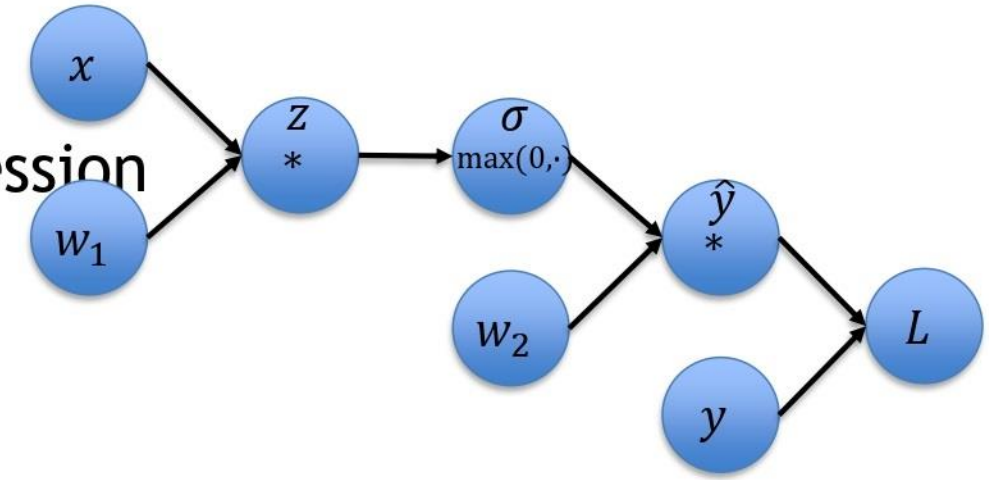
Lecture 6
Optimization

Back to Compute Graphs & NNs

- Inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Two-layer NN for regression with ReLU activation
- Function we want to optimize:

$$\sum_{i=1}^{n} \|w_2 \max(0, w_1 x_i) - y_i\|_2^2$$



Gradient Descent for Neural Networks

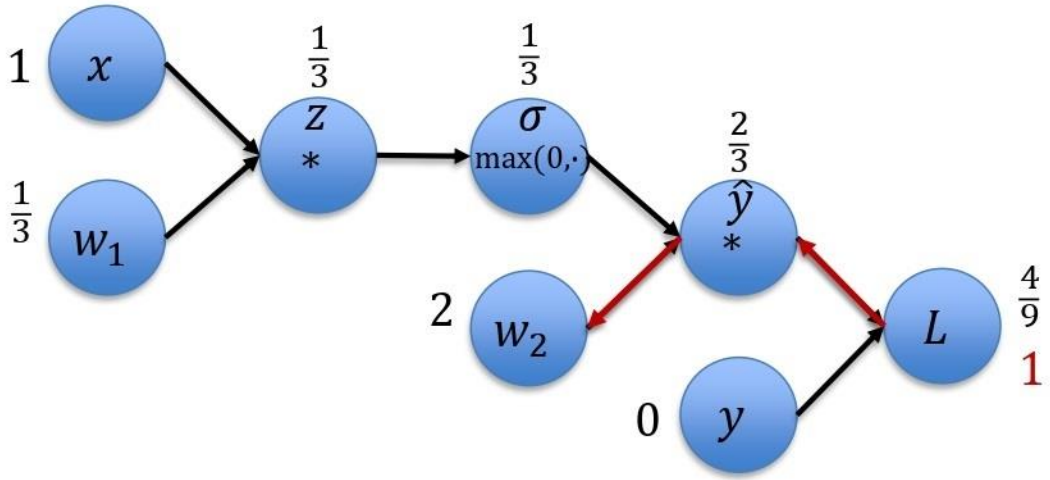
Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

$L(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta}) = \frac{1}{n}\sum_i^n ||\hat{y}_i - y_i||_2^2$

In our case $n, d = 1$:

$L = (\hat{y} - y)^2 \quad \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma \quad \Rightarrow \frac{\partial \hat{y}}{\partial w_2} = \sigma$



Backpropagation

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$

Gradient Descent for Neural Networks

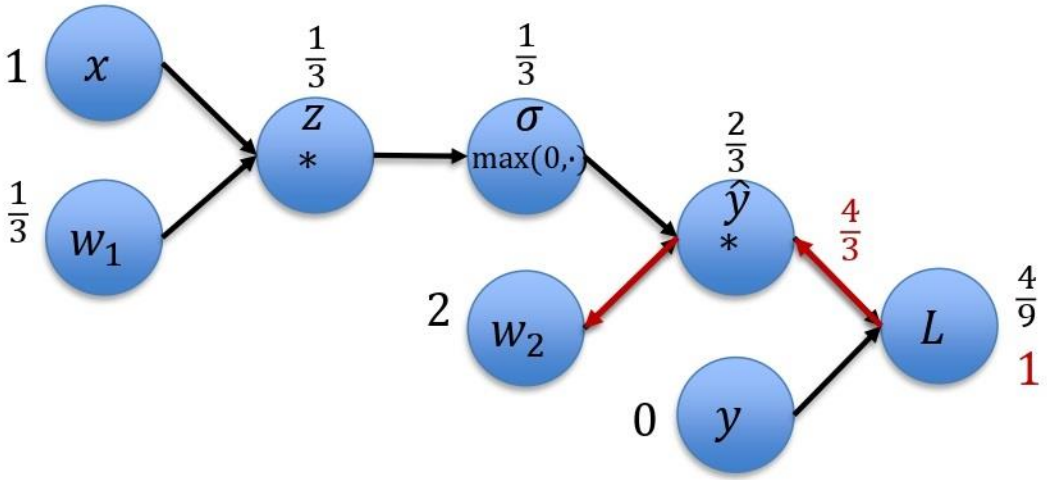
Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

$L(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta}) = \frac{1}{n}\sum_i^n ||\hat{y}_i - y_i||_2^2$

In our case $n, d = 1$:

$L = (\hat{y} - y)^2 \quad \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma \quad \Rightarrow \frac{\partial \hat{y}}{\partial w_2} = \sigma$



Backpropagation

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$

$2 \cdot \frac{2}{3}$

Gradient Descent for Neural Networks

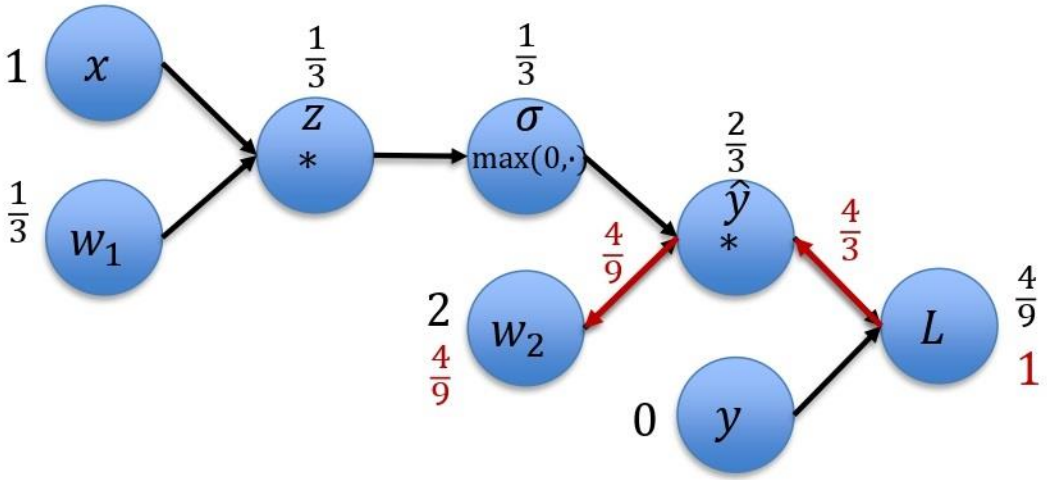
Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

$L(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta}) = \frac{1}{n}\sum_i^n ||\hat{y}_i - y_i||_2^2$

In our case $n, d = 1$:

$L = (\hat{y} - y)^2 \ \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma \ \Rightarrow \boxed{\frac{\partial \hat{y}}{\partial w_2} = \sigma}$



Backpropagation

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$

$$2 \cdot \frac{2}{3} \cdot \frac{1}{3}$$

Gradient Descent for Neural Networks

Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

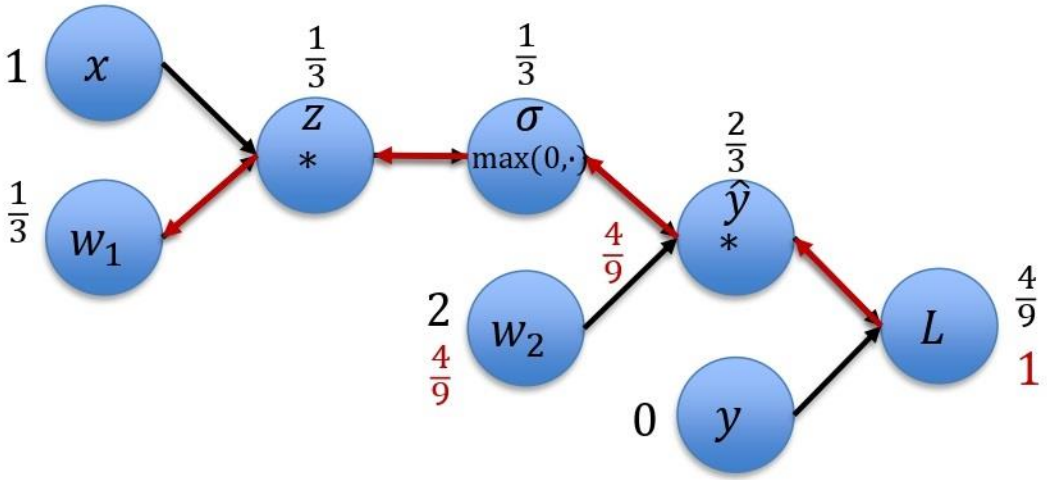
In our case $n, d = 1$:

$$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial \hat{y}}{\partial \sigma} = w_2$$

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

$$z = x \cdot w_1 \Rightarrow \frac{\partial z}{\partial w_1} = x$$



Backpropagation

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

Gradient Descent for Neural Networks

Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

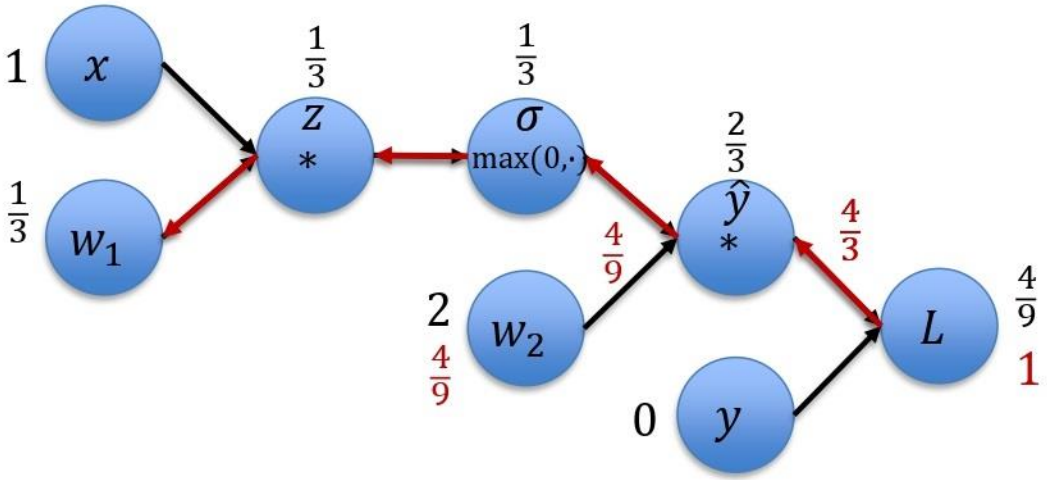
In our case $n, d = 1$:

