Lecture 3 recap

Exercise 1: Loss cheat sheet

- Softmax loss or cross-entropy loss

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right)$$

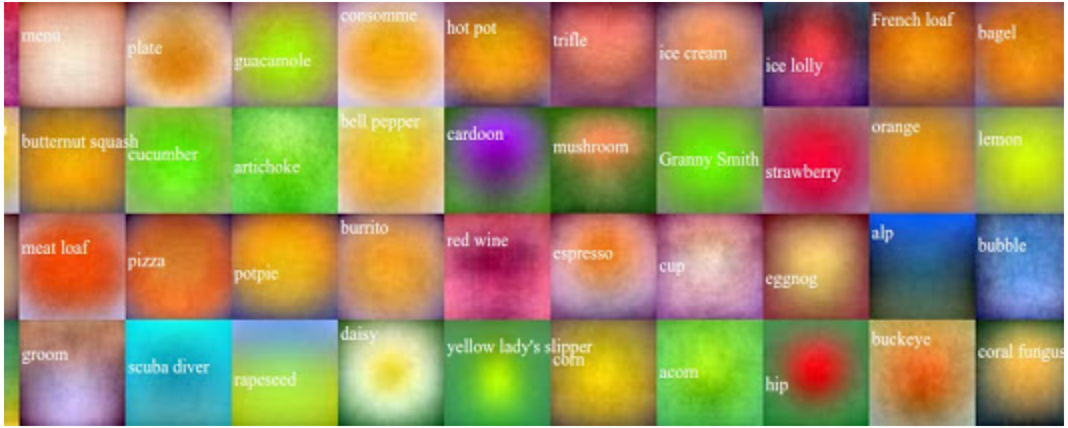
Score of the ground truth class

Scores or predictions

Beyond linear

- Linear score function $f = Wx$

On CIFAR-10

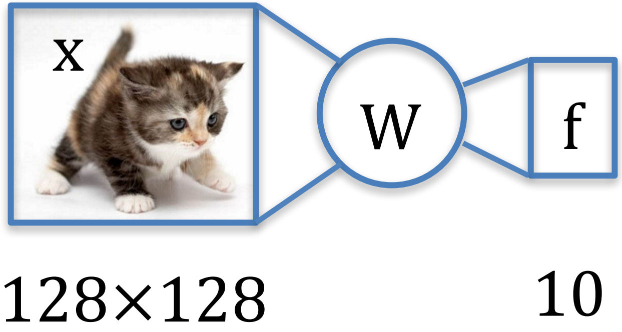


On ImageNet

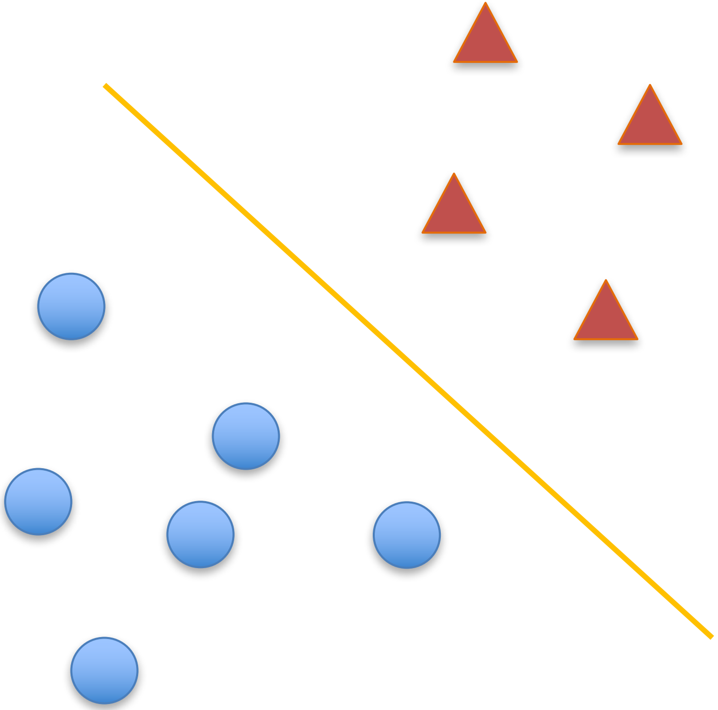
Credit: Li/Karpathy/Johnson

Beyond linear

1-layer network: $f = \mathbf{W}\mathbf{x}$

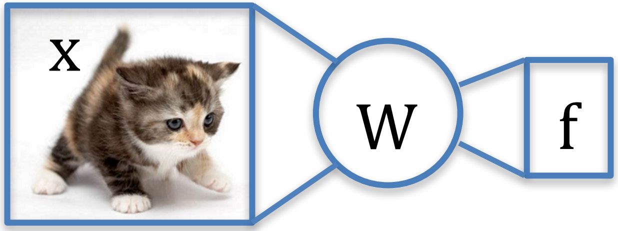


LINEAR TRANSFORMATION



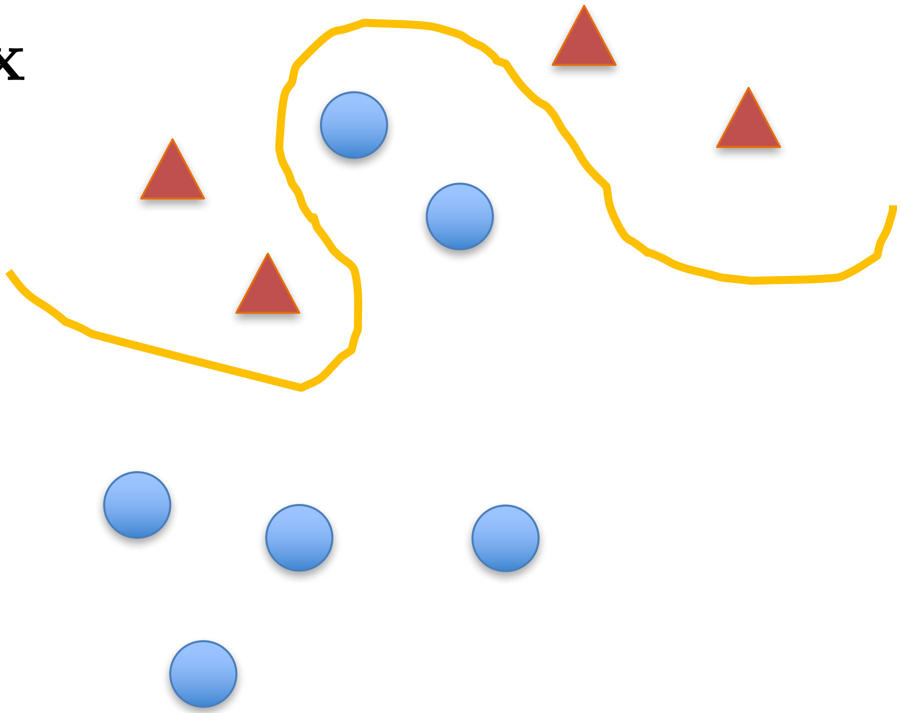
Beyond linear

1-layer network: $f = \mathbf{W}\mathbf{x}$



128×128 10

LINEAR TRANSFORMATION

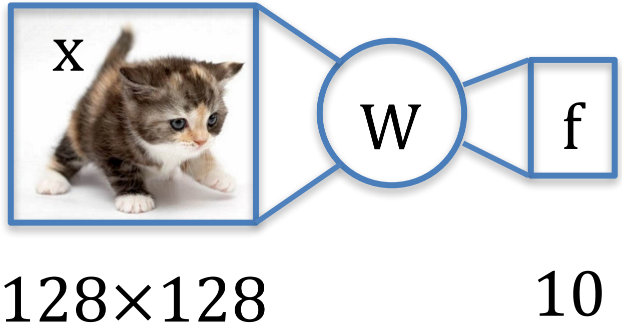


Kernel trick

1-layer network: $f = \mathbf{W}\mathbf{x}$

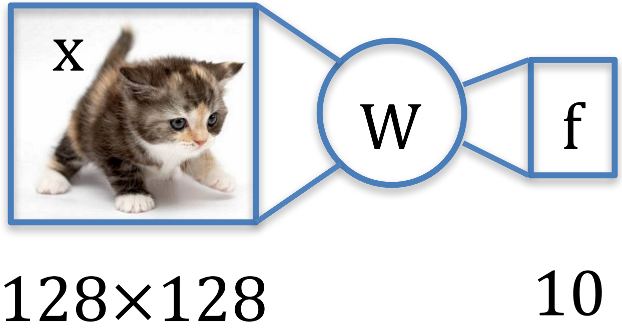
$$f = \mathbf{W}\phi(\mathbf{x})$$

kernel



Neural networks

1-layer network: $f = \mathbf{W}\mathbf{x}$



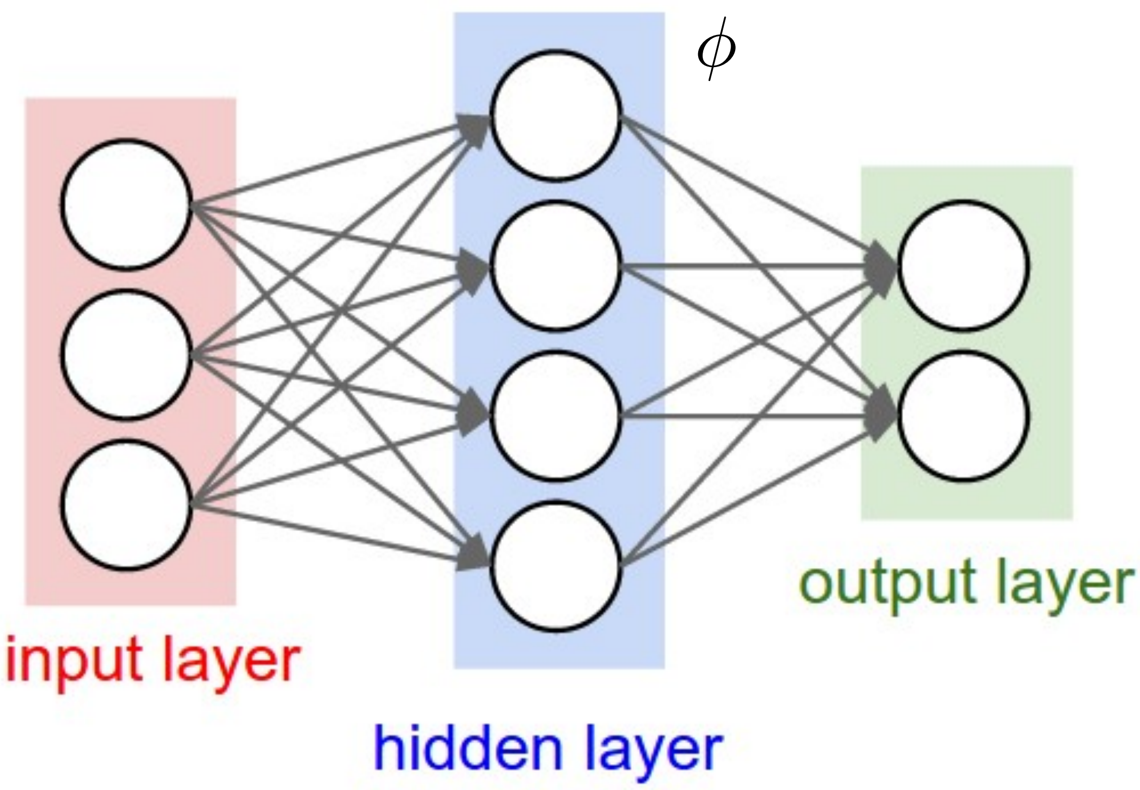
$$f = \mathbf{W}\phi(\mathbf{x};\boldsymbol{\theta})$$

kernel

parameters

From the broad family of functions ϕ we learn the best representation by learning the parameters $\boldsymbol{\theta}$

