Neural Network Training: Implementation

Danna Gurari
University of Colorado Boulder
Spring 2024

https://dannagurari.colorado.edu/course/neural-networks-and-deep-learning-spring-2024/

Review

- Last lecture:
  - Objective function: what to learn
  - Gradient descent: how to learn
  - Training a neural network: optimization
  - Gradient descent for different activation functions
  - Lab assignment expectations
  - Programming tutorial

- Assignments (Canvas):
  - Problem set 1 grades out (send regrade requests by Feb 14 to our TA, Everley)
  - Lab assignment 1 due Wednesday

- Questions?

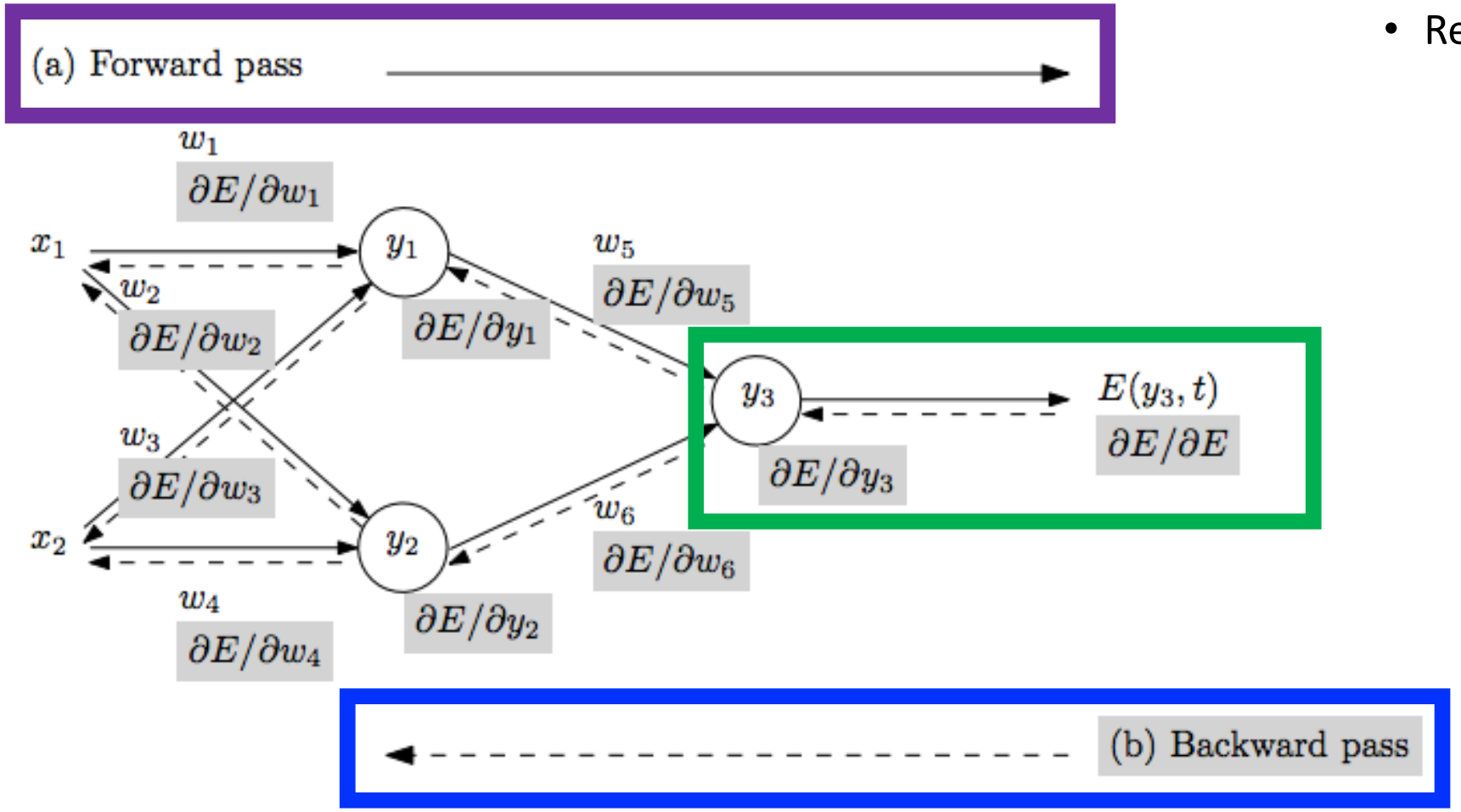
Today’s Topics

- Motivation
- Activation functions
- Initialization
- Efficient learning
- Analyzing Loss Curves
- Programming tutorial

Today's Topics

- Motivation
- Activation functions
- Initialization
- Efficient learning
- Analyzing Loss Curves
- Programming tutorial

Recall: We Optimize NNs with Gradient Descent

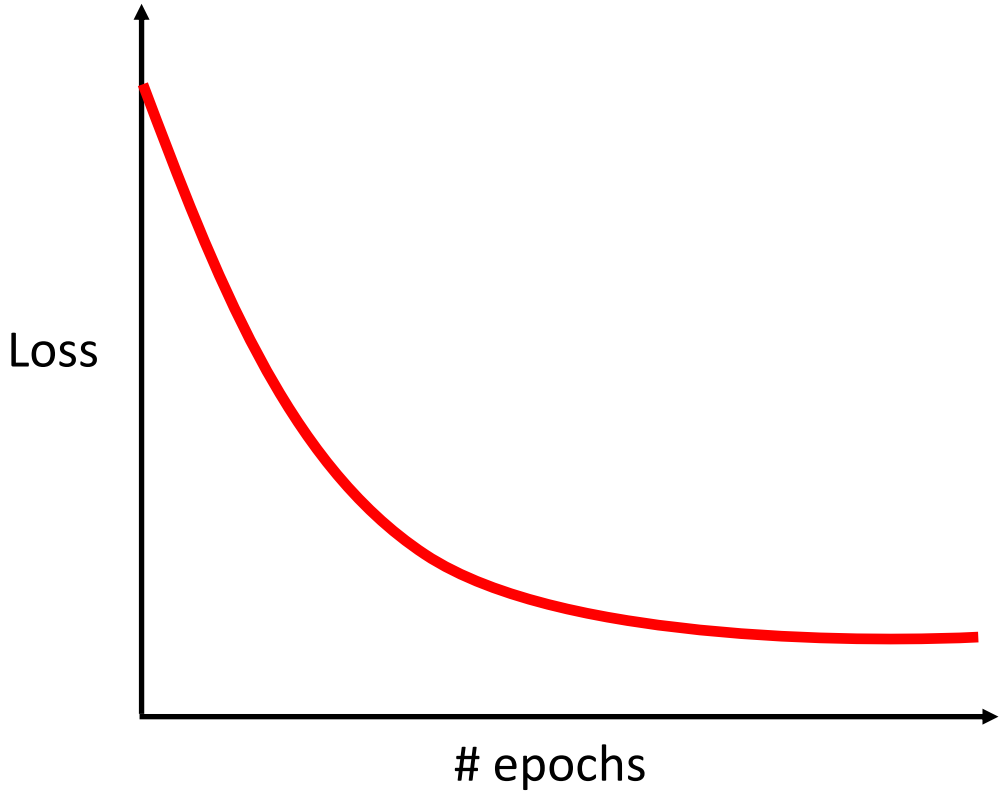


- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through model to make predictions
  2. **Error quantification**: measure dissatisfaction with a model's predictions on training data
  3. **Backward pass**: using predicted output, calculate gradients backward to assign blame to each model parameter
  4. Update each parameter using calculated gradients

Figure from: Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind; Automatic Differentiation in Machine Learning: a Survey; 2018

NN Optimization Goal

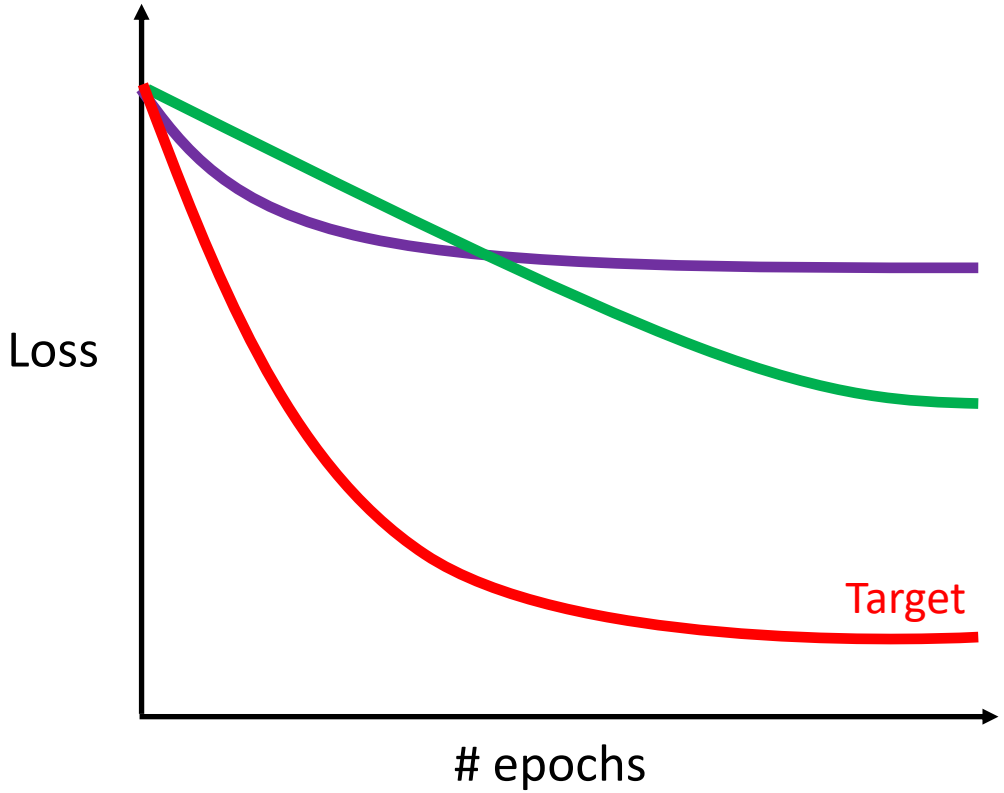
What do we want to see from the error/loss during training?



- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through model to make predictions
  2. **Error quantification**: measure dissatisfaction with a model's predictions on training data
  3. **Backward pass**: using predicted output, calculate gradients backward to assign blame to each model parameter
  4. Update each parameter using calculated gradients

NN Optimization Status Quo

What do we often see from the error/loss during training?



- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through model to make predictions
  2. **Error quantification**: measure dissatisfaction with a model's predictions on training data
  3. **Backward pass**: using predicted output, calculate gradients backward to assign blame to each model parameter
  4. Update each parameter using calculated gradients

Learning Challenge: Having "Fuel" (Gradients)

Recall: an **algorithm** learns from **data** on a **processor** the patterns that will be used to make a prediction

Analogous to a love story of partnering up and road tripping somewhere

- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through model to make predictions
  2. **Error quantification**: measure dissatisfaction with a model's predictions on training data
  3. **Backward pass**: using predicted output, calculate gradients backward to assign blame to each model parameter
  4. Update each parameter using calculated gradients

$w = w - lr * dw$

Today's Scope: "Looking Under the Hood" on What Impacts Gradients During Training

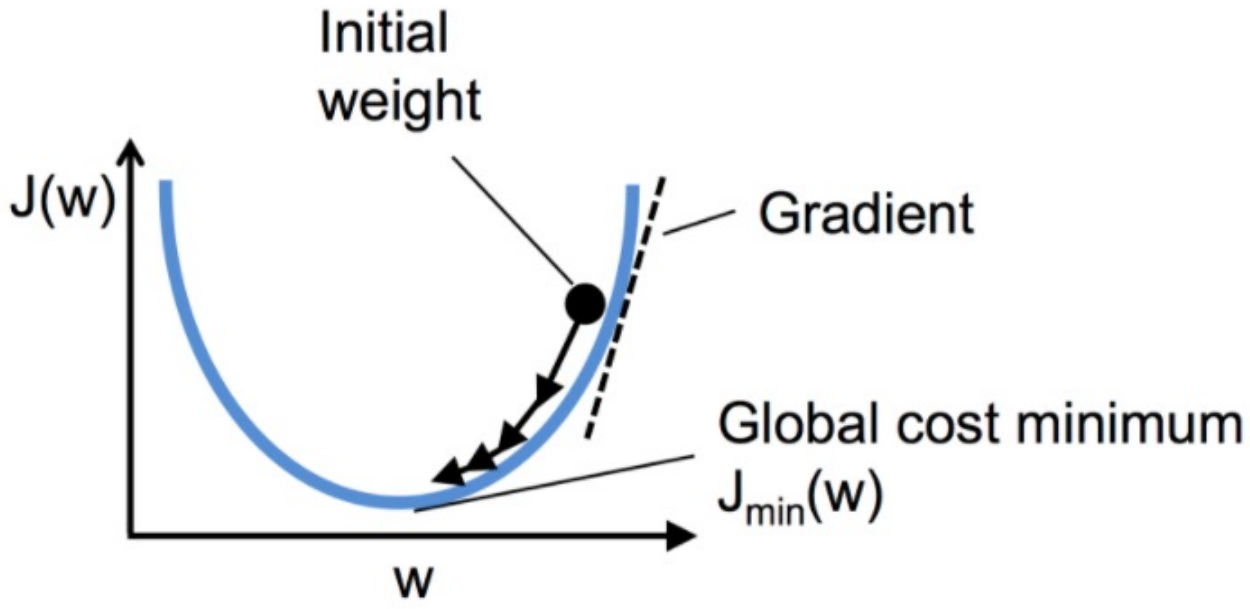
https://www.etftrends.com/etfs-the-importance-of-looking-under-the-hood/

Today's Topics

- Motivation
- Activation functions
- Initialization
- Efficient learning
- Analyzing Loss Curves
- Programming tutorial

Activation Function Overview

- **Want**: function with a gradient large enough to support efficient learning

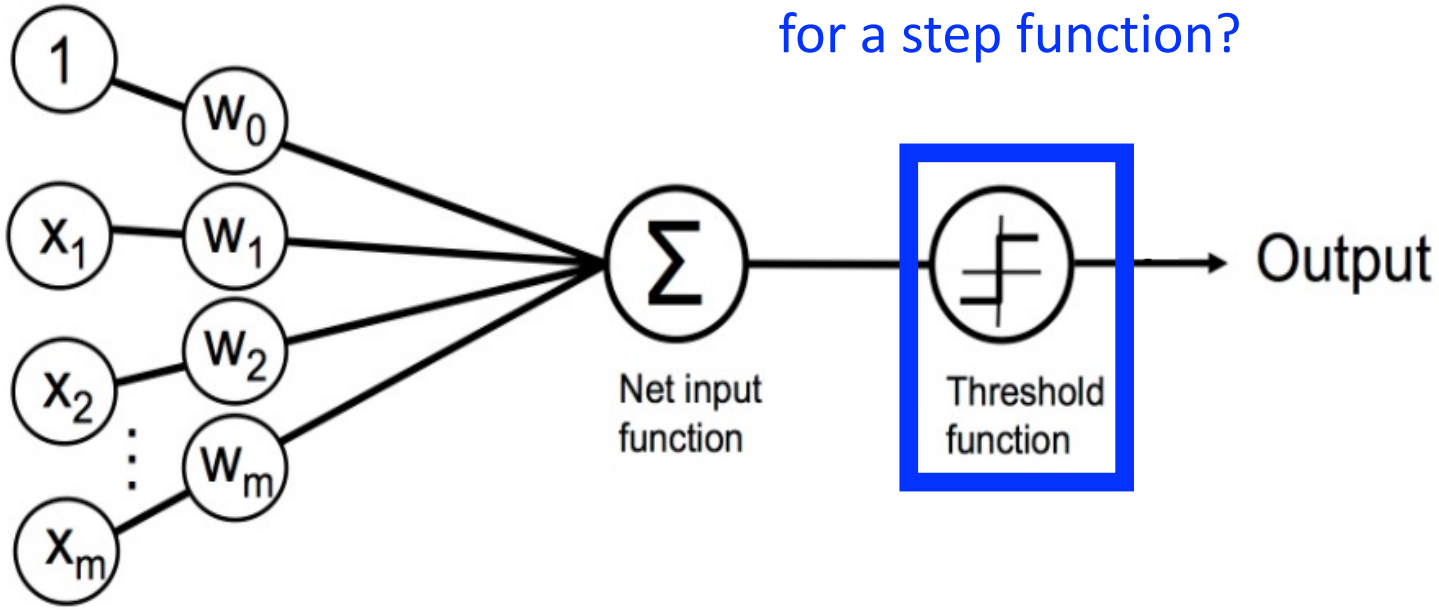


- **Implied requirement:** function should be differentiable

https://github.com/rasbt/python-machine-learning-book-2nd-edition/blob/master/code/ch02/ch02.ipynb

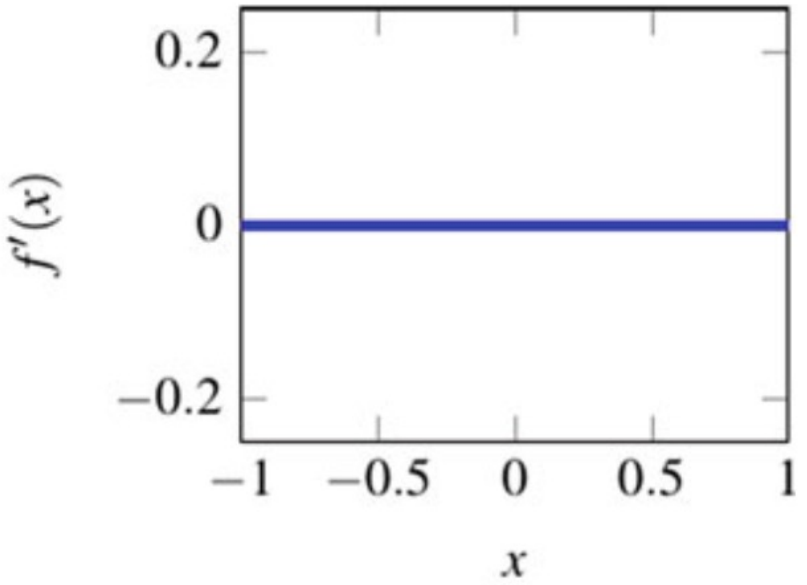
Activation Functions with Gradients: Revisiting Perceptron

What is the gradient for a step function?



Python Machine Learning; Raschka & Mirjalili

0 everywhere except 0 where it is non-differentiable

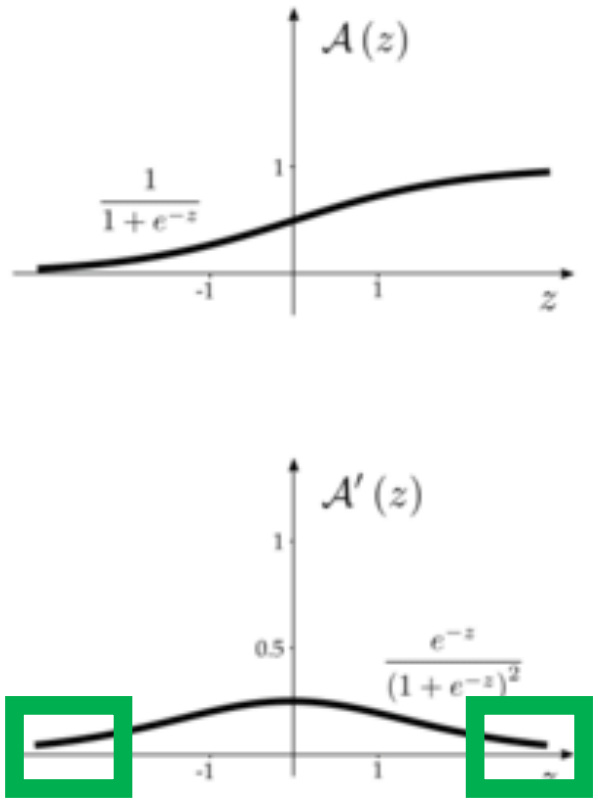


Deep Learning for NLP and Speech Recognition; Kamath, Liu, & Whitaker

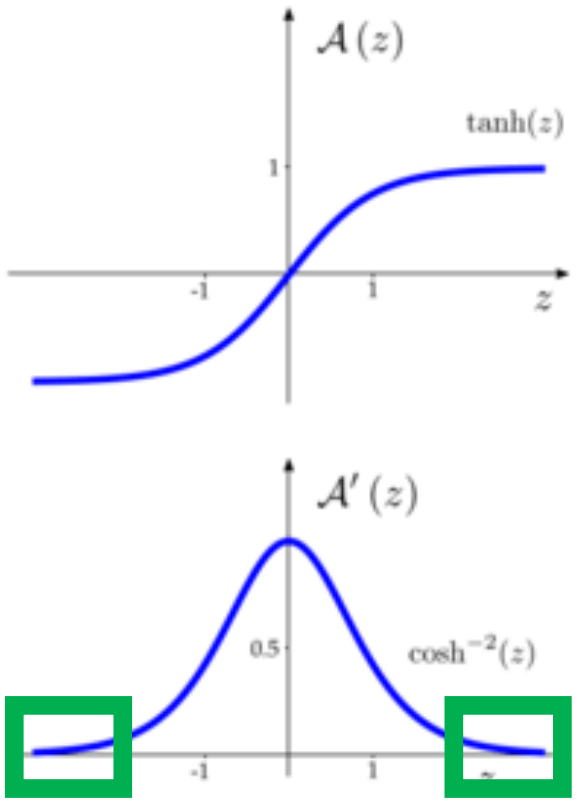
No gradient means model parameters wouldn't change with gradient descent!

Activation Functions with Gradients: Nonlinear Activation Functions

Sigmoid



Tanh



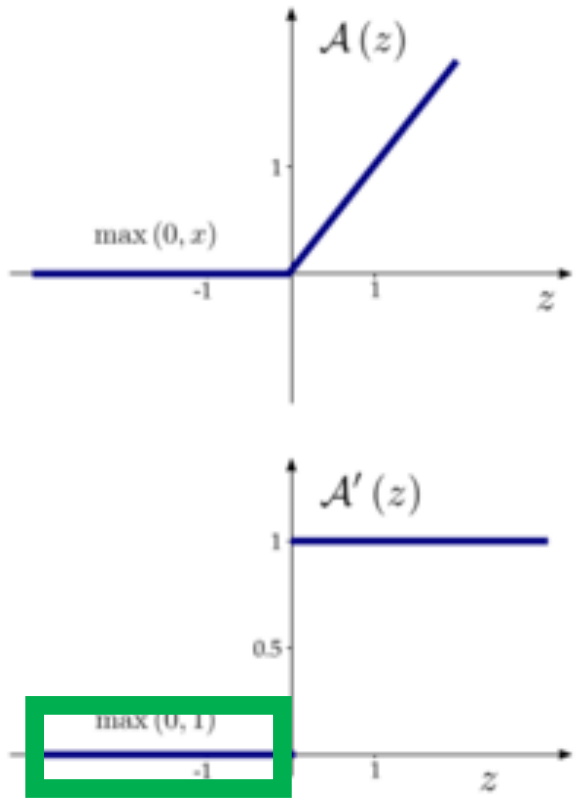
Problem: units with small or large “z” values lead to slow/no learning; why?

