Lecture Notes for

Machine Learning in Python

Revisiting CV

A "Way too Early" History of Deep Learning

Online Education!

- Hope everyong is doing as well as possible
- I have no idea how this is going to go!
- It would help me if 4-5 people turn on their video
- Try the “Side-by-side” mode.
- Audio: keep muted unless there is a question
  - try raising hand, or just hold space bar to speak!
- Use chat sparingly: hard for me notice it
- If Zoom fails, I will just record lectures
- Recording of the lecture will always be uploaded to Panopto (but it will take some time and audio of questions might not get captured)
- Working in groups: use Colab and something like Zoom!
- Any questions before we get started?

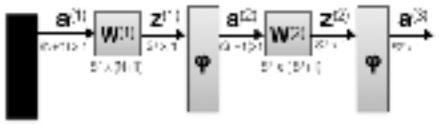
Logistics and Agenda

- Logistics
  - Welcome to online learning!
- 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 $\mathbf{y}^{(k)}$

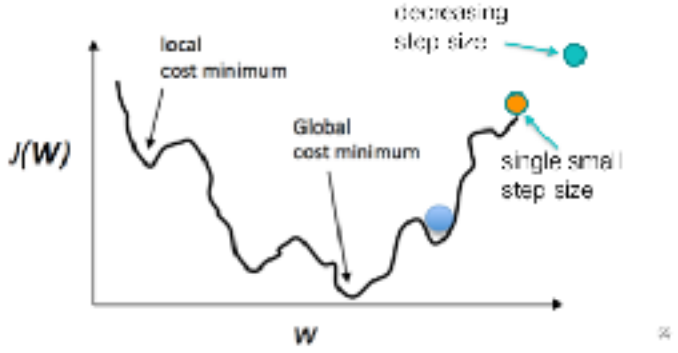
$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(2)}} = -2(\mathbf{y}^{(k)} - \mathbf{a}^{(3)}) * \mathbf{a}^{(3)} * (1 - \mathbf{a}^{(3)})$$

