CS60010: Deep Learning

Sudeshna Sarkar

Spring 2018

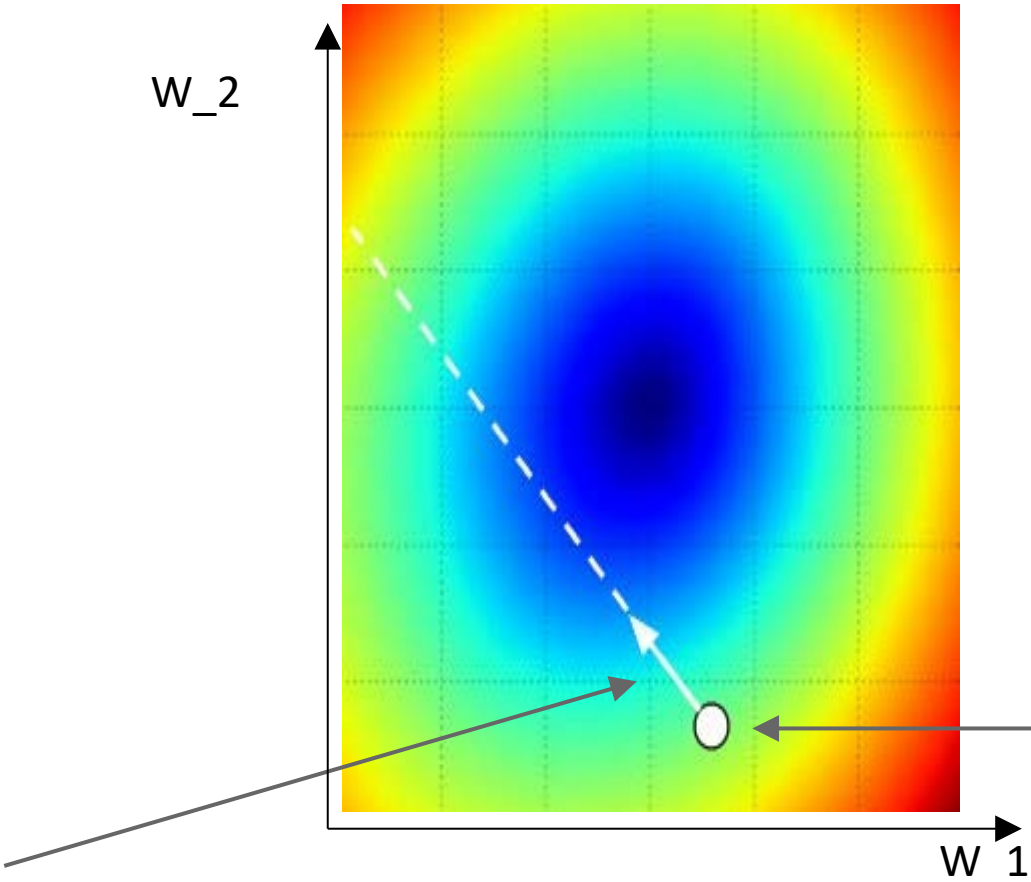
16 Jan 2018

FFN

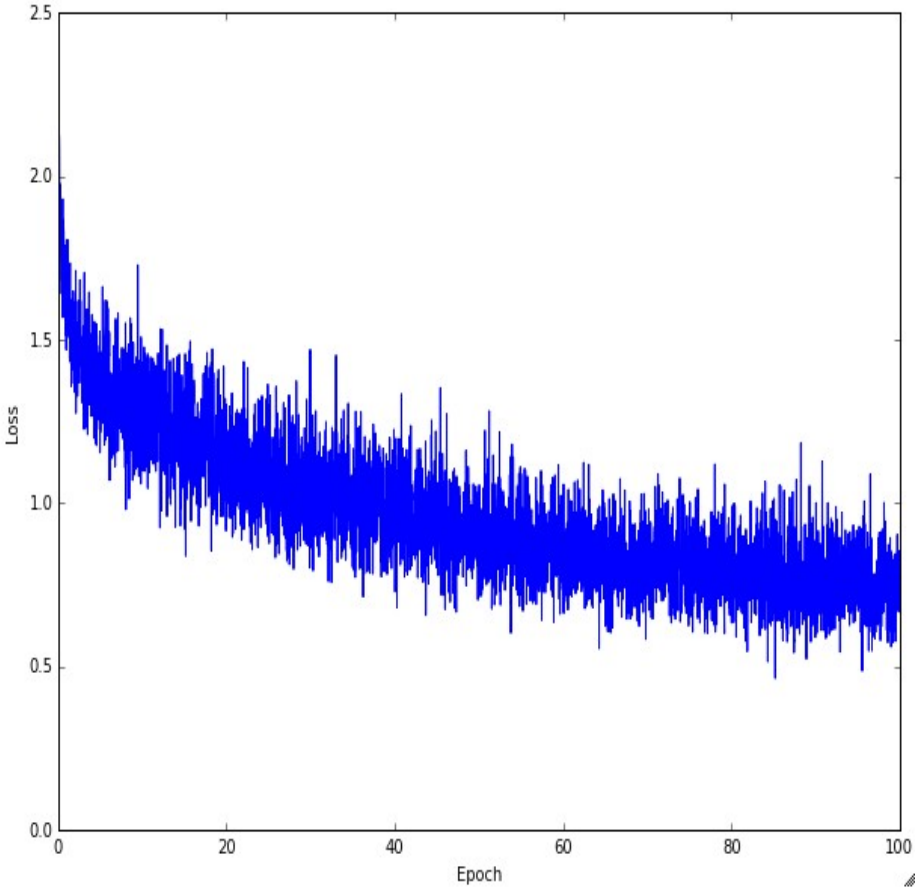
- Goal: Approximate some unknown ideal function f : X ! Y
- Ideal classifier: y = f*(x) with x and category y
