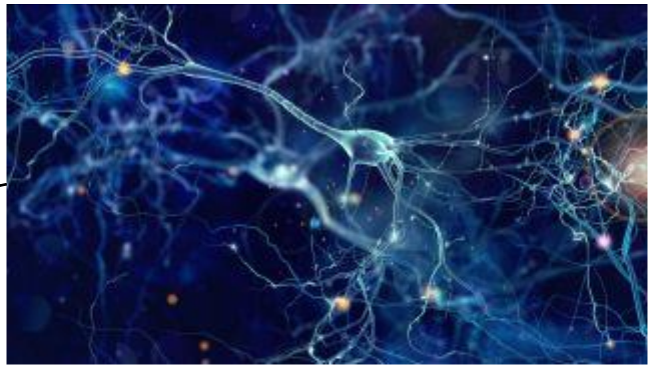
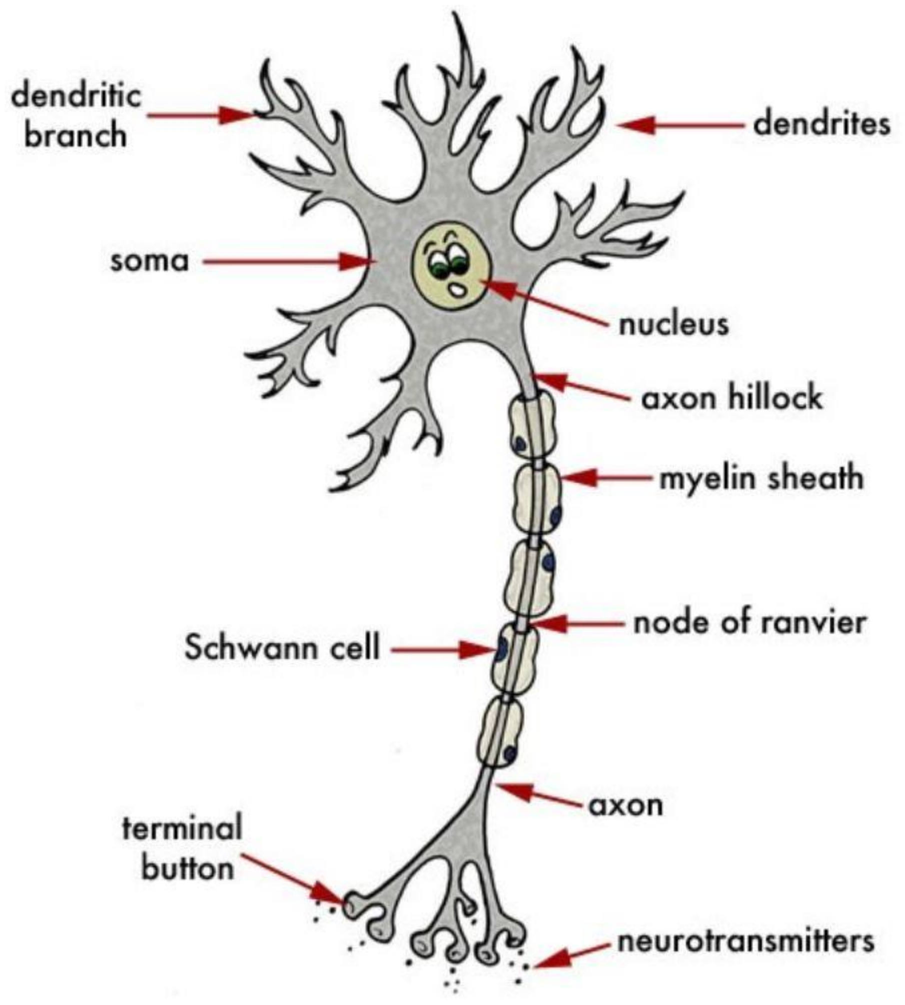
# Introduction to Artificial Neural Networks (ANN) – Part 1



