

---

# Lecture Notes for Machine Learning in Python

---

Professor Eric Larson

Revisiting CV

A “Way too Early” History of Deep Learning

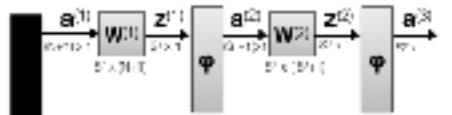
# Logistics and Agenda

---

- Logistics
  - No projects due this week
  - Next time: Wide and Deep Networks (in TF)
- Agenda
  - Cross Validation
  - “Deep Learning” History
  - Remaining Topics
    - TensorFlow from 10,000 feet
    - Wide and Deep Networks
    - Convolution Neural Networks
    - Recurrent Neural Networks

# Last time:

## Back propagation summary



$$J(\mathbf{W}) = \sum_k^M (y^{(k)} - a^{(L)})^2$$

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta \frac{\partial J(\mathbf{W})}{\partial w^{(l)}} a_i^{(l)}$$

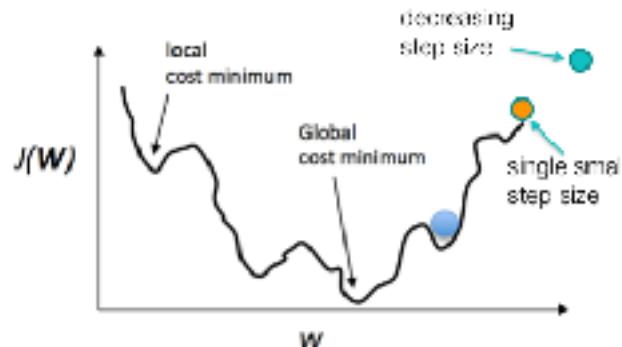
1. Forward propagate to get  $\mathbf{z}, \mathbf{a}$  for all layers
2. Get final layer gradient
3. Update back propagation variables
4. Update each  $\mathbf{W}^{(l)}$ 
  - for each  $y^{(l)}$
  - $\frac{\partial J(\mathbf{W})}{\partial a^{(l)}} = -2(y^{(l)} - a^{(l)}) \cdot a^{(l)} + (1 - a^{(l)})$
  - $\frac{\partial J(\mathbf{W})}{\partial z^{(l)}} = \text{diag}[a^{(l+1)} * (1 - a^{(l+1)})] \cdot \mathbf{W}^{(l+1)} \frac{\partial J(\mathbf{W})}{\partial z^{(l+1)}}$
  - $\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial J(\mathbf{W}^{(l)})}{\partial \mathbf{z}^{(l)}} \cdot a^{(l)}$

## Problems with Advanced Architectures

- Space is no longer convex

- One solution:**

- start with large step size
- \*cool down\* by decreasing step size for higher iterations



28

## Practical Implementation of Architectures

- A new cost function: **Cross entropy**

$$J(\mathbf{W}) = -[y^{(l)} \ln a^{(L)} + (1 - y^{(l)}) \ln(1 - a^{(L)})]$$

speeds up initial training

$$\frac{\partial J(\mathbf{W})}{\partial z^{(l)}} = (\mathbf{a}^{(l+1)} - \mathbf{y}^{(l)})$$

# vectorized backpropagation  
signal1 = (A2-Y\_enc) # <- this is only line  
signal2 = (M2.T \* signal1)\*A2\*(1-A2)

$$\frac{\partial J(\mathbf{W})}{\partial z^{(2)}} = (\mathbf{a}^{(3)} - \mathbf{y}^{(2)})$$

grad1 = signal2[1:,:]\*A1  
grad2 = signal3 \* M2.T

new update

# vectorized backpropagation  
signal1 = -2\*(Y\_enc-A2)\*A2\*(1-A2)  
signal2 = (M2.T \* signal1)\*A2\*(1-A2)

grad1 = signal2[1:,:]\*A1  
grad2 = signal3 \* M2.T

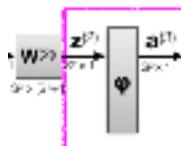
$$\frac{\partial J(\mathbf{W})}{\partial z^{(2)}} = -2(y^{(2)} - a^{(2)}) \cdot a^{(2)} + (1 - a^{(2)})$$

old update

bp-5  
62

## Practical Implementation of Architectures

- A new nonlinearity: **rectified linear units**



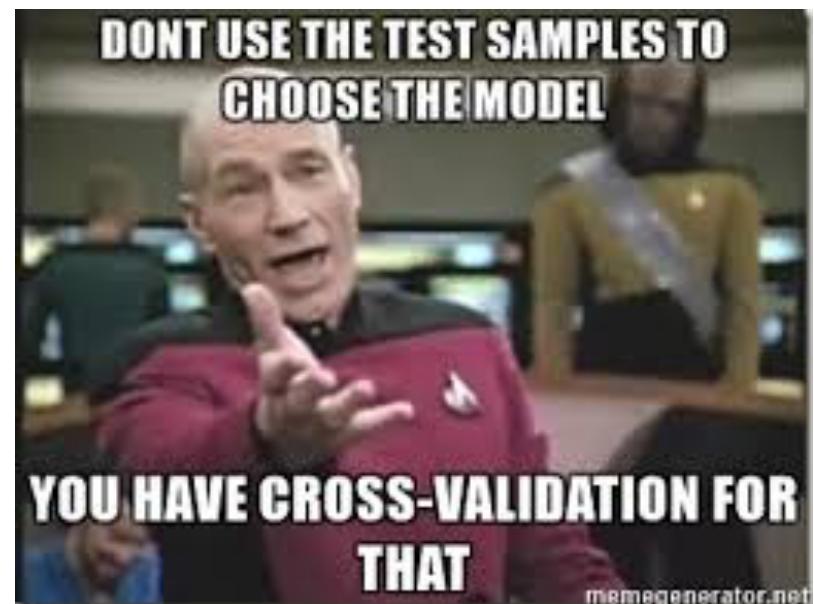
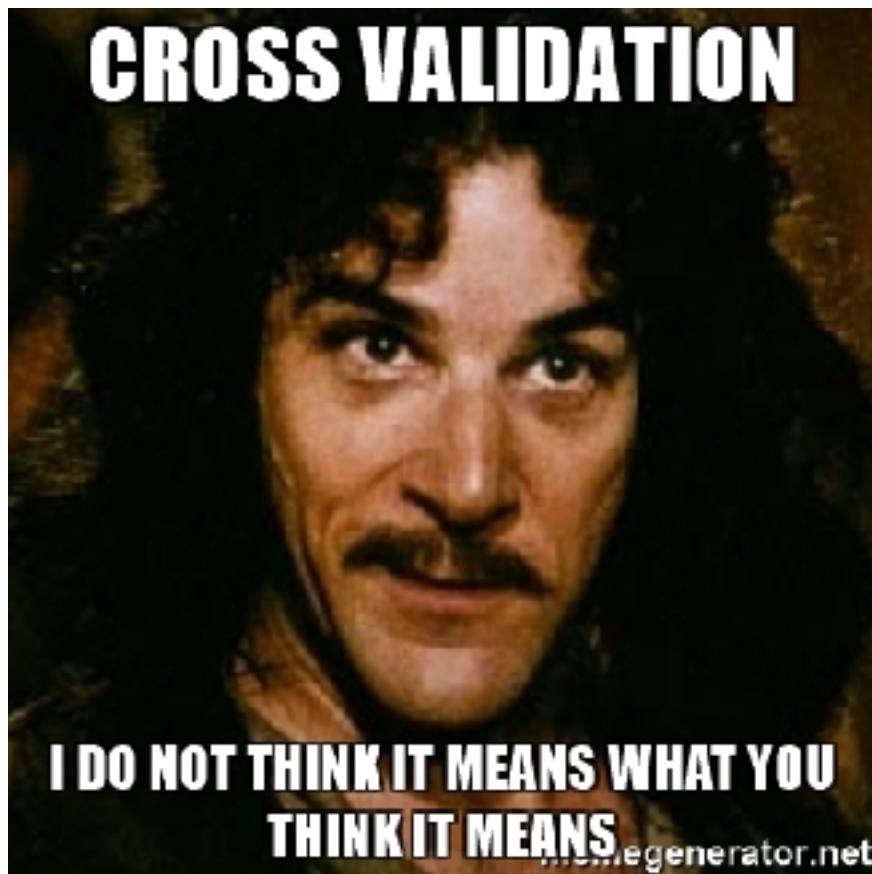
$$\phi(z^{(l)}) = \begin{cases} z^{(l)}, & \text{if } z^{(l)} > 0 \\ 0, & \text{else} \end{cases}$$

it has the advantage of **large gradients** and **extremely simple** derivative

$$\frac{\partial \phi(z^{(l)})}{\partial z^{(l)}} = \begin{cases} 1, & \text{if } z^{(l)} > 0 \\ 0, & \text{else} \end{cases}$$

3

# Revisiting Cross Validation



# Grid Searching

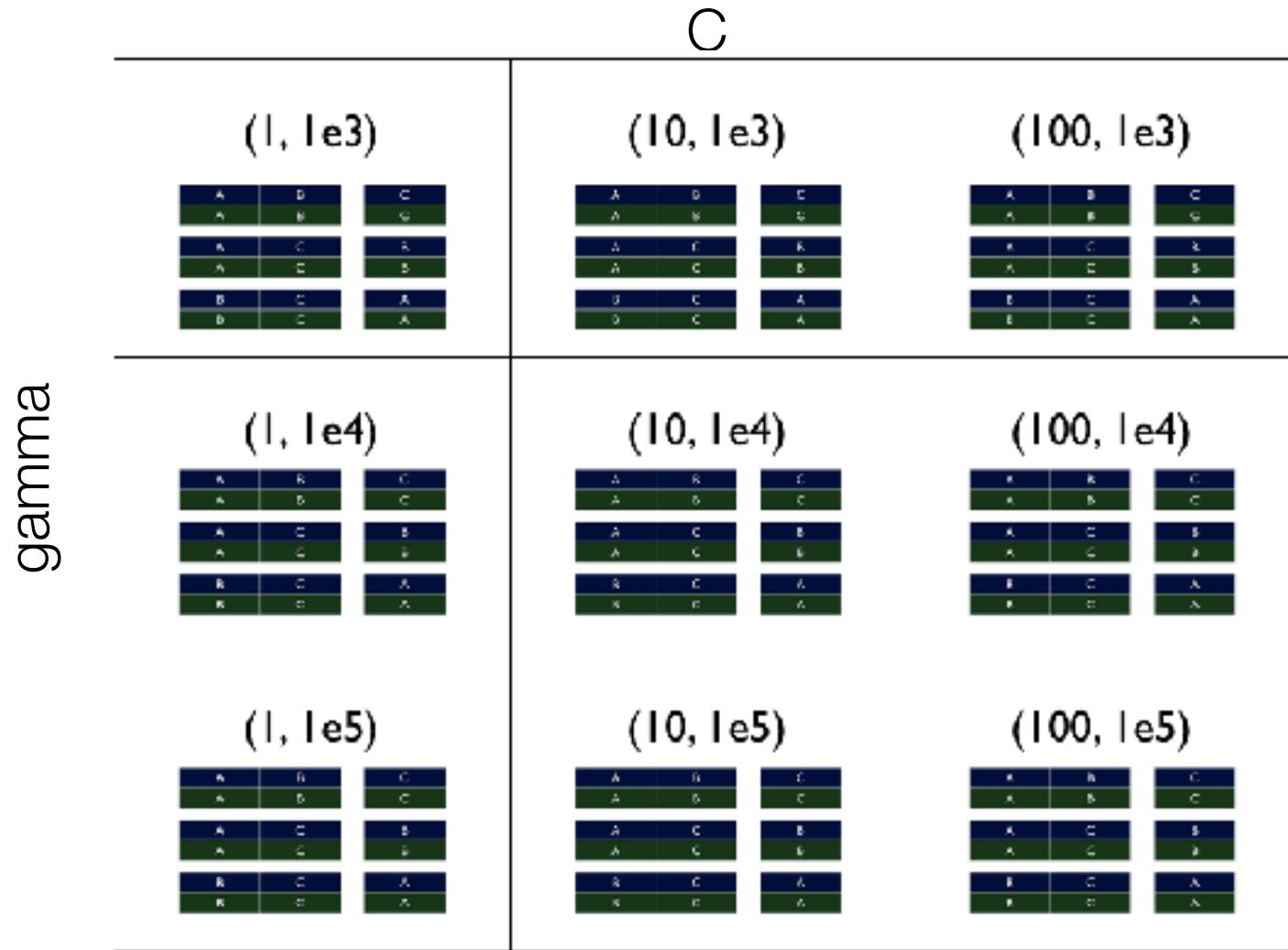
---

- Trying to find the best parameters
  - SVM:  $C=[1, 10, 100]$   $\gamma=[1e3, 1e4, 1e5]$

		C		
		(1, 1e3)	(10, 1e3)	(100, 1e3)
		(1, 1e4)	(10, 1e4)	(100, 1e4)
		(1, 1e5)	(10, 1e5)	(100, 1e5)

