

Lecture Notes for **Machine Learning in Python**

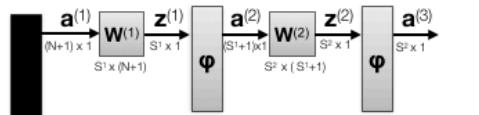
Revisiting CV A “Way too Early” History of Deep Learning

Logistics and Agenda

- Logistics
 - None!
- Agenda
 - Cross Validation
 - Town Hall
 - “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



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

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

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 $\mathbf{y}^{(k)}$

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(2)}} = -2(\mathbf{y}^{(k)} - \mathbf{a}^{(3)}) * \mathbf{a}^{(3)} * (1 - \mathbf{a}^{(3)})$$

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(l+1)}} = \text{diag}[\mathbf{a}^{(l+1)} * (1 - \mathbf{a}^{(l+1)})] \cdot \mathbf{W}^{(l+1)} \frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(l+1)}}$$

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial J(\mathbf{W}^{(l)})}{\partial \mathbf{z}^{(l)}} \cdot \mathbf{a}^{(l)}$$

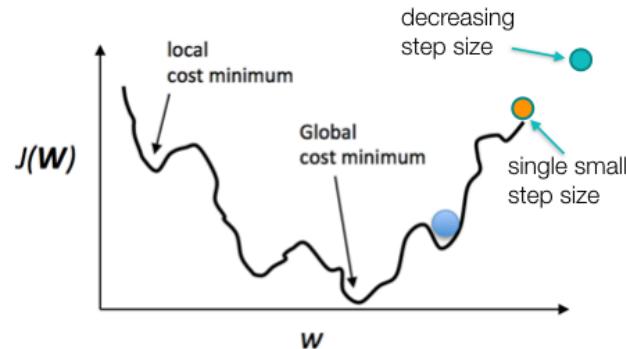
43

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



55

Practical Implementation of Architectures

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

$$J(\mathbf{W}) = -[\mathbf{y}^{(i)} \ln \mathbf{a}^{(L)} + (1 - \mathbf{y}^{(i)}) \ln(1 - \mathbf{a}^{(L)})] \quad \text{speeds up initial training}$$

$$\frac{\partial J(\mathbf{W})}{\mathbf{z}^{(L)}} = (\mathbf{a}^{(L+1)} - \mathbf{y}^{(i)}) \quad \# \text{vectorized backpropagation}$$

```
sigma3 = (A3-Y_enc) # <- this is only line
sigma2 = (W2.T @ sigma3)*A2*(1-A2)
```

$$\frac{\partial J(\mathbf{W})}{\mathbf{z}^{(2)}} = (\mathbf{a}^{(3)} - \mathbf{y}^{(i)}) \quad \# \text{vectorized backpropagation}$$

```
grad1 = sigma2[1:,:] @ A1
grad2 = sigma3 @ A2.T
```

new update

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(2)}} = -2(\mathbf{y}^{(k)} - \mathbf{a}^{(3)}) * \mathbf{a}^{(3)} * (1 - \mathbf{a}^{(3)}) \quad \text{old update}$$

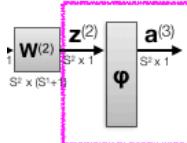
```
# vectorized backpropagation
sigma3 = -2*(Y_enc-A3)*A3*(1-A3)
sigma2 = (W2.T @ sigma3)*A2*(1-A2)

grad1 = sigma2[1:,:] @ A1
grad2 = sigma3 @ A2.T
```

bp-5
63

Practical Implementation of Architectures

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



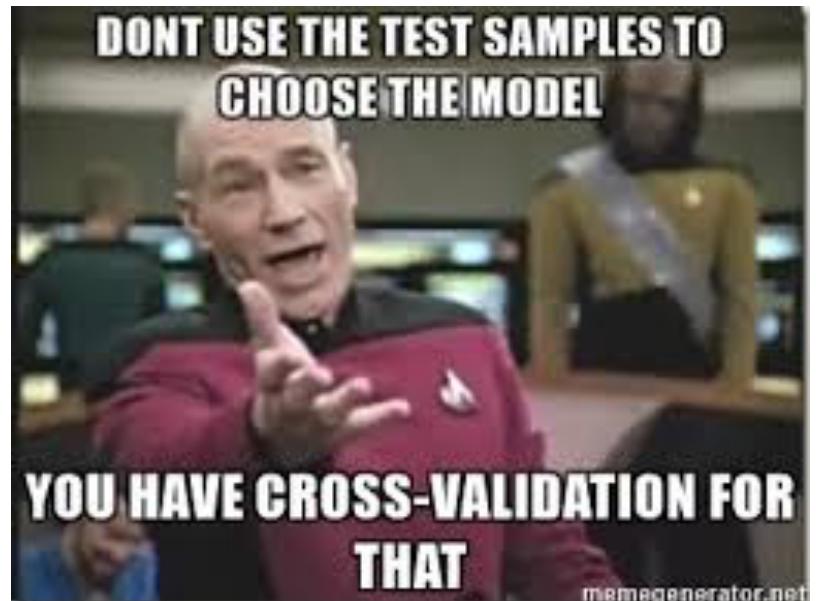
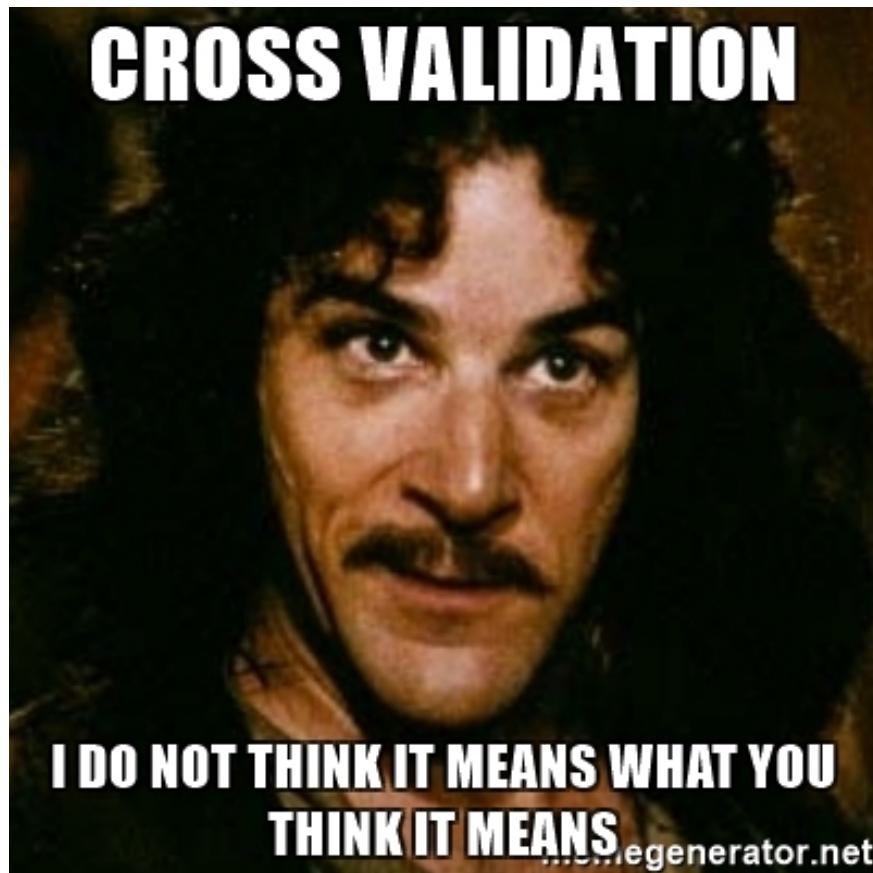
$$\phi(\mathbf{z}^{(i)}) = \begin{cases} \mathbf{z}^{(i)}, & \text{if } \mathbf{z}^{(i)} > 0 \\ 0, & \text{else} \end{cases}$$

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

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

73

Revisiting Cross Validation



Grid Searching

Trying to find the best parameters

LR: C1=[1, 10, 100] C2=[1e3, 1e4, 1e5]

C1			
C2	(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...

C2

C1

(1, 1e3)	(10, 1e3)	(100, 1e3)																																																						
<table border="1"><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>B</td><td>C</td><td>A</td></tr><tr><td>B</td><td>C</td><td>A</td></tr></table>	A	B	C	A	B	C	A	C	B	A	C	B	B	C	A	B	C	A	<table border="1"><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>B</td><td>C</td><td>A</td></tr><tr><td>B</td><td>C</td><td>A</td></tr></table>	A	B	C	A	B	C	A	C	B	A	C	B	B	C	A	B	C	A	<table border="1"><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>B</td><td>C</td><td>A</td></tr><tr><td>B</td><td>C</td><td>A</td></tr></table>	A	B	C	A	B	C	A	C	B	A	C	B	B	C	A	B	C	A
A	B	C																																																						
A	B	C																																																						
A	C	B																																																						
A	C	B																																																						
B	C	A																																																						
B	C	A																																																						
A	B	C																																																						
A	B	C																																																						
A	C	B																																																						
A	C	B																																																						
B	C	A																																																						
B	C	A																																																						
A	B	C																																																						
A	B	C																																																						
A	C	B																																																						
A	C	B																																																						
B	C	A																																																						
B	C	A																																																						
(1, 1e4)	(10, 1e4)	(100, 1e4)																																																						
<table border="1"><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>B</td><td>C</td><td>A</td></tr><tr><td>B</td><td>C</td><td>A</td></tr></table>	A	B	C	A	B	C	A	C	B	A	C	B	B	C	A	B	C	A	<table border="1"><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>B</td><td>C</td><td>A</td></tr><tr><td>B</td><td>C</td><td>A</td></tr></table>	A	B	C	A	B	C	A	C	B	A	C	B	B	C	A	B	C	A	<table border="1"><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>B</td><td>C</td><td>A</td></tr><tr><td>B</td><td>C</td><td>A</td></tr></table>	A	B	C	A	B	C	A	C	B	A	C	B	B	C	A	B	C	A
A	B	C																																																						
A	B	C																																																						
A	C	B																																																						
A	C	B																																																						
B	C	A																																																						
B	C	A																																																						
A	B	C																																																						
A	B	C																																																						
A	C	B																																																						
A	C	B																																																						
B	C	A																																																						
B	C	A																																																						
A	B	C																																																						
A	B	C																																																						
A	C	B																																																						
A	C	B																																																						
B	C	A																																																						
B	C	A																																																						
(1, 1e5)	(10, 1e5)	(100, 1e5)																																																						
<table border="1"><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>B</td><td>C</td><td>A</td></tr><tr><td>B</td><td>C</td><td>A</td></tr></table>	A	B	C	A	B	C	A	C	B	A	C	B	B	C	A	B	C	A	<table border="1"><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>B</td><td>C</td><td>A</td></tr><tr><td>B</td><td>C</td><td>A</td></tr></table>	A	B	C	A	B	C	A	C	B	A	C	B	B	C	A	B	C	A	<table border="1"><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>A</td><td>C</td><td>B</td></tr><tr><td>B</td><td>C</td><td>A</td></tr><tr><td>B</td><td>C</td><td>A</td></tr></table>	A	B	C	A	B	C	A	C	B	A	C	B	B	C	A	B	C	A
A	B	C																																																						
A	B	C																																																						
A	C	B																																																						
A	C	B																																																						
B	C	A																																																						
B	C	A																																																						
A	B	C																																																						
A	B	C																																																						
A	C	B																																																						
A	C	B																																																						
B	C	A																																																						
B	C	A																																																						
A	B	C																																																						
A	B	C																																																						
A	C	B																																																						
A	C	B																																																						
B	C	A																																																						
B	C	A																																																						

Grid Searching

Could perform iteratively

C1

(1, 1e3)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

(10, 1e3)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

(100, 1e3)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

C2

(1, 1e4)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

(10, 1e4)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

(100, 1e4)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

(1, 1e5)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

(10, 1e5)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

(100, 1e5)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

Grid Searching

or at random...

