Deep Learning

Angshuman Paul

Assistant Professor

Department of Computer Science & Engineering

Syllabus

Theory

- **Neural networks:** DL Optimizers (SGD, MBGD, AdaGrad, Adam) and Regularization, Initialization Methods
- **DL Models:** Autoencoder, Convolutional Neural Networks, Recurrent Neural Networks, LSTM, Network Architecture Search (NAS)
- **Deep Generative Models:** Deep Belief Networks, Variational Autoencoders, Generative Adversarial Networks, Deep Convolutional GAN
- **Representation learning:** Unsupervised Pre-training, Transfer learning and Domain adaptation, Distributed representation, Discovering underlying causes

Lab

- Autoencoder, CNN, LSTM, VAE, GANs (variants), Transfer Learning, NLM, Graph NN, Adversarial losses

Course Logistics

- Instructor: Angshuman Paul
- Contact: apaul@iitj.ac.in
- TA Team
  - Jayant Mahawar
  - Obed Jamir
  - Avadhut Eknath Kabadi
  - Kalpesh Soni
- Lab Instructors
  - Angshuman Paul
  - Yashaswi Verma
- Class hours:
  - Monday: 10-10:50 AM
  - Wednesday: 10-10:50 PM
  - Thursday: 10-10:50 PM
- Lab Hours
  - Tuesday: 2-5 PM
  - Wednesday: 2-5 PM
  - Thursday: 1-4 PM

Evaluation Scheme (Tentative)

- Course project (10%)
- Quiz (10%)
- Viva (10%)
- Minor (10%)
- Major (40%)
- Class Notes (5%)
- Lab (15%)
- Slides will be shared every Monday

Course Project

- Course project
  - Group of 3 or 4
    - Groups must be formed by 10/01/25
  - You may choose your own project (needs approval)
  - You may choose from a list of projects (FCFS)
  - **Deadline for final report+ code + other materials**
    - **March 31 (no extension under any circumstances)**
  - Course Project + Theory Viva: Between 05-15 April

Lab Sessions

- Each student must choose one of the three lab sessions
  - Must attend the lab in the same lab session every week
- Each lab session
  - Discussion on experiments
  - Hands-on example
  - Assignment (to be submitted within 3 days from the date of the lab)
  - **Viva from last few submitted assignments**
- You must bring charged laptops
- Grading will be based on
  - Viva
  - Submitted codes

Books and Study Materials

- Book:
  - Christopher M. Bishop, Hugh Bishop (2023), Deep Learning Foundations and Concepts, Springer
  - I. Goodfellow, Y. Bengio, A. Courville (2016), Deep Learning, The MIT Press, 1st Edition.
  - A. Zhang, Z. Lipton, M. Li, A. Smola (2020) Dive into Deep Learning (Release 0.7.1), https://d2l.ai/d2l-en.pdf

- Other useful books:
  - D. FOSTER (2019), Generative Deep Learning, O'Reilly Media, 1st Edition

- Other books & Online materials:
  - https://www.youtube.com/watch?v=RLH2meHRHHc&list=PLehuLRPyt1HxuYpdlW4KevYJVOSDG3DEz
  - https://www.youtube.com/playlist?list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI
  - https://www.youtube.com/watch?v=XTWPyW2mTUg&list=PLehuLRPyt1HxTolYUWeyyIoxDabDmaOSB

Prerequisites

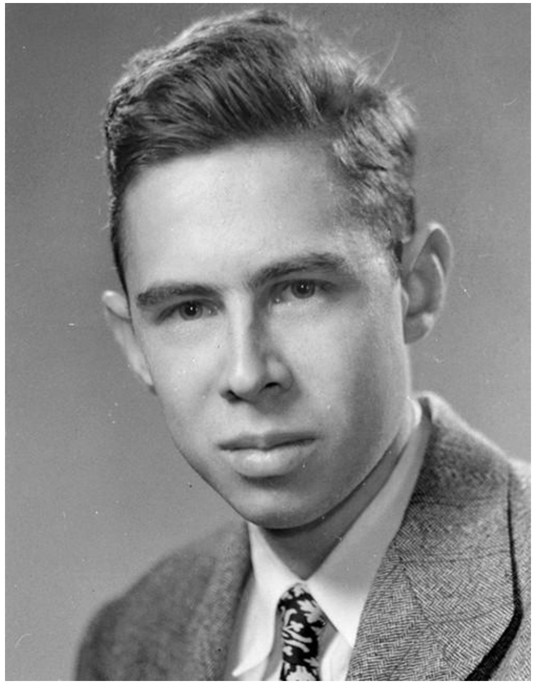
- IML/ PRML
- Basics of
  - Linear Algebra
  - Calculus
  - Probability

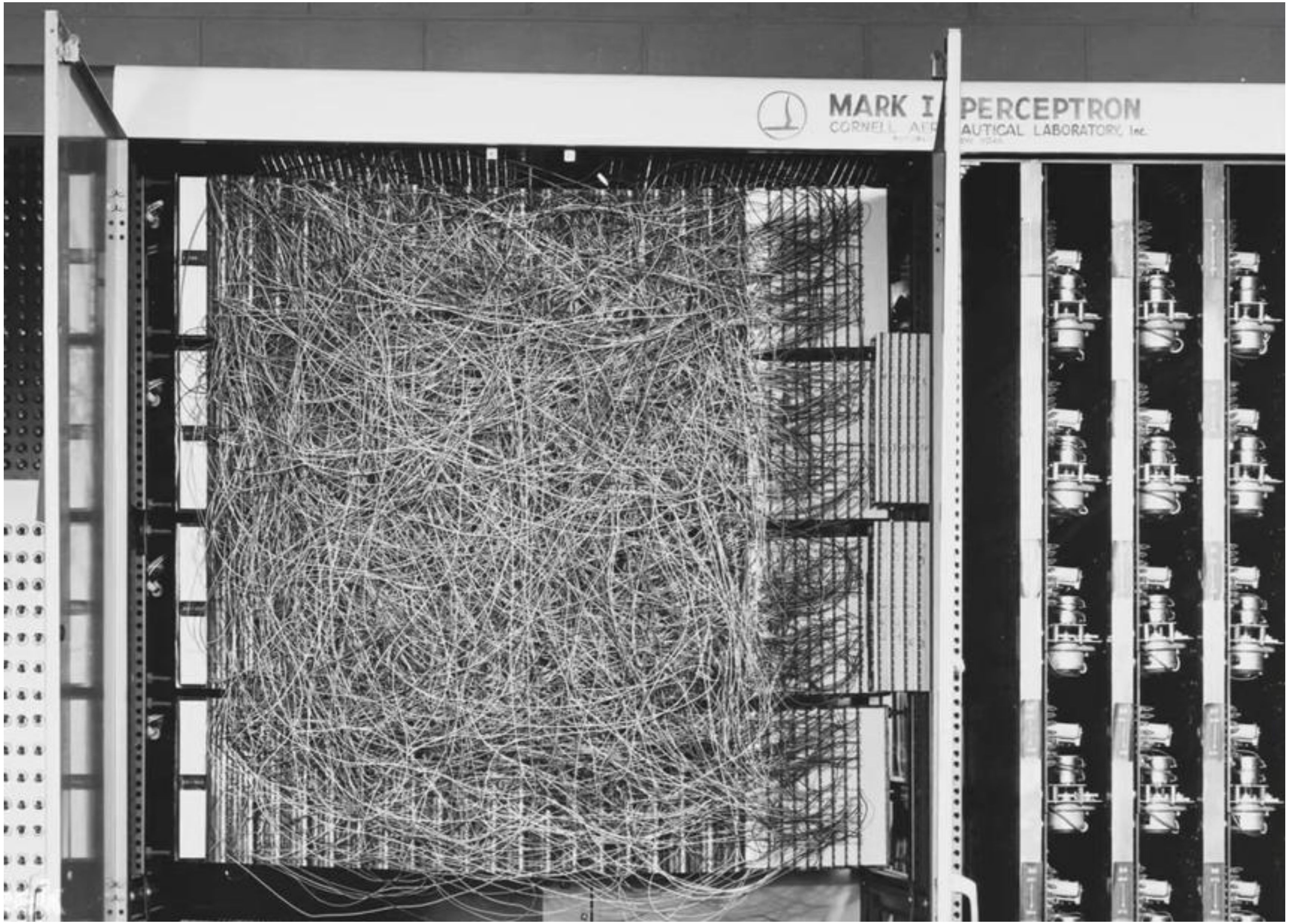
Why Deep Learning?

Why Deep Learning?

- Arguably the most successful ML approach
    - Supervised learning
    - Unsupervised/ weakly-supervised learning
    - Reinforcement learning

- Useful designs exist for handling different types of data
    - Attribute-based data: deep FCNs
    - Image: CNNs, Vision Transformers
    - Audio/ video/ other sequence data: RNN, LSTM, GRU, Transformers

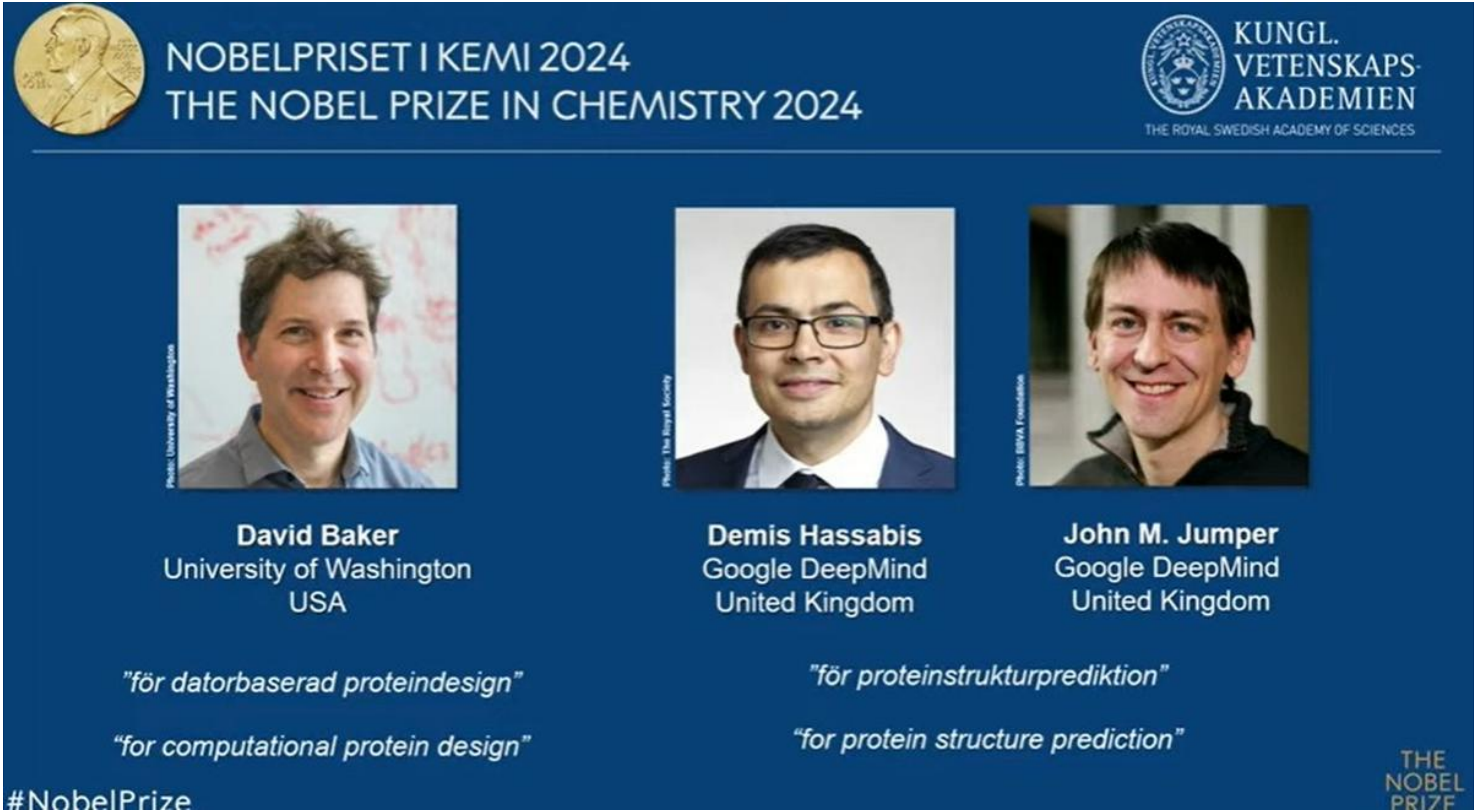




https://images.app.goo.gl/NZeehjDfmTzT4HVa7

https://www.youtube.com/watch?v=l9RWTMNnvi4&t=6s

https://images.app.goo.gl/9PmTEGvjHSMTVX2E6



https://images.app.goo.gl/8ncFJUXP5K6SRC769

https://images.app.goo.gl/sUAZ7q7vudgjT8WFA

Massachusetts Institute of Technology

Education | Research | Innovation | Admissions + Aid | Campus Life | News | Alumni | About MIT

MIT News
ON CAMPUS AND AROUND THE WORLD

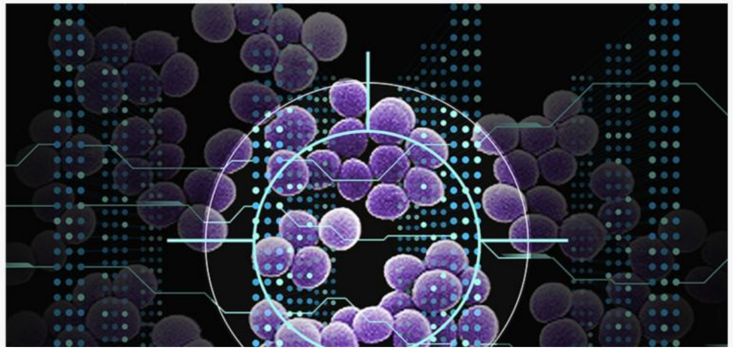
SUBSCRIBE | BROWSE | SEARCH NEWS

Using AI, MIT researchers identify a new class of antibiotic candidates

These compounds can kill methicillin-resistant Staphylococcus aureus (MRSA), a bacterium that causes deadly infections.

Anne Trafton | MIT News
December 20, 2023

PRESS INQUIRIES

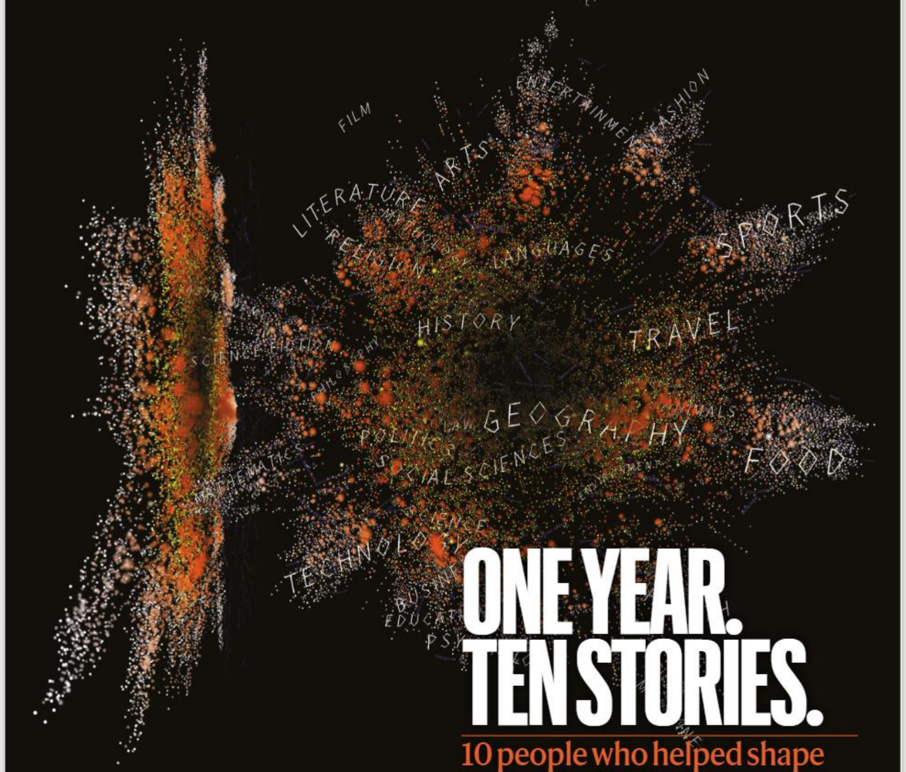


Using a type of artificial intelligence known as deep learning, MIT researchers have discovered a class of compounds that can kill a drug-resistant bacterium that causes more than 10,000 deaths in the United States every year.

Image: Christine Daniloff, MIT; Janice Haney Carr, CDC; iStock

The international journal of science / 21/28 December 2023

nature



ONE YEAR. TEN STORIES.

10 people who helped shape science in 2023

Gross habit
Why the world needs to rethink its addiction to GDP growth

Artificial chemist
Large language model designs and performs chemical experiments

Deep breath
A next-generation inhalable vaccine for COVID-19

Vol. 624, No. 7992
nature.com

2. AI finally starting to feel like AI

The stratospheric ascent of OpenAI's ChatGPT this year has prompted furious debate in the media and elsewhere about the future role of artificial intelligence and its implications for everything from employment to healthcare. Photograph: Lionel Bonaventure/AFP/Getty Images

It's often hard to spot technological watersheds until long after the fact, but 2023 is one of those rare years in which we can say with certainty that the world changed. It was the year in which artificial intelligence (AI) finally went mainstream. I'm referring, of course, to ChatGPT and its stablemates - large language models. Released late in 2022, ChatGPT went viral in 2023, dazzling users with its fluency and seemingly encyclopaedic knowledge. The tech industry - led by trillion-dollar companies - was wrong-footed by the success of a product from a company with just a few hundred employees. As I write, there is desperate jostling to take the lead in the new "generative AI" marketplace heralded by ChatGPT.

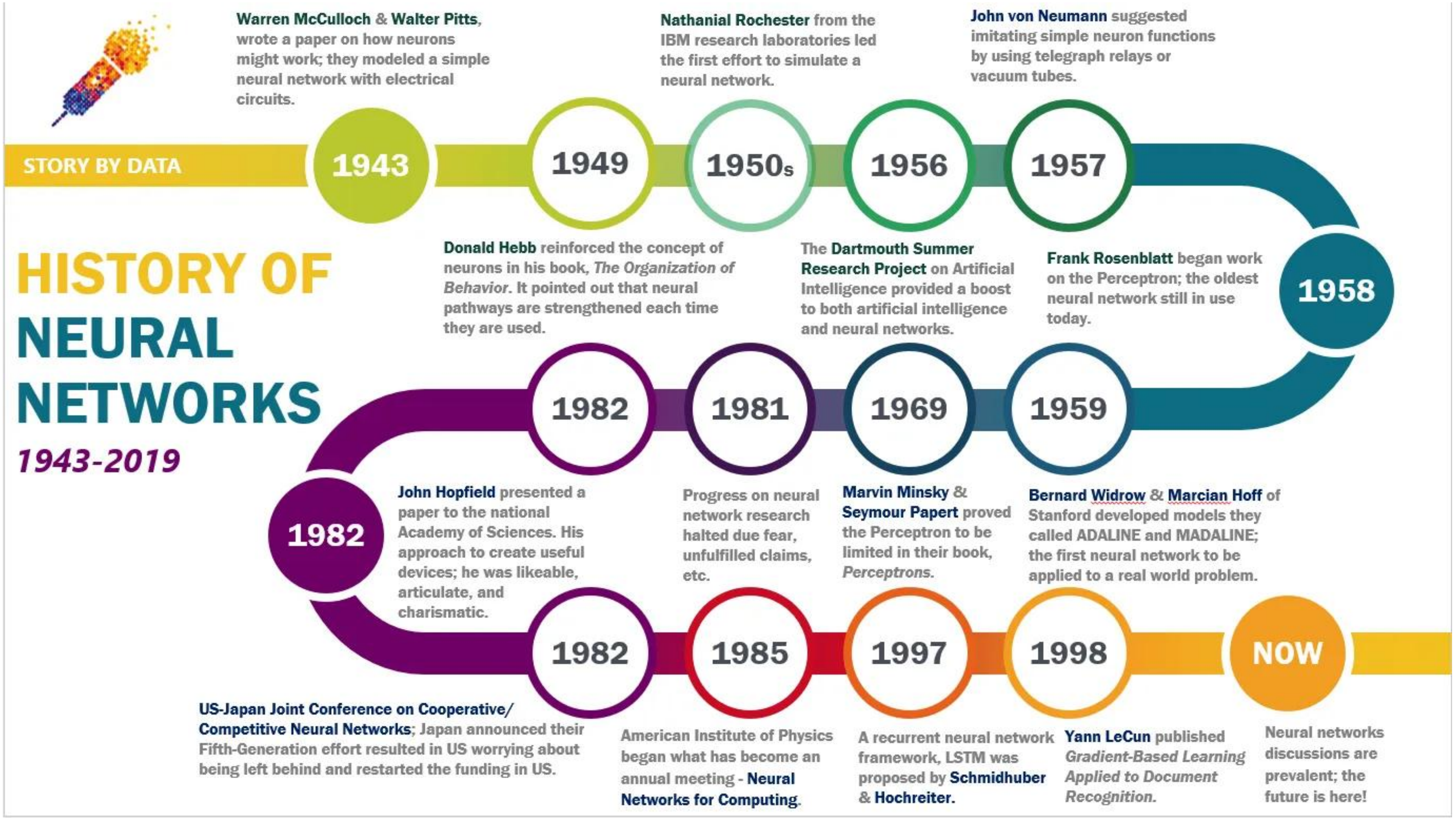
Why Deep Learning?

- Useful for different types of tasks
  - Classification
  - Regression
  - Detection
  - Segmentation
  - Translation
  - Feature extraction
  - **Generation**
  - Many more …

Is it Only the Success Story?

- Explainability
- Generalizability
- Computational requirements
- Data requirements

A Brief History of Neural Networks



https://medium.com/analytics-vidhya/brief-history-of-neural-networks-44c2bf72eec#:~:text=The%20first%20step%20towards%20neural,neural%20network%20with%20electrical%20circuits.

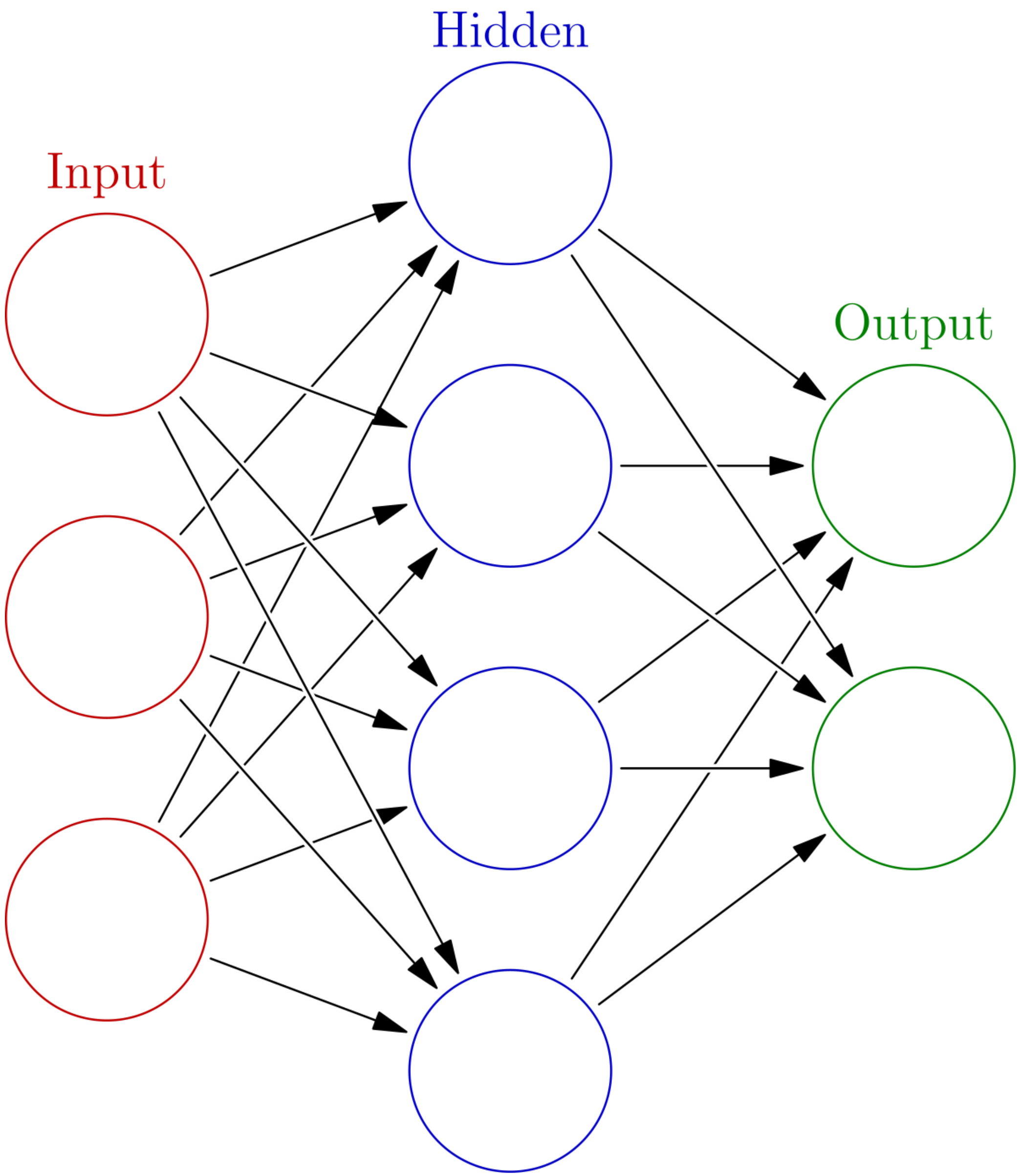
Why Now?

- Availability of hardware
  - GPUs
  - Storage
  - Cloud services

- Availability of large datasets

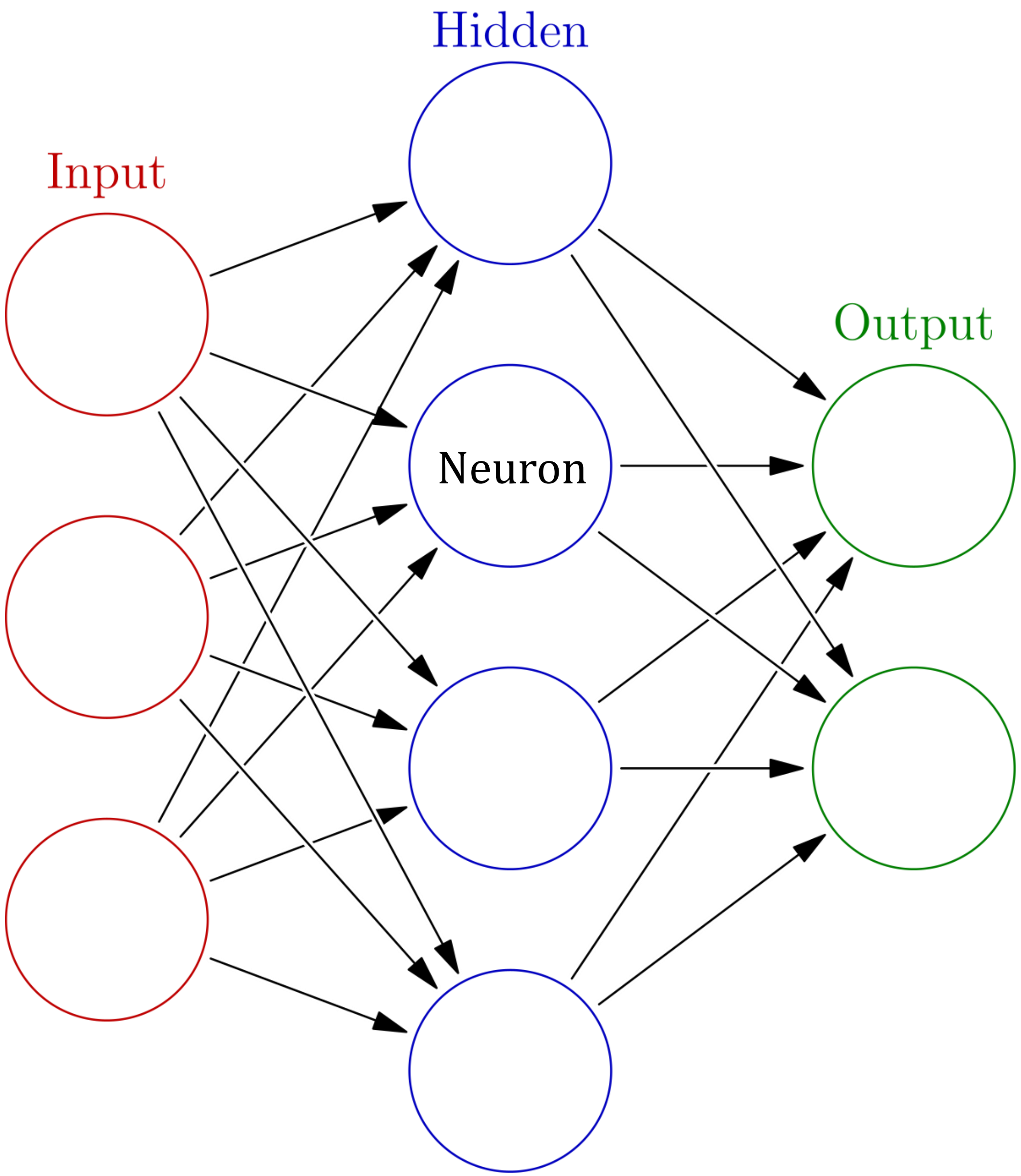
- Availability of implementation platforms
  - PyTorch
  - Tensorflow

What is a Neural Network?



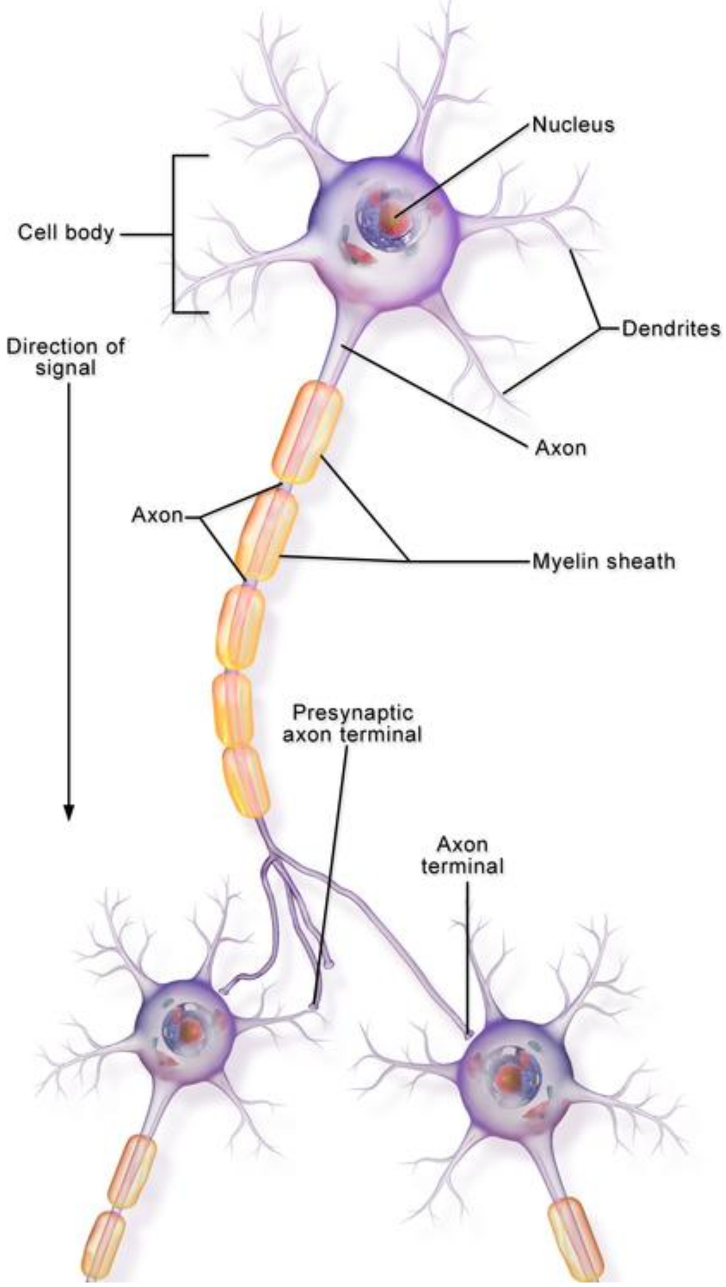
https://en.wikipedia.org/wiki/Artificial_neural_network

What is a Neural Network?



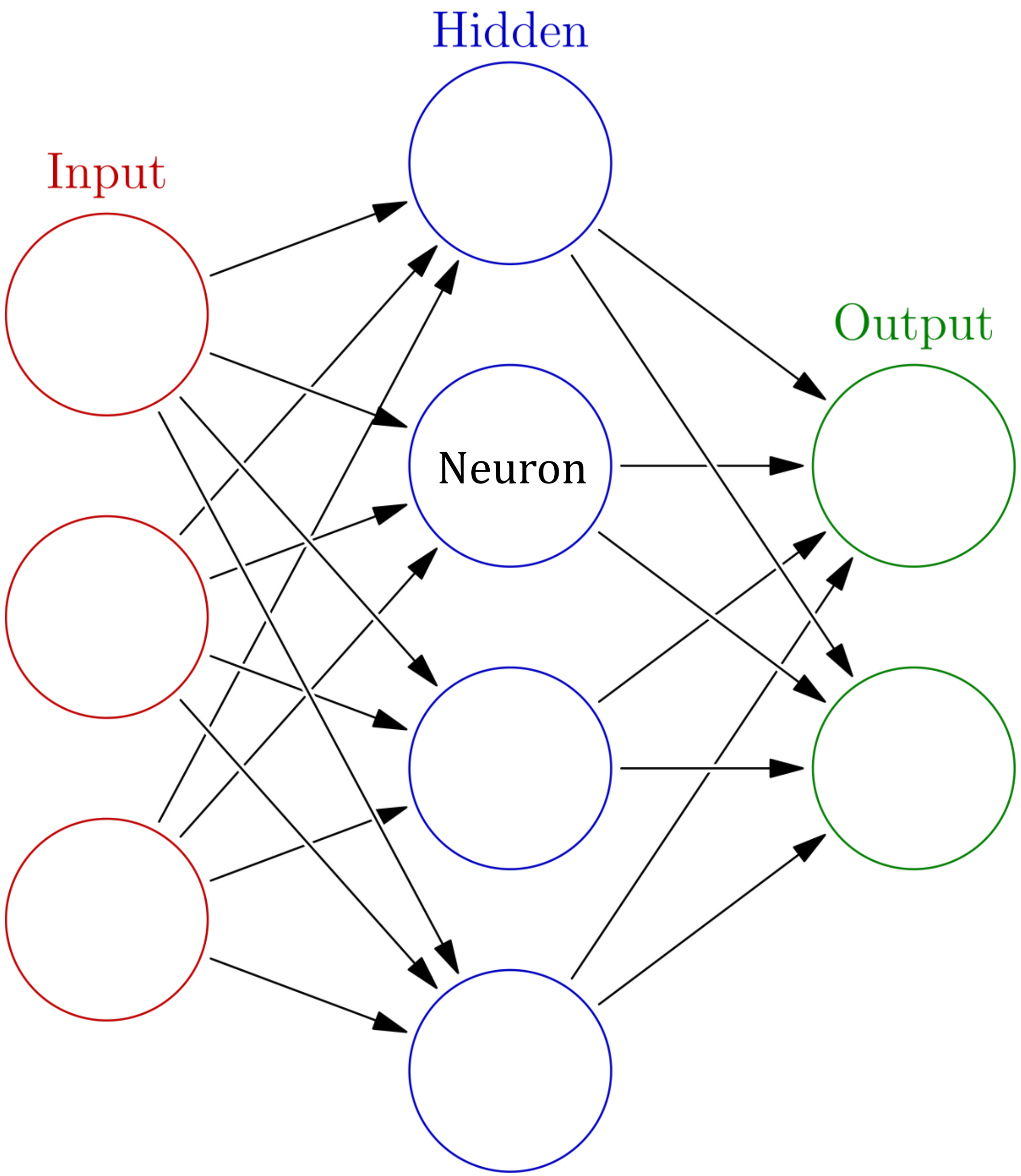
https://en.wikipedia.org/wiki/Artificial_neural_network

Neuron: The Biological Context



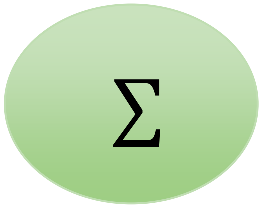
https://www.physio-pedia.com/Neurone

What is a Neural Network?

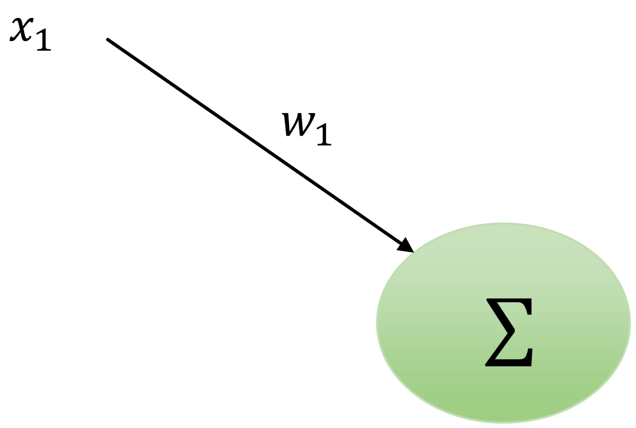


https://en.wikipedia.org/wiki/Artificial_neural_network

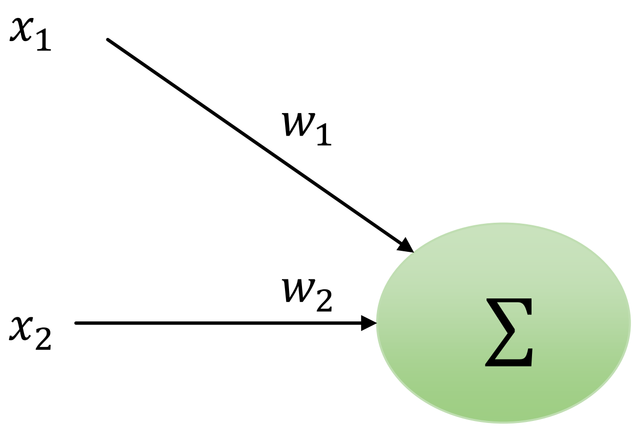
What Happens Inside a Neuron?



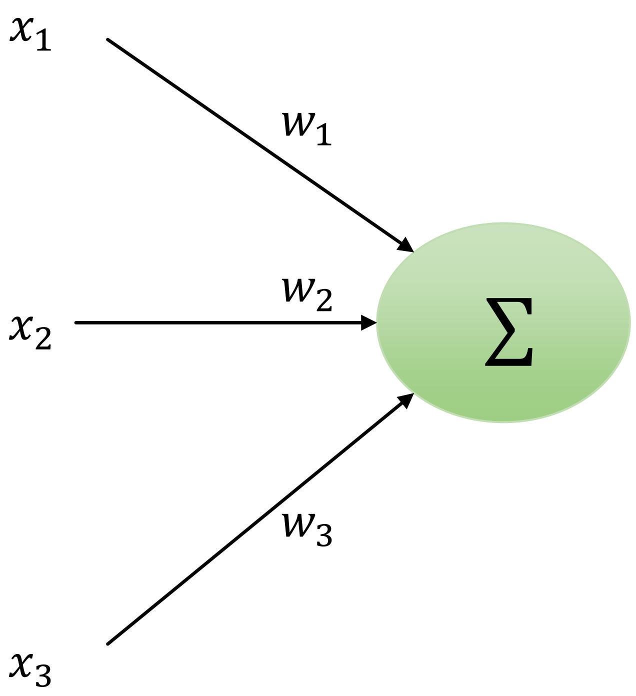
Neuron: The Computational Context



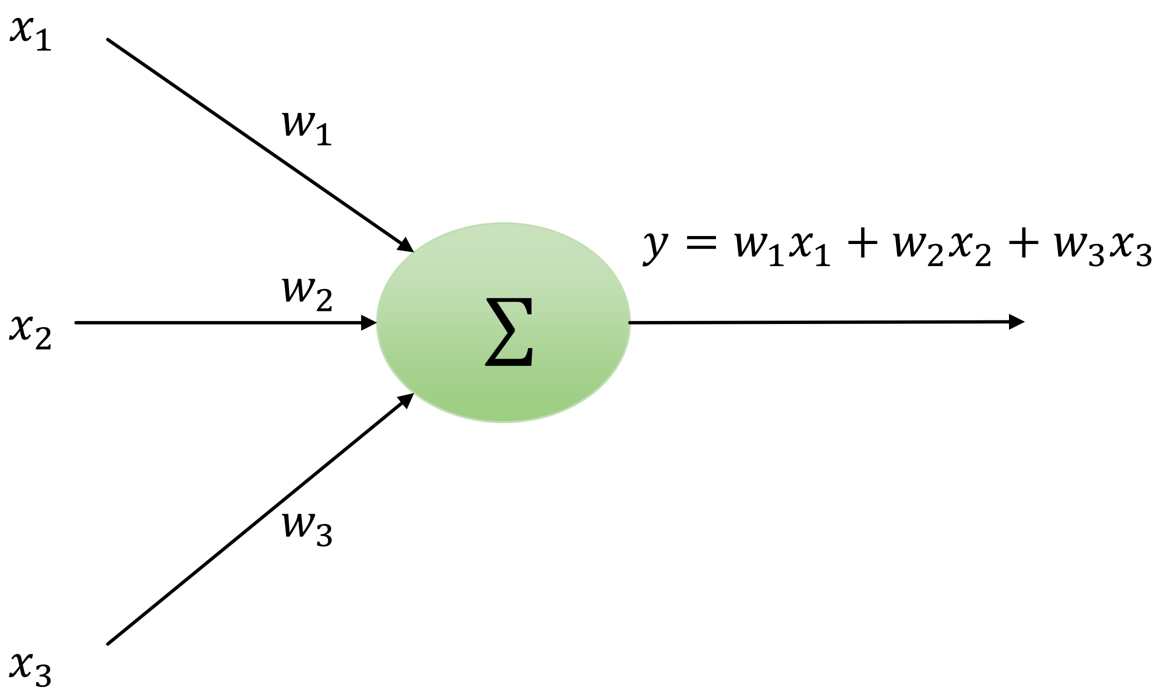
Neuron: The Computational Context



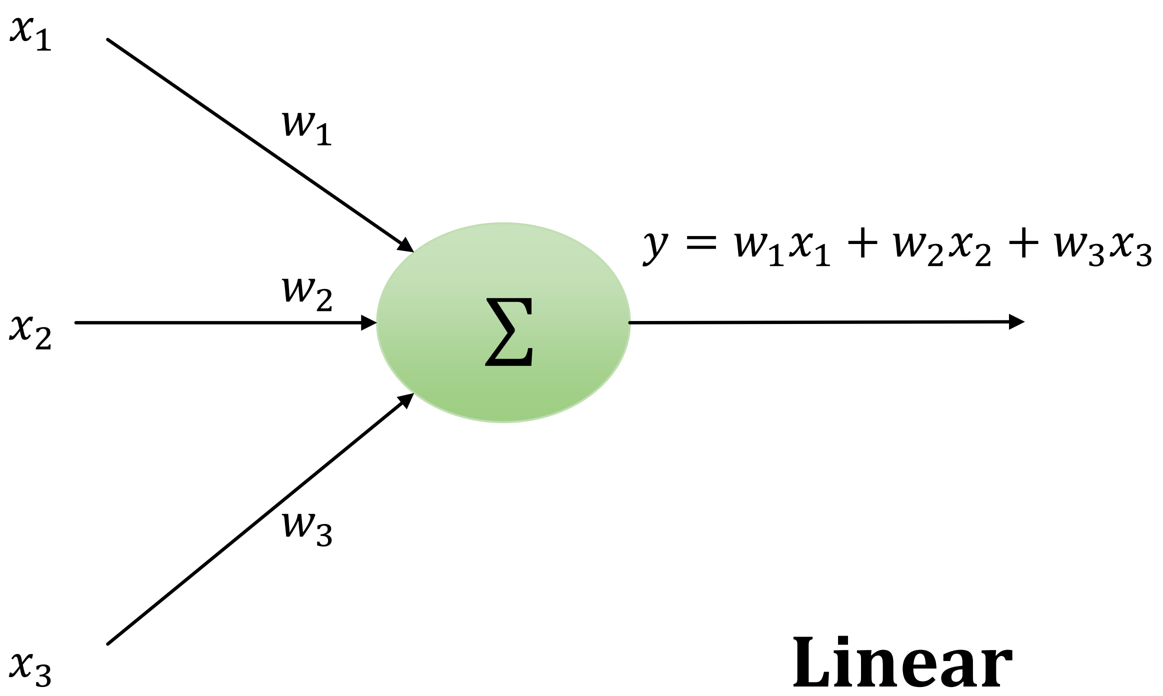
Neuron: The Computational Context



Neuron: The Computational Context

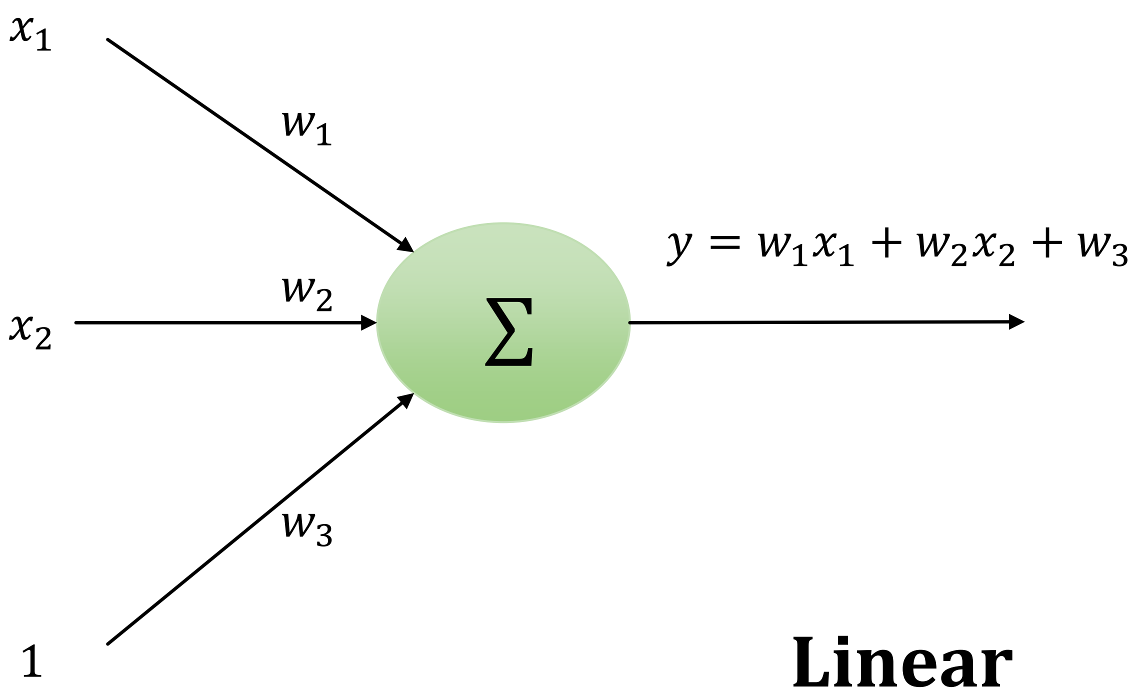


Neuron: The Computational Context



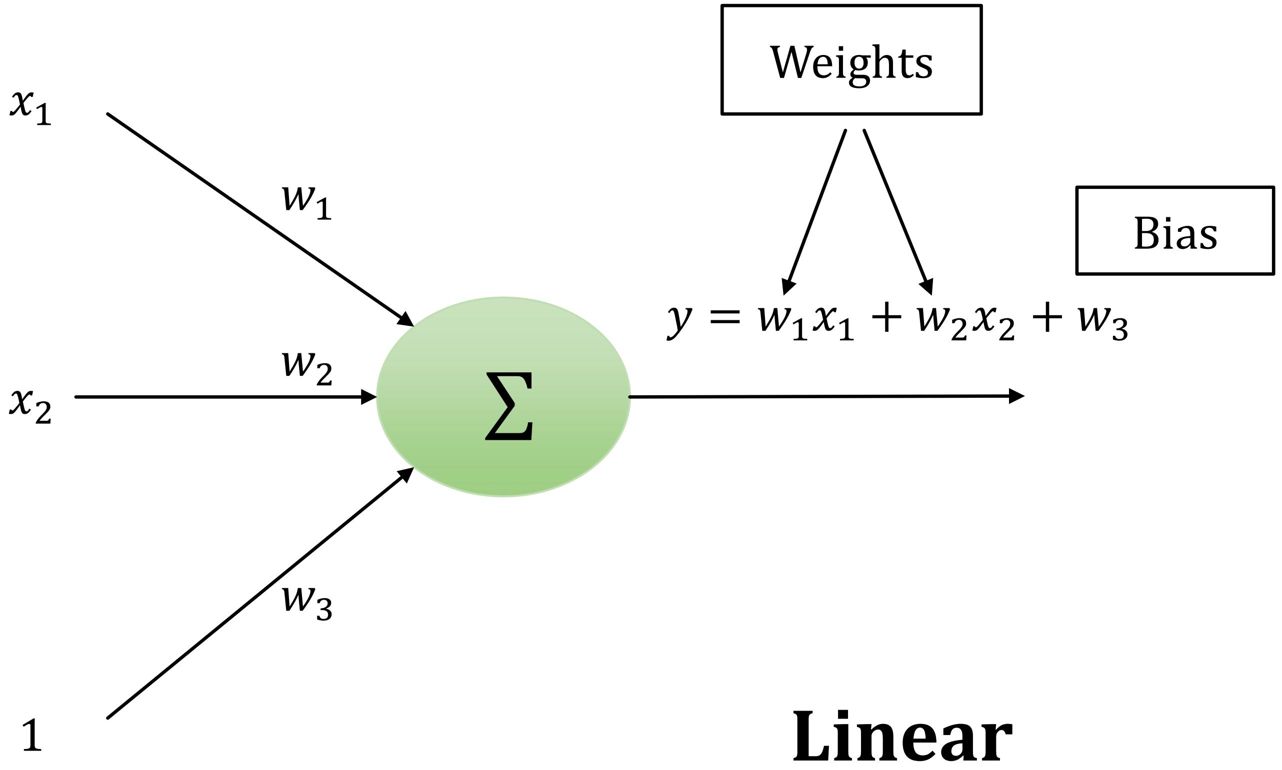
Linear

Neuron: The Computational Context

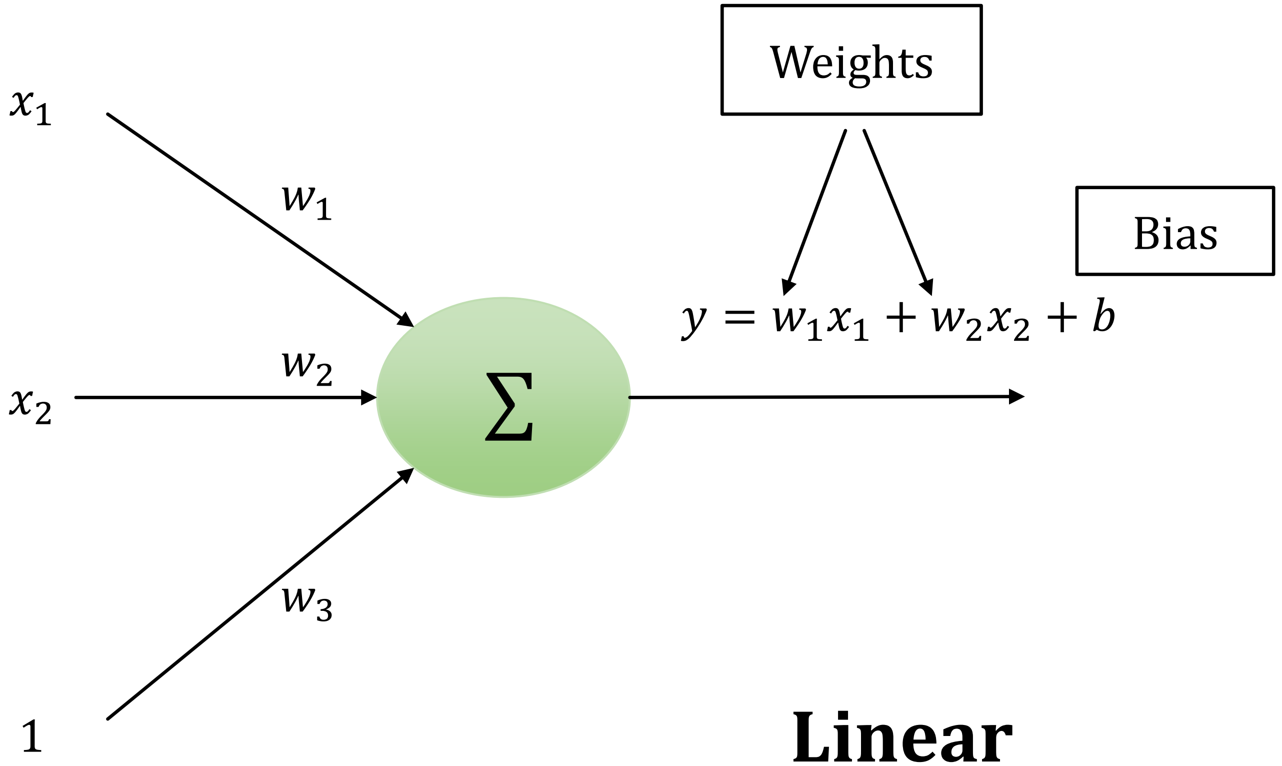


Linear

Neuron: The Computational Context

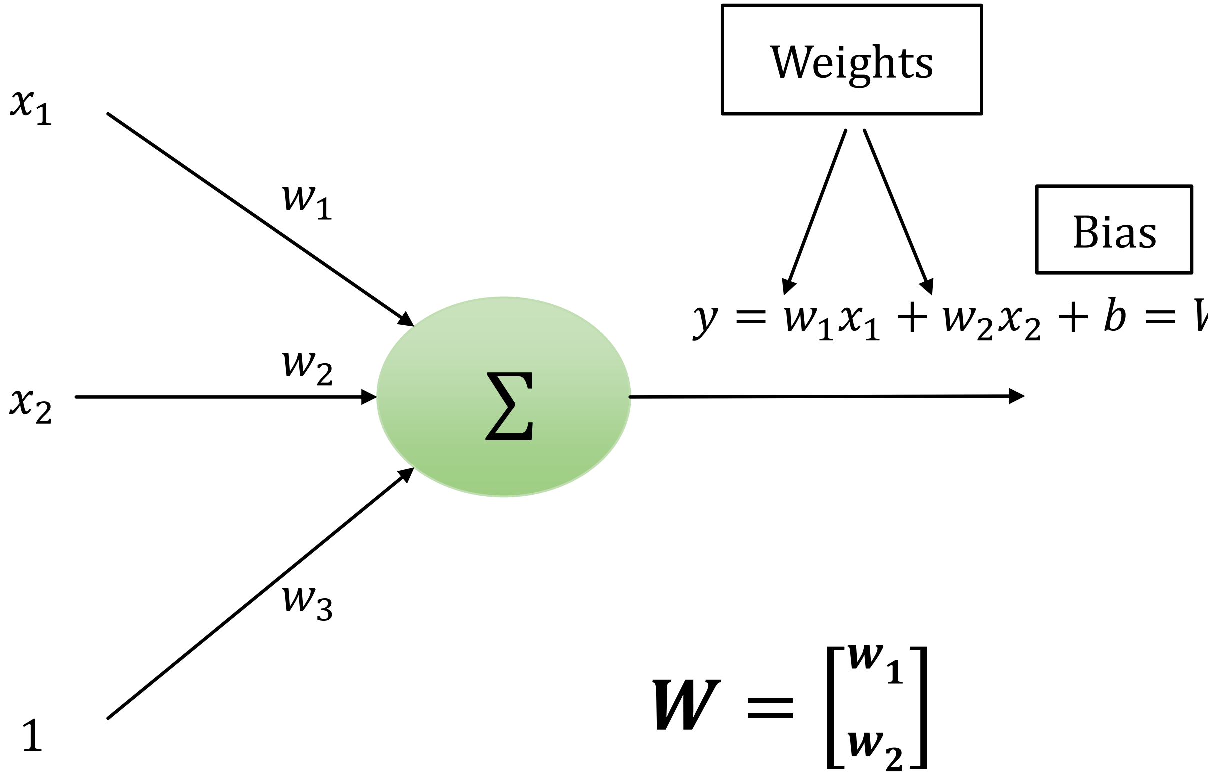


Neuron: The Computational Context



Linear

Neuron: The Computational Context

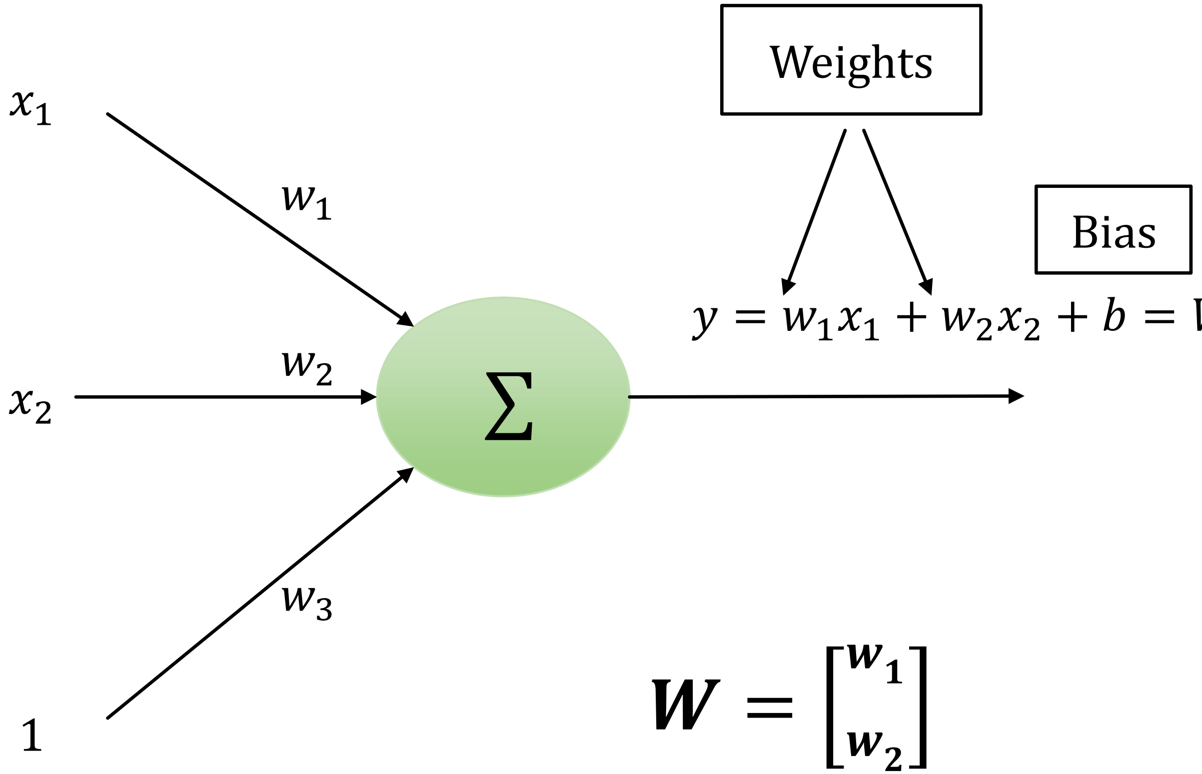


$$y = w_1 x_1 + w_2 x_2 + b = W^T x + b$$

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w_1} \\ \boldsymbol{w_2} \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x_1} \\ \boldsymbol{x_2} \end{bmatrix}$$

Linear

What Can the NN Learn?