**By: Dr Ng Gee Wah**
**ESP 3201**

https://scx2.b-cdn.net/gfx/news/hires/2018/1-neuralnetwork.jpg

# Neural Network In Our Brains



A nerve cell ( neuron)

**A typical biological neuron comprised of:**

- A cell body
- Hair-like dendrites (input channels)
- Axon: output cable

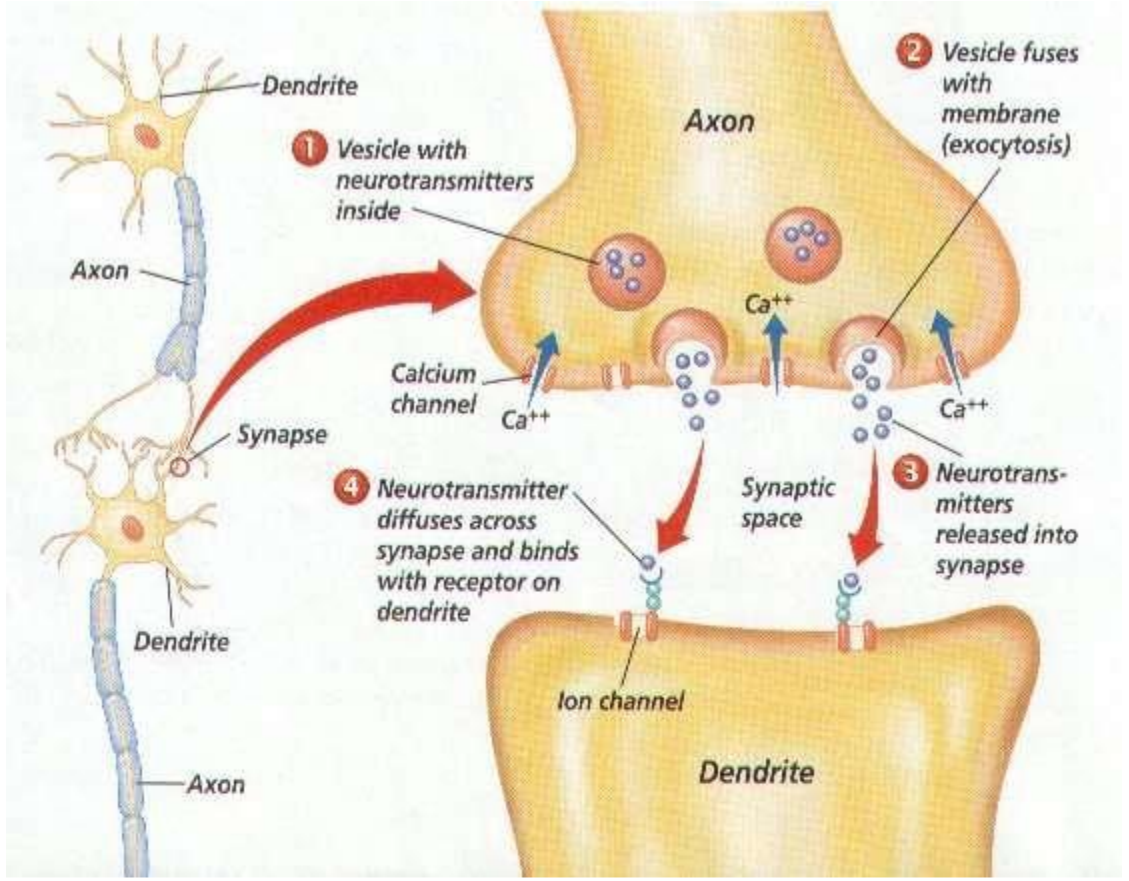
- ➢ Many such nerve cells are arranged together in our brain to form a network of nerves.
- ➢ They pass electrical impulses i.e the excitation from one neuron to the other.
- ➢ The dendrites receive the impulse from the terminal button or synapse of an adjoining neuron.
- ➢ Dendrites carry the impulse to the nucleus of the nerve cell which is also called as soma. The electrical impulse is processed and then passed on to the axon.
- ➢ The axon then carries the impulse from the soma to the synapse.
- ➢ Next, the synapse passes the impulse to dendrites of the second neuron.
- ➢ There are many of this connectivity in the human brain.

https://miro.medium.com/max/1050/1*6MjLJe2Uev7zkfCDw1Wzfg.jpeg

# Synaptic Transmission of Neurons

Synaptic transmission is the process whereby one neuron communicates with other neurons, at a synapse.