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

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

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



Practical Implementation of Architectures

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

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

speeds up initial training

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

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

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

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

new update

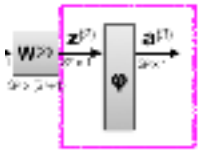
```
# vectorized backpropagation
sigma3 = -2*(Y_enc-A3)*A3*(1-A3)
sigma2 = (W2.T @ sigma3)*A2*(1-A2)

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

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

bp=5

Practical Implementation of Architectures

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

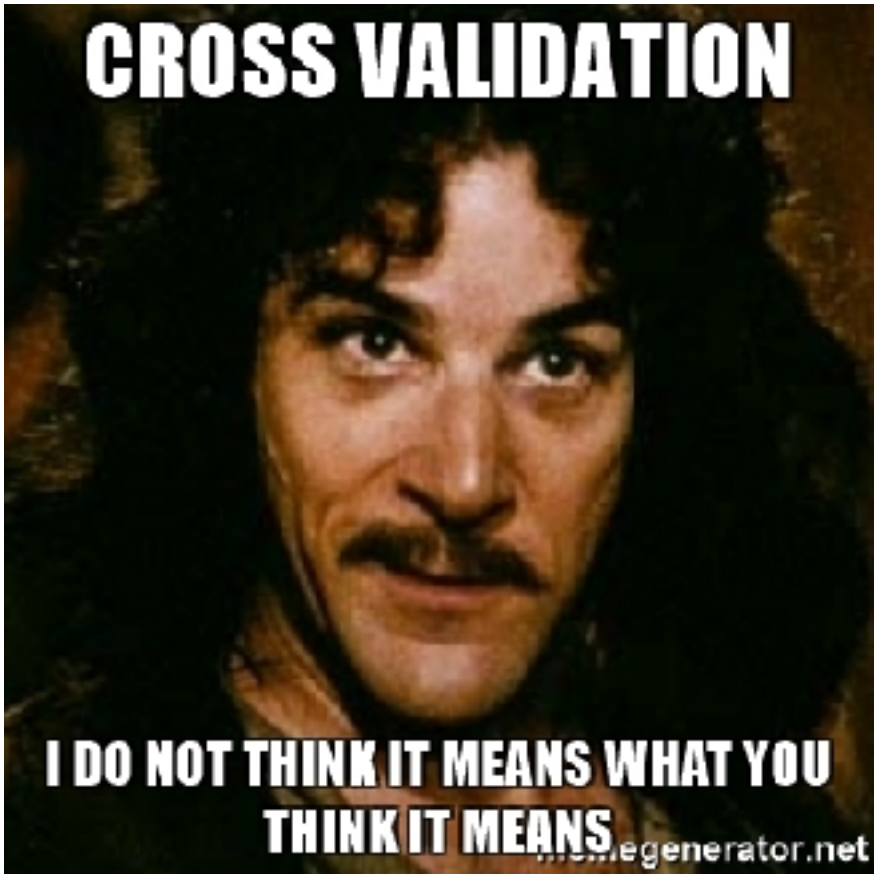


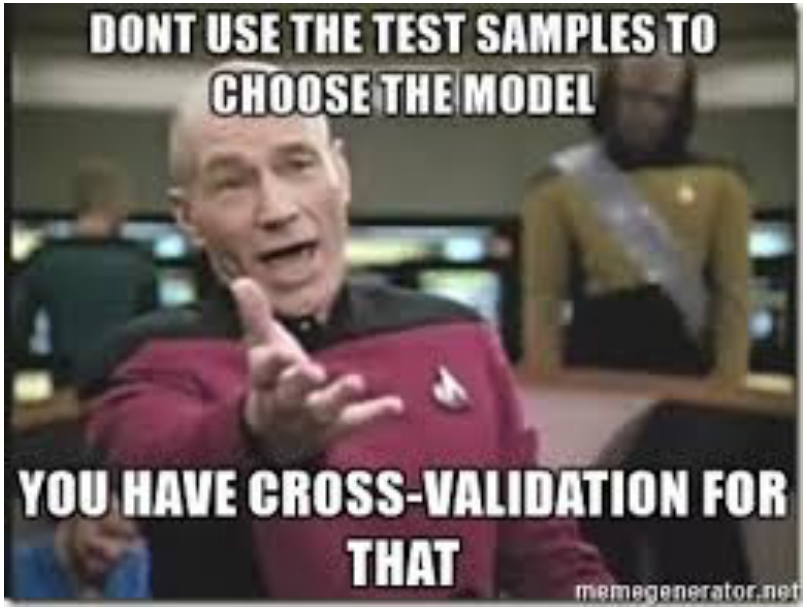
$$\phi(\mathbf{z}^{(l)}) = \begin{cases} \mathbf{z}^{(l)}, \text{ if } \mathbf{z}^{(l)} > 0 \\ 0, \text{ else} \end{cases}$$

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

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

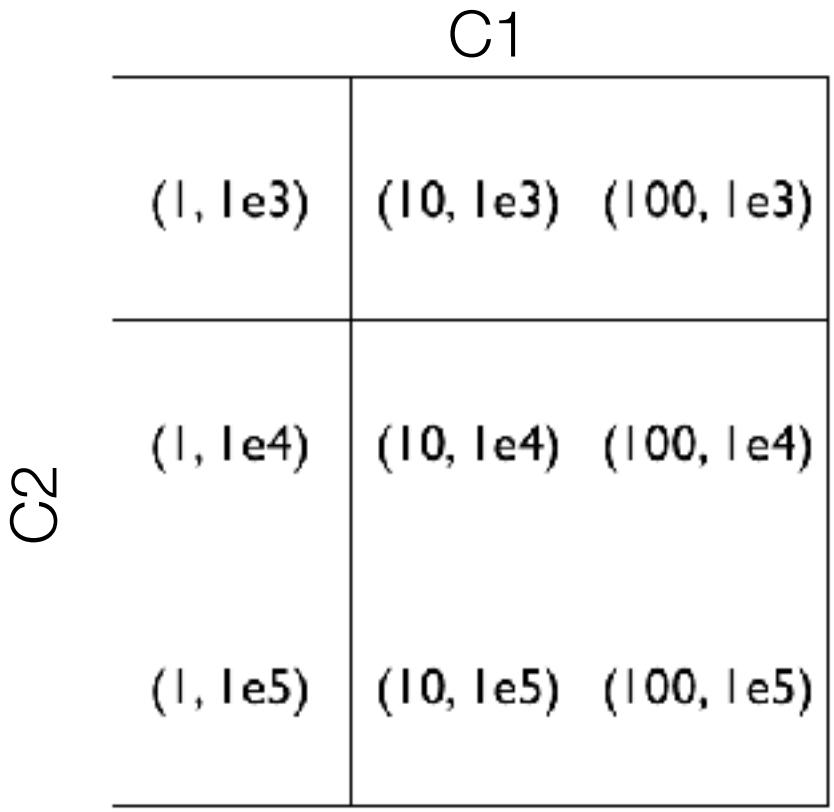
Revisiting Cross Validation





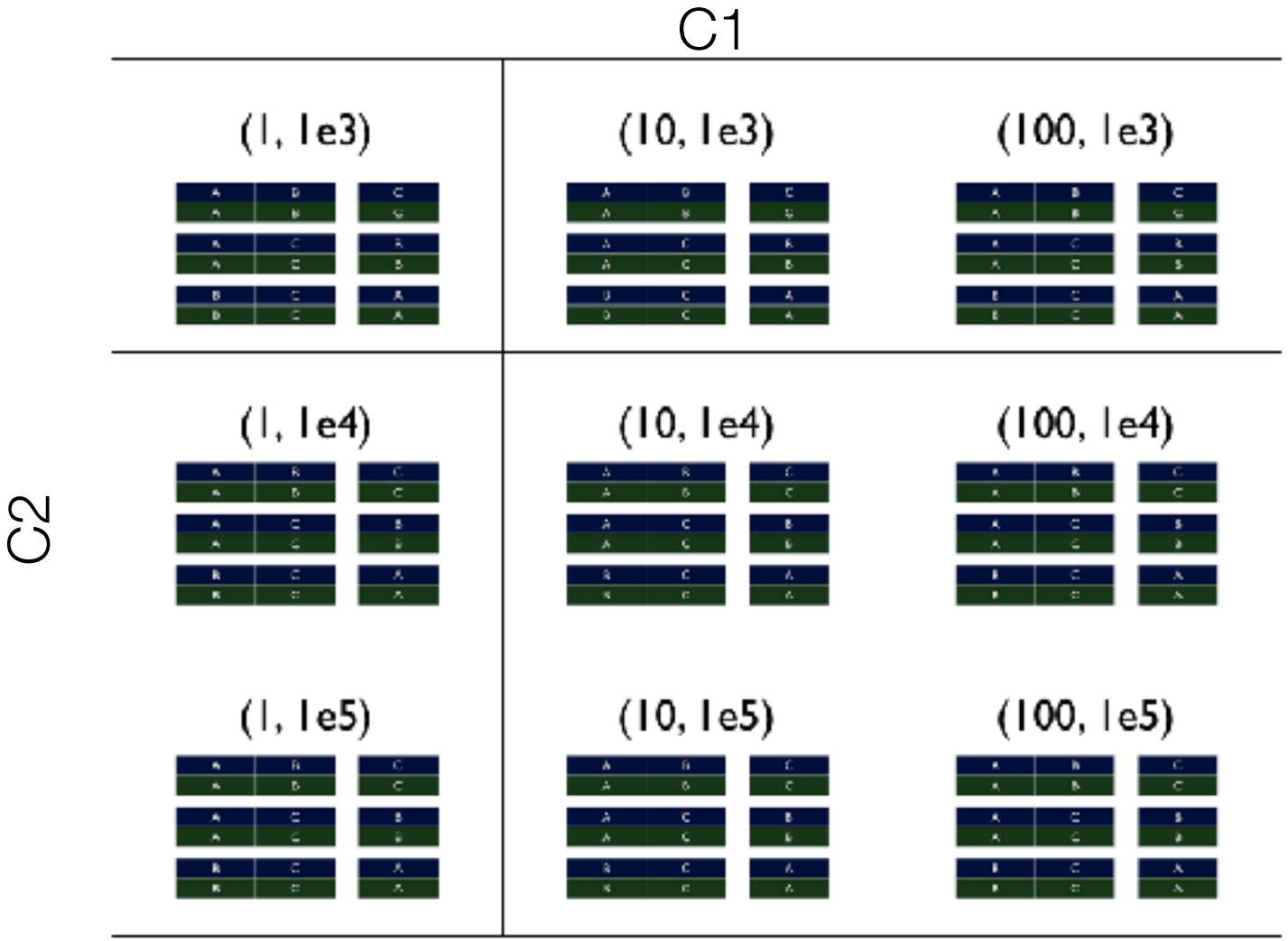
Grid Searching

- Trying to find the best parameters
  - `LR: C1=[1, 10, 100] C2=[1e3,1e4,1e5]`



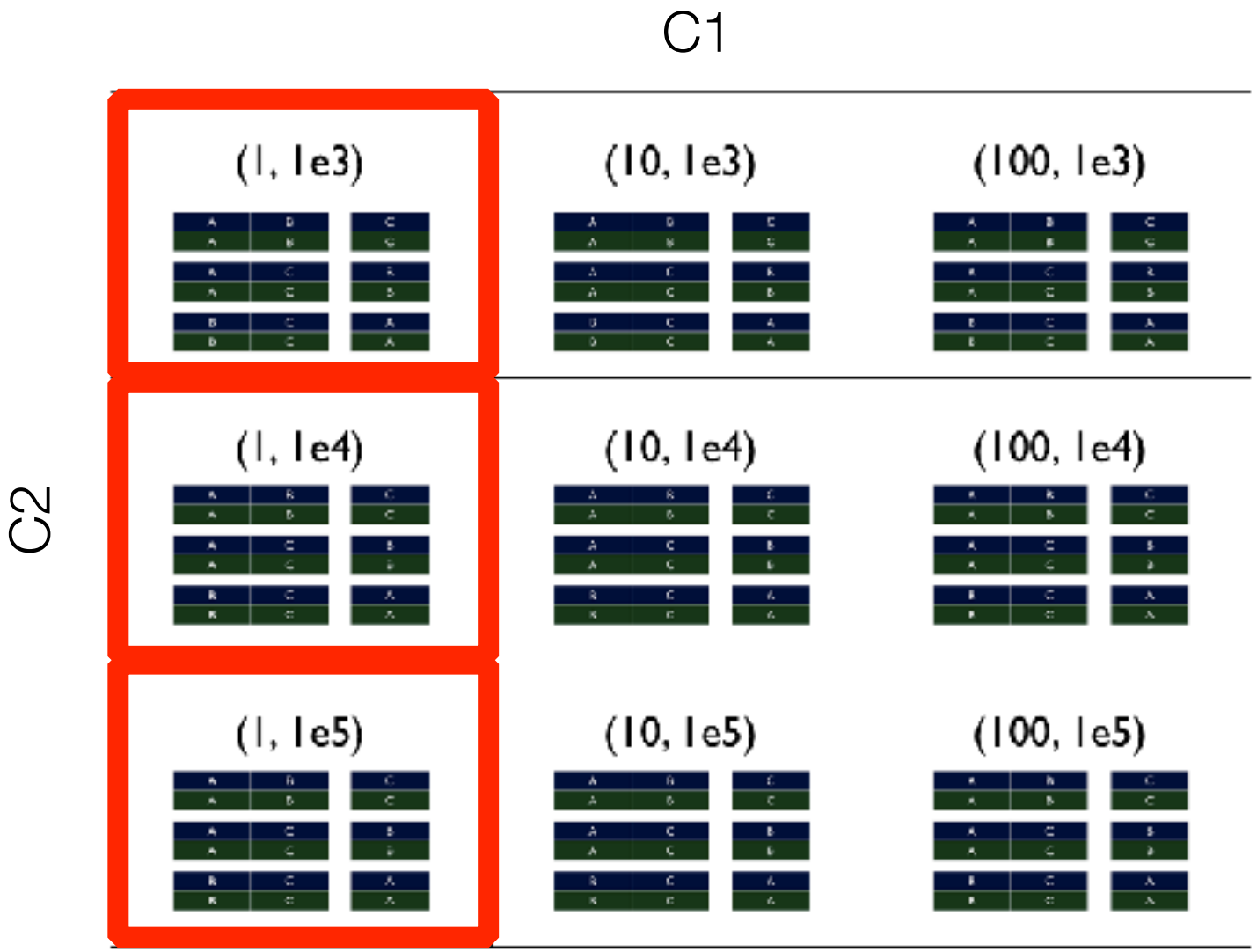
Grid Searching

- For each value, want to run cross validation…



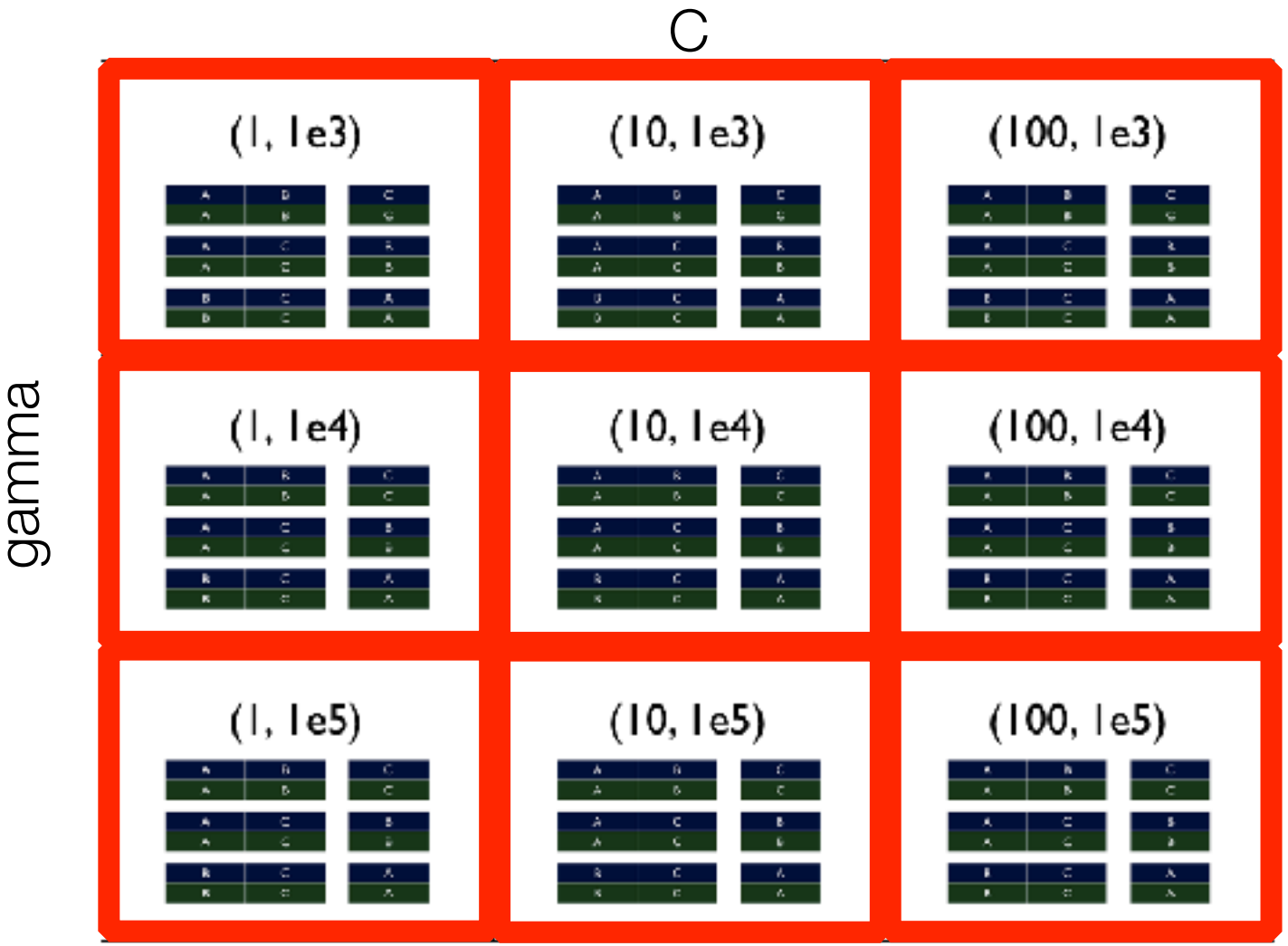
Grid Searching

- Could perform iteratively

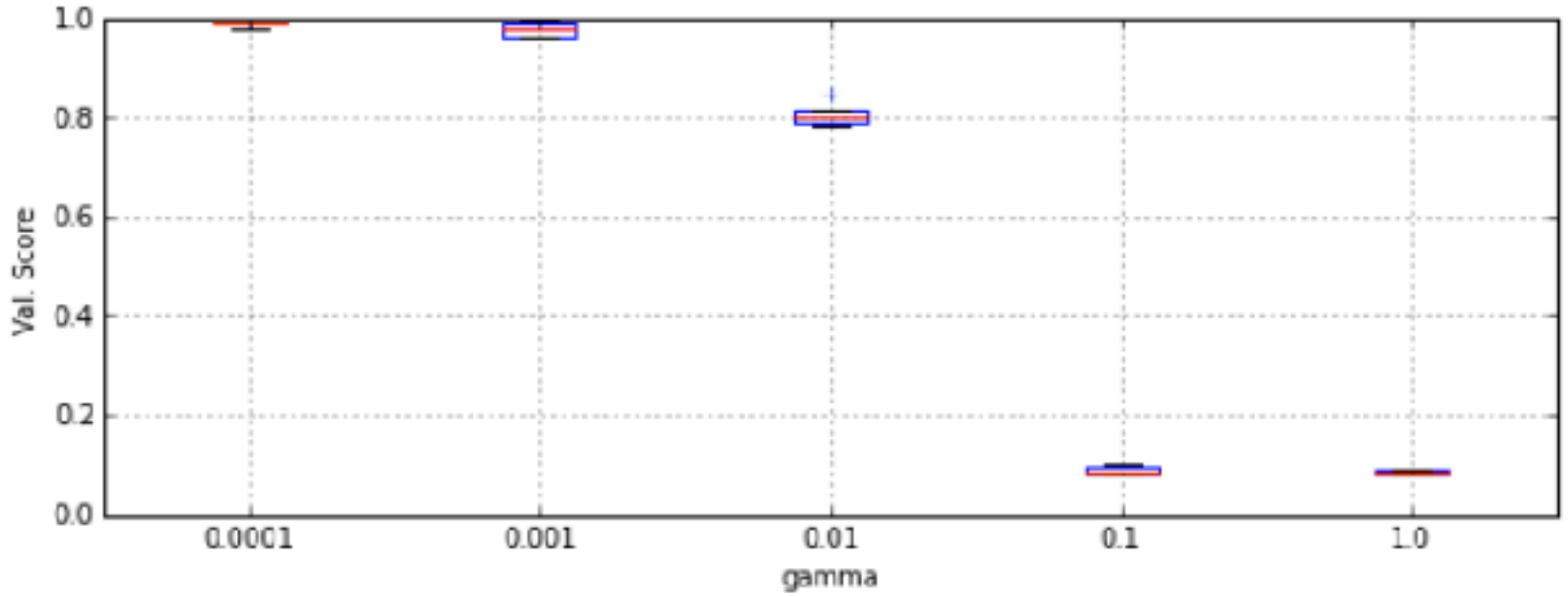


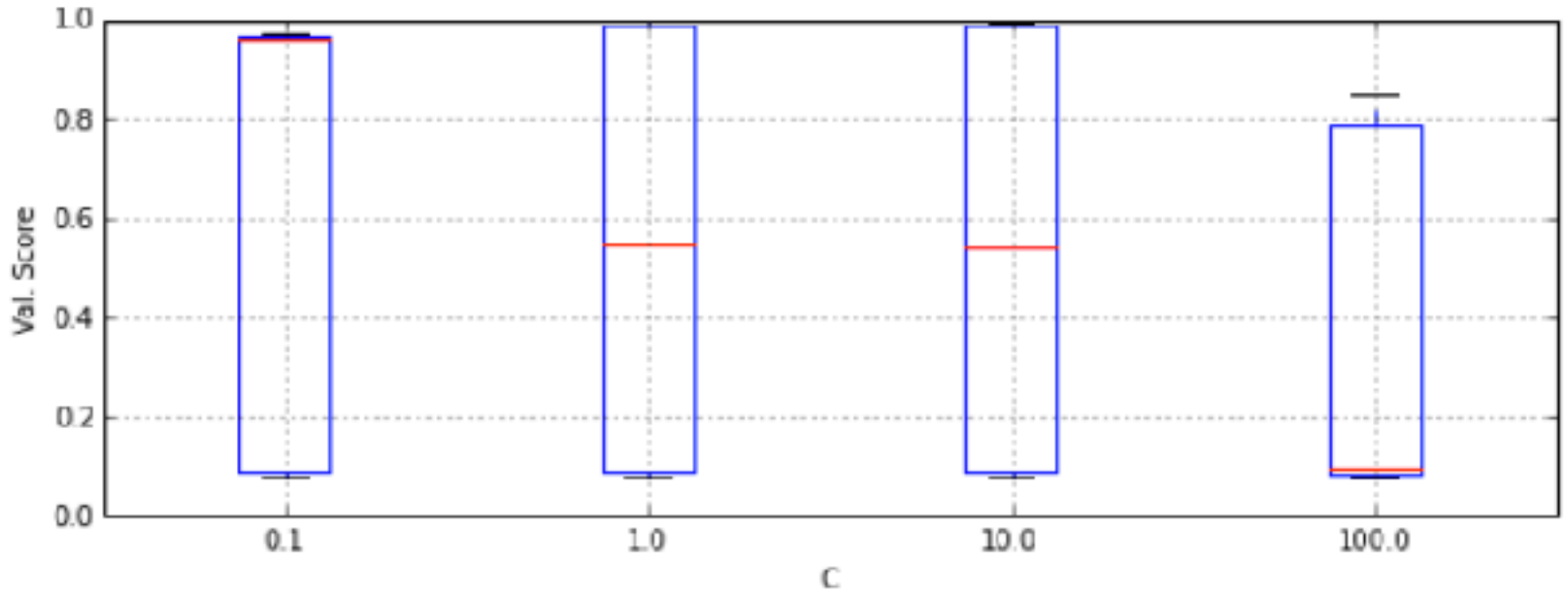
Grid Searching

- or at random…



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

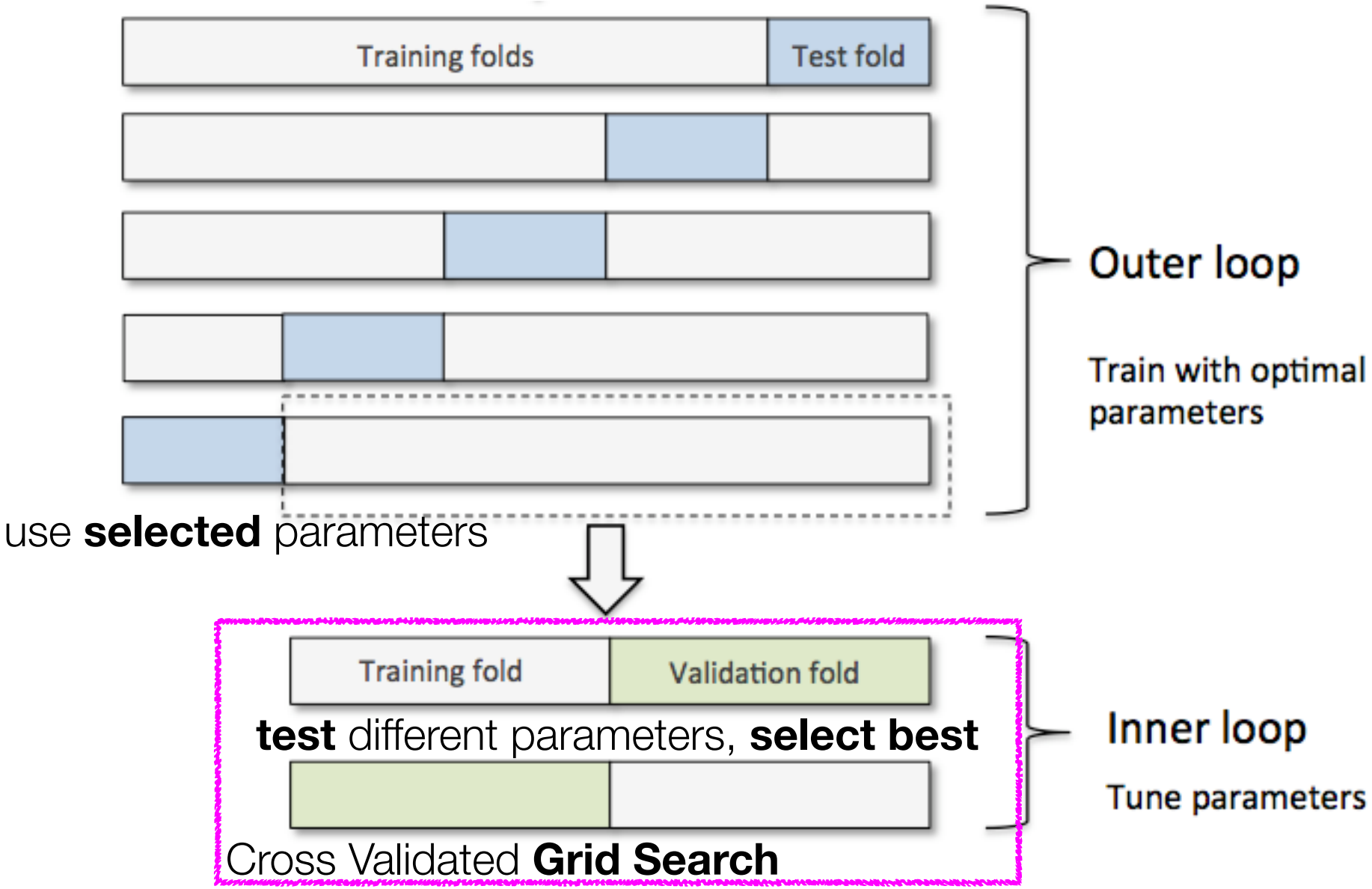




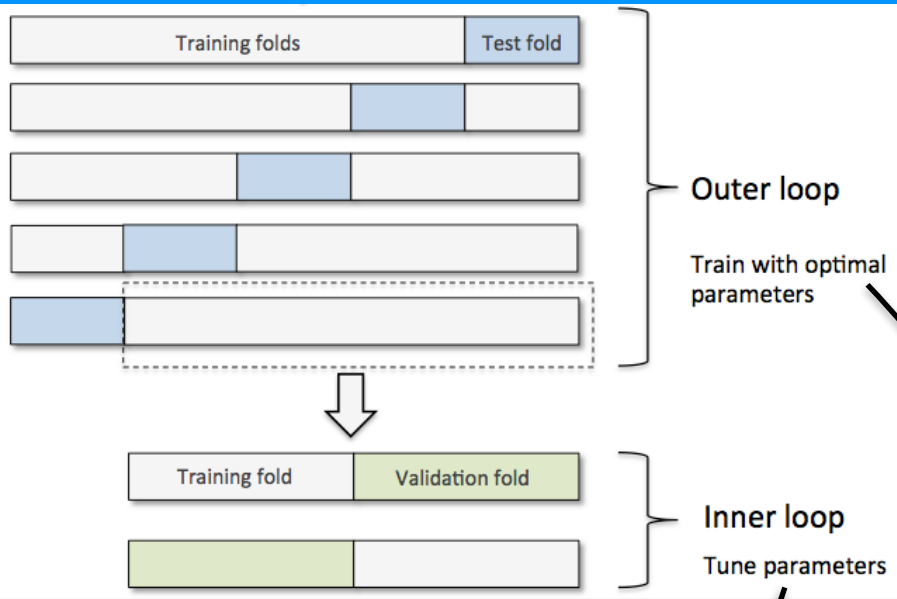
The Problem

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

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)))
```

