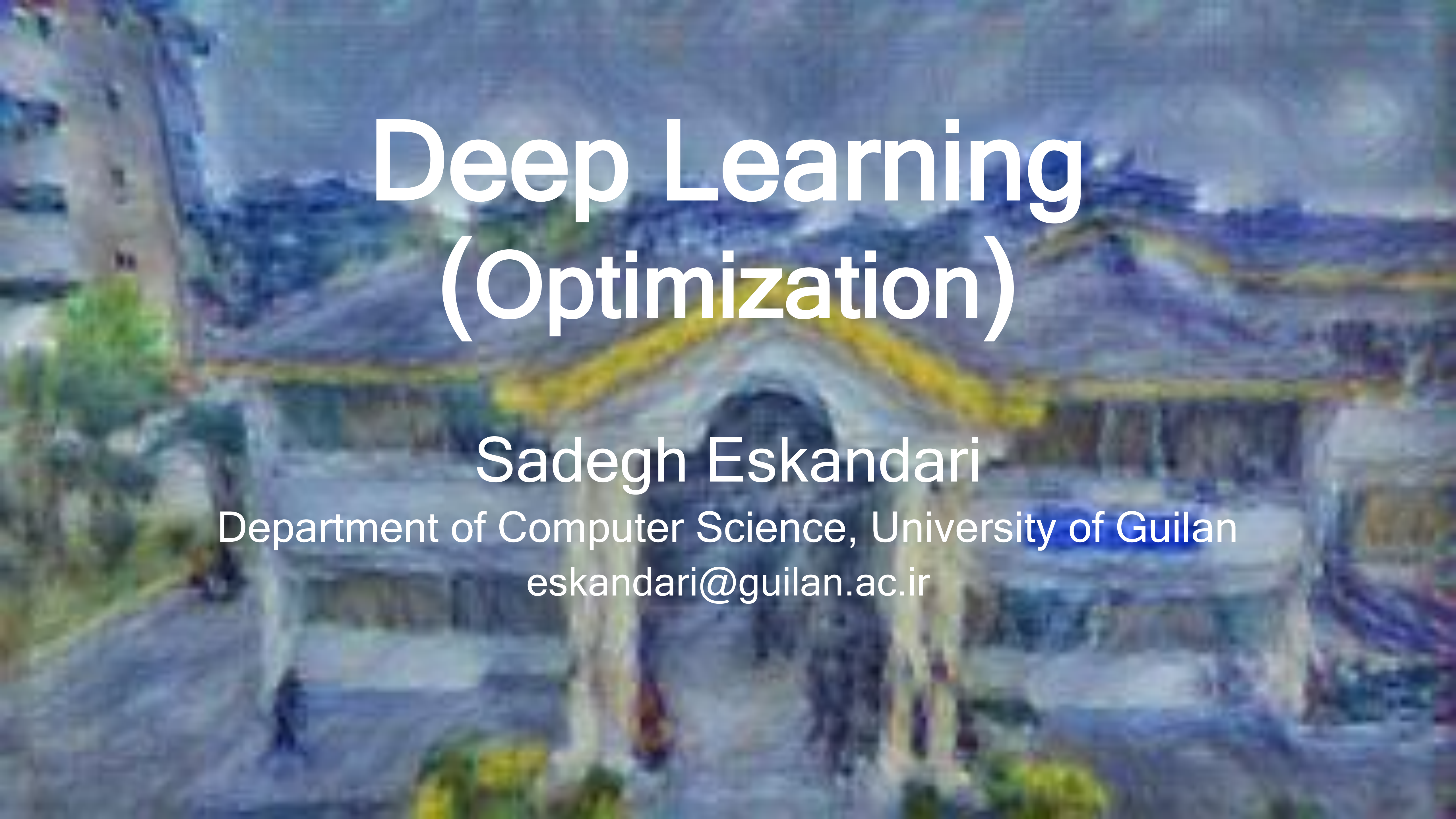


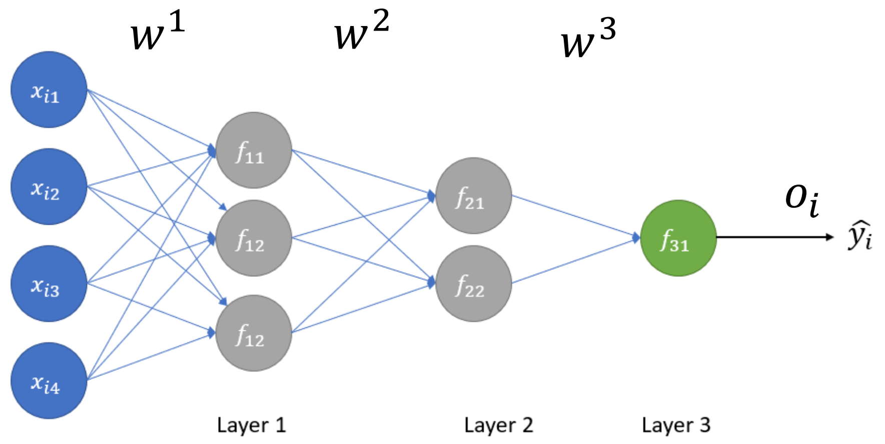
Deep Learning (Optimization)

Sadegh Eskandari
Department of Computer Science, University of Guilan
eskandari@guilan.ac.ir

Today …

- Intro to Optimization
- Gradient Descent
- Hessian Matrix
- Convex Optimization
- Optimization For Deep Learning

Recall …



$$\mathbf{X} = (\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_N})^{\boldsymbol{T}}$$

$$\mathbf{x_i} = (x_{i1}, x_{i2}, \dots, x_{in})^{\boldsymbol{T}}$$

$$\mathbf{y} = (y_1, y_2, \dots, y_N)^{\boldsymbol{T}}$$

Training data

N: num of data

n: dimension of each data point

Step 1: Loss function

$$l_i = (y_i - o_i)^2 = \left(y_i - \varphi(\boldsymbol{w}^T \boldsymbol{x}_i)\right)^2$$

$$L = \sum_{i=1}^{N} l_i$$

Step 2: The objective function

$$\arg \min_{w^1, w^2, w^3} \boxed{L}$$

Step 3: Optimization

```
Initialize $w_{ij}^k$
for $iter = 1$ to $K$:
    for each i, $j, k$:
        $(w_{ij}^k)_{new} = (w_{ij}^k)_{old} - \gamma \boxed{\frac{\partial L}{\partial w_{ij}^k}}$
```

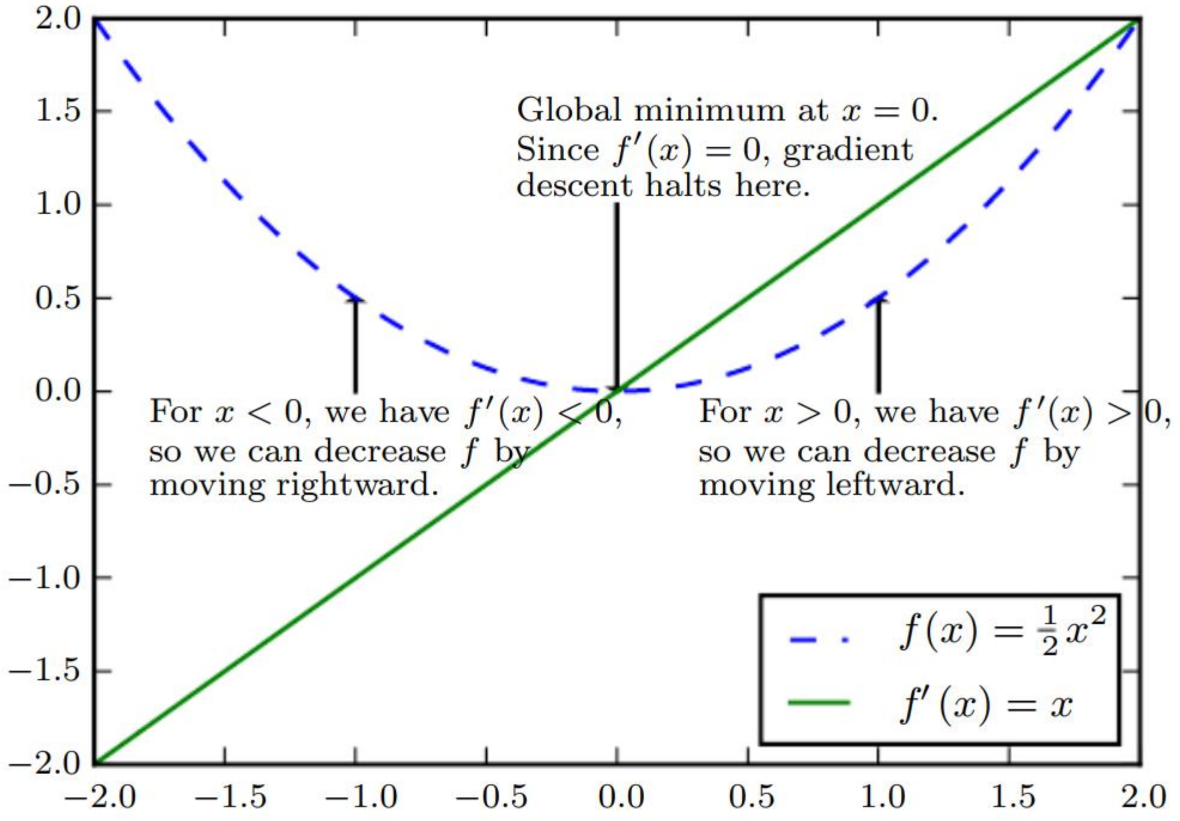
How to calculate L and how to update w_{ij}^k?

Intro to Optimization

- Optimization refers to the task of either minimizing or maximizing some function $f(x)$ by altering x.
- We usually phrase most optimization problems in terms of minimizing $f(x)$.
  - Maximization may be accomplished via a minimization algorithm by minimizing $-f(x)$.
- The function we want to minimize or maximize is called the objective function , or criterion.
  - When we are minimizing it, we may also call it the cost function , loss function, or error function.
- We often denote the value that minimizes or maximizes a function with a superscript $*$. For example, we might say $x^* = \arg\min f(x)$.
- The derivative $f'(x)$ gives the slope of $f(x)$ at the point x
  - It specifies how to scale a small change in the input to obtain the corresponding change in the output: $f(x + \epsilon) \approx f(x) + \epsilon f'(x)$.
  - Therefore, it is useful for minimizing a function because it tells us how to change x in order to make a small improvement in y.

