Lecture Notes for Machine Learning in Python

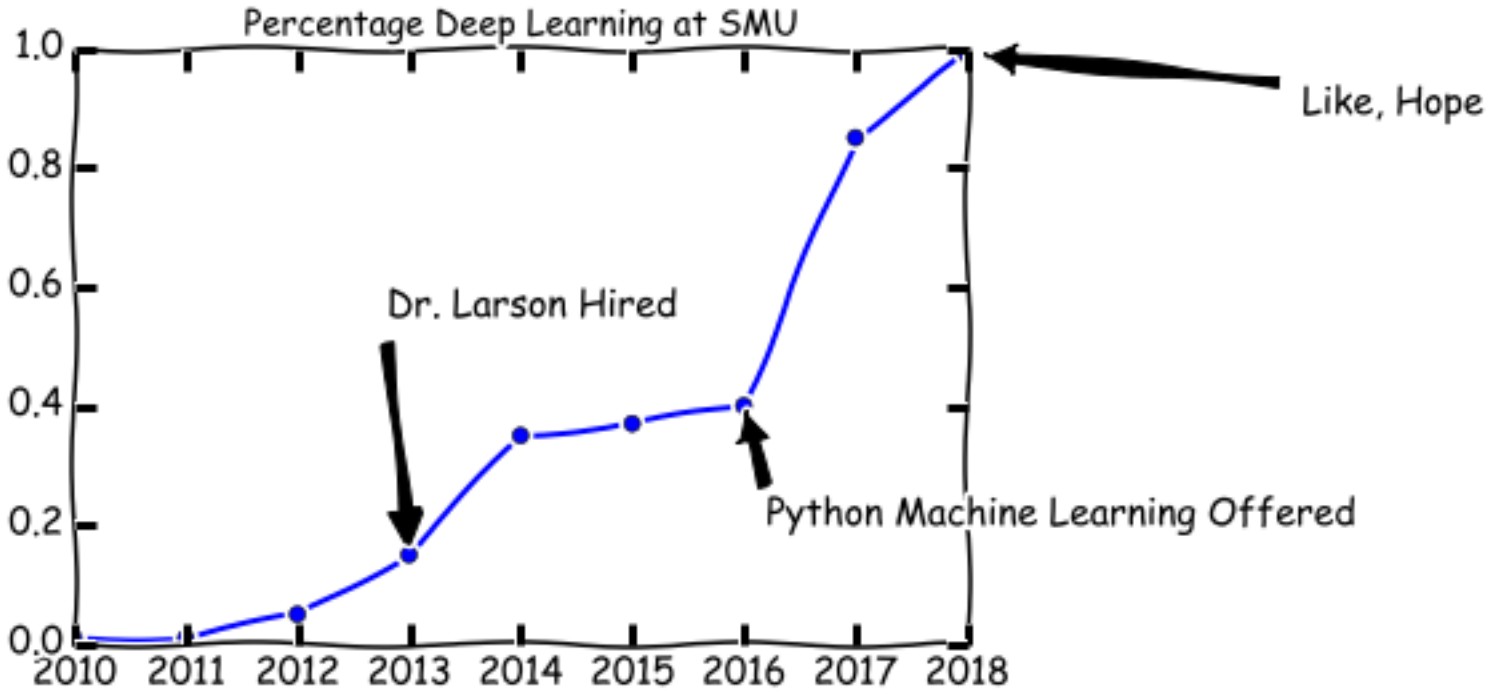
Professor Eric Larson

A “Way too Early” History of Deep Learning

Logistics and Agenda

- Logistics
  - No projects due this week
  - Next Week: Wide and Deep Networks
- Agenda
  - “Deep Learning” History
  - Remaining Lectures
    - TensorFlow from 10,000 feet
    - Wide and Deep Networks
    - Convolution Neural Networks
    - Long- Short-term Memory Networks

Some History of Deep Learning

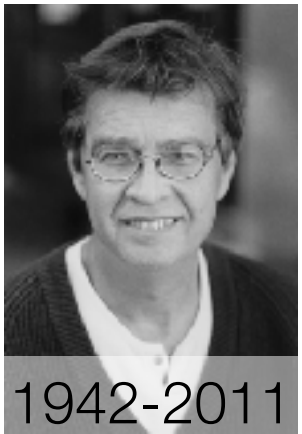



```
# Data to plot
dates = range(2010,2019)
percents = [0.01, 0.01, 0.05, 0.15, 0.35, 0.37, 0.40, 0.85, 0.99]

# Set the style to XKCD
plt.xkcd()

# Plot the percents
plt.plot(dates,percents, marker='o')
```

Neural Networks: Where we left it

- Before 1986: AI Winter
- 1986: *Rumelhart*, *Hinton*, and *Williams* popularize gradient calculation for multi-layer network
  - *technically* introduced by Werbos in 1982
- **difference**: Rumelhart *et al.* validated ideas with a computer
- until this point no one could train a multiple layer network consistently
- algorithm is popularly called **Back-Propagation**
- wins pattern recognition prize in 1993, becomes de-facto machine learning algorithm in the 90's

David Rumelhart



1942-2011

Geoffrey Hinton

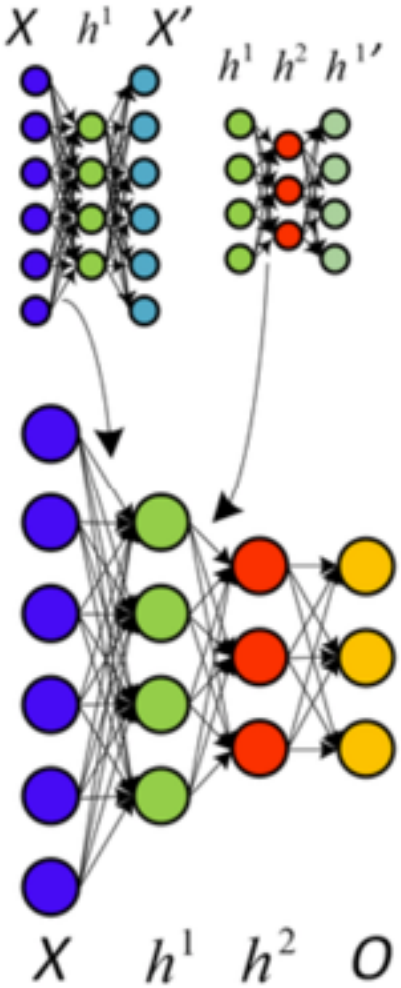
History of Deep Learning: Winter

- Up to this point: back propagation saved AI winter for NN (Hinton and others!)
- 80's, 90's, 2000's: neural networks for image processing start to get deeper
  - but back propagation no longer efficient for training
  - Back propagation gradient **stagnates** research—can't train **deeper** networks
- 2001: SVMs and Random Forests gain traction…
  - The second AI winter begins, research in NN plummets
  - Funding for and accepted papers that incorporate Neural Networks asymptotically approaches zero

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

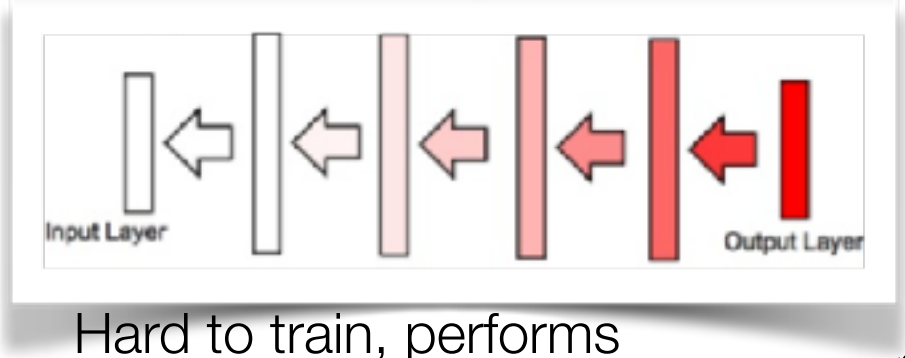
History of Deep Learning: Winter

- Winter is coming:



Easy to train, performs on par with other methods

memorization



Hard to train, performs worse than other methods

generalization

Researcher have difficulty reconciling expressiveness with performance

~chance (untrainable)

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

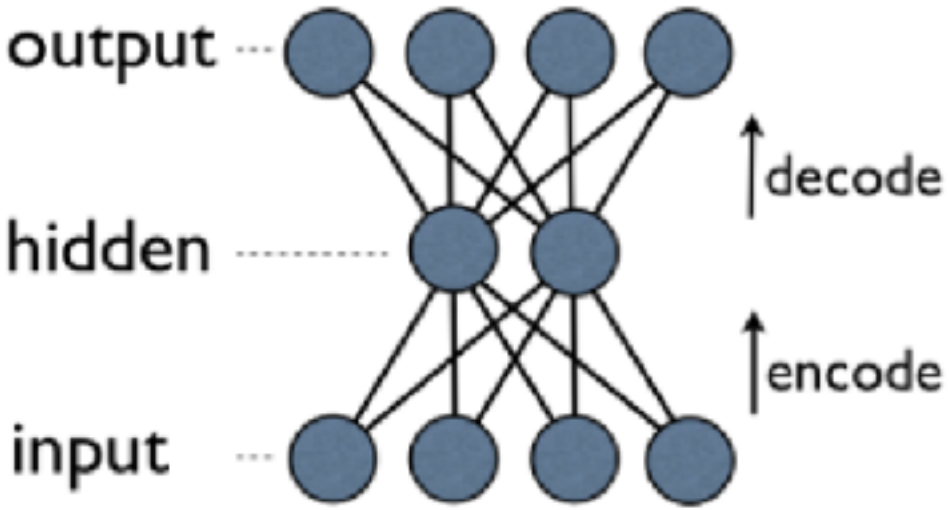
History of Deep Learning

- 2004: Hinton secures funding from CIFAR based on his reputation
  - *eventually*: Canada would be savior for NN
  - Hinton rebrands: Deep Learning
- 2006: Hinton publishes paper on using pre-training and Restricted Boltzmann Machines
- 2007: Another paper: Deep networks are more efficient when pre-trained
  - RBMs not really the important part

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

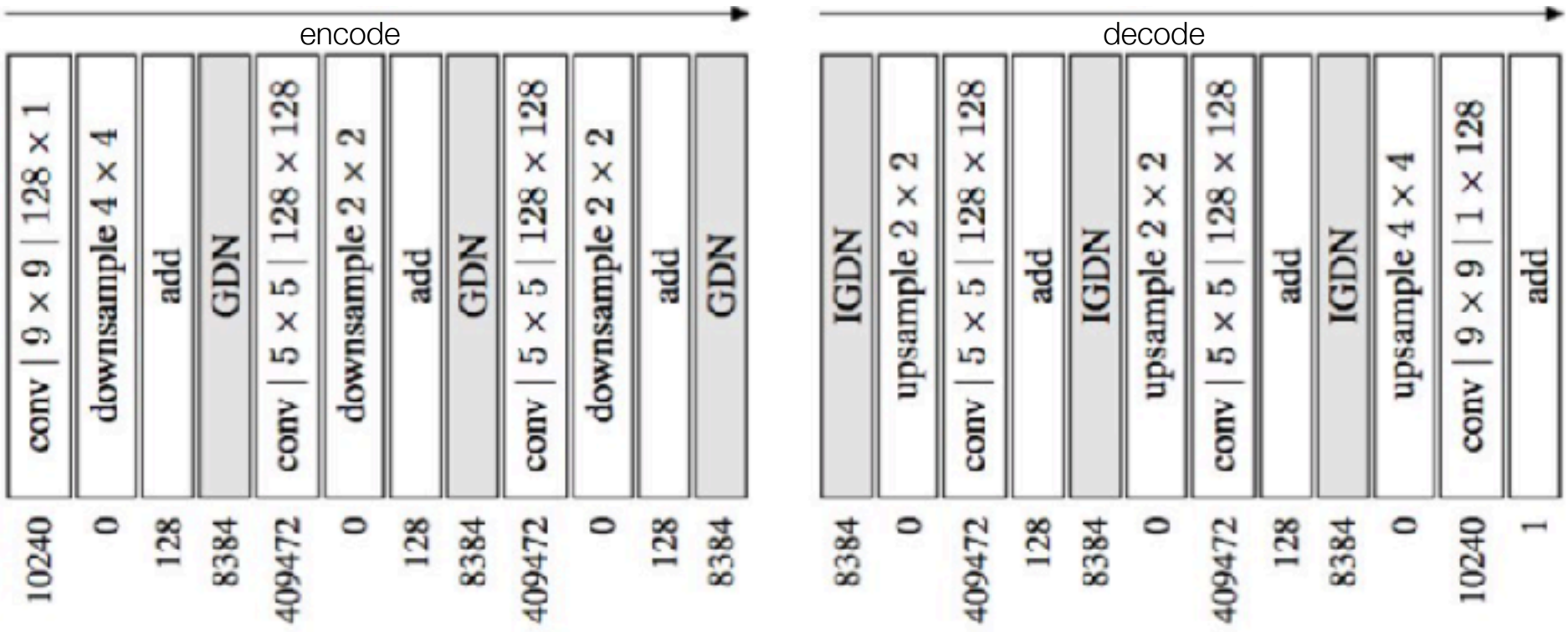
Pre-training: still in the long winter

- auto-encoding

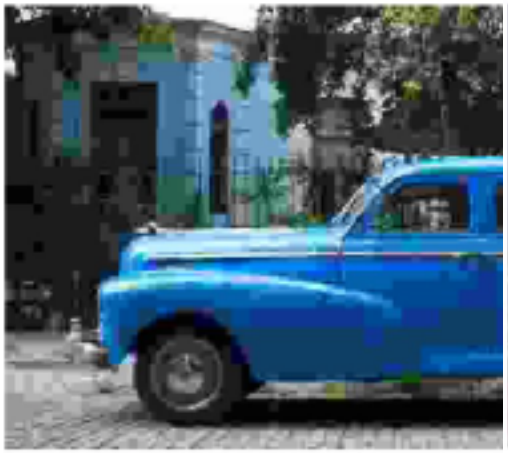


Pre-training: modern example

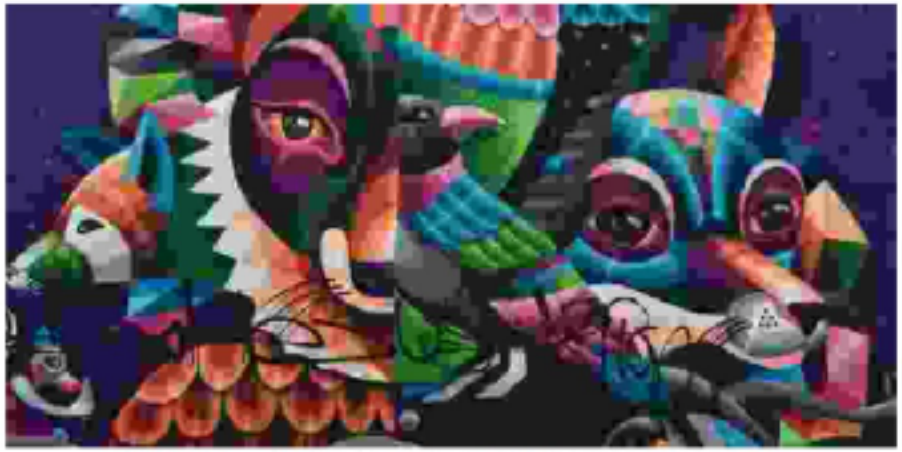
- auto-encoding: example from paper published Nov 5, 2016



