Lecture Notes for Machine Learning in Python

Professor Eric Larson

Revisiting CV

A “Way too Early” History of Deep Learning

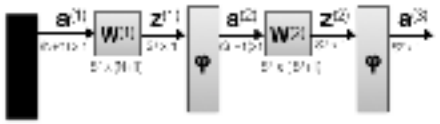
Logistics and Agenda

- Logistics
  - Next time: Wide and Deep Networks (in TF)
- Agenda
  - Cross Validation
  - “Deep Learning” History
  - Remaining Topics
    - TensorFlow from 10,000 feet
    - Wide and Deep Networks
    - Convolution Neural Networks
    - Recurrent Neural Networks

Last time:

Back propagation summary

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



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

1. Forward propagate to get **z**, **a** for all layers
2. Get final layer gradient
3. Update back propagation variables
4. Update each $\mathbf{W}^{(l)}$

for each $\mathbf{y}^{(i)}$

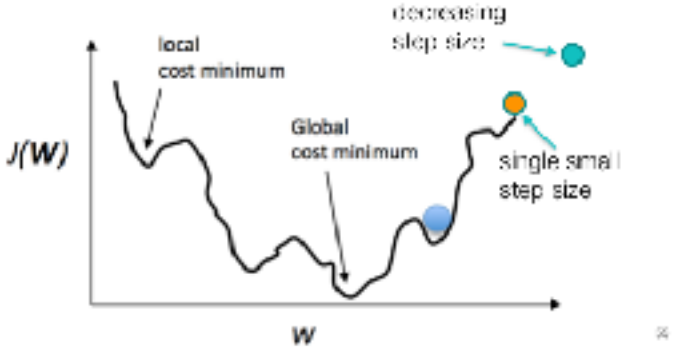
$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(2)}} = -2(\mathbf{y}^{(i)} - \mathbf{a}^{(3)}) * \mathbf{a}^{(3)} * (1 - \mathbf{a}^{(3)})$$

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

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

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**

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

speeds up initial training

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

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

```
# 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

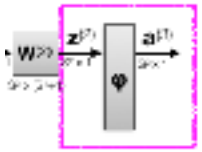
```
# vectorized backpropagation
sigma3 = -2*(Y_enc-A3)*A3*(1-A3)
sigma2 = (W2.T @ sigma3)*A2*(1-A2)

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

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

bp~5

Practical Implementation of Architectures

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

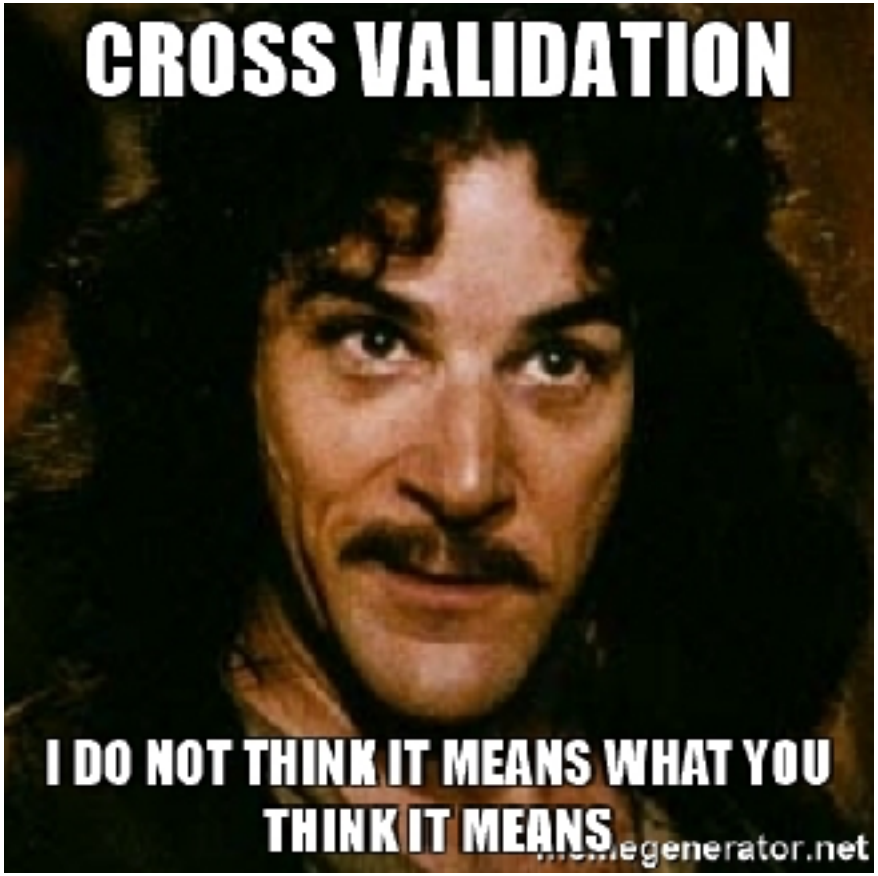


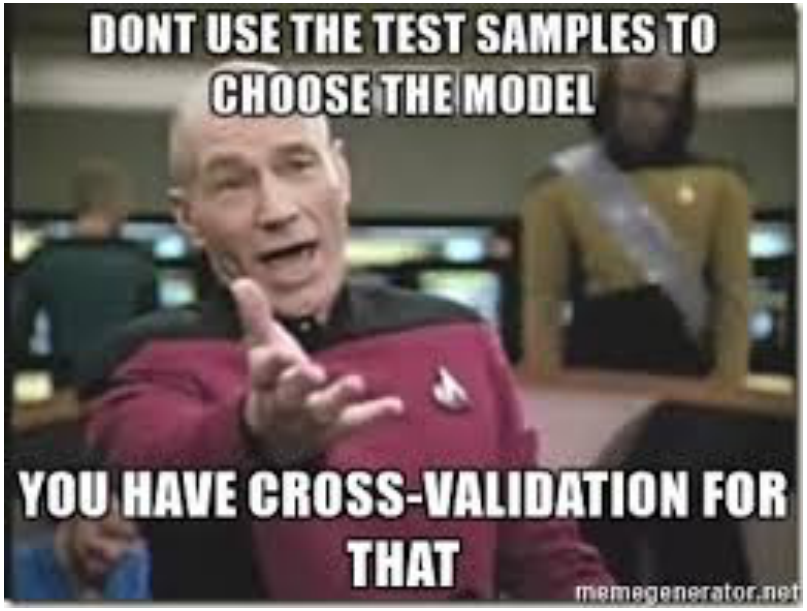
$$\phi(\mathbf{z}^{(i)}) = \begin{cases} \mathbf{z}^{(i)}, \text{ if } \mathbf{z}^{(i)} > 0 \\ \mathbf{0}, \text{ else} \end{cases}$$