gamma

C

(1, 1e3)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

(10, 1e3)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

(100, 1e3)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

(1, 1e4)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

(10, 1e4)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

(100, 1e4)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

(1, 1e5)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

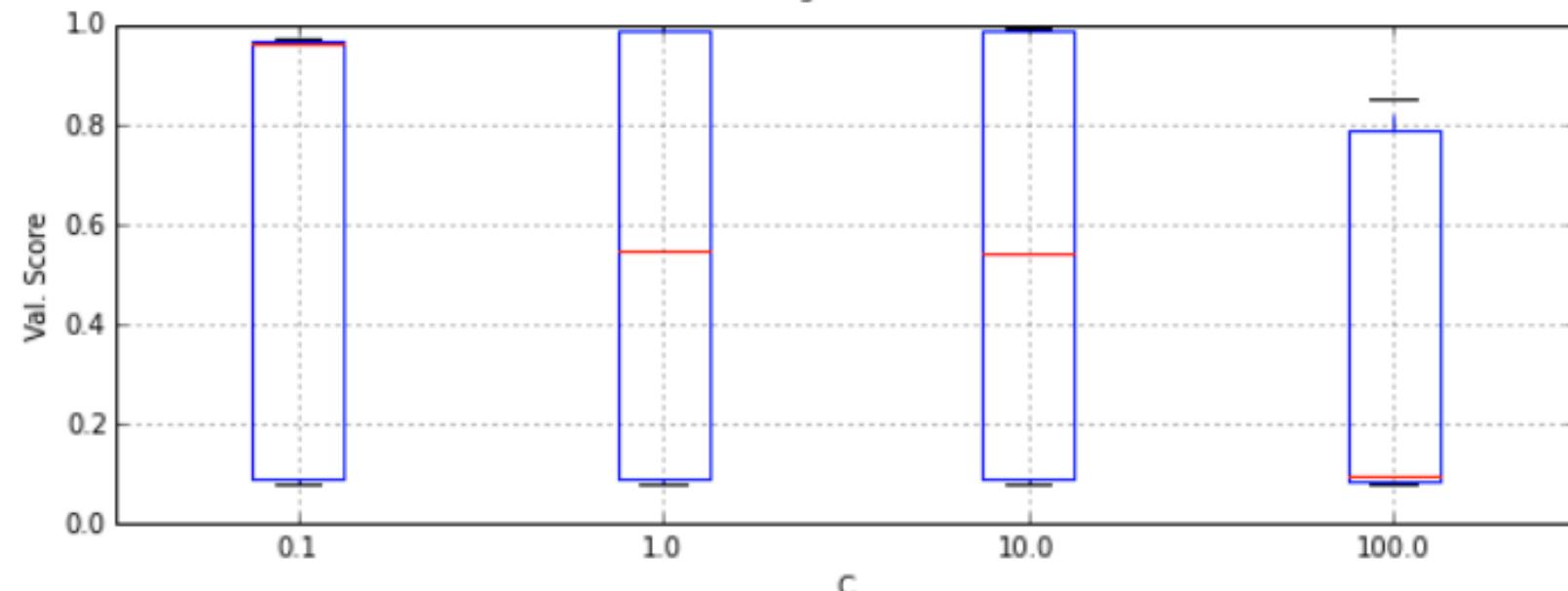
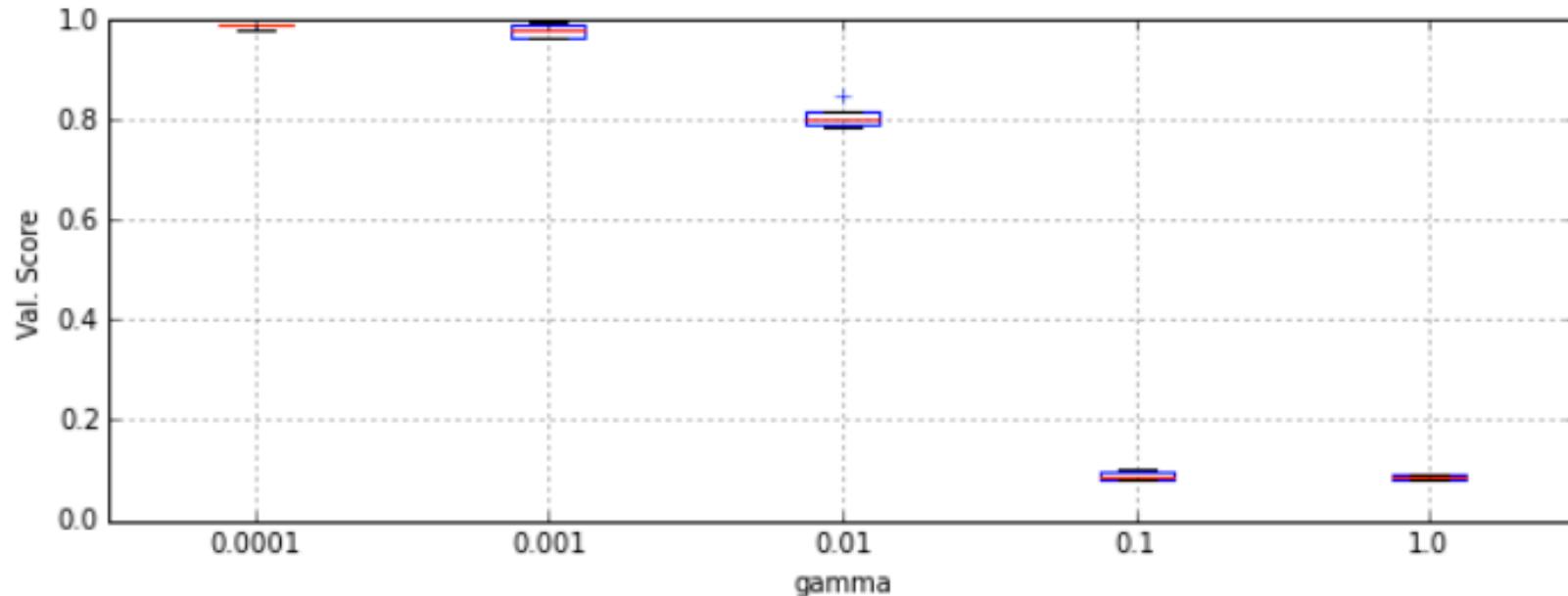
(10, 1e5)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

(100, 1e5)

A	B	C
A	B	C
A	C	B
A	C	B
B	C	A
B	C	A

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

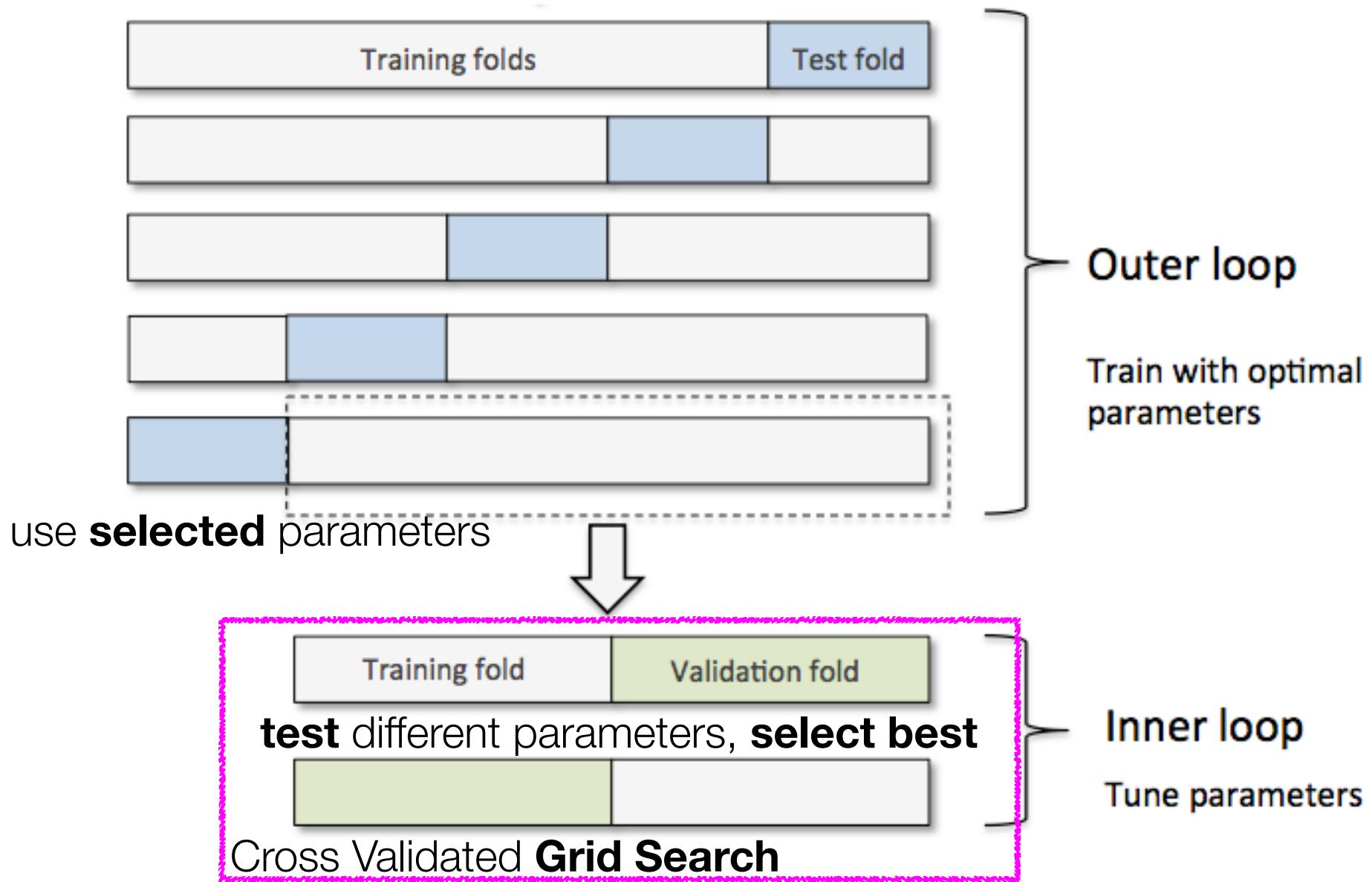


The Problem

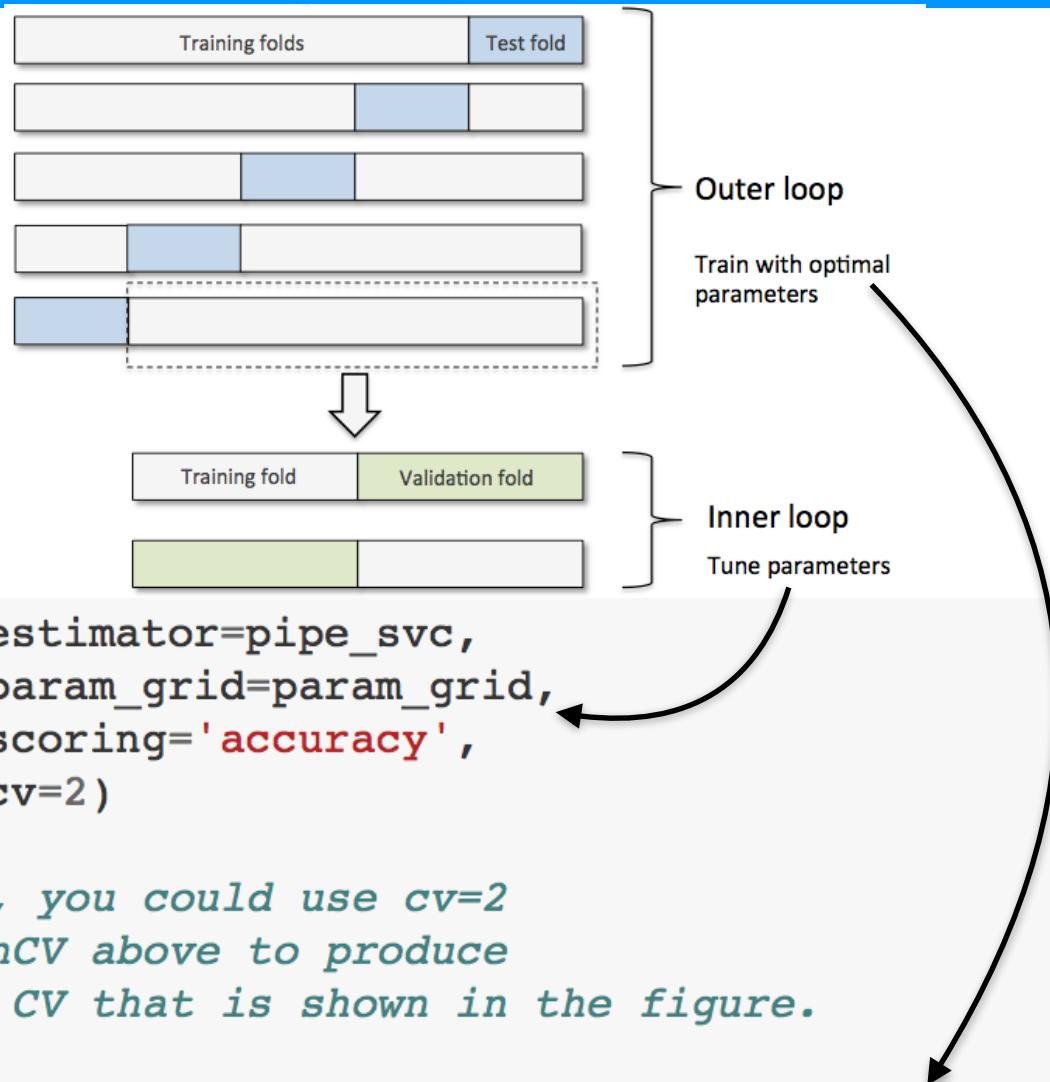
- Using the grid search parameters and testing on the same set...
 - the **performance on the dataset** could now be **biased**
 - cannot determine the **expected performance** on **new data**
 - this is **data snooping**



Solution: Nested Cross Validation



Nested Cross Validation: Hyper-parameters



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

McNemar Testing

A final statistical test, Null hypothesis: these models are the same!

