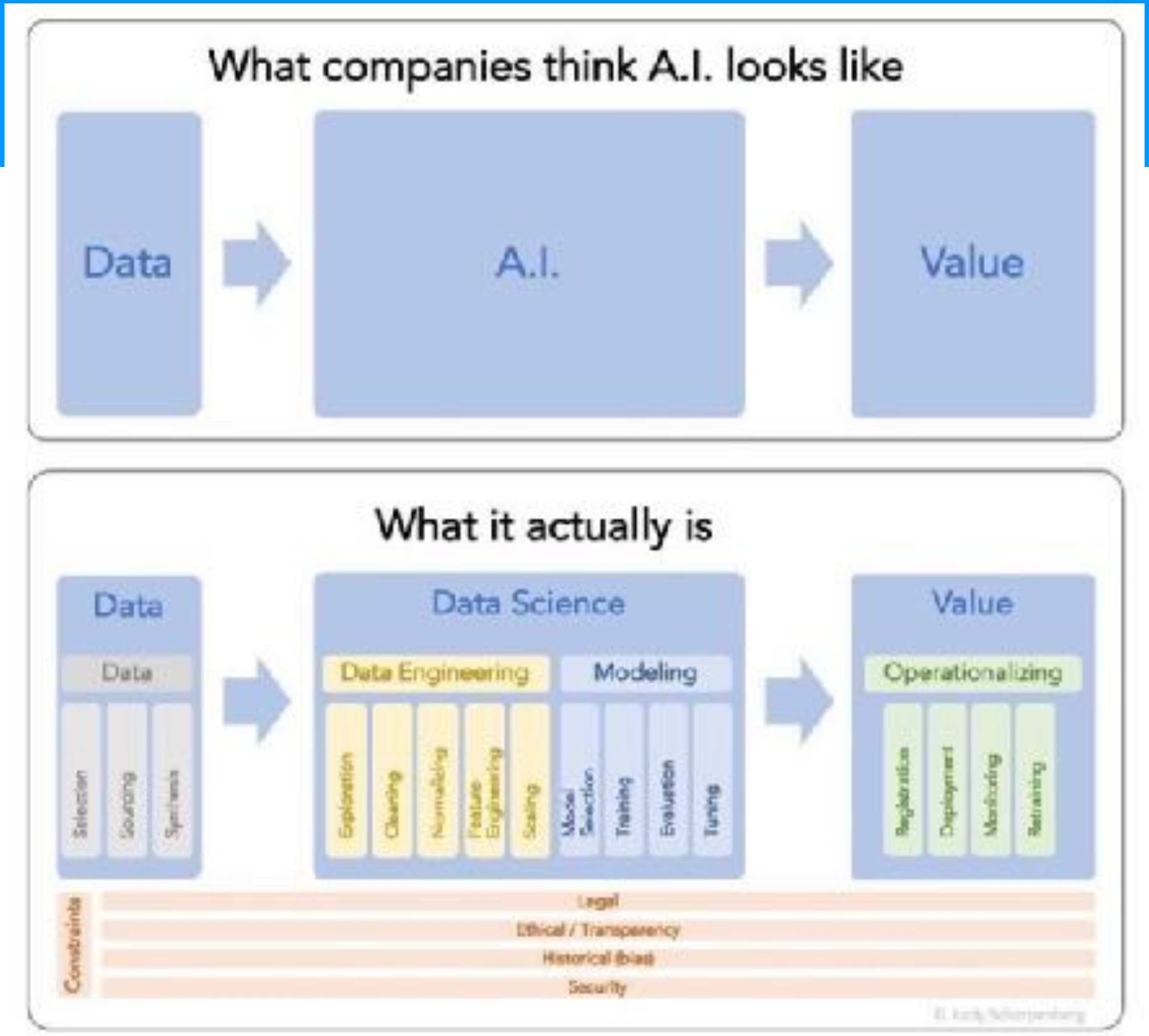
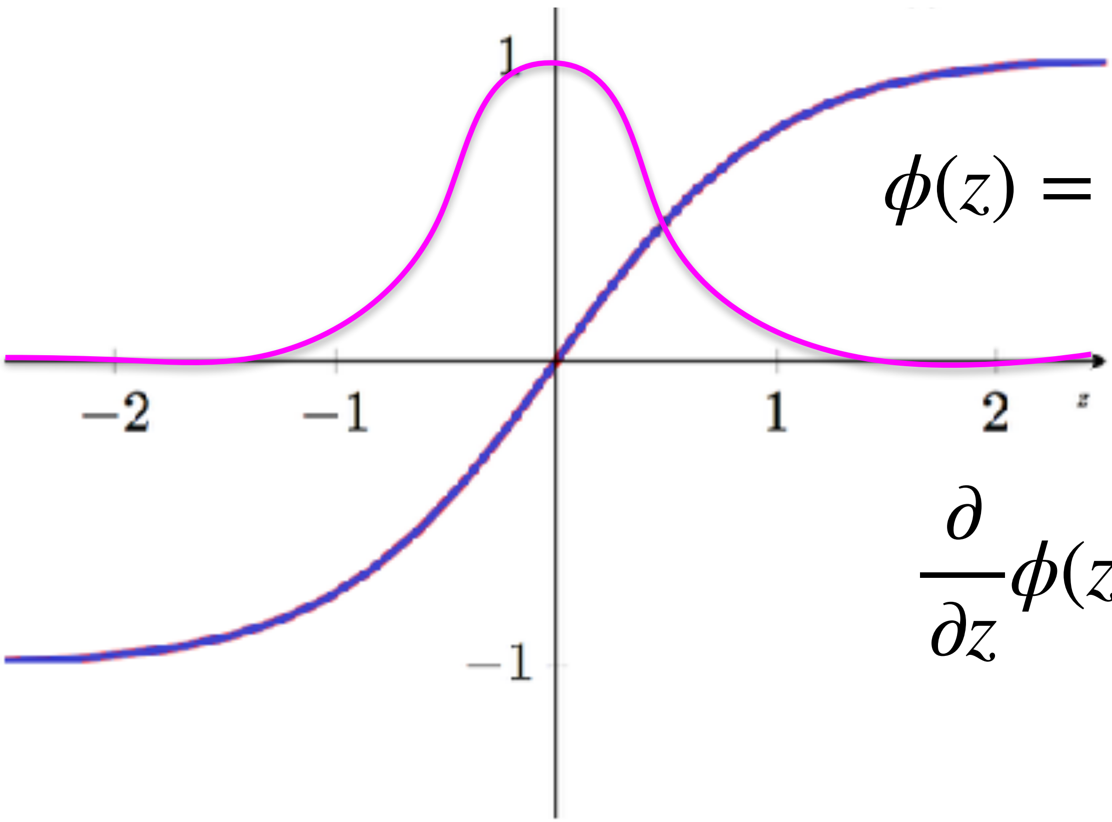
Beyond Sigmoid: Other Activations



New Activation: Hyperbolic Tangent

- Basically a sigmoid from -1 to 1

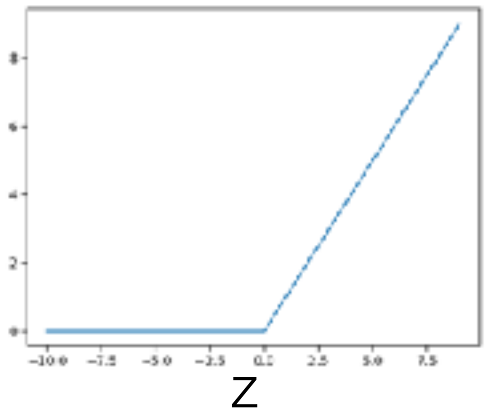


$$\phi(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$\frac{\partial}{\partial z}\phi(z) = \text{sech}^2(z)$$