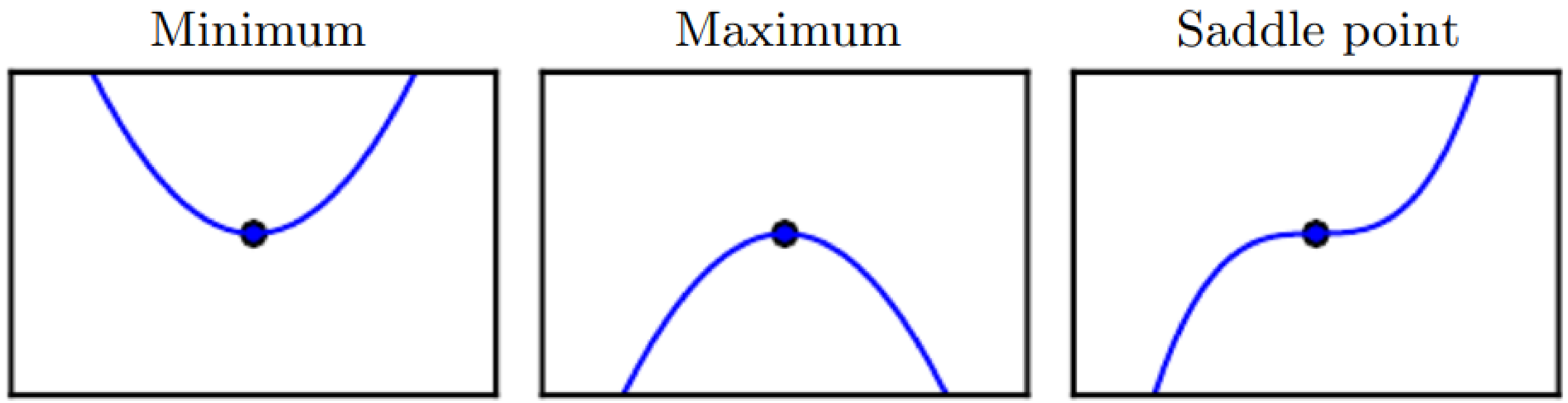
Gradient Descent

- We can reduce $f(x)$ by moving x in small steps with the opposite sign of the derivative.
  - This technique is called gradient descent (Cauchy, 1847).



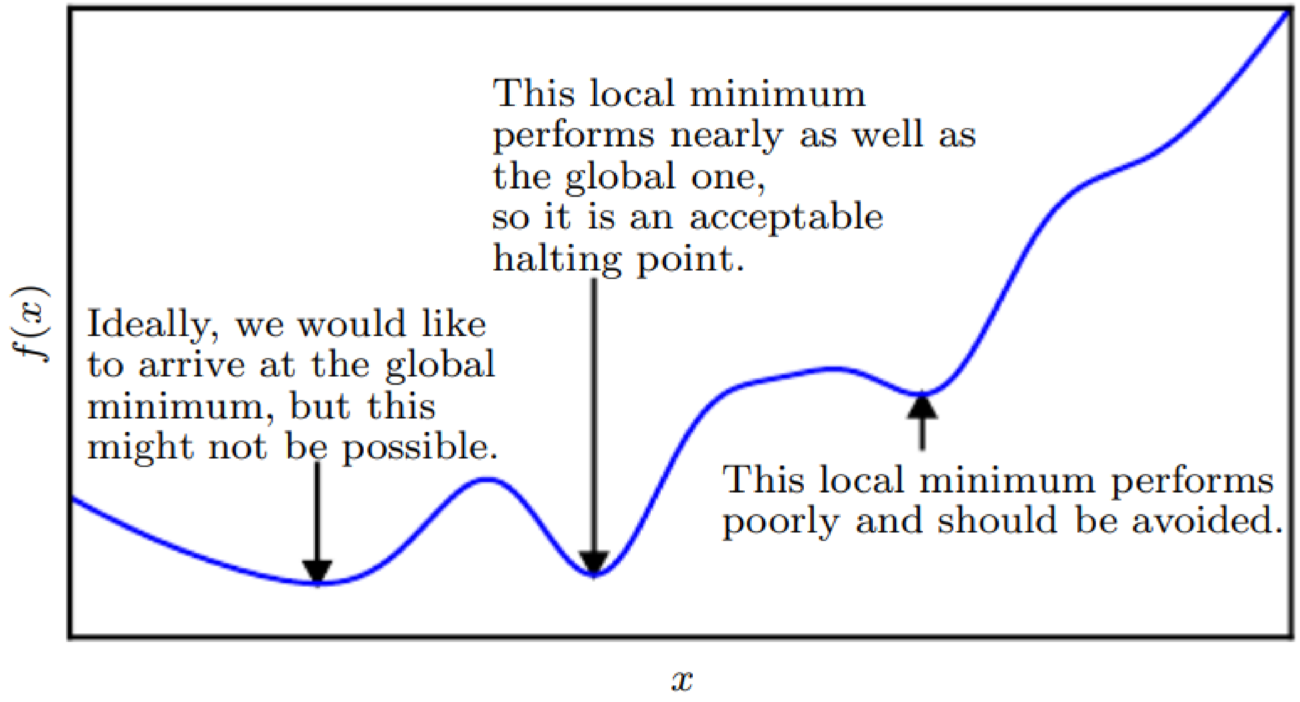
Gradient Descent: Critical Points

- When $f'(x) = 0$, the derivative provides no information about which direction to move.
- Points where $f'(x) = 0$ are known as critical points, or stationary points.



Gradient Descent: Approximate Minimization

- **Approximate minimization**: Optimization algorithms may fail to find a global minimum when there are multiple local minima or plateaus present.
- In the context of deep learning, we generally accept such solutions even though they are not truly minimal



Gradient Descent: Multivariate Functions

- For functions with multiple inputs, we must make use of the concept of partial derivatives.
- The partial derivative $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$ measures how f changes as only the variable x_i increases at point $\boldsymbol{x}$.
- The gradient generalizes the notion of derivative to the case where the derivative is with respect to a vector
  - the gradient of f is the vector containing all the partial derivatives, denoted $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$.

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_D}\right]^T$$

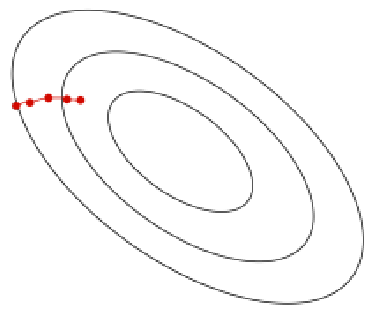
- Therefore:

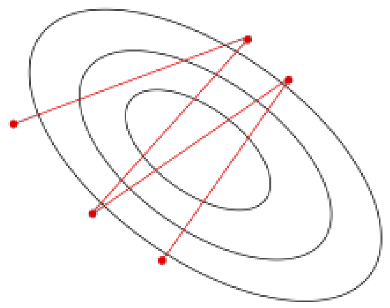
$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\mathbf{x}} f(\boldsymbol{x}) = \left[x_1 - \epsilon \frac{\partial}{\partial x_1} f(\boldsymbol{x}),\ x_2 - \epsilon \frac{\partial}{\partial x_2} f(\boldsymbol{x}), \dots, x_D - \epsilon \frac{\partial}{\partial x_D} f(\boldsymbol{x})\right]^{\boldsymbol{T}}$$

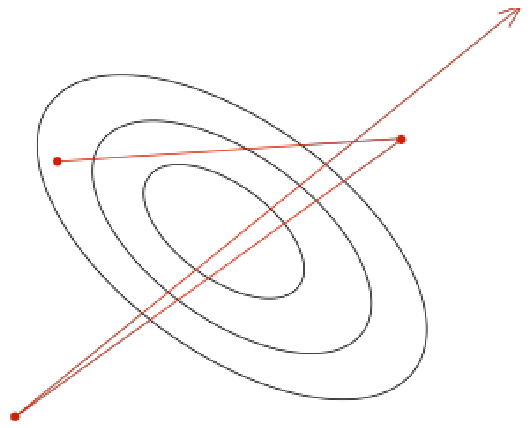
Gradient Descent: Learning Rate

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

- The learning rate ϵ is a hyperparameter we need to tune:


ϵ too small:
Slow progress


ϵ too large:
Oscillations


ϵ much too large:
Instability

Source: Neural Networks and Deep Learning course by Jimmy Ba, 2020, University of Toronto: https://csc413-2020.github.io/

- Good values are typically between 0.001 and 0.1. You should do a grid search if you want good performance (i.e. try 0.1, 0.03, 0.01, …).

Jacobian Matrix

- Sometimes we need to find all the partial derivatives of a function whose input and output are both vectors.
- The matrix containing all such partial derivatives is known as a Jacobian matrix.
- If we have a function $\boldsymbol{f} : \mathbb{R}^m \rightarrow \mathbb{R}^n$, then the Jacobian matrix $\mathbf{J} \in \mathbb{R}_{n \times m}$ of $\boldsymbol{f}$ is defined such that $\mathbf{J}_{i,j} = \frac{\partial}{\partial x_j} \boldsymbol{f}(\boldsymbol{x})_i$.