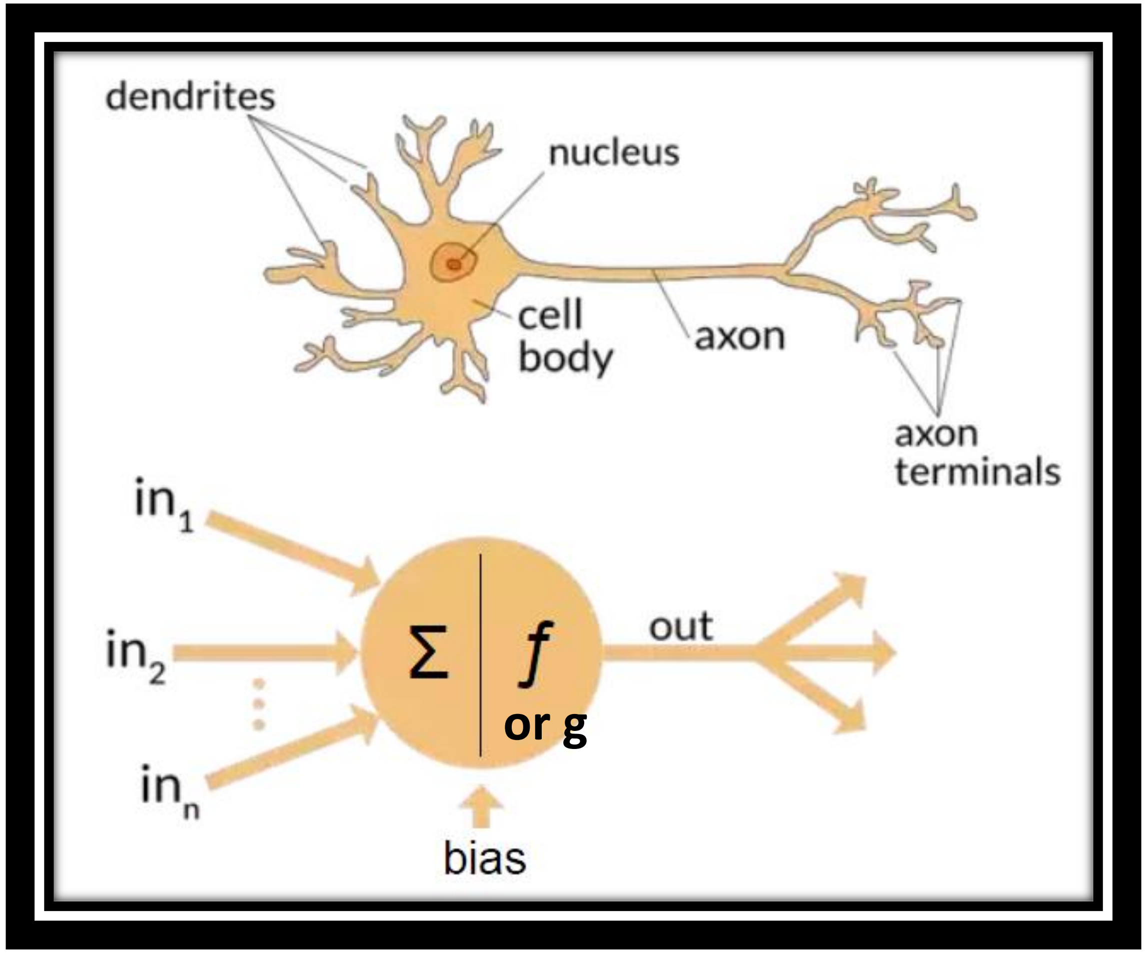
A typical neuron has a cell body (soma), branching processes specialized to receive incoming signals (dendrites), and a single process (axon) that carries electrical signals away from the neuron toward other neurons. Electrical signals carried by axons are action potentials . Axons often have thousands of terminal branches, each ending as synaptic terminal. At the synaptic terminal, the action potential is converted into a chemical message which, in turn, interacts with the recipient neuron.


