Lecture Notes for

Machine Learning in Python

⟦👨‍🏫, 👨‍💻, 🐍, 👨‍🔬⟧

Professor Eric Larson

Adaptive Neural Network Optimization

Class Logistics and Agenda

- Agenda:
  - More optimization techniques
    - Review
    - Adaptive Learning Strategies
  - Town Hall for MLP

Class Overview, by topic

Table Data Visualization

Numpy, Pandas, Seaborn
Overviews with some in-depth discussion

Dimension Reduction and Image Processing

Scikit-learn, Scikit Image,
Intuition only, Some mathematics

Linear and Logistic Regression

Numpy, Recreate API for Scikit-learn
Detailed mathematics for simple optimization
intuition for advanced optimization

Neural Networks and Back Prop.

Numpy
Detailed mathematics for NN operations

Wide and Deep Networks

Convolutional Networks

Recurrent Networks

Keras, Tensorflow
Intuition, Detailed implement.

Ethics in Language Models

ConceptNet
Case studies

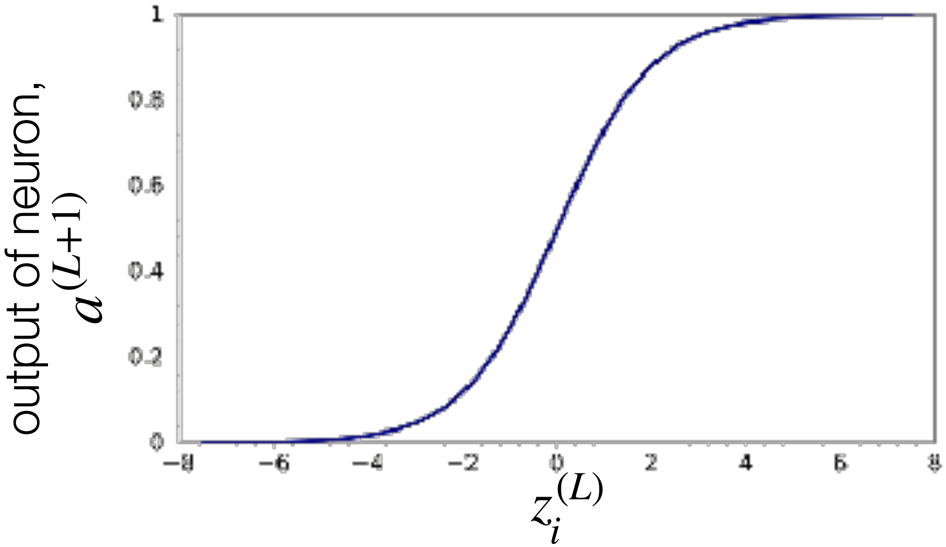
Practical Initialization of Architectures

Formative Self Test

- for adding Gaussian random variables, variances add together

$$\mathbf{a}^{(L+1)} = \phi(\mathbf{W}^{(L)}\mathbf{a}^{(L)})$$ assume each element of $\mathbf{a}$ is Gaussian

- If you initialized the weights, $\mathbf{W}$, with too large variance, you would expect the output of the neuron, $\mathbf{a}^{(L+1)}$, to be:
    - A. saturated to “1”
    - B. saturated to “0”
    - C. could either be saturated to “0” or “1”
    - D. would not be saturated

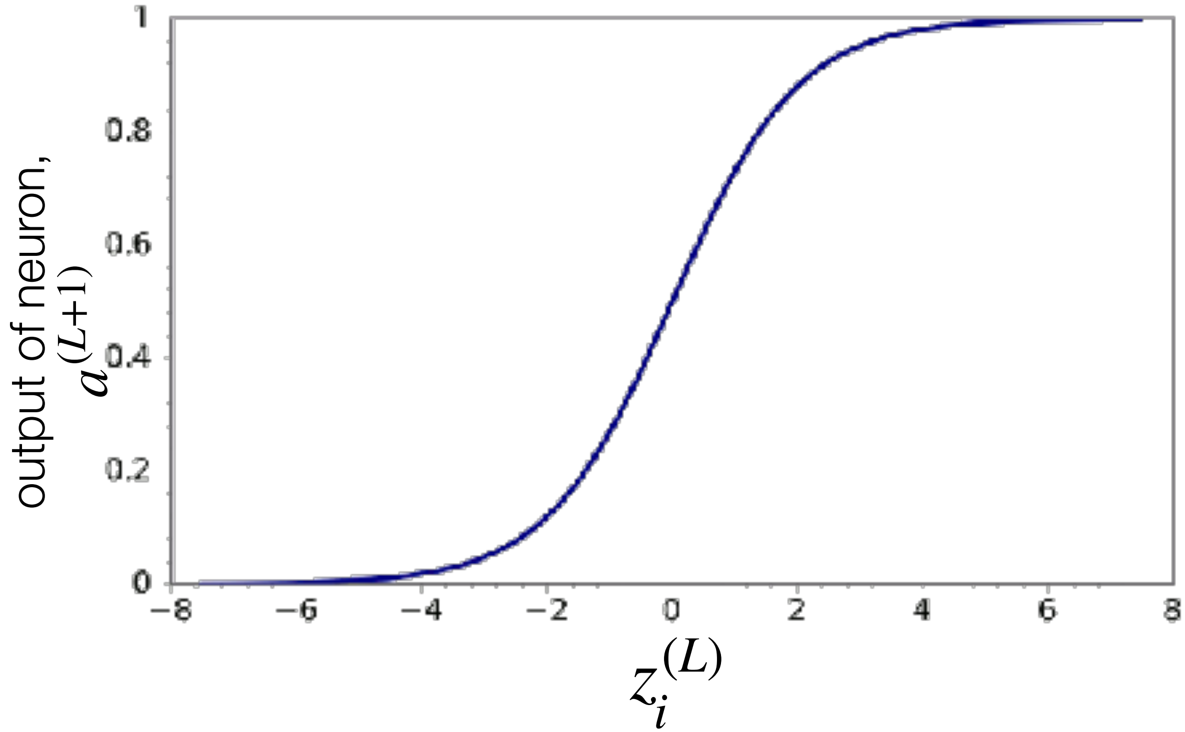


Formative Self Test

- for adding Gaussian distributions, variances add together

  $\mathbf{a}^{(L+1)}=\phi(\mathbf{W}^{(L)}\mathbf{a}^{(L)})$ assume each element of $\mathbf{a}$ is Gaussian