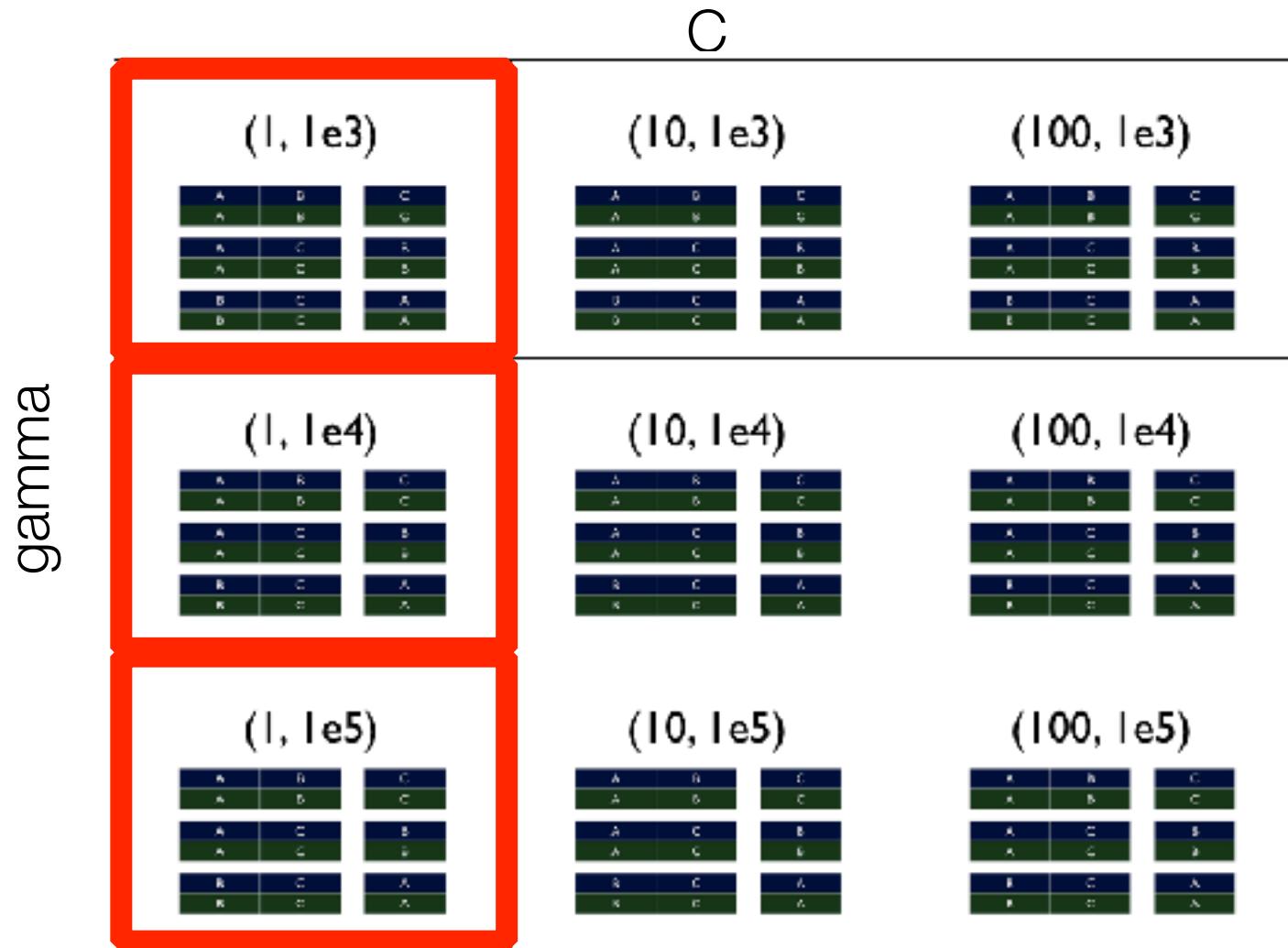
# Grid Searching

- For each value, want to run cross validation...



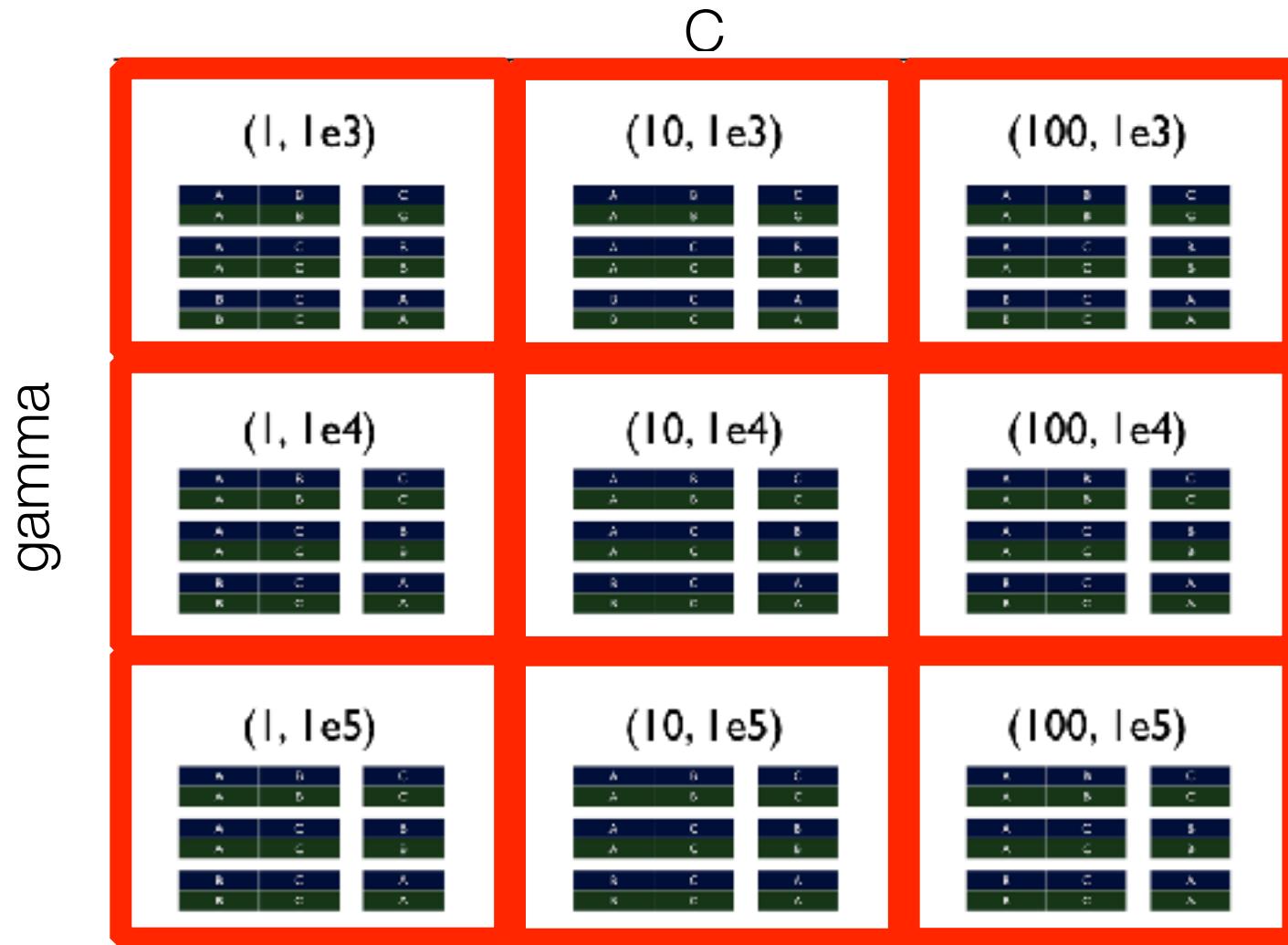
# Grid Searching

- Could perform iteratively

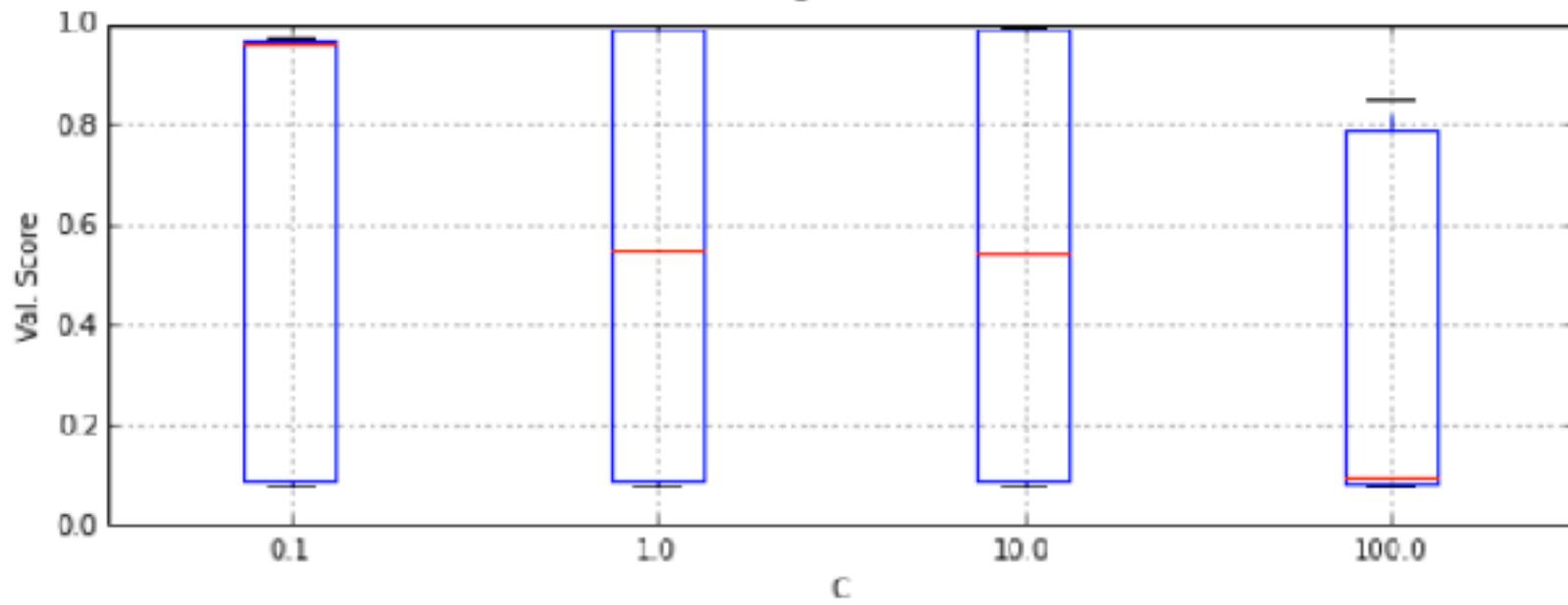
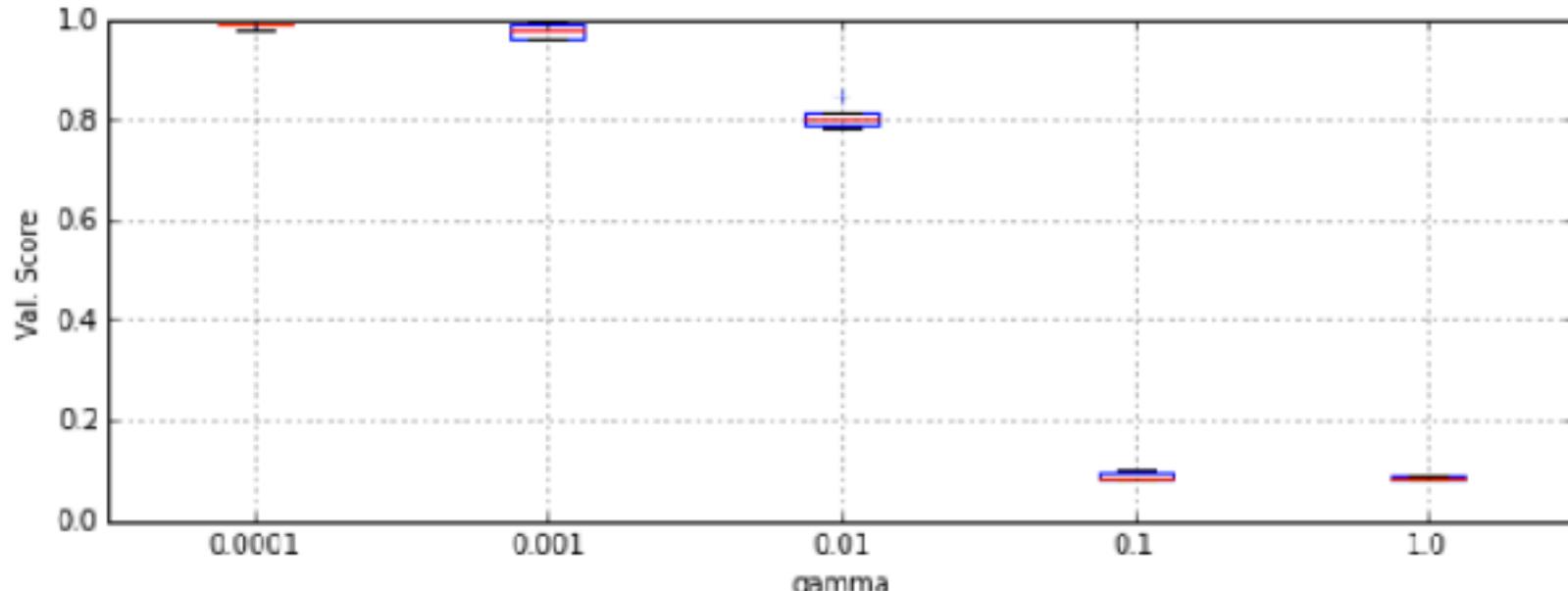