Dendrite
Axon
Synapse
Dendrite
Axon
Axon
1 Vesicle with neurotransmitters inside
2 Vesicle fuses with membrane (exocytosis)
Ca++
Calcium channel
Ca++
Ca++
3 Neurotransmitters released into synapse
4 Neurotransmitter diffuses across synapse and binds with receptor on dendrite
Synaptic space
Ion channel
Dendrite

https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.researchgate.net%2Ffigure%2FFigure117-Transmission-between-two-neuron-across-the-synapses_fig14_320264164&psig=AOvVaw2IdhxyEb6Py6yaT2Ka0e1E&ust=1590213478779000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCMCj5_3kxukCFQAAAAAdAAAAABAD

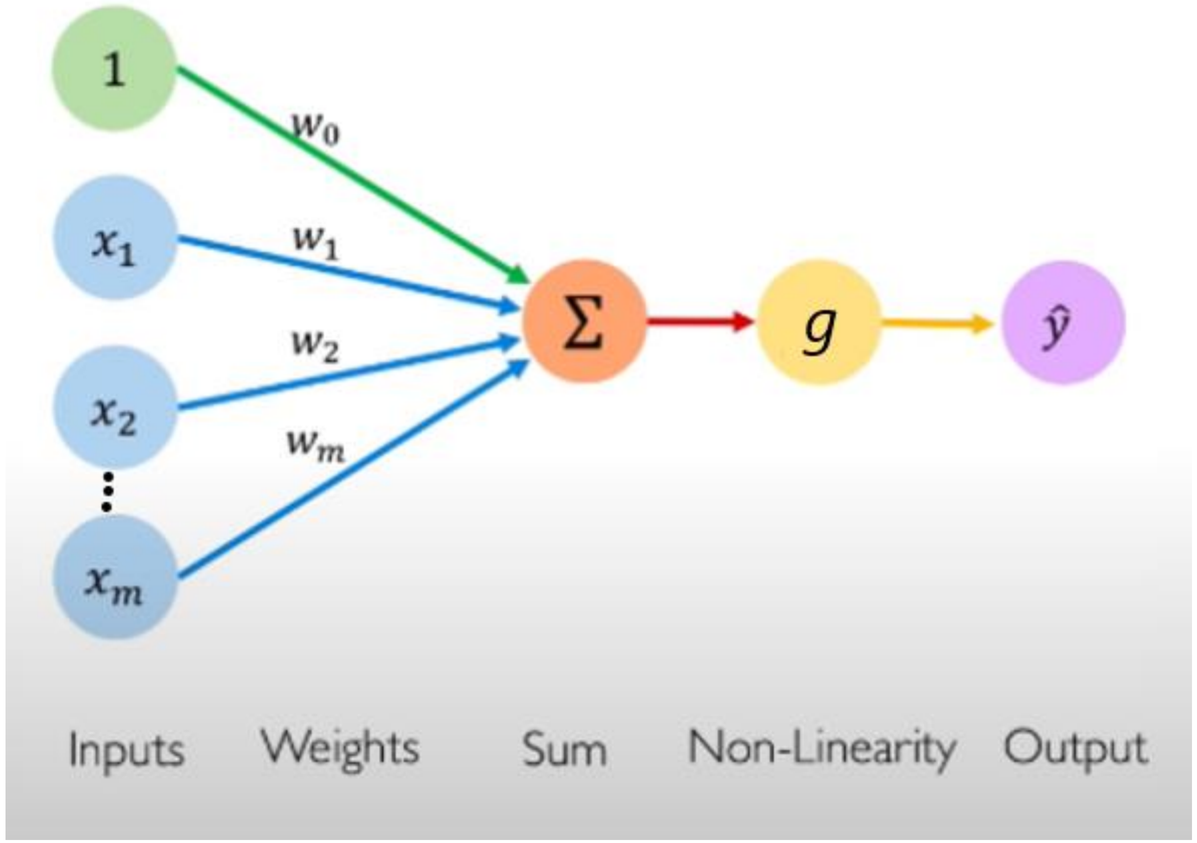
# Artificial and Biological Neural Networks