- What is the derivative of a saturated sigmoid neuron?
  - A. zero
  - B. one
  - C. $a \times (1 - a)$
  - D. it depends

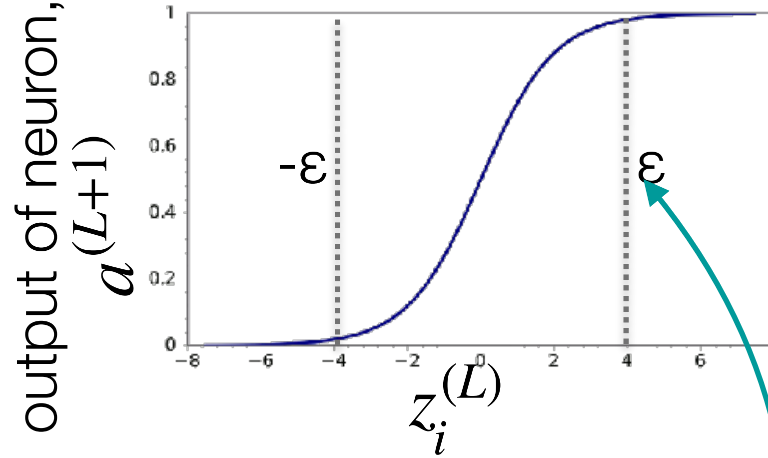


Practical Implementation of Architectures

- **Weight initialization**
  - try not to **saturate** your neurons right away!

$$\mathbf{a}^{(L+1)} = \phi(\mathbf{z}^{(L)})$$
$$\mathbf{z}^{(L)} = \mathbf{W}^{(L)}\mathbf{a}^{(L)} \qquad z_i^{(L)} = \sum_j^{n^{(L)}} w_{ij} a_j^{(L)}$$

each row is summed before sigmoid



want each $z_i^{(L)}$ to be between -ε<Σ<ε for no saturation

solution: squash initial weights magnitude

- one choice: each element of **W** selected from a Gaussian with **zero mean** and **specific standard deviation**

$$w_{ij}^{(L)} \approx \mathcal{N}\left(0,\, 4 \cdot \sqrt{\frac{1}{n^{(L)}}}\right)$$

For a sigmoid if, $-\epsilon < z_i^{(L)} < \epsilon$ where $\epsilon = 4$

then $a^{(L+1)}$ is well distributed [0,1]

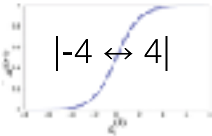
Glorot Weight Initialization

Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot JMLR 2010 Yoshua Bengio
DIRO, Université de Montréal, Montréal, Québec, Canada

Goal: We should not saturate **feedforward** or **back propagated** variance

Relate variance of current layer to variance in z, so $\sigma(z_i^{(L)})$ isn't saturated



try not to saturate

$$z_i^{(L)} = \sum_j^{n^{(L)}} w_{ij} a_j^{(L)}$$

break down feed forward by multiply in i^{th} row

$$\text{Var}[z_i^{(L)}] = \sum_j^{n^{(L)}} \underbrace{E[w_{ij}]^2 \text{Var}[a_j^{(L)}] + \text{Var}[w_{ij}] E[a_j^{(L)}]^2}_{\text{0, if uncorrelated}} + \text{Var}[w_{ij}] \text{Var}[a_j^{(L)}]$$

assume i.i.d.
expand variance calc

$$\text{Var}[z_i^{(L)}] = \sum_j^{n^{(L)}} \text{Var}[w_{ij}] \text{Var}[a_j^{(L)}] = n^{(L)} \text{Var}[w_{ij}] \underbrace{\text{Var}[a_j^{(L)}]}_{\approx 1} = n^{(L)} \text{Var}[w_{ij}]$$

$\text{Std}[z_i^{(L)}] = \sqrt{n^{(L)}} \cdot \text{Std}[w_{ij}]$

Want to keep ~4

$$\text{Std}[z_i^{(L)}] = 4 = \sqrt{n^{(L)}} \cdot \text{Std}[w_{ij}]$$

$$\text{Std}[w_{ij}] = 4 \cdot \sqrt{\frac{1}{n^{(L)}}}$$

$$w_{ij}^{(L)} \sim \mathcal{N}\left(0, 4 \cdot \sqrt{\frac{1}{n^{(L)}}}\right)$$

forward
from sigmoid

Glorot Weight Initialization

$$\text{Std}[z_i^{(L)}] = 4 = \sqrt{n^{(L)}} \cdot \text{Std}[w_{ij}]$$

$$w_{ij}^{(L)} \sim \mathcal{N}\left(0,\, 4 \cdot \sqrt{\frac{1}{n^{(L)}}}\right)$$

forward
from sigmoid

$$\mathbf{v}^{(L)} = \mathbf{a}^{(L)}(1 - \mathbf{a}^{(L)})\mathbf{W}^{(L)} \cdot \mathbf{v}^{(L+1)}$$

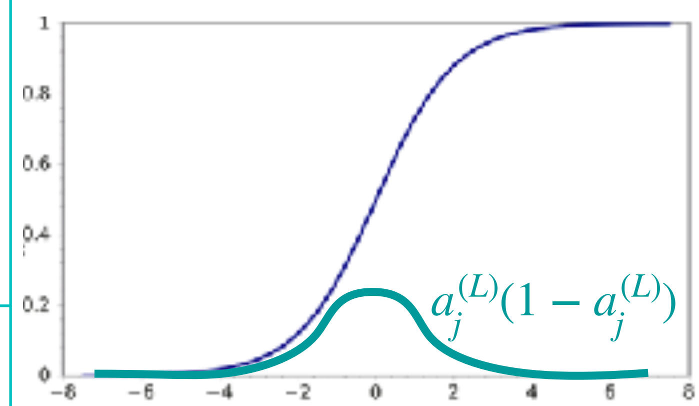
want to keep variance of $\mathbf{v}$ stable magnitude → stable gradient