Reason: small gradients limit amount model parameters change with gradient descent

Masi et al. Journal of the Mechanics and Physics of Solids. 2021

Activation Functions with Gradients: Nonlinear Activation Functions

ReLU



Advantages:

- Fast to compute
- Large gradient when unit is “firing”

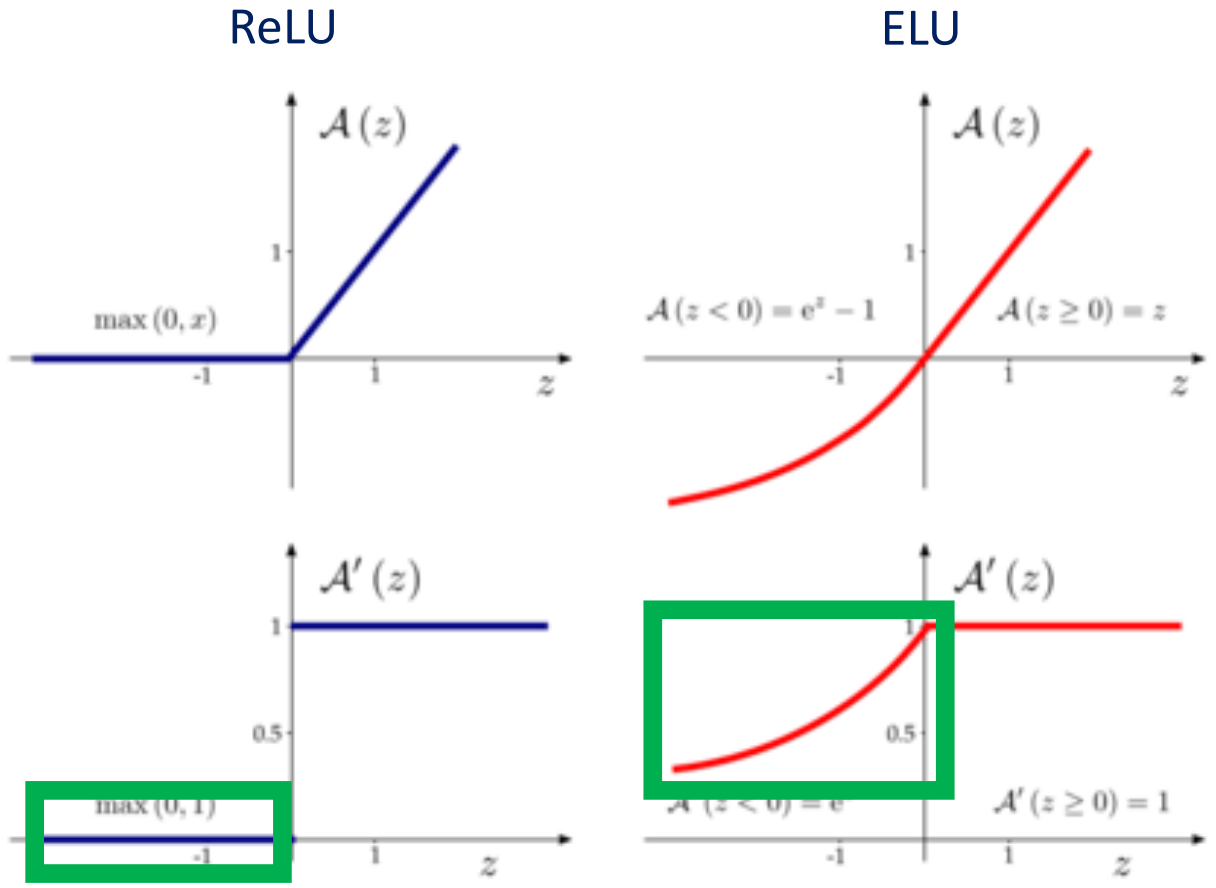
Implementation detail:

- When function is not differentiable at z=0, hard code value (e.g., 0)

Problem: no gradient means units can “die” (analogy: brain damage)

Masi et al. Journal of the Mechanics and Physics of Solids. 2021

Activation Functions with Gradients: Nonlinear Activation Functions

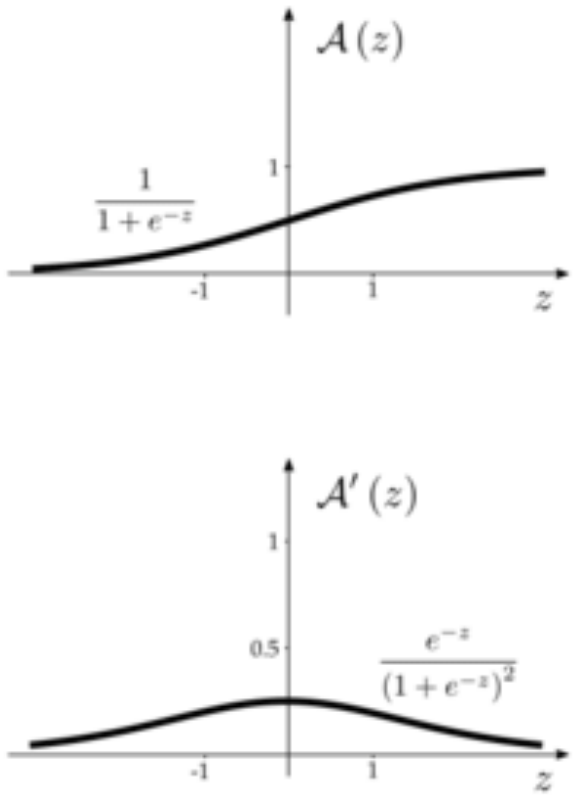


Can avoid dying units by increasing computational complexity of ELU-based activation functions

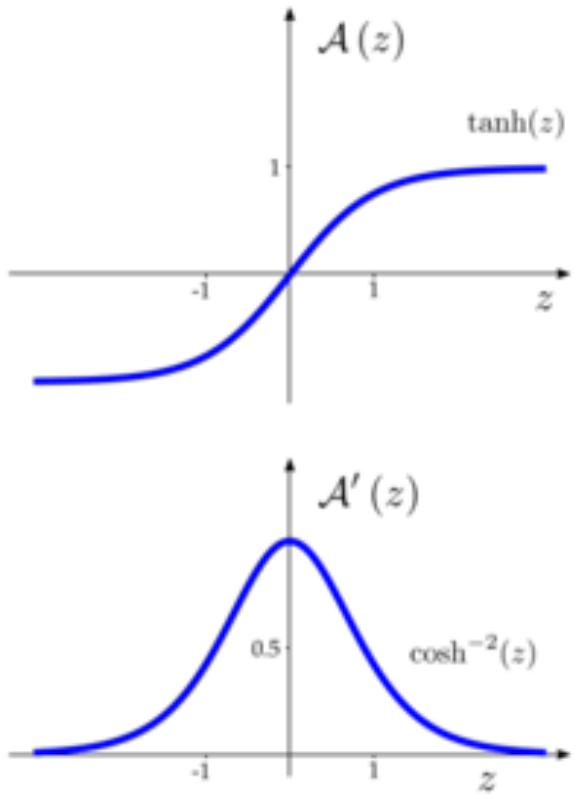
Masi et al. Journal of the Mechanics and Physics of Solids. 2021

Activation Functions with Gradients: Nonlinear Activation Functions

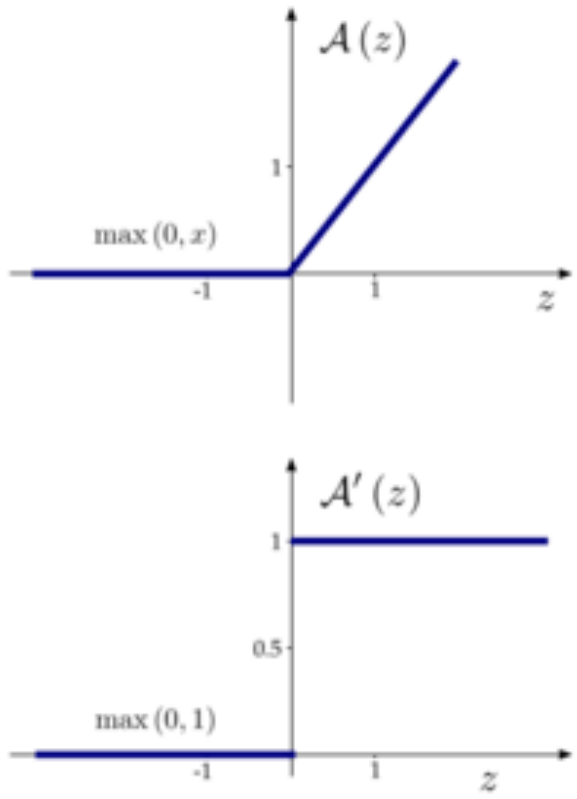
Sigmoid



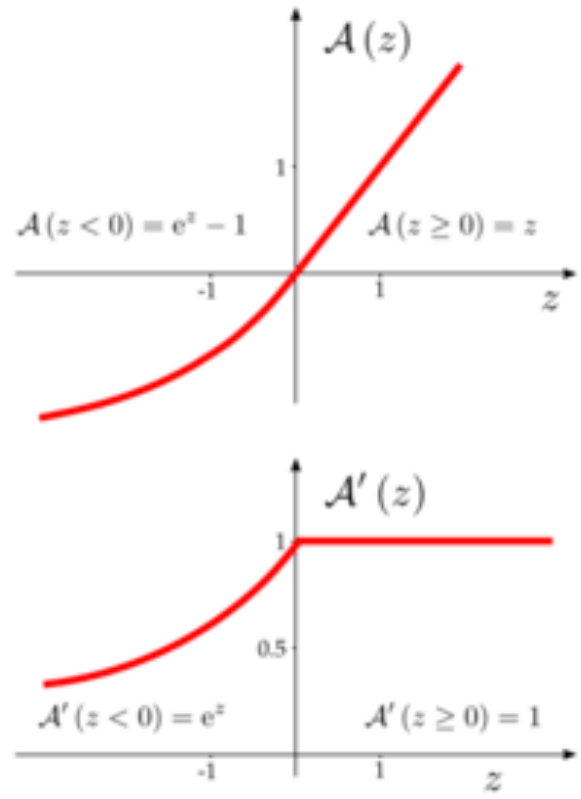
Tanh



ReLU



ELU

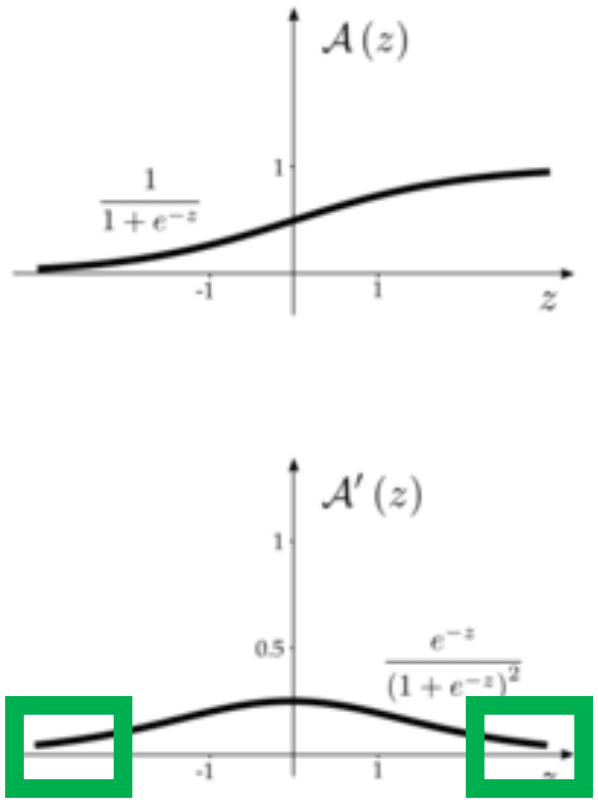


Training goal: large gradients that enable efficient learning

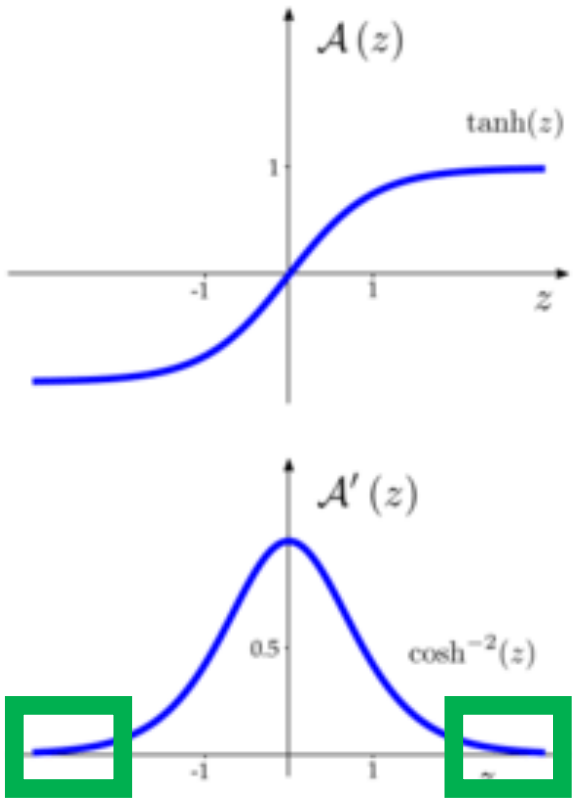
Masi et al. Journal of the Mechanics and Physics of Solids. 2021

