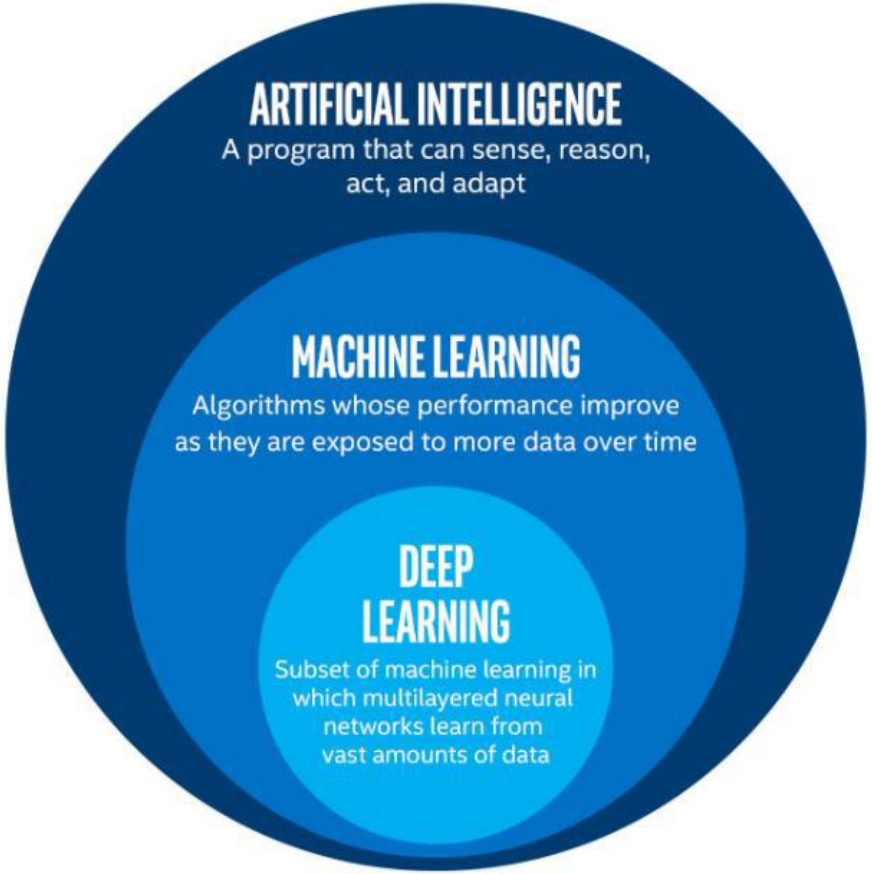
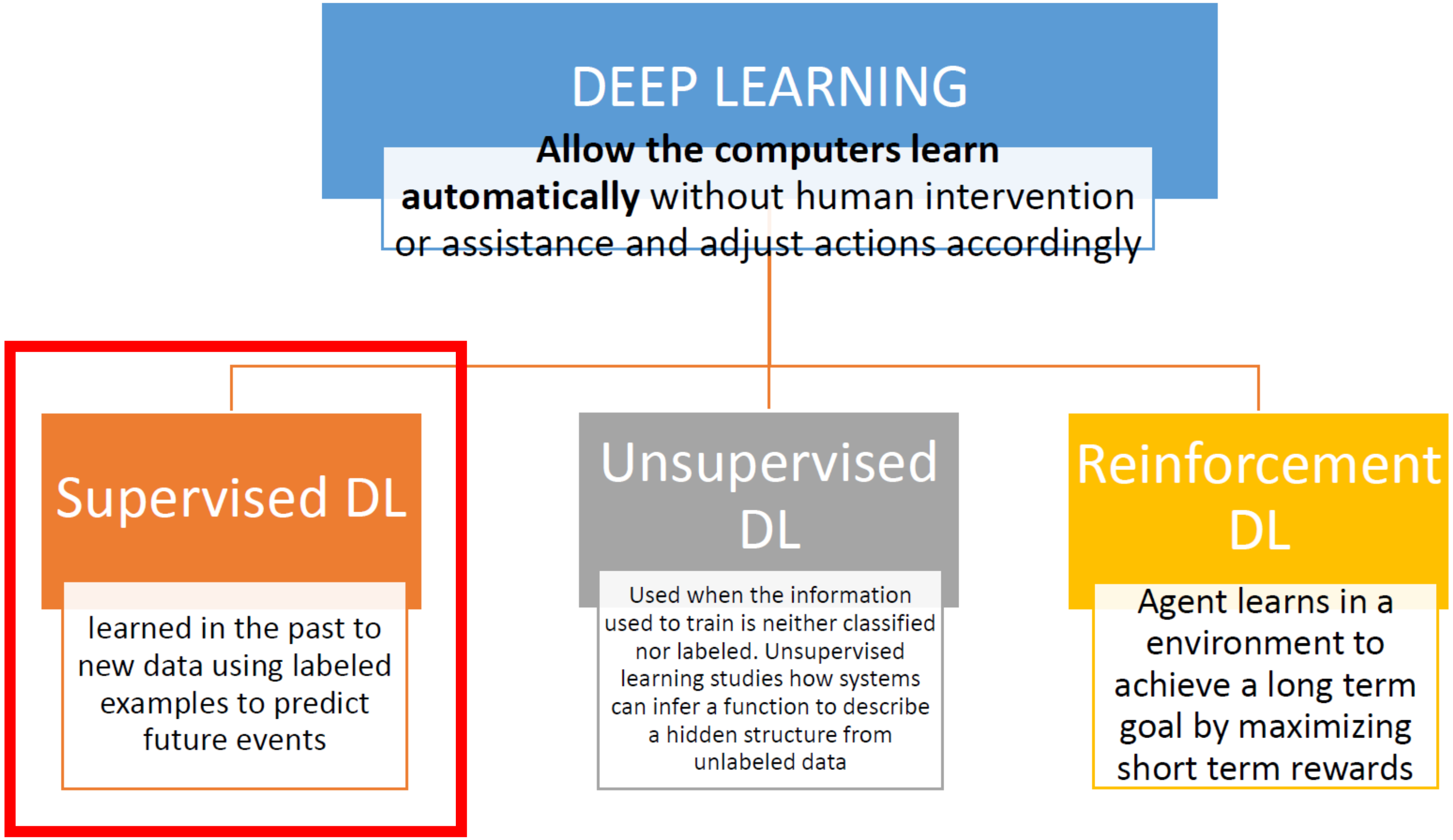
DEEP LEARNING & NEURAL NETWORKS

WEEK 03
SUPERVISED DEEP LEARNING |
MACHINE LEARNING BASICS

SPEAKER
Thakshila Dasun
BSc. Engineering Hons, CIMA UK, IEEE, IMech
Assistant Lecturer at SLIIT | Senior Lecturer at Academy of Innovative Education

Previous Week



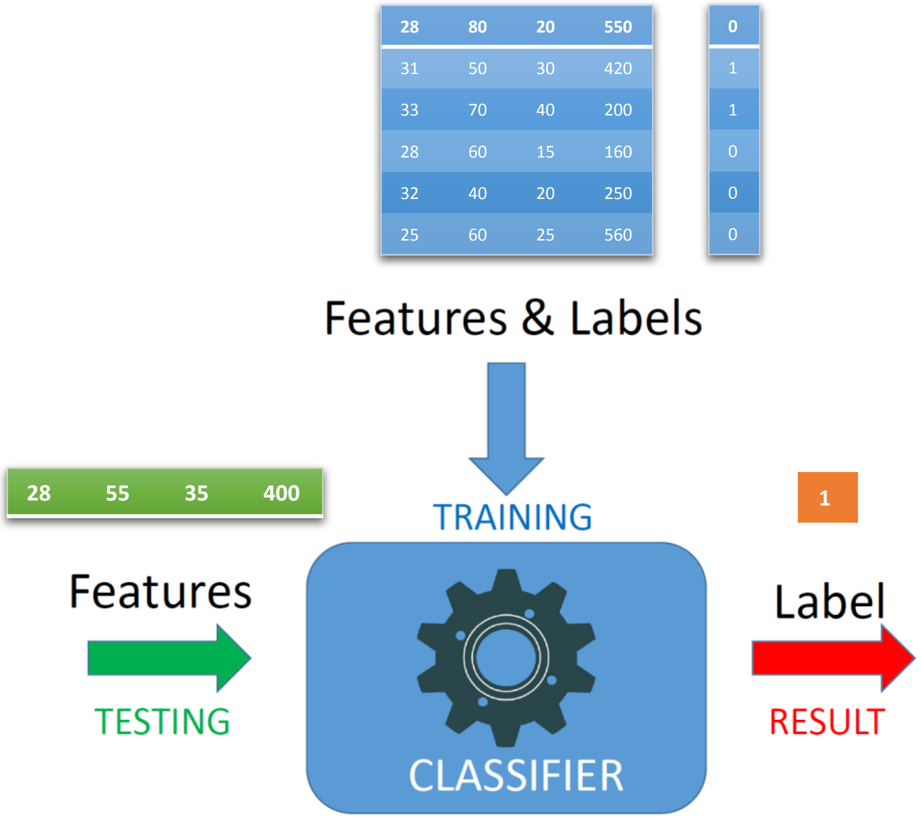


Predicting whether it is going to RAIN today or NOT

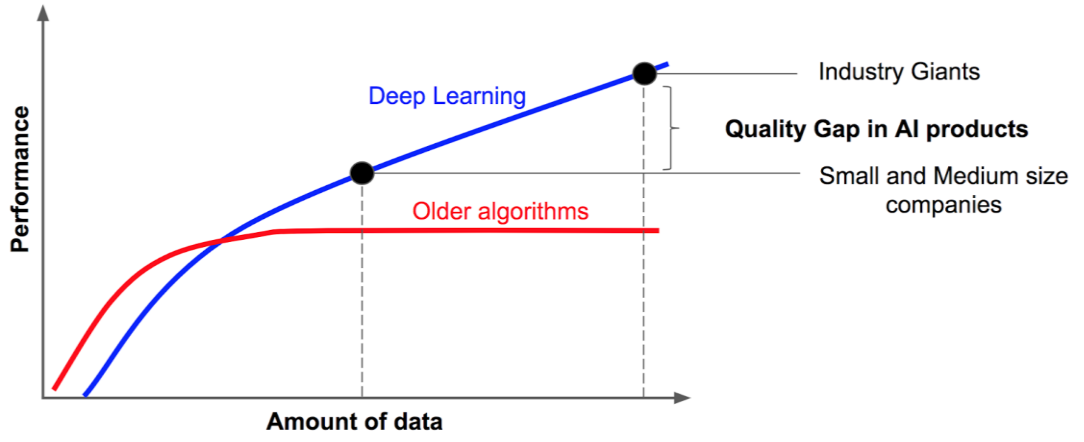
1. Identify Feature and Labels
   - Labels: Possible solution of the problem
   - Feature: Critical Attribute that decides the labels
2. Create a dataset