https://arxiv.org/abs/1611.01704



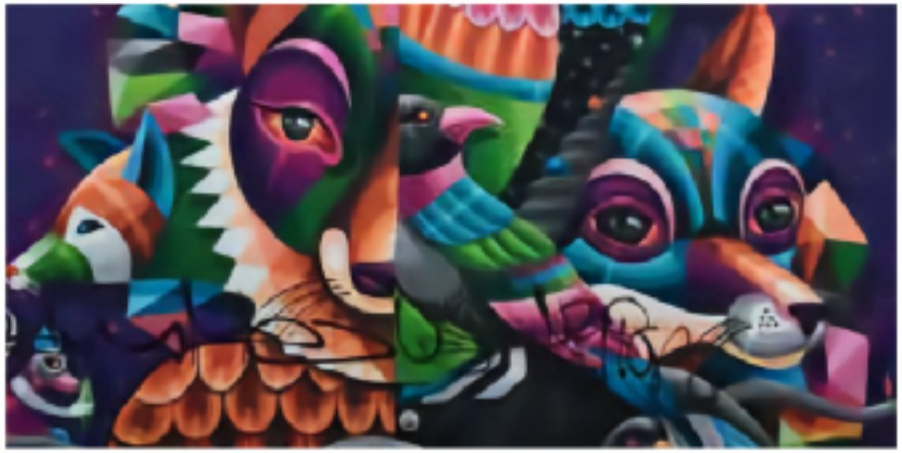
JPEG, 6006 bytes (0.170 bit/px), RMSE: 19.75



JPEG, 5928 bytes (0.168 bit/px), RMSE: 15.44/12.40, PSNR: 24.36 dB/26.26 dB

RMSE: 11.07/10.60, PSNR: 27.25 dB/27.63 dB

Proposed method, 5910 bytes (0.167 bit/px), RMSE



Proposed method, 5685 bytes (0.161 bit/px), RMSE: 10.41/5.98, PSNR: 27.78 dB/32.60 dB

it/px), RMSE: 6.10/5.09, PSNR: 32.43 dB/34.00 dB

JPEG 2000, 5918 bytes (0.167 bit/px), RMSE: 1

JPEG 2000, 5724 bytes (0.162 bit/px), RMSE: 13.75/7.02, PSNR: 25.36 dB/31.20 dB

x), RMSE: 8.56/5.71, PSNR: 29.49 dB/32.99 dB

History of Deep Learning

- 2009:
  - Andrew Ng gets involved
  - Hinton's lab start using GPUs
  - GPUs decrease training time by 70 fold…
- 2010: Hinton's and Ng's students go to internships with Microsoft, Google, IBM, and Facebook

Research at Google

Navdeep Jaitly
Google Brain Team

- Xbox Voice
- Android Speech Recognition
- IBM Watson
- DeepFace
- All of Baidu

George Dahl
Google Brain Team

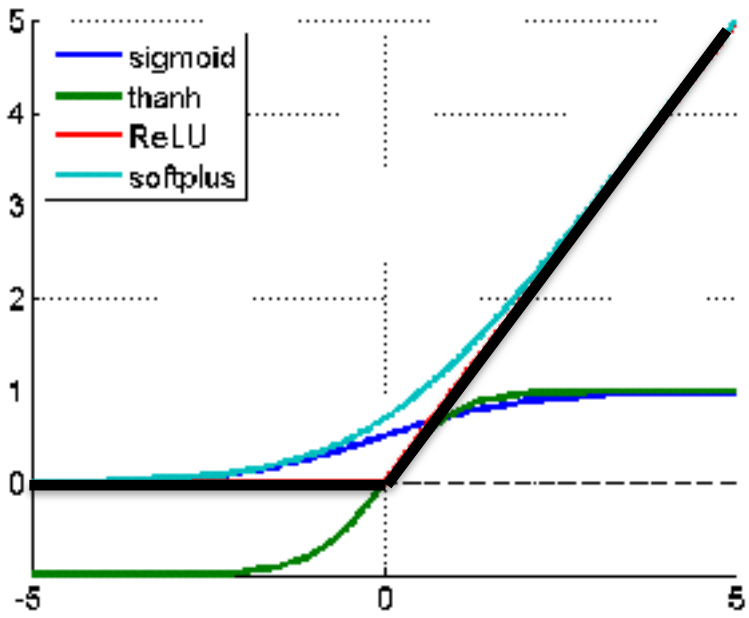
Abdel-rahman Mohamed
Microsoft Research
Redmond, Washington | Computer Software

Current	Microsoft
Previous	University of Toronto, IBM, Microsoft
Education	University of Toronto

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

History of Deep Learning

- 2011: Glort and Y. Bengio investigate more systematic methods for why past deep architectures did not work
  - **discover some interesting, simple fixes**: the type of neurons chosen and the selection of initial weights
  - do not require pre-training to get deep networks properly trained, just sparser representations and less complicated derivatives



$$\text{ReLU: } f(x) = \max(0,x)$$
$$f'(x) = 1 \text{ if } x>0 \text{ else } 0$$

that's a really easy gradient to compute!

…and it makes the weights more sparse

…and helps to solve the vanishing gradient problem

…and its inspired by biological vision community

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

History of Deep Learning

- ReLU not the only way to do it!

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta = \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization

Normalize input layers per mini-batch and add control parameters, γ and β

-help reduce gradient instability
-differentiable normalization==gradient!
-can be applied to each layer input

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

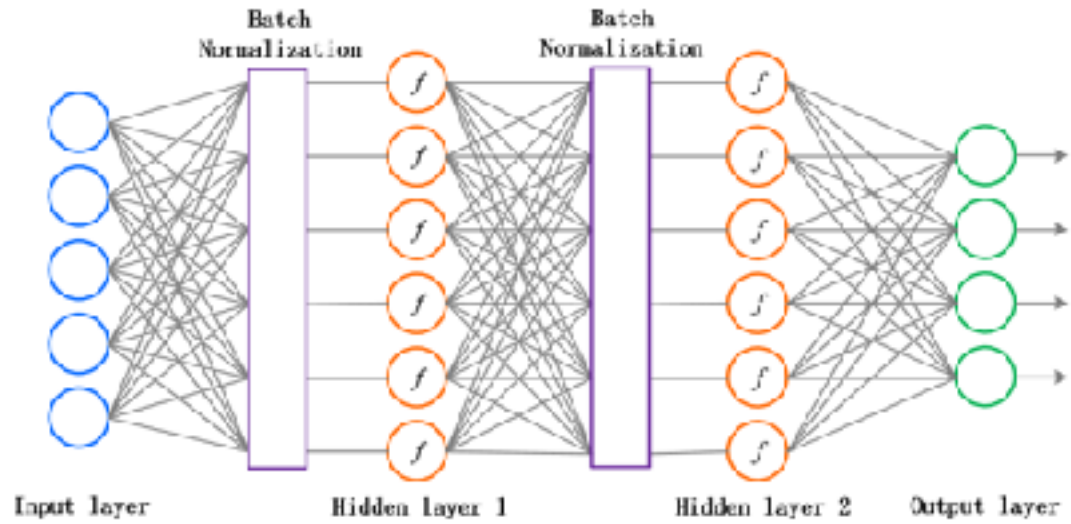
$$\frac{\partial \ell}{\partial \sigma^2_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma^2_{\mathcal{B}} + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}\right) + \frac{\partial \ell}{\partial \sigma^2_{\mathcal{B}}} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} + \frac{\partial \ell}{\partial \sigma^2_{\mathcal{B}}} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$



From Ioffe and Szedegy (2015), Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift: https://arxiv.org/pdf/1502.03167.pdf