Hessian Matrix

Eigenvectors and Eigenvalues

- For a square matrix $\mathbf{A}$ of size $M \times M$, the eigenvector equation is defined by

$$\boxed{\mathbf{Au}_\mathrm{i} = \lambda_i \boldsymbol{u_i}} \quad , i = 1, \dots, M$$

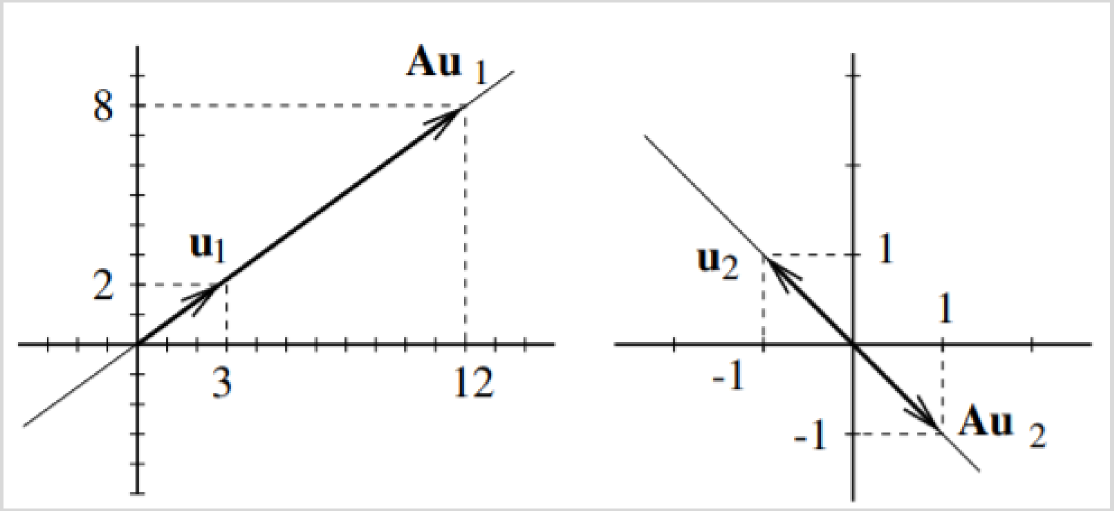
u_i: Eigenvector $\qquad$ λ_i: Eigenvalue

- Characteristic Equation:

$$|\mathbf{A} - \lambda_i \mathbf{I}| = 0$$

- Example:

$$\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \left\{ \begin{array}{l} \mathbf{u_1} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, \lambda_1 = 4 \\ \\ \mathbf{u_2} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \lambda_1 = -1 \end{array} \right.$$



Hessian Matrix

Eigenvectors and Eigenvalues

- For most applications we normalize the eigenvectors (i.e., transform them such that their length is equal to one)

$$\mathbf{u}_i\mathbf{u}_i^T = 1$$

- To normalize, we simply divide $\mathbf{u}_i$ by its length $|\mathbf{u}_i|$

- Example:

Normalized eigenvectors

$$\mathbf{A} = \begin{bmatrix}2 & 3\\ 2 & 1\end{bmatrix} \begin{cases} \mathbf{u_1} = \begin{bmatrix}3\\ 2\end{bmatrix}, \lambda_1 = 4 \xrightarrow{|\mathbf{u_1}| = \sqrt{3^2 + 2^2} = \sqrt{13}} \mathbf{u_1} = \begin{bmatrix}3/\sqrt{13}\\ 2/\sqrt{13}\end{bmatrix} = \begin{bmatrix}0.8331\\ 0.5547\end{bmatrix} \\ \mathbf{u_2} = \begin{bmatrix}-1\\ 1\end{bmatrix}, \lambda_1 = -1 \xrightarrow{|\mathbf{u_2}| = \sqrt{-1^2 + 1^2} = \sqrt{2}} \mathbf{u_2} = \begin{bmatrix}-1/\sqrt{2}\\ 1/\sqrt{2}\end{bmatrix} = \begin{bmatrix}-0.7071\\ 0.7071\end{bmatrix} \end{cases}$$

Hessian Matrix

Eigenvectors and Eigenvalues

- We can re-write the eigenvector equation in matrix form:

$$\begin{tikzcd}
\boxed{\mathbf{A}\mathbf{u}_{\mathrm{i}} = \lambda_i \boldsymbol{u}_i} \arrow[r, "\text{Matrix Form}"] & \boxed{\mathbf{A}\mathbf{U} = \mathbf{U}\boldsymbol{\Lambda}} \arrow[r, "\text{Eigen Decomposition}"] \arrow[dr, "\text{Diagonalization}"'] & \boxed{\mathbf{A} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{-1}} \\
 & & \boxed{\mathbf{U}^{-1}\mathbf{A}\mathbf{U} = \boldsymbol{\Lambda}}
\end{tikzcd}$$

$$\mathbf{U} = \begin{pmatrix} \boxed{\mathbf{u}_1} & \cdots & \boxed{\mathbf{u}_M} \end{pmatrix} \qquad \boldsymbol{\Lambda} = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_M \end{pmatrix}$$

Hessian Matrix

Eigenvectors and Eigenvalues

- If **A** is a real symmetric matrix, then its eigenvalues are real and can be chosen to form orthonormal set, so that

$$\mathbf{u}_i^T\mathbf{u}_j = \begin{cases} 1 & ,\text{if } i = j \\ 0 & ,\text{otherwise} \end{cases}$$

- Or

$$\mathbf{U}^T\mathbf{U} = \mathbf{I} \quad \Rightarrow \quad \mathbf{U}^{\mathrm{T}}\mathbf{U}\mathbf{U}^{-1} = \overbrace{\mathbf{U}^{-1} = \mathbf{U}^{\mathrm{T}}}^{\text{☺}}$$

- Then

$$\begin{tikzcd} & \boxed{\mathbf{A} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{-1} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{\mathrm{T}}} \\ \boxed{\mathbf{AU} = \mathbf{U}\boldsymbol{\Lambda}} \arrow[ur, "\text{Eigen Decomposition}"] \arrow[dr, "\text{Diagonalization}"'] & \\ & \boxed{\mathbf{U}^{-1}\mathbf{AU} = \mathbf{U}^{\mathrm{T}}\mathbf{AU} = \boldsymbol{\Lambda}} \end{tikzcd}$$

Hessian Matrix

- When f has multiple input dimensions, there are many second derivatives. These derivatives can be collected together into a matrix called the Hessian matrix.

$$\boldsymbol{H}(f)(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x})$$

$$\boldsymbol{H} = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_D \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_D^2} \end{pmatrix}$$

- **H** is symmetric because $\frac{\partial^2 f}{\partial \theta_i \partial \theta_j} = \frac{\partial^2 f}{\partial \theta_j \partial \theta_i}$

Hessian Matrix

- Because the Hessian matrix is real and symmetric, we can decompose it into a set of real eigenvalues and an orthogonal basis of eigenvectors.

$$\mathbf{H} = \mathbf{U\Lambda U^{-1}} = \mathbf{U\Lambda U^{T}}$$

$$\mathbf{U^{-1}HU} = \mathbf{U^{T}HU} = \mathbf{\Lambda}$$

- The second derivative in a specific direction represented by a unit vector d is given by $\mathrm{d}^{\mathrm{T}}\mathbf{H}\mathrm{d}$.
  - When d is an eigenvector of **H** , the second derivative in that direction is given by the corresponding eigenvalue.
  - For other directions of d , the directional second derivative is a weighted average of all the eigenvalues, with weights between 0 and 1, and eigenvectors that have a smaller angle with d receiving more weight.

Hessian Matrix

Note: Second order approximation of function $f(x)$ around point a:

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{1}{2}f''(a)(x-a)^2$$

- We can make a second-order Taylor series approximation to the function $f(x)$ around the current point $x^{(0)}$ ($\boldsymbol{g}$: gradient, $\boldsymbol{H}$: hessian):

$$f(\boldsymbol{x}) \approx f\left(\boldsymbol{x}^{(0)}\right) + \left(\boldsymbol{x} - \boldsymbol{x}^{(0)}\right)^T \boldsymbol{g} + \frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{x}^{(0)}\right)^T \mathbf{H}\left(\boldsymbol{x} - \boldsymbol{x}^{(0)}\right)$$

- Using gradient descent method, If we use a learning rate of ϵ, then the new point $\boldsymbol{x}$ will be given by $\boldsymbol{x}^{(0)} - \epsilon\boldsymbol{g}$.

$$f\left(\boldsymbol{x}^{(0)} - \epsilon\boldsymbol{g}\right) \approx f\left(\boldsymbol{x}^{(0)}\right) + \left(\boldsymbol{x}^{(0)} - \epsilon\boldsymbol{g} - \boldsymbol{x}^{(0)}\right)^T \boldsymbol{g} + \frac{1}{2}\left(\boldsymbol{x}^{(0)} - \epsilon\boldsymbol{g} - \boldsymbol{x}^{(0)}\right)^T \mathbf{H}\left(\boldsymbol{x}^{(0)} - \epsilon\boldsymbol{g} - \boldsymbol{x}^{(0)}\right)$$

$$= f\left(\boldsymbol{x}^{(0)}\right) - \epsilon\boldsymbol{g}^T\boldsymbol{g} + \frac{1}{2}\epsilon^2\boldsymbol{g}^{\mathrm{T}}\mathbf{H}\boldsymbol{g}$$

Hessian Matrix

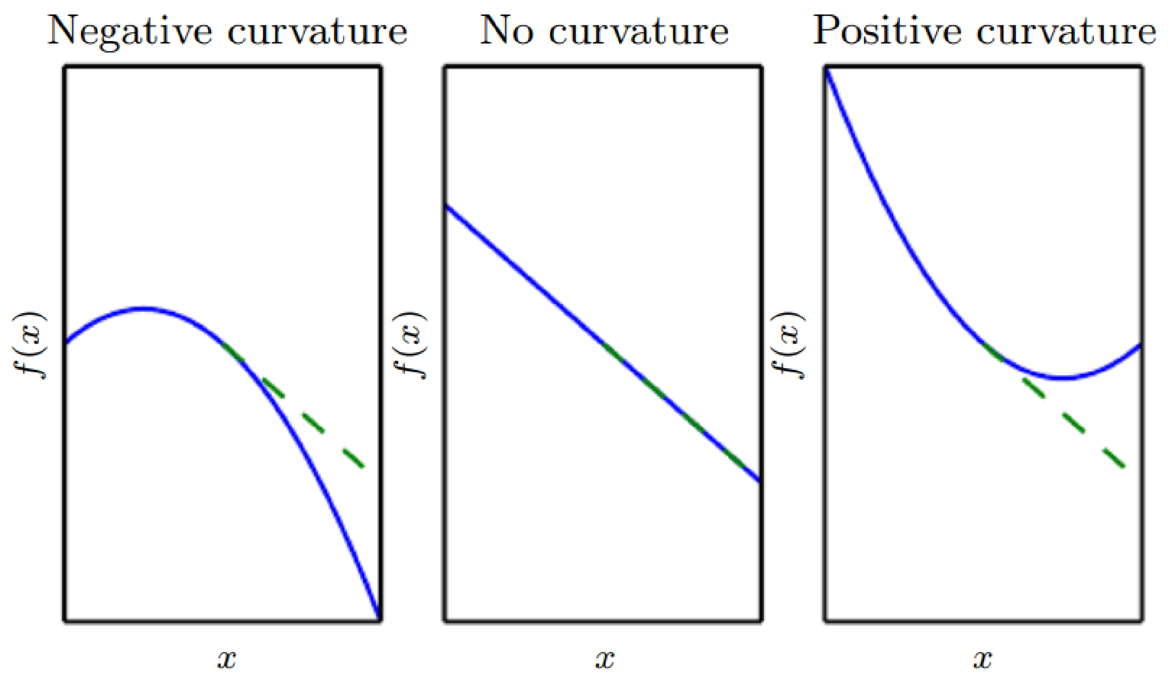
$$f\left(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}\right) \approx f\left(\boldsymbol{x}^{(0)}\right) - \epsilon \boldsymbol{g}^T \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^{\mathrm{T}} \mathbf{H} \boldsymbol{g}$$

- We can think of the $\mathbf{H}$ as measuring curvature.