New Activation: ReLU

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



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

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

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

Other Activation Functions

- Sigmoid Weighted Linear Unit **SiLU** (also called Swish)
- Mixing of sigmoid, σ, and ReLU

$$\phi(z) = \sigma(z) \cdot z$$

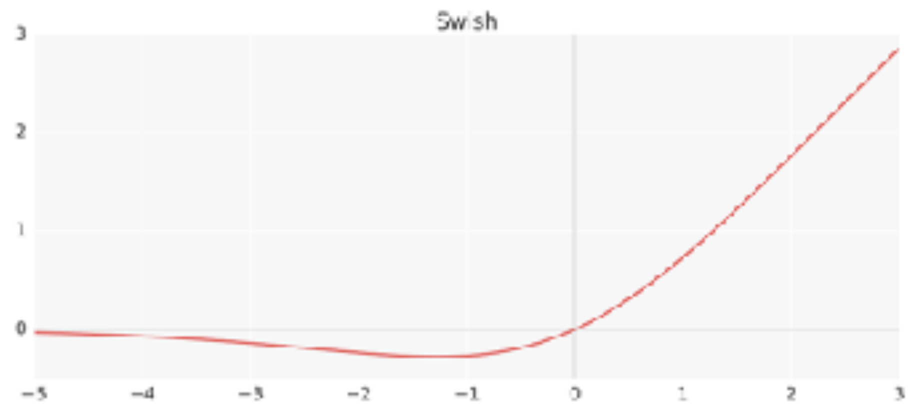


Figure 1: The Swish activation function.

$$\frac{\partial \phi(z)}{\partial z} = \frac{\partial}{\partial z}\sigma(z) \cdot z$$

$$= z \cdot \left[\frac{\partial}{\partial z}\sigma(z)\right] + \sigma(z) \cdot \left[\frac{\partial}{\partial z} z\right]$$

$$= z \cdot \sigma(z)(1 - \sigma(z)) + \sigma(z)$$

$$= z \cdot \sigma(z) + \sigma(z) \cdot (1 - z \cdot \sigma(z))$$

$$= \phi(z) + \sigma(z) \cdot (1 - \phi(z))$$

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

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

Glorot and He Initialization

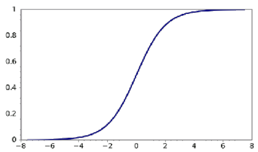
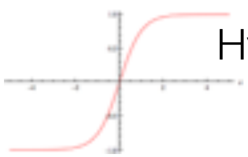
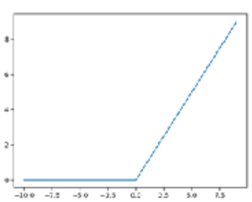
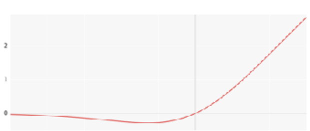
We have solved this assuming the activation output is in the range -4 to 4 (for a sigmoid) and assuming that we use Gaussian for sampling.

This range is different depending on the activation and assuming Gaussian or Uniform sampling.

	Uniform	Gaussian
Tanh	$w_{ij}^{(L)} \sim \sqrt{\frac{6}{n^{(L)} + n^{(L+1)}}}$	$w_{ij}^{(L)} \sim \sqrt{\frac{2}{n^{(L)} + n^{(L+1)}}}$
Sigmoid	$w_{ij}^{(L)} \sim 4\sqrt{\frac{6}{n^{(L)} + n^{(L+1)}}}$	$w_{ij}^{(L)} \sim 4\sqrt{\frac{2}{n^{(L)} + n^{(L+1)}}}$
ReLU SiLU	$w_{ij}^{(L)} \sim \sqrt{2}\sqrt{\frac{6}{n^{(L)} + n^{(L+1)}}}$	$w_{ij}^{(L)} \sim \sqrt{2}\sqrt{\frac{2}{n^{(L)} + n^{(L+1)}}}$

Summarized by Glorot and He

Activations Summary

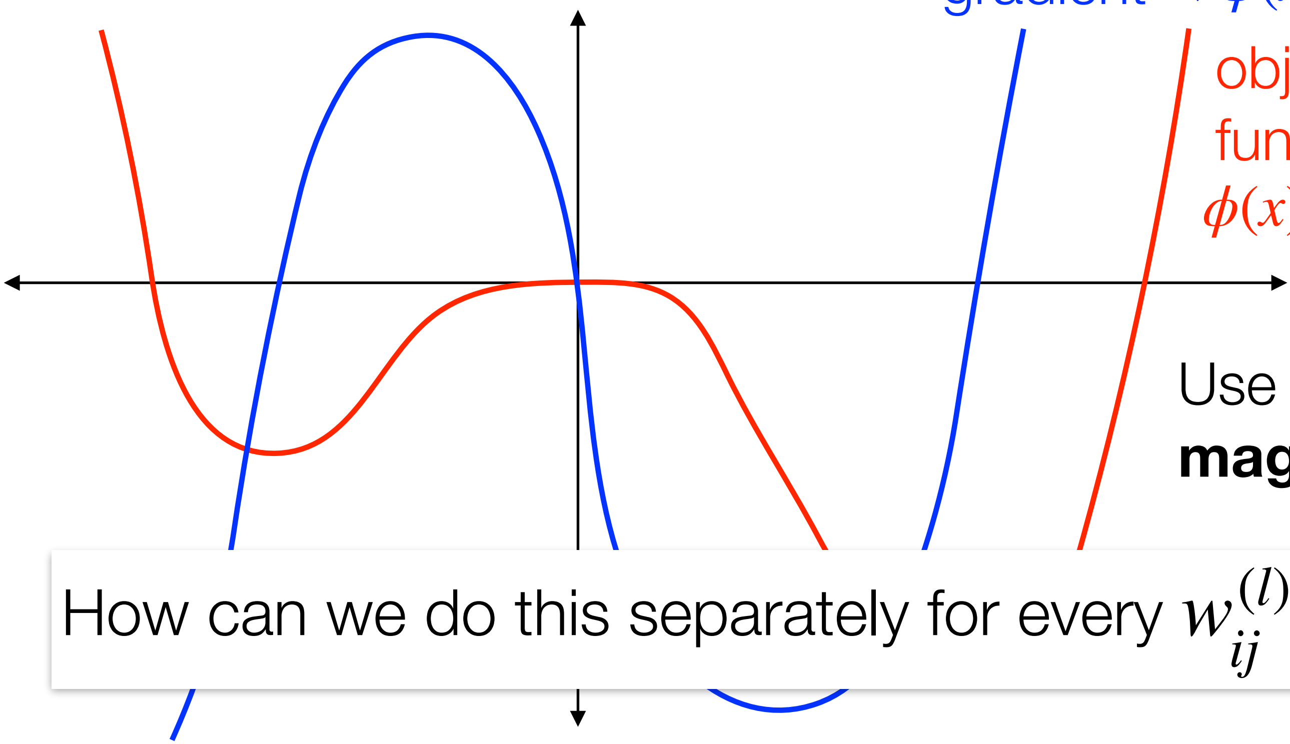
	Definition	Derivative	Weight Init *(Uniform Bounds)*
 Sigmoid	$\phi(z) = \dfrac{1}{1+e^{-z}}$	$\nabla\phi(z) = a(1-a)$	$w_{ij}^{(L)} \sim \pm 4\sqrt{\dfrac{6}{n^{(L)}+n^{(L+1)}}}$
 Hyperbolic Tangent	$\phi(z) = \dfrac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$\nabla\phi(z) = \dfrac{4}{(e^{z}+e^{-z})^2}$	$w_{ij}^{(L)} \sim \pm \sqrt{\dfrac{6}{n^{(L)}+n^{(L+1)}}}$
 ReLU	$\phi(z) = \begin{cases} z, \text{ if } z>0 \\ 0, \text{ else} \end{cases}$	$\nabla\phi(z) = \begin{cases} 1, \text{ if } z>0 \\ 0, \text{ else} \end{cases}$	$w_{ij}^{(L)} \sim \pm \sqrt{2}\sqrt{\dfrac{6}{n^{(L)}+n^{(L+1)}}}$
 SiLU	$\phi(z) = \dfrac{z}{1+e^{-z}}$	$\nabla\phi(z) = \phi(z) + \sigma(z)\cdot(1-\phi(z))$	

More Adaptive Optimization

Going beyond
changing the learning rate

Be adaptive based on Gradient Magnitude?

- Decelerate down regions that are steep
- Accelerate on plateaus



Use **inverse** of **magnitude** of **gradient**!

How can we do this separately for every $w_{ij}^{(l)}$ in every $\mathbf{W}^{(l)}$?

Momentum: be robust to **abrupt changes** in **steepness** (accumulate inverse magnitudes)

http://www.technologyuk.net/mathematics/differential-calculus/higher-derivatives.shtml

Be adaptive based on Gradient Magnitude?