| Biological Neural Network | Artificial Neural Network |
|---|---|
| Neuron | Processor Element (Unit) |
| Dendrite | Inputs |
| Cell Body | Activation Function |
| Axons | Outputs |
| Synapses | Weights |



https://miro.medium.com/max/1220/1*SJPacPhP4KDEB1AdhOFy_Q.pn

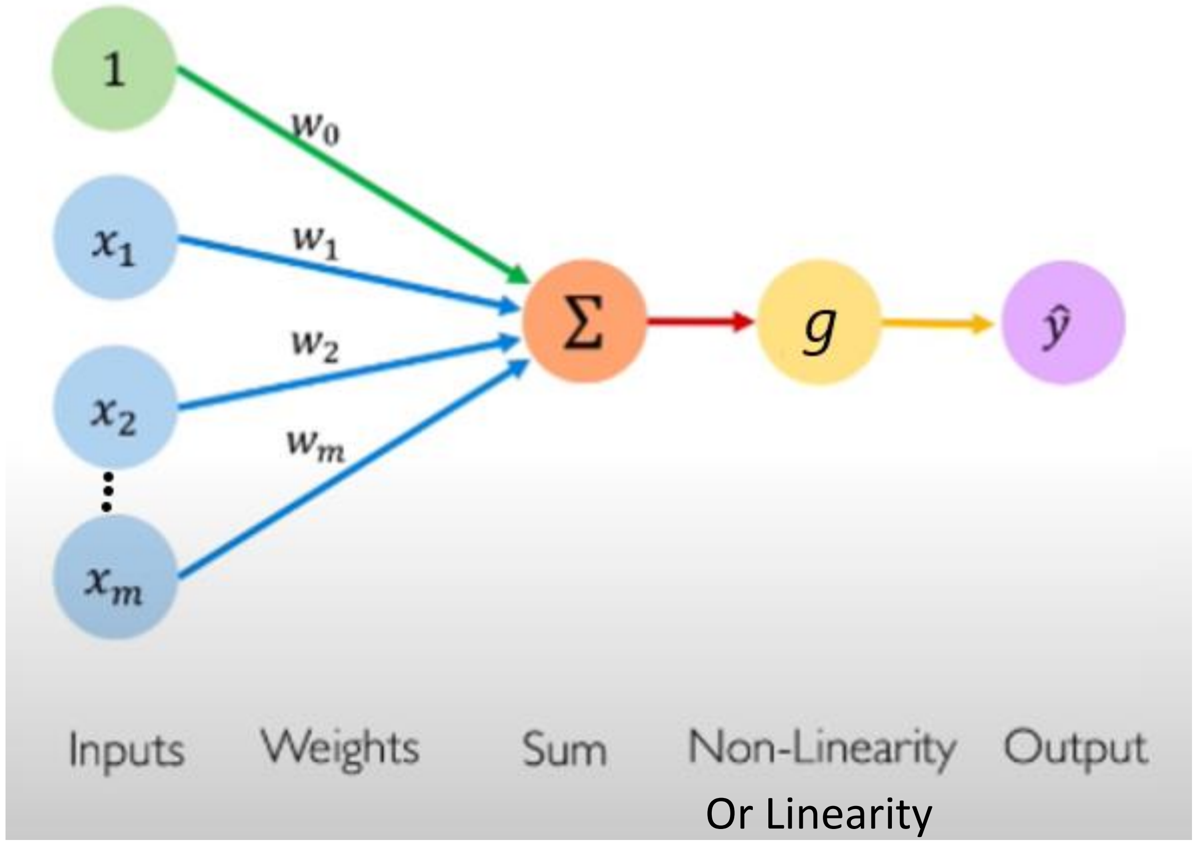
## A Single Artificial Neuron
## The Perceptron: Forward Propagation



$$\underbrace{\hat{y}}_{\text{Output}} = \underbrace{g}_{\text{Activation function (Nonlinear in general)}}\left(\underbrace{\omega_0}_{\text{Bias}} + \underbrace{\sum_{i=1}^{m} x_i\omega_i}_{\text{Linear combination of inputs}}\right)$$