Activation Functions with Gradients: Nonlinear Activation Functions

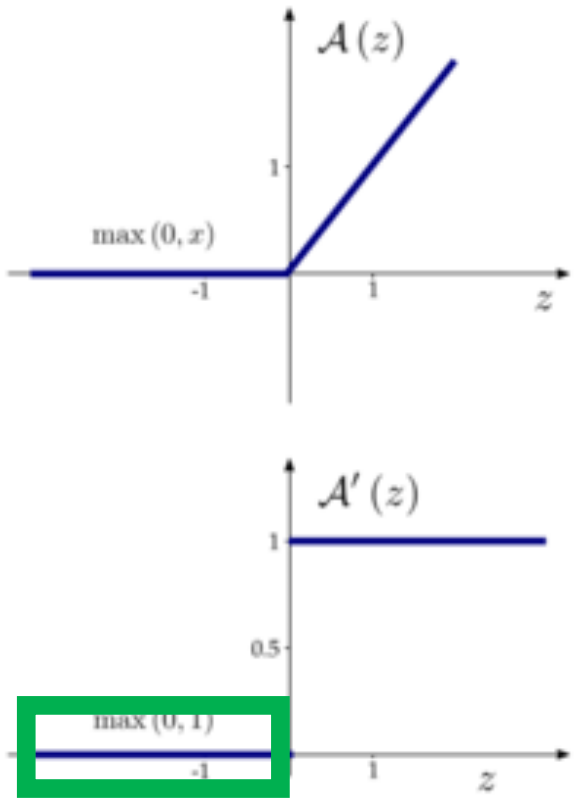
Sigmoid



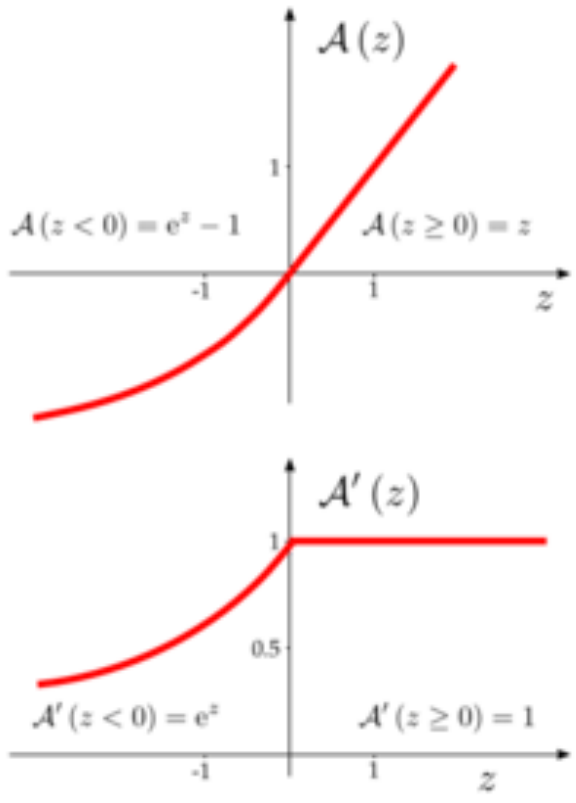
Tanh



ReLU



ELU



Training challenge: preventing *many* neurons from having little/no gradient and so little learning

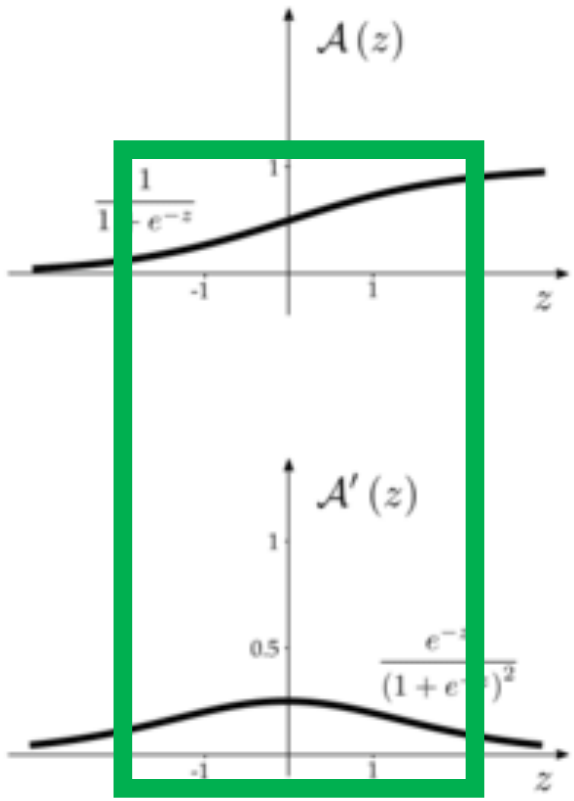
Masi et al. Journal of the Mechanics and Physics of Solids. 2021

Today's Topics

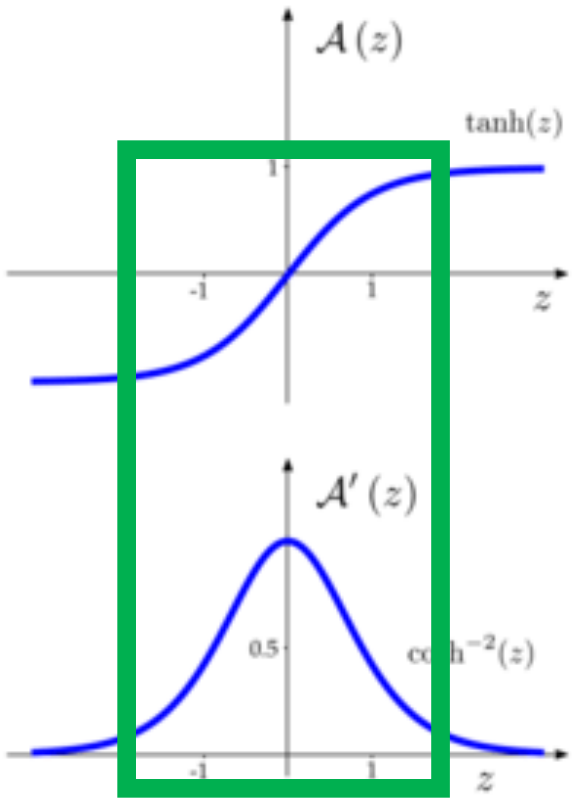
- Motivation
- Activation functions
- Initialization
- Efficient learning
- Analyzing Loss Curves
- Programming tutorial

Model Initialization: Target “Pre-Activation” Values? – for Sigmoid? Tanh? ReLu? ELU?

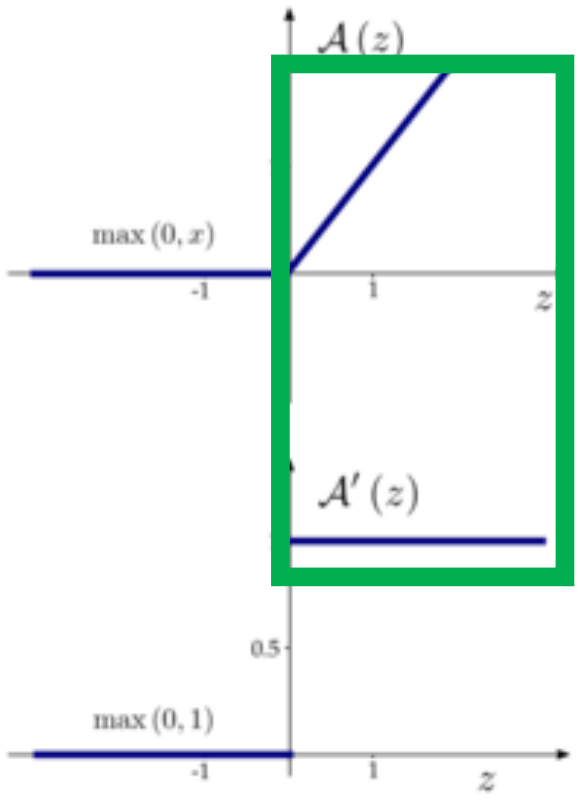
Sigmoid



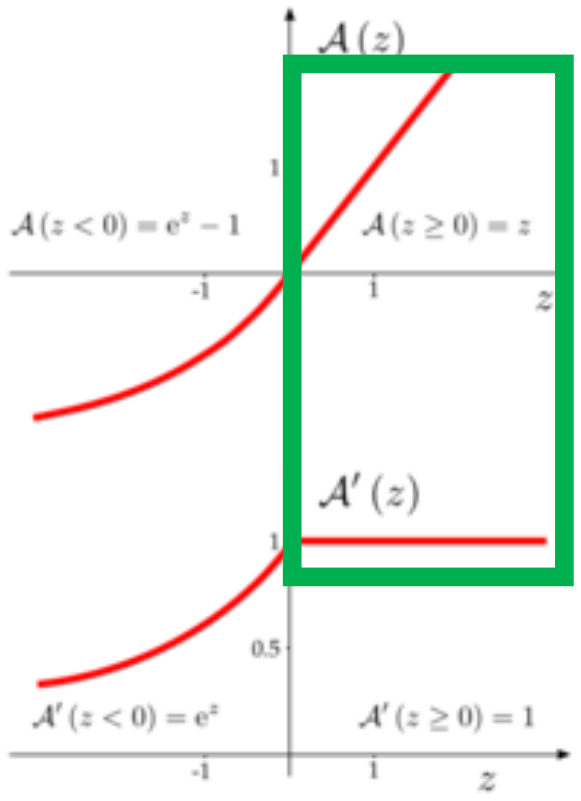
Tanh



ReLU



ELU



Masi et al. Journal of the Mechanics and Physics of Solids. 2021

Model Initialization: Set Weights So Weighted Sum Supports Learning

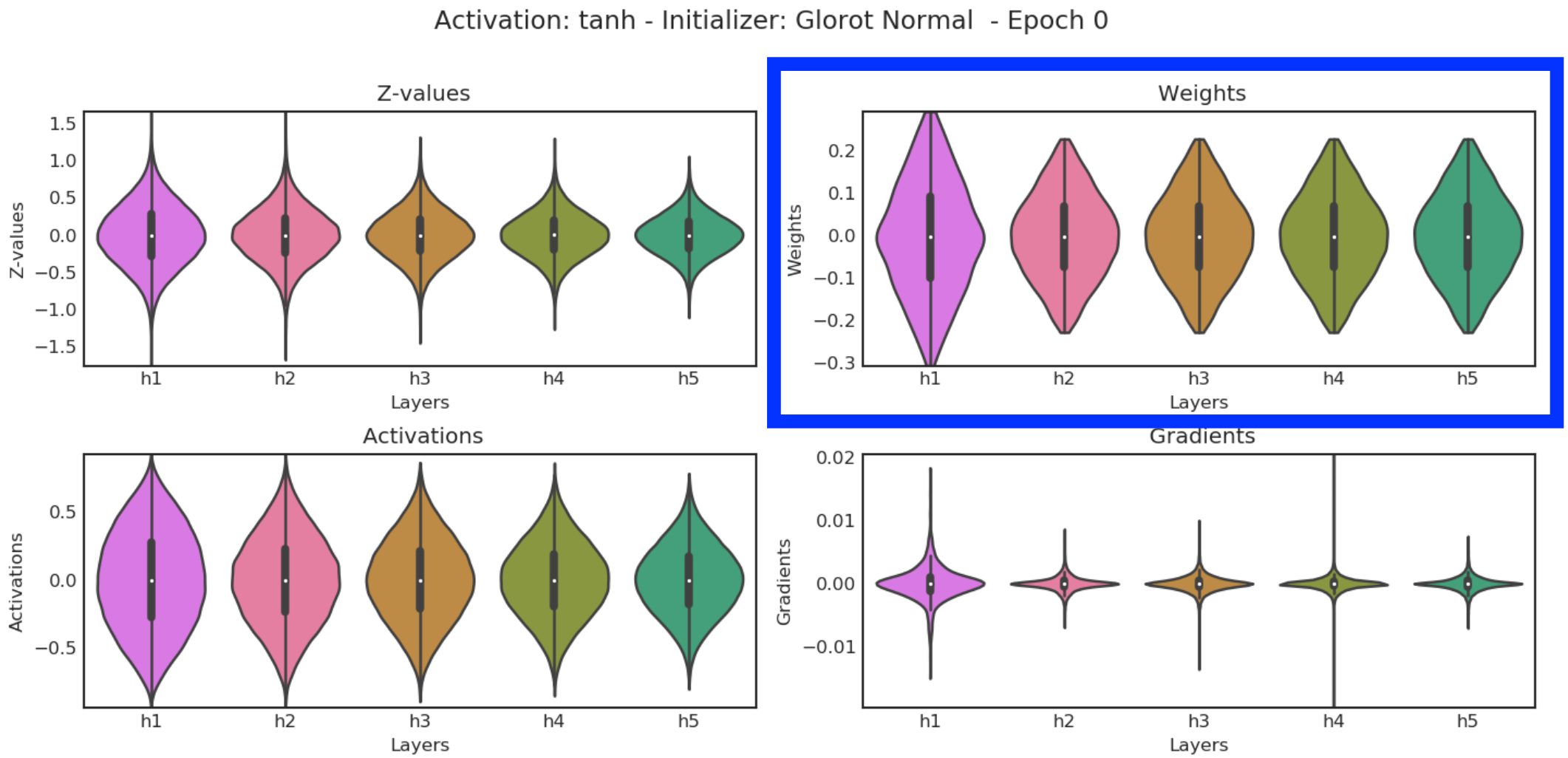
Key considerations:

- avoid symmetry, meaning different neurons compute same functions; why?
- avoid setting all weights to 0; why?
  - excellent tutorial: https://www.youtube.com/watch?v=eoNVmZDnn9w
- avoid large weights; why?