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

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

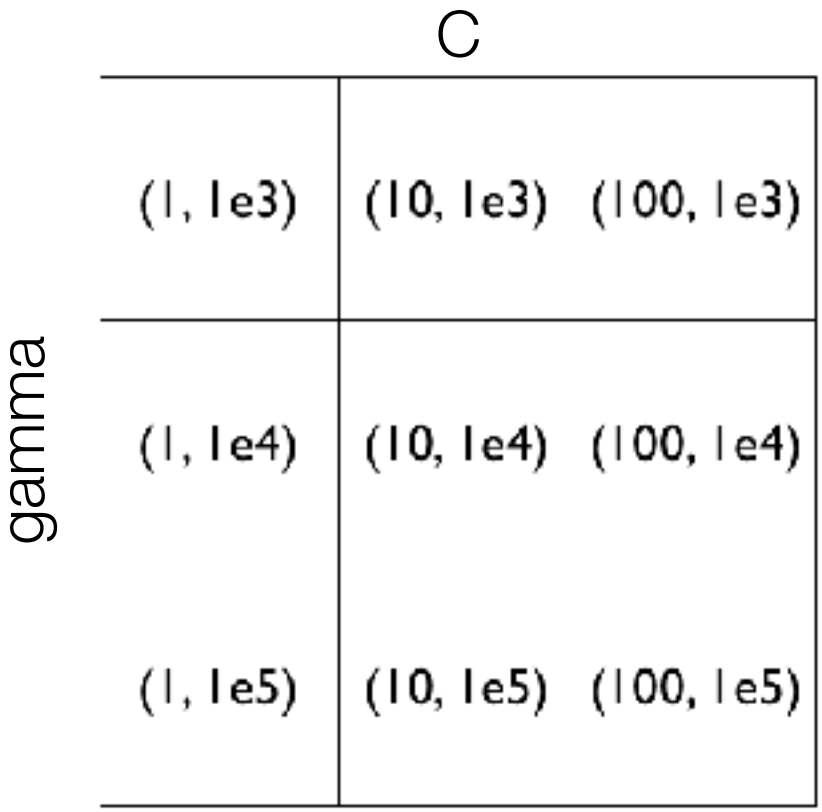
Revisiting Cross Validation





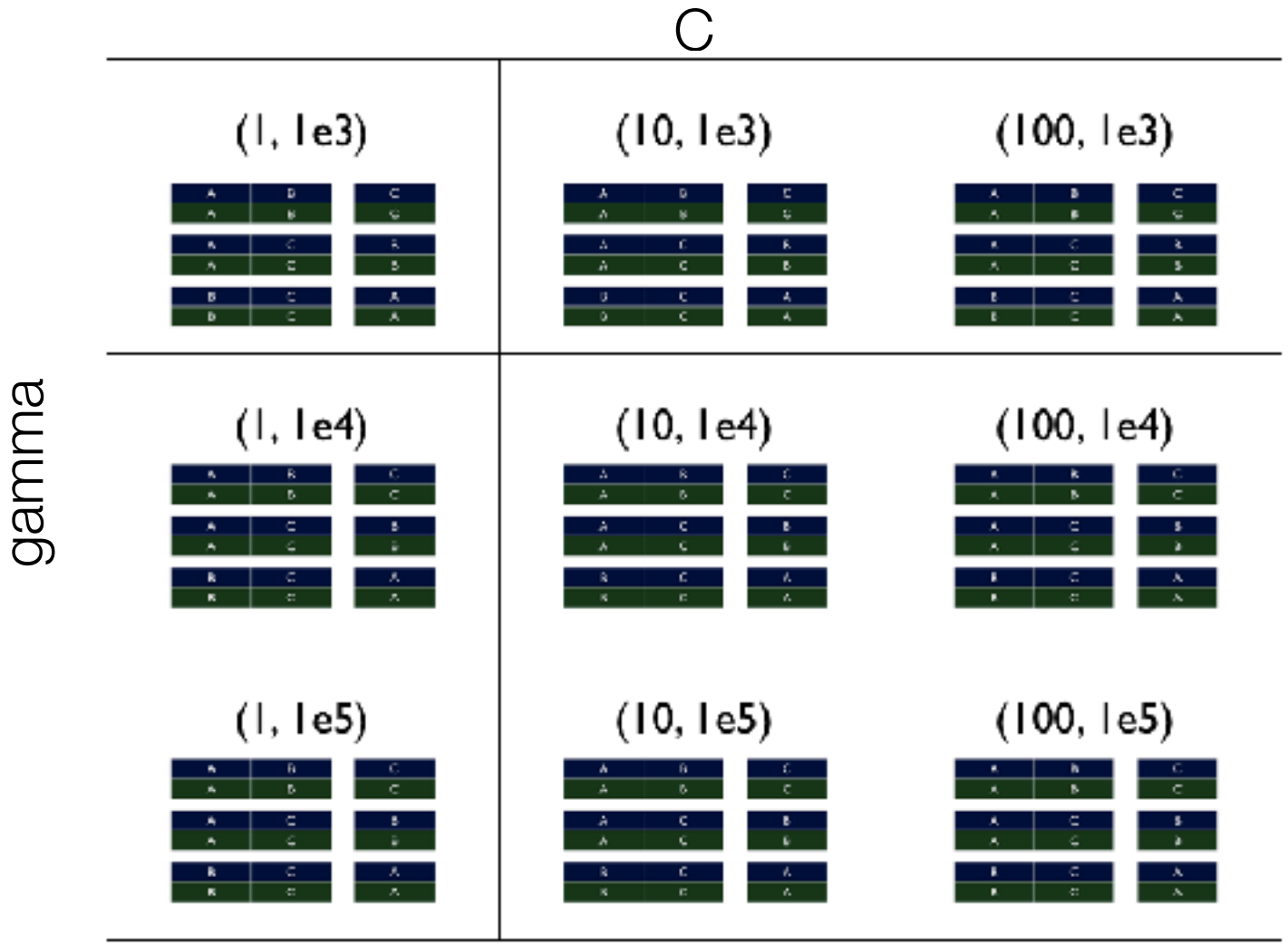
Grid Searching

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



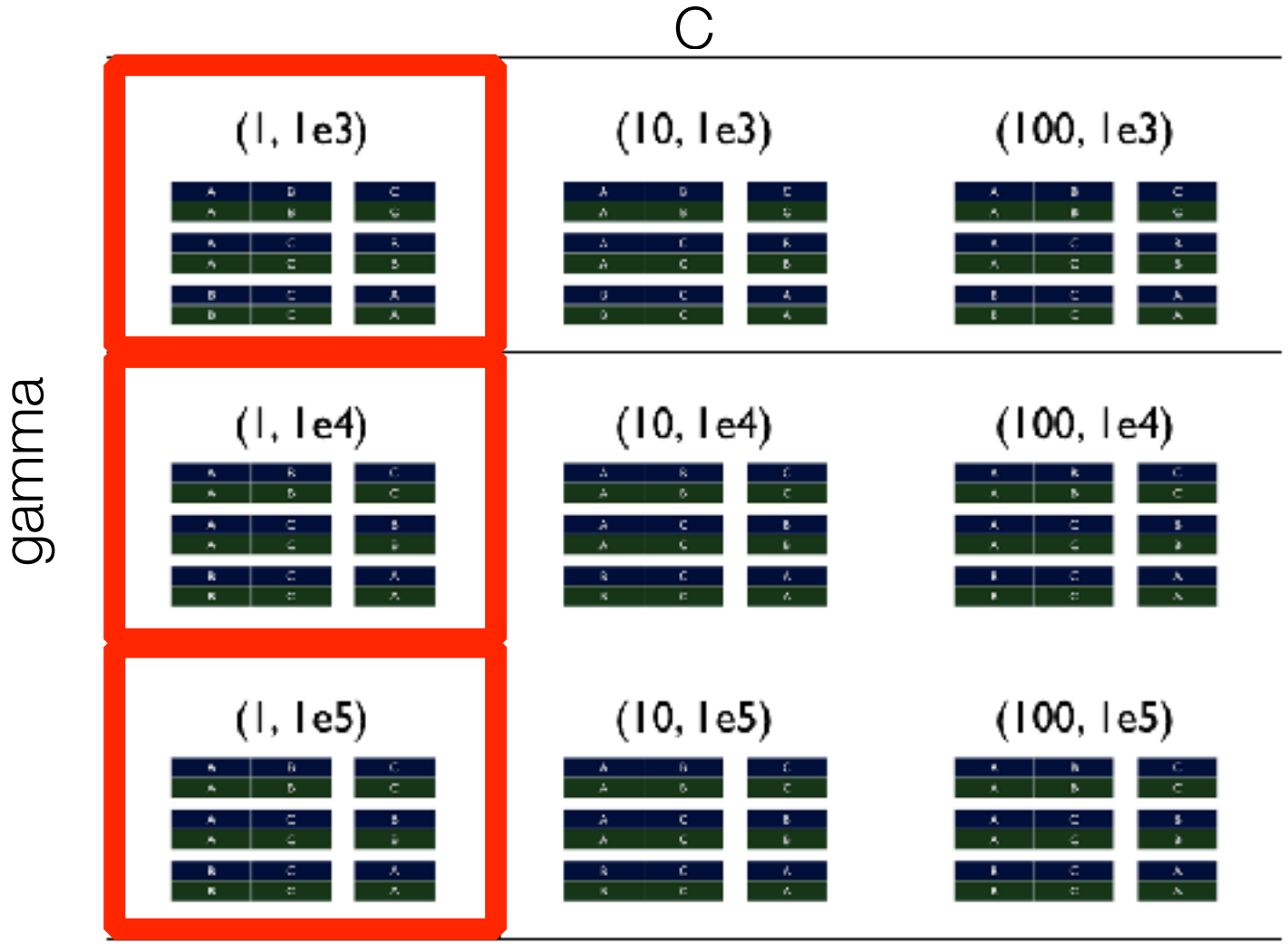
Grid Searching

- For each value, want to run cross validation…



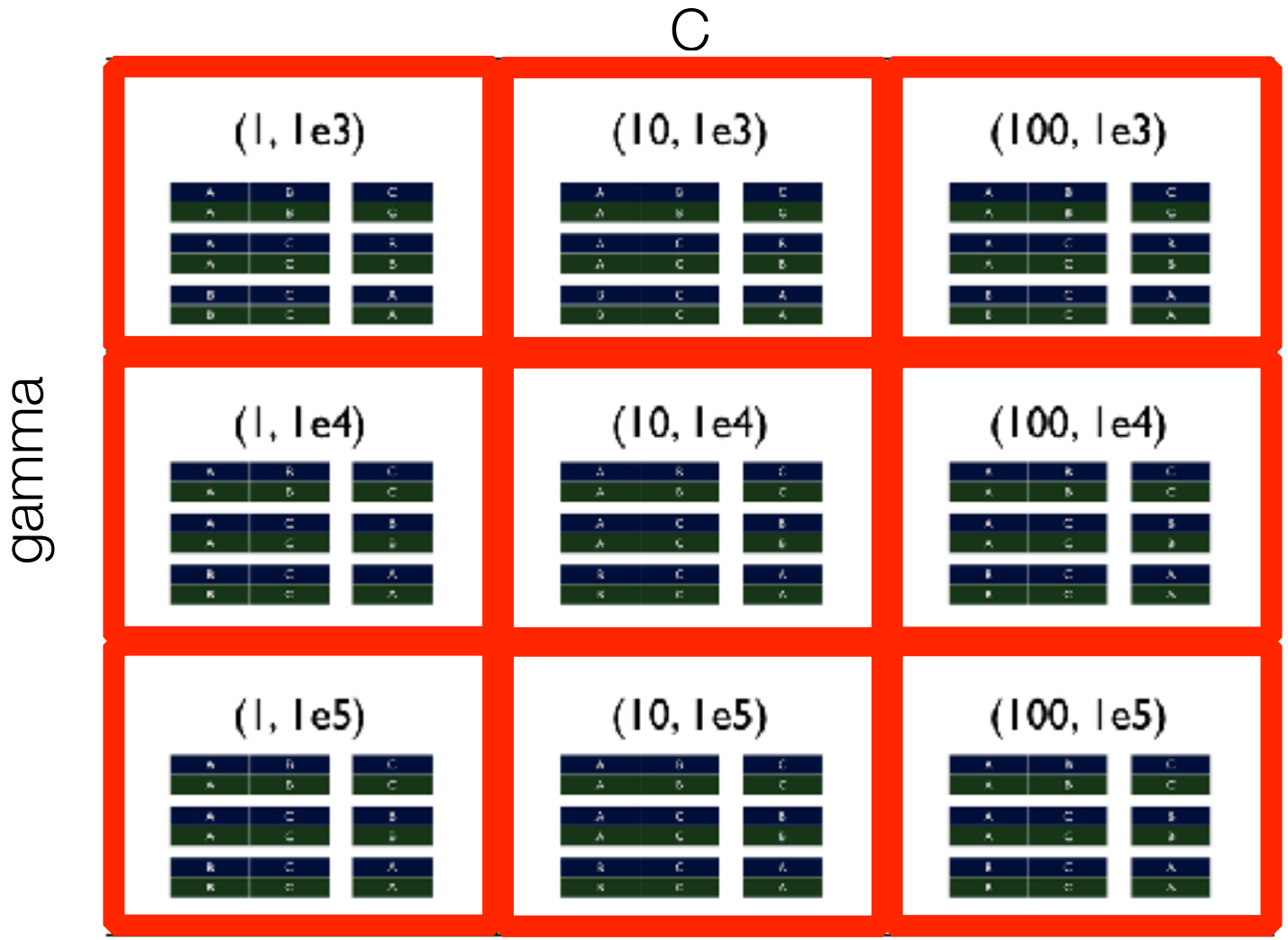
Grid Searching

- Could perform iteratively

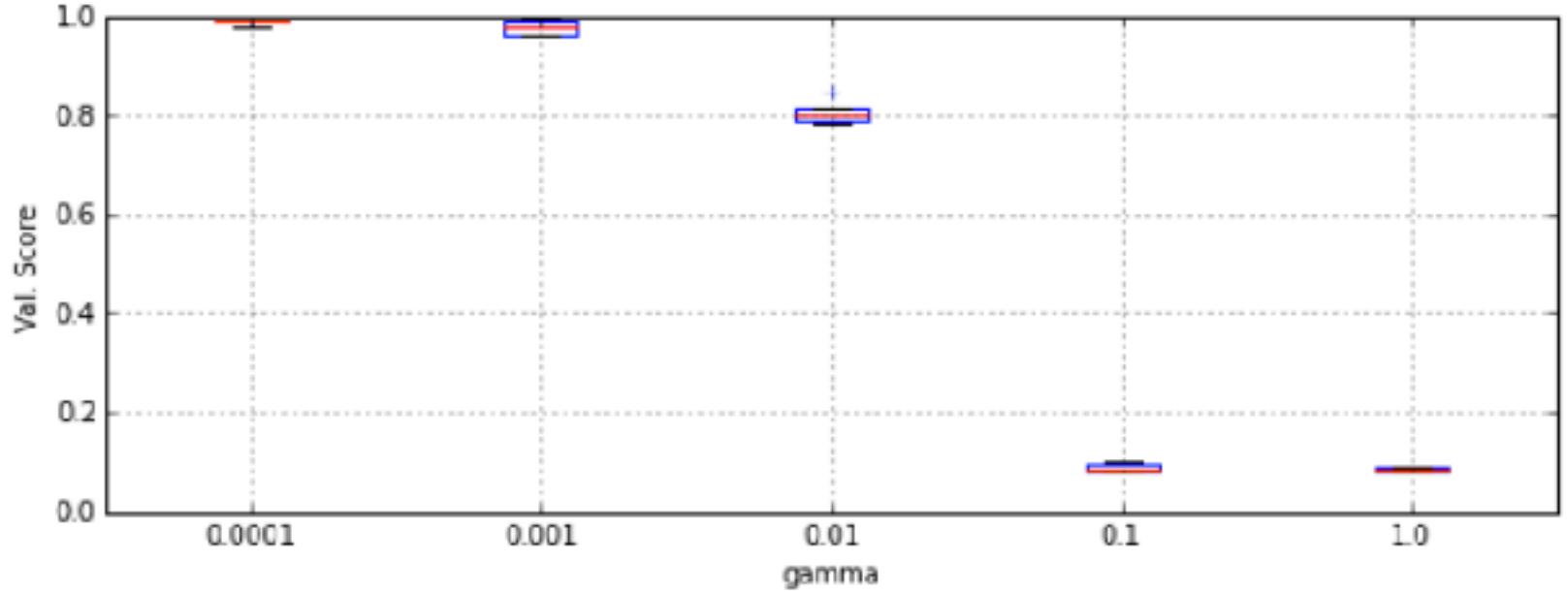


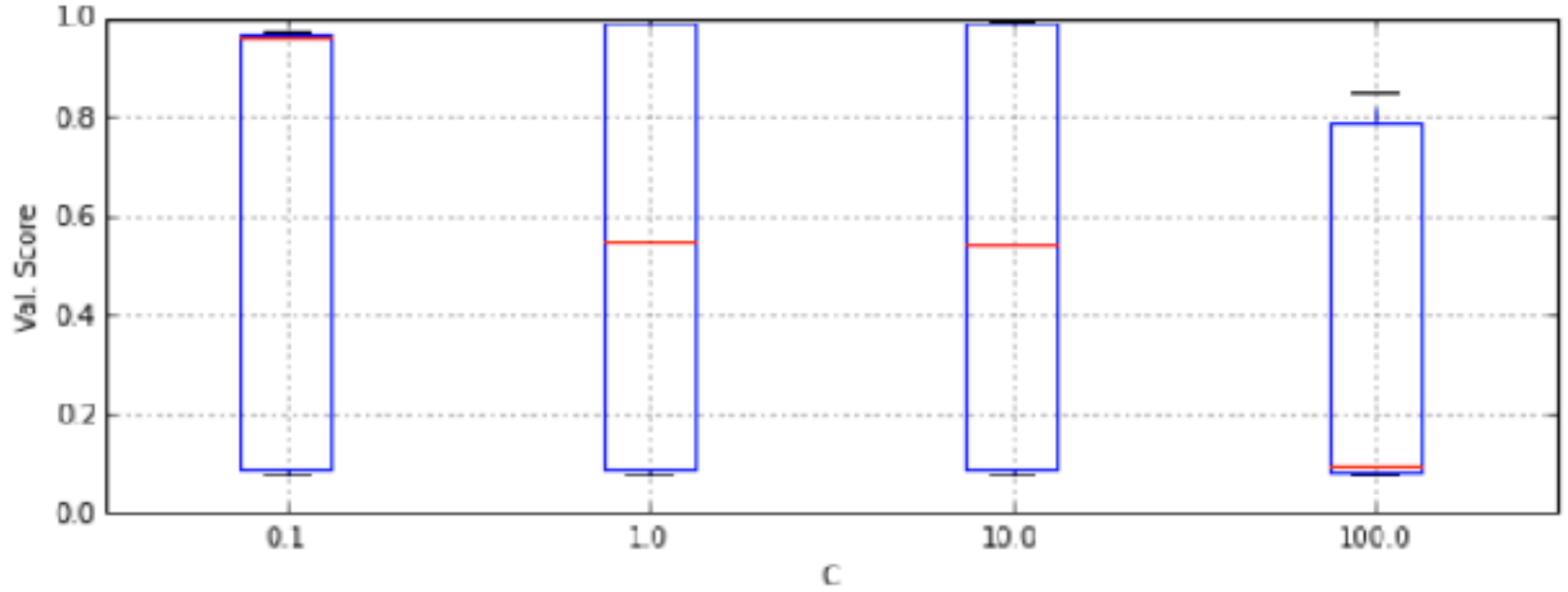
Grid Searching

- or at random…



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

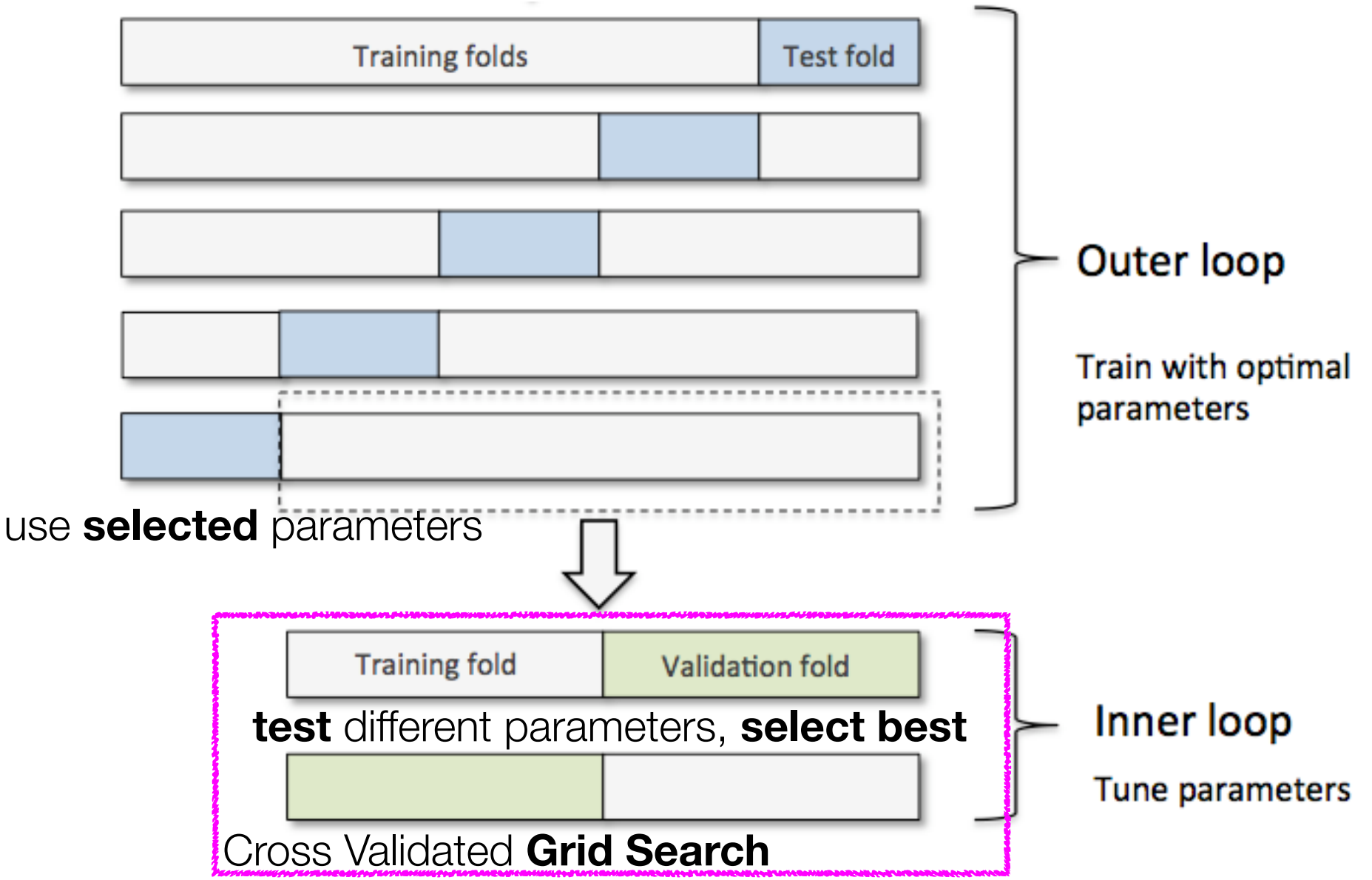




The Problem

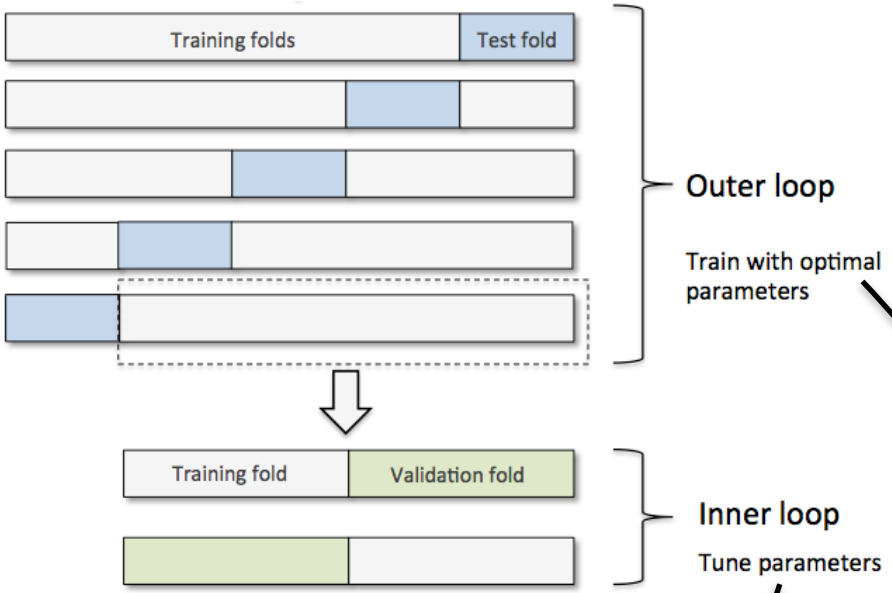
- Using the grid search parameters and testing on the same set…
  - the **performance on the dataset** could now be **biased**
  - cannot realistically determine the **expected performance** on new data

Solution: Nested Cross Validation



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

Nested Cross Validation: Hyper-parameters



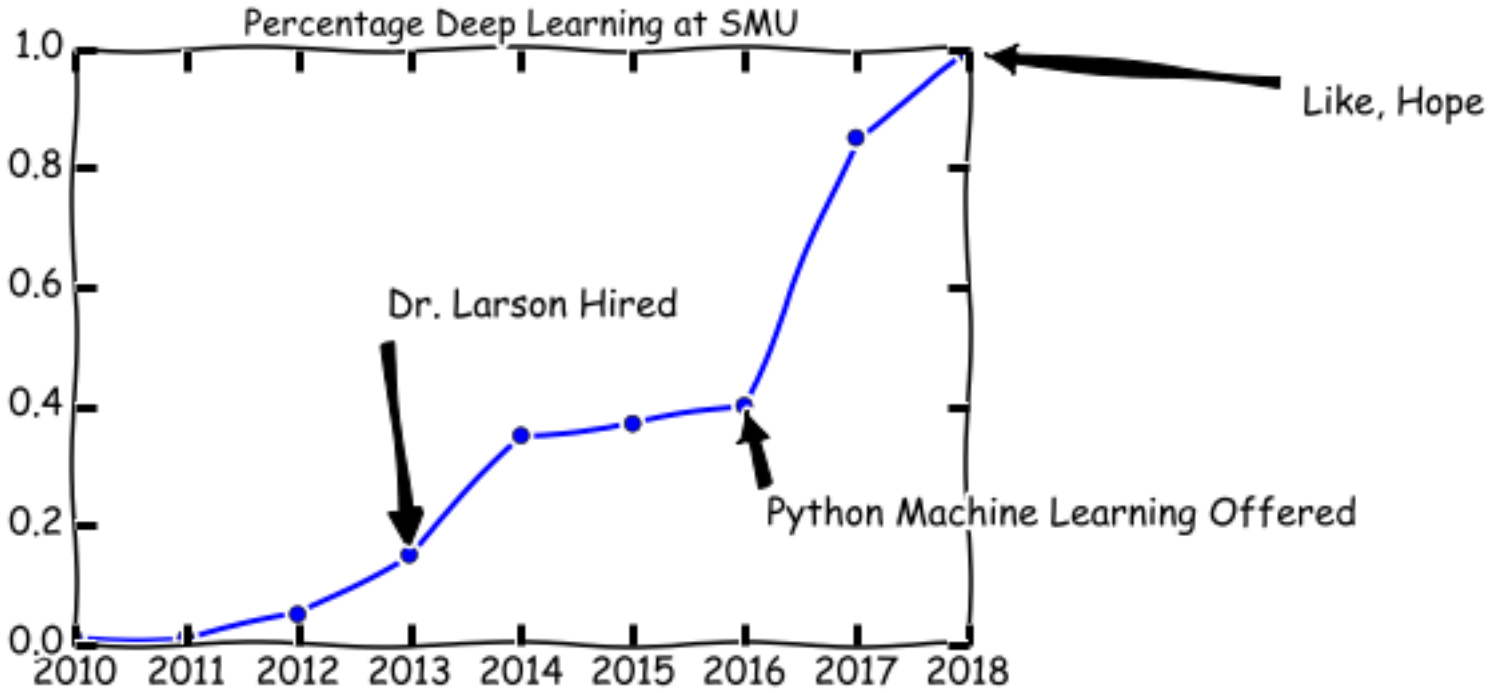

```
gs = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=2)

# 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

Some History of Deep Learning

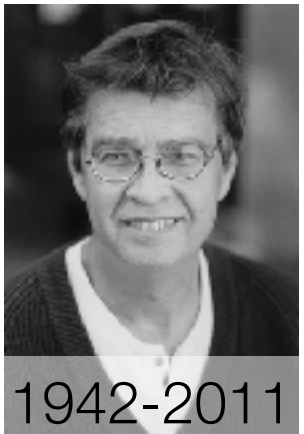



```
# Data to plot
dates = range(2010,2019)
percents = [0.01, 0.01, 0.05, 0.15, 0.35, 0.37, 0.40, 0.85, 0.99]

# Set the style to XKCD
plt.xkcd()

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

Neural Networks: Where we left it

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

David Rumelhart



1942-2011

Geoffrey Hinton

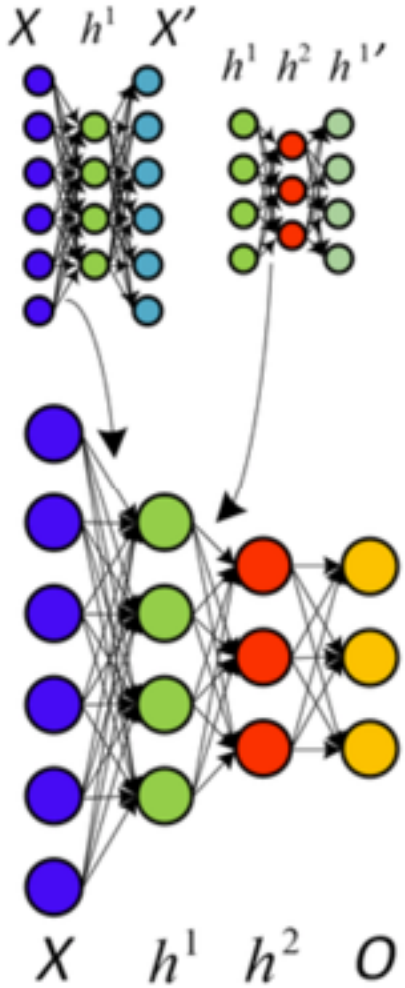
History of Deep Learning: Winter

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

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

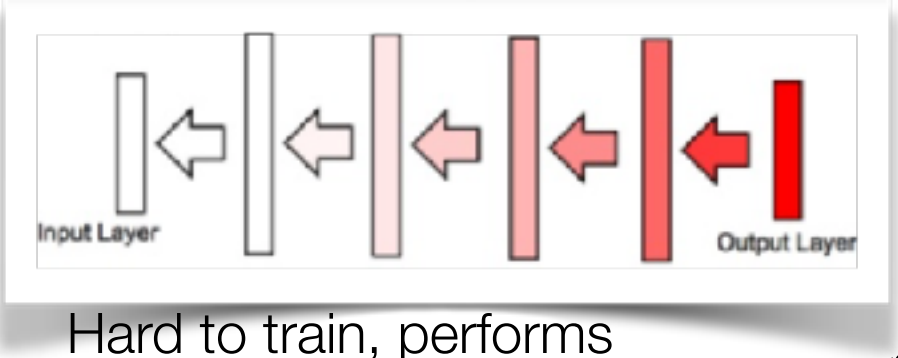
History of Deep Learning: Winter

- Winter is coming:



Easy to train, performs on par with other methods

memorization



Hard to train, performs worse than other methods

generalization

Researcher have difficulty reconciling expressiveness with performance

~chance (untrainable)

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

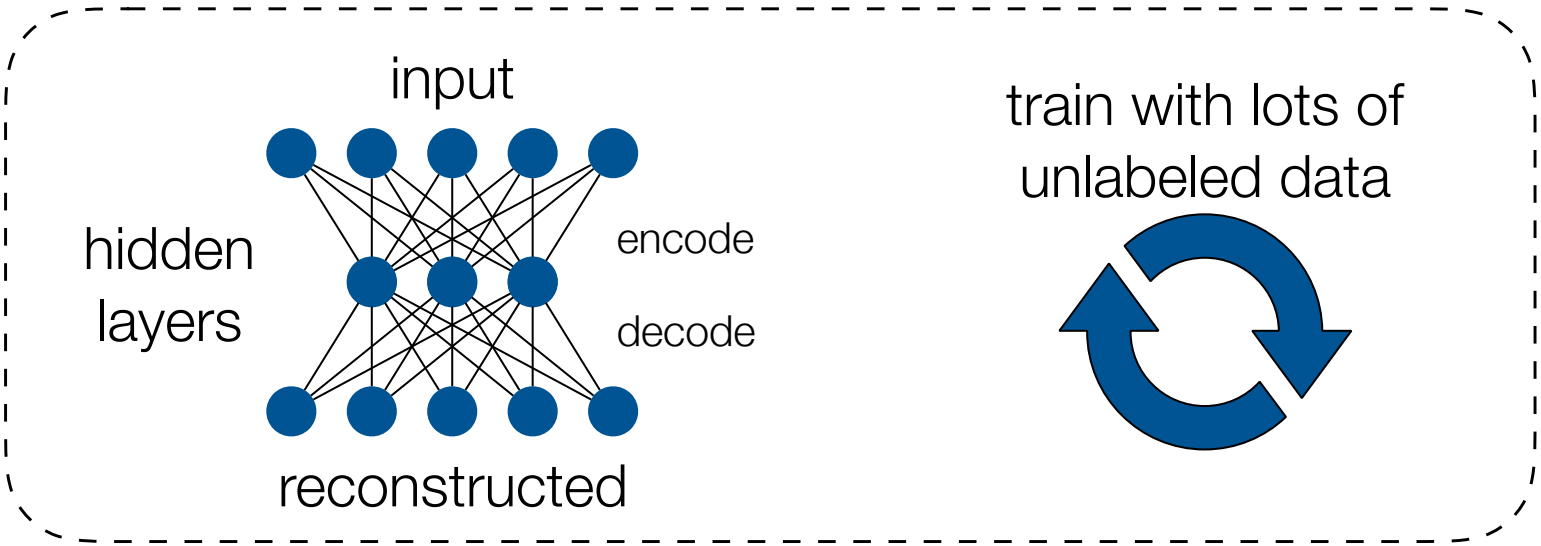
History of Deep Learning

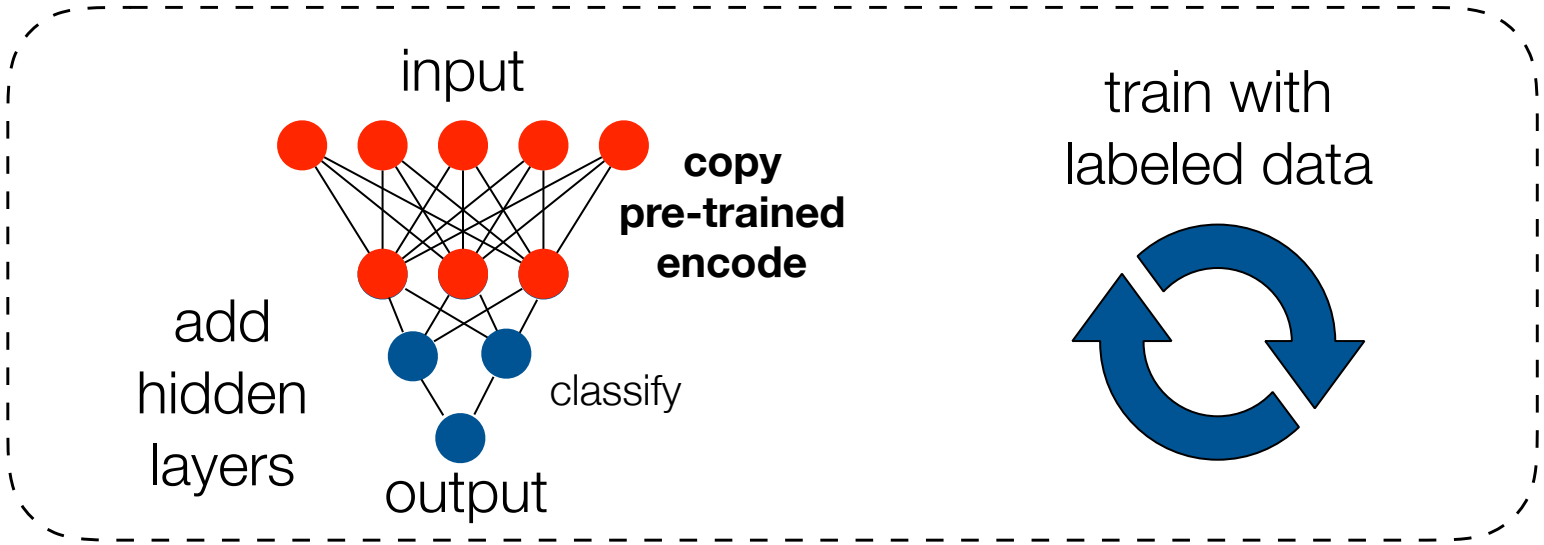
- 2004: Hinton secures funding from CIFAR based on his reputation
  - *eventually*: Canada would be savior for NN
  - Hinton rebrands: Deep Learning
- 2006: Hinton publishes paper on using pre-training and Restricted Boltzmann Machines
- 2007: Another paper: Deep networks are more efficient when pre-trained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