Reference: https://www.youtube.com/watch?v=njKP3FqW3Sk

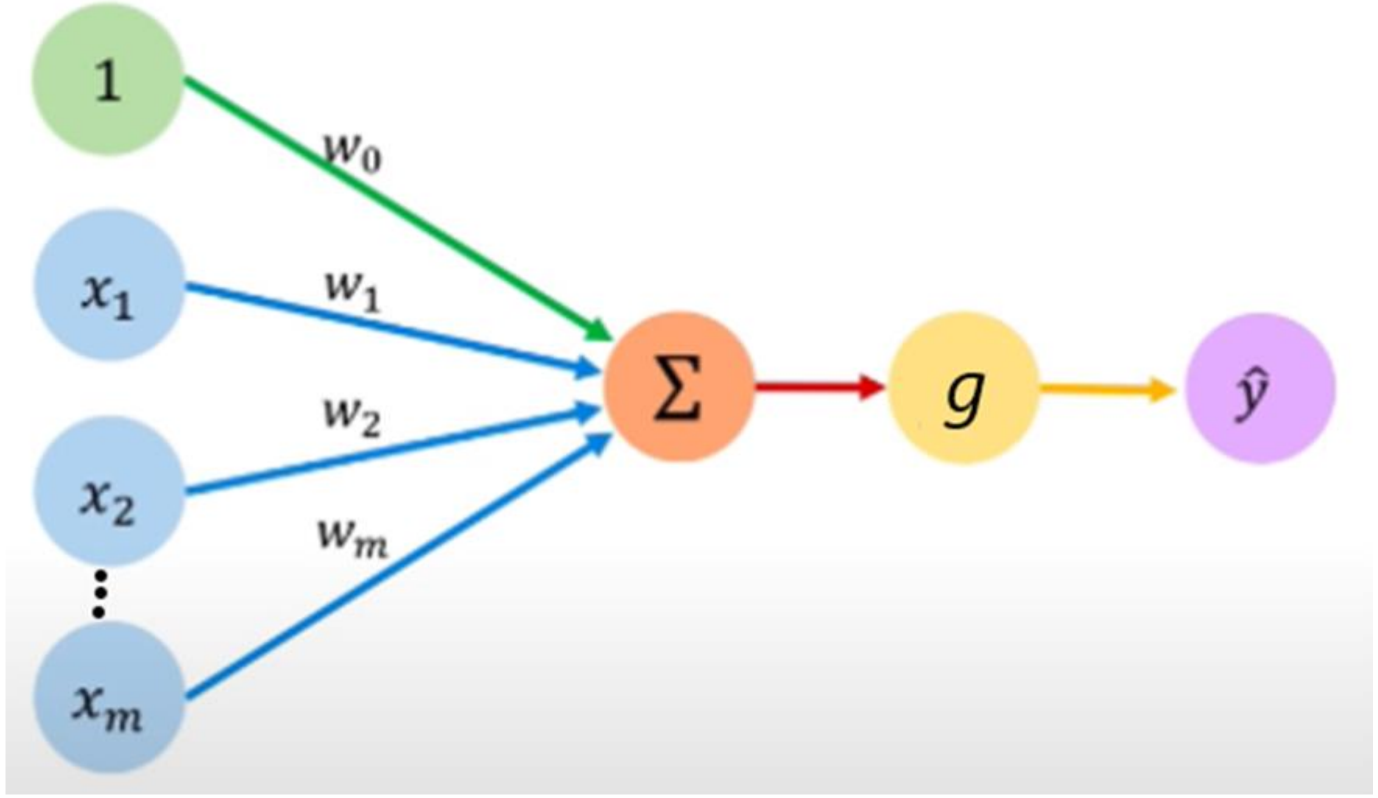
# A Single Artificial Neuron
# The Perceptron: Forward Propagation