- Negative curvature: the cost function decreases faster than the gradient predicts.

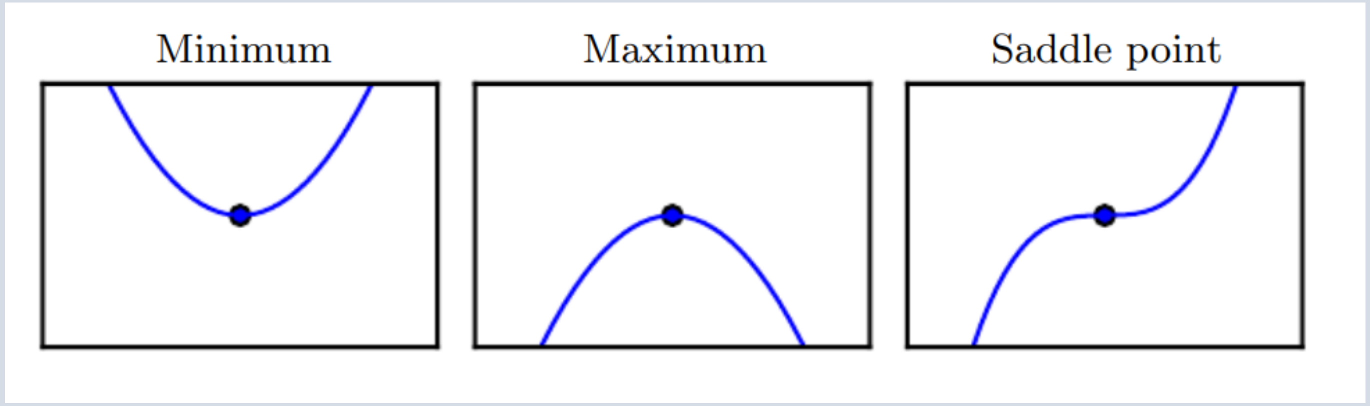
- No curvature: the gradient predicts the decrease correctly.

- Positive curvature: the function decreases more slowly than expected



Hessian Matrix

Remember:



Critical Points:
$f'(x) = 0$

- Second derivative test:

$f'(x) = 0$ and $f''(x) > 0$ → x is a local minimum

$f'(x) = 0$ and $f''(x) < 0$ → x is a local maximum

$f'(x) = 0$ and $f''(x) = 0$ → Inconclusive, may be a saddle point or a part of a flat region

Hessian Matrix

Second derivative test (multiple dimensions):

- Critical points: $\nabla_x f(\boldsymbol{x}) = \mathbf{0}$
- A matrix $\mathbf{A}$ is called positive definite (PD) if its eigenvalues are strictly positive.
- A matrix $\mathbf{A}$ is called negative definite (ND) if its eigenvalues are strictly negative.

$\boxed{\nabla_x f(\boldsymbol{x}) = \mathbf{0} \text{ and } \mathbf{H} \text{ is PD}} \longrightarrow \boxed{x \text{ is a local minimum}}$

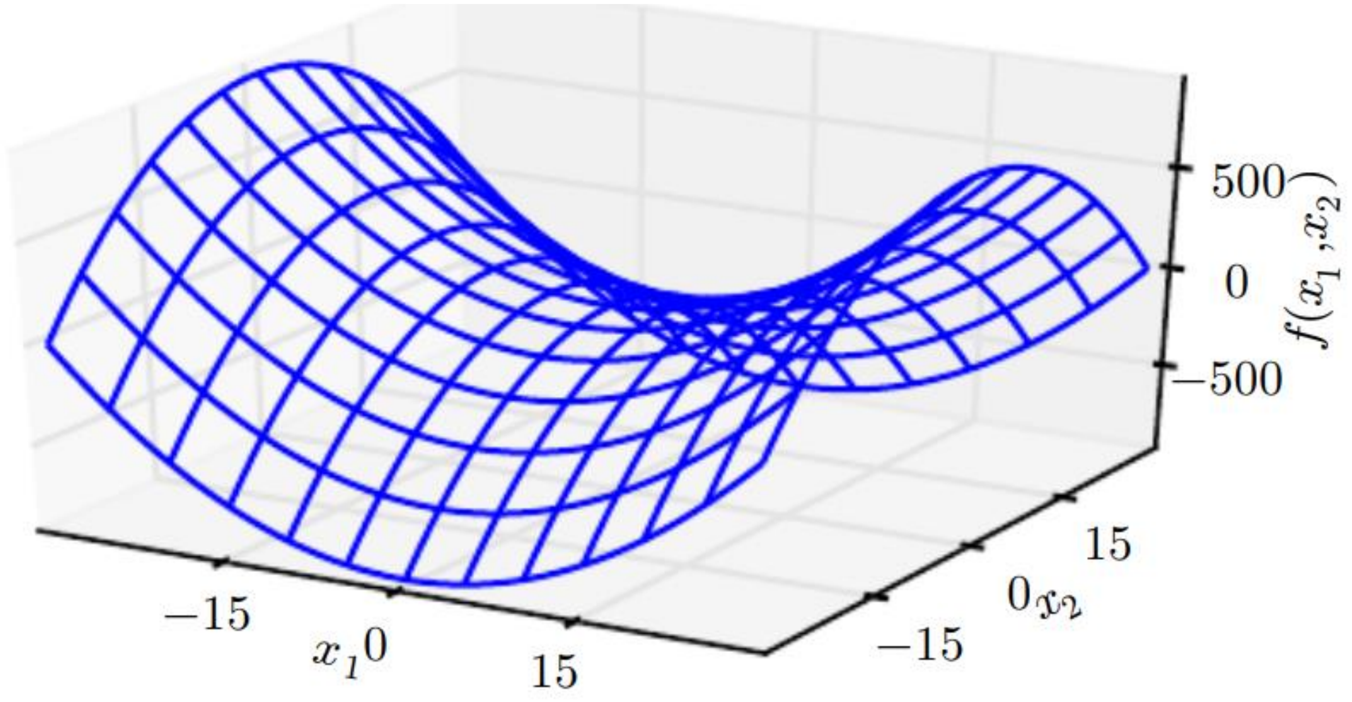
$\boxed{\nabla_x f(\boldsymbol{x}) = \mathbf{0} \text{ and } \mathbf{H} \text{ is ND}} \longrightarrow \boxed{x \text{ is a local maximum}}$

$\boxed{\nabla_x f(\boldsymbol{x}) = \mathbf{0} \text{ and } \mathbf{H} \text{ is not PD or ND}} \longrightarrow \boxed{\text{Inconclusive, may be a saddle point or a part of a flat region}}$

Example: next slide

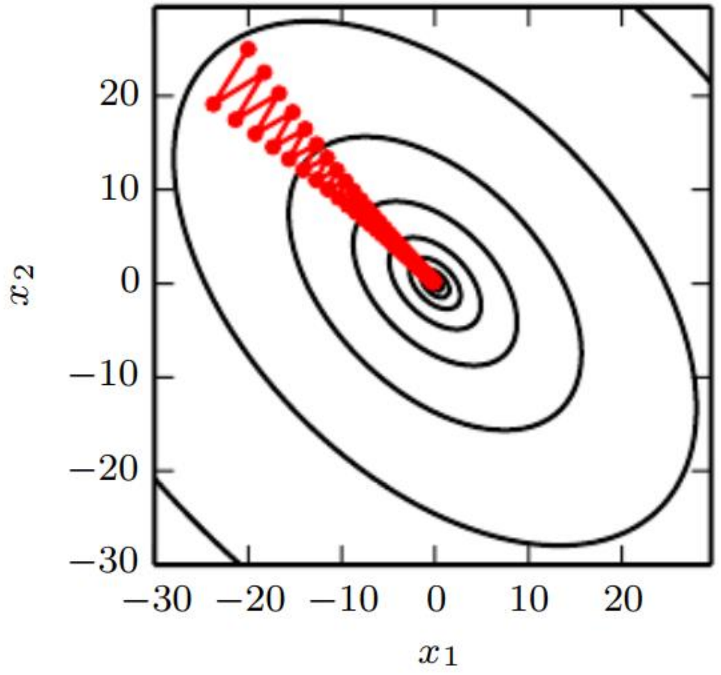
Hessian Matrix

$f'(\boldsymbol{x}) = \mathbf{0}$ and $\mathbf{H}$ is not PD or ND $\longrightarrow$ Inconclusive, may be a saddle point or a part of a flat region



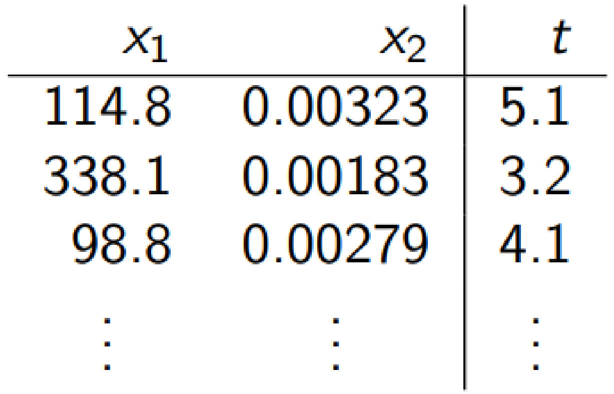
Hessian Matrix

- The quantity $\frac{\lambda_{\max}}{\lambda_{\min}}$ is known as the condition number of a matrix $\boldsymbol{A}$.
- When the Hessian has a large condition number (Ill-conditioned), gradient descent performs poorly.
  - This is because in one direction, the derivative increases rapidly, while in another direction, it increases slowly.
- Poor condition number also makes choosing a good step size (ϵ) difficult.
  - The step size must be too small to avoid overshooting the minimum and going uphill in directions with strong positive curvature.



