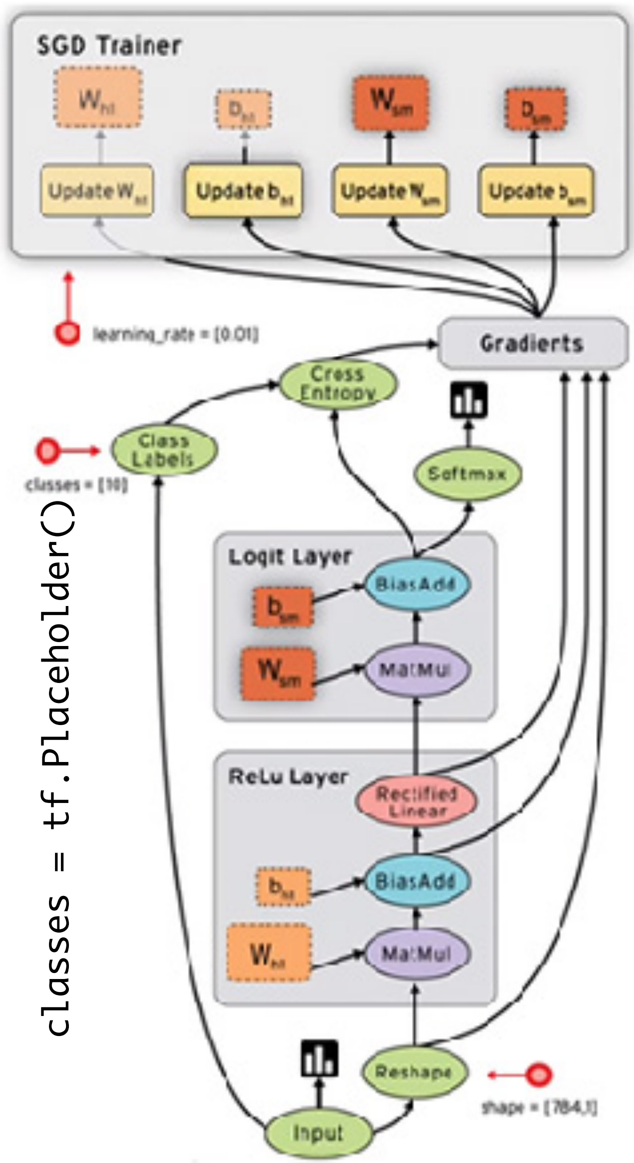
Get random training batch

Run optimization and return loss

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

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

Computation Graph, Two Layer Network



classes = tf.Placeholder()

```
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(…)
```

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

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

```
opt = tf.train.SGDOptimizer(learning_rate=0.01)
opt_operation = opt.minimize(loss)
```

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

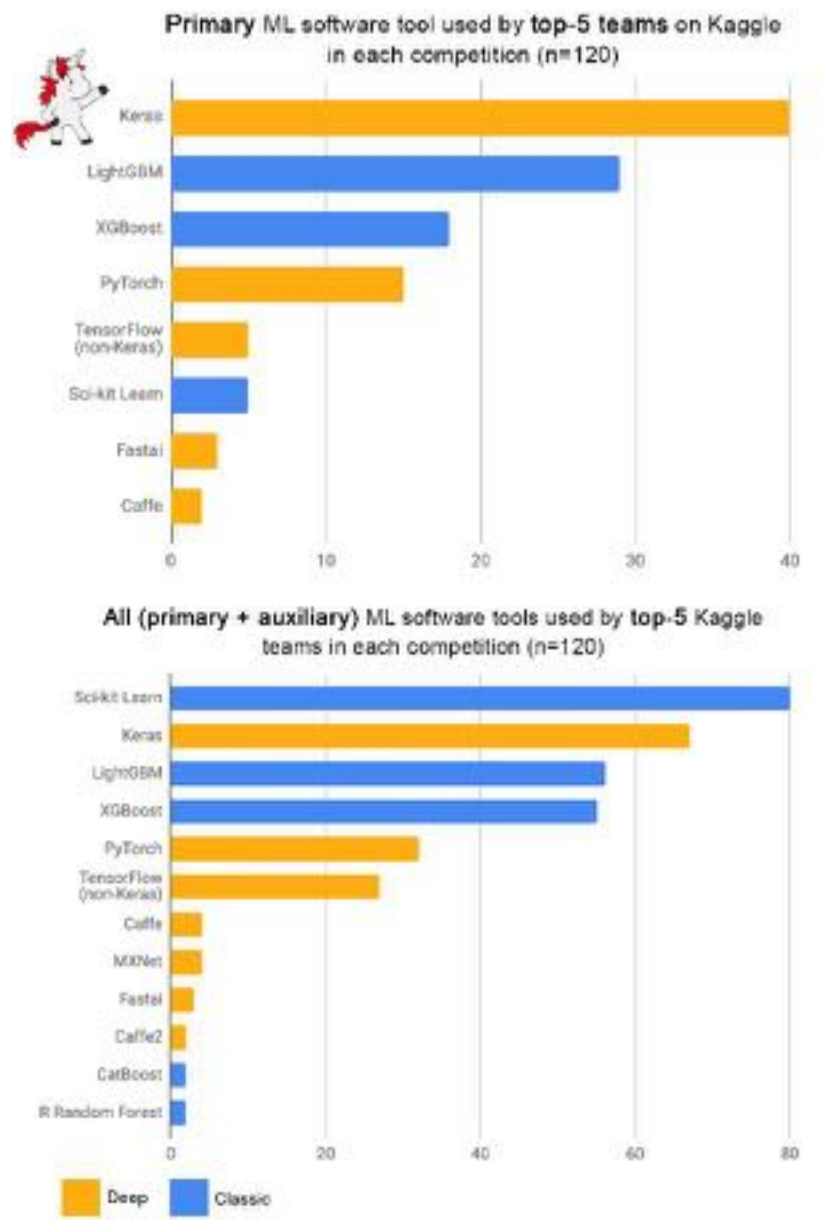
Tensorflow Simplification

- **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

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

Make me slow down if I go too fast!!