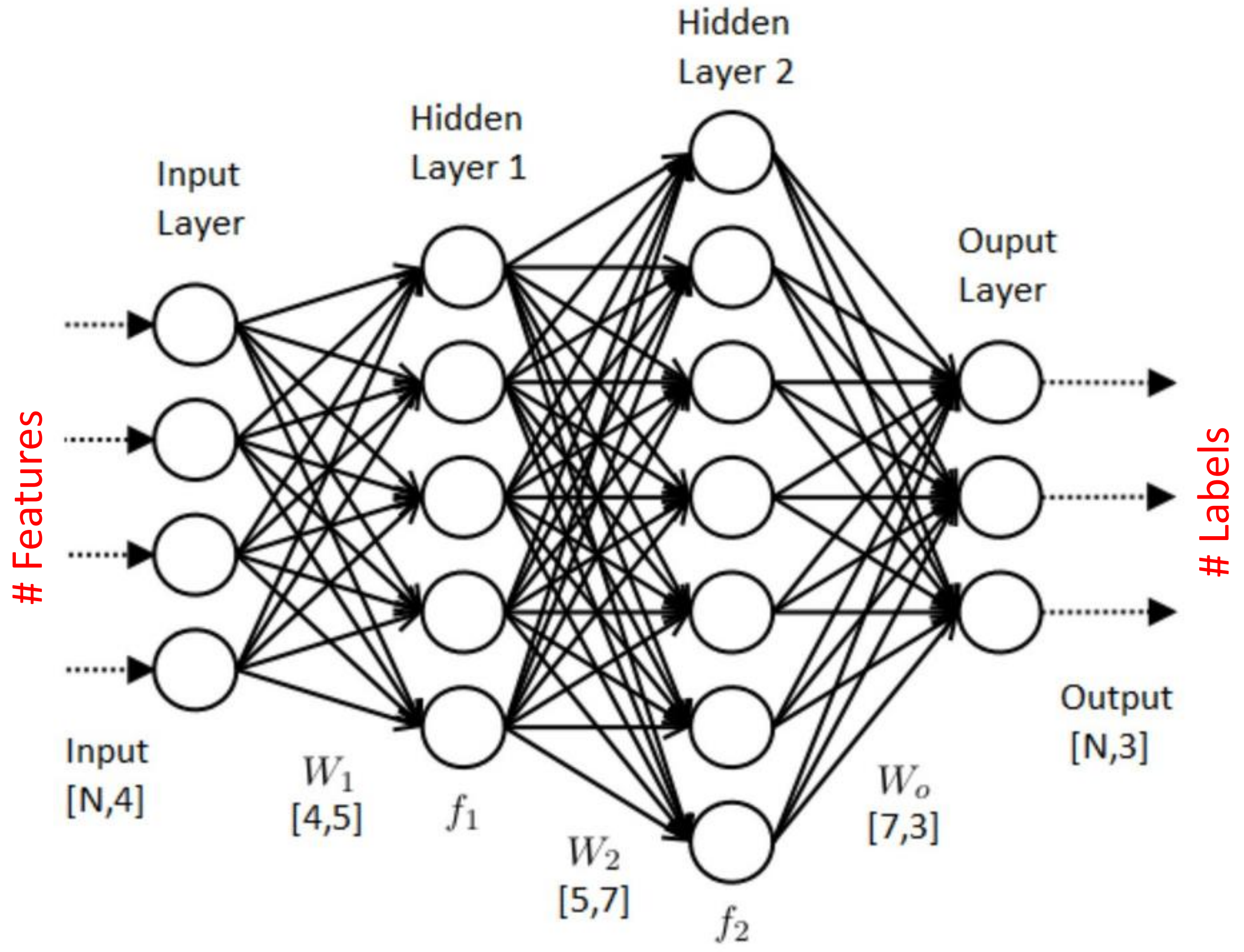
Features				Labels
Temperature C	Humidity %	Wind Speed/ Kmph	Light Intensity/ lux	
28	80	20	550	0
31	50	30	420	1
33	70	40	200	1
28	60	15	160	0
32	40	20	250	0
25	60	25	560	0

3. Train the NN
4. Test & Results

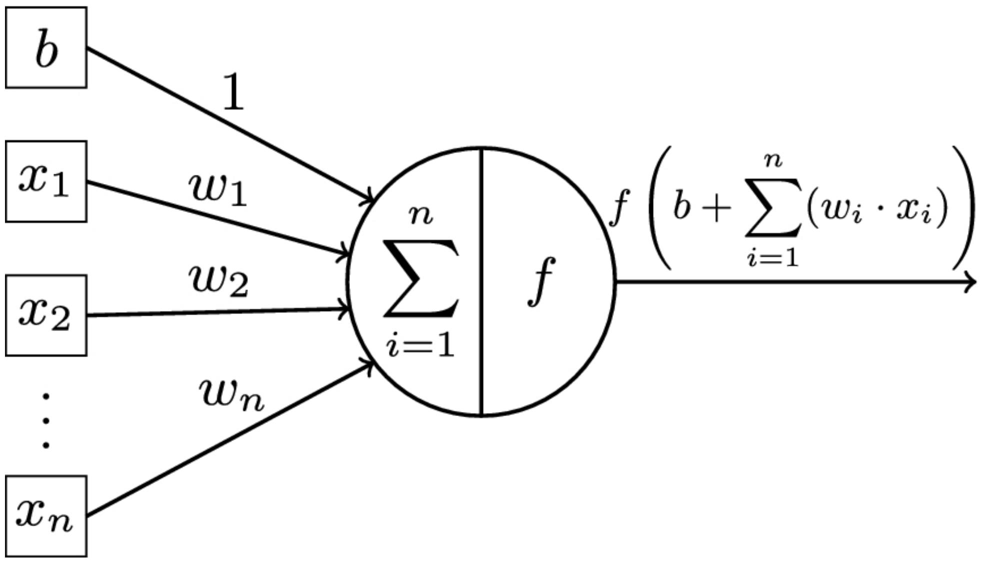


- Classification: Predicting discrete labels
- Regression: Predicting continuous labels