Inverse magnitude of gradient in multiple directions?

$$\mathbf{W}_{k+1} \leftarrow \mathbf{W}_k + \eta \frac{1}{\sqrt{\mathbf{G}_k + \epsilon}} \odot \nabla J(\mathbf{W}_k)$$

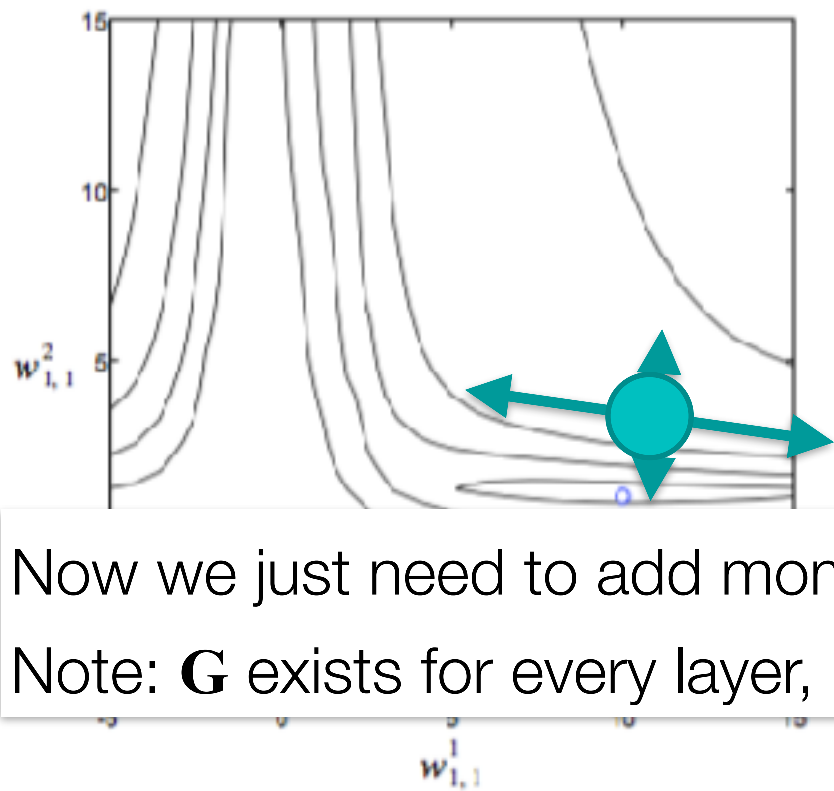
new matrix for normalizing

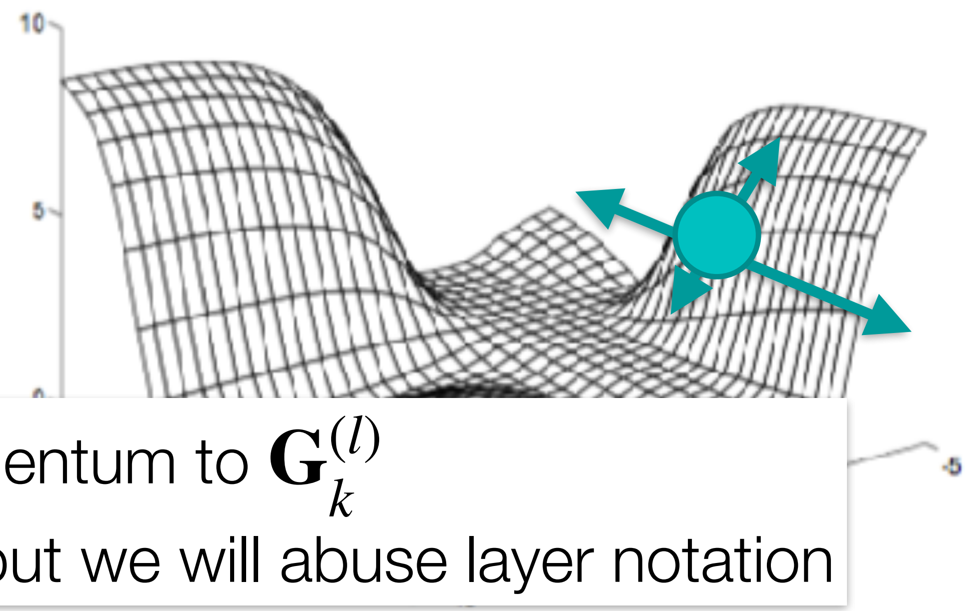
gradient

$$\mathbf{G}_k = \nabla J(\mathbf{W}_k) \odot \nabla J(\mathbf{W}_k)$$

same size as $\mathbf{W}$

```
G = gradW1 * gradW1
W1 += eta*gradW1 / sqrt(G+eps)
```





Now we just need to add momentum to $\mathbf{G}_k^{(l)}$

Note: $\mathbf{G}$ exists for every layer, but we will abuse layer notation

Common Adaptive Strategies $\mathbf{W}_{k+1} = \mathbf{W}_k - \eta \cdot \rho_k$

Adjust each element of gradient by the steepness

• AdaGrad all operations are per element	$\rho_k = \frac{1}{\sqrt{\mathbf{G}_k + \epsilon}} \odot \nabla J(\mathbf{W}_k)$	where $\mathbf{G}_k = \gamma \cdot \mathbf{G}_{k-1} + \nabla J(\mathbf{W}_k) \odot \nabla J(\mathbf{W}_k)$
• RMSProp all operations are per element	$\rho_k = \frac{1}{\sqrt{\mathbf{V}_k + \epsilon}} \odot \nabla J(\mathbf{W}_k)$	$\mathbf{G}_k = \nabla J(\mathbf{W}_k) \odot \nabla J(\mathbf{W}_k)$ $\mathbf{V}_k = \gamma \cdot \mathbf{V}_{k-1} + (1-\gamma) \cdot \mathbf{G}_k$
• AdaDelta all operations are per element	$\rho_k = \frac{\mathbf{M}_k}{\sqrt{\mathbf{V}_k + \epsilon}}$	$\mathbf{M}_{k+1} = \gamma \cdot \mathbf{M}_k + (1-\gamma) \cdot \nabla J(\mathbf{W}_k)$
• AdaM	$\mathbf{G}$ updates with decaying momentum of J and J^2	
• NAdaM	same as Adam, but with nesterov's acceleration	

None of these are **"one-size-fits-all"** because the space of neural network **optimization varies** by problem, AdaM is **popular** but **not a panacea**

Adaptive Momentum

All operations are element wise:

$$\beta_1 = 0.9,\ \beta_2 = 0.999,\ \eta = 0.001,\ \epsilon = 10^{-8}$$

$$k = 0,\ \mathbf{M}_0 = \mathbf{0},\ \mathbf{V}_0 = \mathbf{0}$$

Published as a conference paper at ICLR 2015

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Diederik P. Kingma*
University of Amsterdam, OpenAI

Jimmy Lei Ba*
University of Toronto

For each epoch:

update iteration $k \leftarrow k + 1$

get gradient $\nabla J(\mathbf{W}_k)$

for large k, $\hat{\mathbf{M}} \approx \mathbf{M},\ \hat{\mathbf{V}} \approx \mathbf{V}$

accumulated gradient $\mathbf{M}_k \leftarrow \beta_1 \cdot \mathbf{M}_{k-1} + (1 - \beta_1) \cdot \nabla J(\mathbf{W}_k)$

accumulated squared gradient $\mathbf{V}_k \leftarrow \beta_2 \cdot \mathbf{V}_{k-1} + (1 - \beta_2) \cdot \nabla J(\mathbf{W}_k) \odot \nabla J(\mathbf{W}_k)$

boost moments magnitudes (notice k in exponent)

$$\hat{\mathbf{M}}_k \leftarrow \frac{\mathbf{M}_k}{(1 - [\beta_1]^k)} \qquad \hat{\mathbf{V}}_k \leftarrow \frac{\mathbf{V}_k}{(1 - [\beta_2]^k)}$$

update gradient, normalized by second moment similar to AdaDelta

$$\mathbf{W}_k \leftarrow \mathbf{W}_{k-1} - \eta \cdot \frac{\hat{\mathbf{M}}_k}{\sqrt{\hat{\mathbf{V}}_k + \epsilon}}$$

