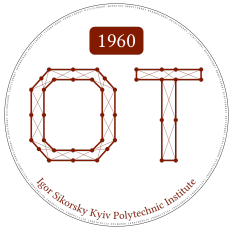



# Computer Vision

Lecture 5: Training neural networks

Yuriy Kochura
iuriy.kochura@gmail.com
@y_kochura

# Today

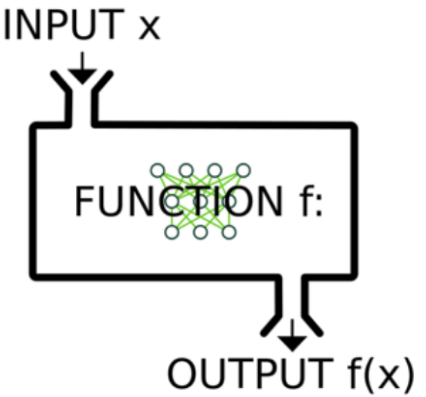
How to optimize parameters efficiently?

- Gradient descent
- Stochastic Gradient Descent
- Mini-batching
- Momentum
- Adaptive methods:
  - AdaGrad
  - RMSProp
  - Adam

# Parameters optimization

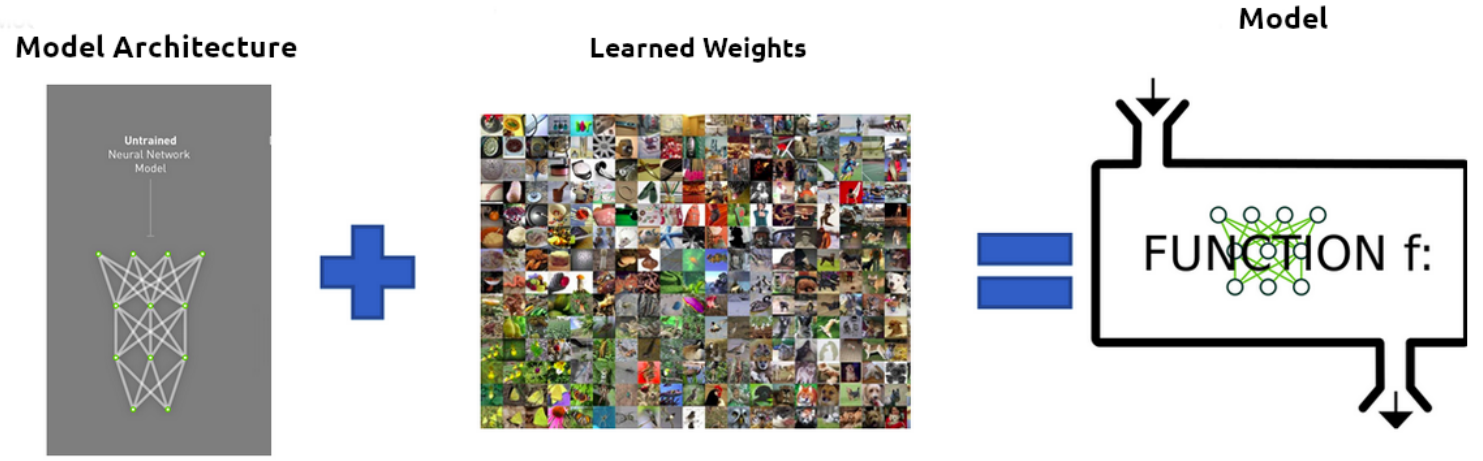
# A model

While what is inside of a deep neural network may be complex, at their core, they are simply functions. They take some input and generate some output.