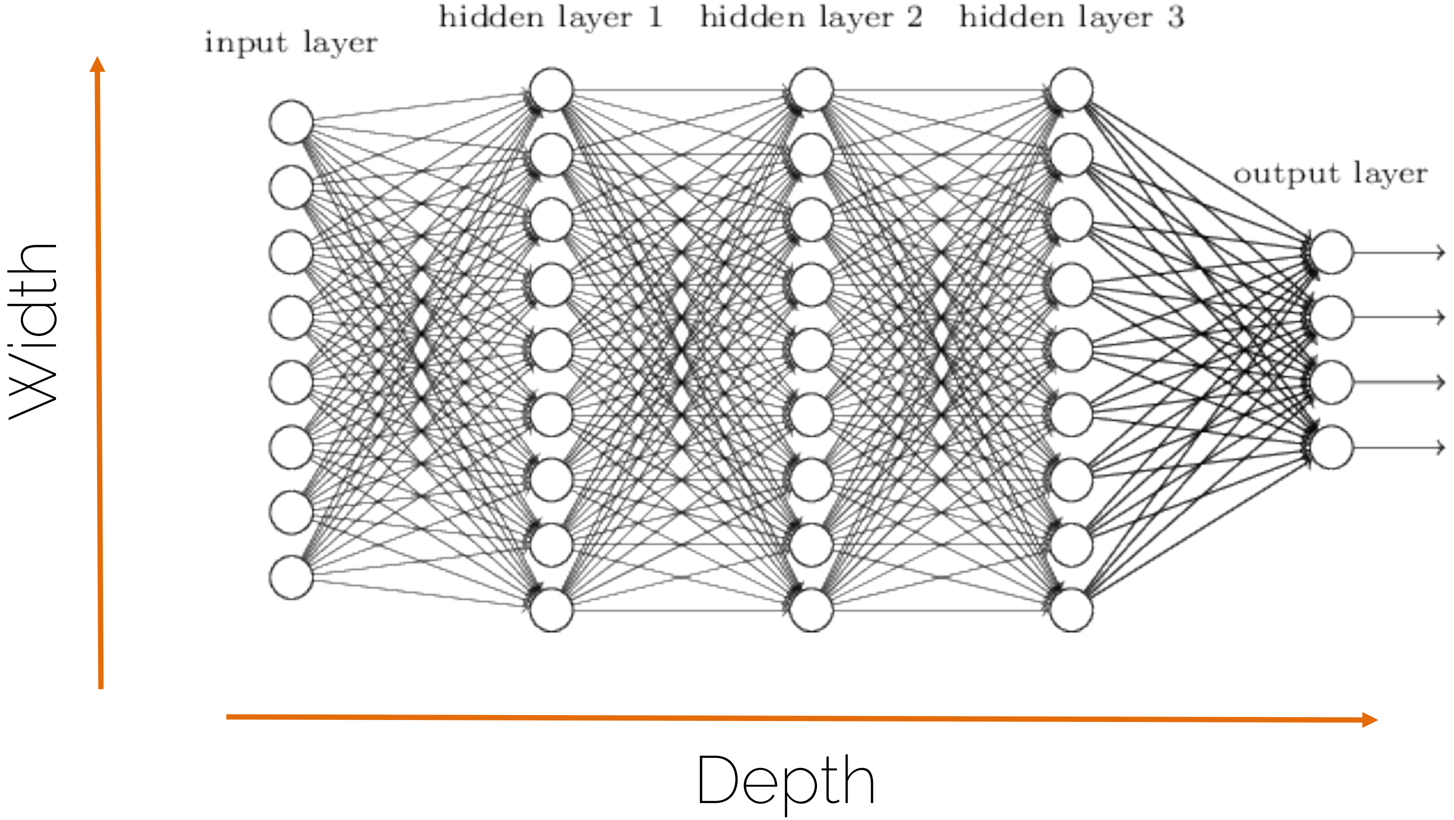
Neural Network



Also SVM is in this category

Credit: Li/Karpathy/Johnson

Neural Network



Neural Network

- Linear score function $f = Wx$

- Neural network is a nesting of 'functions'
  - 2-layers: $f = W_2 \max(0, W_1 x)$
  - 3-layers: $f = W_3 \max(0, W_2 \max(0, W_1 x))$
  - 4-layers: $f = W_4 \tanh\,(\mathrm{W}_3, \max(0, W_2 \max(0, W_1 x)))$
  - 5-layers: $f = W_5 \sigma(W_4 \tanh(\mathrm{W}_3, \max(0, W_2 \max(0, W_1 x))))$
  - … up to hundreds of layers

Computational Graphs

- Neural network is a computational graph
  - It has compute nodes
  - It has edges that connect nodes
  - It is directional
  - It is organized in 'layers'

Backprop

The importance of gradients

- All optimization schemes are based on computing gradients

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$$

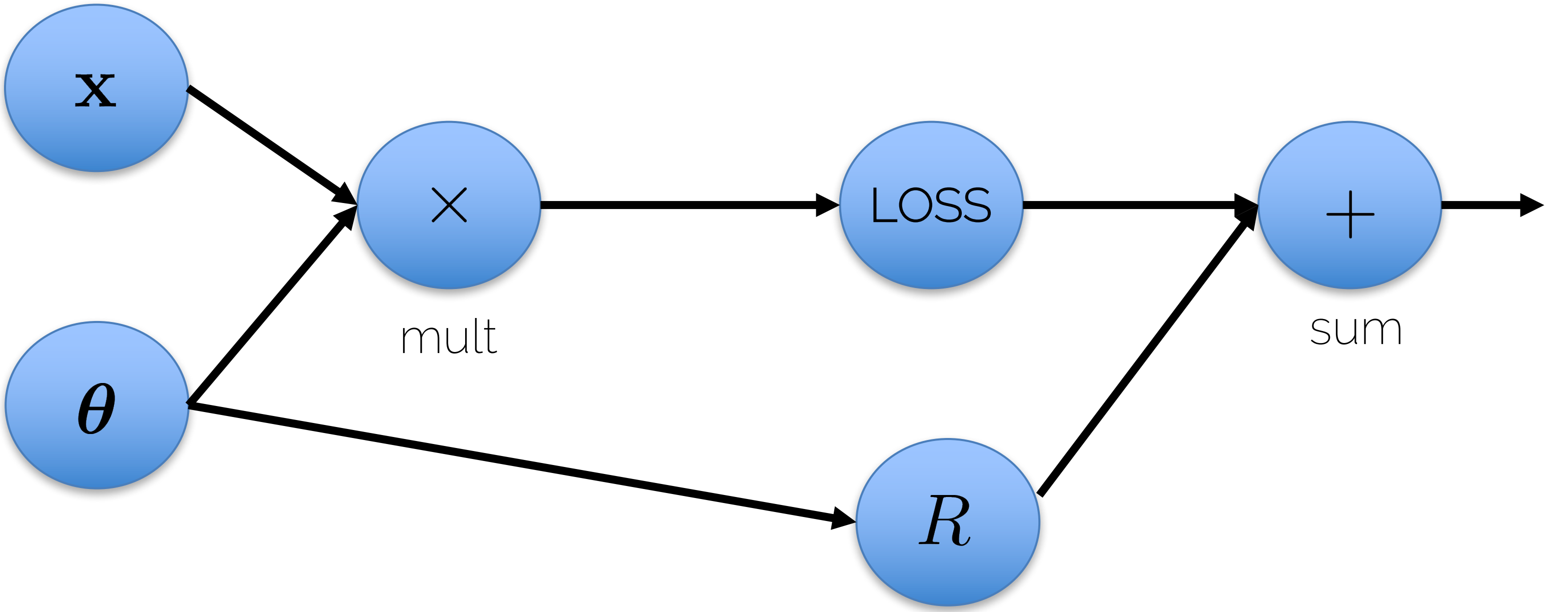
- One can compute gradients analytically but what if our function is too complex?

- Break down gradient computation $\boxed{\text{Backpropagation}}$

Rumelhart 1986

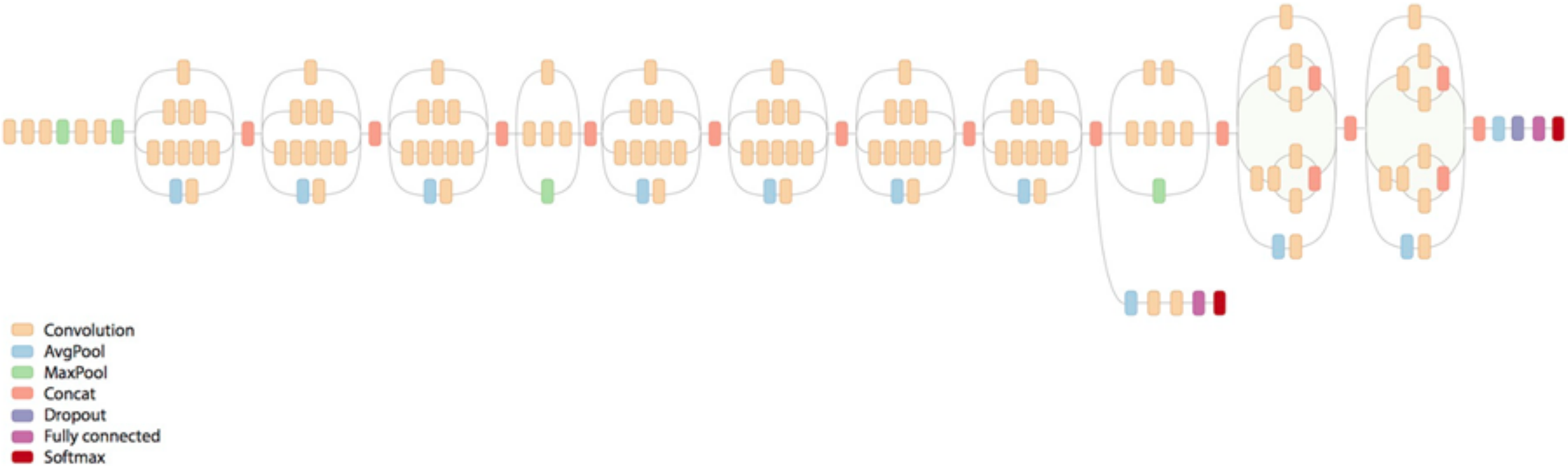
Computational graphs

$$J(\boldsymbol{\theta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) + \lambda R(\boldsymbol{\theta})$$



Computational graphs

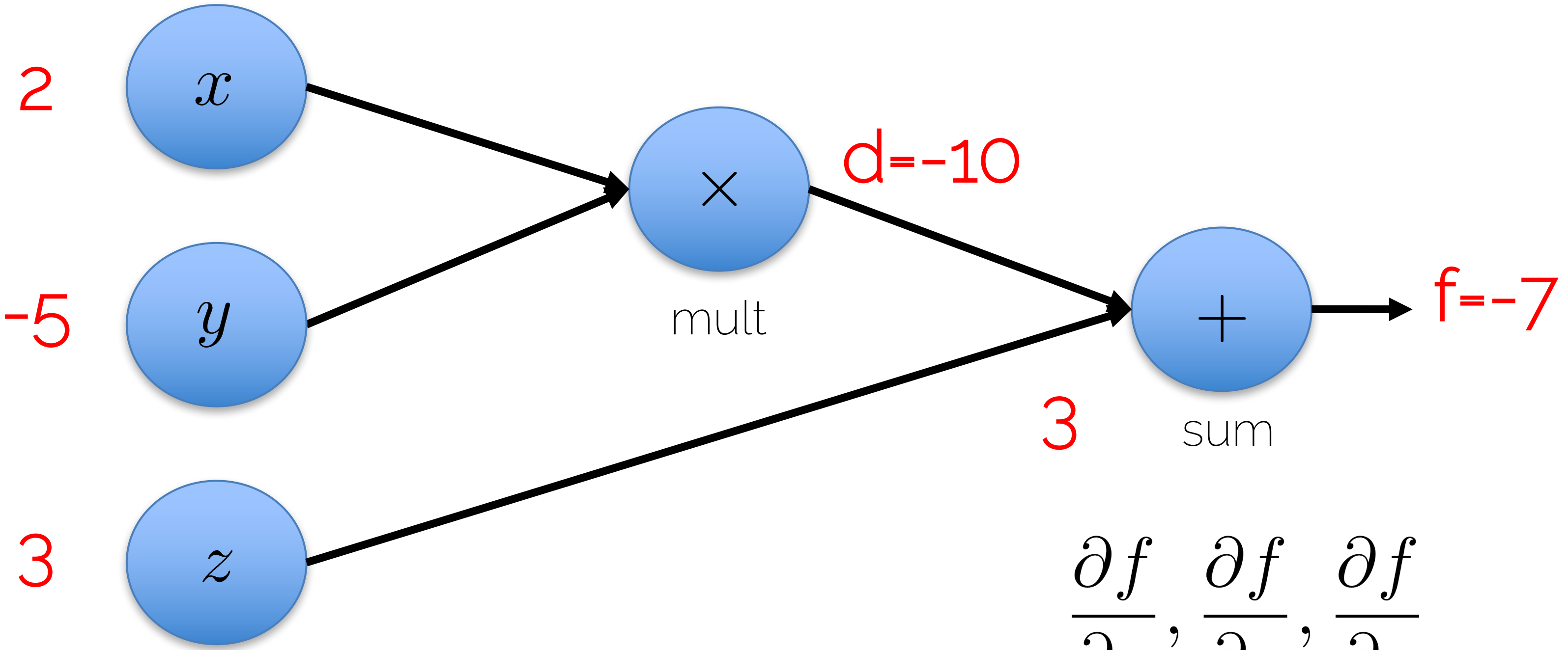
- These graphs can be huge!