$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial \hat{y}}{\partial \sigma} = w_2$

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

$z = x \cdot w_1 \Rightarrow \frac{\partial z}{\partial w_1} = x$



Backpropagation

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

$2 \cdot \frac{2}{3}$

Gradient Descent for Neural Networks

Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

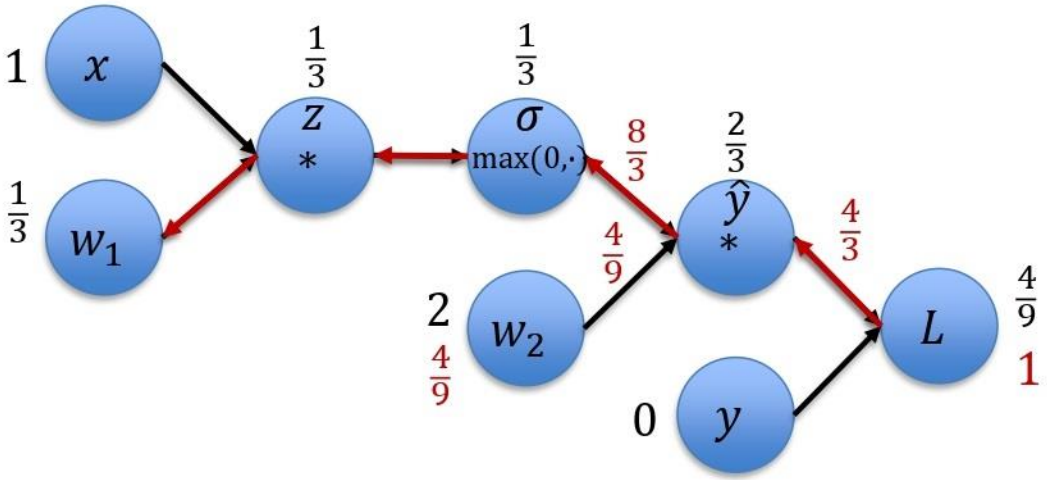
In our case $n, d = 1$:

$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial \hat{y}}{\partial \sigma} = w_2$

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

$z = x \cdot w_1 \Rightarrow \frac{\partial z}{\partial w_1} = x$



Backpropagation

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

$2 \cdot \frac{2}{3} \cdot 2$

Gradient Descent for Neural Networks

Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

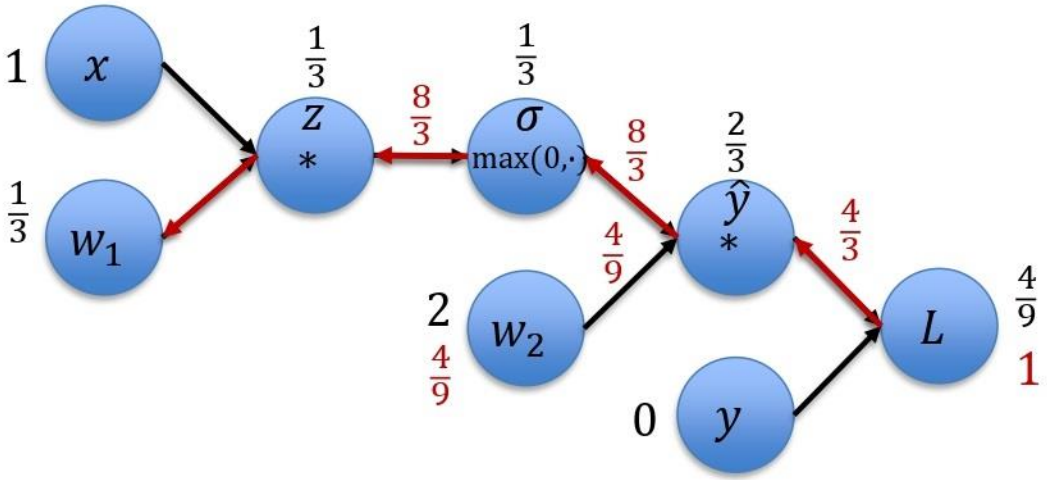
In our case $n, d = 1$:

$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial \hat{y}}{\partial \sigma} = w_2$

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

$z = x \cdot w_1 \Rightarrow \frac{\partial z}{\partial w_1} = x$



Backpropagation

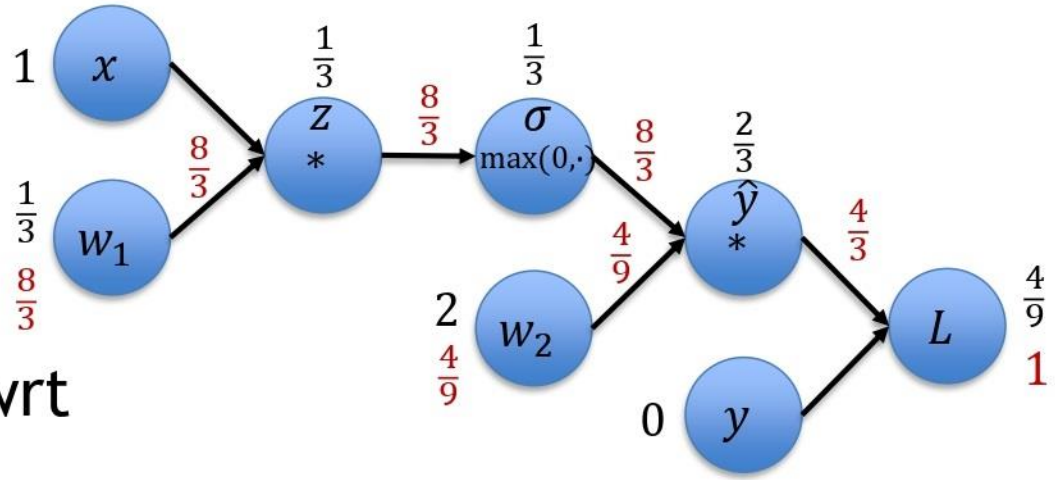
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

$$2 \cdot \frac{2}{3} \cdot 2 \cdot 1$$

Gradient Descent for Neural Networks

- Function we want to optimize:

$$f(x, \boldsymbol{w}) = \sum_{i=1}^{n} \|w_2 \max(0, w_1 x_i) - y_i\|_2^2$$



- Computed gradients wrt to weights w_1 and w_2
- Now: update the weights

$$\boldsymbol{w}' = \boldsymbol{w} - \alpha \cdot \nabla_{\boldsymbol{w}} f = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \alpha \cdot \begin{pmatrix} \nabla_{w_1} f \\ \nabla_{w_2} f \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{3} \\ 2 \end{pmatrix} - \alpha \cdot \begin{pmatrix} \frac{8}{3} \\ \frac{4}{9} \end{pmatrix}$$

But: how to choose a good learning rate α ?

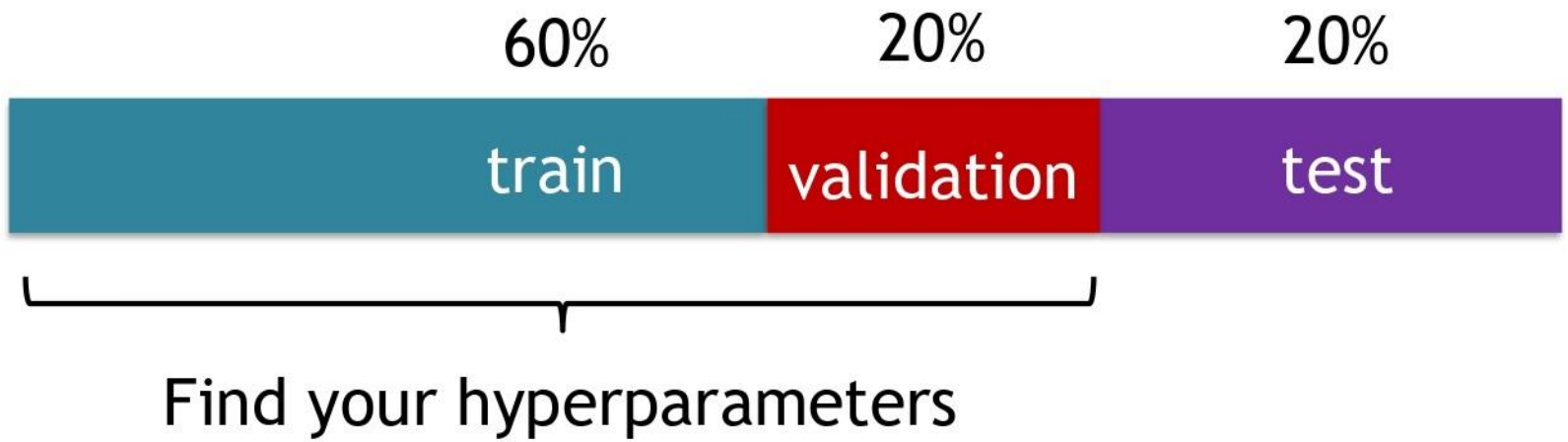
Gradient Descent

- How to pick good learning rate?
- How to compute gradient for single training pair?
- How to compute gradient for large training set?
- How to speed things up? More to see in next lectures…

Regularization

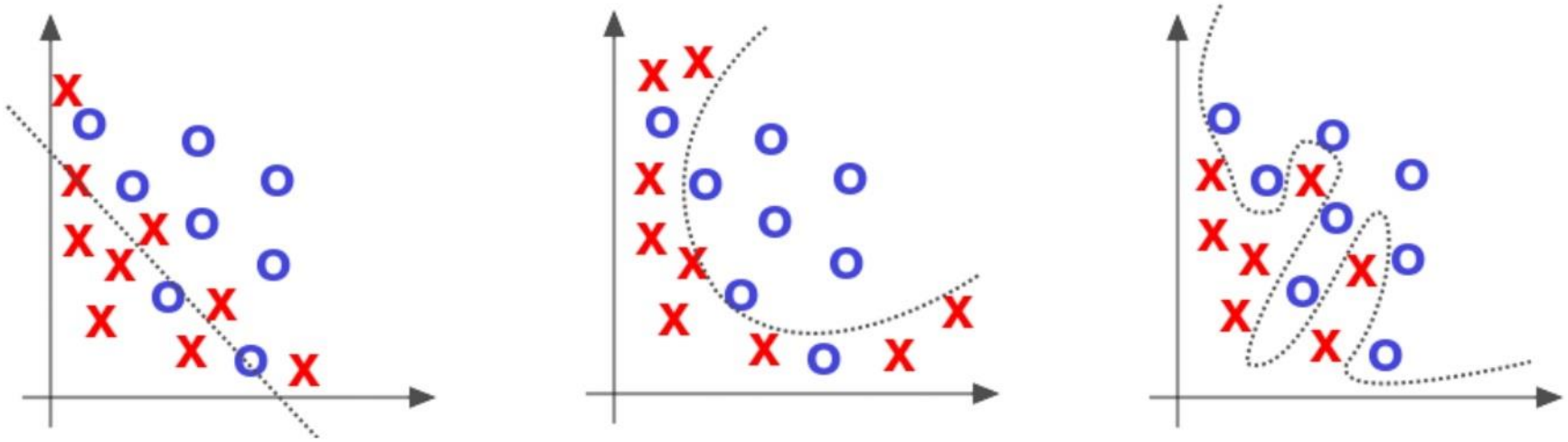
Recap: Basic Recipe for ML

- Split your data



Other splits are also possible (e.g., 80%/10%/10%)