gradient with momentum

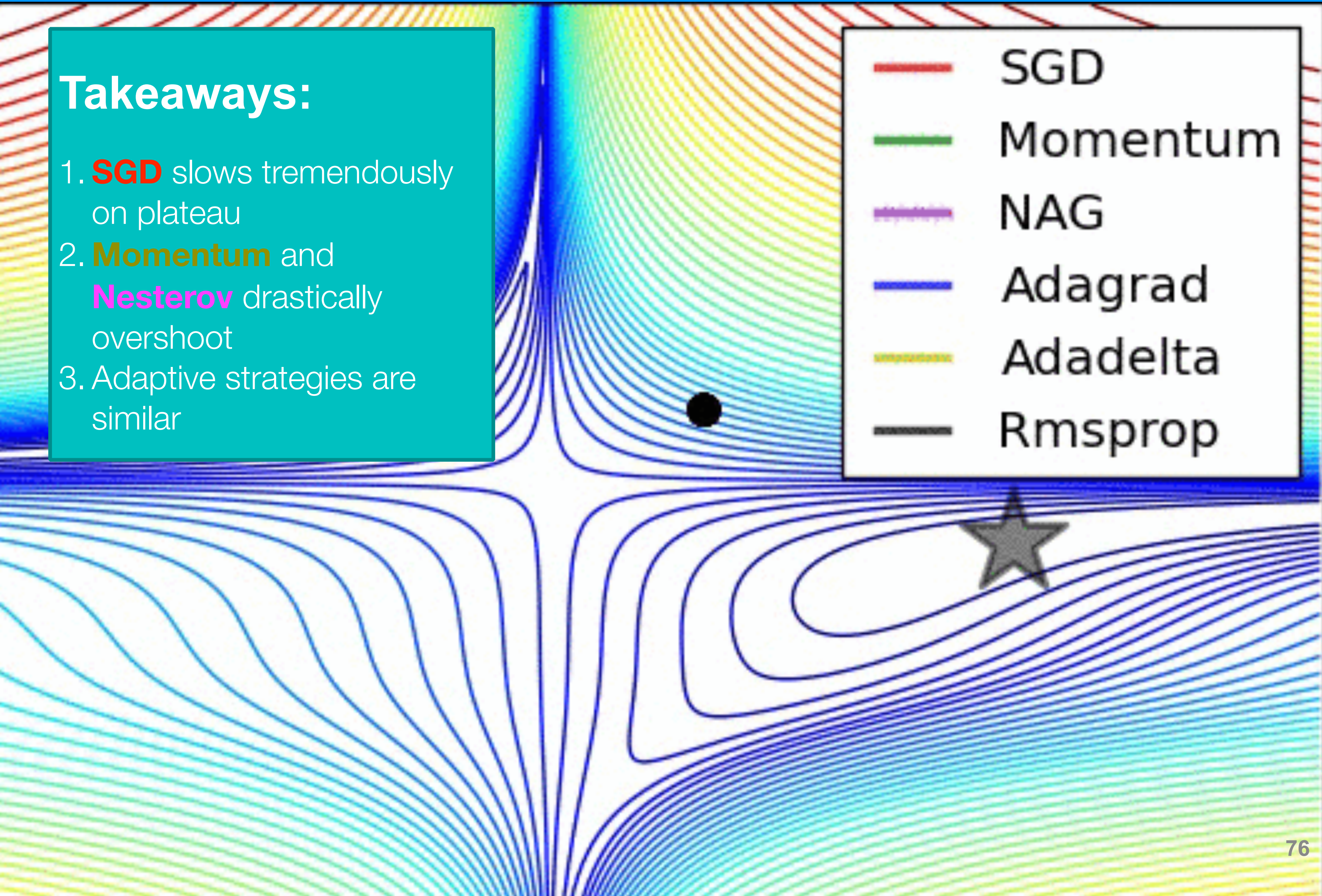
squared magnitude normalizer

Visualization of Optimization

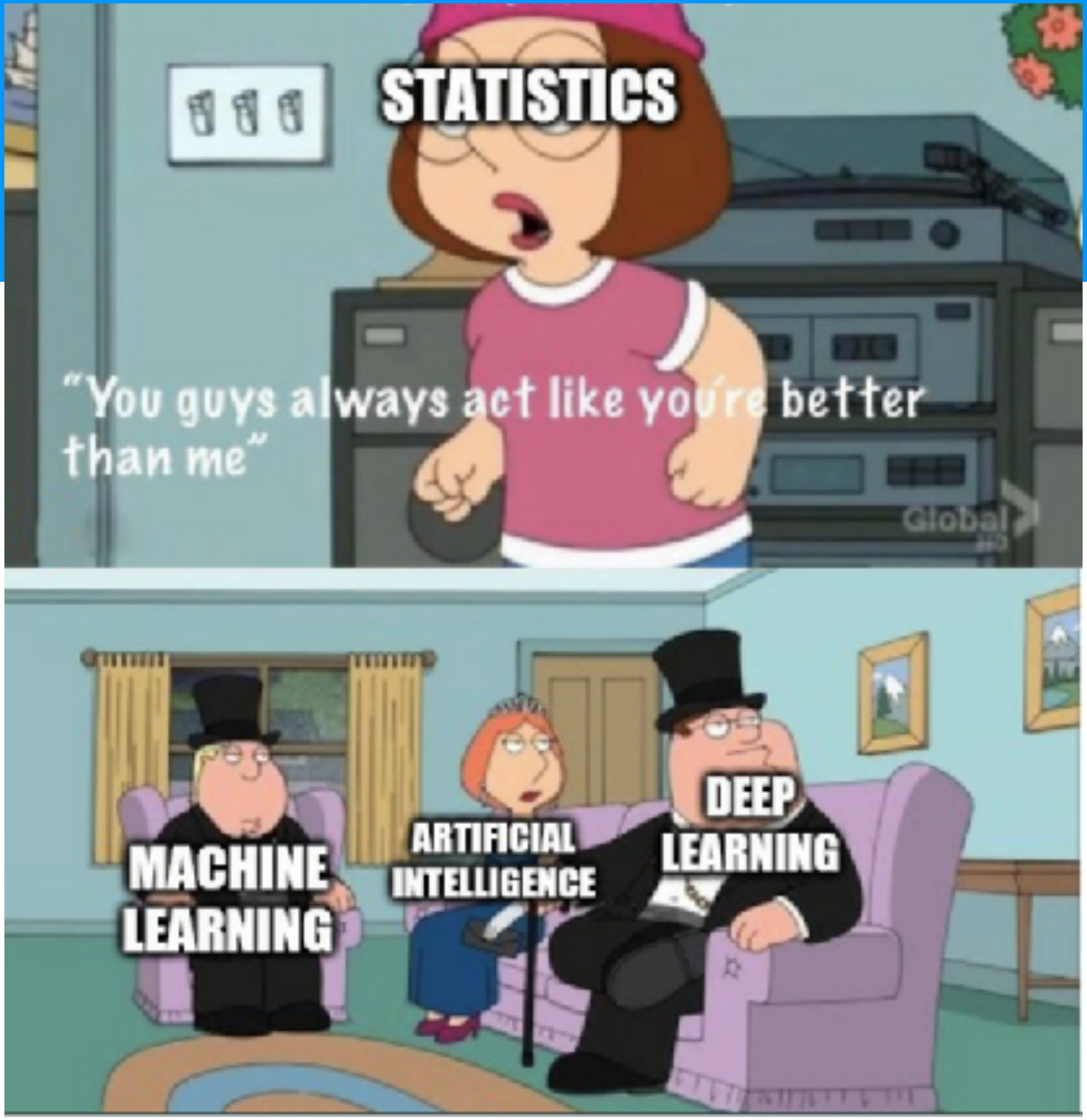
https://ruder.io/optimizing-gradient-descent/

Takeaways:

1. **SGD** slows tremendously on plateau
2. **Momentum** and **Nesterov** drastically overshoot
3. Adaptive strategies are similar



Review


STATISTICS
"You guys always act like you're better than me"
Global
MACHINE LEARNING
ARTIFICIAL INTELLIGENCE
DEEP LEARNING

Review

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \rho_k$$

- Cross entropy

$$\mathbf{A}^{(3)} - \mathbf{Y}$$

new final layer update

- Momentum $\rho_k = \alpha \nabla J(\mathbf{W}_k) + \beta \nabla J(\mathbf{W}_{k-1})$