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

Pre-training: still in the long winter

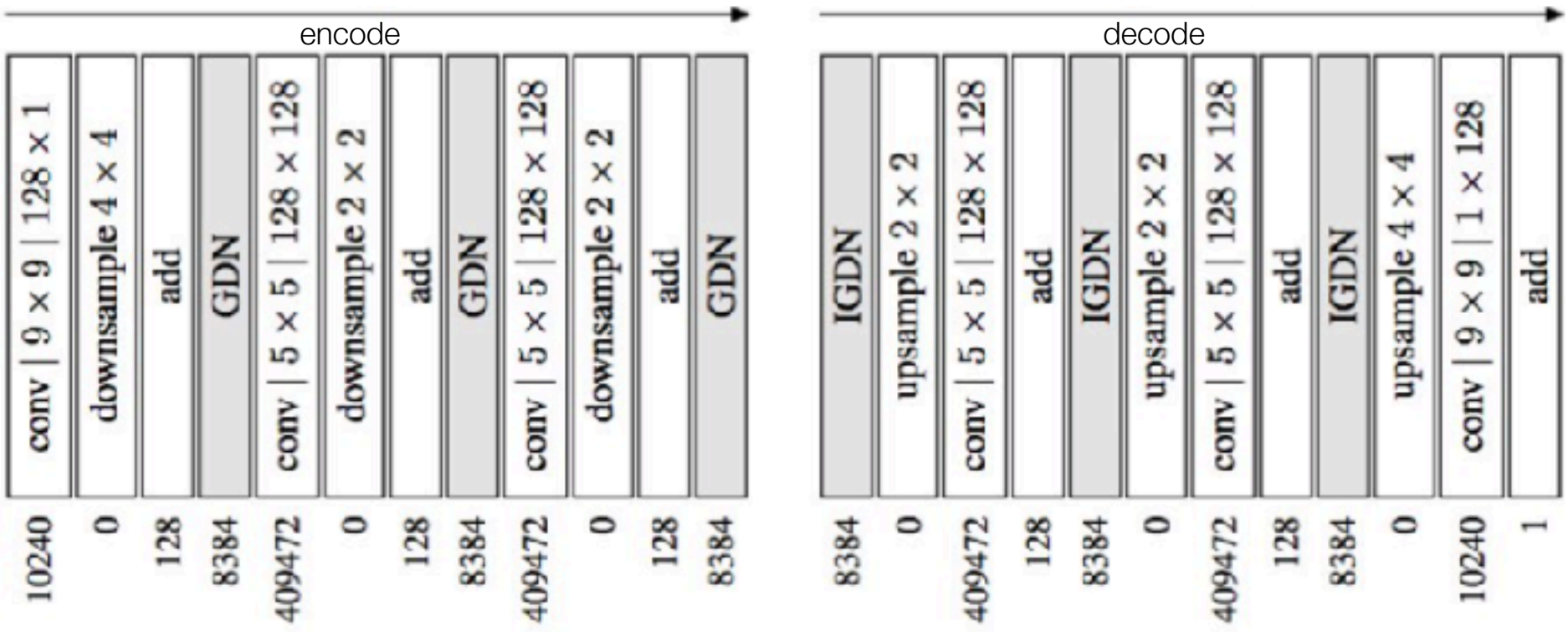
- auto-encoding



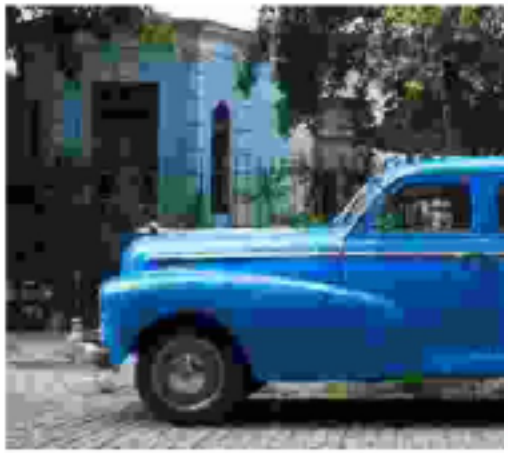


Pre-training: modern example

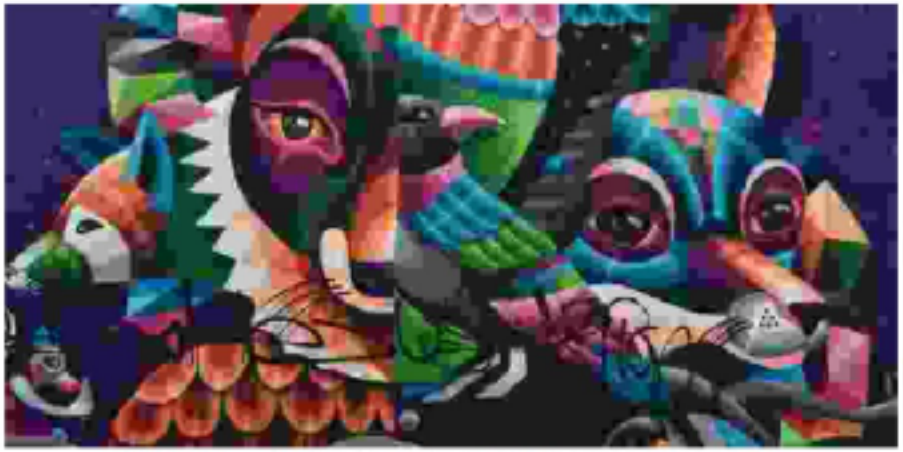
- auto-encoding: example from paper published Nov 5, 2016



https://arxiv.org/abs/1611.01704



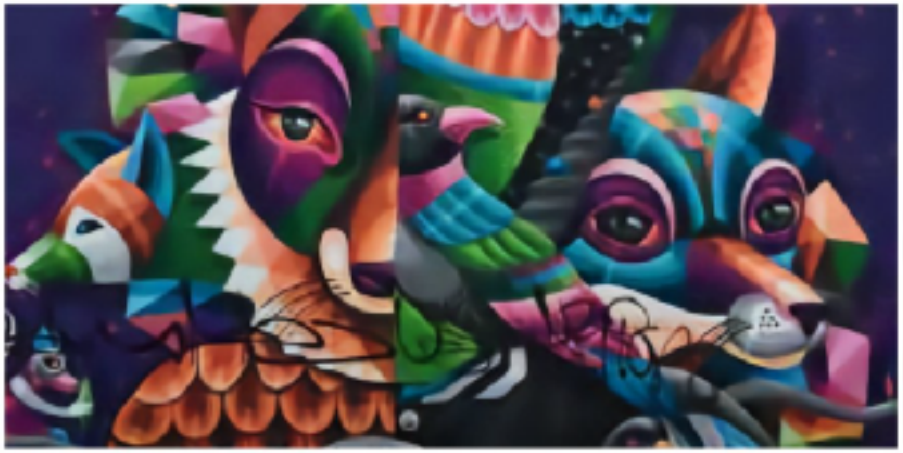
JPEG, 6006 bytes (0.170 bit/px), RMSE: 19.75



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

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

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



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

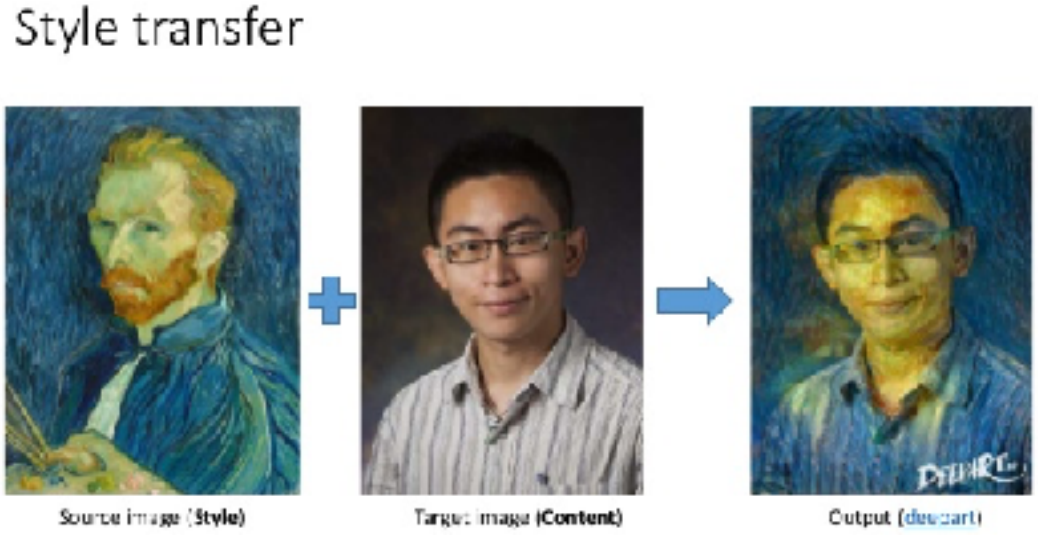
it/px), RMSE: 6.10/5.09, PSNR: 32.43 dB/34.00 dB

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

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

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

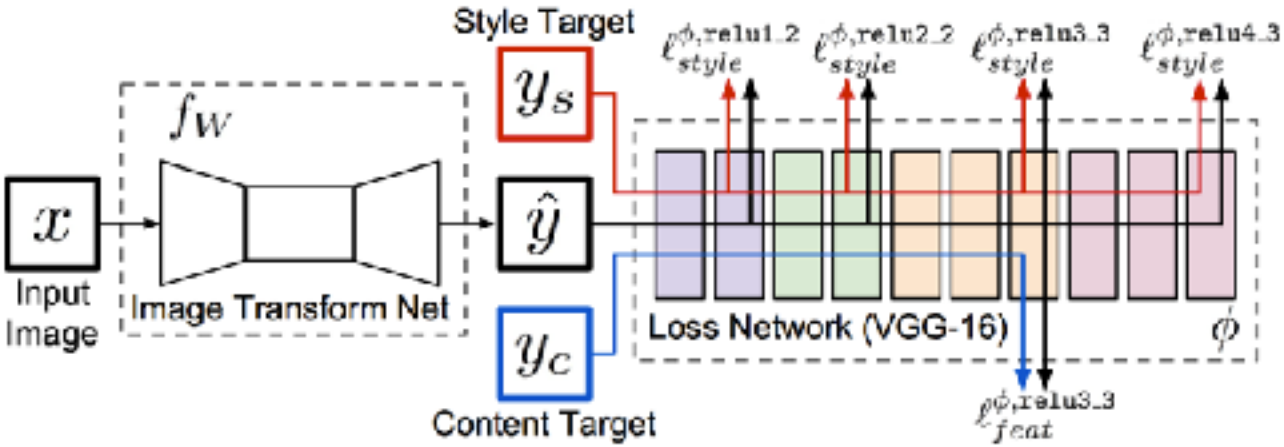
Getting to basic descriptors of images



Gatys *et al.* 2015

Wood, Ledford, Pop 2017

Johnson Alahi, Fei-Fei, 2016



History of Deep Learning

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

Research at Google

Navdeep Jaitly
Google Brain Team

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

George Dahl
Google Brain Team

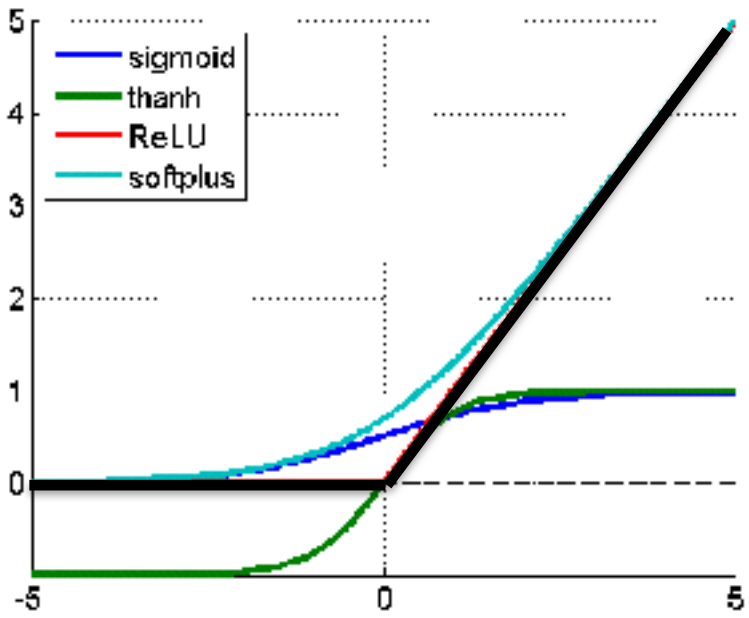
Abdel-rahman Mohamed
Microsoft Research
Redmond, Washington | Computer Software

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

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

History of Deep Learning

- 2011: 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



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

that's a really easy gradient to compute!

…and it makes the weights more sparse

…and helps to solve the vanishing gradient problem

…and its inspired by biological vision community

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

History of Deep Learning

- ReLU not the only way to do it!

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

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

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

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

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

Batch Normalization

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

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

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

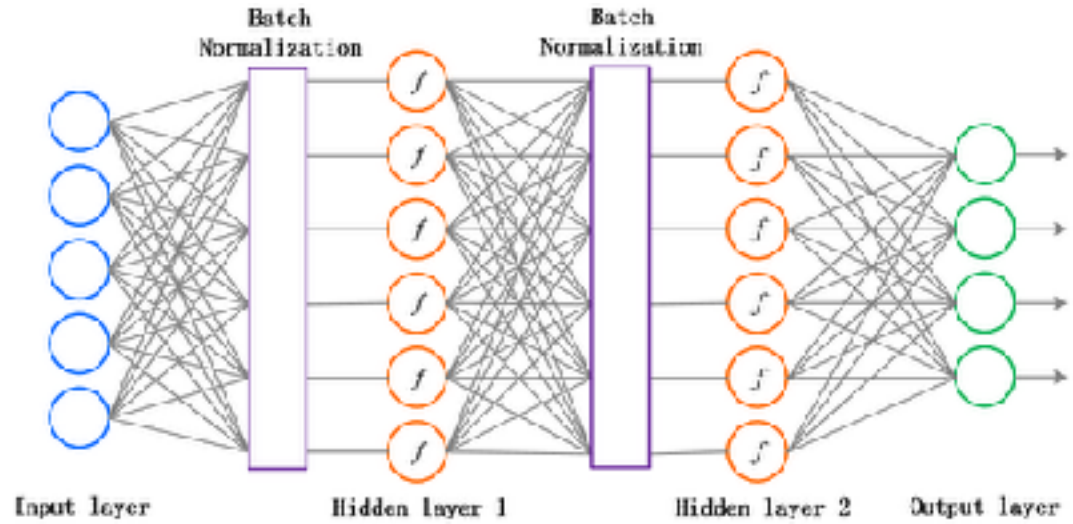
$$\frac{\partial \ell}{\partial \sigma^2_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma^2_{\mathcal{B}} + \epsilon)^{-3/2}$$

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

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

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

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



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

History of Deep Learning

- ReLU not the only way to do it!

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

2 SWISH SiLU

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

$$f(x) = x \cdot \sigma(x) \tag{1}$$

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

*Work done as a member of the Google Brain Residency program (g.co/brainresidency).

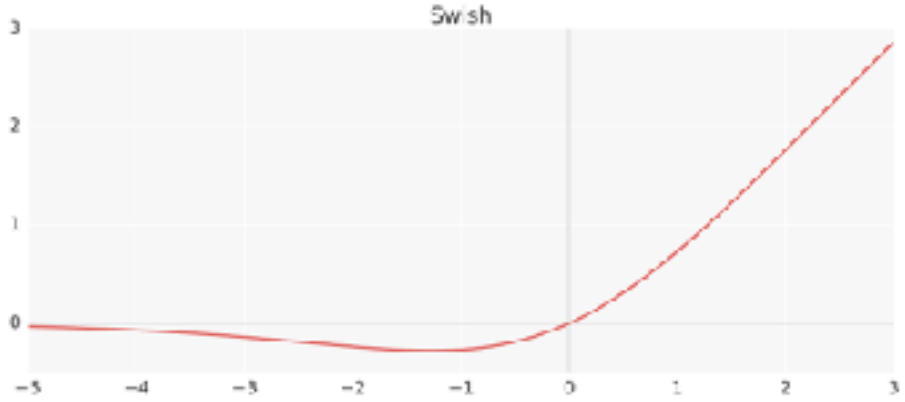
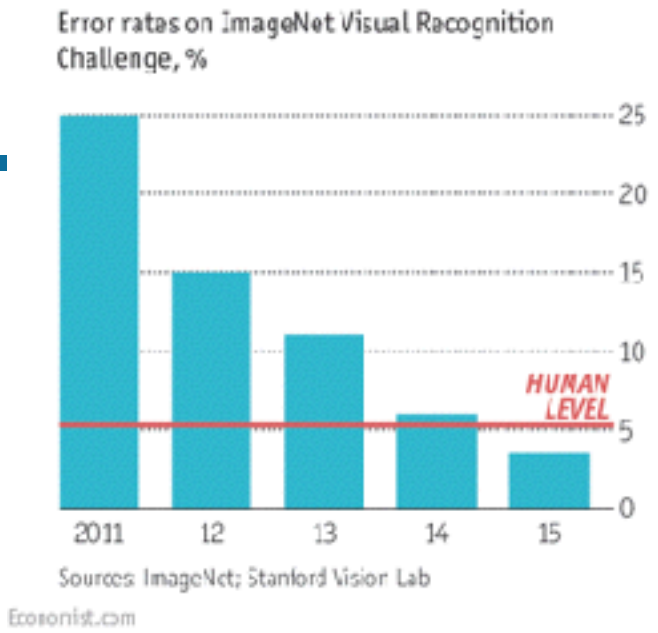


Figure 1: The Swish activation function.