Over- and Underfitting

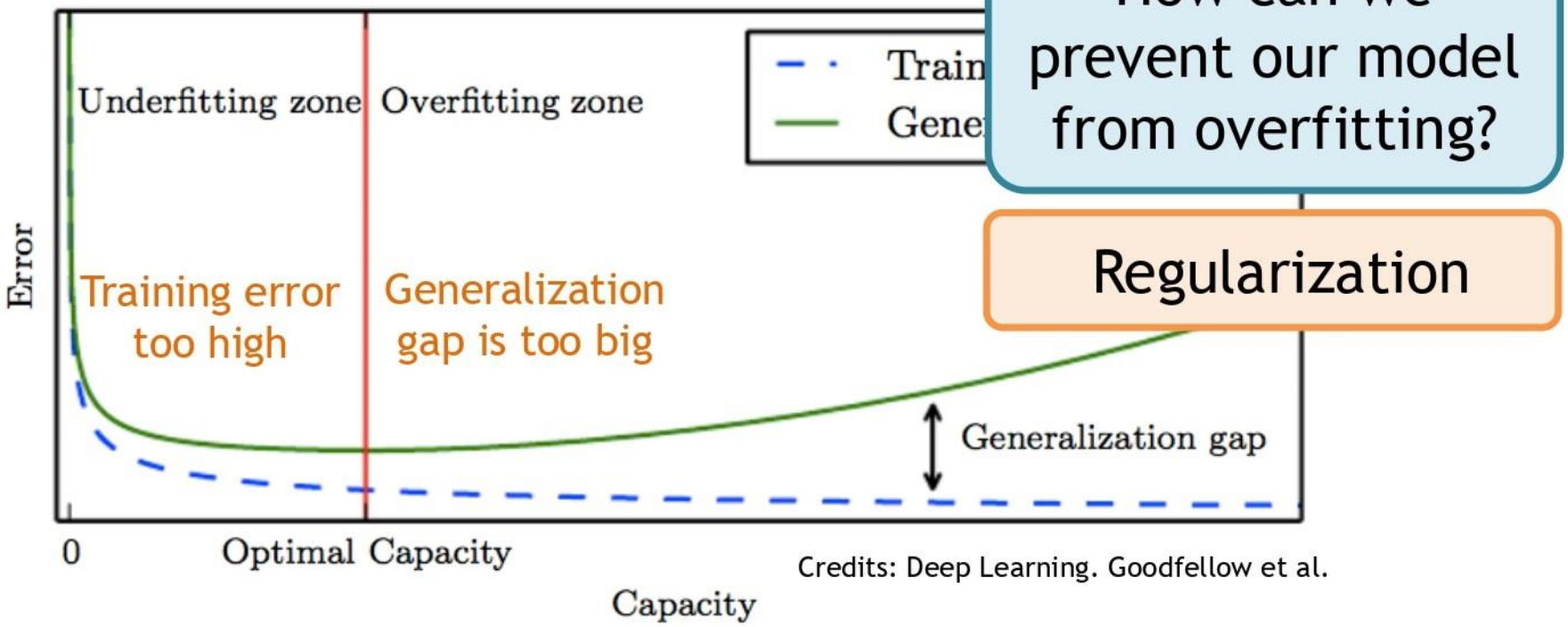


Underfitted **Appropriate** **Overfitted**

Source: Deep Learning by Adam Gibson, Josh Patterson, O'Reily Media Inc., 2017

Training a Neural Network

- Training/ Validation curve



How can we prevent our model from overfitting?

Regularization

Credits: Deep Learning. Goodfellow et al.

Regularization

- Loss function $L(\boldsymbol{y}, \hat{\boldsymbol{y}}, \boldsymbol{\theta}) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$

- Regularization techniques
  - L2 regularization
  - L1 regularization
  - Max norm regularization
  - Dropout
  - Early stopping
  - ...

Add **regularization term** to loss function

More details later

Regularization: Example

- Input: 3 features $\boldsymbol{x} = [1, 2, 1]$
- Two linear classifiers that give the same result:
- $\theta_1 = [0, 0.75, 0]$ → Ignores 2 features
- $\theta_2 = [0.25, 0.5, 0.25]$ → Takes information from all features

Regularization: Example

- Loss $L(\boldsymbol{y}, \hat{\boldsymbol{y}}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \left(x_i \theta_{ji} - y_i\right)^2 + \lambda R(\boldsymbol{\theta})$
- L2 regularization $R(\boldsymbol{\theta}) = \sum_{i=1}^{n} \theta_i^2$

$\theta_1 \longrightarrow 0 + 0.75^2 + 0 = 0.5625$

$\theta_2 \longrightarrow 0.25^2 + 0.5^2 + 0.25^2 = 0.375$ Minimization

$x = [1, 2, 1], \theta_1 = [0, 0.75, 0], \theta_2 = [0.25, 0.5, 0.25]$

Regularization: Example

- Loss $L(\boldsymbol{y}, \hat{\boldsymbol{y}}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \left(x_i \theta_{ji} - y_i\right)^2 + \lambda R(\boldsymbol{\theta})$
- L1 regularization $R(\boldsymbol{\theta}) = \sum_{i=1}^{n} |\theta_i|$

$\theta_1 \longrightarrow 0 + 0.75 + 0 = 0.75$

$\theta_2 \longrightarrow 0.25 + 0.5 + 0.25 = 1$ Minimization

$x = [1, 2, 1], \theta_1 = [0, 0.75, 0], \theta_2 = [0.25, 0.5, 0.25]$

Regularization: Example

- Input: 3 features $\boldsymbol{x} = [1, 2, 1]$

- Two linear classifiers that give the same result:

$\theta_1 = [0, 0.75, 0]$ ⟶ Ignores 2 features

$\theta_2 = [0.25, 0.5, 0.25]$ ⟶ Takes information from all features

Regularization: Example

- Input: 3 features $\boldsymbol{x} = [1, 2, 1]$

- Two linear classifiers that give the same result:

$\theta_1 = [0, 0.75, 0]$ → L1 regularization enforces **sparsity**

$\theta_2 = [0.25, 0.5, 0.25]$ → Takes information from all features

Regularization: Example

- Input: 3 features $\boldsymbol{x} = [1, 2, 1]$

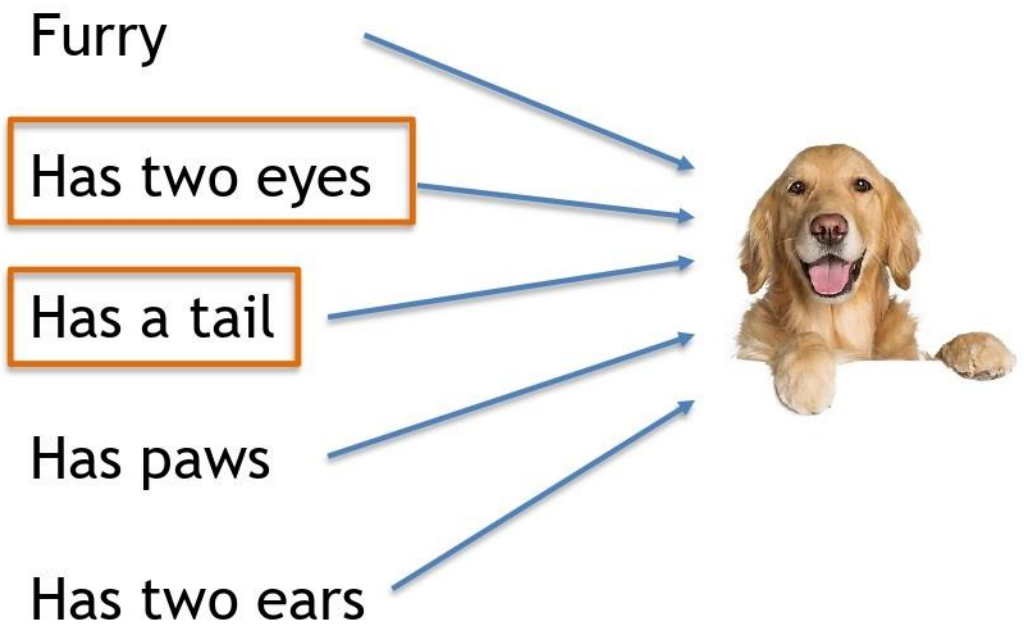
- Two linear classifiers that give the same result:

$\theta_1 = [0, 0.75, 0]$ → L1 regularization enforces **sparsity**

$\theta_2 = [0.25, 0.5, 0.25]$ → L2 regularization enforces that the weights have **similar values**