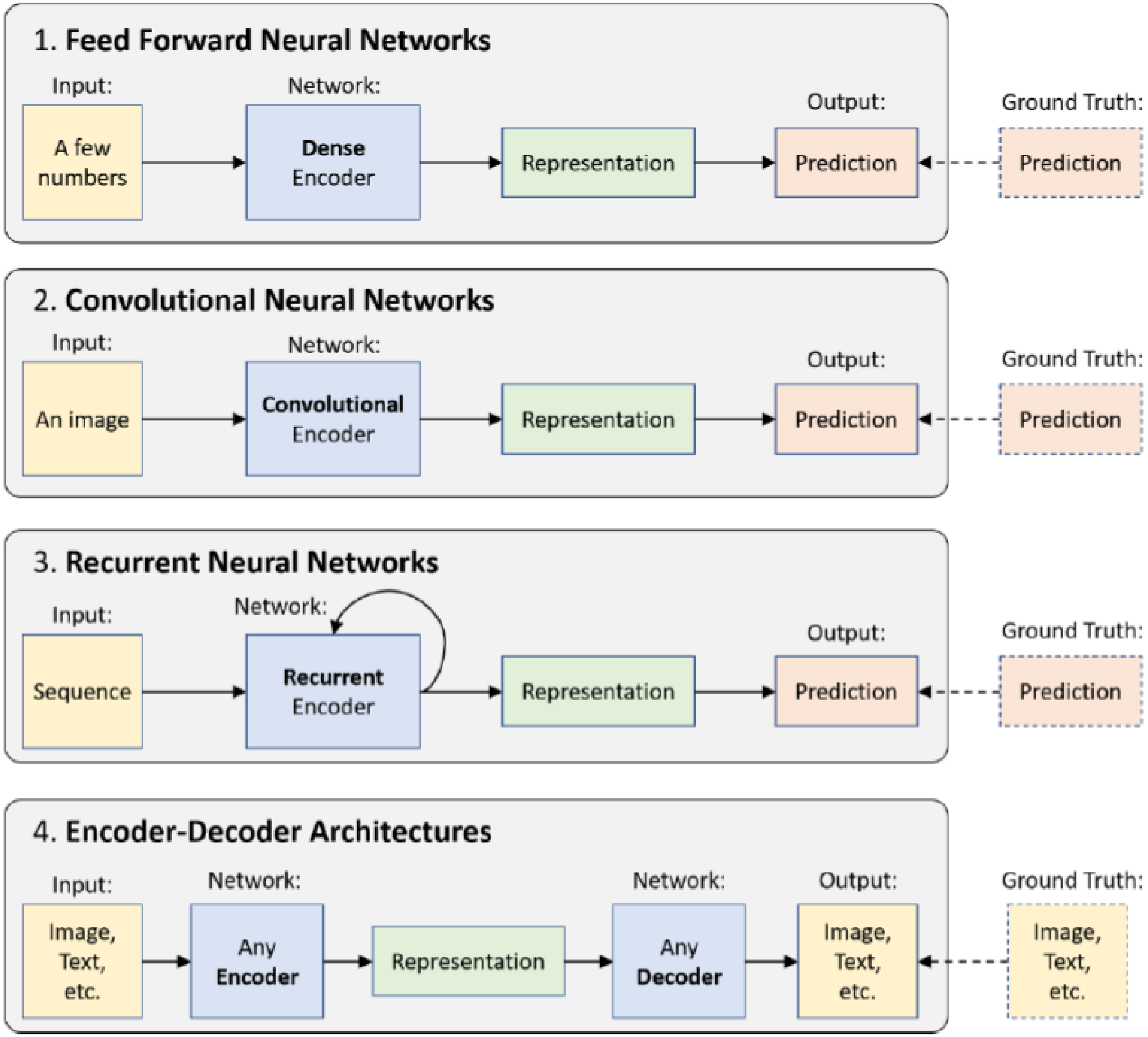




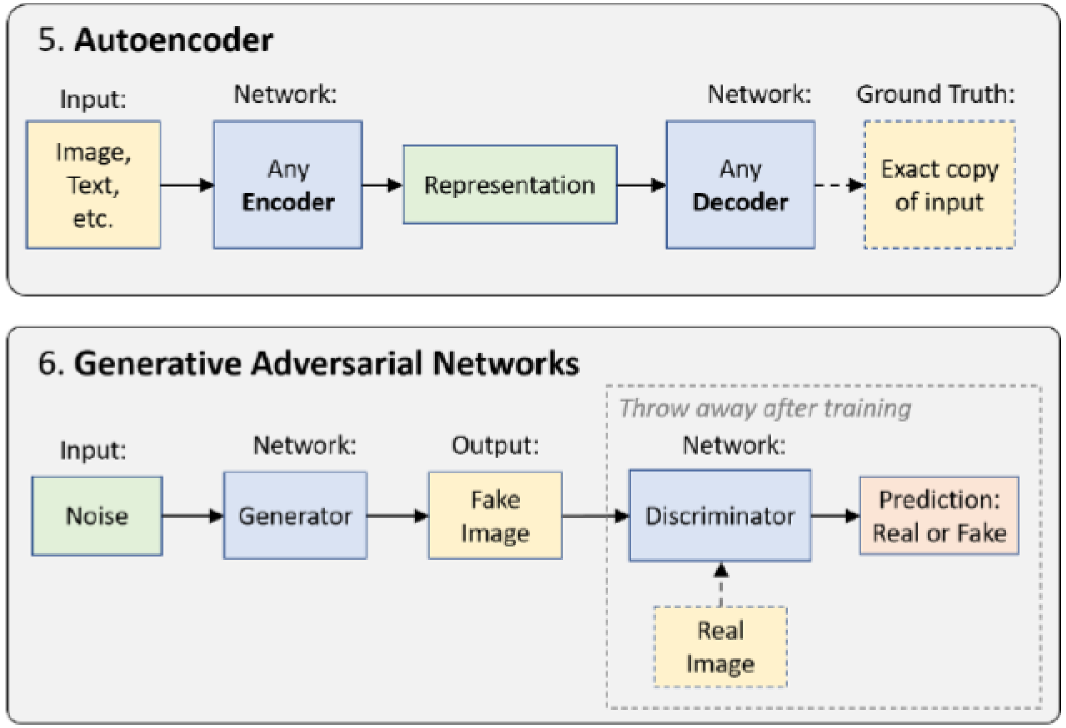
Weights (W) : All the Nets have
Biases (b) : All the Neuron other than neurons in the input layer has



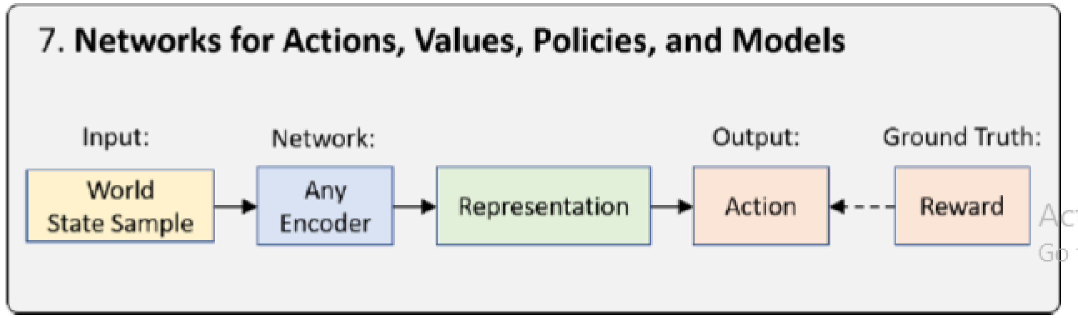
Supervised Learning



Unsupervised Learning



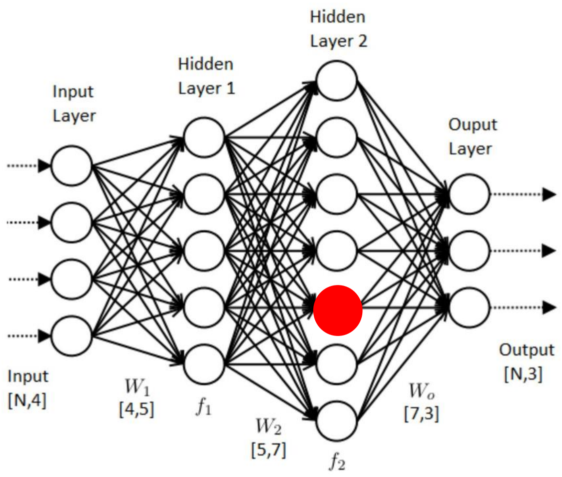
Reinforcement Learning



Today

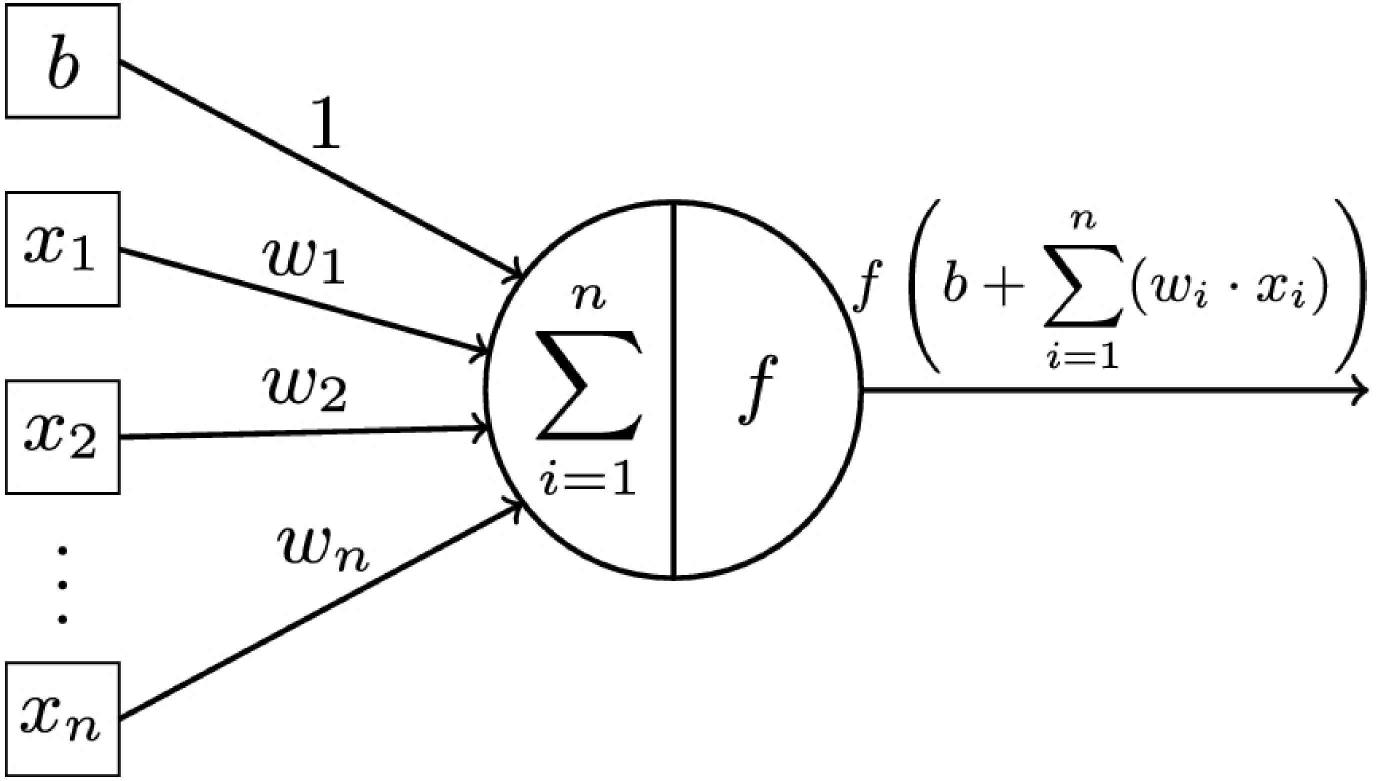
1. Activation Functions
2. Loss Functions
3. Optimizers
4. FFNN for MNIST Dataset
5. In-Class Practical I : Web Application for Handwritten Digits

Activation Functions(1)



$$Z = \left(\sum_{i=1}^{n} wi \cdot xi \right) + \mathrm{b}$$

$$Y = F(Z)$$



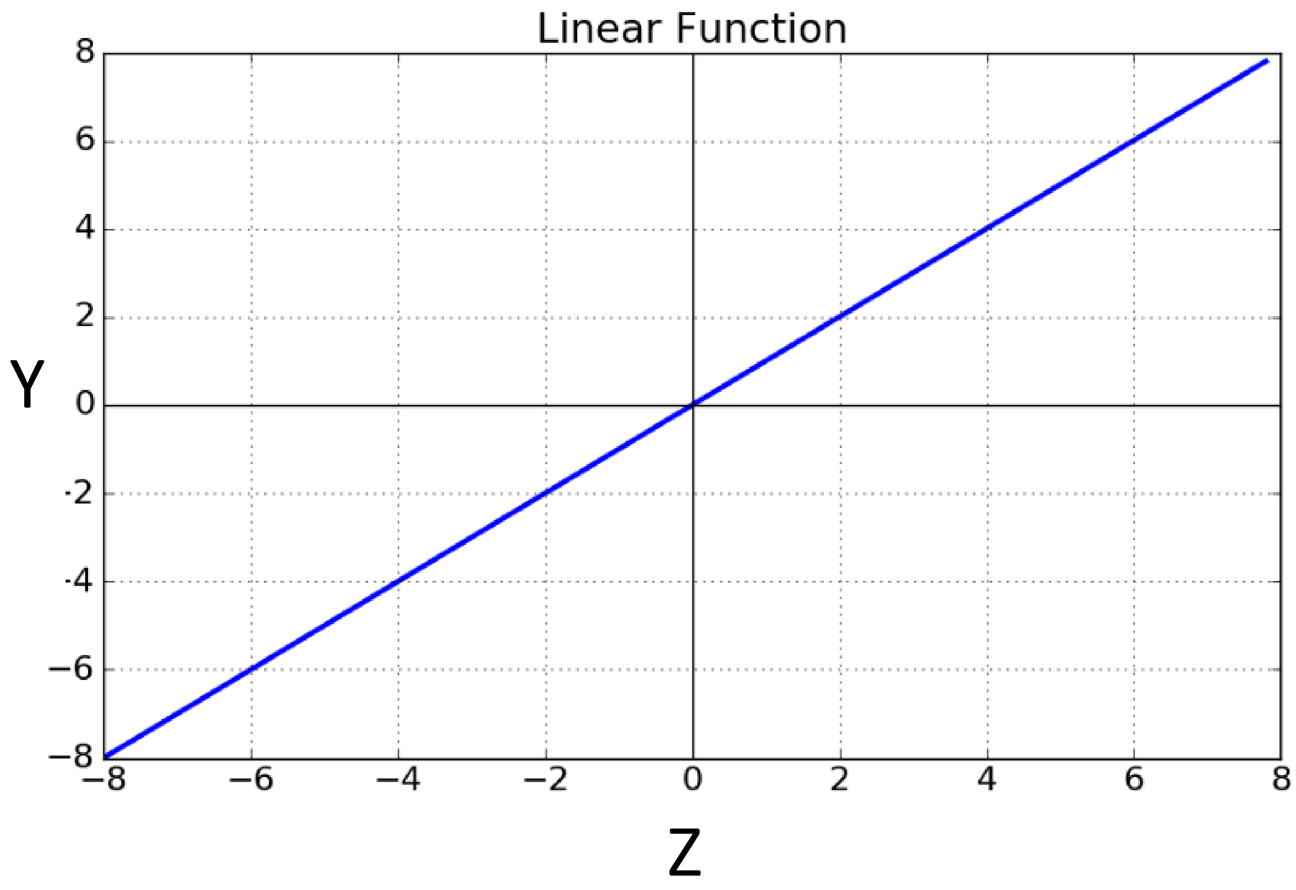
Types of activation functions

- The Activation Functions can be basically divided into 2 types
  - Linear or Identity Activation Function
  - Non-linear Activation Functions