	Model 2 correct	Model 2 wrong
Model 1 correct	A	B
Model 1 wrong	C	D

One caveat: Statistical power depends upon $B+C$, which might be small, even with lots of test data.

McNemar and Edwards, 1948

$$\chi^2 = \frac{(|B - C| - 1)^2}{B + C}$$

χ squared statistic, one DOF

Steps:

1. Compare each model's performance on the same test data (2x2 matrix)
2. Calculate χ statistic
3. Look up p -value associated with χ statistic
4. Can you reject the null hypothesis that the models are the same ($p < 0.05$)?

McNemar Example

Model 1	Model 2	Label	Matrix
T-shirt	T-shirt	T-shirt	A
Sneaker	T-shirt	Sneaker	B
T-shirt	Pullover	Pullover	C
Sneaker	Sneaker	Sneaker	A
T-shirt	Sneaker	Sneaker	C
Pullover	Pullover	T-shirt	D
Pullover	T-shirt	Pullover	B
Sneaker	Sneaker	Sneaker	A
Sneaker	Sneaker	Sneaker	A

Model 2			
		correct	wrong
Model 1	correct	4 A	2 B
	wrong	2 C	1 D

McNemar and Edwards, 1948

$$\chi^2 = \frac{(|B - C| - 1)^2}{B + C}$$

$$\chi^2 = \frac{(|2 - 2| - 1)^2}{2 + 2} = 0.25$$

Confidence	0.90	0.95	0.99
1 DOF, Critical Value	2.706	3.841	6.635

<https://www.itl.nist.gov/div898/handbook/eda/section3/eda3674.htm>

Since $0.25 < 3.841$, we cannot reject the null hypothesis.
This means **we cannot say the models are different** based on the evidence.

Town Hall



Some History of Deep Learning

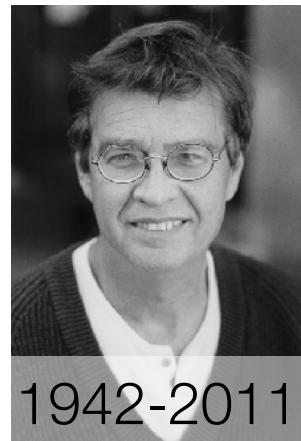
When you move on to
Deep Learning



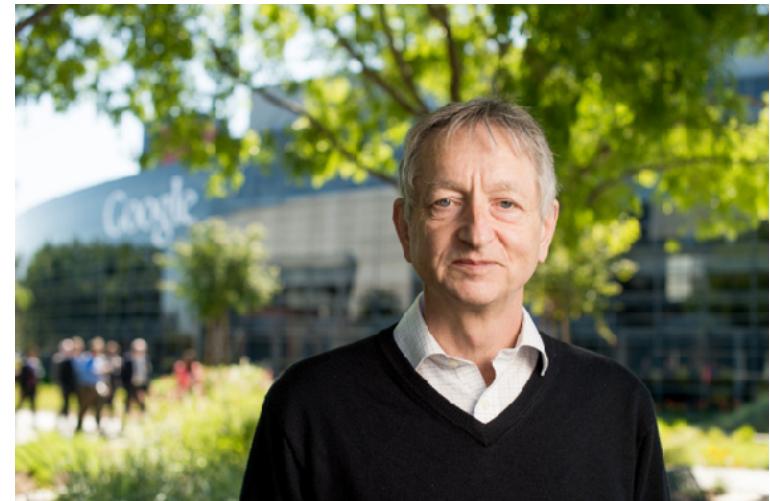
Neural Networks: Where we left it

- 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



Geoffrey Hinton



Machine Learning Timeline (Neural Nets)

- Up to this point: back propagation saved AI winter
- 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



1949, Hebb's Law
Close neuron fire together



1960, Widrow & Hoff
Adaline Network



1986, Rumelhart & Hinton
Back-propagation



2003, Vapnik
Kernel SVMs

BRACE YOURSELF



AI WINTER IS
COMING

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

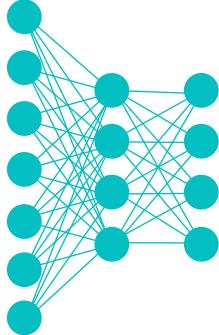
Read this: <http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/>



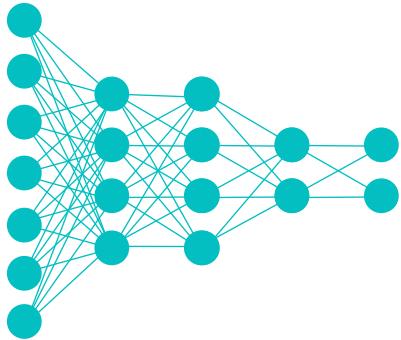
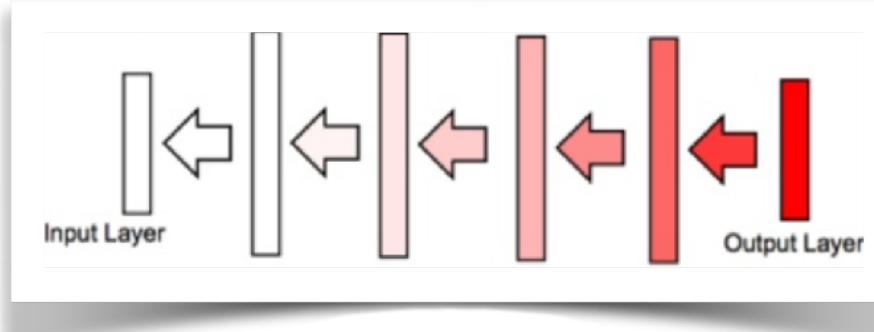


History of Deep Learning: Winter

- AI Winter is coming:



Easy to train, performs on par with other methods



Hard to train, performs worse than other methods

Researchers have difficulty reconciling expressiveness with performance

~chance (untrainable)

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

Machine Learning Timeline (Neural Nets)

- 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



1949, Hebb's Law
Close neuron fire together



1960, Widrow & Hoff
Adaline Network



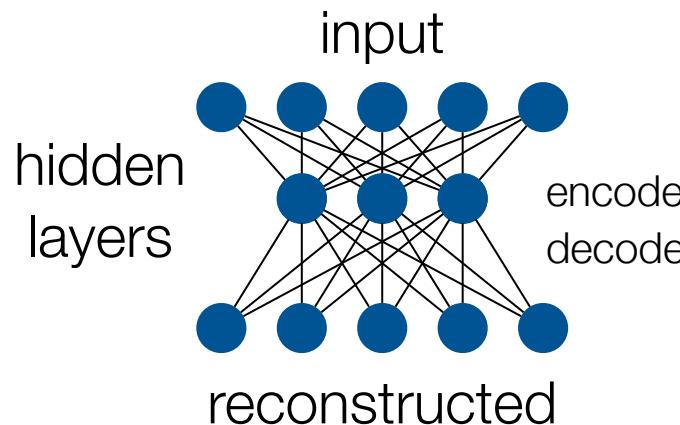
1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs

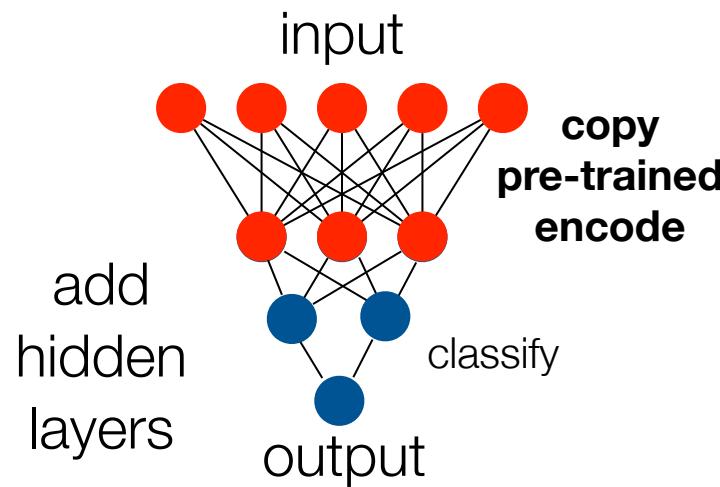
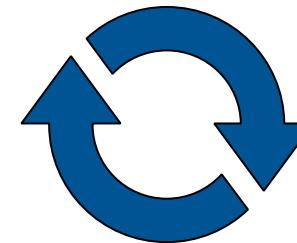


Pre-training: still in the long winter

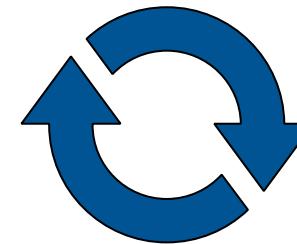
- auto-encoding (a form of pre-training)