Similar calculation for back prop.

$$\text{Var}[v_i^{(L)}] = n^{(L+1)}\text{Var}[w_{ij}]\text{Var}[v_j^{(L+1)} \cdot a_j^{(L)}(1 - a_j^{(L)})]$$

$$\text{Std}[v_i^{(L)}] = \sqrt{n^{(L+1)}} \cdot \text{Std}[w_{ij}] \cdot 0.25$$

want = 1



$$w_{ij}^{(L)} \sim \mathcal{N}\left(0,\, 4 \cdot \sqrt{\frac{1}{n^{(L+1)}}}\right)$$

backward
from sensitivity

$$w_{ij}^{(L)} \sim \mathcal{N}\left(0,\, 4 \cdot \sqrt{\frac{2}{n^{(L)} + n^{(L+1)}}}\right)$$

compromise

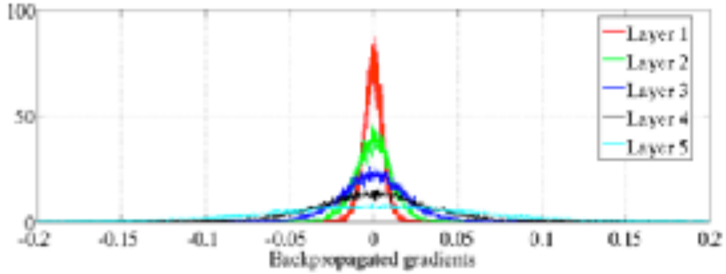
$$w_{ij}^{(L)} \sim U\left[\pm 4\sqrt{\frac{6}{n^{(L)} + n^{(L+1)}}}\right]$$

if drawn from uniform dist.

Glorot Weight Initialization

Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot **Yoshua Bengio**
DIRO, Université de Montréal, Montréal, Québec, Canada

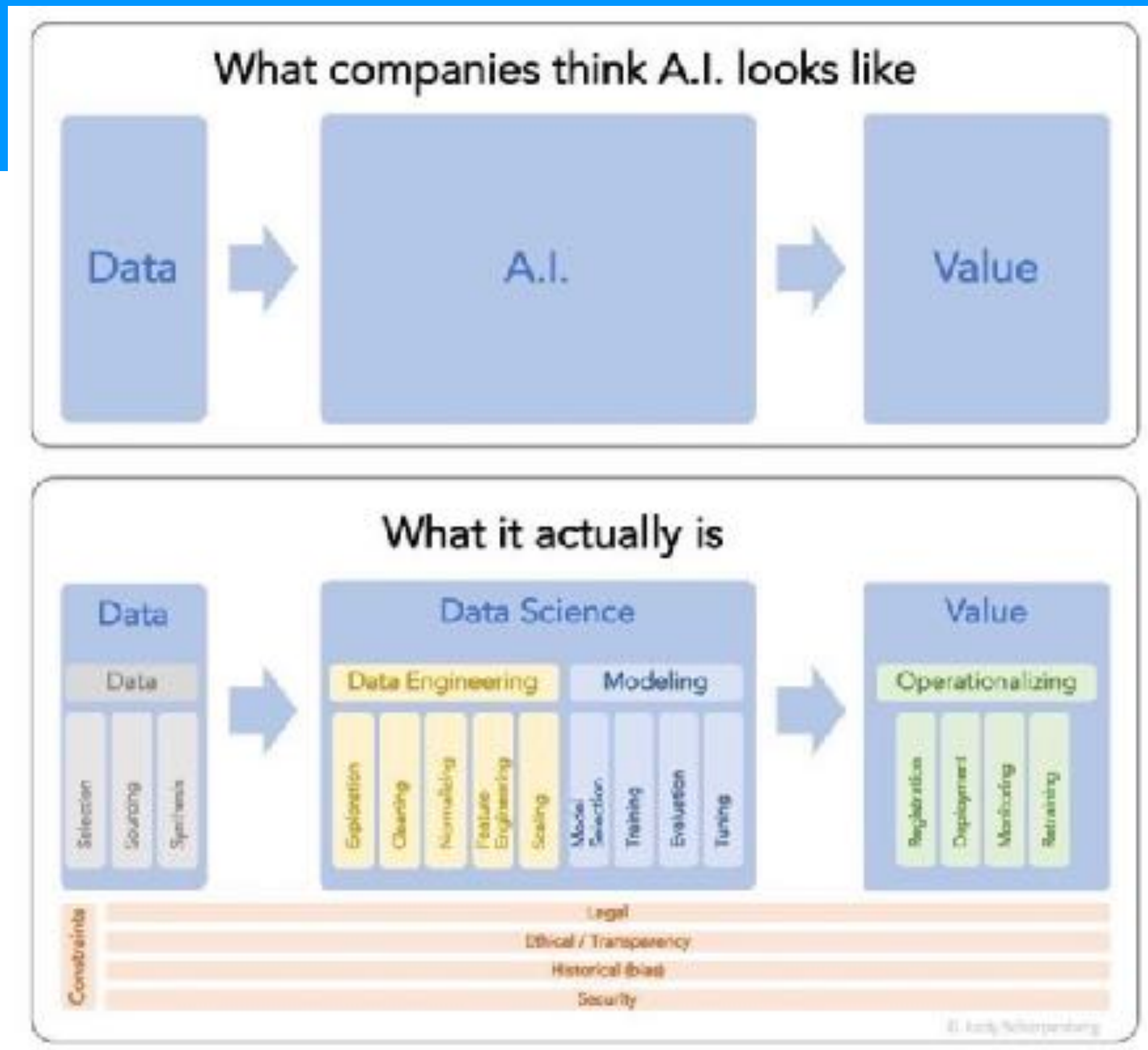


Starting gradient histograms per layer
standard initialization

Figure 7: *Back-propagated gradients normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized (bottom) initialization. Top: 0-peak decreases for higher layers.*