Regularization: Effect

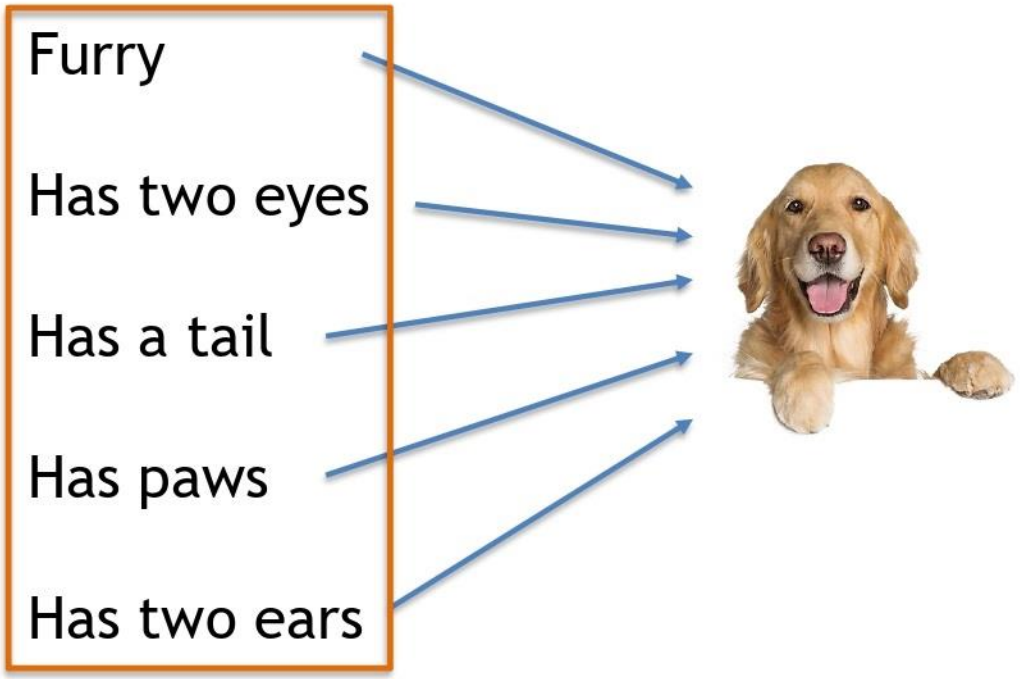
- Dog classifier takes different inputs



L1 regularization will focus all the attention to a few key features

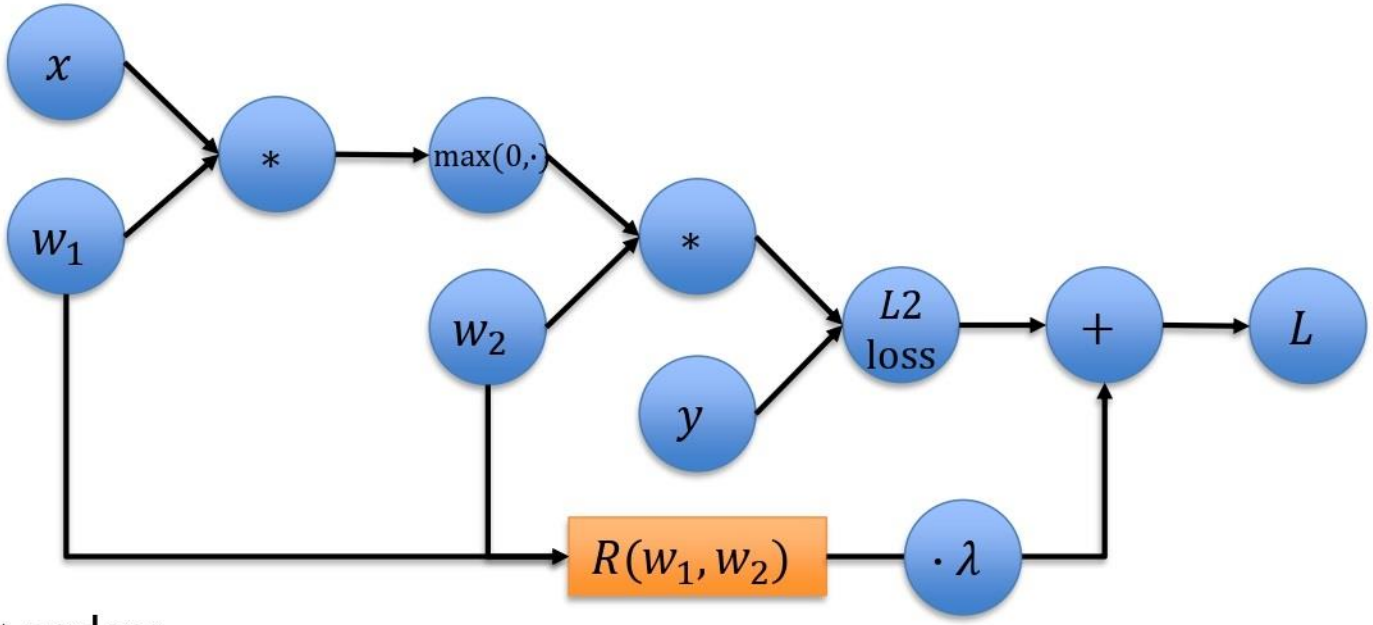
Regularization: Effect

- Dog classifier takes different inputs



L2 regularization will take all information into account to make decisions

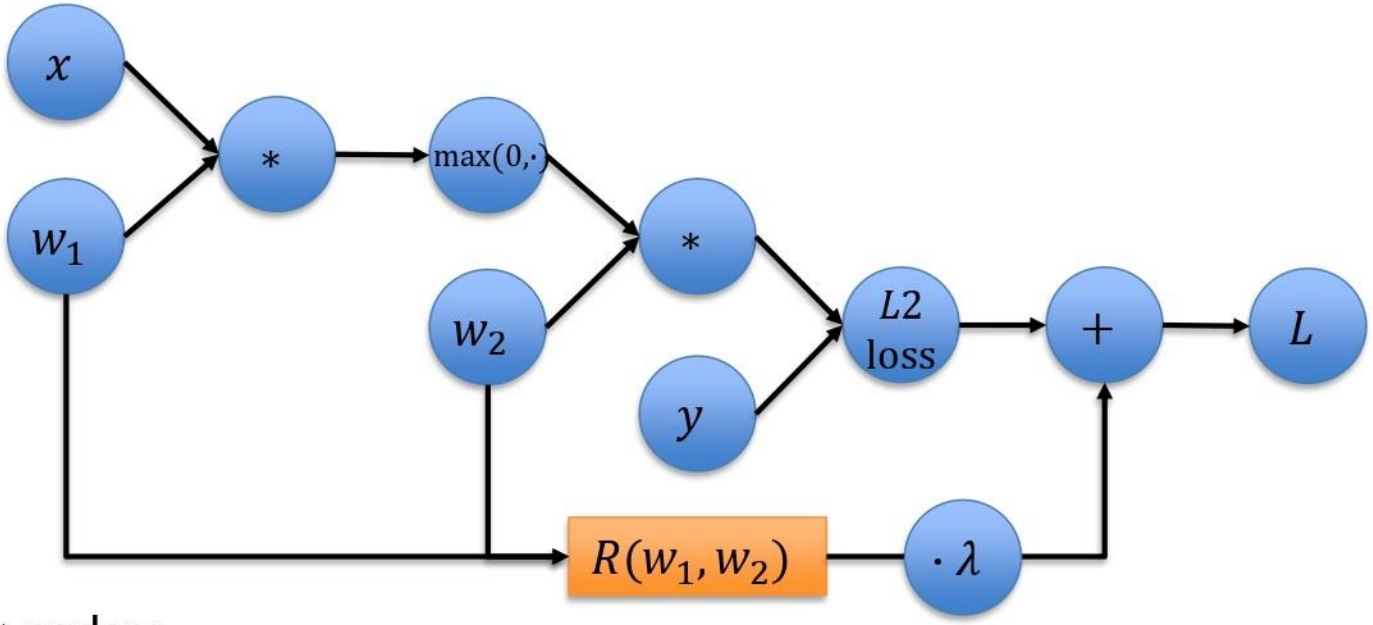
Regularization for Neural Networks



Combining nodes:
Network output + L2-loss + regularization

$$\sum_{i=1}^{n} \|w_2 \max(0, w_1 x_i) - y_i\|_2^2 + \lambda R(w_1, w_2)$$

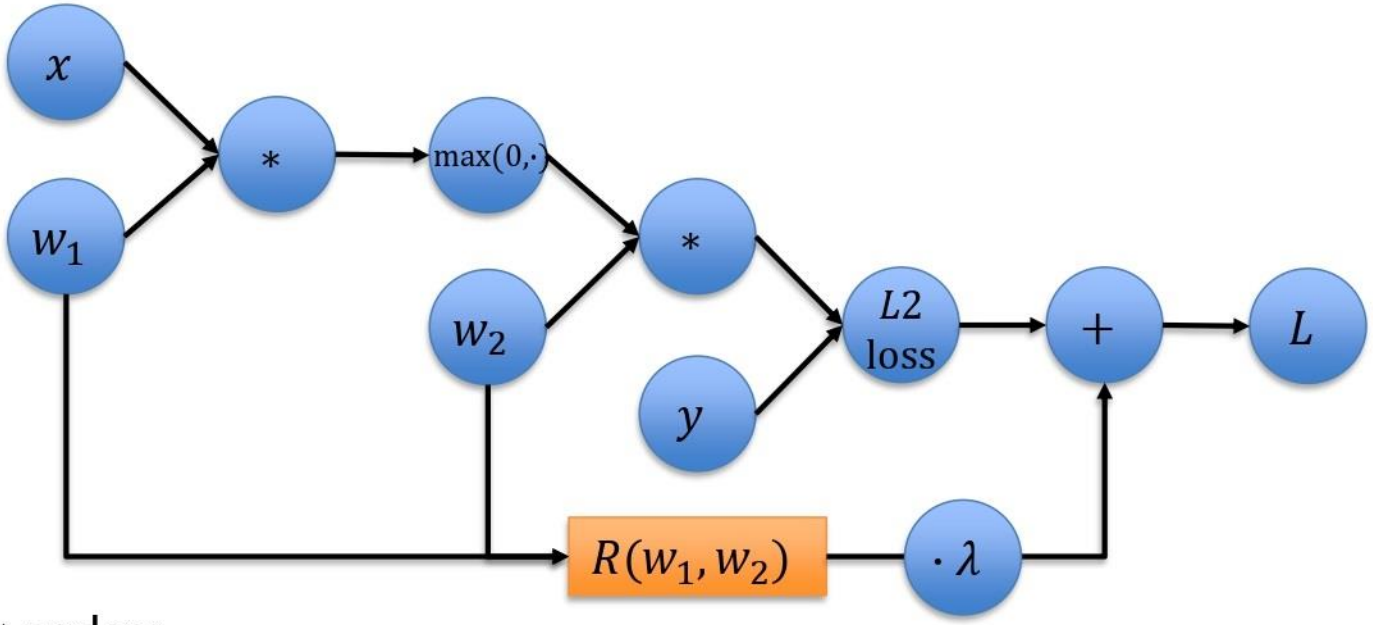
Regularization for Neural Networks



Combining nodes:
Network output + L2-loss + regularization

$$\sum_{i=1}^{n} \|w_2 \max(0, w_1 x_i) - y_i\|_2^2 + \lambda \left\| \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \right\|_2^2$$

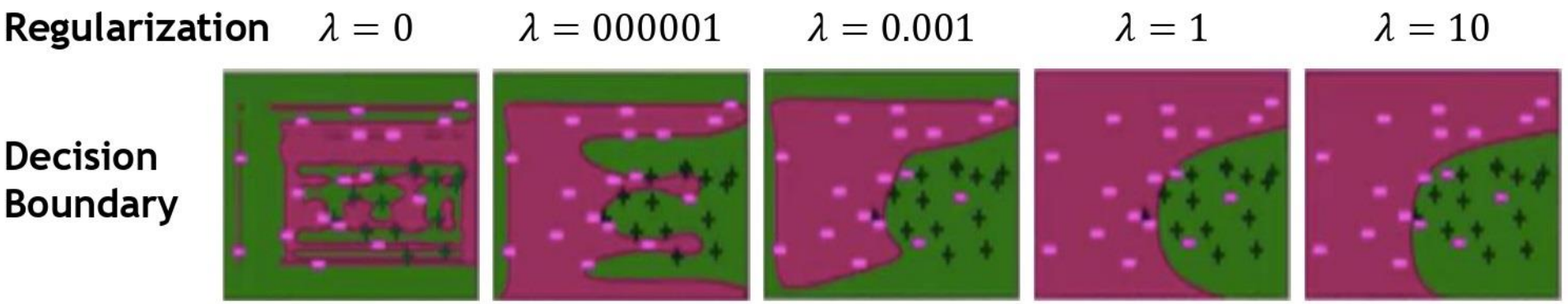
Regularization for Neural Networks



Combining nodes:
Network output + L2-loss + regularization

$$\sum_{i=1}^{n} \|w_2 \max(0, w_1 x_i) - y_i\|_2^2 + \lambda(w_1^2 + w_2^2)$$

Regularization



Credits: University of Washington

What is the goal of regularization?

What happens to the training error?