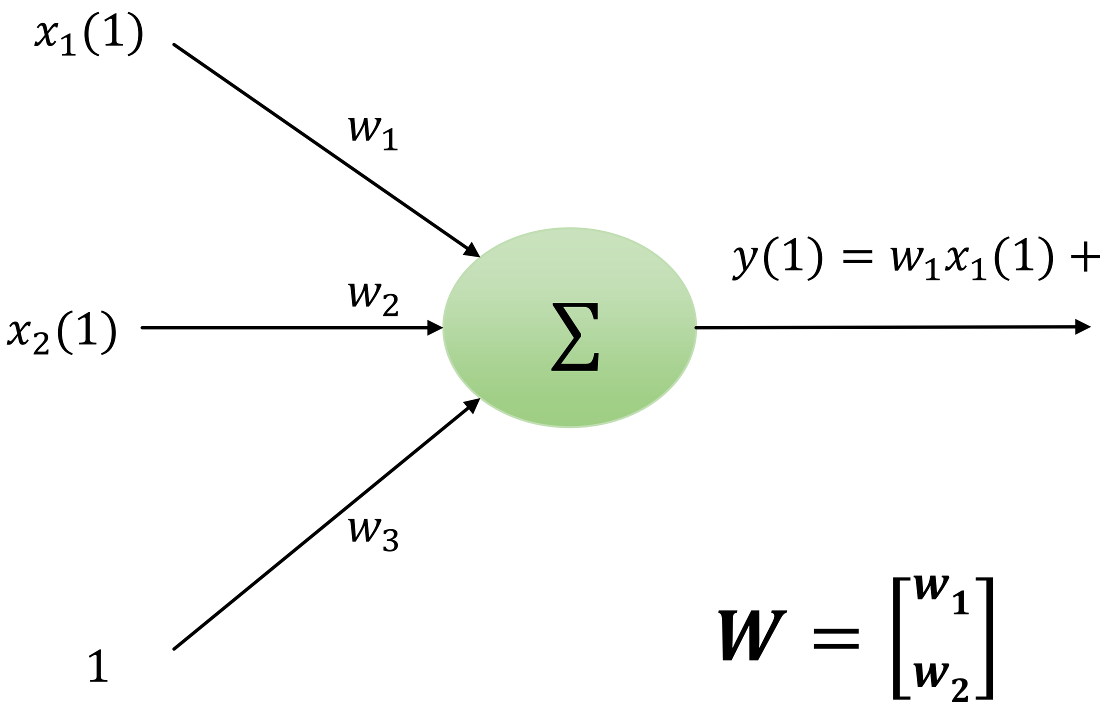
$$y = w_1 x_1 + w_2 x_2 + b = W^T x + b$$

$$\boldsymbol{W} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

NN has to learn *W* and *b*

The Input-Output Relationship

$\boldsymbol{x(1)}$

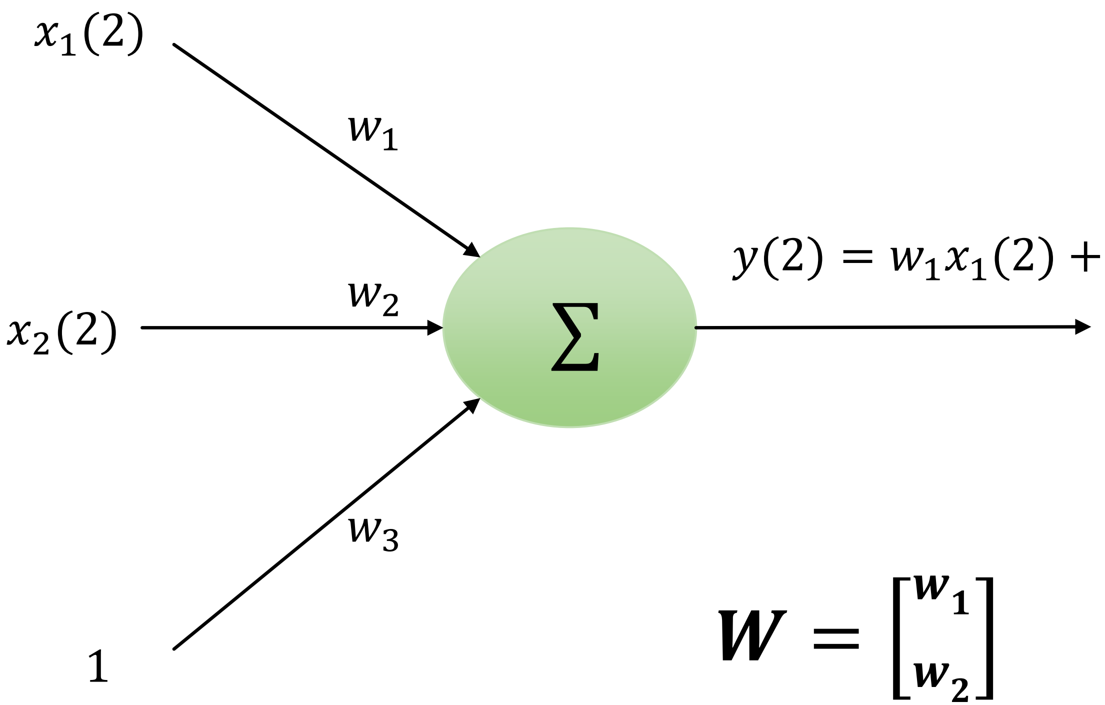


$$y(1) = w_1 x_1(1) + w_2 x_2(1) + b = W^T x(1) + b$$

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w_1} \\ \boldsymbol{w_2} \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x_1(1)} \\ \boldsymbol{x_2(1)} \end{bmatrix}$$

The Input-Output Relationship

$\boldsymbol{x(2)}$

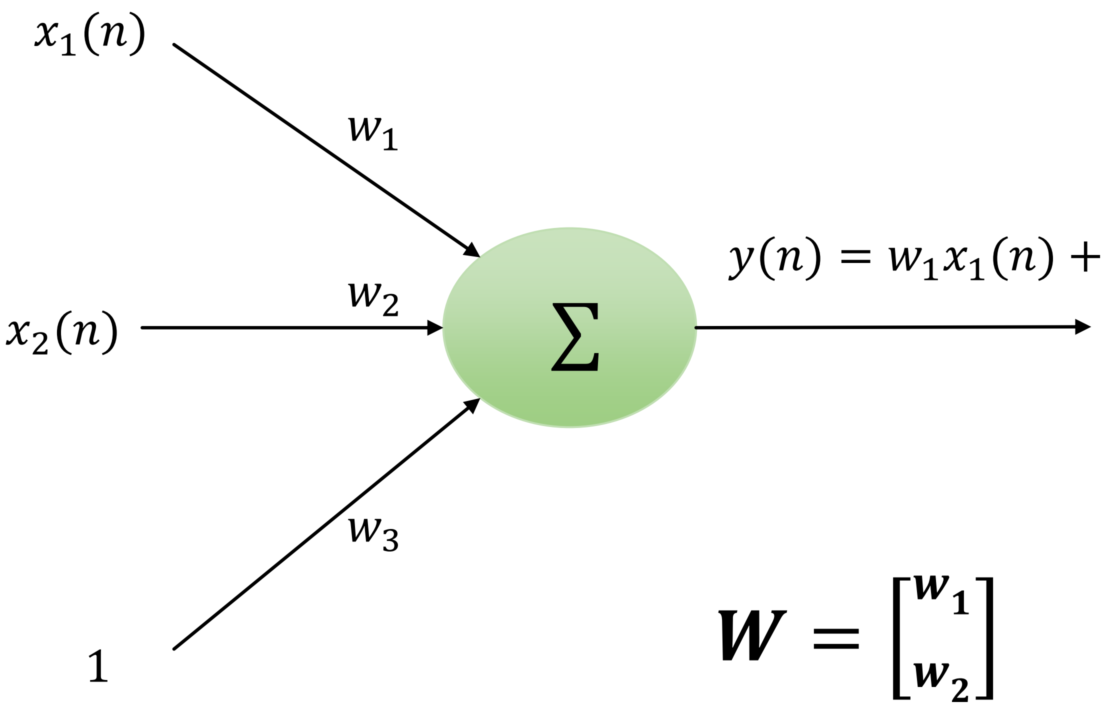


$$y(2) = w_1 x_1(2) + w_2 x_2(2) + b = W^T x(2) + b$$

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w_1} \\ \boldsymbol{w_2} \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x_1(2)} \\ \boldsymbol{x_2(2)} \end{bmatrix}$$

The Input-Output Relationship

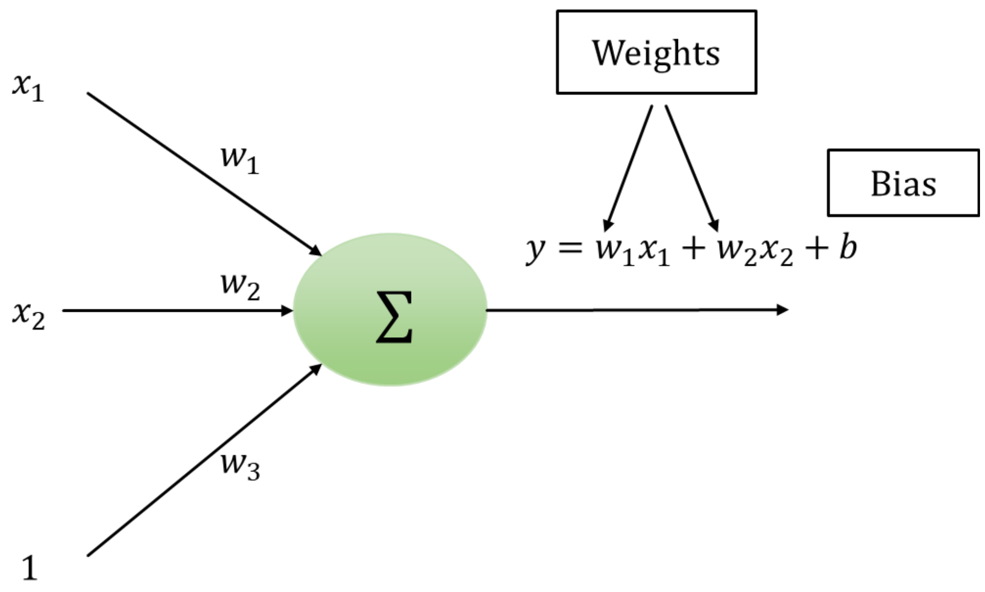
$\boldsymbol{x(n)}$



$$y(n) = w_1 x_1(n) + w_2 x_2(n) + b = W^T x(n) + b$$

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w_1} \\ \boldsymbol{w_2} \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x_1(n)} \\ \boldsymbol{x_2(n)} \end{bmatrix}$$

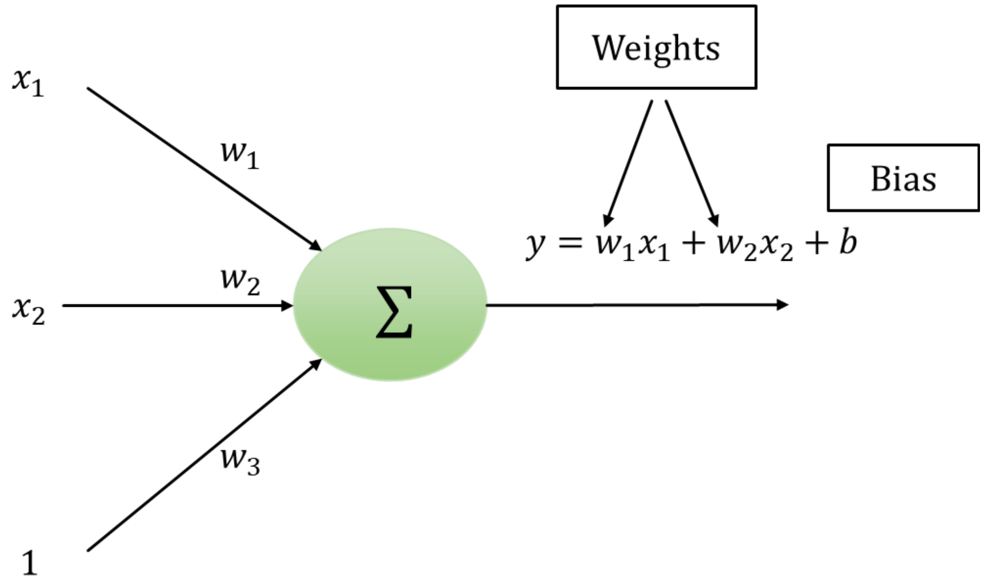
How can the NN Learn?



Consider n input samples

$$X = [\boldsymbol{x}(\mathbf{1}) \quad \boldsymbol{x}(\mathbf{2}) \ \ldots \ \boldsymbol{x}(\boldsymbol{n})]$$

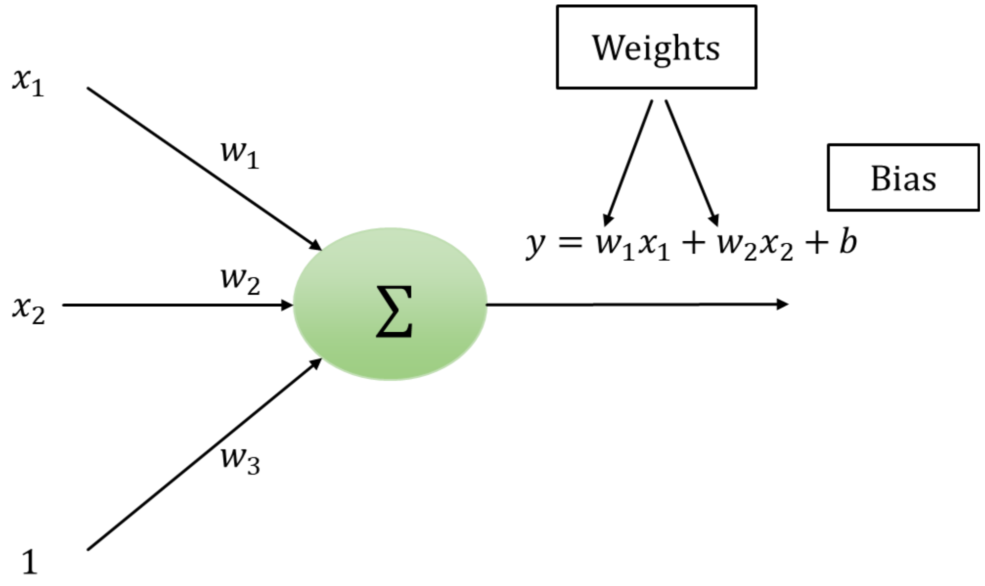
How can the NN Learn?



Consider n input samples

$$X = \begin{bmatrix} \boldsymbol{x_1(1)} & \boldsymbol{x_1(2)} & \dots & \boldsymbol{x_1(n)} \\ \boldsymbol{x_2(1)} & \boldsymbol{x_2(2)} & \dots & \boldsymbol{x_2(n)} \end{bmatrix}$$

How can the NN Learn?



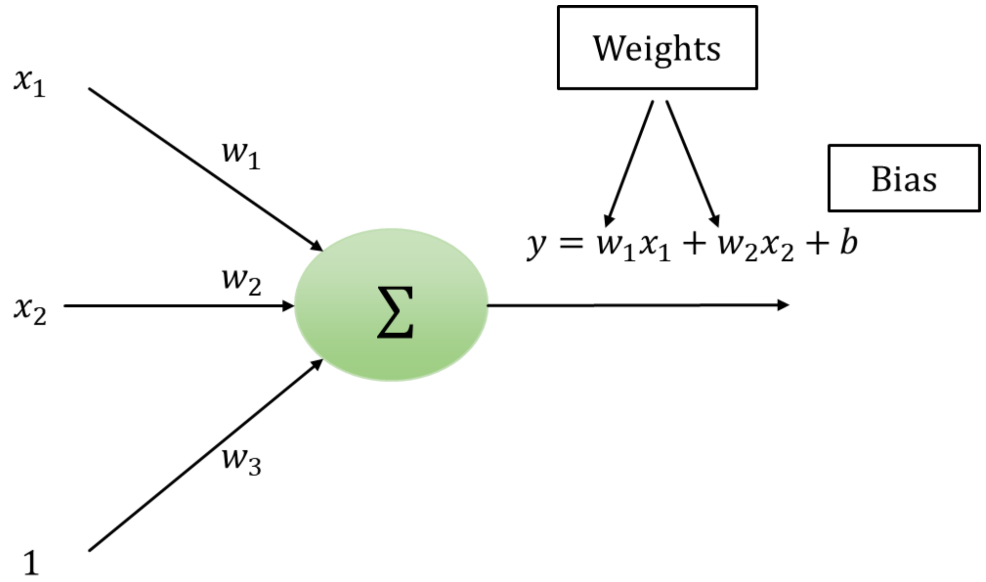
Consider n input samples

$$X = \begin{bmatrix} \boldsymbol{x}_1(\mathbf{1}) & \boldsymbol{x}_1(\mathbf{2}) & \dots & \boldsymbol{x}_1(\boldsymbol{n}) \\ \boldsymbol{x}_2(\mathbf{1}) & \boldsymbol{x}_2(\mathbf{2}) & \dots & \boldsymbol{x}_2(\boldsymbol{n}) \end{bmatrix}$$

The actual observation for X is

$$Y_0 = [y_o(\mathbf{1}) \quad y_o(\mathbf{2}) \quad \dots \quad y_o(\boldsymbol{n})]$$

How can the NN Learn?



Consider n input samples

$$X = \begin{bmatrix} \boldsymbol{x_1(1)} & \boldsymbol{x_1(2)} & \dots & \boldsymbol{x_1(n)} \\ \boldsymbol{x_2(1)} & \boldsymbol{x_2(2)} & \dots & \boldsymbol{x_2(n)} \end{bmatrix}$$

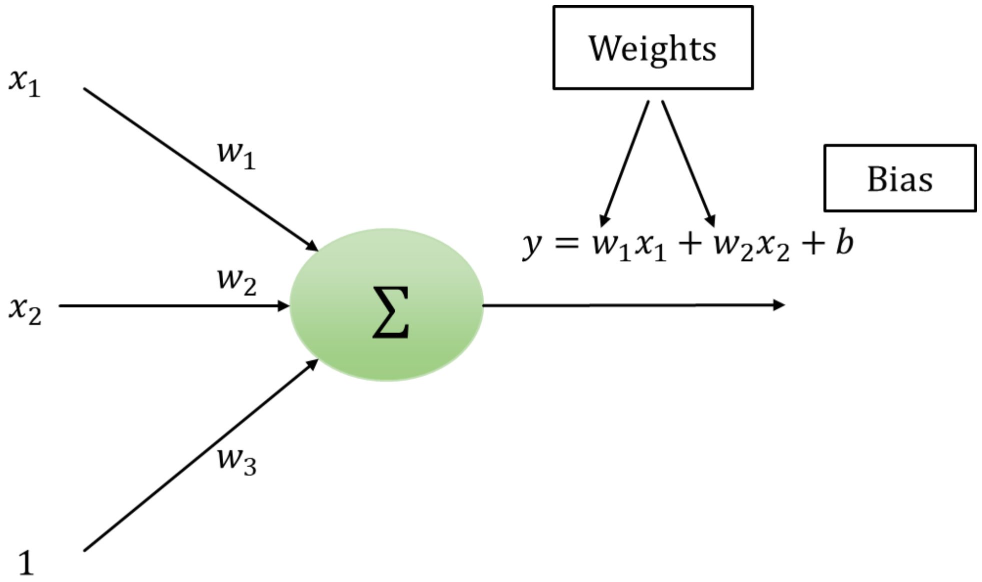
The actual observation for X is

$$Y_0 = [y_o(\mathbf{1}) \quad y_o(\mathbf{2}) \quad \dots \quad y_o(\boldsymbol{n})]$$

When X is applied to the neural network, we get the output

$$Y = [y(\mathbf{1}) \quad y(\mathbf{2}) \quad \dots \quad y(\boldsymbol{n})]$$

How can the NN Learn?



Consider n input samples

$$X = \begin{bmatrix} \boldsymbol{x}_1(\mathbf{1}) & \boldsymbol{x}_1(\mathbf{2}) & \dots & \boldsymbol{x}_1(\boldsymbol{n}) \\ \boldsymbol{x}_2(\mathbf{1}) & \boldsymbol{x}_2(\mathbf{2}) & \dots & \boldsymbol{x}_2(\boldsymbol{n}) \end{bmatrix}$$

The actual observation for X is

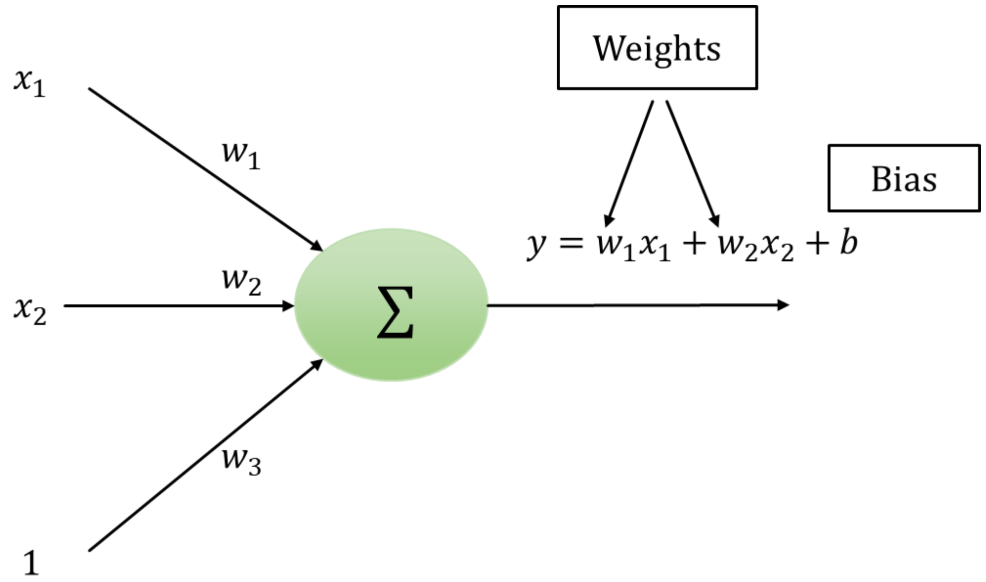
$$Y_0 = [y_o(\mathbf{1}) \quad y_o(\mathbf{2}) \quad \dots \quad y_o(\boldsymbol{n})]$$

When X is applied to the neural network, we get the output

$$Y = [y(\mathbf{1}) \quad y(\mathbf{2}) \quad \dots \quad y(\boldsymbol{n})]$$

$$Y = [(w_1x_1(\mathbf{1}) + w_2x_2(\mathbf{1}) + b) \quad (w_1x_1(\mathbf{2}) + w_2x_2(\mathbf{2}) + b) \quad \dots (w_1x_1(\boldsymbol{n}) + w_2x_2(\boldsymbol{n}) + b)]$$

How can the NN Learn? Analytical Approach



Consider n input samples

$$X = \begin{bmatrix} \boldsymbol{x}_1(\mathbf{1}) & \boldsymbol{x}_1(\mathbf{2}) & \dots & \boldsymbol{x}_1(\boldsymbol{n}) \\ \boldsymbol{x}_2(\mathbf{1}) & \boldsymbol{x}_2(\mathbf{2}) & \dots & \boldsymbol{x}_2(\boldsymbol{n}) \end{bmatrix}$$

The actual observation for X is

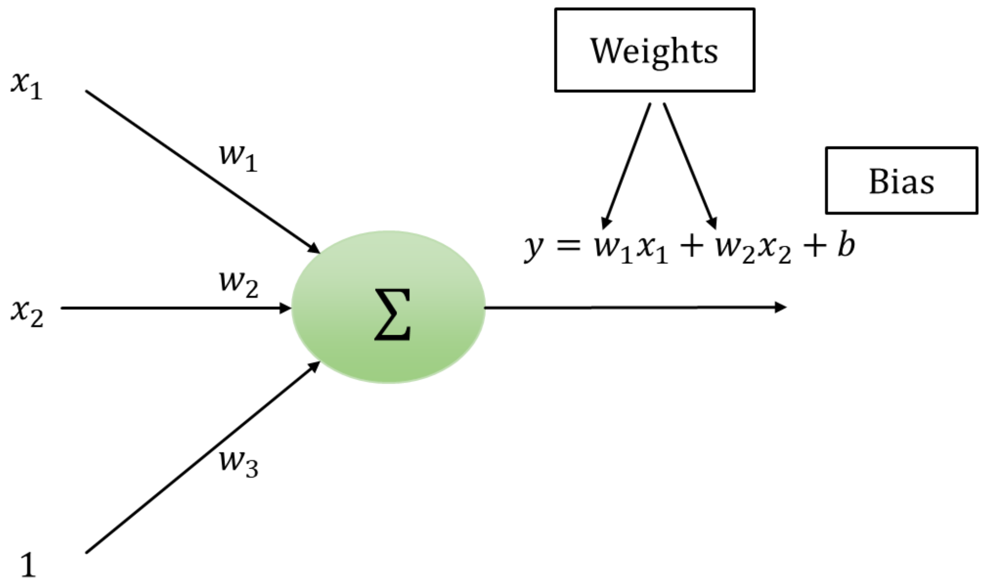
$$Y_0 = [y_o(\mathbf{1}) \quad y_o(\mathbf{2}) \quad \dots \quad y_o(\boldsymbol{n})]$$

When X is applied to the neural network, we get the output

$$Y = [y(\mathbf{1}) \quad y(\mathbf{2}) \quad \dots \quad y(\boldsymbol{n})]$$

$$Y = [(w_1 x_1(\mathbf{1}) + w_2 x_2(\mathbf{1}) + b) \quad (w_1 x_1(\mathbf{2}) + w_2 x_2(\mathbf{2}) + b) \quad \dots (w_1 x_1(\boldsymbol{n}) + w_2 x_2(\boldsymbol{n}) + b)]$$

How to find w_1, w_2, b analytically?



Consider n input samples

$$X = \begin{bmatrix} \boldsymbol{x}_1(\mathbf{1}) & \boldsymbol{x}_1(\mathbf{2}) & \dots & \boldsymbol{x}_1(\boldsymbol{n}) \\ \boldsymbol{x}_2(\mathbf{1}) & \boldsymbol{x}_2(\mathbf{2}) & \dots & \boldsymbol{x}_2(\boldsymbol{n}) \end{bmatrix}$$

The actual observation for X is

$$Y_0 = [y_o(\mathbf{1}) \quad y_o(\mathbf{2}) \quad \dots \quad y_o(\boldsymbol{n})]$$

When X is applied to the neural network, we get the output

$$Y = [y(\mathbf{1}) \quad y(\mathbf{2}) \quad \dots \quad y(\boldsymbol{n})]$$

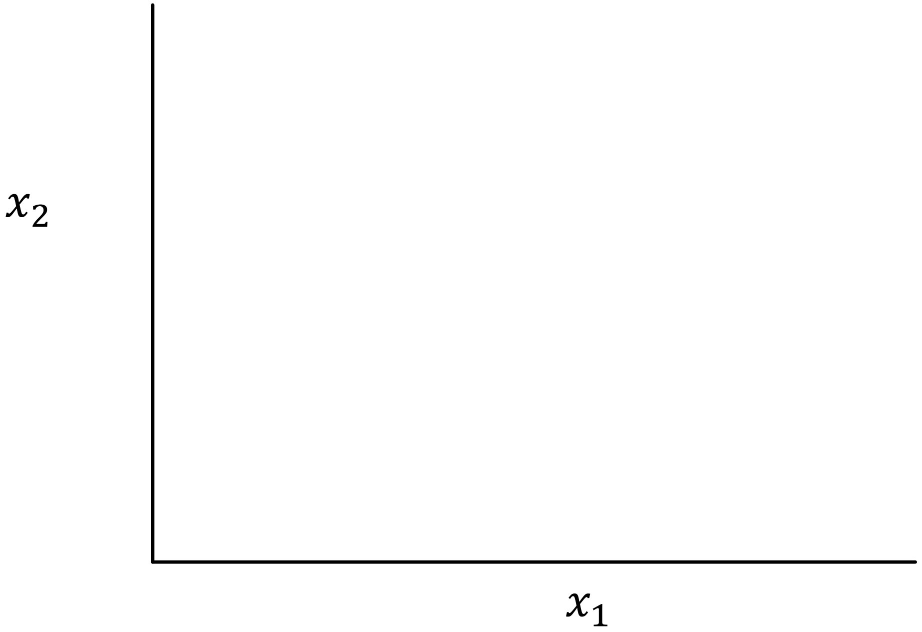
$$Y = [(w_1x_1(\mathbf{1}) + w_2x_2(\mathbf{1}) + b) \quad (w_1x_1(\mathbf{2}) + w_2x_2(\mathbf{2}) + b) \quad \dots (w_1x_1(\boldsymbol{n}) + w_2x_2(\boldsymbol{n}) + b)]$$

How to find w_1, w_2, b analytically? **Find $\boldsymbol{w_1, w_2, b}$ that minimizes $\|\boldsymbol{Y - Y_0}\|^2$**

How?

What Can this NN Do?

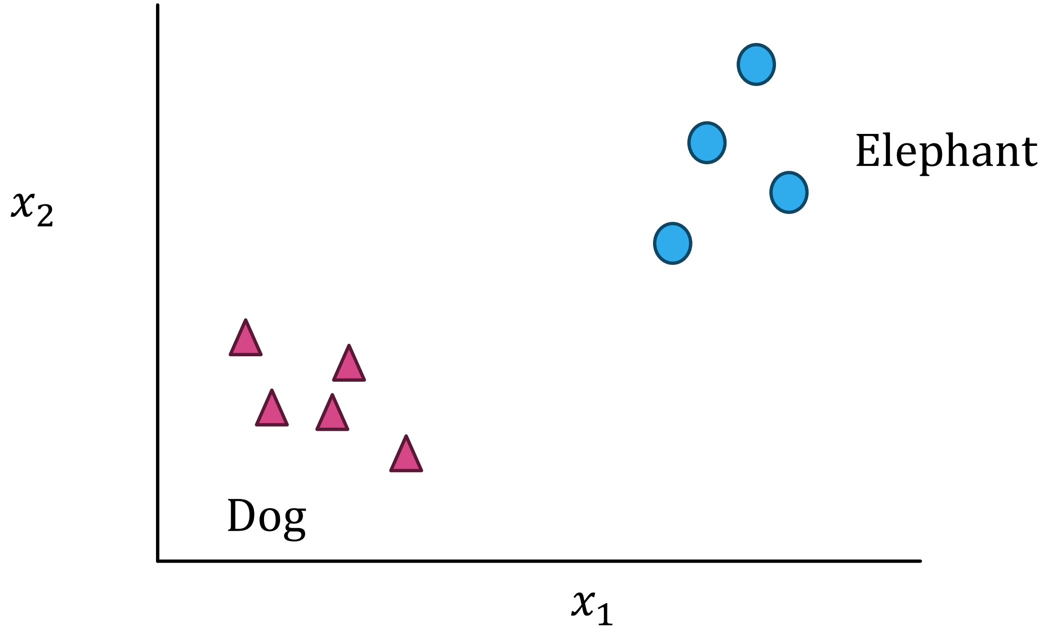
x_1: Height
x_2: Weight



Elephant vs dog classification

Plot of Data

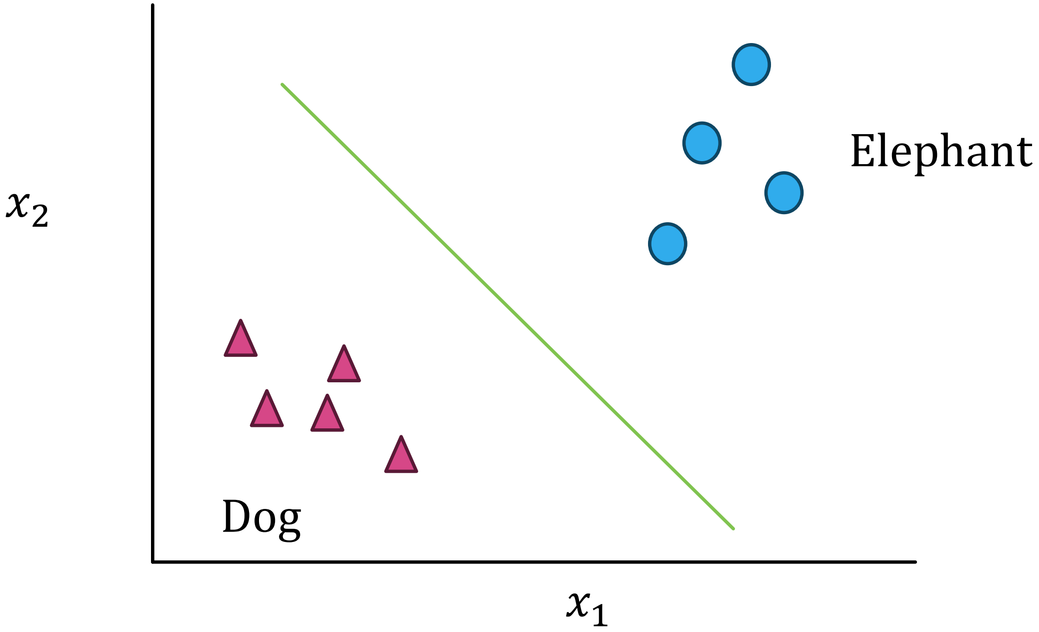
x_1: Height
x_2: Weight



Elephant vs dog classification

I plot some sample data

Separability



x_1: Height
x_2: Weight

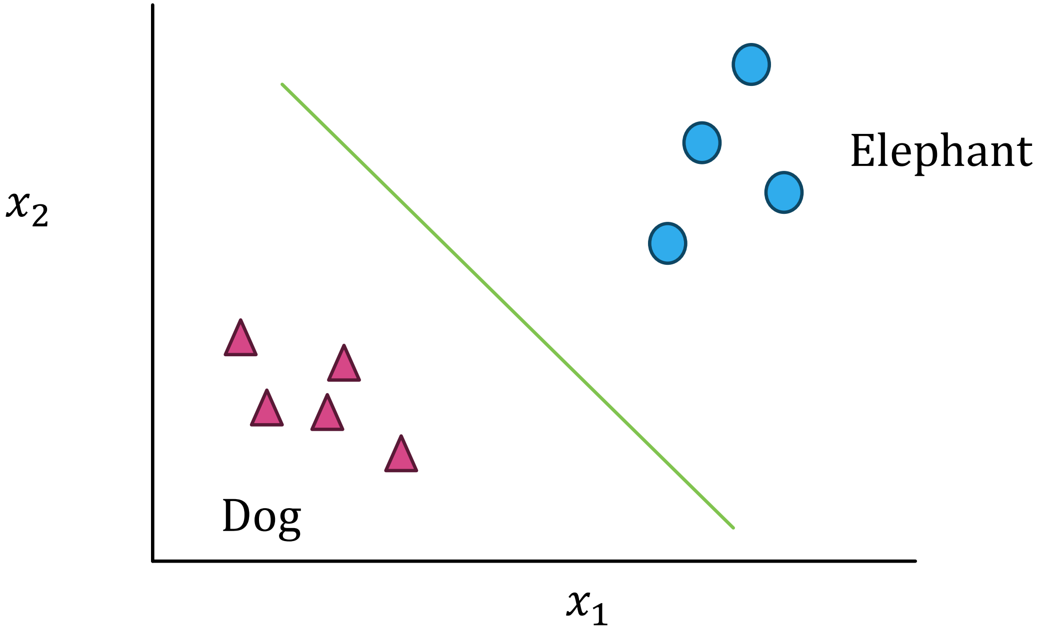
Elephant vs dog classification

I plot some sample data

Linearly separable

Class Encoding: One-hot Encoding

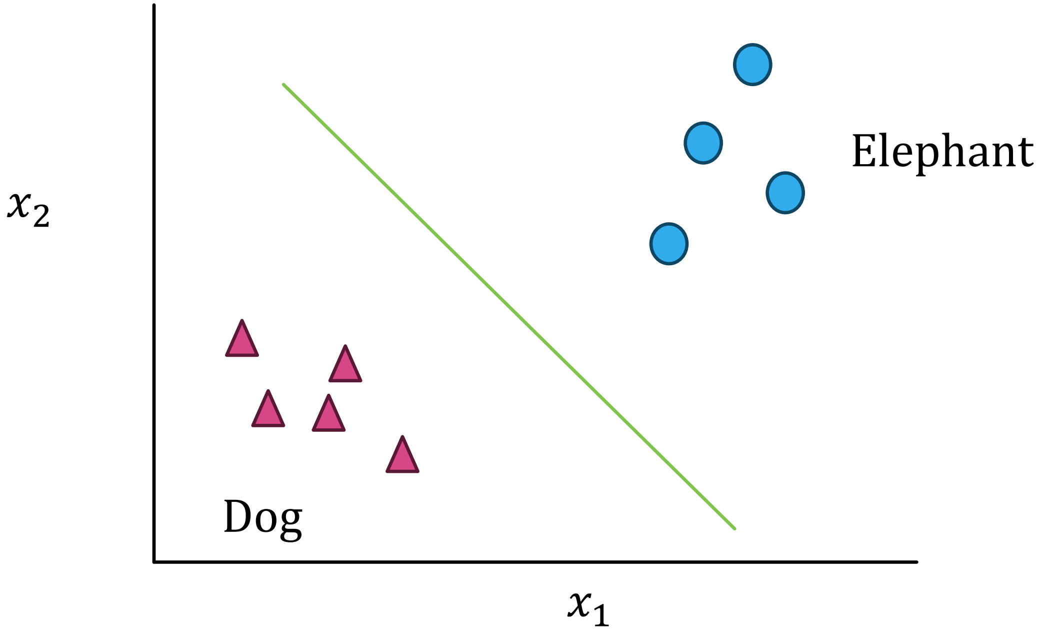
x_1: Height
x_2: Weight



Class Name	Class Number	Encoding
Elephant	1	1 0
Dog	2	0 1

Advantage of One-hot Encoding

x_1: Height
x_2: Weight

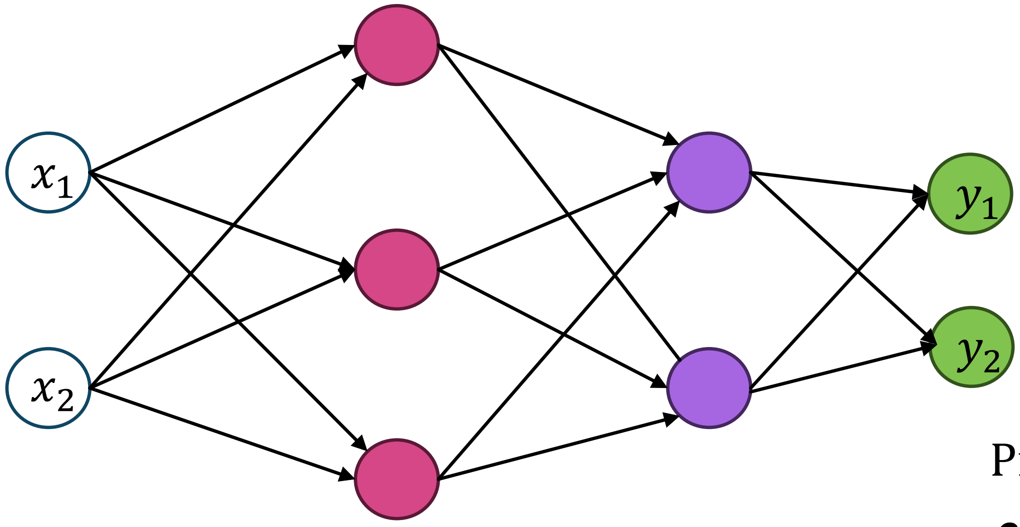


Class Name	Class Number	Encoding
Elephant	1	1 0
Dog	2	0 1

At each dimension of encoding, we perform a binary decision making

What Can this NN Do?

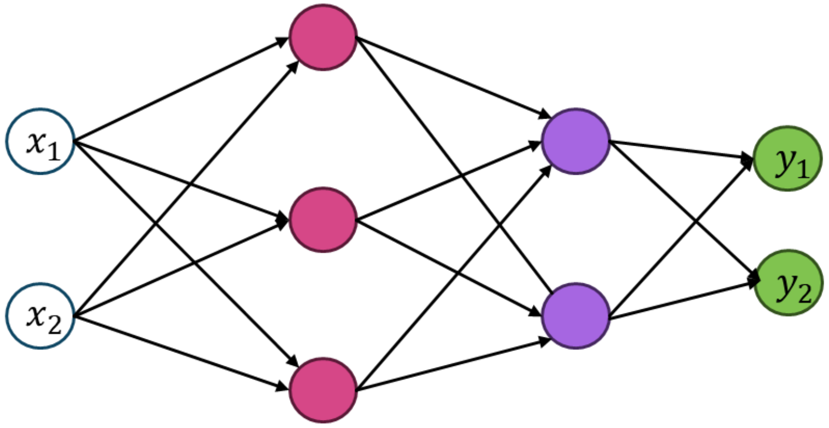
x_1: Height
x_2: Weight



Predicted output

$$\boldsymbol{y_1 = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3}$$
$$\boldsymbol{y_2 = \beta_1 x_1 + \beta_2 x_2 + \beta_3}$$

The Softmax operation



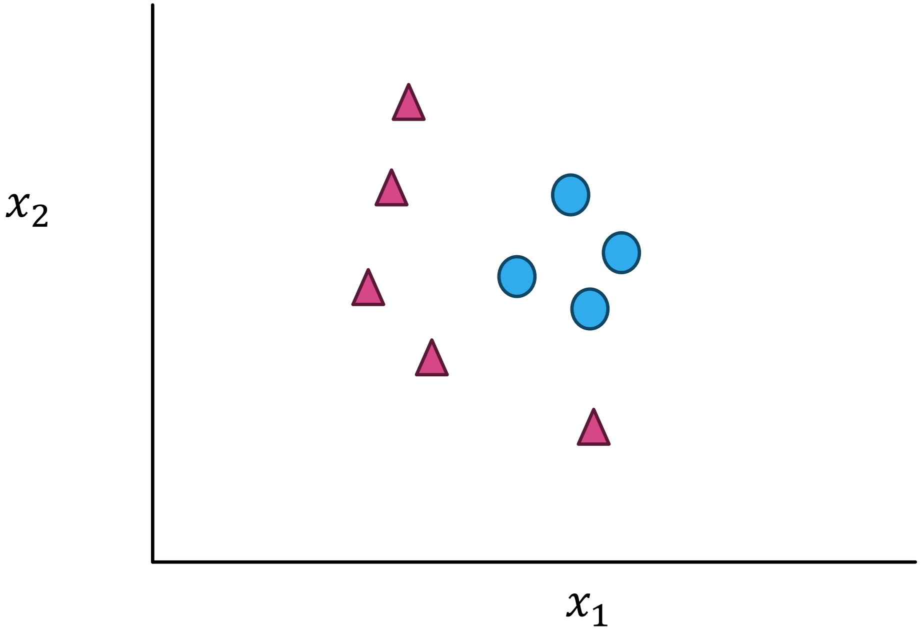
x_1: Height
x_2: Weight

Predicted output

$$\boldsymbol{y_1 = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3}$$
$$\boldsymbol{y_2 = \beta_1 x_1 + \beta_2 x_2 + \beta_3}$$

- There is no guarantee that y_i will be 0 or 1
- So, we do softmax
- $o_i = \dfrac{\exp(y_i)}{\sum_{\forall j} \exp(y_j)}$
- Output class label
  - $\underbrace{argmax}_{j}\ o_j$

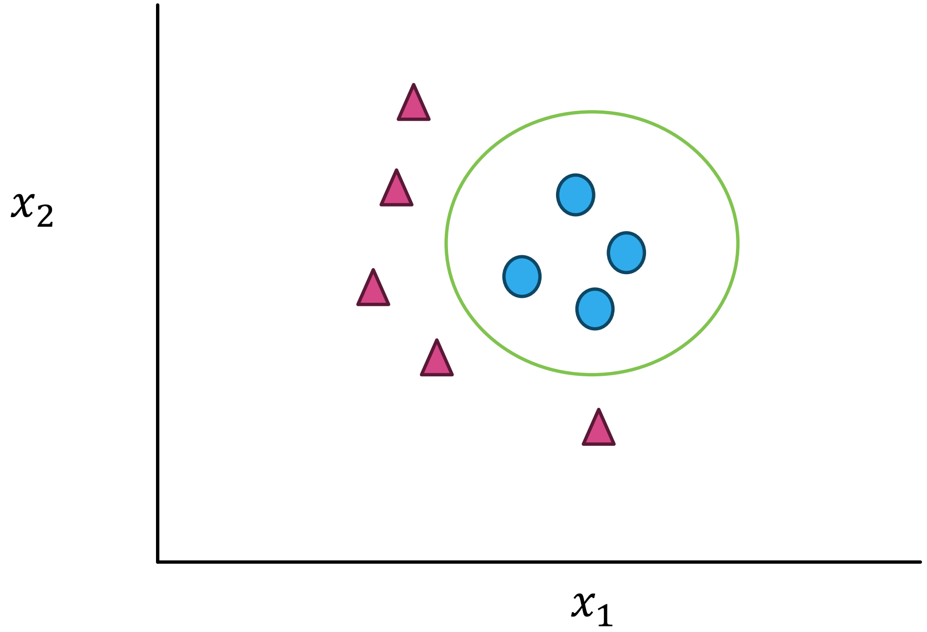
But, Life May not be So Simple



I plot some sample data

Linearly non-separable

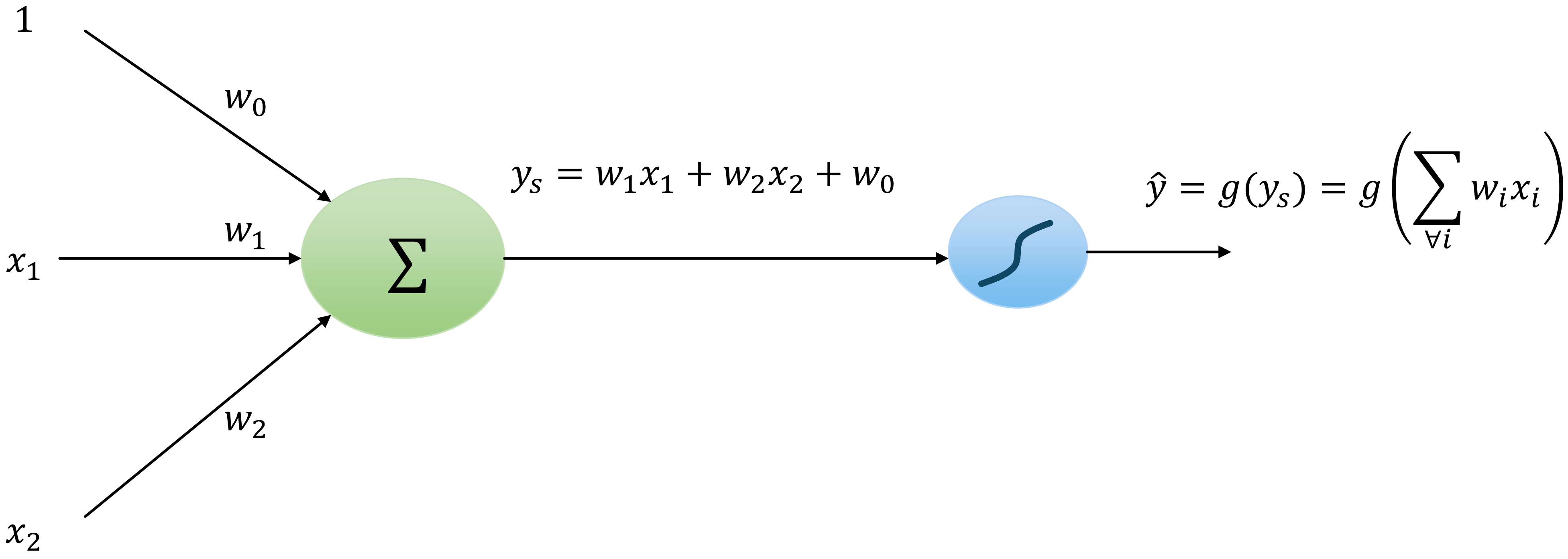
But, Life May not be So Simple



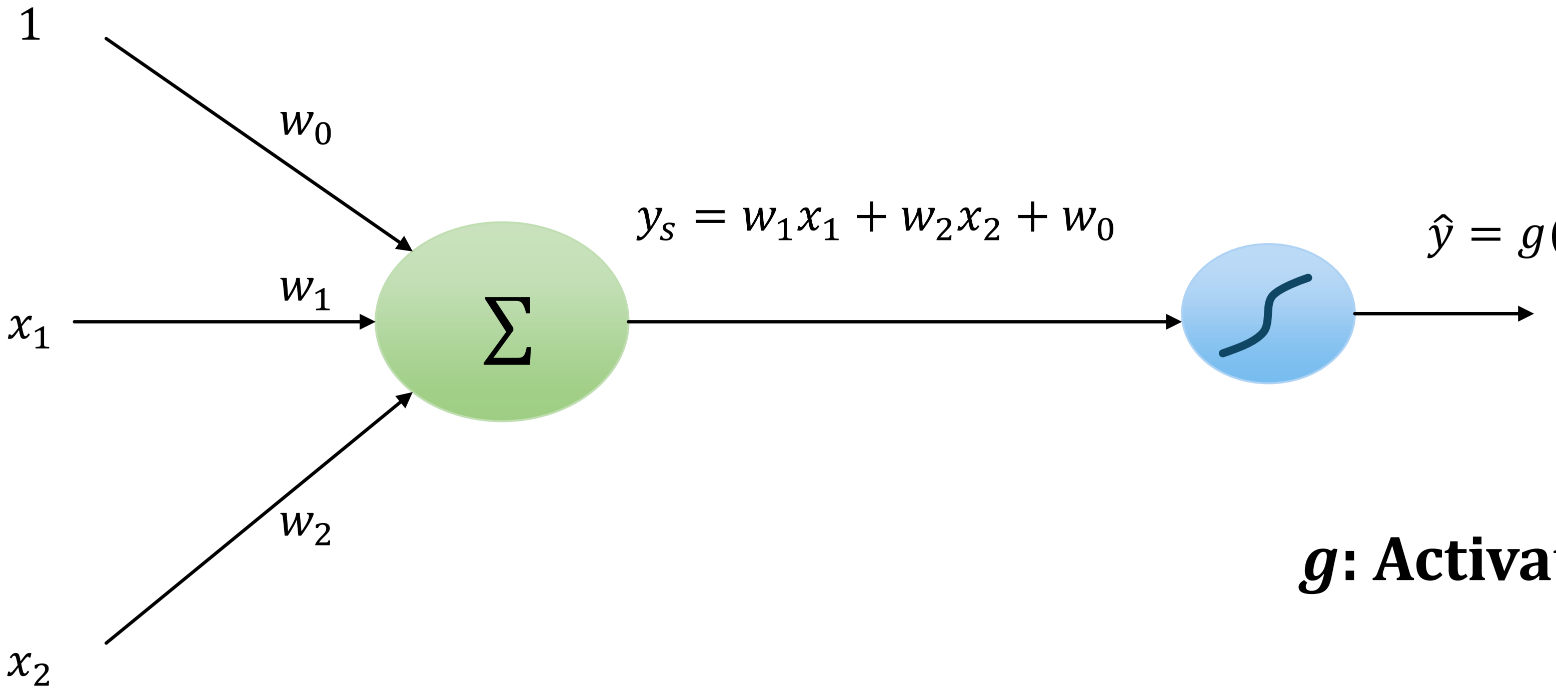
I plot some sample data

Linearly non-separable

Neuron: How to Bring in Non-linearity



Neuron: How to Bring in Non-linearity

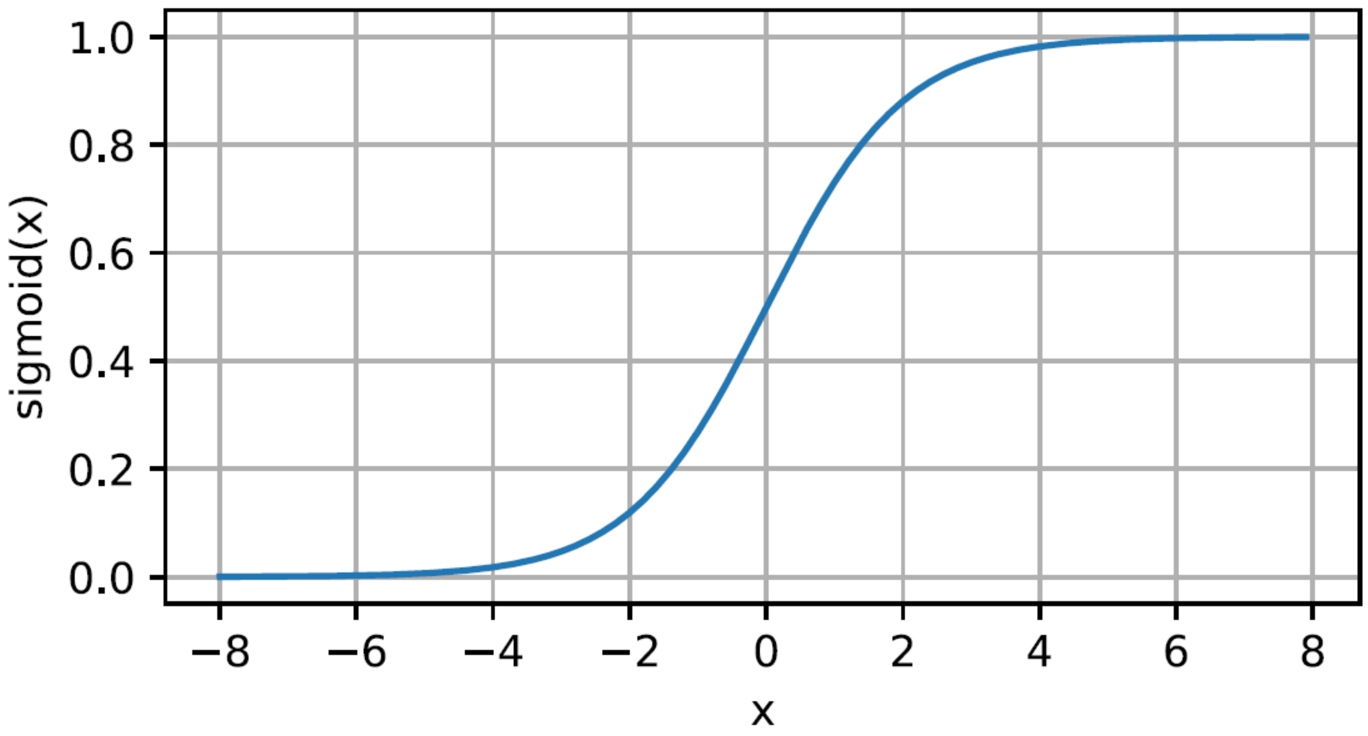


$$y_s = w_1 x_1 + w_2 x_2 + w_0$$

$$\hat{y} = g(y_s) = g\left(\sum_{\forall i} w_i x_i\right)$$

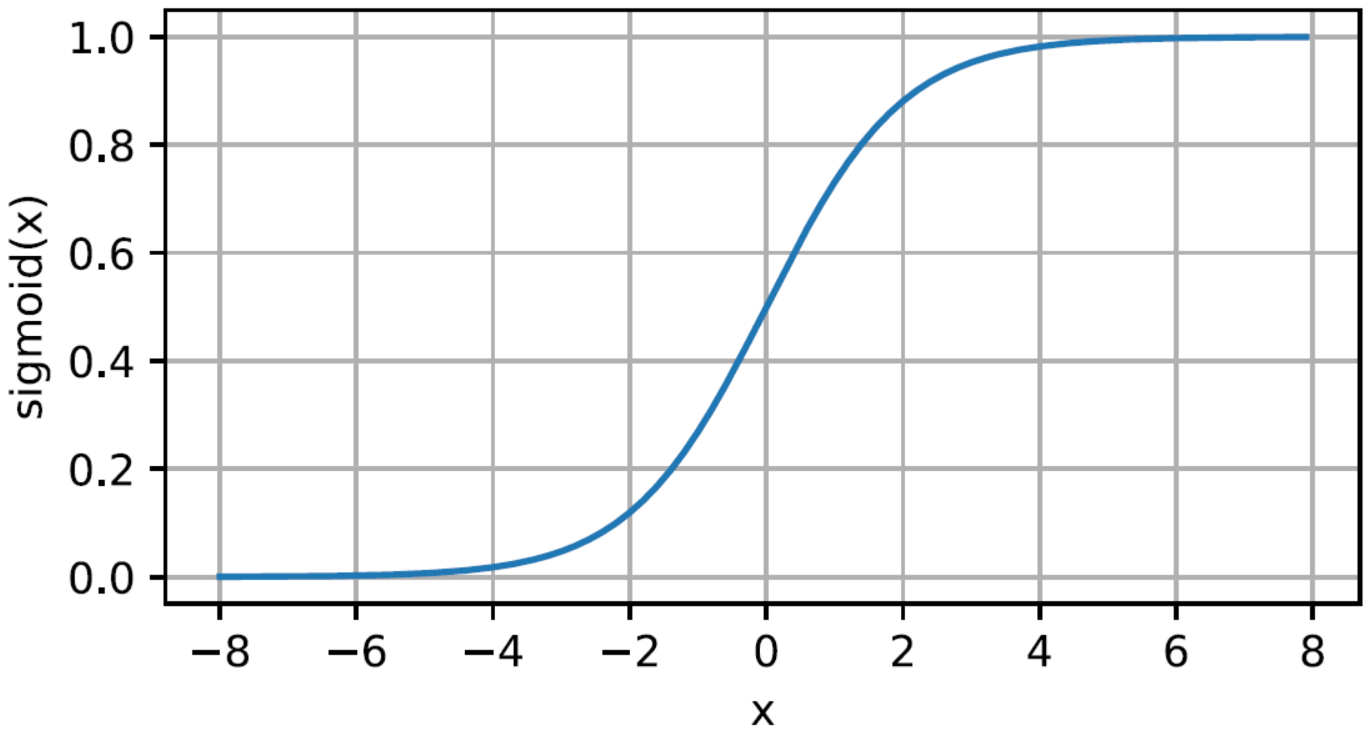
$\boldsymbol{g}$**: Activation Function**