Linear or Identity Activation Function

- The activation is proportional to the input. . This can be applied to various neurons and multiple neurons can be activated at the same time

- **Equation : Y=** F(Z) = Z
- **Range :** (-infinity to infinity)



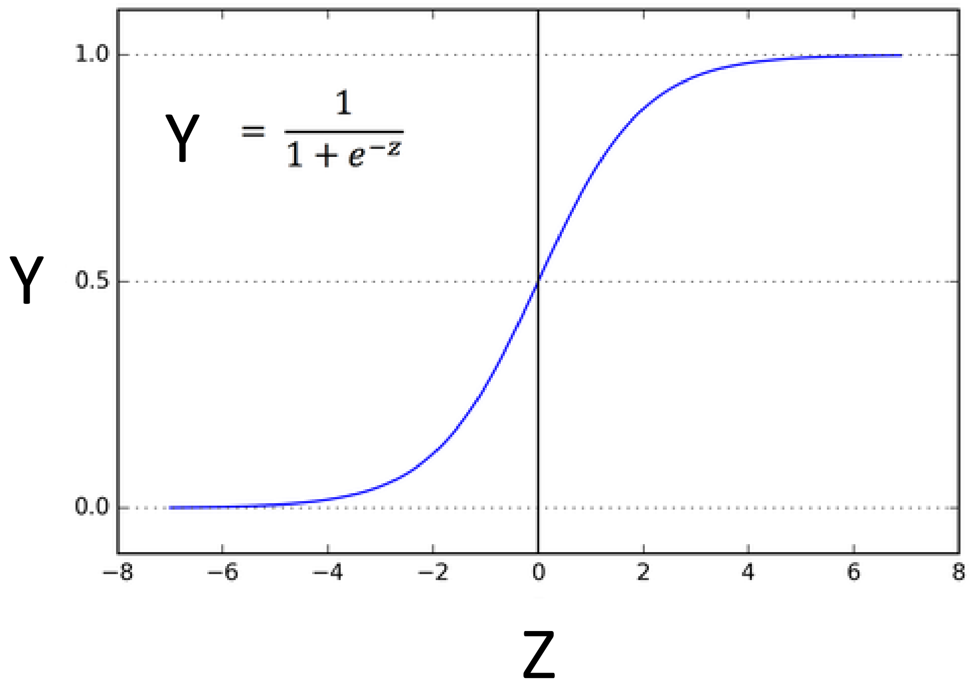
Non-linear Activation Functions

- The Nonlinear Activation Functions are the most used activation functions. It makes it easy for the model to generalize or adapt with variety of data and to differentiate between the output.

1. **Sigmoid or Logistic Activation Function**
2. **Tanh or hyperbolic tangent Activation Function:**
3. **ReLU (Rectified Linear Unit) Activation Function**
4. **Leaky ReLU**
5. **Softmax**

Sigmoid or Logistic Activation Function

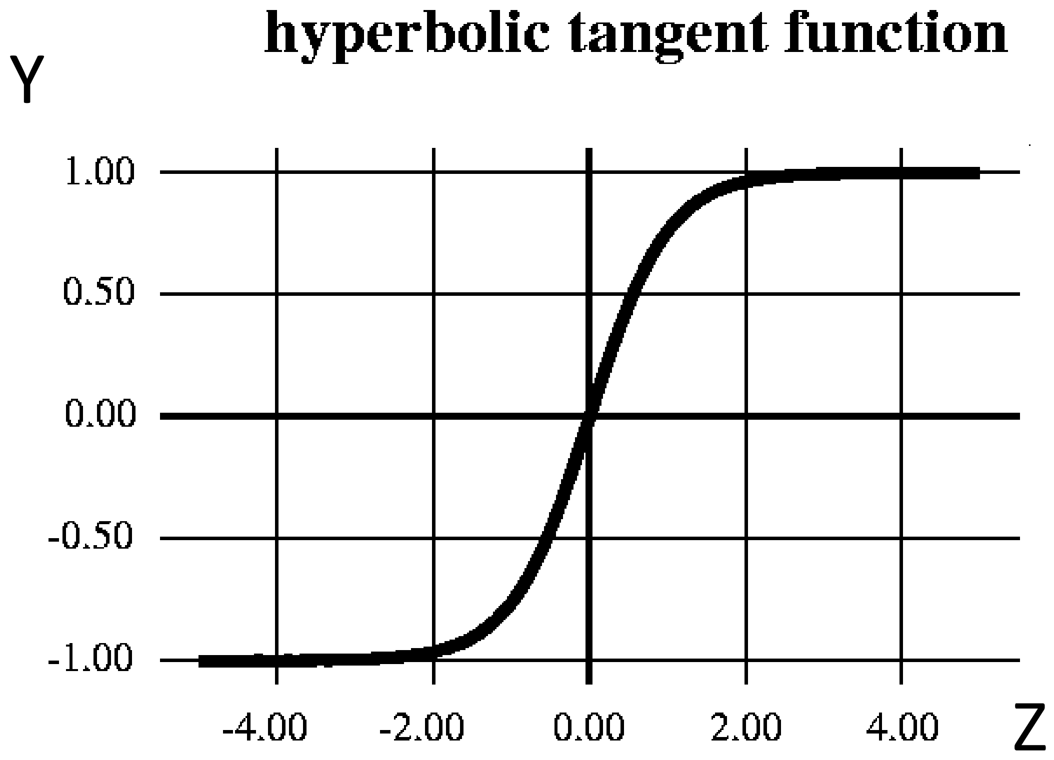
- **Equation : Y=** F(Z) = $\frac{1}{1+e^{-z}}$
- **Range :** (0 to 1)
- It gives rise to a problem of "**vanishing gradients**", since the Y values tend to respond very less to changes in X
- It saturate and kill gradients.



Tanh or hyperbolic tangent Activation Function

- **Equation : Y=** F(Z) = $\frac{1-e^{-2z}}{1+e^{-2z}}$

- **Range :** (-1 to 1)
- It also suffers vanishing gradient problem
- It saturate and kill gradients.

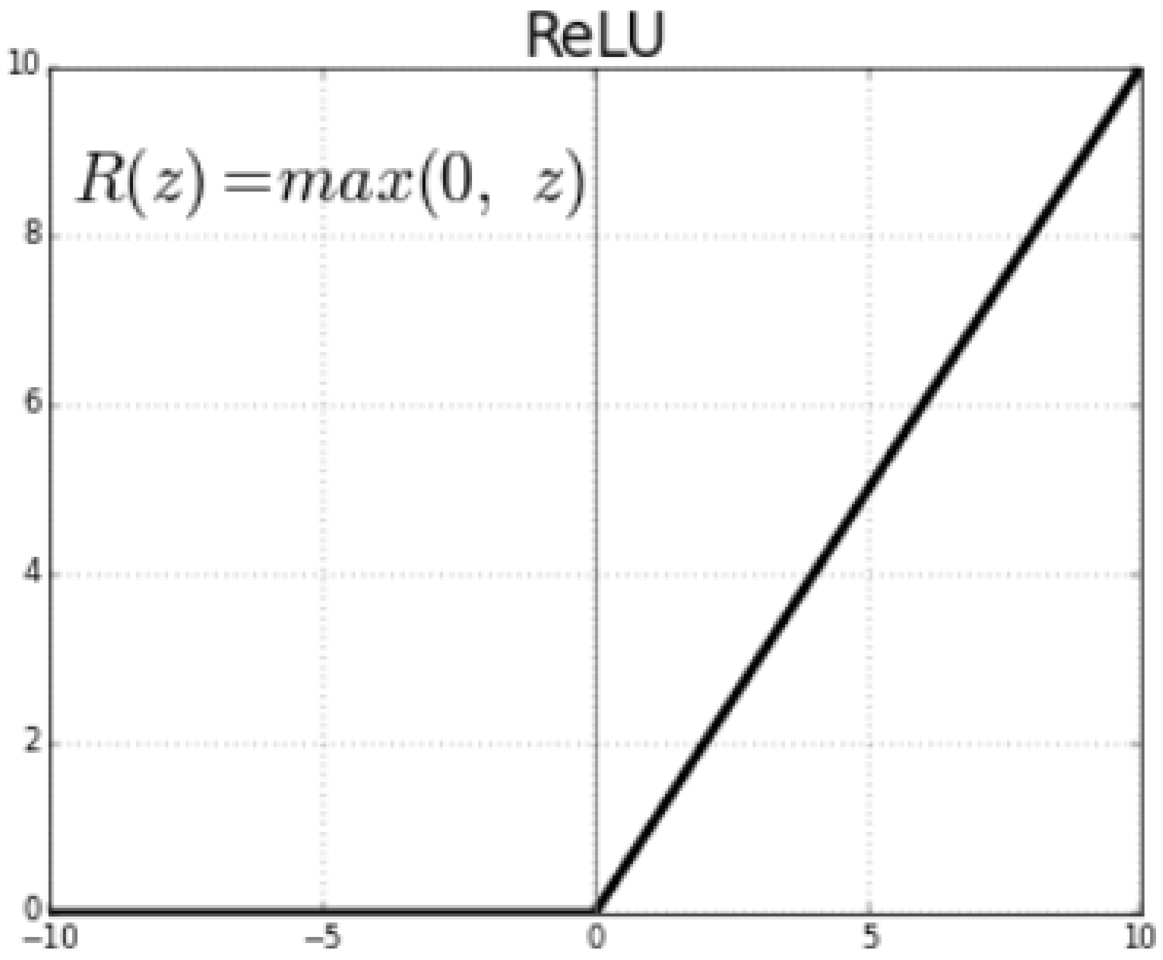


ReLU (Rectified Linear Unit)

- The ReLU is the most used activation function in the world right now

- **Equation : f(Z) = max(0,Z)**
- **Range :** (0 to infinity)

