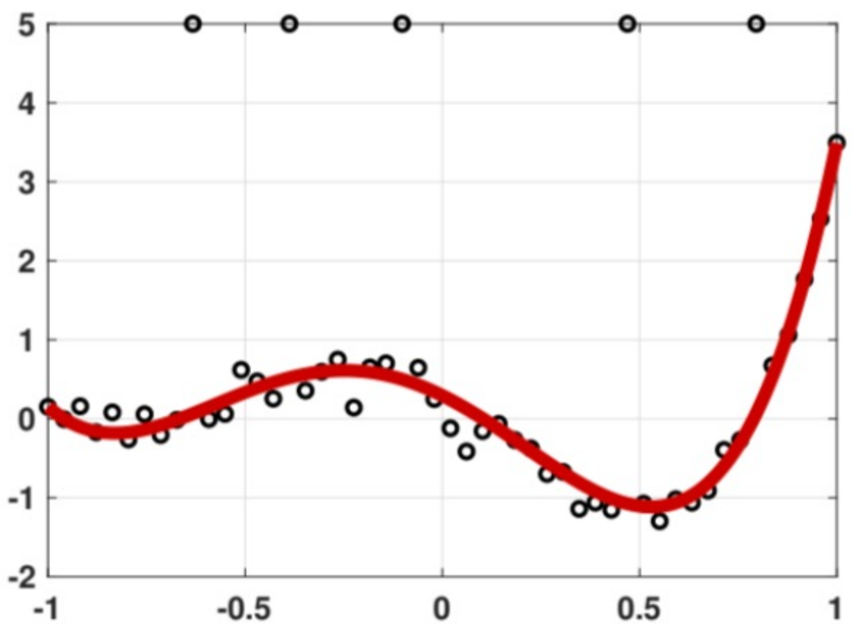
AMA 564 Deep Learning

2025 Spring

Lecture 3

Revisit: Deep Nonparametric Regression

Recall the regression problem



- **Data** $(X_i, Y_i), i = 1, \dots, n.$

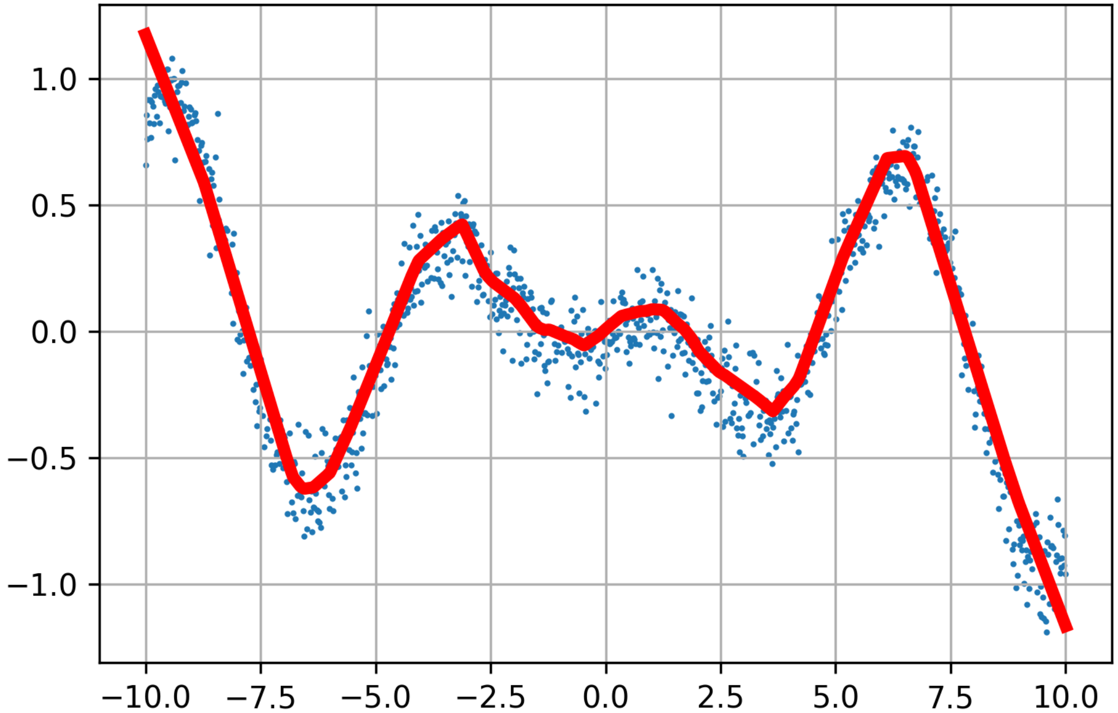
- **To find a network**
  $$f(x; \theta)$$
  such that
  $$\sum \phi(Y_i - f(X_i; \theta))$$
  is minimized over

$$\mathcal{F} = \{f: f(x; \theta) \text{ is a neural network parameterized by } \theta \in \mathbb{R}^s\}$$

- **How do we solve for θ ?**
  1. Initialize $\theta_0 \in \mathbb{R}^s$
  2. Calculate the gradient at θ_t (with different ϕ)
  3. Move a step α_t
  4. Iterate until stop

Revisit: Deep Nonparametric Regression

Recall the regression problem



- **Data** $(X_i, Y_i), i = 1, \dots, n.$
- **To find a network** $f(x; \theta)$ **such that** $\sum \left(Y_i - f(X_i; \theta)\right)^2$ **is minimized over**

$$\mathcal{F} = \{f : f(x; \theta) \text{ is a neural network parameterized by } \theta \in \mathbb{R}^s\}$$

- **How do we solve for θ ?**
  1. **Initialize $\theta_0 \in \mathbb{R}^s$**
  2. **Calculate the gradient at θ_t**
  3. **Move a step α_t**
  4. **Iterate until stop.**

Revisit: The Optimization Problem

The optimization problem

- Data $(X_i, Y_i), i = 1, \dots, n.$
- The empirical risk

$$R_n(\theta) = R_n\big(f(\cdot, \theta)\big) = \frac{1}{n}\sum\big(Y_i - f(X_i; \theta)\big)^2 .$$

- To minimize $R_n(\theta)$ over $\theta \in \mathbb{R}^s$.

Initialize $\boldsymbol{\theta_0} \in \mathbb{R}^s$ by some randomization

For $\boldsymbol{t = 1, \cdots, T}$

Calculate $\boldsymbol{\frac{dR_n(\theta)}{d\theta}|_{\theta=\theta_{t-1}}}$

Set stepsize $\boldsymbol{\alpha_t > 0}$

Update $\boldsymbol{\theta_t = \theta_{t-1} - \alpha_t \cdot [\frac{dR_n(\theta)}{d\theta}|_{\theta=\theta_{t-1}}]}$

After *T* times iterations, we got $\boldsymbol{\theta_T}$ such that $\boldsymbol{R_n(\theta_T)}$ is small.

The Optimization Problem

Question

How to calculate the gradient

$$\frac{d}{d\boldsymbol{\theta}}\boldsymbol{R_n}(\boldsymbol{\theta}) = -\frac{2}{n}\sum(\boldsymbol{Y_i} - \boldsymbol{f}(\boldsymbol{X_i};\boldsymbol{\theta}))\frac{d}{d\boldsymbol{\theta}}\boldsymbol{f}(\boldsymbol{X_i};\boldsymbol{\theta})$$

especially how to compute $\frac{d}{d\boldsymbol{\theta}}\boldsymbol{f}(\boldsymbol{X_i};\boldsymbol{\theta})$ exactly?

BackPropogation

Warm-Start: Chain Rule

$$\boxed{\frac{\boldsymbol{d}}{\boldsymbol{dx}}[\boldsymbol{f}(\boldsymbol{g}(\boldsymbol{x}))] = \frac{\boldsymbol{df}}{\boldsymbol{dg}} \times \frac{\boldsymbol{dg}}{\boldsymbol{dx}}}$$

Warm-Start: Chain Rule

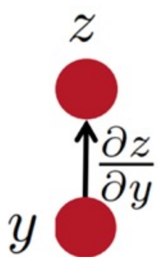
$$\boxed{\frac{d}{dx}[f(g(x))] = \frac{df}{dg} \times \frac{dg}{dx}}$$

The arrow shows functional dependence of z on y

- i.e. given y, we can calculate z.
- e.g., for example: $z(y) = 2y^2$

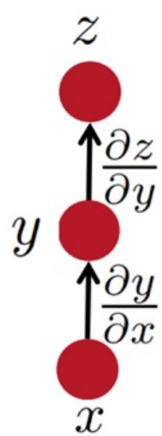
The derivative of z, with respect to y.