History of Deep Learning

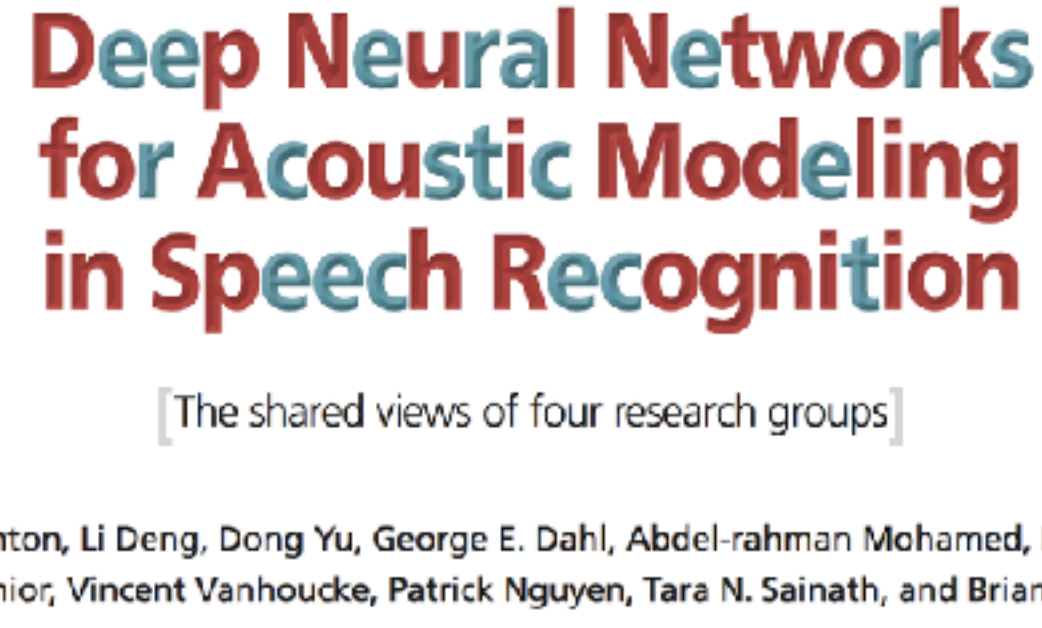
- 2012: ImageNet competition occurs
  - **Second place**: 26.2% error rate
  - **First place**:
    - From Hinton's lab, uses convolutional neural networks with ReLU neurons and dropout
    - 15.2% error rate
  - Computer vision adopts deep learning with convolutional neural networks en mass

Fei Fei Li
Former
Director of Stanford's
AI Lab

I happened to witness this critical juncture in time first hand because the ImageNet challenge was over the last few years organized by Fei-Fei Li's lab (my lab), so I remember when my labmate gasped in disbelief as she noticed the (very strong) ConvNet submission come up in the submission logs. And I remember us pacing around the room trying to digest what had just happened. In the next few months ConvNets went from obscure models that were shrouded in skepticism to rockstars of Computer Vision, present as a core building block in almost every new Computer Vision paper.

History of Deep Learning

- 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

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

History of Deep Learning

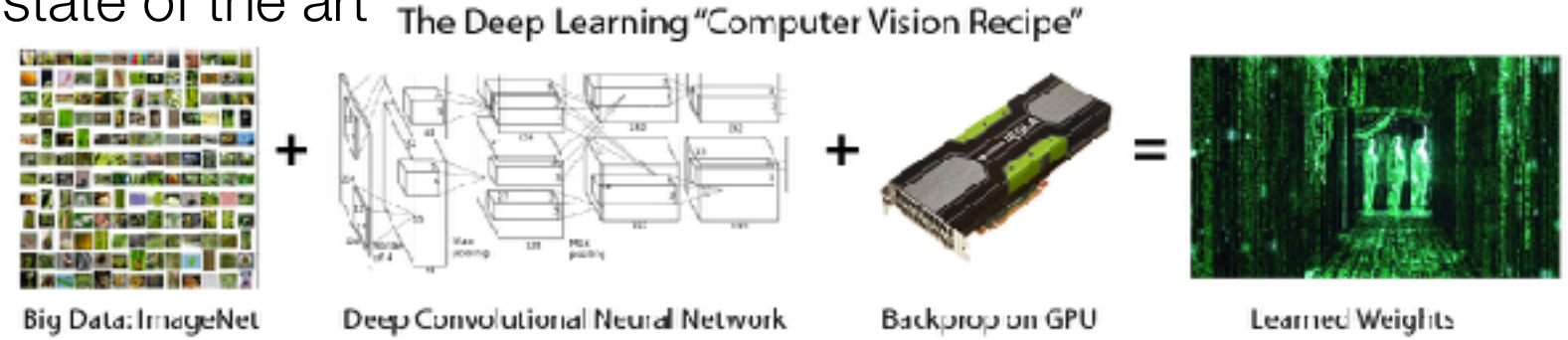
- 2013: Ng and Google (founded BrainTeam)
  - run unsupervised feature creation on YouTube videos (becomes computer vision benchmark)

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.

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

History of Deep Learning

- Hinton summarized what we learned in deep learning from the 2006 to present. Where we went wrong before present day:
  - labeled dataset were 1000s of times too small
  - computers were millions of times too slow
  - weights were initialized in stupid ways
  - we used the wrong non-linearities
- Or in my terms:
  - use a GPU, dropout, ReLU or BN where it makes sense (like in early feedforward layers)
- Modern day deep learning uses simple, tried methods to achieve state of the art



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 University

Advises IBM

Heads Facebook AI Team

Google

Google Cloud

Coursera
Baidu
Google

Read this paper from 2015, as it sums up advancements nicely

And predicts the future of deep learning

REVIEW

doi:10.1038/nature14539

Deep learning

Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

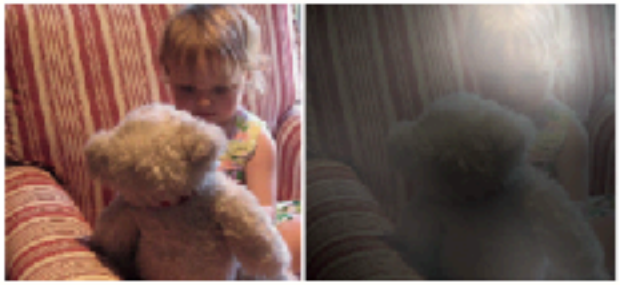
Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other 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 audio, whereas recurrent nets have shone light on sequential data such as text and speech.

Famous examples:

A woman is throwing a **frisbee** in a park.