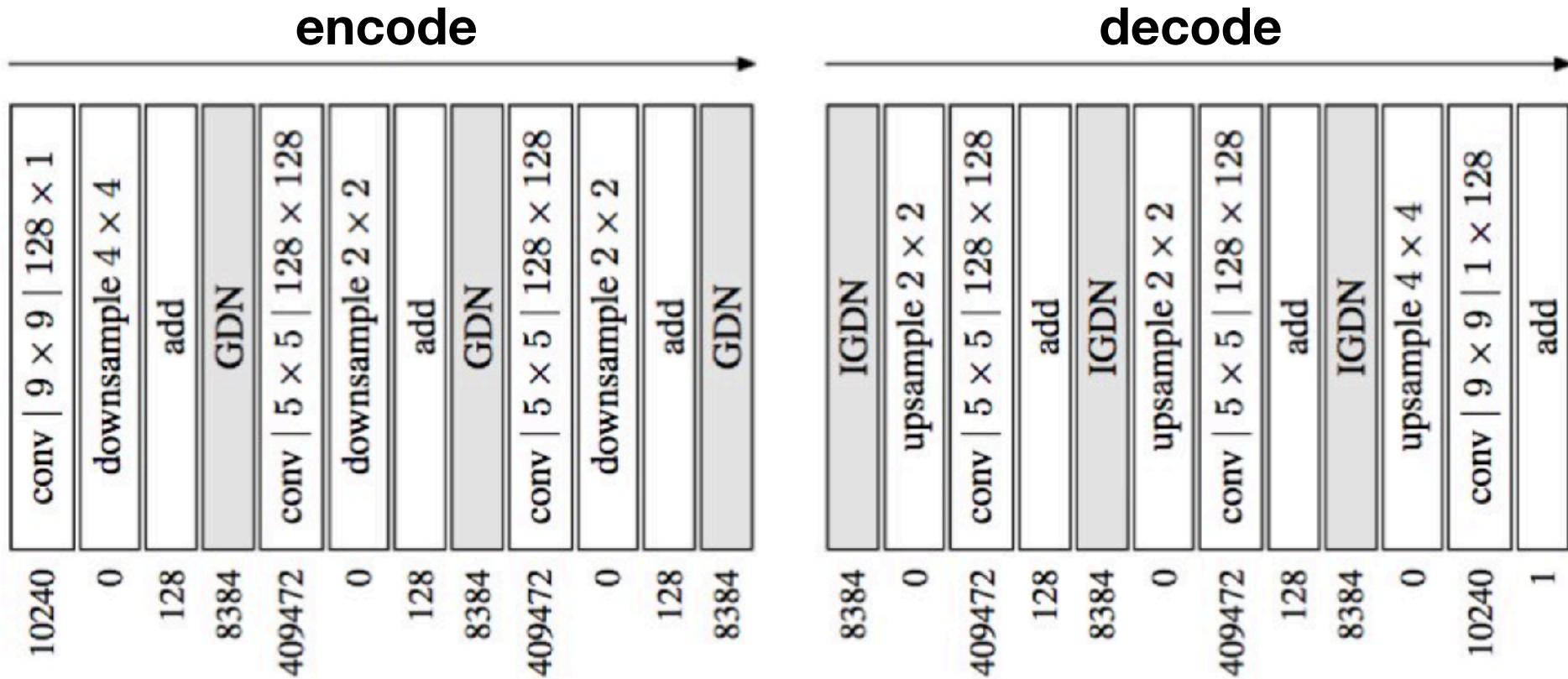
train with lots of
unlabeled data



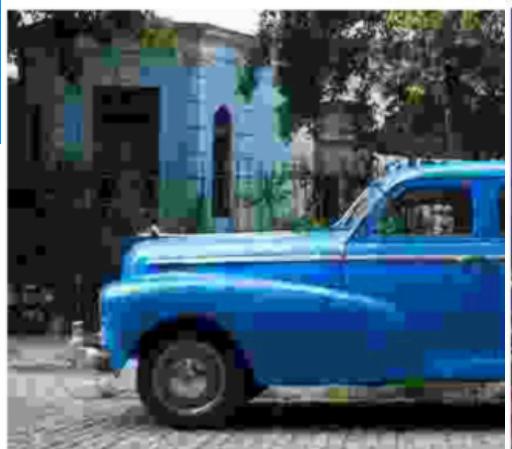
train with
labeled data



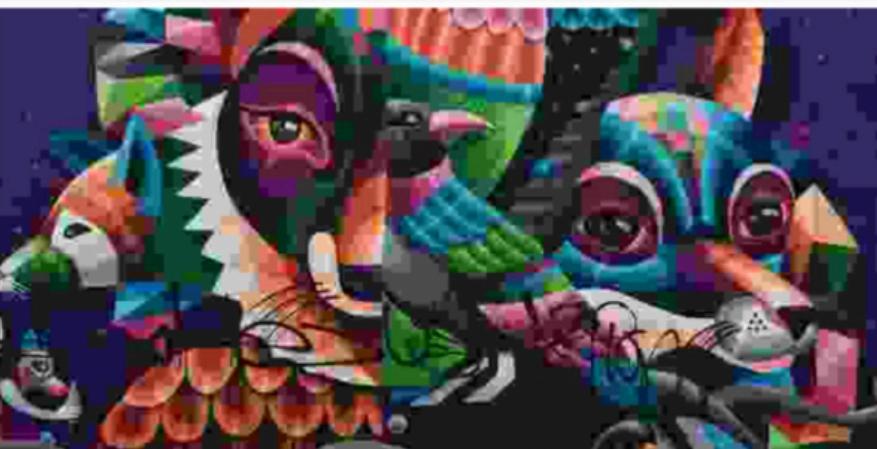
Pre-training: modern example



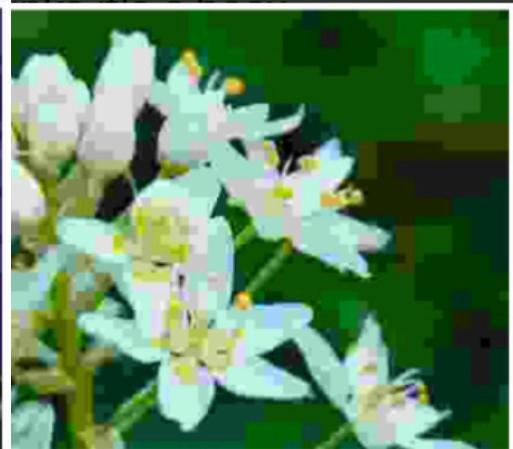
<https://arxiv.org/abs/1611.01704>



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



bit/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

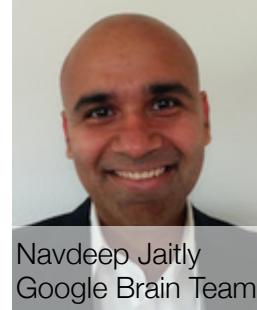


xx), RMSE: 8.56/5.71, PSNR: 29.49 dB/32.99 dB

Still in the Long Winter

- 2009: Hinton's lab starts using GPUs, Also Andrew Ng
 - 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



George Dahl
Google Brain Team

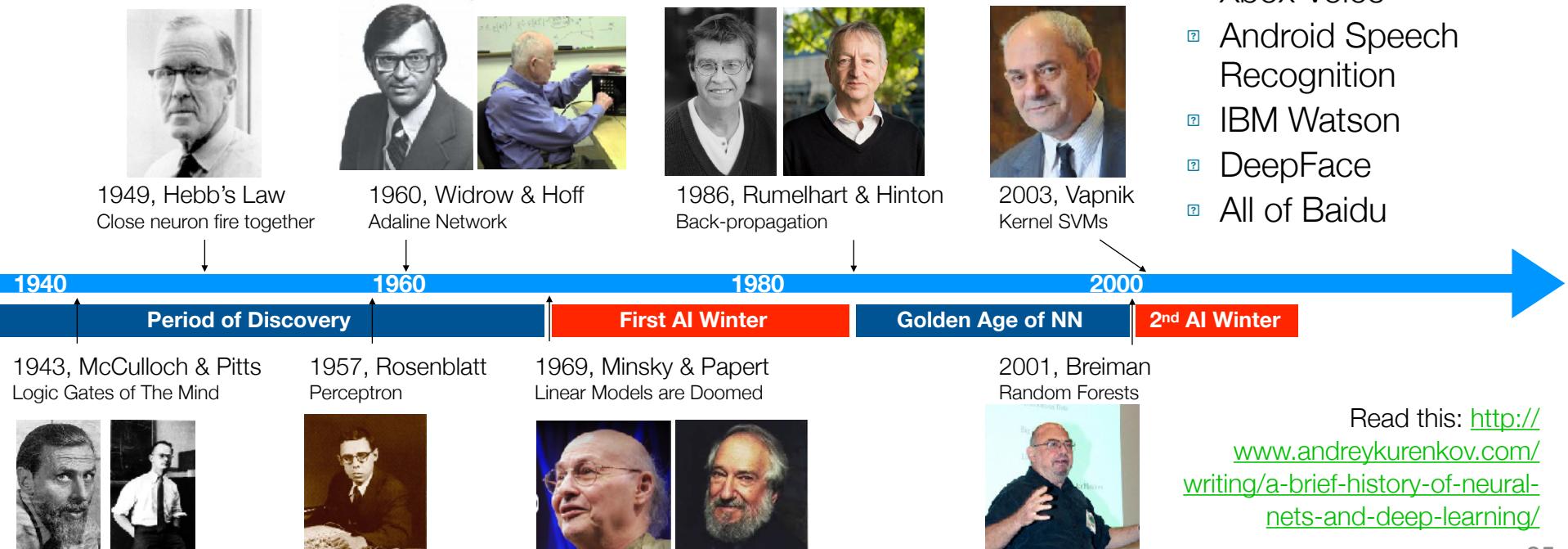
Abdel-rahman Mohamed

Microsoft Research
Redmond, Washington | Computer Software



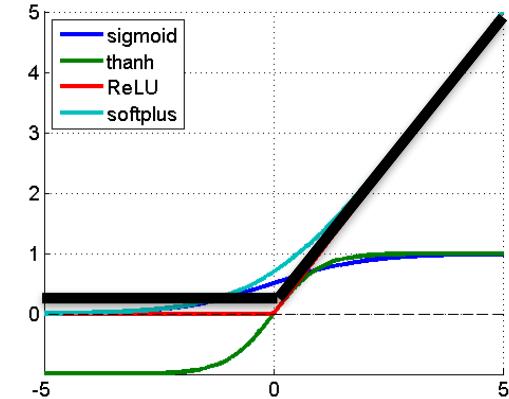
Current Microsoft
Previous University of Toronto, IBM, Microsoft
Education University of Toronto

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



Still in the long Winter

- 2011: Glorot and 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: $f(x) = \max(0, x)$
 $f'(x) = 1 \text{ if } x > 0 \text{ else } 0$



1949, Hebb's Law
Close neuron fire together



1960, Widrow & Hoff
Adaline Network



1986, Rumelhart & Hinton
Back-propagation



2003, Vapnik
Kernel SVMs



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

End of Session

- Next Time:
 - How we got out of the AI winter
 - Introduction to TensorFlow
 - Wide and Deep Networks

Lecture Notes for **Machine Learning in Python**

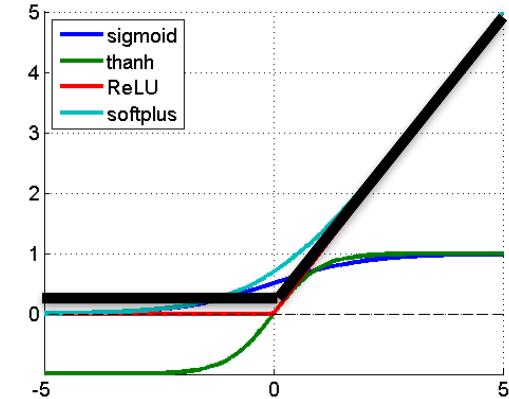
Professor Eric Larson
Wide and Deep Networks

Lecture Agenda

- Logistics: CS 8321 in Spring
 - Grading and lab deadlines
- Get out of the long winter...
- Introduction to TensorFlow
 - Tensors, Namespaces, Numerical methods
 - Deep APIs
- Wide and Deep Networks

Still in the long Winter

- 2011: Glorot and 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: $f(x) = \max(0, x)$
 $f'(x) = 1 \text{ if } x > 0 \text{ else } 0$



1949, Hebb's Law
Close neuron fire together



1960, Widrow & Hoff
Adaline Network



1986, Rumelhart & Hinton
Back-propagation



2003, Vapnik
Kernel SVMs



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

Still in the long Winter

- ReLU not the only way to do it!
 - Sigmoid Weighted Linear Unit
SiLU
 - also called Swish
 - Mixing of sigmoid, σ , and ReLU

Ramachandran P, Zoph B, Le QV.
Swish: a Self-Gated Activation
Function. arXiv preprint
arXiv:1710.05941. 2017 Oct 16

Elfwing, Stefan, Eiji Uchibe, and Kenji
Doya. "Sigmoid-weighted linear units
for neural network function
approximation in reinforcement
learning." Neural Networks (2018).

$$\frac{\partial \varphi(z)}{\partial z} = \varphi(z) + \sigma(z)[1 - \varphi(z)]$$
$$= a^{(l+1)} + \sigma(z^{(l)}) \cdot [1 - a^{(l+1)}]$$

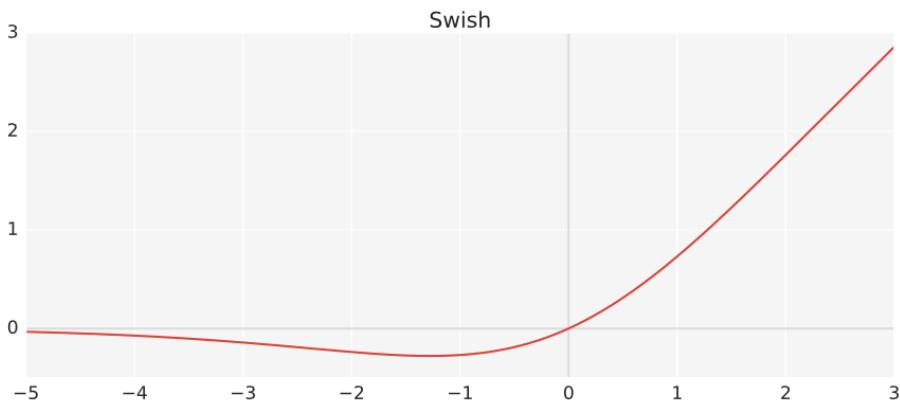


Figure 1: The Swish activation function.