- e.g., for example: $\frac{\partial z(y)}{\partial y} = 4y$.



Simple chain rule

- If z is a function of y, and y is a function of x
  - Then z is a function of x, as well.
- Question: how to find $\frac{\partial z}{\partial x}$



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

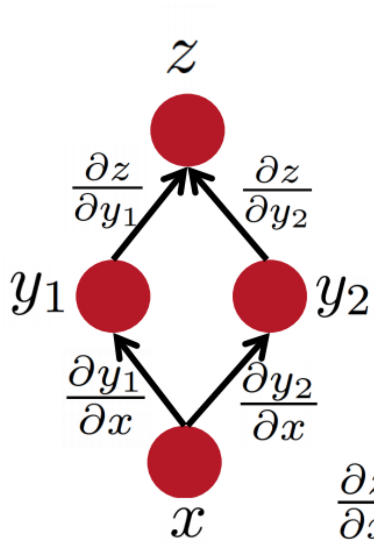
We will use these facts to derive the details of the Backpropagation algorithm.

z will be the error (loss) function.
- We need to know how to differentiate z

Intermediate nodes use a logistics function (or another differentiable step function).
- We need to know how to differentiate it.

Multiple Path Chain Rule

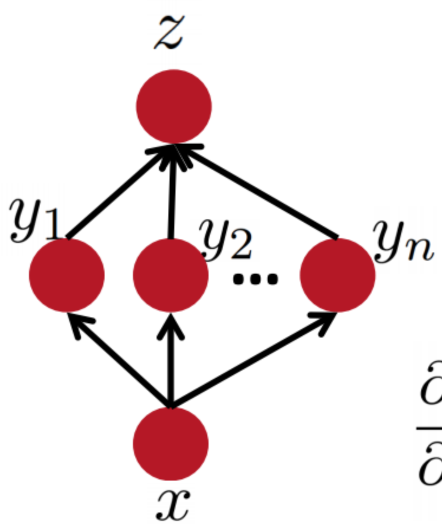
$$z(x) \coloneqq z(y_1(x), y_2(x))$$



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

Multiple Path Chain Rule

$$\boldsymbol{z}(\boldsymbol{x}) \coloneqq \boldsymbol{z}(\boldsymbol{y_1}(\boldsymbol{x}), \boldsymbol{y_2}(\boldsymbol{x}), \dots, \boldsymbol{y_n}(\boldsymbol{x}))$$



$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Backpropogation

Loop over instances:

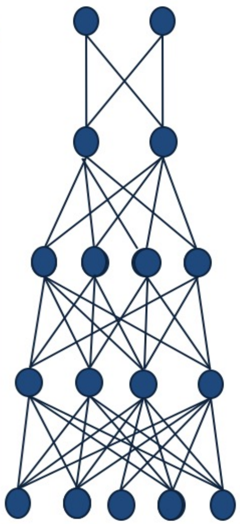
1. The forward steps

- Given the input, make predictions layer-by-layer, starting from the first layer)

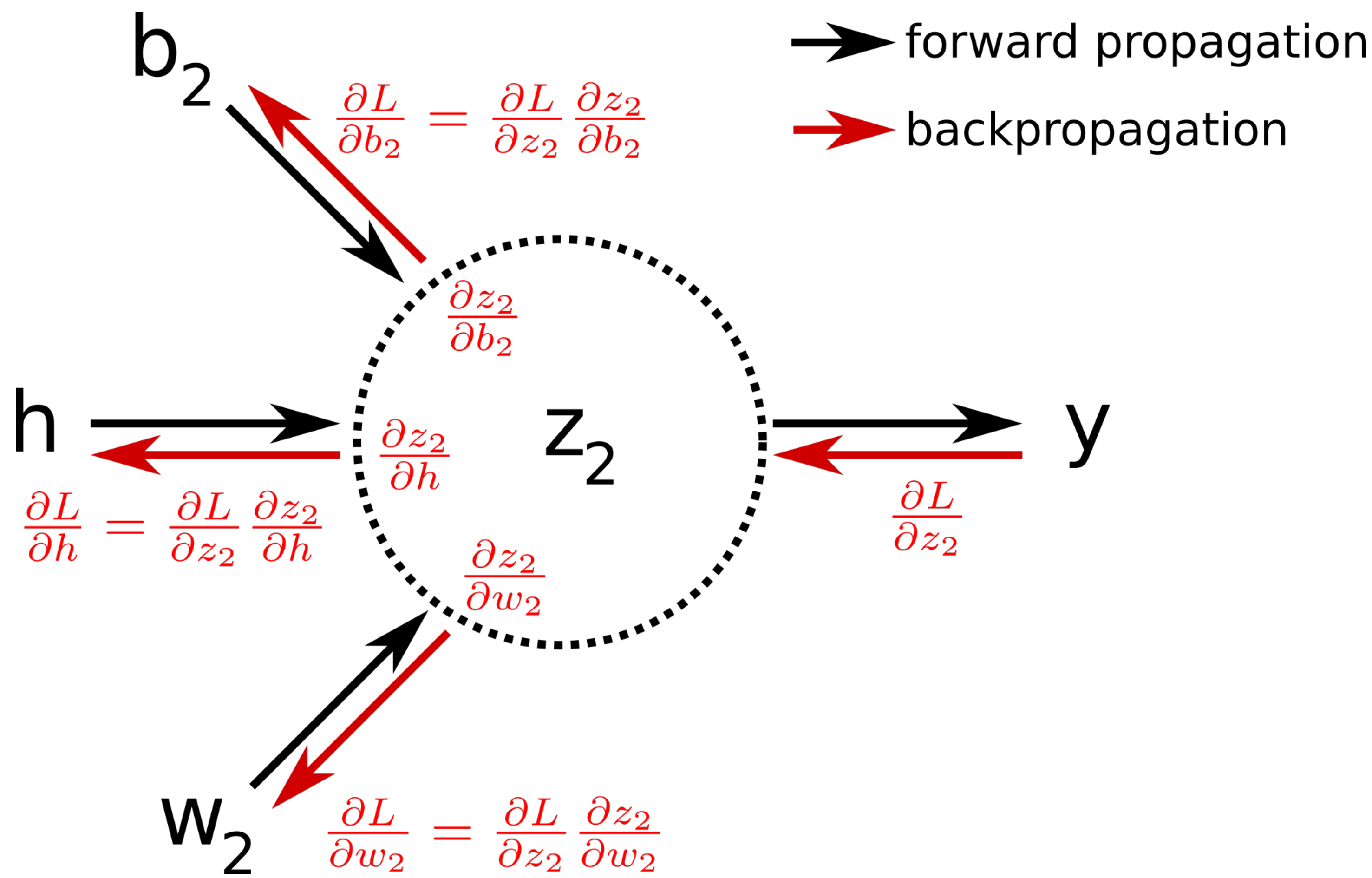
2. The backward steps

- Calculate the error in the output
- Update the weights layer-by-layer, starting from the final layer

$\frac{\partial f}{\partial \theta_{ij}}$



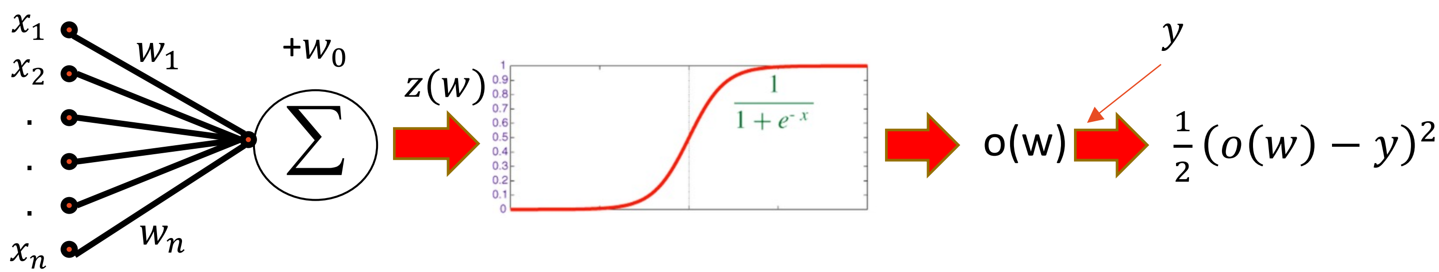
A simple example



Backpropogation: An example

One Layer Percepton

Neuron is modeled by a unit connected by weighted links w_i to other units i.