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

History of Deep Learning



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

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

History of Deep Learning

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

Deep Neural Networks for Acoustic Modeling in Speech Recognition

[The shared views of four research groups]

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

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

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

History of Deep Learning

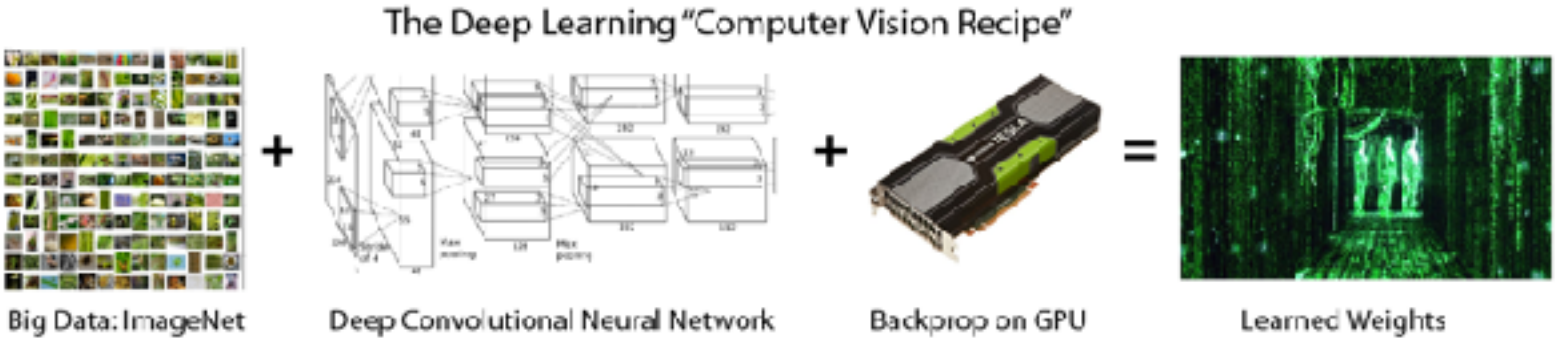
- 2013: Ng and Google (founded BrainTeam)
  - run unsupervised feature creation on YouTube videos (becomes computer vision benchmark)

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

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

History of Deep Learning

- **Hinton** summarized what we learned in deep learning from the 2006 to present. Where we went wrong before present day:
  - labeled dataset were 1000s of times too small
  - computers were millions of times too slow
  - weights were initialized in stupid ways
  - we used the wrong non-linearities
- Or **Larson's Law**:
  - use a GPU when possible, init weights for consistent gradient magnitude, ReLU/BN/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/

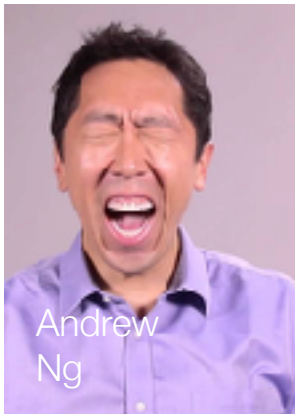
A summary of the Deep Learning people:

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

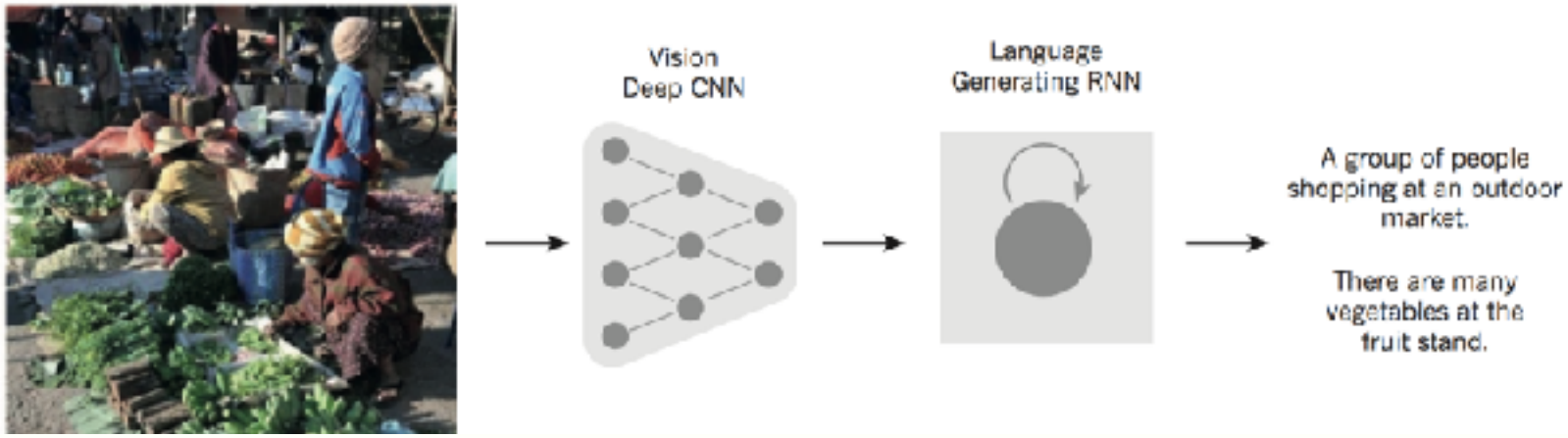
doi:10.1038/nature14539

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

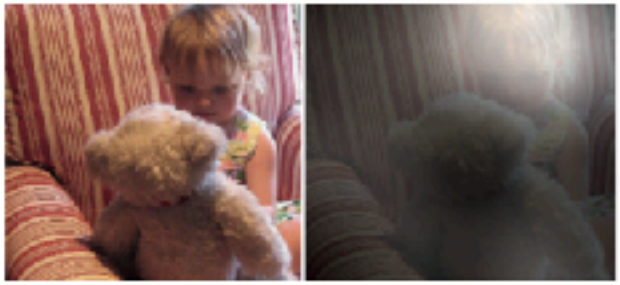
Famous examples:



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

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

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

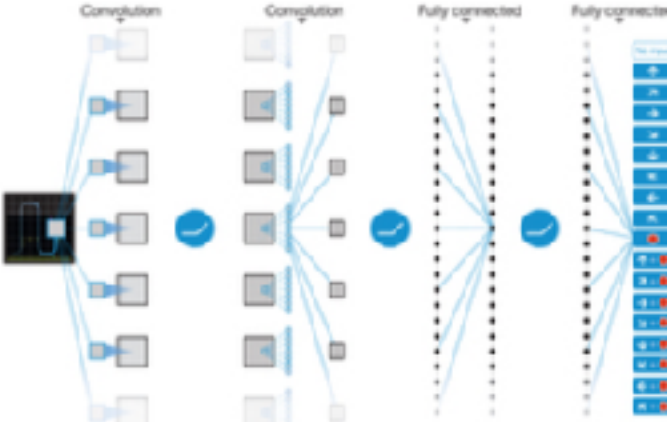


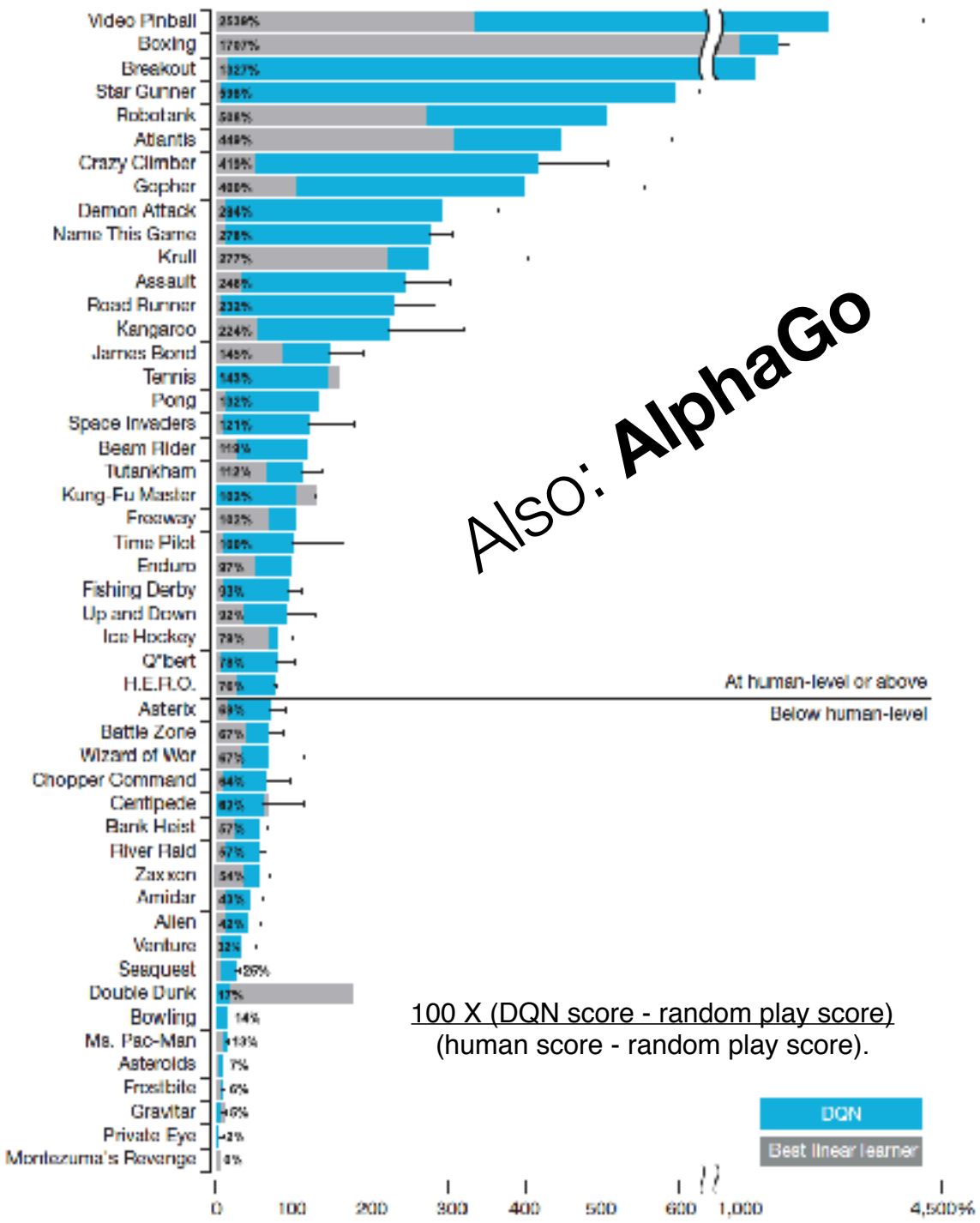
A little **girl** sitting on a bed with a teddy bear.

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

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

More Famous examples: using reinforcement learning





Also: **AlphaGo**

End of Session

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

End of Session

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

Lecture Notes for Machine Learning in Python

Professor Eric Larson

TensorFlow via Wide and Deep Networks

Lecture Agenda

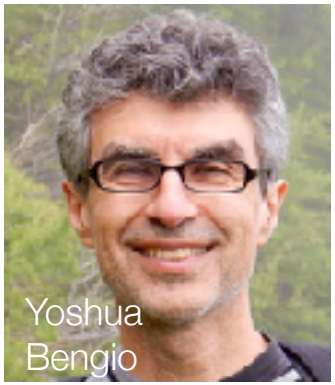
- Introduction to TensorFlow
  - Tensors, Namespaces, Numerical methods
  - Deep APIs
- Wide and Deep Networks

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 (not called neural networks)
- 2006: Auto-encoding and Restricted Boltzmann Machines
- 2007: Deep networks are more efficient when pre-trained
  - RBMs not really the important part
- 2009: GPUs decrease training time by 70 fold…
- 2010: Hinton's students go to internships with Microsoft, Google, and IBM, making their speech recognition systems faster, more accurate and deployed in only 3 months…
- 2012: Hinton Lab, Google, IBM, and Microsoft jointly publish paper, popularity sky-rockets for deep learning methods
- 2011-2013: Ng and Google run unsupervised feature creation on YouTube videos (becomes computer vision benchmark)
- 2012+: Pre-training is not actually needed, just solutions for vanishing gradients (like ReLU, BP, Swish)

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

What else did we talk about last time?



Stayed at
University, Montreal
Advises IBM

Heads
Facebook
AI Team,
NYU

Toronto
GoogleBrain

Chief Scientist, AI/ML
Google Cloud

Stanford
Founded Coursera
MacArthur Genius

Stanford
Co-Founded Coursera
Baidu, GoogleBrain

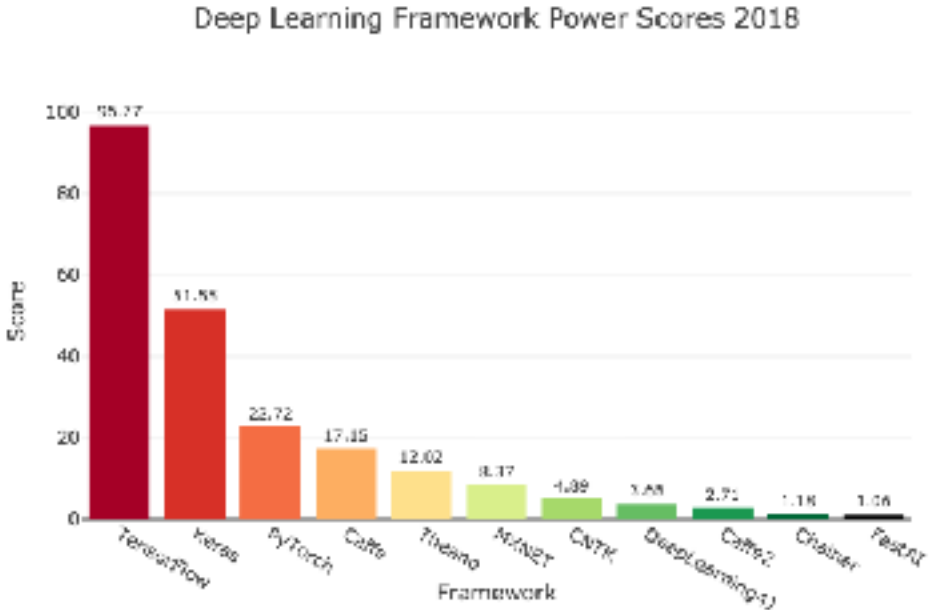
TensorFlow

Matthew Rocklin @mrocklin · Apr 5

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

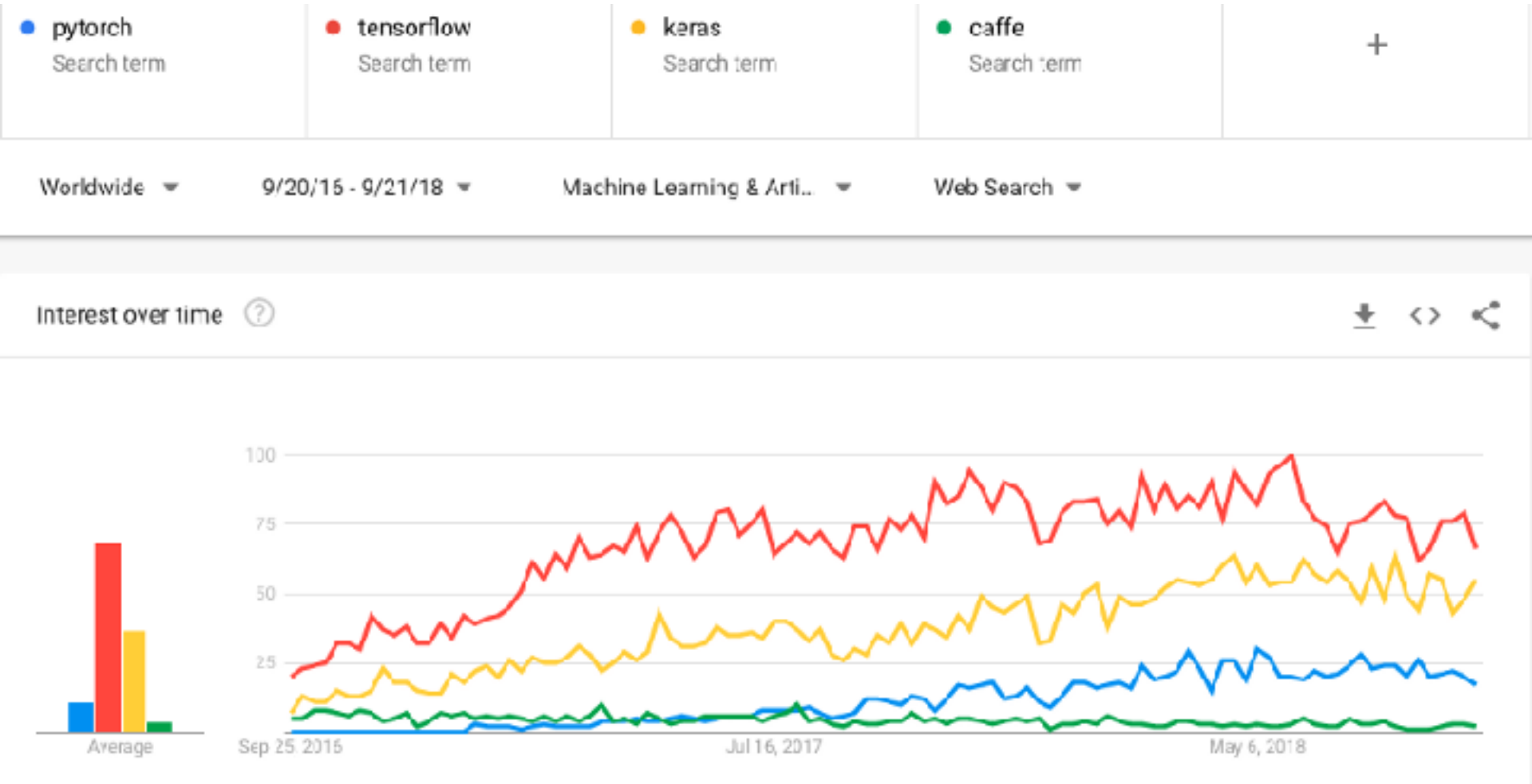
Options for Deep Learning Toolkits

- Caffe/Caffe2
- TensorFlow
- PyTorch
- MxNet
- CNTK
- [Theano]
- DL4J
- 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	552	29.90%	73	6,200	202	3,700	109,578	8,334	67,551	1,642
Keras	884	364	449	695	177	22.20%	53	9,120	79	1,390	33,558	1,647	12,655	719
PyTorch	486	309	428	885	120	6.40%	19	1,780	18	1,560	18,718	952	4,474	760
Caffe	607	399	615	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	423	17	652	8,477	585	2,447	328
MXNET	266	154	200	298	29	1.50%	2	524	32	260	15,200	1,170	5,496	687
CNTK	126	96	97	160	12	3.00%	0	223	1	88	15,106	1,368	4,029	189
DeepLearning4J	17	6	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

Options for Deep Learning Toolkits



Options for Deep Learning Toolkits

	Languages	Tutorials and training materials	CNN modeling capability	RNN modeling capability	Architecture: easy-to-use and modular front end	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	+	+
Tensor-Flow	Python	+++	+++	++	+++	++	++	+
Torch	Lua, Python (new)	+	+++	++	++	+++	++	
Caffe	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	

Programmatic creation

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

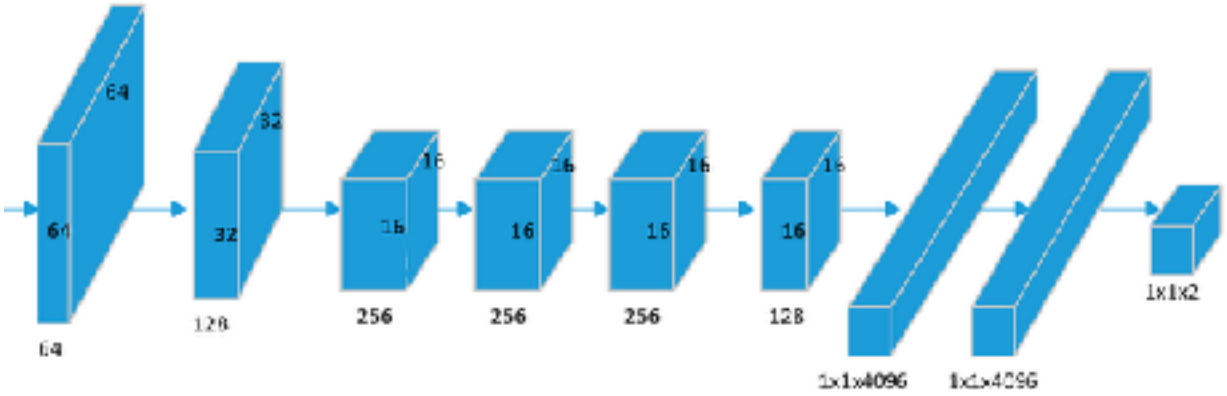
Tensorflow

- 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)
- **Supports:**
  - tensor creation
  - functions on tensors
  - automatic derivative computation

Tensors

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

```
a = tf.constant(5.0)

b = tf.constant(6.0)
```



Tensor basic functions

```
a = tf.constant(5.0)

b = tf.constant(6.0)

c = a * b
```

- Easy to define operations on tensors

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

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

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

Tensor neural network functions

- Easy to define operations on layers of networks

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

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

Tensor function evaluation

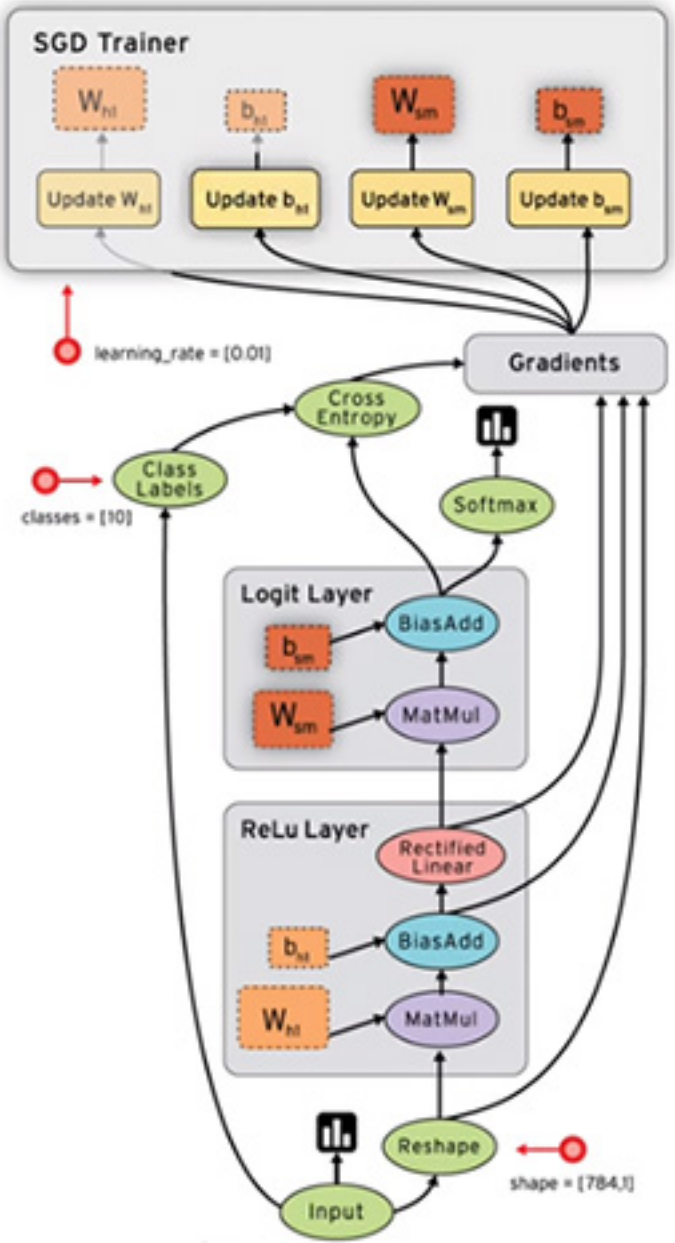
- Easy to define operations on tensors
- Nothing evaluated until you define a session and tell it to evaluate it
- Session defines configuration of execution
  - like GPU versus CPU

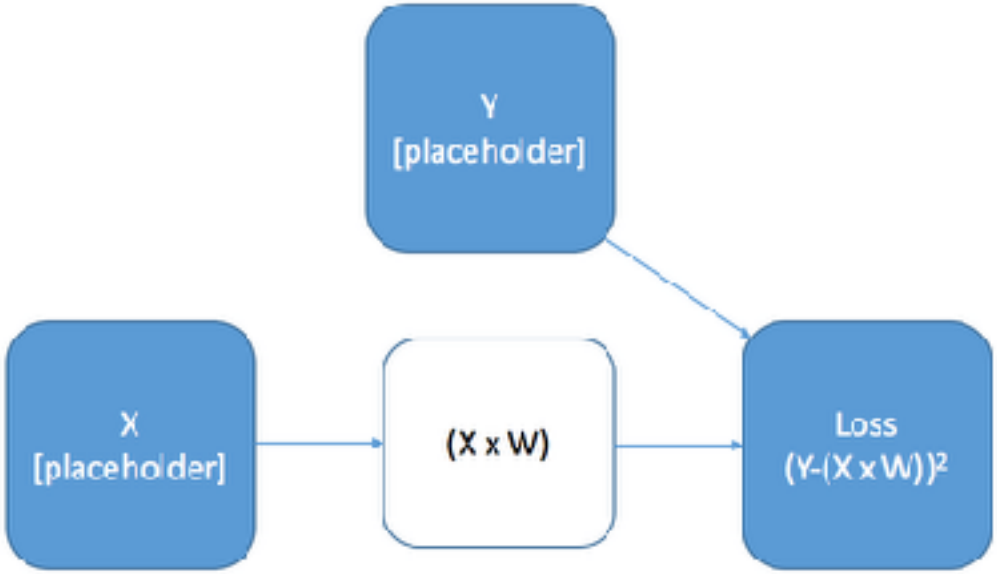
```
a = tf.constant(5.0)

b = tf.constant(6.0)

c = a * b

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

output = 30

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

Computation Graph





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

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

Tensorflow for Neural Networks

- Starting Simple…

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

```
# Define variables to be learned
with tf.variable_scope("linear-regression"):
  W = tf.get_variable("weights", (1, 1),
                      initializer=tf.random_normal_initializer())
  b = tf.get_variable("bias", (1,),
                      initializer=tf.constant_initializer(0.0))
  y_pred = tf.matmul(X, W) + b
  loss = tf.reduce_sum((y - y_pred)**2/n_samples)

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


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

Tensor Mini-batching?