# Grid Searching

- or at random...



```
print(search.report())
search.boxplot_parameters(display_train=False)
```



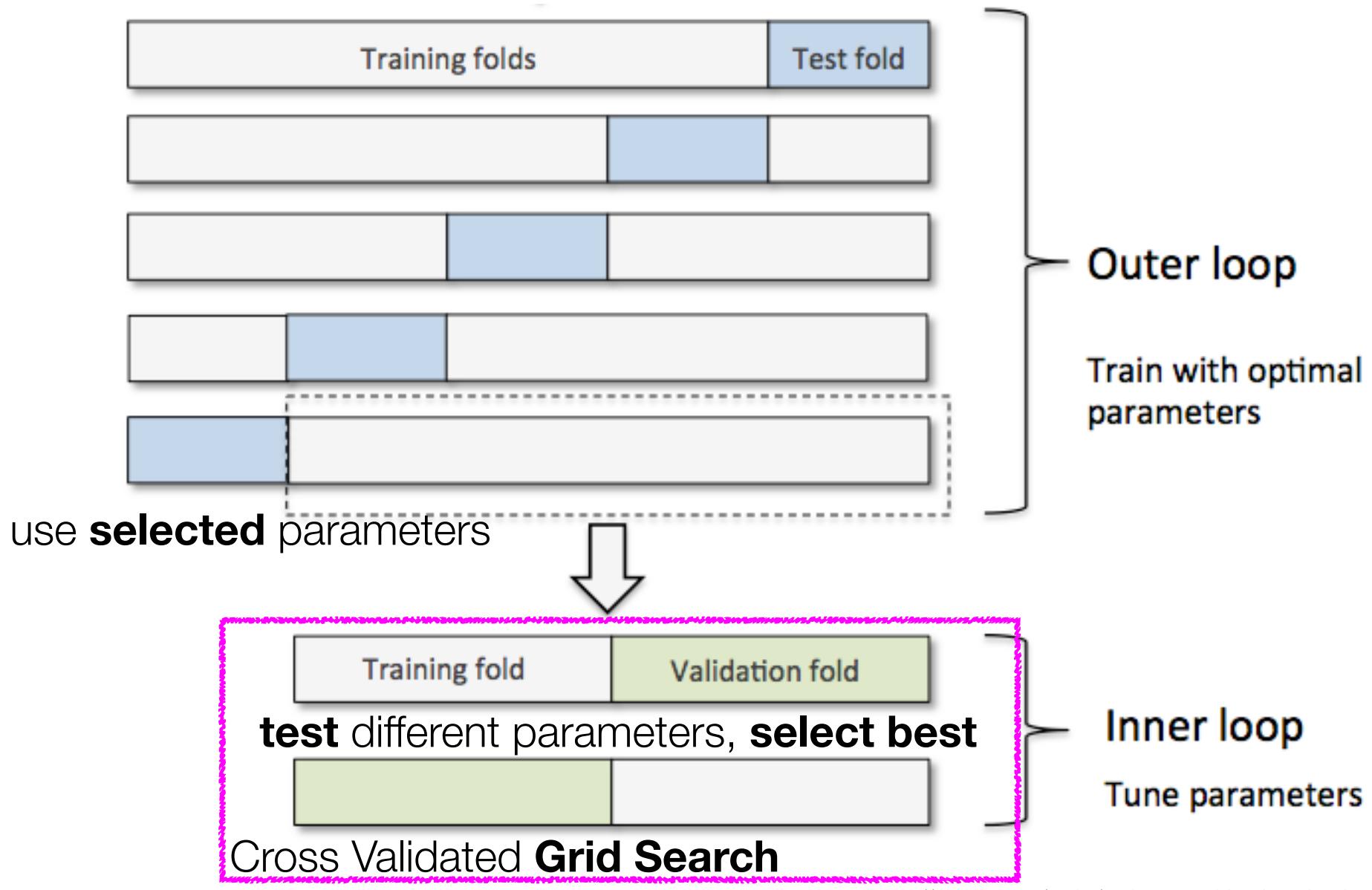
# The Problem

---

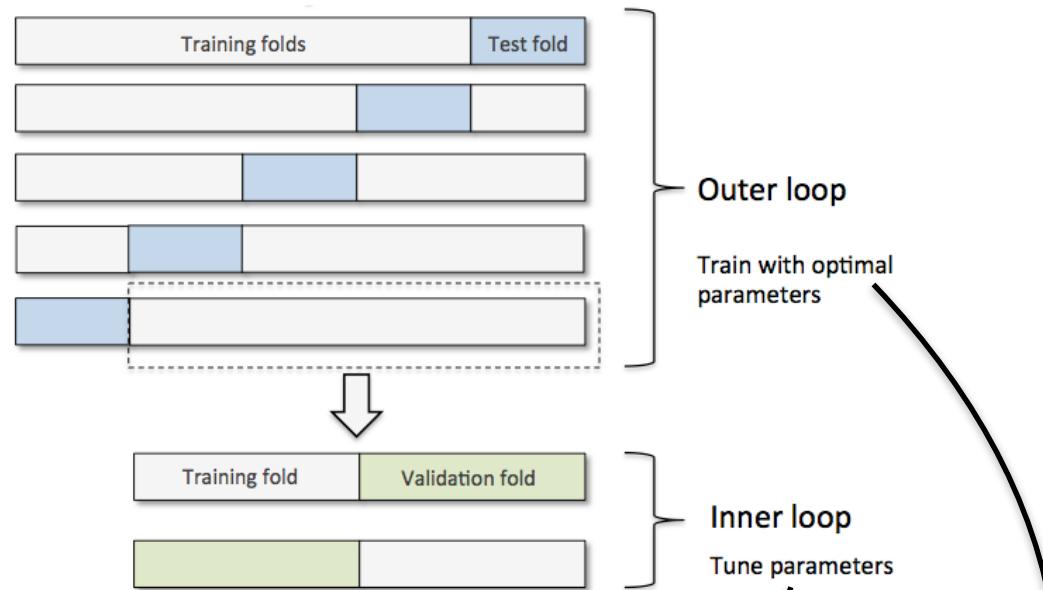
- Using the grid search parameters and testing on the same set...
  - the **performance on the dataset** could now be **biased**
  - cannot realistically determine the **expected performance** on new data



# Solution: Nested Cross Validation



# Nested Cross Validation: Hyper-parameters



```
gs = GridSearchCV(estimator=pipe_svc,  
                  param_grid=param_grid,  
                  scoring='accuracy',  
                  cv=2)
```

# Note: Optionally, you could use cv=2  
# in the GridSearchCV above to produce  
# the 5 x 2 nested CV that is shown in the figure.