Regularization

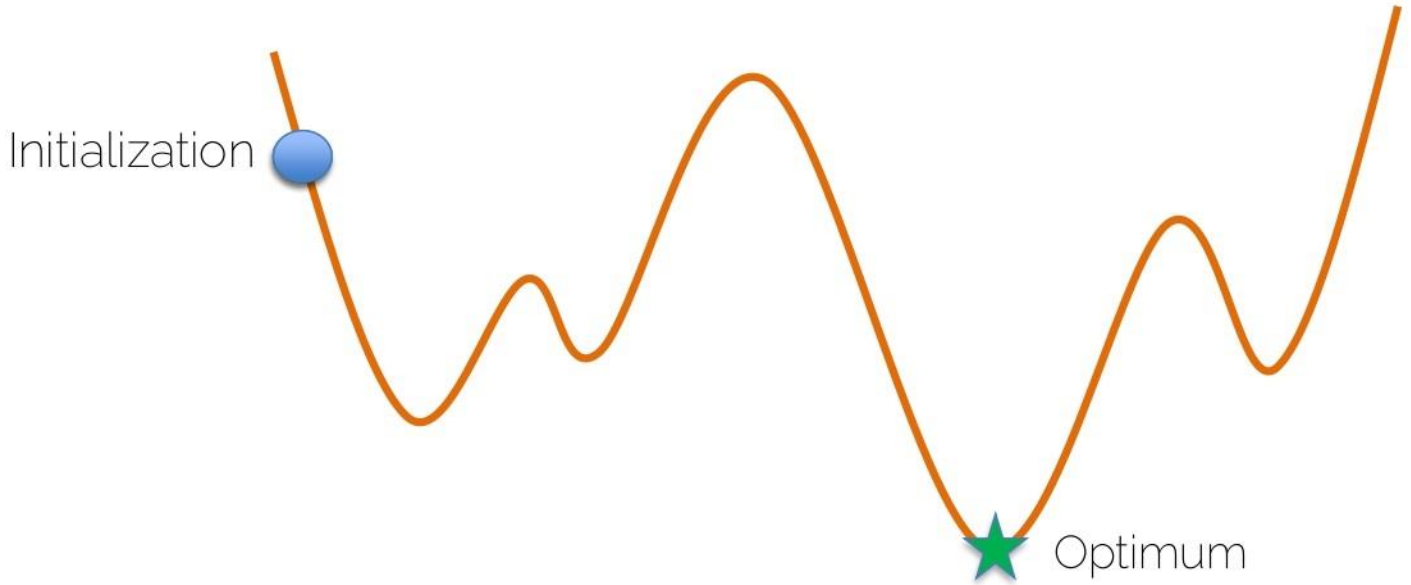
- Any strategy that aims to

Lower validation error

Increasing training error

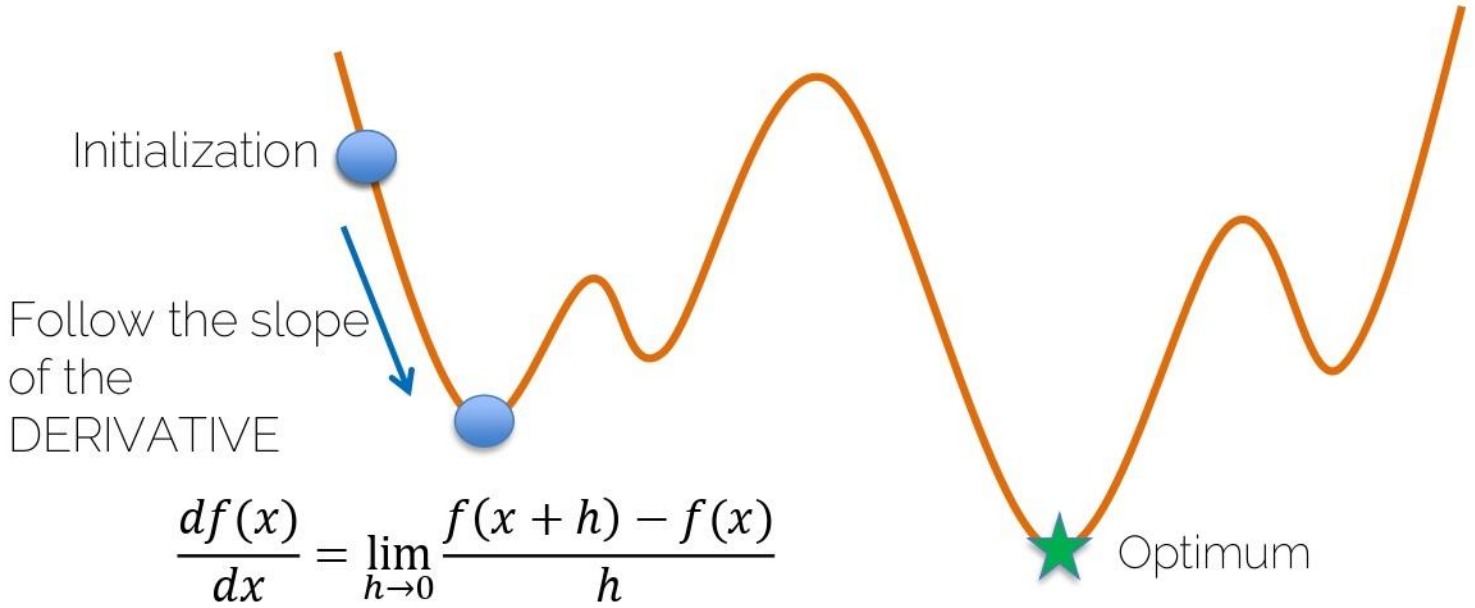
Gradient Descent

$$x^* = \arg\min f(x)$$



Gradient Descent

$$x^* = \arg\min f(x)$$



$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

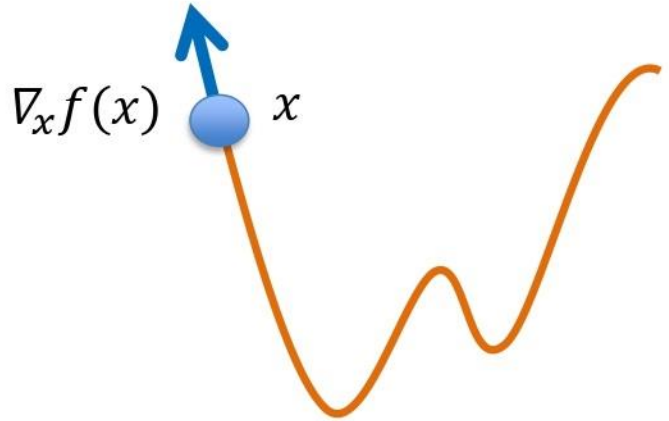
Gradient Descent

- From derivative to gradient

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} \longrightarrow \nabla_x f(x)$$

Direction of greatest increase of the function

- Gradient steps in direction of negative gradient



$$x' = x - \alpha \nabla_x f(x)$$

Learning rate

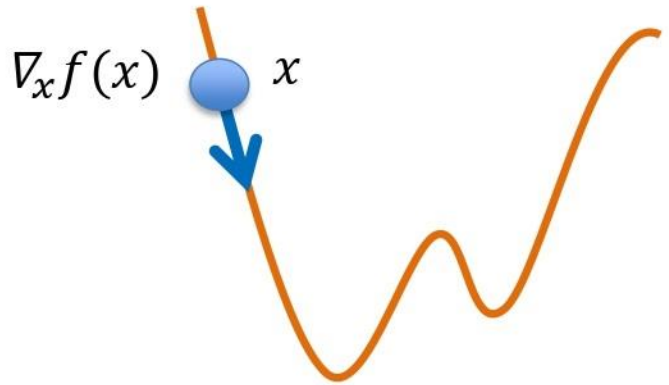
Gradient Descent

- From derivative to gradient

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} \longrightarrow \nabla_x f(x)$$

Direction of greatest increase of the function

- Gradient steps in direction of negative gradient



$$x' = x - \alpha \nabla_x f(x)$$

SMALL Learning rate

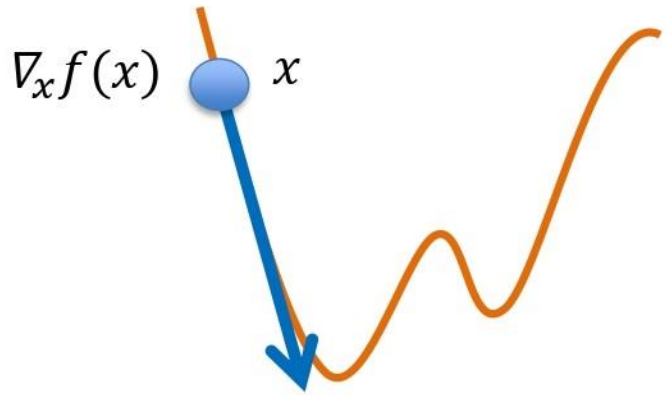
Gradient Descent

- From derivative to gradient

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} \longrightarrow \nabla_x f(x)$$