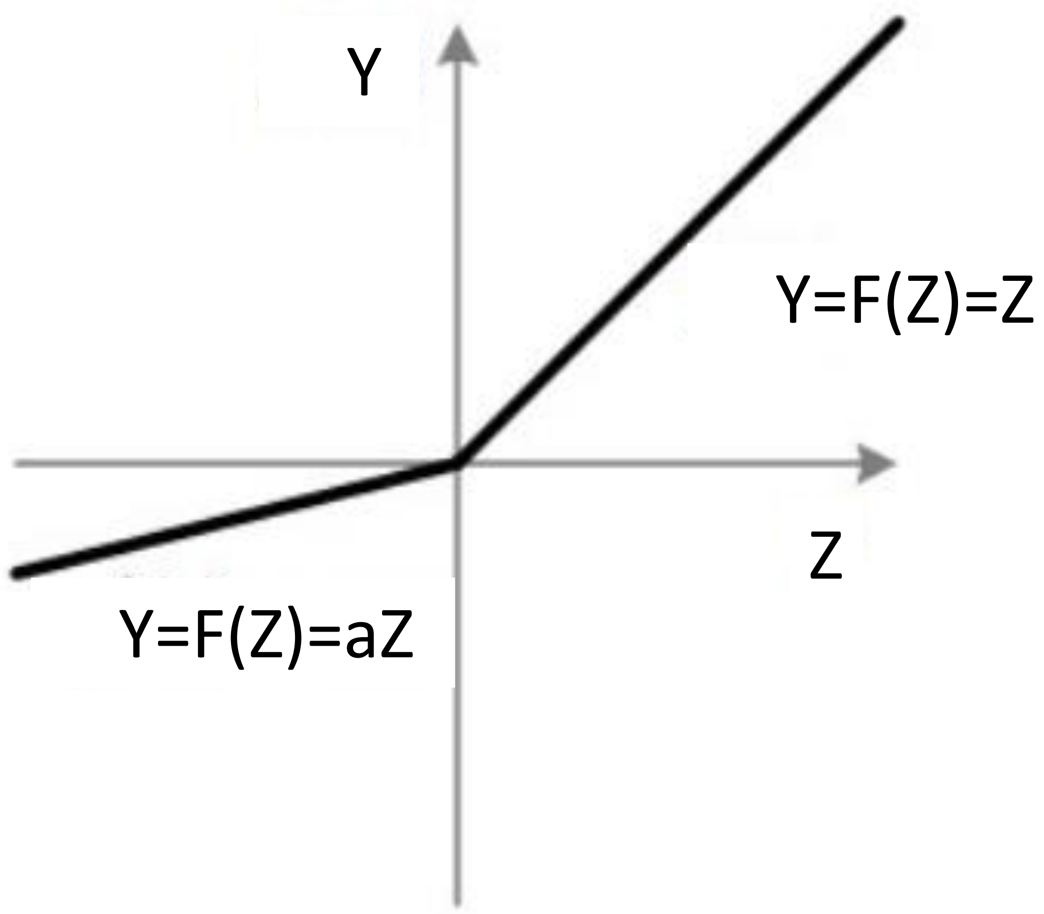
- The outputs are not zero centered similar to the sigmoid activation function
- When the gradient hits zero for the negative values, it does not converge towards the minima which will result in a dead neuron while back propagation



Leaky ReLU

- To solve the ReLU problem we have leaky ReLU

- **Equation :** f(x) = ax for x<0 and x for x>0
- **Range :** (0.01 to infinity)



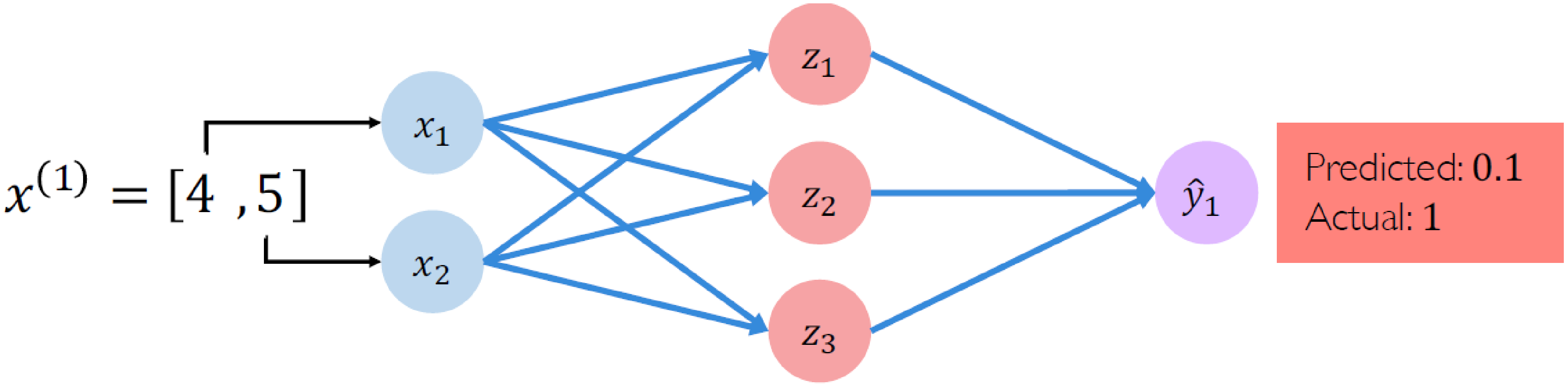
Softmax

- The softmax function is also a type of sigmoid function but it is very useful to handle classification problems having multiple classes.
- The softmax function is shown above, where z is a vector of the inputs to the output layer
- The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Loss Functions

The loss of our network measures the cost incurred from incorrect predictions

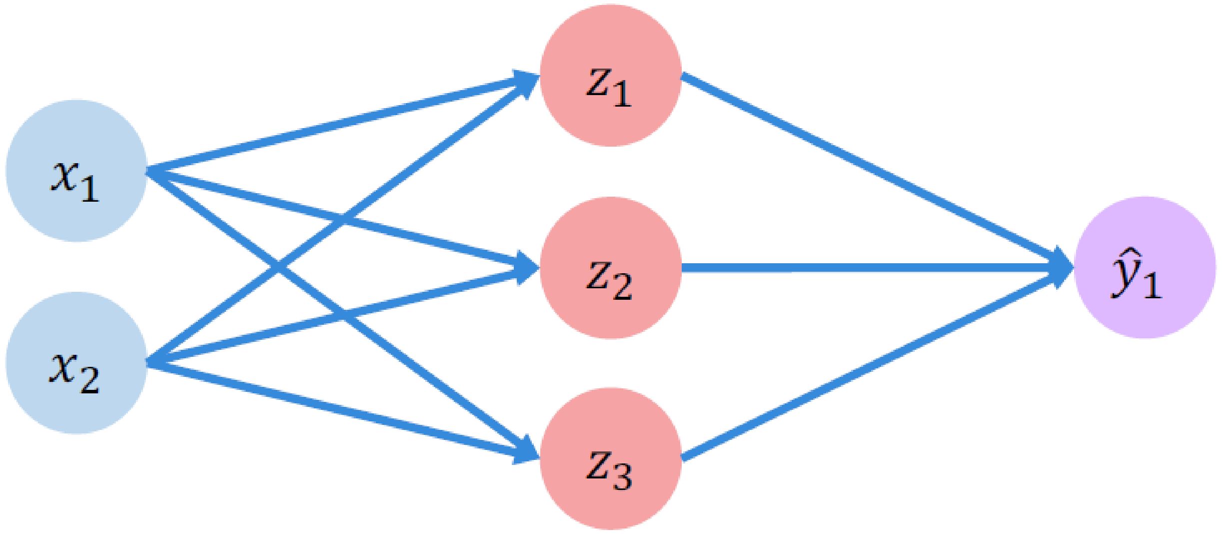


$$\mathcal{L}\big(\underbrace{f\big(x^{(i)}; \boldsymbol{W}\big)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}\big)$$

Mean Squared Error

Mean squared error loss can be used with regression models that output continuous real numbers

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$

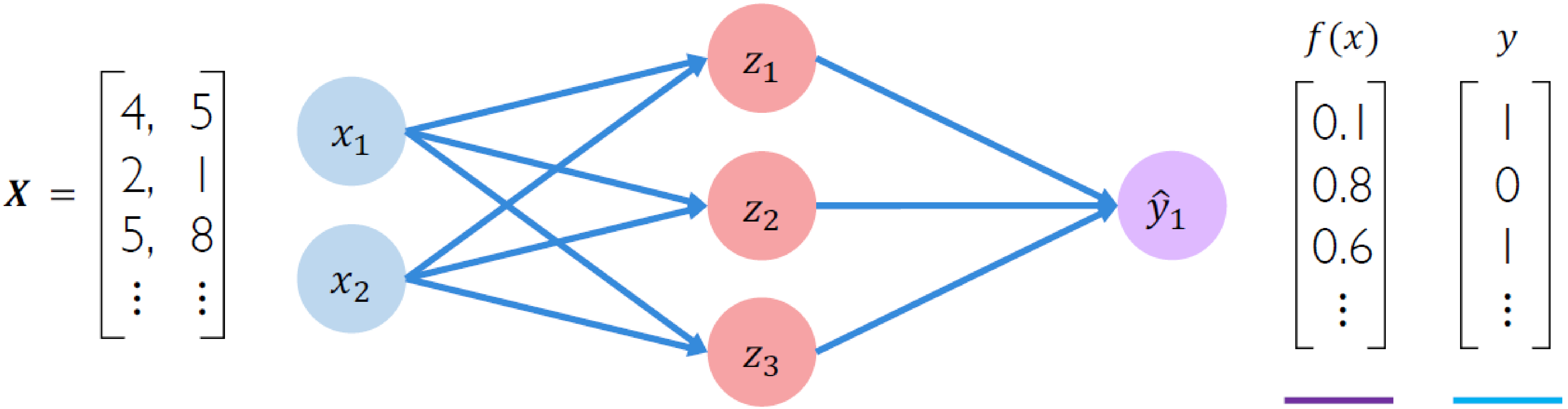


$$\underbrace{\begin{array}{cc} f(x) & y \\ \begin{bmatrix} 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix} & \begin{bmatrix} 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix} \end{array}}_{\text{Final Grades (percentage)}}$$

$$\boldsymbol{J}(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\Big(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big)^2$$

Cross Entropy

Cross entropy loss can be used with models that output a probability between 0 and 1



$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log\Big(\underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big) + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log\Big(1 - \underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big)$$