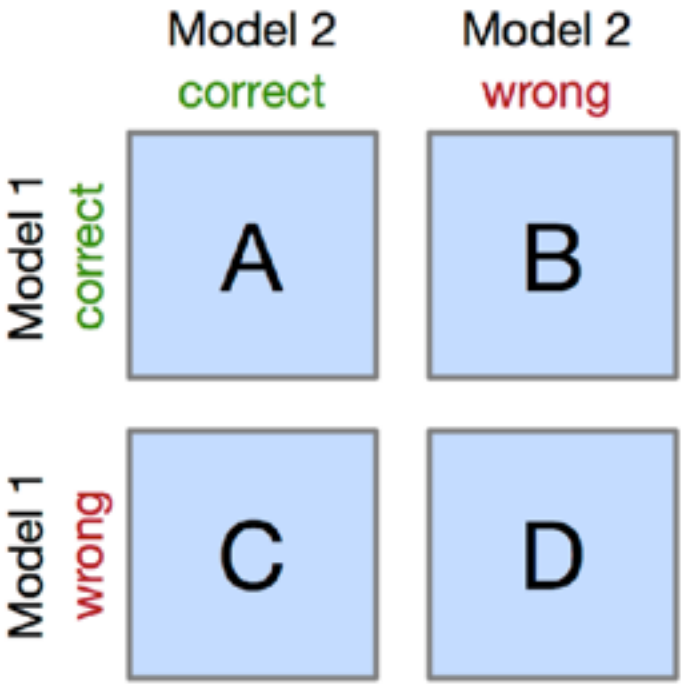
https://github.com/rasbt/python-machine-learning-book/

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!



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

McNemar and Edwards, 1948

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

Chi squared statistic, one DOF

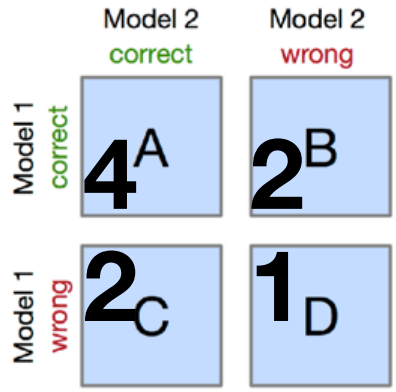
Steps:

1. Compare each model's performance on the same test data
2. Calculate Chi statistic
3. Look up P-value associated with CHi statistic
4. Can you reject the null hypthesis that the models are the same?

https://sebastianraschka.com/blog/2018/model-evaluation-selection-part4.html

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



McNemar and Edwards, 1948

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

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

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 < 2.706, we cannot reject the null hypothesis. This means **we cannot say the models are different** based on the evidence.

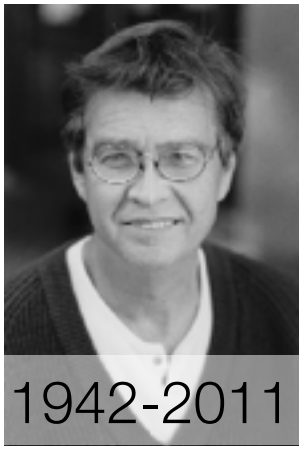
Town Hall

Some History of Deep Learning

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

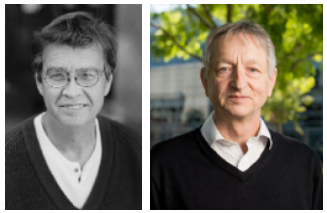
Geoffrey Hinton

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
- Second AI winter begins, research in NN plummets
- Funding for and accepted papers with Neural Networks asymptotically approaches zero

1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network



1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs

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

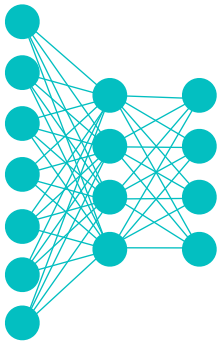
1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

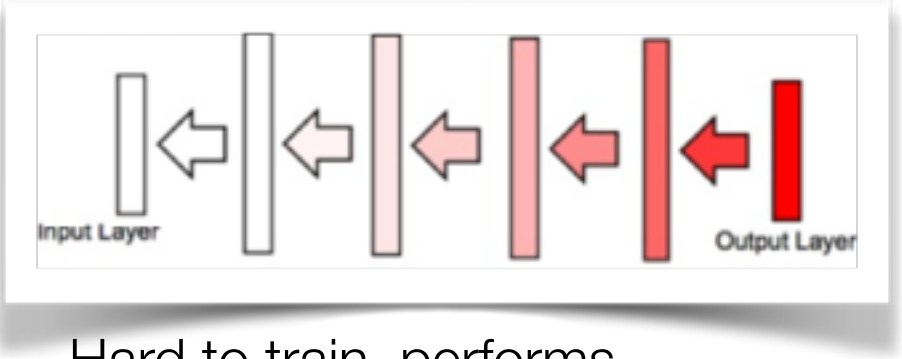
Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

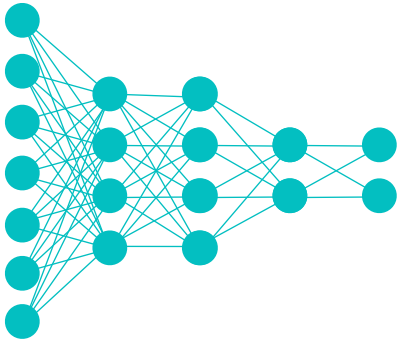
History of Deep Learning: Winter

- AI Winter is coming:



Easy to train, performs on par with other methods





Hard to train, performs worse than other methods

Researcher have difficulty reconciling expressiveness with performance

~chance (untrainable)

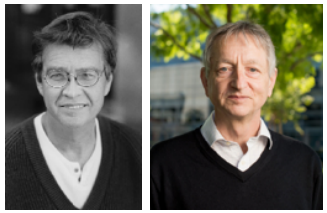
Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