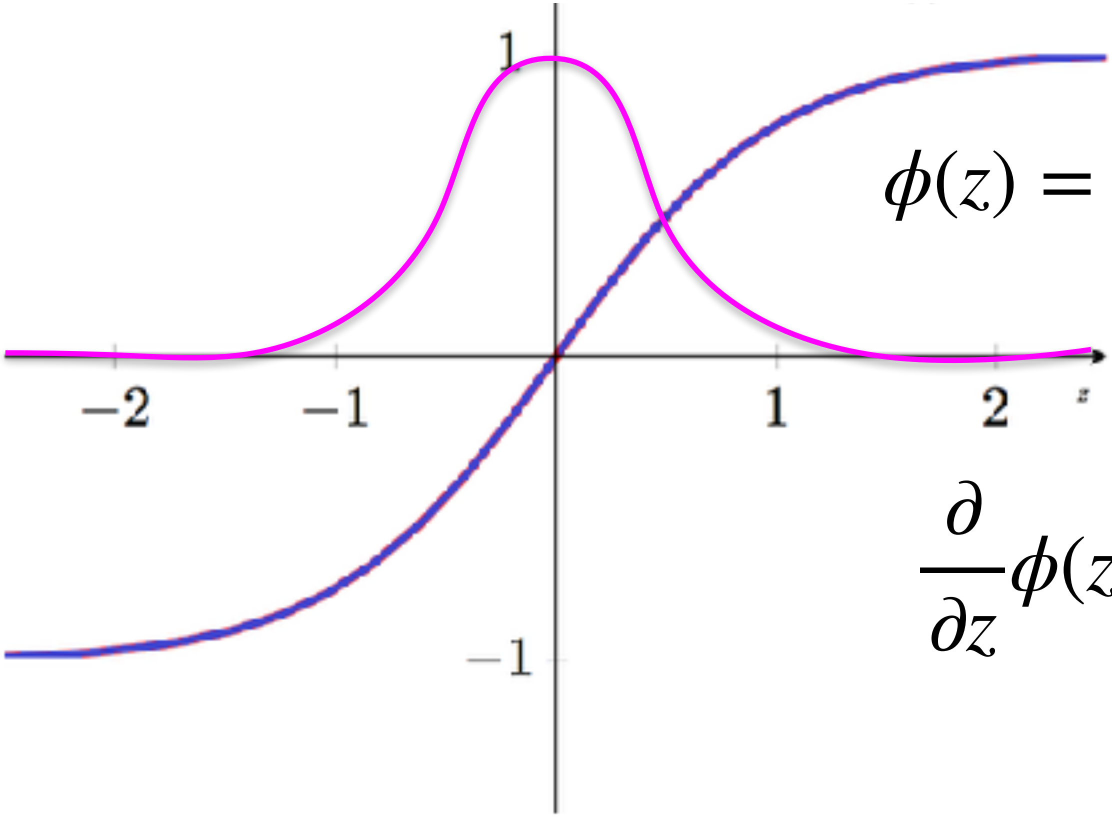
Smarter Weight Initialization

Beyond Sigmoid: Other Activations



New Activation: Hyperbolic Tangent

- Basically a sigmoid from -1 to 1

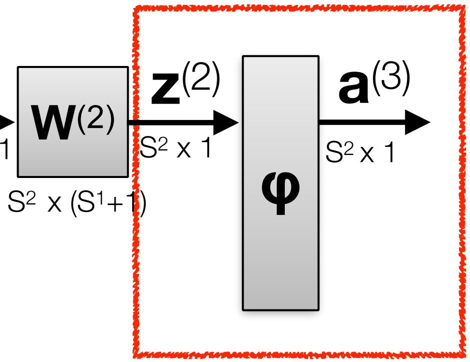


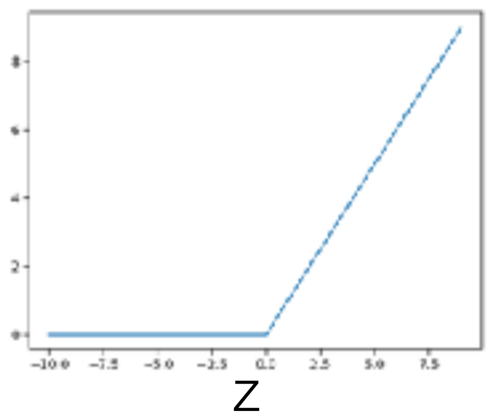
$$\phi(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

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

New Activation: ReLU

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



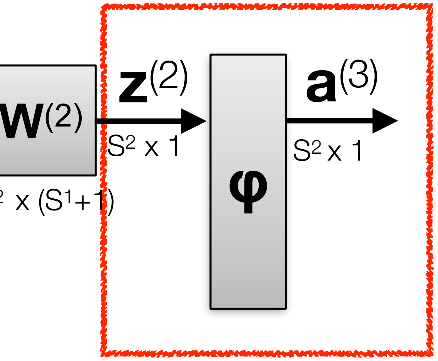


$$\phi(z) = \begin{cases} z, \text{ if } z > 0 \\ 0, \text{ else} \end{cases}$$

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

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

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

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

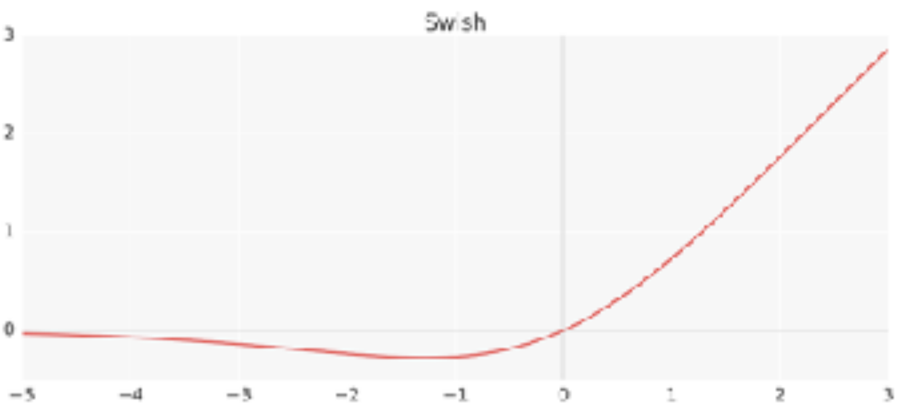


Figure 1: The Swish activation function.