A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background

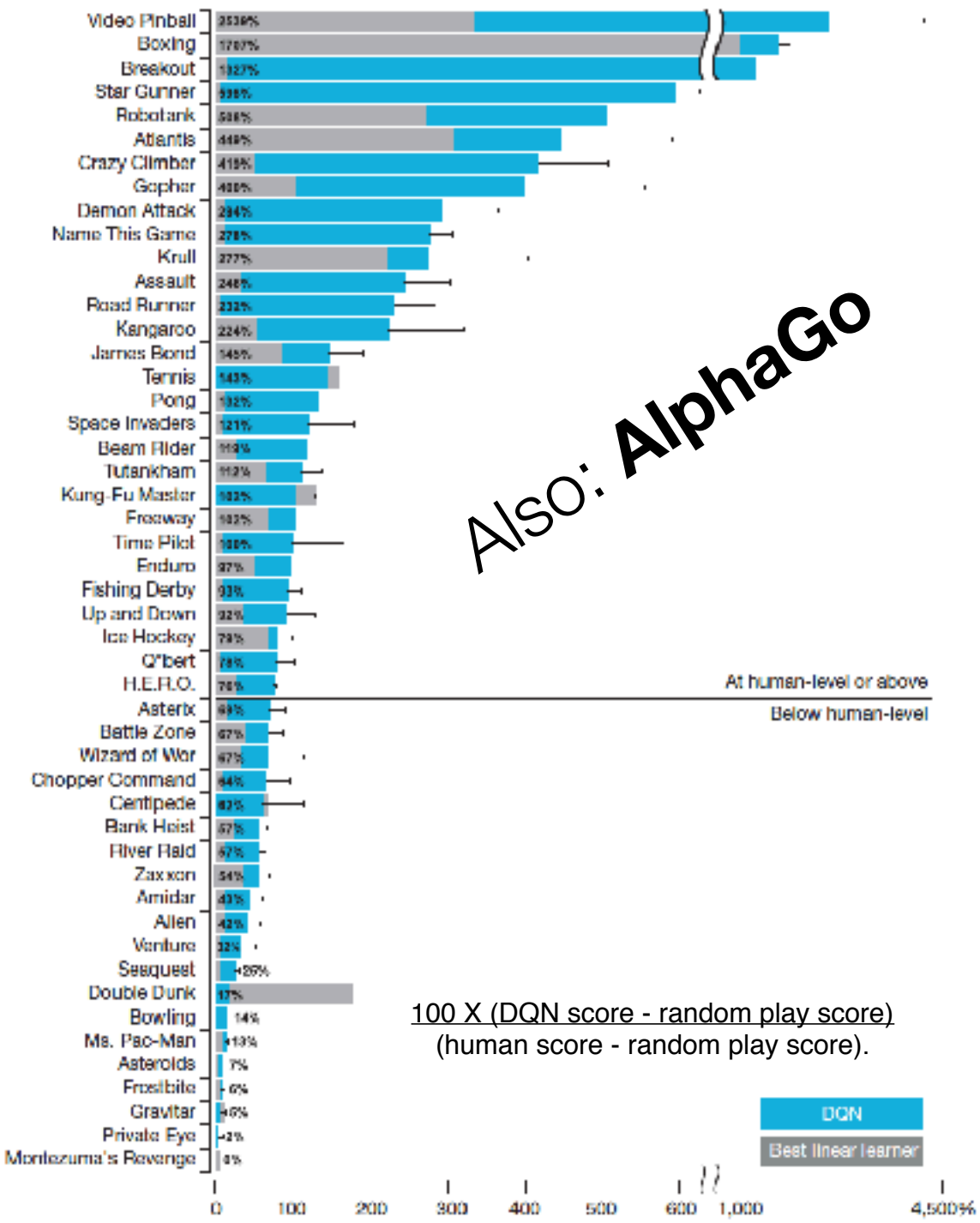


A little **girl** sitting on a bed with a teddy bear.

A group of **people** sitting on a boat in the water.

A giraffe standing in a forest with **trees** in the background.

Famous examp



Also: **AlphaGo**

100 X (DQN score - random play score)
(human score - random play score).

End of Session

- Next Time:
  - Introduction to TensorFlow
  - Wide and Deep Networks

End of Session

- if time:
  - more on auto encoding and transfer learning!
  - to the white board…

Lecture Notes for Machine Learning in Python

Professor Eric Larson

TensorFlow via Wide and Deep Networks

Lecture Agenda

- Introduction to TensorFlow
  - Tensors, Namespaces, Numerical methods
  - Using simplified API Manager!
- Wide and Deep Networks

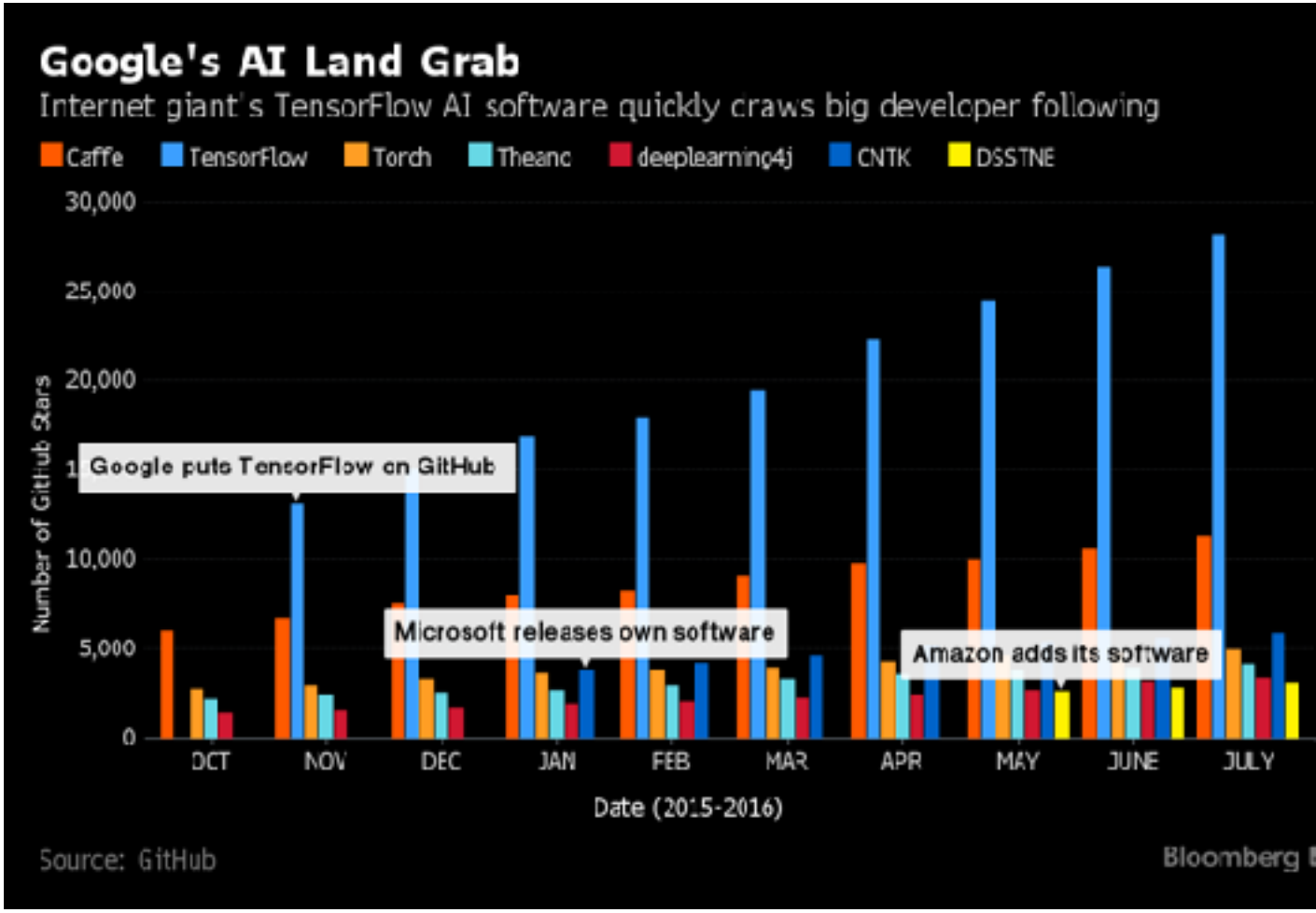
TensorFlow

Matthew Rocklin @mrocklin · Apr 5

Hello world. Please stop calling multi-dimensional arrays "tensors". This angers mathematicians and physicists to no end.