Another view of GoogLeNet's architecture.

An example: forward pass

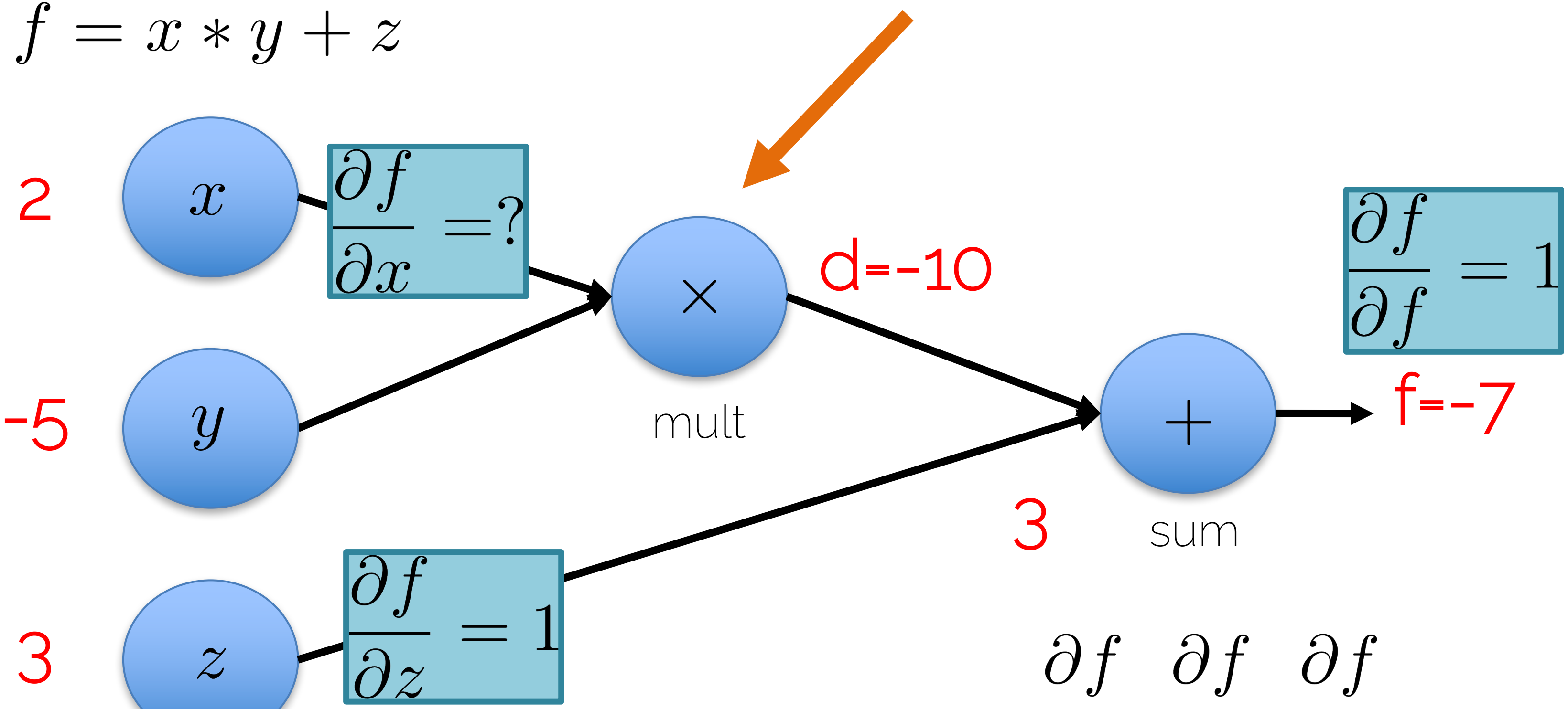
$f = x * y + z$ Initialization $x = 2, y = -5, z = 3$



$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

An example: backward pass

$$f = x * y + z$$

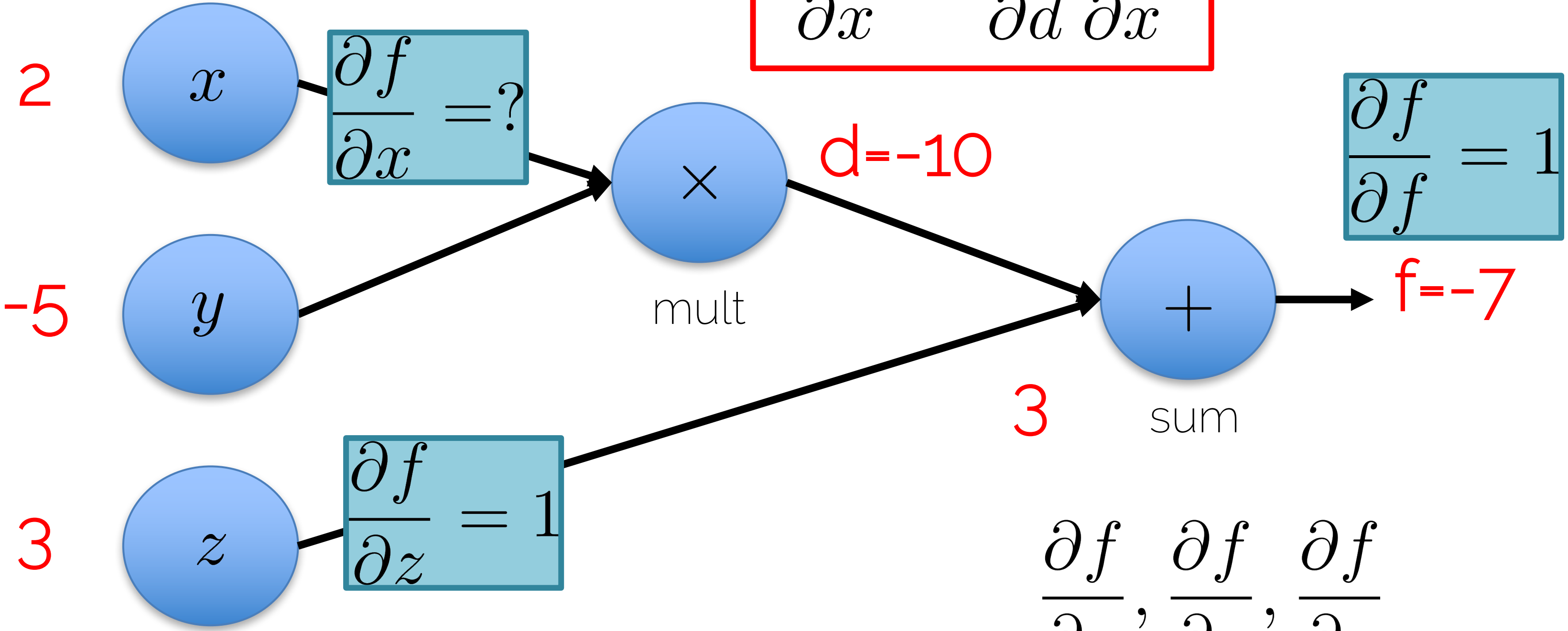


$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

An example: chain rule

$$f = x * y + z$$

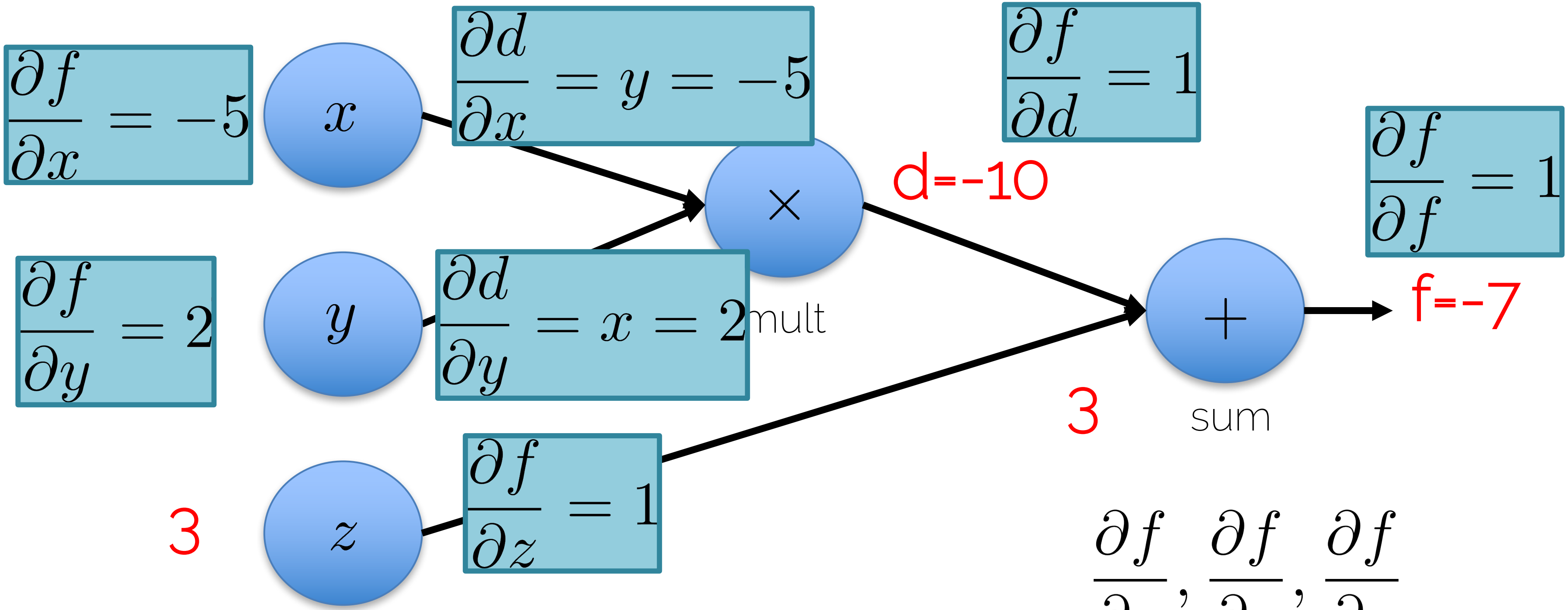
$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial d}\frac{\partial d}{\partial x}}$$