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

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

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

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

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

Glorot and He Initialization

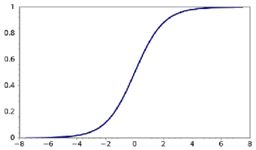
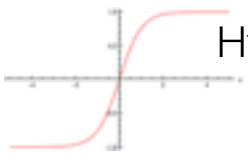
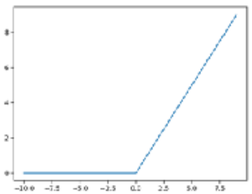
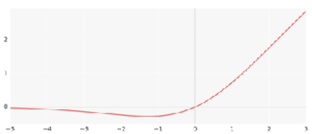
We have solved this assuming the activation output is in the range -4 to 4 (for a sigmoid) and assuming that we use Gaussian for sampling.

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	$\phi(z) = \dfrac{z}{1+e^{-z}}$	$\nabla\phi(z) = \phi(z) + \sigma(z)\cdot(1-\phi(z))$	

Demo

`08. Practical_NeuralNets.ipynb`

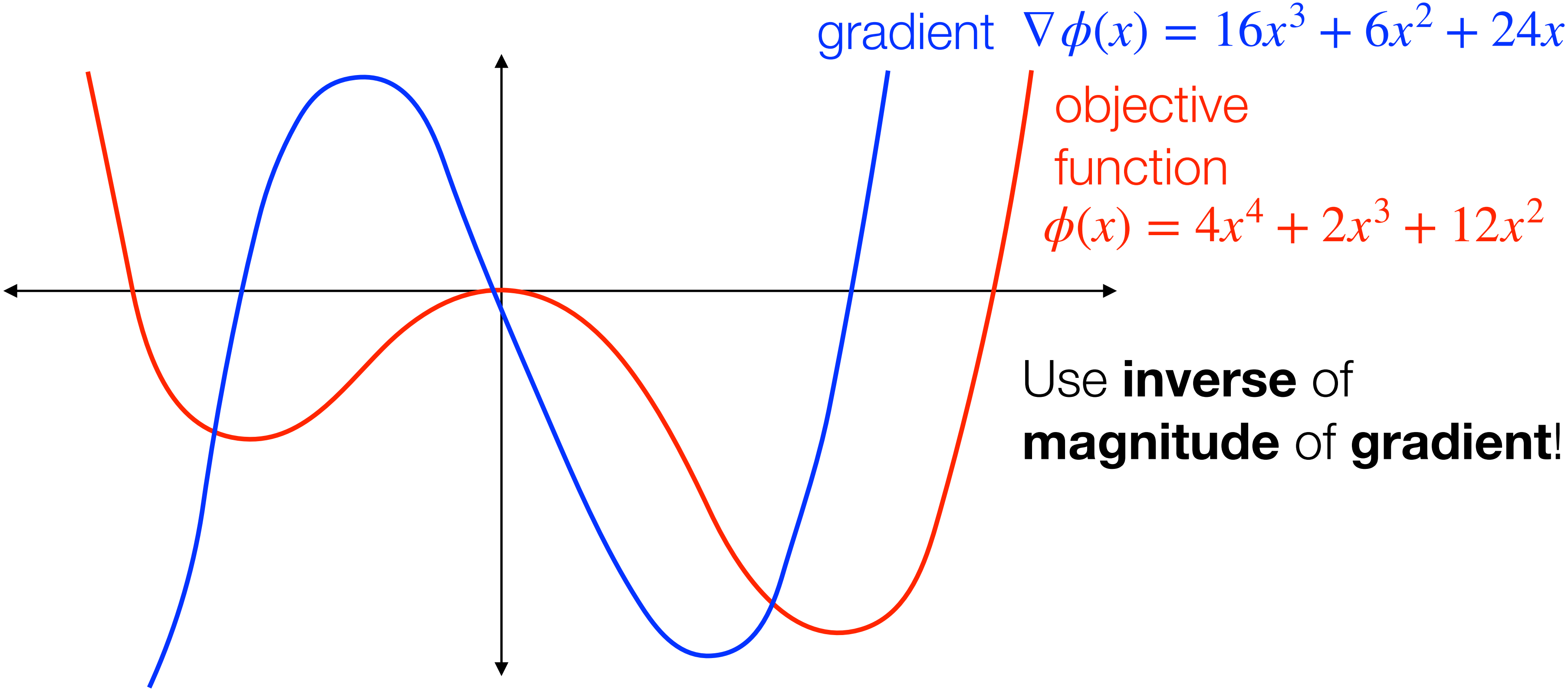
- ReLU Nonlinearities
- Important for deep networks

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**!