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

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

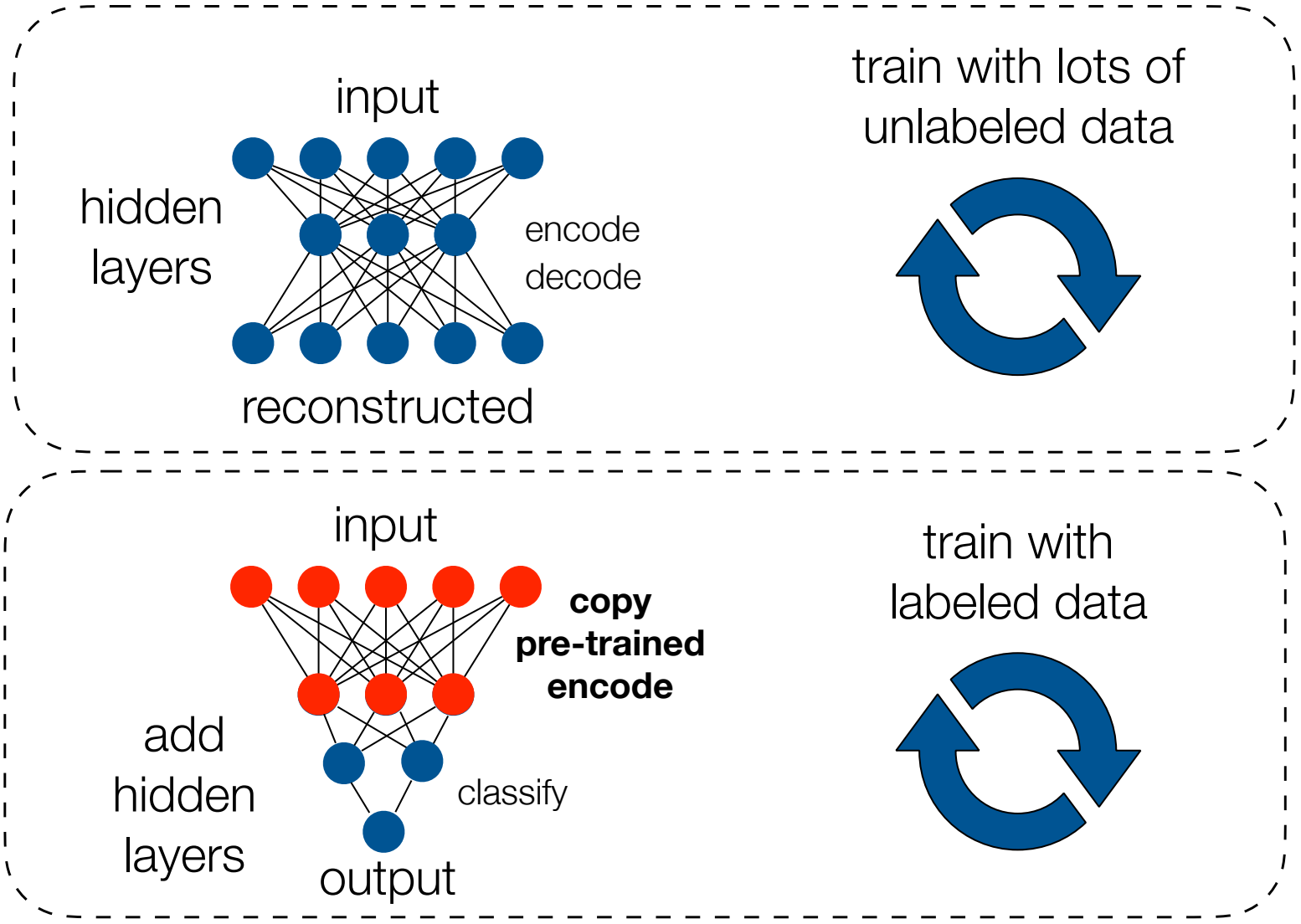
1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

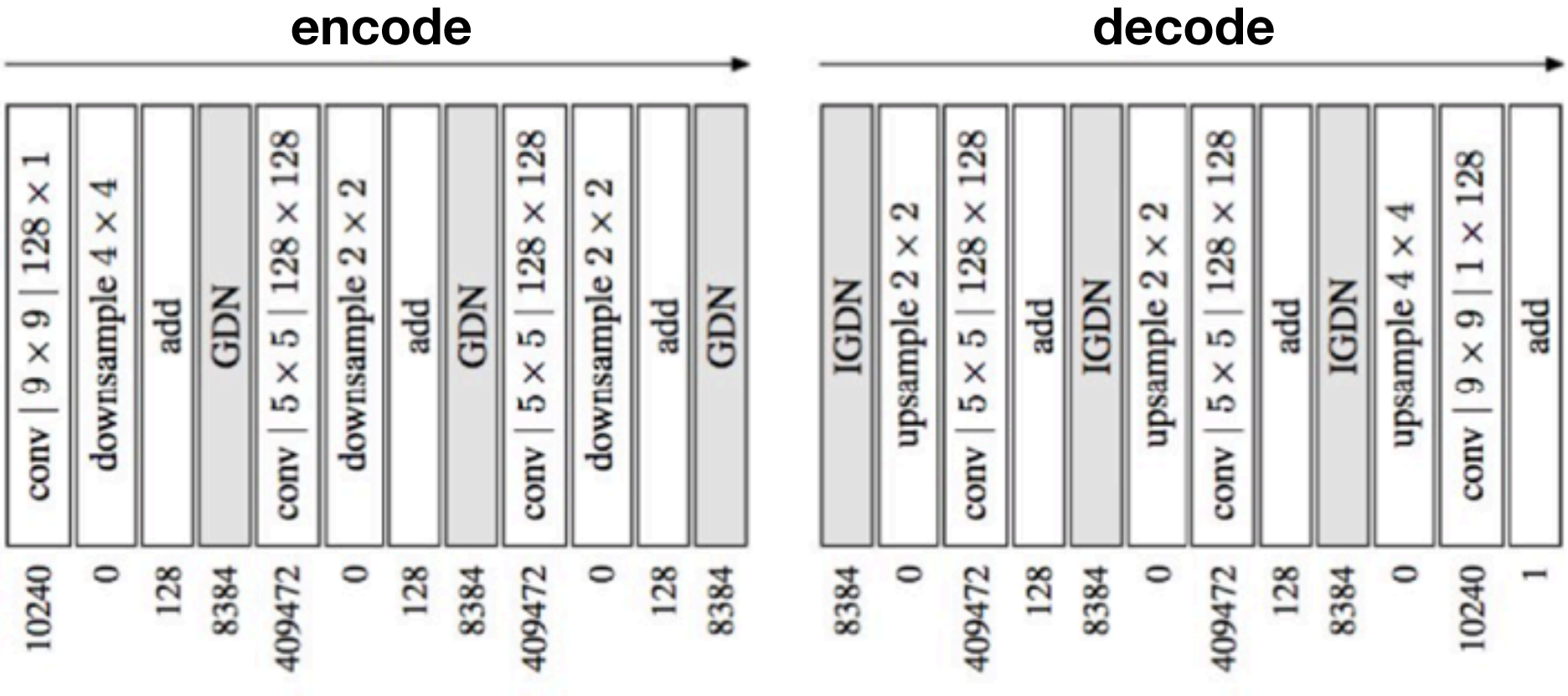
Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

Pre-training: still in the long winter

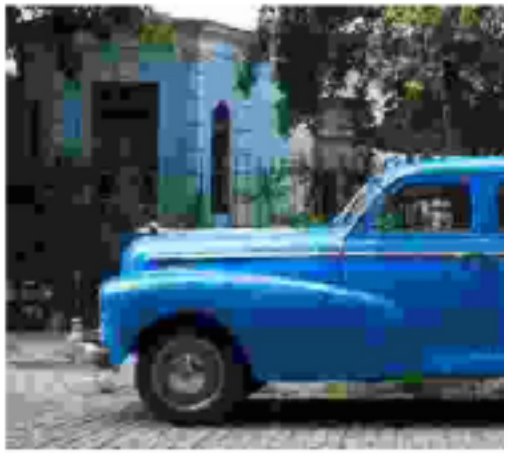
- auto-encoding (a form of pre-training)



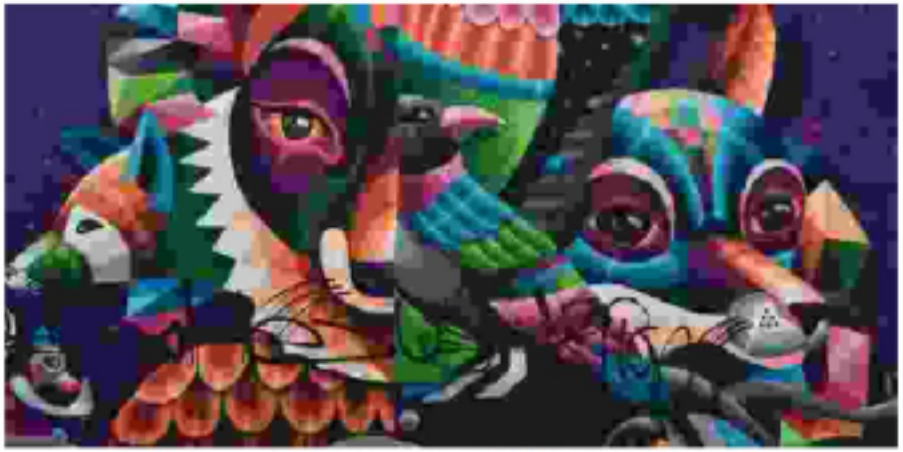
Pre-training: modern example



https://arxiv.org/abs/1611.01704



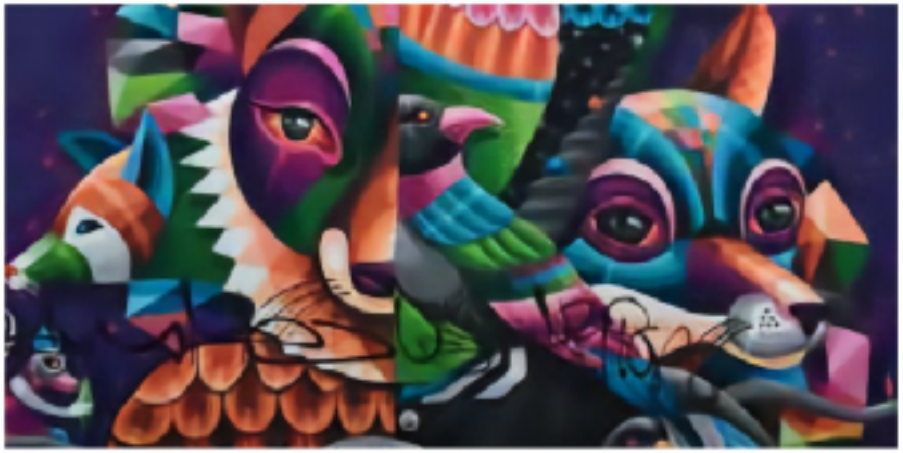
JPEG, 6006 bytes (0.170 bit/px), RMSE: 19.75



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

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

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

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

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

Still in the Long Winter

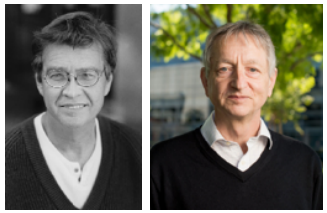
- 2009: Hinton's lab starts using GPUs, Also Andrew Ng
  - GPUs decrease training time by 70 fold…
- 2010: Hinton's and Ng's students go to internships with Microsoft, Google, IBM, and Facebook

Navdeep Jaitly
Google Brain Team


George Dahl
Google Brain Team


1949, Hebb's Law
Close neuron fire together


1960, Widrow & Hoff
Adaline Network


1986, Rumelhart & Hinton
Back-propagation


2003, Vapnik
Kernel SVMs

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

1940 | 1960 | 1980 | 2000

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

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

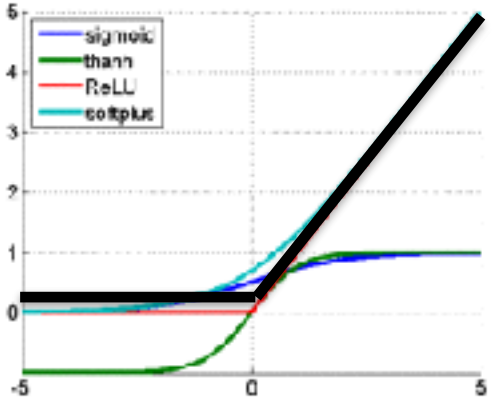
1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

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

Still in the long Winter

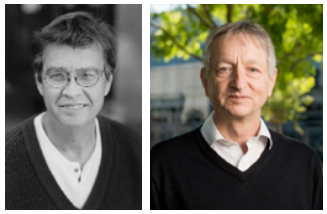
- 2011: Glorot and Bengio investigate more systematic methods for why past deep architectures did not work
  - **discover some interesting, simple fixes**: the type of neurons chosen and the selection of initial weights
  - do not require pre-training to get deep networks properly trained, just sparser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