Different Activation Functions: Sigmoid



$$sigmoid(x) = \frac{1}{1 + \exp(-x)}$$

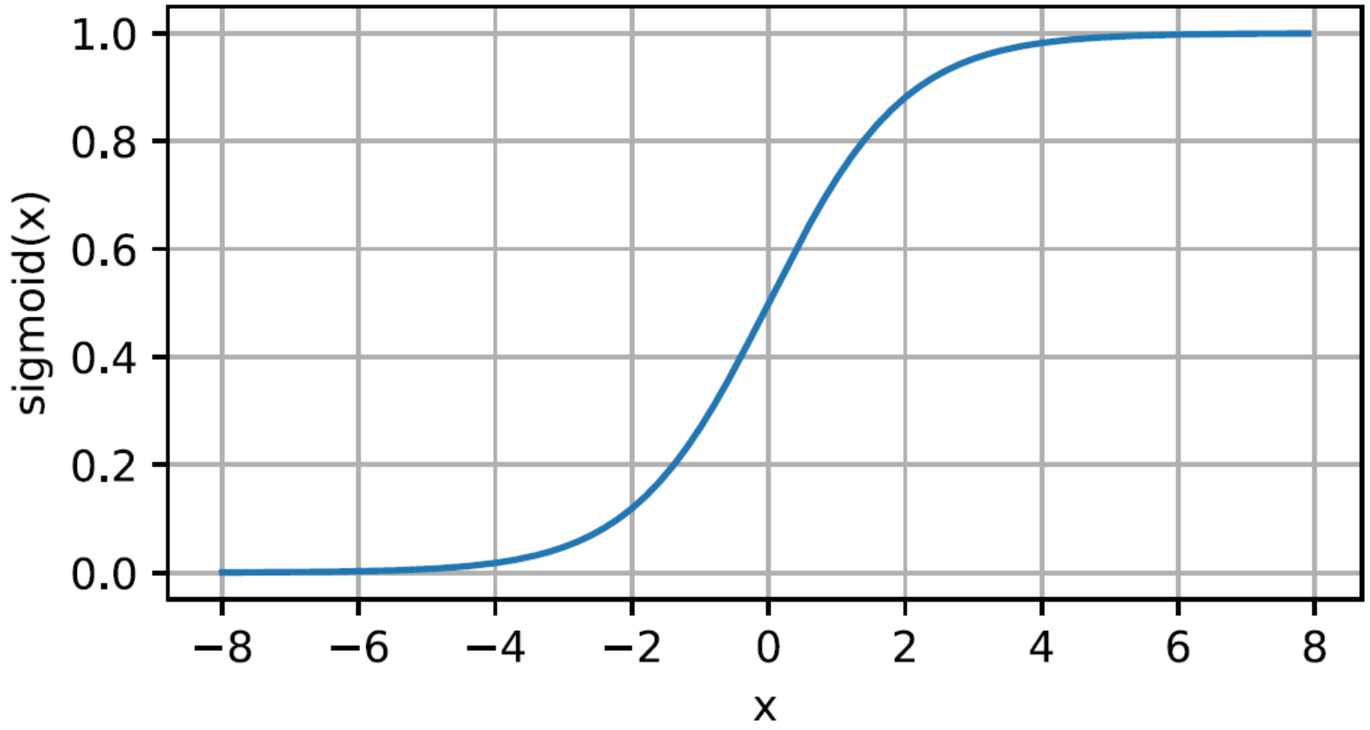
Different Activation Functions: Sigmoid



$$sigmoid(x) = \frac{1}{1 + \exp(-x)}$$

$$\frac{d}{dx} sigmoid(x) = f(sigmoid(x))$$

Different Activation Functions: Sigmoid

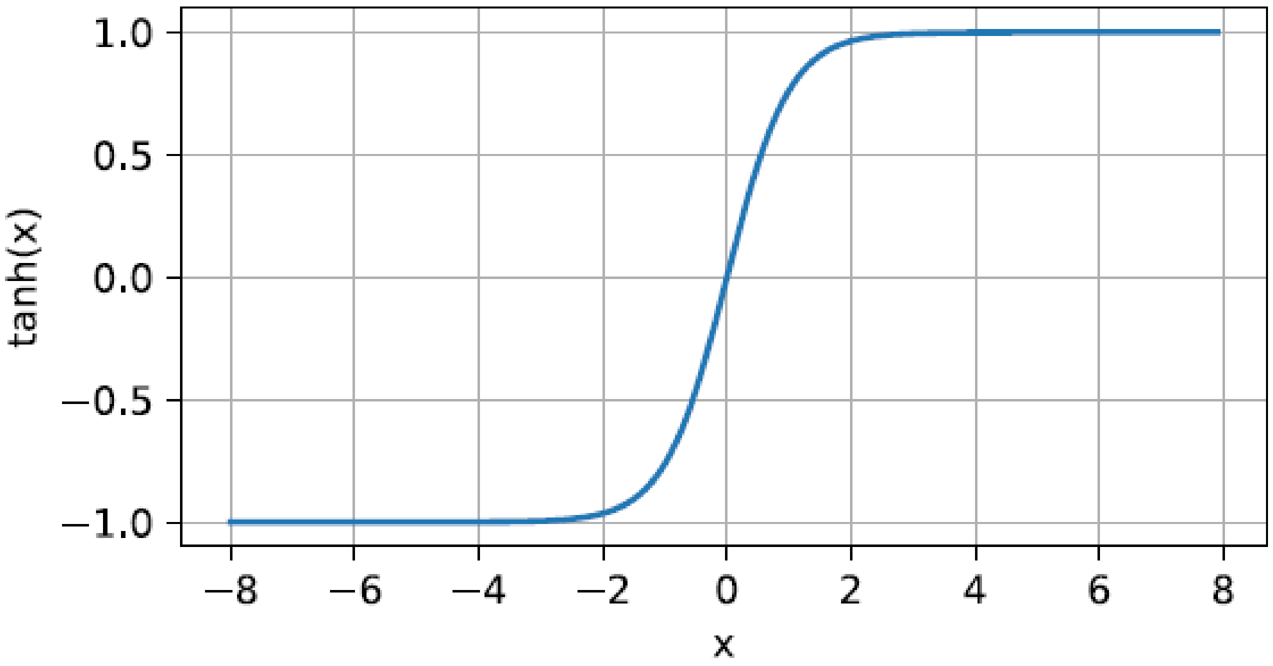


$$sigmoid(x) = \frac{1}{1 + \exp(-x)}$$

$$\frac{d}{dx} sigmoid(x) = f(sigmoid(x))$$

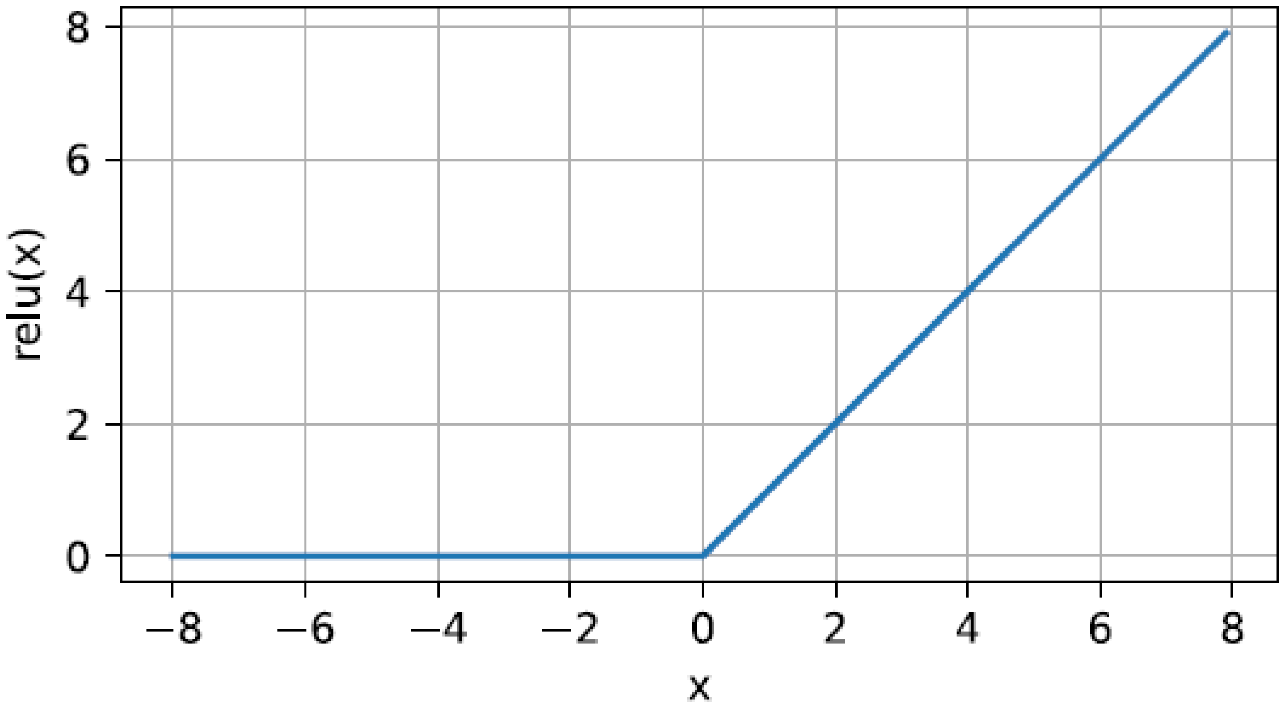
Find the value of $\boldsymbol{f(sigmoid(x))}$

Different Activation Functions: tanh



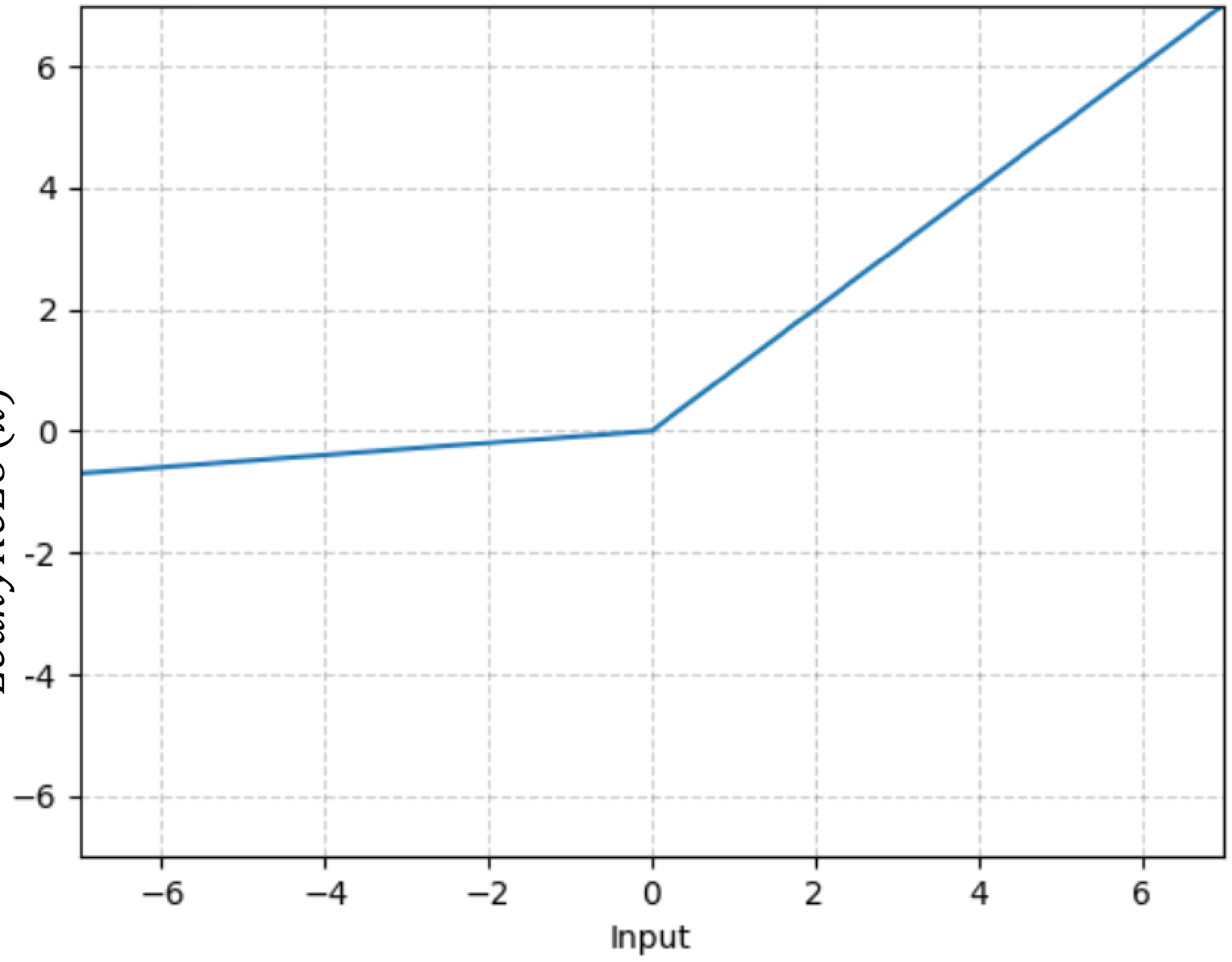
$$tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$

Different Activation Functions: ReLU



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

Different Activation Functions: Leaky ReLU

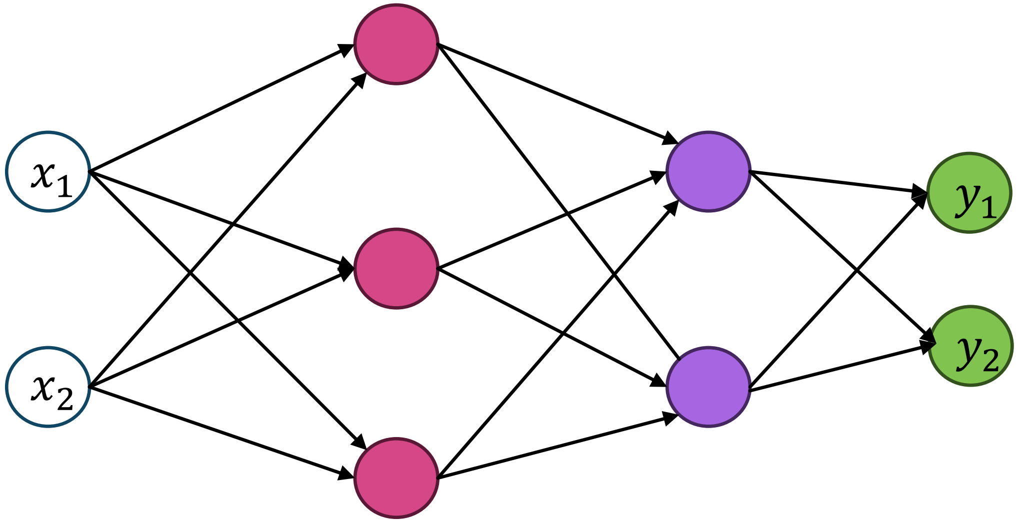


$LeakyReLU(x)$

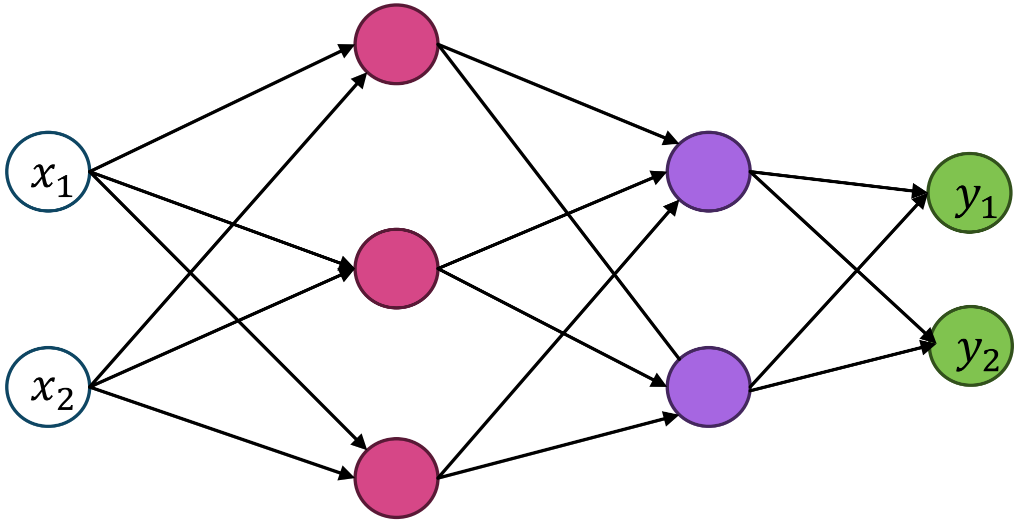
x

$$LeakyReLU(x) = \max(mx, x)$$

How to Measure the Performance of this NN?

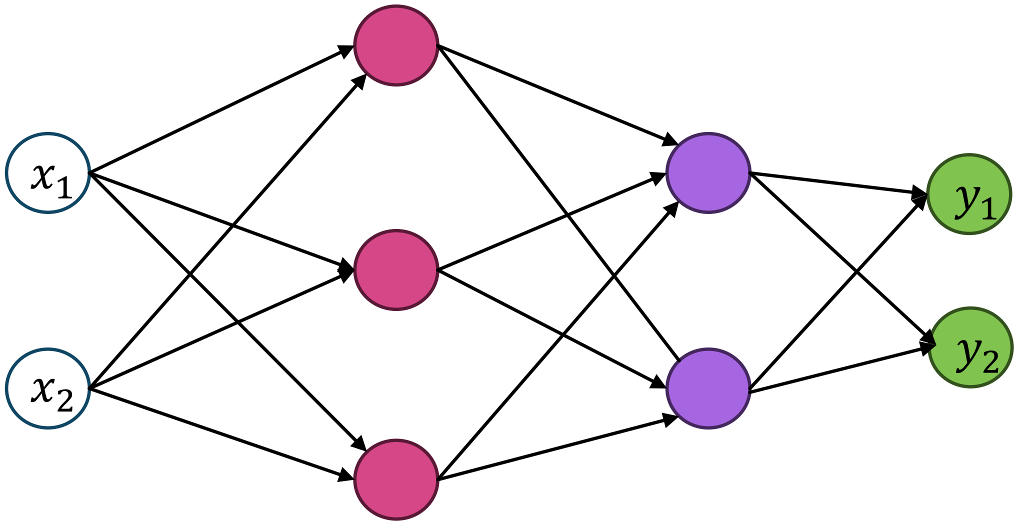


How to Measure the Performance of this NN?



Difference between the output and reality (observation)

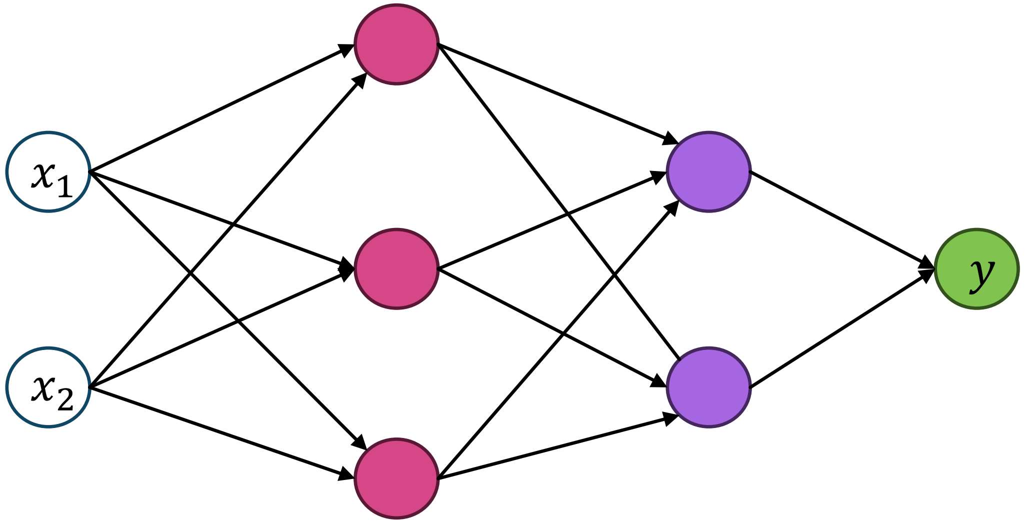
How to Measure the Performance of this NN?



Difference between the output and reality (observation)

Loss

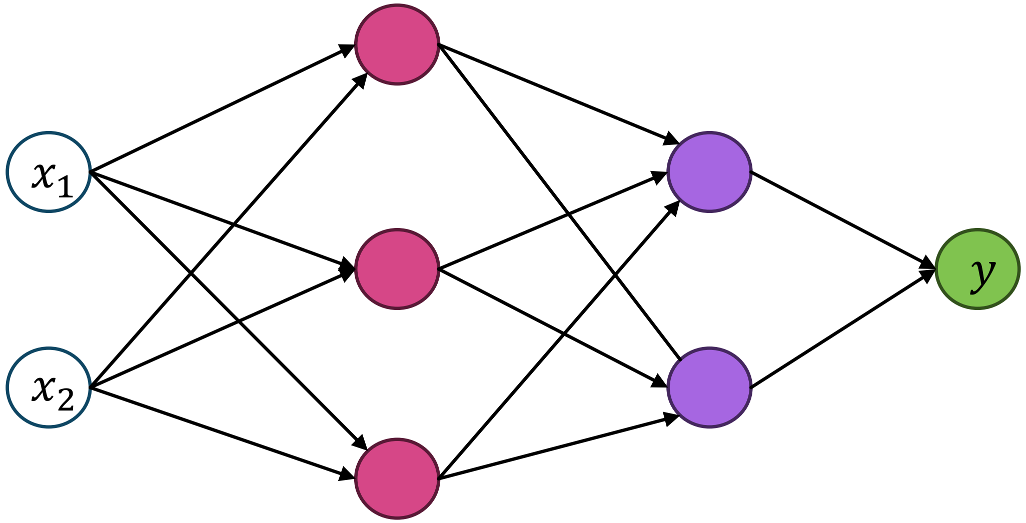
Losses for Regression: Mean Squared Error

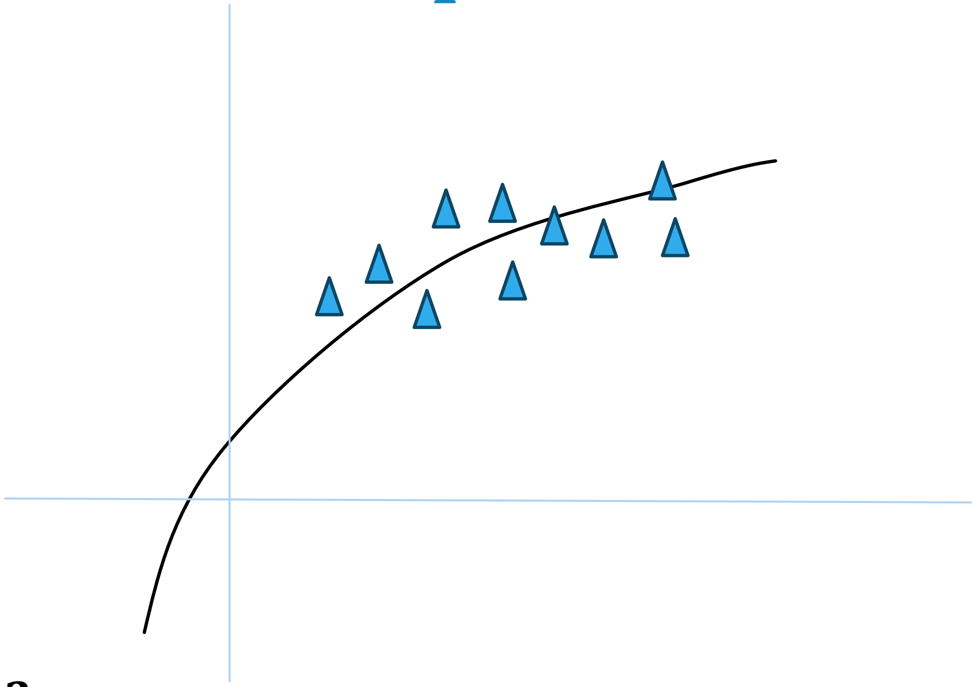


- For input $x(i)$, Output $= y(i)$
- i^{th} observation $= y_o(i)$
- No. of observation: N

$$\boldsymbol{MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_o(i) - y(i)\right)^2}$$

Losses for Regression: Mean Squared Error

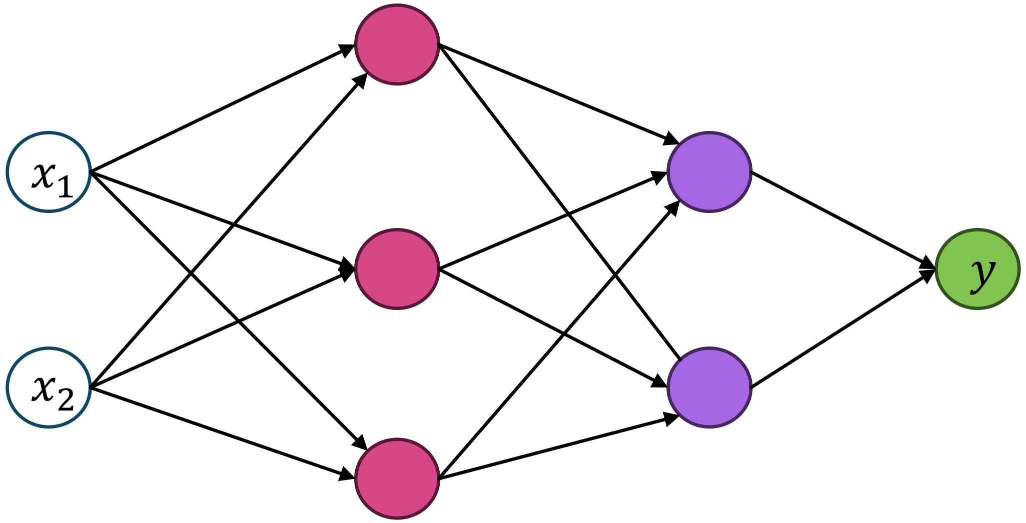


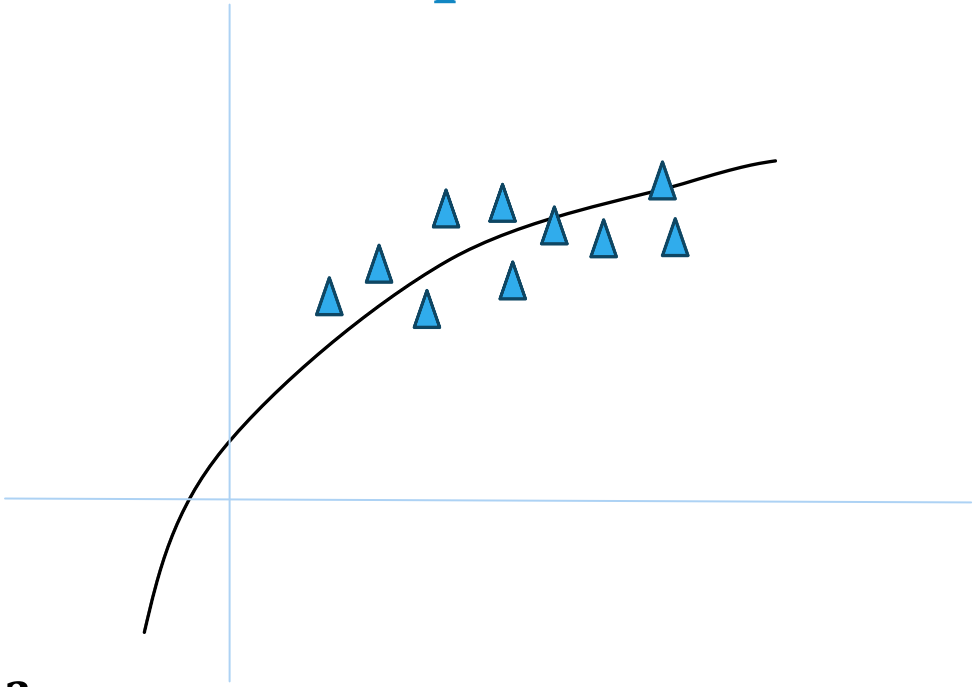


$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_o(i) - y(i)\right)^2$$

- Low MSE

Losses for Regression: Mean Squared Error





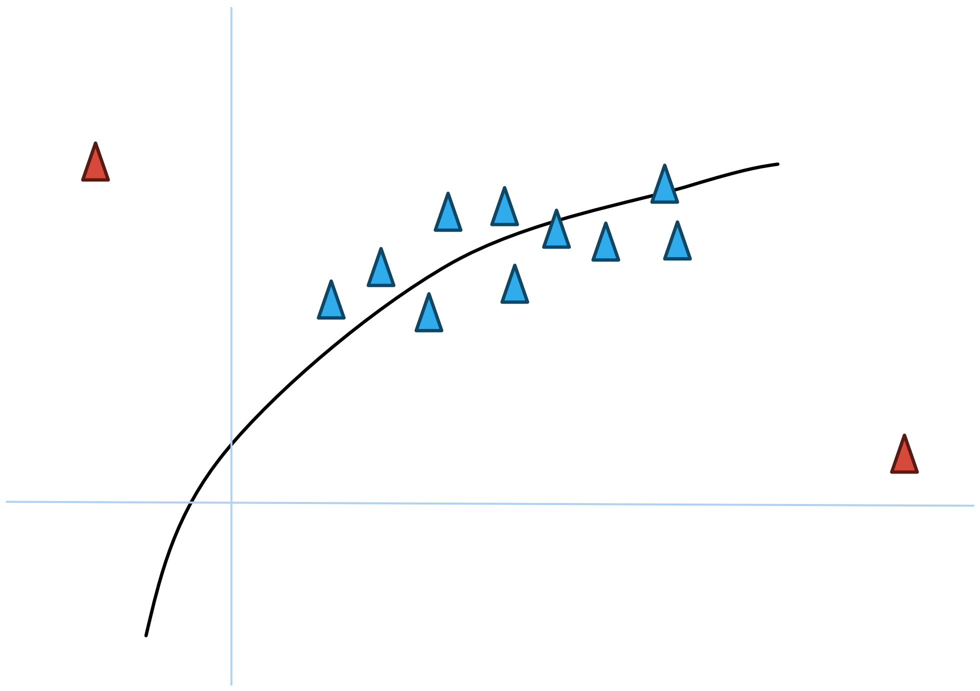
$$\boldsymbol{MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_o(i) - y(i)\right)^2}$$

- Low MSE

Drawback?

Losses for Regression: Mean Squared Error

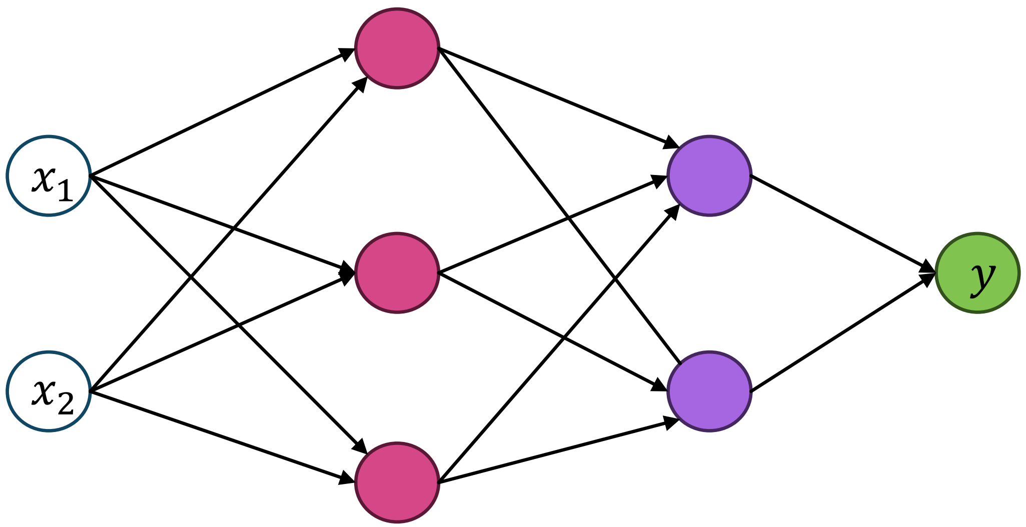
$$\boldsymbol{MSE} = \frac{\mathbf{1}}{\boldsymbol{N}} \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{N}} \left(\boldsymbol{y_o}(\boldsymbol{i}) - \boldsymbol{y}(\boldsymbol{i})\right)^{\mathbf{2}}$$



Drawback?

- High MSE, even for a few major outliers
- Very sensitive to outliers

Losses for Regression: Mean Absolute Error



- For input $x(i)$, Output $= y(i)$
- i^{th} observation $= y_o(i)$
- No. of observation: N

$$\boldsymbol{MAE = \frac{1}{N}\sum_{i=1}^{N}|y_o(i) - y(i)|}$$

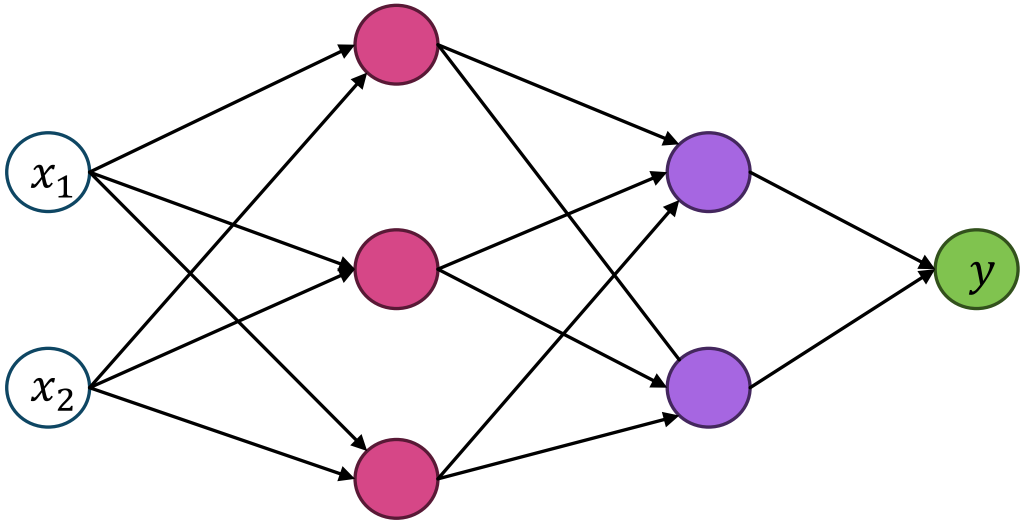
Losses for Regression: Mean Absolute Error

$$\boldsymbol{MAE = \frac{1}{N}\sum_{i=1}^{N}|y_o(i) - y(i)|}$$

What is the drawback of MAE?

- For input $x(i)$, Output $= y(i)$
- i^{th} observation $= y_o(i)$
- No. of observation: N

Losses for Regression: Huber Loss

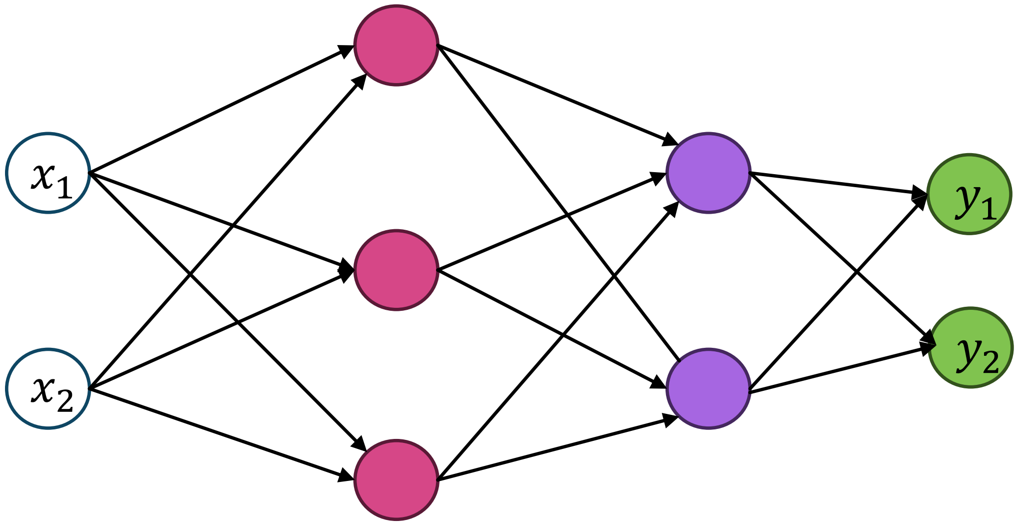


- For input $x(i)$, Output $= y(i)$
- i^{th} observation $= y_o(i)$
- No. of observation: N

Find out what is this and how it helps

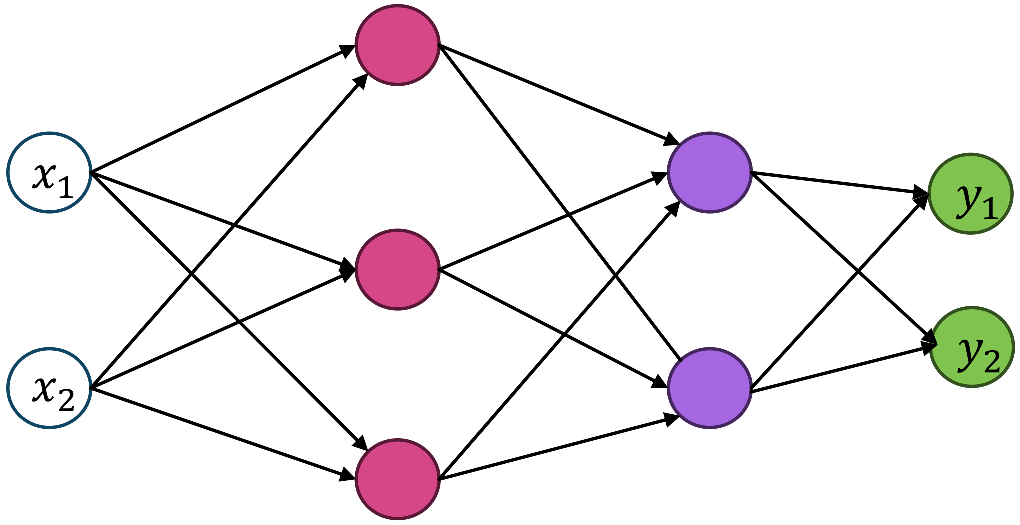
Losses for Classification: Cross Entropy Loss

Observed label $= y_o = [y_{o1} \;\; y_{o2}]$
Predicted probability $= p = [p_1 \;\; p_2]$



$$L = -\sum_{i=1}^{C} y_{oi} \log p_i$$

Losses for Classification: Cross Entropy Loss

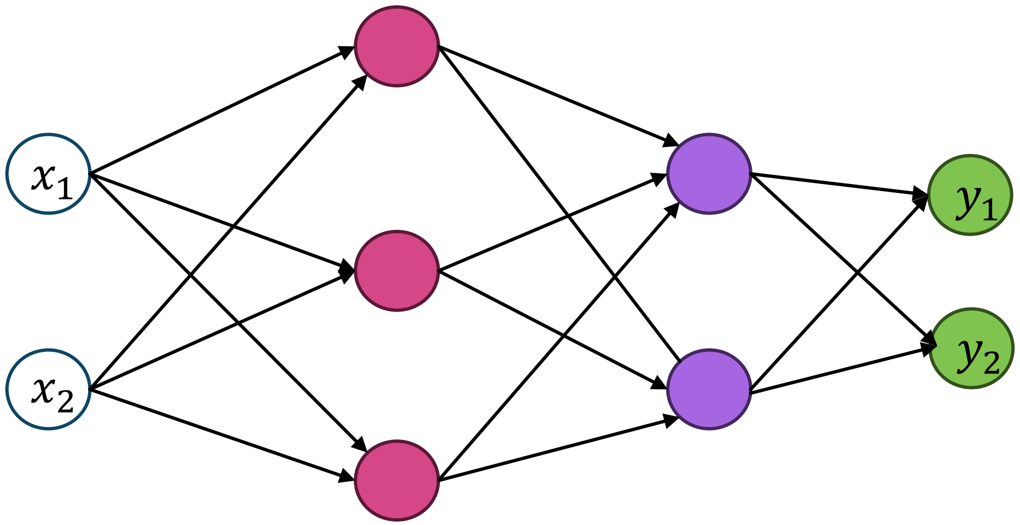


Observed label $= y_o = [y_{o1} \;\; y_{o2}]$
Predicted probability $= p = [p_1 \;\; p_2]$

$$L = -\sum_{i=1}^{C} y_{oi} \log p_i = f(W)$$

Loss is a function of weights and biases (parameters)

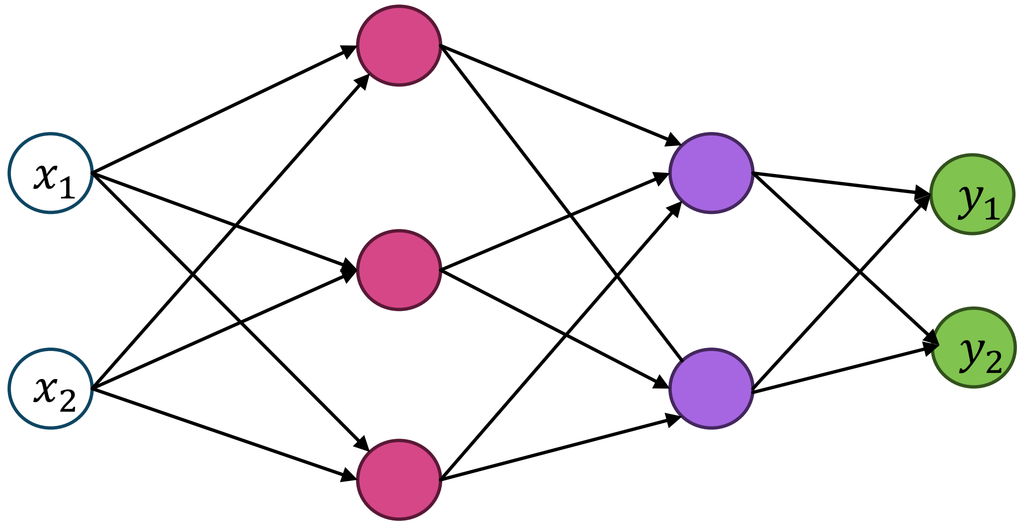
Training of NNs



We can't have an analytical solution for NNs with nonlinearity

The Goal of Training an NN is to get a set of weights and biases (parameters) that would minimize the loss

Training of NNs

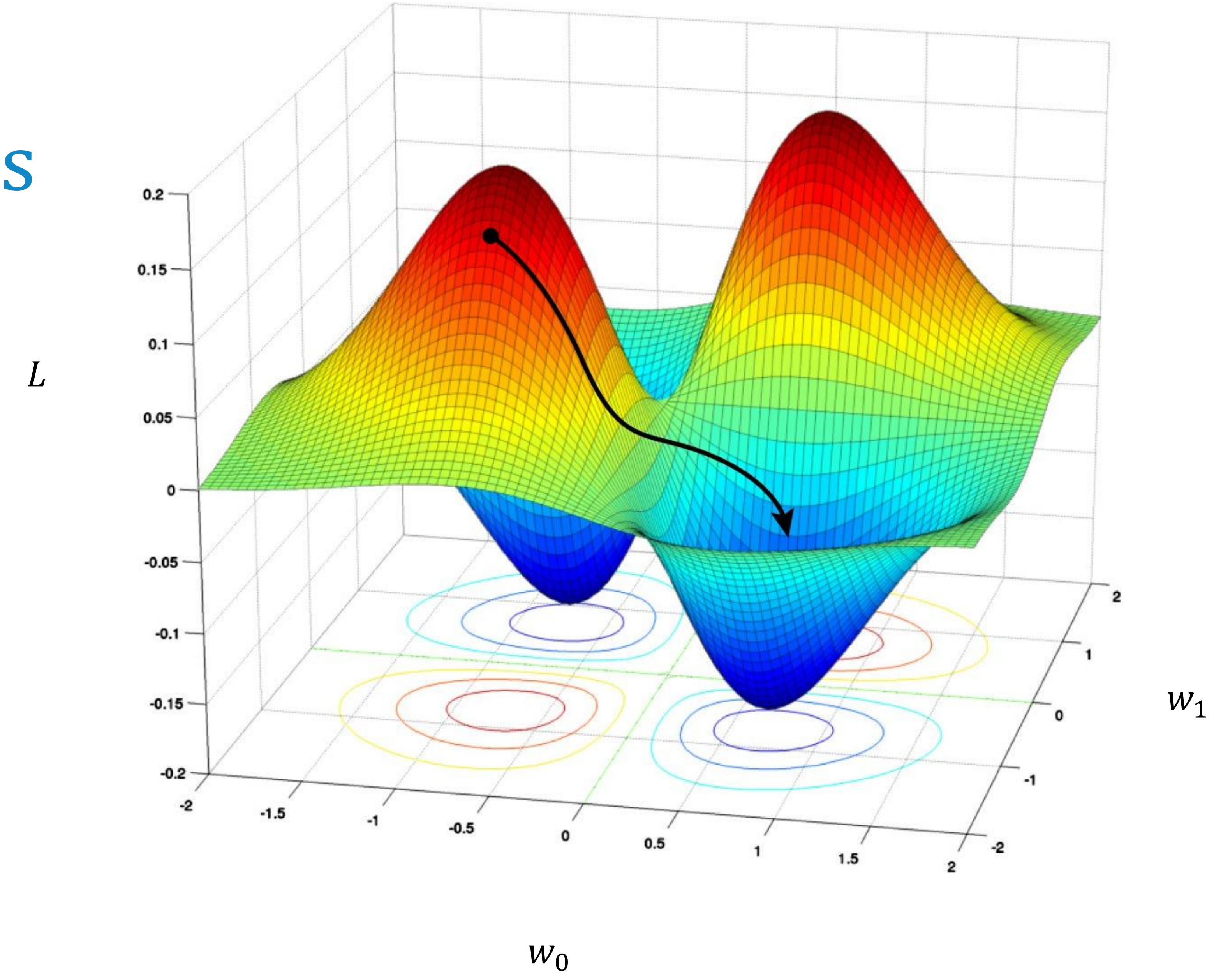


We can't have an analytical solution for NNs with nonlinearity

The Goal of Training an NN is to get a set of weights and biases (parameters) that would minimize the loss

$$\boldsymbol{W^* = \underbrace{argmin}_{W}\ L(W)}$$

How to Minimize Loss

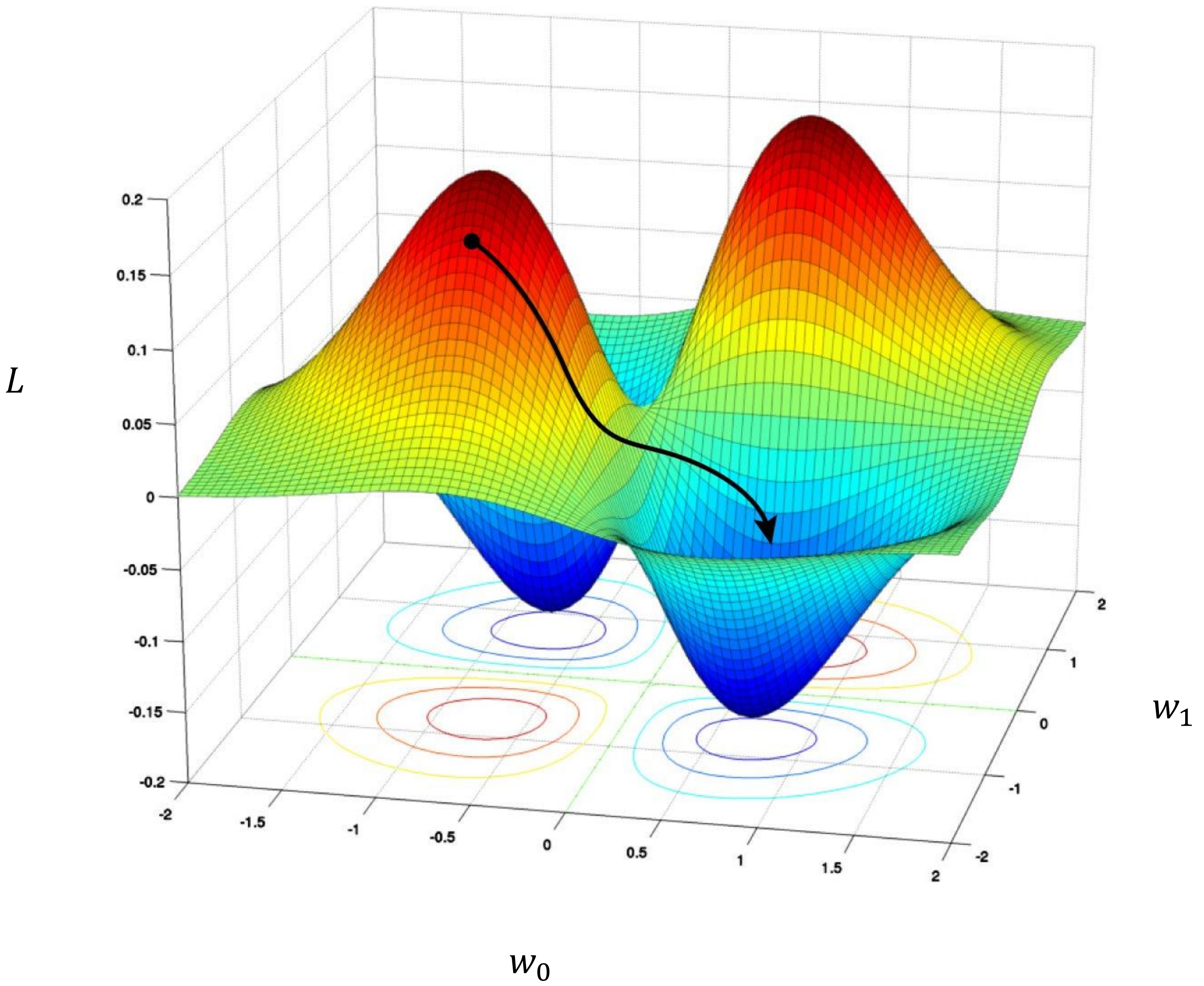


How to Minimize Loss

The Goal of Training an NN is to get a set of weights and biases (parameters) that would minimize the loss

$$W^* = \underbrace{argmin}_{W} \; L(W)$$

Our goal is to reach the global minima

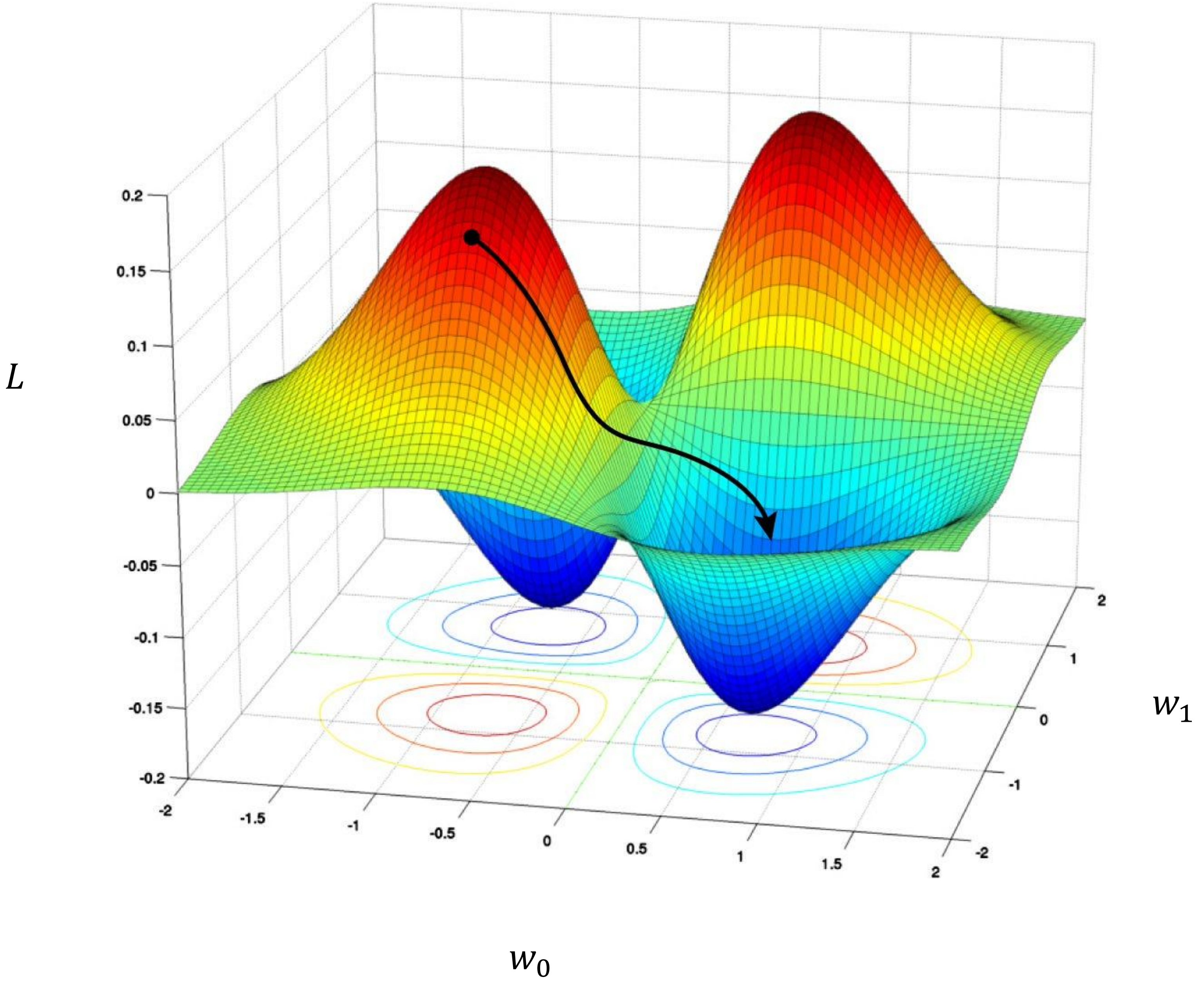


How to Reach the Global Minima?