Options for Deep Learning Toolkits

- Caffe
- TensorFlow
- Torch
- CuDNN
- MxNet
- Theano
- CNTK
- DSSTNE



Programmatic creation

- Most toolkits use python to build a computation graph of operations
  - Build up computations
  - Execute computations
- Theano/CNTK are completely valid alternatives
  - its really up to you what you want to use
  - Theano originated at Berkeley and can be wrapped with Keras or Lasagne
    - http://deeplearning.net/software/theano/
  - CNTK is a Microsoft product with python wrappers and has some impressive speed graphics compared to all other languages (as of one year ago)
    - https://github.com/Microsoft/CNTK

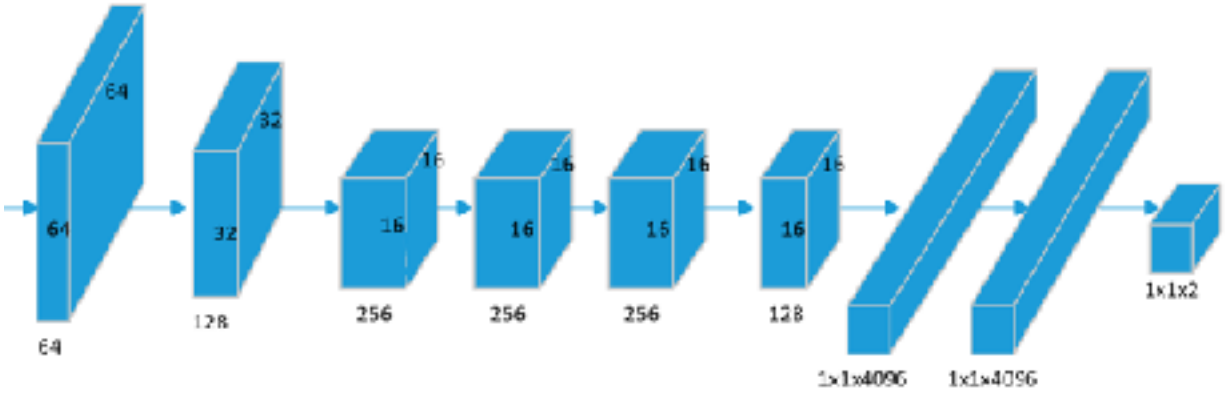
Tensorflow

- Open sourced library from Google
- Second generation release from Google Brain
  - supported for Linux and Unix (and recently windows!)
  - Also works on Android/iOS
- Released November 9th, 2015
- **Supports:**
  - tensor creation
  - functions on tensors
  - automatic derivative computation

Tensors

- Tensors are just multidimensional arrays
  - like in Numpy
  - typically, for NN,
    - 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)

```
a = tf.constant(5.0)

b = tf.constant(6.0)
```



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

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

- Each function creates layers easily, *knows its gradient*
- But… lets start simple…

Tensor function evaluation

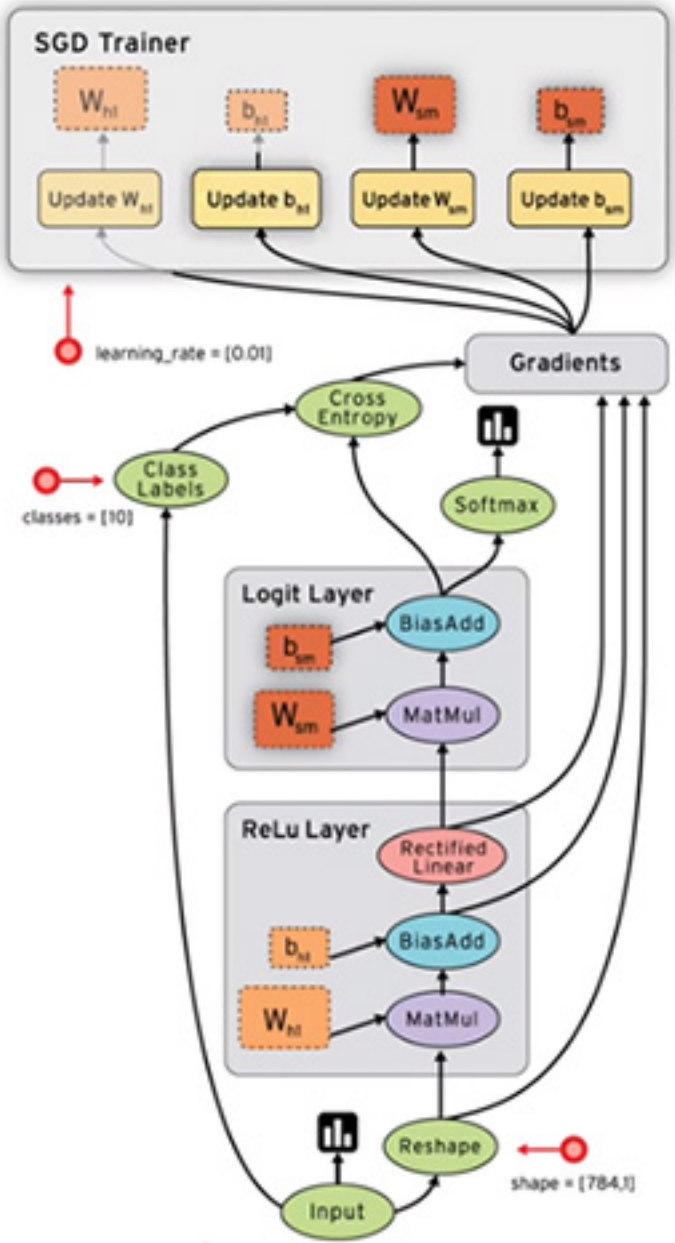
- Easy to define operations on tensors
- Nothing evaluated until you define a session and tell it to evaluate it
- Session defines configuration of execution
  - like GPU versus CPU

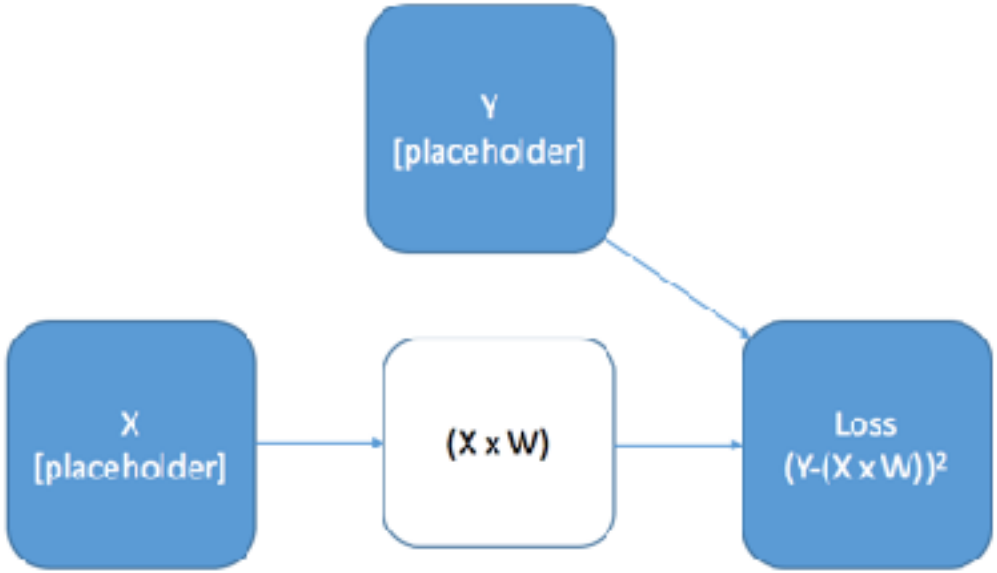
```
a = tf.constant(5.0)

b = tf.constant(6.0)

c = a * b

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

output = 30

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

Computation Graph





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

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

Tensorflow for Neural Networks

- Seems like a lot of new syntax
- Why can't we just use numpy?
  - automatic differentiation, i.e., "*your life, easier*"

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

```
# Define variables to be learned
with tf.variable_scope("linear-regression"):
    W = tf.get_variable("weights", (1, 1),
                        initializer=tf.random_normal_initializer())
    b = tf.get_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)

    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_data, y: y_data})
```


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

Tensor 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})
```

- Each node in a TensorFlow graph knows its gradient
- Which means back-propagation is easy to carry out computationally

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

Tensor-flow Simplification

- It seems like many common NN optimizations can be preprogrammed
- **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) **Yes**, and Dr. Larson is going to make us write it ourselves

Tensor General Training Overview

- High level wrappers help standardize
  - wrappers for TensorFlow typically follow this paradigm:
  - ✓ deep MLPs can be created/fit in one or two lines of code
  - customizing architectures requires about 10x code, but
  - ✓ also has high level wrappers
    - CNNs, RNNs have good support here
    - implemented through “model function”
  - ✓ dealing with different datatypes adds more complexity through “input functions”
  - X even more advanced architectures require using pure TensorFlow
  - X generative, adversarial networks, transfer learning, etc. usually require pure TensorFlow