$$L(w) = \frac{1}{2}(o(w) - y)^2,$$

$$o(w) = sigmoid(z(w)),$$

$$z(w) = \ w_0 + \sum_{i=1}^{n} w_i x_i .$$

One Layer Percepton

$$\frac{\partial L}{\partial w_i} = \frac{dL}{do(w)} \times \frac{do(z(w))}{dz(w)} \times \frac{\partial z(w)}{\partial w_i}$$

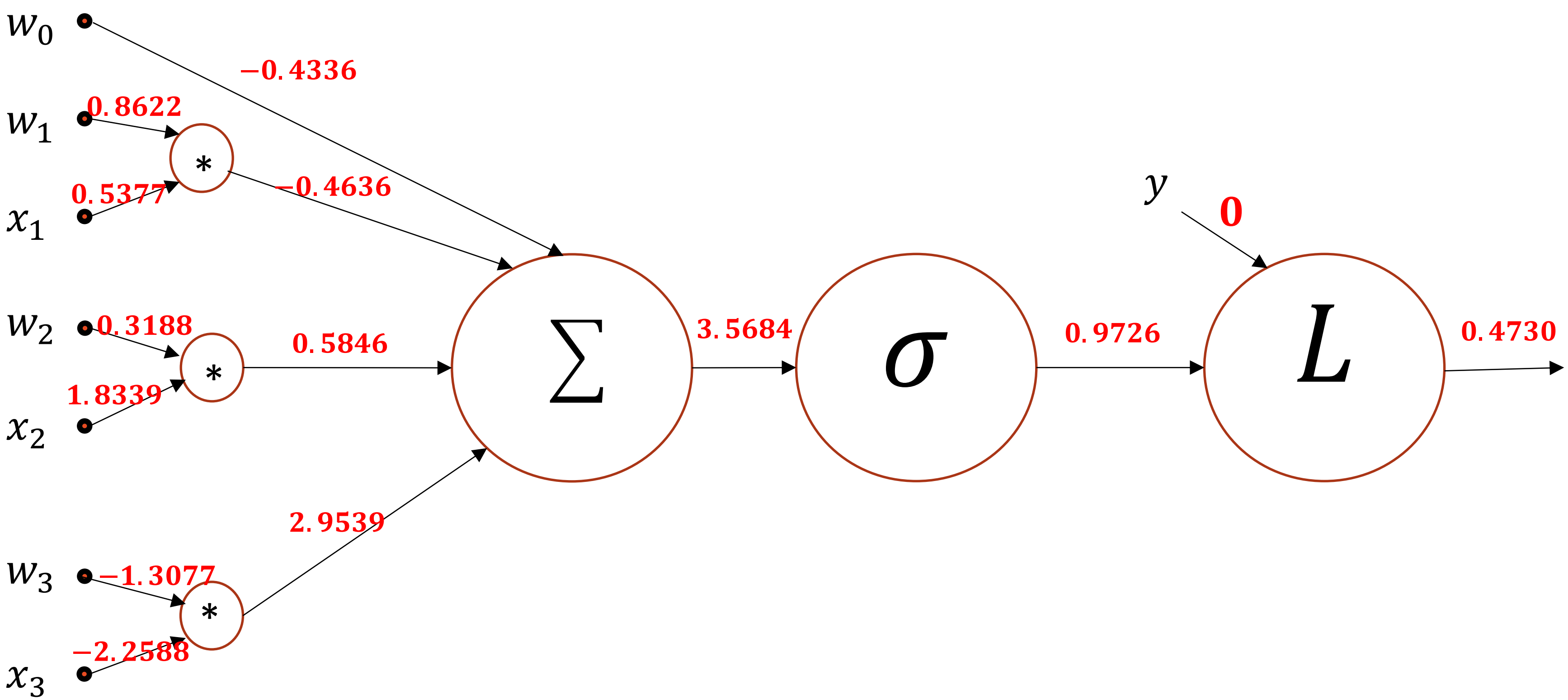
$$\frac{dL}{do(w)} = \frac{d}{do(w)}\left[\frac{1}{2}(o(w) - y)^2\right] = o(w) - y$$

$$\frac{do(z(w))}{dz(w)} = \frac{d}{dz(w)}\left[\frac{1}{1 + e^{-z(w)}}\right] = \frac{1}{1 + e^{-z(w)}} \times (1 - \frac{1}{1 + e^{-z(w)}})$$

$$\frac{\partial z(w)}{\partial w_i} = \frac{\partial}{\partial w_i}\left[w_0 + \sum_{j=1}^{n} w_j x_j\right] = x_i$$

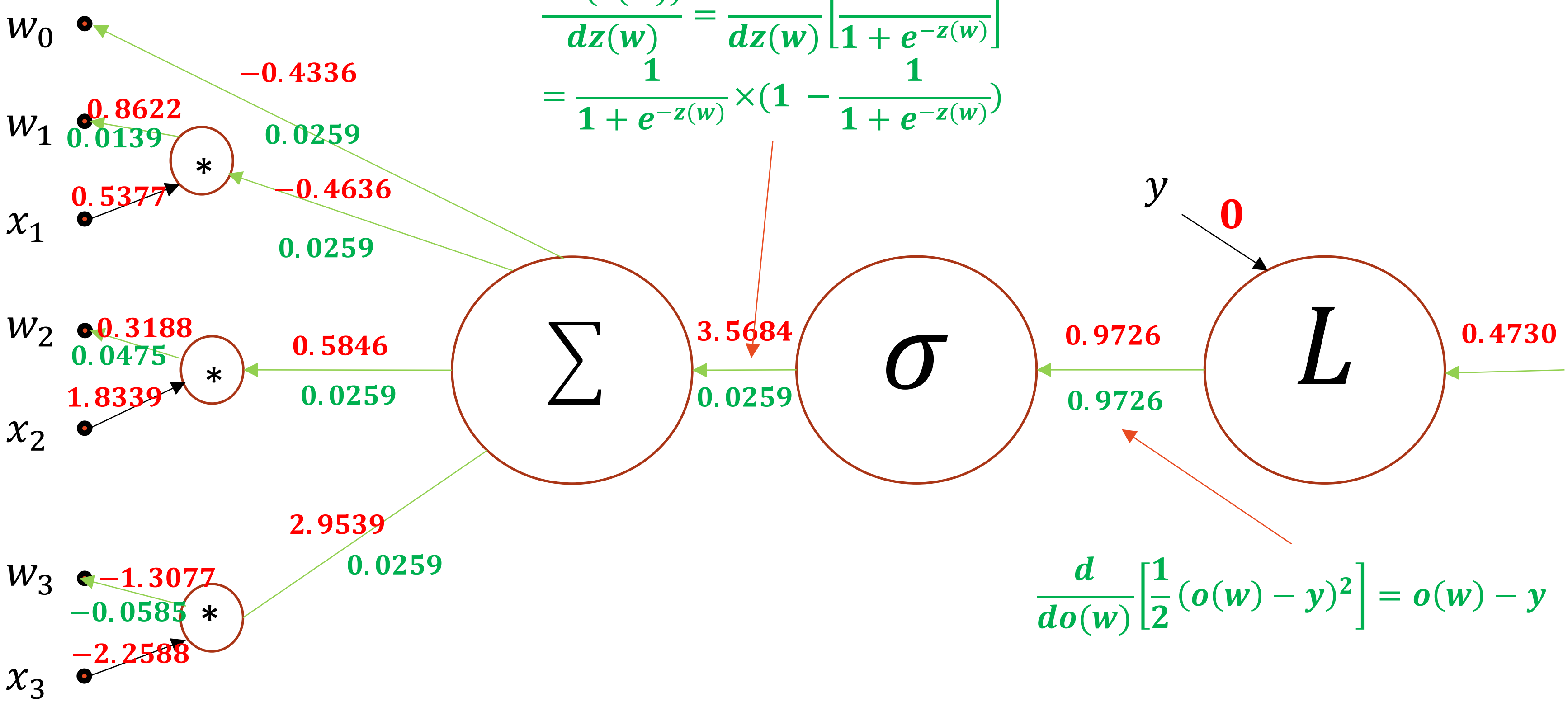
Note: we denote $\boldsymbol{x_0 = 1}$.