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

1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network



1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs

1940 — 1960 — 1980 — 2000 — 2020

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

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

2011, Bengio
Init and ReLU

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

Machine Learning Timeline (

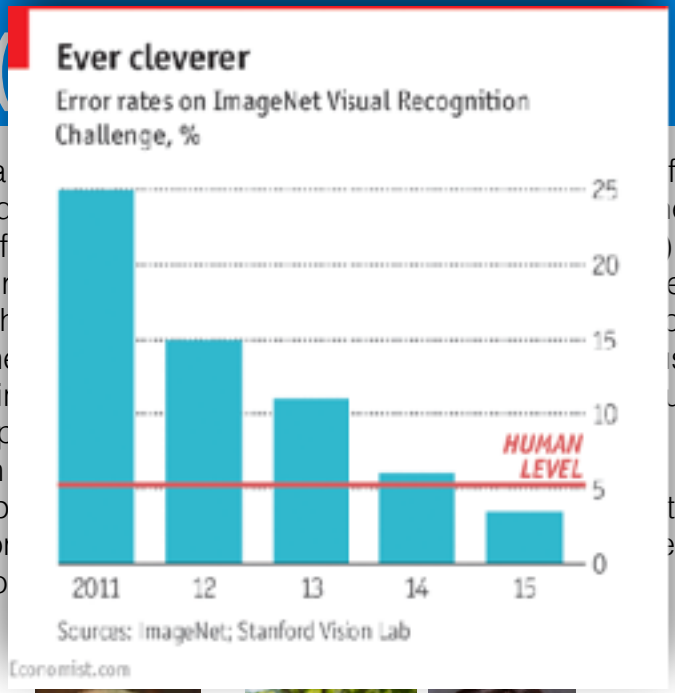
ImageNet competition occurs

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

"I ha
hanc
last f
so I r
as sh
come
pacir
happ
from
skep
a cor
Visio

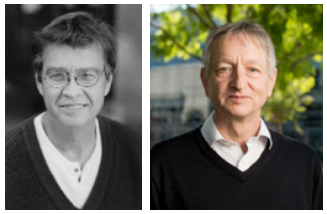
first
ıe
),
ef
on
s
ust

t as
er



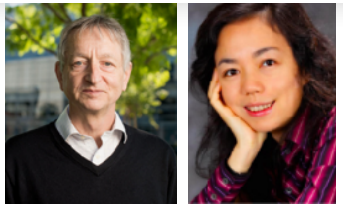
1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network



1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs



2012, Hinton, Fei-Fei Li
CNNs win ImageNet

1940 — 1960 — 1980 — 2000 — 2020

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

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

1969, Minsky & Papert
Linear Models are Doomed

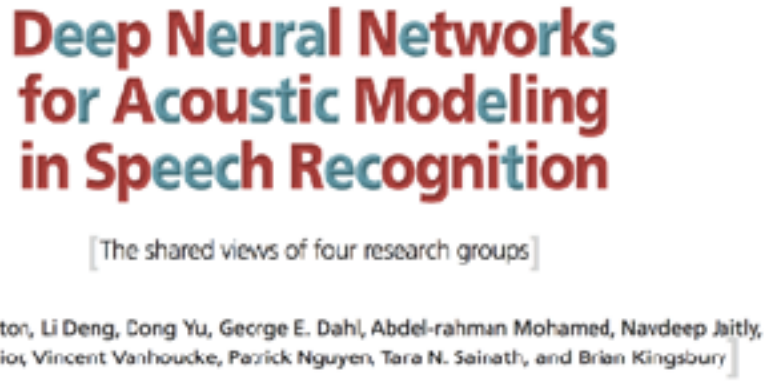
2001, Breiman
Random Forests

2011, Bengio
Init and ReLU

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

Machine Learning Timeline (Neural Nets)

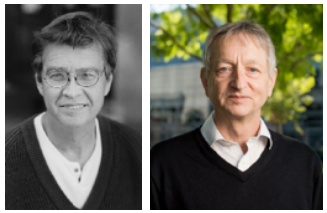
- 2012: Hinton Lab, Google, IBM, and Microsoft jointly publish paper, popularity for deep learning methods increases



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

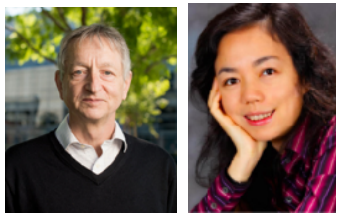
1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network



1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs



2012, Hinton, Fei-Fei Li
CNNs win ImageNet

1940 | 1960 | 1980 | 2000 | 2020

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

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

2011, Bengio
Init and ReLU

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

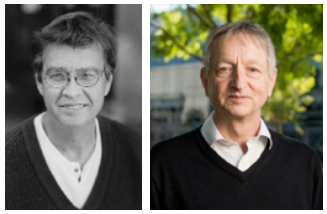
Machine Learning Timeline (Neural Nets)

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

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

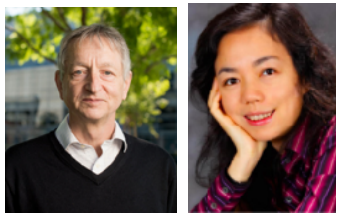
1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network



1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs



2012, Hinton, Fei-Fei Li
CNNs win ImageNet

1940 | 1960 | 1980 | 2000 | 2020

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

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

2011, Bengio
Init and ReLU

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

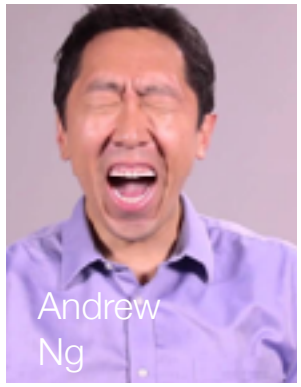
A summary of the Deep Learning people:

Stayed at
University
Advises IBM

Heads
Facebook
AI Team

Google

Chief Scientist,
AI/ML
Google Cloud



Coursera
Baidu
Google

Stanford
Founded Coursera
MacArthur Genius

Popularized CNNs, RNNs, and Mitigated Gradient Instability

Made Deep Learning Instruction Accessible

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

And predicts the future of deep learning

REVIEW

doi:10.1038/nature14539

Deep learning

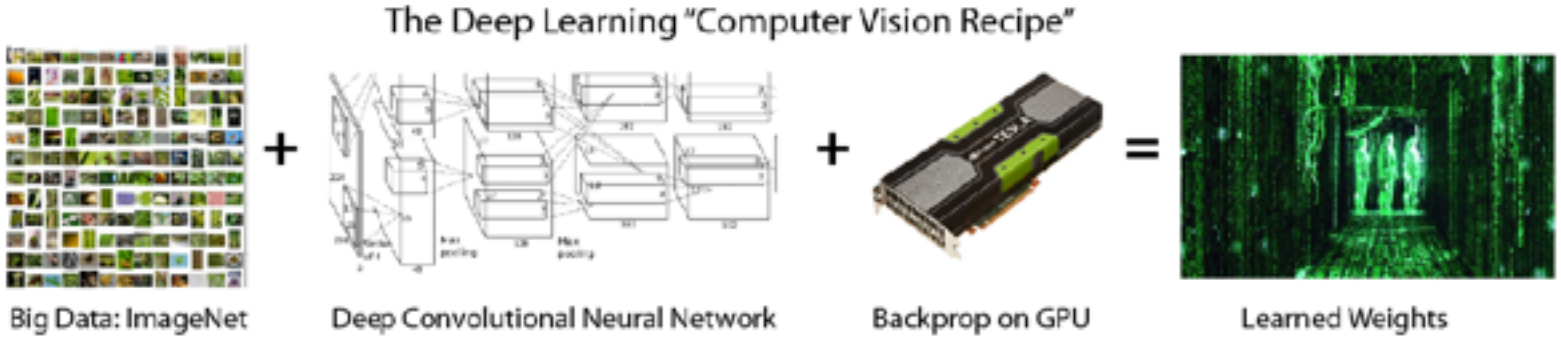
Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

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

Lecture Notes for Machine Learnin

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!

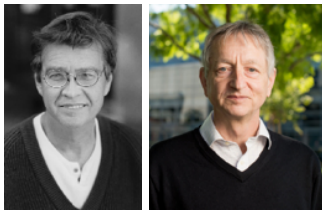


Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

Machine Learning Timeline (Neural Nets)

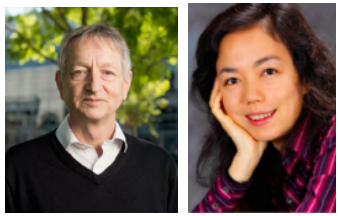
1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network



1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs



2012, Hinton, Fei-Fei Li
CNNs win ImageNet

1940 | 1960 | 1980 | 2000 | 2020

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

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

2011, Bengio
Init and ReLU

2015, Google
Tensorflow Open Sourced

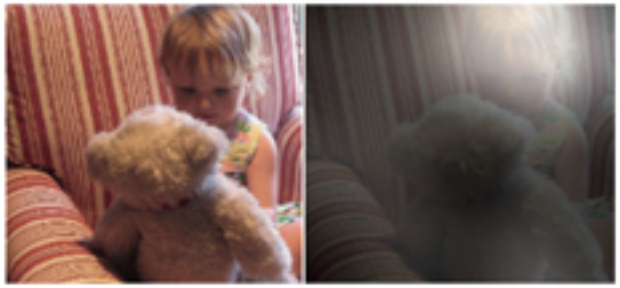
TensorFlow

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

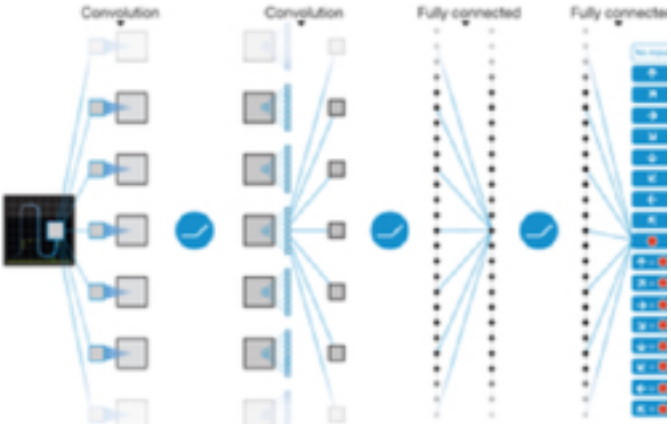


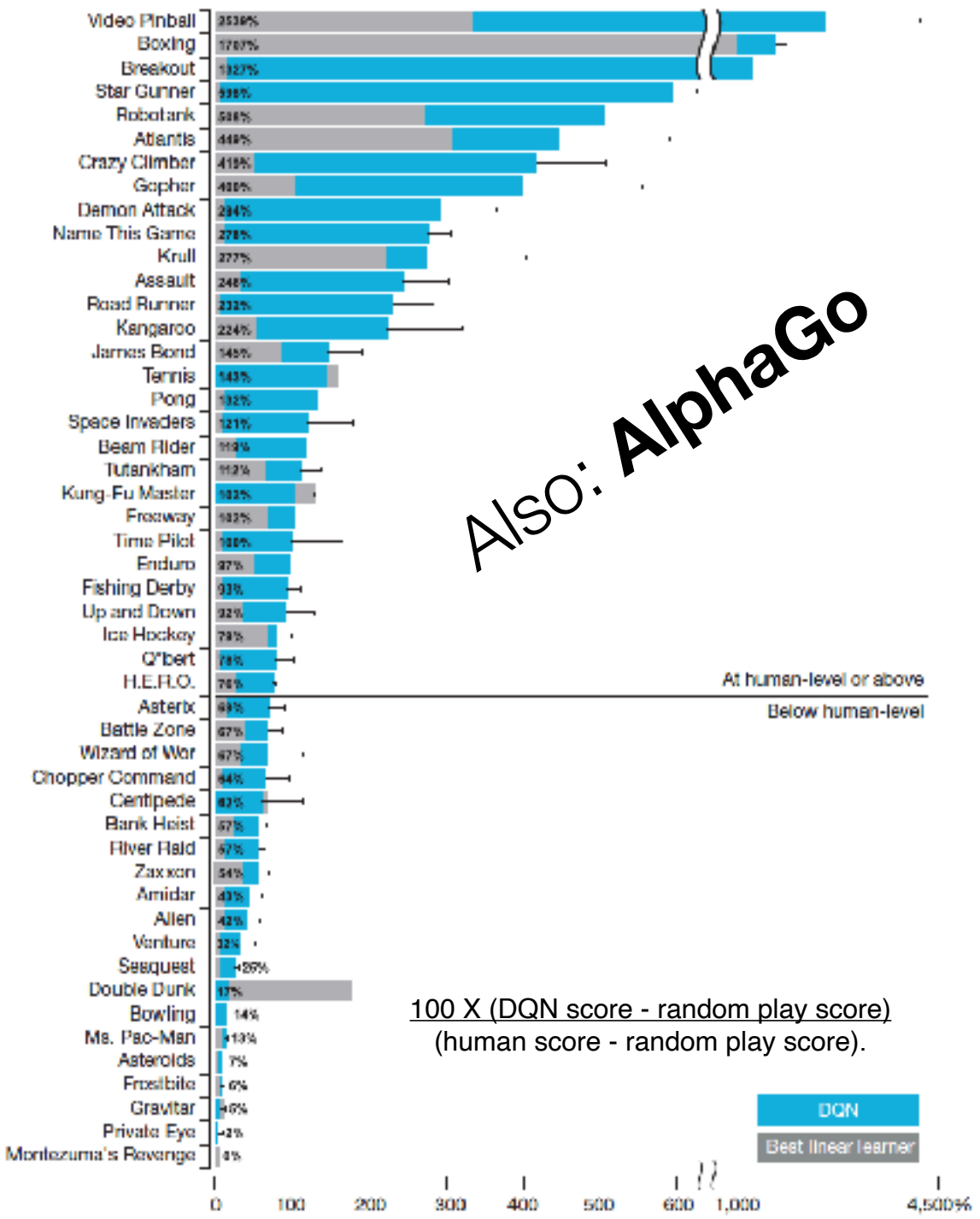
A little **girl** sitting on a bed with a teddy bear.