- Nesterov's Accelerated Gradient

$$\rho_k = \underbrace{\beta \nabla J\left(\mathbf{W}_k + \alpha \nabla J(\mathbf{W}_{k-1})\right)}_{\text{step twice}} + \alpha \nabla J(\mathbf{W}_{k-1})$$

- Mini-batching

←all data→

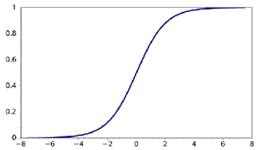
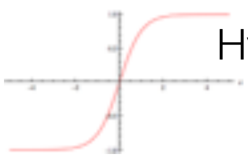
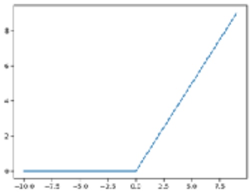
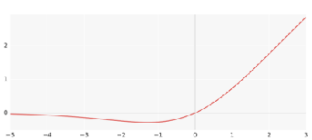
	batch 1	batch 2	batch 3	batch 4	batch 5	batch 6	batch 7	batch 8	batch 9
Epoch 1									
Epoch 2									
Epoch 3									
Epoch 4									
...									

shuffle ordering each epoch and update W's after each batch

- Learning rate adaptation (eta)

$$\eta_e = \eta_0^{(1+e\cdot\epsilon)} \qquad \eta_e = \eta_0 \cdot d^{\lfloor \frac{e}{e_d} \rfloor}$$

Review: Activations Summary

	Definition	Derivative	Weight Init *(Uniform Bounds)*
 Sigmoid	$\phi(z) = \frac{1}{1+e^{-z}}$	$\nabla\phi(z) = a(1-a)$	$w_{ij}^{(L)} \sim \pm 4\sqrt{\frac{6}{n^{(L)}+n^{(L+1)}}}$
 Hyperbolic Tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$\nabla\phi(z) = \frac{4}{(e^z + e^{-z})^2}$	$w_{ij}^{(L)} \sim \pm\sqrt{\frac{6}{n^{(L)}+n^{(L+1)}}}$
 ReLU	$\phi(z) = \begin{cases} z, \text{ if } z > 0 \\ 0, \text{ else} \end{cases}$	$\nabla\phi(z) = \begin{cases} 1, \text{ if } z > 0 \\ 0, \text{ else} \end{cases}$	$w_{ij}^{(L)} \sim \pm\sqrt{2}\sqrt{\frac{6}{n^{(L)}+n^{(L+1)}}}$
 SiLU	$\phi(z) = \frac{z}{1+e^{-z}}$	$\nabla\phi(z) = \phi(z) + \sigma(z)\cdot(1-\phi(z))$	

08a. Practical_NeuralNetsWithBias.ipynb

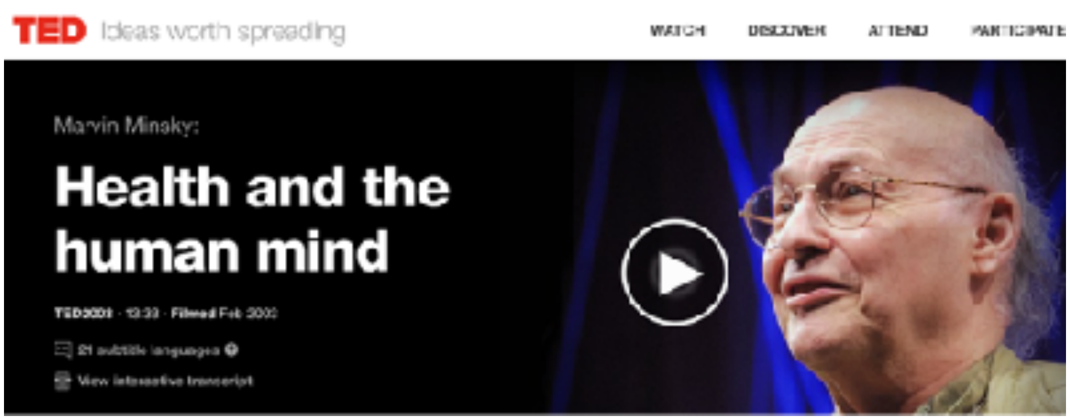
~~Momentum~~
~~Cooling~~
~~Cross Entropy~~
~~Smarter Weight Initialization~~
ReLU Nonlinearities
Adaptive training with AdaGrad

Revisiting Universality (if time)

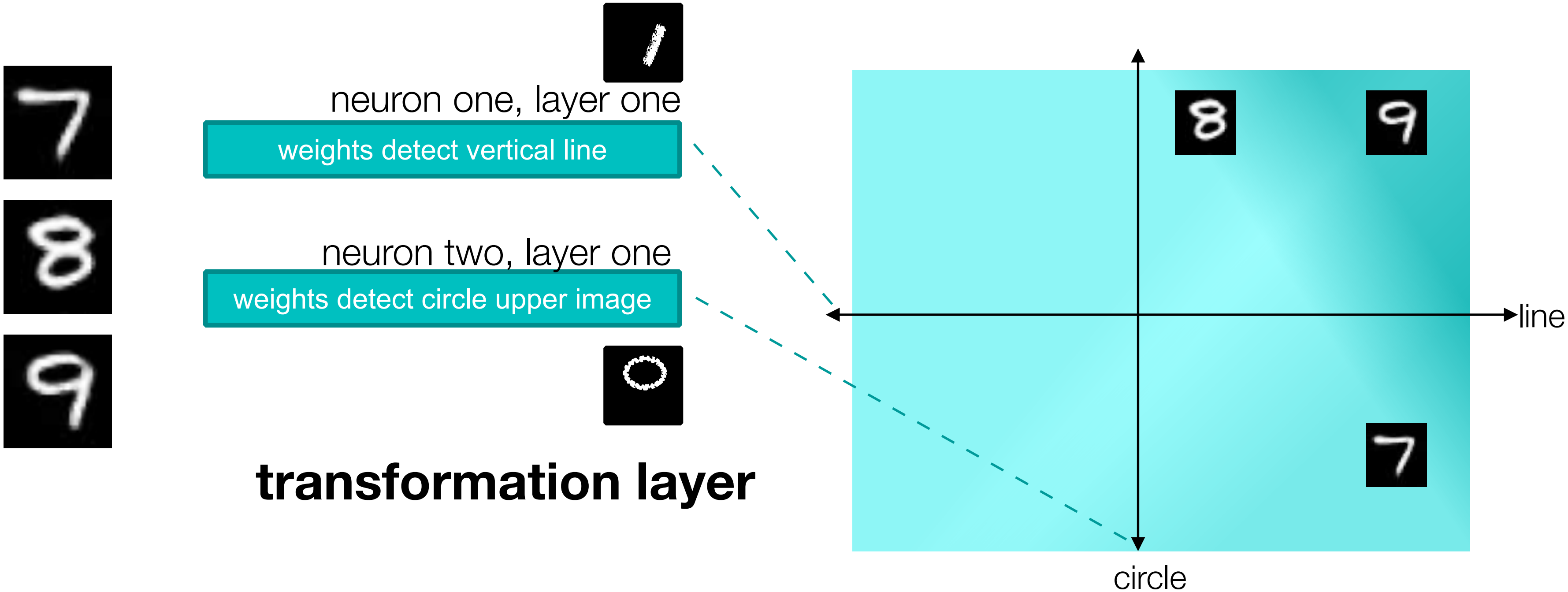
- Neural networks can separate any data through multiple layers. The true realization of Rosenblatt:

  > "Given an elementary α-perceptron, a stimulus world W, and any classification C(W) for which a solution exists; let all stimuli in W occur in any sequence, provided that each stimulus must reoccur in finite time; then beginning from an arbitrary initial state, an error correction procedure will always yield a solution to C(W) in finite time…"

- **Universality**: No matter what function we want to compute, we know that there is a neural network which can do the job.