Backpropogation



A simple example

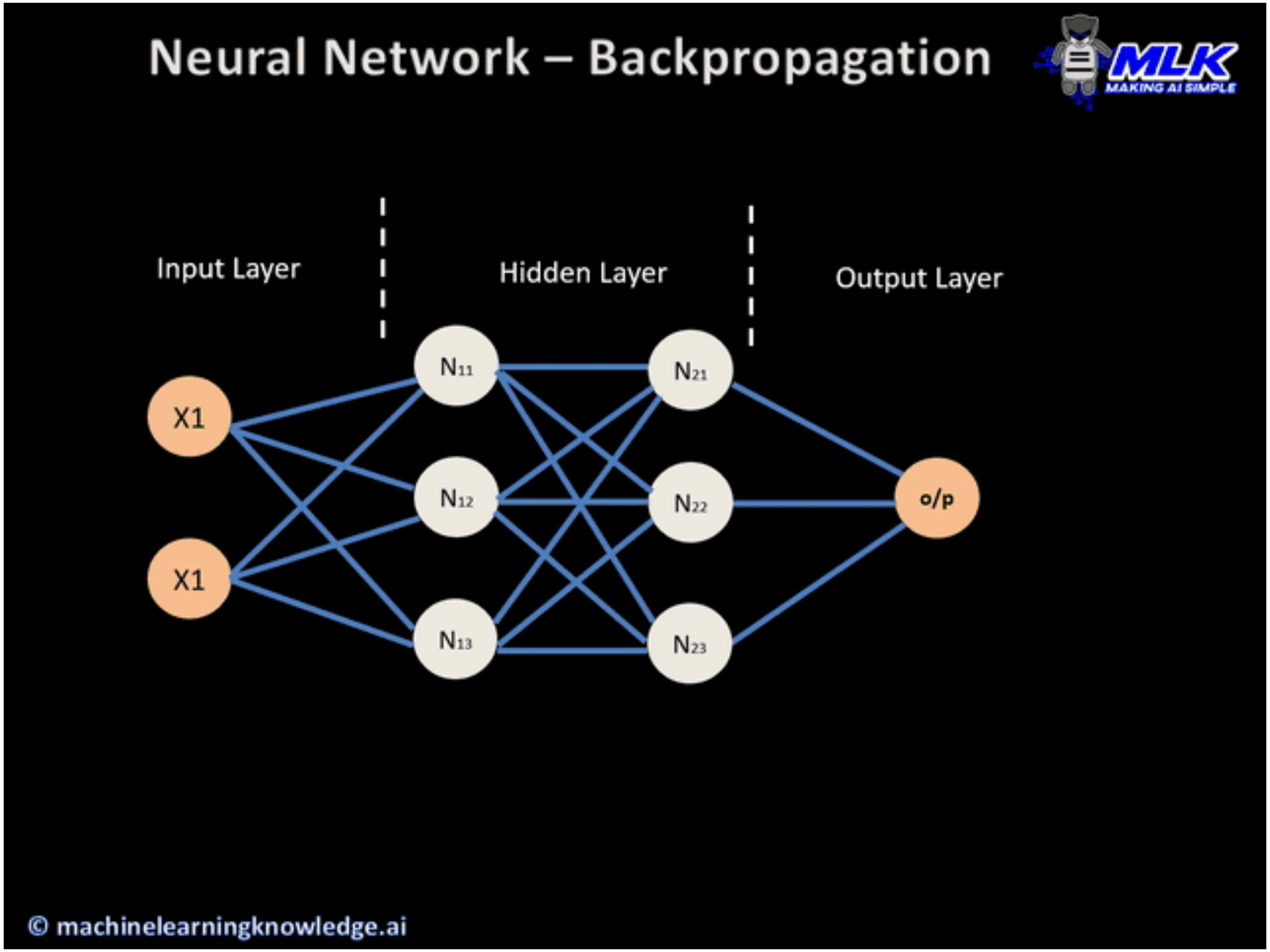
$$\frac{do(z(w))}{dz(w)} = \frac{d}{dz(w)}\left[\frac{1}{1+e^{-z(w)}}\right] = \frac{1}{1+e^{-z(w)}} \times \left(1 - \frac{1}{1+e^{-z(w)}}\right)$$



$$\frac{d}{do(w)}\left[\frac{1}{2}(o(w)-y)^2\right] = o(w) - y$$

$$\frac{\partial z(w)}{\partial w_i} = \frac{\partial}{\partial w_i}\left[w_0 + \sum_{j=1}^{n} w_j x_j\right] = x_i$$

A simple example



Source:https://medium.com/analytics-vidhya/backpropagation-for-dummies-e069410fa585

A good tutorial: https://youtu.be/tIeHLnjs5U8?si=fmmHycZ7rscG8eJE

The Optimization Problem

Questions

1. How to initialize $\theta_0 \in \mathbb{R}^s$?

2. How to choose the stepsize $\alpha_t > 0$?

3. What if the sample size $\boldsymbol{n}$ is very large?

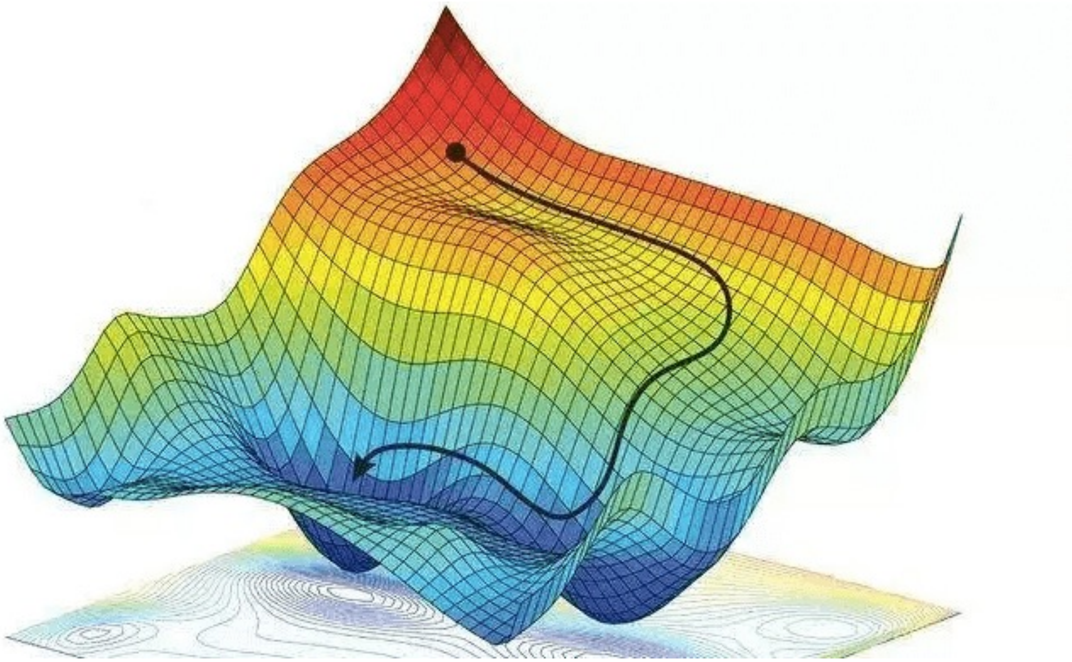
Optimization Algorithms

1. **Stochastic Gradient Descent**
2. **Momentum Acceleration**
3. **AdaGrad**
4. **ADAM**

Problem Set-up

Given data $(X_i, Y_i), i = 1, \ldots, n$. Minimize a **loss function** over $\theta \in \mathbb{R}^s$**:**

$$\min_{\theta \in \mathbb{R}^s} \; f(\theta) := \frac{1}{n} \sum_{i=1}^{n} l(\theta; X_i, Y_i).$$



Source:https://www.nature.com/articles/s41592-019-0369-z

Gradient Descent Algorithm

Start from some $\theta^0 \in \mathbb{R}^s$, gradient descent (GD) algorithm updates as:

$$\theta^{k+1} = \theta^k - \alpha_k \nabla f(\theta^k),$$

until

$$||\nabla f\left(\theta^{k+1}\right)|| \leq \varepsilon,$$

for some tolerance $\varepsilon > 0$.

Gradient Descent Algorithm

Start from some $\theta^0 \in \mathbb{R}^s$, gradient descent (GD) algorithm updates as:

$$\theta^{k+1} = \theta^k - \alpha_k \nabla f(\theta^k),$$

until

$$||\nabla f(\theta^{k+1})|| \leq \varepsilon,$$

for some tolerance $\varepsilon > 0$.