- Feedforward Network: Define parametric mapping
- y = f(x; theta)
- Learn parameters theta to get a good approximation to f* from available sample

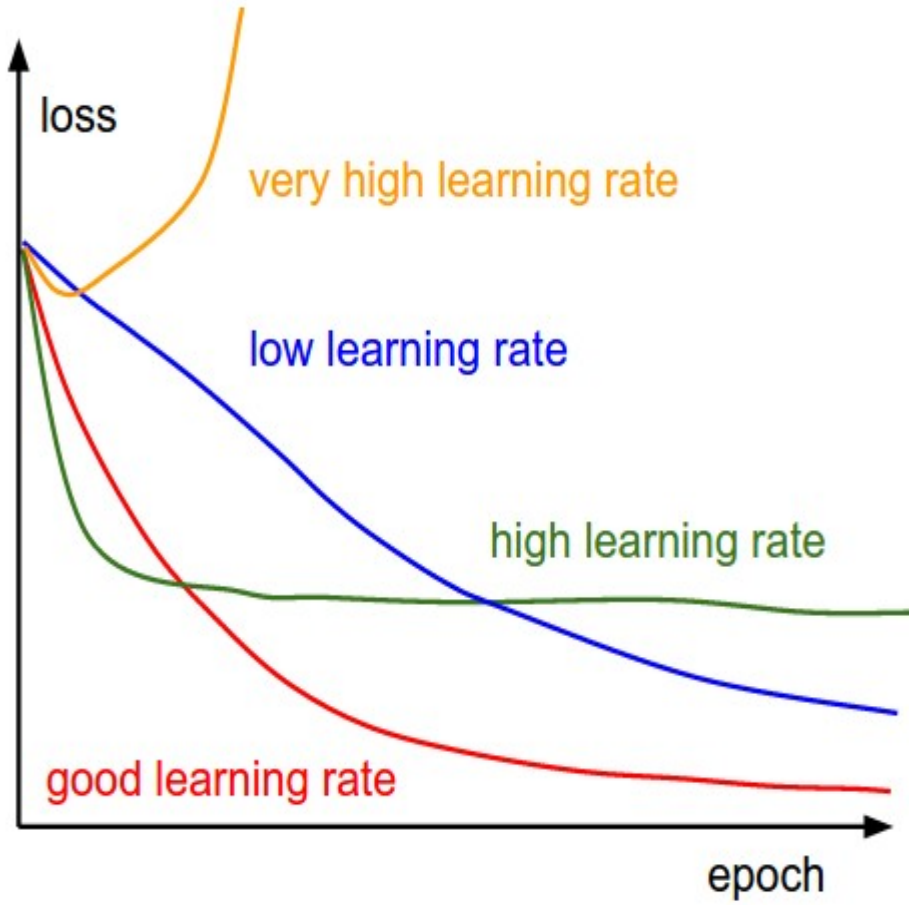
- Function f is a composition of many different functions