—
Credits: NVIDIA

# Components of a model



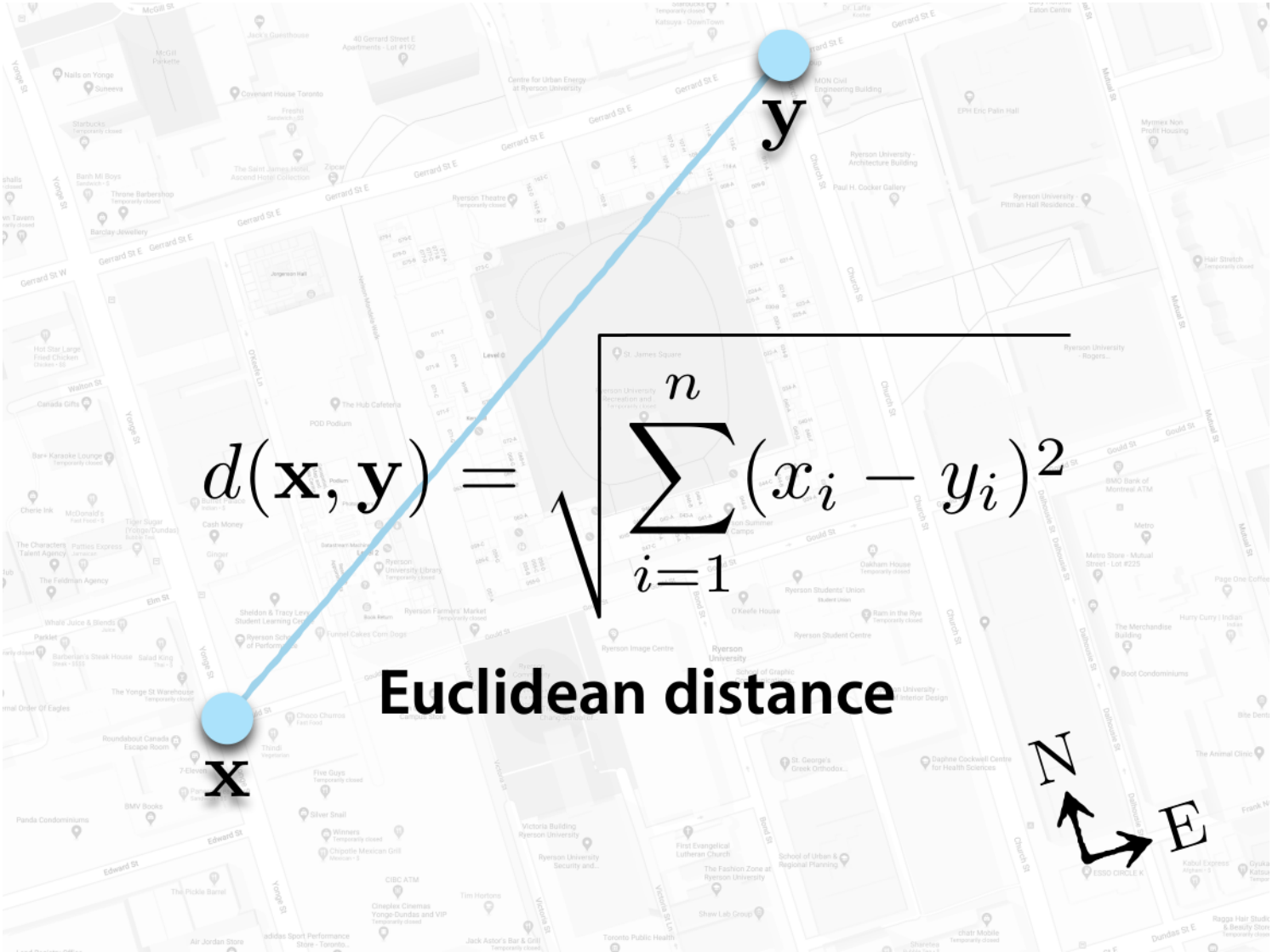

—
Credits: NVIDIA

# A common training process for neural networks

1. Define a task + collect data
2. Initialize the parameters
3. Choose an optimization algorithm
4. Repeat these steps:
    4.1. Forward propagate an input
    4.2 Compute the cost function
    4.3 Compute the gradients of the cost with respect to parameters using backpropagation
    4.4 Update each parameter using the gradients, according to the optimization algorithm

# Common Loss functions

- Euclidean distance (L2 Loss)
- Mean Square Error (MSE)
- Manhattan distance (L1 Loss)
- Mean Absolute Error (MAE)
- Cross Entropy Loss




Slide source: Deep Learning in Computer Vision

# Euclidean distance (L2 Loss)

$$d(\hat{y}, y) = \sqrt{\sum_{i=1}^{n} \left(\hat{y}^{(i)} - y^{(i)}\right)^2} = \|\hat{y} - y\|_2$$

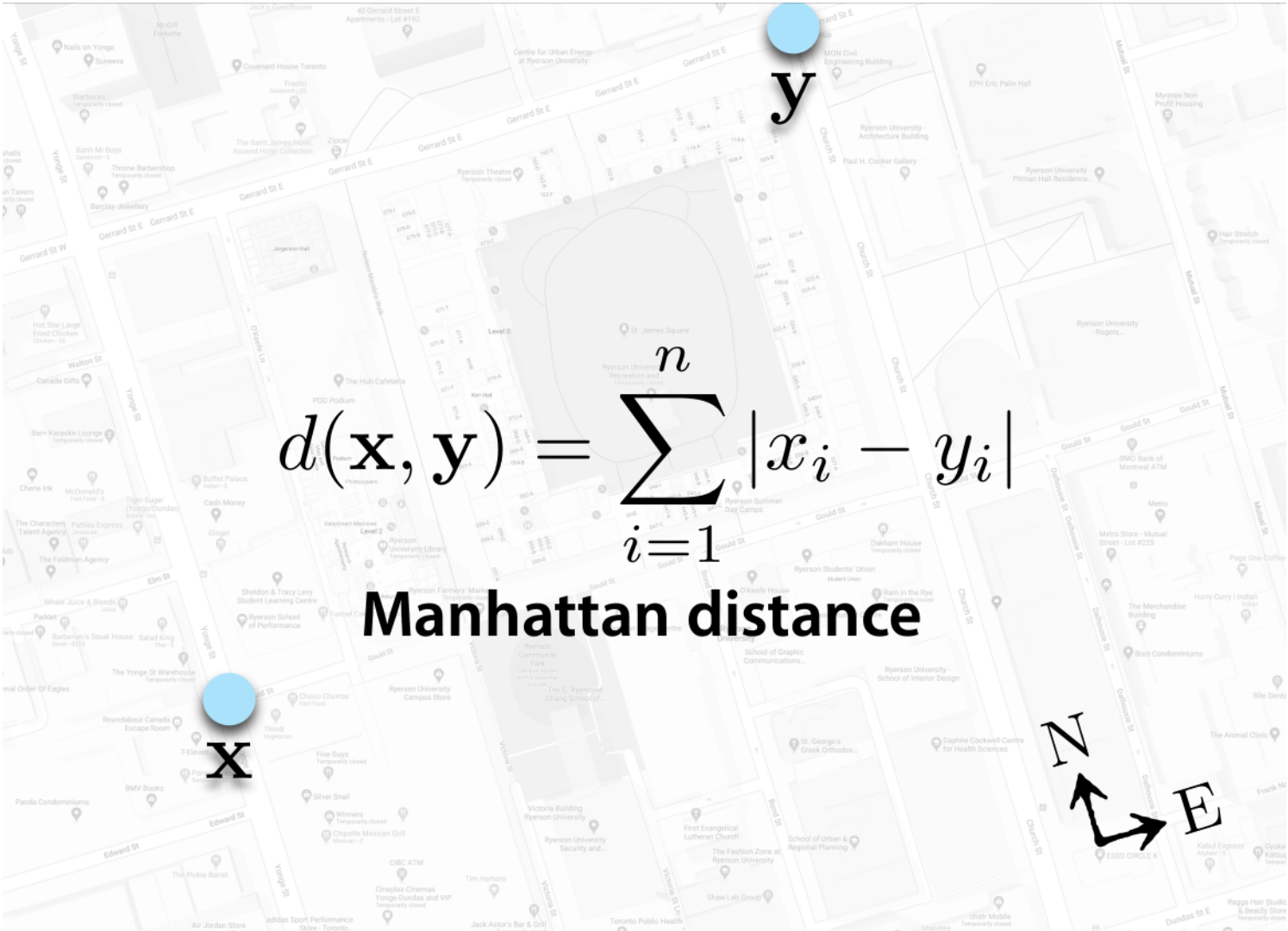
where $n$ is a total number of training examples