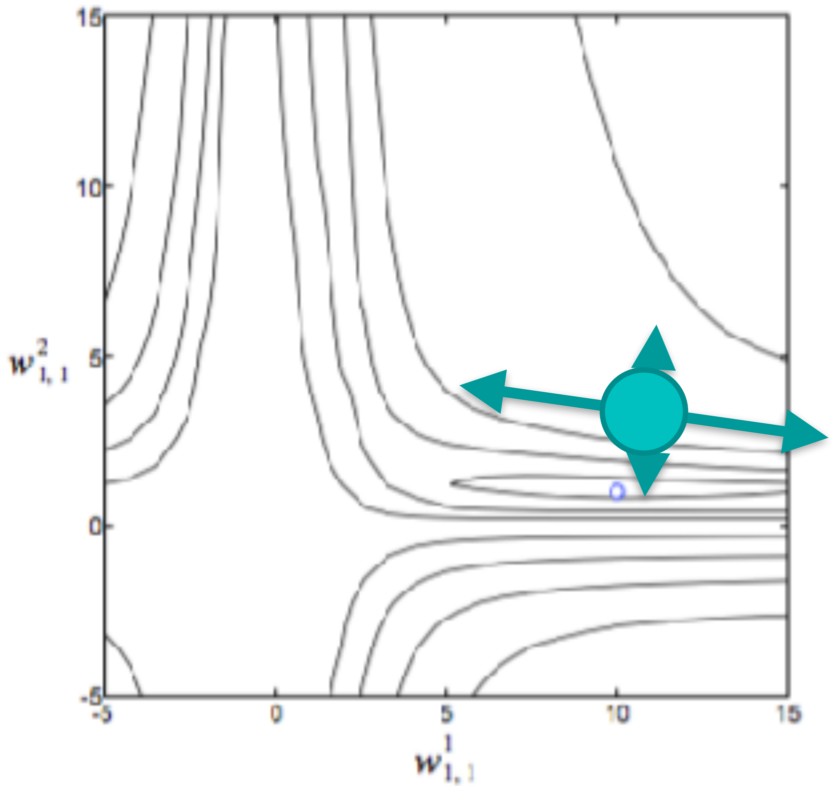
Momentum: be robust to **abrupt changes** in **steepness** (accumulate inverse magnitudes)

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

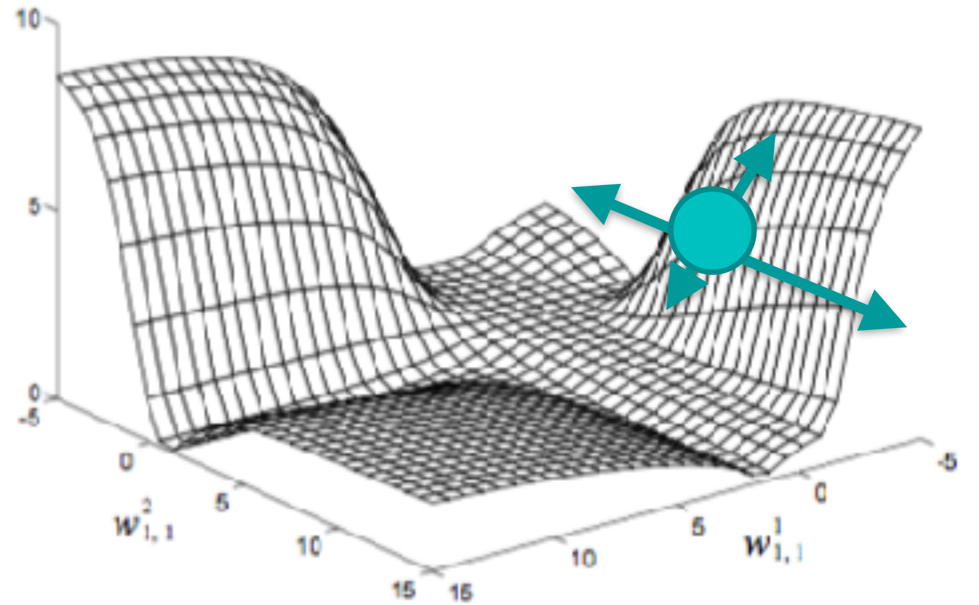
Be adaptive based on Gradient Magnitude?

Inverse magnitude of gradient in multiple directions?