Loss Optimization

We want to find the network weights that achieve the lowest loss

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

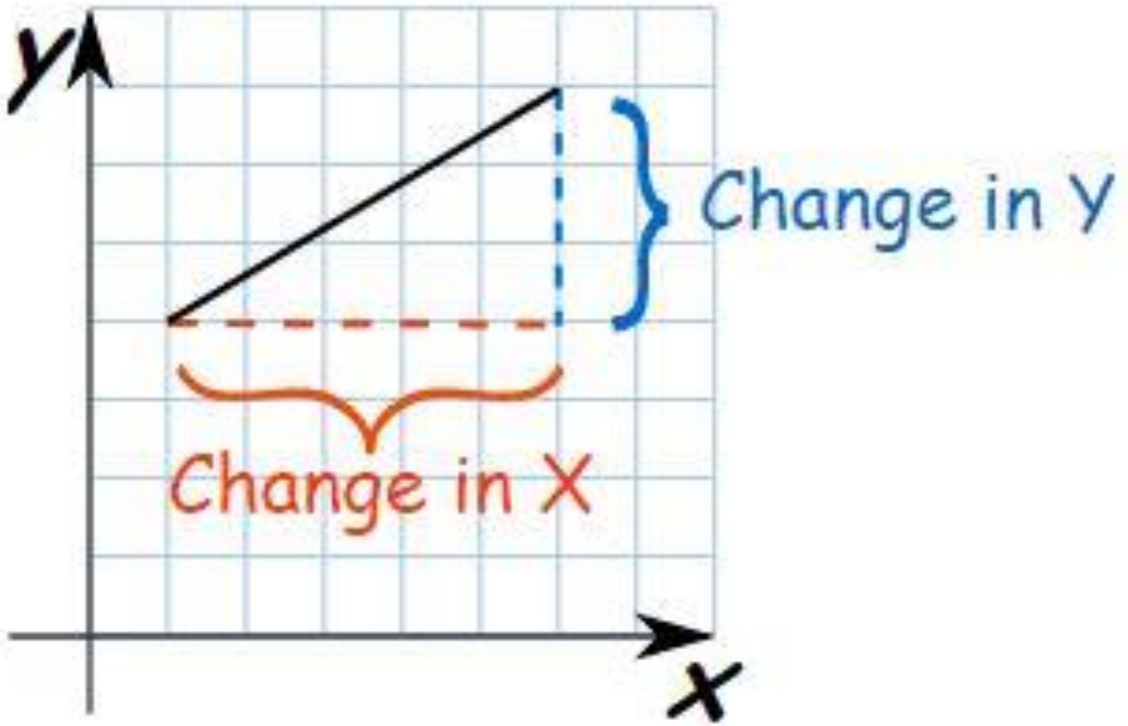
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

Remember:

$$\boldsymbol{W} = \left\{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \cdots\right\}$$

Gradient

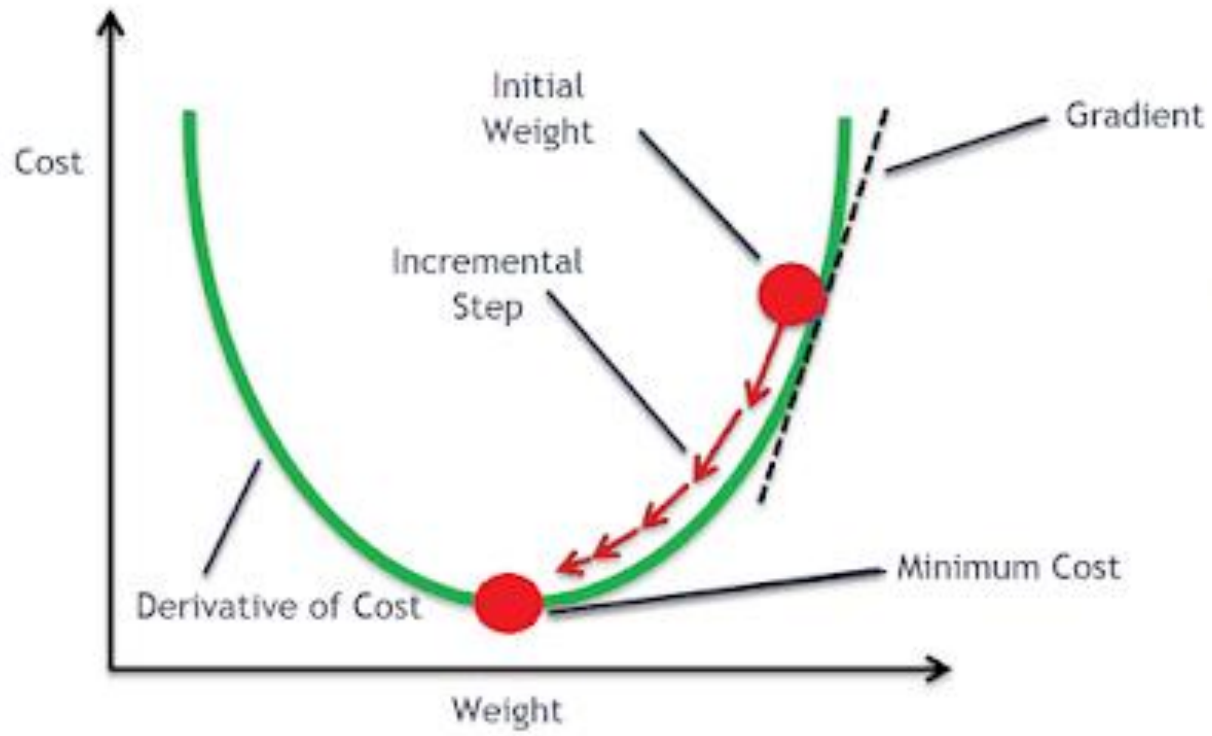
$$\textbf{Gradient} = \frac{\text{Change in Y}}{\text{Change in X}}$$

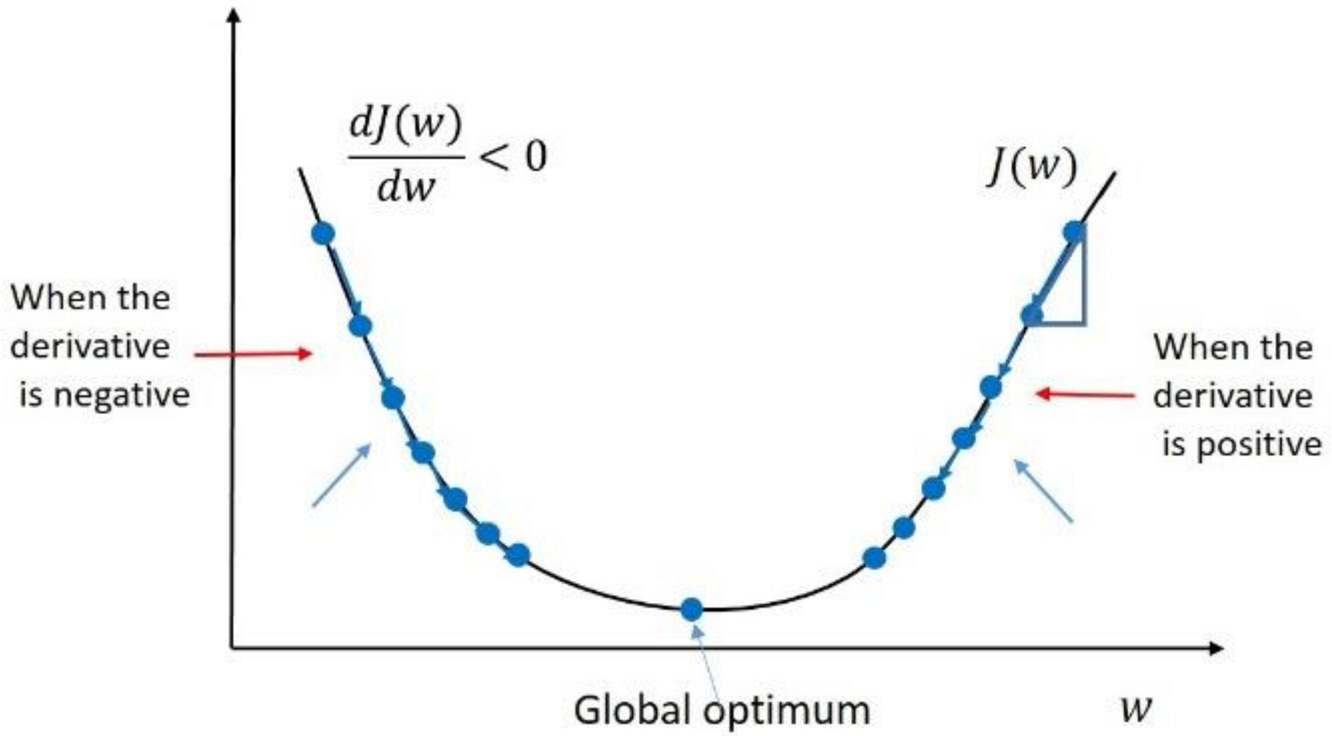


Gradient Descent

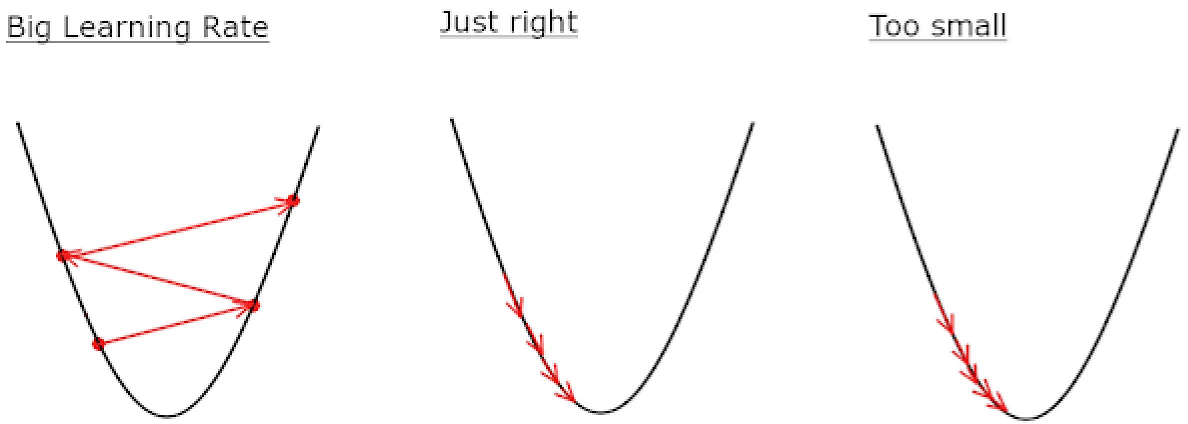
Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$