$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

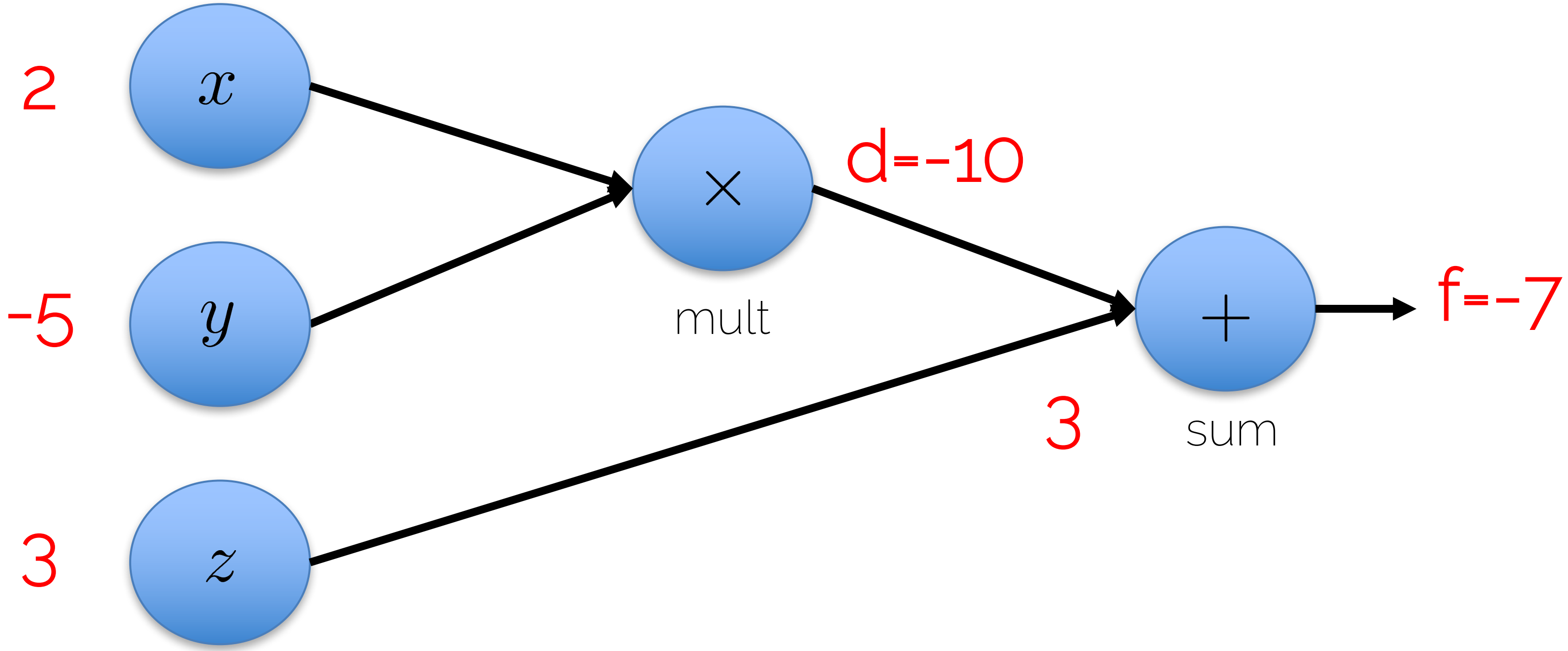
An example: chain rule $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial d}\frac{\partial d}{\partial x}$

$$f = x * y + z$$



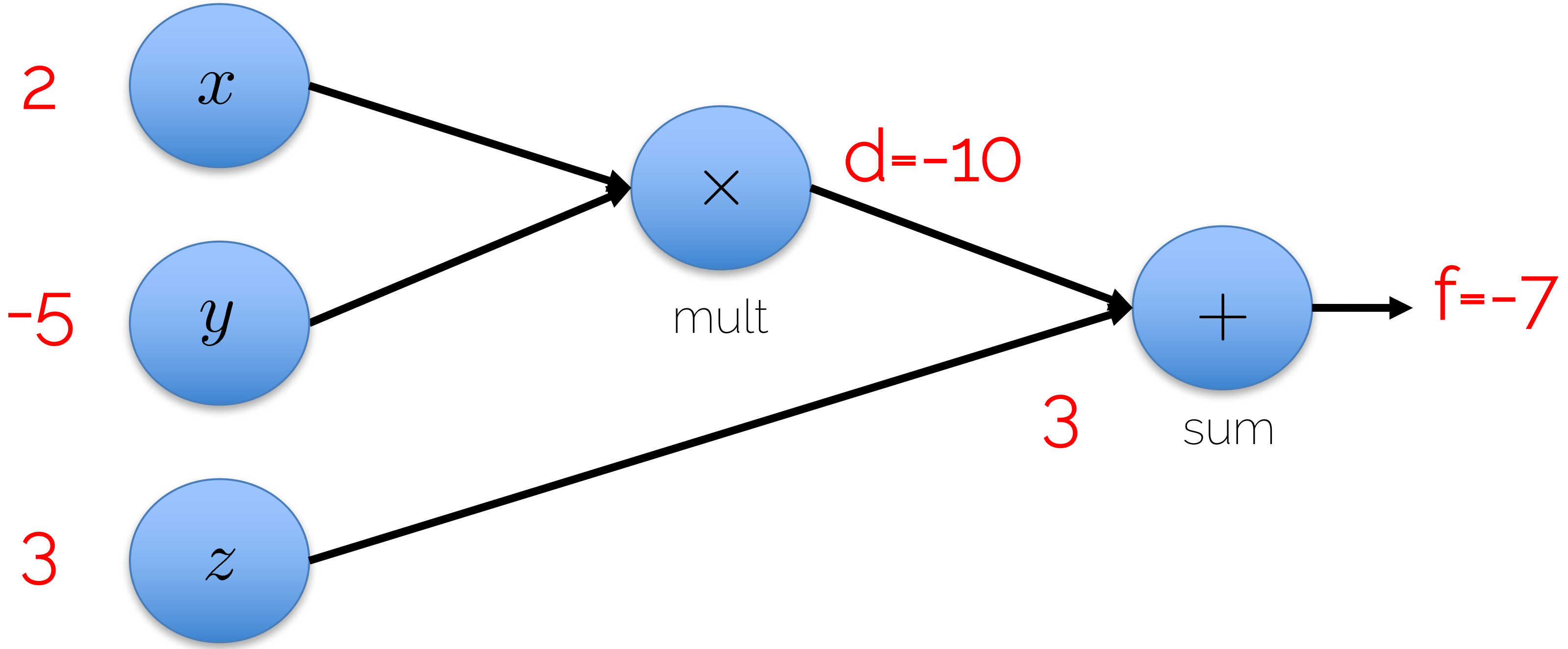
$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