# Mean Square Error (MSE)

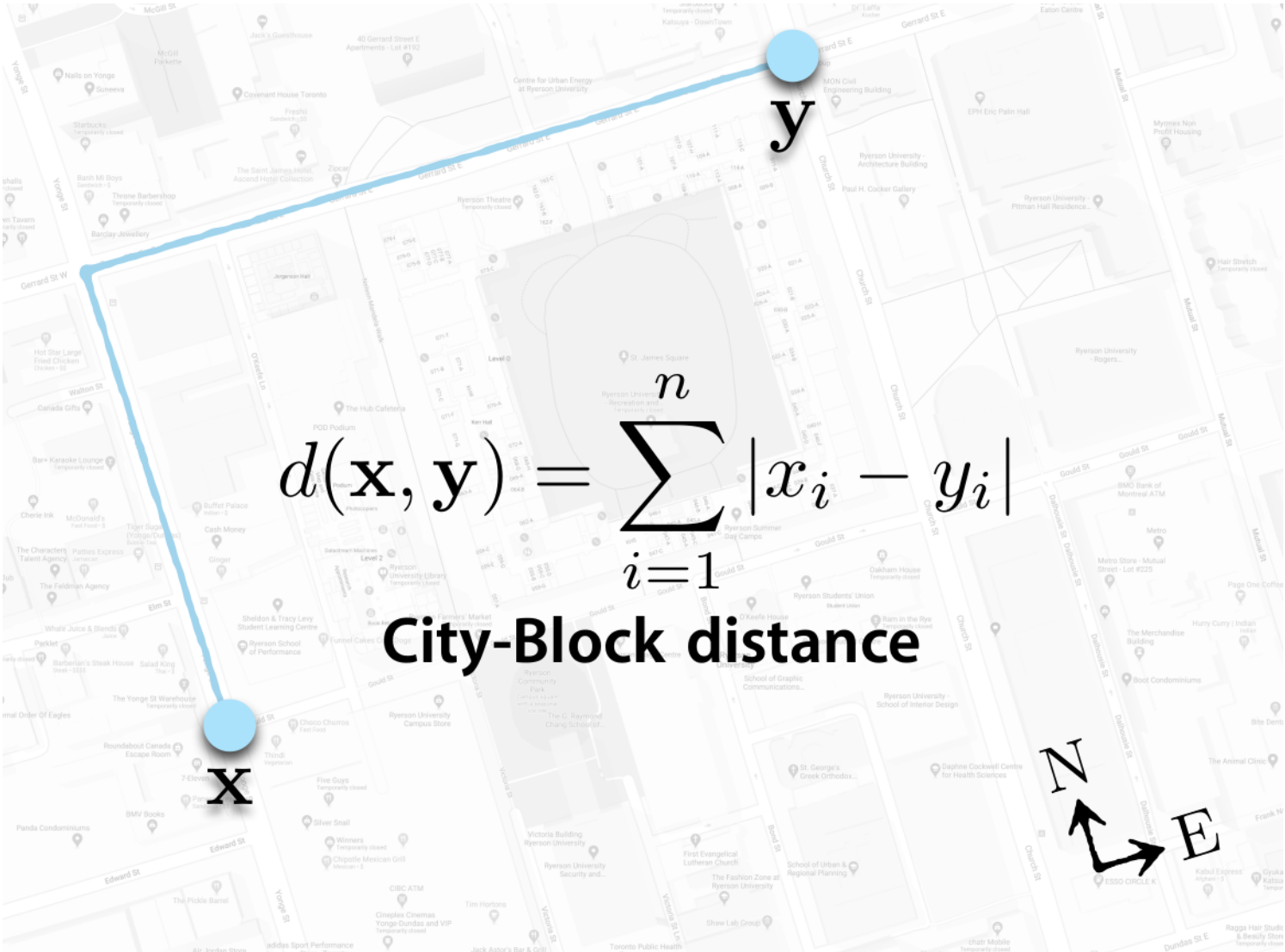
$$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = (\hat{y}^{(i)} - y^{(i)})^2$$

$$\mathcal{J}(\hat{y}, y) = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\left(\hat{y}^{(i)} - y^{(i)}\right)^2} = \frac{1}{n}\left\|\hat{y} - y\right\|_2$$

where $n$ is a total number of training examples




Slide source: Deep Learning in Computer Vision




—

Slide source: Deep Learning in Computer Vision

# Manhattan distance (L1 Loss)

$$d(\hat{y}, y) = \sum_{i=1}^{n} |\hat{y}^{(i)} - y^{(i)}| = \|\hat{y} - y\|_1$$

where $n$ is a total number of training examples

# Mean Absolute Error (MAE)

$$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = |\hat{y}^{(i)} - y^{(i)}|$$

$$\mathcal{J}(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} |\hat{y}^{(i)} - y^{(i)}| = \frac{1}{n} \|\hat{y} - y\|_1$$

where $n$ is a total number of training examples

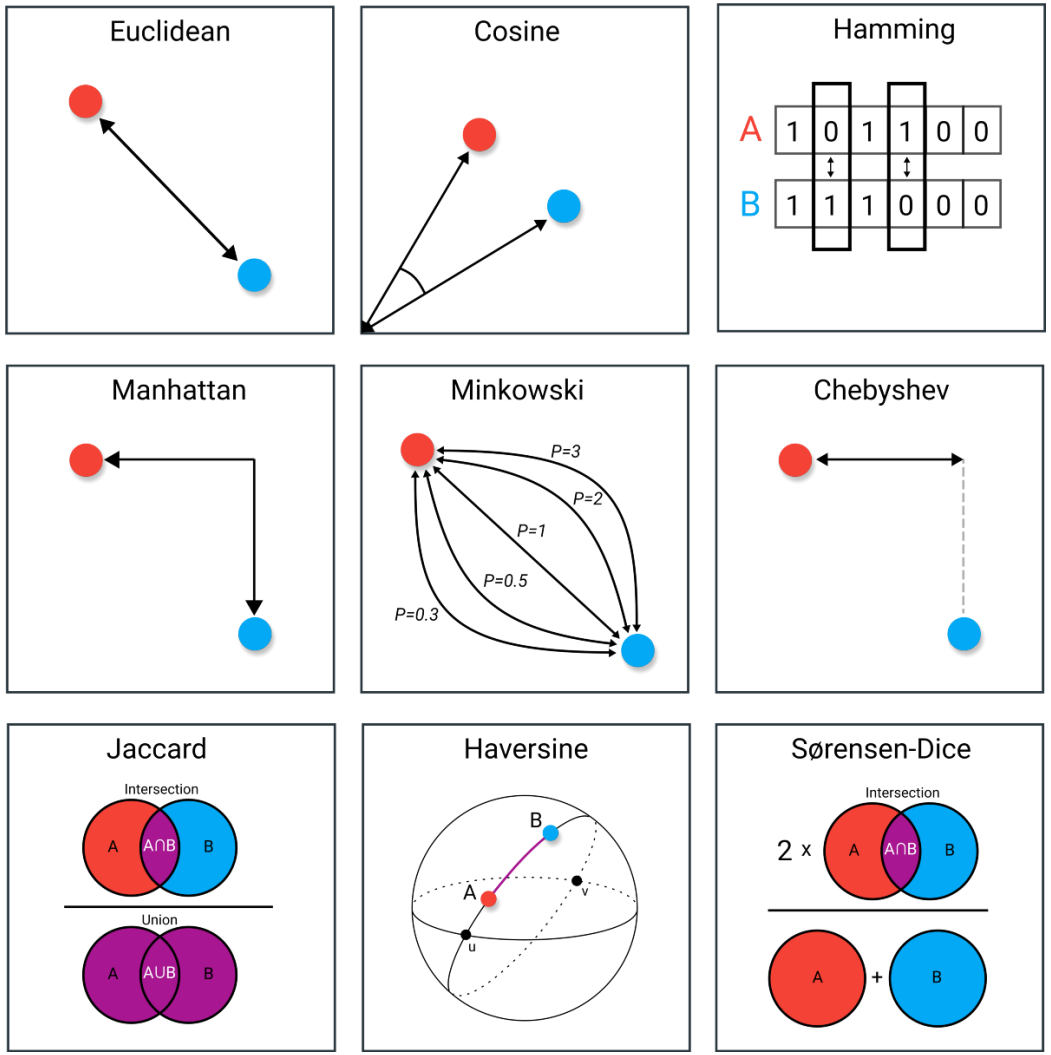
# Cross Entropy Loss (binary)