Approach:

- weights initialized to random, small values where the scale of “small” is often driven by the number of input nodes to the layer
- biases set to 0

e.g., Xavier/Glorot Initialization



Weights set so **z-values** have similar variance across layers in range of (1, -1), **activations** are similar across layers without going to extreme large/small values, and consequently **gradients** have similar variance across layers that support learning

https://towardsdatascience.com/hyper-parameters-in-action-part-ii-weight-initializers-35aee1a28404

e.g., Xavier/Glorot Initialization

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

(W is the weight matrix between layers *j* and *j+1*, n_j *is* # of neurons entering layer *j+1 or "fan in"*, n_{j+1} is # of neurons leaving layer *j+1 or "fan out"*, and U[−a, a] is the uniform distribution in the interval (−a, a))

(JMLR Workshop and Conference Proceedings, 2010)

Understanding the difficulty of training deep feedforward neural networks

---

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

e.g., Xavier/Glorot Initialization

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

(W is the weight matrix between layers *j* and *j+1*, n_j *is* # of neurons entering layer *j+1 or "fan in"*, n_{j+1} is # of neurons leaving layer *j+1 or "fan out"*, and U[−a, a] is the uniform distribution in the interval (−a, a))

Limitation: does not account for non-linearities introduced by activation functions

e.g., He/Kaiming/MSRA Initialization

$$\sigma = \sqrt{2.0/n_{in}}$$

(n_{in} *is* # of neurons entering the layer *or "fan in"*)

(ICCV, 2015)

Delving Deep into Rectifiers:
Surpassing Human-Level Performance on ImageNet Classification

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun

e.g., He/Kaiming/MSRA Initialization

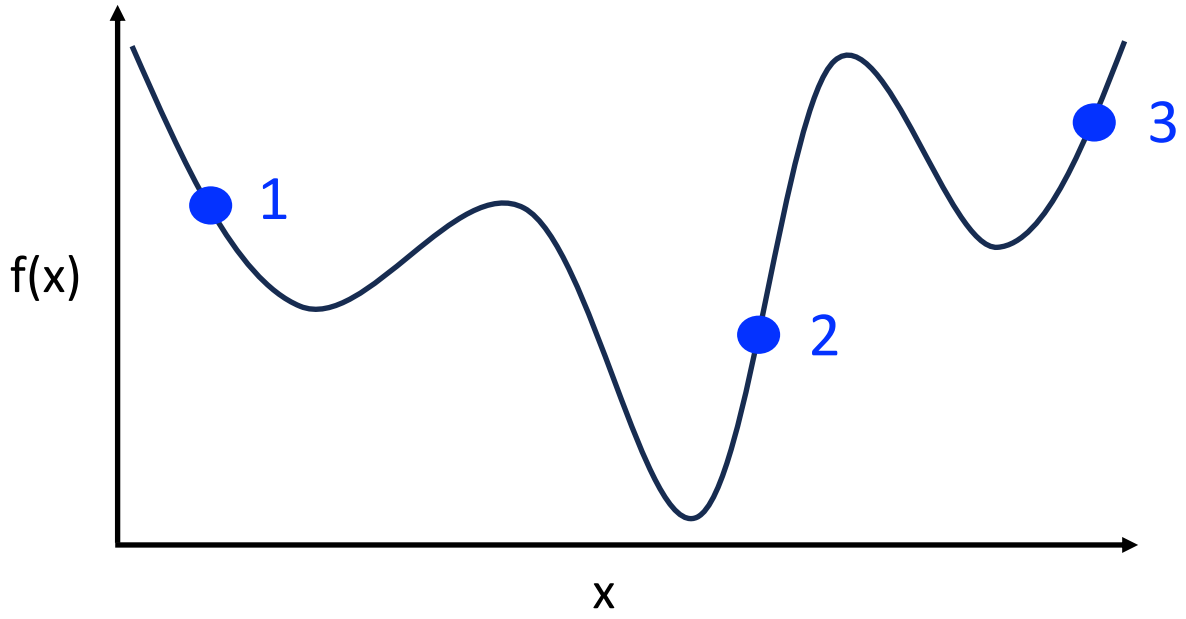
nonlinearity	gain
Linear / Identity	1
Conv{1,2,3}D	1
Sigmoid	1
Tanh	$\frac{5}{3}$
ReLU	$\sqrt{2}$

https://pytorch.org/docs/stable/nn.init.html

- Different scaling factors are used for different activation functions to account for how they squash the pre-activations differently
- Example below is for ReLU

$$\sigma = \sqrt{2.0/n_{in}}$$

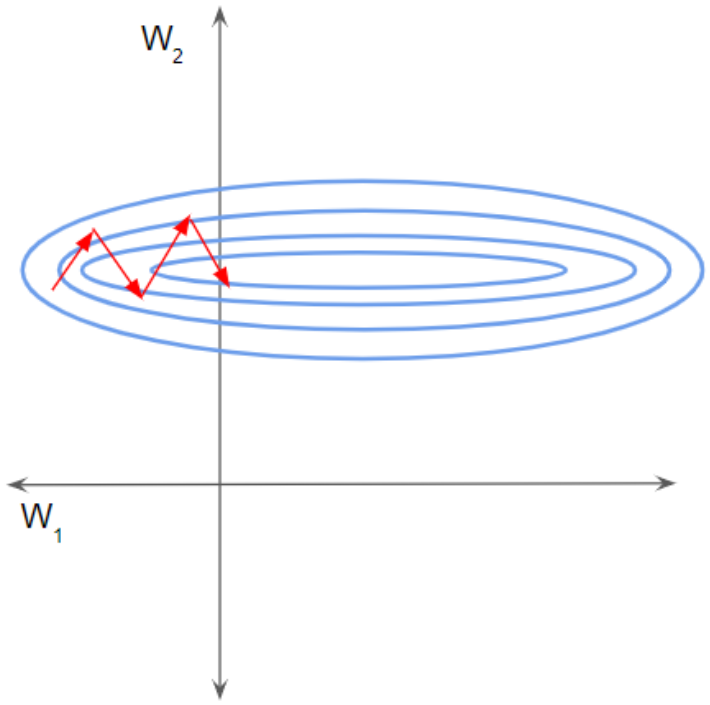
Practical Note: Where to Start When Learning?



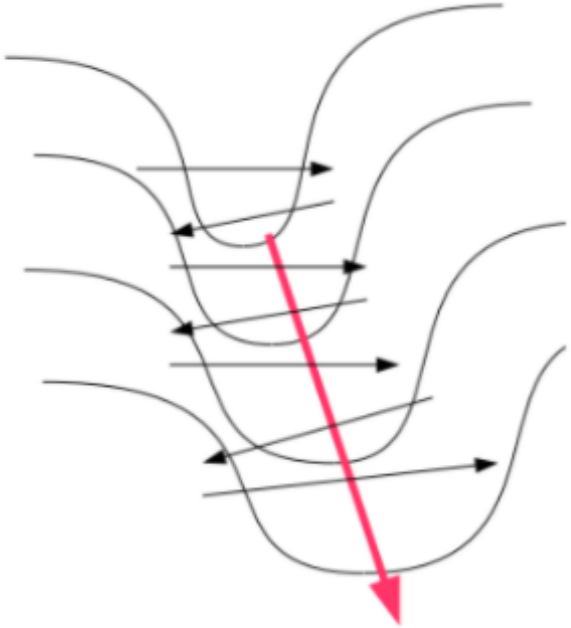
May need to repeat initialization to arrive as close as possible to the target solution to accelerate learning and improve final performance

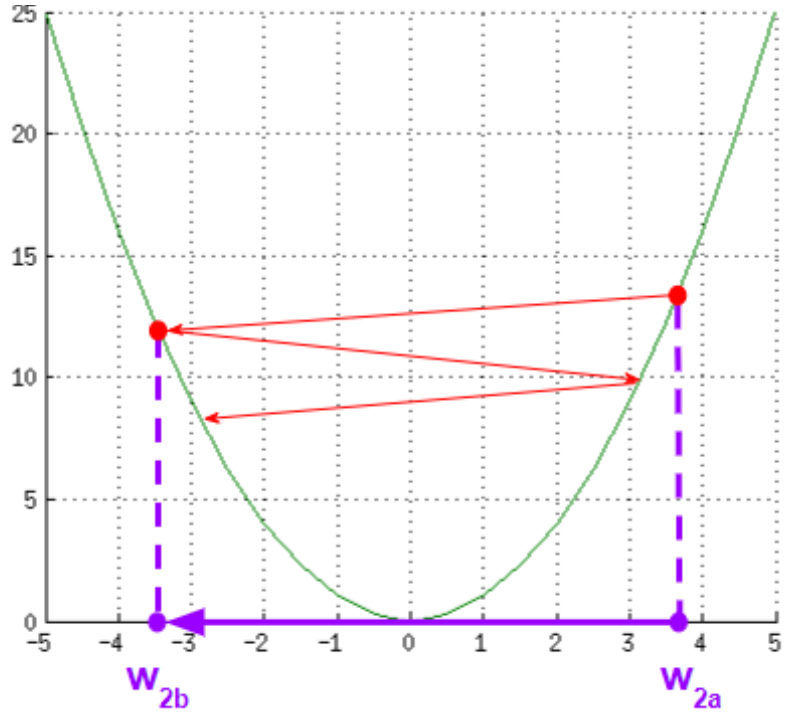
Data Initialization: Features On Different Scales Can Cause Learning To Be Slower and Poor Performance

e.g., 2D loss function:



Inefficient bouncing can occur during learning when larger updates are needed for some weights to minimize the loss during gradient descent