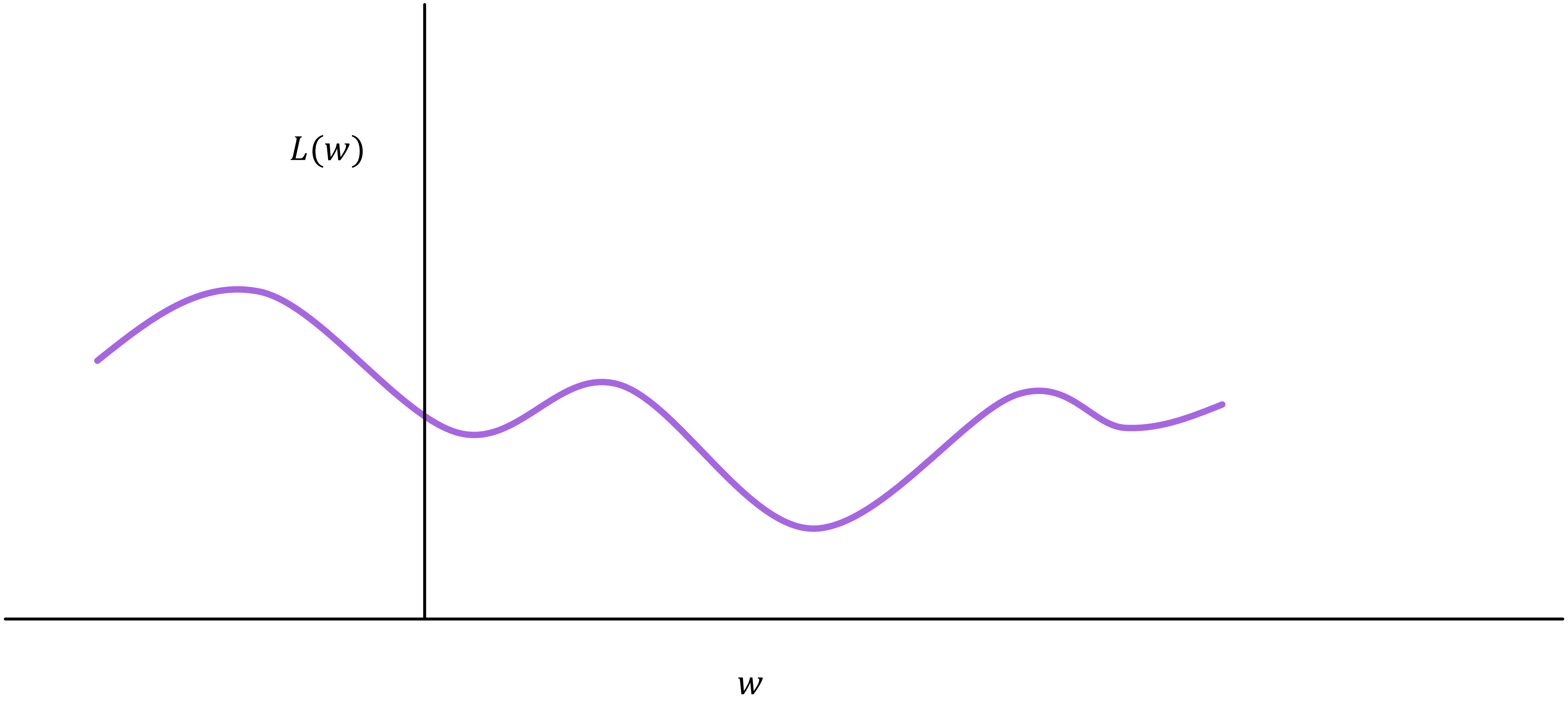
The Goal of Training an NN is to get a set of weights and biases (parameters) that would minimize the loss

$$W^* = \underbrace{argmin}_{W} \; L(W)$$

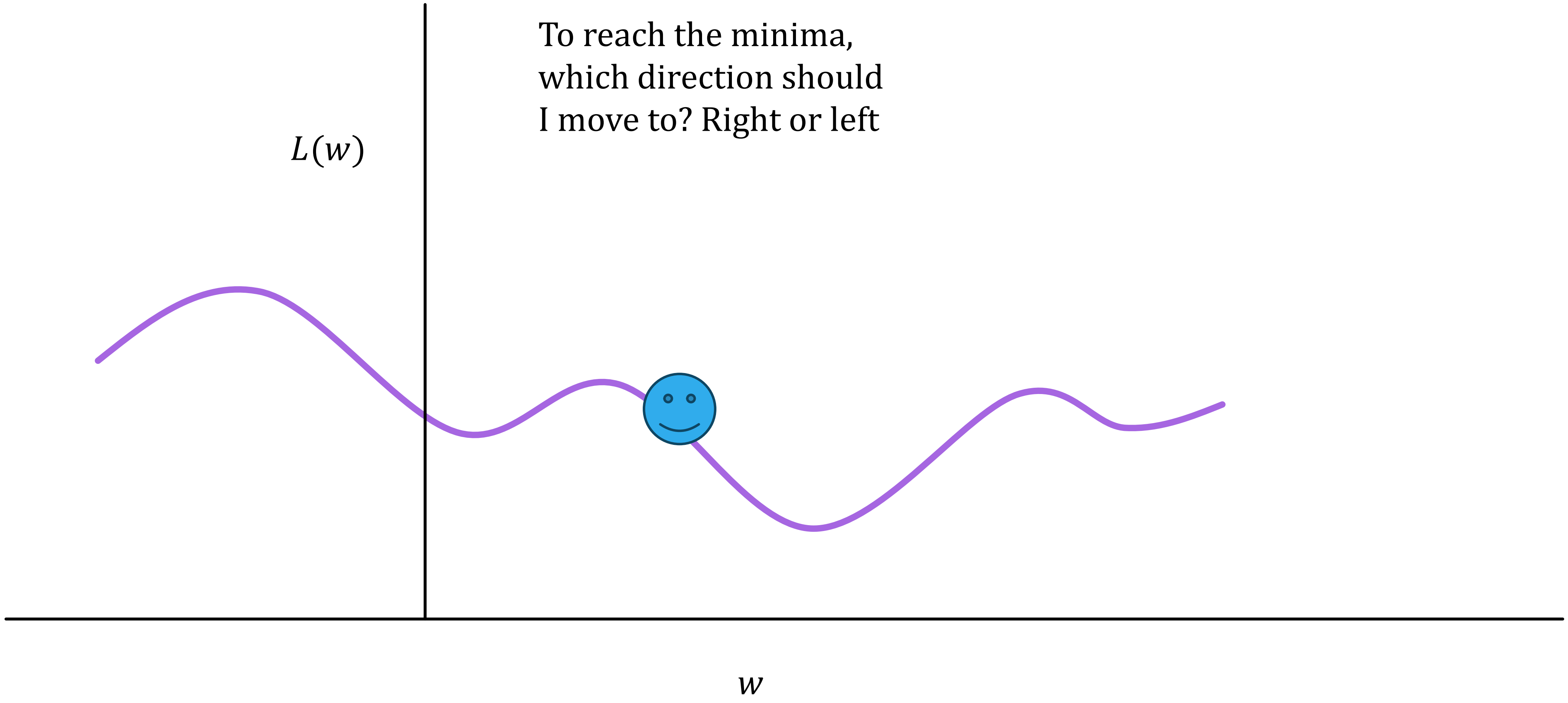
Our goal is to reach the global minima



How to Reach the Global Minima: A 2D Example



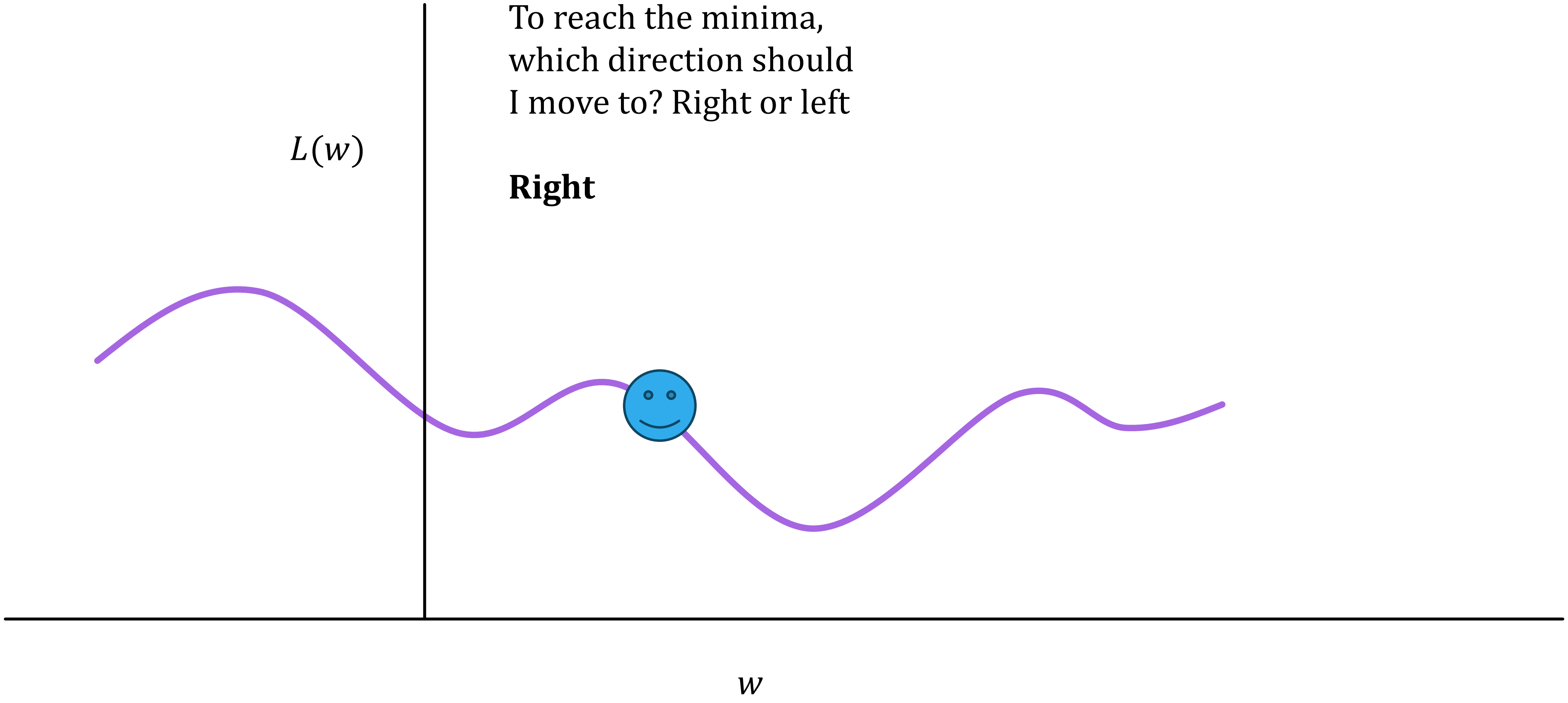
How to Reach the Global Minima: A 2D Example



How to Reach the Global Minima: A 2D Example

To reach the minima,
which direction should
I move to? Right or left

Right

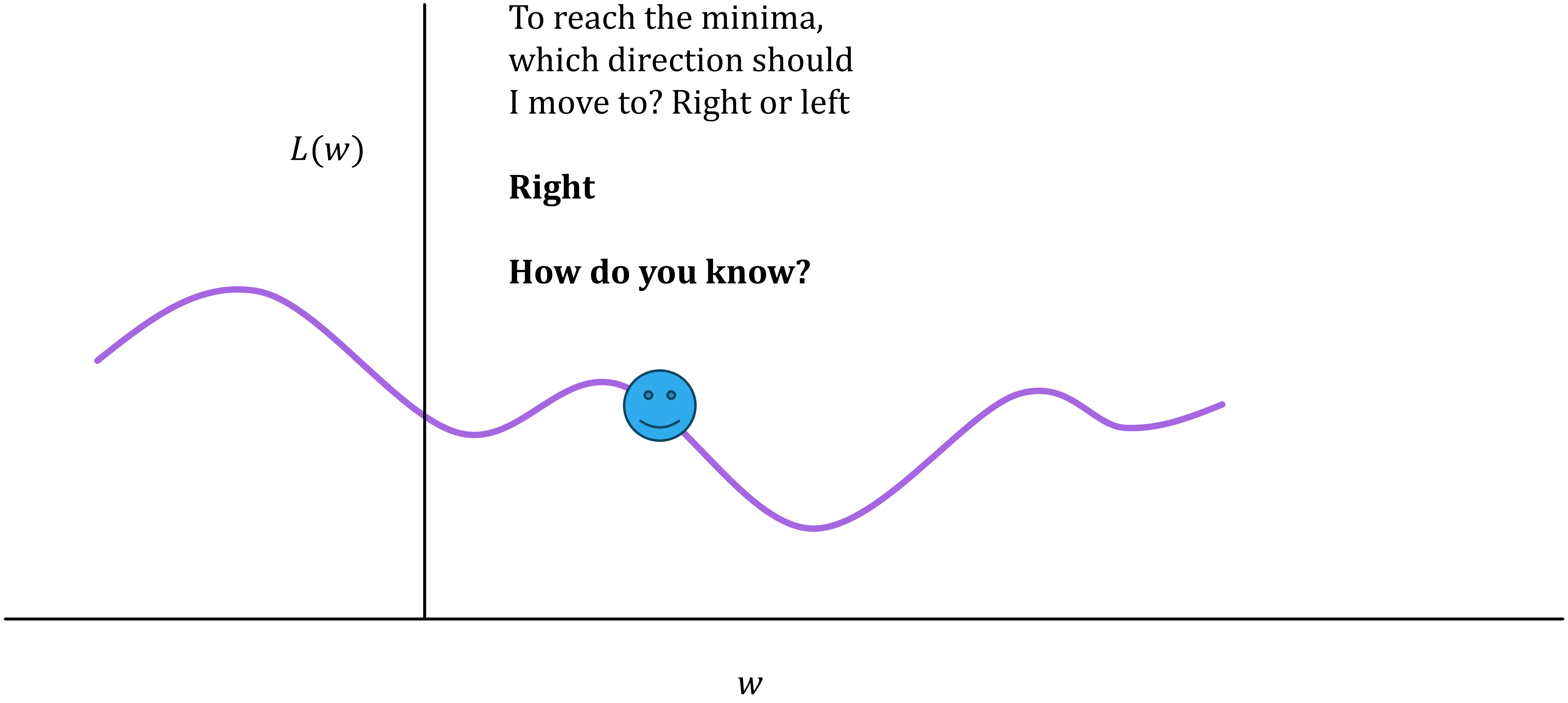


How to Reach the Global Minima: A 2D Example

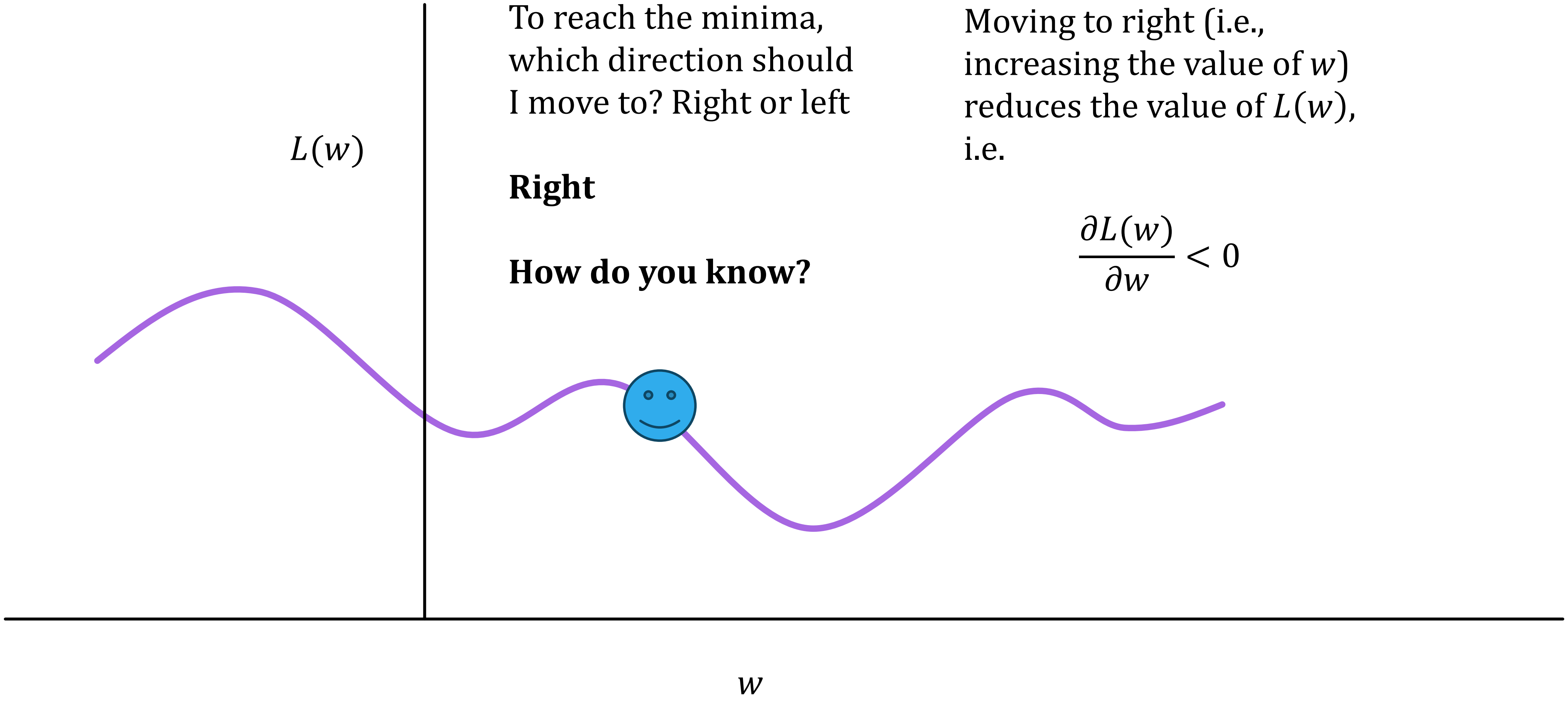
To reach the minima, which direction should I move to? Right or left

Right

How do you know?



How to Reach the Global Minima: A 2D Example



To reach the minima, which direction should I move to? Right or left

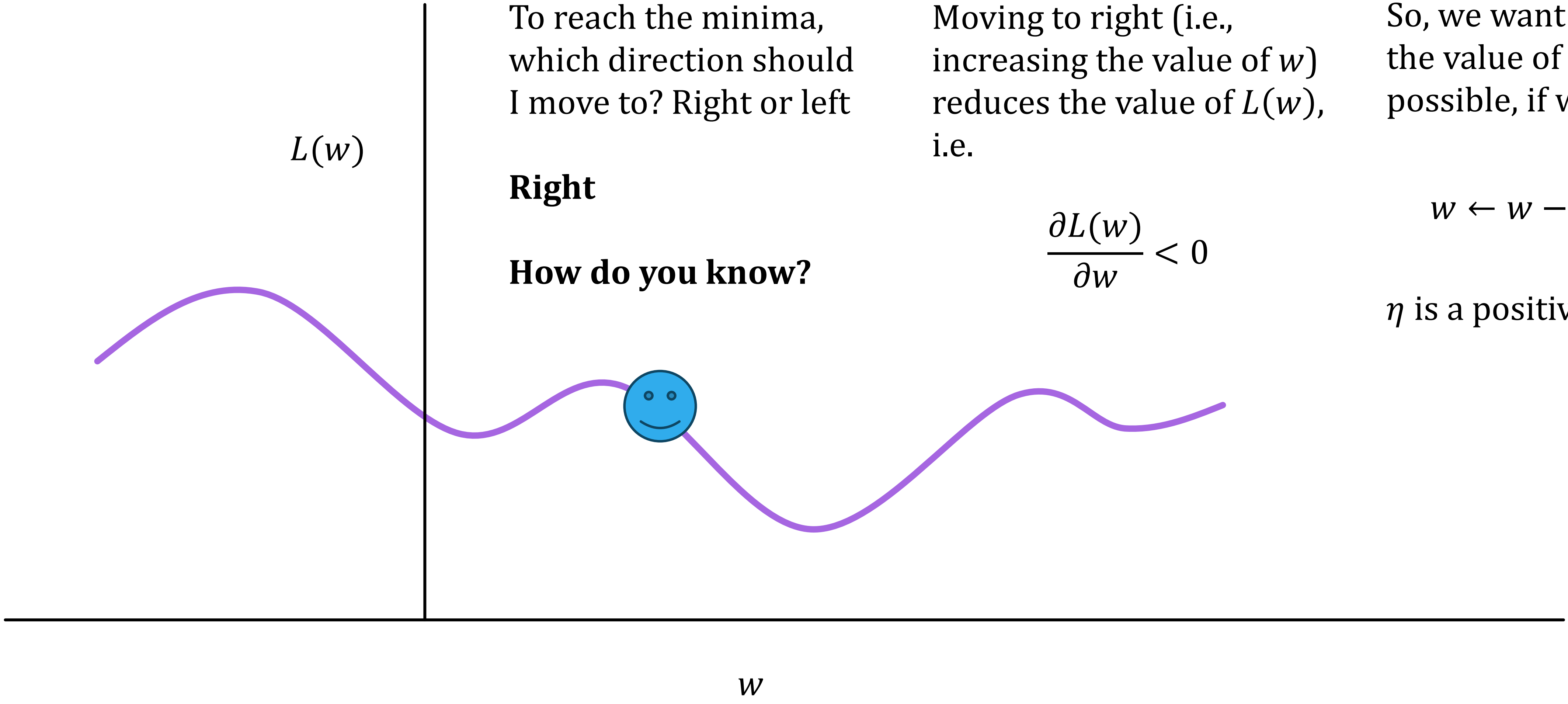
Right

How do you know?

Moving to right (i.e., increasing the value of w) reduces the value of $L(w)$, i.e.

$$\frac{\partial L(w)}{\partial w} < 0$$

How to Reach the Global Minima: A 2D Example



To reach the minima, which direction should I move to? Right or left

Right

How do you know?

Moving to right (i.e., increasing the value of w) reduces the value of $L(w)$, i.e.

$$\frac{\partial L(w)}{\partial w} < 0$$

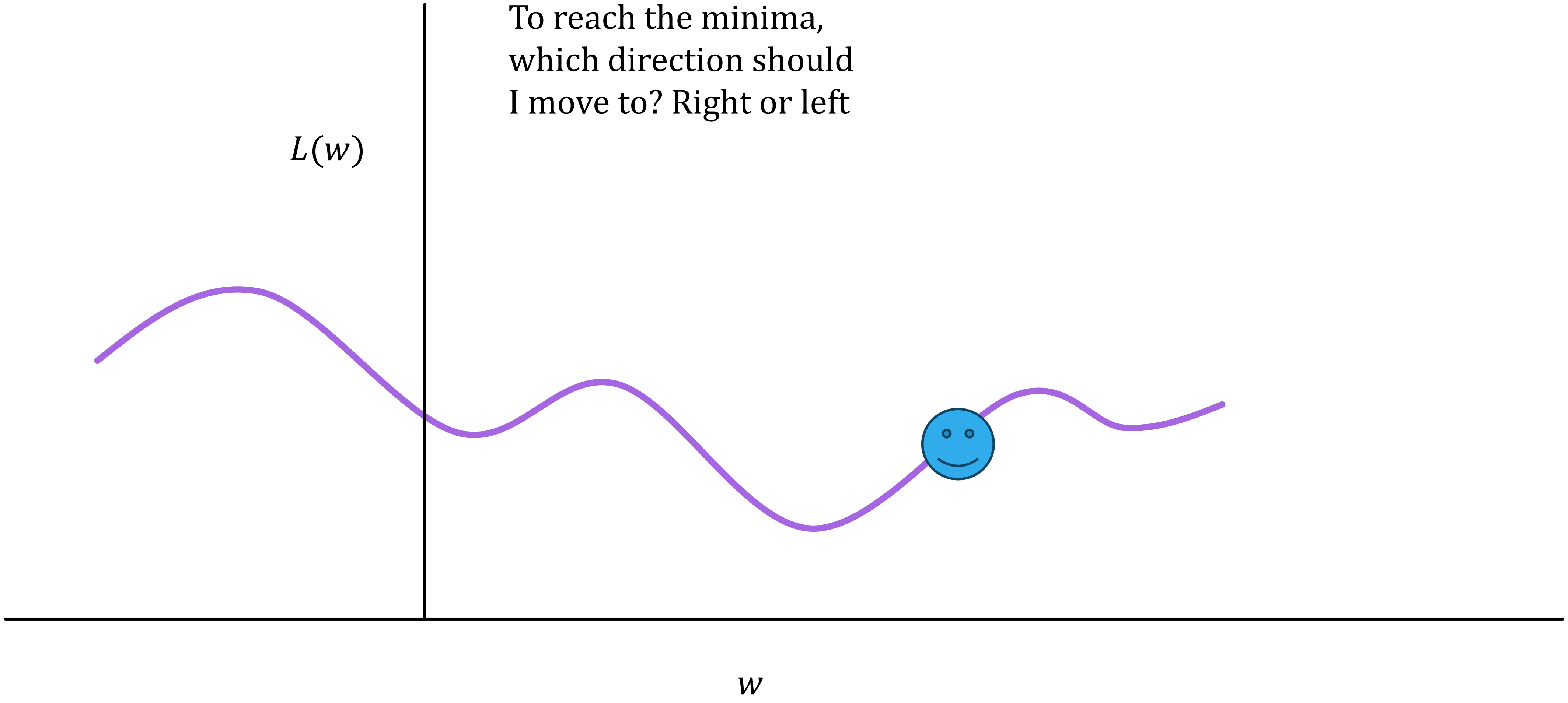
So, we want to increase the value of w. That is possible, if we do

$$w \leftarrow w - \eta \frac{\partial L(w)}{\partial w}$$

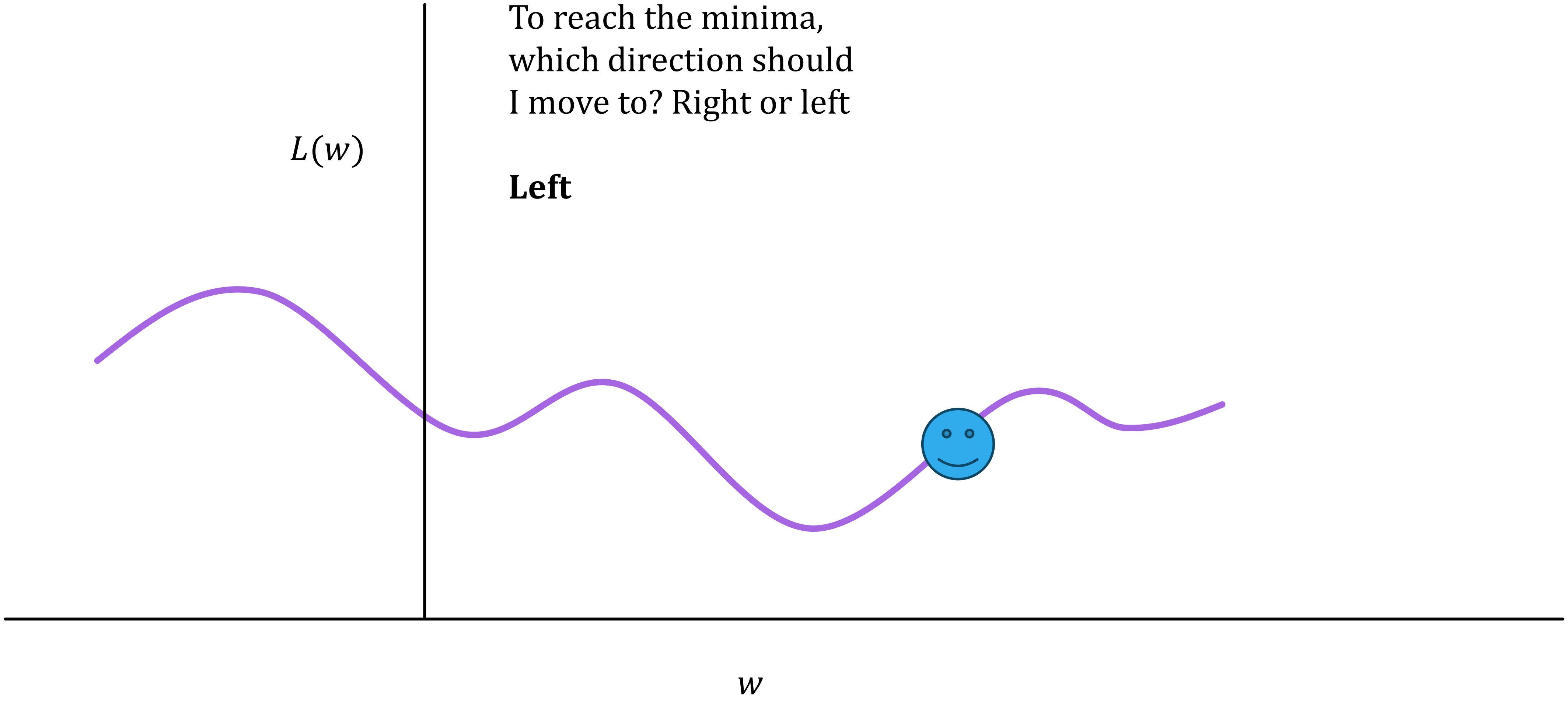
η is a positive constant

How to Reach the Global Minima: A 2D Example

To reach the minima, which direction should I move to? Right or left



How to Reach the Global Minima: A 2D Example



To reach the minima, which direction should I move to? Right or left

Left

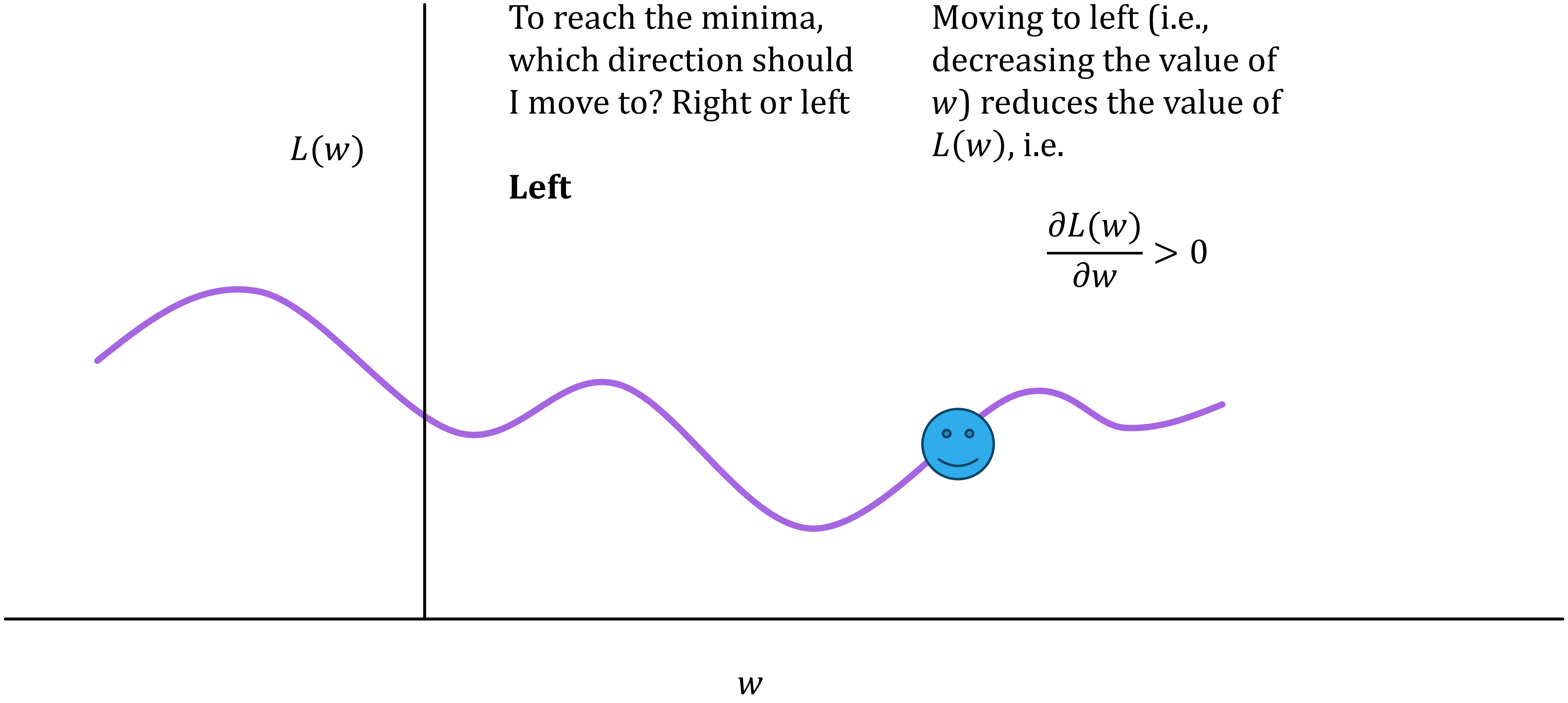
How to Reach the Global Minima: A 2D Example

To reach the minima, which direction should I move to? Right or left

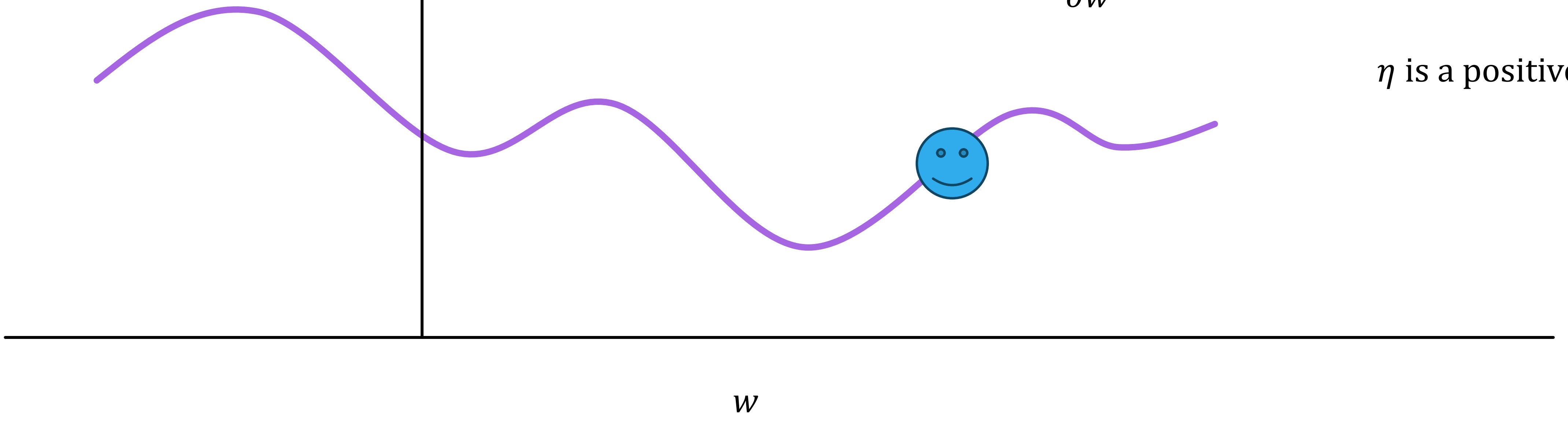
Left

Moving to left (i.e., decreasing the value of w) reduces the value of $L(w)$, i.e.

$$\frac{\partial L(w)}{\partial w} > 0$$



How to Reach the Global Minima: A 2D Example



To reach the minima, which direction should I move to? Right or left

Left

Moving to left (i.e., decreasing the value of w) reduces the value of $L(w)$, i.e.

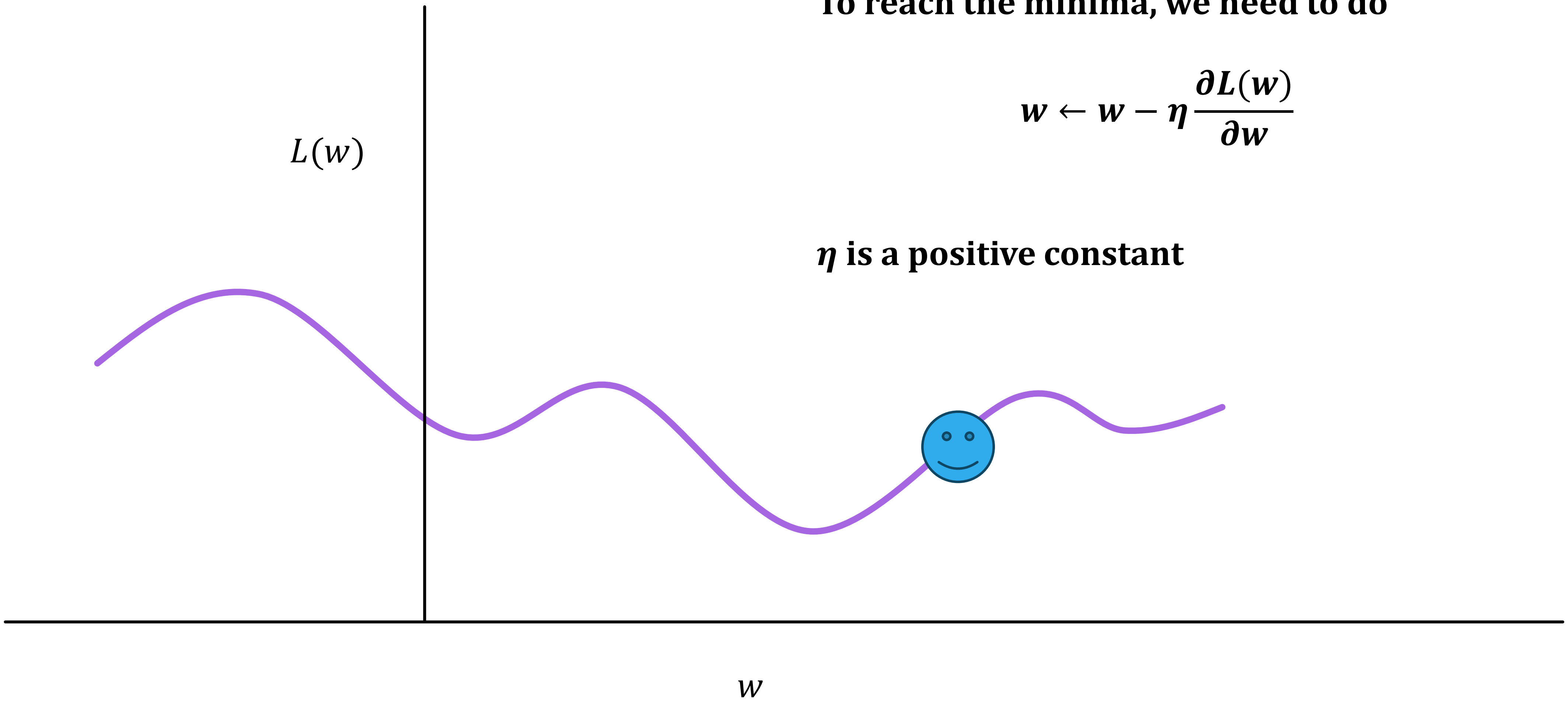
$$\frac{\partial L(w)}{\partial w} > 0$$

So, we want to decrease the value of w. That is possible, if we do

$$w \leftarrow w - \eta \frac{\partial L(w)}{\partial w}$$

η is a positive constant

How to Reach the Global Minima: A 2D Example



To reach the minima, we need to do

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{\eta}\frac{\partial \boldsymbol{L}(\boldsymbol{w})}{\partial \boldsymbol{w}}$$

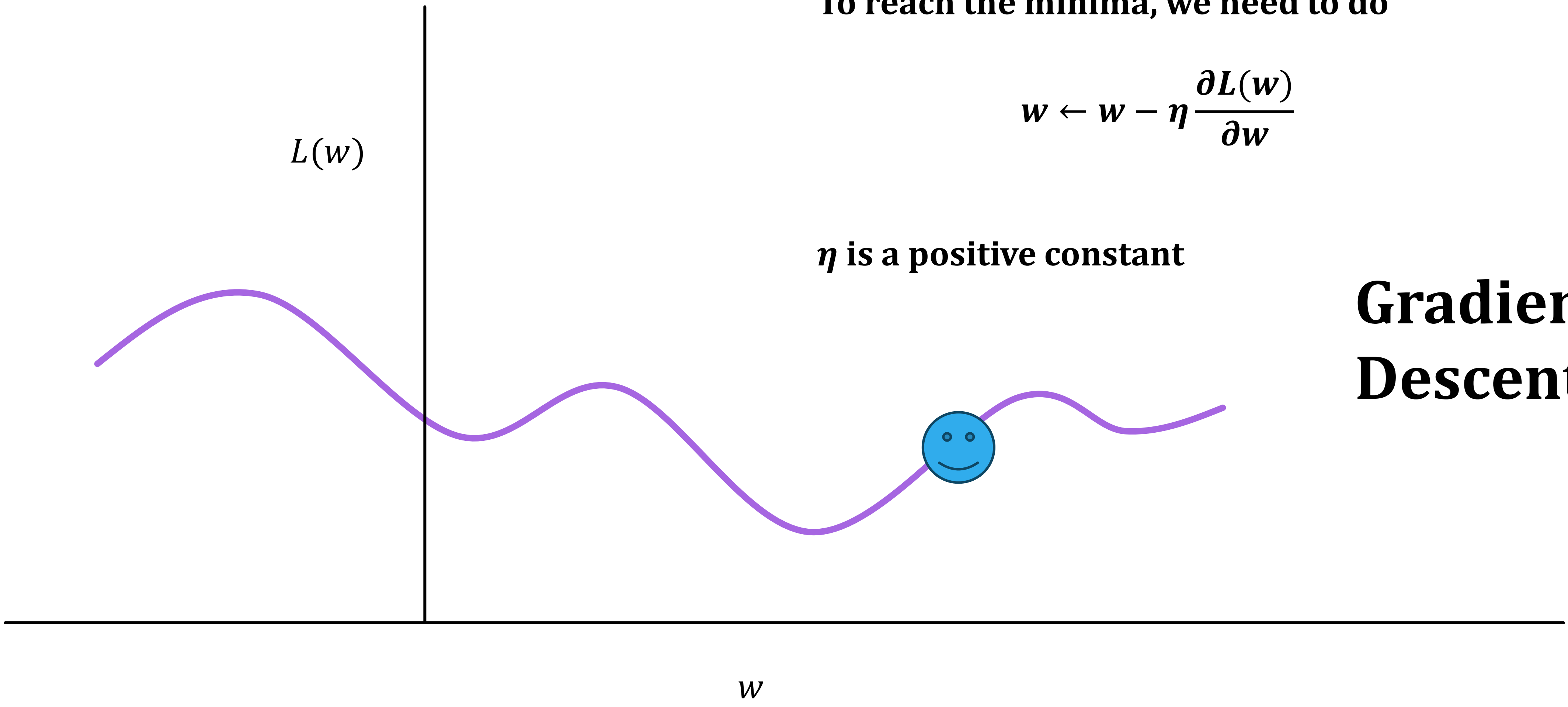
$\boldsymbol{\eta}$ **is a positive constant**

How to Reach the Global Minima: A 2D Example

To reach the minima, we need to do

$$\boldsymbol{w \leftarrow w - \eta \frac{\partial L(w)}{\partial w}}$$

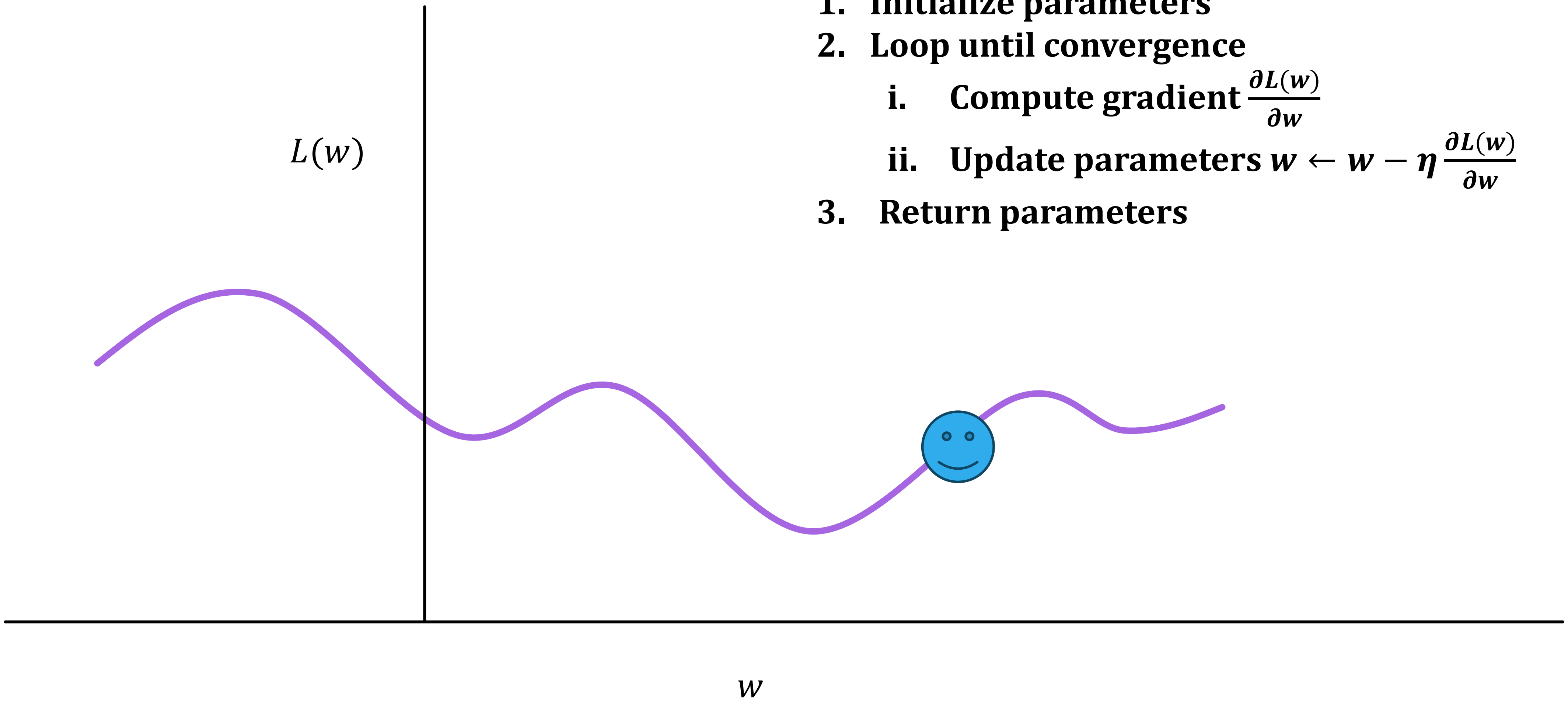
η is a positive constant



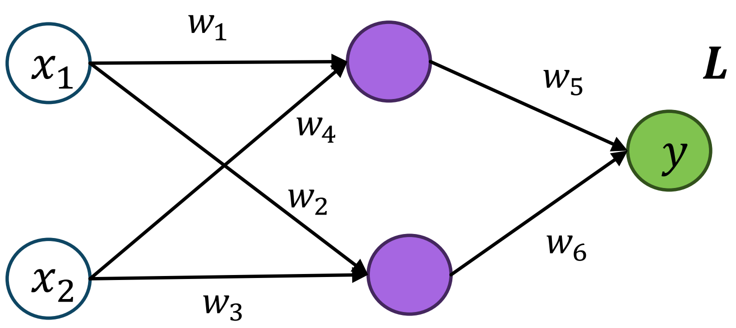
Gradient Descent

Gradient Descent

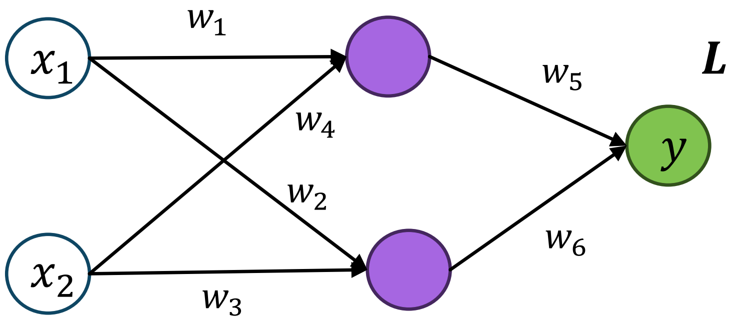
1. **Initialize parameters**
2. **Loop until convergence**
   i. **Compute gradient** $\frac{\partial L(w)}{\partial w}$
   ii. **Update parameters** $w \leftarrow w - \eta \frac{\partial L(w)}{\partial w}$
3. **Return parameters**



Computing Gradients: Backpropagation



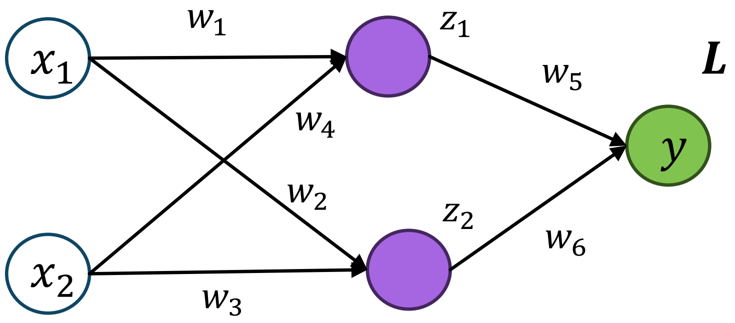
Computing Gradients: Backpropagation



To update w_5, we will do

$$w_5 \leftarrow w_5 - \eta \frac{\partial L}{\partial w_5}$$

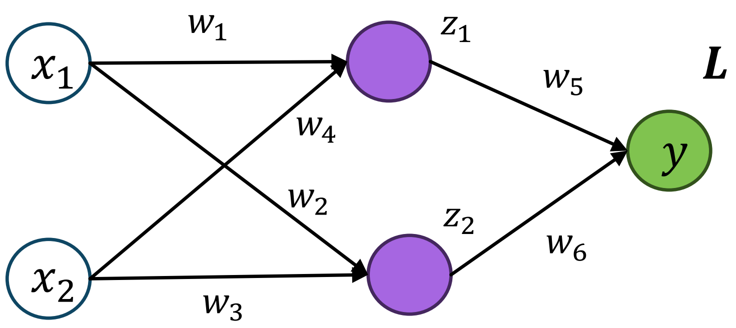
Computing Gradients: Backpropagation



To update w_6, we will do

$$w_6 \leftarrow w_6 - \eta \frac{\partial L}{\partial w_6}$$

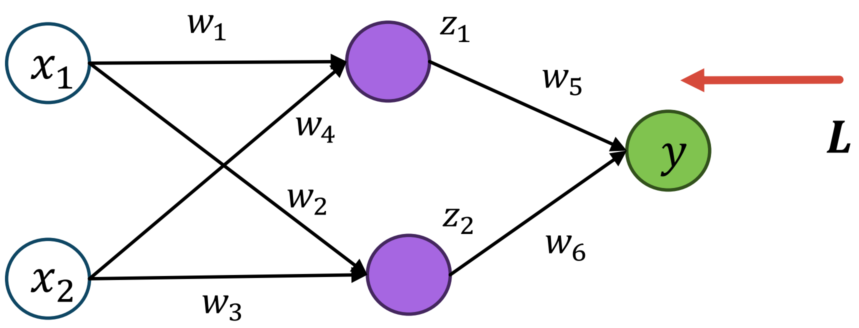
Computing Gradients: Backpropagation



To update w_1, we will do

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

Computing Gradients: Backpropagation

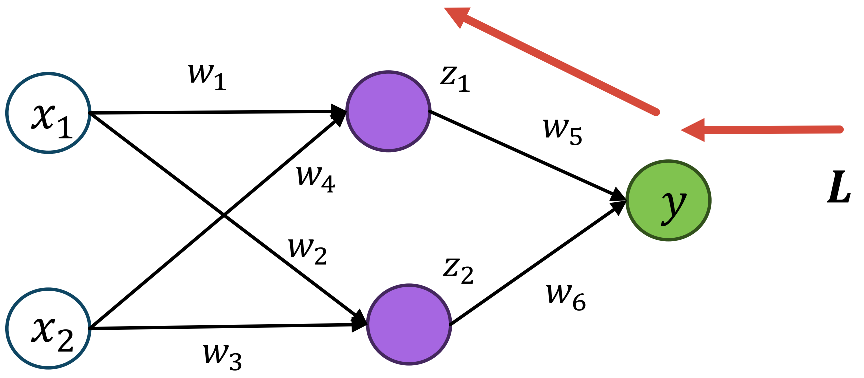


To update w_1, we will do

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial y} \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$