$$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

$$\mathcal{J}(\hat{y}, y) = -\frac{1}{n} \sum_{i=1}^{n} \left[ y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

where $n$ is a total number of training examples

# 9 Distance Measures in Data Science




—

Slide source: 9 Distance Measures in Data Science


—

Slide source: Deep Learning in Computer Vision

# Optimization algorithms

# Hyperparameters & Parameters

- Result of optimization is a set of parameters
- Hyperparameters vs. Parameters

## Hyperparameters

- Train-test split ratio
- Learning rate in optimization algorithms
- Number of hidden layers in a NN
- Number of activation units in each layer
- Batch size

⋮

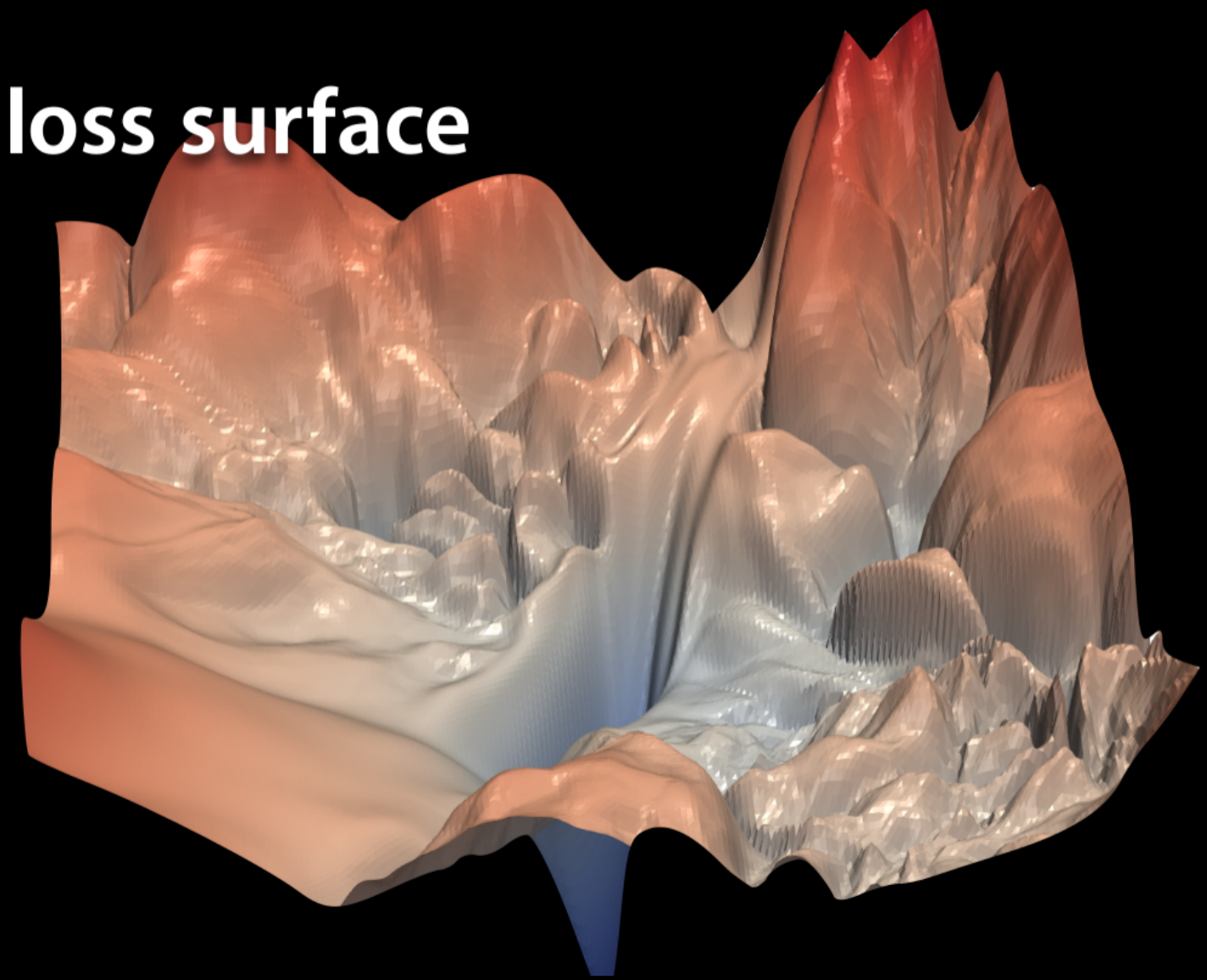
## Parameters

- Weights and biases of a NN
- The cluster centroids in clustering

# Problem of optimization

## Empirical risk minimization

$$W_*^{\mathbf{d}} = \arg\min_W \mathcal{J}(W) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(y^{(i)}, f(\mathbf{x}^{(i)}, W)\right)$$