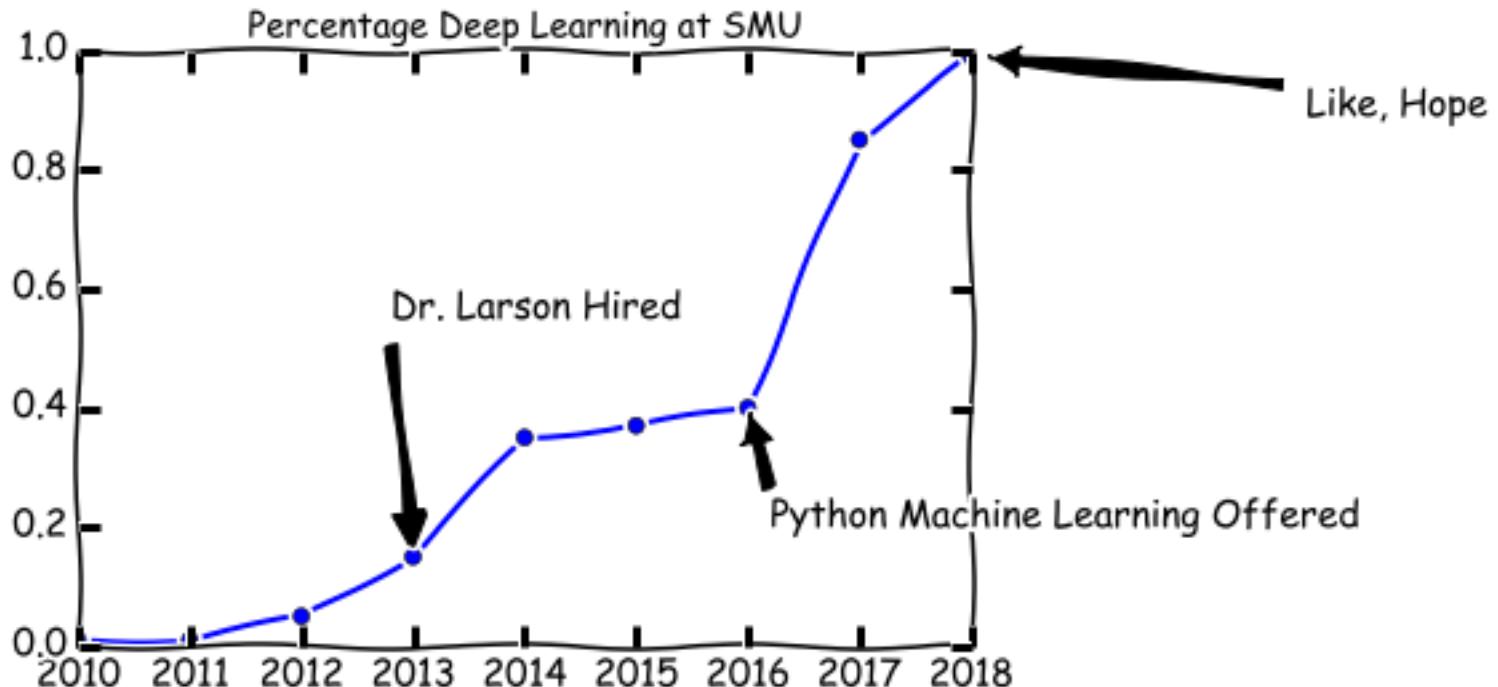
```
scores = cross_val_score(gs, X_train, y_train, scoring='accuracy', cv=5)  
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

# Self Test

---

- **What is the end goal of nested cross-validation?**
  - A. To determine hyper parameters
  - B. To estimate generalization performance
  - C. To estimate generalization performance when performing hyper parameter tuning
  - D. To estimate the variation in tuned hyper parameters

# 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



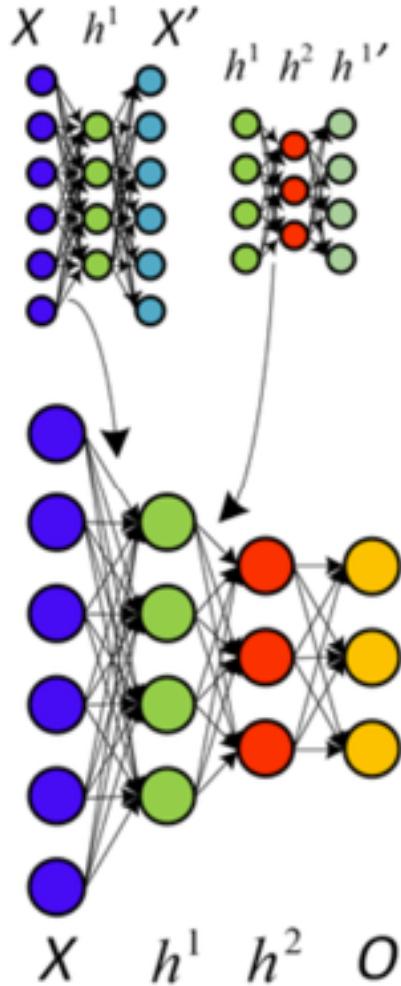
# History of Deep Learning: Winter

---

- 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

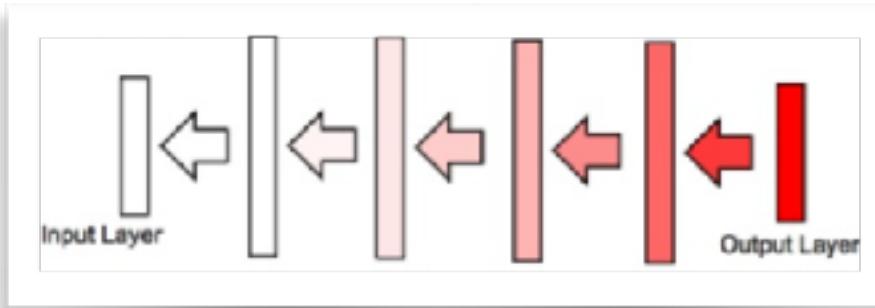
# History of Deep Learning: Winter

- 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)

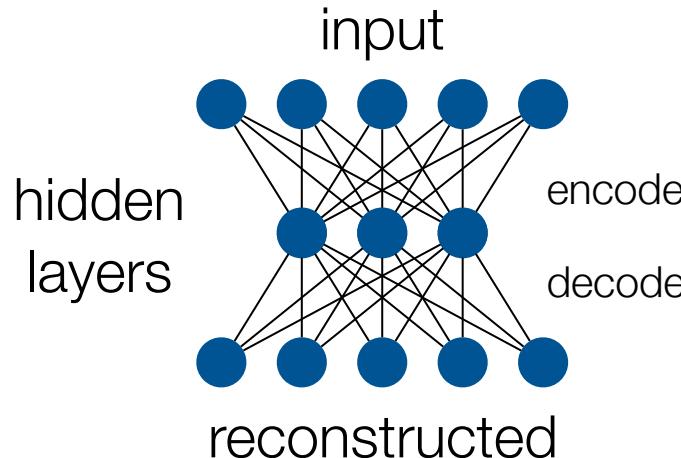
# History of Deep Learning

---

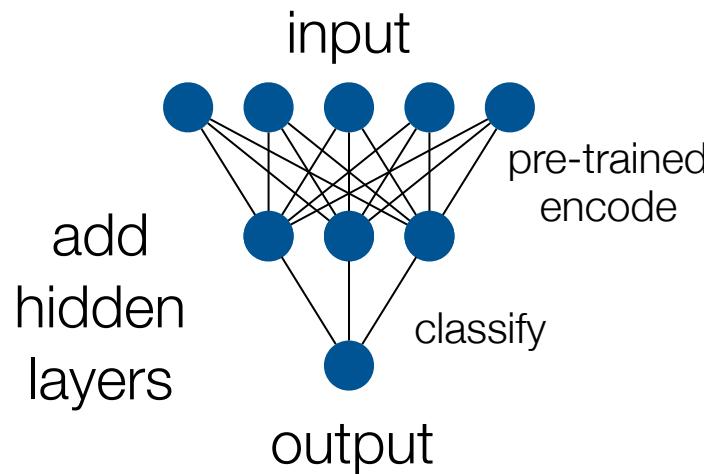
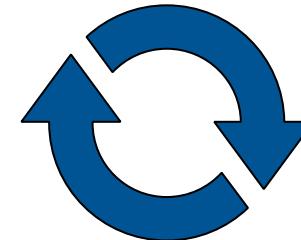
- 2004: Hinton secures funding from CIFAR based on his reputation
  - *eventually*: Canada would be savior for NN
  - Hinton rebrands: Deep Learning
- 2006: Hinton publishes paper on using pre-training and Restricted Boltzmann Machines
- 2007: Another paper: Deep networks are more efficient when pre-trained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

# Pre-training: still in the long winter

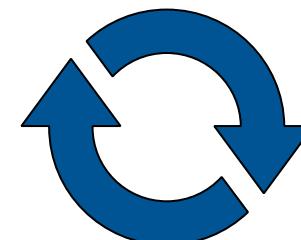
- auto-encoding



train with lots of  
unlabeled data

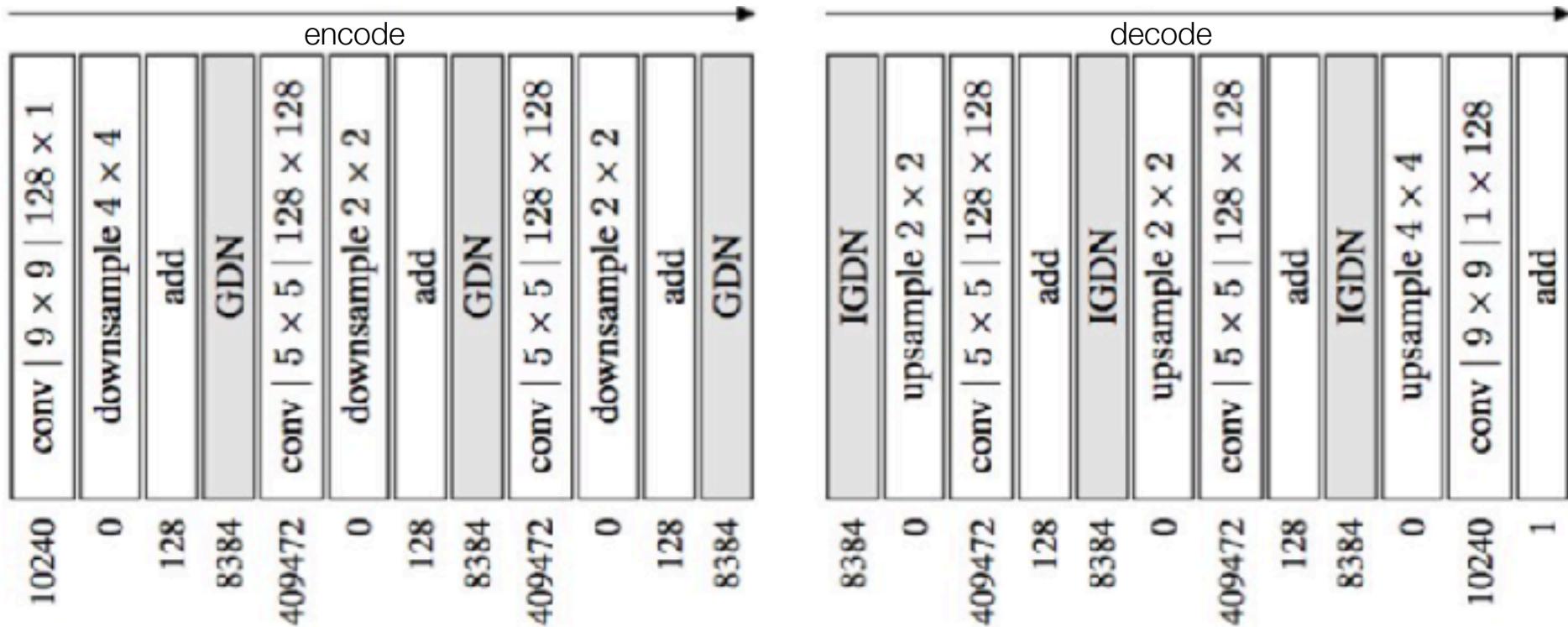


train with  
labeled data



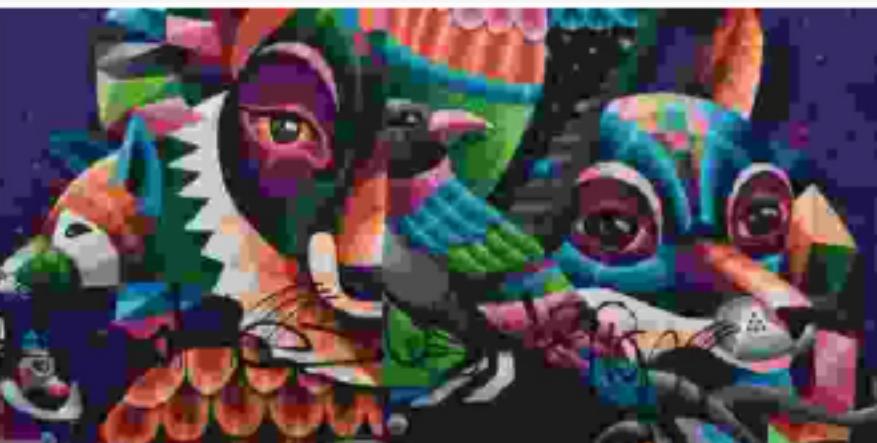
# Pre-training: modern example

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





JPEG, 5006 bytes (0.170 bit/px), RMSE: 19.75



JPEG, 5923 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



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



JPEG 2000, 5918 bytes (0.167 bit/px), RMSE: 17



JPEG 2000, 5724 bytes (0.162 bit/px), RMSE: 13.75/7.00, PSNR: 25.36 dB/31.20 dB



bit/px), RMSE: 8.56/5.71, PSNR: 29.49 dB/32.99 dB

# Getting to basic descriptors of images

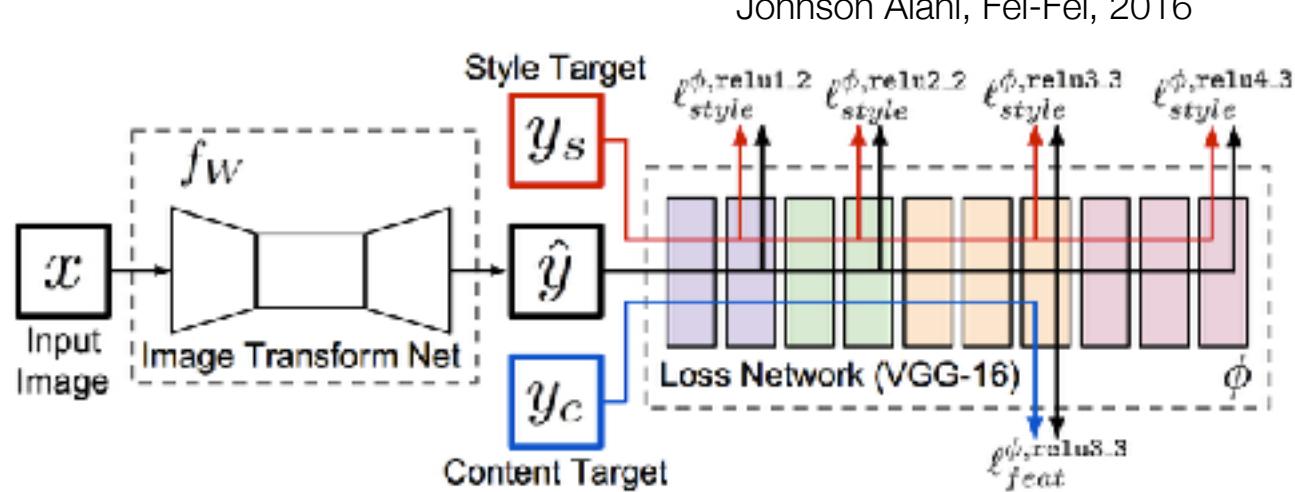
Style transfer



Gatys et al. 2015



Wood, Ledford, Pop 2017



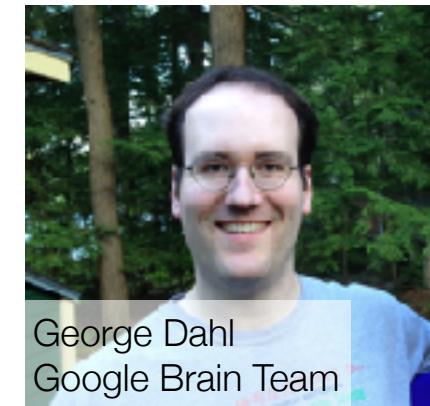
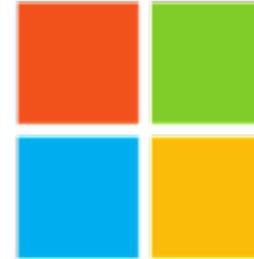
# History of Deep Learning

- 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



Navdeep Jaitly  
Google Brain Team

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



George Dahl  
Google Brain Team



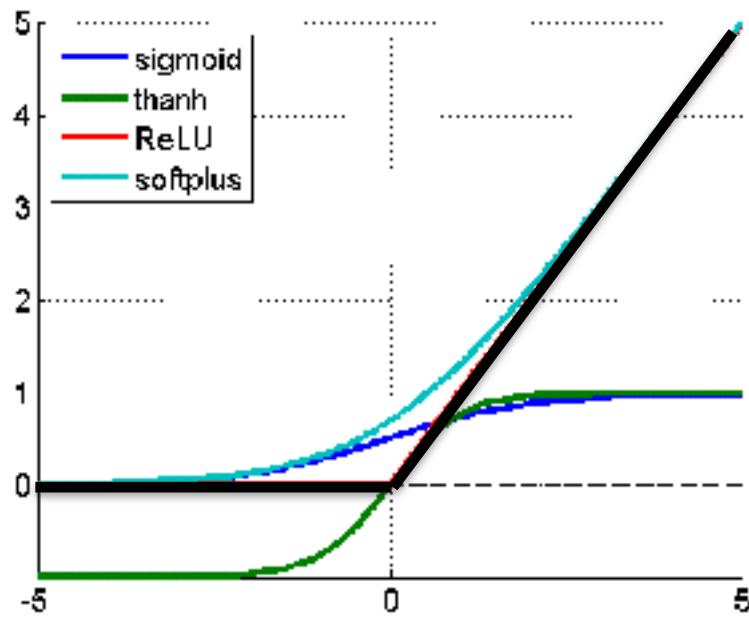
**Abdel-rahman Mohamed**

Microsoft Research  
Redmond, Washington | Computer Software

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

# 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



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

# History of Deep Learning

- ReLU not the only way to do it!

**Input:** Values of  $x$  over a mini-batch:  $B = \{x_1 \dots m\}$ ;  
 Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\begin{aligned} \mu_B &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && // \text{mini-batch mean} \\ \sigma_B^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2 && // \text{mini-batch variance} \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} && // \text{normalize} \\ y_i &\leftarrow \gamma \hat{x}_i + \beta = \text{BN}_{\gamma, \beta}(x_i) && // \text{scale and shift} \end{aligned}$$

## Batch Normalization

Normalize input layers per mini-batch and add control parameters,  $\gamma$  and  $\beta$

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

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

$$\frac{\partial \ell}{\partial \sigma_B^2} = \sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \cdot (x_i - \mu_B) \cdot \frac{-1}{2} (\sigma_B^2 + \epsilon)^{-3/2}$$

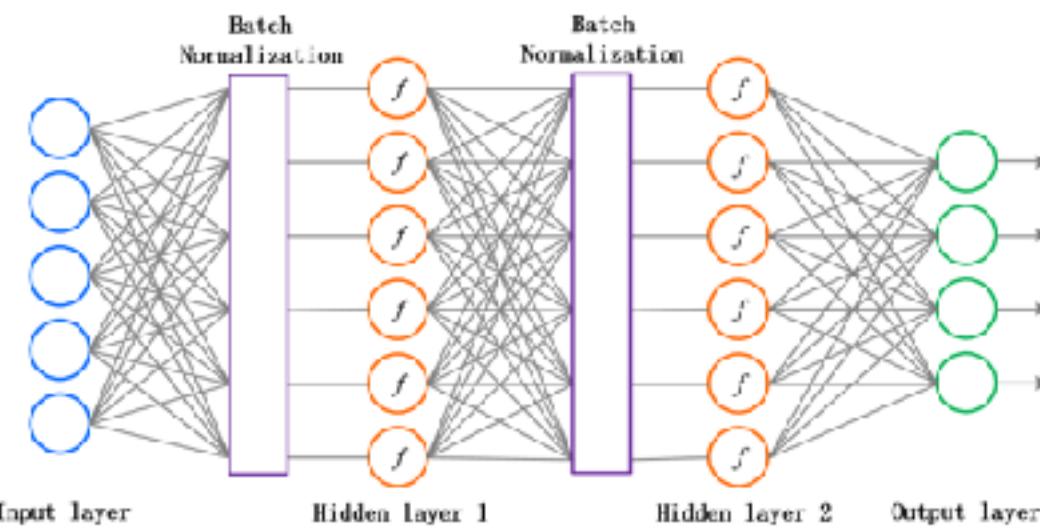
$$\frac{\partial \ell}{\partial \mu_B} = \left( \sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_B^2} \cdot \frac{\sum_{i=1}^m -2(x_i - \mu_B)}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_B^2} \cdot \frac{2(x_i - \mu_B)}{m} + \frac{\partial \ell}{\partial \mu_B} \cdot \frac{1}{m}$$