Derivative Calculation:

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

Machine Learning Timeline (1940-2020)

- **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

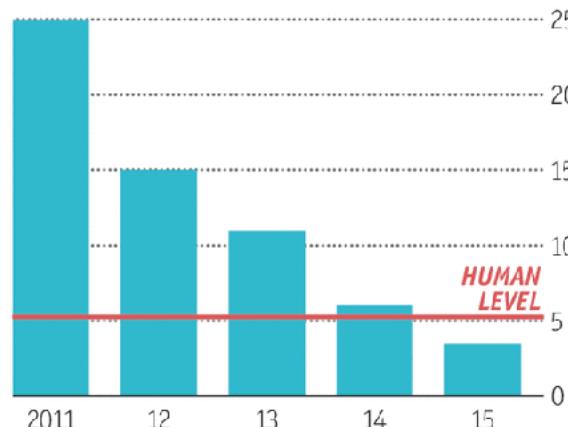


Fei Fei Li
Director of Stanford's
AI Lab (Former)
HAI Founder

"I have had a hand in last few years so I must say as she comes along pacifying people happens from time to time a colleague of mine

Ever cleverer

Error rates on ImageNet Visual Recognition Challenge, %



Sources: ImageNet; Stanford Vision Lab

Economist.com



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



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

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]

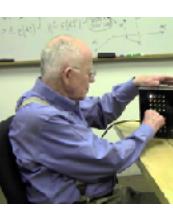
[https://www.cs.toronto.edu/~gdahl/papers/
deepSpeechReviewSPM2012.pdf](https://www.cs.toronto.edu/~gdahl/papers/deepSpeechReviewSPM2012.pdf)



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



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



1943, McCulloch & Pitts
Logic Gates of The Mind

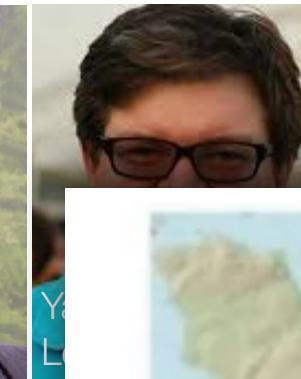


A summary of the Deep Learning people:

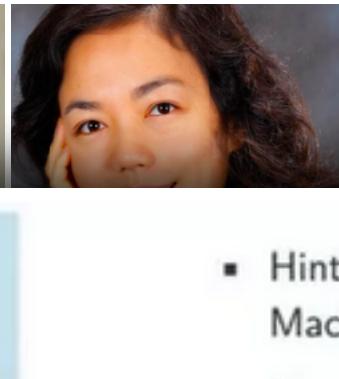


Yoshua
Bengio

Stayed at Univ.
Montreal
Advises IBM

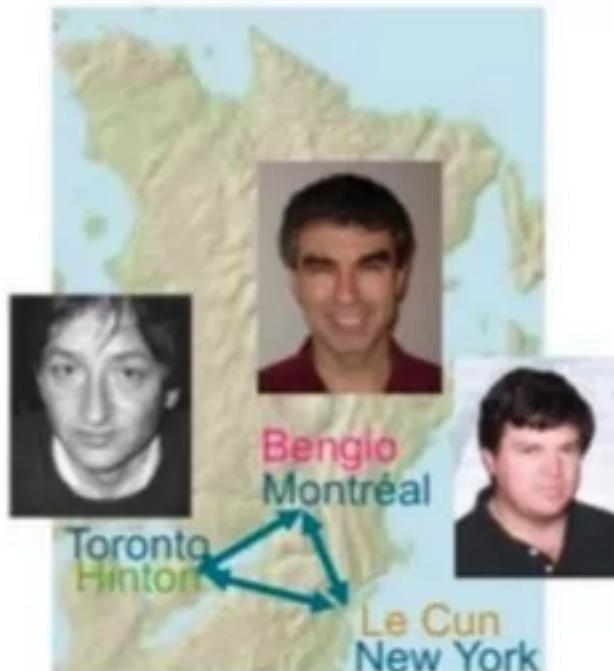


Yann
LeCun



Power
Machine

Read this paper
as it sums up ac
nicely



- Hinton: Restricted Boltzmann Machine, Deep autoencoder
- Bengio: neural language modeling.
- LeCun: Convolutional Neural Network
- NIPS, ICML, CVPR, ACL
- Google Brain, Deep Mind.
- FaceBook AI.

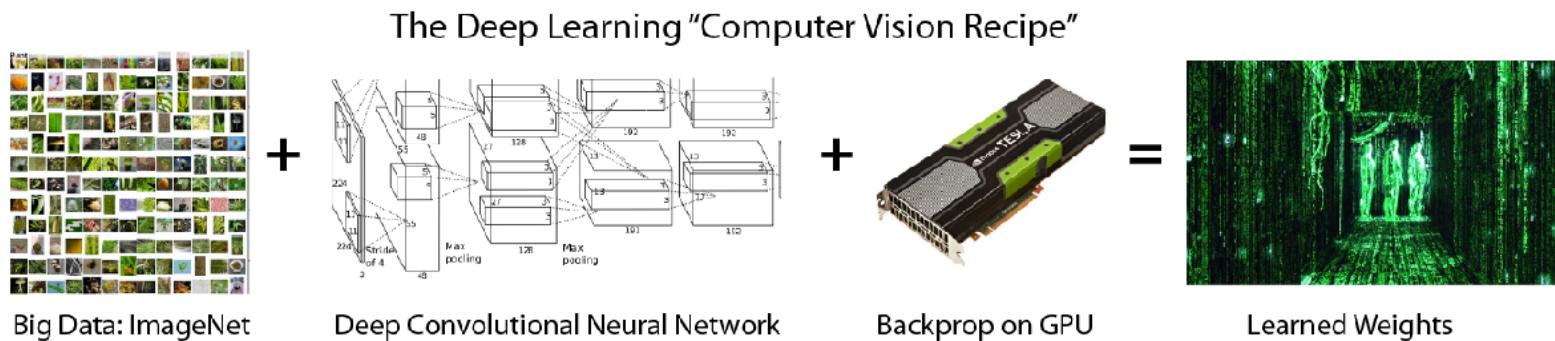
Yann LeCun^{1,2}, Yoshua Bengio³ & Geoffrey Hinton^{4,5}

And predicts the future of
deep learning

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

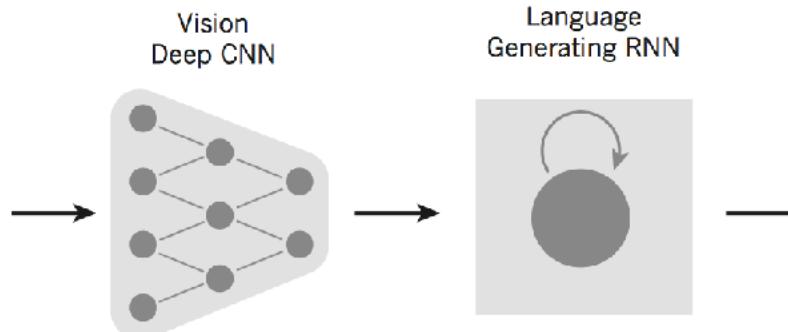
History of Deep Learning

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



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

Famous examples:



A group of people shopping at an outdoor market.

There are many vegetables at the fruit stand.



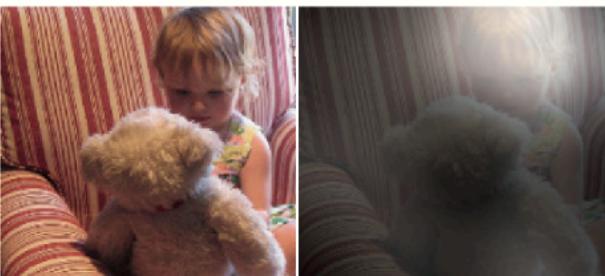
A woman is throwing a **frisbee** in a park.



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.

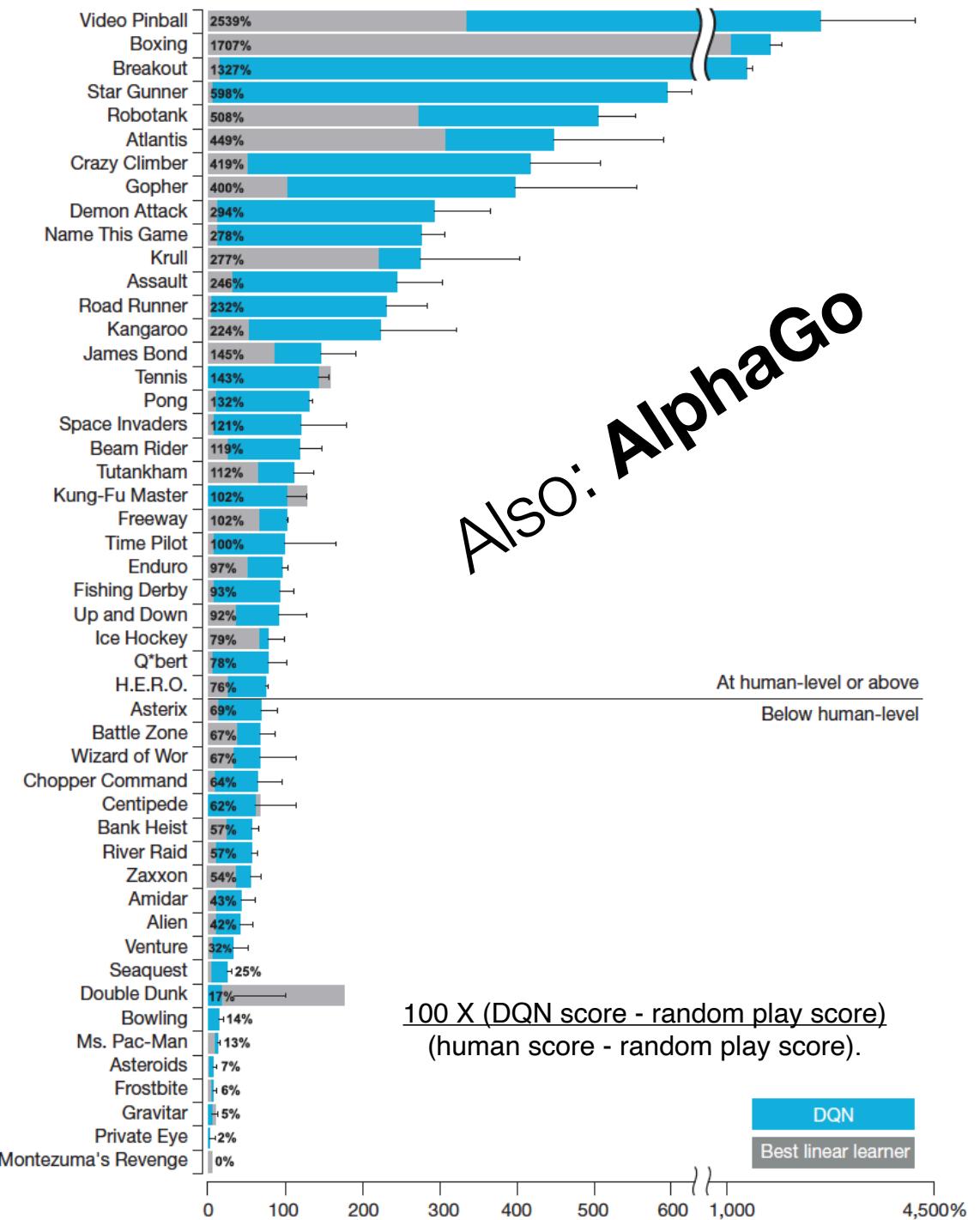
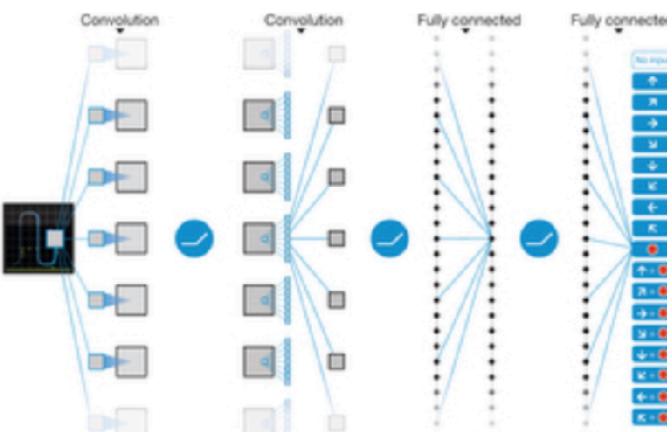


A group of **people** sitting on a boat in the water.