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

with tf.Session() as sess:
  # Initialize Variables in graph
  sess.run(tf.initialize_all_variables())
  # Gradient descent loop for 500 steps
  for _ in range(500):
    # Select random minibatch
    indices = np.random.choice(n_samples, batch_size)
    X_batch, y_batch = X_data[indices], y_data[indices]
    # Do gradient descent step
    _, loss_val = sess.run([opt_operation, loss], feed_dict={X: X_batch, y: y_batch})
```


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

Tensor-flow Simplification

- It seems like many common NN optimizations can be preprogrammed
- **Self Test**: Can the syntax be simplified?
  - (A) **Yes**, we could write a generic mini-batch optimization computation graph, then use it for arbitrary inputs
  - (B) **Yes**, but we lose control over the optimization procedures
  - (C) **Yes**, but we lose control over the NN models that we can create via Tensorflow
  - (D) **Yes**, and Dr. Larson is going to make us write it ourselves

Tensor General Training Overview

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

Tensor General Training Overview

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

10. Keras Wide and Deep.ipynb

Demo

TensorFlow with Keras

Reinventing the MLP Wheel

Other tutorials:

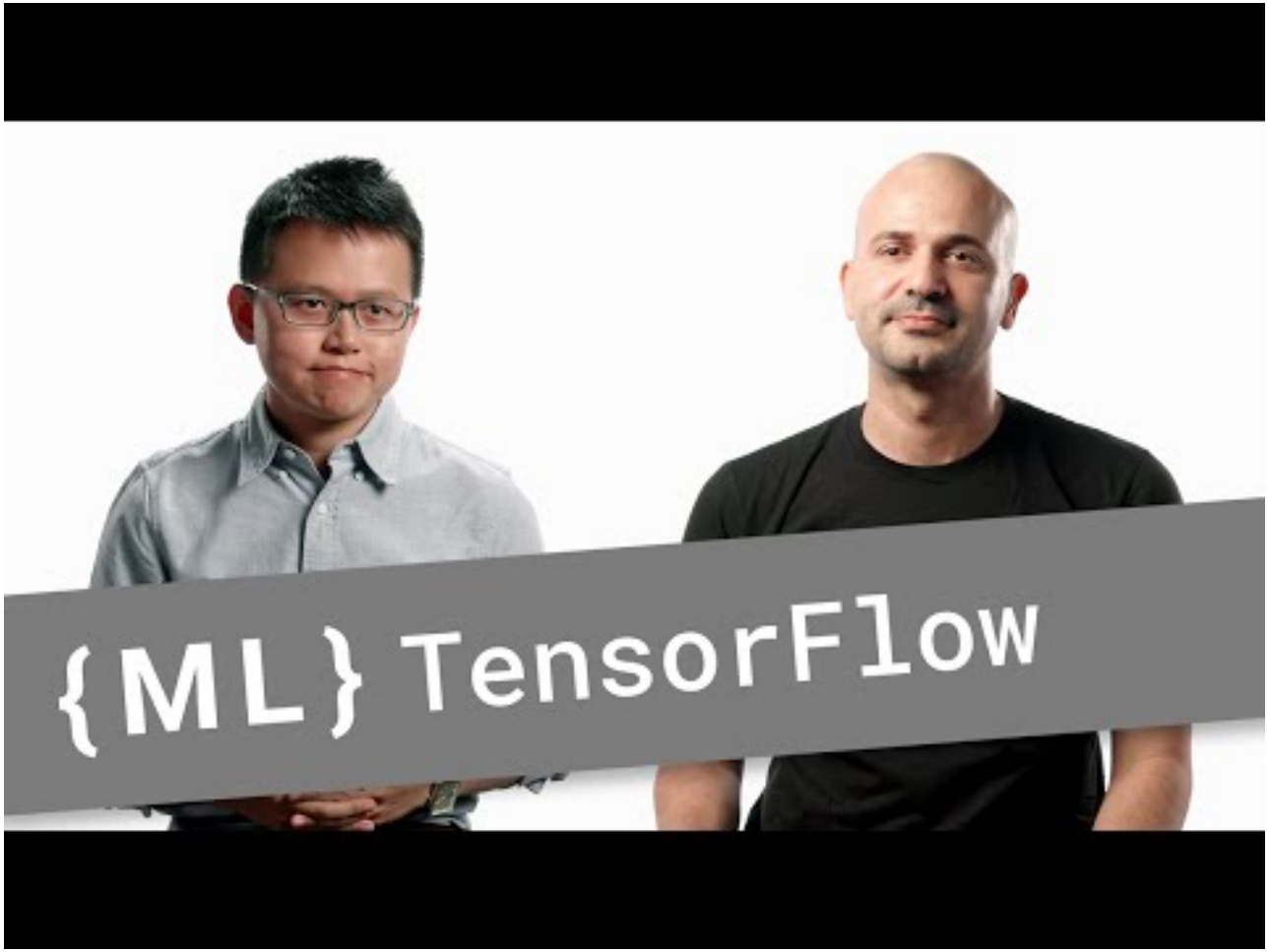
https://github.com/jtoy/awesome-tensorflow

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

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

Make me slow down if I go too fast!!

Wide and Deep Networks



Wide and Deep

Wide & Deep Learning for Recommender Systems

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

Google Inc.*

ABSTRACT

Generalized linear models with nonlinear feature transfor-

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

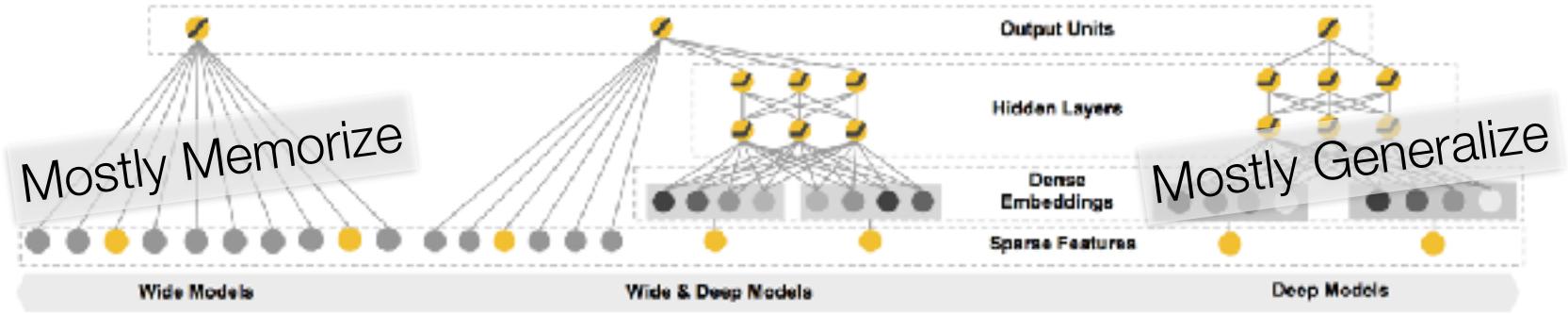
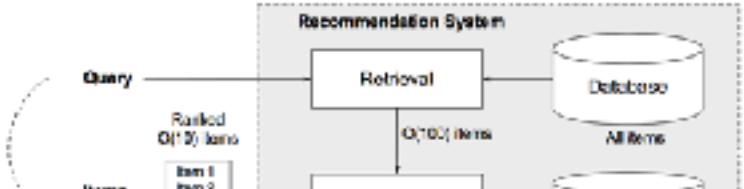


Figure 1: The spectrum of Wide & Deep models.

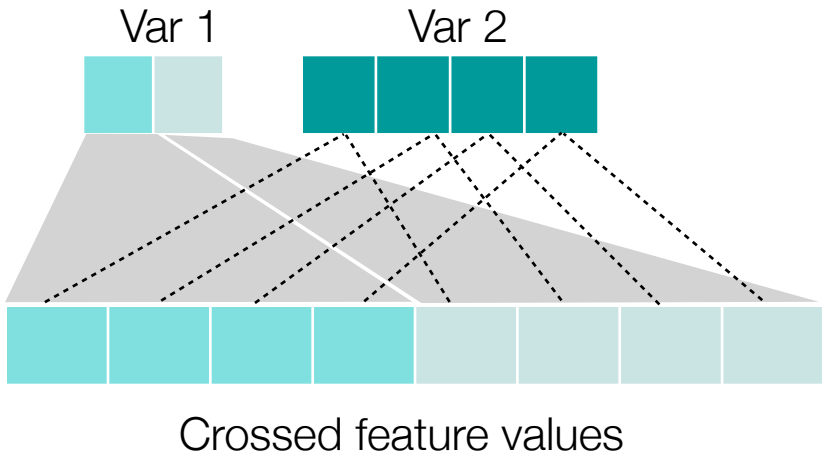
linear model with feature transformations for generic recommender systems with sparse inputs.

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



Wide networks (Memorize?)

- Wide refers to the expansion of your features set
- Crossed feature columns of categorical features
  - Movie Rating
    - G
    - PG
    - PG-13
    - R
    - Else
  - Movie Genre
    - Action
    - Drama
    - Comedy
    - Horror
    - Else

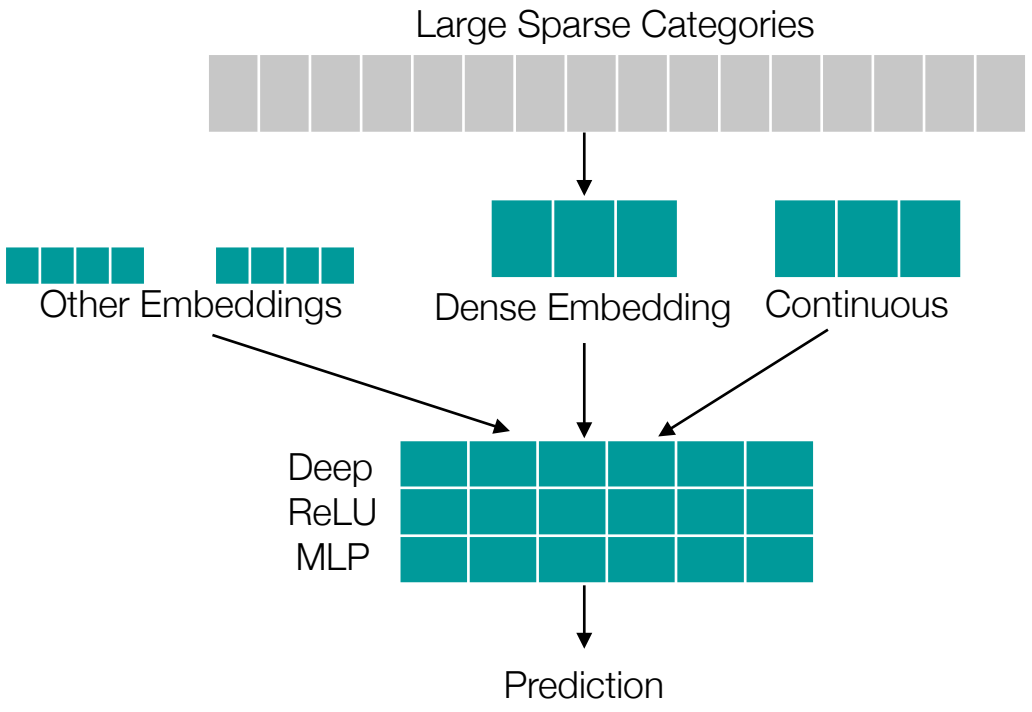


- Crossed feature “Rating-Genre”
  - G-Action, G-Drama, G-Comedy, G-Horror, G-else
  - PG-Action, PG-Drama, PG-Comedy, PG-Horror, G-else
  - and so on … one hot encoded

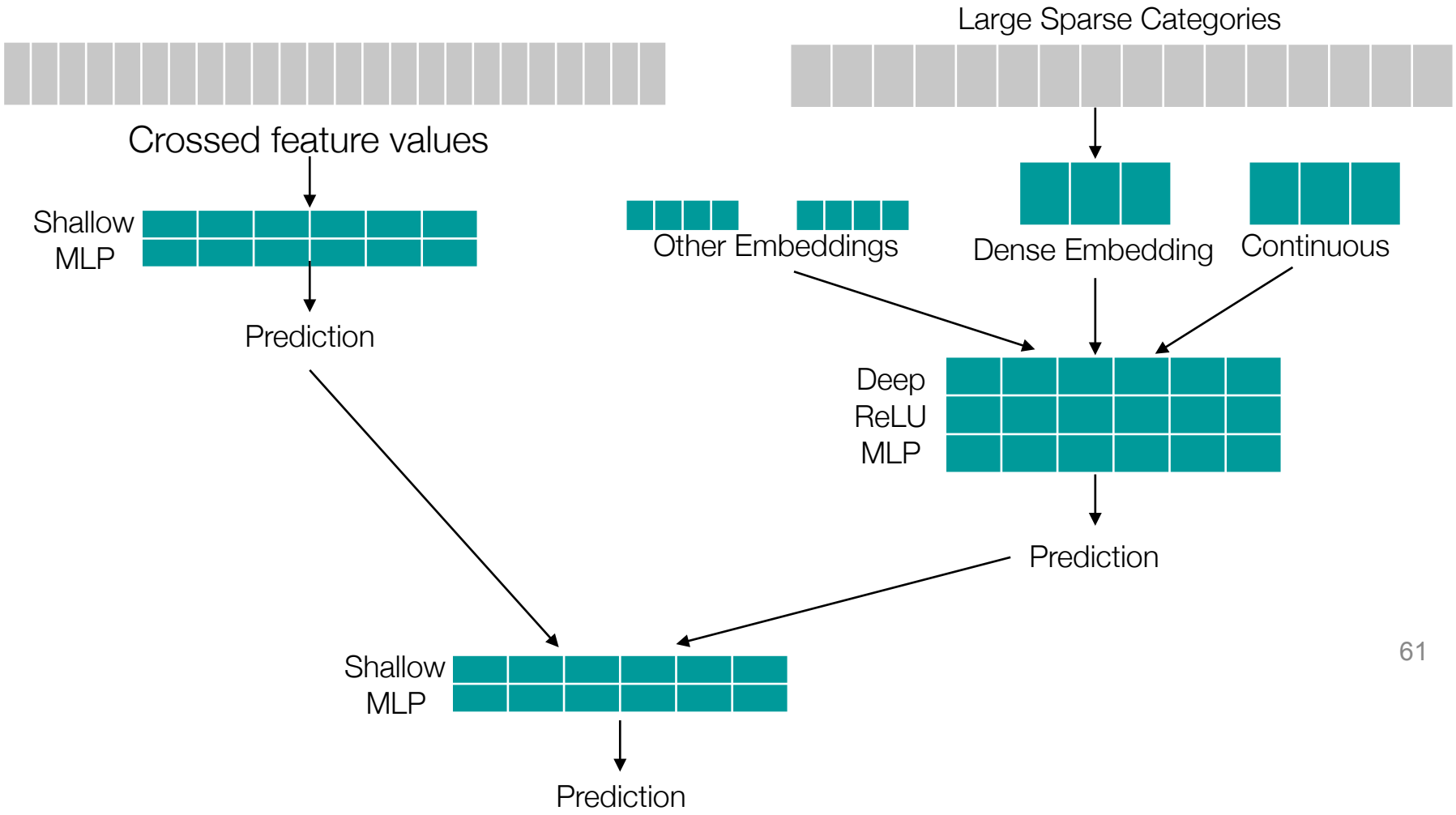
Sparse Embedding and Deep MLP (Generalize?)

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

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



Combining Memorization and Generalization



Demo

Wide and Deep

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

Back Up Slides

History of Deep Learning: summary

- Up to this point: back propagation saved AI winter for NN (Hinton and others!)
- 80's, 90's, 2000's: convolutional networks for image processing start to get deeper
  - but back propagation no longer does great job at training them
- SVMs and Random Forests gain traction…
  - The second AI winter begins, research in NN plummets
- 2004: Hinton secures funding from CIFAR in 2004
  - Hinton rebrands: Deep Learning (not called neural networks)
- 2006: Auto-encoding and Restricted Boltzmann Machines
- 2007: Deep networks are more efficient when pre-trained
  - RBMs not really the important part
- 2009: GPUs decrease training time by 70 fold…
- 2010: Hinton's students go to internships with Microsoft, Google, and IBM, making their speech recognition systems faster, more accurate and deployed in only 3 months…
- 2012: Hinton Lab, Google, IBM, and Microsoft jointly publish paper, popularity sky-rockets for deep learning methods
- 2011-2013: Ng and Google run unsupervised feature creation on YouTube videos (becomes computer vision benchmark)
- 2012+: Pre-training is not actually needed, just solutions for vanishing gradients (like ReLU)

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

Tensor Variables

- Variables that must be initialized before use

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

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

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

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


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

Tensor Variables

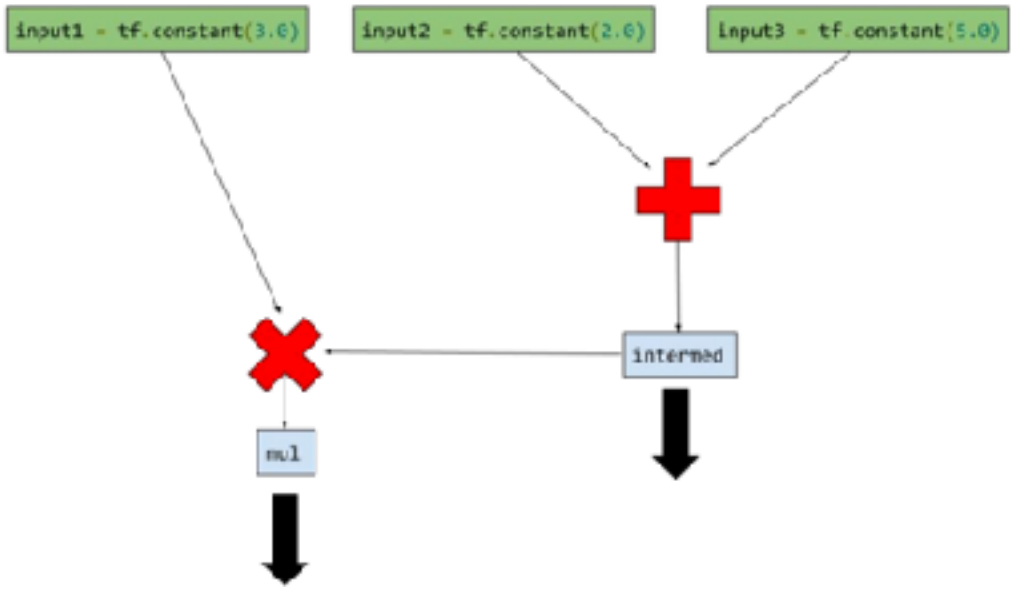
- Variables that must be initialized before use

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

mutable variable

mutable variable

fetch variable values

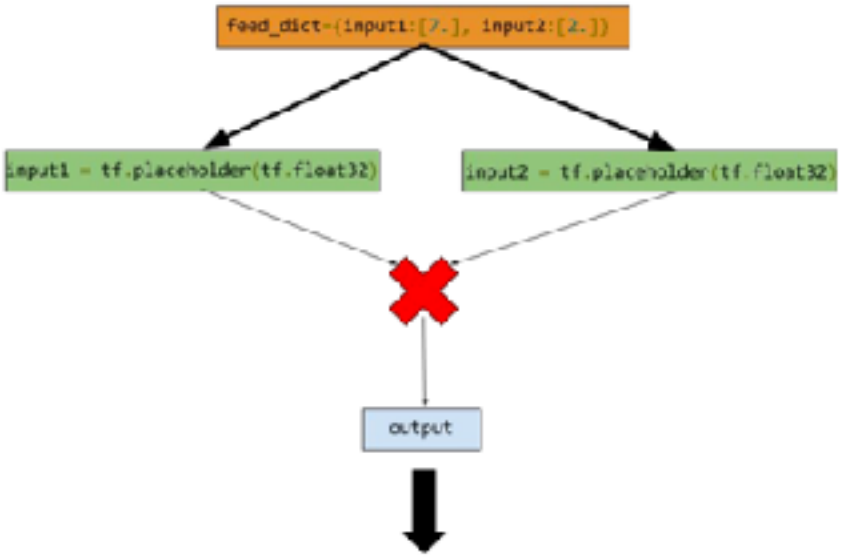


https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Tensor Placeholders

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

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

input2 = tf.placeholder(tf.float32)

output = tf.mul(input1, input2)

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



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

Tensor get_variable and scope

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

Variable Scope Example

Variable Scope mechanism in TensorFlow consists of 2 main functions:

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