Direction of greatest increase of the function

- Gradient steps in direction of negative gradient

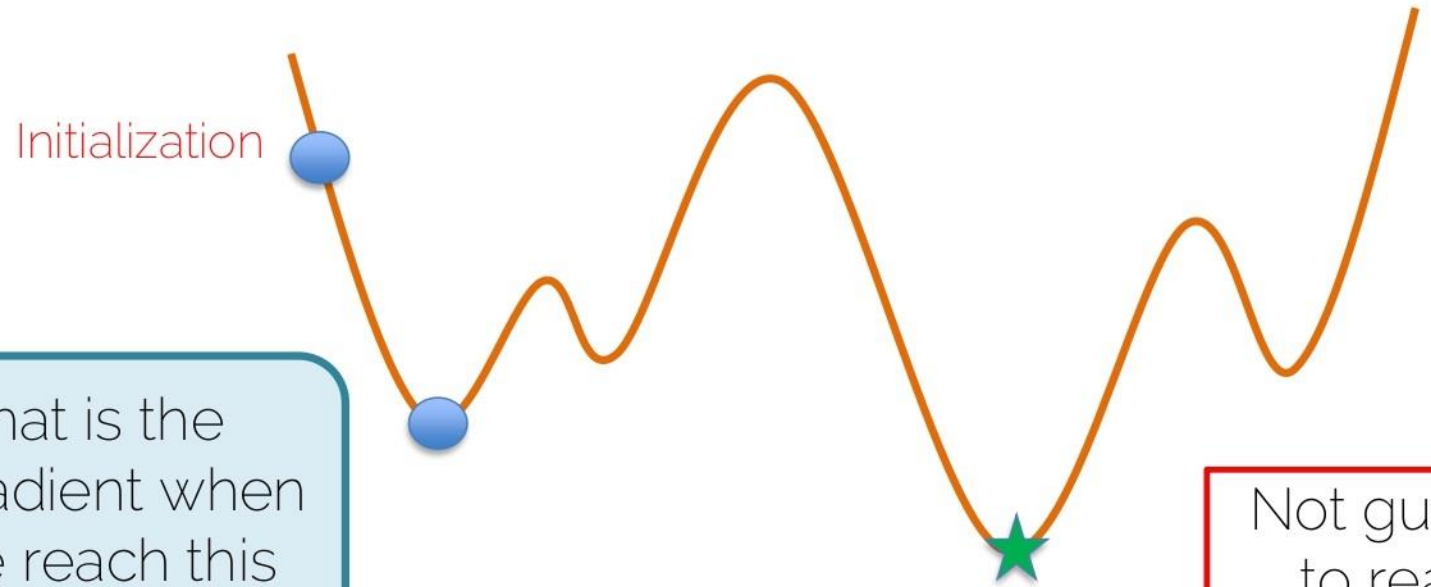


$$x' = x - \alpha \nabla_x f(x)$$

LARGE Learning rate

Gradient Descent

$$\boldsymbol{x}^* = \arg\min f(\boldsymbol{x})$$

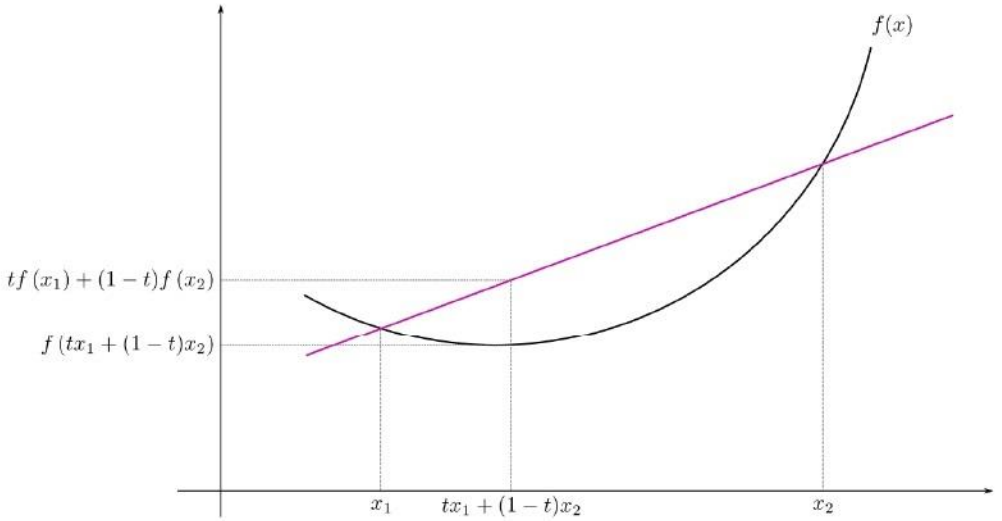


What is the gradient when we reach this point?

Not guaranteed to reach the global optimum

Convergence of Gradient Descent

- Convex function: all local minima are global minima

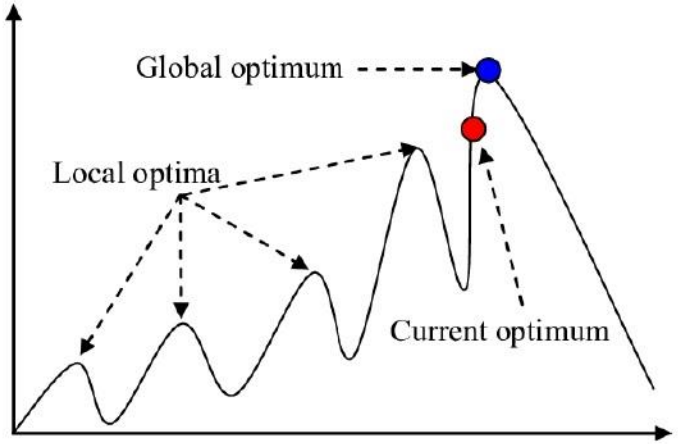


Source: https://en.wikipedia.org/wiki/Convex_function#/media/File:ConvexFunction.svg

If line/plane segment between any two points lies above or on the graph

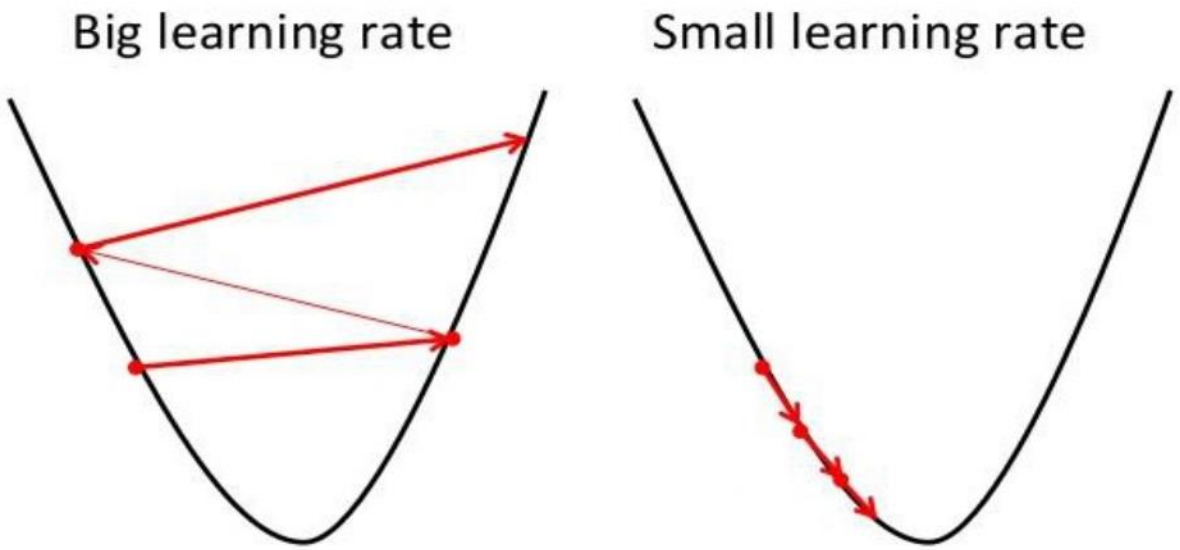
Convergence of Gradient Descent

- Neural networks are non-convex
  - many (different) local minima
  - no (practical) way to say which is globally optimal



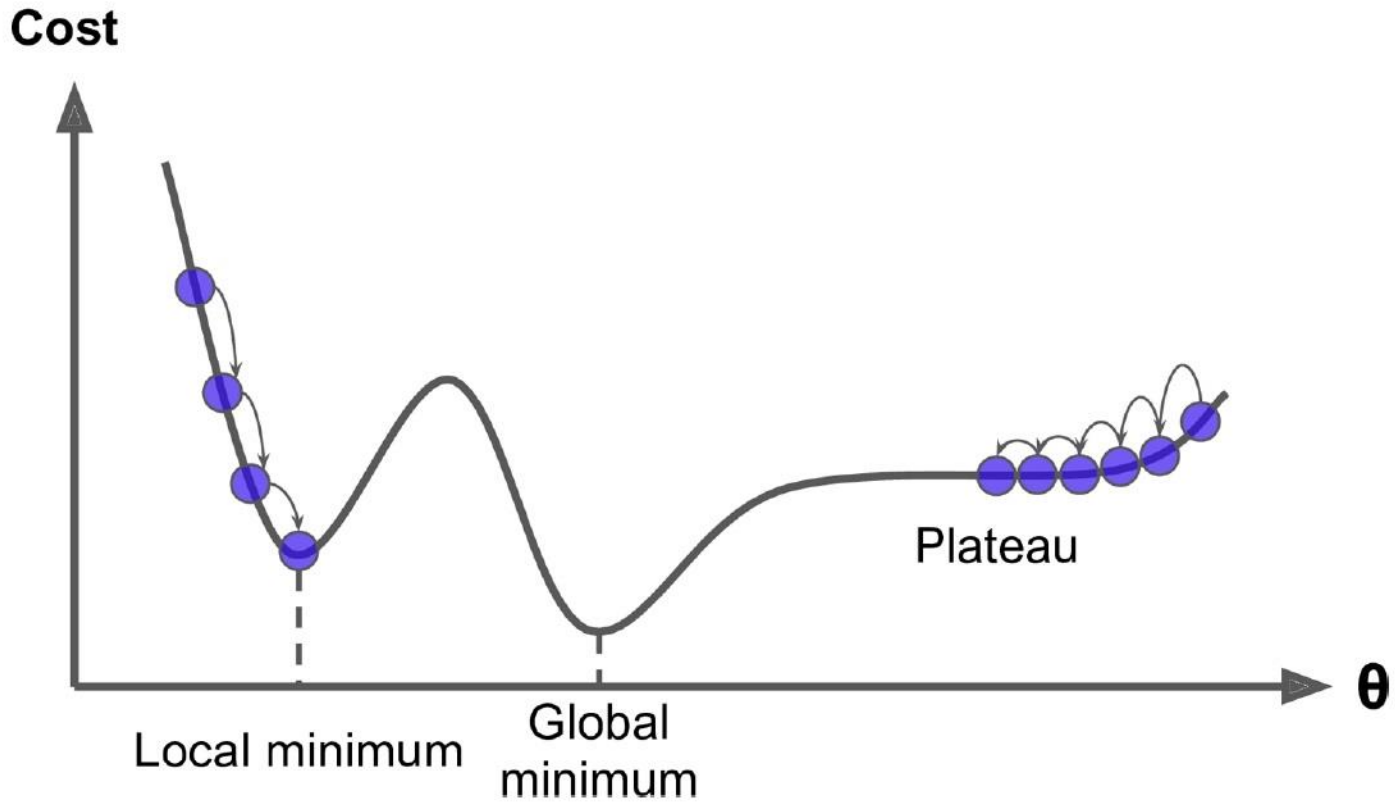
Source: Li, Qi. (2006). Challenging Registration of Geologic Image Data

Convergence of Gradient Descent



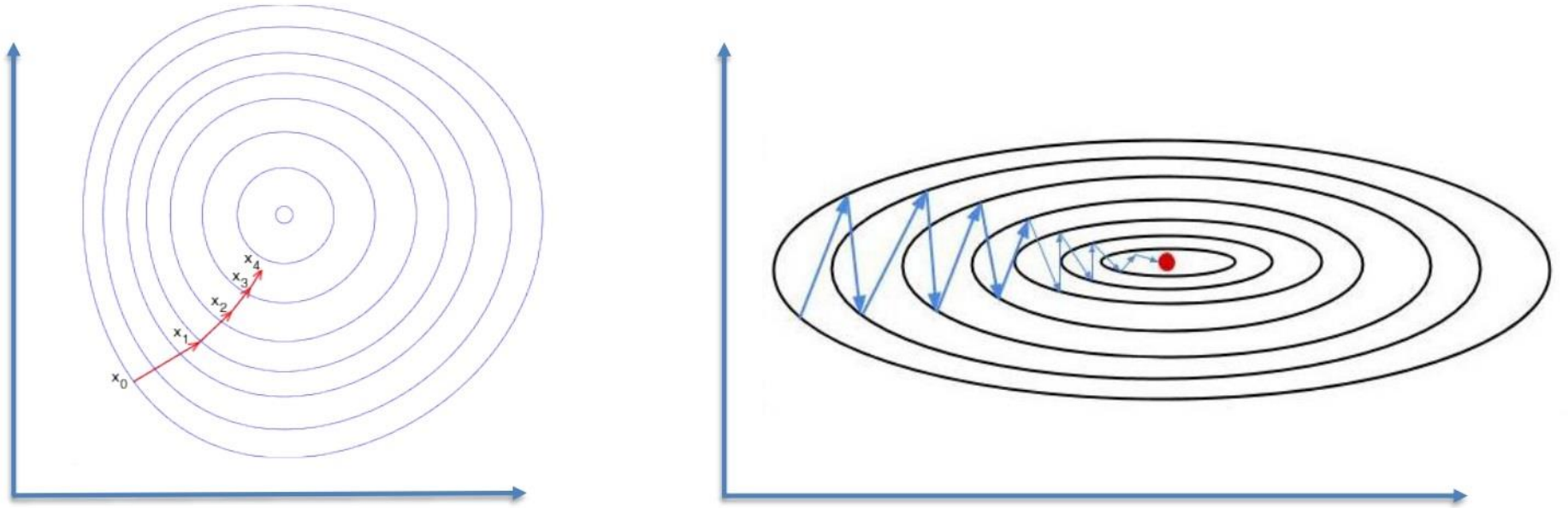
Source: https://builtin.com/data-science/gradient-descent