—

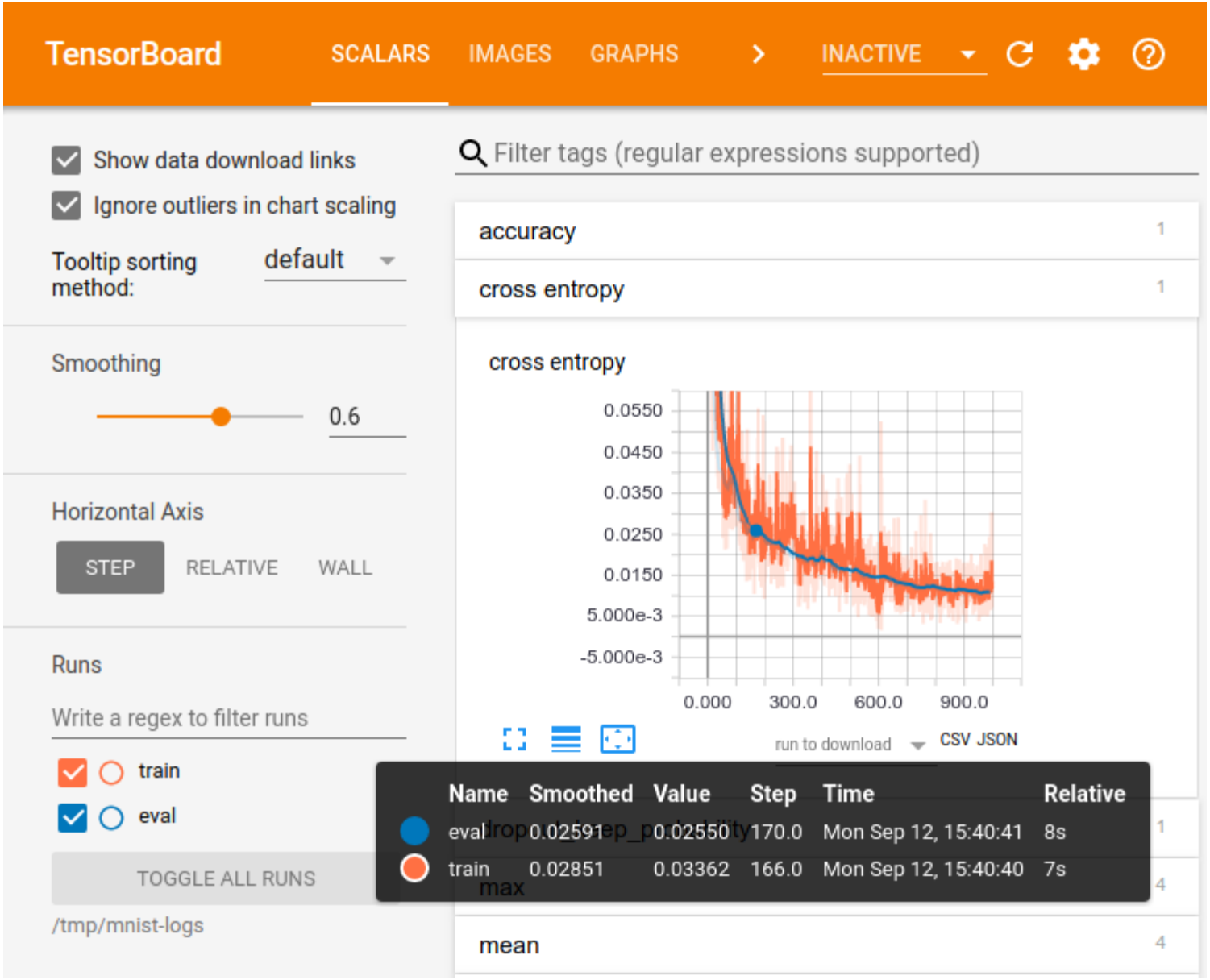
Slide source: Deep Learning in Computer Vision

## A practical recommendation

Training a massive deep neural network is long, complex and sometimes confusing.

A first step towards understanding, debugging and optimizing neural networks is to make use of visualization tools for

- plotting losses and metrics,
- visualizing computational graphs,
- or showing additional data as the network is being trained.


TensorBoard
SCALARS
IMAGES
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method:
default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
train
eval
TOGGLE ALL RUNS
/tmp/mnist-logs
Filter tags (regular expressions supported)
accuracy 1
cross entropy 1
cross entropy
0.0550
0.0450
0.0350
0.0250
0.0150
5.000e-3
-5.000e-3
0.000
300.0
600.0
900.0
run to download
CSV JSON
Name Smoothed Value Step Time Relative
eval 0.02591 0.02550 170.0 Mon Sep 12, 15:40:41 8s
train 0.02851 0.03362 166.0 Mon Sep 12, 15:40:40 7s
max 4
mean 4

# Demo

# Gradient Descent

GD

# Gradient descent

To minimize $\mathcal{J}(W)$, standard batch gradient descent (GD) consists in applying the update rule

$$g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla_W \mathcal{L}\left(y^{(i)}, f(\mathbf{x}^{(i)}, W)\right) = \nabla_W \mathcal{J}(W)$$
$$W_{t+1} = W_t - \alpha g_t,$$

where $\alpha$ is the learning rate.

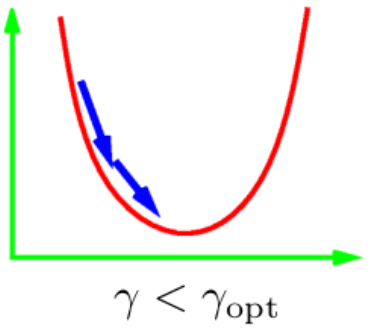
# GD

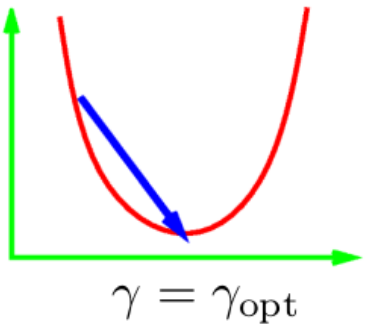
The worst optimization method in the world

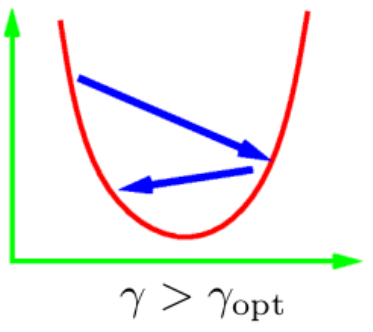
Side note: You should probably never use gradient descent directly, consider it a building block for other methods.