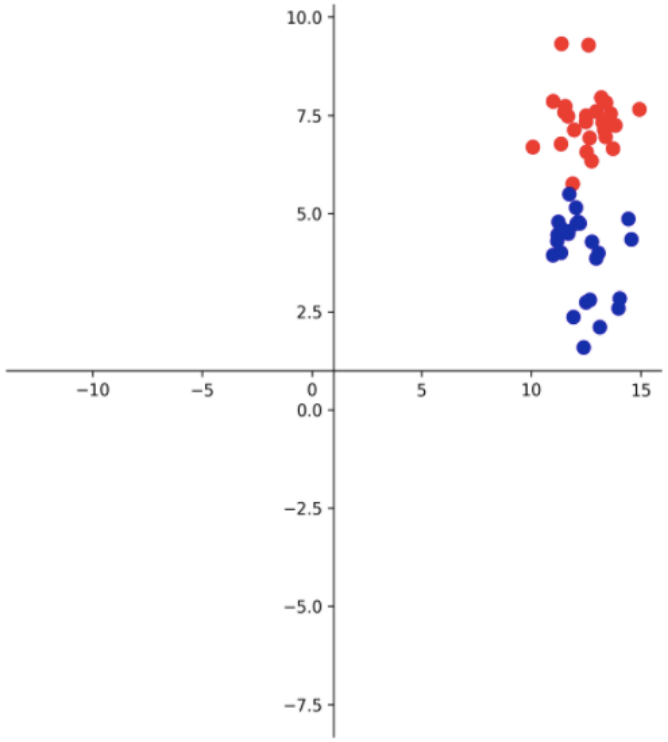


https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

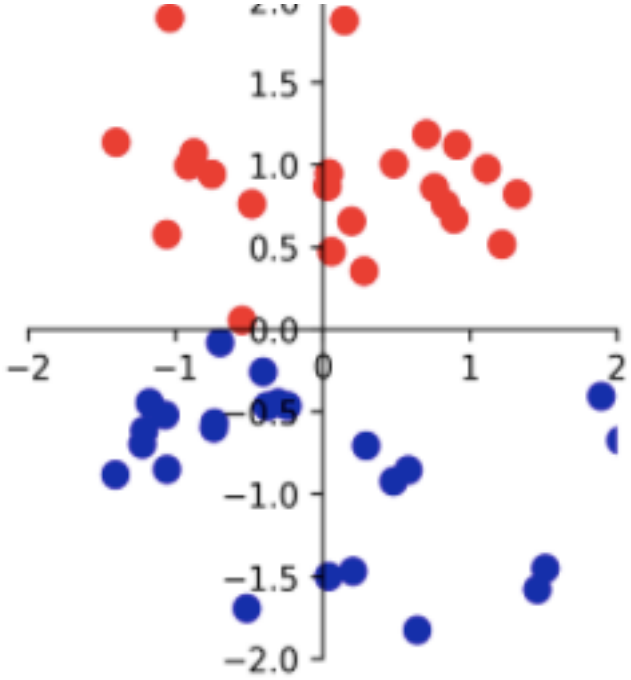
Data Initialization: Features On Different Scales Can Cause Learning To Be Slower and Poor Performance

* Simplify learning by standardizing input data so mean is 0 and standard deviation 1

Original data:



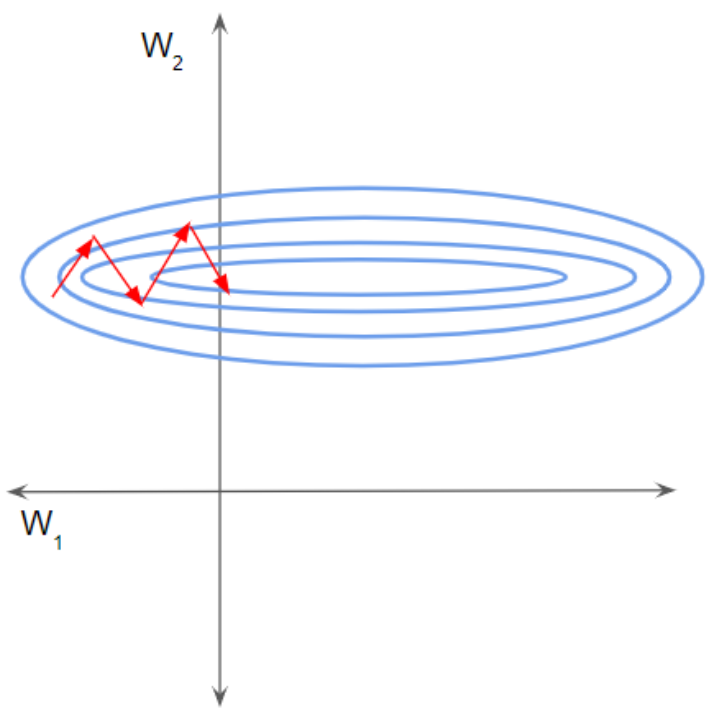
Standardized data:



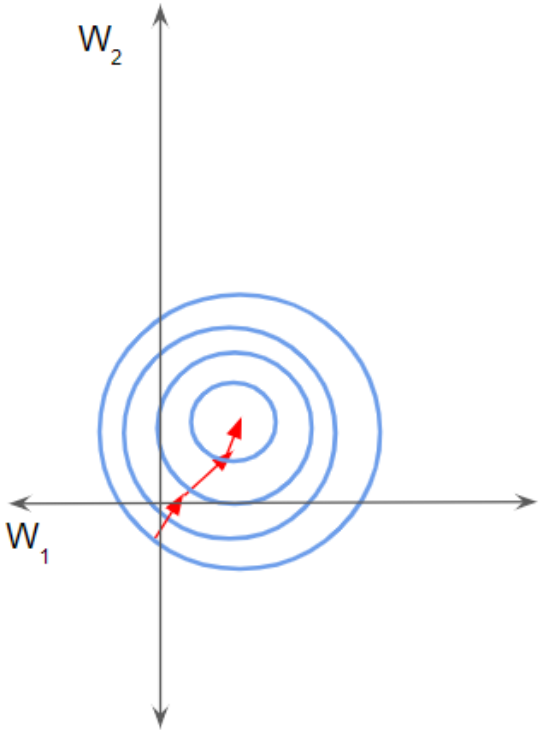
https://github.com/amueller/introduction_to_ml_with_python/blob/master/03-unsupervised-learning.ipynb

Data Initialization: Features On Different Scales Can Cause Learning To Be Slower and Poor Performance

* Simplify learning by standardizing input data so mean is 0 and standard deviation 1



Standardization changes the loss function so gradient descent can more smoothly arrive at the minimum!



https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

Today's Topics

- Motivation
- Activation functions
- Initialization
- Efficient learning
- Analyzing Loss Curves
- Programming tutorial

Challenge: Train Faster!!!

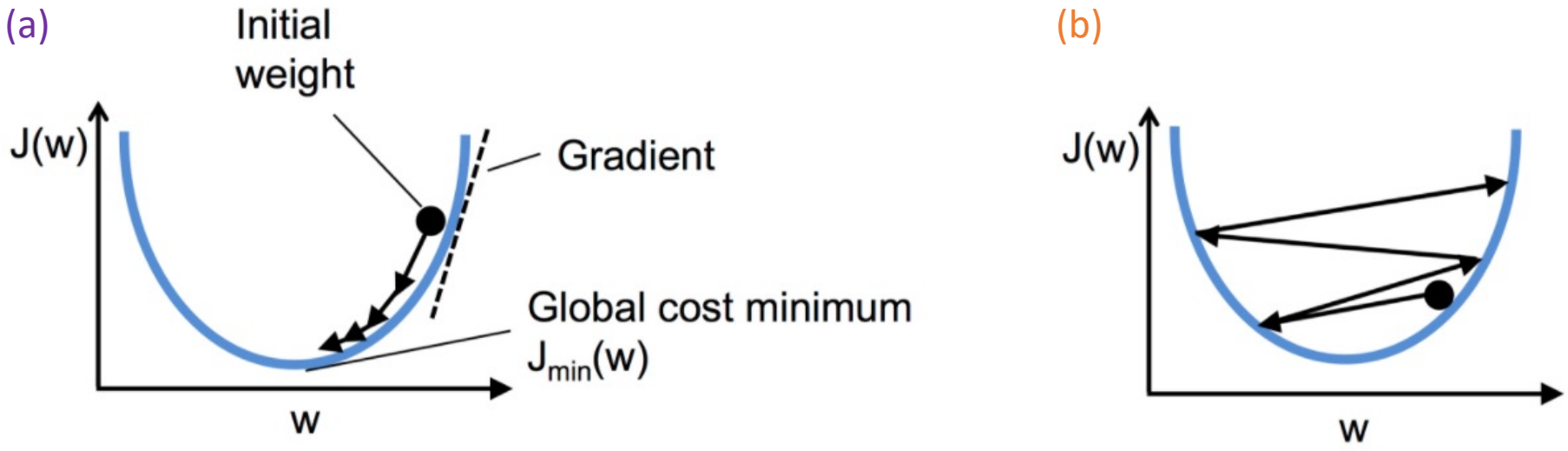
Algorithm training can take hours, days, weeks, months, or more with big data and so many parameters…

How Often to Update?

- Use calculations over ***all training examples*** (Batch gradient descent)
  - Less bouncing but can be slow or infeasible when dataset is large
- Use calculations from ***one training example*** (Stochastic gradient descent)
  - Fast to compute and can train using huge datasets (stores one instance in memory at each iteration) but updates are expected to bounce a lot
- Use calculations over ***subset of training examples*** (Mini-batch gradient descent)
  - Bounces less erratically than SGD and can train using huge datasets (store some instances in memory at each iteration) but can be slow or infeasible when dataset is large
- Often mini-batch gradient descent is used with maximum # of examples that fit in memory

How Much to Update?

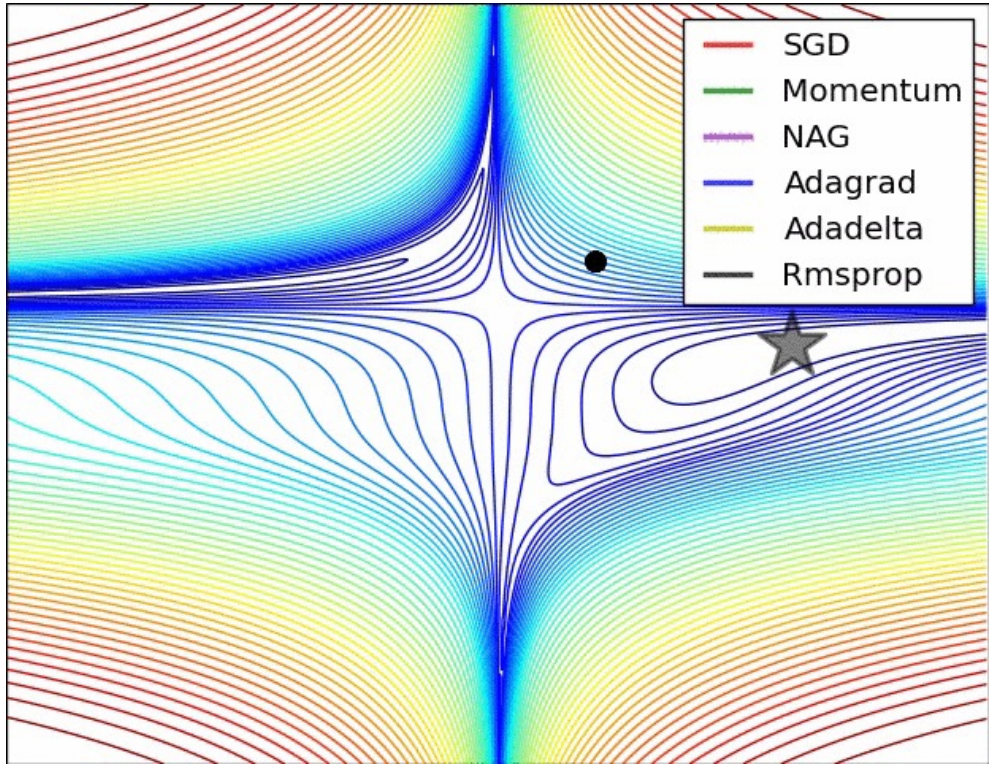
- Step size = learning rate
  - (a) When learning rate is too small, convergence to good solution will be slow
  - (b) When learning rate is too large, convergence to a good solution is not possible