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

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

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



# History of Deep Learning

- ReLU not the only way to do it!

## 2 SWISH

We propose a new activation function that we name *Swish*:

$$f(x) = x \cdot \sigma(x) \quad (1)$$

where  $\sigma(x) = (1 + \exp(-x))^{-1}$  is the sigmoid function. See Figure 1 for the graph of Swish.

\*Work done as a member of the Google Brain Residency program ([g.co/brainresidency](https://g.co/brainresidency)).

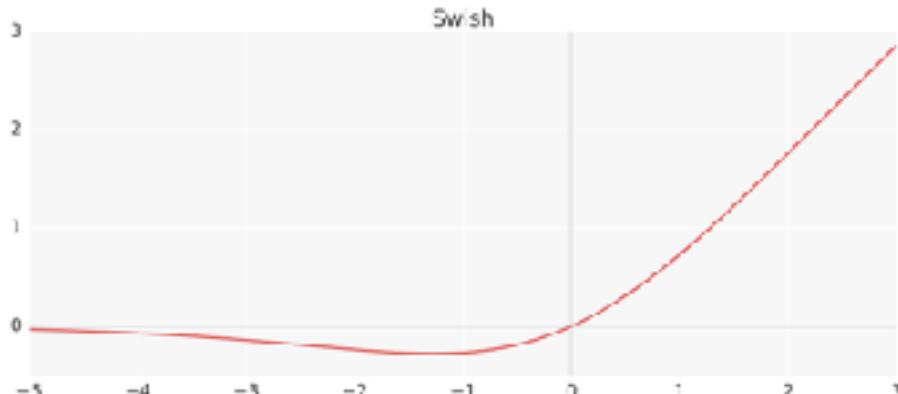
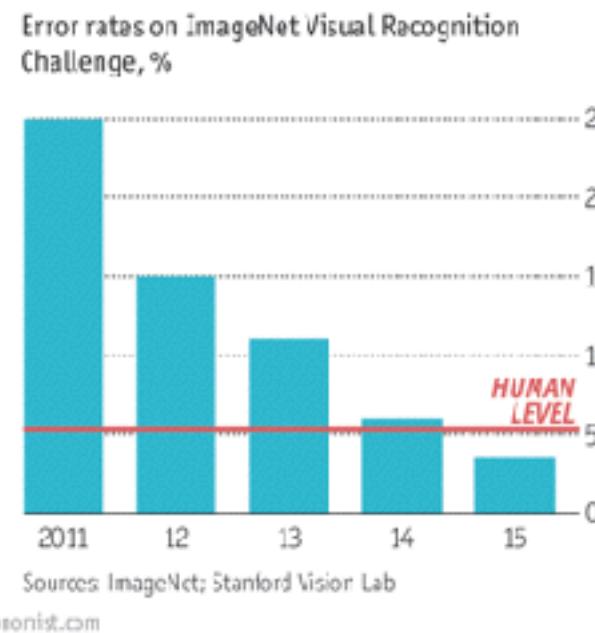


Figure 1: The Swish activation function.

$$\begin{aligned}f'(x) &= \sigma(x) + x \cdot \sigma(x)(1 - \sigma(x)) \\&= \sigma(x) + x \cdot \sigma(x) - x \cdot \sigma(x)^2 \\&= x \cdot \sigma(x) + \sigma(x)(1 - x \cdot \sigma(x)) \\&= f(x) + \sigma(x)(1 - f(x))\end{aligned}$$

# History of Deep Learning

- 2012: 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 masse



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

## Deep Neural Networks for Acoustic Modeling in Speech Recognition

[The shared views of four research groups]

[Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly,  
Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury]

[https://www.cs.toronto.edu/~gdahl/papers/  
deepSpeechReviewSPM2012.pdf](https://www.cs.toronto.edu/~gdahl/papers/deepSpeechReviewSPM2012.pdf)

# History of Deep Learning

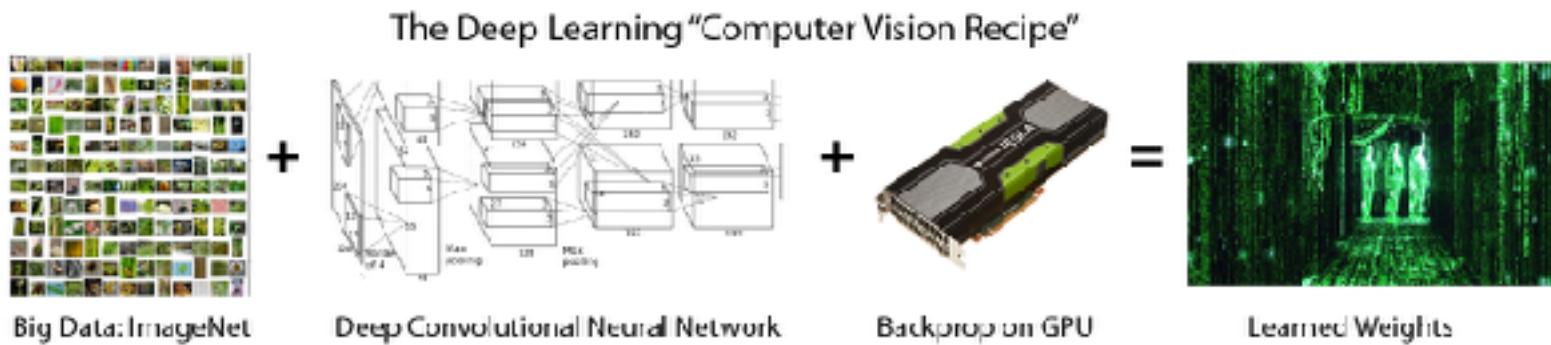
- 2013: Ng and Google (founded BrainTeam)
  - run unsupervised feature creation on YouTube videos (becomes computer vision benchmark)

*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.*



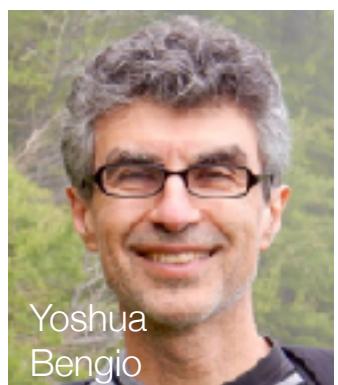
# History of Deep Learning

- 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 Larson's terms:
  - use a GPU when possible, init weights for consistent gradient magnitude, ReLU or BN where it makes sense (like in early feedforward layers), and lots of dropout in the final layers that tend to over learn quickly!
- 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:



Yoshua  
Bengio

Stayed at  
University

Advises IBM



Yann  
LeCun

Heads  
Facebook  
AI Team



Geoffrey  
Hinton

Google



FeiFei  
Li

Google  
Cloud



Andrew  
Ng

Coursera  
Baidu  
Google

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

And predicts the future of  
deep learning

## REVIEW

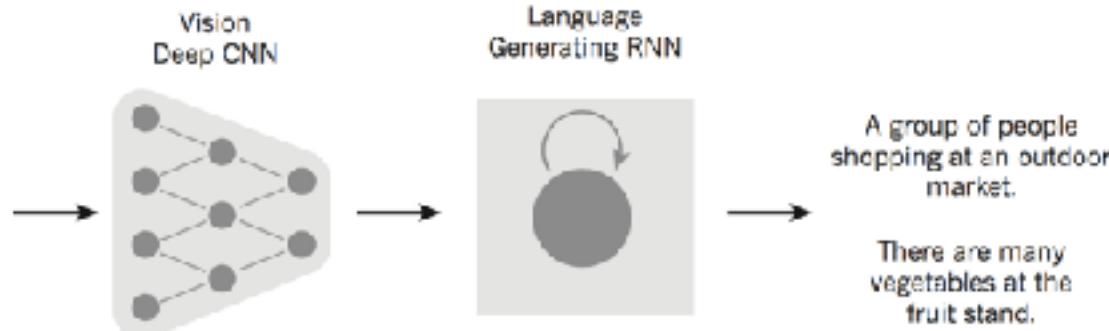
doi:10.1038/nature14539

### Deep learning

Yann LeCun<sup>1,2</sup>, Yoshua Bengio<sup>3</sup> & Geoffrey Hinton<sup>4,5</sup>

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 genetics. 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:



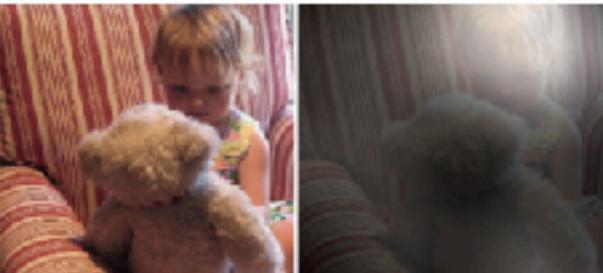
A woman is throwing a frisbee in a park.



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.

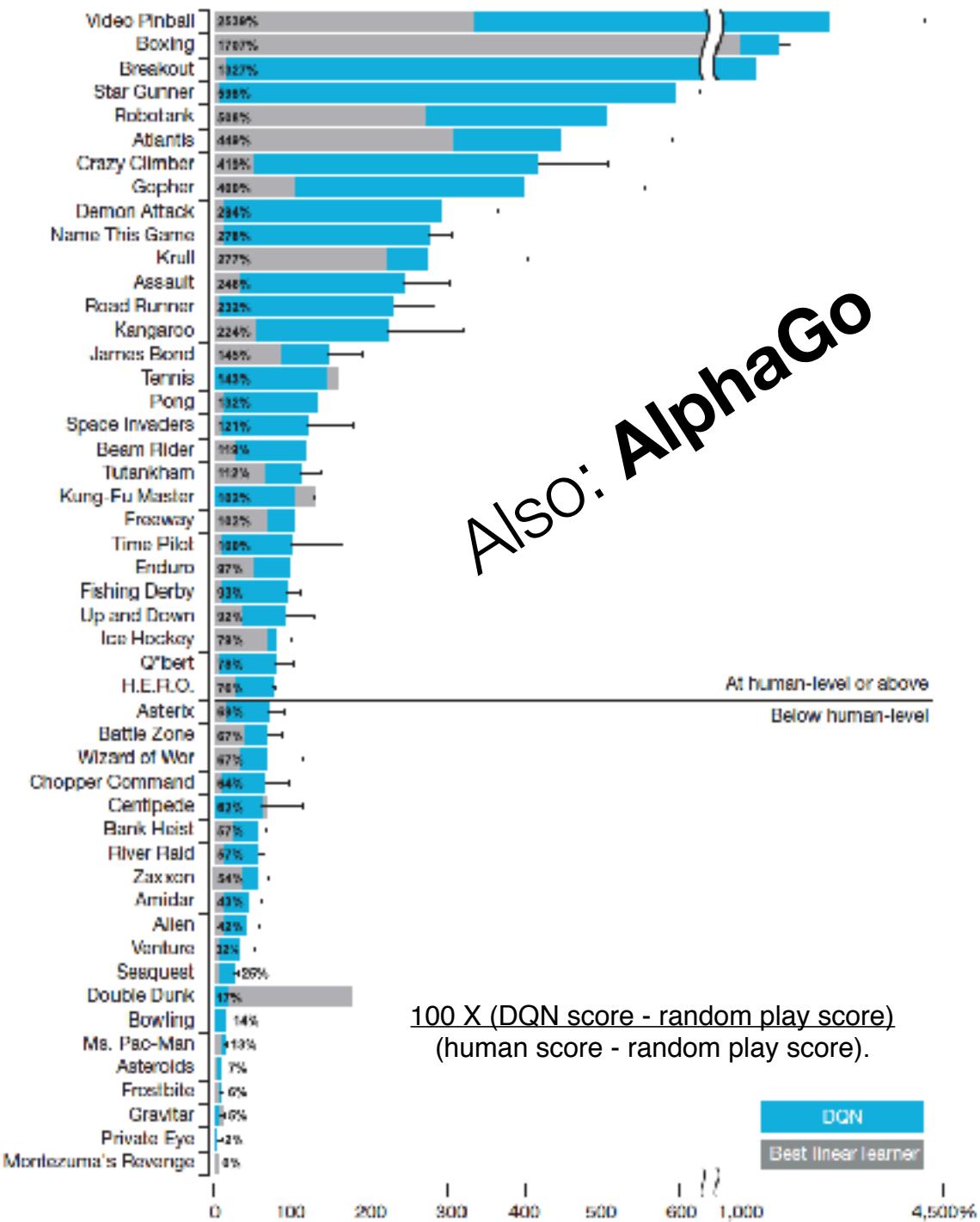
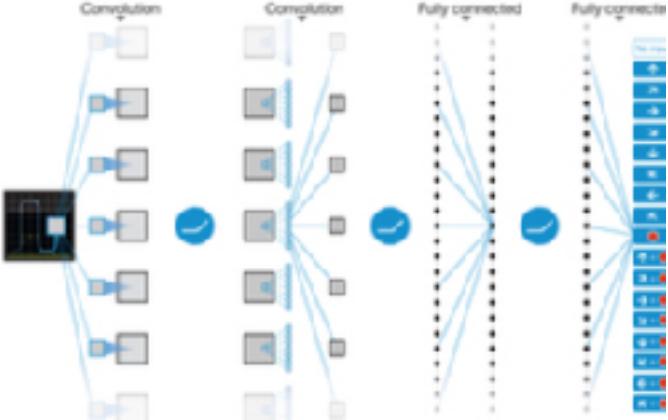


A group of people sitting on a boat in the water.



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

# More Famous examples: using reinforcement learning



# 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

# 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 (not called neural networks)
- 2006: Auto-encoding and Restricted Boltzmann Machines
- 2007: Deep networks are more efficient when pre-trained
  - RBMs not really the important part
- 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, BP, Swish)

# What else did we talk about last time?

---



Yoshua  
Bengio



Yann  
LeCun



Geoffrey  
Hinton



FeiFei  
Li



Andrew  
Ng

Stayed at  
University

Advises IBM

Heads  
Facebook  
AI Team

Google

Google  
Cloud

Coursera  
Baidu  
Google

- Now lets look at the assignment...

# TensorFlow



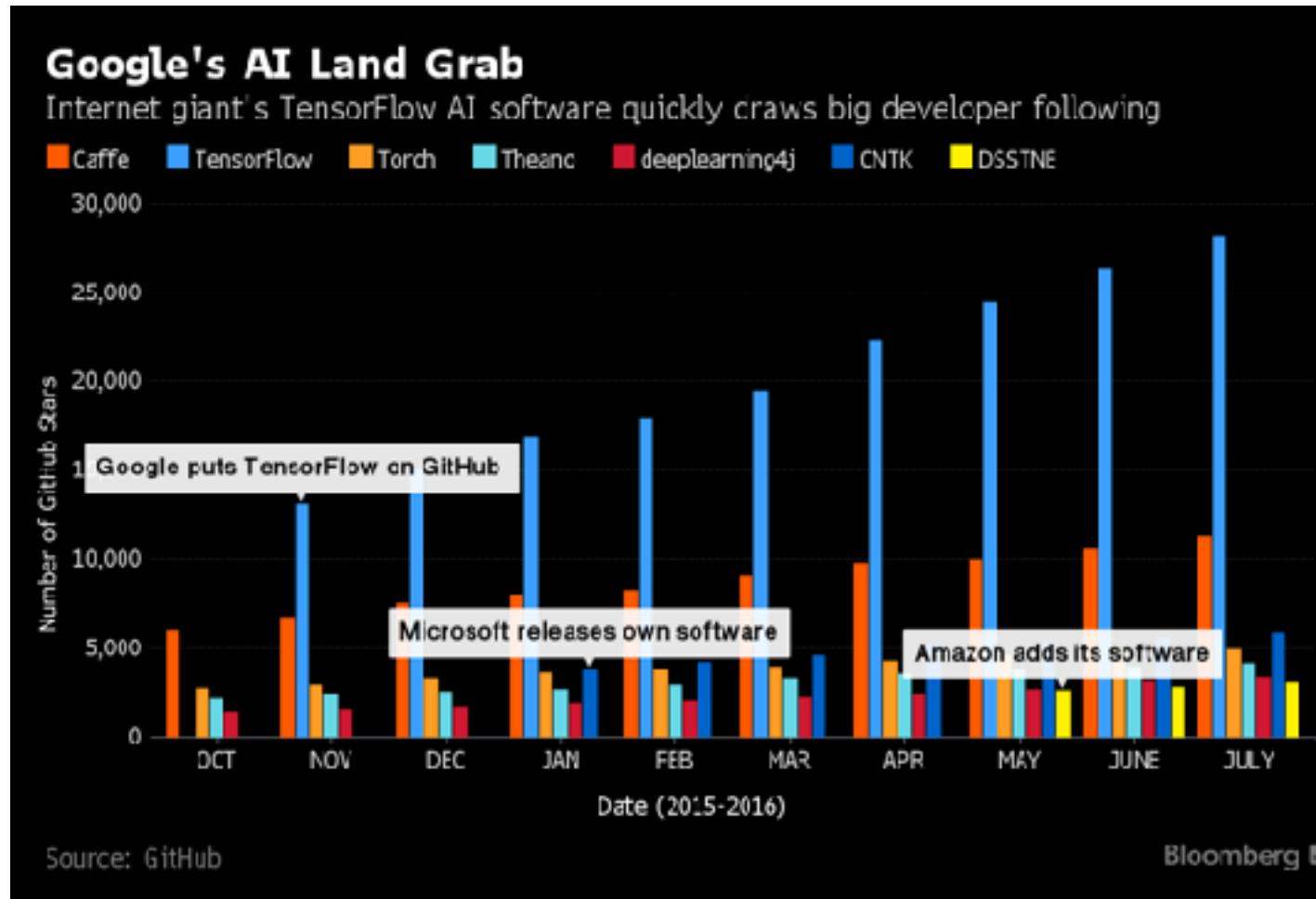
Matthew Rocklin @mrocklin · Apr 5



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

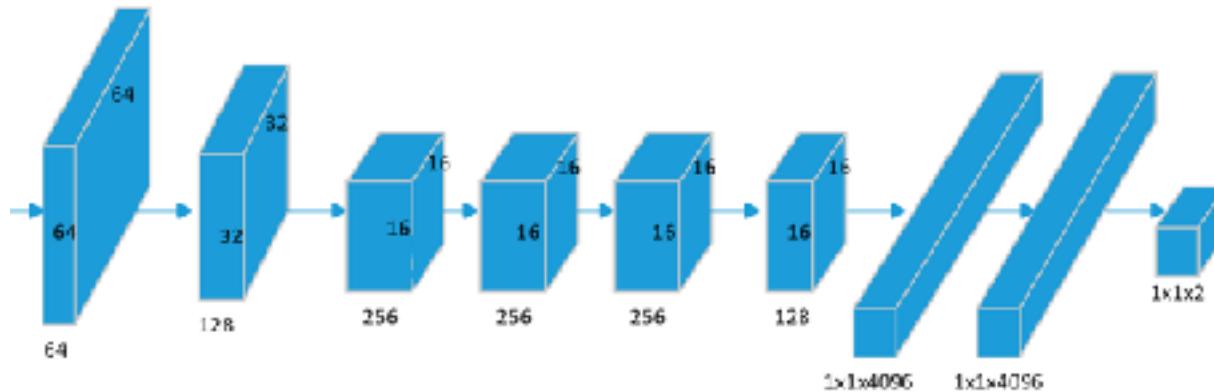
---

- 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
<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
- `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)
```