Key points:

1. Compute $\boldsymbol{\nabla f(\theta^k)}$.
2. Choose step size $\alpha_k > 0$ satisfying

$$\boldsymbol{f(\theta^{k+1}) < f(\theta^k)}.$$

Feasible step sizes for L-smooth functions

Assumption 3.1

$\boldsymbol{f}(\boldsymbol{\theta})$ is continuously differentiable and $\nabla \mathbf{f}(\boldsymbol{\theta})$ is **Lipschitz continuous**:

$$||\nabla \mathbf{f}(\mathbf{x}) - \nabla \mathbf{f}(\mathbf{y})|| \leq \boldsymbol{L}||\boldsymbol{x} - \boldsymbol{y}||$$

for some L > 0. We call $\boldsymbol{f}$ satisfying this property is a **L-smooth function**.

Lemma 3.1 Given an L-smooth function f, then for any $x, y \in dom(f)$, we have

$$f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{L}{2}||y - x||^2.$$

Consequence: If we choose $\alpha_k = 1/L$, then

$$\begin{aligned} f(\theta^{k+1}) - f(\theta^k) &= f(\theta^k - \tfrac{1}{L}\nabla f(\theta^k)) - f(\theta^k) \\ &\leq \nabla f(\theta^k)^T(-\tfrac{1}{L}\nabla f(\theta^k)) + \tfrac{L}{2}\|\tfrac{1}{L}\nabla f(\theta^k)\|^2 \\ &\leq -\tfrac{1}{2L}\|\nabla f(\theta^k)\|^2. \end{aligned}$$

Question: If we apply Lemma 3.1, what is the optimal fixed step size ?

Convergence of Gradient Descent Algorithm

Theorem 3.1 Let f be a L-smooth function and $f(\theta) \geq \bar{f} > -\infty$ for any θ. Let $\{\theta^k\}_{k=0}^T$ be the sequence generated by the gradient descent algorithm with step size 1/L, then

$$\min_{1 \leq k \leq T} \|\nabla f(\theta^k)\|^2 \leq \frac{2L(f(\theta^0) - \bar{f})}{T}.$$

Convergence of Gradient Descent Algorithm

Theorem 3.1 Let f be a L-smooth function and $f(\theta) \geq \bar{f} > -\infty$ for any θ. Let $\{\theta^k\}_{k=0}^T$ be the sequence generated by the gradient descent algorithm with step size 1/L, then

$$\min_{1 \leq k \leq T} \|\nabla f(\theta^k)\|^2 \leq \frac{2L(f(\theta^0) - \bar{f})}{T}.$$

We leave the proof as a question in Assignment 1.

Hint:

Step 1: Apply Lemma 3.1 at step k.

Step 2: Sum them up for k = 0, 1, …, T.

Step 3: Realize that f is bounded from below.

Computation Bottleneck in Deep Learning

In gradient descent, we need to compute

$$\nabla f(\theta^k) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta l(\theta^k; X_i, Y_i).$$

This computation is expensive if n is huge !!!

Question: How to overcome it?

Hint: How to estimate the expectation of a random variable?

Stochastic Gradient

Instead of computing the exact gradient, we consider

$$g(\theta, \xi),$$

which is a stochastic estimation satisfying

$$\mathbb{E}_{\xi}[g(\theta, \xi)] = \nabla f(\theta).$$

Stochastic Gradient

Instead of computing the exact gradient, we consider

$$g(\theta, \xi),$$

which is a stochastic estimation satisfying

$$\mathbb{E}_\xi[g(\theta, \xi)] = \nabla f(\theta).$$

Examples:

Noisy gradients: Assume ξ is a random noise satisfying $\mathbb{E}[\xi] = 0$, we consider

$$g(\theta, \xi) = \nabla f(\theta) + \xi.$$

Stochastic gradients: Assume ξ is an index uniformly sampling from {1, 2, …, n}, we consider

$$g(\theta, \xi) = \nabla_\theta \, l(\theta; X_\xi, Y_\xi).$$

Stochastic Gradient Descent (SGD)

Start from some $\theta^0 \in \mathbb{R}^s$, the SGD algorithm updates iteratively as:

$$\theta^{k+1} = \theta^k - \alpha_k g(\theta^k, \xi_k),$$

where $g(\theta^k, \xi_k)$ is the stochastic gradient computed at θ^k.

Key points:

1. Sampling strategy to compute $g(\theta^k, \xi_k)$.
2. Choose step size $\alpha_k > 0$.

A natural question: How to check the quality of the solution?

We measure $\mathrm{E}_\xi || g(\theta^k, \xi) ||$!!!

Convergence of SGD

Assumption 3.2 f is a **convex function** and

$$\mathbb{E}_\xi[g(\theta,\xi)] = \nabla f(\theta),$$
$$\mathbb{E}_\xi[\|g(\theta,\xi)\|^2] \le B^2,\ \forall\theta.$$

where B is a given parameters.

Theorem 3.2 Let $\{\theta^k\}$ be the sequence generated by SGD with step size $\alpha_k > 0$, under Assumption 3.2, for any T > 0,

$$\mathbb{E}[f(\bar{\theta}^T) - f^*] \le \frac{\|\theta^0 - \theta^*\|^2 + B^2 \sum_{j=0}^T \alpha_j^2}{2\sum_{j=0}^T \alpha_j},$$

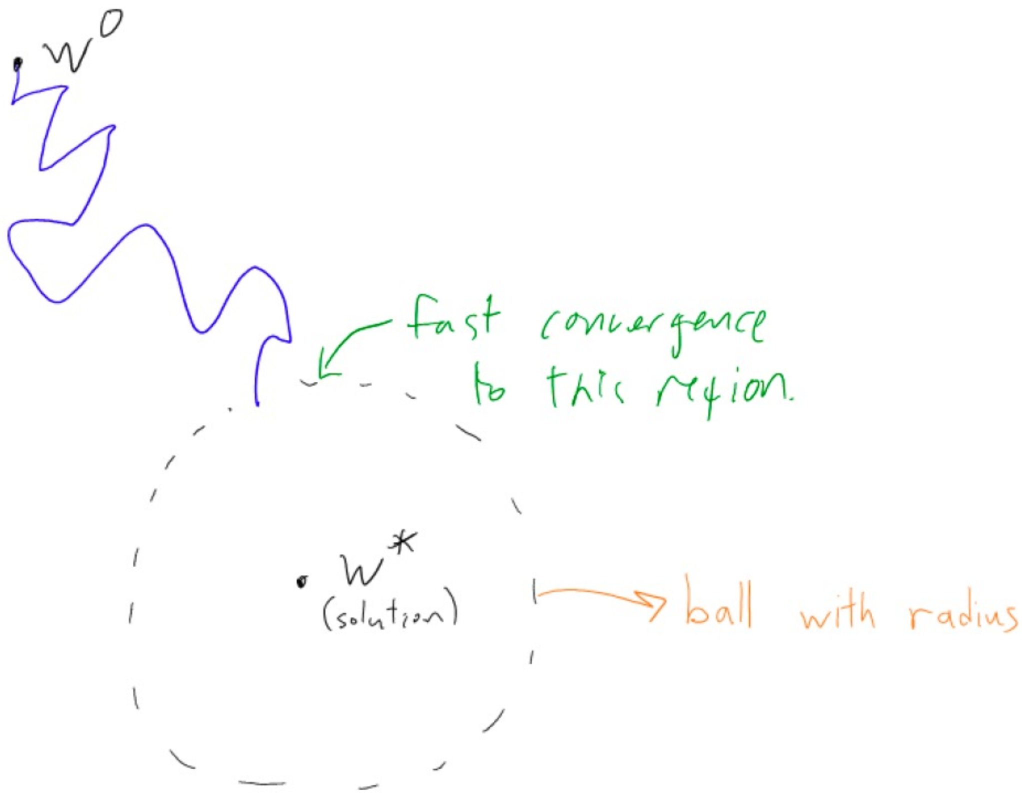
where

$$\lambda_k = \sum_{j=0}^k \alpha_j,\ \bar{\theta}^k = \lambda_k^{-1} \sum_{j=0}^k \alpha_j \theta^j.$$

Implications

Proposition 3.1 If we take $\alpha_j = \alpha > 0$, then

$$\mathbb{E}[f(\bar{\theta}^T) - f^*] \leq \frac{\|\theta^0 - \theta^*\|^2 + B^2(T+1)\alpha^2}{2(T+1)}.$$



As $T \to \infty$, the estimator

$$\hat{\theta}^T$$

will be in a ball with radius

$$B^2\alpha^2/2$$

Implications

Proposition 3.1 If we take $\alpha_j = \alpha > 0$, then

$$\mathbb{E}[f(\bar{\theta}^T) - f^*] \leq \frac{\|\theta^0 - \theta^*\|^2 + B^2(T+1)\alpha^2}{2(T+1)}.$$

Implication: We need to choose decreasing step size.