Gradient Descent



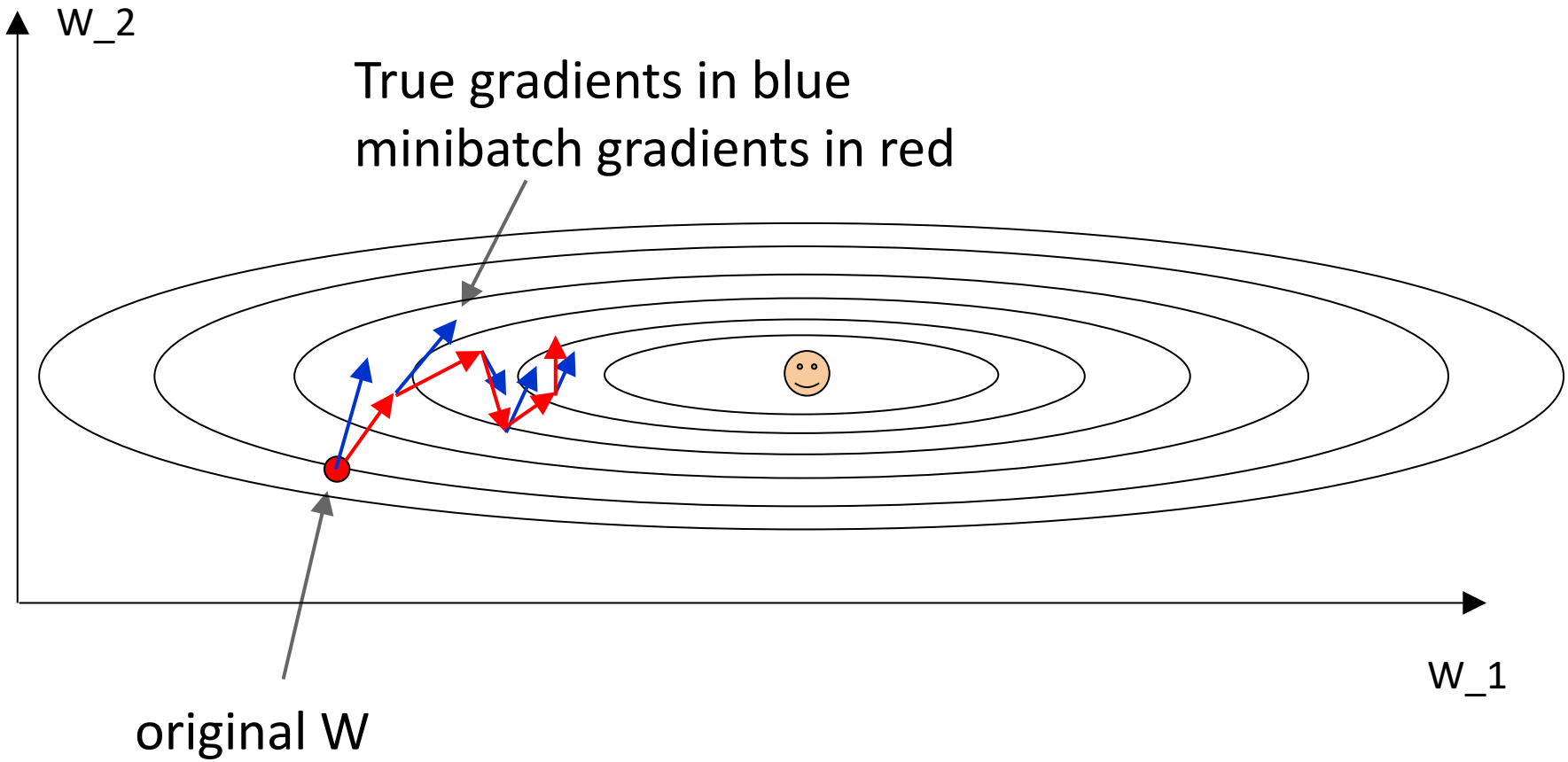
The effects of step size (or “learning rate”)