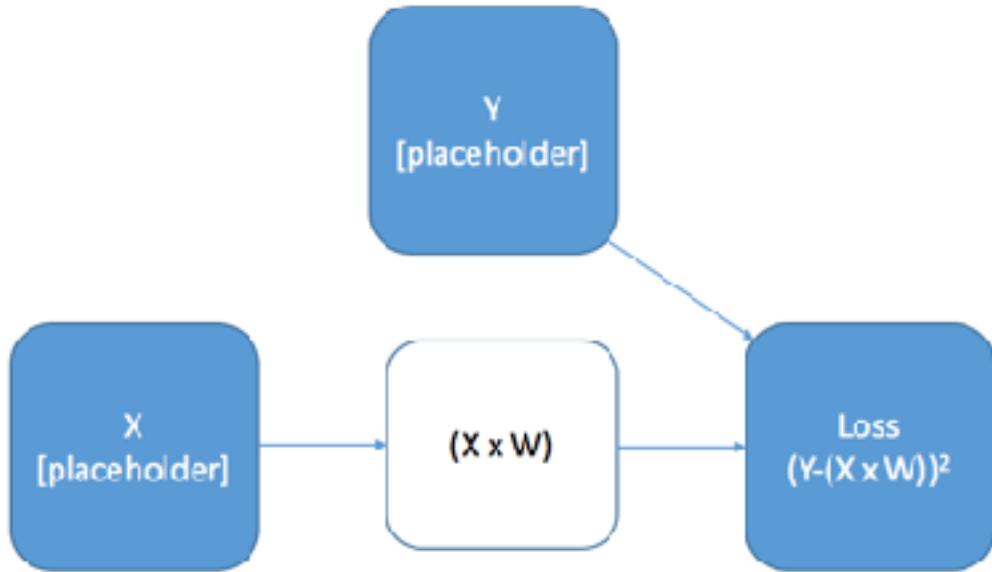
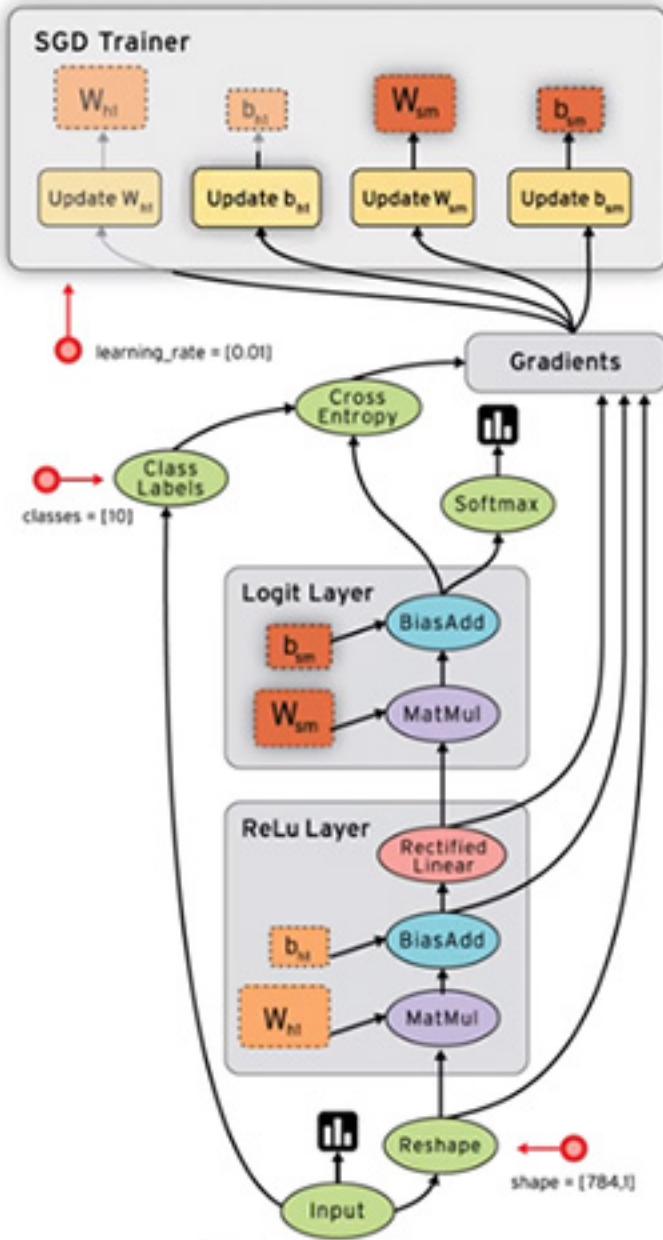
```
b = tf.constant(6.0)
```

```
c = a * b
```