For example, choose $\alpha_t = \frac{1}{t+1}$, then

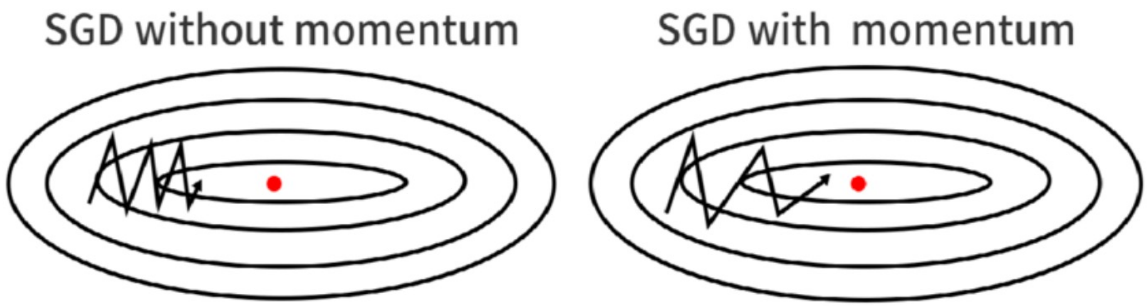
$$\sum_{t=0}^{\infty} \alpha_t = \sum_{t=1}^{\infty} \frac{1}{t} = \infty$$

and

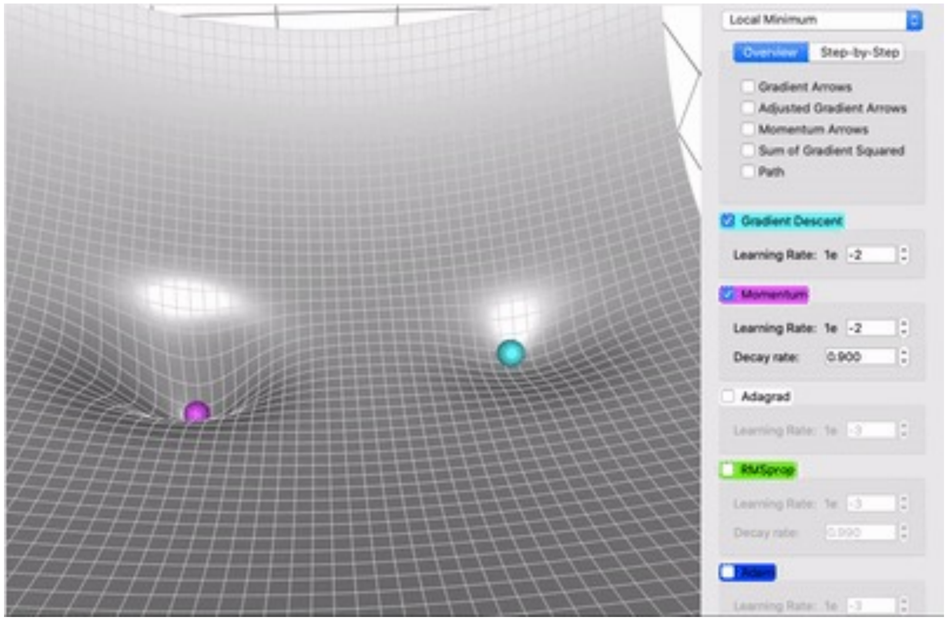
$$\sum_{t=0}^{\infty} \alpha_t^2 = \sum_{t=1}^{\infty} \frac{1}{t^2} = \frac{\pi^2}{6} < \infty$$

Is SGD Good Enough?

1. **Slow convergence**.



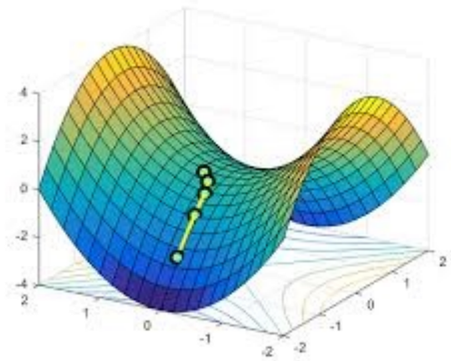
2. **Converge to local optimal solution**.



Is SGD Good Enough?

3. **Converge to saddle points**.

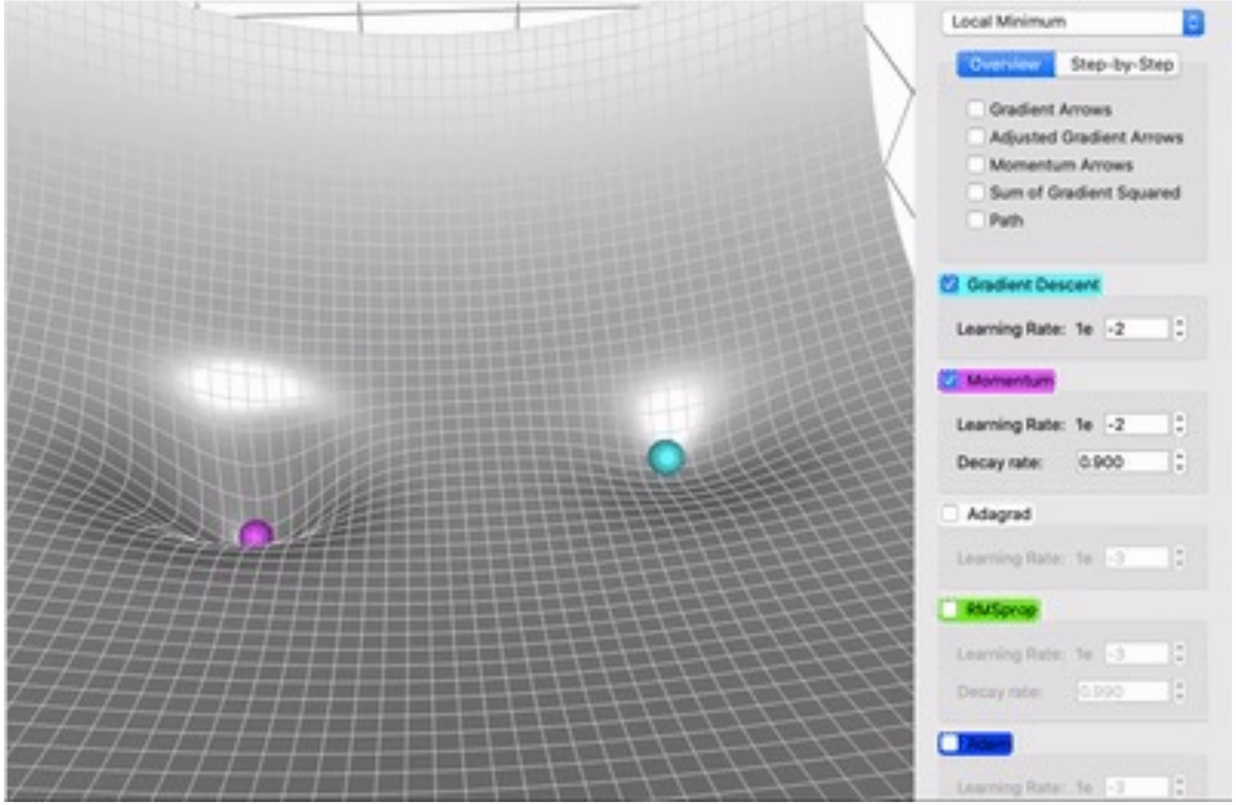
$$f(x, y) = x^2 - y^2.$$
$$\frac{\partial}{\partial x} f(0, 0) = 2 * 0 = 0,$$
$$\frac{\partial}{\partial y} f(0, 0) = -2 * 0 = 0.$$