Hessian Matrix

- Suppose we have the following dataset for linear regression:



x_1	x_2	t
114.8	0.00323	5.1
338.1	0.00183	3.2
98.8	0.00279	4.1
⋮	⋮	⋮

Source: Neural Networks and Deep Learning course by Jimmy Ba, 2020, University of Toronto: https://csc413-2020.github.io/

- Which weight, x_1 or x_2, will receive a larger gradient descent update?

Hessian Matrix

- To avoid these problems, it’s a good idea to center your inputs to zero mean and unit variance, especially when they’re in arbitrary units

$$\hat{x}_j = \frac{x_j - \mu_j}{\sigma_j}$$

Hessian Matrix

- The Ill-conditioned issue can be resolved by using information from the Hessian matrix to guide the search.
- The simplest method for doing so is known as Newton's method:

$$\boldsymbol{x}' = \boldsymbol{x} - \boldsymbol{H}(x)^{-1}\nabla_{\mathbf{x}} f(\boldsymbol{x})$$

- Optimization algorithms that use only the gradient, such as gradient descent, are called first-order optimization algorithms.
- Optimization algorithms that also use the Hessian matrix, such as Newton's method, are called second-order optimization algorithms

Convex Optimization

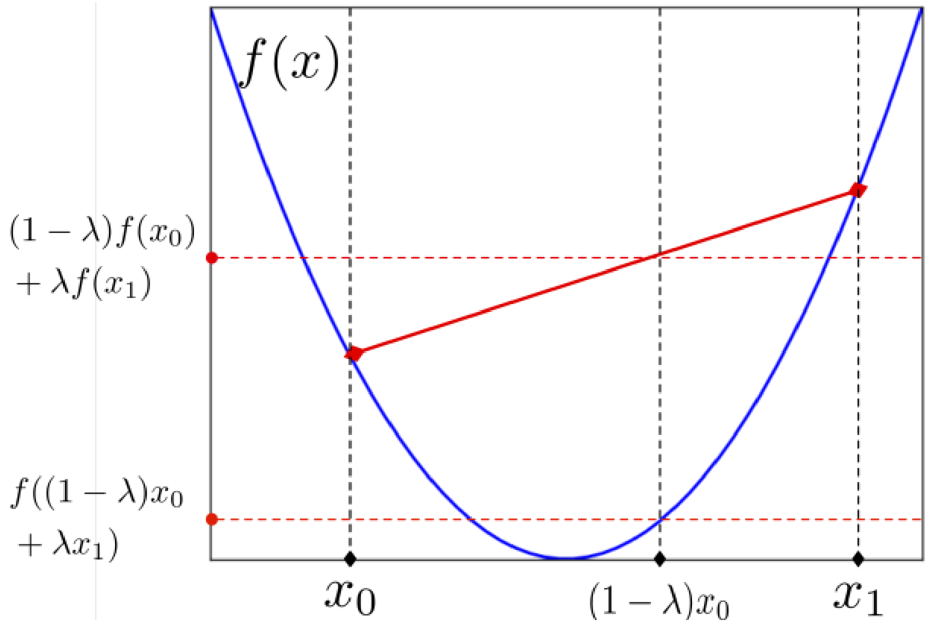
- A set S is convex, if for any $x_0, x_1 \in S$,

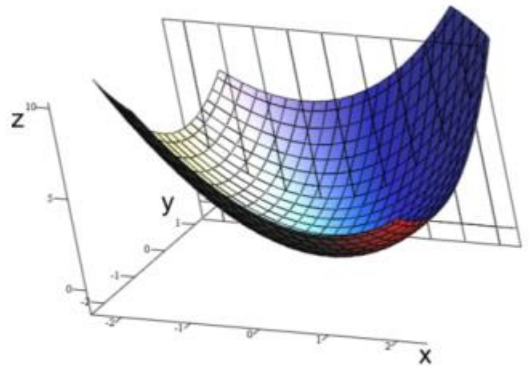
$$(1-\lambda)x_0 + \lambda x_1 \in S, \quad \text{for } 0 \le \lambda \le 1.$$

- A function f is convex, if for any $x_0, x_1 \in S$,

$$f\big((1-\lambda)x_0 + \lambda x_1\big) \le (1-\lambda)f(x_0) + \lambda f(x_1), \quad \text{for } 0 \le \lambda \le 1.$$

Source: Neural Networks and Deep Learning course by Jimmy Ba, 2020, University of Toronto: https://csc413-2020.github.io/



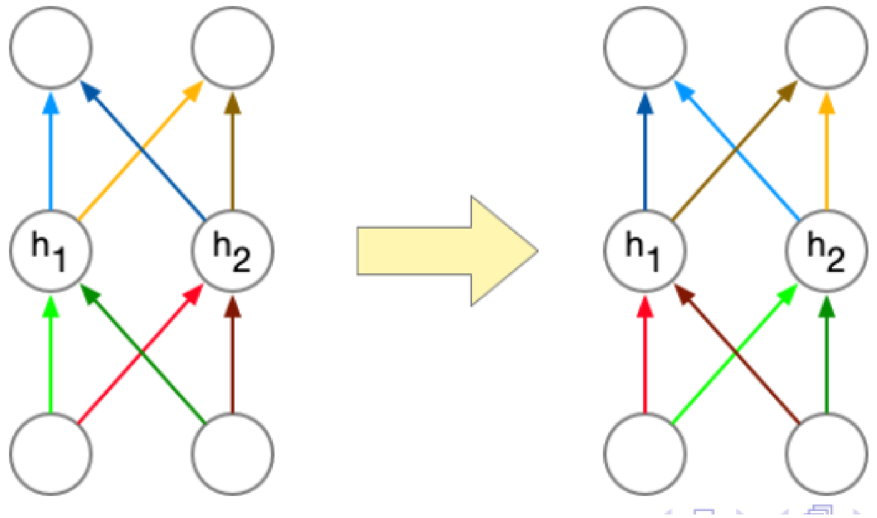


Convex Optimization

- If a function is convex, any local minimum is also a global minimum.
- This is very convenient for optimization since if we keep going downhill, we’ll eventually reach a global minimum.

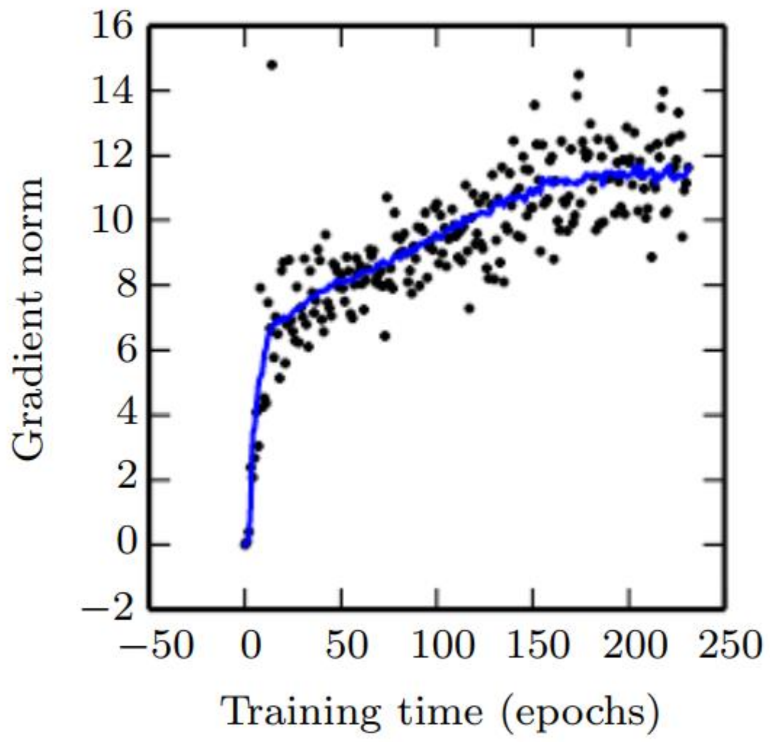
Optimization for Training Deep Models

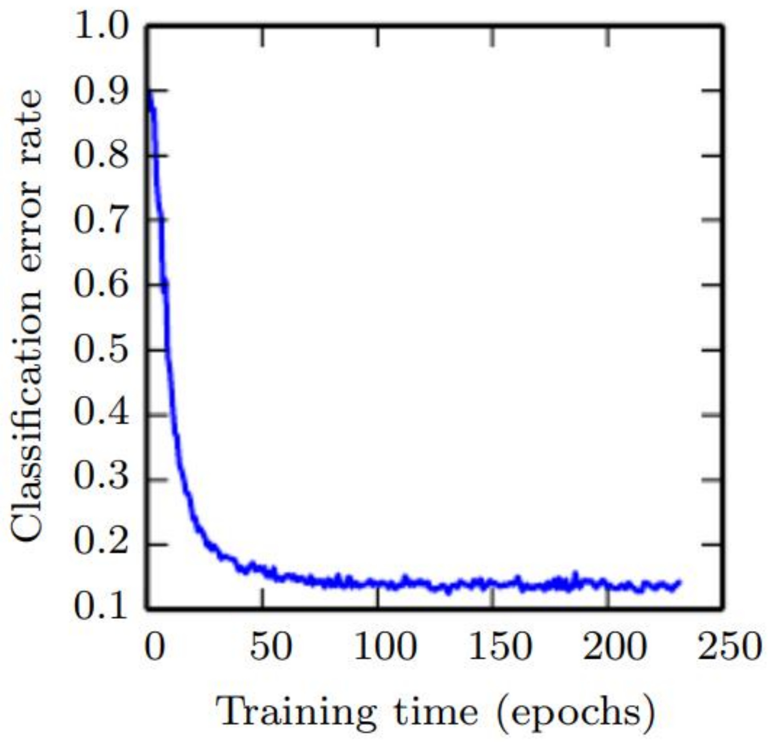
- Training multilayer neural nets is non-convex ☹☹☹
- Why?
- Permutation symmetry or weight space symmetry
- Consider that a given network is optimized using parameters W^*
- If the loss function was convex, then $\text{Loss}(W^*)$ would be the global minimum value
- Suppose we exchange the parameters of two nodes in the same layer
- Then the value computed by the network will be preserved
- If the function were convex, this value would have to be larger that $\text{Loss}(W^*)$