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

output = 30

# Computation Graph



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

48  
<http://www.datasciencecentral.com/profiles/blogs/google-open-source-tensorflow>

# Tensorflow for Neural Networks

- Starting Simple...

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

```
# 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})
```

# 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})
```

# 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”
    - ✓ • common methods (generative/adversarial, transfer learning, etc.) already have high level wrappers
    - ✗ • even more advanced architectures require using pure TensorFlow

# Tensor General Training Overview

---

- Keras Functional API
  - build models through series of nested functions
  - each “function” represents an operation in the NN
  - “sparse” has multiple representations
    - scipy sparse
    - one-hot encoded sparse

## TensorFlow with Keras

Reinventing the MLP  
Wheel

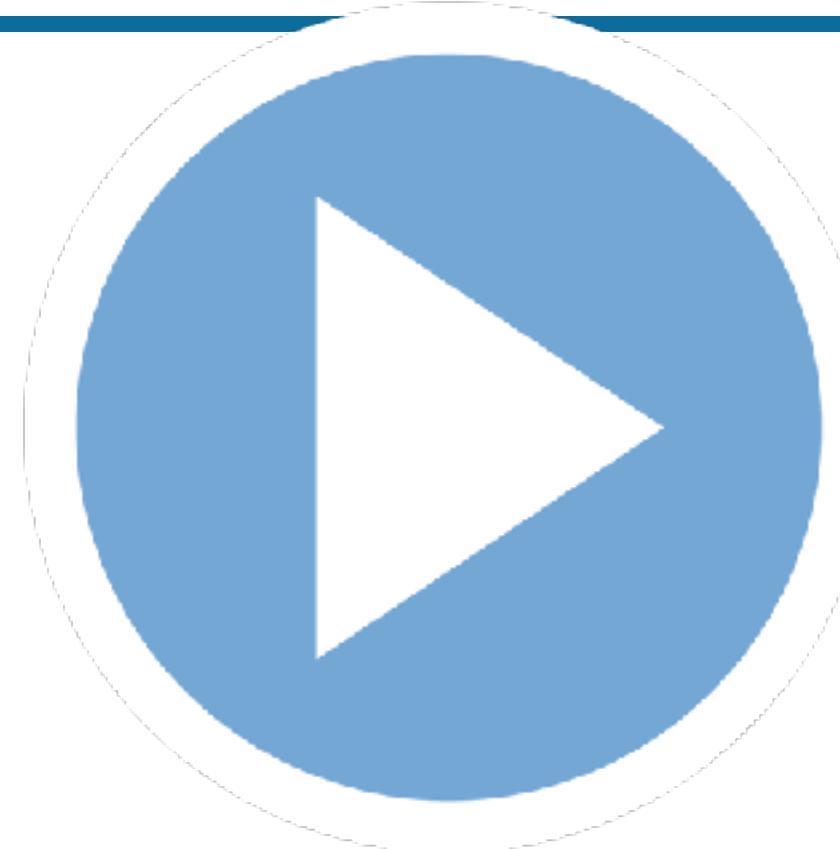
Other tutorials:

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

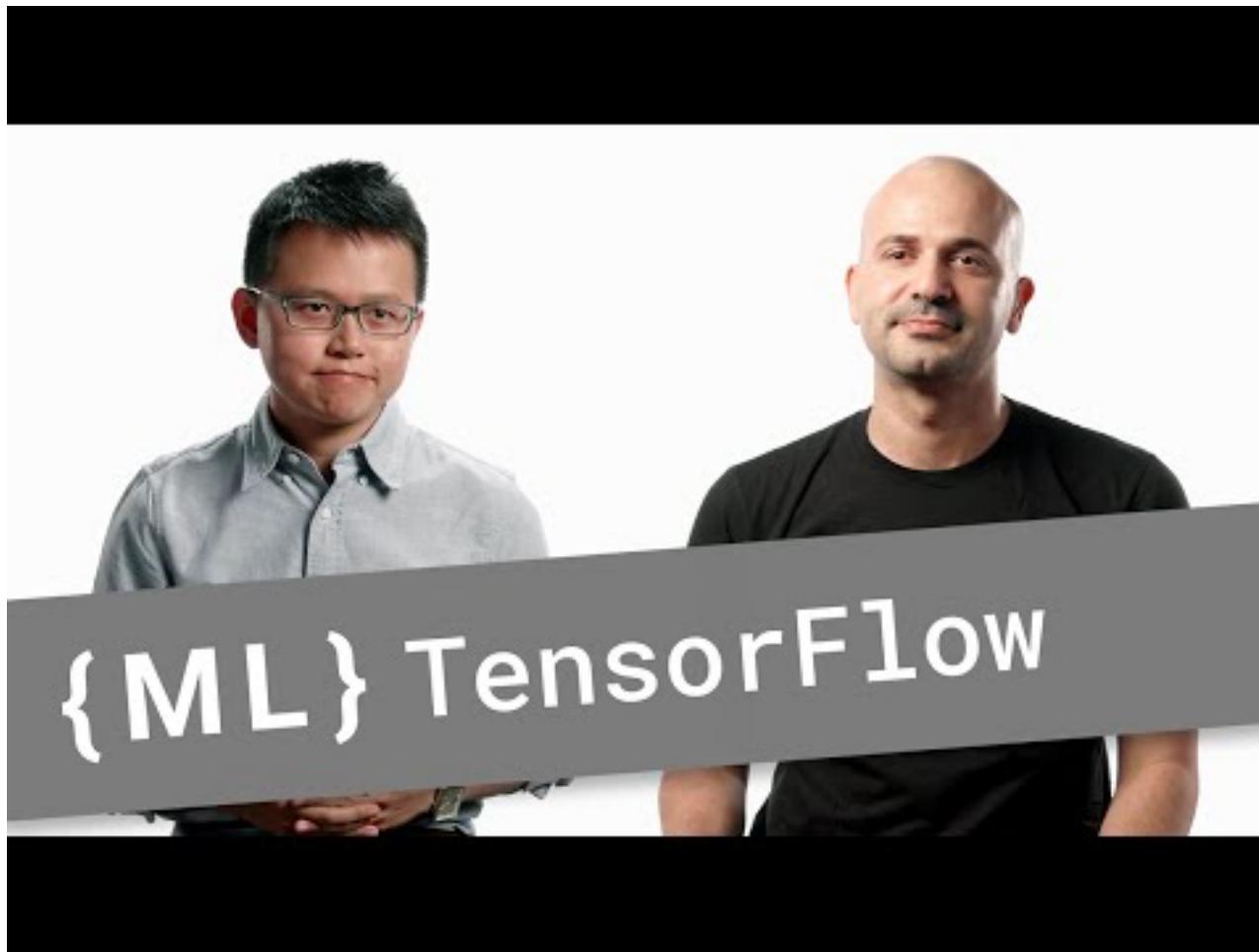
<https://elitedatascience.com/keras-tutorial-deep-learning-in-python>

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

Make me slow down if I go too fast!!



# 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. Recommendations based on memorization are usually more topical and

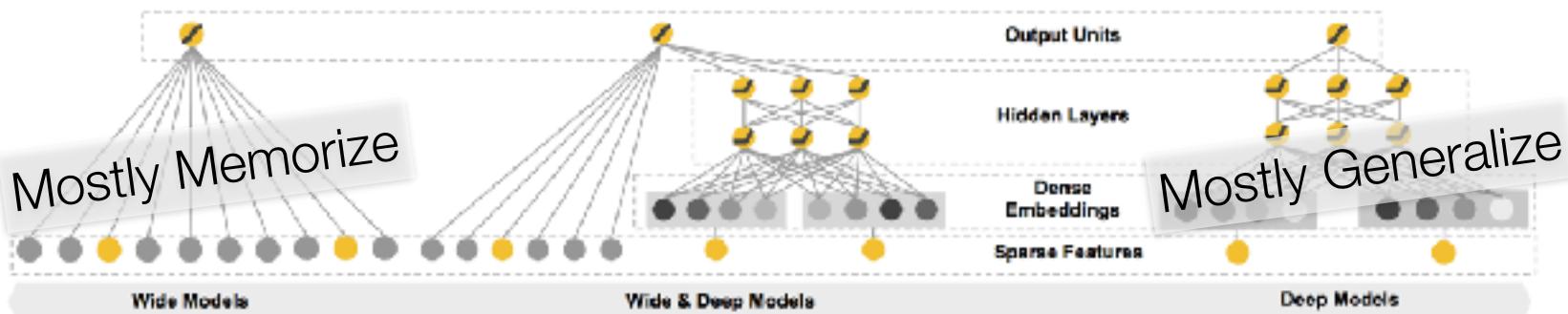
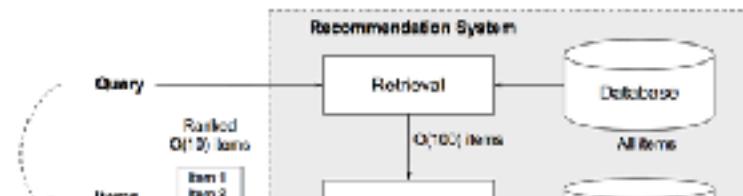


Figure 1: The spectrum of Wide & Deep models.

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

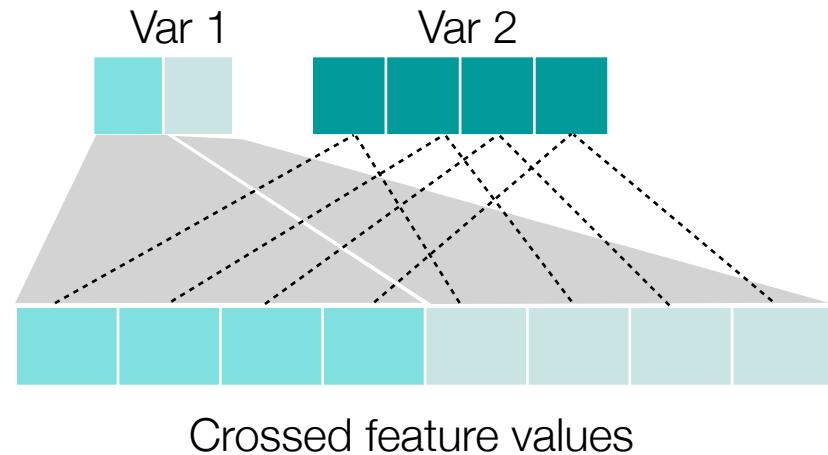
- The implementation and evaluation of the Wide & Deep recommender system productionized on Google



# Wide networks (Memorize?)

---

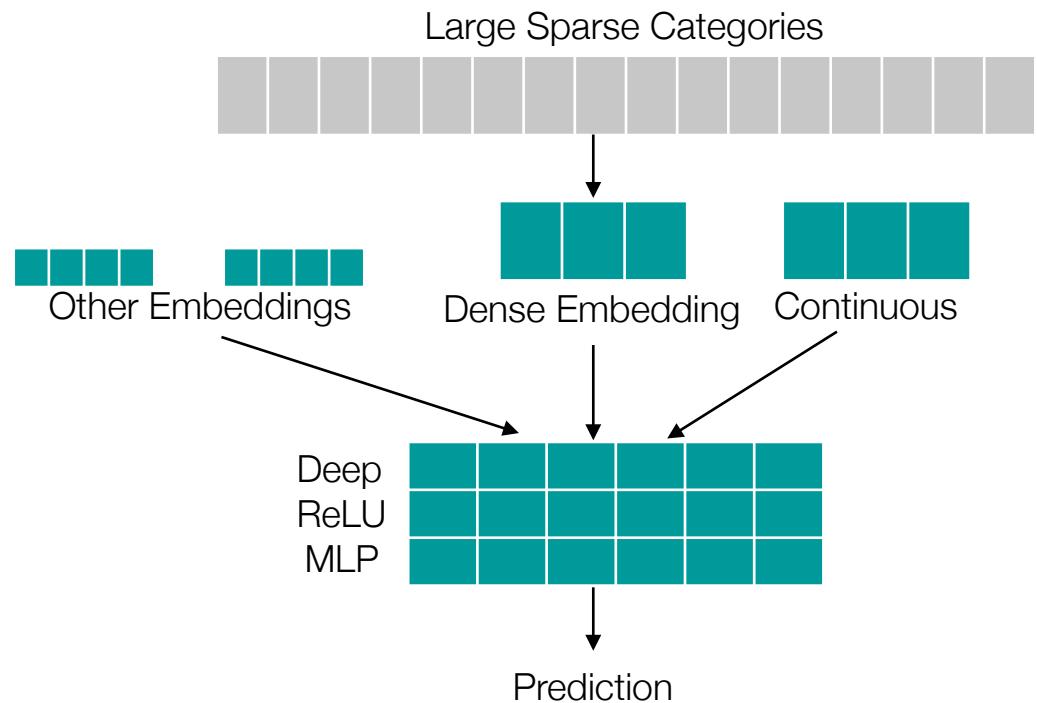
- Wide refers to the expansion of your features set
- Crossed feature columns of categorical features
  - Movie Rating
    - G
    - PG
    - PG-13
    - R
    - Else
  - Movie Genre
    - Action
    - Drama
    - Comedy
    - Horror
    - Else
- Crossed feature “Rating-Genre”
  - G-Action, G-Drama, G-Comedy, G-Horror, G-else
  - PG-Action, PG-Drama, PG-Comedy, PG-Horror, G-else
  - and so on ... one hot encoded



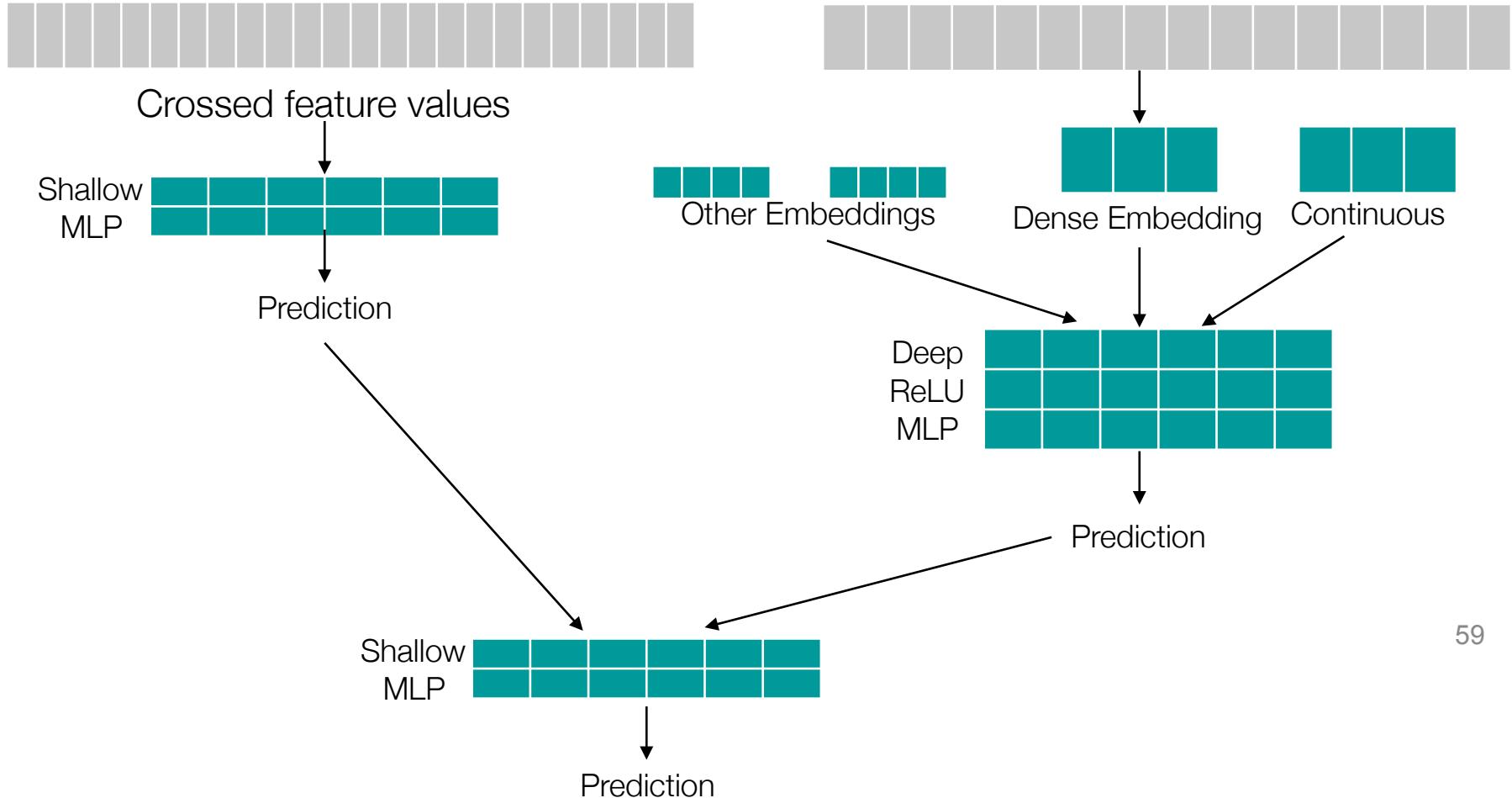
# Sparse Embedding and Deep MLP (Generalize?)

---

- 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

The awful dataset:  
Toy Census Data Example

Other tutorials:

[https://www.tensorflow.org/tutorials/wide\\_and\\_deep](https://www.tensorflow.org/tutorials/wide_and_deep)



# End of Session

---

- Next Time:
  - Convolutional Neural Networks

---

# Back Up Slides

# History of Deep Learning: summary

---

- 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 (not called neural networks)
- 2006: Auto-encoding and Restricted Boltzmann Machines
- 2007: Deep networks are more efficient when pre-trained
  - RBMs not really the important part
- 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)

# Tensor Variables

---

- Variables that must be initialized before use