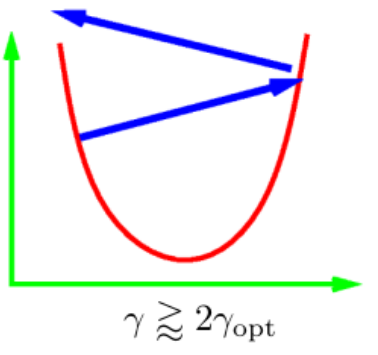
—
Credits: Aaron Defazio, Facebook AI Research

# Learning Rate



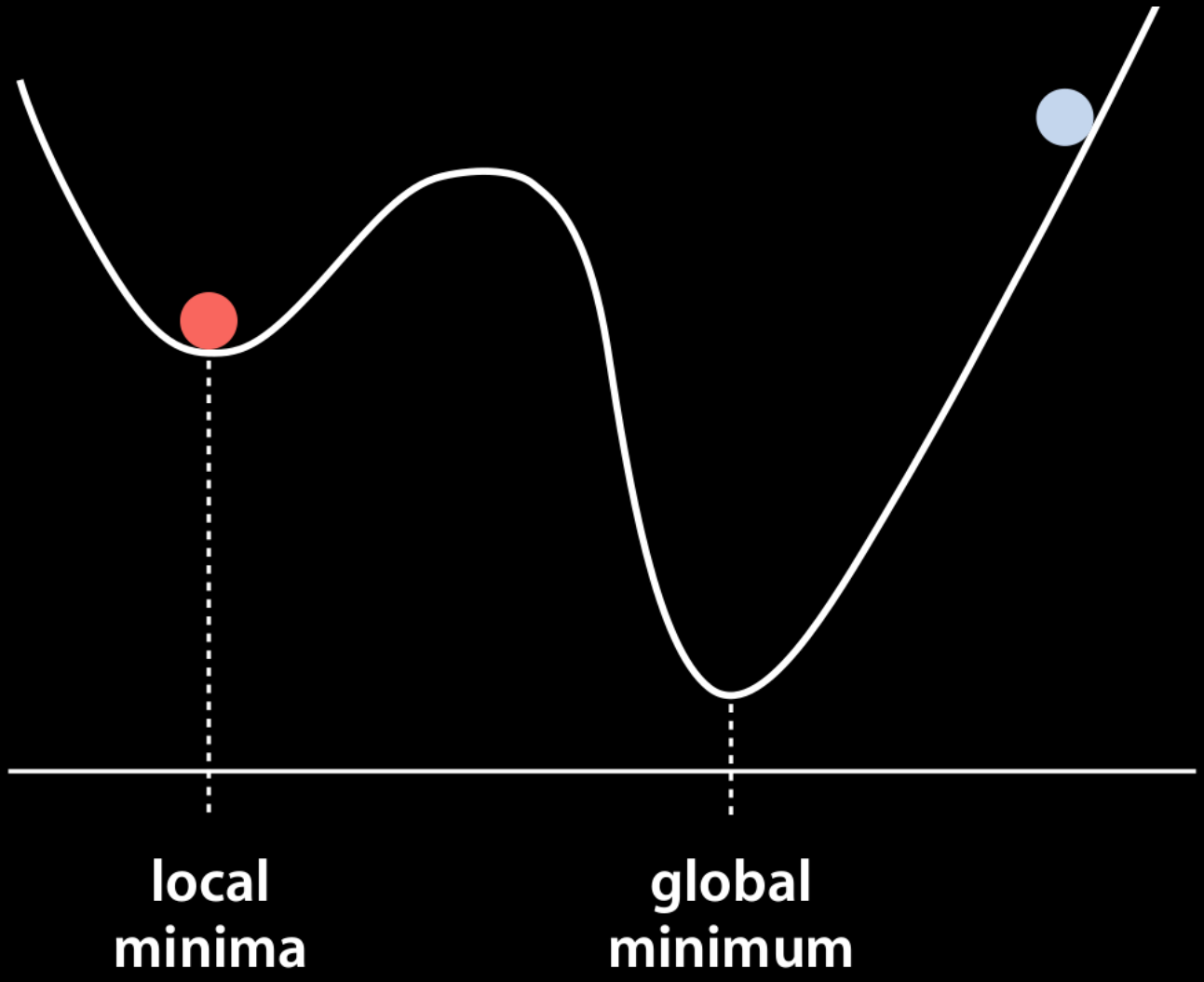



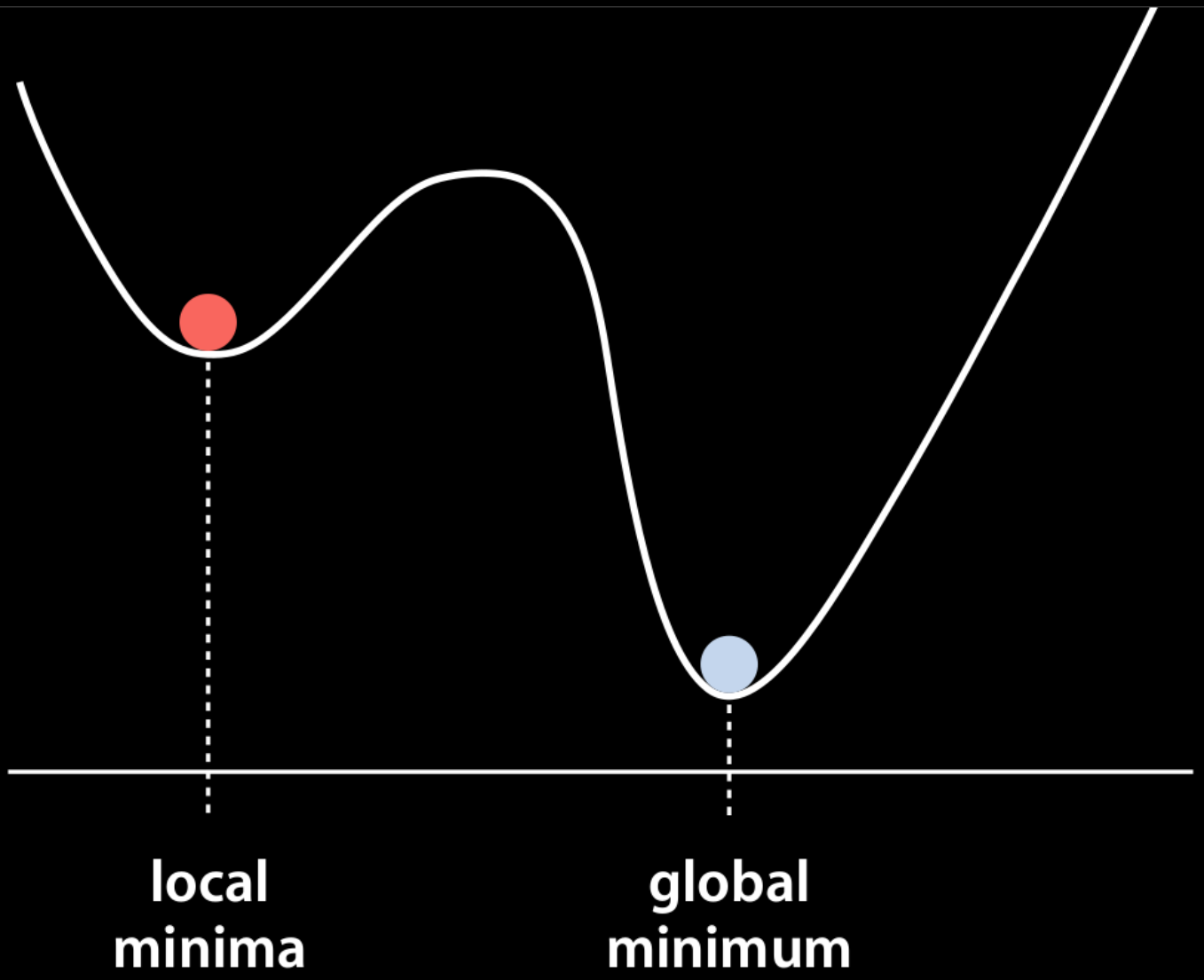

Here $\gamma = \alpha$ is the learning rate.