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

More Famous



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



Yoshua Bengio

Geoffrey Hinton

Yann LeCun



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

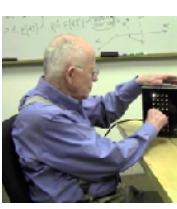
- Up to this point: back propagation saved AI winter for NN (Hinton and others!)
- 80's, 90's, 2000's: convolutional networks for image processing start to get deeper
 - but back propagation no longer does great job at training them
- SVMs and Random Forests gain traction...
 - The second AI winter begins,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



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

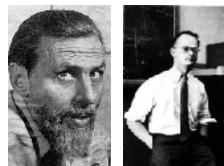
1980

2000

2020

Period of Discovery

1943, McCulloch & Pitts
Logic Gates of The Mind



1957, Rosenblatt
Perceptron



First AI Winter

1969, Minsky & Papert
Linear Models are Doomed



Golden Age of NN

2001, Breiman
Random Forests



2nd AI Winter

2011, Bengio
Init and ReLU



Age of Deep Learning

2015, Google
Tensorflow Open Source



TensorFlow

TensorFlow

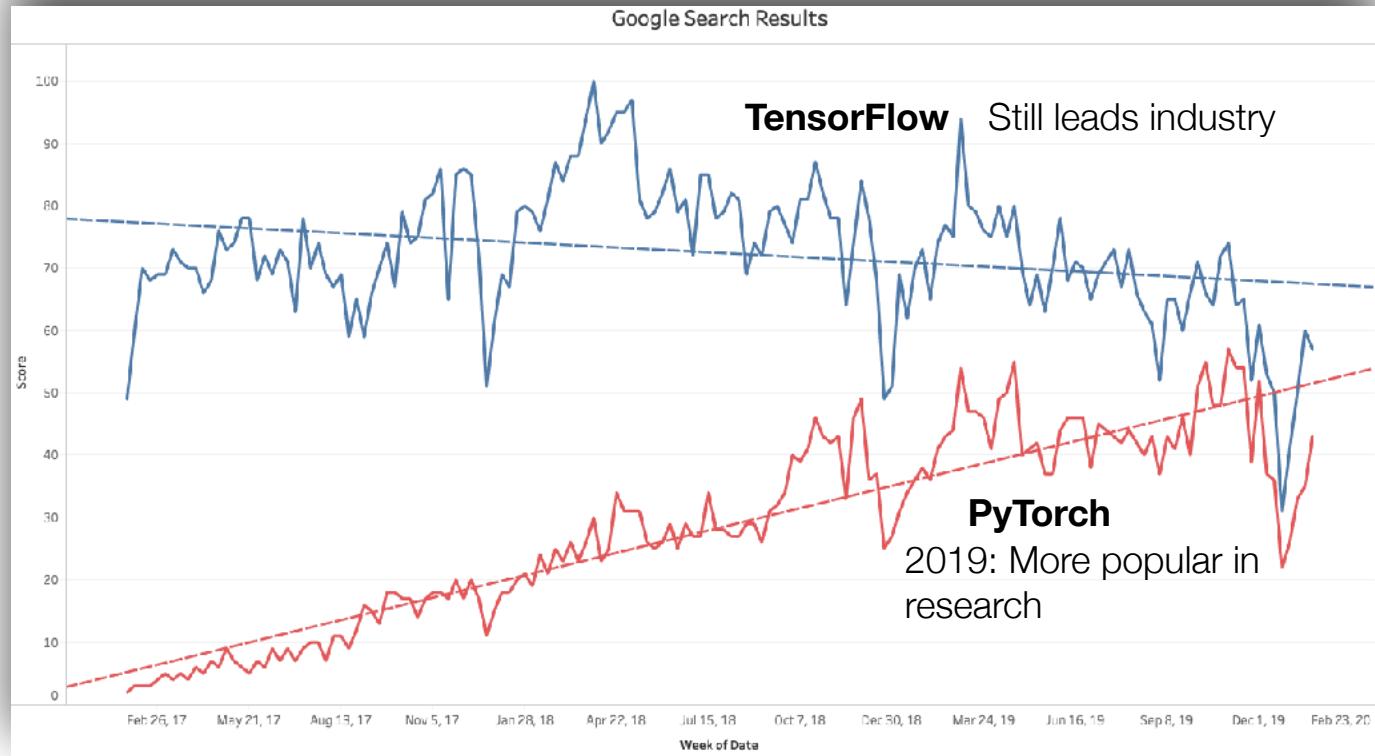
“Further discussion of it merely incumbers the literature and befogs the mind of fellow students.”

- 2007: NIPS program committee rejects a paper on deep learning by *al. et.* Hinton because they already accepted a paper on deep learning and two papers on the same topic would be excessive.
- ~2009: A reviewer tells Yoshua Bengio that papers about neural nets have no place in ICML.
- ~2010: A CVPR reviewer rejects Yann LeCun's paper even though it beats the state-of-the-art. The reviewer says that it tells us nothing about computer vision because everything is learned.



Options for Deep Learning Toolkits

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



Framework	Online Job Listings					KDnuggets Usage Survey	Google Search Volume	Medium Articles	Amazon Books	ArXiv Articles	GitHub Activity			
	Indeed	Monster	Simply Hired	LinkedIn	Angel List						Stars	Watchers	Forks	Contributors
TensorFlow	2,079	1,253	1,582	2,610	552	29.90%	73	6,200	202	3,700	109,576	8,334	67,551	1,642
Keras	684	364	449	695	177	22.20%	53	9,120	79	1,390	33,558	1,847	12,658	719
PyTorch	486	309	428	665	120	6.40%	19	1,780	18	1,560	18,716	952	4,474	760
Caffe	607	399	515	866	123	1.50%	4	815	14	1,360	25,604	2,218	15,633	270
Theano	356	316	279	508	95	4.90%	0	428	17	652	8,477	585	2,447	328
MXNET	266	154	200	298	29	1.50%	2	524	32	260	15,200	1,170	5,498	587
CNTK	126	96	97	160	12	3.00%	0	223	1	88	15,106	1,368	4,029	189
DeepLearning4J	17	5	9	35	3	3.40%	2	70	11	27	9,615	829	4,441	232
Caffe2	55	51	49	109	12	1.20%	2	335	2	67	8,284	577	2,102	193
Chainer	19	19	19	28	3	0.00%	2	91	3	164	4,128	325	1,095	182
FastAI	0	0	0	0	0	0.00%	0	858	0	11	7,268	432	2,647	195

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
 - 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)
- Toolkits Support:**
- tensor creation
 - functions on tensors
 - automatic differentiation

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`

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 creates layers easily, *knows its gradient*
- **Automatic Differentiation** is just **chain rule**
- But... lets start simple...

Tensor function evaluation

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

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

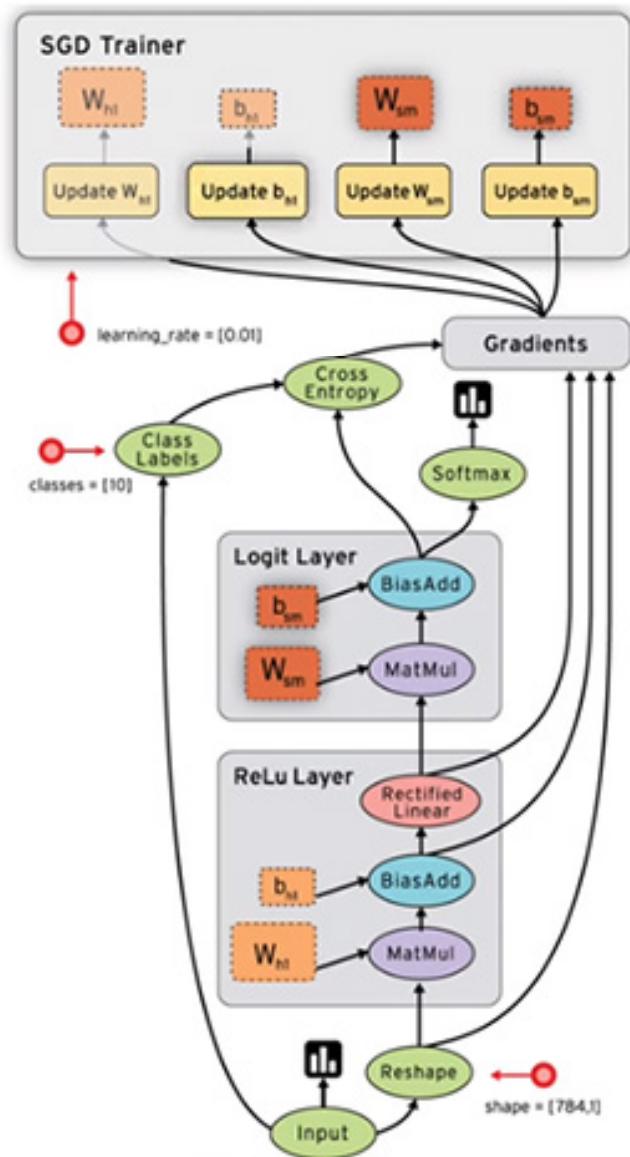
```
c = a * b
```

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

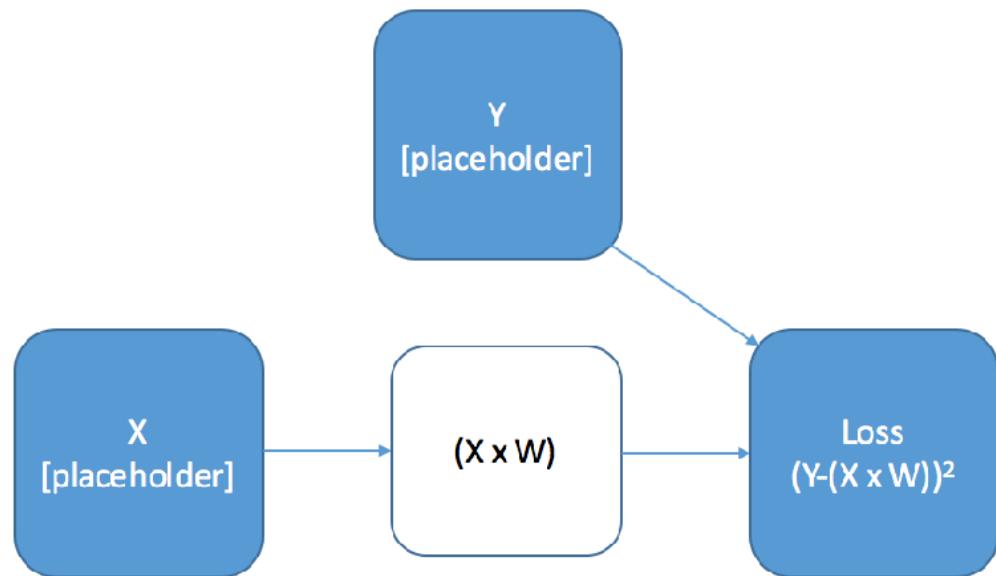
output = 30

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

Computation Graph



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



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

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

Tensorflow with Linear Regression

- Simple Computation Graph

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

```
X = tf.placeholder()
y = tf.placeholder()

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})
```

1. **Setup** Variables

2. Add **optimization** operation to computation graph
Adjusts variables to minimize loss with
automatic differentiation

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

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

TensorFlow Mini-batching

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

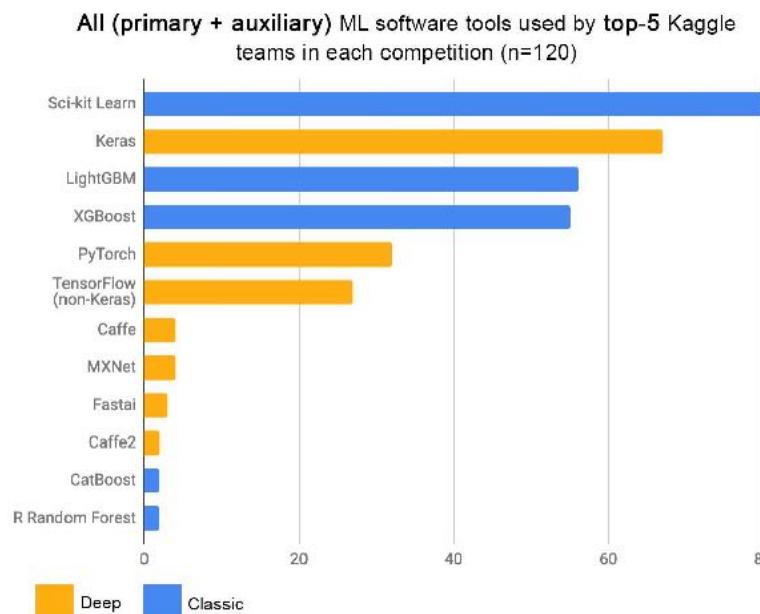
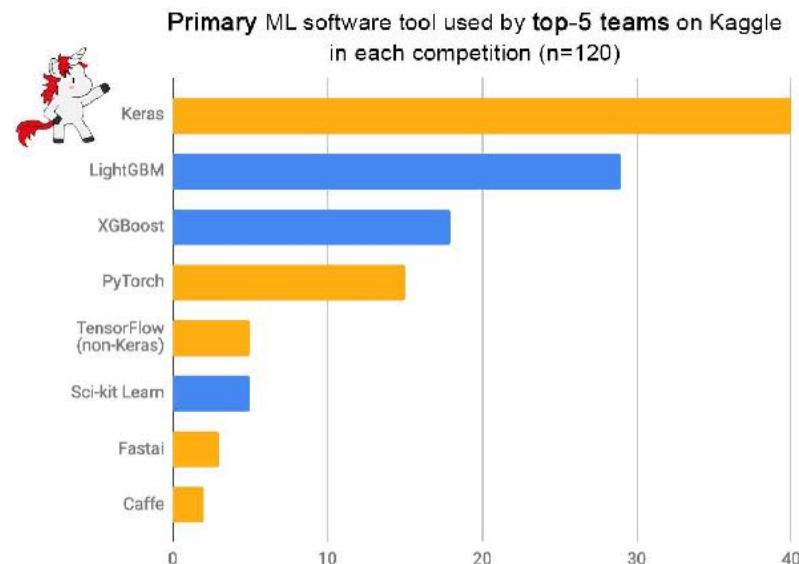
with tf.Session() as sess:
    # Initialize Variables in graph
    sess.run(tf.initialize_all_variables())
    # Gradient descent loop for 500 steps
    for _ in range(500):
        # Select random minibatch
        indices = np.random.choice(n_samples, batch_size)
        X_batch, y_batch = X_data[indices], 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