Convergence of Gradient Descent



Source: A. Geron

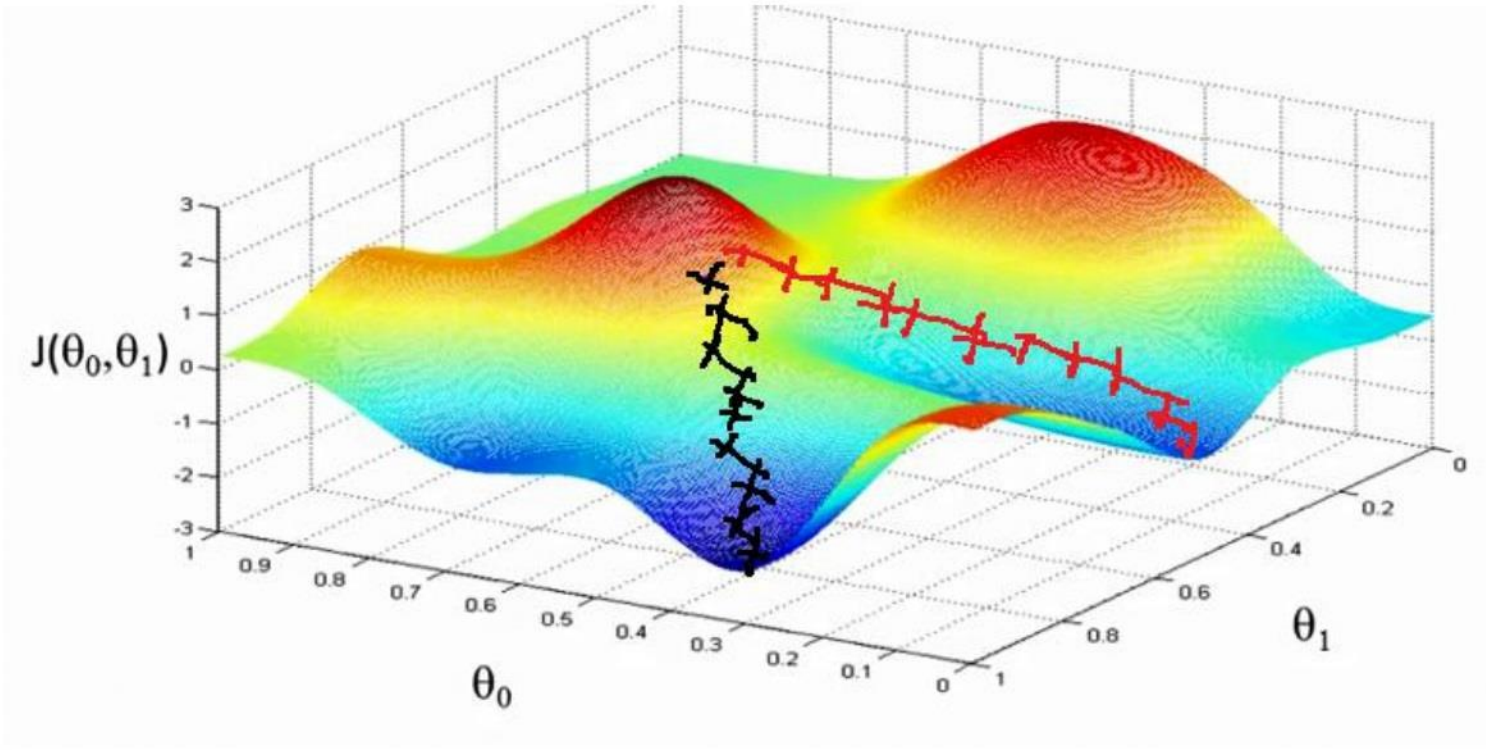
Gradient Descent: Multiple Dimensions



Source: builtin.com/data-science/gradient-descent

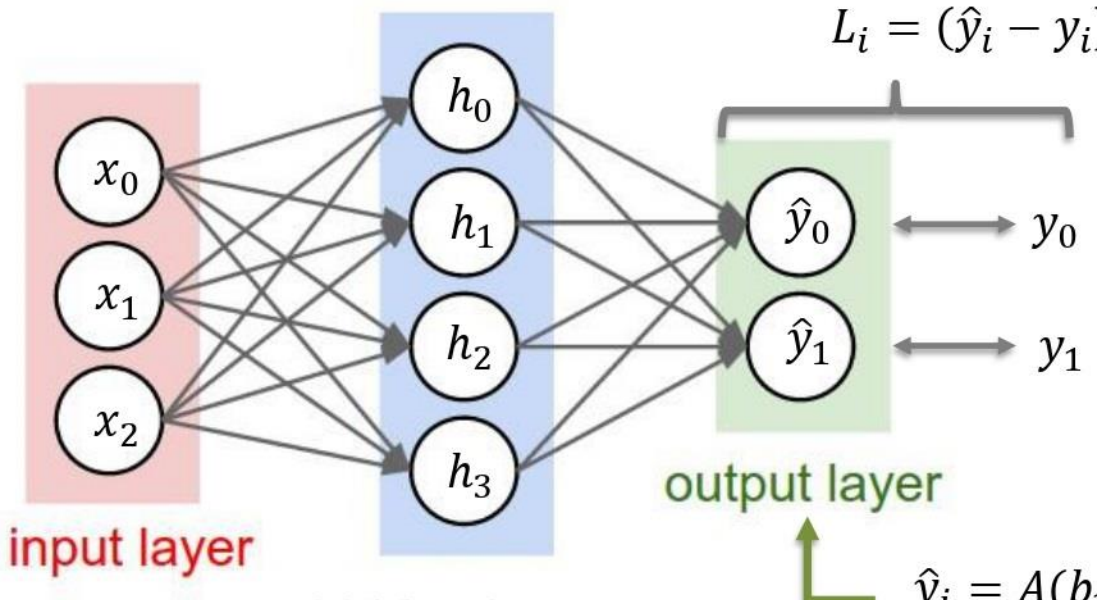
Various ways to visualize…

Gradient Descent: Multiple Dimensions



Source: http://blog.datumbox.com/wp-content/uploads/2013/10/gradient-descent.png

Gradient Descent for Neural Networks



Loss function

$$L_i = (\hat{y}_i - y_i)^2$$

$$\nabla_{\boldsymbol{W},\boldsymbol{b}} f_{\{\boldsymbol{x},\boldsymbol{y}\}}(\boldsymbol{W}) = \begin{bmatrix} \frac{\partial f}{\partial w_{0,0,0}} \\ \cdots \\ \cdots \\ \frac{\partial f}{\partial w_{l,m,n}} \\ \cdots \\ \cdots \\ \frac{\partial f}{\partial b_{l,m}} \end{bmatrix}$$

$$\hat{y}_i = A(b_{1,i} + \sum_j h_j w_{1,i,j})$$

$$h_j = A(b_{0,j} + \sum_k x_k w_{0,j,k})$$

Just simple:

$$A(x) = \max(0, x)$$

Gradient Descent: Single Training Sample

- Given a loss function $\boldsymbol{L}$ and a single training sample $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}$
- Find best model parameters $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}\}$
- Cost $L_i(\boldsymbol{\theta}, \boldsymbol{x}_i, \boldsymbol{y}_i)$
  - $\boldsymbol{\theta} = \arg\min L_i(\boldsymbol{x}_i, \boldsymbol{y}_i)$
- Gradient Descent:
  - Initialize $\boldsymbol{\theta}^1$ with 'random' values (more to that later)
  - $\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha \nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta}^k, \boldsymbol{x}_i, \boldsymbol{y}_i)$
  - Iterate until convergence: $\left|\boldsymbol{\theta}^{k+1} - \boldsymbol{\theta}^k\right| < \epsilon$

Gradient Descent: Single Training Sample

- $\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} - \alpha \nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta}^{k}, \boldsymbol{x}_i, \boldsymbol{y}_i)$

- $\nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta}^{k}, \boldsymbol{x}_i, \boldsymbol{y}_i)$ computed via backpropagation
- Typically: $\dim\left(\nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta}^{k}, \boldsymbol{x}_i, \boldsymbol{y}_i)\right) = \dim(\boldsymbol{\theta}) \gg 1\,million$

Gradient Descent: Multiple Training Samples

- Given a loss function $\boldsymbol{L}$ and multiple ($\boldsymbol{n}$) training samples $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}$
- Find best model parameters $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}\}$

- Cost $L = \frac{1}{n}\sum_{i=1}^{n} L_i(\boldsymbol{\theta}, \boldsymbol{x}_i, \boldsymbol{y}_i)$
  - $\boldsymbol{\theta} = \arg\min L$

Gradient Descent: Multiple Training Samples

- Update step for multiple samples

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} - \alpha \nabla_{\boldsymbol{\theta}} L\left(\boldsymbol{\theta}^{k}, \boldsymbol{x}_{\{1..n\}}, \boldsymbol{y}_{\{1..n\}}\right)$$

- Gradient is average / sum over residuals

$$\nabla_{\boldsymbol{\theta}} L\left(\boldsymbol{\theta}^{k}, \boldsymbol{x}_{\{1..n\}}, \boldsymbol{y}_{\{1..n\}}\right) = \frac{1}{n} \sum_{i=1}^{n} \underbrace{\nabla_{\boldsymbol{\theta}} L_i\left(\boldsymbol{\theta}^{k}, \boldsymbol{x}_i, \boldsymbol{y}_i\right)}$$

Reminder: this comes from backprop.