—
Credits: Aaron Defazio, Facebook AI Research



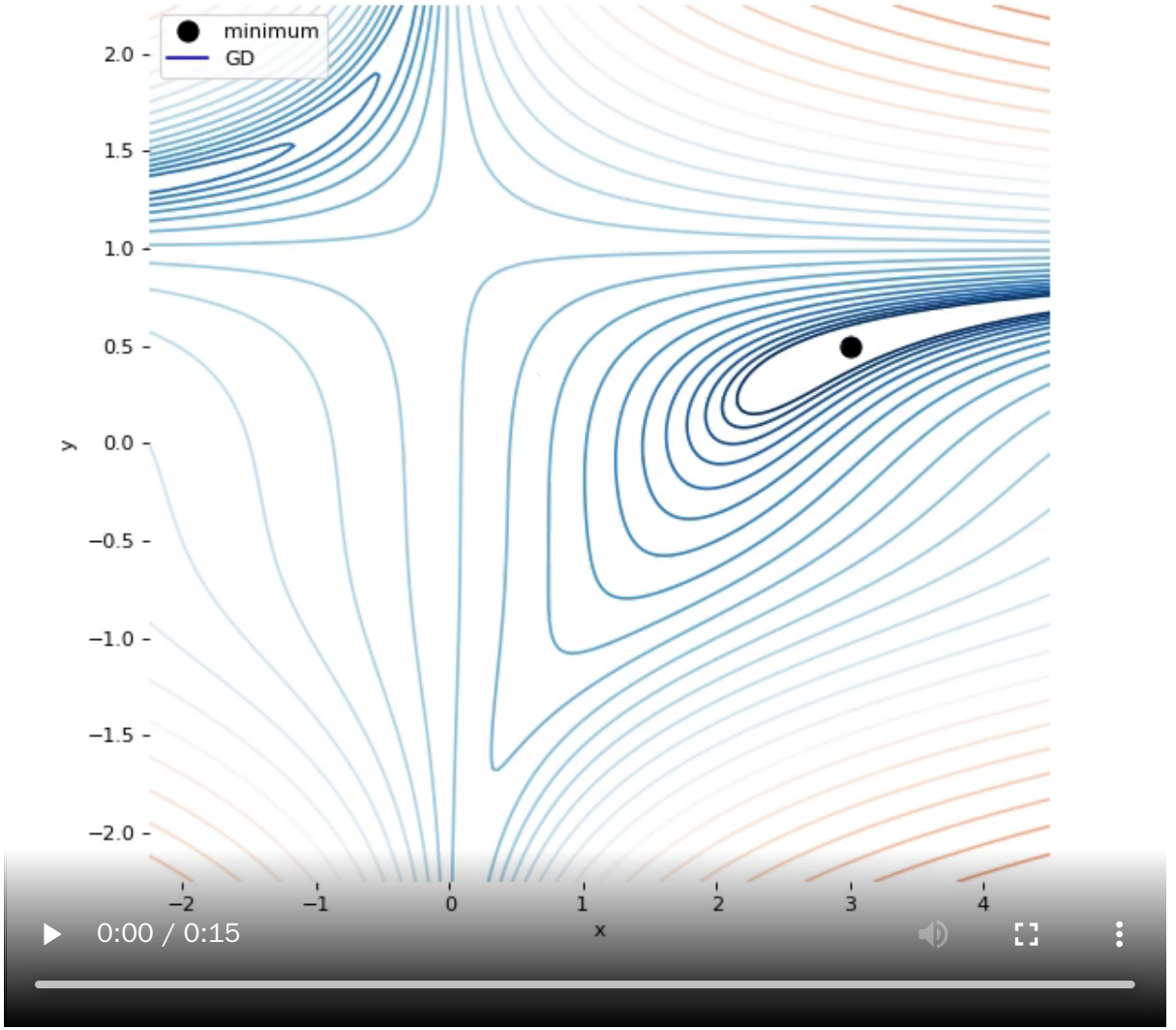

—

Slide source: Deep Learning in Computer Vision




—

Slide source: Deep Learning in Computer Vision


minimum
GD
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
−2
−1
0
1
2
3
4
x
0:00 / 0:15

# Stochastic Gradient Descent

SGD

# SGD

To reduce the computational complexity, stochastic gradient descent (SGD) consists in updating the parameters after every sample

$$\begin{aligned} \ell^{(i)} &= \mathcal{L}\left(y^{(i)}, f(\mathbf{x}^{(i)}, W)\right) \\ g_t^{(i)} &= \nabla_W \ell^{(i)} \\ W_{t+1} &= W_t - \alpha g_t^{(i)} \end{aligned}$$

# Advantages of SGD

- Stochastic gradients are drastically cheaper to compute (proportional to your dataset size), so you can often take thousands of SGD steps for the cost of one GD step.
- On average, the stochastic gradient is a good estimate of the gradient.
- The noise can prevent the optimizing converging to bad local minima, a phenomena called annealing.

—
Credits: Aaron Defazio, Facebook AI Research

# Mini-batching

# Mini-batching

Visiting the samples in mini-batches and updating the parameters each time

$$g_t^{(k)} = \frac{1}{B} \sum_{i=1}^{B} \nabla_W \mathcal{L} \left( y_k^{(i)}, f(\mathbf{x}_k^{(i)}, W) \right)$$
$$W_{t+1} = W_t - \alpha g_t^{(k)},$$

where $k$ is an index of mini-batch.

- Increasing the batch size $B$ reduces the variance of the gradient estimates and enables the speed-up of batch processing.
- The interplay between $B$ and $\alpha$ is still unclear.