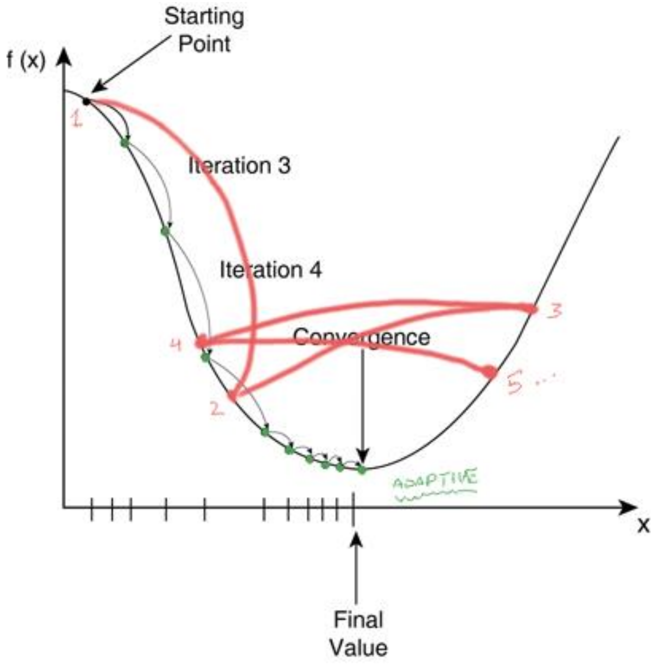
Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$

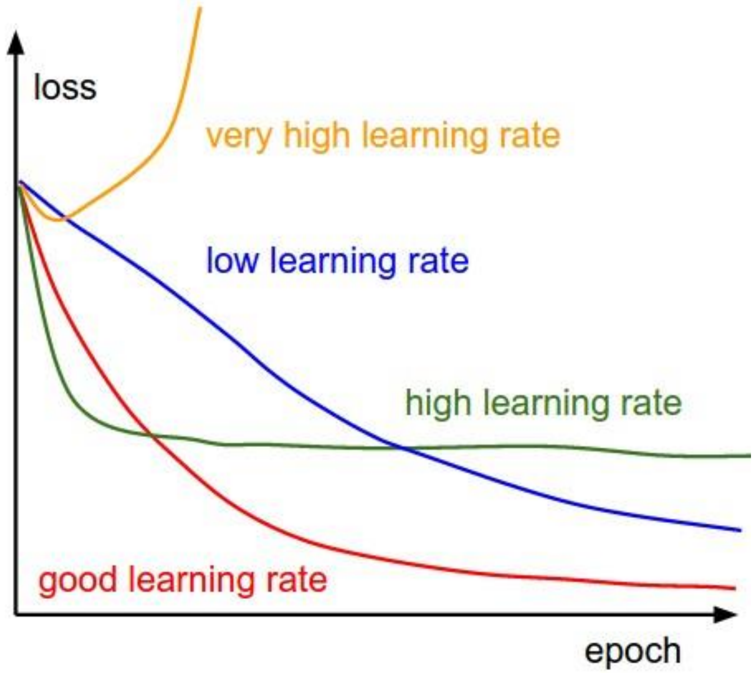




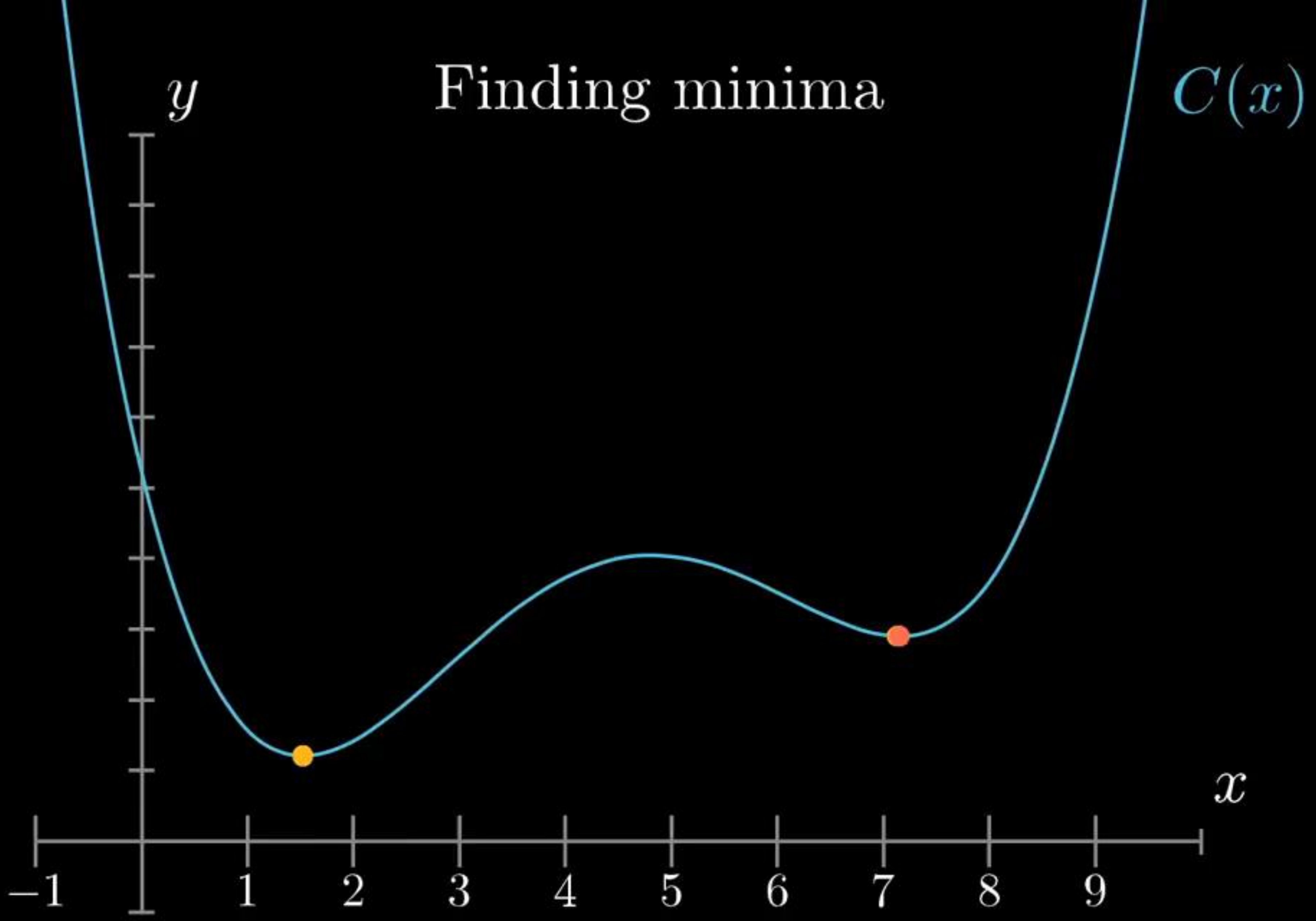
The Learning Rate (η)







Finding minima



Adam

- Adam stands for **Adaptive Moment Estimation.** Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter.

AdaDelta

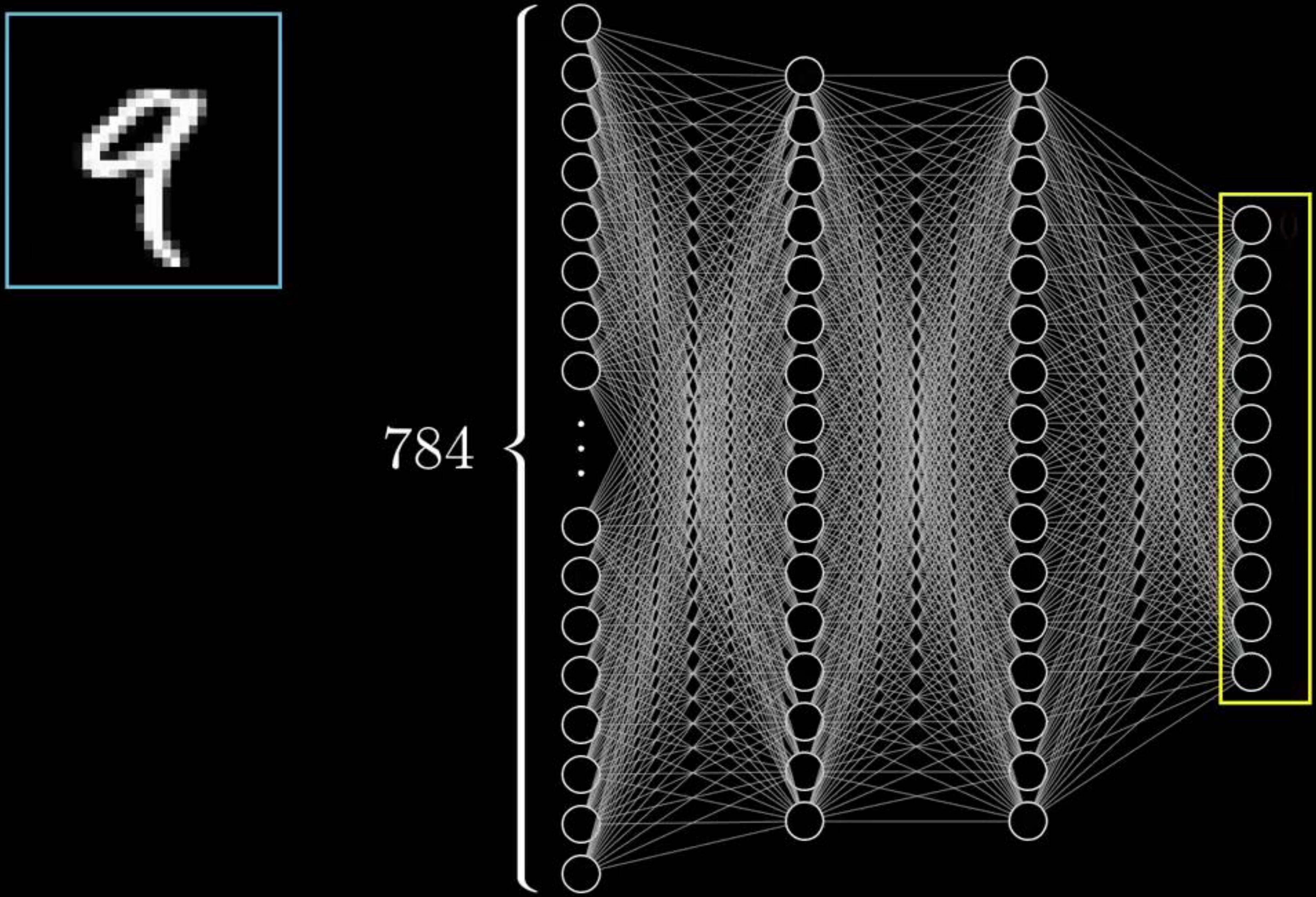
- It is an extension of **AdaGrad** which tends to remove the *decaying learning Rate* problem of it. Instead of accumulating all previous squared gradients, ***Adadelta*** limits the window of accumulated past gradients to some fixed size **w**.