Slides based on cs231n by Fei-Fei Li & Andrej Karpathy & Justin Johnson

Stochastic Gradient Descent



Gradients are noisy but still make good progress on average

Cost functions:

- In most cases, our parametric model defines a distribution $p(y|x;\theta)$
- Use the principle of maximum likelihood

- The cost function is often the negative log-likelihood
- equivalently described as the cross-entropy between the training data and the model distribution.

$$J(\theta) = -E_{x,y\sim\hat{p}_{data}} \log p_{model}(y|x)$$

- .

Conditional Distributions and Cross-Entropy

$$J(\theta) = -E_{x,y\sim\hat{p}_{data}} \log p_{model}(y|x)$$

- The specific form of the cost function changes from model to model, depending on the specific form of $\log p_{model}$
- For example, if $p_{model}(y|x) = \mathcal{N}(\mathrm{y}; \mathrm{f}(\mathrm{x}, \theta), \mathrm{I})$ then we recover the mean squared error cost,

$$J(\theta) = \frac{1}{2} E_{x,y\sim\hat{p}_{data}} \|y - f(x;\theta)\|^2 + \mathrm{Const}$$

- For predicting median of Gaussian, the equivalence between maximum likelihood estimation with an output distribution and minimization of mean squared error holds
- Specifying a model $p(y|x)$ automatically determines a cost function $\log p(y|x)$

- The gradient of the cost function must be large and predictable enough to serve as a good guide for the learning algorithm.
- The negative log-likelihood helps to avoid this problem for many models.

Learning Conditional Statistics

- Sometimes we merely predict some statistic of y conditioned on x.
  Use specialized loss functions
  - For example, we may have a predictor $f(x;\theta)$ that we wish to employ to predict the mean of y.
- With a sufficiently powerful neural network, we can think of the NN as being able to represent any function f from a wide class of functions.
  - view the cost function as being a *functional* rather than just a function.
  - A functional is a mapping from functions to real numbers.
- We can thus think of learning as choosing a function rather than a set of parameters.
- We can design our cost functional to have its minimum occur at some specific function we desire. For example, we can design the cost functional to have its minimum lie on the function that maps x to the expected value of y given x.

Learning Conditional Statistics

- Results derived using calculus of variations:

1. Solving the optimization problem

$$f^* = \underset{f}{argmin}\, \mathbb{E}_{x,y\sim p_{data}} \|y - f(x)\|^2$$

yields

$$f^*(x) = \mathbb{E}_{x,y\sim p_{data}(y|x)}[y]$$

2. Solving

$$f^* = \underset{f}{argmin}\, \mathbb{E}_{x,y\sim p_{data}} \|y - f(x)\|_1$$

yields a function that predicts the median value of y for each x.

Mean Absolute Error (MAE)

- MSE and MAE often lead to poor results when used with gradient-based optimization. Some output units that saturate produce very small gradients when combined with these cost functions.
- Thus use of cross-entropy is popular even when it is not necessary to estimate the distribution $p(y|x)$

Output Units

1. *Linear units for Gaussian Output Distributions*. Linear output layers are often used to produce the mean of a conditional Gaussian distribution $p(y|x) = \mathcal{N}(y; \hat{y}, I)$
   - Maximizing the log-likelihood is then equivalent to minimizing the mean squared error
   - Because linear units do not saturate, they pose little difficulty for gradient-based optimization algorithms
2. *Sigmoid Units for Bernoulli Output Distributions*. 2-class classification problem. Needs to predict $P(y = 1|x)$.
$$\hat{y} = \sigma(w^T h + b)$$
3. *Softmax Units for Multinoulli Output Distributions*. Any time we wish to represent a probability distribution over a discrete variable with n possible values, we may use the softmax function. Softmax functions are most often used as the output of a classifier, to represent the probability distribution over n different classes.