Universality



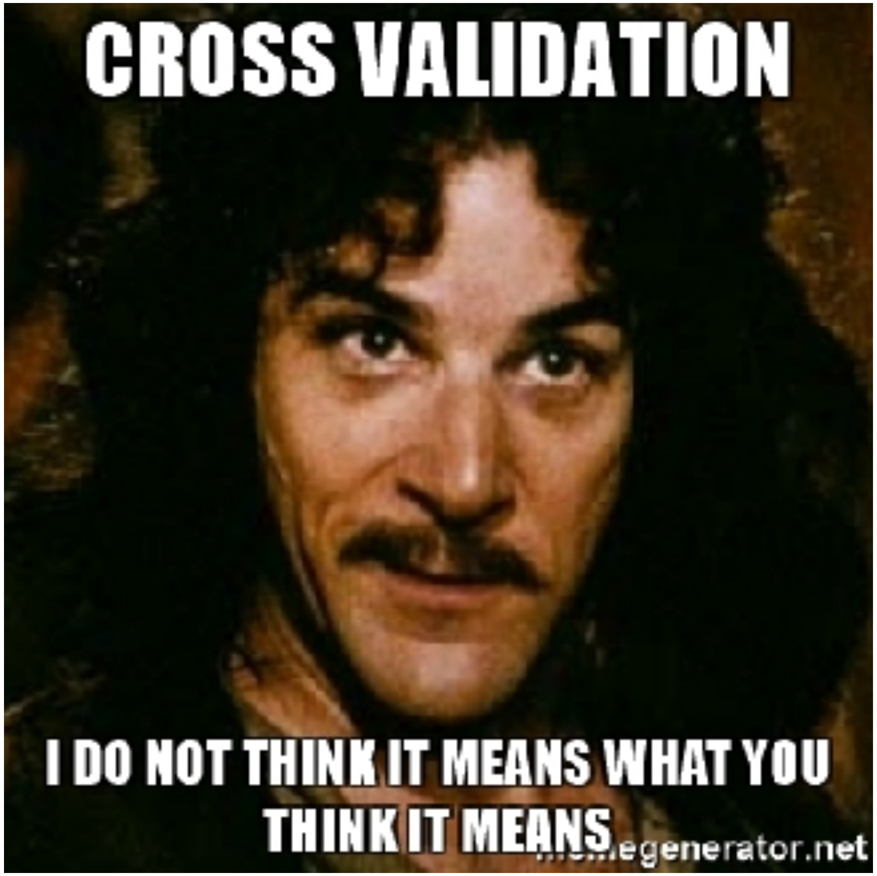
classification layer is like
one versus all logistic regression

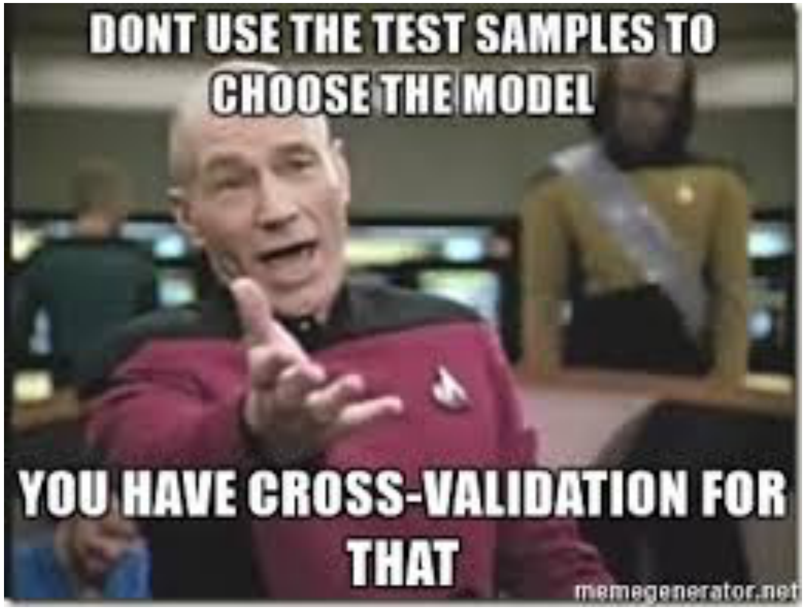
- One nonlinear hidden layer with an output layer can perfectly train any problem with enough data, but might just be memorizing…
  - … it might be better to have even more layers for decreased computation and generalizability

Agenda

- Now: Cross validation + Lab 4 Town Hall
- Next Time: **Final Flipped Module!**
- Then: **Deep Learning**

Revisiting Cross Validation (if time)





Review: Grid Searching

Trying to find the best parameters

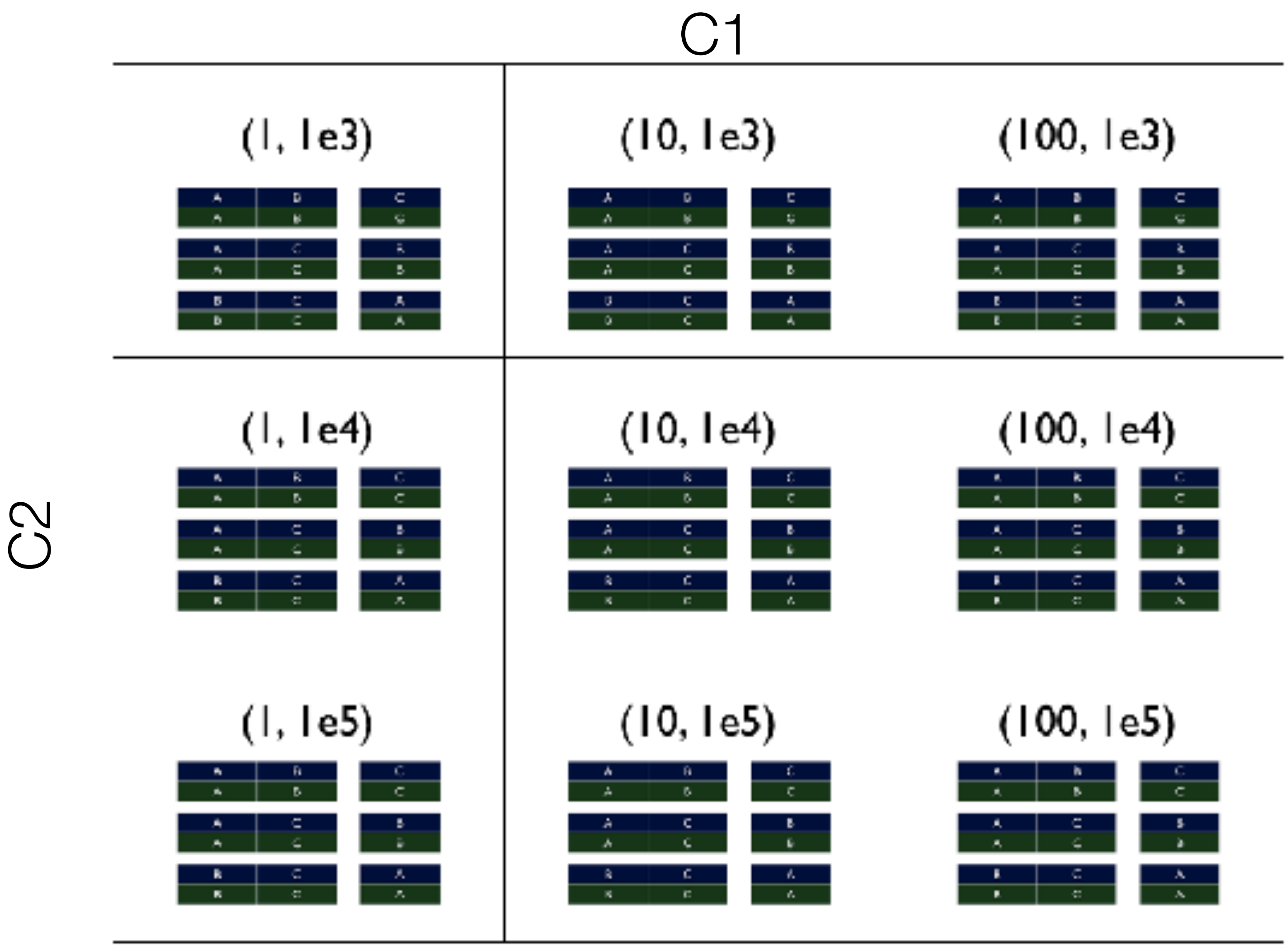
```
NN: C1=[1, 10, 100] C2=[1e3,1e4,1e5]
```

C1

C2			
	(1, 1e3)	(10, 1e3)	(100, 1e3)
	(1, 1e4)	(10, 1e4)	(100, 1e4)
	(1, 1e5)	(10, 1e5)	(100, 1e5)