An example: the chain rule



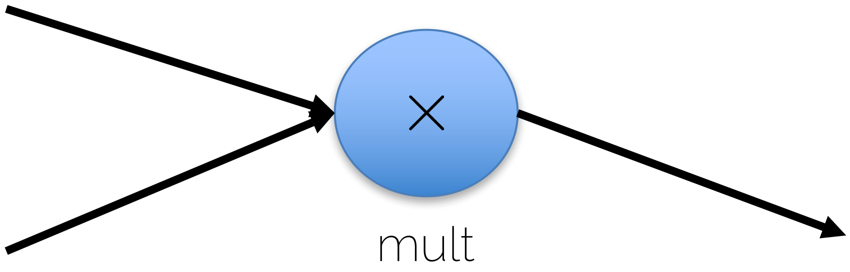
An example: the chain rule

- Each node is only interested in its own inputs and outputs

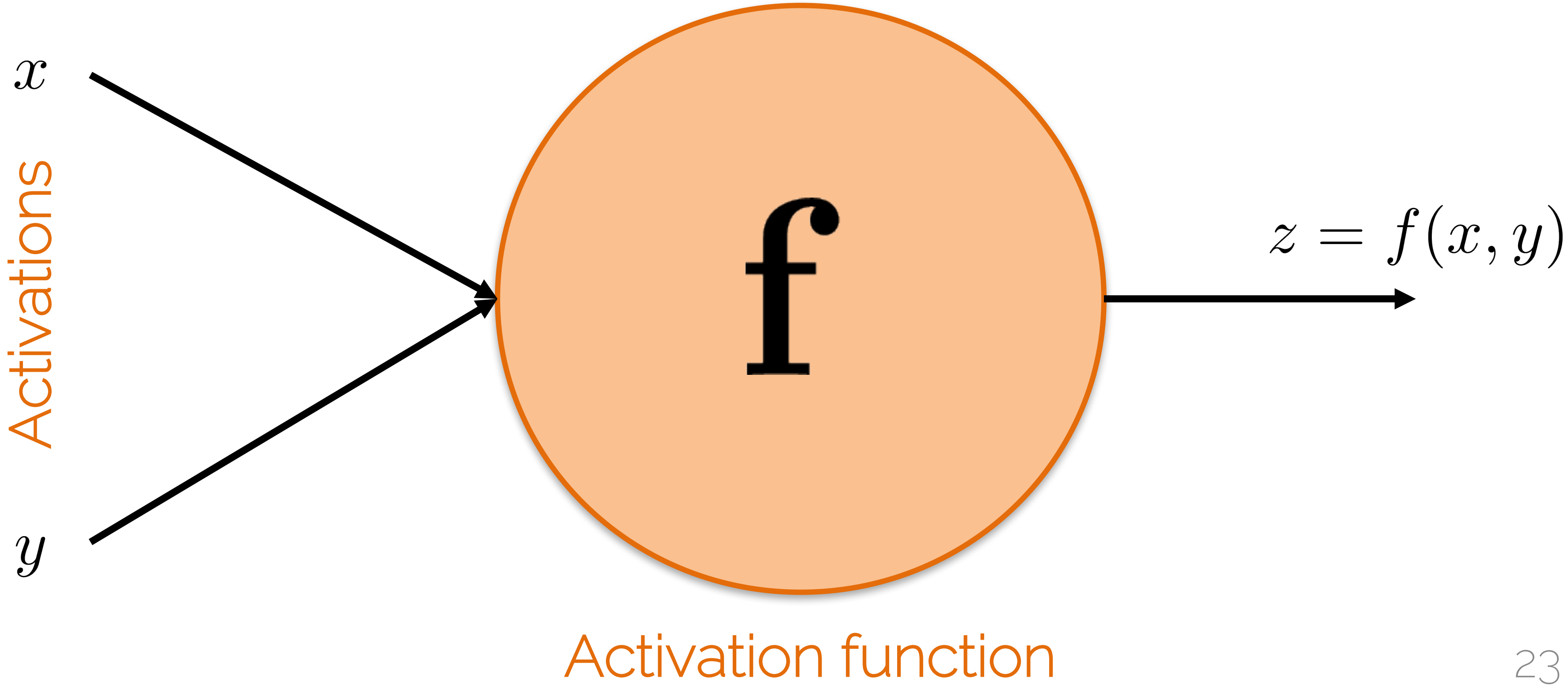


An example: the chain rule

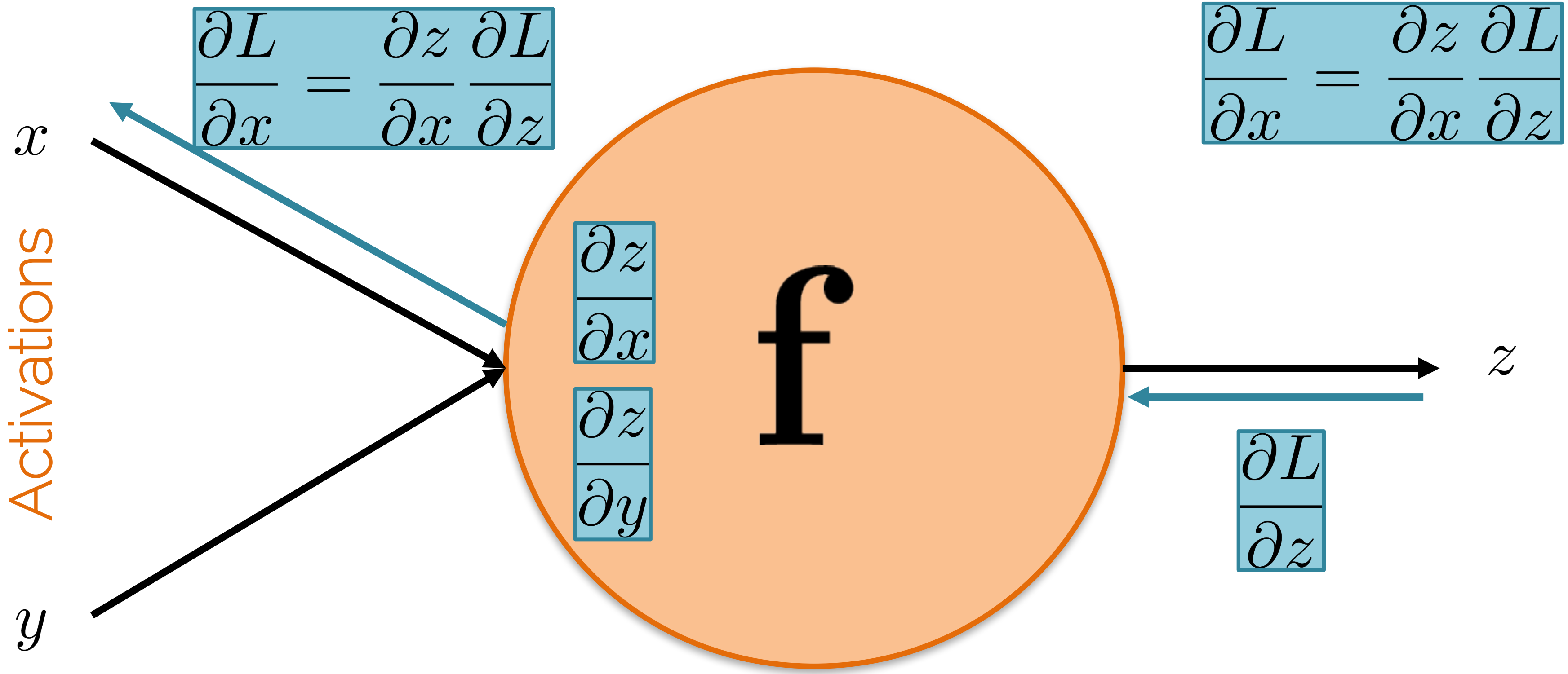
- Each node is only interested in its own inputs and outputs



The flow of the gradients



The flow of the gradients



Activation function

The flow of the gradients

- Many many many many of these nodes form a neural network

NEURONS

- Each one has its own work to do

FORWARD AND BACKWARD PASS

Optimization

Optimization

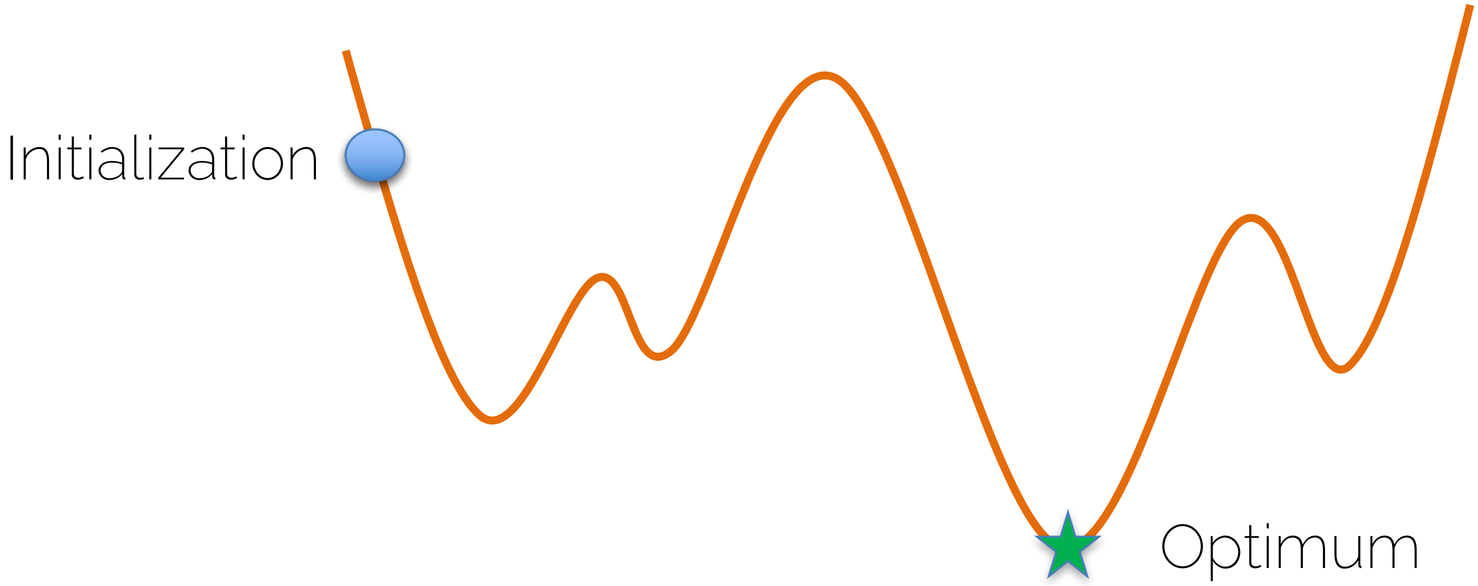
$$\boldsymbol{\theta}_{ML} = \arg\max_{\theta} p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})$$

- Complex function that cannot be derived in closed form
- Fast way to find a minimum
- Scales to large datasets

Gradient descent

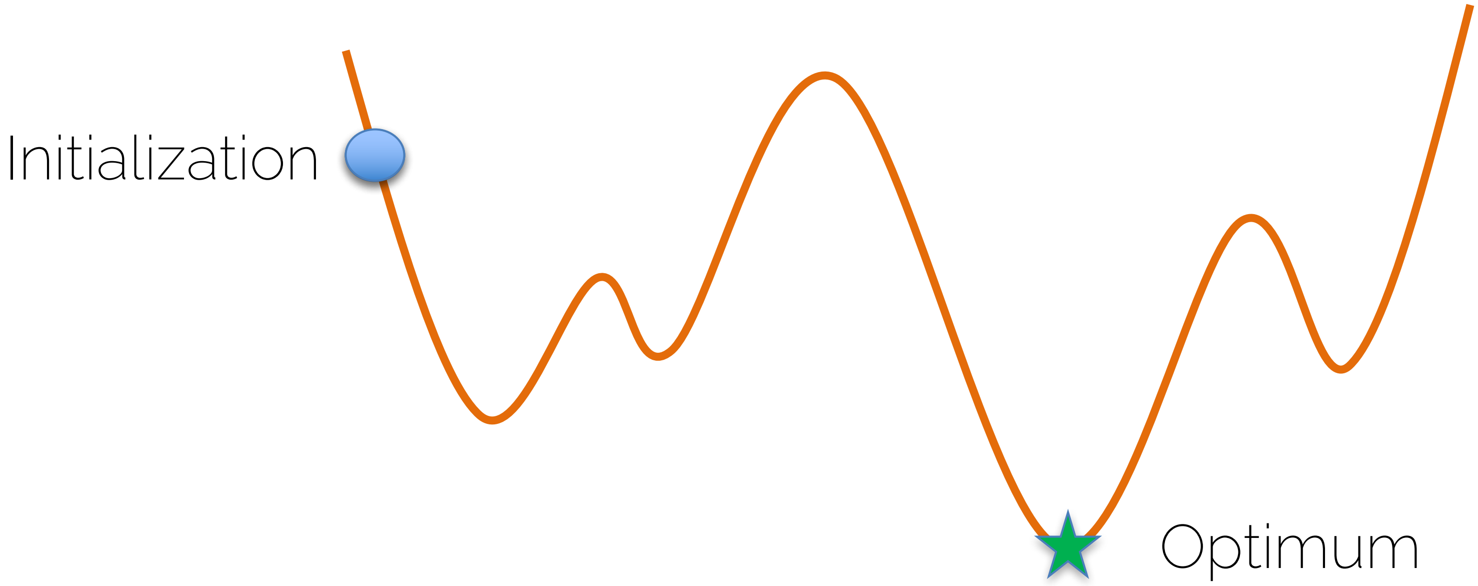
Following the slope

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



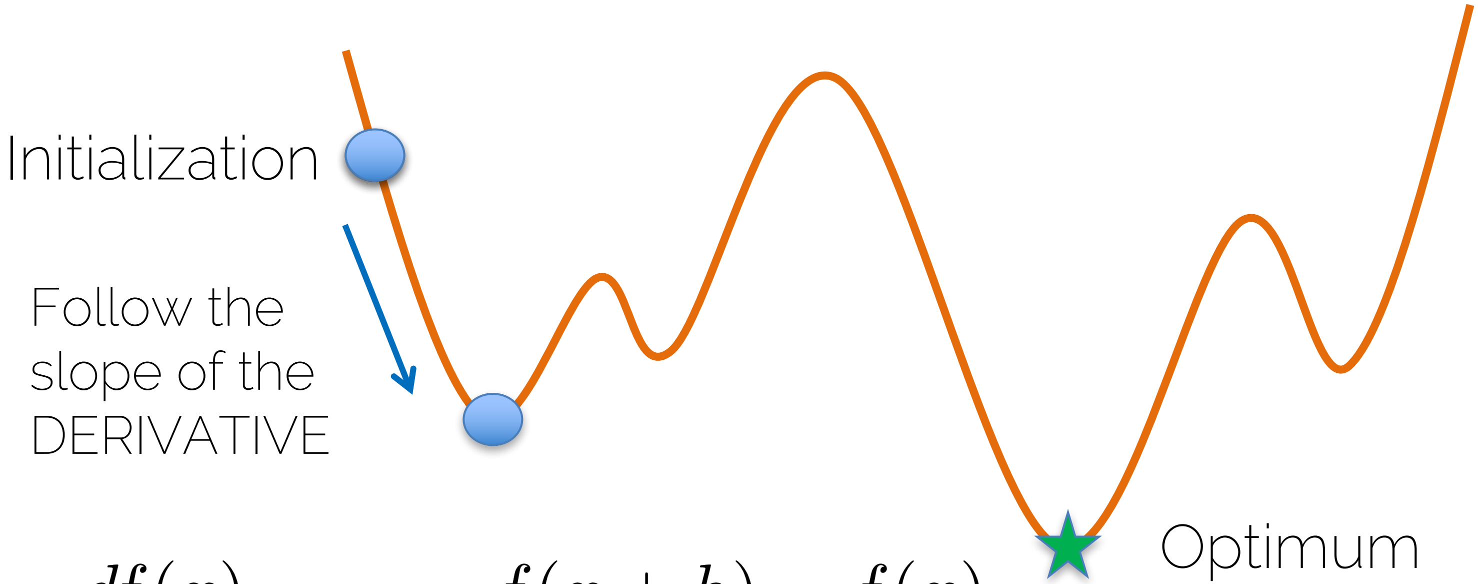
Following the slope

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



Following the slope

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



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

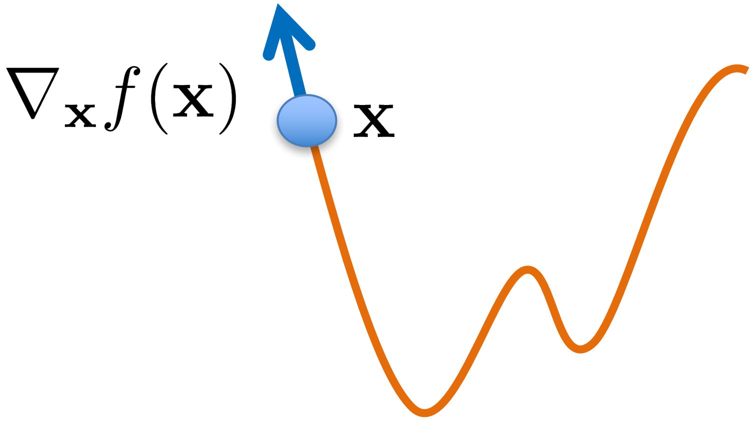
Gradient steps

- From derivative to gradient

$$\frac{df(x)}{dx} \longrightarrow \nabla_{\mathbf{x}} f(\mathbf{x})$$

Direction of greatest increase of the function

- Gradient steps in direction of negative gradient



$$\mathbf{x}' = \mathbf{x} - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x})$$