Softmax output

- In case of a discrete variable with k values, produce a vector $\hat{\boldsymbol{y}}$ with $\hat{y}_i = P(y = i|x)$.
- A linear layer predicts unnormalized log probabilities: $\boldsymbol{z} = \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{h} + \boldsymbol{b}$
- where $z_i = \log \tilde{P}(y = i|x)$

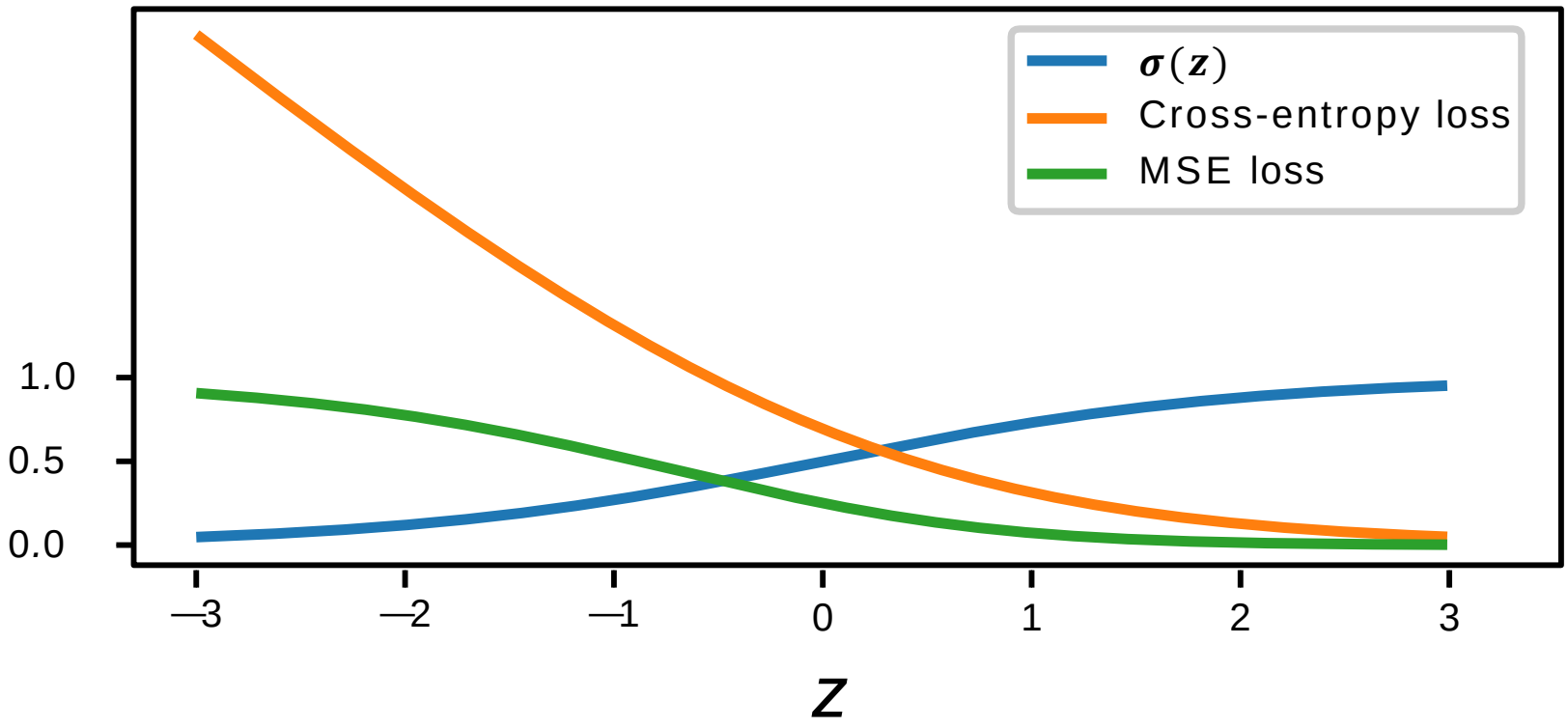
$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- When training the softmax to output a target value y using maximum log-likelihood
- Maximize $\log P(y = i; z) = \log \ softmax(z)_i$
- $\log \ softmax(z)_i = z_i - \log \sum_j \exp(z_j)$