Computing Gradients: Backpropagation

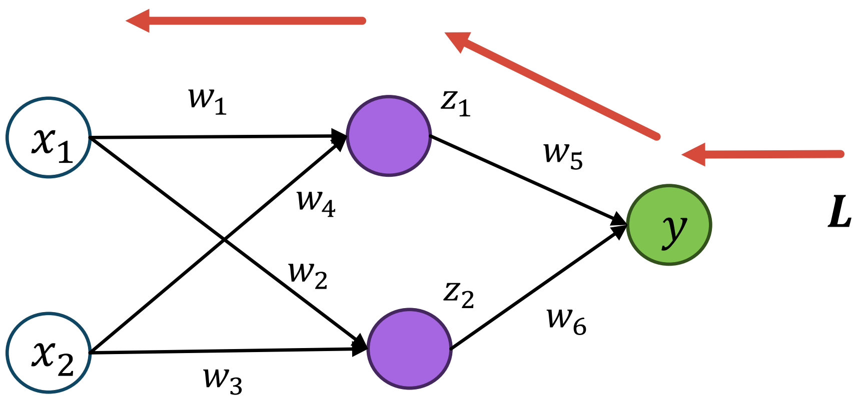


To update w_1, we will do

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial y} \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$

Computing Gradients: Backpropagation



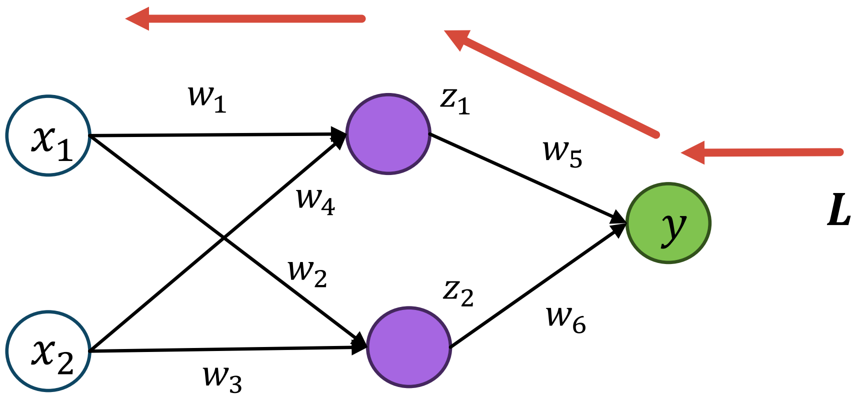
To update w_1, we will do

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial y} \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$

Chain rule

Computing Gradients: Backpropagation



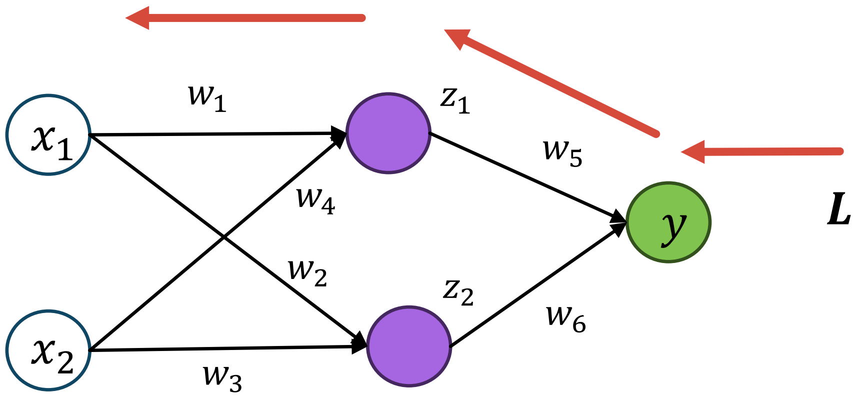
To update w_1, we will do

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial y} \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$

Similarly for other parameters

Differentiability



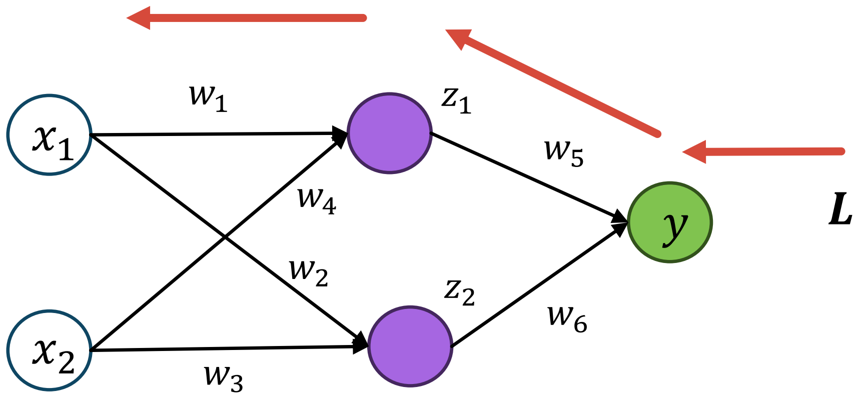
To update w_1, we will do

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial y} \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$

The loss function and the output of the nodes need to be differentiable

Differentiability



To update w_1, we will do

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial y} \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$

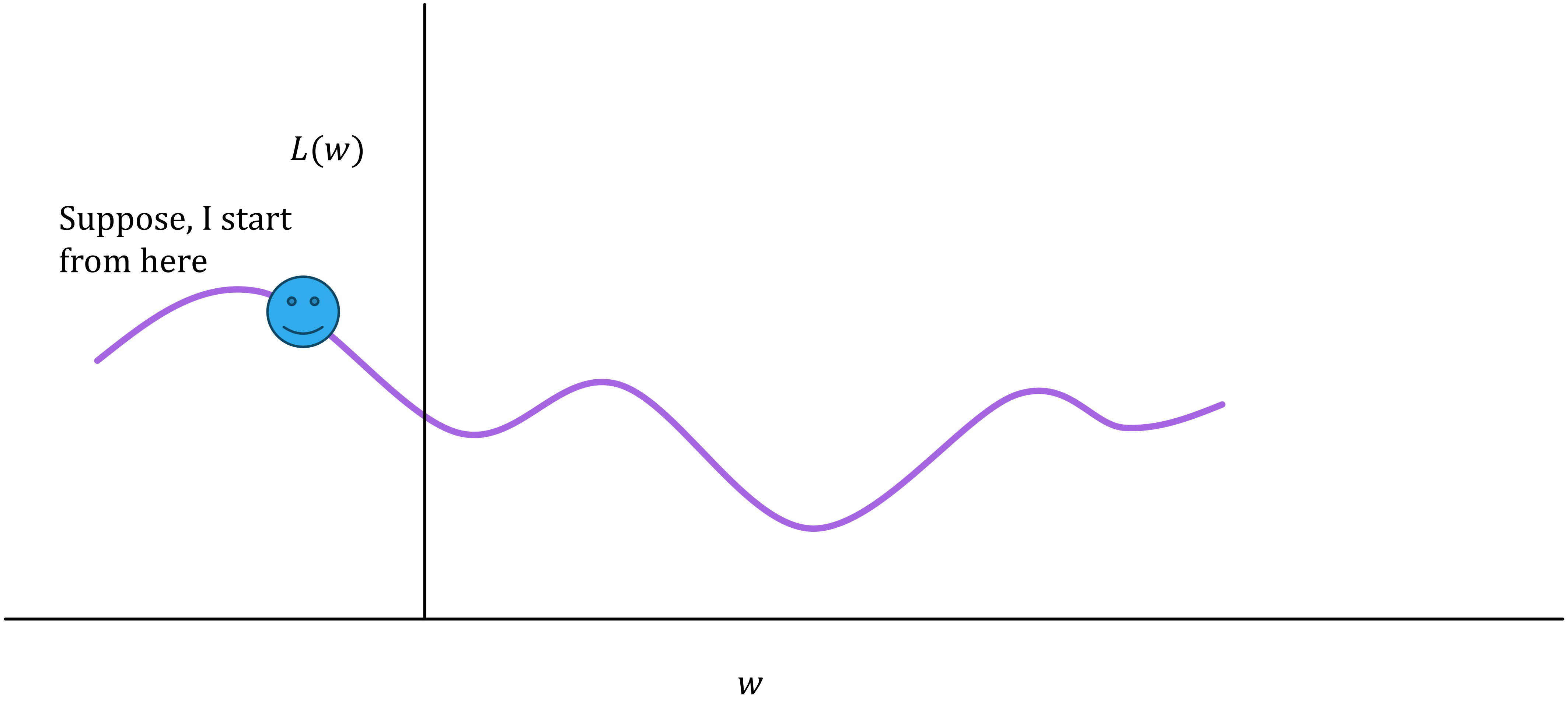
The loss function and the output of the nodes need to be differentiable

This is possible only if the activation function is differentiable

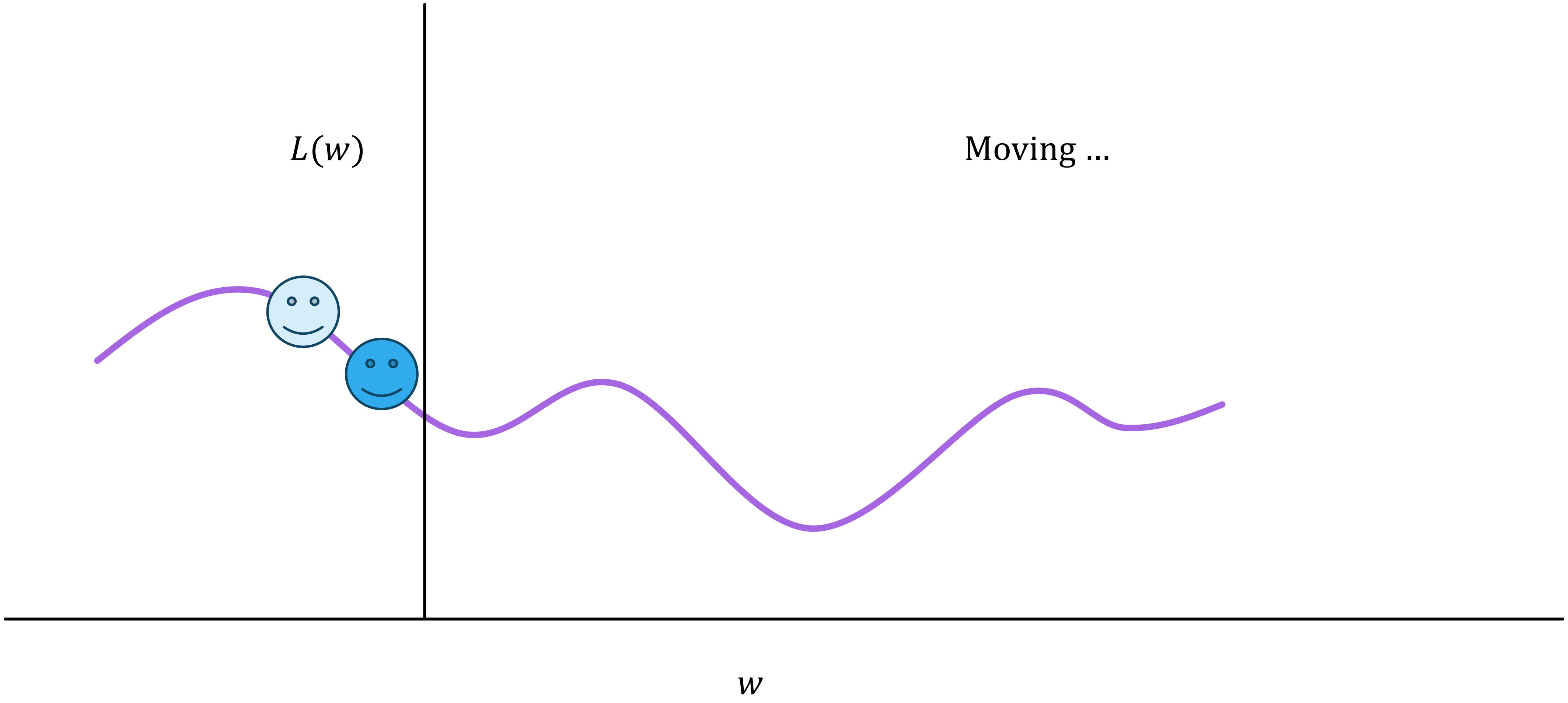
Optimizing the Loss

- May be challenging because of
  - Local minima
  - Saddle points
  - Vanishing gradients

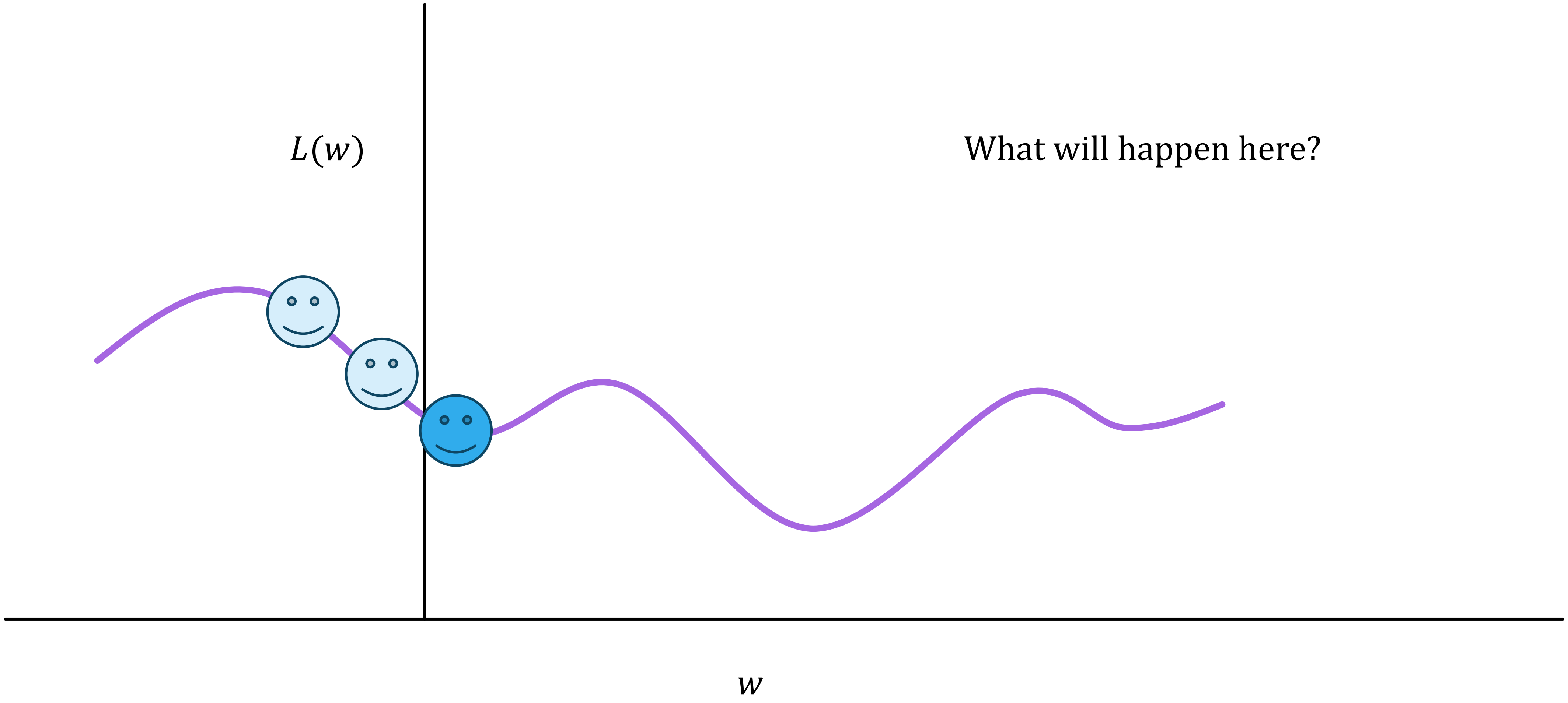
Local Minima



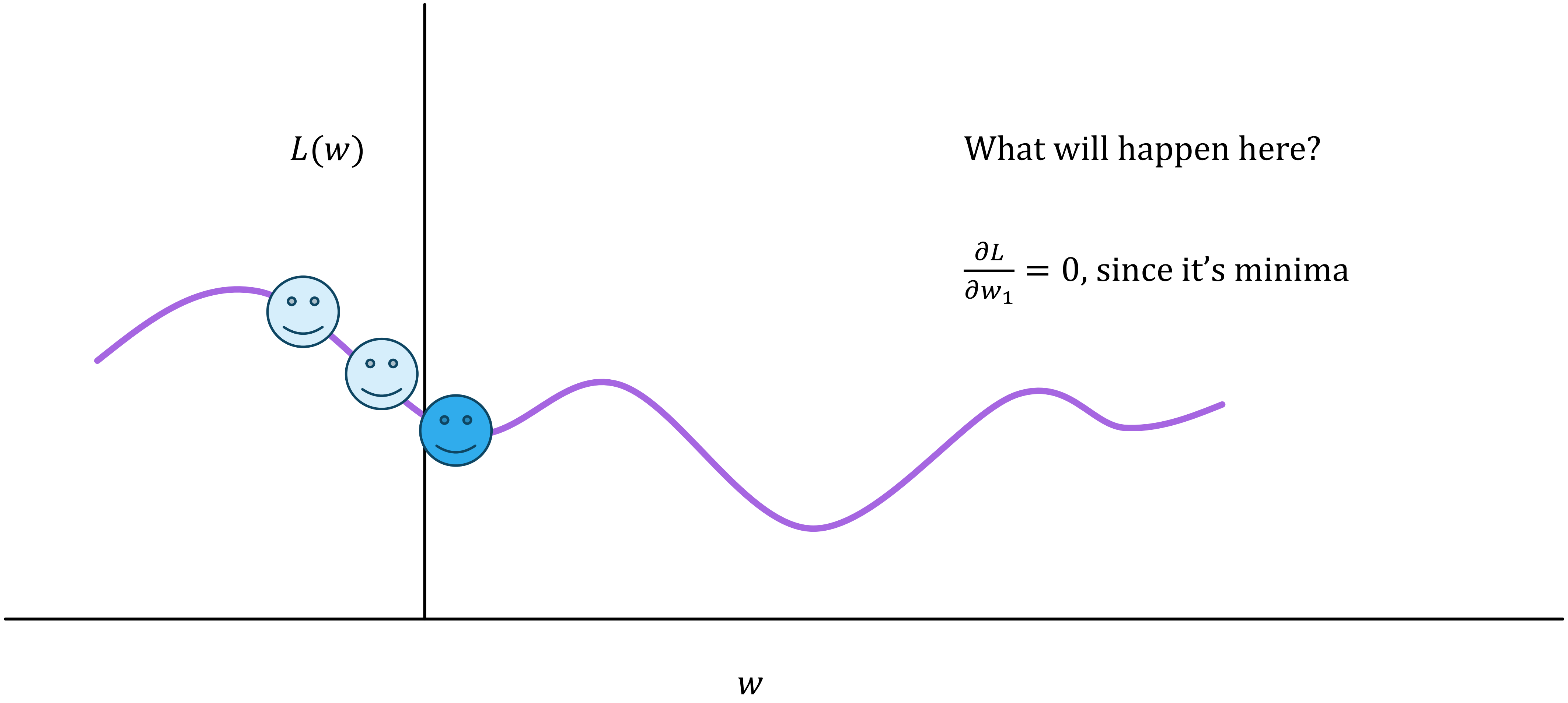
Local Minima



Local Minima



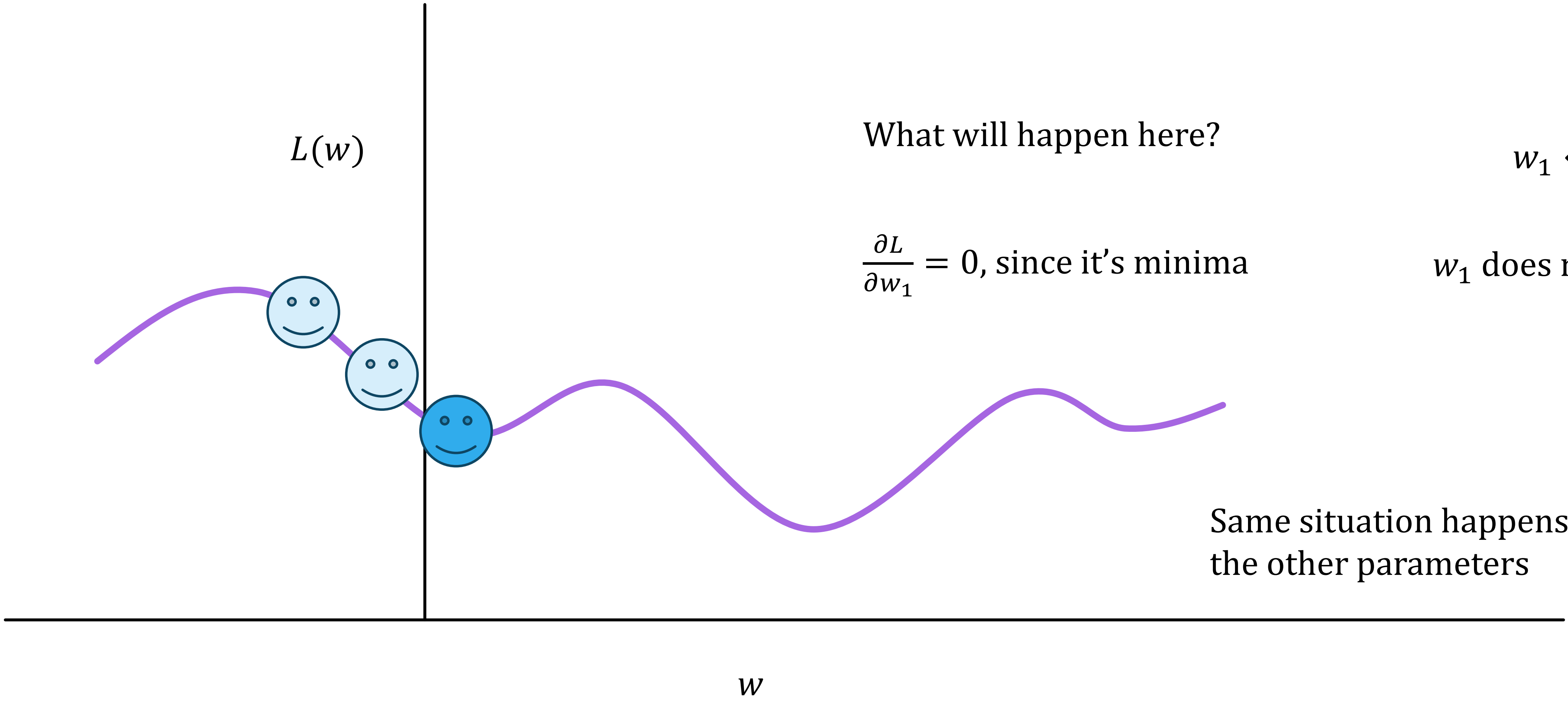
Local Minima



What will happen here?

$\frac{\partial L}{\partial w_1} = 0$, since it's minima

Local Minima



What will happen here?

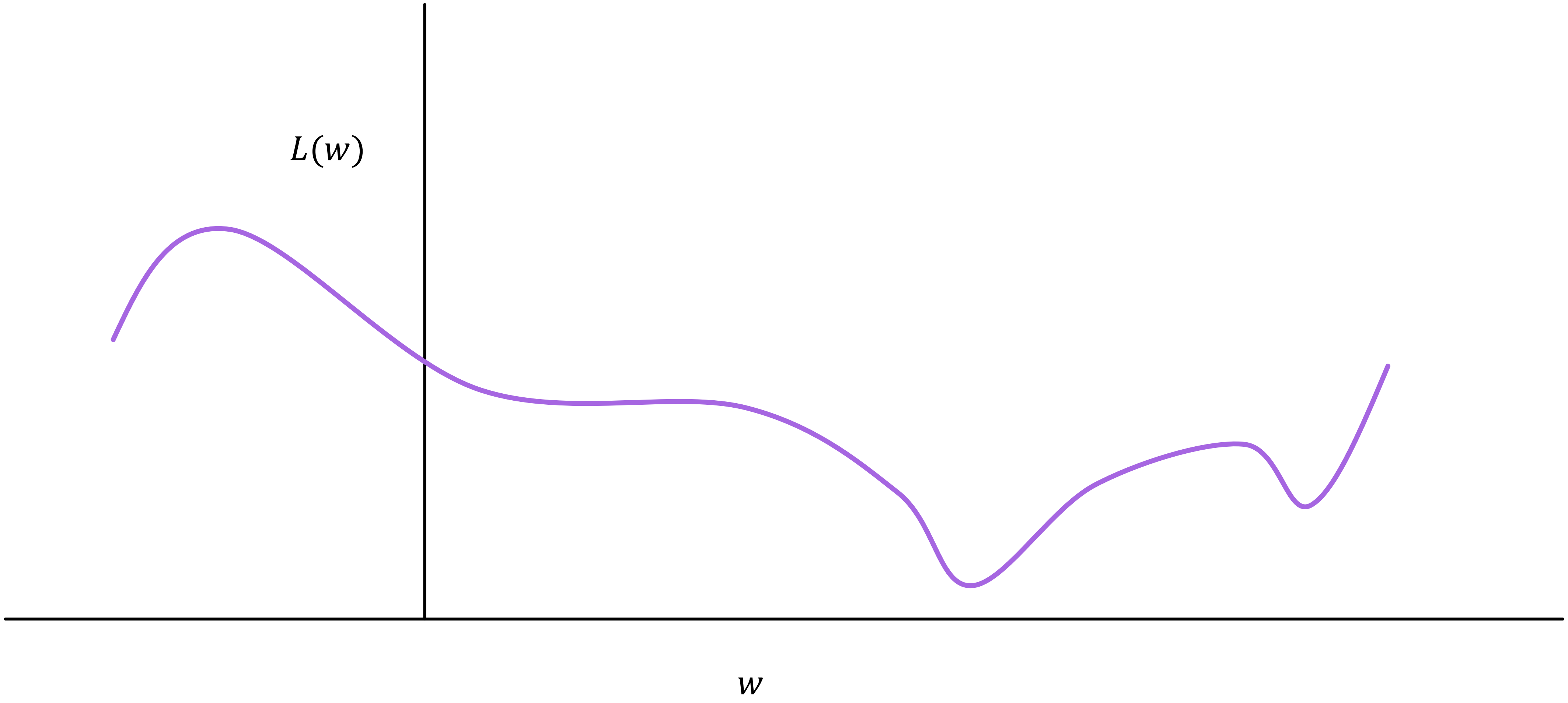
$\frac{\partial L}{\partial w_1} = 0$, since it's minima

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

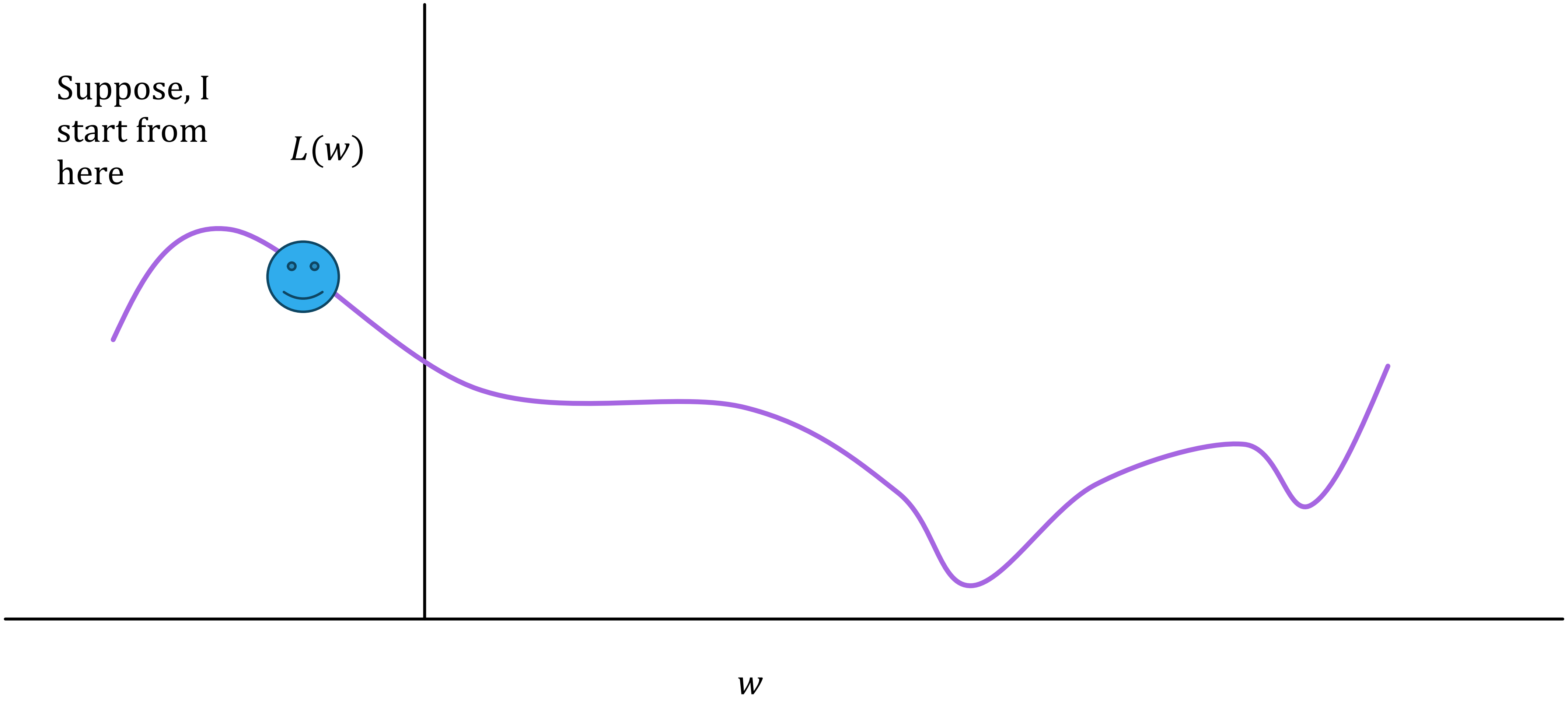
w_1 does not change further

Same situation happens for the other parameters

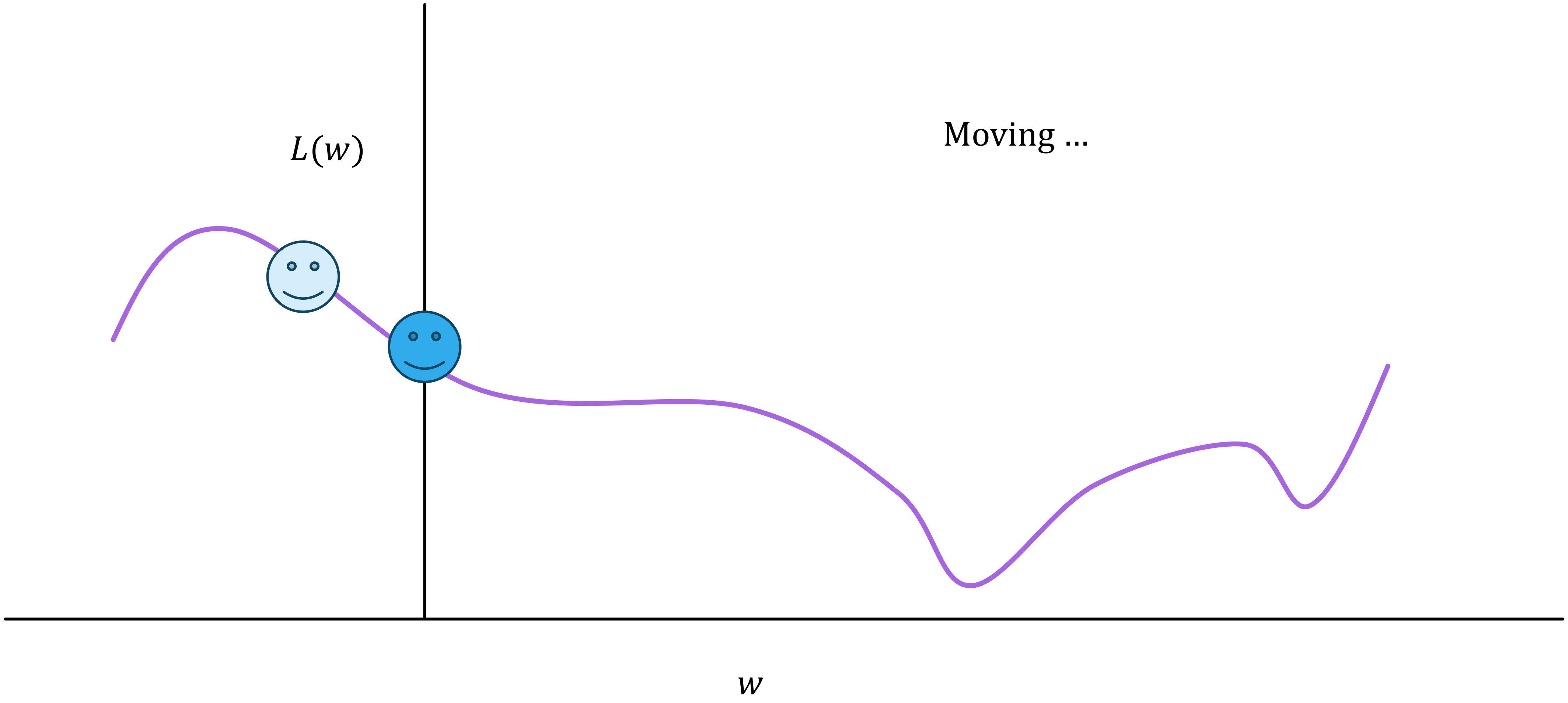
Saddle Points



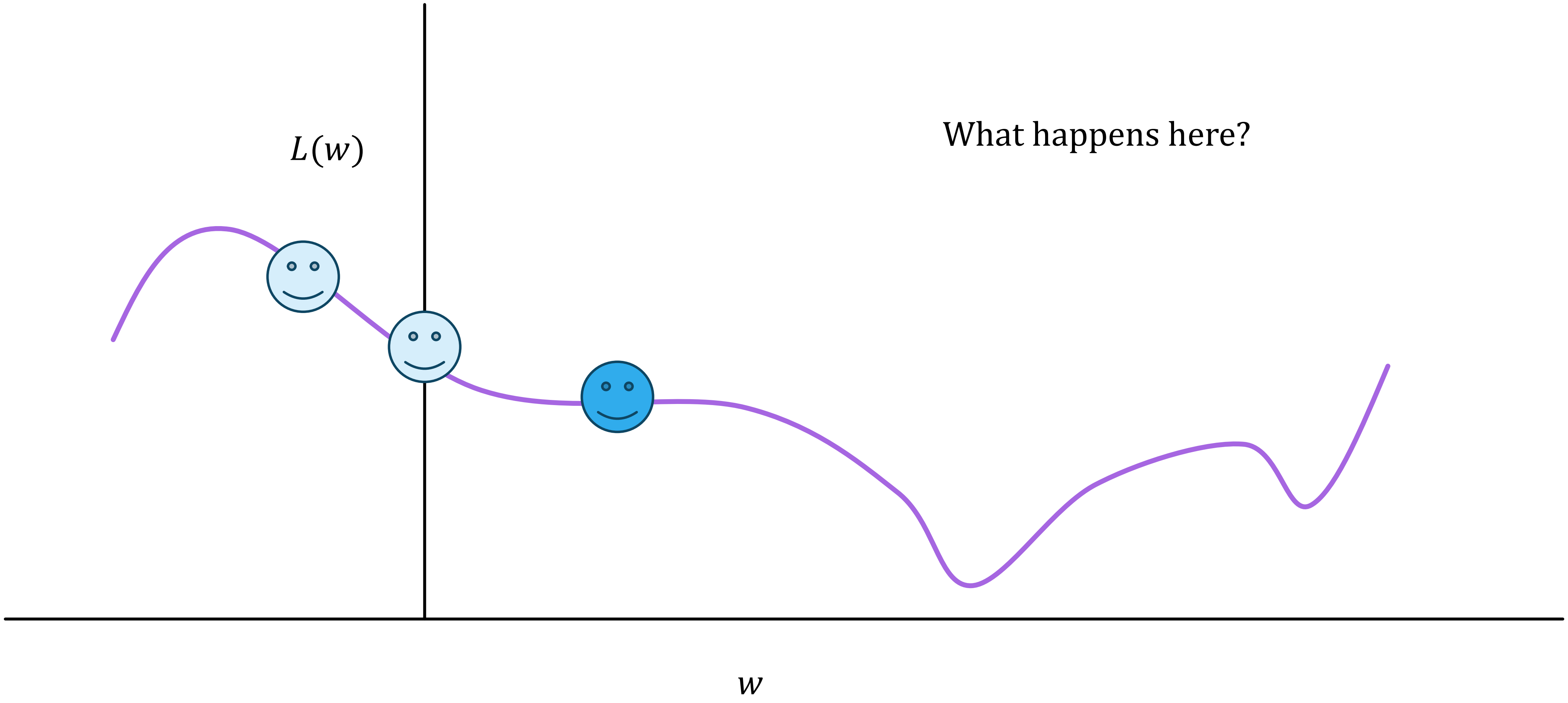
Saddle Points



Saddle Points



Saddle Points



Saddle Points

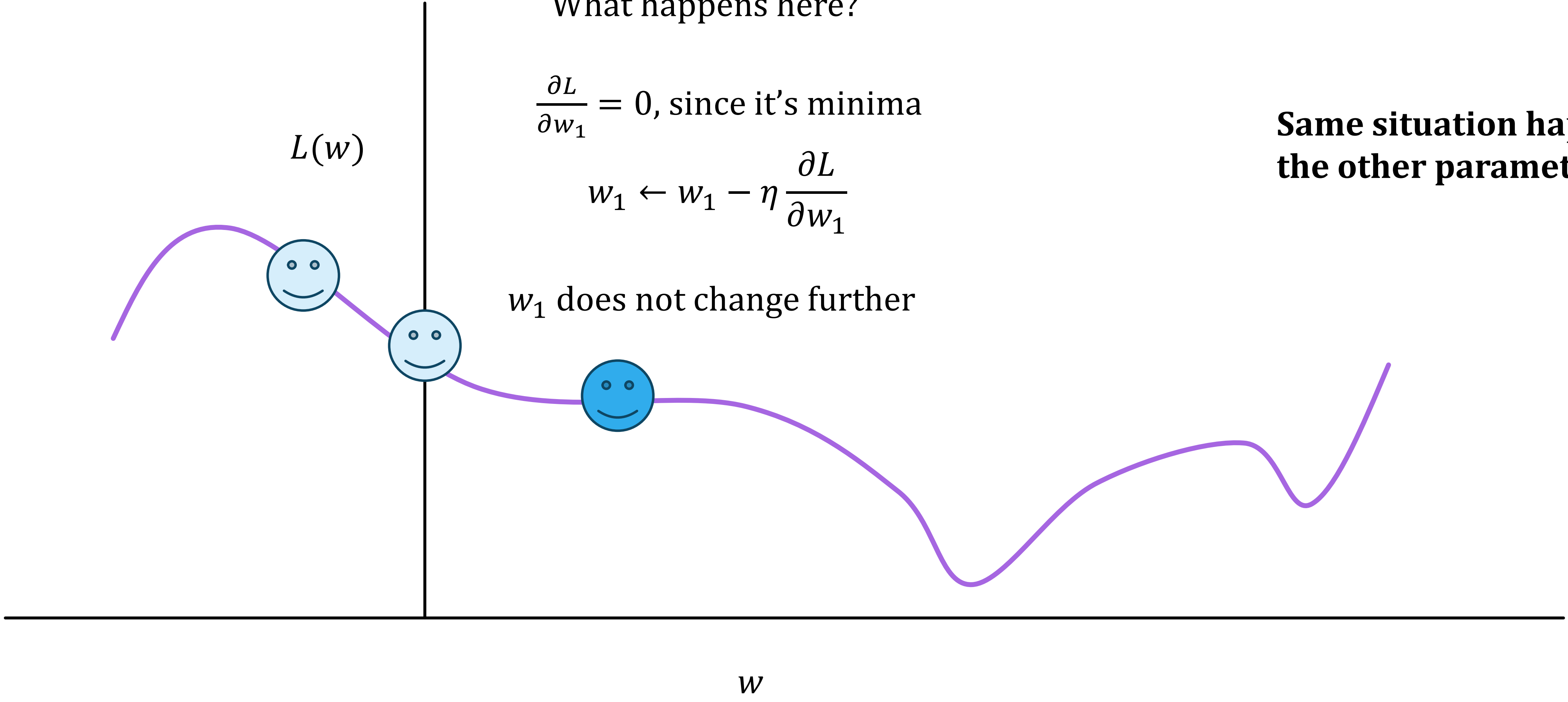
What happens here?

$$\frac{\partial L}{\partial w_1} = 0\text{, since it's minima}$$

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

w_1 does not change further

Same situation happens for the other parameters



Vanishing Gradient

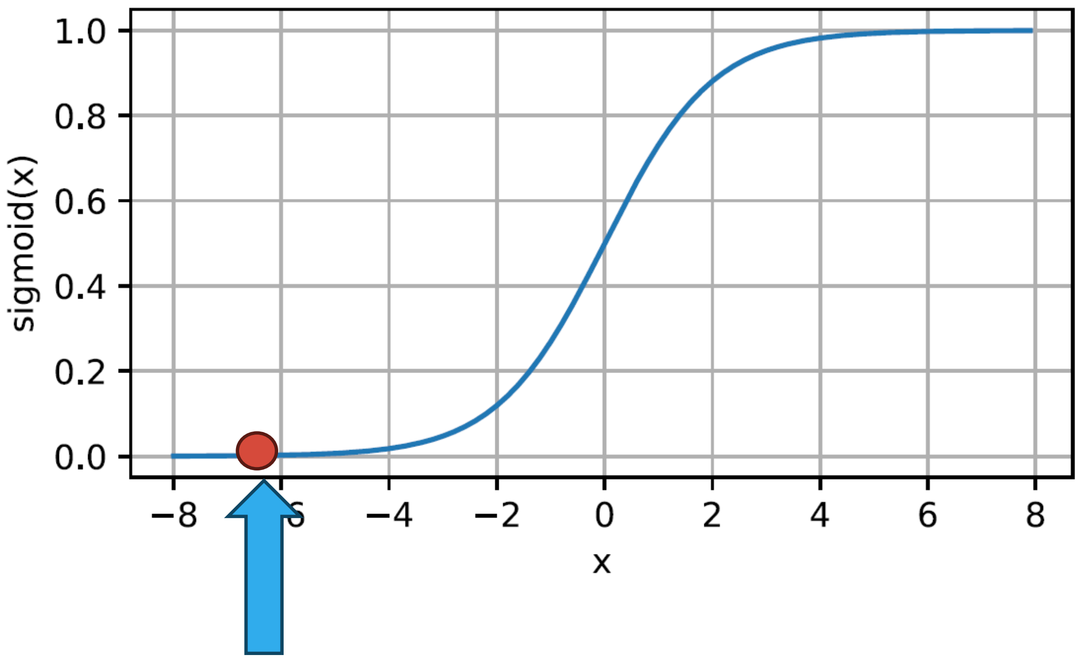
$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$

For w_1 to change, $\frac{\partial L}{\partial z_1}$ and $\frac{\partial z_1}{\partial w_1}$ must be non-zero

Vanishing Gradient

$$w_n \leftarrow w_n - \eta \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial z_2} \dots \frac{\partial z_n}{\partial w_n}$$

For w_1 to change, each of $\frac{\partial L}{\partial z_1}, \frac{\partial z_1}{\partial z_2}, \dots \frac{\partial z_n}{\partial w_n}$ must be non-zero

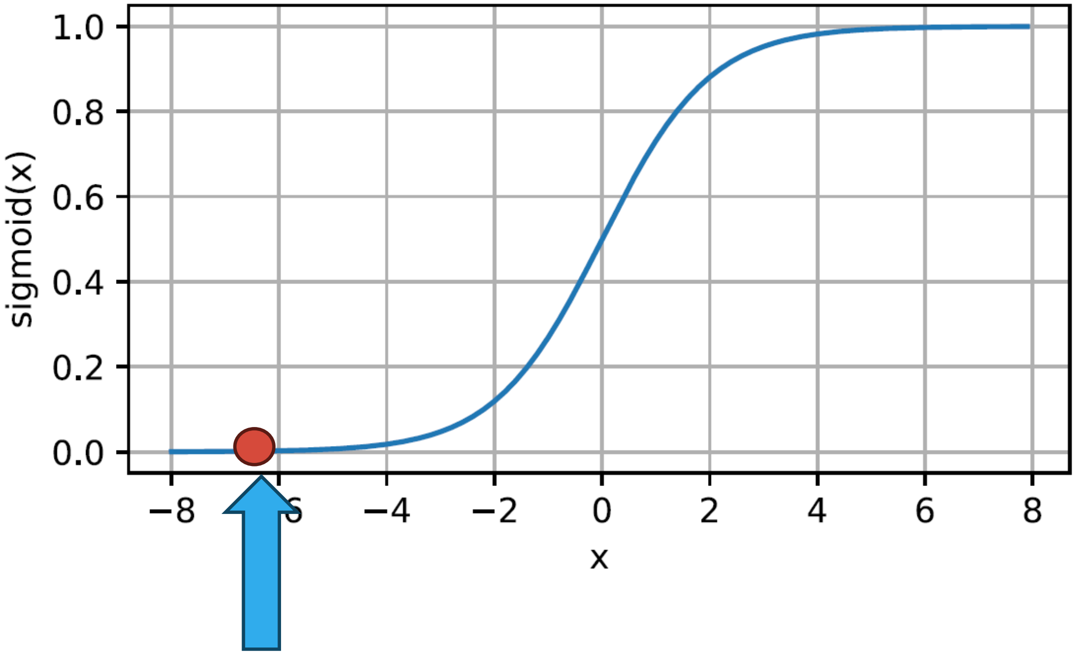


For any of the nodes related to the derivatives, if we are here, the derivative term will be zero

Vanishing Gradient

$$\boldsymbol{w_n \leftarrow w_n - \eta \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial z_2} \cdots \frac{\partial z_n}{\partial w_n}}$$

For w_1 to change, each of $\frac{\partial L}{\partial z_1}, \frac{\partial z_1}{\partial z_2}, \ldots \frac{\partial z_n}{\partial w_n}$ must be non-zero



For any of the nodes related to the derivatives, if we are here, the derivative term will be zero

Since $\boldsymbol{w_n \leftarrow w_n - \eta \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial z_2} \cdots \frac{\partial z_n}{\partial w_n}}$

The parameter will not get updated

Vanishing Gradient

$$\boldsymbol{w_n \leftarrow w_n - \eta \frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial z_2} \cdots \frac{\partial z_n}{\partial w_n}}$$

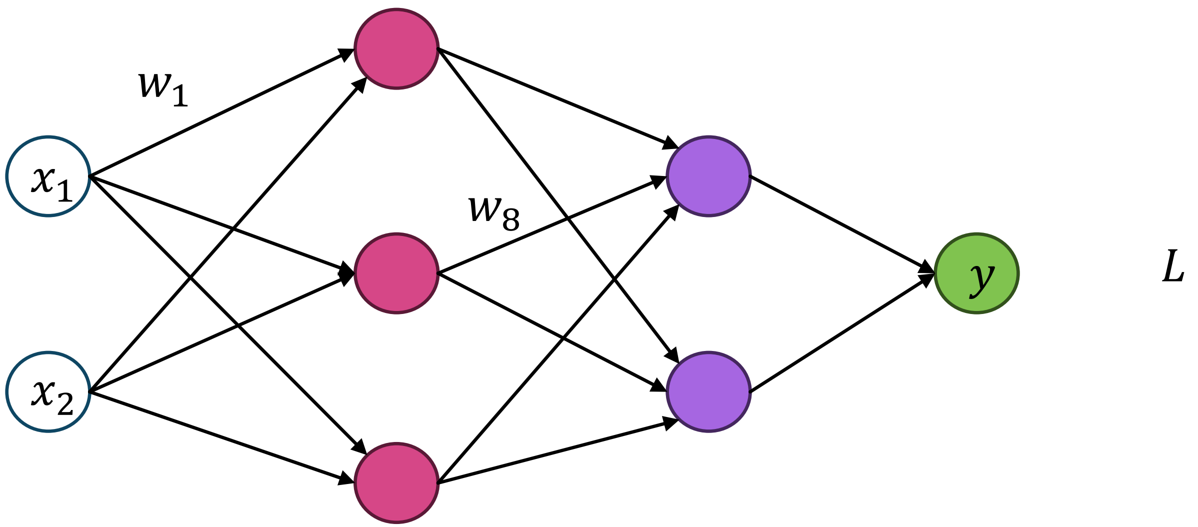
If each of $\frac{\partial L}{\partial z_1}, \frac{\partial z_1}{\partial z_2}, \dots \frac{\partial z_n}{\partial w_n}$ are < 1, the product will be very small and the gradient will not get updated significantly

Exploding Gradient

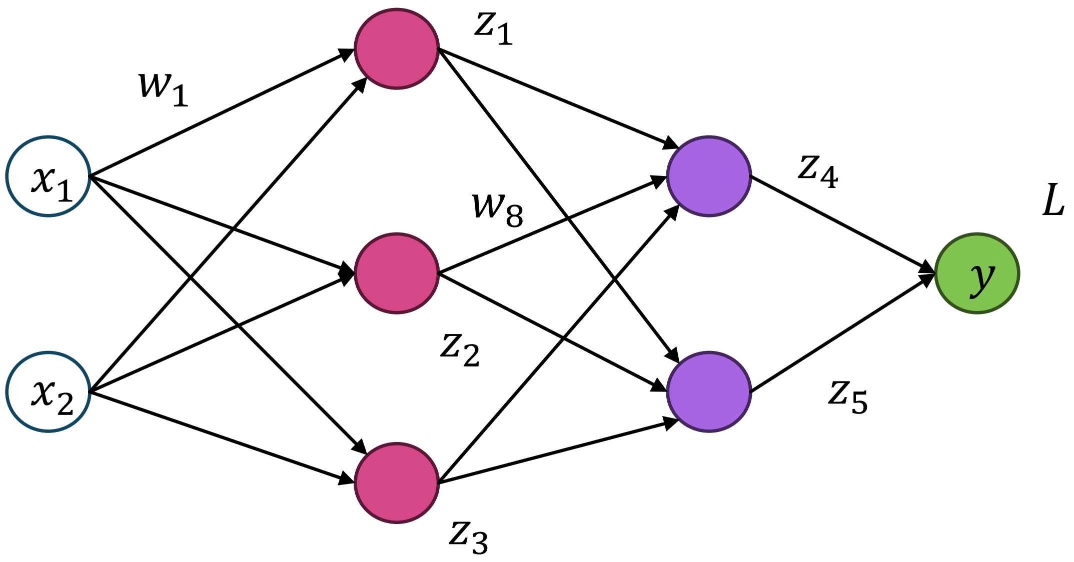
$$w_n \leftarrow w_n - \eta \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial z_2} \dots \frac{\partial z_n}{\partial w_n}$$

If each of $\frac{\partial L}{\partial z_1}, \frac{\partial z_1}{\partial z_2}, \dots \frac{\partial z_n}{\partial w_n}$ are > 1, the product will be very large and the algorithm may overshoot the minima

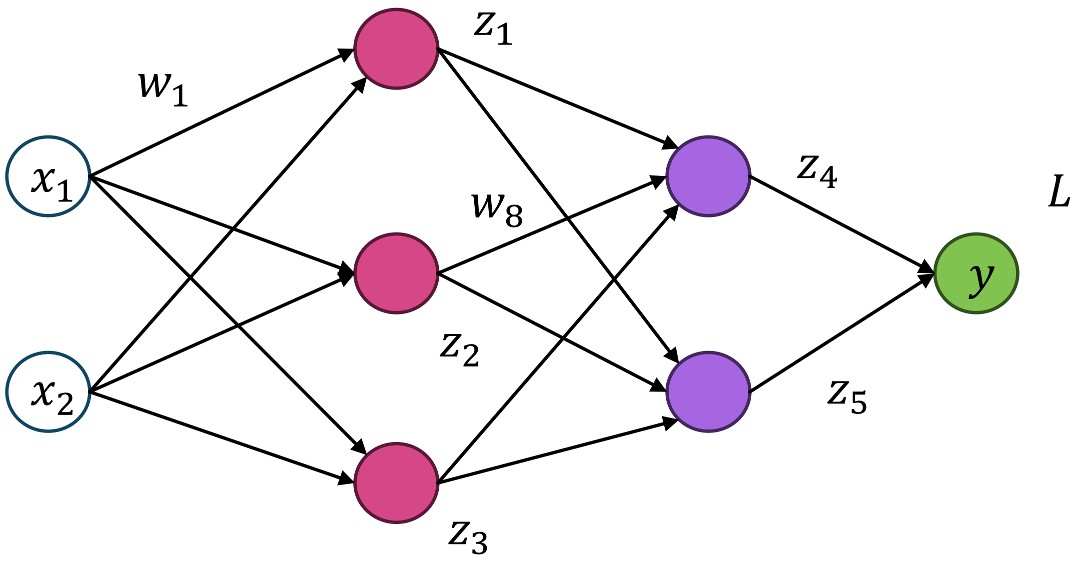
Backpropagation: Example



Backpropagation: Example

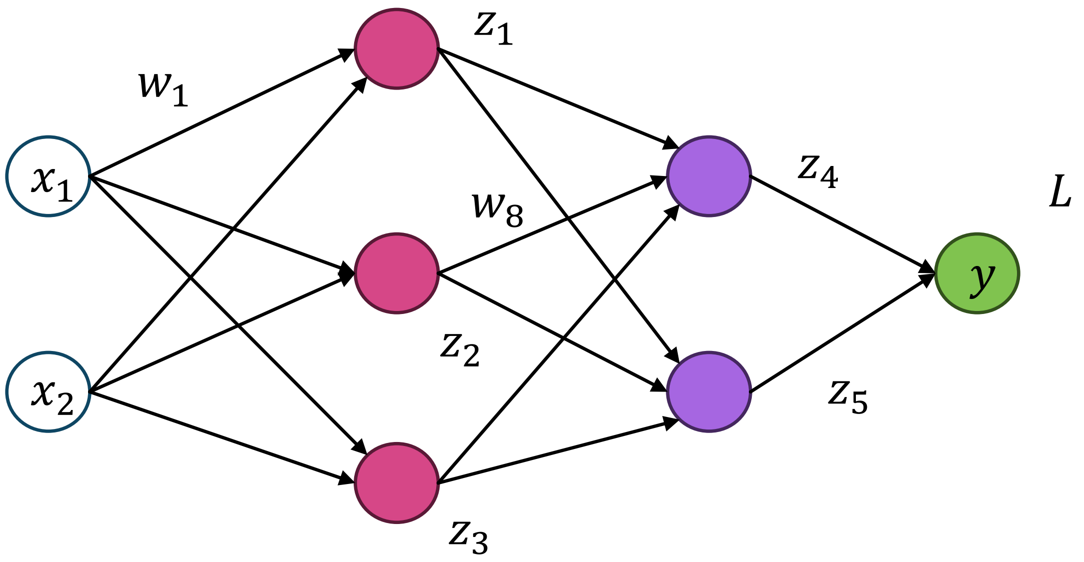


Backpropagation: Example



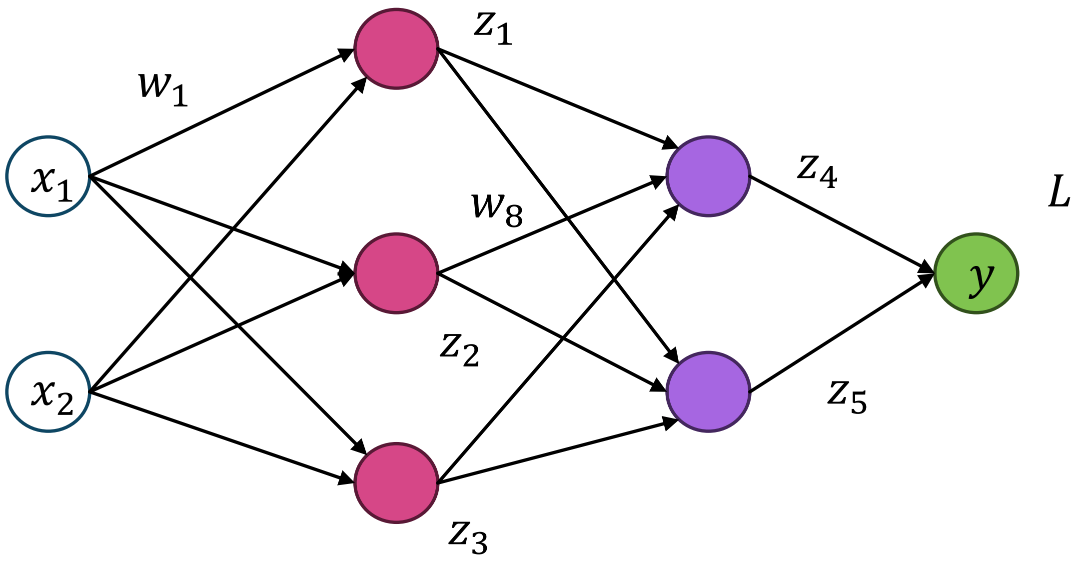
$$\frac{\partial L}{\partial w_8} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_8}$$