Optimization for Training Deep Models

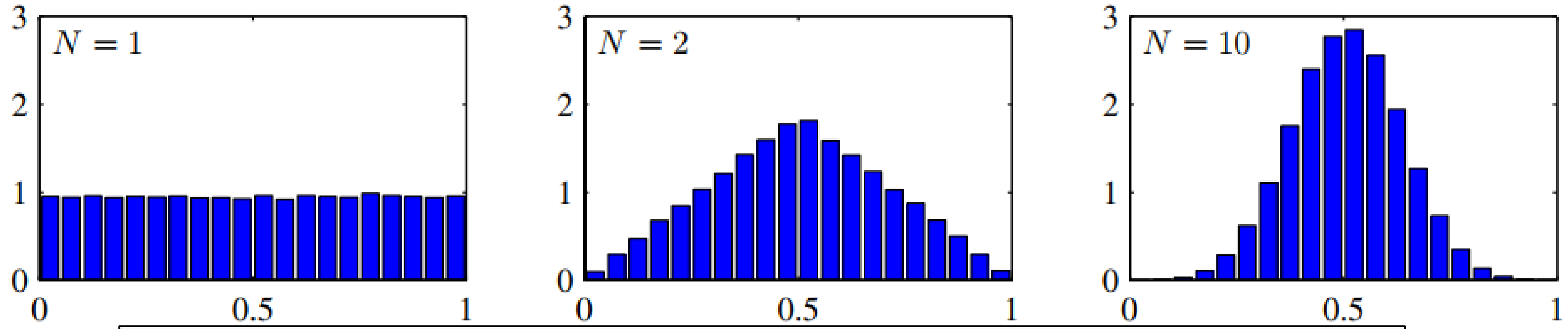
- Ill-Conditioning problem is very common in neural network training
- Ill-conditioning can manifest by causing SGD to get “stuck” in the sense that even very small steps increase the cost function.
- Therefore, gradient descent often does not arrive at a critical point of any kind





Optimization for Training Deep Models

- **The Central Limit Theorem**: Mean of a set of random variables, which is of course itself a random variable, has a distribution that becomes increasingly Gaussian as the number of terms in the sum increases

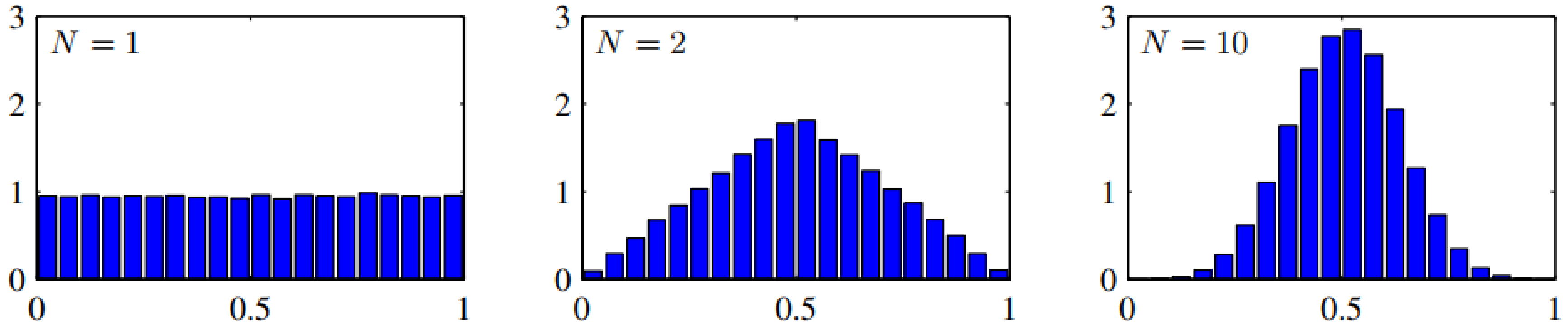


```python
import numpy as np
from matplotlib import pyplot as plt
N = int(input())
means = []
for i in range(1,100000)
    means.append(np.mean(np.random.random(size=(N,))))

plt.hist(means,bins = 100, range=(0,1))
```

Optimization for Training Deep Models

- **The Central Limit Theorem**: Mean of a set of random variables, which is of course itself a random variable, has a distribution that becomes increasingly Gaussian as the number of terms in the sum increases



- The standard error of mean:

$$SE(\hat{\mu}_N) = \sqrt{Var\left(\frac{1}{N}\sum_{i=1}^{N} x_i\right)} = \frac{\sigma}{\sqrt{N}}$$

- The denominator of $\sqrt{N}$ shows that there are less than linear returns to using more examples to estimate the mean.

- Compare two hypothetical estimates of the gradient, one based on 100 examples and another based on 10,000 examples. The latter requires 100 times more computation than the former but reduces the standard error of the mean only by a factor of 10

Optimization for Training Deep Models

- In machine learning the objective function usually decomposes as a sum over the training examples.

$$L(\boldsymbol{\Theta}) = \frac{1}{N}\sum_{i=1}^{N} l_i(\boldsymbol{\Theta})$$

- By linearity

$$\nabla L(\boldsymbol{\Theta}) = \frac{1}{N}\sum_{i=1}^{N} \nabla l_i(\boldsymbol{\Theta})$$

- Computing the gradient requires summing over all of the training examples. This is known as batch training.

$$\boldsymbol{\Theta} = \boldsymbol{\Theta} - \epsilon \nabla L(\boldsymbol{\Theta})$$

- Batch training is impractical if you have a large dataset (e.g. millions of training examples)!

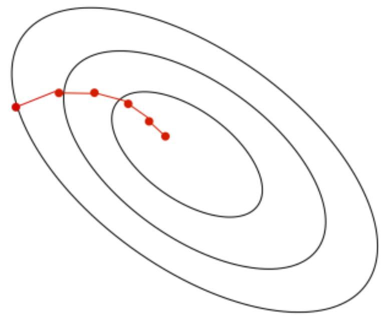
Optimization for Training Deep Models

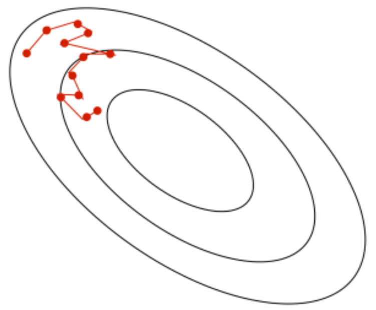
- Stochastic gradient descent (SGD): update the parameters based on the gradient for a single training example:

$$\boldsymbol{\Theta} = \boldsymbol{\Theta} - \epsilon \nabla L_i(\boldsymbol{\Theta})$$

- Based on the central limit theorem, if you sample a training example at random, the stochastic gradient is an unbiased estimate of the batch gradient

Batch gradient descent moves directly downhill. SGD takes steps in a noisy direction, but moves downhill on average.

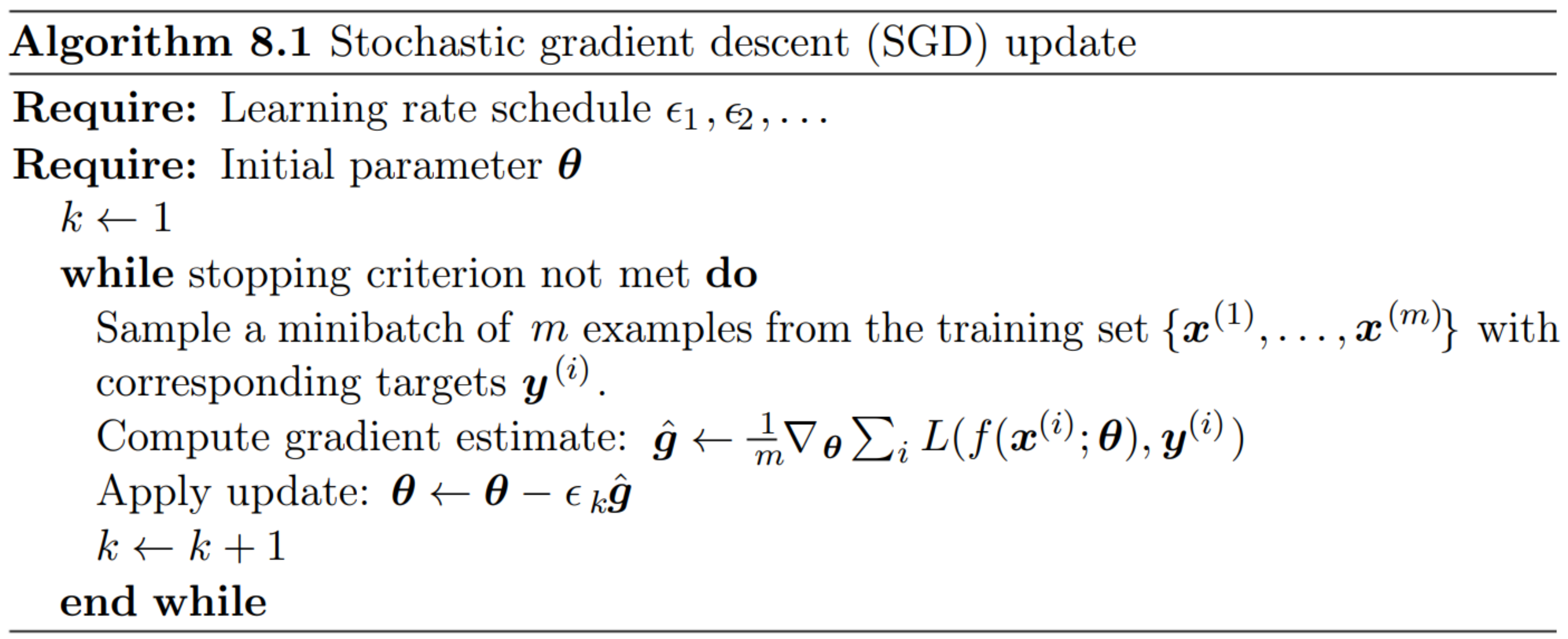

batch gradient descent


stochastic gradient descent

Source: Neural Networks and Deep Learning course by Jimmy Ba, 2020, University of Toronto: https://csc413-2020.github.io/

Optimization for Training Deep Models

- **Problem with SGD**: if we only look at one training example at a time, we can't exploit efficient parallel operations in GPU.
- **Compromise approach:** compute the gradients on a medium-sized set of training examples, called a mini-batch.