Output Types

Output Type	Output Distribution	Output Layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary cross-entropy
Discrete	Multinoulli	Softmax	Discrete cross-entropy
Continuous	Gaussian	Linear	Gaussian cross-entropy (MSE)
Continuous	Mixture of Gaussian	Mixture Density	Cross-entropy
Continuous	Arbitrary	GAN, VAE, FVBN	Various

Sigmoid output with target of 1



Bad idea to use MSE loss with sigmoid unit.

Sigmoid Units

- Task: Predict a binary variable y
- Use a sigmoid unit:

$$\hat{y} = \sigma(\mathbf{w}^T \mathbf{h} + b)$$

- Cost:

$$J(\theta) = -\log p(y|\mathbf{x}) = -\log \sigma((2y-1)(\mathbf{w}^T \mathbf{h} + b))$$

- Positive: Only saturates when model already has right answer i.e. when $y = 1$ and $(\mathbf{w}^T \mathbf{h} + b)$ is very positive and vice versa
- When $(\mathbf{w}^T \mathbf{h} + b)$ has wrong sign, a good gradient is returned

Softmax Units

- Need to produce a vector $\hat{\mathbf{y}}$ with $\hat{y}_i = p(y = i|\mathbf{x})$
- Linear layer first produces unnormalized log probabilities: $\mathbf{z} = W^T\mathbf{h} + \mathbf{b}$
- Softmax:

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- Log of the softmax (since we wish to maximize $p(y = i; \mathbf{z})$):