```
with tf.variable_scope("foo"):
    v = tf.get_variable("v", [1])
with tf.variable_scope("foo", reuse=True):
    v1 = tf.get_variable("v", [1])
```

```
v.name == "foo/v:0"
```

without “reuse” this is an error

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

https://www.tensorflow.org/versions/r0.11/how_tos/variable_scope/index.html

Tensor get_variable and scope

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

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

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

explicit definition of repeated operations

https://www.tensorflow.org/versions/r0.11/how_tos/variable_scope/index.html

Tensor get_variable and scope

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

more readable code

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

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


https://www.tensorflow.org/versions/r0.11/how_tos/variable_scope/index.html

Sampling and Modeling

Sampling and Modeling

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

Lecture Notes for Machine Learning in Python

Professor Eric Larson

Archive: Dealing with Larger Data

Class logistics

- A2 is over!
  - thanks for all your work on this
- Remainder of semester:
  - A3 (no resubmits)
  - final in-class-assignment
- Grading is coming
  - resubmissions will be manipulated if I take too much time…

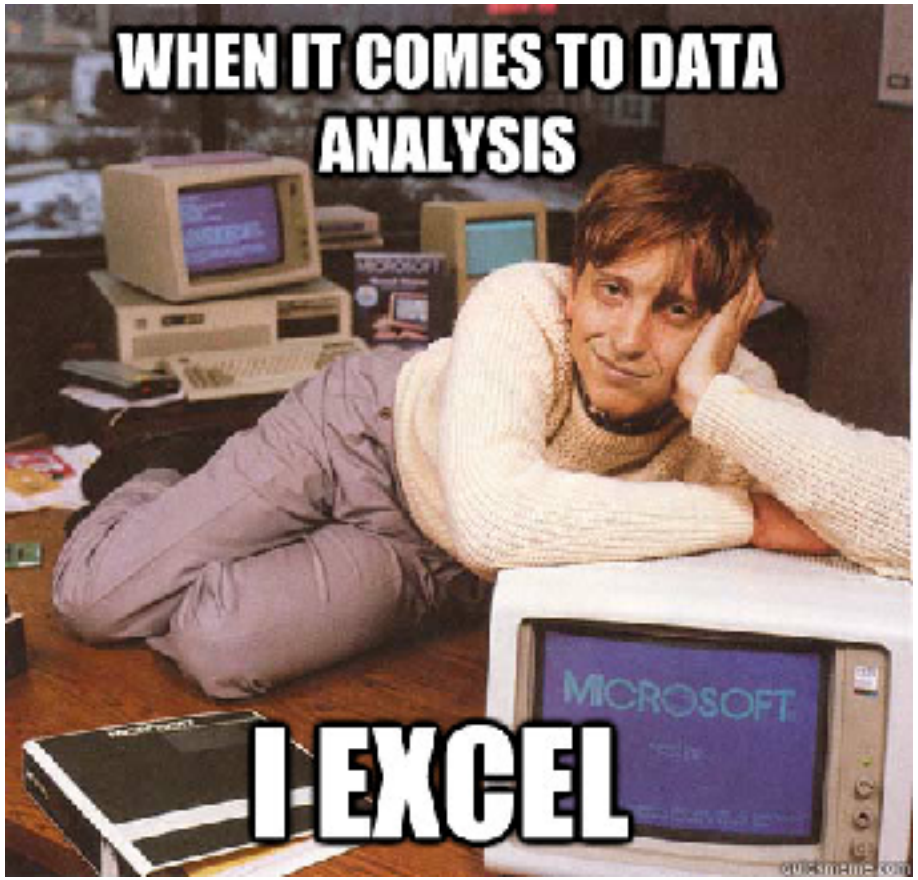
A2 Retrospective

- What problems did you run into?
- How did you solve them?

Two Lecture Agenda

- Today:
  - Defining larger data
  - Data Parallel: Grid Search
  - Out of core processing
  - Smarter processing

Big Data, Massive Data, Medium Data, and Maybe Massive-ish Data



Big Data

- Refers to the exponential size of data creation

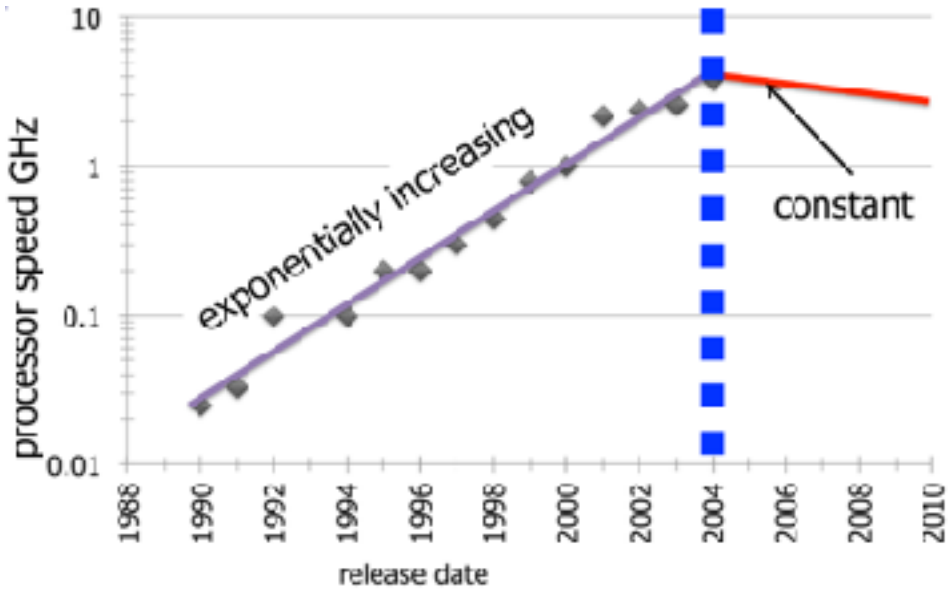
6 Billion
Flickr Photos

28 Million
Wikipedia Pages

1 Billion
Facebook Users

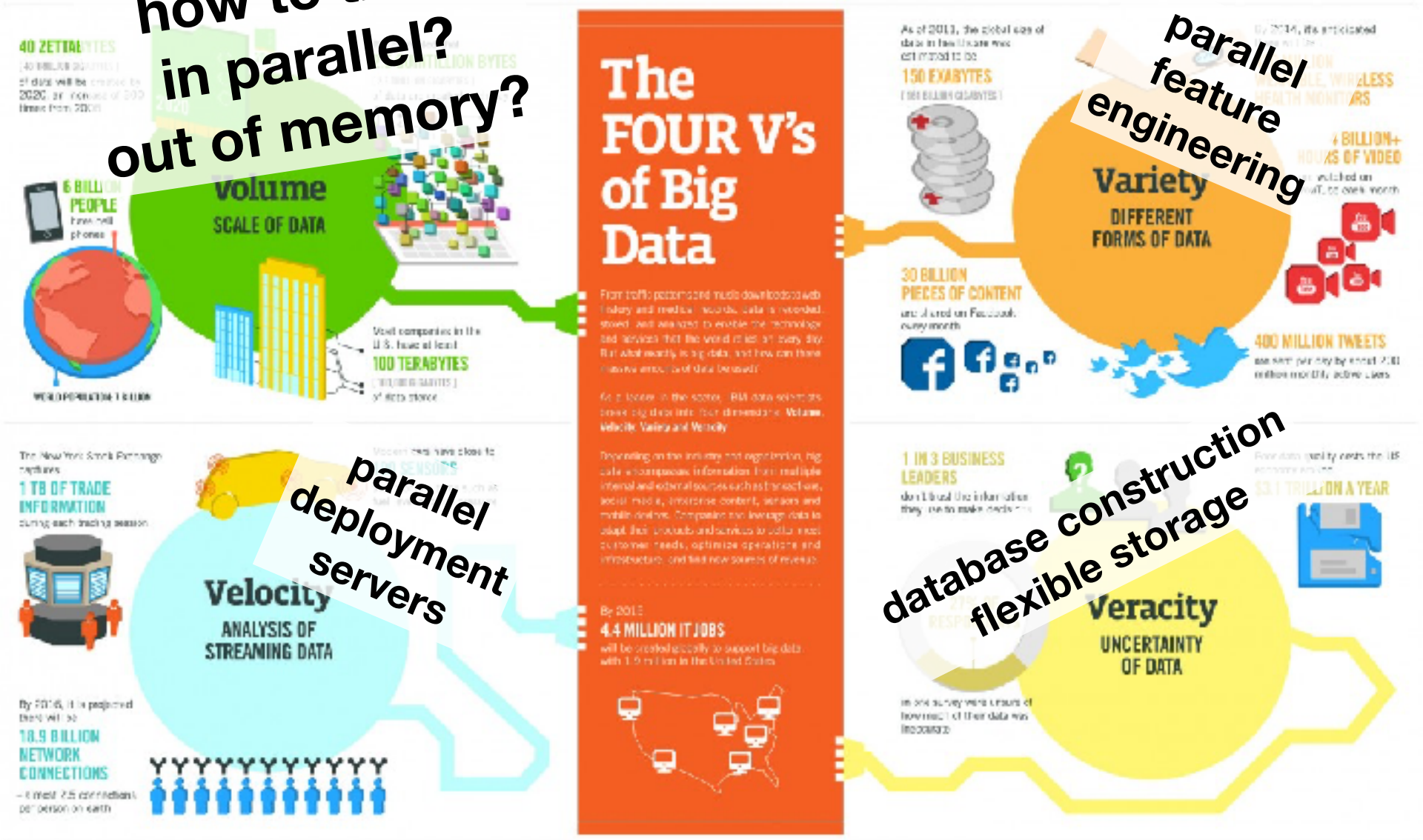
72 Hours a Minute
YouTube

- But is motivated by stagnant processor speed



You will see many definitions use the 4Vs

- **volume**: huge amount of data
- **variety**: types of data
- **velocity**: streaming data
- **veracity**: uncertainty of data



http://www.ibmbigdatahub.com/infographic/four-vs-big-data

Dealing with Larger Data

- Massive is in the **eye of the beholder**
  - some breakdown of data is still too massive to use traditional, in-memory, processing methods
- **out-of-core**: cannot fit into RAM on single machine or requires too much paging to keep in memory efficiently
- **parallelism**: can be broken into smaller subproblems.
  - Usually this is “embarrassingly parallel” that is exploited
  - **distributed**: clusters where code is brought to the data for processing (map-reduce).
- **sub-sampling**: don’t ever rule this out!

“Maybe Massive” Examples: out of core

- **Out-of-core** on a single machine is great for exploratory analysis (10’s of GB up to 1TB)
  - **use case**: want to use similar statistical, visualization, and aggregation tools as you are accustomed
  - **objective**: simplify the data in something manageable
  - **example**: visualizing airline data or other tabular data in a massive relational database
  - typically emulates distributed architecture, used for debugging, **this will be our focus**!

“Massive” Examples: cannot store

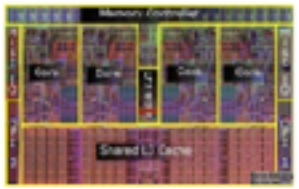
- Distributed
- **forced parallelism**: cannot fit on one hard drive
  - one server cannot store the data
  - streamed from many different sources
- implementing queries or aggregating over **truly massive** amounts of data—example: Google search, facebook, twitter
  - few companies work with data of this size (but not rare)
  - you need to define your problem as a map-reduce step and be conscious of data location or aggregate the data into a graph, then perform analysis

Data on different architectures

- scalable learning is hard because of
  - iterative algorithms
  - programability
  - failures
  - distributing data to parallel processes

abstracting parallel programming is just not there yet

GPUs



Multicore

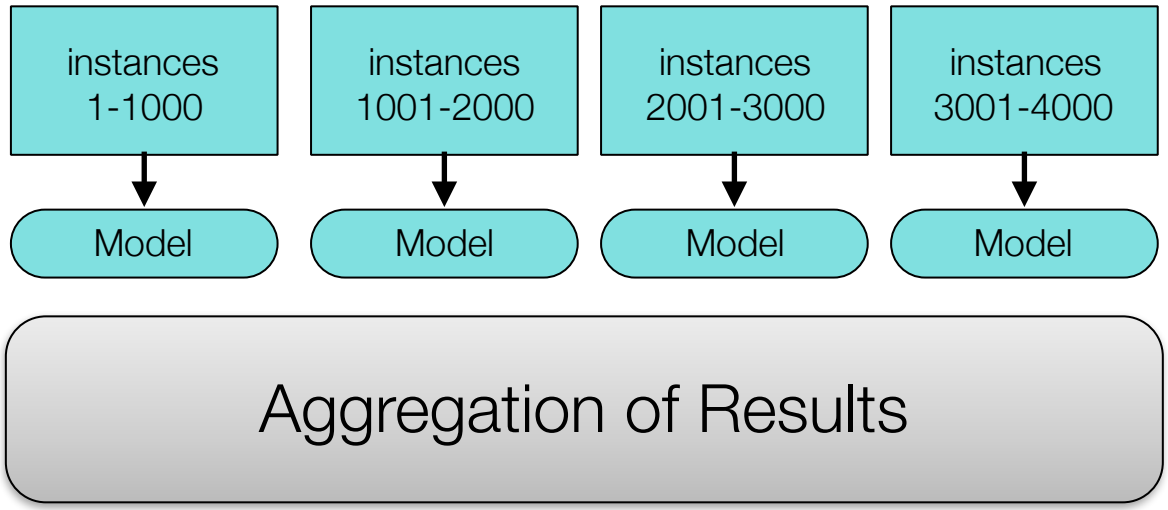
Clusters

Clouds

Supercomputers

Data Parallel (Embarrassingly Parallel)

- simplest type of parallelism, easy to exploit
  - problems can be broken into smaller, independent sub-problems
  - like running prediction on a data set of independent instances



Data Parallel Map Reduce

Map

- Data parallel over the elements (like images)
- Emit key/values (like image features)