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

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

More Famous



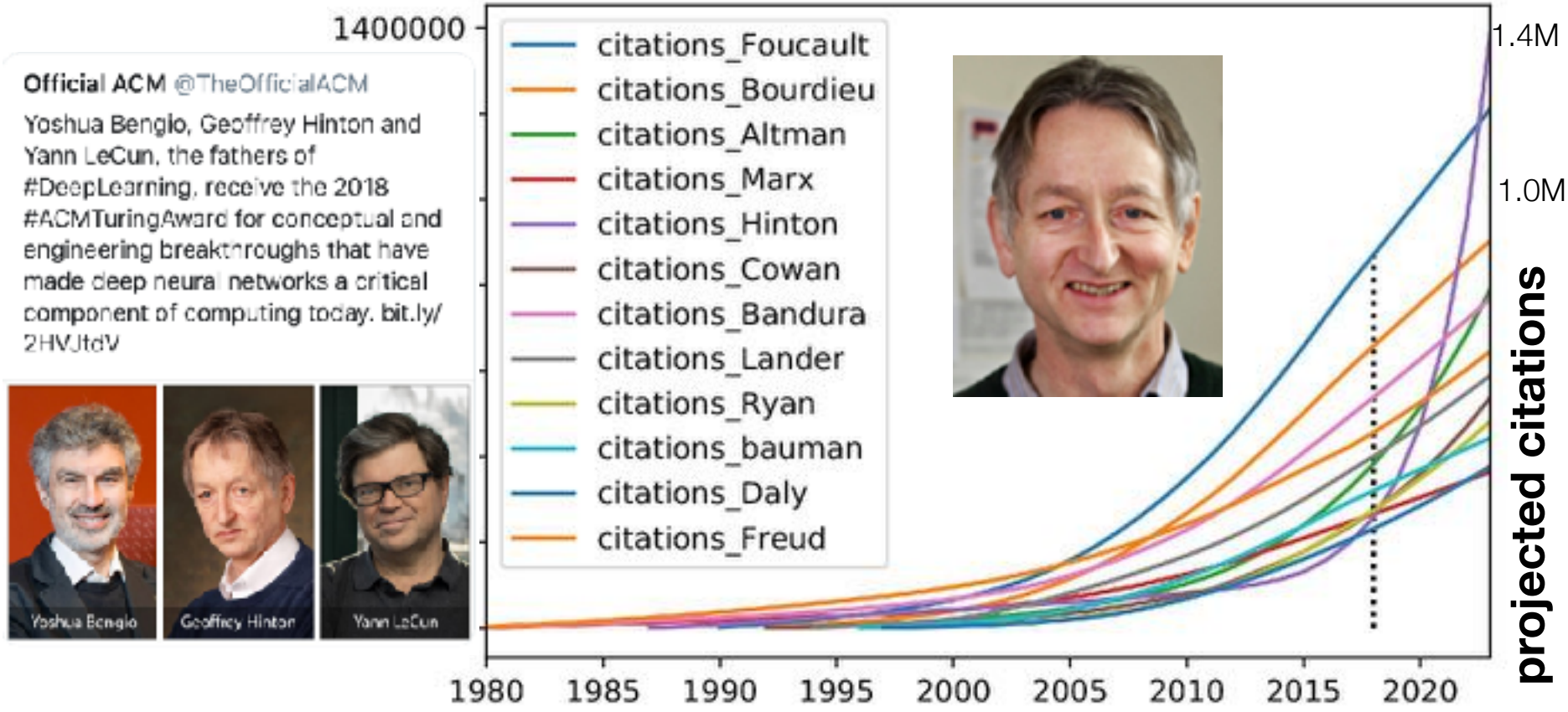


Also: **AlphaGo**

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

End of Session

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



End of Session

- if time:
  - batch normalization

Backup: History of Deep Learning

- ReLU not the only way to do it!

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

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

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

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

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

Batch Normalization

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

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

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

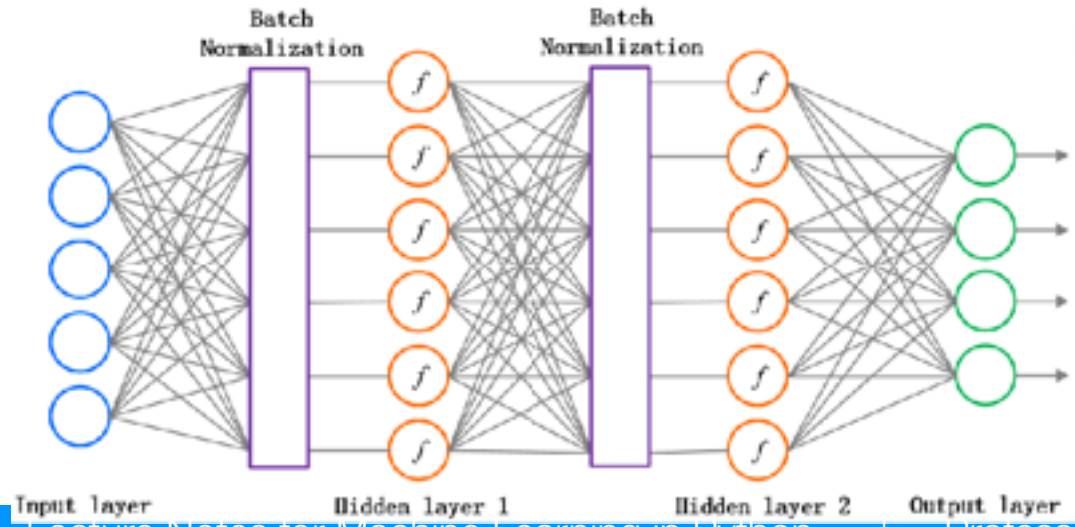
$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

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

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

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

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



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

Lecture Notes for

Machine Learning in Python

Professor Eric Larson

Wide and Deep Networks

Lecture Agenda

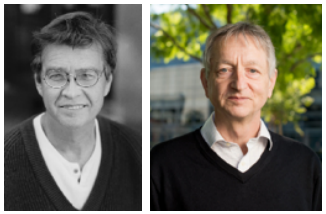
- Logistics: CS 8321 in Spring
  - COVID Grading
- Introduction to TensorFlow
  - Tensors, Namespaces, Numerical methods
  - Deep APIs
- Wide and Deep Networks

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
- 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)

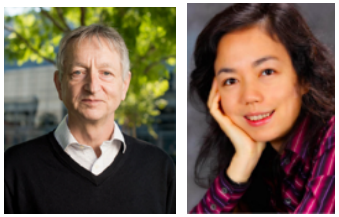
1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network

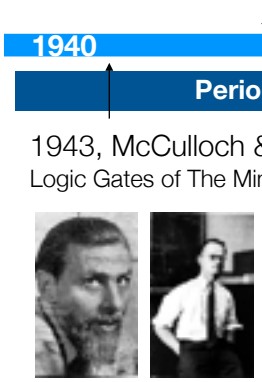


1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs



2012, Hinton, Fei-Fei Li
CNNs win ImageNet



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

2015, Google
Tensorflow Open Source

1940 | 1960 | 1980 | 2000 | 2020

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

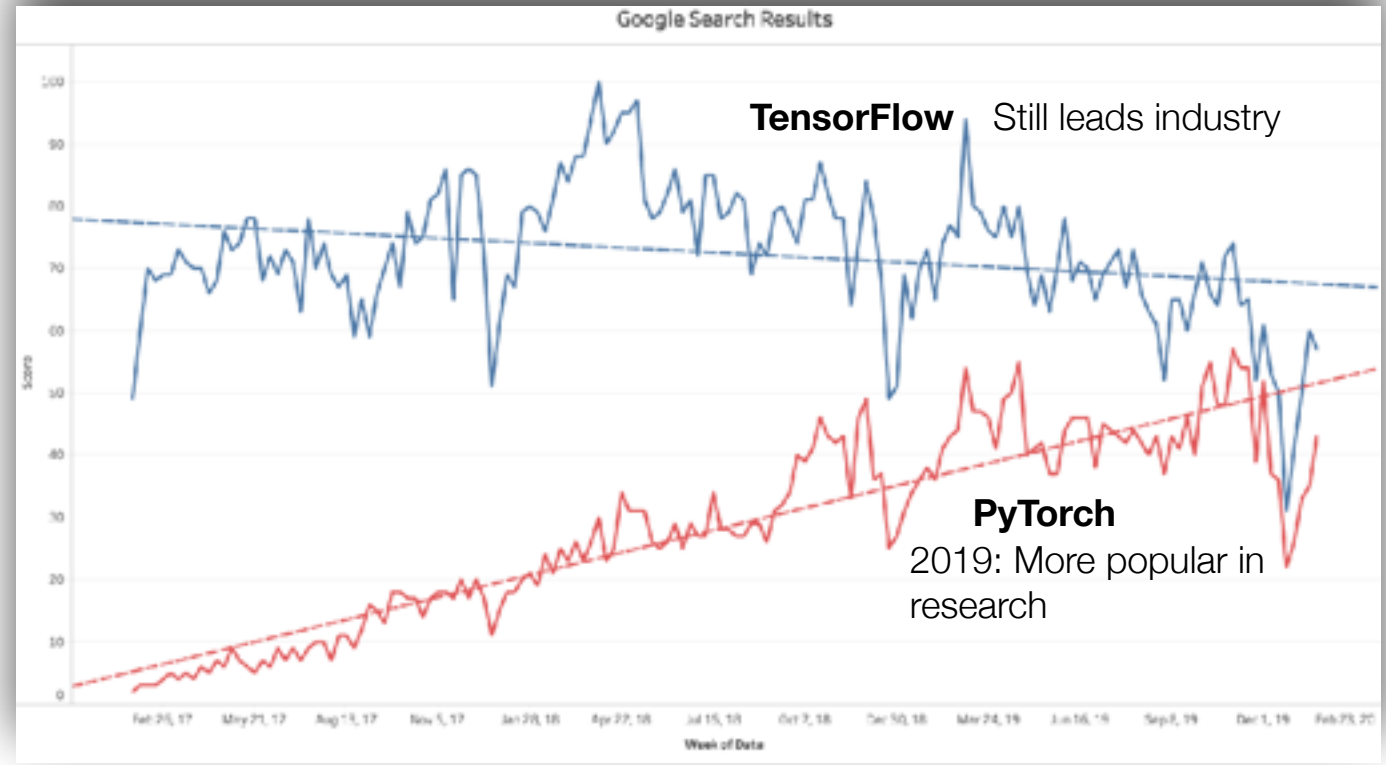
TensorFlow

"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



	Online Job Listings					KDnuggets	Google	Medium	Amazon	ArXiv	GitHub Activity			
Framework	Indeed	Monster	Simply Hired	LinkedIn	Angel List	Usage Survey	Search Volume	Articles	Books	Articles	Stars	Watchers	Forks	Contributors
TensorFlow	2,079	1,253	1,582	2,610	662	25.90%	73	6,200	202	3,700	136,376	8,334	87,501	1,042
Keras	654	364	448	685	177	22.20%	53	9,100	78	1,360	33,558	1,647	12,658	719
PyTorch	488	309	428	685	120	8.40%	19	1,780	18	1,560	18,718	952	4,474	780
Caffe	607	399	516	868	123	1.50%	4	816	14	1,300	25,804	2,218	16,533	270
Theano	356	316	278	508	95	4.90%	0	436	17	852	8,477	585	3,447	328
MXNET	266	154	200	298	29	1.50%	2	524	32	260	15,200	1,170	5,498	687
CNTK	126	86	97	160	12	3.00%	0	223	1	68	15,105	1,368	4,029	169
DeepLearning4J	17	5	9	35	3	3.40%	2	70	11	27	9,815	629	4,441	232
Caffe2	55	51	49	109	12	1.20%	2	336	2	67	8,284	577	2,102	193
Chainer	19	19	18	28	3	0.00%	2	91	3	164	4,198	325	1,085	102
FastAI	0	0	0	0	0	0.00%	0	558	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
- **Toolkits Support:**
  - tensor creation
  - functions on tensors
  - automatic differentiation

- Tensors are just multidimensional arrays
  - like in Numpy
    - scalars (biases and constants)
    - vectors (e.g., input arrays)
    - 2D matrices (e.g., images)
    - 3D matrices (e.g., color images)
    - 4D matrices (e.g., batches of color images)

Tensor basic functions

```
a = tf.constant(5.0)

b = tf.constant(6.0)

c = a * b
```

- Easy to define operations on tensors

Numpy	TensorFlow
`a = np.zeros((2,2)); b = np.ones((2,2))`	`a = tf.zeros((2,2)), b = tf.ones((2,2))`
`np.sum(b, axis=1)`	`tf.reduce_sum(a,reduction_indices=[1])`
`a.shape`	`a.get_shape()`
`np.reshape(a, (1,4))`	`tf.reshape(a, (1,4))`
`b * 5 + 1`	`b * 5 + 1`
`np.dot(a,b)`	`tf.matmul(a, b)`
`a[0,0], a[:,0], a[0,:]`	`a[0,0], a[:,0], a[0,:]`