$$\log \text{softmax}(\mathbf{z})_i = z_i - \log \sum_j \exp(z_j)$$

Benefits

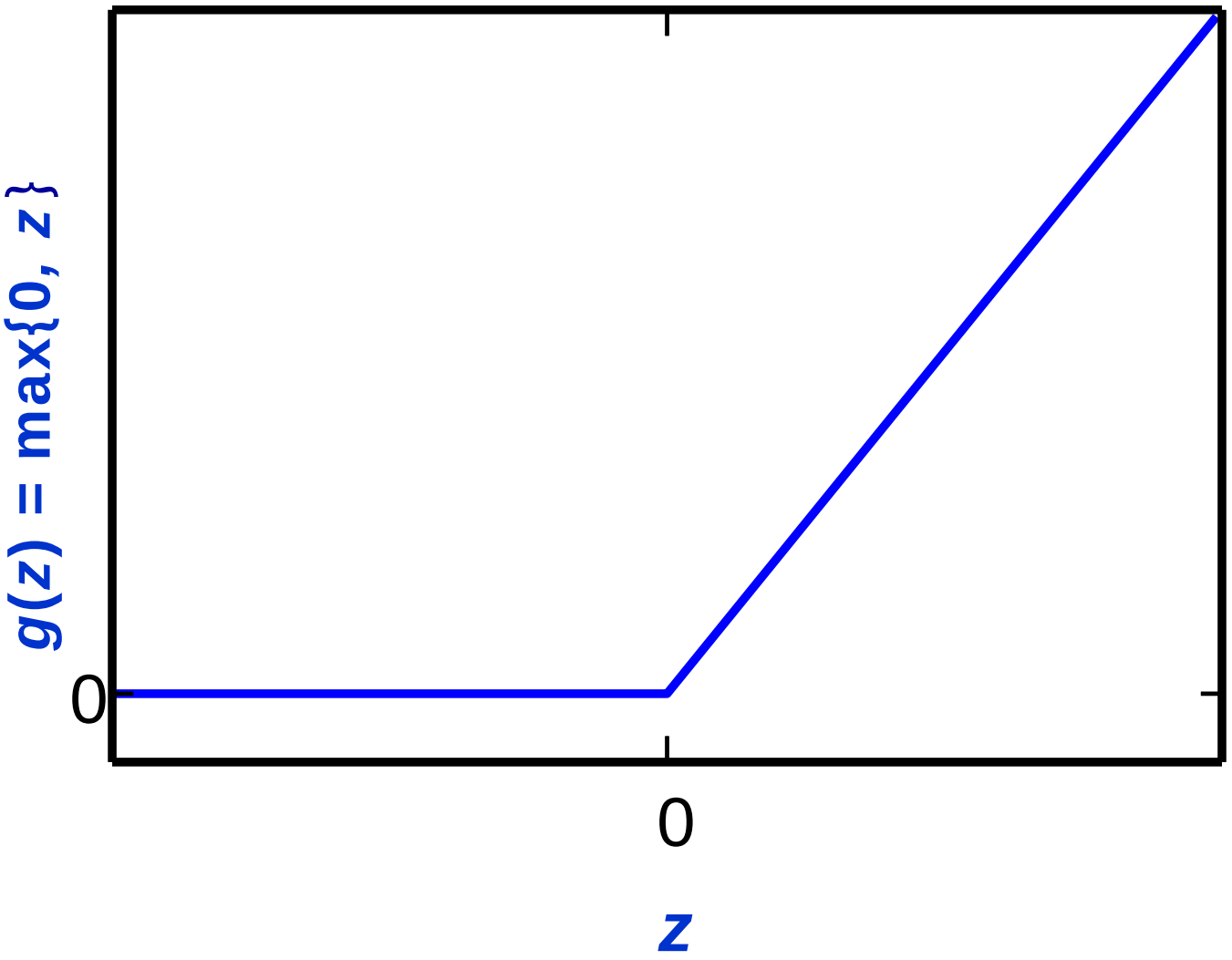
$$\log \text{softmax}(\mathbf{z})_i = z_i - \log \sum_j \exp(z_j)$$

- z_i term never saturates, making learning easier
- Maximizing log-likelihood encourages z_i to be pushed up, while encouraging all $\mathbf{z}$ to be pushed down (Softmax encourages competition)
- More intuition: Think of $\log \sum_j \exp(z_j) \approx \max_j z_j$ (why?)
- log-likelihood cost function ($\sim z_i - \max_j z_j$) strongly penalizes the most active *incorrect* prediction
- If model already has correct answer then $\log \sum_j \exp(z_j) \approx \max_j z_j$ and z_i will roughly cancel out
- Progress of learning is dominated by incorrectly classified examples

Hidden Units

- Rectified linear units are an excellent default choice of hidden unit.
- Use ReLUs, 90% of the time
- Many hidden units perform comparably to ReLUs.
  New hidden units that perform comparably are rarely interesting.

Rectified Linear Activation



ReLU

- Positives:
  - Gives large and consistent gradients (does not saturate) when active
  - Efficient to optimize, converges much faster than sigmoid or tanh
- Negatives:
  - Non zero centered output
  - Units "die" i.e. when inactive they will never update

Architecture Basics

- Depth
- Width

Universal Approximator Theorem

- One hidden layer is enough to *represent* (not *learn*) an approximation of any function to an arbitrary degree of accuracy
- So why deeper?
  - Shallow net may need (exponentially) more width
  - Shallow net may overfit more

- http://mcneela.github.io/machine_learning/2017/03/21/Universal-Approximation-Theorem.html
- https://blog.goodaudience.com/neural-networks-part-1-a-simple-proof-of-the-universal-approximation-theorem-b7864964dbd3
- http://neuralnetworksanddeeplearning.com/chap4.html