Learning rate

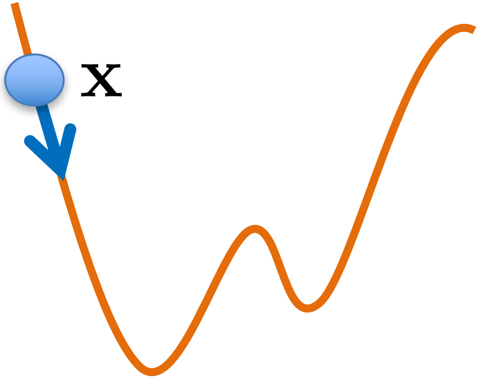
Gradient steps

- From derivative to gradient

$$\frac{df(x)}{dx} \longrightarrow \nabla_{\mathbf{x}} f(\mathbf{x})$$

Direction of greatest increase of the function

- Gradient steps in direction of negative gradient

$\nabla_{\mathbf{x}} f(\mathbf{x})$  $\mathbf{x}$

$$\mathbf{x}' = \mathbf{x} - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x})$$

SMALL Learning rate

Gradient steps

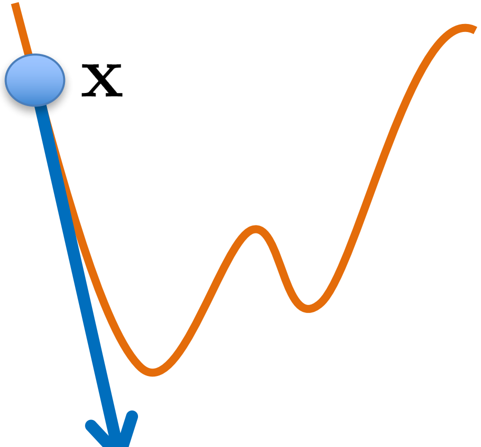
- From derivative to gradient

$$\frac{df(x)}{dx} \longrightarrow \nabla_{\mathbf{x}} f(\mathbf{x})$$

Direction of greatest increase of the function

- Gradient steps in direction of negative gradient

$\nabla_{\mathbf{x}} f(\mathbf{x})$



$\mathbf{x}$

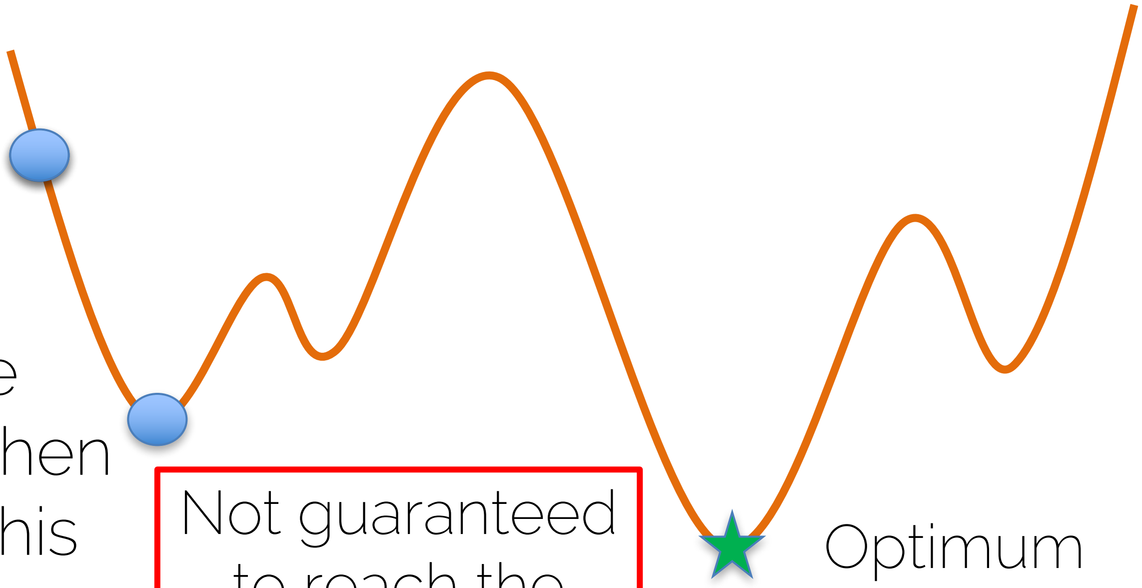
$$\mathbf{x}' = \mathbf{x} - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x})$$

LARGE Learning rate

Convergence

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

Initialization



What is the gradient when we reach this point?

Not guaranteed to reach the optimum

Optimum

Numerical gradient

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

- Approximate
- Slow evaluation

Analytical gradient

- Exact and fast

Remember Linear Regression

$$f(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

$$f(\boldsymbol{\theta}) = \frac{1}{n}(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

Analytical gradient

$$2\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} - 2\mathbf{X}^T\mathbf{y}$$

Gradient descent for least squares

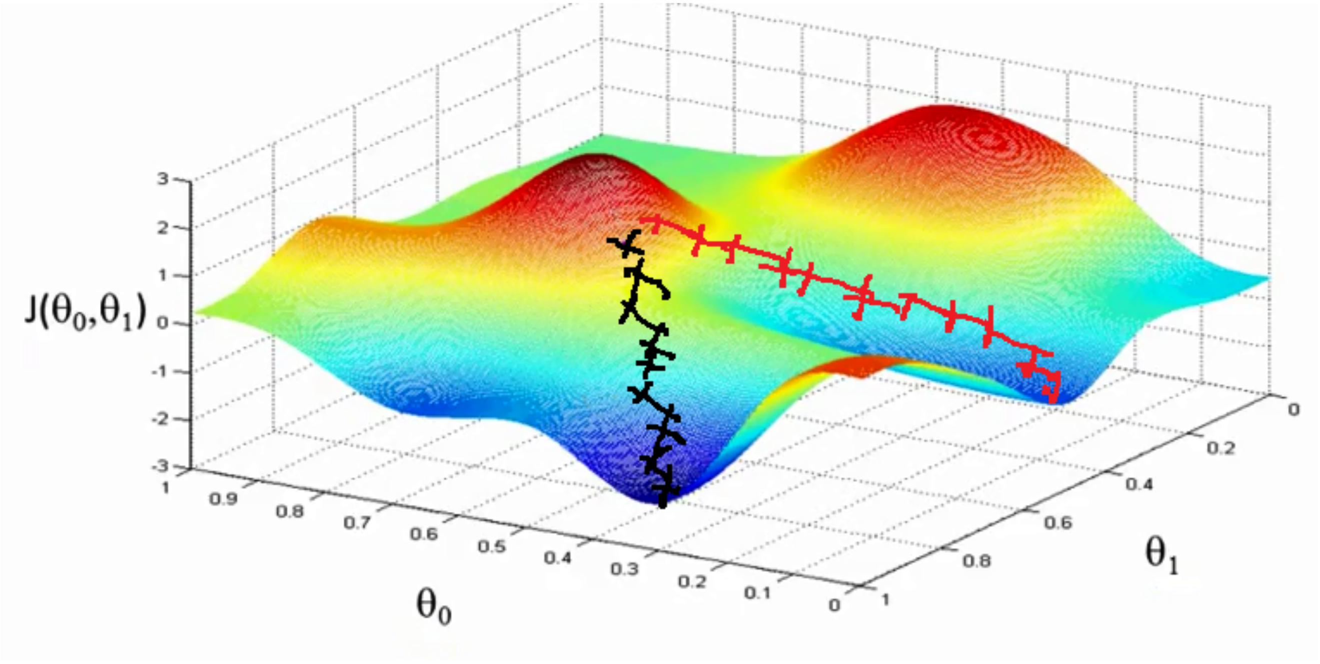
$$f(\boldsymbol{\theta}) = \frac{1}{n}(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon\, 2\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} - 2\mathbf{X}^T\mathbf{y}$$

Convex, always converges to the same solution

Non-linear least squares

- Not necessarily convex



Stochastic Gradient Descent

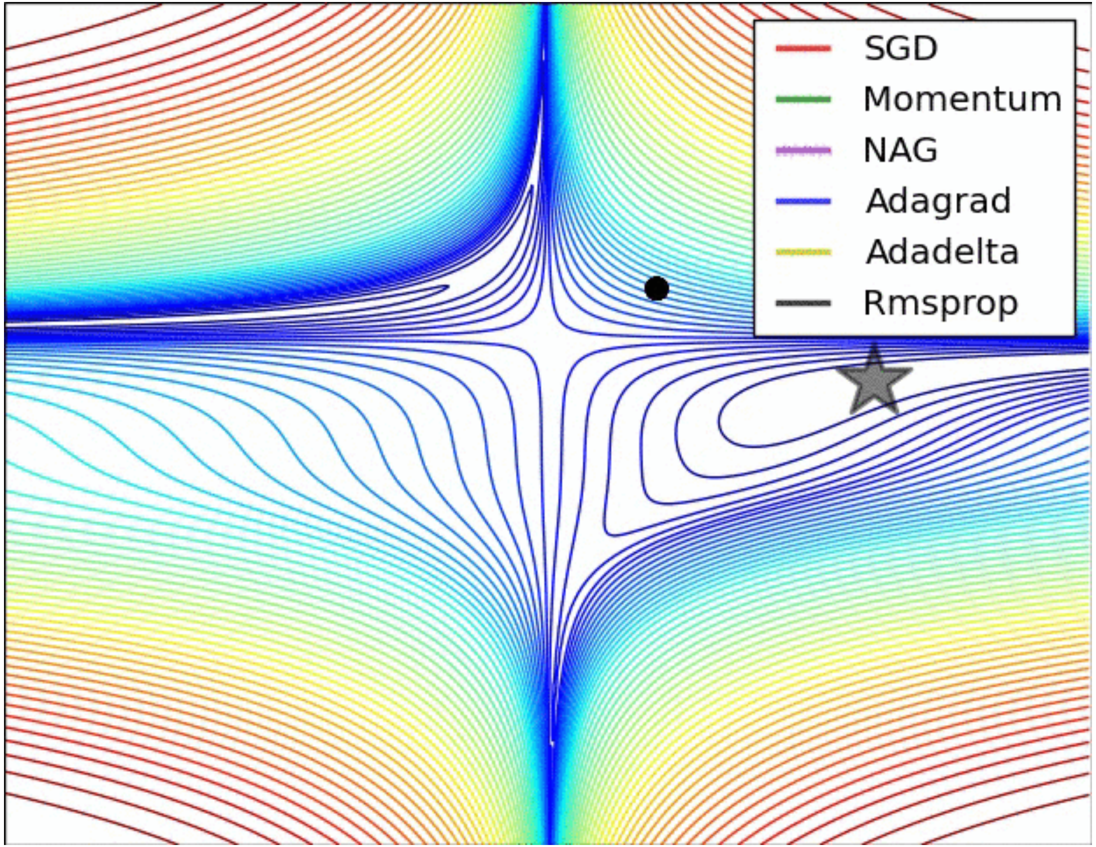
- If we have m training samples we need to compute the gradient for all of them which is $\mathcal{O}(m)$

- Gradient is an expectation, and so it can be approximated with a small number of samples

Minibatch $\quad \mathbb{B} = \{x^1, \cdots, x^{m'}\}$

Epoch = complete pass through all the data

Convergence



Stochastic gradient descent

Gradient

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon \boxed{\frac{1}{m}\sum_i \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i)}$$

Model

Loss

Ignore the sum for convenience ☺

SGD $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i)$

Momentum update

- Designed to accelerate training
- Define a new term called velocity $\mathbf{v}$

$$\mathbf{v}_{k+1} = \alpha \mathbf{v}_k - \epsilon \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i)$$