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



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



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

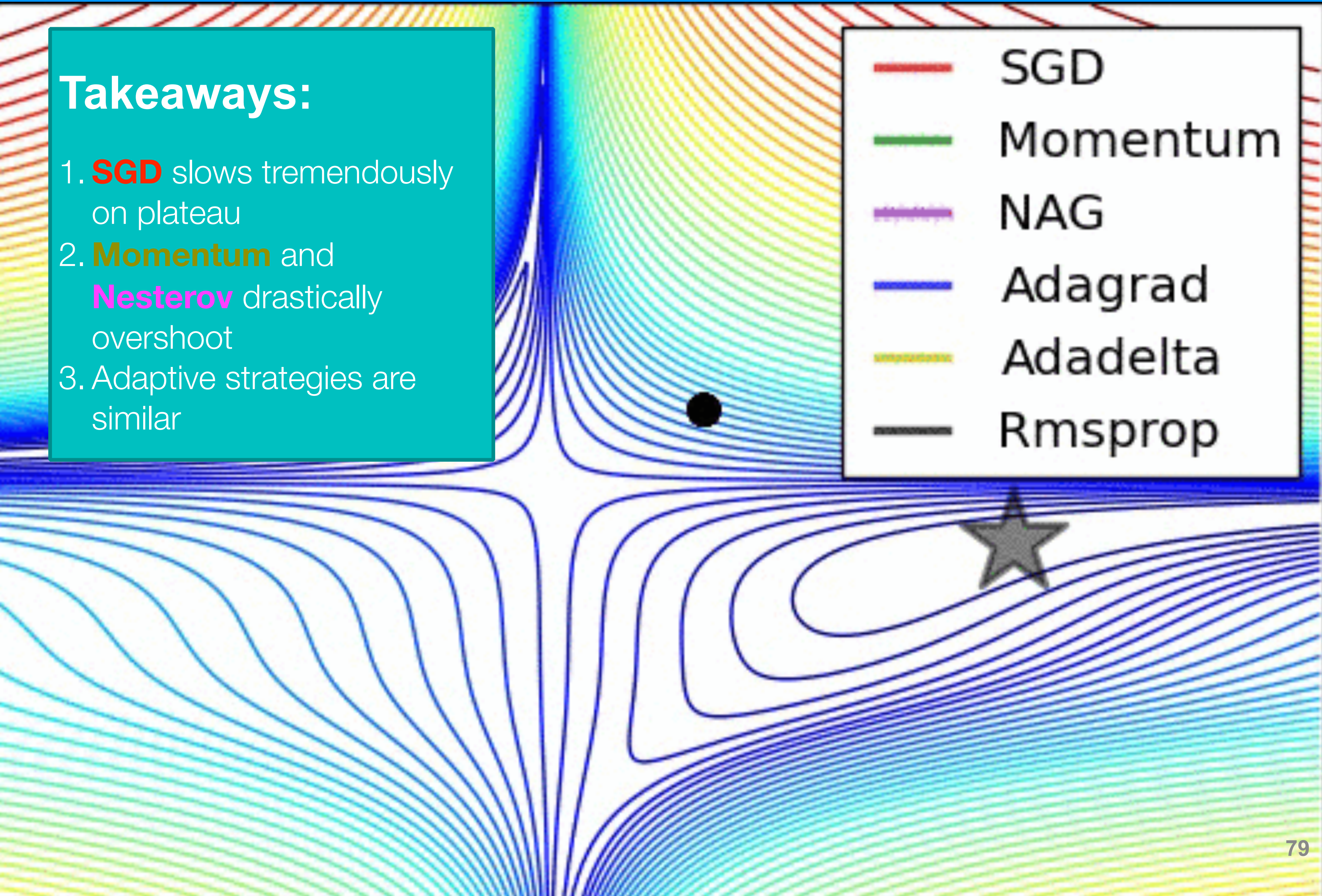
squared magnitude normalizer

Visualization of Optimization

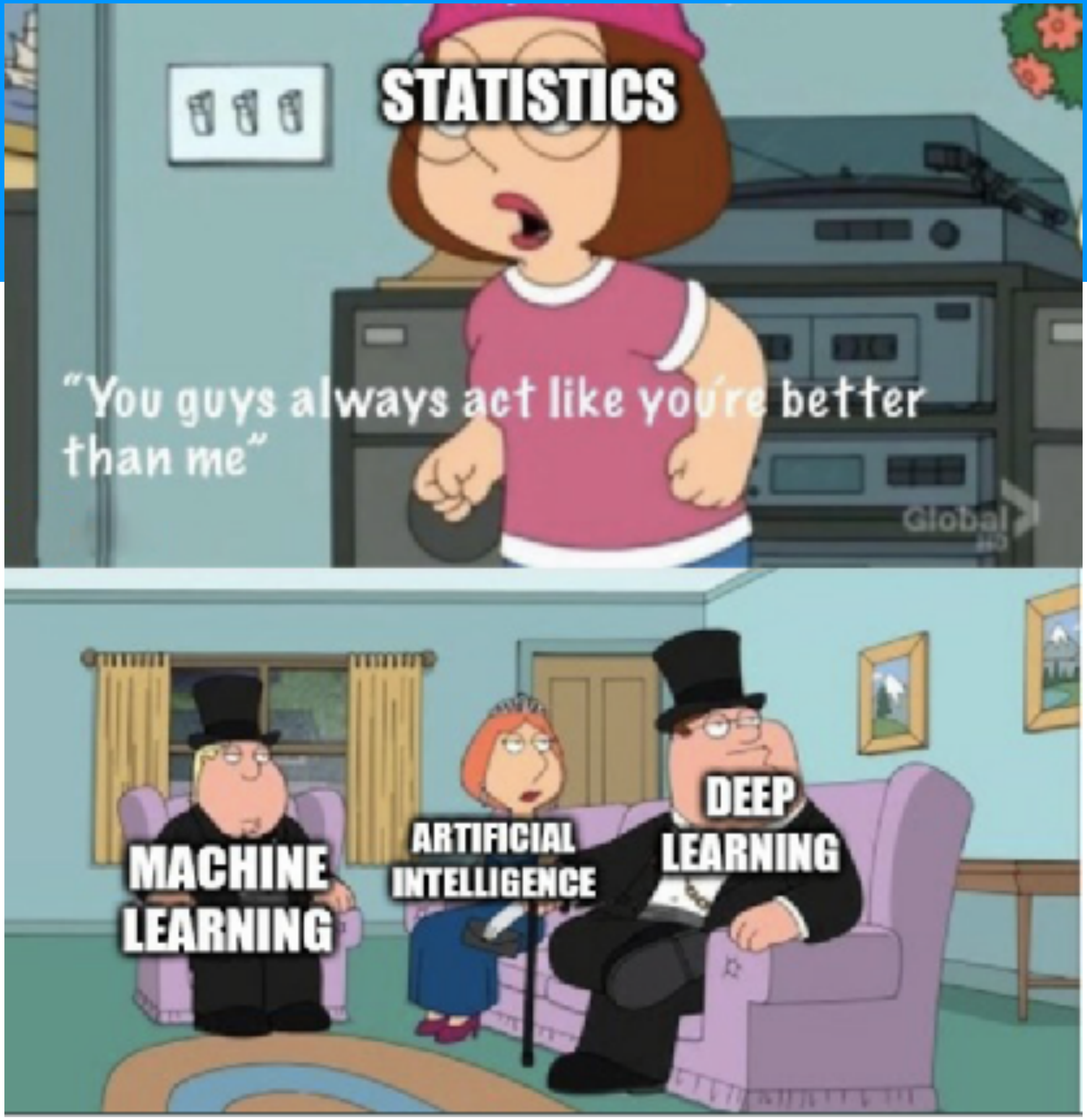
https://ruder.io/optimizing-gradient-descent/

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→

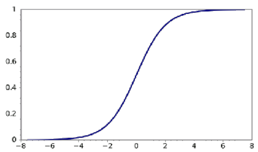
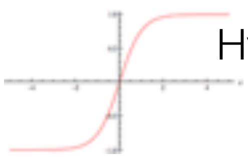
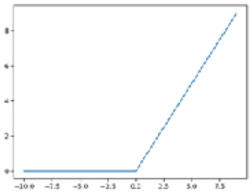
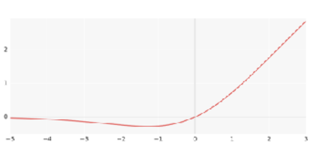
	batch 1	batch 2	batch 3	batch 4	batch 5	batch 6	batch 7	batch 8	batch 9
Epoch 1									
Epoch 2									
Epoch 3									
Epoch 4									
...									