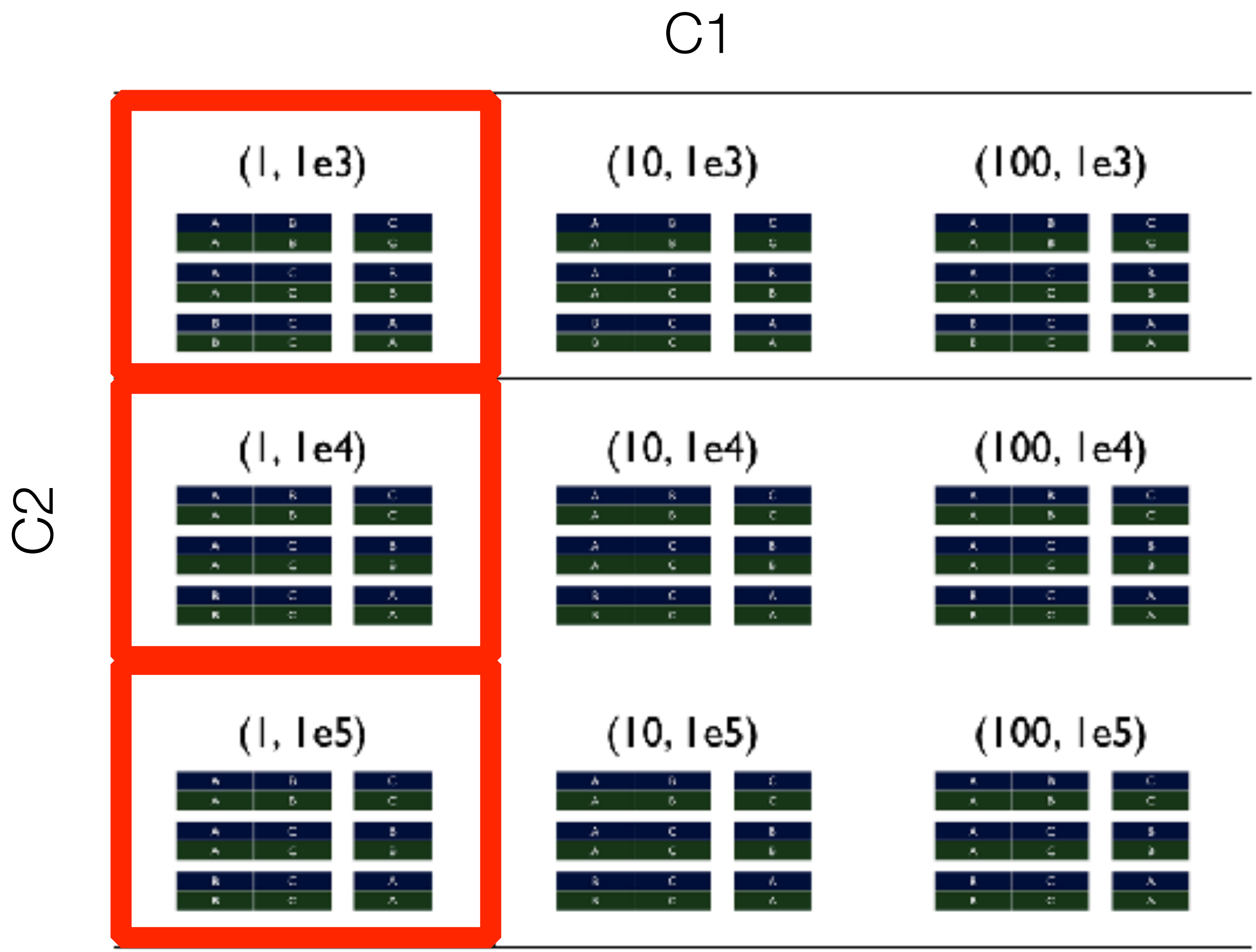
Review: Grid Searching

For each value, want to run cross validation…



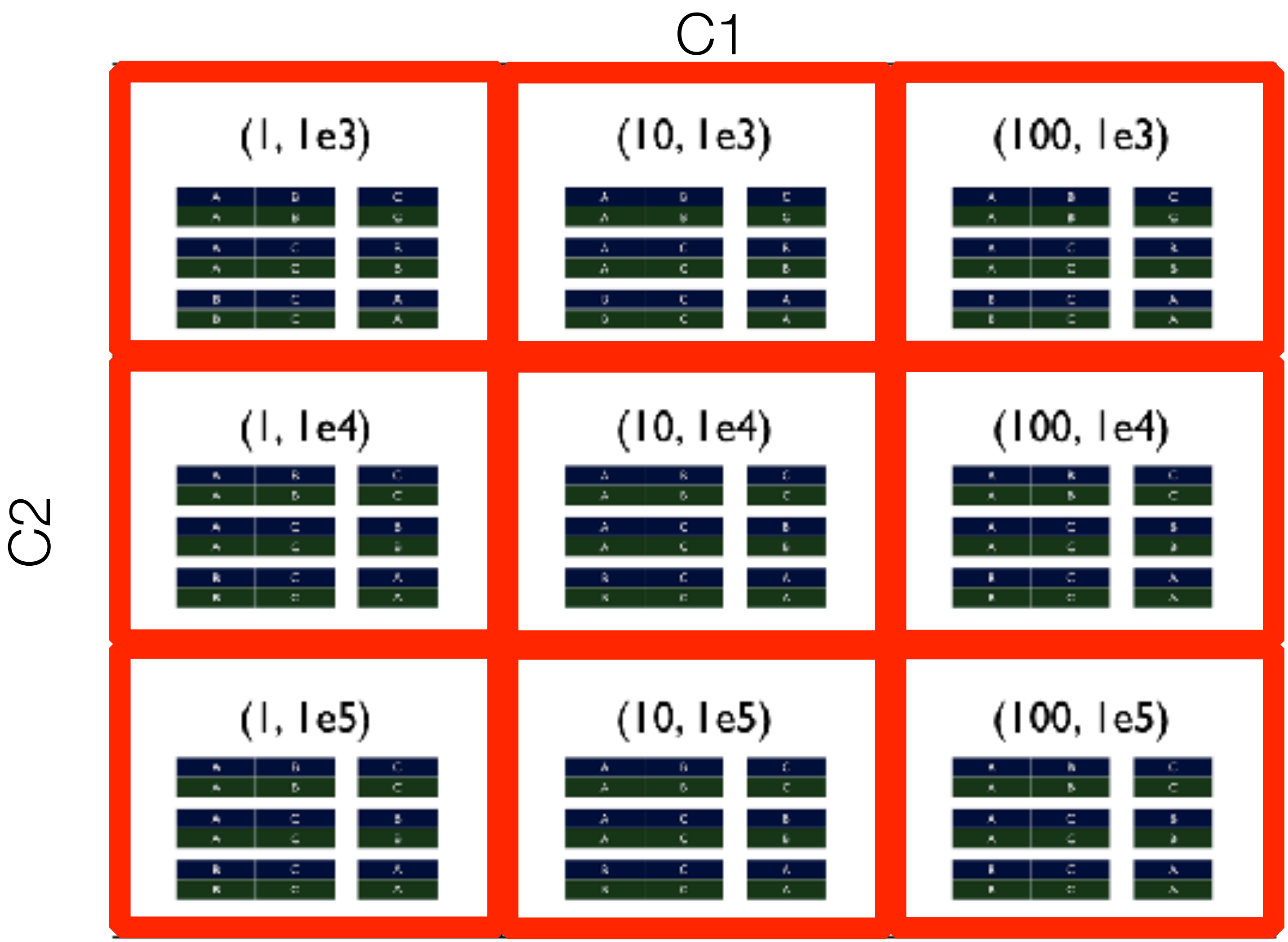
Review: Grid Searching

Could perform iteratively



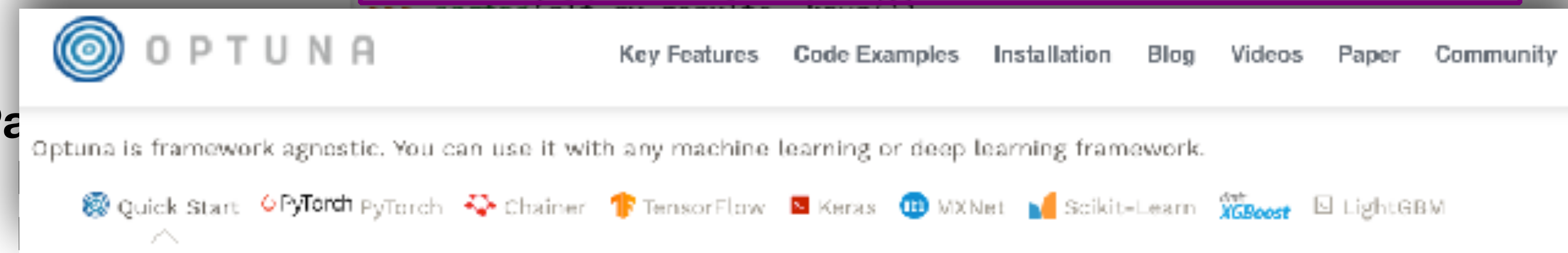
Review: Grid Searching

or at random…



Review: Grid Searches in Scikit-learn

```
>>> from sklearn import svm, datasets
>>> from sklearn.model_selection import GridSearchCV
>>> iris = datasets.load_iris()
>>> parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
>>> svc = svm.SVC()
>>> clf = GridSearchCV(svc, parameters)
>>> clf.fit(iris.data, iris.target)
GridSearchCV(estimator=SVC(),
             param_grid={'C': [1, 10], 'kernel': ('linear', 'rbf')})
```