Algorithm 8.1 Stochastic gradient descent (SGD) update

```
Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while
```

Source: Goodfellow et al. (2016), Deep Learning

Optimization for Training Deep Models

- **It is necessary to decrease the learning rate over time**.
- Sufficient condition for convergence:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \quad \text{and}$$
$$\sum_{k=1}^{\infty} \epsilon_k^2 < \infty.$$

Optimization for Training Deep Models

Momentum

- When there is high curvature, SGD could be slow.

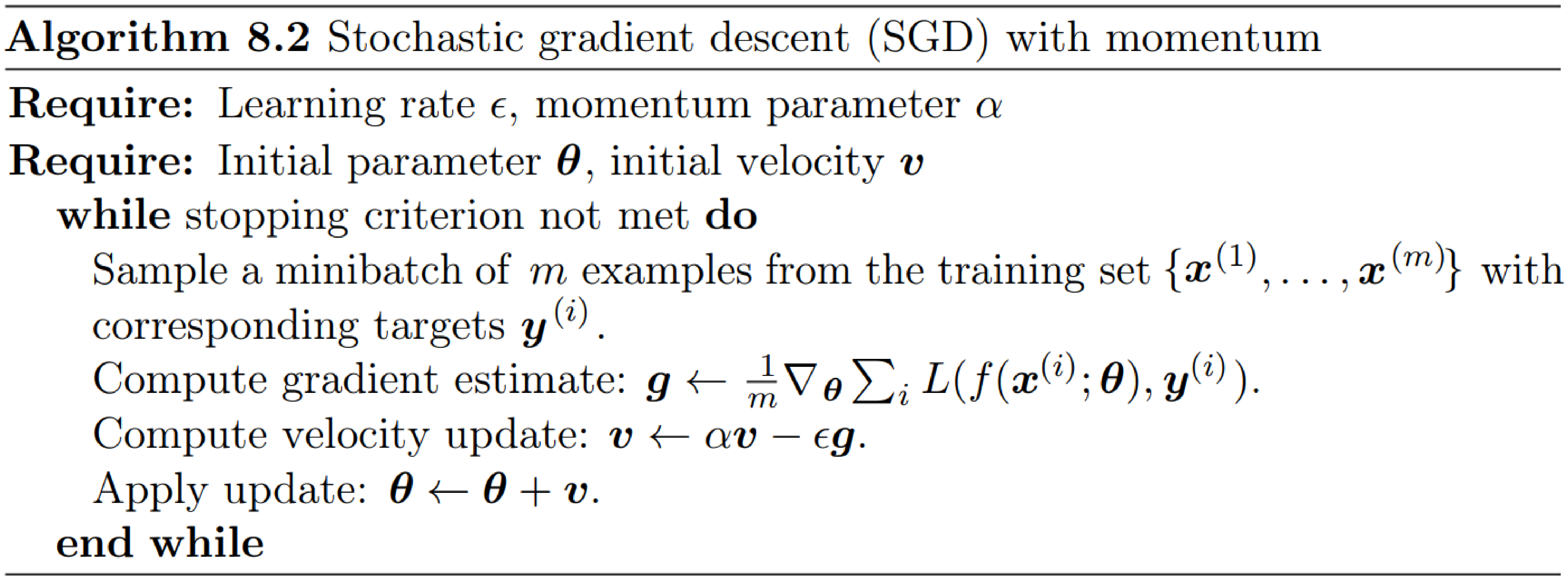
- The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.

$$\boldsymbol{v} = \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\Theta}} \left(\frac{1}{m} \sum_{i=1}^{m} l_i(\boldsymbol{\Theta}) \right), \alpha \in [0,1)$$

$$\boldsymbol{\Theta} = \boldsymbol{\Theta} + \boldsymbol{v}$$

Optimization for Training Deep Models

Momentum



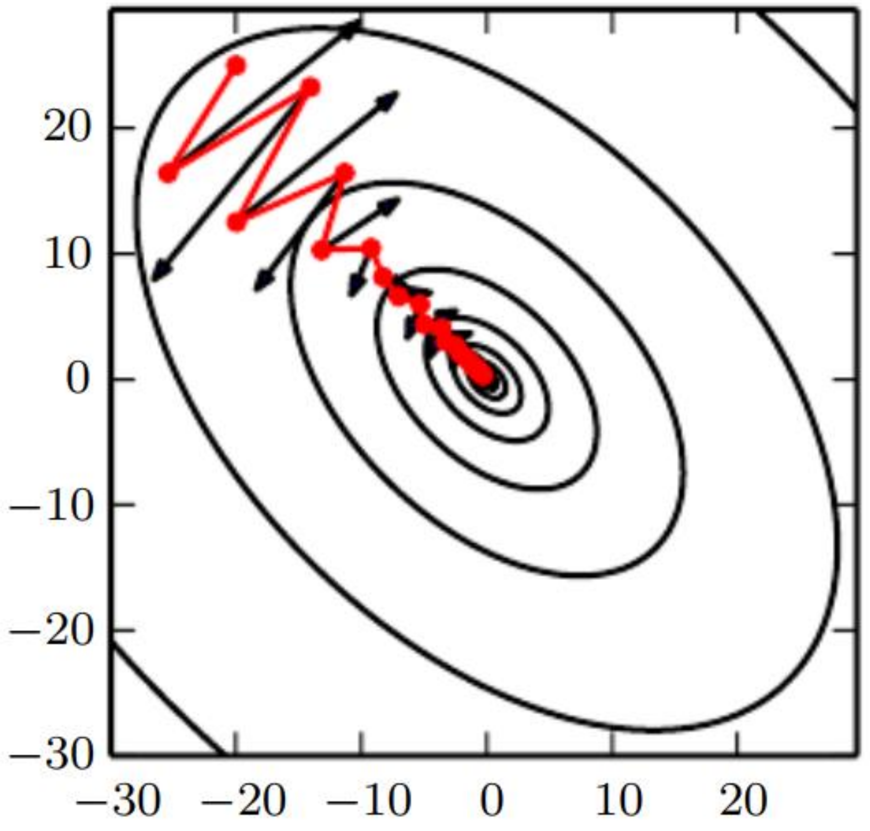
Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

```
Require: Learning rate $\epsilon$, momentum parameter $\alpha$
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
  end while
```

Source: Goodfellow et al. (2016), Deep Learning

Optimization for Training Deep Models

Momentum



Source: Goodfellow et al. (2016), Deep Learning

Optimization for Training Deep Models

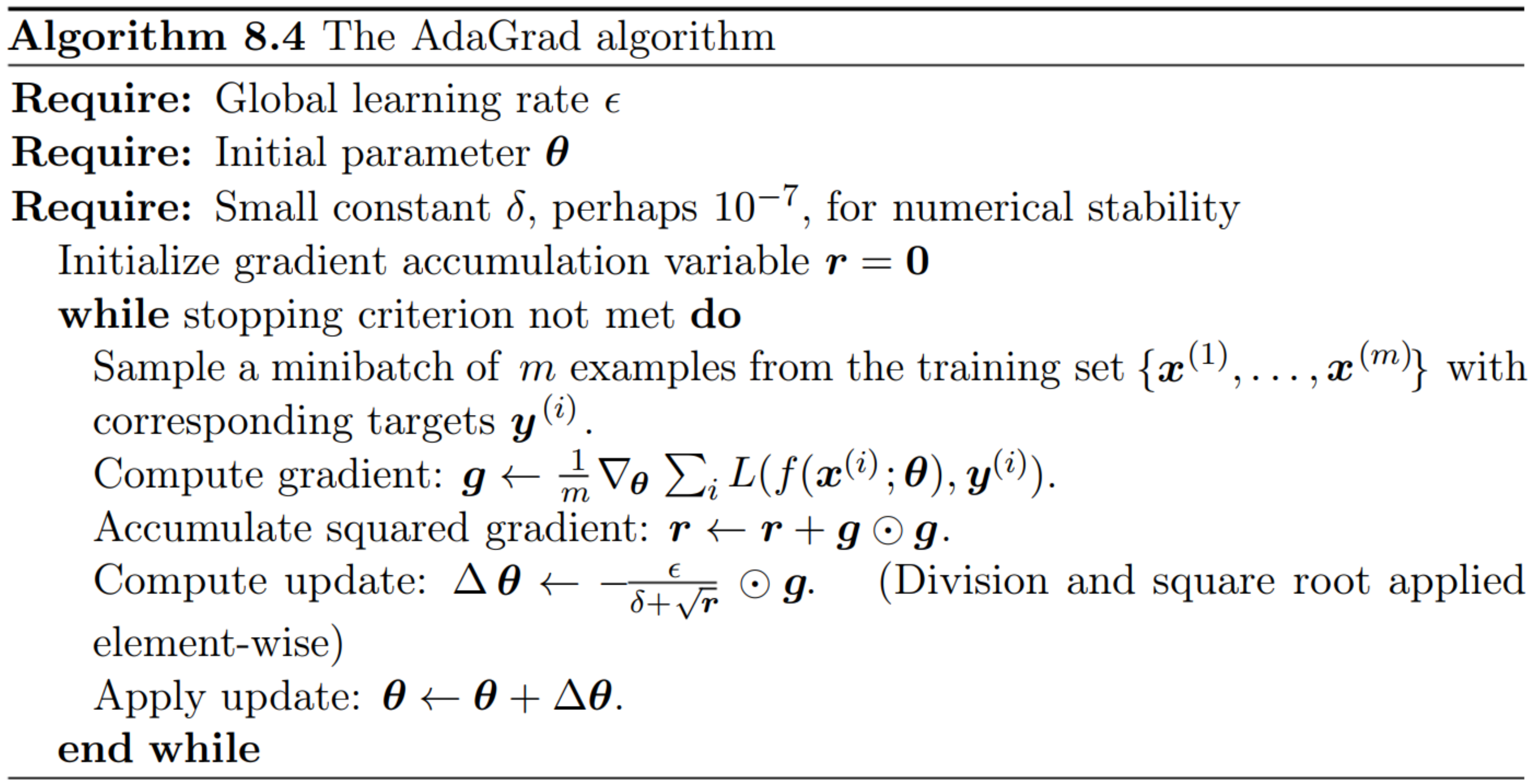
AdaGrad [1]

- Learning rate is reliably one of the most difficult to set hyperparameters because it significantly affects model performance.
- The main idea in AdaGrad is to use a separate learning rate for each parameter and automatically adapt these learning rates throughout the course of learning
- The parameters with the largest partial derivative of the loss have a correspondingly rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate.