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

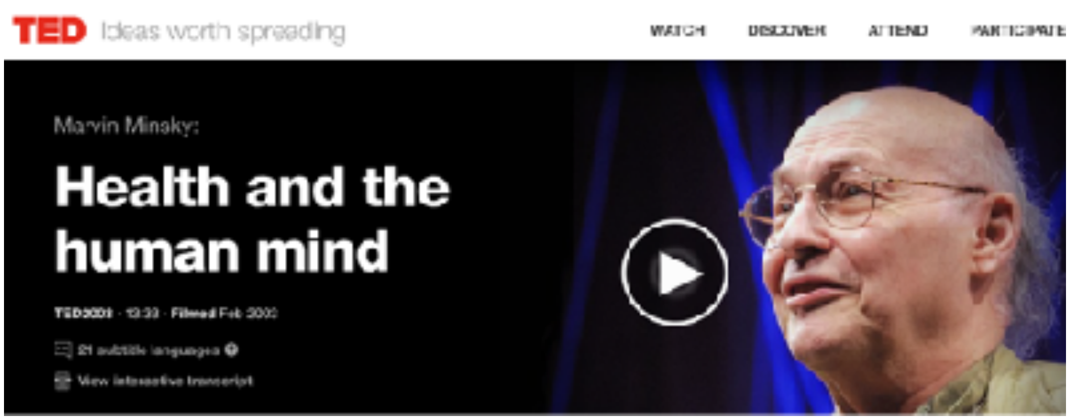
	Definition	Derivative	Weight Init *(Uniform Bounds)*
 Sigmoid	$\phi(z) = \frac{1}{1+e^{-z}}$	$\nabla\phi(z) = a(1-a)$	$w_{ij}^{(L)} \sim \pm 4\sqrt{\frac{6}{n^{(L)}+n^{(L+1)}}}$
 Hyperbolic Tangent	$\phi(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$\nabla\phi(z) = \frac{4}{(e^{z}+e^{-z})^2}$	$w_{ij}^{(L)}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

Revisiting Universality

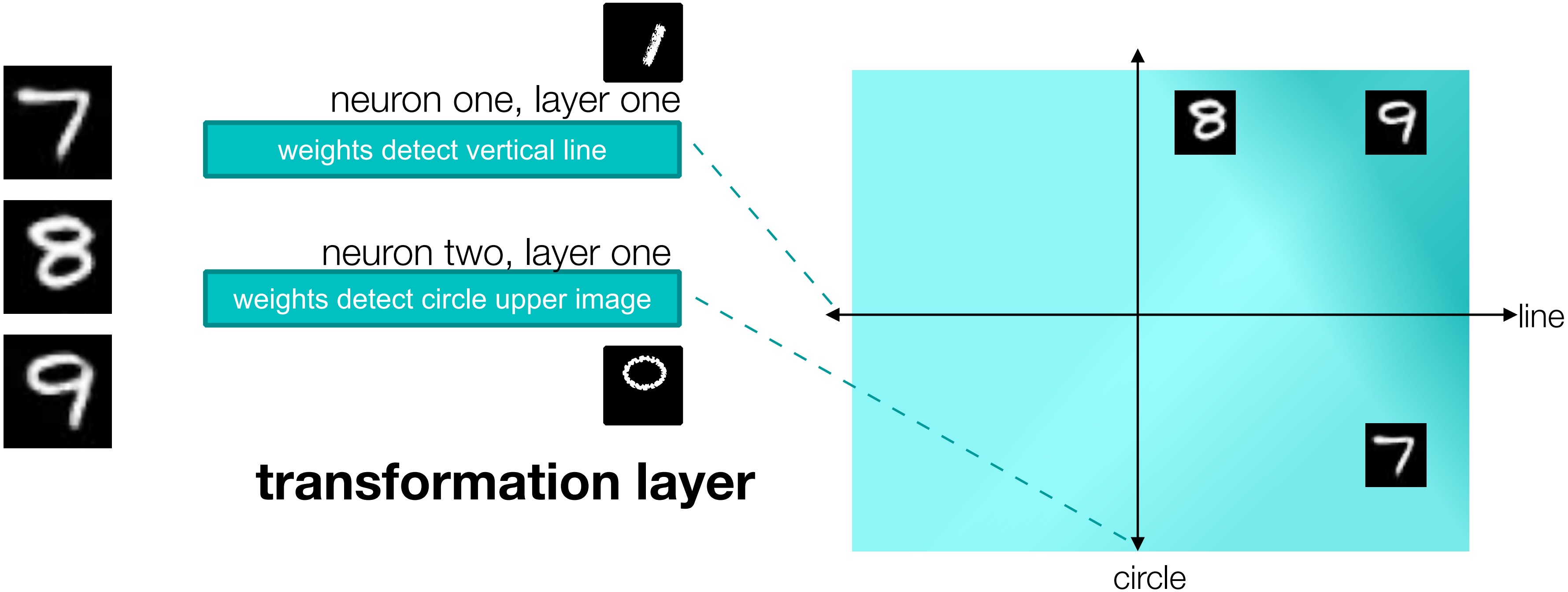
- Neural networks can separate any data through multiple layers. The true realization of Rosenblatt:

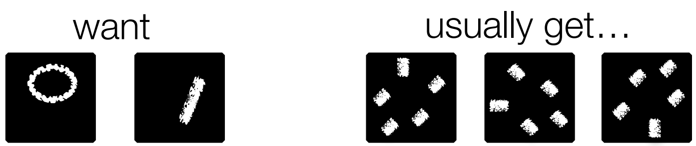
  "Given an elementary α-perceptron, a stimulus world W, and any classification C(W) for which a solution exists; let all stimuli in W occur in any sequence, provided that each stimulus must reoccur in finite time; then beginning from an arbitrary initial state, an error correction procedure will always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

End of Session

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

Town Hall