- **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

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



10. Keras Wide and Deep.ipynb

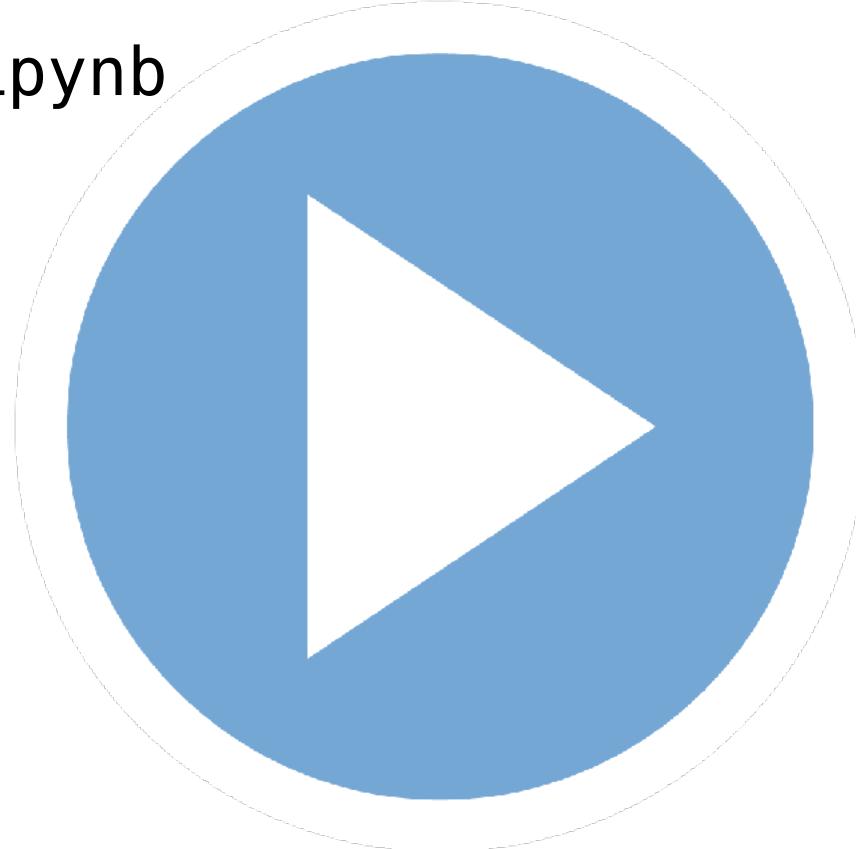
Reinventing the MLP
Wheel

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!!

Lecture Notes for **Machine Learning in Python**

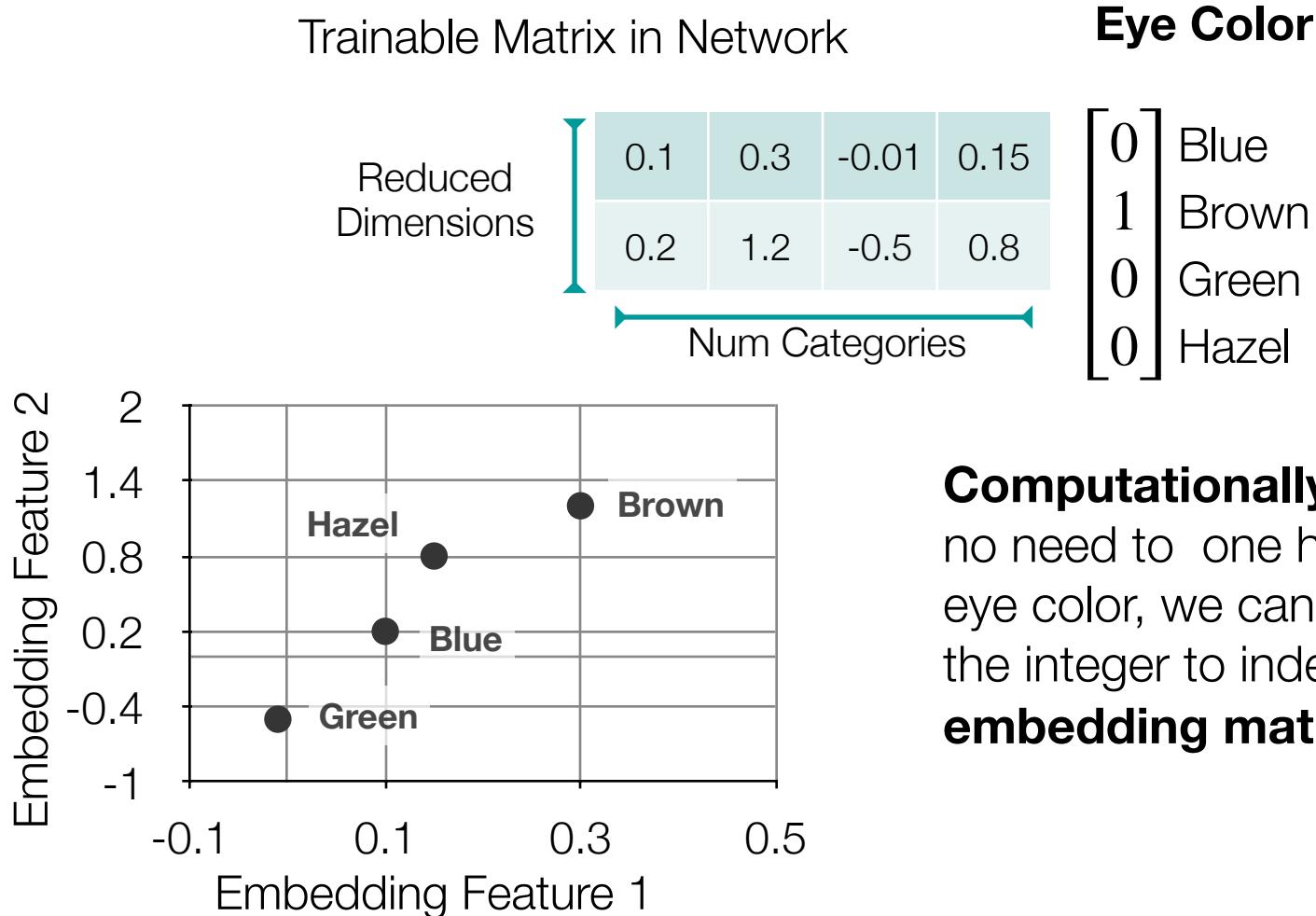
Professor Eric Larson
Wide and Deep Networks

Lecture Agenda

- Finish Keras Demo
- Wide and Deep Networks
- Wide and Deep Town Hall

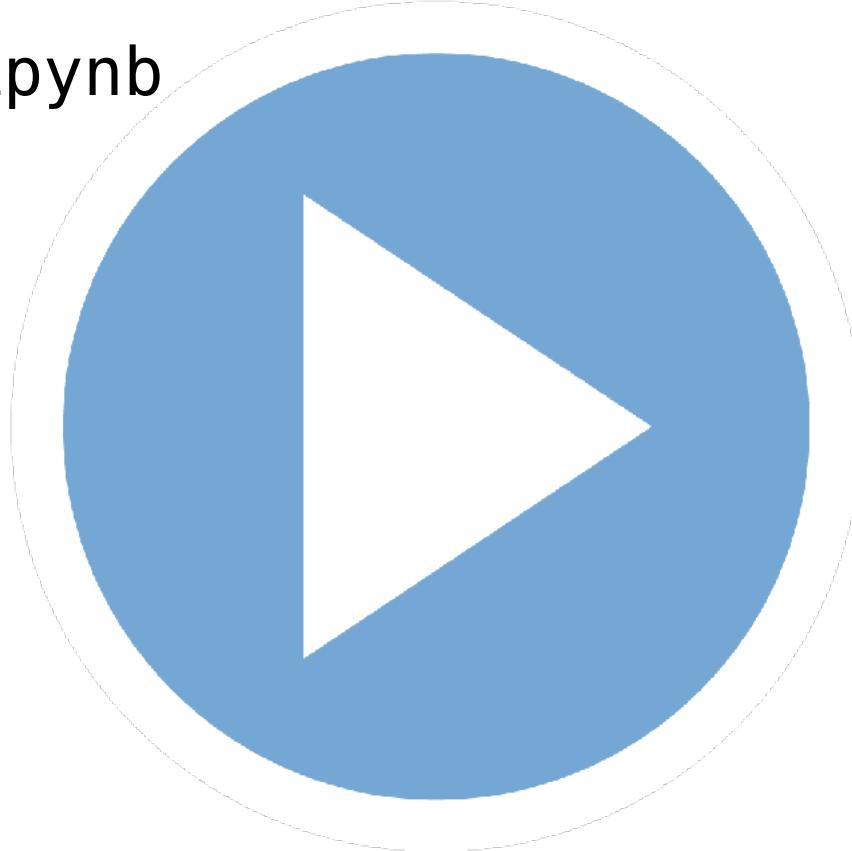
Categorical Feature Embeddings

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



10. Keras Wide and Deep.ipynb

Reinventing the MLP
Wheel



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

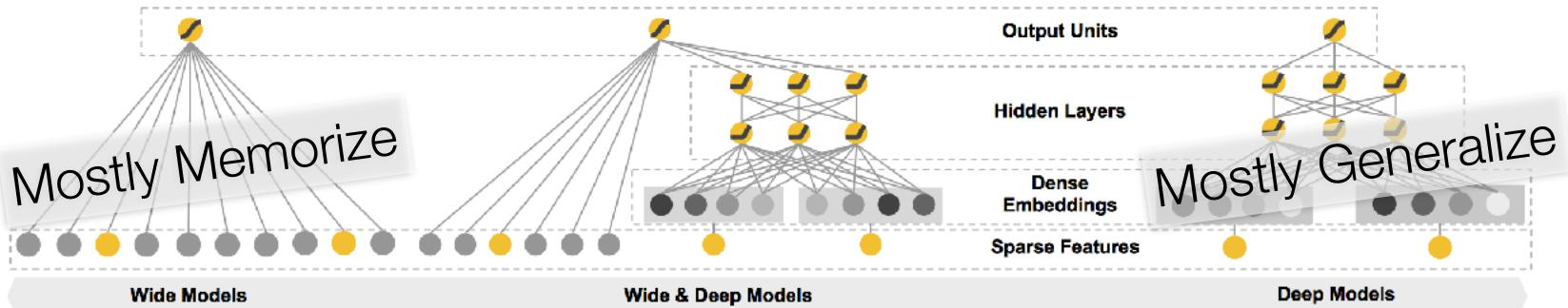
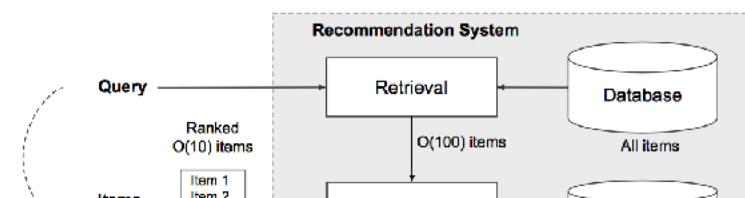