Backpropagation: Example



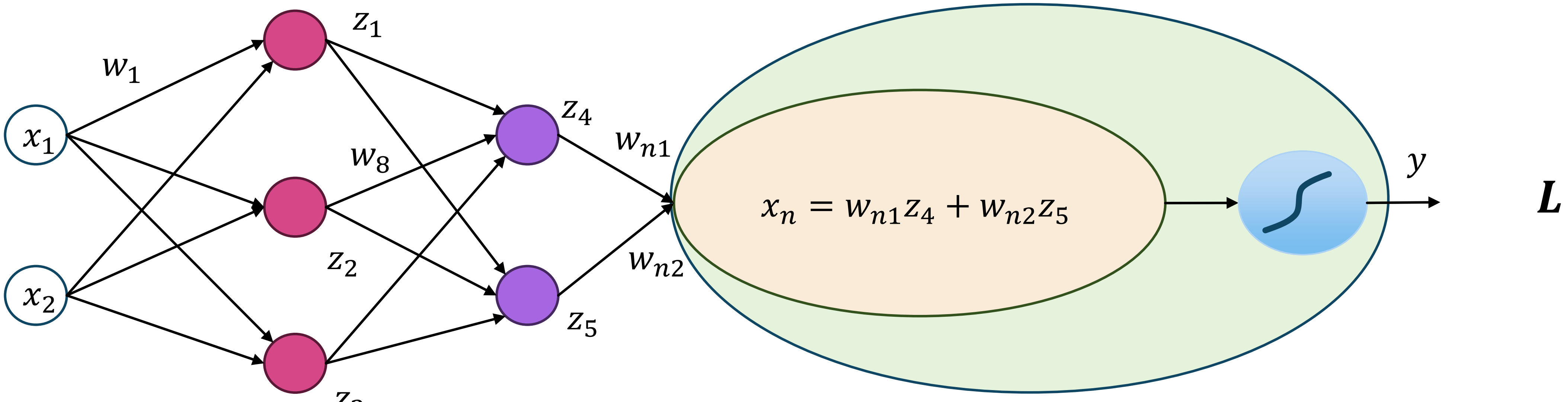
$$\frac{\partial L}{\partial w_1} = ?$$

Backpropagation: Example



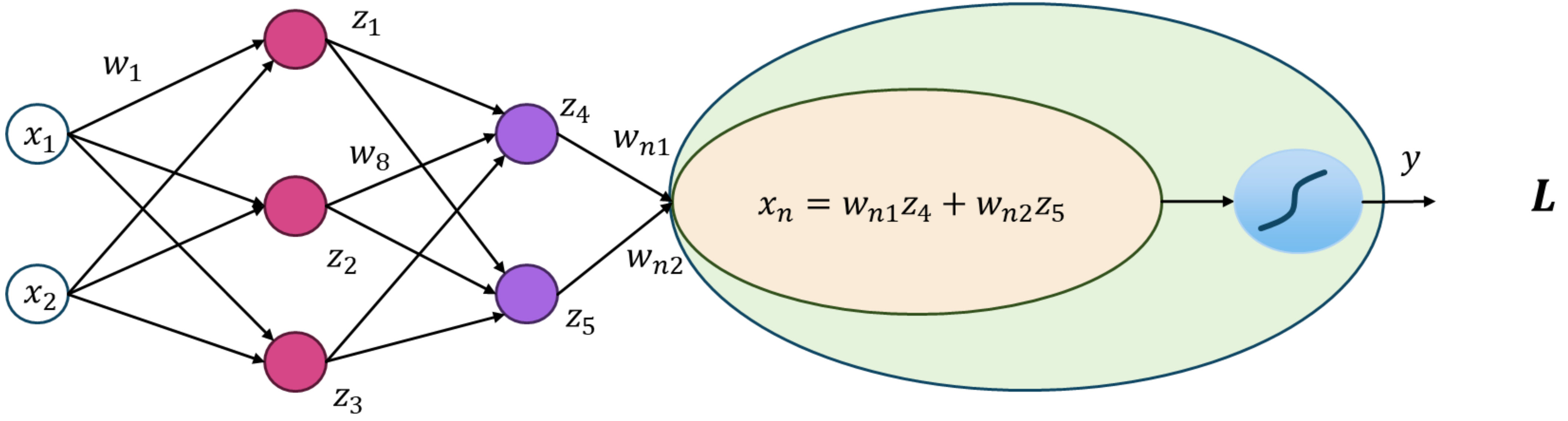
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \; Then?$$

Backpropagation: Example



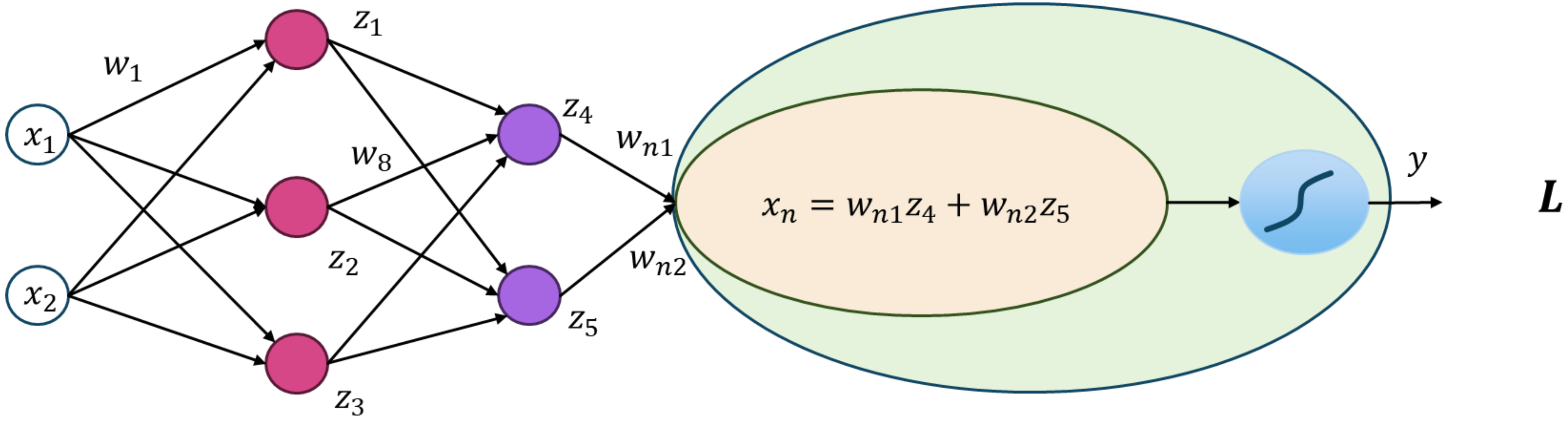
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \; Then?$$

Backpropagation: Example



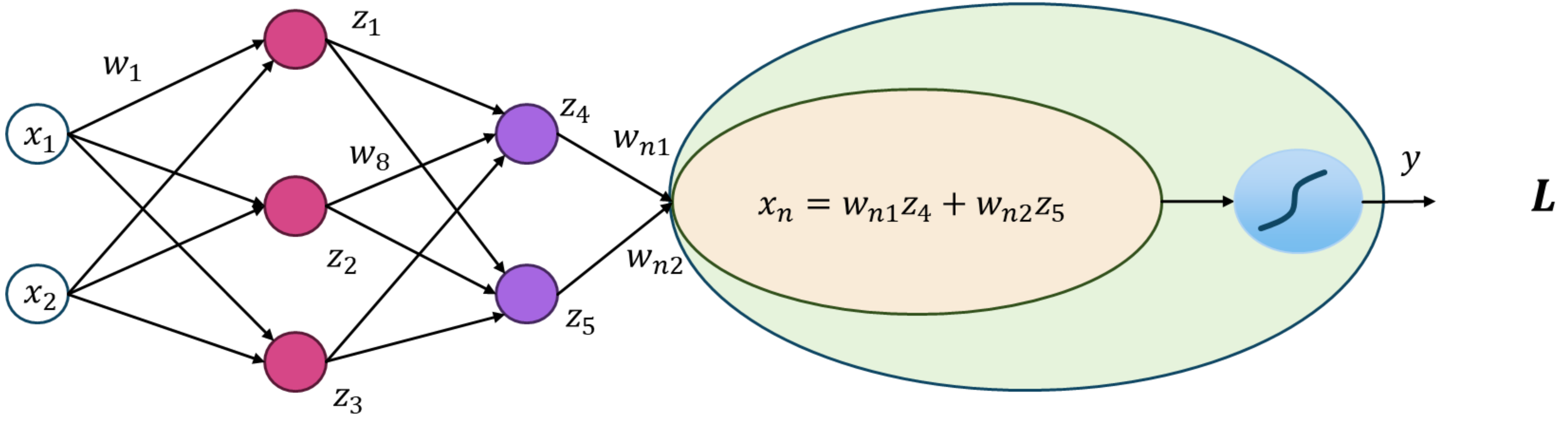
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x_n}\frac{\partial x_n}{\partial w_1}$$

Backpropagation: Example



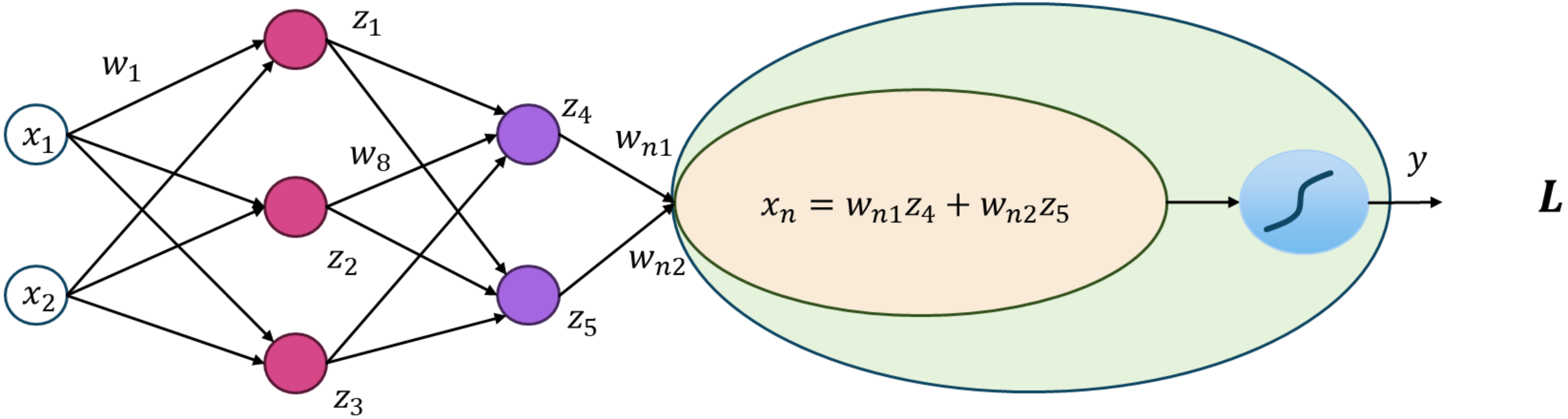
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x_n}\frac{\partial (w_{n1}z_4 + w_{n2}z_5)}{\partial w_1}$$

Backpropagation: Example



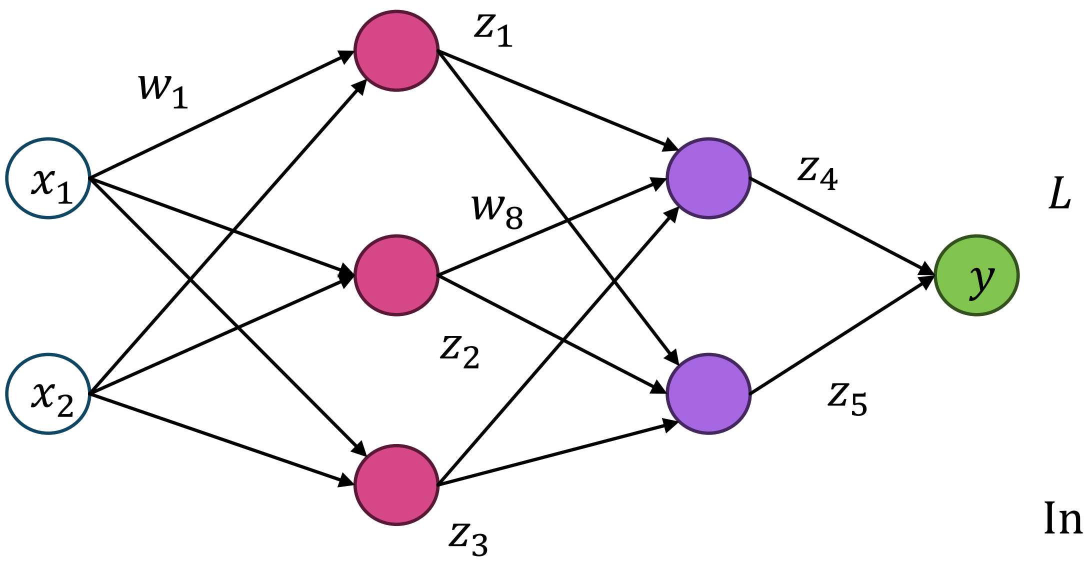
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x_n}\left(w_{n1}\frac{\partial z_4}{\partial w_1} + w_{n2}\frac{\partial z_5}{\partial w_1}\right)$$

Backpropagation: Example



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x_n}\left(w_{n1}\frac{\partial z_4}{\partial z_1}\frac{\partial z_1}{\partial w_1} + w_{n2}\frac{\partial z_5}{\partial z_1}\frac{\partial z_1}{\partial w_1}\right)$$

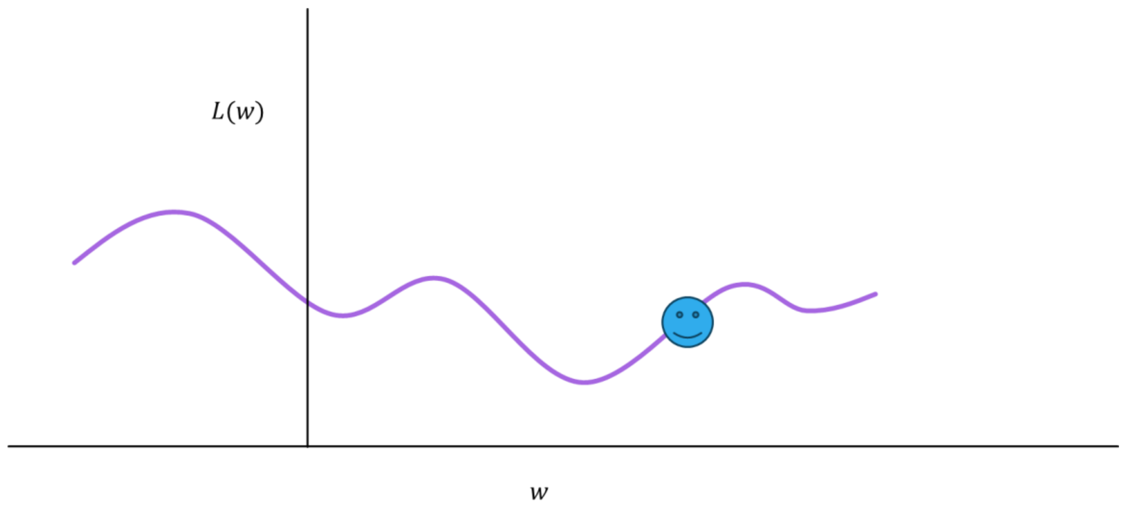
Backpropagation: Example



$$\frac{\partial L}{\partial w_8} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_8}$$

In the same manner as that of the previous, we can prove this as well

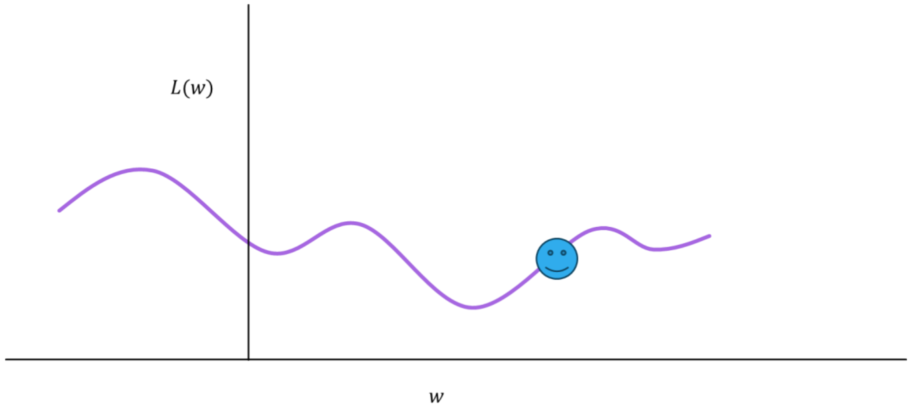
Gradient Descent: Challenges



1. **Initialize parameters**
2. **Loop until convergence**
   i. **Compute gradient** $\frac{\partial L(w)}{\partial w}$
   ii. **Update parameters** $w \leftarrow w - \eta \frac{\partial L(w)}{\partial w}$
3. **Return parameters**

Epoch

Gradient Descent: Challenges



1. **Initialize parameters**
2. **Loop until convergence**
   i. **Calculate loss $L(w)$**
   ii. **Compute gradient $\frac{\partial L(w)}{\partial w}$**
   iii. **Update parameters $w \leftarrow w - \eta \frac{\partial L(w)}{\partial w}$**
3. **Return parameters**

Epoch

In Batch Gradient Descent with N samples (data points),

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} L_i(w)$$

$L_i(w)$: Loss for the i^{th} data point

Gradient Descent: Challenges

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Calculate loss $\boldsymbol{L(w)}$**
   ii. **Compute gradient $\boldsymbol{\frac{\partial L(w)}{\partial w}}$**
   iii. **Update parameters $\boldsymbol{w \leftarrow w - \eta \frac{\partial L(w)}{\partial w}}$**
3. **Return parameters**

Epoch

In Batch Gradient Descent with N samples (data points),

$$L(w) = \frac{1}{N} \sum_{i=1}^{N} L_i(w)$$

$$\frac{\partial L(w)}{\partial w} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial L_i(w)}{\partial w}$$

Gradient Descent: Challenges

In Batch Gradient Descent with N samples (data points),

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Calculate loss $L(w)$**
   ii. **Compute gradient $\frac{\partial L(w)}{\partial w}$**
   iii. **Update parameters $w \leftarrow w - \eta \frac{\partial L(w)}{\partial w}$**
3. **Return parameters**

Epoch

$$L(w) = \frac{1}{N} \sum_{i=1}^{N} L_i(w)$$

$$\frac{\partial L(w)}{\partial w} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial L_i(w)}{\partial w}$$

For each update, it will take a long time if we have many training samples

We will also have to compute N gradients for each update

Solution: Stochastic Gradient Descent

In Batch Gradient Descent with N samples (data points),

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} L_i(w) = E(L_i(w))$$

So, if we calculate loss for each sample and take the expected value, it will be equal to the loss for the batch

Solution: Stochastic Gradient Descent

In Batch Gradient Descent with N samples (data points),

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} L_i(w) = E(L_i(w))$$

So, if we calculate loss for each sample and take the expected value, it will be equal to the loss for the batch

So, suppose, I calculate loss for one sample, say $L_j(w)$ and do the parameter update

Do it again for another sample, and continue doing it for individual samples, I am expected to make the same impact on the parameters

However, each update would require only one gradient computation (less time, less storage for gradient values)

Stochastic Gradient Descent

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Randomly shuffle samples in the training dataset**
   ii. **For training sample $i = 1, 2, 3, \dots, N$**
      a. **Compute gradient $\frac{\partial L_i(w)}{\partial w}$**
      b. **Update parameters $w \leftarrow w - \eta \frac{\partial L_i(w)}{\partial w}$**
3. **Return parameters**

Epoch

Challenges in Stochastic and Batch Gradient Descent

- BGD: not data efficient when data is very similar
- BGD: High computational time for each step

- SGD: not computationally efficient
- SGD: noisy results since at a time, one data point is considered

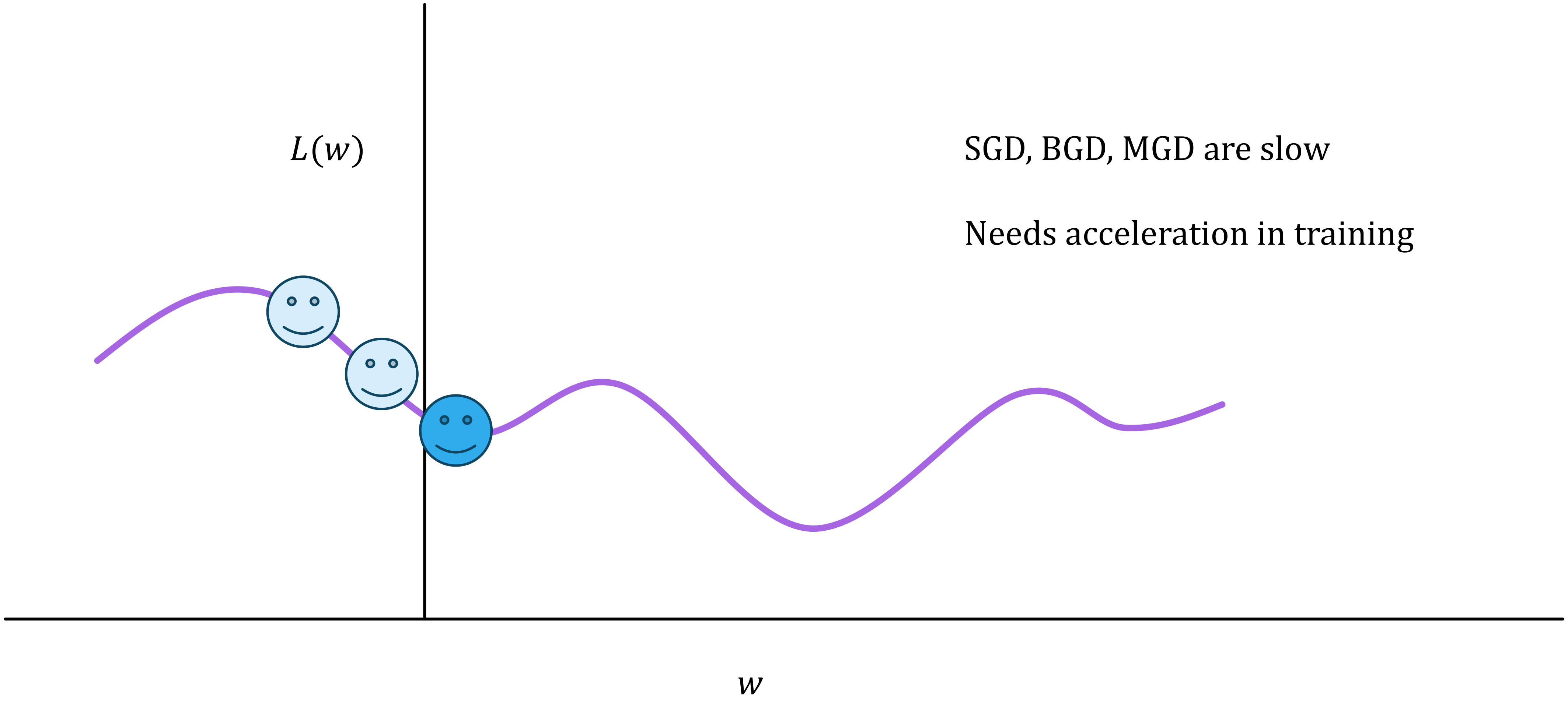
- Solution: minibatch GD

Minibatch Gradient Descent

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Randomly shuffle samples in the training dataset**
   ii. **From training dataset, create b minibatches of size m**
   iii. **For each minibatches $i = 1, 2, 3, \ldots, b$**
      a. **Compute gradient $\frac{\partial L_i(w)}{\partial w}$**
      b. **Update parameters $w \leftarrow w - \eta \frac{\partial L_i(w)}{\partial w}$**
3. **Return parameters**

Epoch

Need of Momentum



SGD, BGD, MGD are slow

Needs acceleration in training

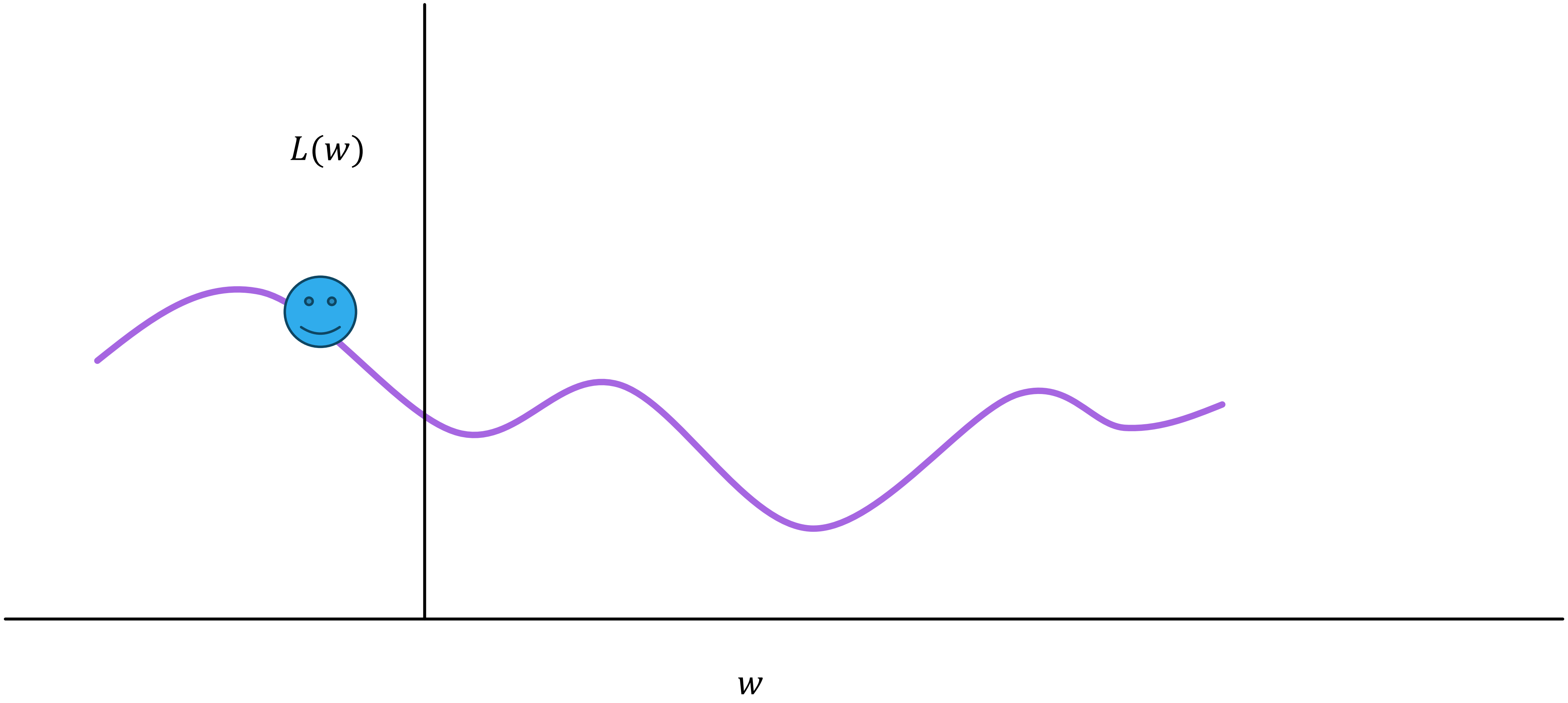
Momentum

Momentum $= mass \times velocity = mv$

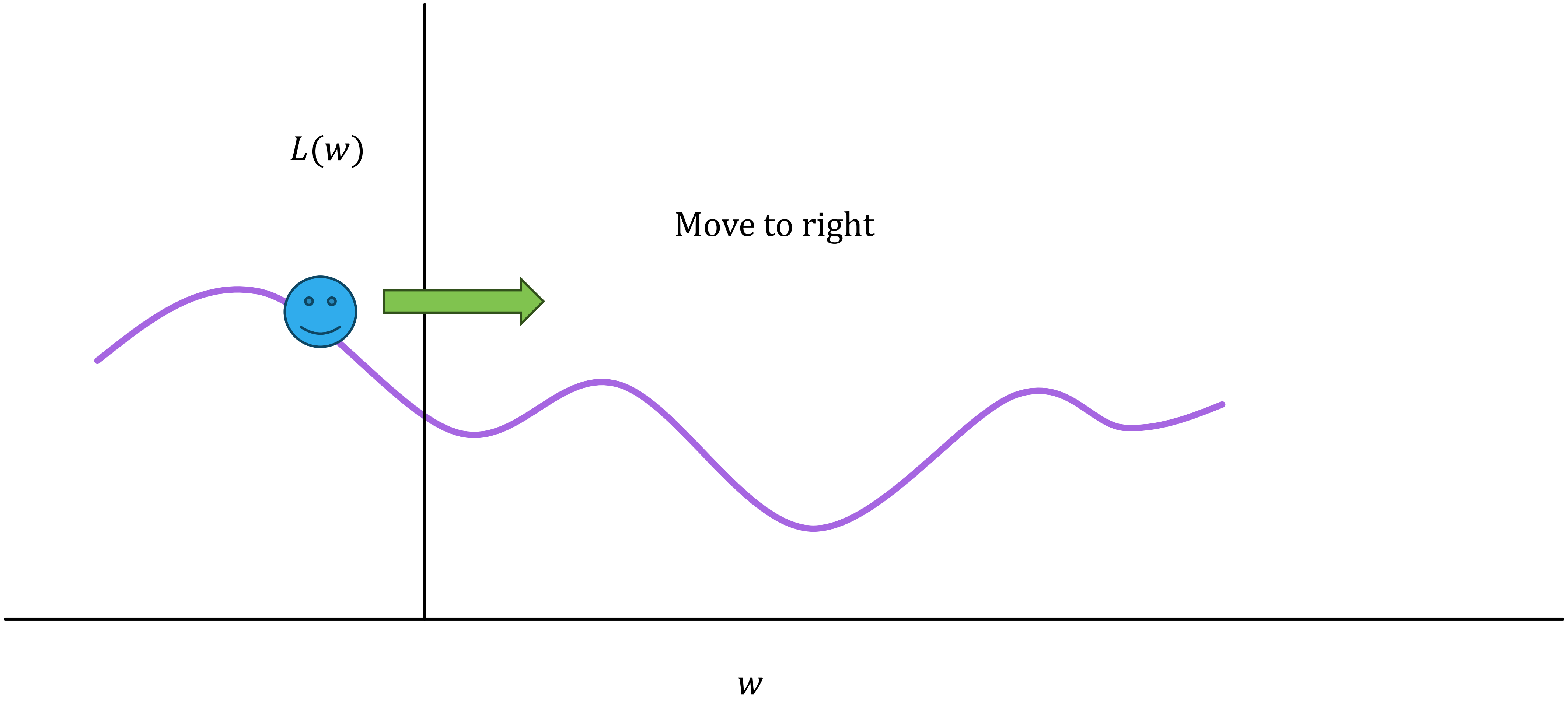
For unit mass, momentum$= v$

Information from previous gradients may help in deciding the path to the minima

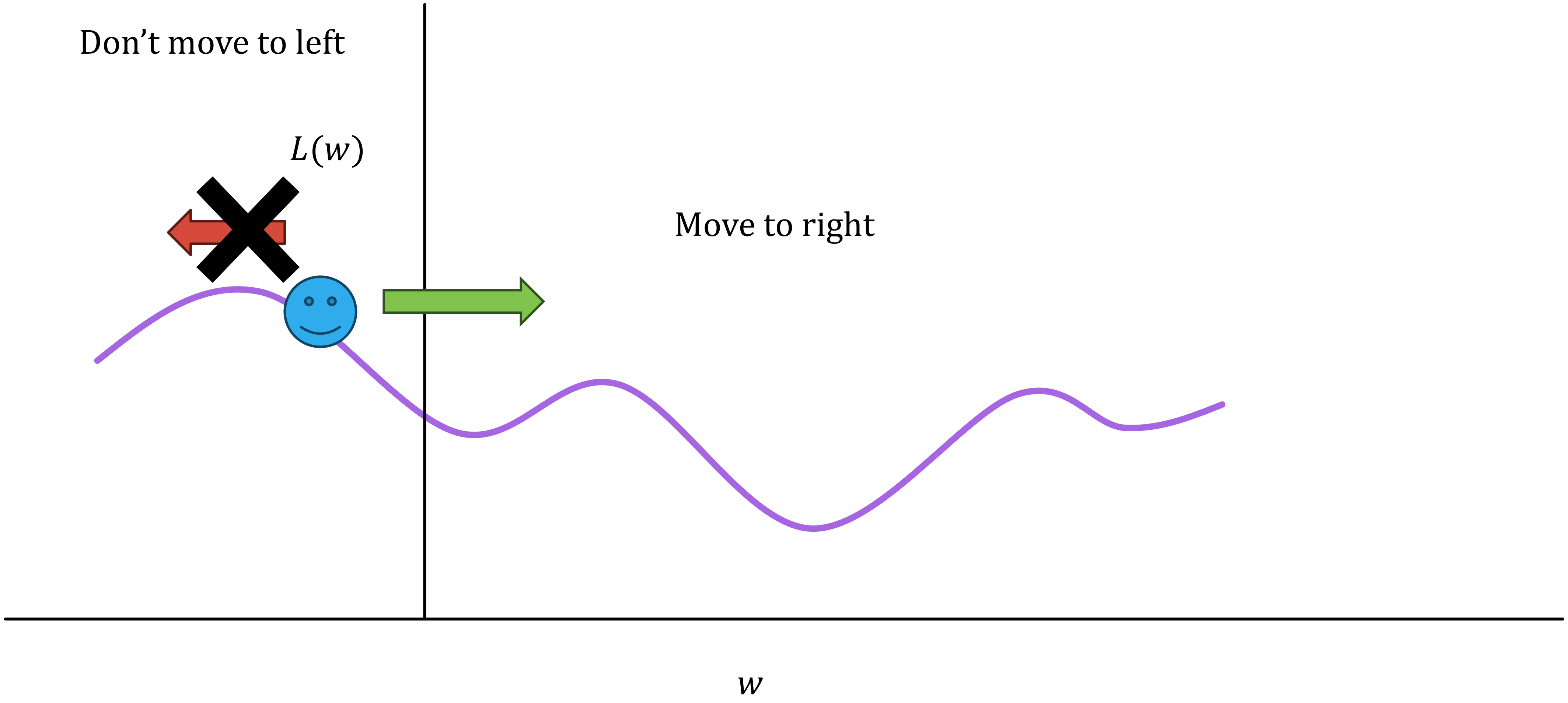
Momentum



Momentum



Momentum



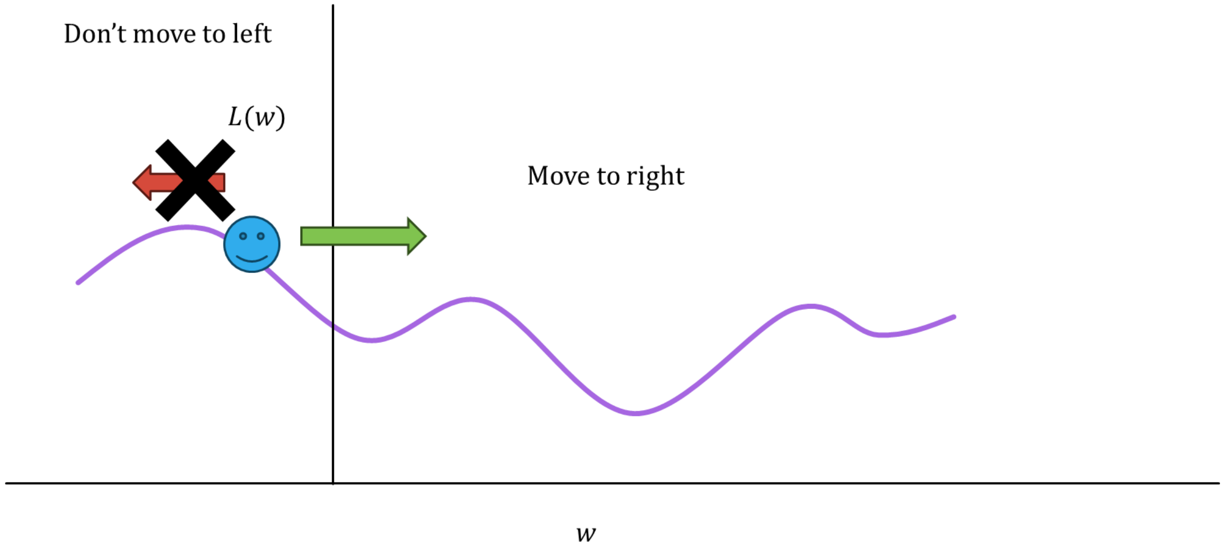
Momentum

Momentum $= mass \times velocity = mv$

For unit mass, momentum$= v$

Information from previous gradients may help in deciding the path to the minima

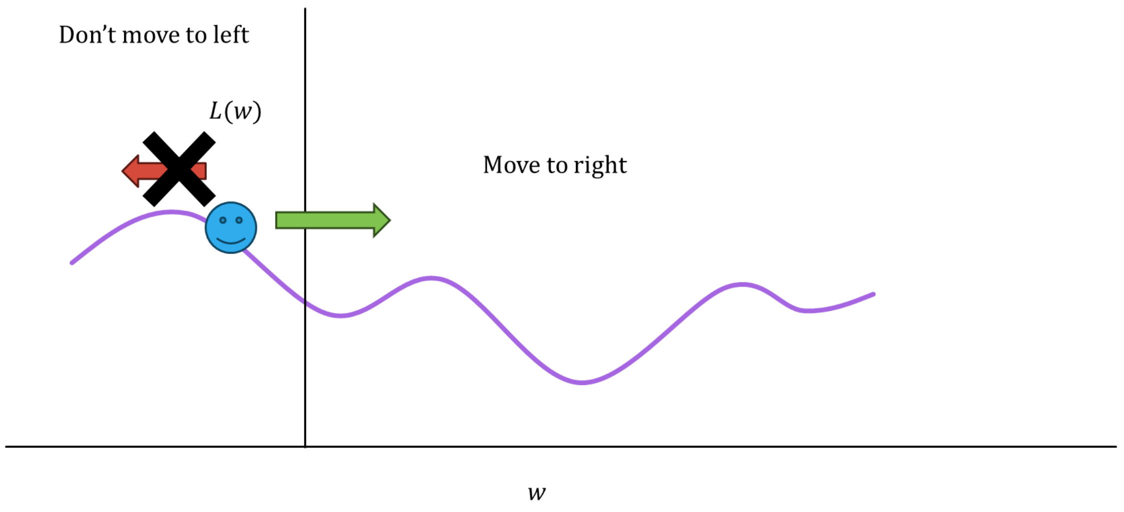
Use past gradients



Momentum

Use past gradients

$$\Delta w_{(t-1)} \leftarrow -\eta \frac{\partial L\left(w_{(t-1)}\right)}{\partial w_{(t-1)}} + \mu \Delta w_{(t-2)}$$

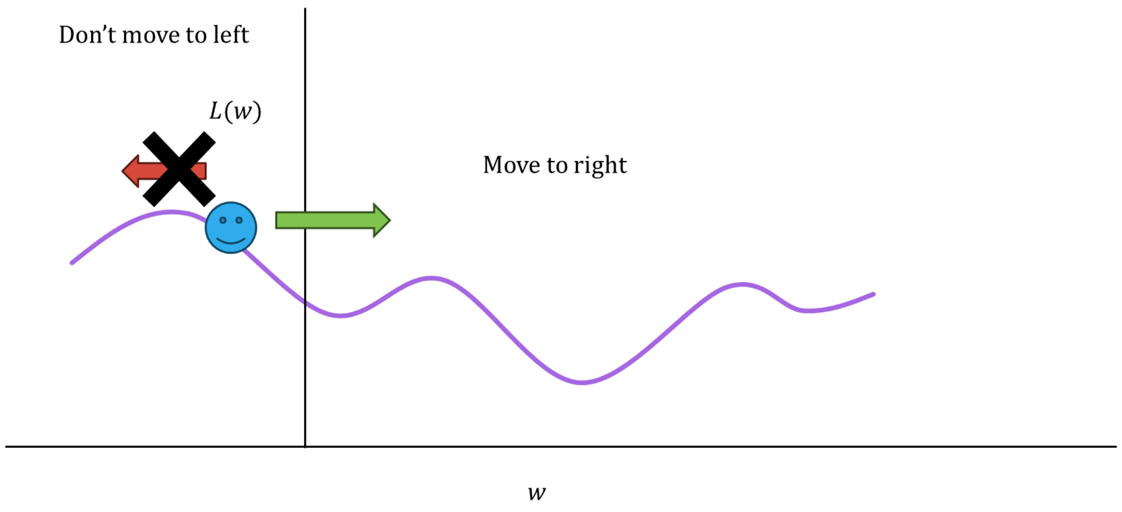


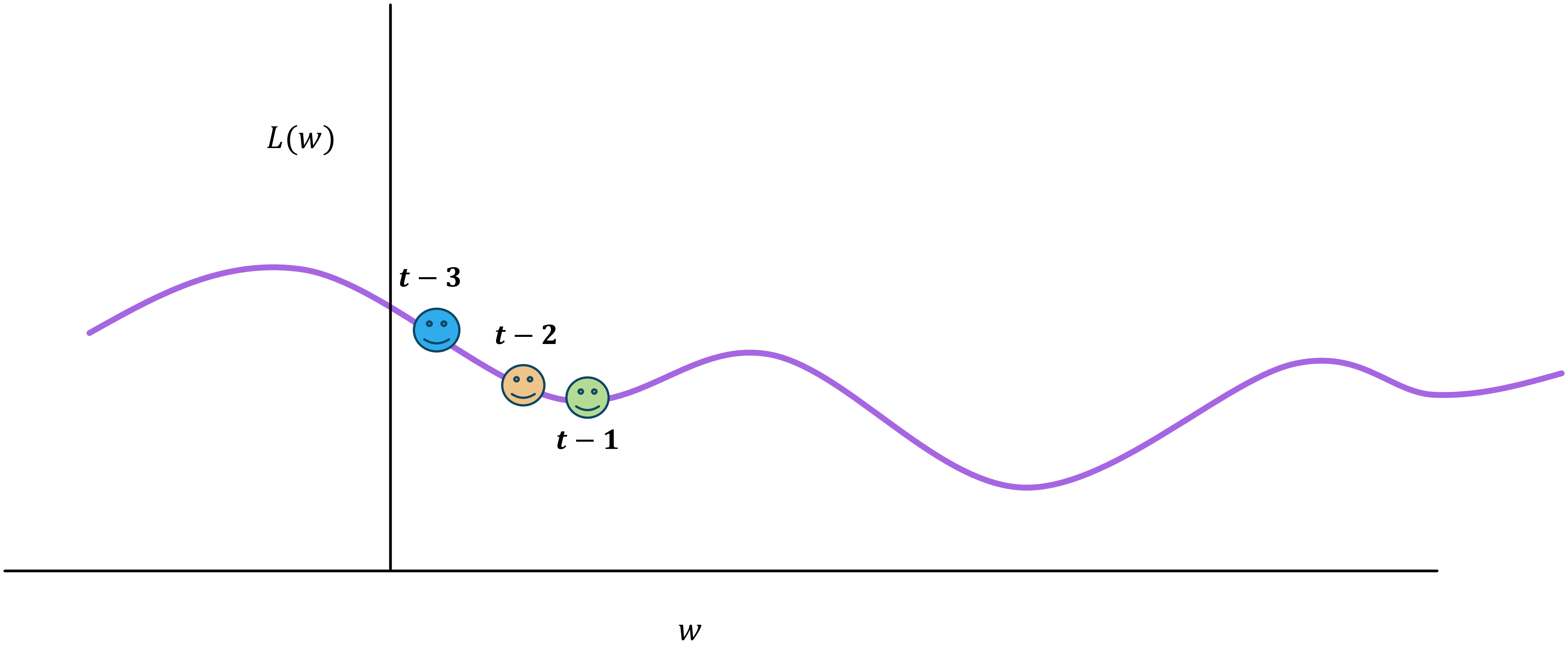
Momentum

Use past gradients

$$\Delta w_{(t-1)} \leftarrow -\eta \frac{\partial L\left(w_{(t-1)}\right)}{\partial w_{(t-1)}} + \mu \Delta w_{(t-2)}$$

$$w_t \leftarrow w_{(t-1)} + \Delta w_{(t-1)}$$



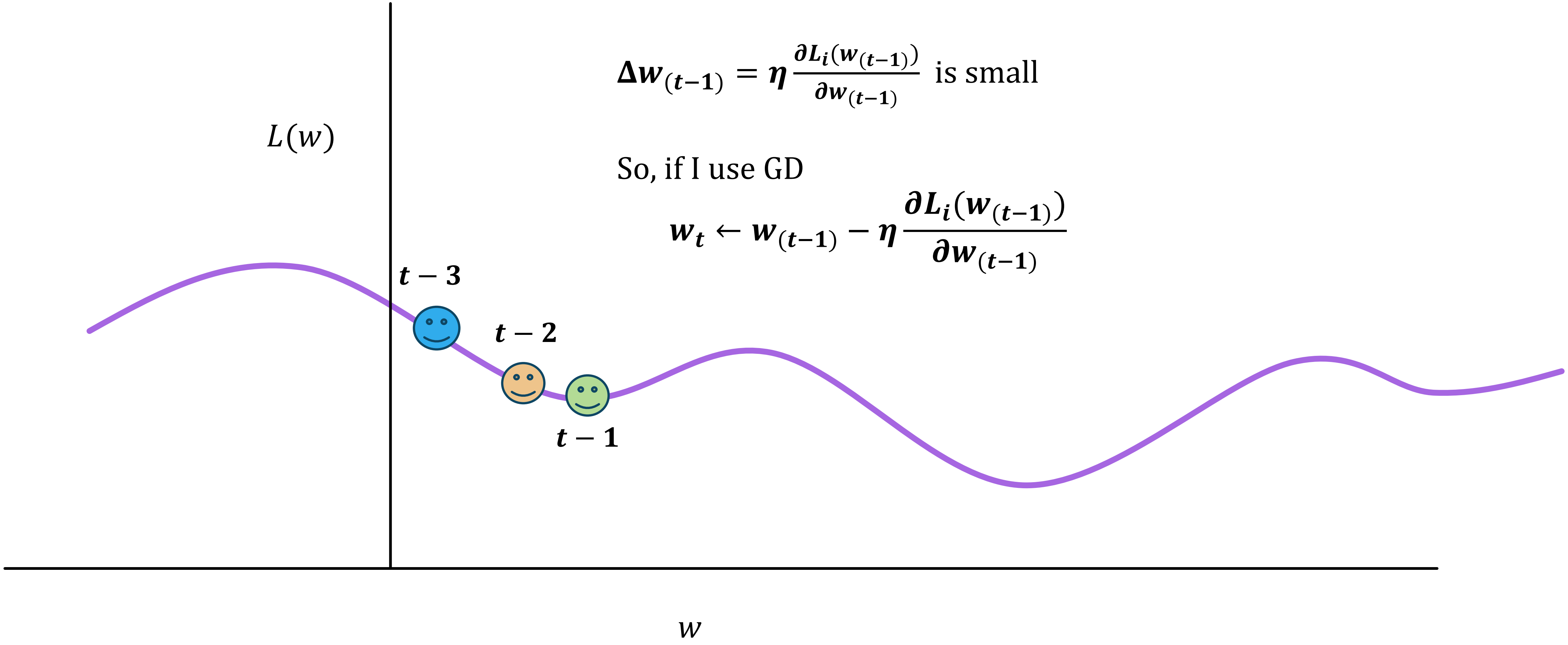


$L(w)$
$t-3$
$t-2$
$t-1$
w

$$\boldsymbol{\Delta w_{(t-1)} = \eta \frac{\partial L_i(w_{(t-1)})}{\partial w_{(t-1)}}}$$ is small

So, if I use GD

$$\boldsymbol{w_t \leftarrow w_{(t-1)} - \eta \frac{\partial L_i(w_{(t-1)})}{\partial w_{(t-1)}}}$$

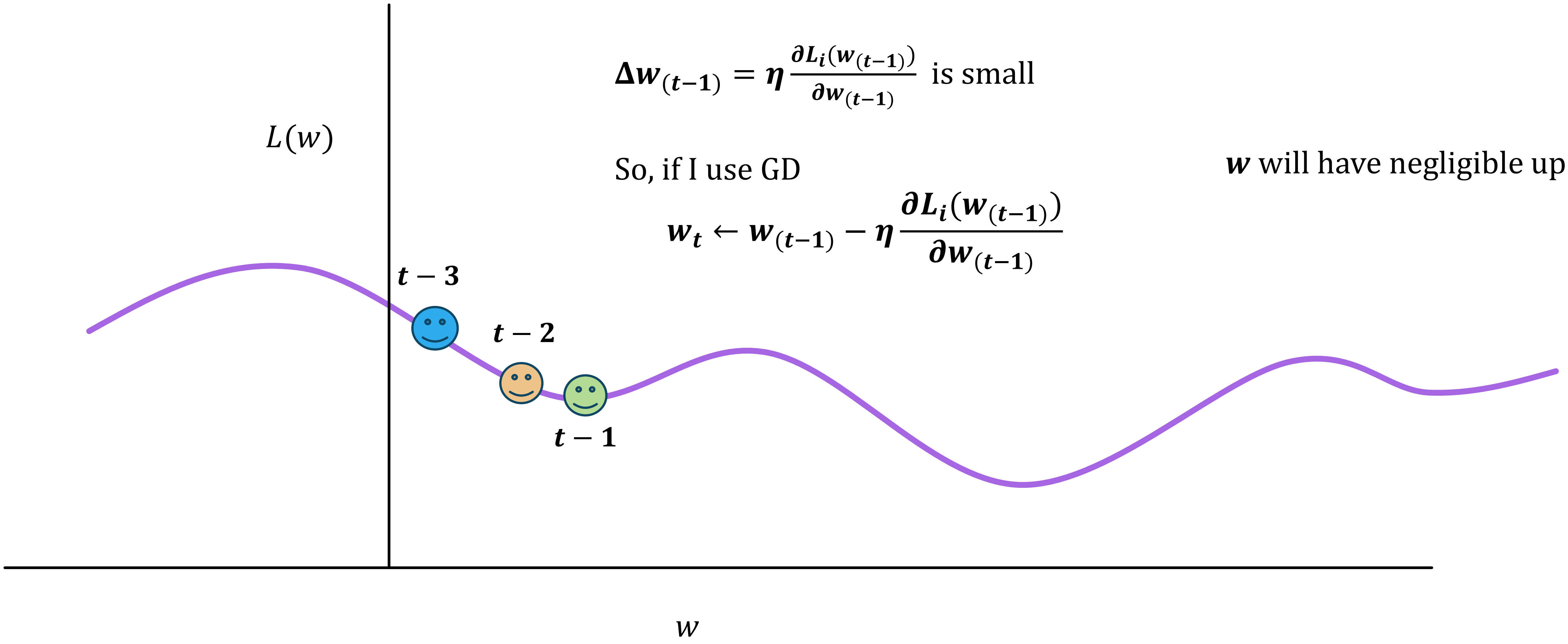


$$\boldsymbol{\Delta w_{(t-1)}} = \boldsymbol{\eta} \frac{\boldsymbol{\partial L_i(w_{(t-1)})}}{\boldsymbol{\partial w_{(t-1)}}} \text{ is small}$$

So, if I use GD

$$\boldsymbol{w_t} \leftarrow \boldsymbol{w_{(t-1)}} - \boldsymbol{\eta} \frac{\boldsymbol{\partial L_i(w_{(t-1)})}}{\boldsymbol{\partial w_{(t-1)}}}$$

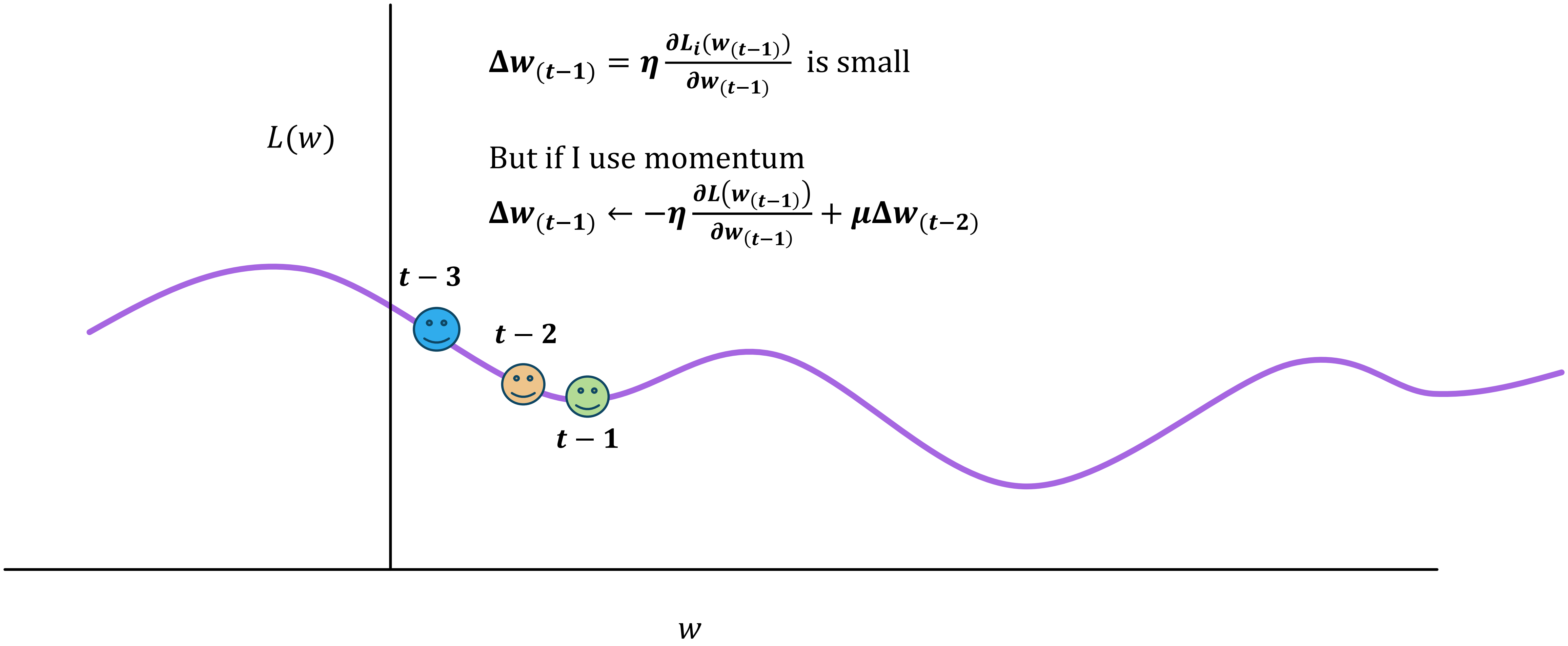
$\boldsymbol{w}$ will have negligible update



$$\boldsymbol{\Delta w_{(t-1)} = \eta \frac{\partial L_i(w_{(t-1)})}{\partial w_{(t-1)}}} \text{ is small}$$