```
W1 = tf.ones((2,2)) immutable
```

```
W2 = tf.Variable(tf.zeros((2,2)), name="weights") mutable
```

```
with tf.Session() as sess:  
    print(sess.run(W1))  
    sess.run(tf.initialize_all_variables())  
    print(sess.run(W2))
```

---

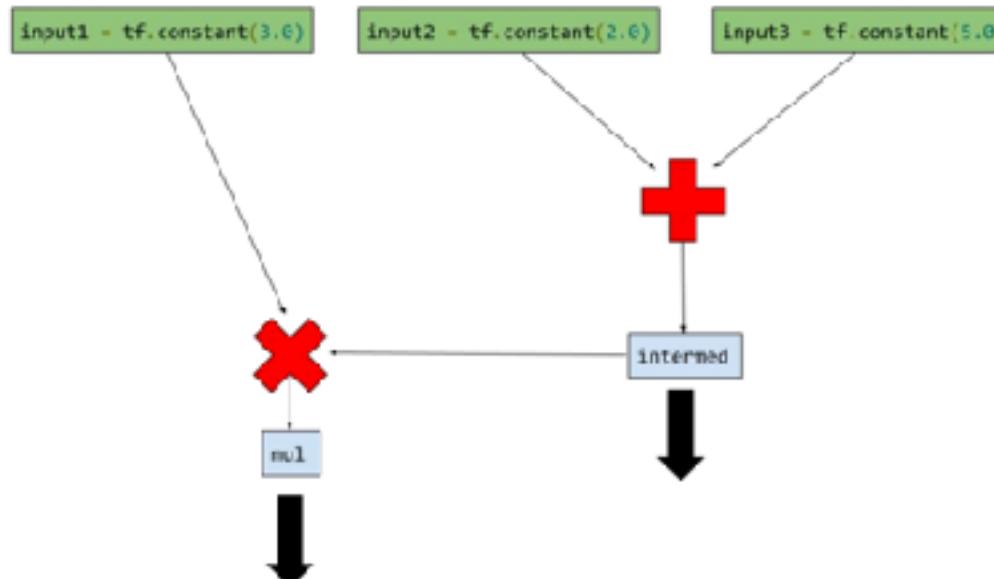
```
W = tf.Variable(tf.zeros((2,2)), name="weights")
```

```
R = tf.Variable(tf.random_normal((2,2)), name="random_weights")
```

# Tensor Variables

- Variables that must be initialized before use

```
input1 = tf.constant(3.0)
input2 = tf.constant(2.0)
input3 = tf.constant(5.0)
intermed = tf.add(input2, input3) mutable variable
mul = tf.mul(input1, intermed) mutable variable
with tf.Session() as sess:
    result = sess.run([mul, intermed]) fetch variable values
    print(result)
```



# Tensor Placeholders

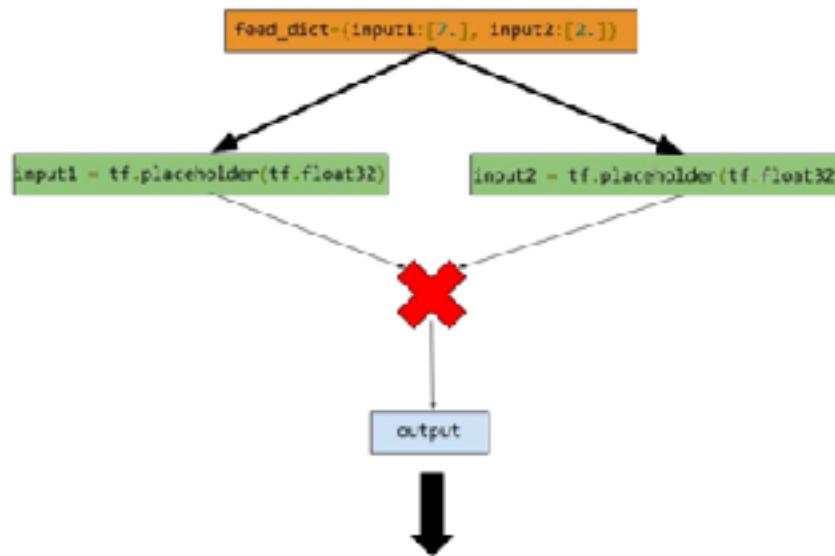
- Placeholders await values and can be streamed in from the computation graph

```
input1 = tf.placeholder(tf.float32)

input2 = tf.placeholder(tf.float32)

output = tf.mul(input1, input2)

with tf.Session() as sess:
    print(sess.run([output], feed_dict={input1:[7.], input2:[2.]}))
```



# Tensor get\_variable and scope

- Just a form of scoping
  - define and reuse variables
  - if you have not defined variable as “reuse,” then you can only access it once via get\_variable

## Variable Scope Example

Variable Scope mechanism in TensorFlow consists of 2 main functions:

- `tf.get_variable(<name>, <shape>, <initializer>)`: Creates or returns a variable with a given name.
- `tf.variable_scope(<scope_name>)`: Manages namespaces for names passed to `tf.get_variable()`.

```
with tf.variable_scope("foo"):  
    v = tf.get_variable("v", [1])      v.name == "foo/v:0"  
  
with tf.variable_scope("foo", reuse=True):  
    v1 = tf.get_variable("v", [1])      without "reuse" this is an error
```

# Tensor get\_variable and scope

- Why do we need this?
  - Reusing variable creation code

```
def my_image_filter(input_images):
    conv1_weights = tf.Variable(tf.random_normal([5, 5, 32, 32]),
                               name="conv1_weights")
    conv1_biases = tf.Variable(tf.zeros([32]), name="conv1_biases")
    conv1 = tf.nn.conv2d(input_images, conv1_weights,
                        strides=[1, 1, 1, 1], padding='SAME')
    relu1 = tf.nn.relu(conv1 + conv1_biases)

    conv2_weights = tf.Variable(tf.random_normal([5, 5, 32, 32]),
                               name="conv2_weights")
    conv2_biases = tf.Variable(tf.zeros([32]), name="conv2_biases")
    conv2 = tf.nn.conv2d(relu1, conv2_weights,
                        strides=[1, 1, 1, 1], padding='SAME')
    return tf.nn.relu(conv2 + conv2_biases)
```

**explicit definition of  
repeated operations**

# Tensor get\_variable and scope

- Why do we need this?
  - Reusing variable creation code

```
def conv_relu(input, kernel_shape, bias_shape):
    # Create variable named "weights".
    weights = tf.get_variable("weights", kernel_shape,
        initializer=tf.random_normal_initializer())
    # Create variable named "biases".
    biases = tf.get_variable("biases", bias_shape,
        initializer=tf.constant_initializer(0.0))
    conv = tf.nn.conv2d(input, weights,
        strides=[1, 1, 1, 1], padding='SAME')
    return tf.nn.relu(conv + biases)

def my_image_filter(input_images):
    with tf.variable_scope("conv1"):
        # Variables created here will be named "conv1/weights", "conv1/biases".
        relu1 = conv_relu(input_images, [5, 5, 32, 32], [32])
    with tf.variable_scope("conv2"):
        # Variables created here will be named "conv2/weights", "conv2/biases".
        return conv_relu(relu1, [5, 5, 32, 32], [32])
```

more readable code

THE NOVEMBER 2016  
GUIDE TO MAKING PEOPLE  
**FEEL OLD**

IF THEY'RE [AGE], YOU SAY:

"DID YOU KNOW [THING] HAS BEEN  
AROUND FOR A MAJORITY OF YOUR LIFE?"

AGE	THING
16	GRAND THEFT AUTO IV
17	RICKROLLING
18	AQUARIUM HUNGER FORCE COLON MOVIE FILM FOR THEATRES
19	THE NINTENDO WII
20	TWITTER
21	THE XBOX 360, XKCD
22	CHUCK NORRIS FACTS
23	OPPORTUNITY'S MARS EXPLORATION
24	FACEBOOK
25	GMAIL, PIRATES OF THE CARIBBEAN
26	IN DA CLUB
27	FIREFOX
28	THE WAR IN AFGHANISTAN
29	THE IPOD
30	SHREK, WIKIPEDIA
31	THOSE X-MEN MOVIES
32	THE SIMS
33	AUTOTUNED HIT SONGS
34	THE STAR WARS PREQUELS
35	THE MATRIX
36	POKÉMON RED & BLUE
37	NETFLIX, HARRY POTTER, GOOGLE
38	DEEP BLUE'S VICTORY
39	TUPAC'S DEATH
40	THE LAST CALVIN AND HOBBES STRIP
41	JOY STICKY
>41	[DON'T WORRY, THEY'VE GOT THIS COVERED]

# Sampling and Modeling

# Sampling and Modeling

---

- As of November 8th (morning) the highest rank for Trump winning the presidency was 33% (Nate Silver)
- Most other models were at <1% and were critical of *fivethirtyeight* (Nate Silver)
- **SELF TEST:** Why was this polling/modeling flawed?
  - (A) Insufficient sampling in key geographic areas
    - Under-sampling voters in rural areas (more difficult to contact)
    - Under-sampling minority voters (assume mirroring to Obama)
  - (B) Reliance on Bayesian estimation models
  - (C) Impossible cross validation due to changing environment
  - (D) Insufficient previous examples, even if environment was not changing
- Was there a way to make a better model of the election outcome?
- Some ensemble models performed well, but used features that only included:
  - time incumbent was in office, approval ratings for incumbent, and GDP growth, (other economic factors decreased performance of models)
  - even these models had wide confidence intervals, no clear winner