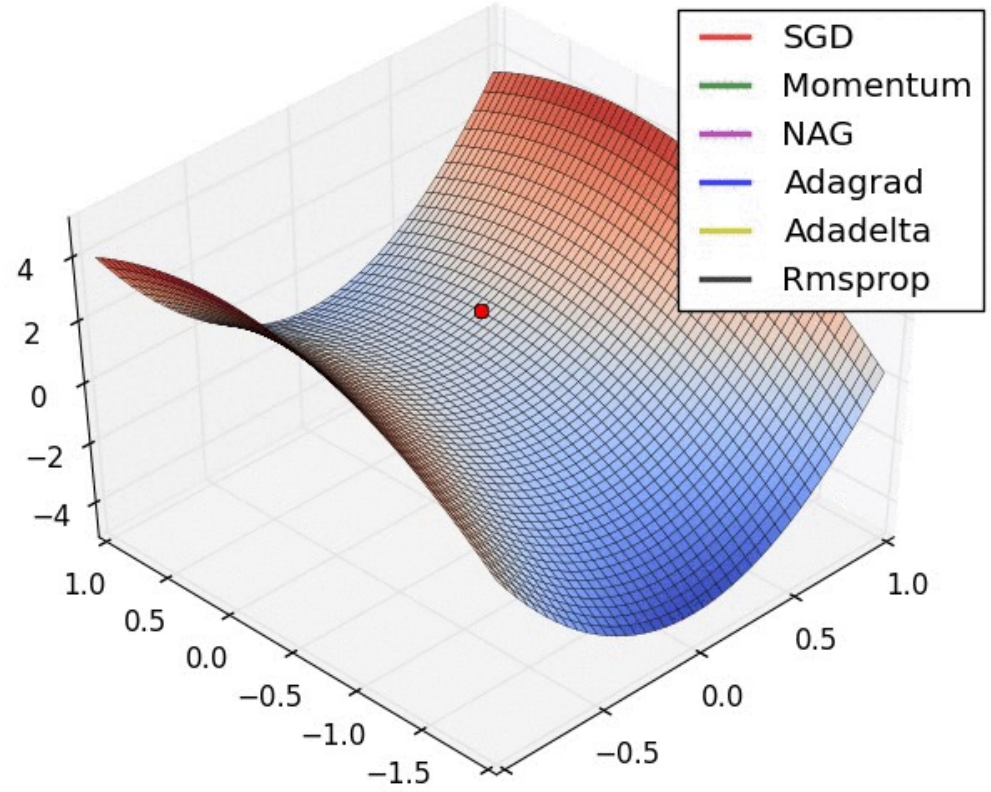
- Many ways to use the gradients

https://github.com/rasbt/python-machine-learning-book-2nd-edition/blob/master/code/ch02/ch02.ipynb

How Much to Update? – Learning Visualizations



Trajectory of optimization algorithms on contours of a loss surface



Optimization on a 3D function showing how some algorithms can learn for saddle points

Source: http://cs231n.github.io/neural-networks-3/#update

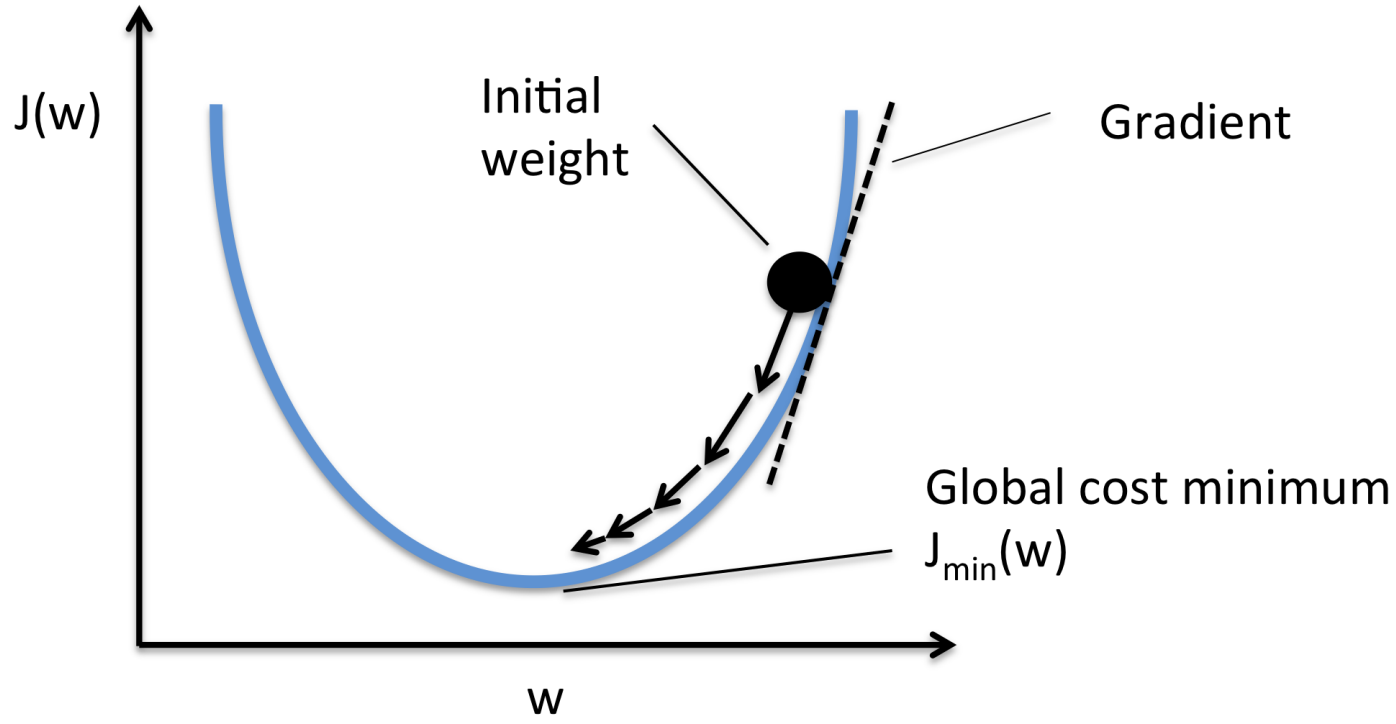
How Much to Update?

- Vanilla Approach:

Parameters Gradient

```
x += - learning_rate * dx
```

Inefficient since steps get smaller as gradient gets smaller



http://cs231n.github.io/neural-networks-3/#update

Figure from: https://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/

How Much to Update?

- Momentum optimization:
  - Analogy: roll a ball down a hill and it will pick up momentum

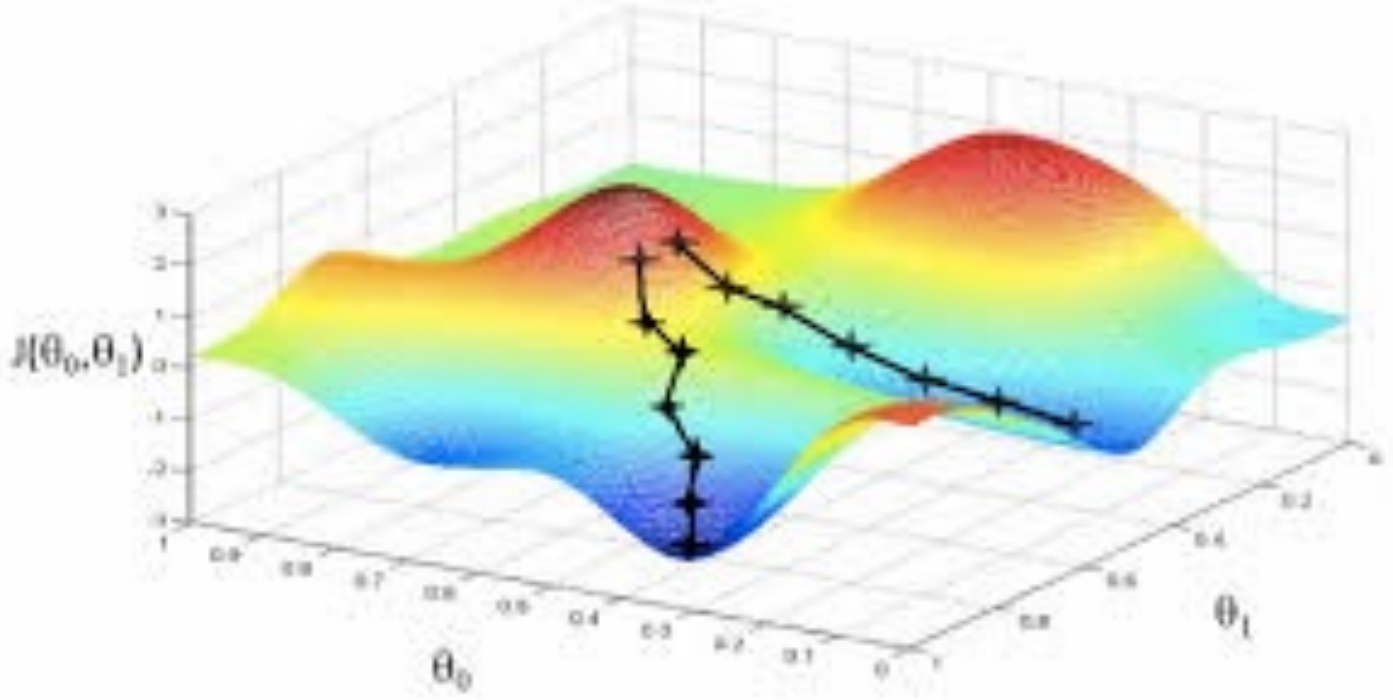


Figure from: https://medium.com/ai-society/hello-gradient-descent-ef74434bdfa5

How Much to Update?

- Momentum optimization:
  - Analogy: roll a ball down a hill and it will pick up momentum

Like friction; values range from 0 to 1 with larger being greater friction

Velocity vector captures cumulative direction of previous gradients; initialized to 0