Pa

-
-

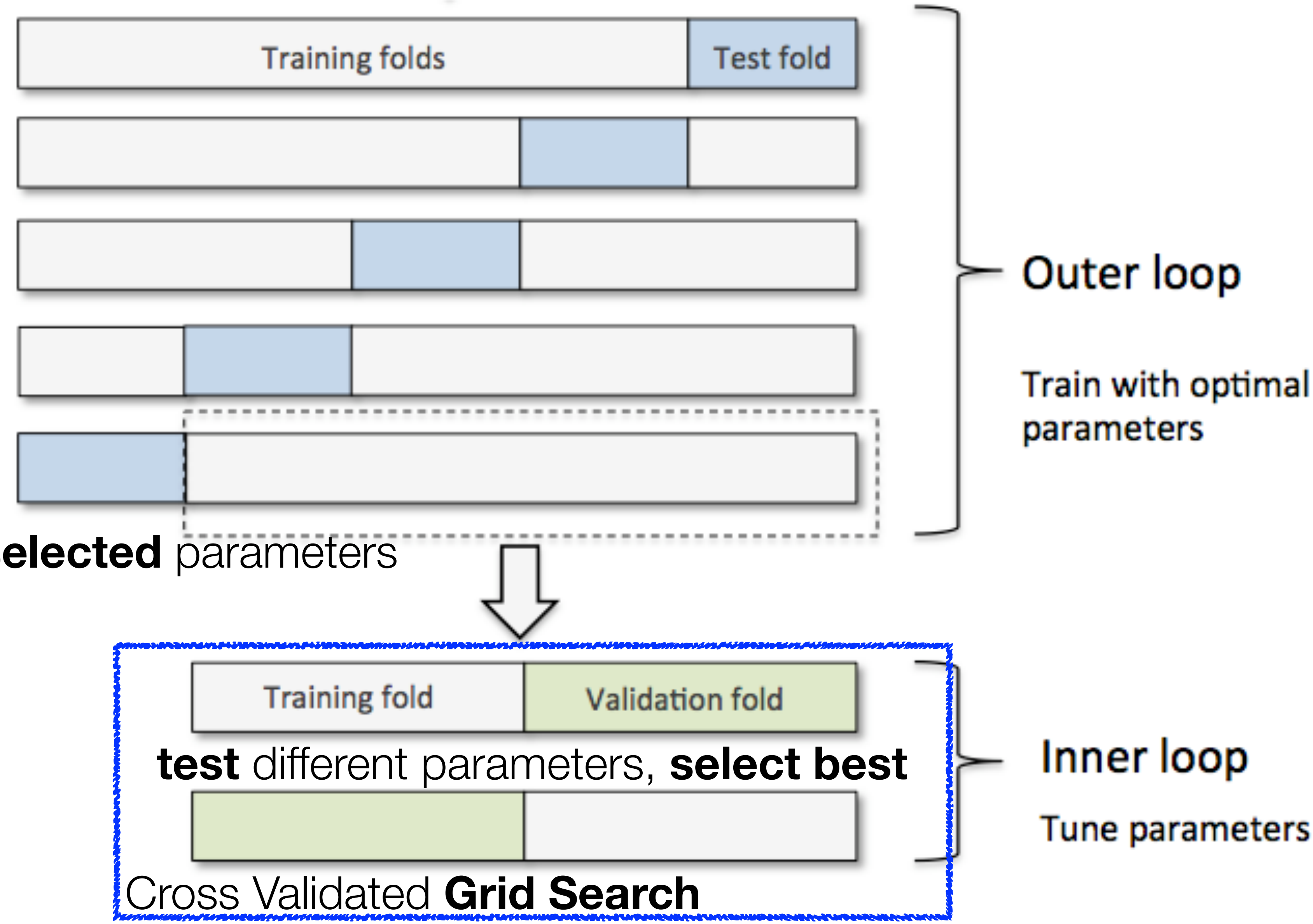
values, sampled

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.model_selection import RandomizedSearchCV
>>> from scipy.stats import uniform
>>> iris = load_iris()
>>> logistic = LogisticRegression(solver='saga', tol=1e-2, max_iter=200,
...                               random_state=0)
>>> distributions = dict(C=uniform(loc=0, scale=4),
...                      penalty=['l2', 'l1'])
>>> clf = RandomizedSearchCV(logistic, distributions, random_state=0)
>>> search = clf.fit(iris.data, iris.target)
>>> search.best_params_
{'C': 2..., 'penalty': 'l1'}
```

Review: Self Test

- Using the grid search parameters and testing on the same set…
- Is this **data snooping**?
  - A. True, this is snooping because it uses test set to define parameters
  - B. True, this is snooping because we can no longer reliably define the expected performance on new data
  - C. False, this is not snooping because we still separated train and test data
  - D. False, this is not snooping because hyper parameters are not trainable

A Costly Solution: Nested Cross Validation



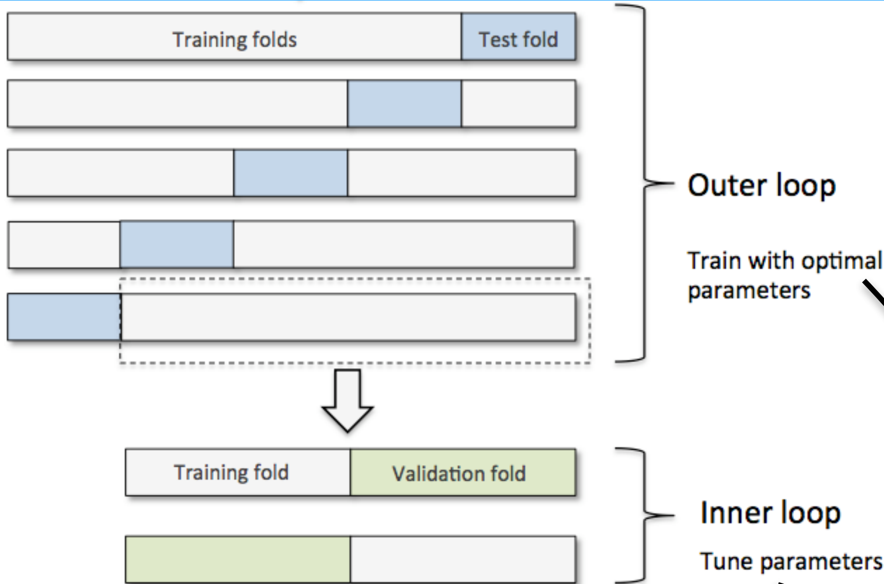
use **selected** parameters

test different parameters, **select best**

Cross Validated **Grid Search**

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

Review: 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 for Comparing Performance

Few assumptions, **Null hypothesis**: predictions are not different!

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

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

McNemar and Edwards, 1948

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

If predictions are drawn from the same distributions, then this equation follows χ **squared statistic with one DOF**

Steps:

1. Compare each model's predictions on the same test data (2x2 matrix)
2. Calculate χ^2 statistic
3. Look up *critical value* associated with χ^2 statistic for given confidence
4. Are you confident enough to **reject the null hypothesis** that the performance is the same (p<0.05)?

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

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

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

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

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

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

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

Town Hall