SGD with Momentum

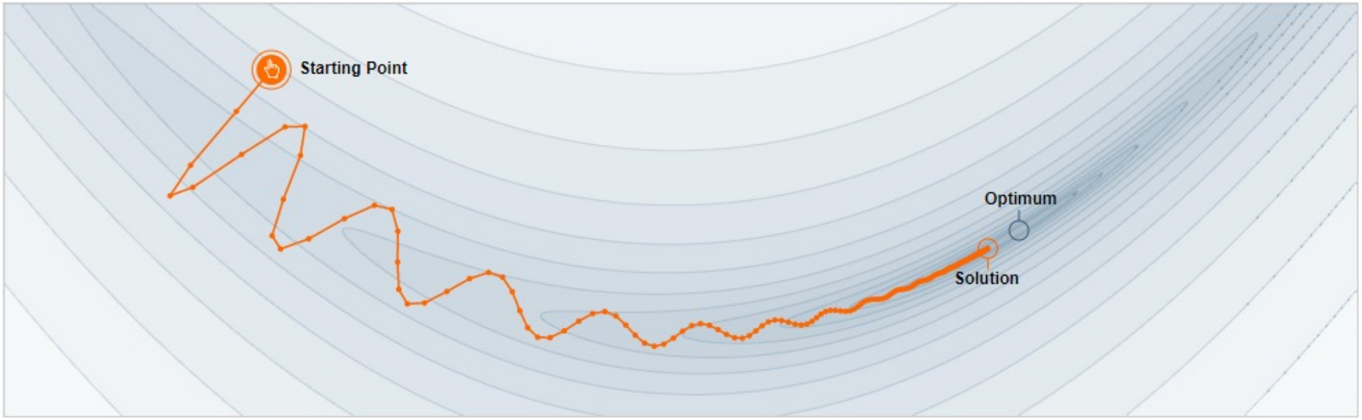
Start from some $\theta^0 \in \mathbb{R}^s$, $v_0 = g(\theta^0, \xi_0)$, for $k \geq 0$:

$$v^{k+1} = \gamma v^k + (1-\gamma) g(\theta^k, \xi_k),$$
$$\theta^{k+1} = \theta^k - v^{k+1}.$$

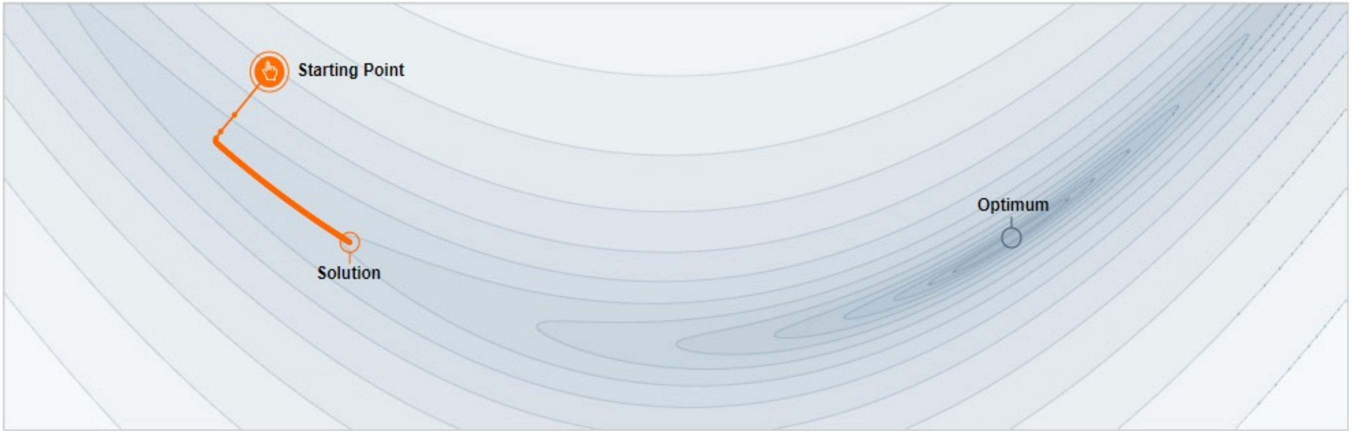


γ is usually chosen to be 0.9 in practice.

A Simple Example



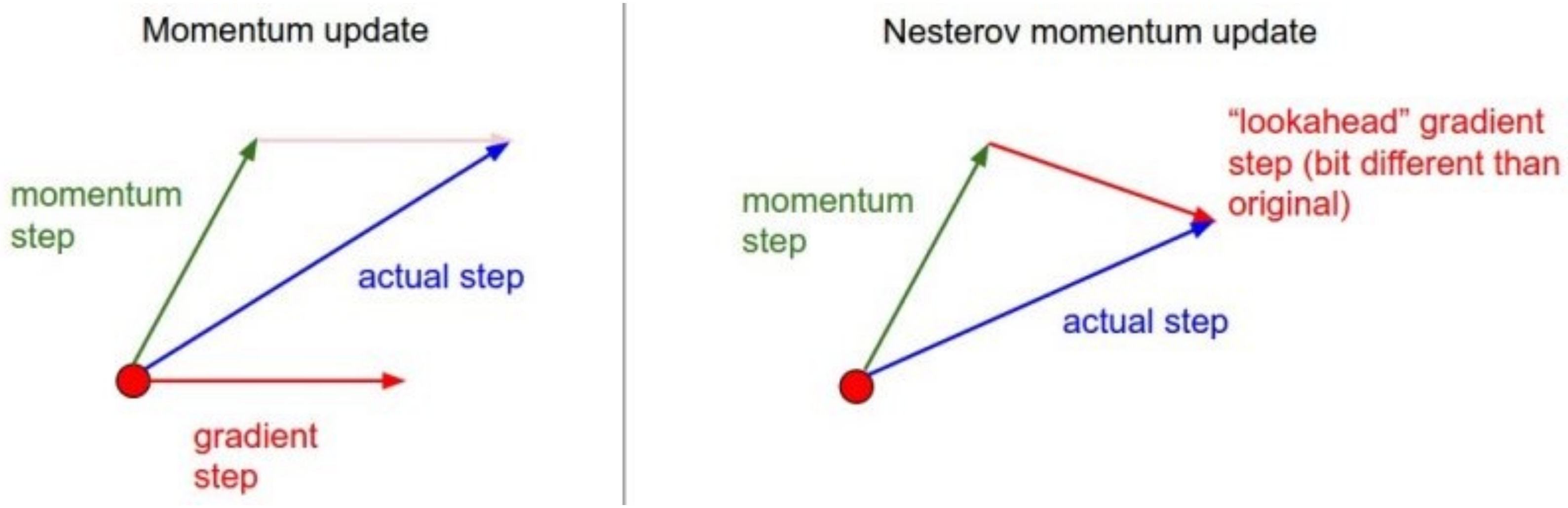
$\gamma = 0.9$



$\gamma = 0$

Reading Material: [Why Momentum Real Works?]

Nesterov Momentum



Start from some $\theta^0 \in \mathbb{R}^s$, $v_0 = g(\theta^0, \xi_0)$, for $k \geq 0$:

$$
\begin{aligned}
\vartheta^k &= \theta^k - \beta_k v^k, \\
v^{k+1} &= \beta_k v^k + \alpha_k g(\vartheta^k, \xi_k), \\
\theta^{k+1} &= \theta^k - v^{k+1}.
\end{aligned}
$$

An advantage: prevent overshot!

AdaGrad: Adaptive Learning Rates

Key idea: Rescale the learning rate of each coordinate by the historical progress.

Start from some $\theta^0 \in \mathbb{R}^s$, $n_g = 0$, for $k \geq 0$:

$$\begin{aligned} n_g &= n_g + g(\theta^k, \xi_k) .* g(\theta^k, \xi_k), \\ \theta^{k+1} &= \theta^k - \alpha_k g(\theta^k, \xi_k) ./ (n_g + 10^{-8}). \end{aligned}$$

Issue: The learning rate (step size) goes to zero quickly.

RMSProp: “Leaky AdaGrad”

Key idea: Discount the accumulated norm of the gradients.