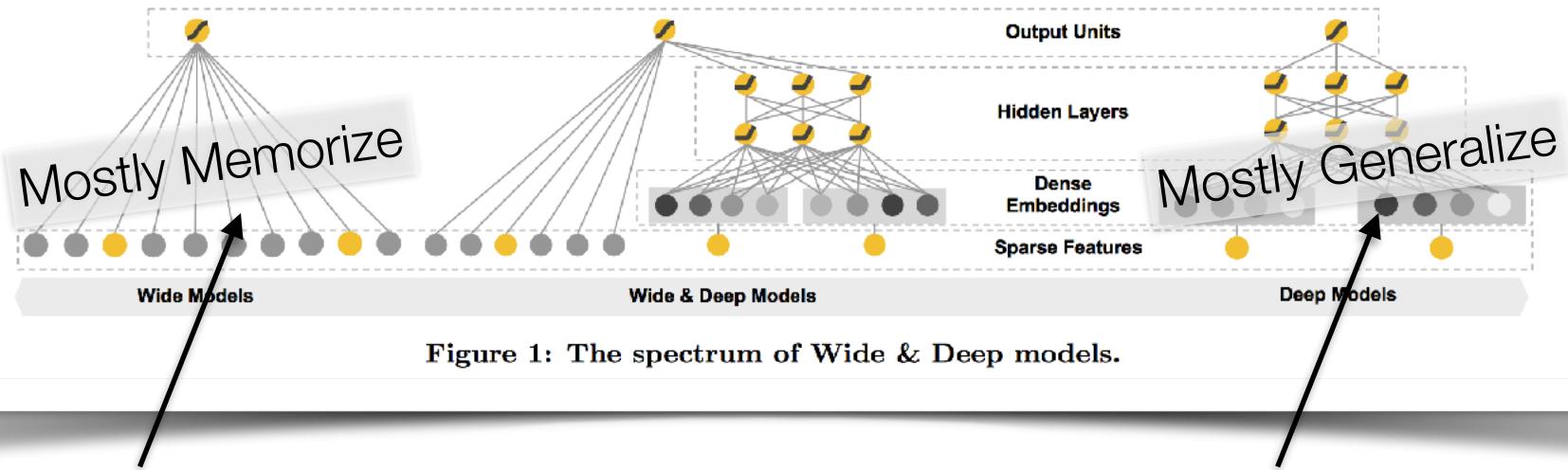
Figure 1: The spectrum of Wide & Deep models.

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



Why wide and deep?



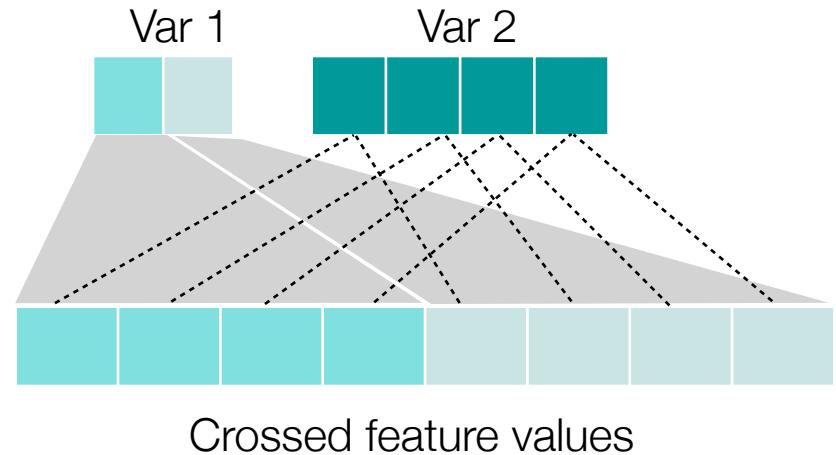
But why memorize?

Obvious!

- Categorical values have combinations that repeat!
 - so memorizing these values is not necessarily a bad strategy
 - let's make memorizing easy on one network

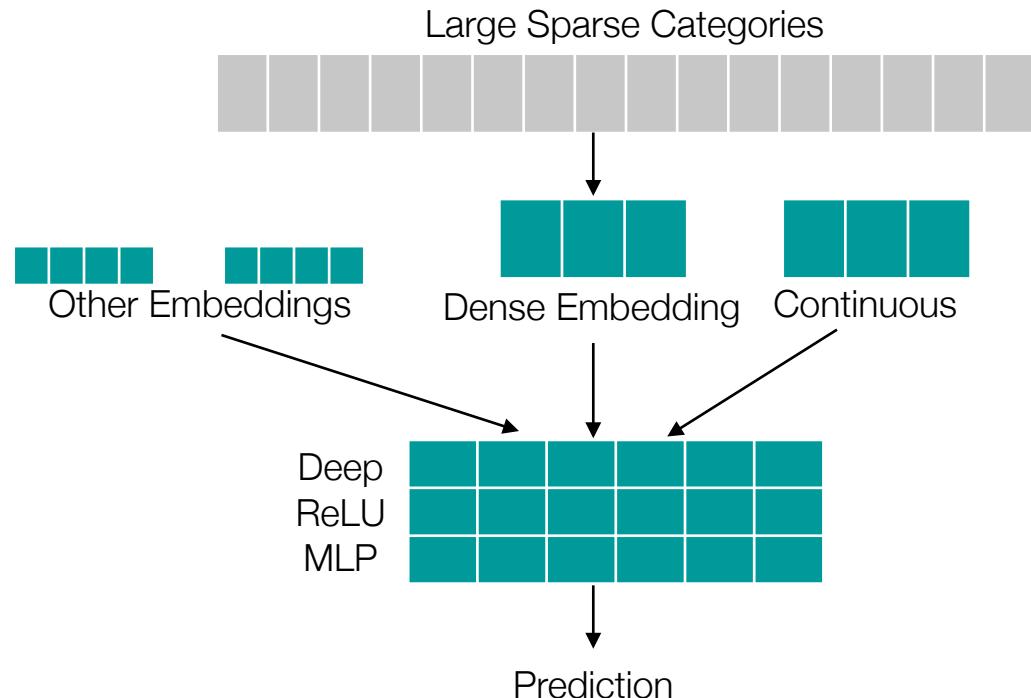
Wide networks (Memorize?)

- Wide refers to the expansion of 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



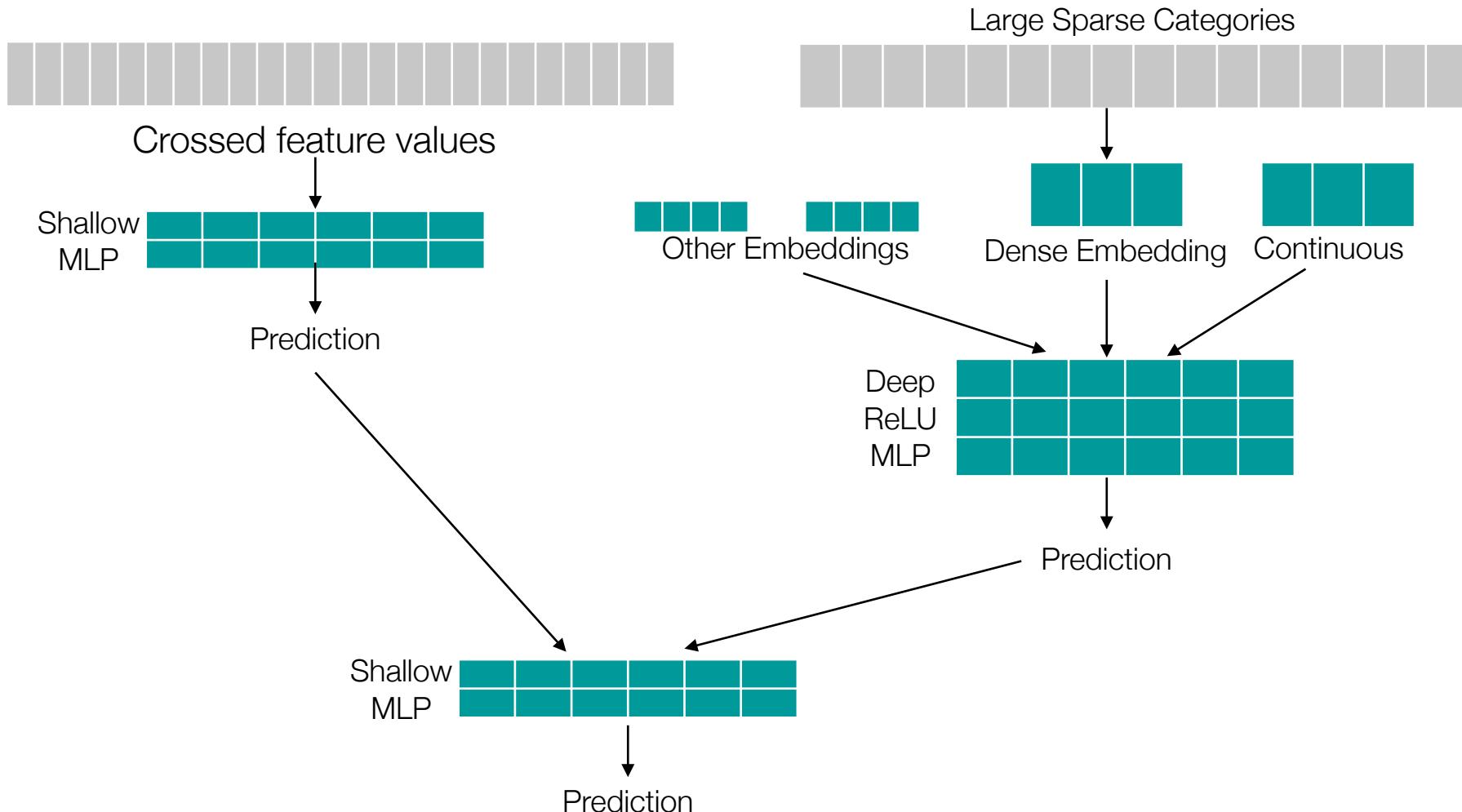
Sparse Embedding and Deep MLP (Generalize?)

- Deep refers to increasingly smaller hidden layers
- Embed into sparse representations via ReLU
- Movie Actors
 - Armand Assante
 - Meryl Streep
 - Danny Trejo
 - Kevin Bacon
 - Audrey Hepburn
 - ...
 -



Combining Memorization and Generalization

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

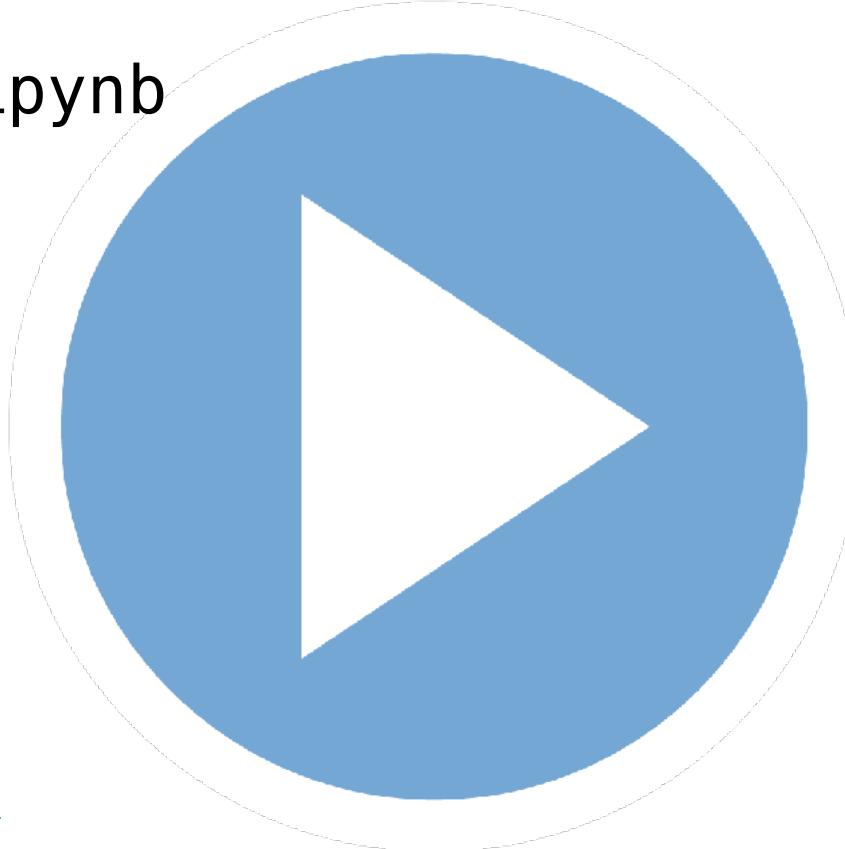


10. Keras Wide and Deep.ipynb

The awful dataset:
Toy Census Data Example

Other tutorials:

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



Wide and Deep Town Hall

End of Session

■ Next Time:

■ **Convolutional Neural Networks**