[1] Duchi, J., Hazan, E. and Singer, Y., 2011. Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7).

Optimization for Training Deep Models

AdaGrad



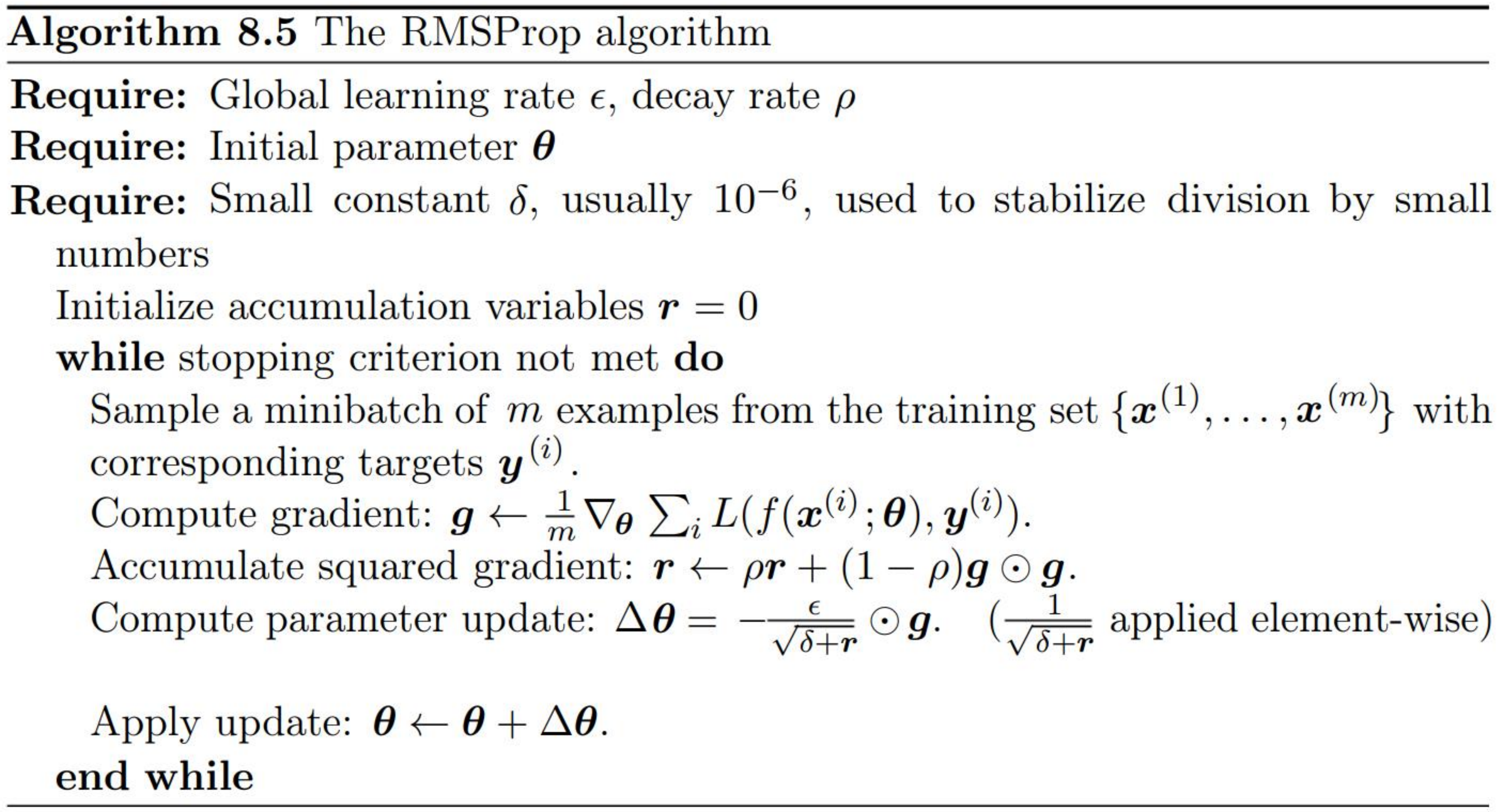
Algorithm 8.4 The AdaGrad algorithm

```
Require: Global learning rate $\epsilon$
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, perhaps $10^{-7}$, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.
    Compute update: $\Delta \boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
  end while
```

Optimization for Training Deep Models

RMSProp (Hinton, 2012)

- RMSProp is an extension to AdaGrad that uses an exponentially decaying average to discard history from the extreme past



Algorithm 8.5 The RMSProp algorithm

```
Require: Global learning rate $\epsilon$, decay rate $\rho$
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small
    numbers
  Initialize accumulation variables $\boldsymbol{r} = 0$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.
    Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$.   ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)

    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
  end while
```

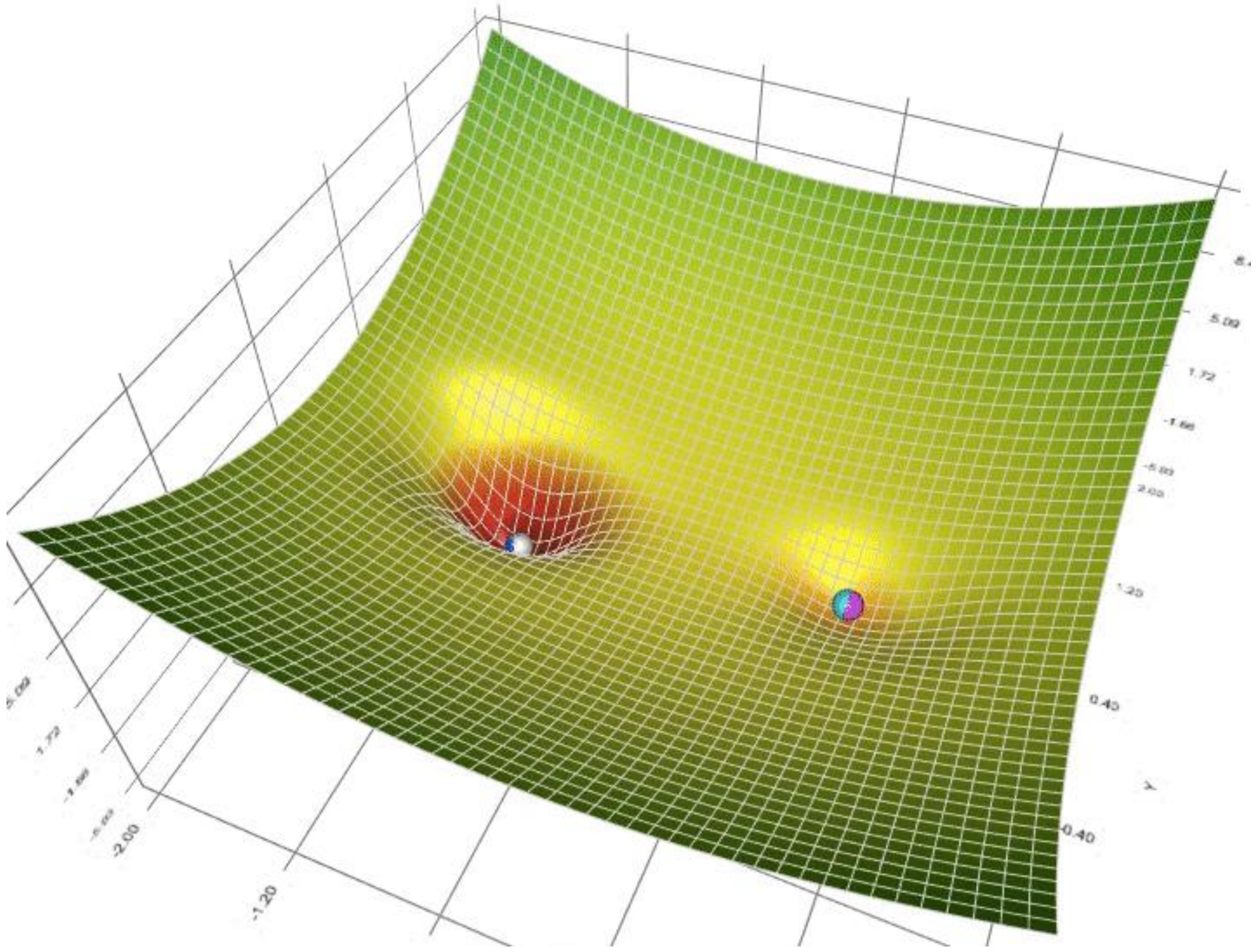
Optimization for Training Deep Models

Adam [1]

Adam = RMSProp + Momentum

[1] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Optimization for Training Deep Models



Animation of 5 gradient descent methods on a surface:

- gradient descent (cyan),
- momentum (magenta),
- AdaGrad (white),
- RMSProp (green),
- Adam (blue).
- Left: global minimum;
- Right: local minimum.

Next …

Convolutional Neural Networks