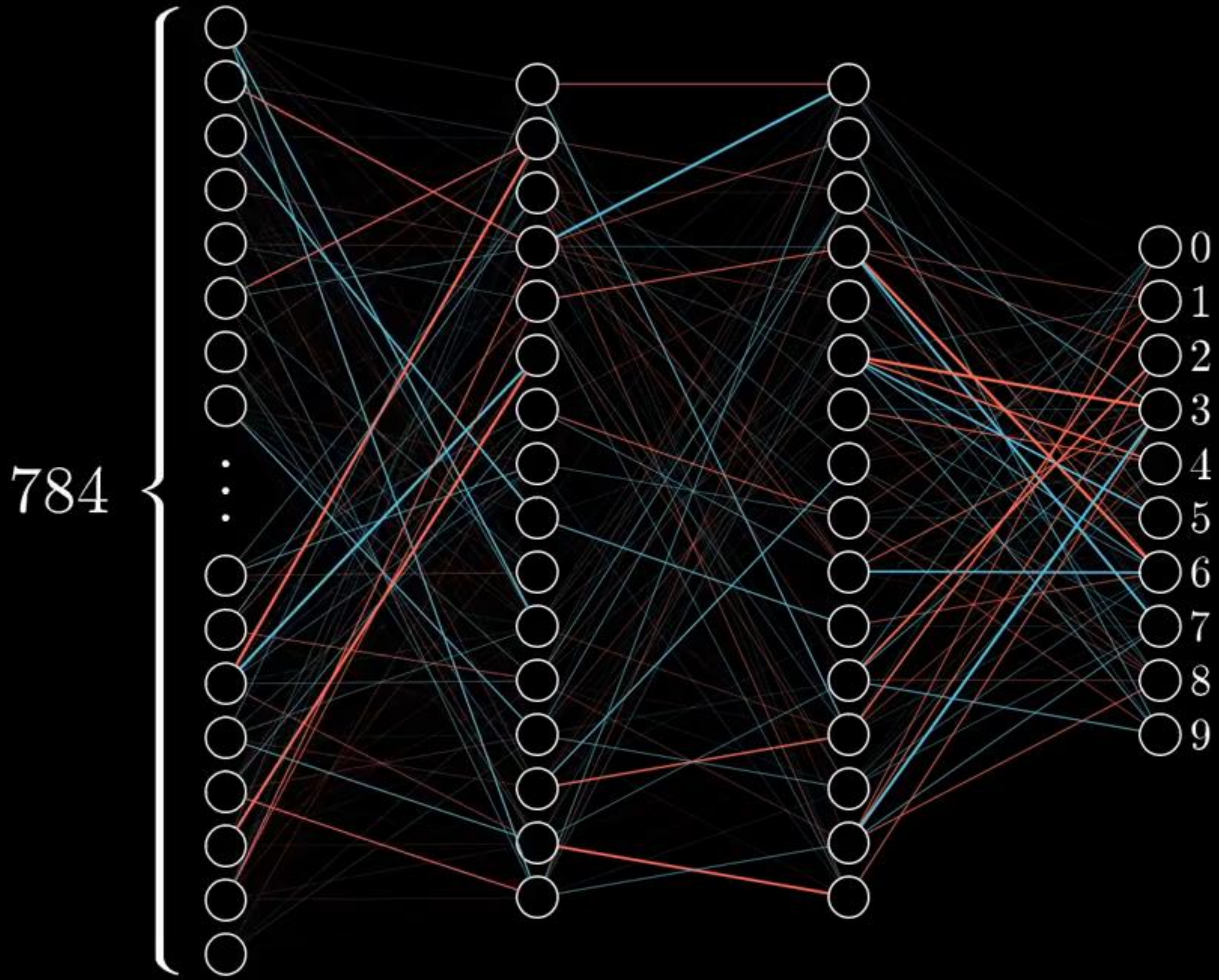
Adagrad

- It simply allows the learning Rate -**η** to **adapt** based on the parameters. So it makes big updates for infrequent parameters and small updates for frequent parameters. For this reason, it is well-suited for dealing with sparse data.

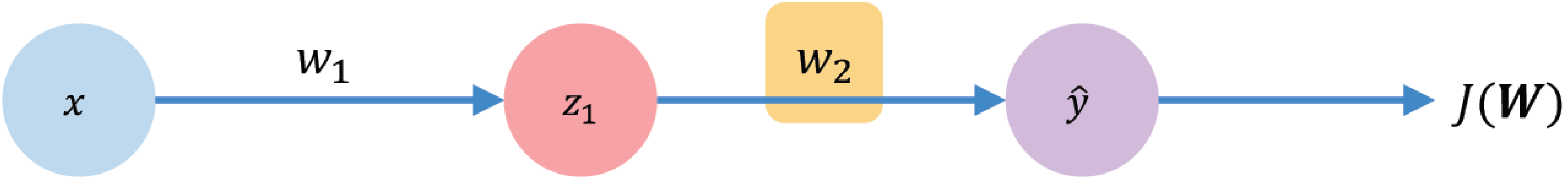
Gradient Vector and backpropagation

$$W = \begin{bmatrix} w1 \\ w2 \\ w3 \\ . \\ . \\ wn \end{bmatrix} \qquad -\eta\frac{\partial J(W)}{\partial W} = \begin{bmatrix} \Delta w1 \\ \Delta w2 \\ \Delta w3 \\ . \\ . \\ \Delta wn \end{bmatrix}$$


784

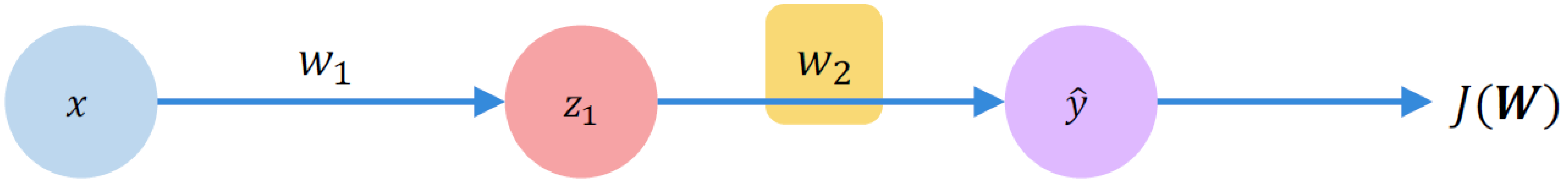

784
0
1
2
3
4
5
6
7
8
9

Computing Gradients: Backpropagation



How does a small change in one weight (ex. $\mathbf{w_2}$) affect the final loss $\boldsymbol{J(W)}$?

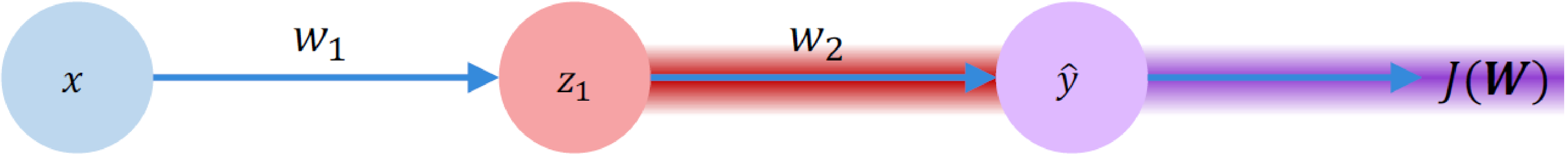
Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial \boxed{w_2}} =$$

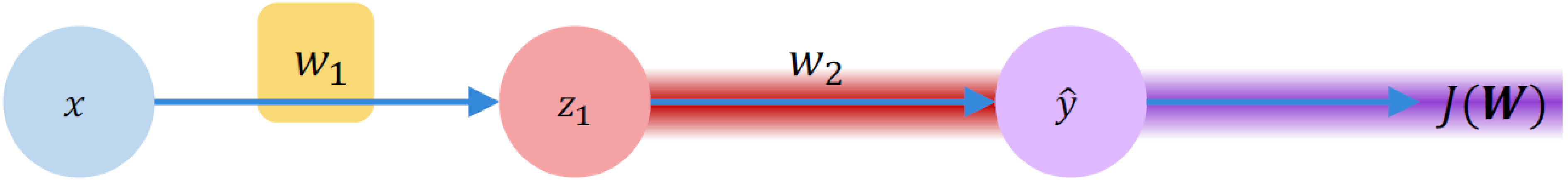
Let's use the chain rule!

Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \underbrace{\frac{\partial J(\boldsymbol{W})}{\partial \hat{y}}} * \underbrace{\frac{\partial \hat{y}}{\partial w_2}}$$

Computing Gradients: Backpropagation

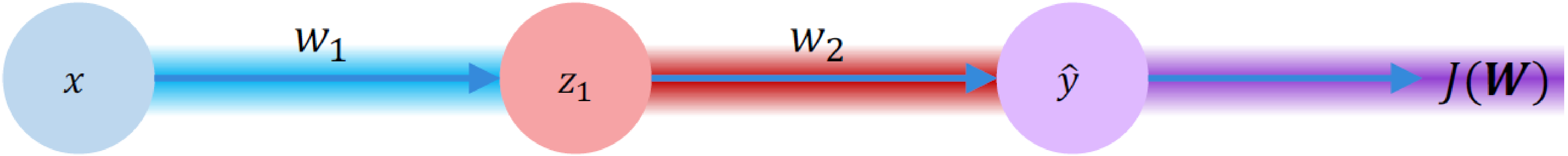


$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule!

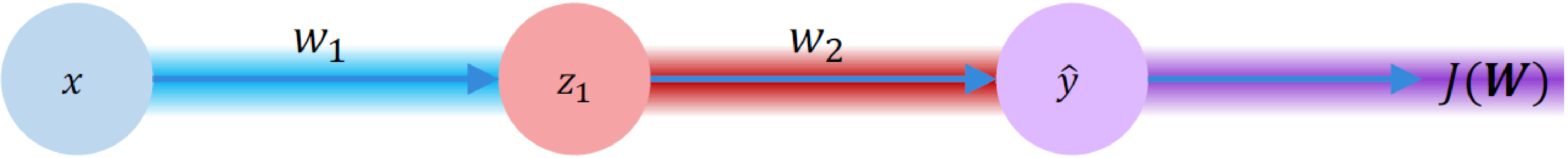
Apply chain rule!

Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

References

- 3BLUE1BROWN SERIES S3 (https://www.youtube.com/watch?v=aircAruvnKk&list=PL_h2yd2CGtBHEKwEH5iqTZH85wLS-eUzv&index=1)
- MIT 6.S191 Introduction to Deep Learning (introtodeeplearning.com)

(Some slides are taken here)