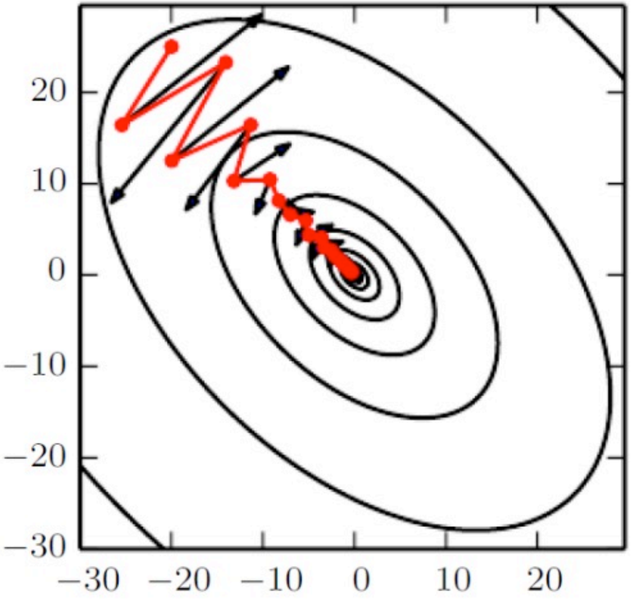
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \mathbf{v}_{k+1}$$

- The velocity accumulates gradients

SGD $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i)$ Polyack 1964

Momentum update

$$\mathbf{v}_{k+1} = \alpha \mathbf{v}_k - \epsilon \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i) \qquad \boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \mathbf{v}_{k+1}$$

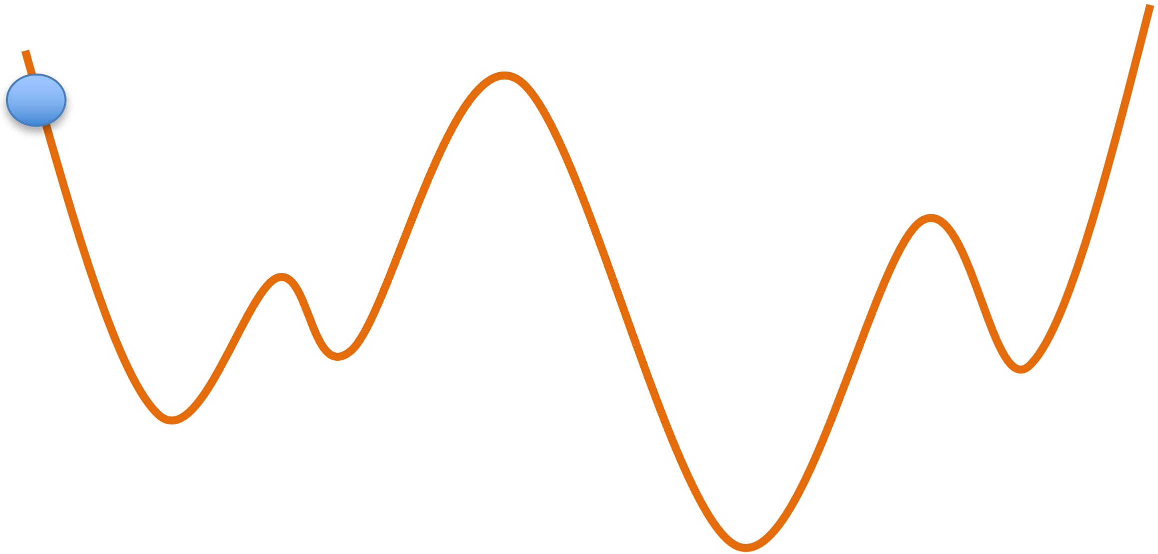


Step will be largest when a sequence of gradients all point to the same direction

Image: Goodfellow et al.

Momentum update

- Can it overcome local minima?



$$\mathbf{v}_{k+1} = \alpha \mathbf{v}_k - \epsilon \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i)$$

Nesterov's momentum

- *Look-ahead* momentum

$$\widetilde{\boldsymbol{\theta}}_{k+1} = \boldsymbol{\theta}_k + \mathbf{v}_k$$

$$\mathbf{v}_{k+1} = \alpha \mathbf{v}_k - \epsilon \nabla_{\boldsymbol{\theta}} L(\widetilde{\boldsymbol{\theta}}_{k+1}, \mathbf{x}^i, \mathbf{y}^i)$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \mathbf{v}_{k+1}$$

SGD $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i)$ Sutskever 2013, Nesterov 1983

Nesterov's momentum

- *Look-ahead* momentum

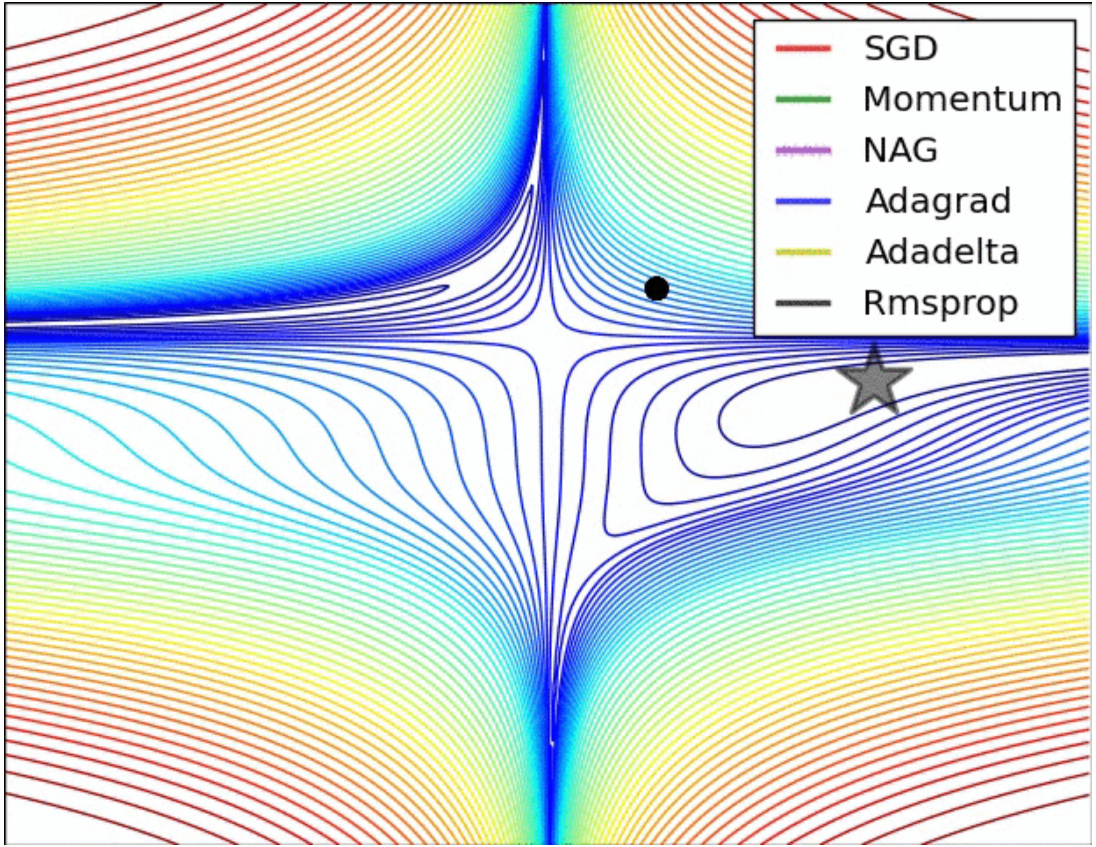
$$\widetilde{\boldsymbol{\theta}}_{k+1} = \boldsymbol{\theta}_k + \mathbf{v}_k$$

$$\mathbf{v}_{k+1} = \alpha \mathbf{v}_k - \epsilon \nabla_{\boldsymbol{\theta}} L(\widetilde{\boldsymbol{\theta}}_{k+1}, \mathbf{x}^i, \mathbf{y}^i)$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \mathbf{v}_{k+1}$$

SGD $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i)$ Sutskever 2013, Nesterov 1983

Convergence



More parameters…

$$\mathbf{v}_{k+1} = \boxed{\alpha}\mathbf{v}_k - \boxed{\epsilon}\nabla_{\boldsymbol{\theta}} L(\widetilde{\boldsymbol{\theta}}_{k+1}, \mathbf{x}^i, \mathbf{y}^i)$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \boxed{\epsilon}\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i)$$

Can we relax the dependence on the hyperparameters?

AdaGrad update

- Adapt the learning rate of all model parameters

$$\mathbf{g} = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_{k+1}, \mathbf{x}^i, \mathbf{y}^i)$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k + \boxed{\mathbf{g} \odot \mathbf{g}}$$

Element-wise multiplication

Diagonal matrix with entries that are the square of the gradient

Duchi 2011

AdaGrad update

- Adapt the learning rate of all model parameters

$$\mathbf{g} = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_{k+1}, \mathbf{x}^i, \mathbf{y}^i)$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k + \mathbf{g} \odot \mathbf{g}$$

Accumulating gradients

AdaGrad update

- Adapt the learning rate of all model parameters

$$\mathbf{g} = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i)$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k + \mathbf{g} \odot \mathbf{g}$$

Learning rate

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \frac{\epsilon}{\delta + \sqrt{\mathbf{r}_{k+1}}} \odot \mathbf{g}$$

Small constant for
numerical stability

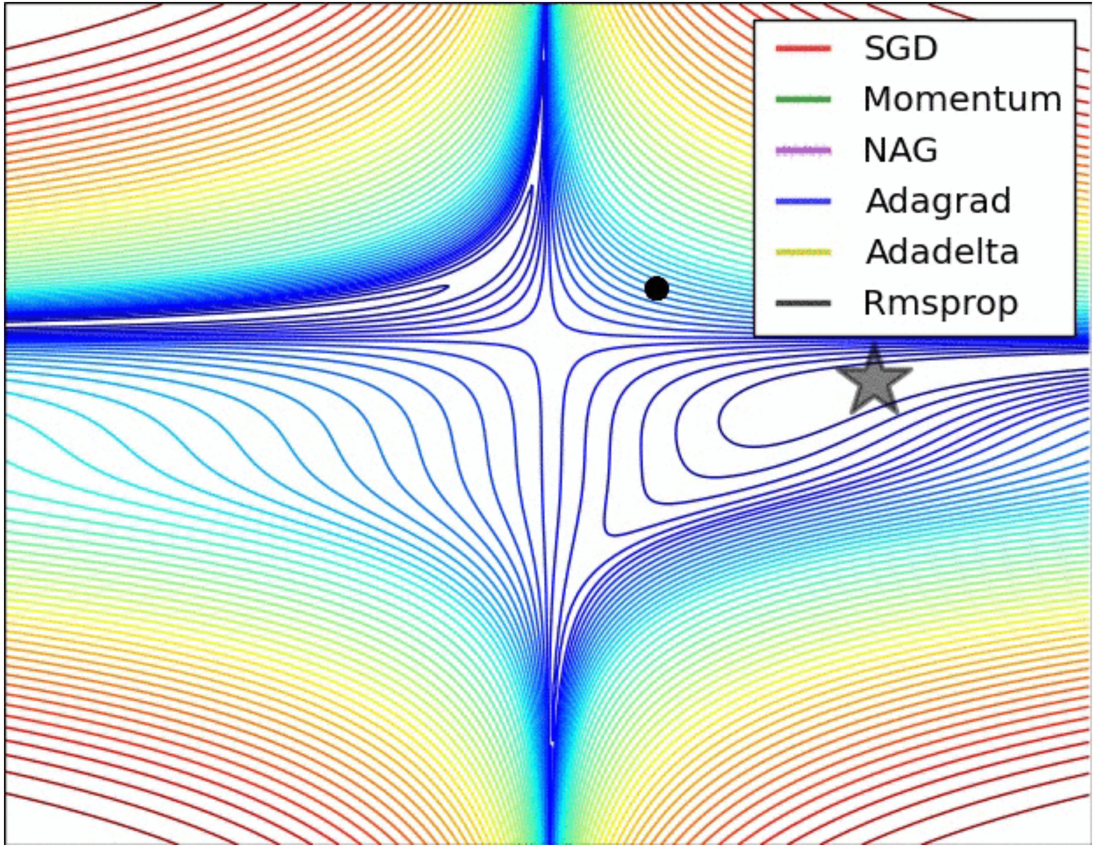
AdaGrad update

- Theory: more progress in regions where the function is more flat

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \frac{\epsilon}{\delta + \sqrt{\mathbf{r}_{k+1}}} \odot \mathbf{g}$$