- Also supports convolution: `tf.nn.conv2d, tf.nn.conv3D`

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

Tensor neural network functions

- Easy to define operations on layers of networks

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

- Each function 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())
```

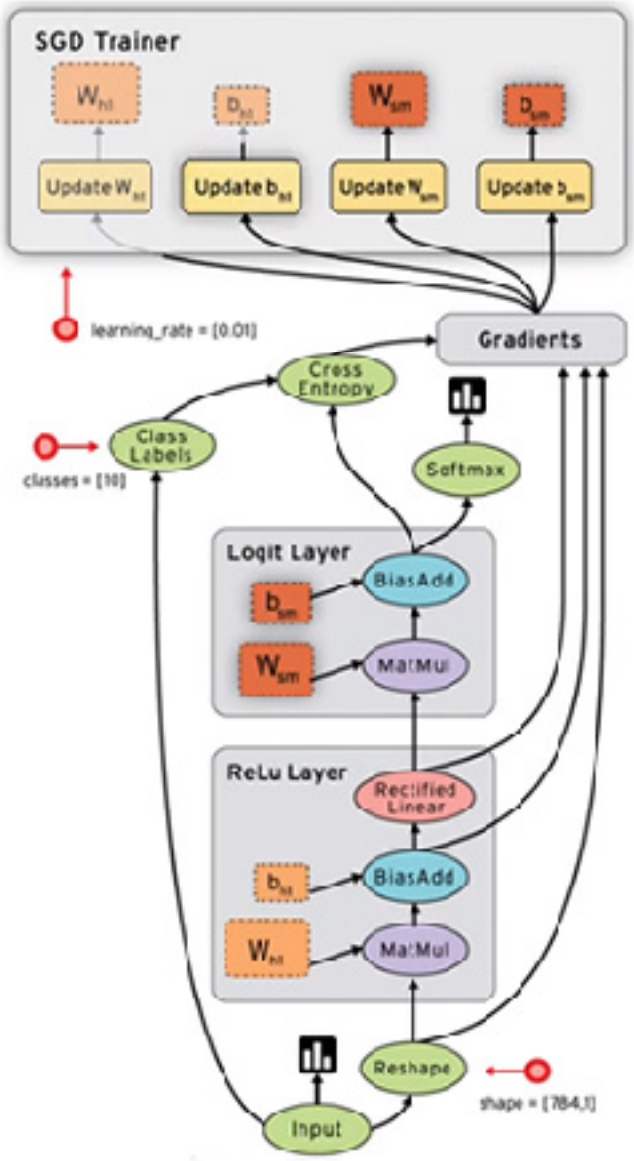
output = 30

- Easy to define operations on tensors

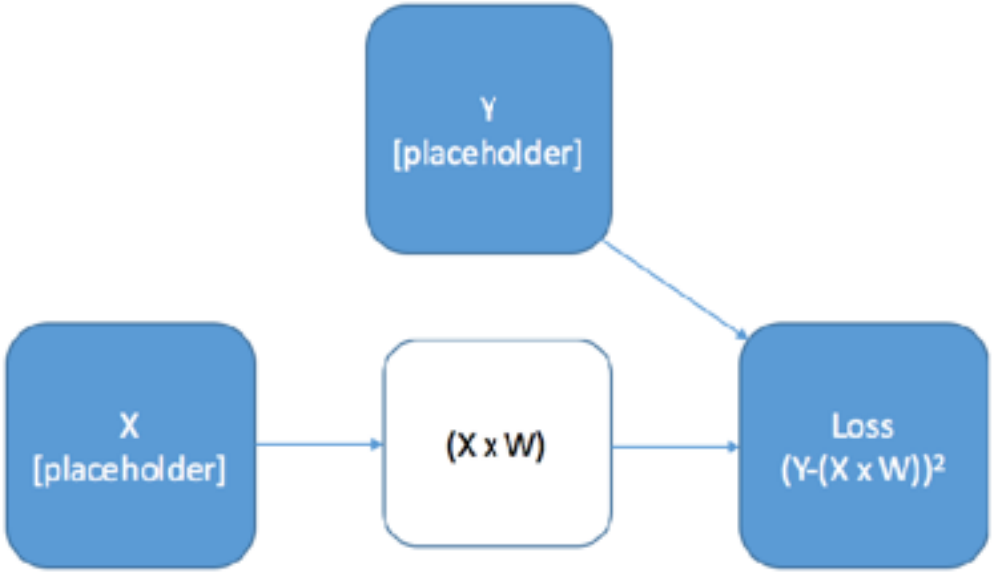
- Nothing evaluated until you define a session and tell it to evaluate it
- Session defines configuration of execution
  - like GPU versus CPU

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

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)
```

1. **Setup** Variables

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

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

```
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())
    sess.run([opt_operation], feed_dict={X: X_data, y: y_data})
```