input -> list(key, value)

Reduce

- Aggregate over the keys
- Must be commutative and associative operations
- Data parallel over keys
- Emit reduced output

list(key, value) -> output

Map Reduce

Self Test 11.1:

We are performing Logistic regression on a large dataset located upon a cluster of machines (each cluster has a portion of the data). Can map-reduce help our computation?

(A) Yes, we can run gradient estimation separately on each cluster

(B) No, gradient estimation is iterative and the overhead of transferring gradient data negates the speed up in distributed computation

(C) Yes, we can train a full Logistic regression model on each machine and combine the outputs

(D) No, its better to sub-sample from each cluster and train a model on a single machine

Map Reduce in Machine Learning

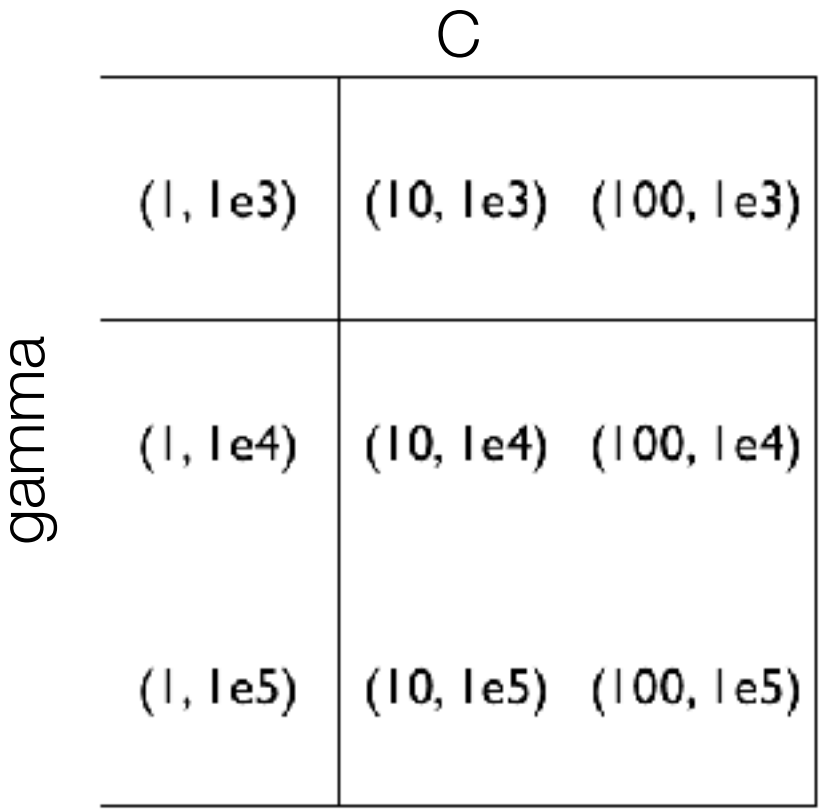
- Much of learning is iterative
- Though not always
  - ensembles
  - multi-class training (for large classes)
- Data gets moved around your cluster, possibly replicated, overhead might eliminate benefits
  - keep data near computation (or on HDFS)
- Mostly, Map Reduce is great for embarrassingly parallel
  - feature extraction
  - grid searching
  - cross validation
  - computing statistics

lets look at these!!

a good tutorial on map reduce in data mining: http://videolectures.net/kdd2010_papadimitriou_sun_yan_lsdm/

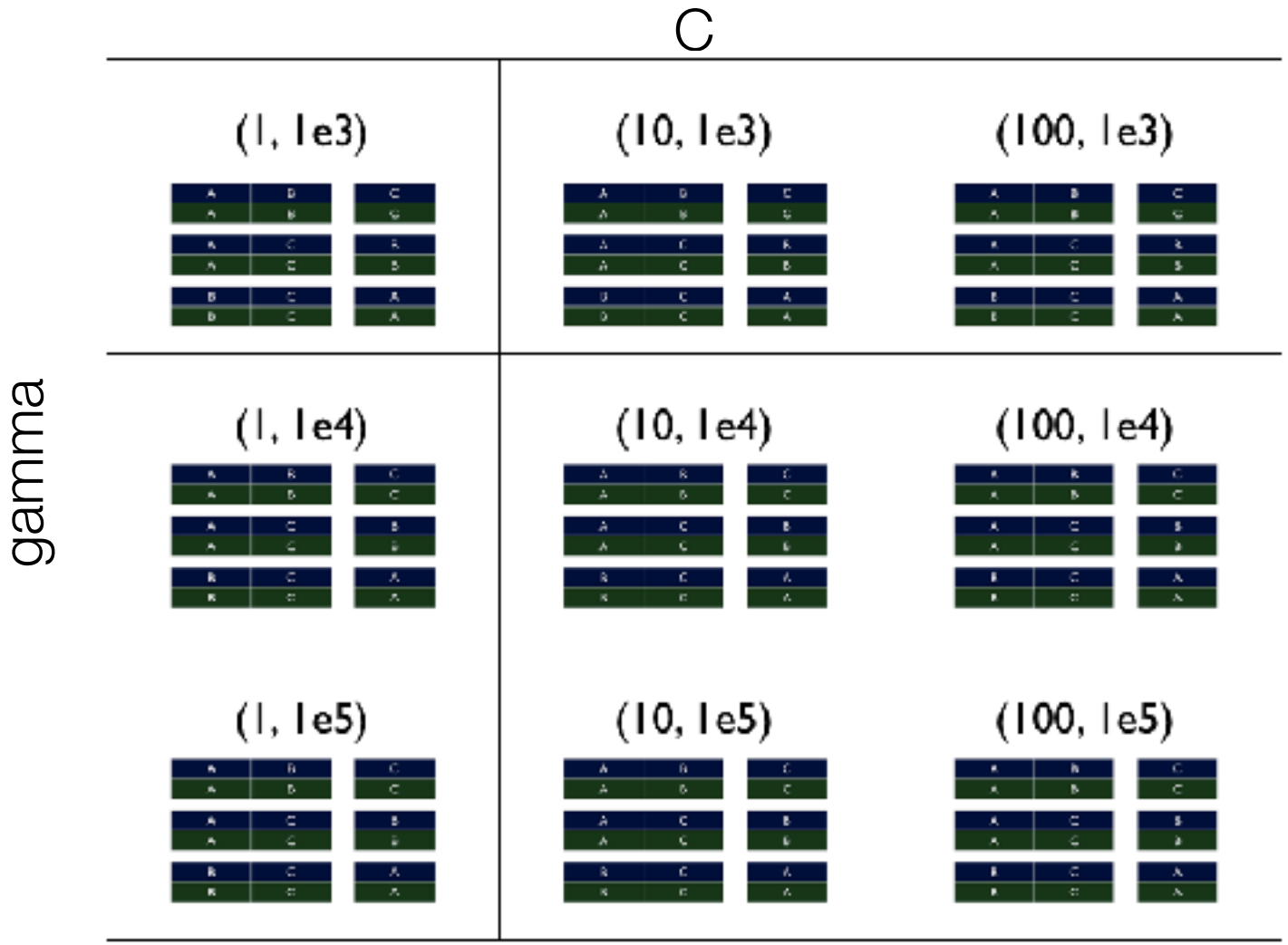
Grid Searching

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



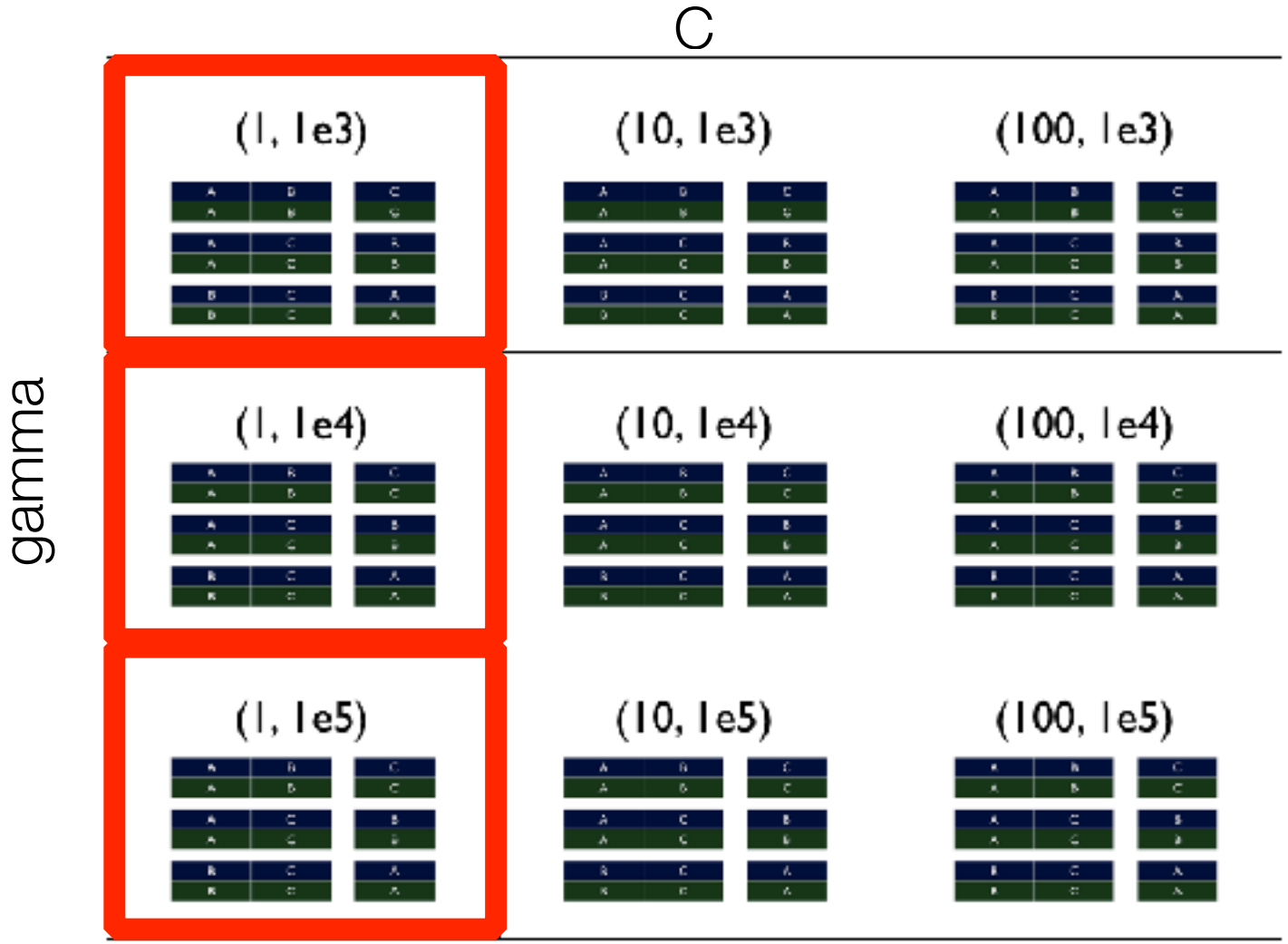
Grid Searching

- For each value, want to run cross validation…



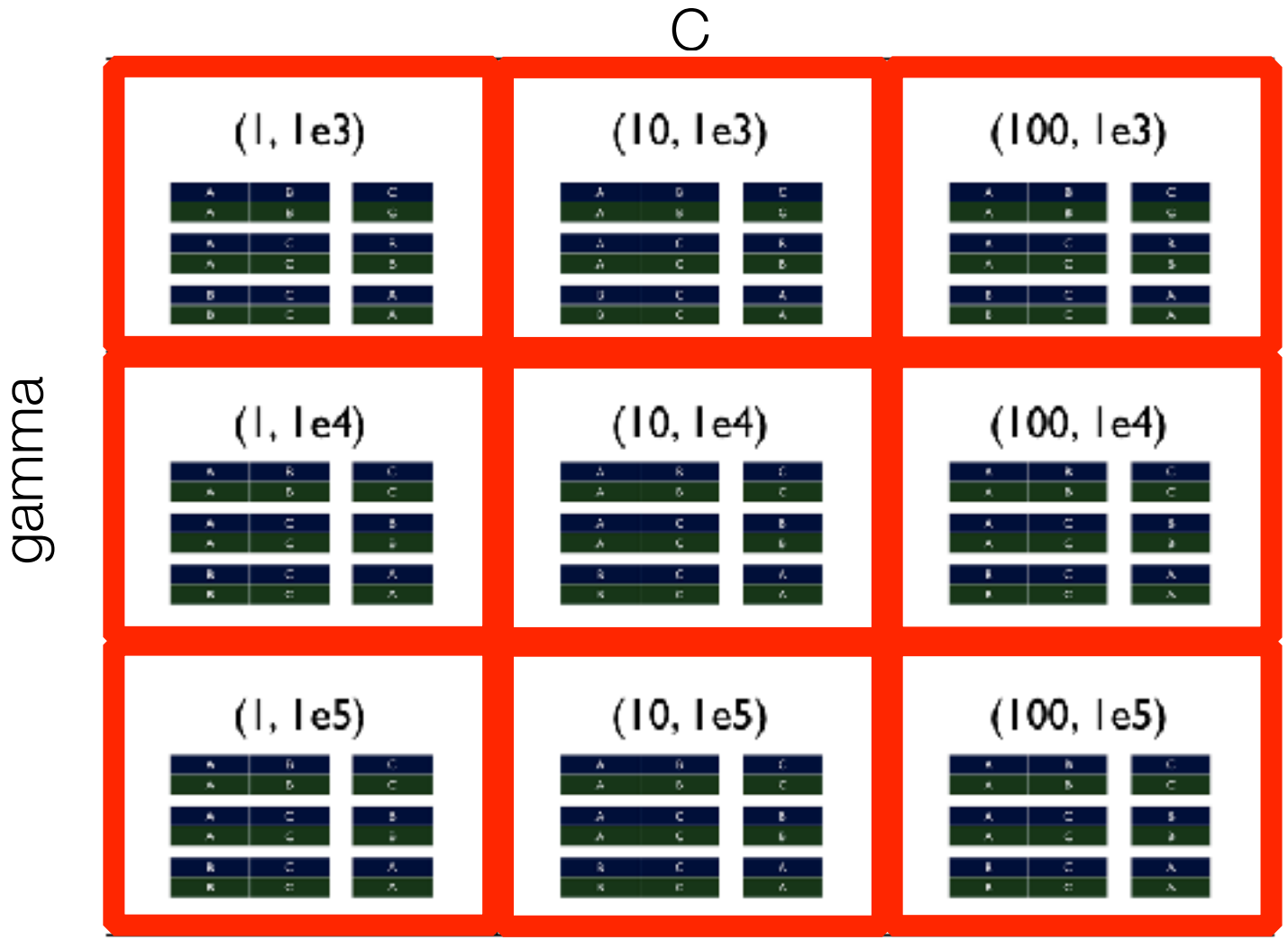
Grid Searching

- Could perform iteratively



Grid Searching

- or at random…



Demo

Parallel Parameter Tuning in sklearn

notebook 10

Other tutorials:

Olivier Grisel's Tutorial:

https://www.youtube.com/watch?v=iFkRt3BCctg

~3 hours

https://github.com/ogrisel/parallel_ml_tutorial/blob/master/rendered_notebooks/06%20-%20Distributed%20Model%20Selection%20and%20Assessment.ipynb

Lecture Notes for Machine Learning in Python

Professor Eric Larson

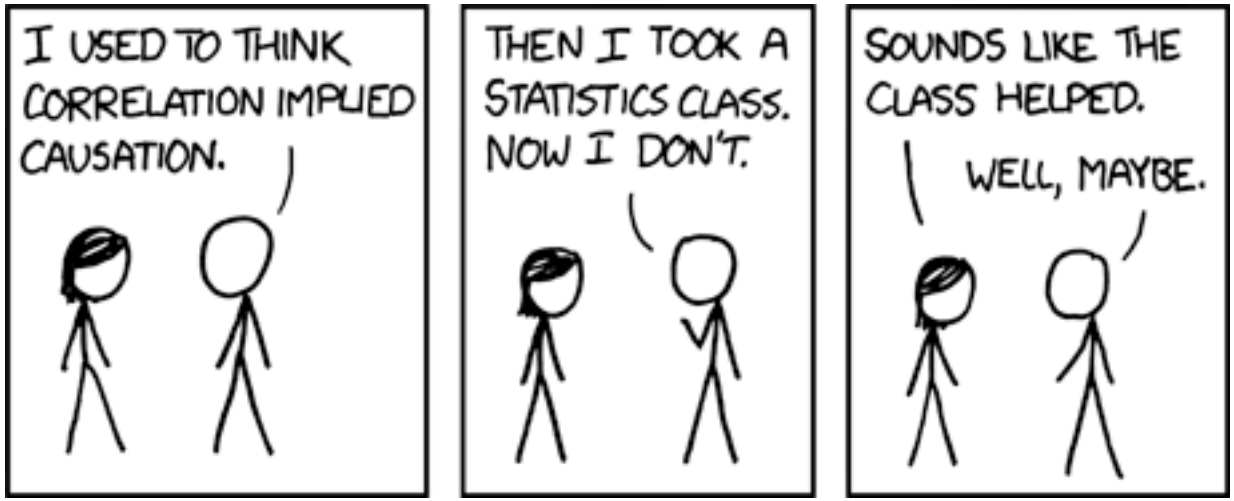
Archive Dealing with Larger Data

Two Lecture Agenda

- *Last Lecture*
  - *Defining larger data*
  - *Data Parallel: Grid Search*

- Today:
  - Out of core processing
  - Smarter processing

Managing Memory



Out of Core Memory

- **Data cannot fit into RAM**
  - Classic **space-time** tradeoff
  - Load from disk in **chunks**
- Each chunk: **memory mapping**
  - map on disk instead of RAM
  - works, but is **slow**
  - need to optimize for **sequential access**

Out of Core Memory

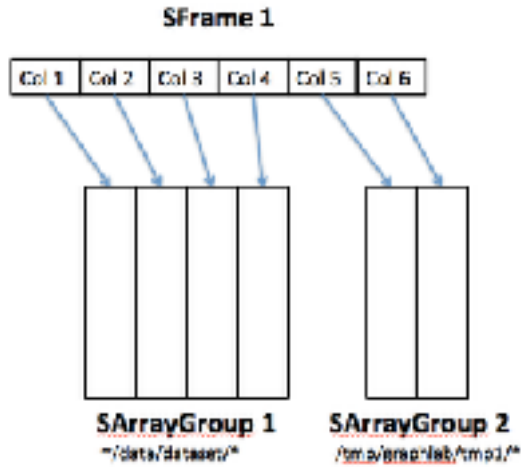
- **Data Format** is paramount: disk I/O and processing power
  - **Formatted file**: like a CSV, slow to read and requires parsing. Avoid this at all costs! But, this is a likely format to start with…
  - **Database**: optimized access, we want to use or emulate this type of memory access
  - **Uncompressed Raw Data**: once read, no further processing is needed, stored exactly as it is in memory
  - **Compressed**: Less to read from the disk, but requires further decoding

To Compress or not to Compress

- **Uncompressed**
  - fast indexing of **single datatype** arrays (like accessing struct in c++)
  - faster to know **where to read from file**, only need to read memory that is needed (fseek, fread)
  - quick sequential access, but **expensive** to add/delete data
- **Compressed** (or slightly compressed)
  - if compression is **hardware optimized**, then very little overhead for the decoding operation
  - need to read chunks of **possibly irrelevant data** to decompress what you are interested in
  - much **less** hard disk **memory footprint** required, could be as much as 80-90% savings

Out of Core: Representation

- **single file**: all data stored back to back in one file exactly like dumping a c-struct to file (raw bytes)
  - **tough to add columns** — need to read and write file completely to keep continuity
  - typically only **support one data type** or only numeric data
  - **examples**: numpy memmap and bigmemory in R
- **multiple files**:
  - separated by row chunks
    - each file has identical format, multiple data types
    - **example**: Dask (many pandas chunks)
  - separated by columns
    - trivial to **add columns** of new data
    - string and categorical data are easier to support
    - for lots of columns, can overwhelm the file system
    - **example**: GraphLab SFrames