- Practice: for most deep learning models, accumulating gradients from the beginning results in excessive decrease in the effective learning rate

Convergence

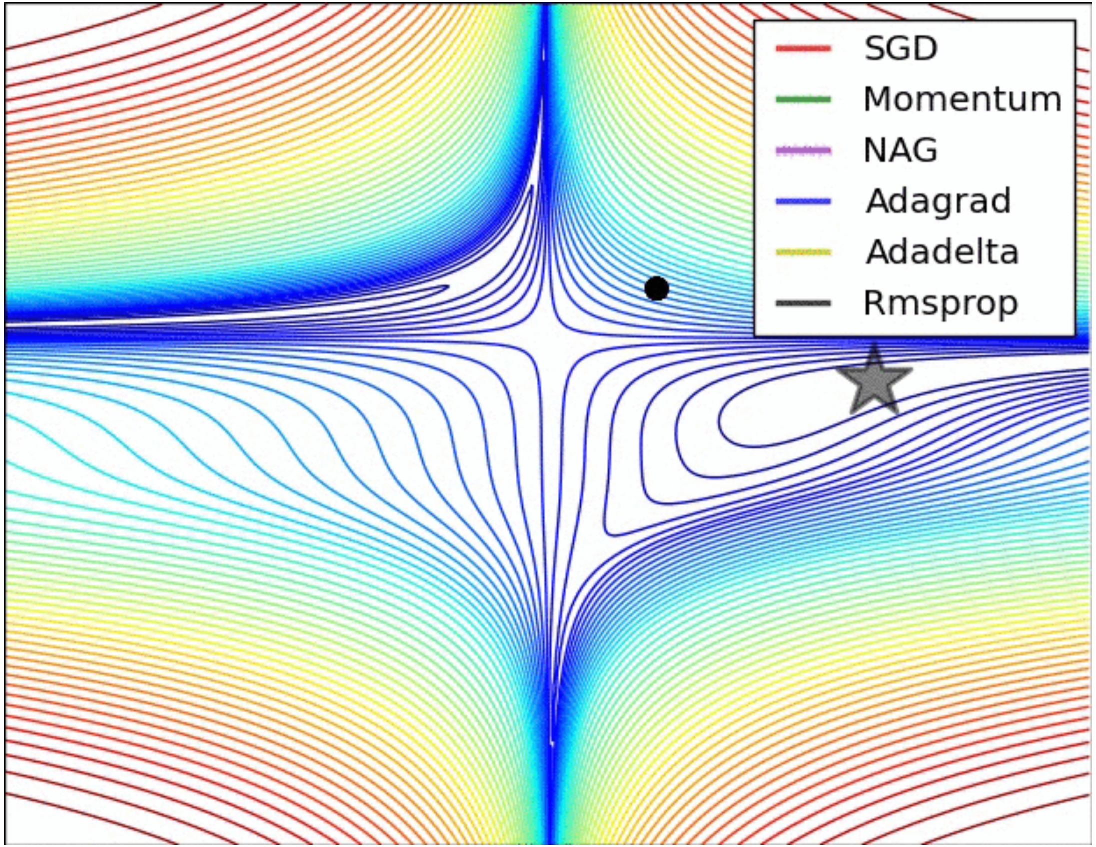


RMSProp and Adadelta

- Improvements to AdaGrad to avoid the problem of diminishing learning rate
- Decaying factor applied to the accumulation of gradients
- Old gradients are slowly forgotten

Zeiler, 2012. Hinton, 2012

Convergence



Adam

- Optimizer of choice for most neural networks
- Adam = adaptive moments
- It can be seen as an RMSProp with momentum

Kingma and Ba 2014

AdaGrad

$$\mathbf{g} = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i)$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k + \mathbf{g} \odot \mathbf{g}$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \frac{\epsilon}{\delta + \sqrt{\mathbf{r}_{k+1}}} \odot \mathbf{g}$$

Adam

$$\mathbf{g} = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i)$$

Second order moment

$$\mathbf{r}_{k+1} = \rho_2 \mathbf{r}_k + (1 - \rho_2)\mathbf{g} \odot \mathbf{g}$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon \frac{\hat{\mathbf{s}}}{\delta + \sqrt{\hat{\mathbf{r}}_{k+1}}}$$

Adam

We can consider it as momentum

Gradient $\mathbf{g} = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i)$

First order moment $\mathbf{s}_{k+1} = \rho_1 \mathbf{s}_k + (1 - \rho_1)\mathbf{g}$

Second order moment $\mathbf{r}_{k+1} = \rho_2 \mathbf{r}_k + (1 - \rho_2)\mathbf{g} \odot \mathbf{g}$

Unbias the moments $\hat{\mathbf{s}}_{k+1} = \dfrac{\mathbf{s}_{k+1}}{1 - \rho_1}$ $\hat{\mathbf{r}}_{k+1} = \dfrac{\mathbf{r}_{k+1}}{1 - \rho_2}$

Update step $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon \dfrac{\hat{\mathbf{s}}}{\delta + \sqrt{\hat{\mathbf{r}}_{k+1}}}$

Adam

Unbias the moments $\hat{\mathbf{s}}_{k+1} = \frac{\mathbf{s}_{k+1}}{1-\rho_1}$ $\hat{\mathbf{r}}_{k+1} = \frac{\mathbf{r}_{k+1}}{1-\rho_2}$

- Both moments are initialized to zero, which means that specially at the beginning they have a tendency to converge to zero

$$\rho_1 = 0.9 \qquad \rho_2 = 0.999$$

$\boxed{\text{Go-to optimizer}}$

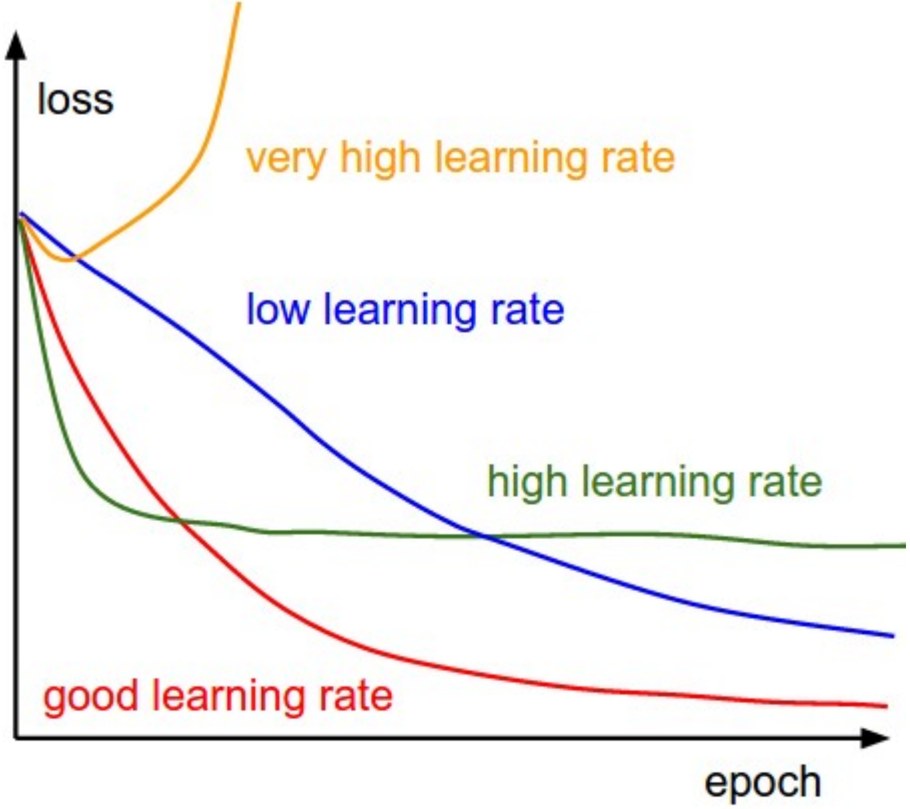
So far

- Classic optimizers: SGM, Momentum, Nesterov's momentum

- Adaptive learning rates: AdaGrad, Adadelta, RMSProp and Adam

Can we get rid of the learning rate?

Importance of the learning rate



Jacobian and Hessian

- Derivative $\mathbf{f}:\mathbb{R}\rightarrow\mathbb{R}$ $\frac{df(x)}{dx}$
- Gradient $\mathbf{f}:\mathbb{R}^m\rightarrow\mathbb{R}$ $\nabla_{\mathbf{x}}f(\mathbf{x})$ $\left(\frac{df(x)}{dx_1},\frac{df(x)}{dx_2}\right)$
- Jacobian $\mathbf{f}:\mathbb{R}^m\rightarrow\mathbb{R}^n$ $\mathbf{J}\in\mathbb{R}^{n\times m}$
- Hessian $\mathbf{f}:\mathbb{R}^m\rightarrow\mathbb{R}$ $\mathbf{H}\in\mathbb{R}^{m\times m}$ SECOND DERIVATIVE

Newton's method

- Approximate our function by a second-order Taylor series expansion

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

First derivative

Second derivative (curvature)

https://en.wikipedia.org/wiki/Taylor_series

Newton's method

- SGD (green)

- Newton's method exploits the curvature to take a more direct route

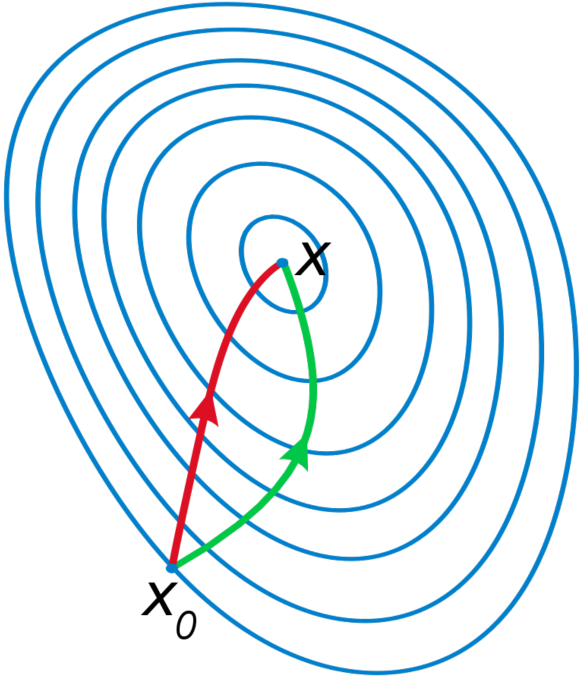


Image from Wikipedia

Newton's method

- Differentiate and equate to zero

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 \boxed{- \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})}$$ Update step

We got rid of the learning rate!

SGD $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i)$

Newton's method

- Differentiate and equate to zero

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 \boxed{- \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})}$$

Update step

Parameters of a network (millions)	Number of elements in the Hessian	Computational complexity of inversion per iteration
k	k^2	$\mathcal{O}(k^3)$

Only small networks can be trained with this method

Newton's method

$$J(\boldsymbol{\theta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$$

Can you apply Newton's method for linear regression? What do you get as a result?

BFGS and L-BFGS

- Broyden-Fletcher-Goldfarb-Shanno algorithm
- Belongs to the family of quasi-Newton methods
- Have an approximation of the inverse of the Hessian

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boxed{\mathbf{H}^{-1}} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$$

- BFGS $\mathcal{O}(n^2)$
- Limited memory: L-BFGS $\mathcal{O}(n)$

Which, what and when?

- Standard: Adam
- Fall-back option: SGD with momentum
- L-BFGS if you can do full batch updates (forget applying it to minibatches!!)

Next lecture

- NO LECTURE on November 14^{th}!
- Thursday November 16^{th}: exercise 1 solution and presentation of exercise 2