Tensor General Training Overview

- Model Functions (tf.contrib)
  - these functions create the architecture for the network
  - called for inference and training
  - can use pure TensorFlow code or wrappers
  - need to return:
    - training operations (and make graph, if needed)
    - loss function (where to read from graph)
    - predictions (how you interpret outputs)
  - you have lots of control and lots of responsibilities as a programmer

Demo

TensorFlow tf.contrib (and Keras)

Reinventing the MLP Wheel

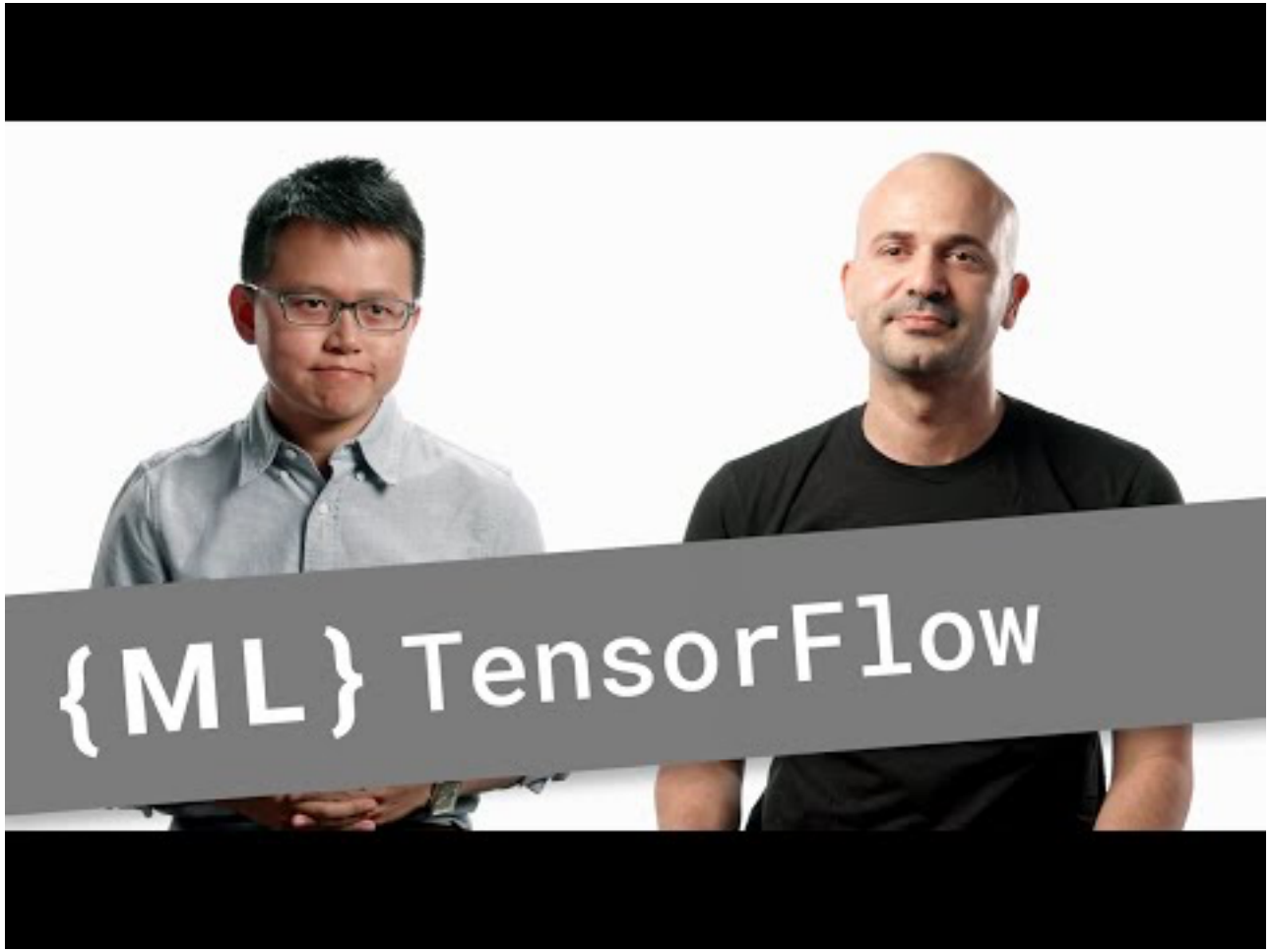
Other tutorials:

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

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

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

Wide and Deep Networks



Wide and Deep

Wide & Deep Learning for Recommender Systems

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, Hemal Shah

Google Inc.*

ABSTRACT

Generalized linear models with nonlinear feature transfor-

have never or rarely occurred in the past. Recommenda-tions based on memorization are usually more topical and

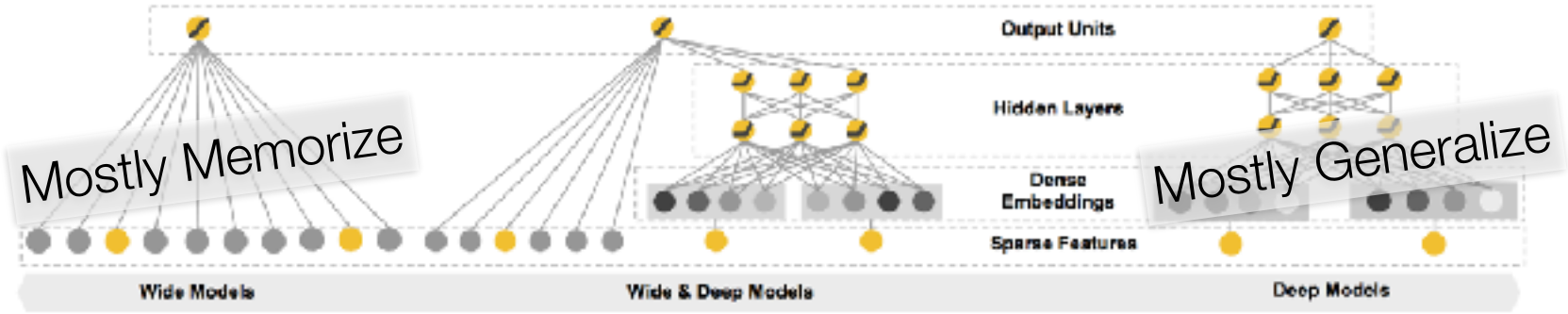
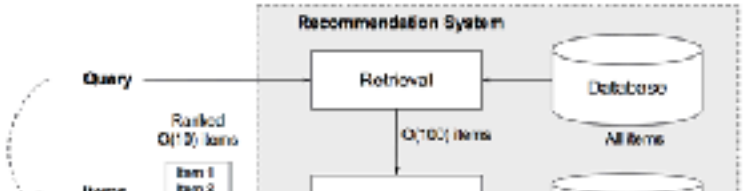


Figure 1: The spectrum of Wide & Deep models.

linear model with feature transformations for generic recommender systems with sparse inputs.

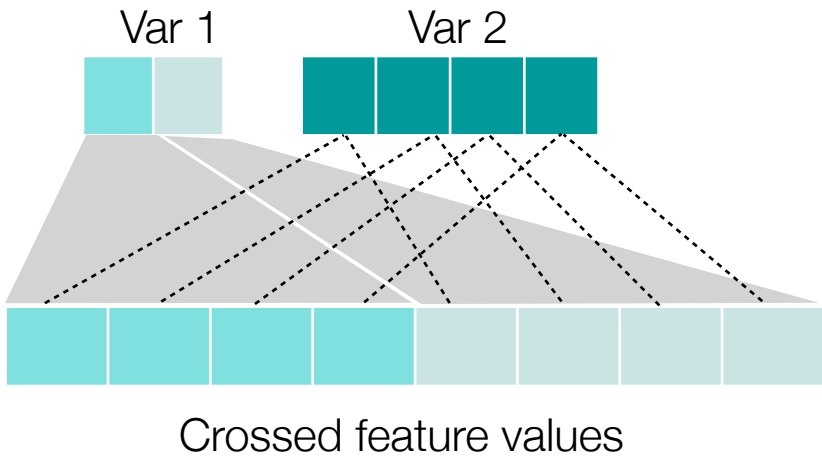
- The implementation and evaluation of the Wide & Deep recommender system productionized on Google



Wide networks (Memorize?)