But if I use momentum

$$\boldsymbol{\Delta w_{(t-1)} \leftarrow -\eta \frac{\partial L(w_{(t-1)})}{\partial w_{(t-1)}} + \mu \Delta w_{(t-2)}}$$



$$\boldsymbol{\Delta w_{(t-1)} = \eta \frac{\partial L_i(w_{(t-1)})}{\partial w_{(t-1)}}}$$ is small

But if I use momentum

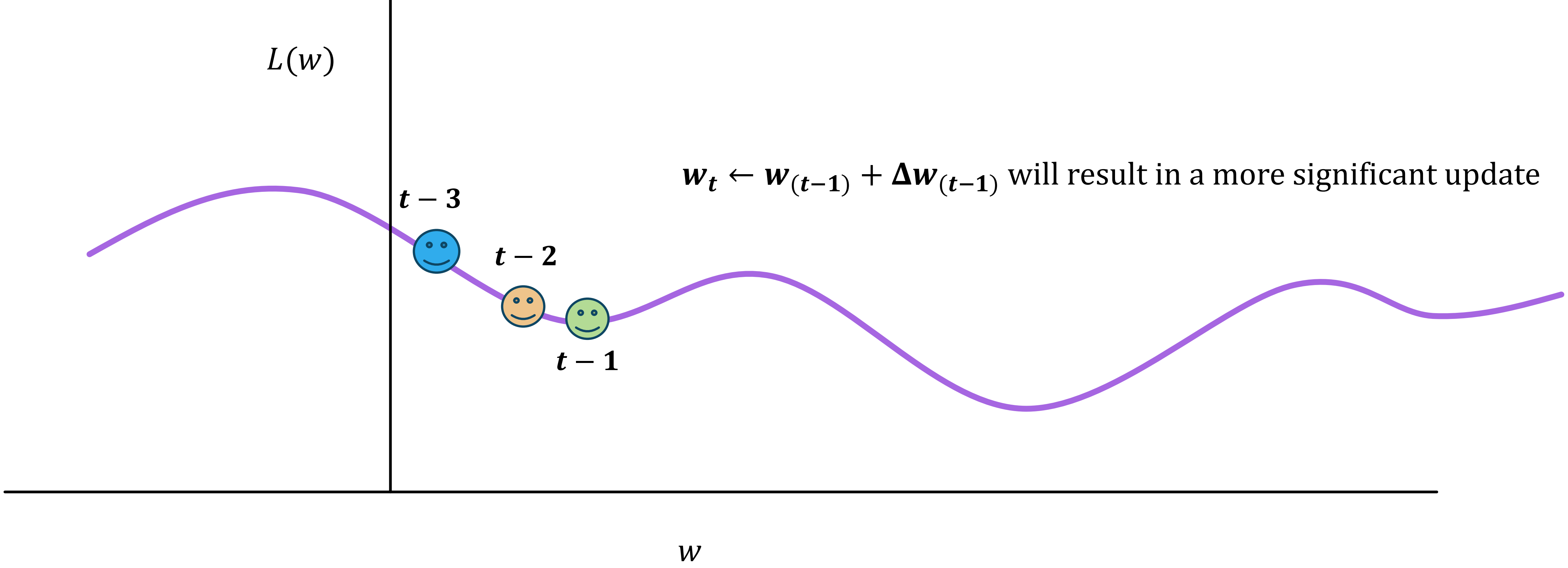
$$\boldsymbol{\Delta w_{(t-1)} \leftarrow -\eta \frac{\partial L(w_{(t-1)})}{\partial w_{(t-1)}} + \mu \Delta w_{(t-2)}}$$

Although $\boldsymbol{\eta \frac{\partial L_i(w_{(t-1)})}{\partial w_{(t-1)}}}$ is small

$\boldsymbol{\Delta w_{(t-2)}}$ is larger

Hence, $\boldsymbol{\Delta w_{(t-1)}}$ will not be negligible

$\boldsymbol{w_t \leftarrow w_{(t-1)} + \Delta w_{(t-1)}}$ will result in a more significant update



Momentum

$$\Delta w \leftarrow -\eta \frac{\partial L(w)}{\partial w} + \mu \Delta w$$

$$w \leftarrow w + \Delta w$$

Equivalently

$$v \leftarrow \alpha v - \eta \frac{\partial L(w)}{\partial w}$$

$$w \leftarrow w + v$$

v **is sometimes called velocity**

Minibatch Gradient Descent with Momentum

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Randomly shuffle samples in the training dataset**
   ii. **From training dataset, create b minibatches of size m**
   iii. **For each minibatches $i = 1, 2, 3, \ldots, b$**
      a. **Compute gradient $\frac{\partial L_i(w)}{\partial w}$**
      b. **Compute $\Delta w \leftarrow -\eta \frac{\partial L_i(w)}{\partial w} + \mu \Delta w$**
      c. **Update parameters $w \leftarrow w + \Delta w$**
3. **Return parameters**

Nesterov Momentum

$$\boldsymbol{\Delta w_{(t-1)} \leftarrow -\eta \frac{\partial L\left(w_{(t-1)} + \mu \Delta w_{(t-2)}\right)}{\partial w_{(t-1)}} + \mu \Delta w_{(t-2)}}$$

$$\boldsymbol{w_t \leftarrow w_{(t-1)} + \Delta w_{(t-1)}}$$

Adaptive Learning Rate

- Time dependent learning rate

$\eta(t) = \eta_i$ if $t_i \leq t \leq t_{i+1}$	piecewise constant
$\eta(t) = \eta_0 \cdot e^{-\lambda t}$	exponential decay
$\eta(t) = \eta_0 \cdot (\beta t + 1)^{-\alpha}$	polynomial decay

AdaGrad

- Individually adapts the learning rates of all model parameters
  - By scaling them inversely proportional to the square root of the sum of all of their historical squared values
- The parameters with the largest partial derivative of the loss have a correspondingly rapid decrease in their learning rate
- Parameters with small partial derivatives have a relatively small decrease in their learning rate
- Greater progress in the more gently sloped directions of parameter space
- Converges rapidly for convex loss function

AdaGrad

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Randomly shuffle samples in the training dataset**
   ii. **From training dataset, create b minibatches of size m**
   iii. **For each minibatches $i = 1, 2, 3, \dots, b$**
      a. **Compute gradient** $g \leftarrow \frac{\partial L_i(w)}{\partial w}$
      b. **Accumulate squared gradient** $r \leftarrow r + g \odot g$
      c. **Compute Update** $\Delta w \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$
      d. **Update parameters** $w \leftarrow w + \Delta w$
3. **Return parameters**

AdaGrad

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Randomly shuffle samples in the training dataset**
   ii. **From training dataset, create b minibatches of size m**
   iii. **For each minibatches $i = 1, 2, 3, \dots, b$**
      a. **Compute gradient** $g \leftarrow \frac{\partial L_i(w)}{\partial w}$

      Hadamard Product (term by term)

      b. **Accumulate squared gradient** $r \leftarrow r + g \odot g$
      c. **Compute Update** $\Delta w \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$

      (division and square root are applied element-wise)

      a. **Update parameters** $w \leftarrow w + \Delta w$
3. **Return parameters**

AdaGrad: Details of Step 2.iii

Let $r = [r_1 \;\; r_2 \;\; \dots r_n]$, $\;\; g = [g_1 \;\; g_2 \;\; \dots g_n]$, $\;\; \Delta w = [\Delta w_1 \;\; \Delta w_2 \;\; \dots \Delta w_n]$

a. **Accumulate squared gradient** $r \leftarrow r + g \odot g$

Hadamard Product (term by term)

$$[r_1 \;\; r_2 \;\; \dots r_n] \leftarrow [r_1 \;\; r_2 \;\; \dots r_n] + [g_1^2 \;\; g_2^2 \;\; \dots g_n^2]$$

b. **Compute Update** $\Delta w \leftarrow -\frac{\epsilon}{\delta+\sqrt{r}} \odot g$

(division and square root are applied element-wise)

$$\Delta w_1 \leftarrow -\frac{\epsilon}{\delta+\sqrt{r_1}} g_1$$

$$\Delta w_2 \leftarrow -\frac{\epsilon}{\delta+\sqrt{r_2}} g_2$$

$$\dots$$

$$\dots$$

$$\Delta w_n \leftarrow \frac{\epsilon}{\delta+\sqrt{r_n}} g_n$$

c. **Update parameters** $[w_1 \;\; w_2 \;\; \dots w_n] \leftarrow [w_1 \;\; w_2 \;\; \dots w_n] + [\Delta w_1 \;\; \Delta w_2 \;\; \dots \Delta w_n]$

AdaGrad

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Randomly shuffle samples in the training dataset**
   ii. **From training dataset, create b minibatches of size m**
   iii. **For each minibatches $i = 1, 2, 3, \ldots, b$**
      a. **Compute gradient** $g \leftarrow \frac{\partial L_i(w)}{\partial w}$
      b. **Accumulate squared gradient** $r \leftarrow r + g \odot g$
      c. **Compute Update** $\Delta w \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$ Learning rate
      d. **Update parameters** $w \leftarrow w + \Delta w$
3. **Return parameters**

RMSProp

- Based on the entire history of gradient, AdaGrad may make the learning rate too small before it arrives to a convex segment of the loss

- RMSProp discards the gradient from distant past

- It finds a convex segment with a large learning rate

RMSProp

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Randomly shuffle samples in the training dataset**
   ii. **From training dataset, create $\boldsymbol{b}$ minibatches of size $\boldsymbol{m}$**
   iii. **For each minibatches $\boldsymbol{i = 1, 2, 3, \dots, b}$**
      a. **Compute gradient** $\boldsymbol{g \leftarrow \frac{\partial L_i(w)}{\partial w}}$
      b. **Accumulate squared gradient** $\boldsymbol{r \leftarrow \rho r + (1-\rho) g \odot g}$
      c. **Compute Update** $\boldsymbol{\Delta w \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g}$
      d. **Update parameters** $\boldsymbol{w \leftarrow w + \Delta w}$
3. **Return parameters**

RMSProp

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Randomly shuffle samples in the training dataset**
   ii. **From training dataset, create b minibatches of size m**
   iii. **For each minibatches $i = 1, 2, 3, \dots, b$**
      a. **Compute gradient** $g \leftarrow \frac{\partial L_i(w)}{\partial w}$
      b. **Accumulate squared gradient** $r \leftarrow \rho r + (1-\rho) g \odot g$
      c. **Compute Update** $\Delta w \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$
      d. **Update parameters** $w \leftarrow w + \Delta w$
3. **Return parameters**

At epoch t

$$r_t = (1-\rho) g_t^2 + \rho r_{t-1}$$

RMSProp

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Randomly shuffle samples in the training dataset**
   ii. **From training dataset, create b minibatches of size m**
   iii. **For each minibatches $i = 1, 2, 3, \ldots, b$**
      a. **Compute gradient** $g \leftarrow \frac{\partial L_i(w)}{\partial w}$
      b. **Accumulate squared gradient** $r \leftarrow \rho r + (1-\rho) g \odot g$
      c. **Compute Update** $\Delta w \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$
      d. **Update parameters** $w \leftarrow w + \Delta w$
3. **Return parameters**

At epoch t

$$r_t = (1-\rho) g_t^2 + \rho r_{t-1}$$

$$= (1-\rho) g_t^2 + \rho (1-\rho) g_{t-1}^2 + \rho^2 r_{t-2}$$

$$= (1-\rho)(g_t^2 + \rho g_{t-1}^2 + \rho^2 g_{t-2}^2 + \rho^3 g_{t-3}^2 + \ldots)$$

RMSProp

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Randomly shuffle samples in the training dataset**
   ii. **From training dataset, create b minibatches of size m**
   iii. **For each minibatches $i = 1, 2, 3, \dots, b$**
      a. **Compute gradient** $g \leftarrow \frac{\partial L_i(w)}{\partial w}$
      b. **Accumulate squared gradient** $r \leftarrow \rho r + (1-\rho) g \odot g$
      c. **Compute Update** $\Delta w \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$
      d. **Update parameters** $w \leftarrow w + \Delta w$
3. **Return parameters**

Gradients from distant past matters less compared to gradients from recent past

At epoch t

$$r_t = (1-\rho) g_t^2 + \rho r_{t-1}$$

$$= (1-\rho) g_t^2 + \rho(1-\rho) g_{t-1}^2 + \rho^2 r_{t-2}$$

$$= (1-\rho)(g_t^2 + \rho g_{t-1}^2 + \rho^2 g_{t-2}^2 + \rho^3 g_{t-3}^2 + \dots)$$

RMSProp with Nesterov Momentum

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Randomly shuffle samples in the training dataset**
   ii. **From training dataset, create b minibatches of size m**
   iii. **For each minibatches $i = 1, 2, 3, \dots, b$**
      a. **Compute gradient** $g \leftarrow \frac{\partial L(w+\mu\Delta w)}{\partial w}$
      b. **Accumulate squared gradient** $r \leftarrow \rho r + (1-\rho) g \odot g$
      c. **Compute Velocity Update** $v \leftarrow \mu\Delta w - \frac{\epsilon}{\sqrt{r}} \odot g$
      d. **Update parameters** $w \leftarrow w + v$
3. **Return parameters**

Adam: Adaptive Moments

- Momentum is incorporated directly as an estimate of the first order moment (with exponential weighting) of the gradient

- Bias correction in first and second order moments

Adam

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Randomly shuffle samples in the training dataset**
   ii. **From training dataset, create b minibatches of size m**
   iii. **For each minibatches $i = 1, 2, 3, \dots, b$**
      a. **Compute gradient** $g \leftarrow \frac{\partial L_i(w)}{\partial w}$
      b. **Update biased first moment** $s \leftarrow \rho_1 s + (1 - \rho_1) g$
      c. **Update biased second moment** $r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g$
      d. **Correct bias in first moment** $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$
      e. **Correct bias in second moment** $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$
      f. **Compute update** $\Delta w = -\frac{\hat{s}}{\delta+\sqrt{\hat{r}}}$
      g. **Update parameters** $w \leftarrow w + \Delta w$
3. **Return parameters**

t is the number of epochs

Adam: Possible Variation

1. **Initialize parameters**
2. **Loop until convergence**
   i. **Randomly shuffle samples in the training dataset**
   ii. **From training dataset, create b minibatches of size m**
   iii. **For each minibatches $i = 1, 2, 3, \ldots, b$**
      a. **Compute gradient** $g \leftarrow \frac{\partial L_i(w)}{\partial w}$
      b. **Update biased first moment** $s \leftarrow \rho_1 s + (1 - \rho_1) g$
      c. **Update biased second moment** $r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g$
      d. **Correct bias in first moment** $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$
      e. **Correct bias in second moment** $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$
      f. **Compute update** $\Delta w = -\frac{\eta \hat{s}}{\delta + \sqrt{\hat{r}}}$
      g. **Update parameters** $w \leftarrow w + \Delta w$
3. **Return parameters**

t is the number of epochs

η: learning rate

Why Bias Correction

- We are interested to look at the gradient over the epochs and not just at one particular epoch
  - We are interested in the expected value of the gradient
- However, we are updating an exponential moving average
  - $\boldsymbol{s} \leftarrow \boldsymbol{\rho_1 s} + (\mathbf{1} - \boldsymbol{\rho_1})\boldsymbol{g}$
- We do not have any update equation for gradient
- Under certain assumption, it can be shown that
  - $E(g) = E\left(\frac{s}{1-\rho_1^t}\right) = E(\hat{\boldsymbol{s}})$
- Similarly,
  - $E(\boldsymbol{g} \odot \boldsymbol{g}) = E\left(\frac{r}{1-\rho_2^t}\right) = E(\hat{\boldsymbol{r}})$

Gradient Clipping

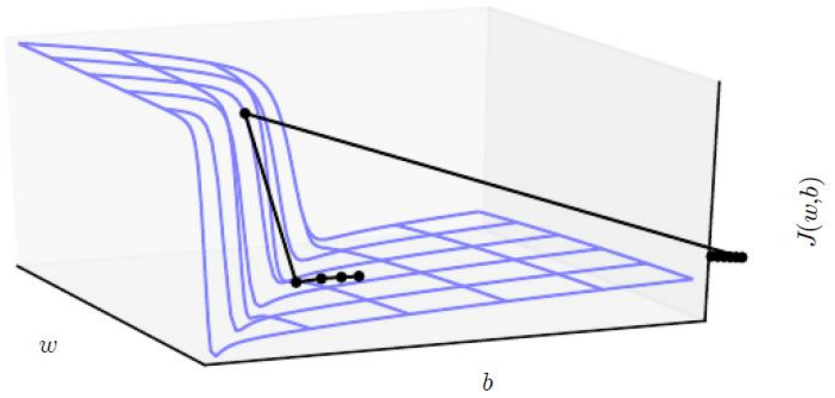
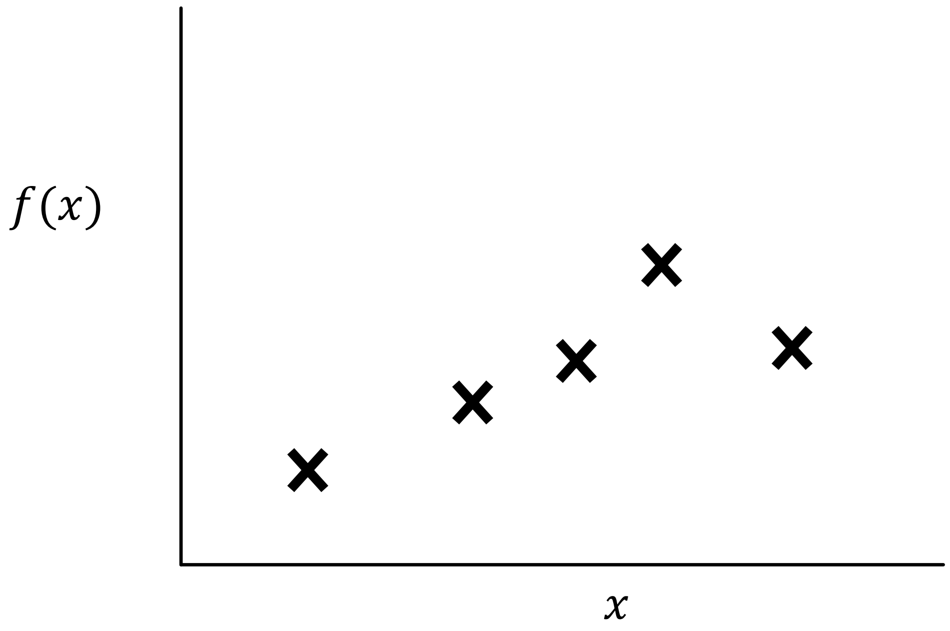


Figure 8.3: The objective function for highly nonlinear deep neural networks or for recurrent neural networks often contains sharp nonlinearities in parameter space resulting from the multiplication of several parameters. These nonlinearities give rise to very high derivatives in some places. When the parameters get close to such a cliff region, a gradient descent update can catapult the parameters very far, possibly losing most of the optimization work that had been done. Figure adapted with permission from Pascanu *et al.* (2013).

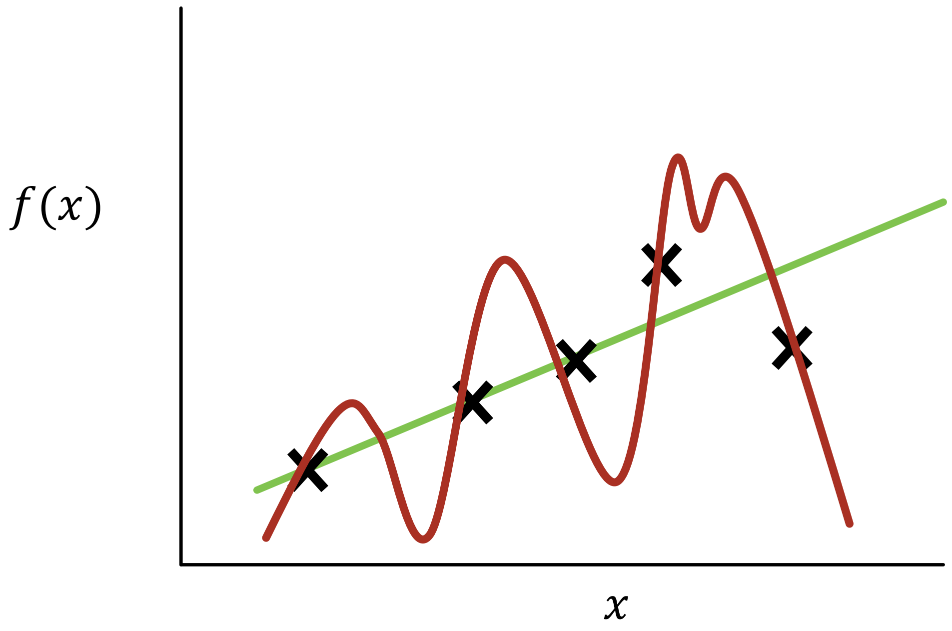
- If GD algorithms try to take a long step, clip (reduce) the step size
- If $|\mathbf{g}| > v$
    - $\mathbf{g} \leftarrow \frac{\mathbf{g}v}{|\mathbf{g}|}$

Some Training Data



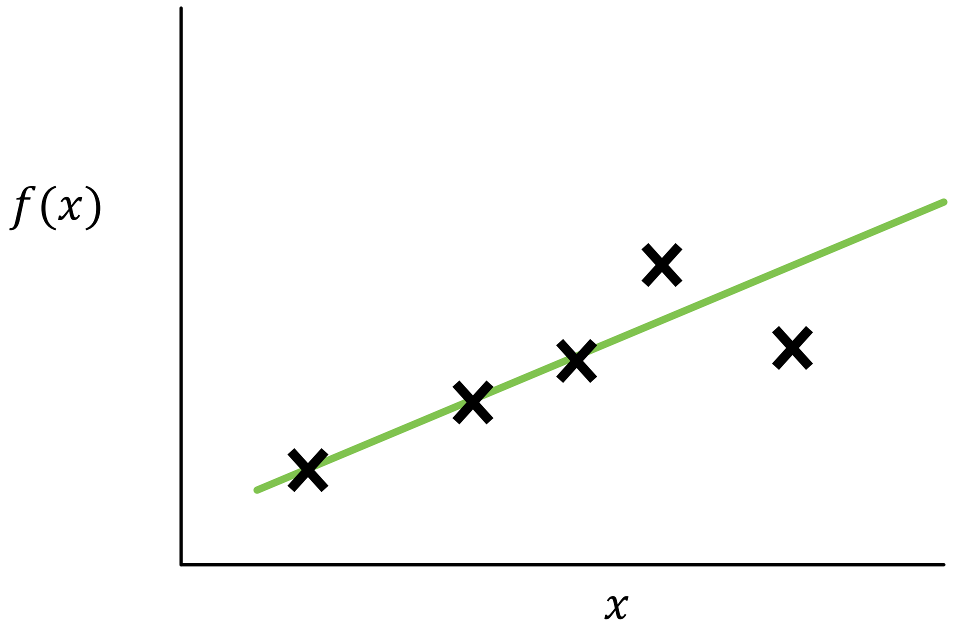
First Set of Training Data

Bias and Variance



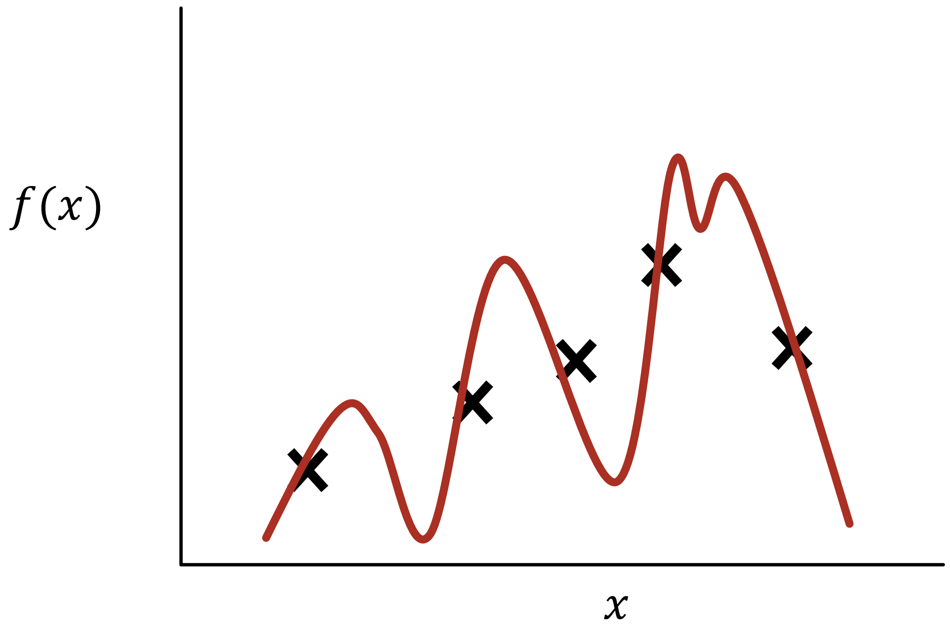
- Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It leads to high error on training and test data.
- Variance is the variability of model's prediction across different sets of data

Bias and Variance



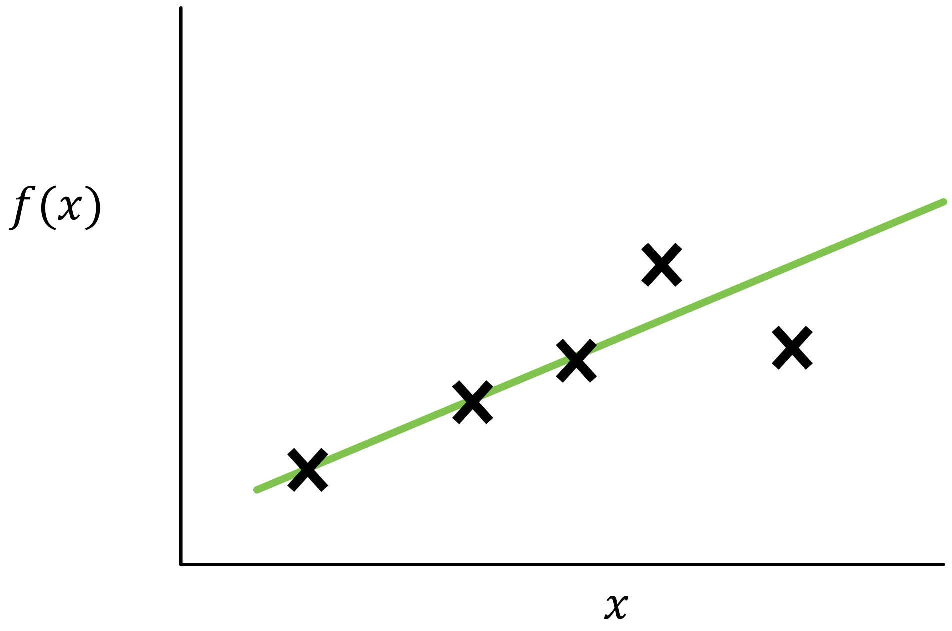
- More assumption: simple model/ simple curve

Bias and Variance

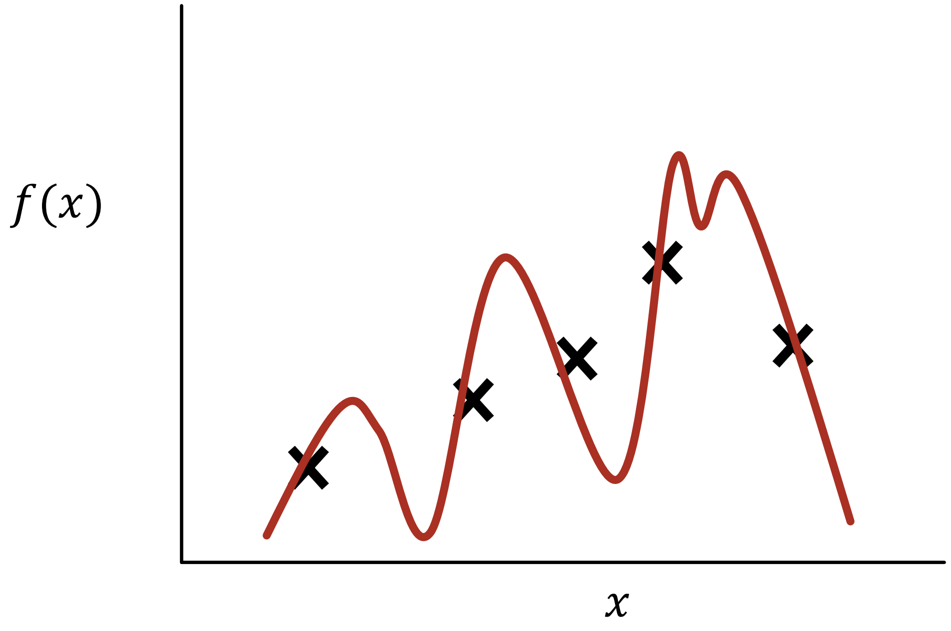


- Less assumption: complex model/ simple curve

Fitting Possible Curves on the Training Data

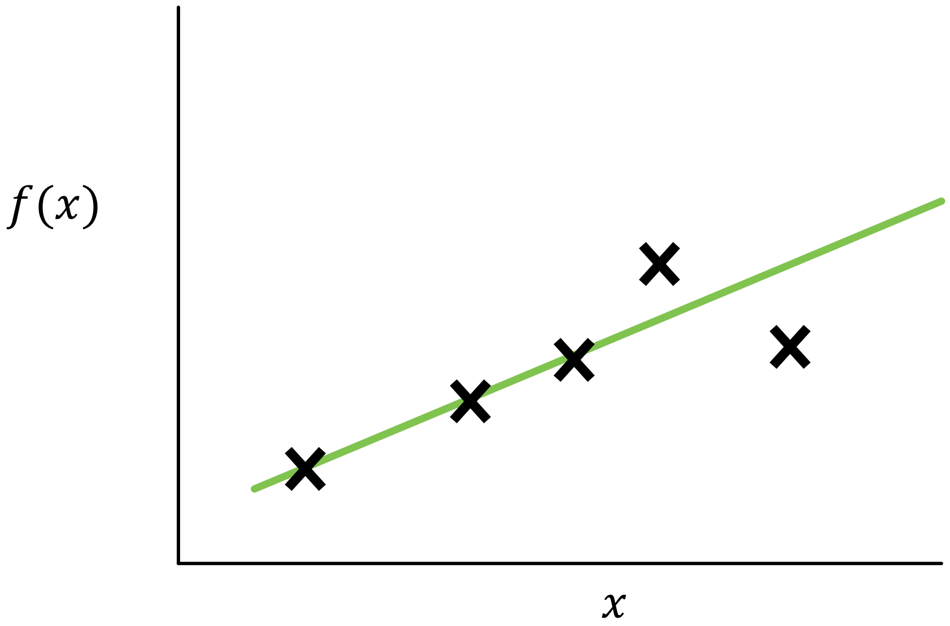


ML Model 1

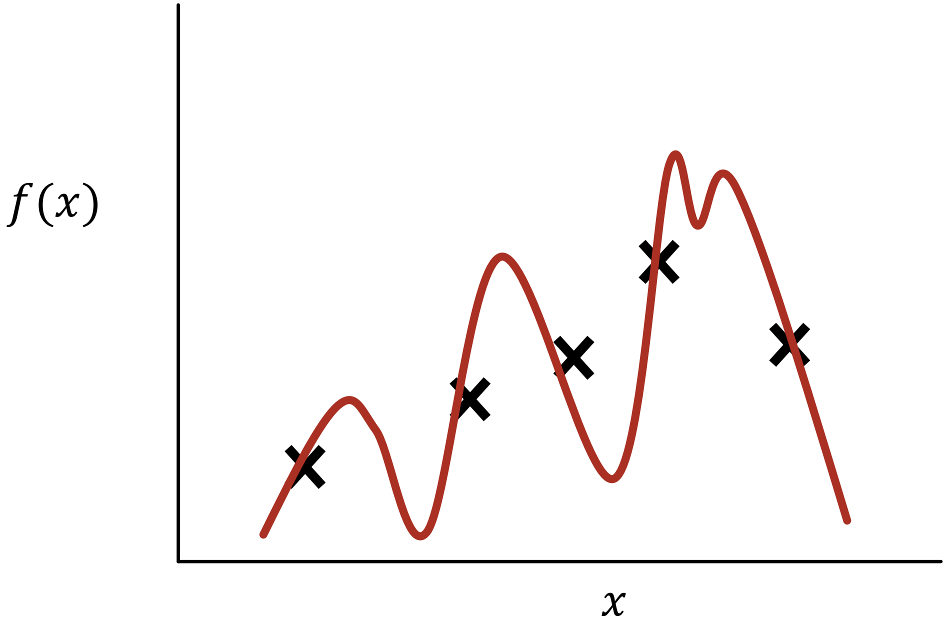


ML Model 2

Fitting Possible Curves on the Training Data



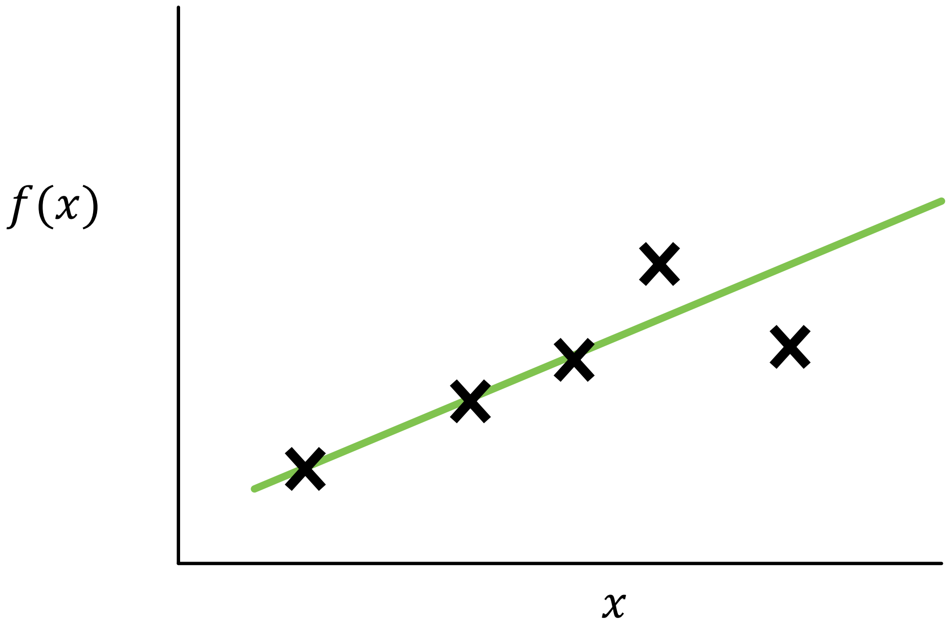
ML Model 1



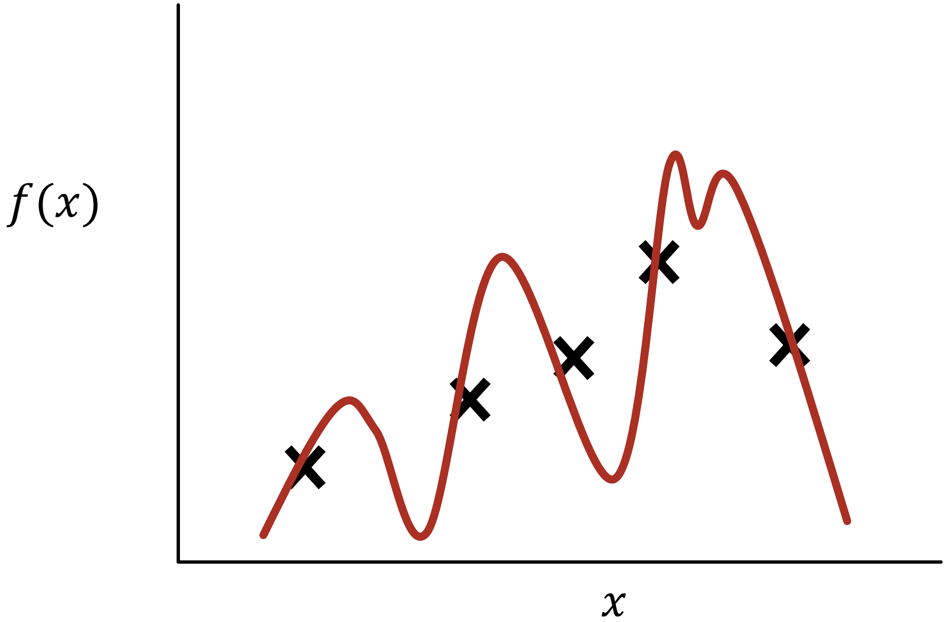
ML Model 2

- More assumption: simple model/ simple curve
  - More bias
- Ranking of bias (high to low)
  - Green curve (model 1)
  - Red curve (model 2)

Fitting Possible Curves on the Training Data



ML Model 1

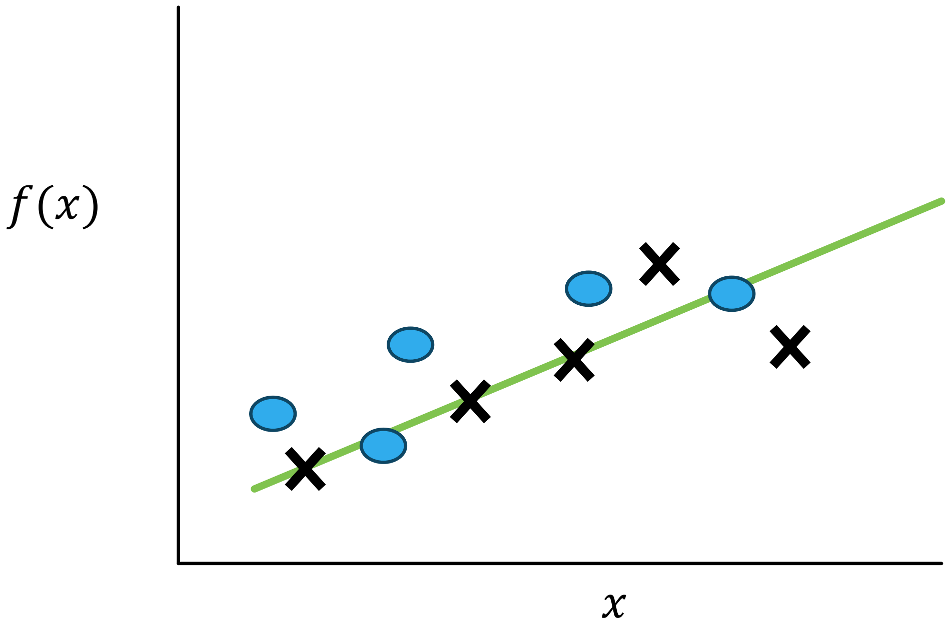


ML Model 2

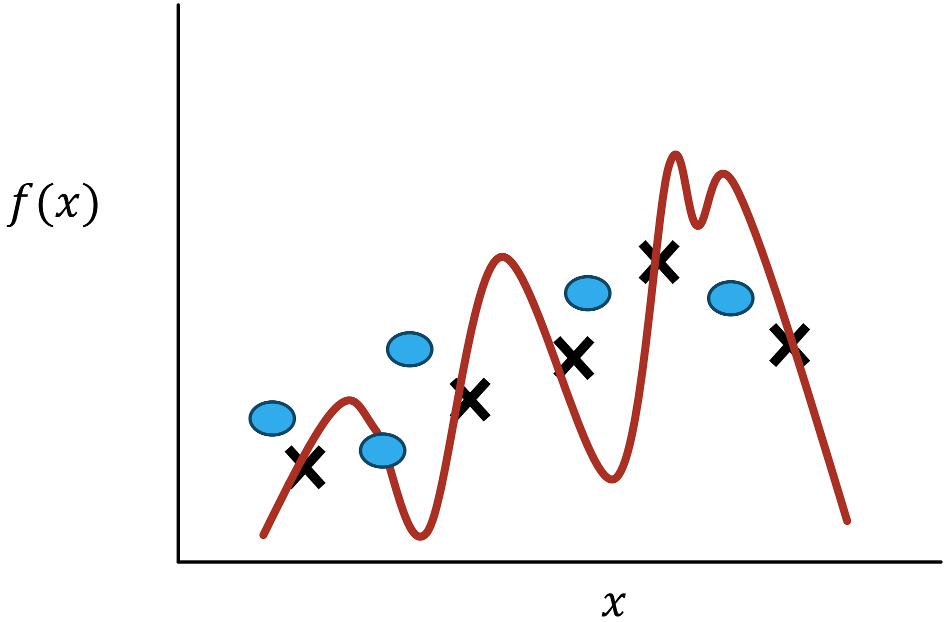
- Ranking of bias (high to low)
  - Green curve (model 1)
  - Red curve (model 2)

- Which curve has more error when compared to training examples?
- Ranking of training error (high to low)
  - Green curve
  - Red curve

Now, Let’s See How They Perform on Test Data 1



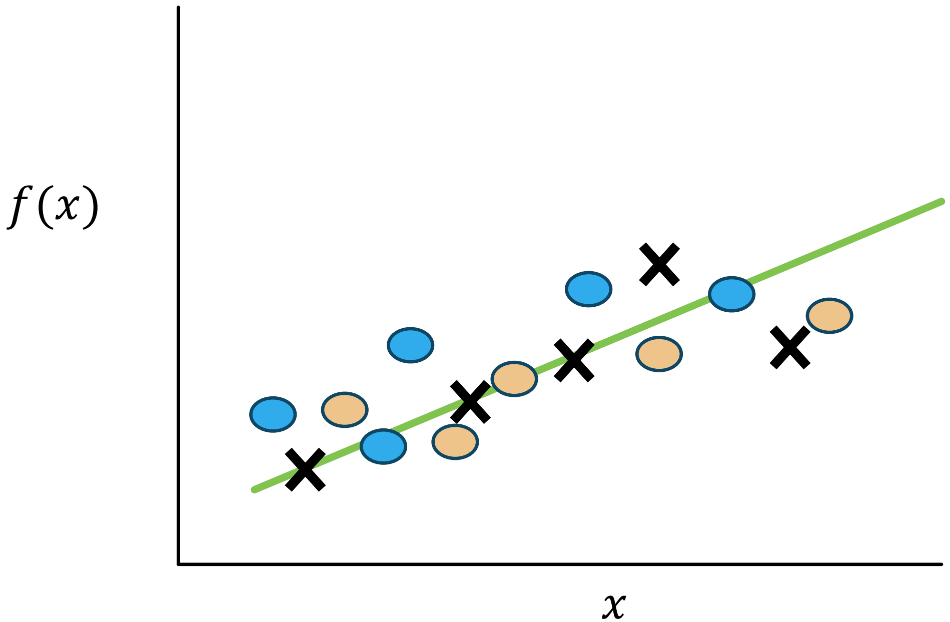
ML Model 1



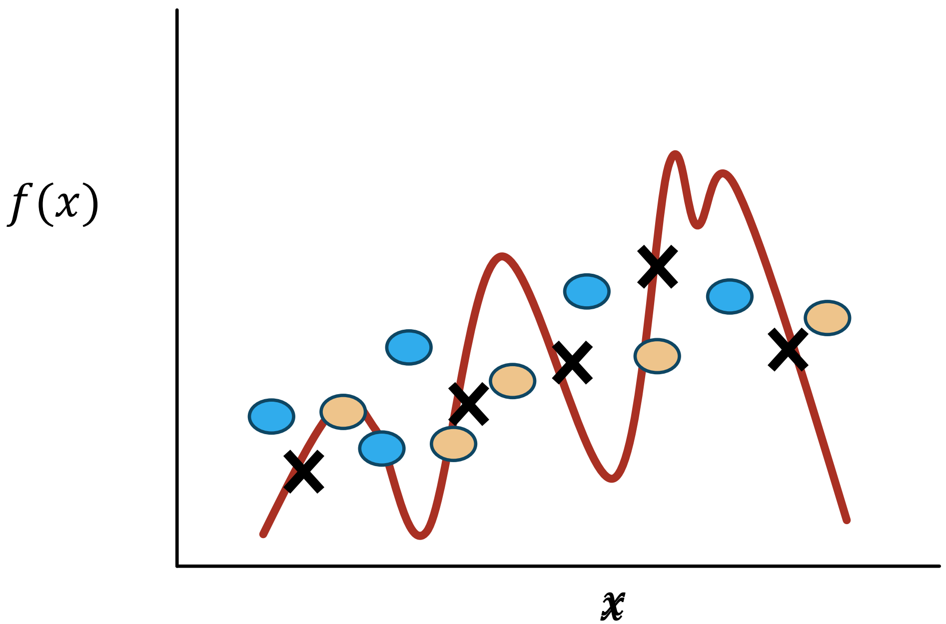
ML Model 2

- Model 1
  - Similar to its performance on training data
  - Test error $e_g(1)$
- Model 2
  - Very different from its performance on training data
  - Test error $e_r(1)$

Now, Let's See How They Perform on Test Data 2



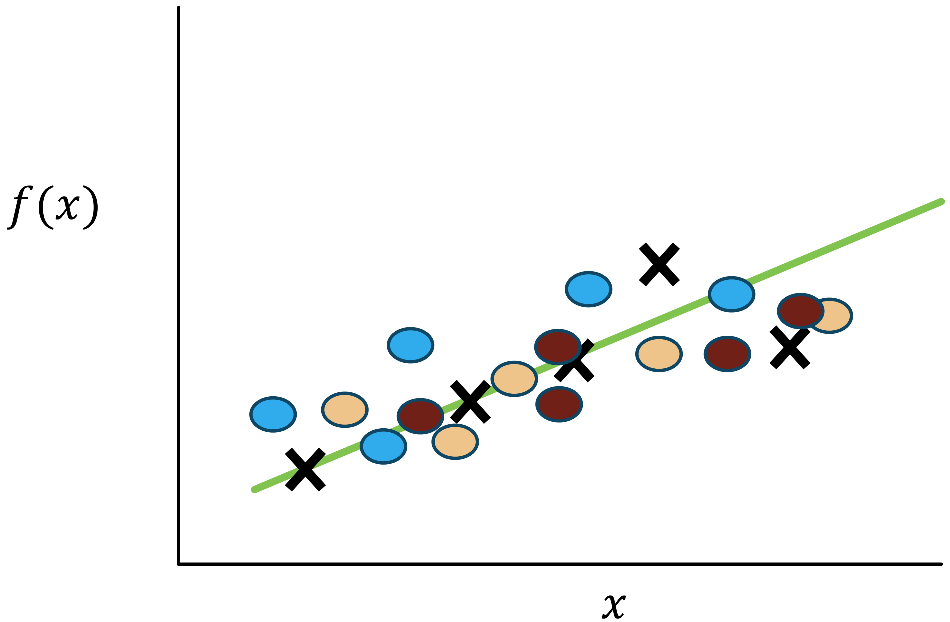
ML Model 1



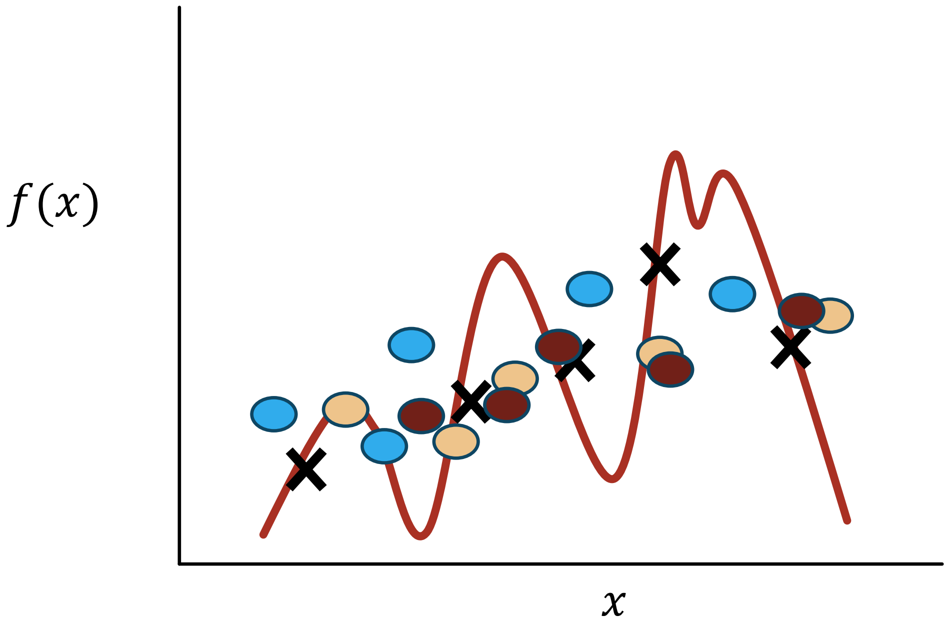
ML Model 2

- Model 1
  - Similar to its performance on training data, and test data 1
  - Test error $e_g(2)$
- Model 2
  - Very different from its performance on training data, and test data 1
  - Test error $e_r(2)$

Now, Let's See How They Perform on Test Data 3



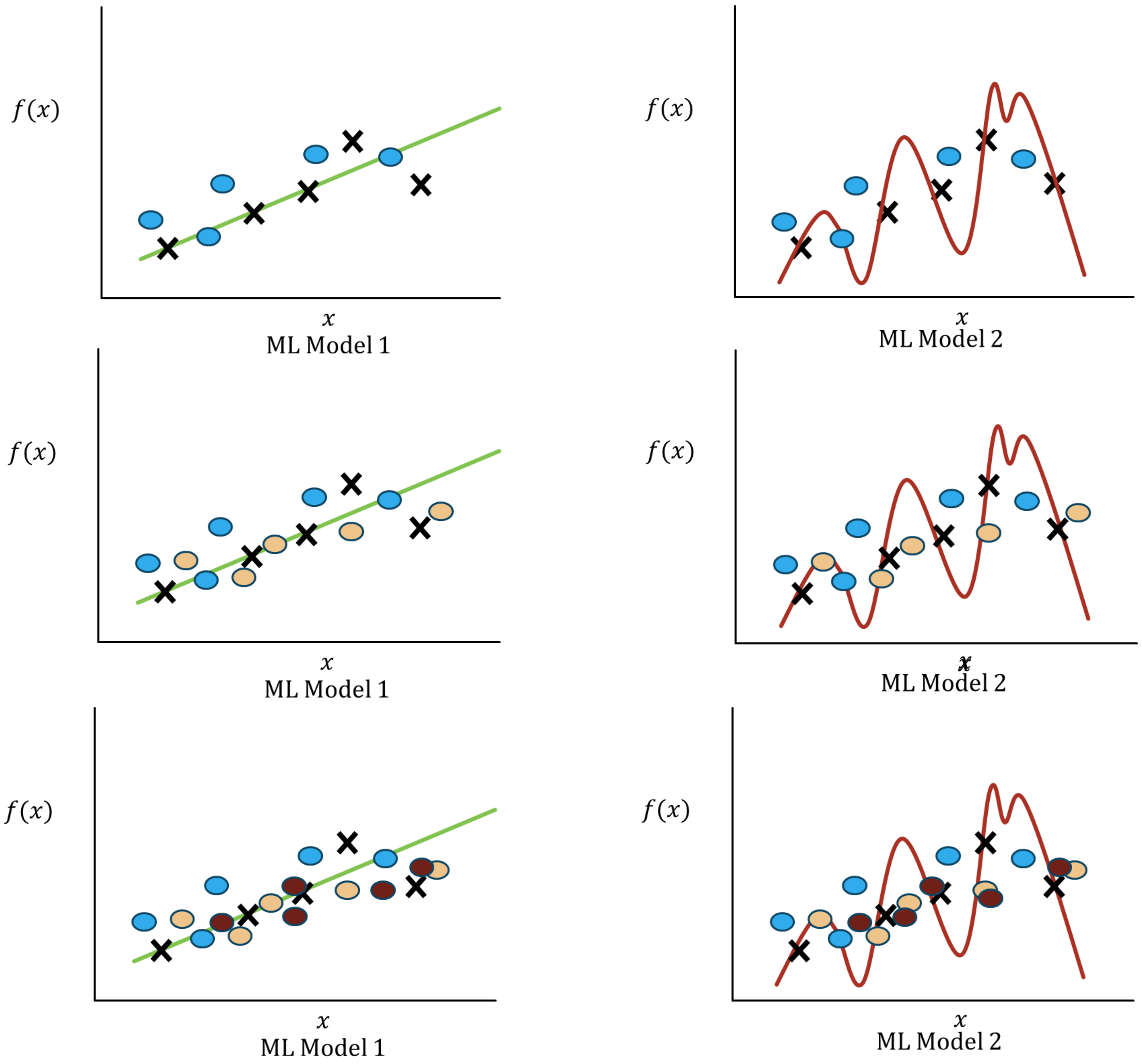
ML Model 1

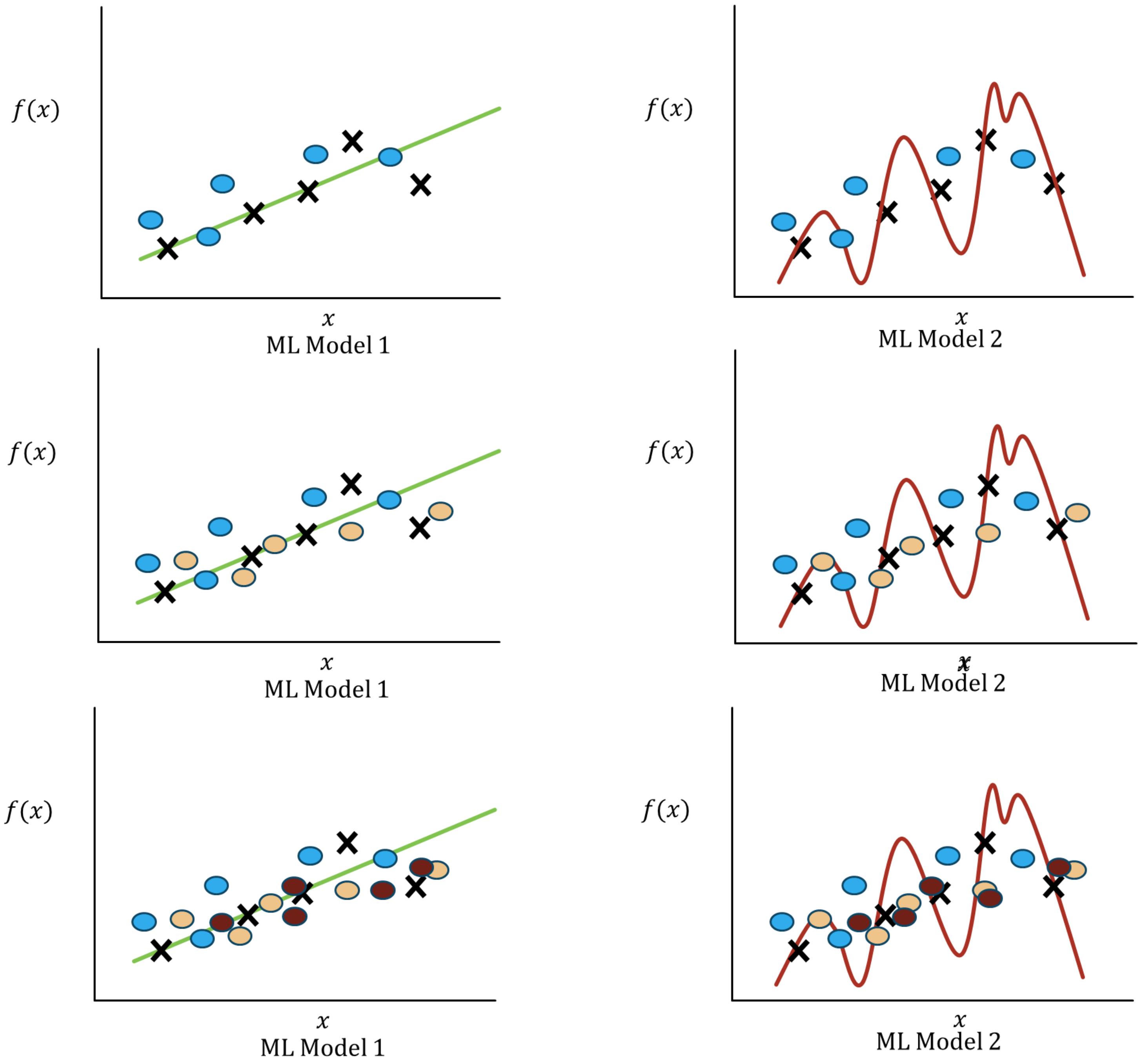


ML Model 2

- Model 1
  - Similar to its performance on training data, test data 1, and test data 2
  - Test error $e_g(3)$
- Model 2
  - Very different from its performance on training data, test data 1, and test data 2
  - Test error $e_r(3)$

So, What do We Observe?