Start from some $\theta^0 \in \mathbb{R}^s$, $n_g = 0$, for $k \geq 0$:

$$
\begin{aligned}
n_g &= \gamma n_g + (1-\gamma) g(\theta^k, \xi_k) .* g(\theta^k, \xi_k), \\
\theta^{k+1} &= \theta^k - \alpha_k g(\theta^k, \xi_k) ./ (n_g + 10^{-8}).
\end{aligned}
$$

ADAM

Key idea: Consider momentum and adaptive learning rate (second-order momentum) together.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Recommended Hyper-parameter Settings in ADAM

Adam: A Method for Stochastic Optimization,

Diederik P. Kingma, Jimmy Ba,

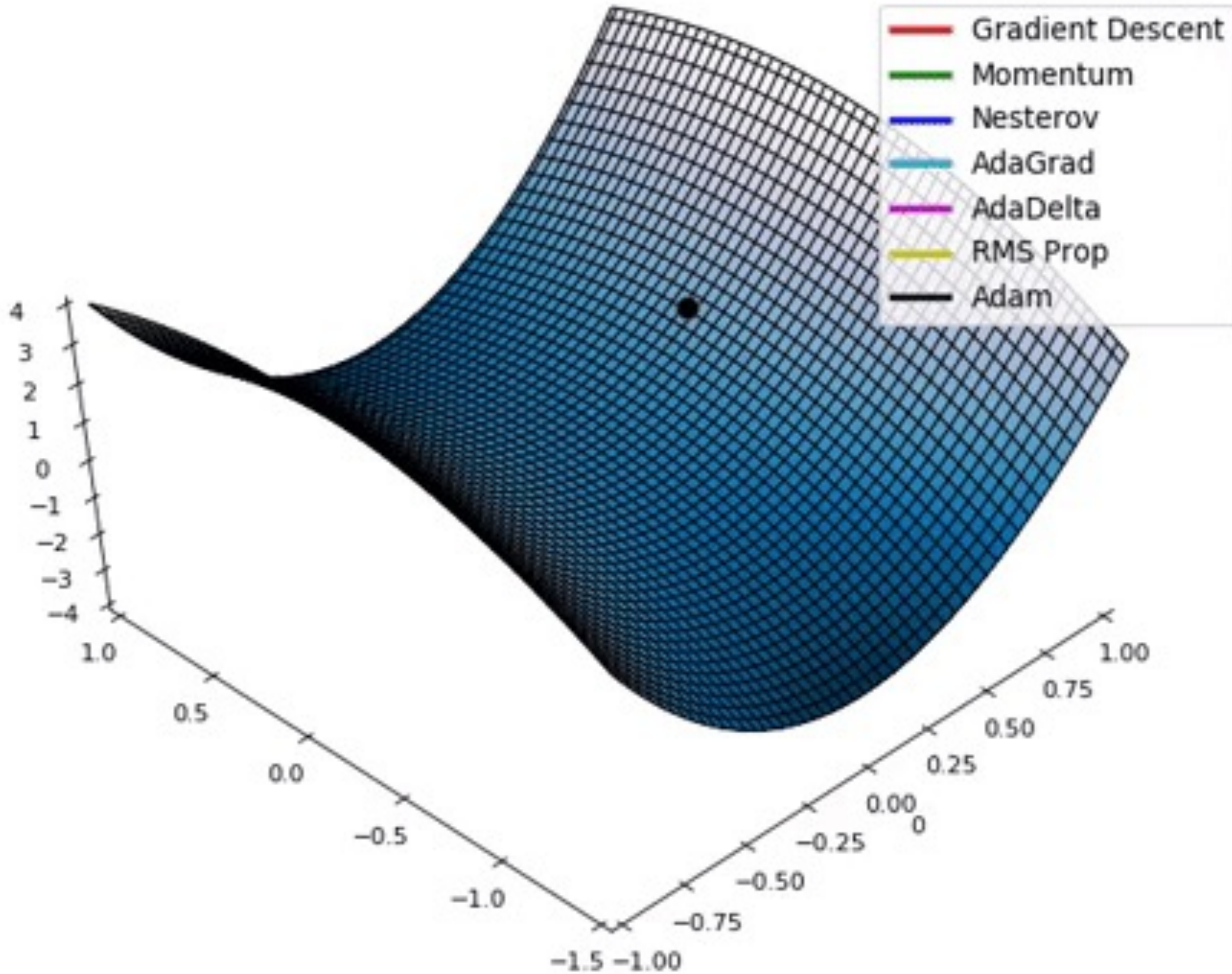
International Conference for Learning Representations, 2015

Google Citation: 130,829

In the original paper of ADAM, the following hyper-parameter settings are recommended:

$$\boldsymbol{\alpha = 0.001, \quad \beta_1 = 0.9, \qquad \beta_2 = 0.999, \qquad \epsilon = 10^{-8}.}$$

Numerical Comparison



Stories about ADAM

ADAM is arguably **the most popular** optimization algorithm in training deep neural networks now. But the convergence analysis contains some **mistakes** in the original paper. ADAM can be **non-convergent** !

S. J. Reddi, S. Kale, and S. Kumar.

On the convergence of adam and beyond.

International Conference for Learning Representations, 2018

Best Paper Award !

Stories about ADAM

RMSProp can be convergent for large parameters ($\boldsymbol{\beta_2}$)!

N. Shi, D. Li, M. Hong, and R. Sun.

RMSprop converges with proper hyper-parameter.

International Conference on Learning Representations, 2020.

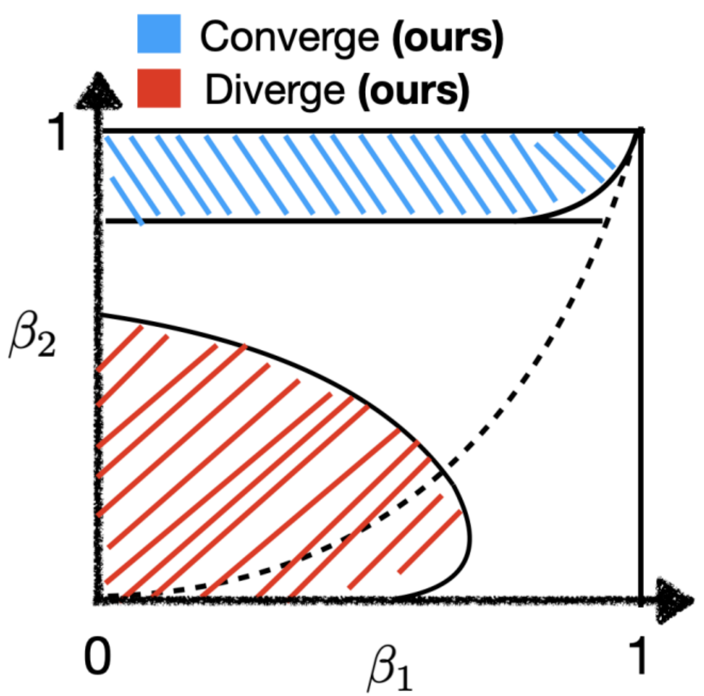
Not address the issue for ADAM since they set $\boldsymbol{\beta_1 = 0}$.

Stories about ADAM

Y. Zhang, C. Chen, N. Shi, R. Sun, Z.-Q. Luo

Adam Can Converge Without Any Modification on Update Rules.

NeurIPS 2022



Practical Implementation Tricks

Minibatch: Instead of sampling one random gradient $g(\theta^k, \xi_k)$, we sample **p** random gradient $g(\theta^k, \xi_{k_1}), \ldots, g\left(\theta^k, \xi_{k_p}\right)$, **Update**

$$\theta^{k+1} = \theta^k - \alpha_k \frac{1}{p} \sum_{i=1}^{p} g(\theta^k, \xi_{k_i}).$$

Practical Implementation Tricks

Minibatch: Instead of sampling one random gradient $g(\theta^k, \xi_k)$, we sample **p** random gradient $g(\theta^k, \xi_{k_1}), \ldots, g\left(\theta^k, \xi_{k_p}\right)$, **Update**

$$\theta^{k+1} = \theta^k - \alpha_k \frac{1}{p} \sum_{i=1}^{p} g(\theta^k, \xi_{k_i}).$$

Epoch: a central concept in training. In each epoch, $\boldsymbol{n_E}$ SGD updates will be executed. Usually, we select

$$\boldsymbol{n_E = ceil(n/p).}$$

Practical Implementation Tricks

Minibatch: Instead of sampling one random gradient $g(\theta^k, \xi_k)$**,** we sample **p** random gradient $g(\theta^k, \xi_{k_1}), \dots, g\left(\theta^k, \xi_{k_p}\right)$, **Update**

Epoch: a central concept in training. In each epoch, $\boldsymbol{n_E}$ SGD updates will be executed. Usually, we select

$$\boldsymbol{n_E = ceil(n/p).}$$

Dynamic Step Size Adjusting:

(a) Decrease the step size by ratio $0 < \gamma < 1$ every K epochs.

(b) **Epoch Doubling Strategy**: Run K epochs with step size α, then, run 2K epochs with step size $\alpha/2$, ……