- Often people are lazy and just write: $\nabla L = \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L_i$
  - omitting $\frac{1}{n}$ is not 'wrong', it just means rescaling the learning rate

Gradient Descent on Train Set

- Given large train set with $\boldsymbol{n}$ training samples $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}$
  - Let's say 1 million labeled images
  - Let's say our network has 500k parameters

- Gradient has 500k dimensions
- $\boldsymbol{n = 1\,million}$

→ Extremely expensive to compute

Stochastic Gradient Descent (SGD)

- If we have $\boldsymbol{n}$ training samples we need to compute the gradient for all of them which is $\boldsymbol{O(n)}$

- If we consider the problem as empirical risk minimization, we can express the total loss over the training data as the expectation of all the samples

$$\frac{1}{n}\left(\sum_{i=1}^{n} L_i(\boldsymbol{\theta}, \boldsymbol{x_i}, \boldsymbol{y_i})\right) = \mathbb{E}_{i\sim[1,\ldots,n]}[L_i(\boldsymbol{\theta}, \boldsymbol{x_i}, \boldsymbol{y_i})]$$

Stochastic Gradient Descent (SGD)

- The expectation can be approximated with a small subset of the data

$$\mathbb{E}_{i\sim[1,\dots,n]}[L_i(\boldsymbol{\theta}, \boldsymbol{x_i}, \boldsymbol{y_i})] \approx \frac{1}{|S|}\sum_{j\in S}\left(L_j(\boldsymbol{\theta}, \boldsymbol{x_j}, \boldsymbol{y_j})\right) \text{ with S} \subseteq \{1, \dots, n\}$$

Minibatch
choose subset of trainset $\boldsymbol{m} \ll \boldsymbol{n}$

$$B_i = \{\{\boldsymbol{x_1}, \boldsymbol{y_1}\}, \{\boldsymbol{x_2}, \boldsymbol{y_2}\}, \dots, \{\boldsymbol{x_m}, \boldsymbol{y_m}\}\}$$
$$\{B_1, B_2, \dots, B_{n/m}\}$$

Stochastic Gradient Descent (SGD)

- Minibatch size is hyperparameter
  - Typically power of 2 → 8, 16, 32, 64, 128…
  - Smaller batch size means greater variance in the gradients

    → noisy updates
  - Mostly limited by GPU memory (in backward pass)
  - E.g.,
    - Train set has $\mathbf{n = 2^{20}}$ (about 1 million) images
    - With batch size $\mathbf{m = 64}$: $B_{1 \ldots n/m} = B_{1 \ldots 16{,}384}$ minibatches

(Epoch = complete pass through training set)

Stochastic Gradient Descent (SGD)

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{k}, \boldsymbol{x}_{\{1..m\}}, \boldsymbol{y}_{\{1..m\}})$$

k now refers to k-th iteration

$$\nabla_{\boldsymbol{\theta}} L = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L_i$$

m training samples in the current minibatch

Gradient for the k-th minibatch

Note the terminology: iteration vs epoch

Convergence of SGD

Suppose we want to minimize the function $F(\theta)$ with the stochastic approximation

$$\theta^{k+1} = \theta^k - \alpha_k H(\theta^k, X)$$

where $\alpha_1, \alpha_2 \ldots \alpha_n$ is a sequence of positive step-sizes and $H(\theta^k, X)$ is the unbiased estimate of $\nabla \mathrm{F}(\theta^k)$, i.e.

$$\mathbb{E}[H(\theta^k, X)] = \nabla \mathrm{F}(\theta^k)$$

Robbins, H. and Monro, S. "A Stochastic Approximation Method" 1951.

Convergence of SGD

$$\theta^{k+1} = \theta^k - \alpha_k H(\theta^k, X)$$

converges to a local (global) minimum if the following conditions are met:

1) $\alpha_n \geq 0, \forall\, n \geq 0$
2) $\sum_{n=1}^{\infty} \alpha_n = \infty$
3) $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$
4) $F(\theta)$ is strictly convex

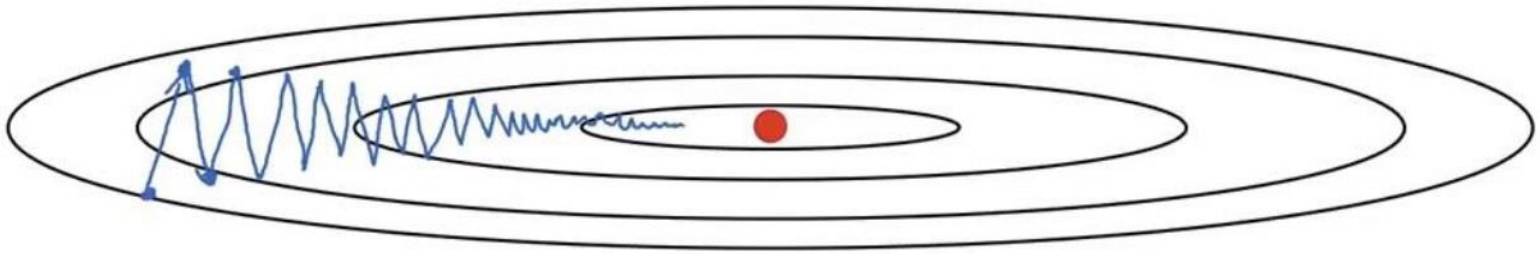
The proposed sequence by Robbins and Monro is $\alpha_n \propto \frac{\alpha}{n}, for\ n > 0$

Problems of SGD

- Gradient is scaled equally across all dimensions
  → i.e., cannot independently scale directions
  → need to have conservative min learning rate to avoid divergence
  → Slower than 'necessary'

- Finding good learning rate is an art by itself
  → More next lecture

Gradient Descent with Momentum



Source: A. Ng

We're making many steps back and forth along this dimension. Would love to track that this is averaging out over time.

Would love to go faster here…
I.e., accumulated gradients over time

Gradient Descent with Momentum

$$\boldsymbol{v}^{k+1} = \beta \cdot \boldsymbol{v}^{k} - \alpha \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{k})$$

accumulation rate ('friction', momentum)

velocity

learning rate

Gradient of current minibatch

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} + \boldsymbol{v}^{k+1}$$

weights of model

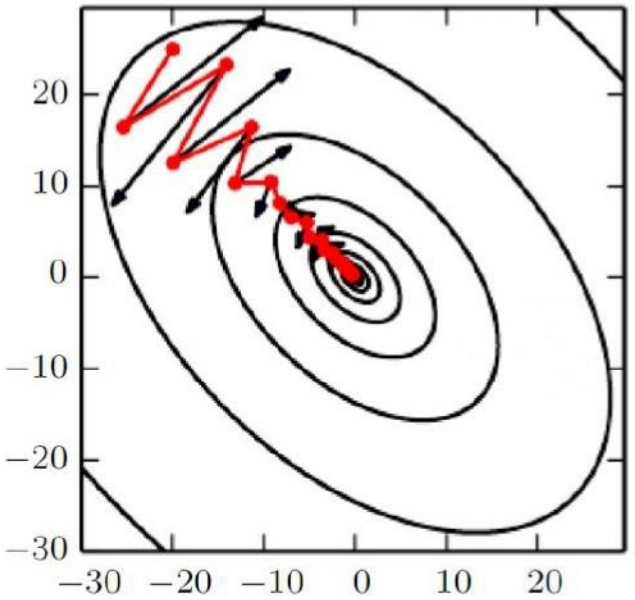
velocity

Exponentially-weighted average of gradient

Important: velocity $\boldsymbol{v}^{k}$ is vector-valued!

[Sutskever et al., ICML'13] On the importance of initialization and momentum in deep learning

Gradient Descent with Momentum



Source: I. Goodfellow

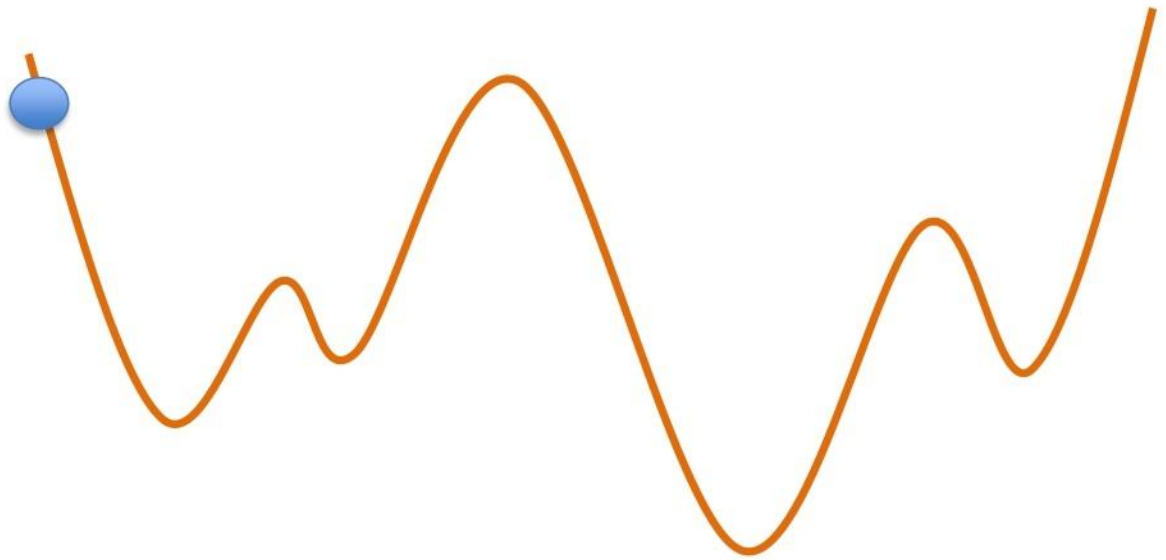
Step will be largest when a sequence of gradients all point to the same direction

Hyperparameters are α, β
β is often set to 0.9

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} + \boldsymbol{v}^{k+1}$$

Gradient Descent with Momentum

- Can it overcome local minima?



$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} + \boldsymbol{v}^{k+1}$$

Nesterov Momentum

- Look-ahead momentum

$$\widetilde{\boldsymbol{\theta}}^{k+1} = \boldsymbol{\theta}^{k} + \beta \cdot \boldsymbol{v}^{k}$$

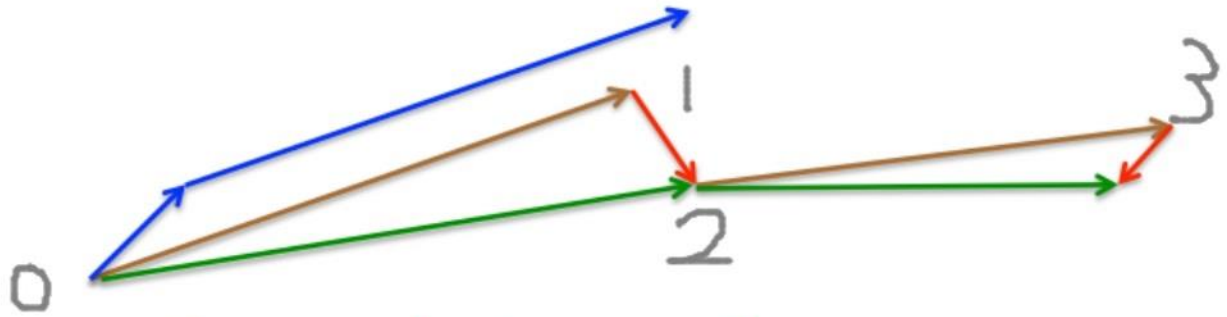
$$\boldsymbol{v}^{k+1} = \beta \cdot \boldsymbol{v}^{k} - \alpha \cdot \nabla_{\boldsymbol{\theta}} L(\widetilde{\boldsymbol{\theta}}^{k+1})$$

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} + \boldsymbol{v}^{k+1}$$

Nesterov, Yurii E. "A method for solving the convex programming problem with convergence rate O (1/k^ 2)." *Dokl. akad. nauk Sssr*. Vol. 269. 1983.

Nesterov Momentum

- First make a big jump in the direction of the previous accumulated gradient.
- Then measure the gradient where you end up and make a correction.



brown vector = jump, red vector = correction, green vector = accumulated gradient

blue vectors = standard momentum

Source: G. Hinton

$$\widetilde{\boldsymbol{\theta}}^{k+1} = \boldsymbol{\theta}^{k} + \beta \cdot \boldsymbol{v}^{k}$$
$$\boldsymbol{v}^{k+1} = \beta \cdot \boldsymbol{v}^{k} - \alpha \cdot \nabla_{\boldsymbol{\theta}} L(\widetilde{\boldsymbol{\theta}}^{k+1})$$
$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} + \boldsymbol{v}^{k+1}$$