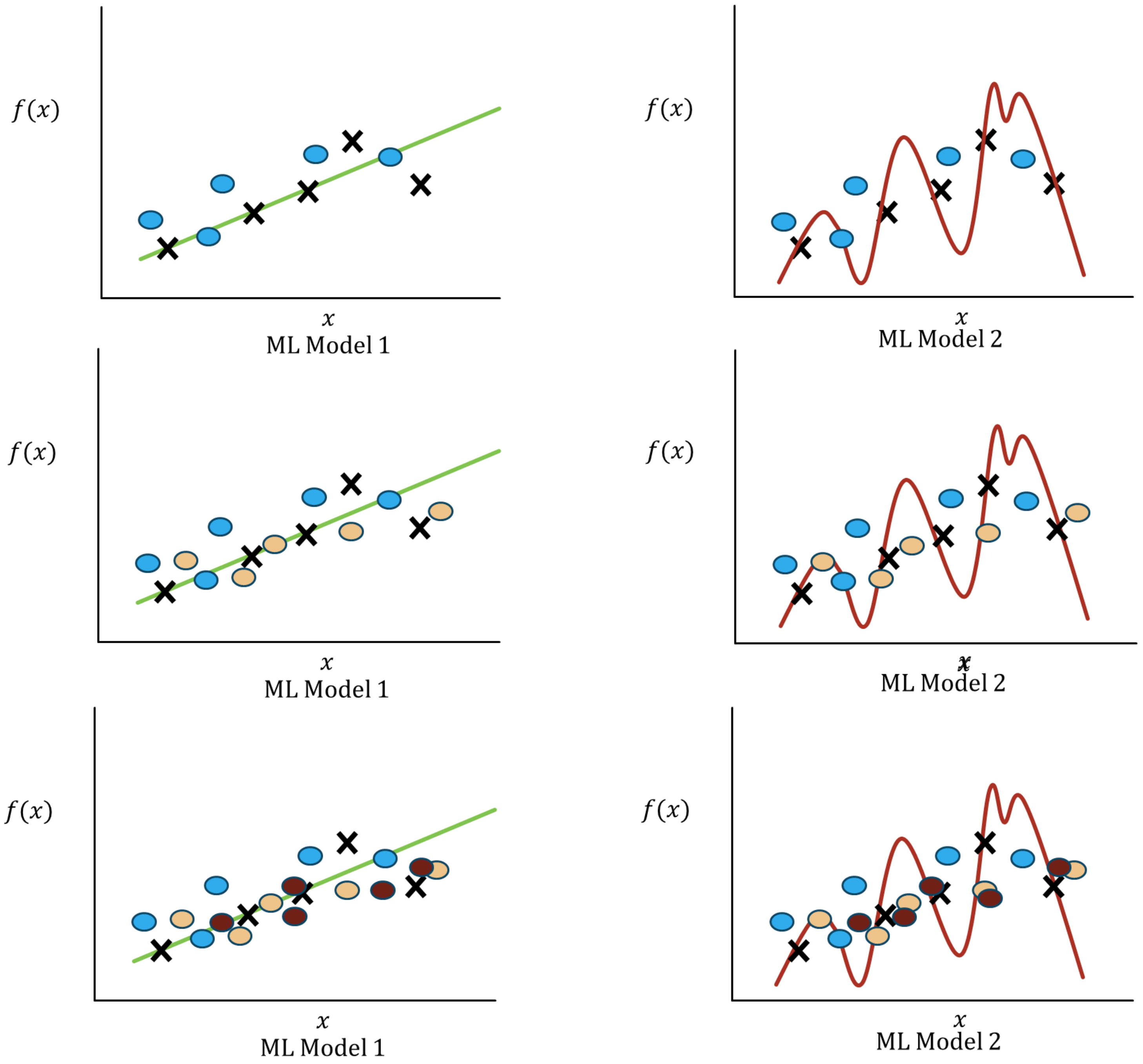




So, What do We Observe?

- The three sets of test data are similar to the training data with slight differences

- For the green curve (model 1), the errors in the test data $e_g(1), e_g(2), e_g(3)$ are similar to each other and similar to training error
  - Variance of the error is low
  - Low variance
  - High bias

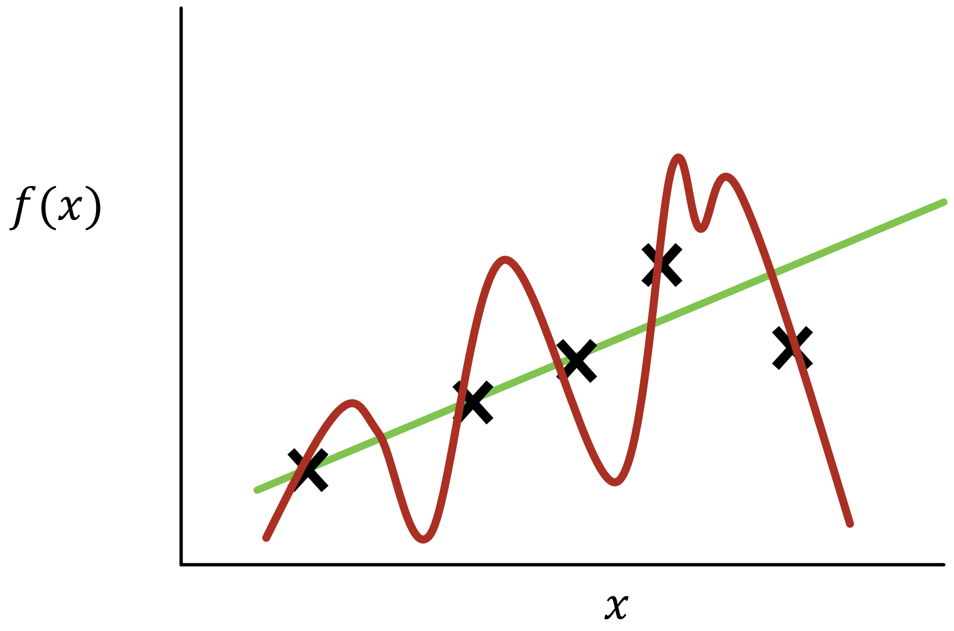


So, What do We Observe?

- The three sets of test data are similar to the training data with slight differences

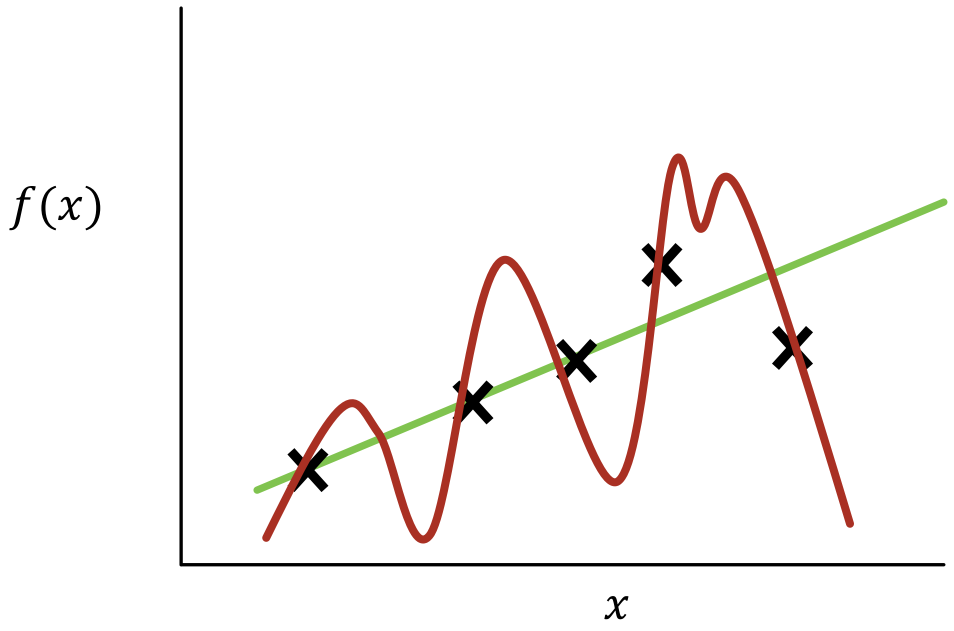
- For the red curve (model 2), the errors in the test data $e_r(1), e_r(2), e_r(3)$ are not similar to each other and not similar to training error
  - Variance of the error is high
  - High variance
  - Low bias

Bias and Variance



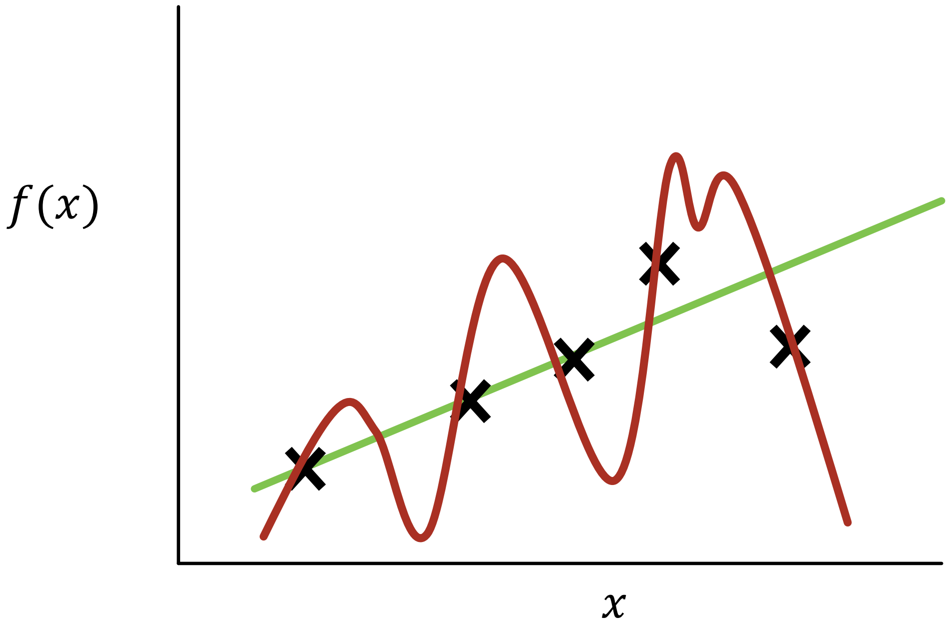
- More assumption: simple model/ simple curve
  - More bias, less variance
- Less assumption: complex model/ simple curve
  - Less bias, more variance

Impact of Bias and Variance



- More assumption: simple model/ simple curve
  - More bias, less variance
  - Underfitting
  - Consistent but relative poorer performance across datasets
- Less assumption: complex model/ simple curve
  - Less bias, more variance
  - Overfitting
  - Inconsistent performance across datasets

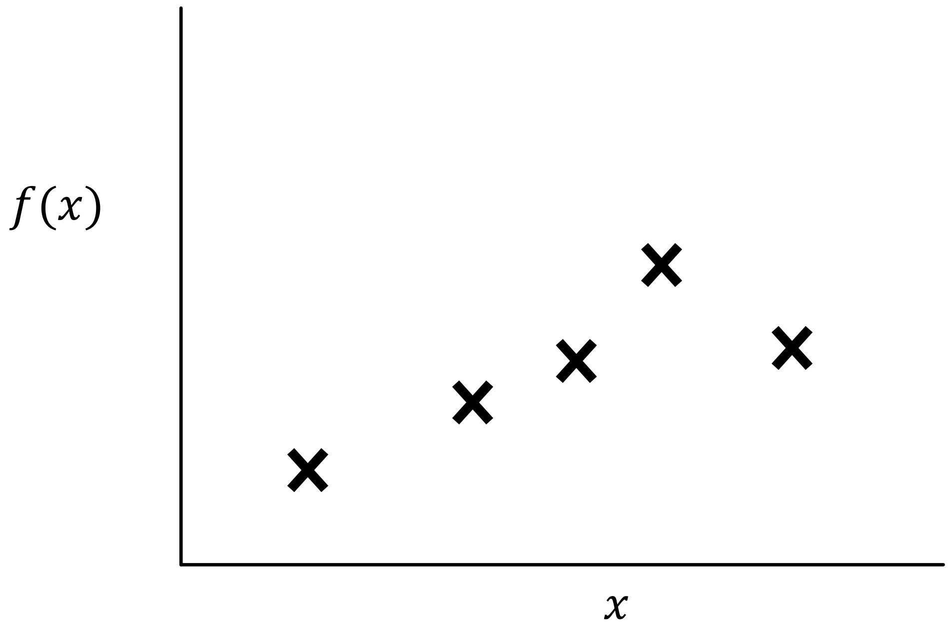
Impact of Bias and Variance



So, we want low bias and low variance in any model. But we can't achieve these at the same time

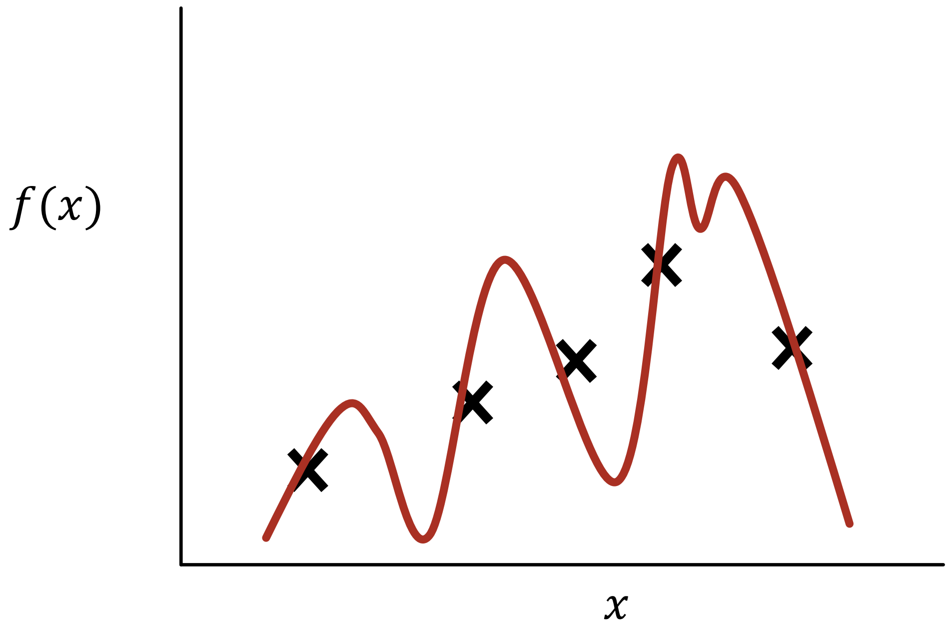
- More assumption: simple model/ simple curve
  - More bias, less variance
  - Underfitting
  - Consistent but relative poorer performance across datasets
- Less assumption: complex model/ simple curve
  - Less bias, more variance
  - Overfitting
  - Inconsistent performance across datasets

Another Aspect of Overfitting



- Suppose, you have a small amount of training data
- Can you design a model to fit all the training data?

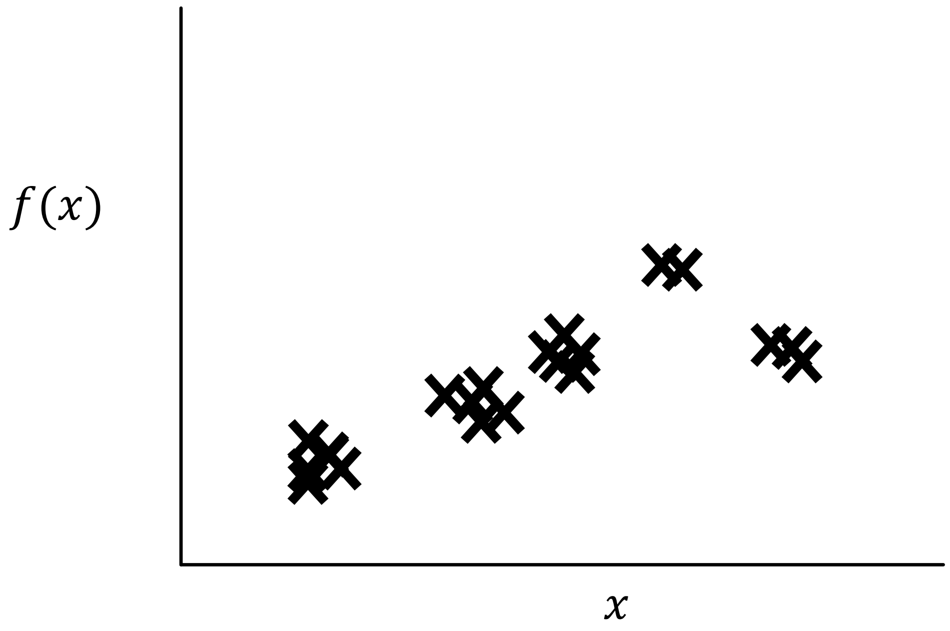
Another Aspect of Overfitting



- Suppose, you have a small amount of training data

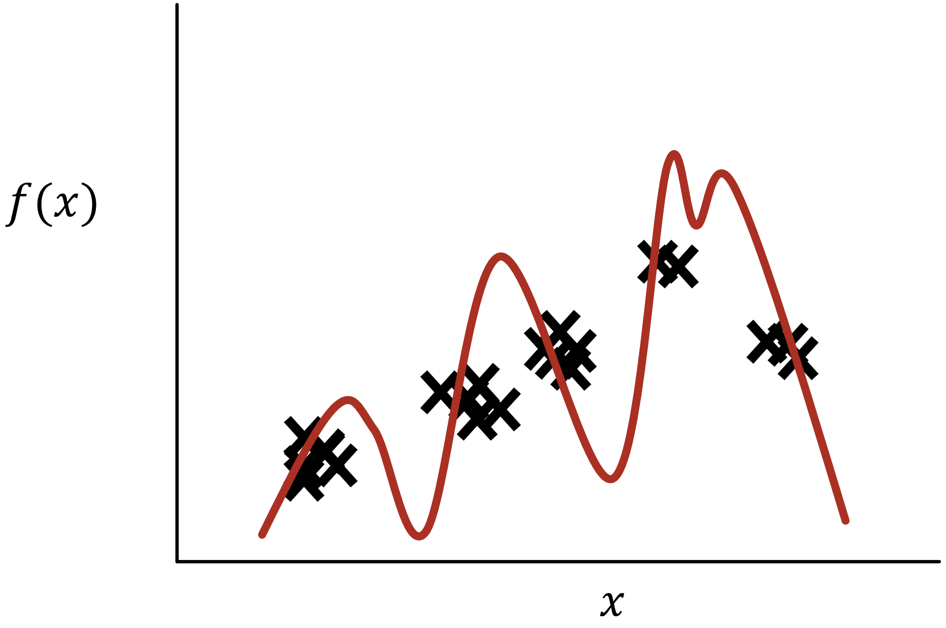
- Can you design a model to fit all the training data?
  - Yes (more or less easily)

Another Aspect of Overfitting



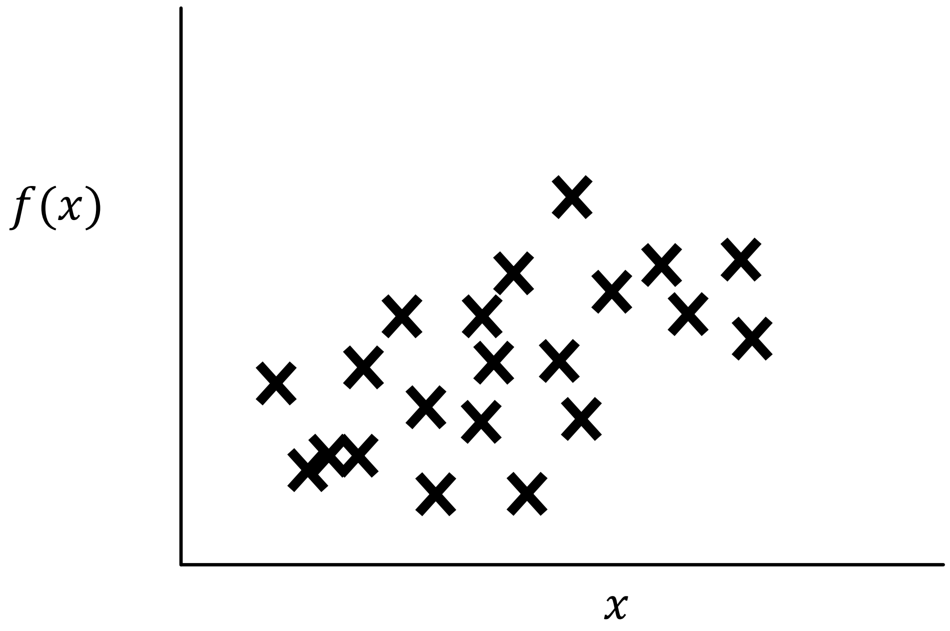
- Now, suppose, you have a lot of training data but of similar types
- Can you design a model to fit all the training data?

Another Aspect of Overfitting



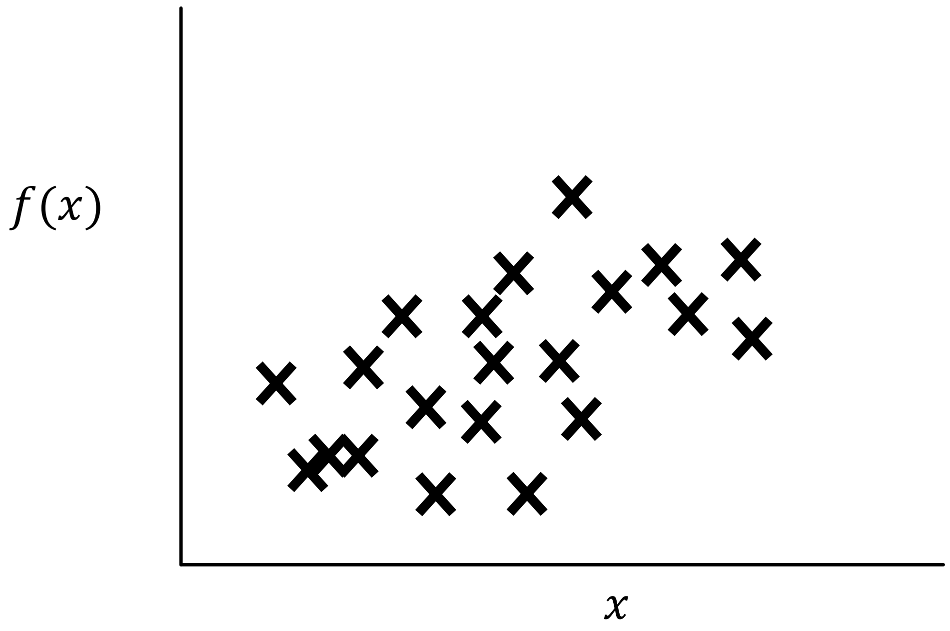
- Now, suppose, you have a lot of training data but of similar types
- Can you design a model to fit all the training data?
  - Yes (more or less)

Another Aspect of Overfitting



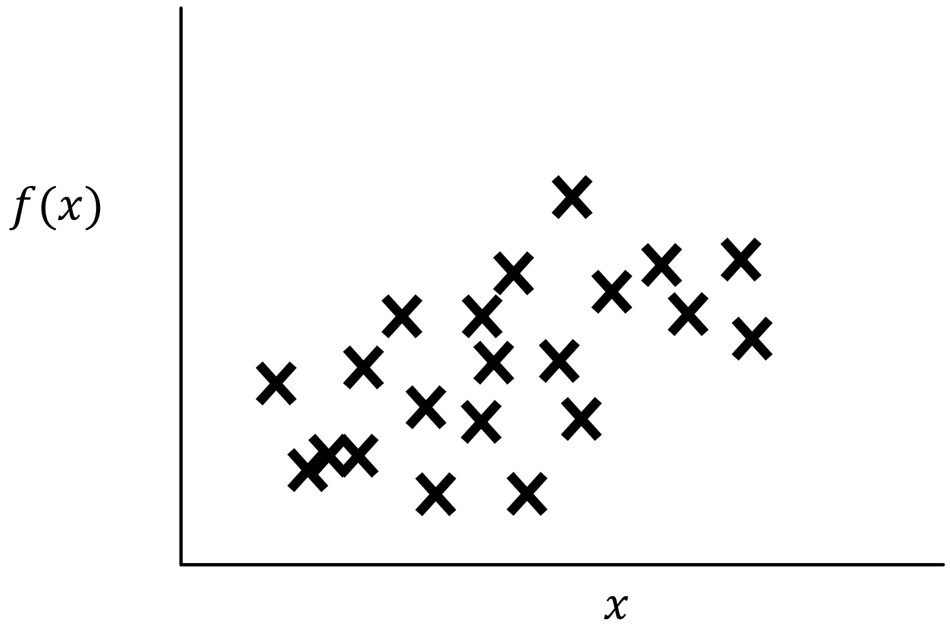
- Now, suppose, you have a lot of training data of sufficient diversity
- Can you design a model to fit all the training data?

Another Aspect of Overfitting



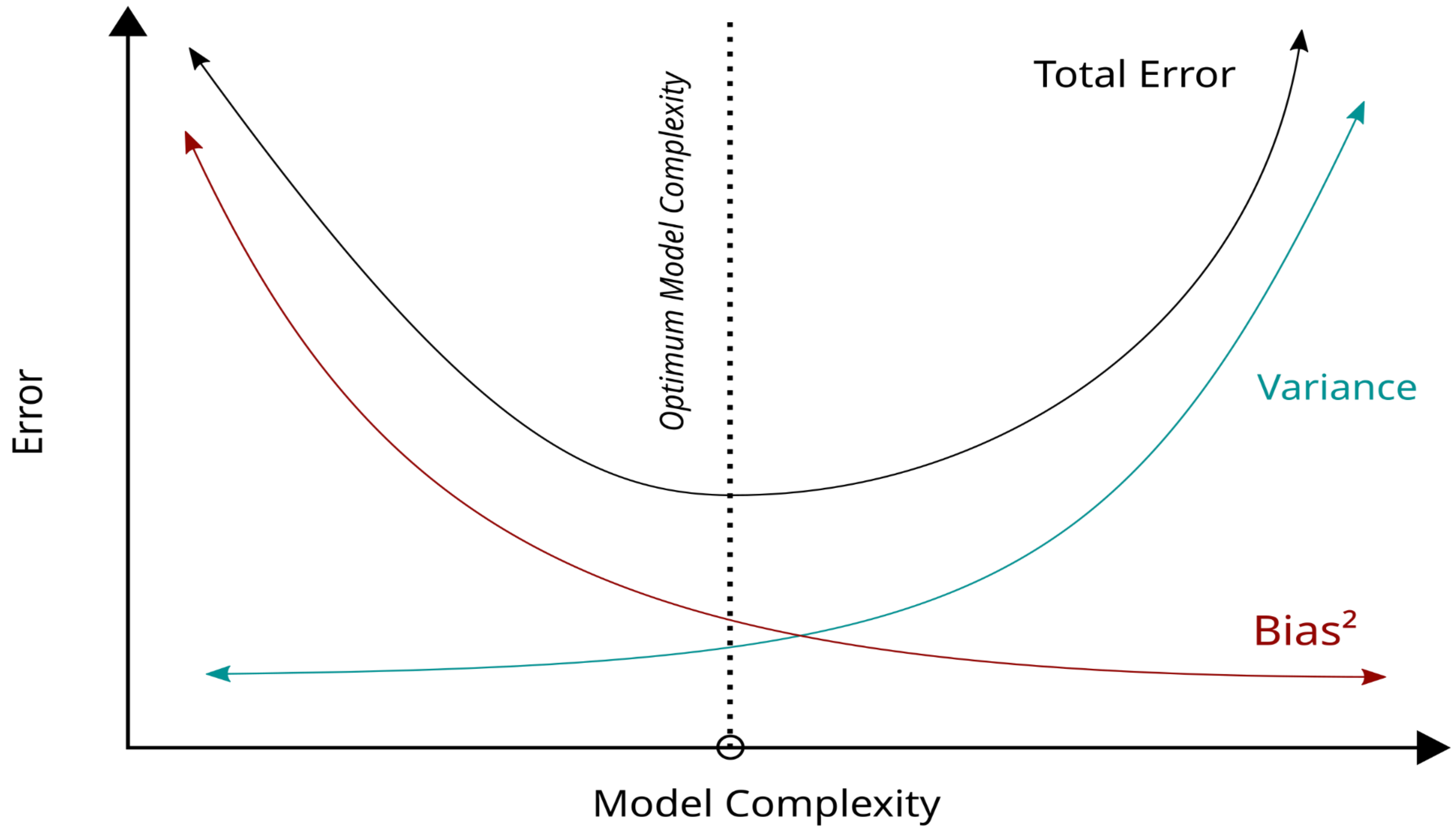
- Now, suppose, you have a lot of training data of sufficient diversity
- Can you design a model to fit all the training data?
  - Very difficult and practically almost impossible

Another Aspect of Overfitting



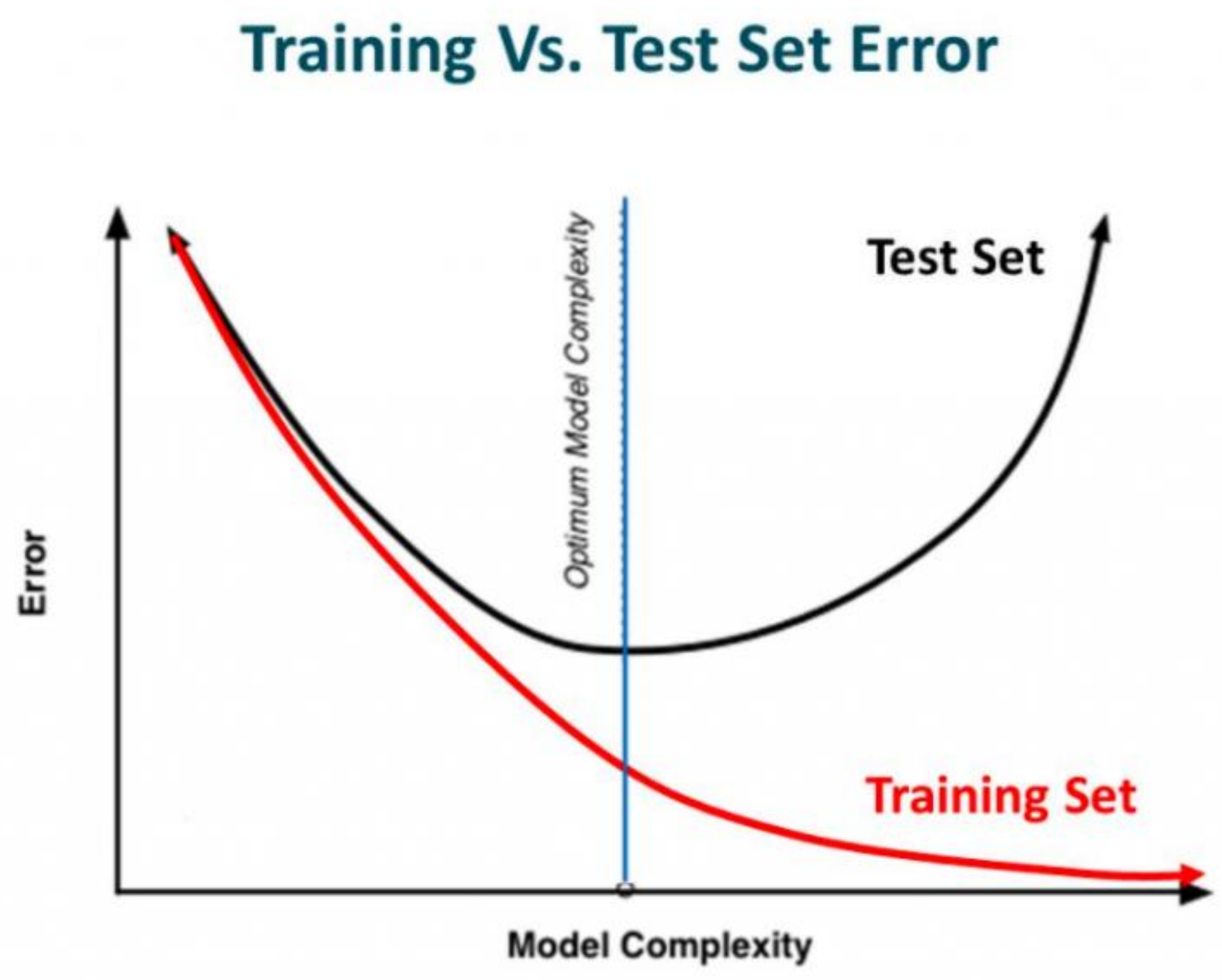
- Training data of sufficient diversity
  - Prevents overfitting

Errors



https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff#/media/File:Bias_and_variance_contributing_to_total_error.svg

Errors

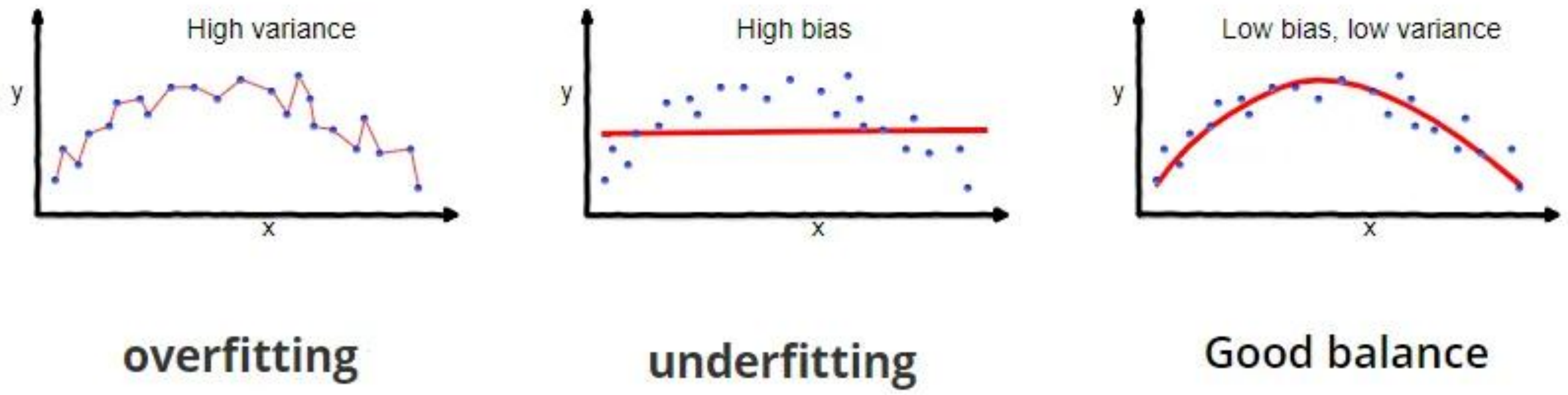


https://www.kaggle.com/code/sid321axn/regularization-techniques-in-deep-learning/notebook

Bias Variance Tradeoff

- We want low bias and low variance

- Need to find a sweet spot between simple and complex model



https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229

Regularization

- Any modification made to a learning algorithm that is intended to reduce its generalization error but not its training error

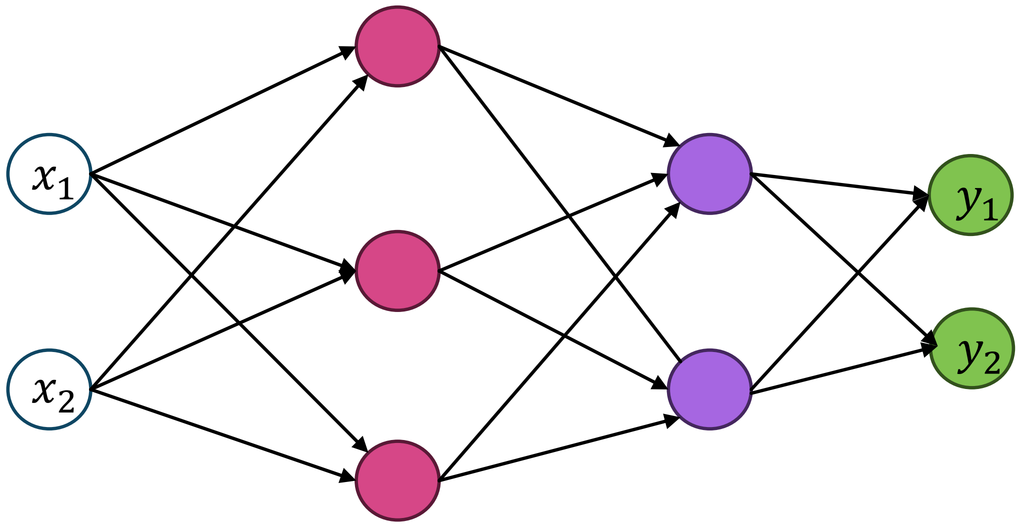
Regularization Strategies

- Adding extra constraint
  - e.g., adding constraint to parameter values
- Extra terms in the objective function
  - Corresponds to a soft constraint on the parameter values
  - Penalties that encode prior knowledge
  - Constraint and penalties to design simpler model
    - Avoiding overfitting

Parameter Norm Penalties

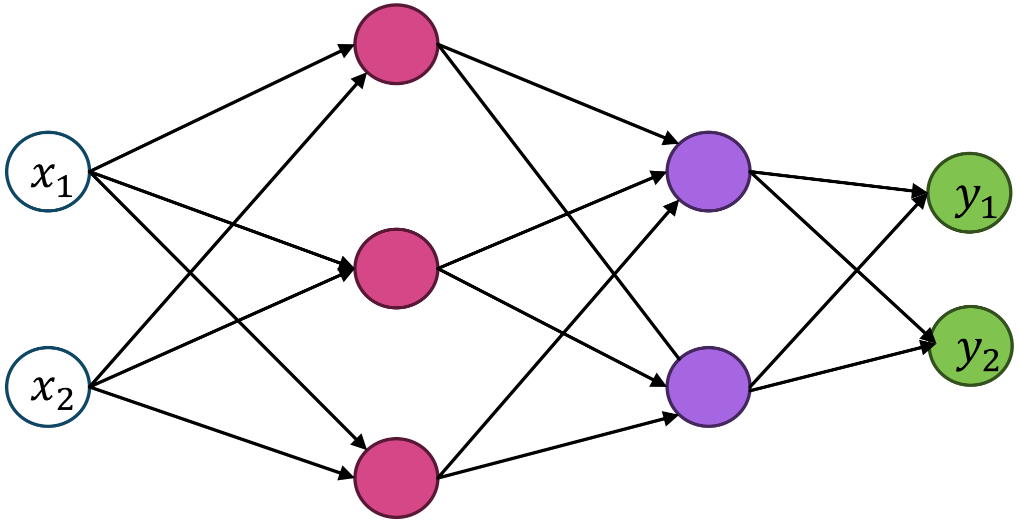
- $L_T(\Theta) = L(\Theta) + \Omega(\Theta)$
  - $L(\Theta)$: Actual loss computed from input data
  - $\Omega(\Theta)$: Parameter norm penalty term
  - Θ: All parameters (weights w and biases)
  - Typically the penalty is applied only on the weights and not biases (why?)
- L^2 regularization:
  - $L_T(w) = L(w) + 0.5 \sum_{i=1}^{M} |w_i|^2$
- L^1 regularization:
  - $L_T(w) = L(w) + \sum_{i=1}^{M} |w_i|$

Dropout



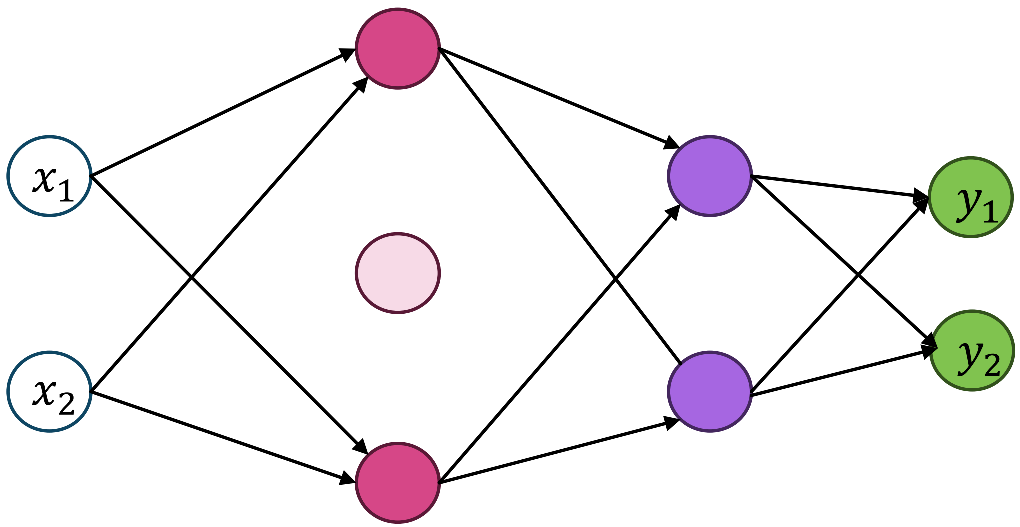
Design philosophy: remove complex interdependence between parameters

Dropout



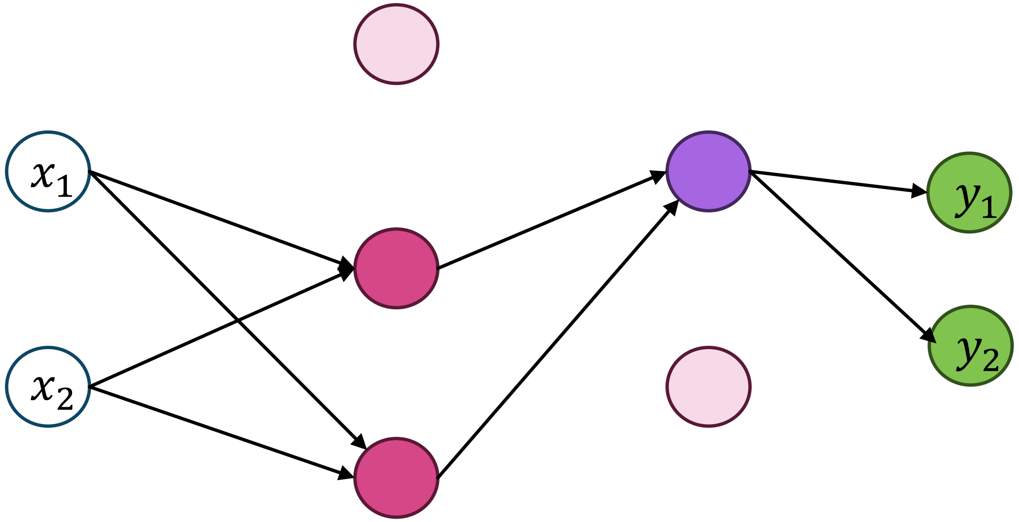
Epoch 1: Randomly choose and drop some nodes

Dropout: Training



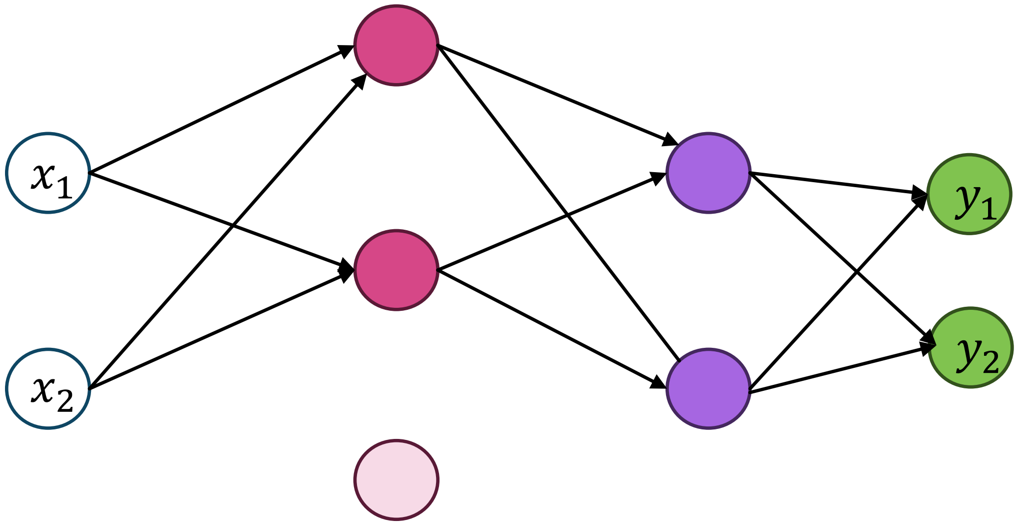
Epoch 1: Randomly choose and drop some nodes

Dropout: Training



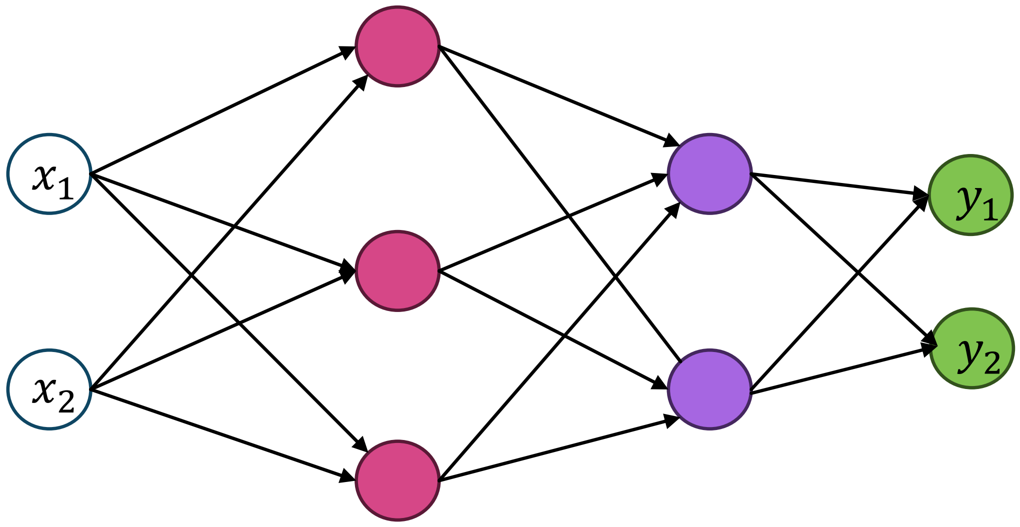
Epoch 2: Randomly choose and drop some nodes

Dropout



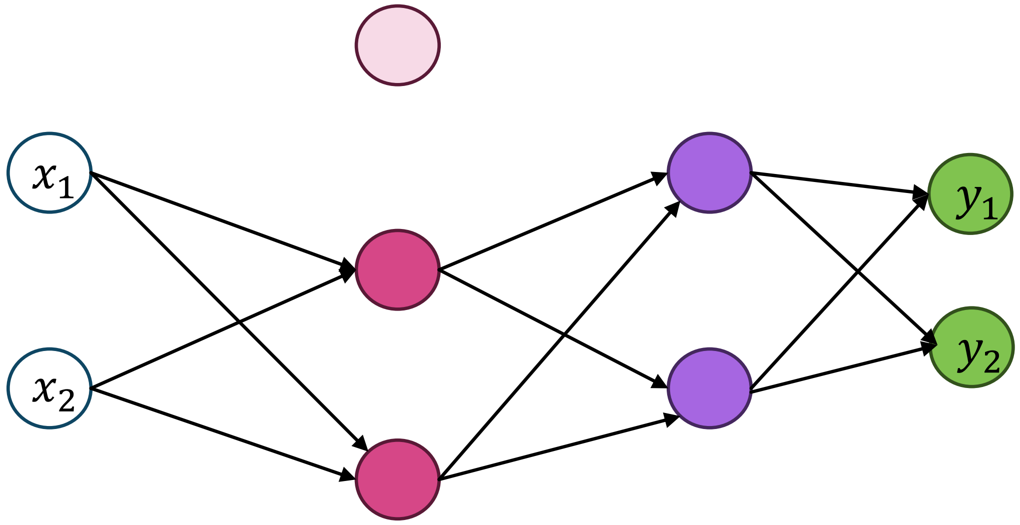
Epoch 3: Randomly choose and drop some nodes

Dropout: Testing



Make the prediction using the complete network at the time of testing

Dropout



Dropout can be implemented on the nodes of multiple layers

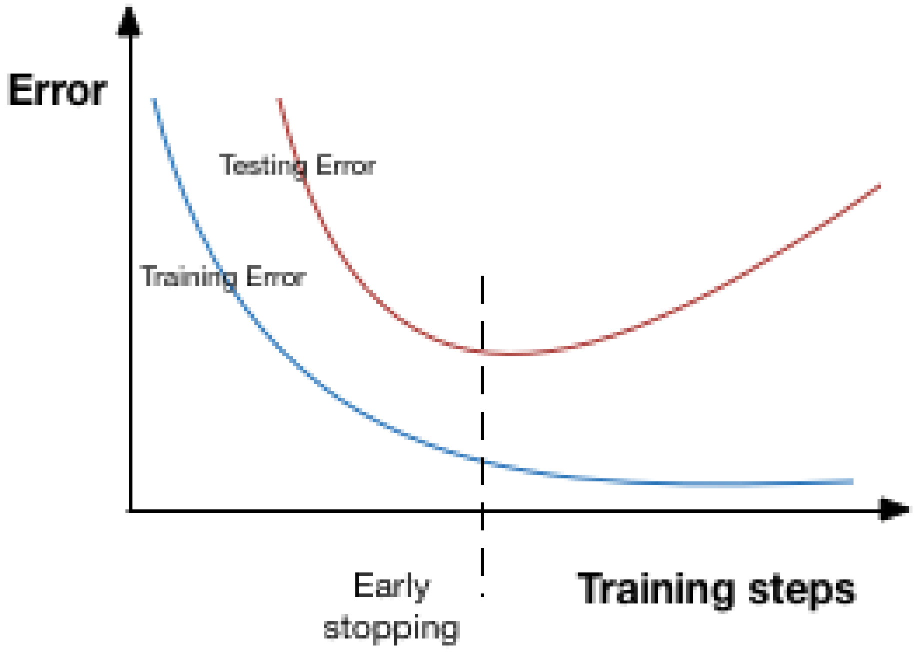
Because of dropout, neurons connected to the dropped neuron may stop to rely heavily on the dropped neuron.

As if other neurons learn to live without the dropped neuron

Dropout reduces **complex co-adaptations**

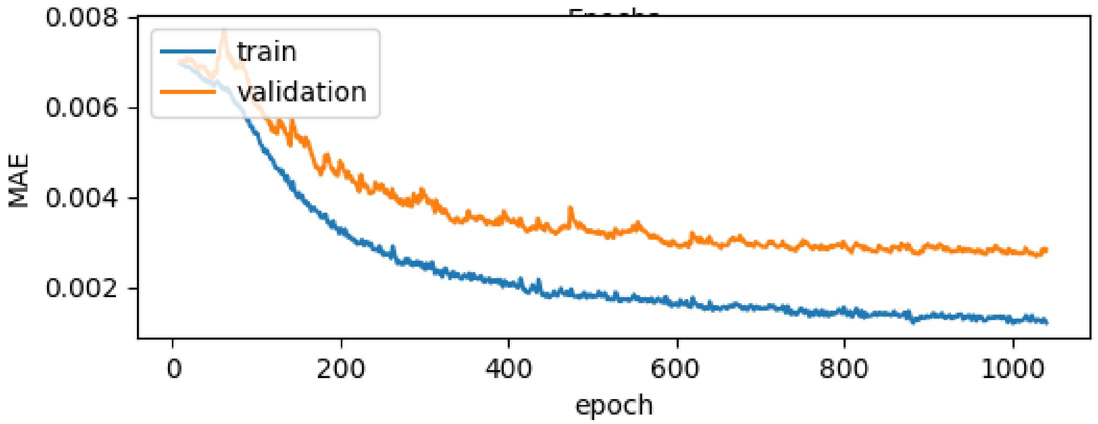
Early Stopping

- After a certain number of epochs, the model starts to overfit
- Stop early



Early Stopping

- After a certain number of epochs, the model starts to overfit
- Stop early



- Use validation data
- Find plateau in validation loss/ accuracy

Parameter Initialization

- Tries to achieve some nice properties at the beginning of training

- An initial point
  - May be helpful from the perspective of optimization
  - May be detrimental from the view point of generalization
  - We do not have a clear understanding of both

- Breaking the symmetry: Something about the initialization that we understand

Xavier Glorot Initialization

- Sample the initial weights from the uniform distribution as follows

$$w_{ij}(l) \sim U\left(-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}\right)$$

$$n_{in}\text{: } Number\ of\ inputs\ to\ the\ nodes\ of\ layer\ l$$

- This expression is obtained using assumption of linearity in each layer
  - Practically not possible
  - However, it works well for nonlinear models also

Xavier Glorot Initialization

- Sample the initial weights from the uniform distribution as follows

$$w_{ij}(l) \sim U\left(-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}\right)$$

$$n_{in}:\ Number\ of\ inputs\ to\ the\ nodes\ of\ layer\ l$$

- Considering linear nodes, the output of node j at layer l is

$$z_j(l) = \sum_{i=1}^{n_{in}} w_{ij}(l)$$

- If $w_{ij}(l)$ does not reduce as n_{in} increases, $z_j(l)$ may become large and cause exploding gradient problem at the next layer
- If $w_{ij}(l)$ does not increase as n_{in} decreases, $z_j(l)$ may become small and cause vanishing gradient problem at the next layer
- The above formulation of $w_{ij}(l)$ solves both the issues

Xavier Glorot Initialization

- Sample the initial weights from the uniform distribution as follows

$$w_{ij}(l) \sim U\left(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right)$$

$$n_{in}\text{: } Number\ of\ inputs\ to\ the\ nodes\ of\ layer\ l\ (fan\_in)$$
$$n_{out}\text{: } Number\ of\ outputs\ from\ the\ nodes\ of\ layer\ l\ (fan\_out)$$

- Some researchers use the above formulation as well

He Initialization

- Does not assume linear activations
- Originally designed for ReLU/ PReLU
- Sample the initial weights from the normal distribution as follows

$$w_{ij}(l) \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_{in}}}\right)$$

$$n_{in}\text{: } Number\ of\ inputs\ to\ the\ nodes\ of\ layer\ l\ (fan\_in)$$

$$n_{out}\text{: } Number\ of\ outputs\ from\ the\ nodes\ of\ layer\ l\ (fan\_out)$$

But Why Are We So Much Bothered About NNs?

- What is so special about NNs?

Universal Approximation Theorem

- A neural network, even with a single layer can approximate any function
- Then, why do we talk about deep networks?

Universal Approximation Theorem

- A neural network, even with a single hidden layer can approximate any function
- Then, why do we talk about deep networks?
  - Because, to approximate with a single layer, we may need absurdly many nodes

Universal Approximation Theorem

- A feedforward network with a linear output layer and at least one hidden layer with any “squashing” activation function (such as the logistic sigmoid activation function) can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units.