- Wide refers to the expansion of your features set
- Crossed feature columns of categorical features

- Movie Rating
  - G
  - PG
  - PG-13
  - R
  - Else
- Movie Genre
  - Action
  - Drama
  - Comedy
  - Horror
  - Else

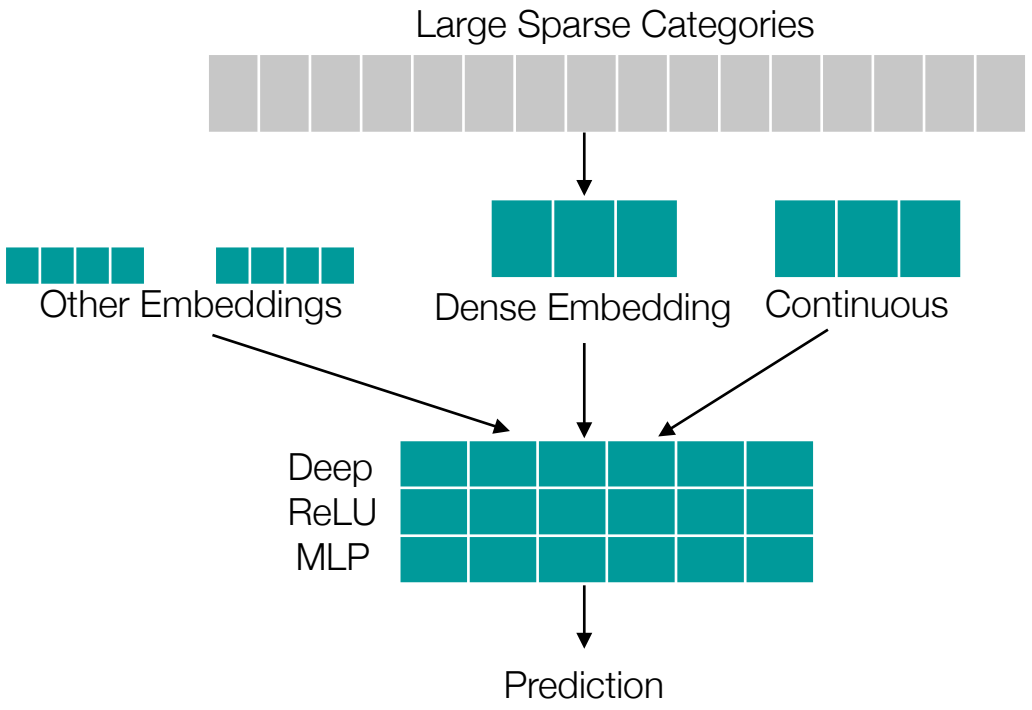


- Crossed feature “Rating-Genre”
  - G-Action, G-Drama, G-Comedy, G-Horror, G-else
  - PG-Action, PG-Drama, PG-Comedy, PG-Horror, G-else
  - and so on … one hot encoded

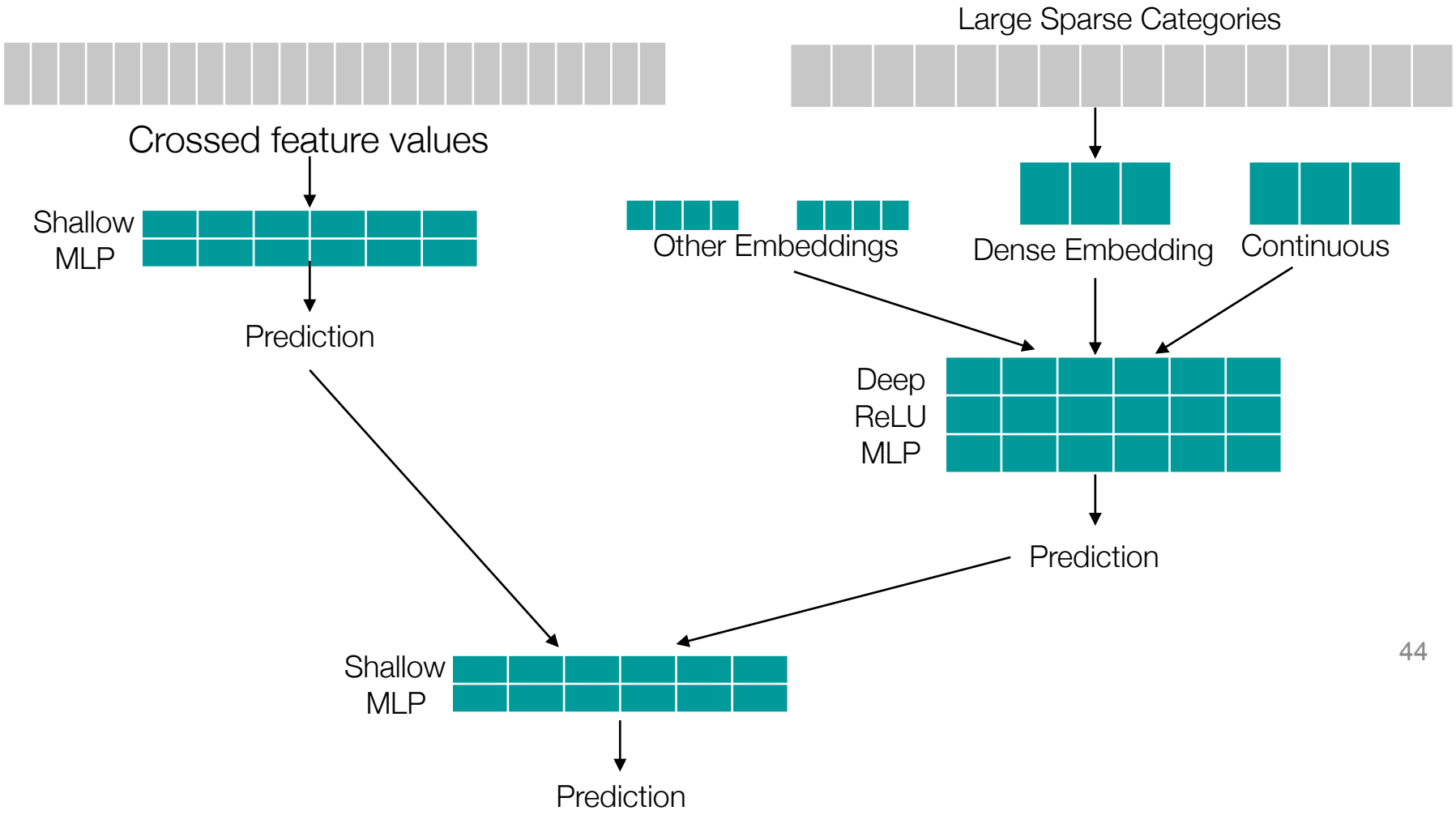
Sparse Embedding and Deep MLP (Generalize?)

- Deep refers to increasingly smaller hidden layers
- Embed into sparse representations via ReLU

- Movie Actors
  - Armand Assante
  - Danny Trejo
  - Kevin Bacon
  - Meryl Streep
  - Audrey Hepburn
  - …



Combining Memorization and Generalization



Wide and Deep

Reproducing Google:
Toy Census Data Example

Other tutorials:

https://www.tensorflow.org/tutorials/wide_and_deep

End of Session

- Next Time:
  - Convolutional Neural Networks