Out of Core: Access

- Usually **subsumed to the programmer** via APIs
  - Numpy's memory map which is simply using c-file pointers rather than in-memory stack pointers
    - set up memory and never load all of it
  - easy to share memory across processes as long as file locking implemented appropriately (much easier to lock on the file system)
  - file system caches sequential access (huge speed up!!)
  - "*easy*" to use in `scikit-learn` with `partial_fit`

Out of Core: Access

- **Self Test 11.2**: A memory mapped file with 1 billion float32 values will be of what size on disk:
  - (A) 32 GB
  - (B) 4 GB
  - (C) 1 GB
  - (D) it depends

Out-of-core Machine Learning

Partial fitting
notebook 11

Other tutorials:

- https://github.com/rasbt/pattern_classification/blob/master/machine_learning/scikit-learn/outofcore_modelpersistence.ipynb

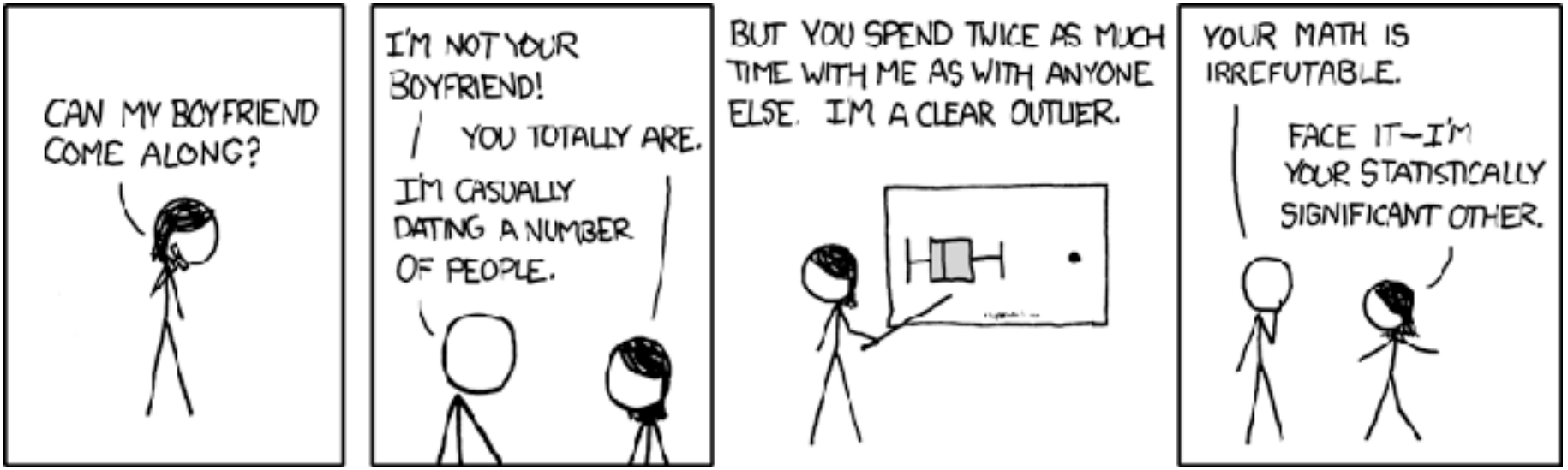
Anything from Jim Crist or Mathew Rocklin:

- https://github.com/jcrist

Alex Perrier:

- https://www.opendatascience.com/blog/riding-on-large-data-with-scikit-learn/

Smarter Processing



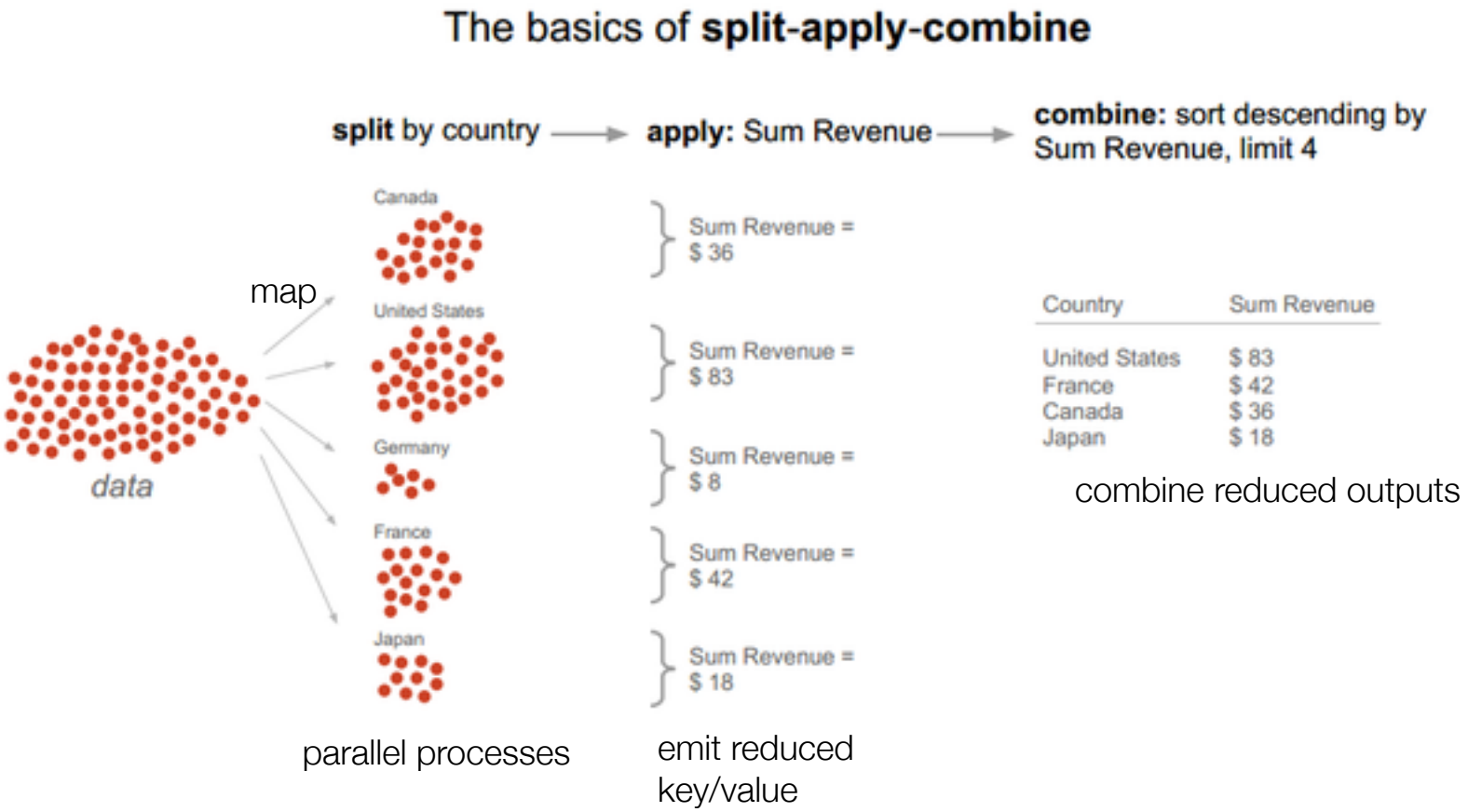
Smarter Processing

- **Data Parallelism**
  - Common form: split-apply-combine for **embarrassingly parallel tasks**
  - Map-Reduce with one reducer
  - **Mini-batch** with averaging
- **Task Scheduling**
  - Tasks **agglomerated** until required to be executed
  - Multiple tasks run in **one** data **access**
  - lazy evaluation
- These are **not mutually exclusive**!

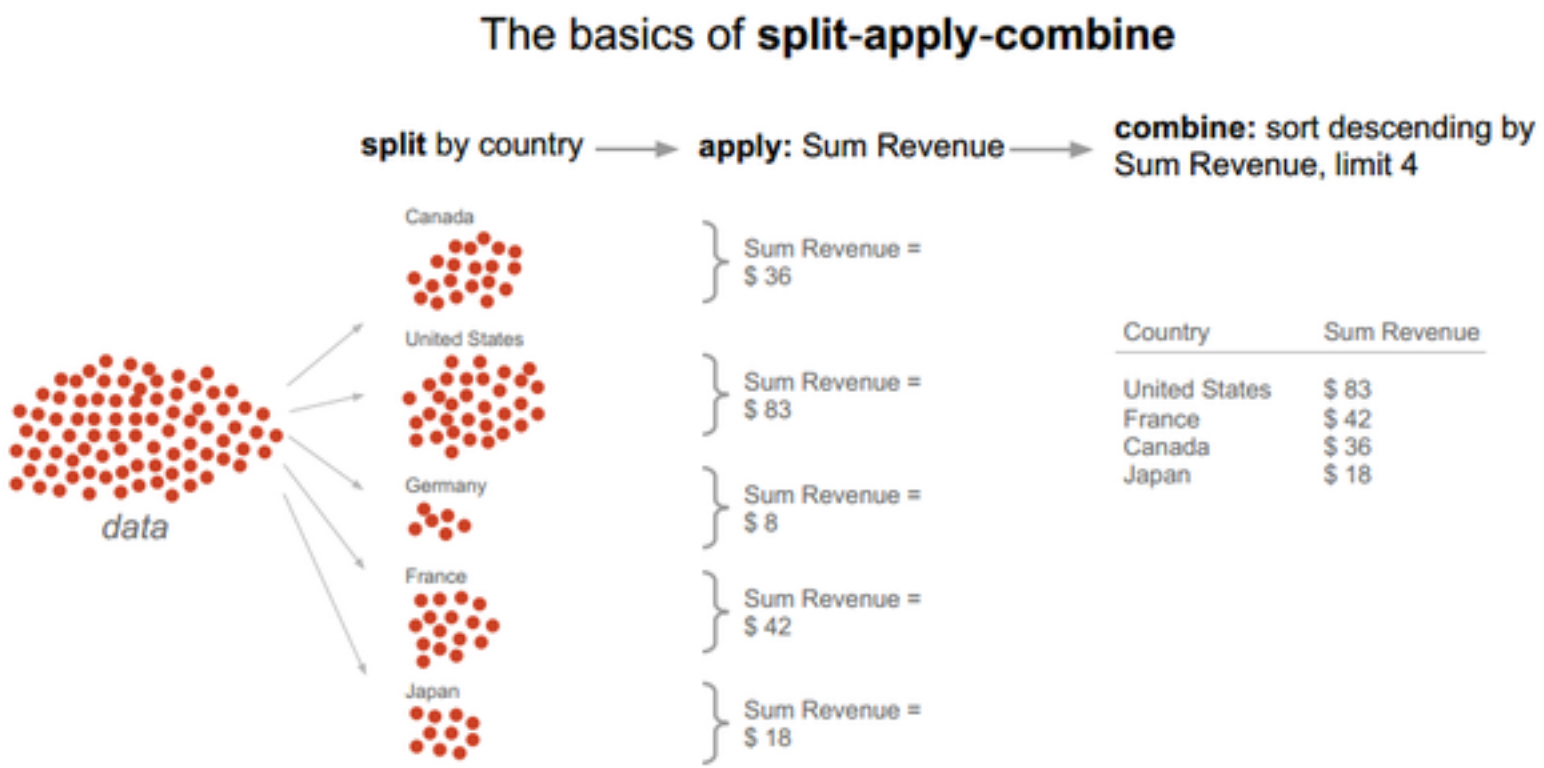
Spilt-Apply-Combine

- Just an embarrassingly parallel form of map-reduce
  - **Split**: partitioned grouping of the data (map)
  - **Apply**: a custom aggregation of the group (reduction, done in parallel)
  - **Combine**: combining the groups together (emission of reduced output)
- **Example**: taking grouped statistics (min, max, mean)

Spilt-Apply-Combine



https://twitter.com/medriscoll/status/557989114805747712



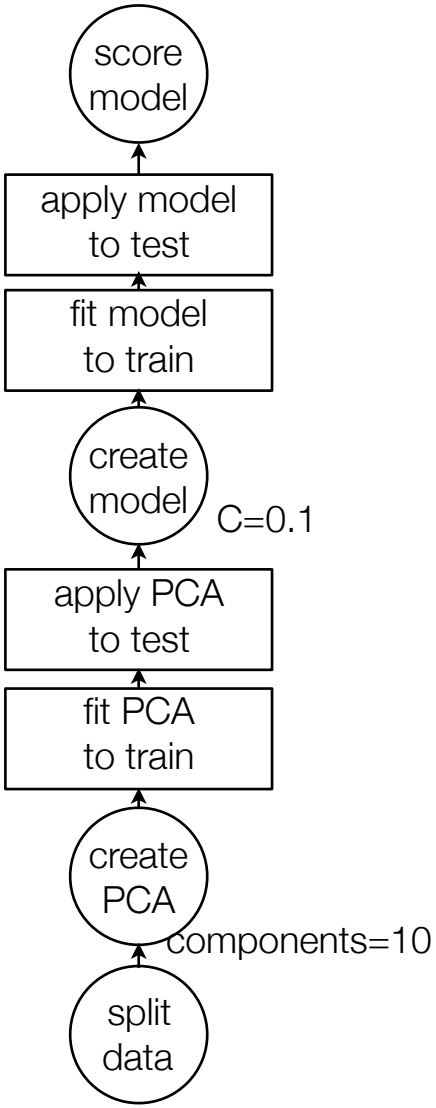
- **Self Test 11.3**: Would split-apply-combine in a distributed environment work well?
  - (A) Yes, data is properly divided for parallelism
  - (B) No, split is not consistent for where data resides in each server
  - (C) Yes, but you need to distribute data by group before carrying out operations
  - (D) No, splits might be unequal sizes, negating advantages

https://twitter.com/medriscoll/status/557989114805747712

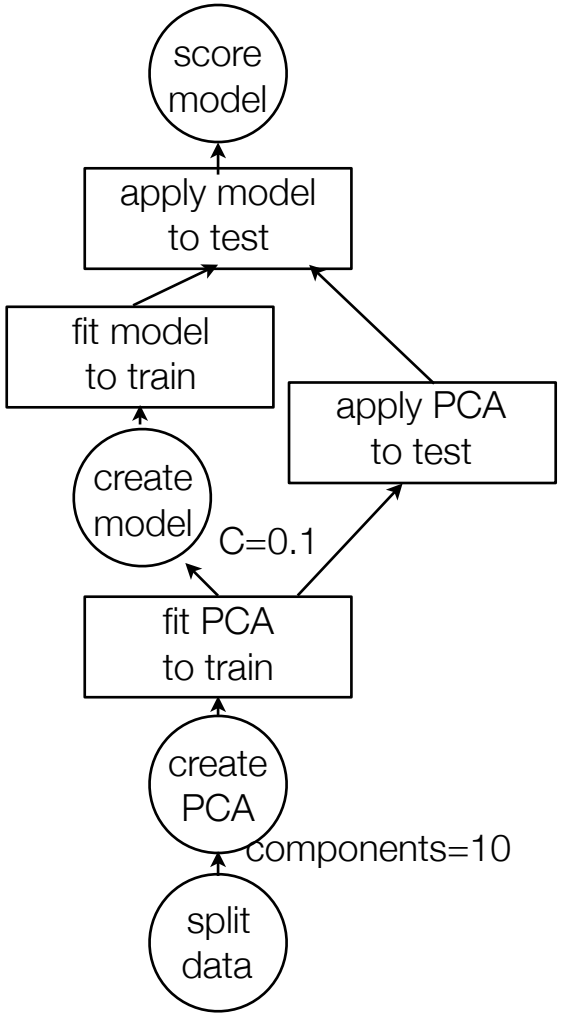
Lazy Evaluation

- **Wait** to evaluate until all tasks are agglomerated
- Typically for single machine architectures where explicit programming of graph is **subsumed**
- Use **Computation Graph**
  - only perform **needed** operations
  - **parallelize** operations where possible
  - **inline** functions for speedy operations

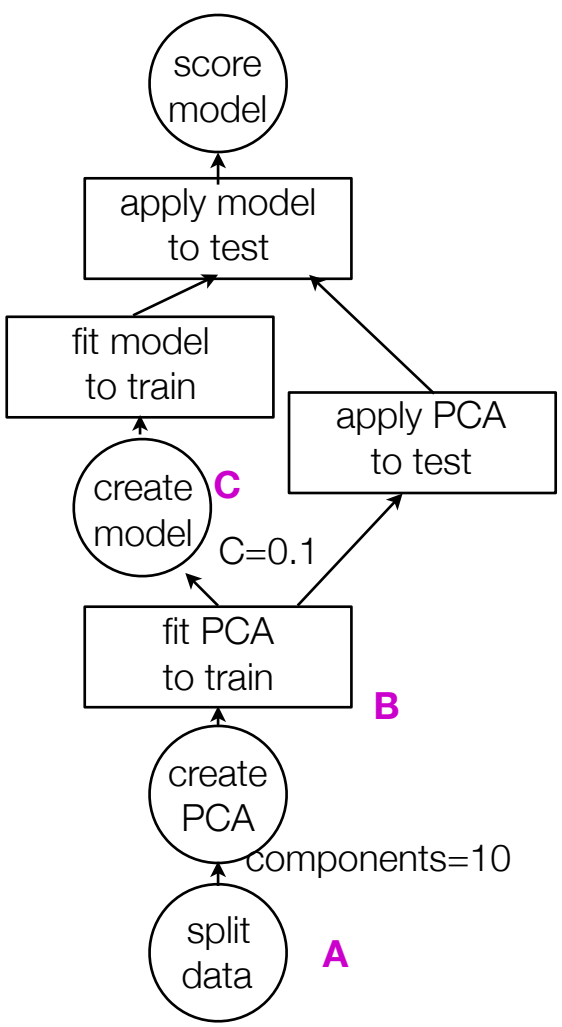
Lazy Evaluation: Example

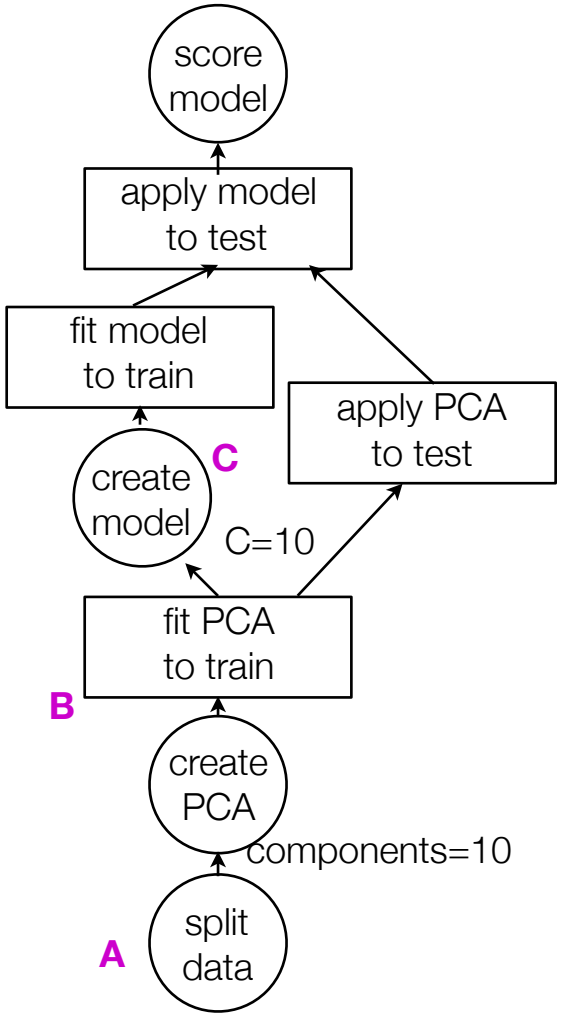


Lazy Evaluation: Example



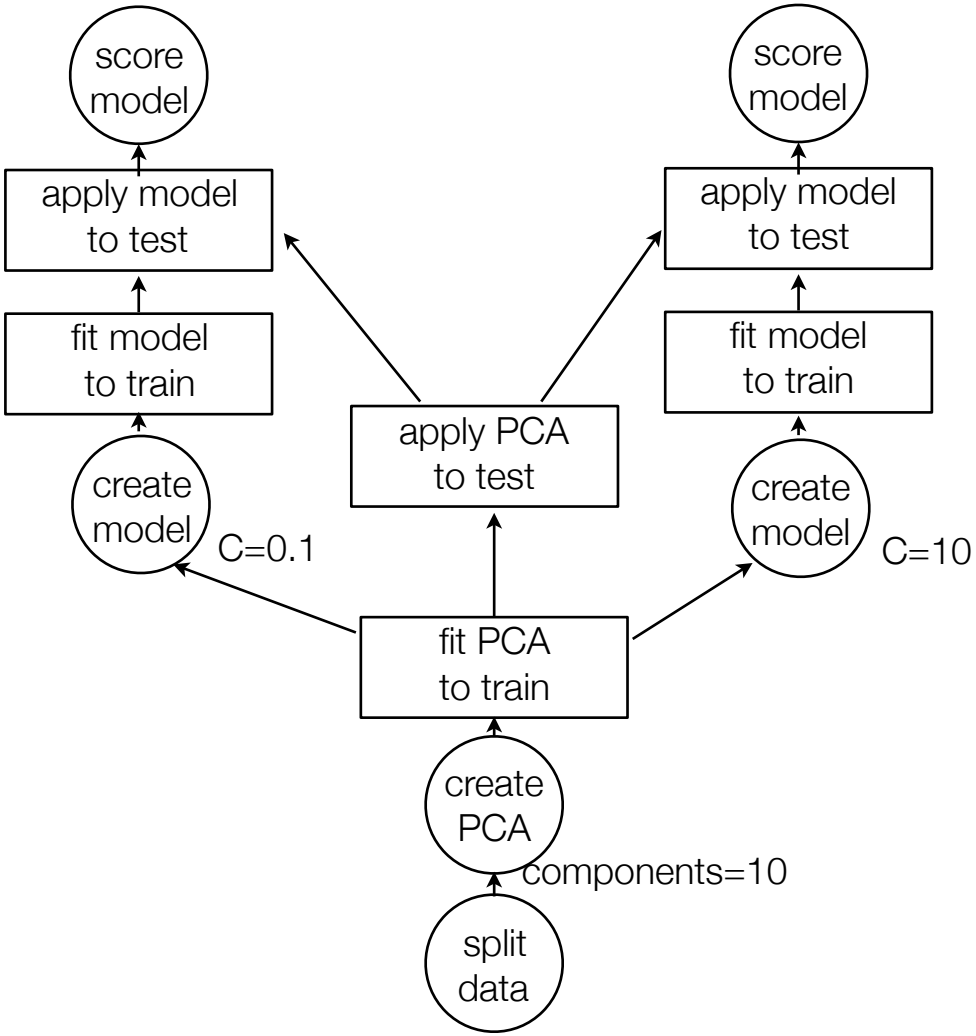
Lazy Evaluation: Example



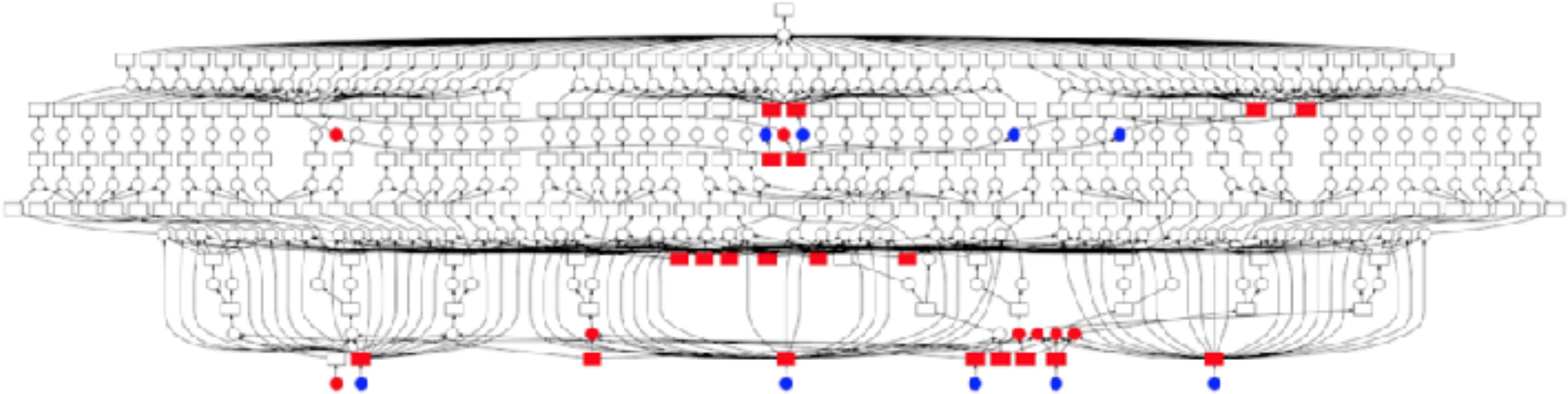


Self test 11.4: Which parts of the two models can be combined?

Lazy Evaluation: Example



Lazy Evaluation: Example



task graph for **GridSearch**

http://matthewrocklin.com/blog/work/2016/07/12/dask-learn-part-1

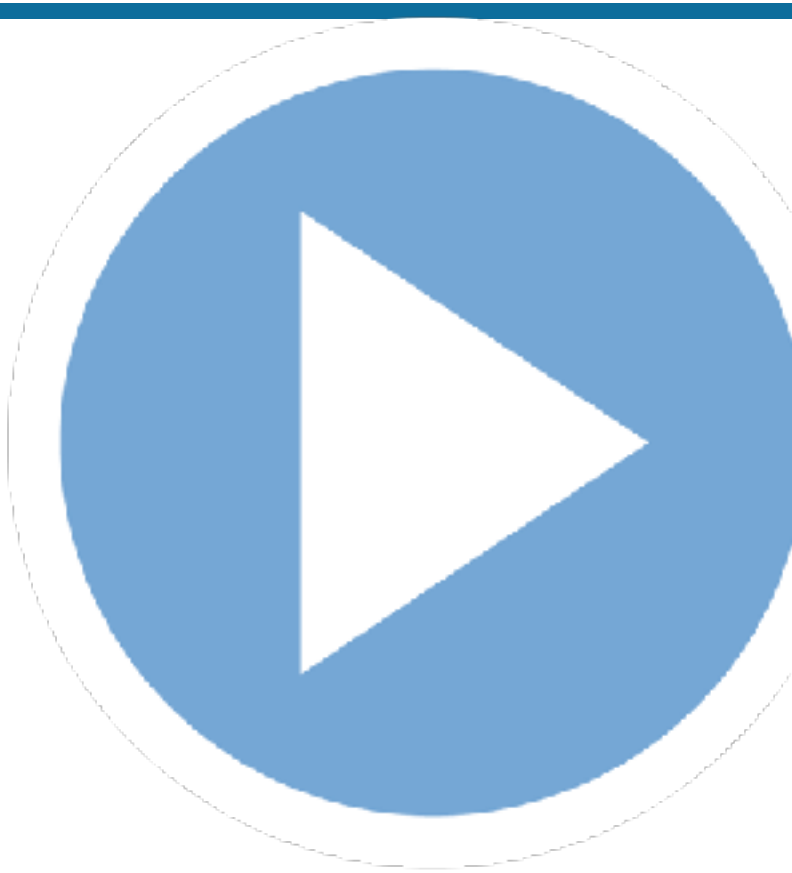
Lazy Evaluation: Caveats

- **Optimization** of task graph **is overhead**
  - Small for long running tasks
  - Might **increase memory footprint** for cached values
  - Doesn't solved any issues with copying data…
- But can easily (and should) be **combined** with **out-of-core** processing

Lazy Evaluation: Caveats

- Self Test 11.4: Computation Graphs are used by which of the following Companies/APIs for interacting with data?
  - (A) Dask
  - (B) Turi (graphlab)
  - (C) TensorFlow
  - (D) Theano

Putting it all together



Out of Core and Lazy Evaluation
in Dask (*notebook 12a*)
in GraphLab (*notebook 12b*)
with Airlines

Other tutorials:

- https://github.com/rasbt/pattern_classification/blob/master/machine_learning/scikit-learn/outofcore_modelpersistence.ipynb

Anything from Jim Crist or Mathew Rocklin:

- https://github.com/jcrist

Alex Perrier:

- https://www.opendatascience.com/blog/riding-on-large-data-with-scikit-learn/

For next time

- Back to Neural Networks
- Deep Learning Architectures
  - will finish out semester
  - convolutional NN
  - recurrent NN
  - TensorFlow examples
    - but you can use whatever API you wish
    - including SKFlow (now a part of TensorFlow)