Question: Express the forward propagation equation when m=1 and when g is

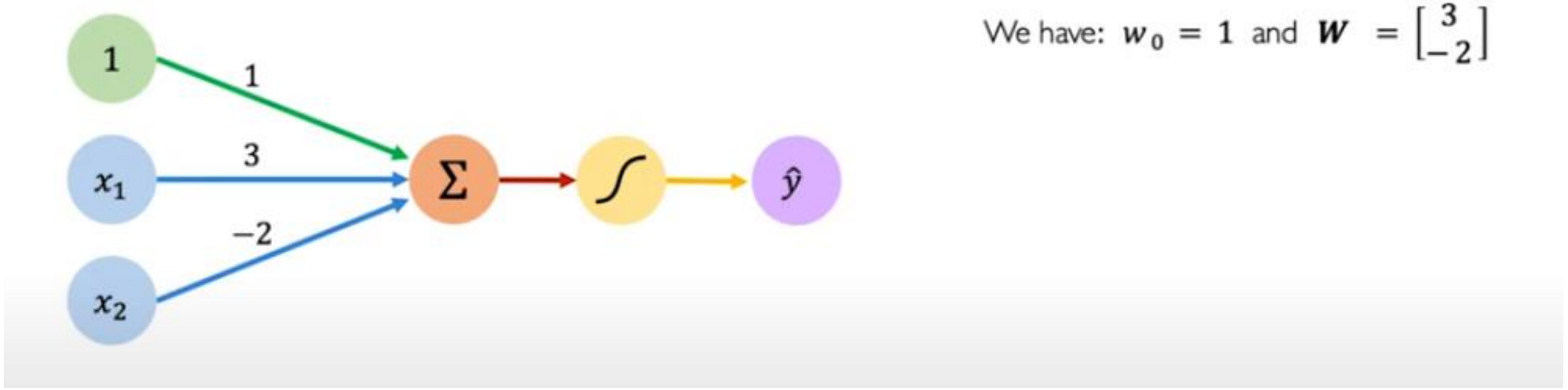
i) A linear function; say g is a proportional gain value of 1.
ii) A non linear function; say g is a sigmoid function.

## The Simplified Perceptron Output Equation



$$\hat{y} = g(w_0 + X^T W)$$

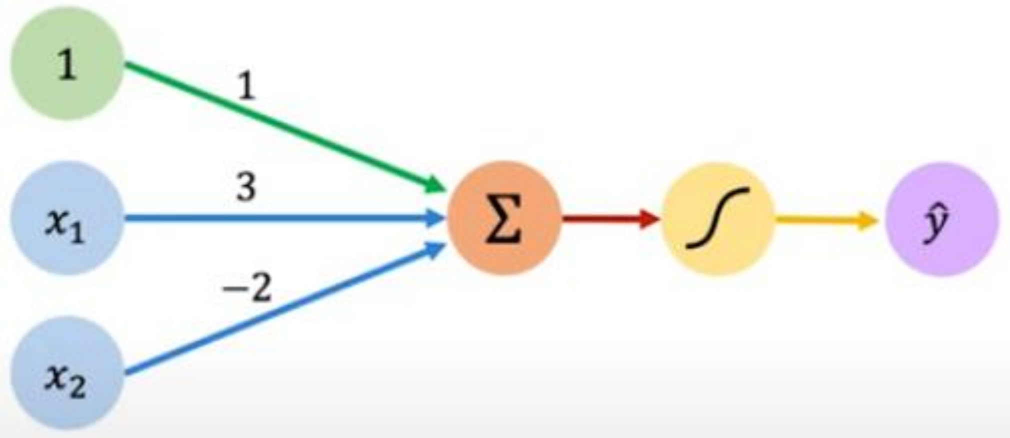
# Exercise case 1



We have: $\boldsymbol{w}_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

1. This network only have two inputs.  What is the input? (answer: $x_1$ and $x_2$)
2. Given the weights value and g as a linear function of say gain 1, what is the output of the neuron equation?
3. Find the output of the neuron if the input $[x_1, x_2] = [-1, 2]$.