Gradient not used for speed but instead acceleration

```
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

- What are advantages and disadvantages?
  - Can roll past local minima ☺
  - It may roll past optimum and oscillate around it ☹
  - Another hyperparameter to tune: mu ☹

http://cs231n.github.io/neural-networks-3/#update

How Much to Update?

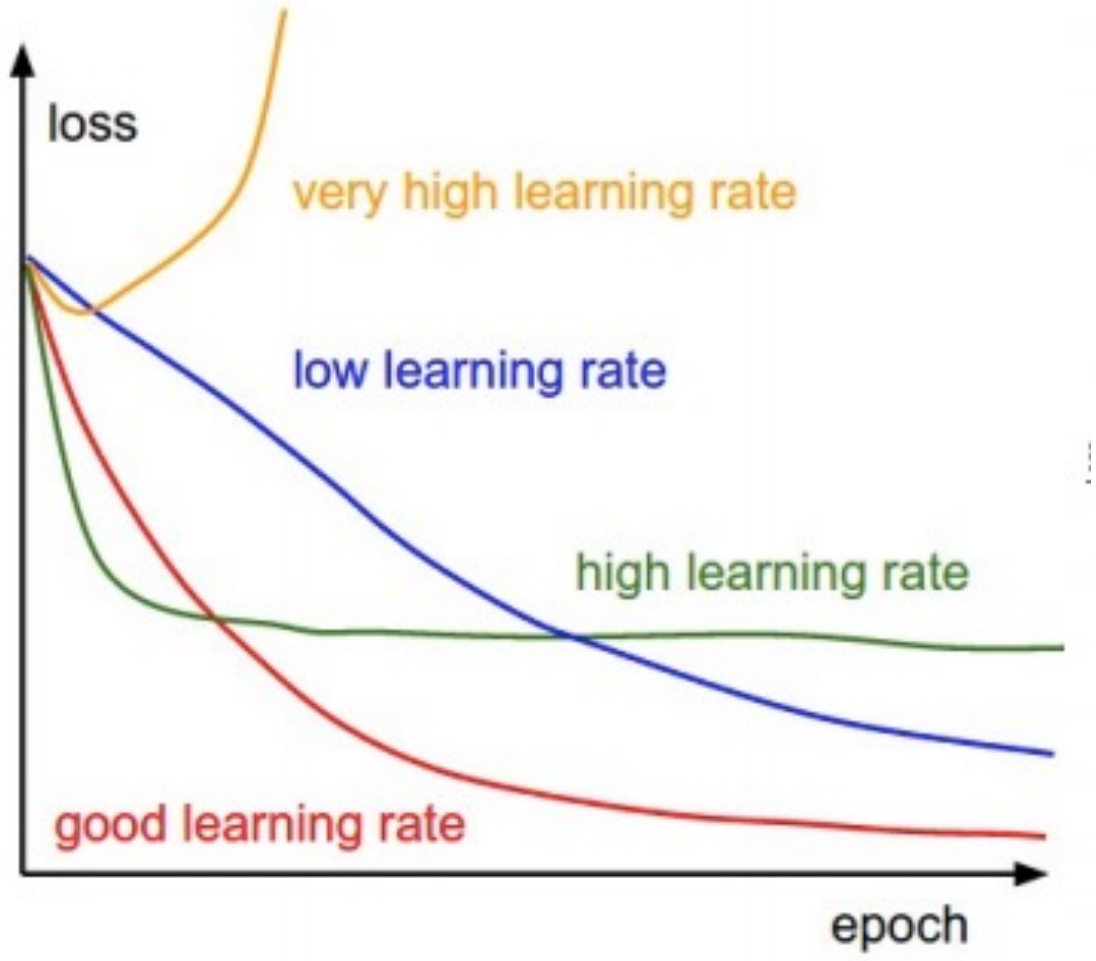
- Step decay:
  - Reduce the learning rate by some factor every few epochs
- Exponential decay
- 1/t decay
- Adapt learning rate per-parameter
- e.g., AdaGrad, RMSprop, and Adam (i.e., adaptive momentum – very popular in practice)

http://cs231n.github.io/neural-networks-3/#update

Today's Topics

- Motivation
- Activation functions
- Initialization
- Efficient learning
- Analyzing Loss Curves
- Programming tutorial

During Training, You Should Ask Yourself: What Does the Observed Loss Behavior Mean?

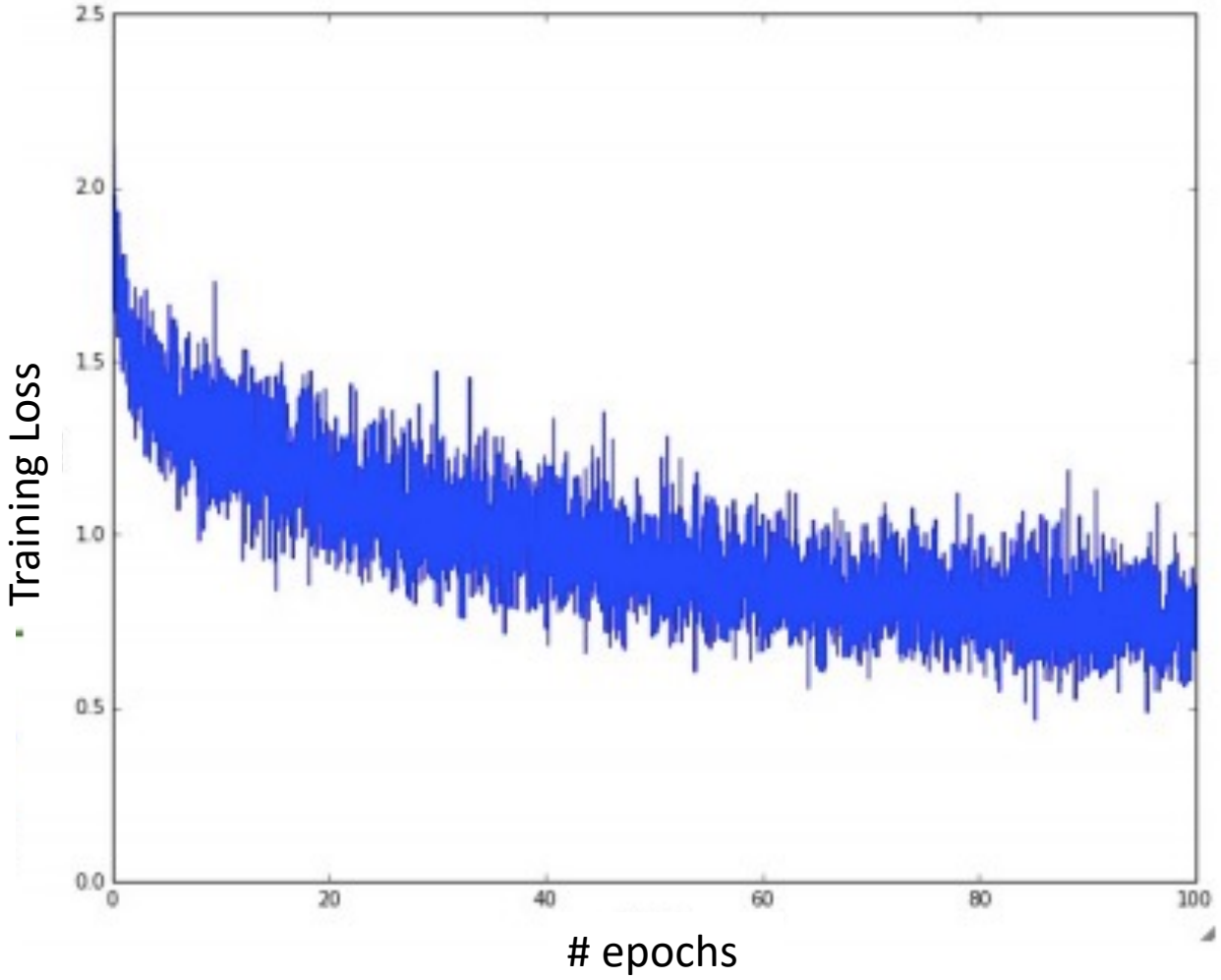


https://cs231n.github.io/neural-networks-3/#update

During Training, You Should Ask Yourself: What Does the Observed Loss Behavior Mean?

- Loss curves signal how well training is going
- Can address potential concerns by debugging the training process for each hypothesized issue one-by-one: e.g.,
  - learning rate too high
  - learning rate too low
  - too small of mini-batch size
  - too many dead neurons resulting from poor weight initialization

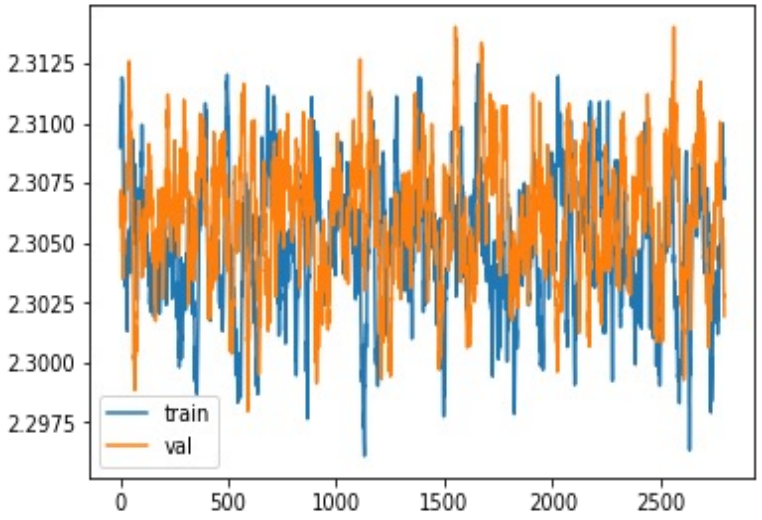
Analysis: Why Might There Be Oscillations in the Learning Curve for the Training Loss?



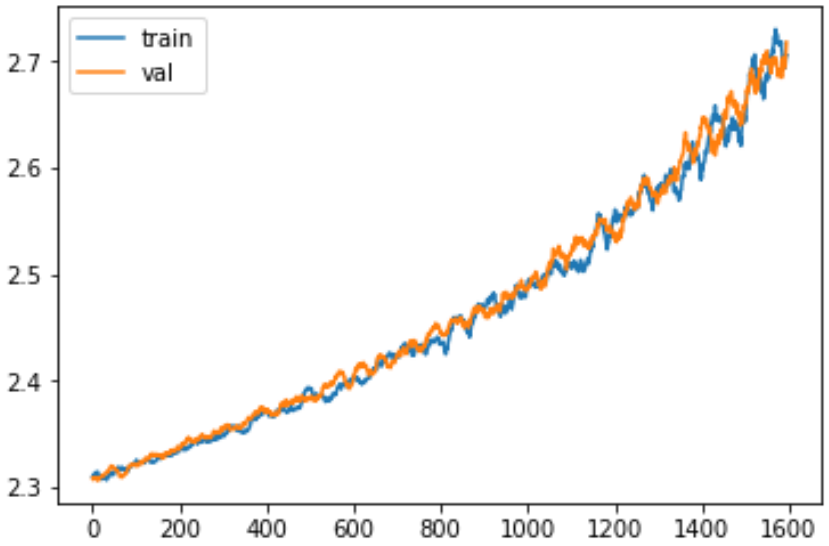
https://cs231n.github.io/neural-networks-3/#update

Discussion: From These Learning Curves, What Do You Think Is Happening and What Might Be a Fix?

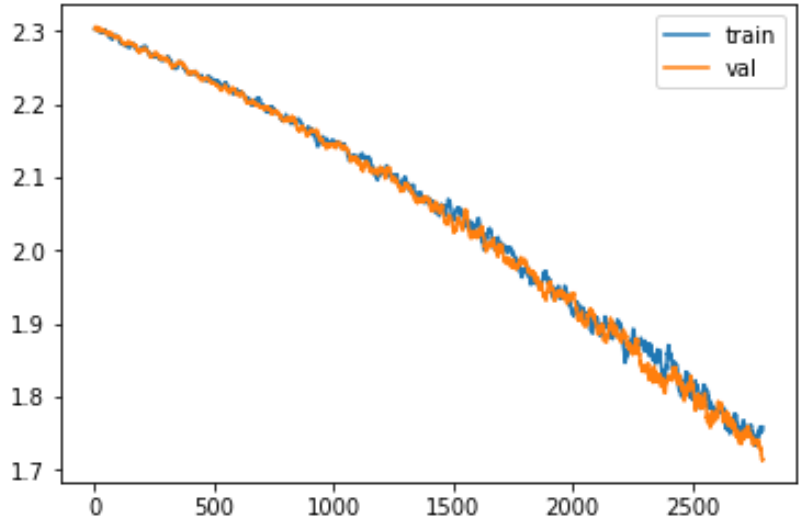
(a)



(b)



(c)



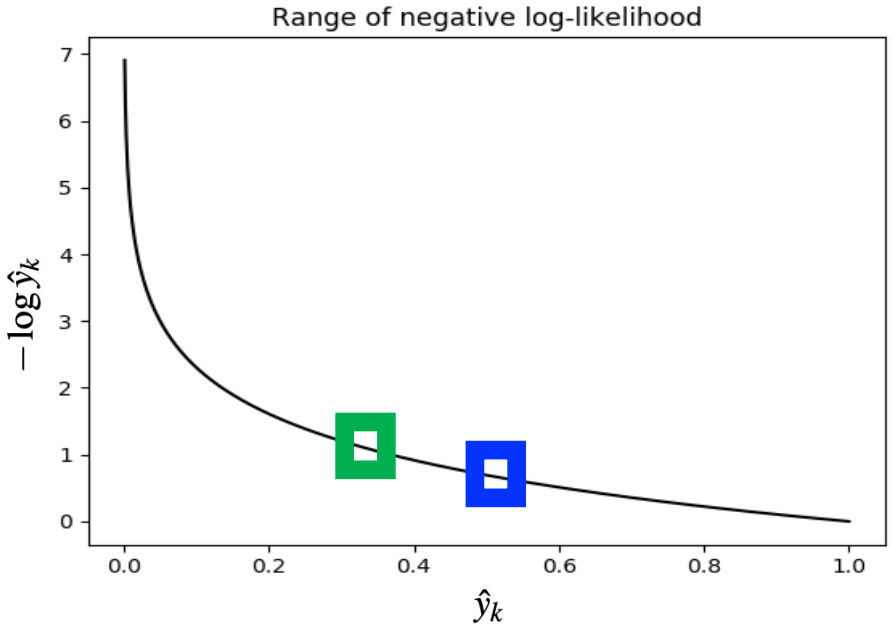
Feeling Bewildered By Your Learning Curves?

You may feel better when looking at this link:

https://lossfunctions.tumblr.com/

Note: Losses Can Mismatch Evaluation Scores

- As loss decreases, the evaluation score *can* increase; e.g., 2 examples with true labels 1 and 0; threshold for predicting label 1 is 0.5; uses cross entropy loss
  - Classifier predicts 0.51 and 1.0 respectively:
    - What is the accuracy: 0, 0.5, or 1?
    - Total loss: 0.97, Accuracy = 0.5
  - Classifier predicts 0.49 and 0.51 respectively:
    - What is the accuracy – 0, 0.5, or 1?
    - Total loss: 0.70, Accuracy: 0
  - The softmax score (the predicted value) from the more similar scores (0.49, 0.51) results in smaller losses (–log scores) than for less similar scores (0.51, 1)



$$= -\log \frac{\exp(w_k \cdot x + b_k)}{\sum_{j=1}^{K} \exp(w_j \cdot x + b_j)}$$

Today's Topics

- Motivation
- Activation functions
- Initialization
- Efficient learning
- Analyzing Loss Curves
- Programming tutorial

Today’s Topics

- Motivation
- Activation functions
- Initialization
- Efficient learning
- Analyzing Loss Curves
- Programming tutorial

The End