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})
```

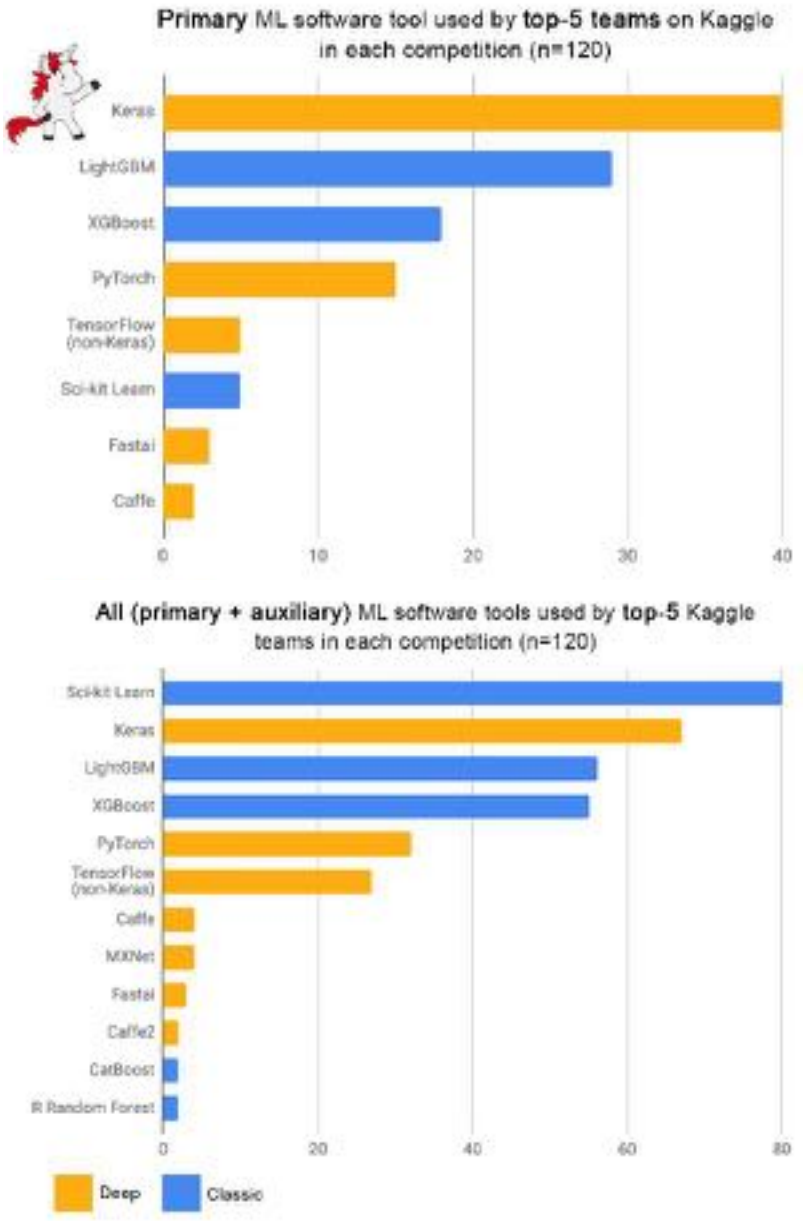
https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

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

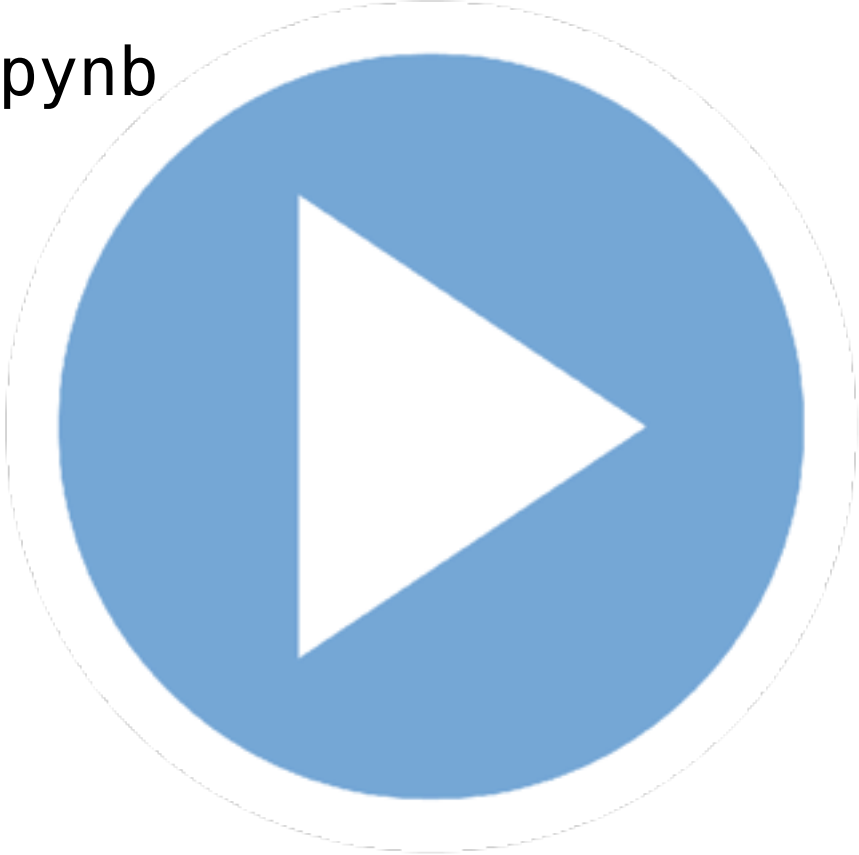
- Keras Sequential API
  - great for simple, feed forward models
- Keras Functional API
  - build models through series of nested functions
  - each “function” represents an operation in the NN
- Keras Classes (Inheritance)
  - good for more advanced functionality



Demo

`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. Recommenda-tions based on memorization are usually more topical and

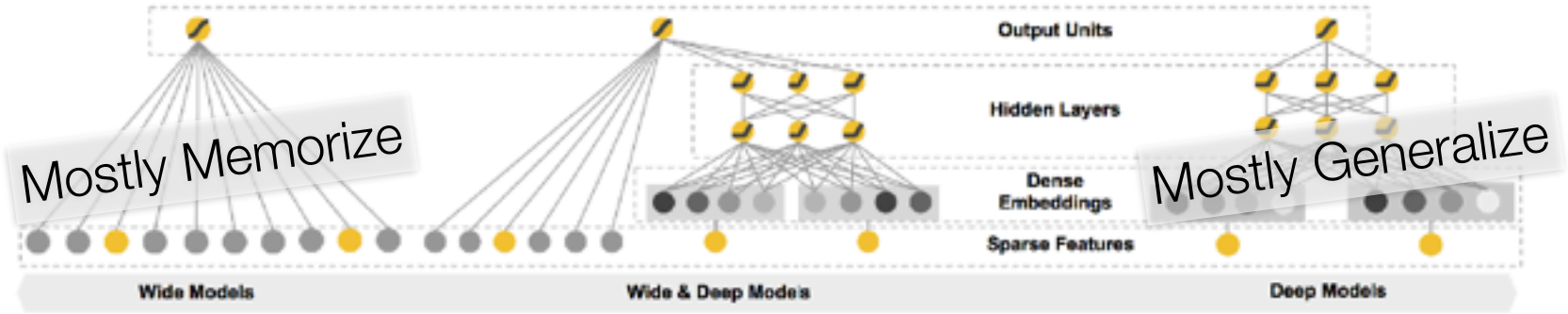
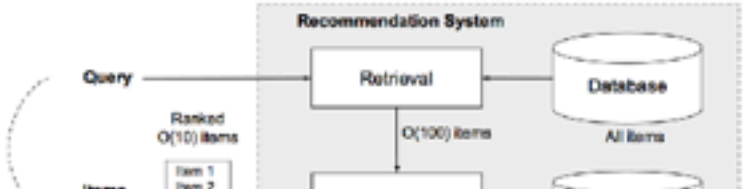


Figure 1: The spectrum of Wide & Deep models.

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

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



Why wide and deep?

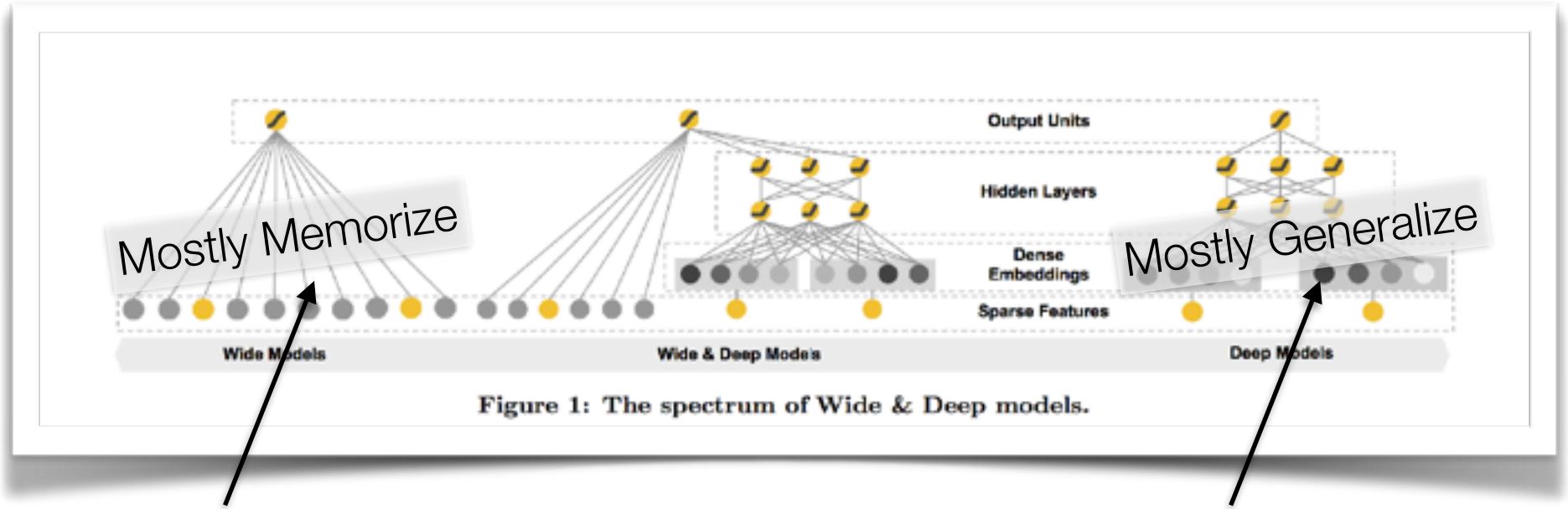


Figure 1: The spectrum of Wide & Deep models.

But why memorize?

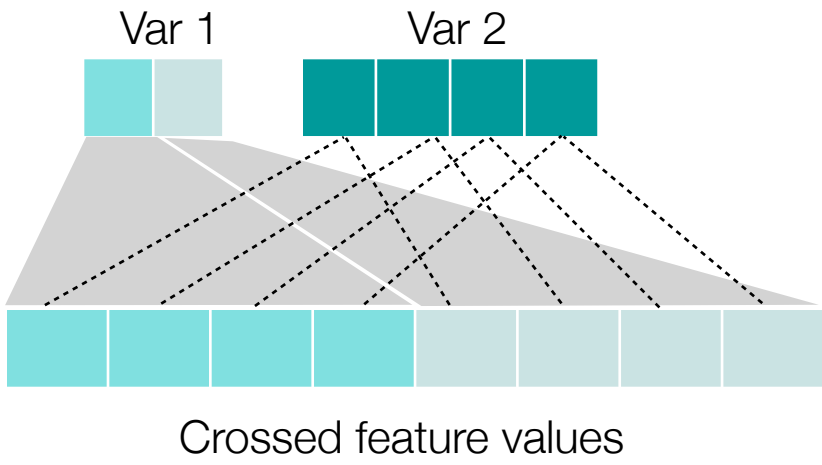
Obvious!

- Categorical values have combinations that repeat!
  - so memorizing these values is not neccessarily 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

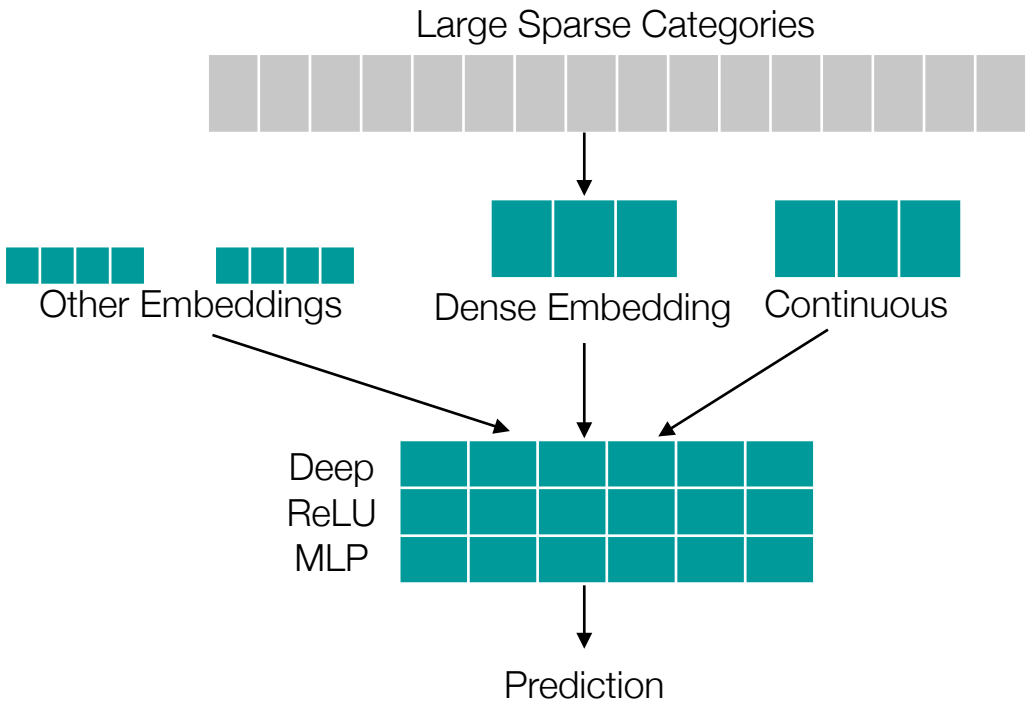


- Crossed feature "Rating-Genre"
  - G-Action, G-Drama, G-Comedy, G-Horror, G-else
  - PG-Action, PG-Drama, PG-Comedy, PG-Horror, G-else
  - and so on … one hot encoded

Sparse Embedding and Deep MLP (Generalize?)

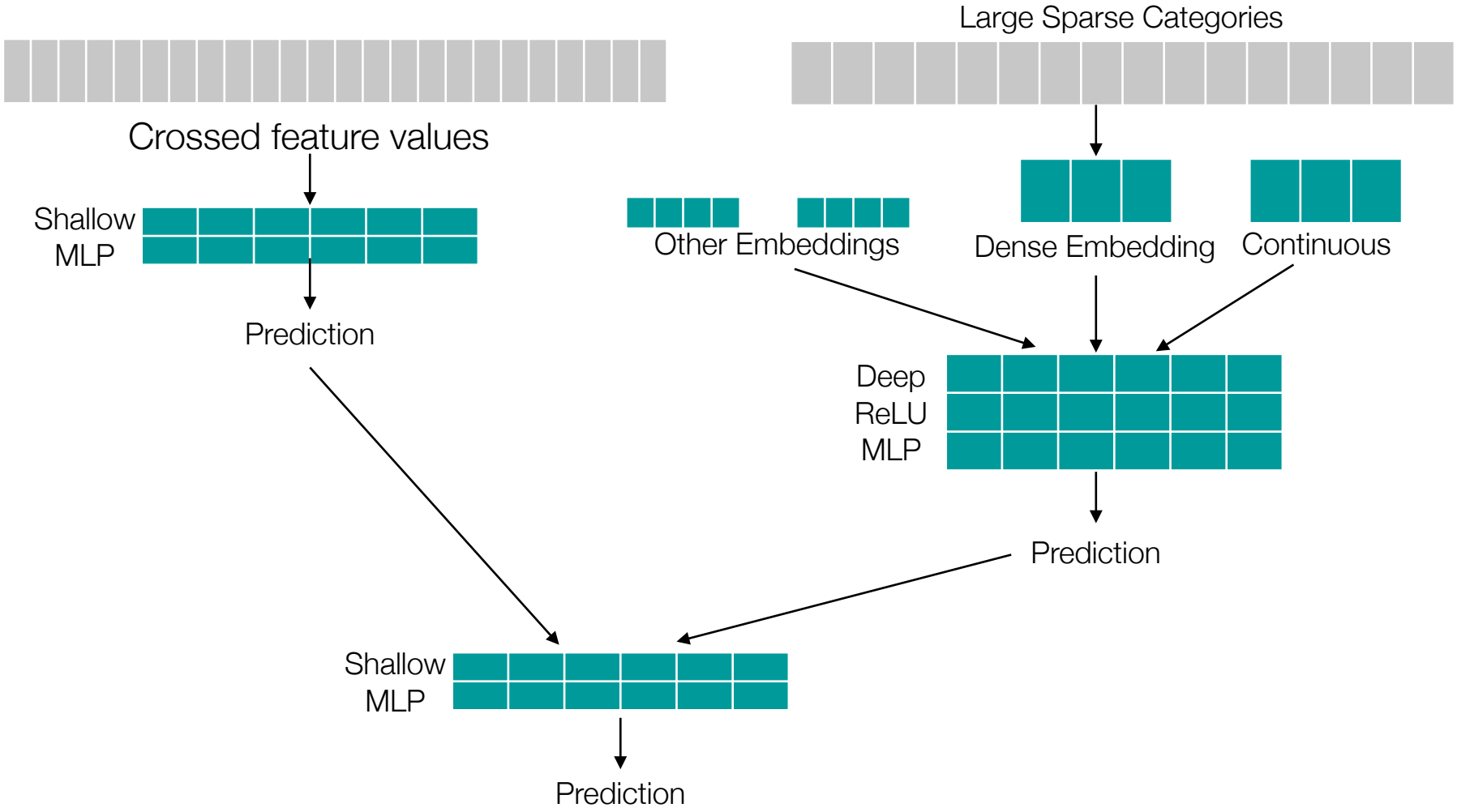
- Deep refers to increasingly smaller hidden layers
- Embed into sparse representations via ReLU

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



Combining Memorization and Generalization

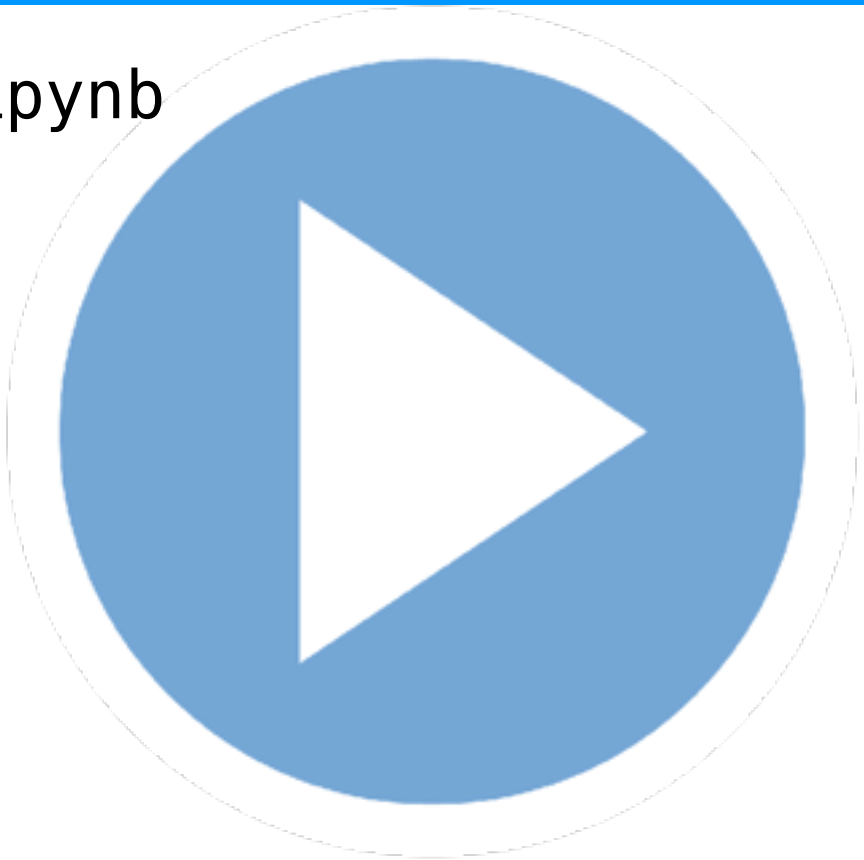
- Deep refers to increasingly smaller hidden layers
- Embed into sparse representations via ReLU



Wide and Deep

Demo

`10. Keras Wide and Deep.ipynb`



The awful dataset:
Toy Census Data Example

Other tutorials:

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

End of Session

- Next Time:
  - **Convolutional Neural Networks**