# Momentum

# Momentum

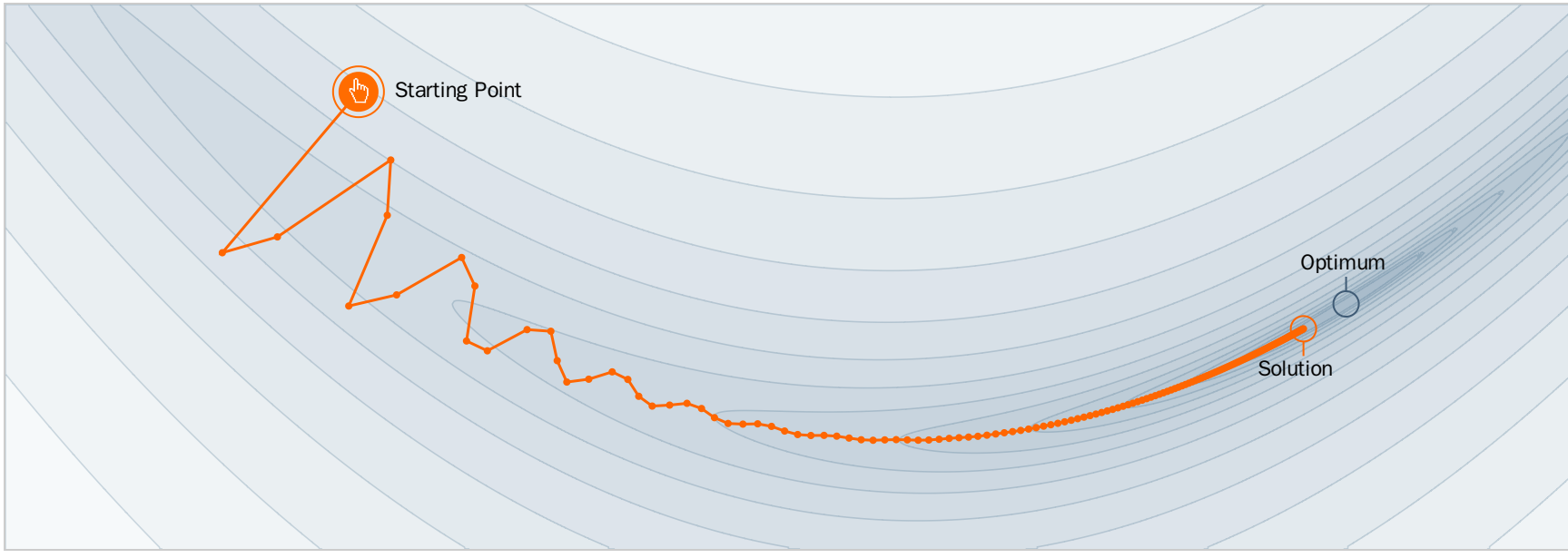
SGD + Momentum = Stochastic heavy ball method

$$p_{t+1} = \beta_t p_t + \nabla \ell^{(i)}(W_t)$$
$$W_{t+1} = W_t - \alpha_t p_{t+1}$$

Rule for updating:

$$W_{t+1} = W_t - \alpha_t \nabla \ell^{(i)}(W_t) + \beta_t (W_t - W_{t-1})$$

Key idea: The next step becomes a combination of the previous step's direction and the new negative gradient.




Step-size α = 0.02

0 0.003 0.006

Momentum β = 0.99

0.00 0.500 0.990

We often think of Momentum as a means of dampening oscil
speeding up the iterations, leading to faster convergence. But
interesting behavior. It allows a larger range of step-sizes to b
creates its own oscillations. What is going on?

—

Credits: Distill, Why Momentum Really Works?


—

Image credits: Kosta Derpanis, Deep Learning in Computer Vision, 2018

# Advantages

SGD with momentum has three nice properties:

- it can go through local barriers
- it accelerates if the gradient does not change much
- it dampens oscillations in narrow valleys

## Practical Aspects of momentum

It's basically "free lunch", in almost all situations, SGD + momentum is better than SGD, and very rarely worse!

## Recommended Parameters:

$\beta = 0.9$ or $0.99$ almost always works well. Sometimes slight gains can be had by tuning it.

The step size parameter ($\alpha$) usually needs to be decreased when the momentum parameter is increased to maintain convergence.

—
Credits: Aaron Defazio, Facebook AI Research

# Demo

# Adaptive methods

# Adaptive methods

The magnitude of the gradients often varies highly between layers due to, so a global learning rate may not work well.

General IDEA: Instead of using the same learning rate for every weight in our network, maintain an estimate of a better rate separately for each weight.

The exact way of adapting to the learning rates varies between algorithms, but most methods either adapt to the variance of the weights, or to the local curvature of the problem.

—
Credits: Aaron Defazio, Facebook AI Research

## AdaGrad

Per-parameter downscale by square-root of sum of squares of all its historical values.

$$r_t = r_{t-1} + g_t \odot g_t$$
$$W_{t+1} = W_t - \frac{\alpha}{\varepsilon + \sqrt{r_t}} \odot g_t$$

- AdaGrad eliminates the need to manually tune the learning rate. Most implementation use $\alpha = 0.01$ as default.
- It is good when the objective is convex.
- $r_t$ grows unboundedly during training, which may cause the step size to shrink and eventually become infinitesimally small.
- $\delta$ is an additive constant that ensures that we do not divide by 0.

## RMSProp

Same as AdaGrad but accumulate an exponentially decaying average of the gradient.

Key IDEA: normalize by the root-mean-square of the gradient

$$r_t = \rho r_{t-1} + (1-\rho) g_t \odot g_t$$
$$W_{t+1} = W_t - \frac{\alpha}{\varepsilon + \sqrt{r_t}} \odot g_t$$

- Perform better in non-convex settings.

## Adam: RMSprop with a kind of momen

"Adaptive Moment Estimation"

Similar to RMSProp with momentum, but with bias correction terms for the first and second moments.

$$
\begin{aligned}
p_t &= \rho_1 p_{t-1} + (1-\rho_1) g_t \\
\hat{p}_t &= \frac{p_t}{1-\rho_1^t} \\
r_t &= \rho_2 r_{t-1} + (1-\rho_2) g_t \odot g_t \\
\hat{r}_t &= \frac{r_t}{1-\rho_2^t} \\
W_{t+1} &= W_t - \alpha \frac{\hat{p}_t}{\varepsilon + \sqrt{\hat{r}_t}}
\end{aligned}
$$

- Good defaults are $\rho_1 = 0.9$ and $\rho_2 = 0.999$.
- Just as momentum improves SGD, it improves RMSProp as well.

# Practical side

For poorly conditioned problems, Adam is often much better than SGD.

Use Adam over RMSprop due to the clear advantages of momentum.

**BUT, Adam is poorly understood theoretically, and has known disadvantages:**

- Does not converge at all on some simple example problems!
- Gives worse generalization error on many computer vision problems (i.e. ImageNet)
- Requires more memory than SGD
- Has 2 momentum parameters, so some tuning may be needed

—

Credits: Aaron Defazio, Facebook AI Research

# Demo

# demo - losslandscape

# The end