# Exercise case 2



We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

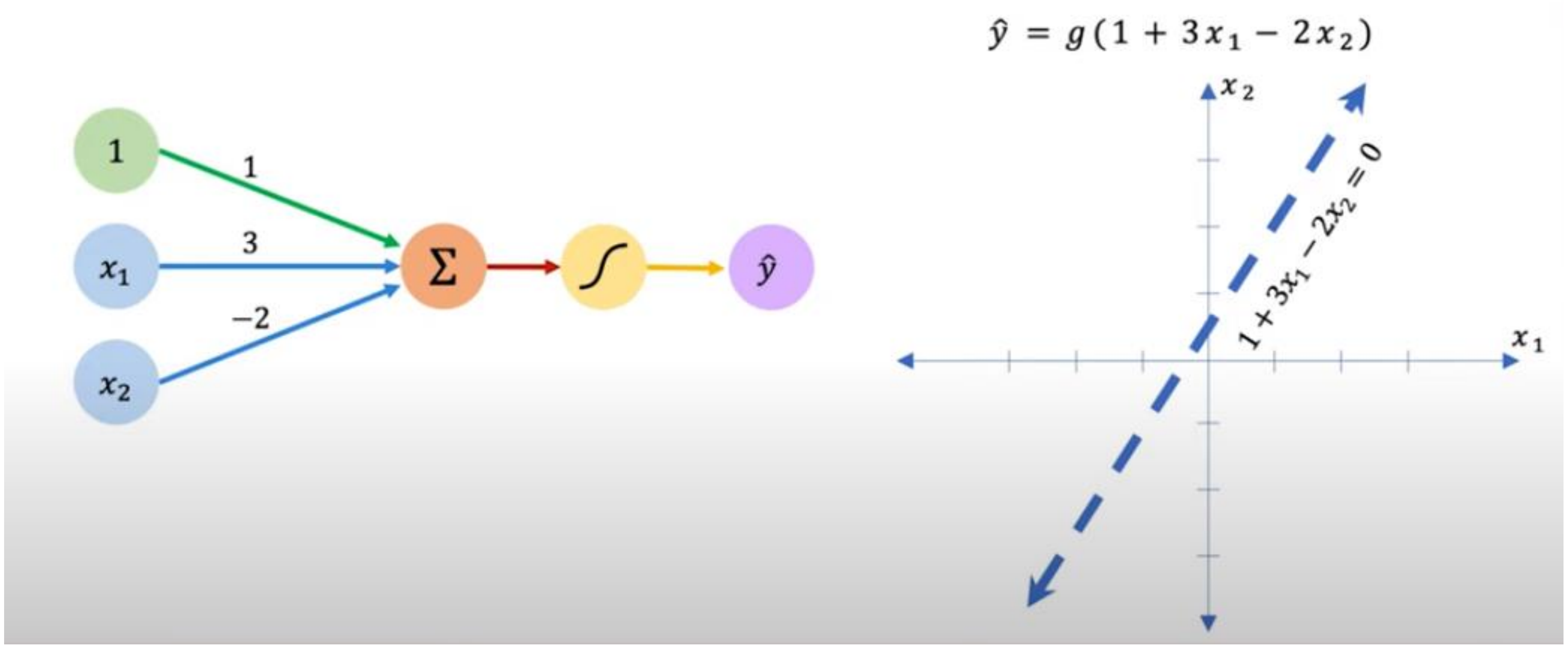
$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$

$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

$$\hat{y} = g(\underbrace{1 + 3x_1 - 2x_2}_{\text{This is just a line in 2D!}})$$

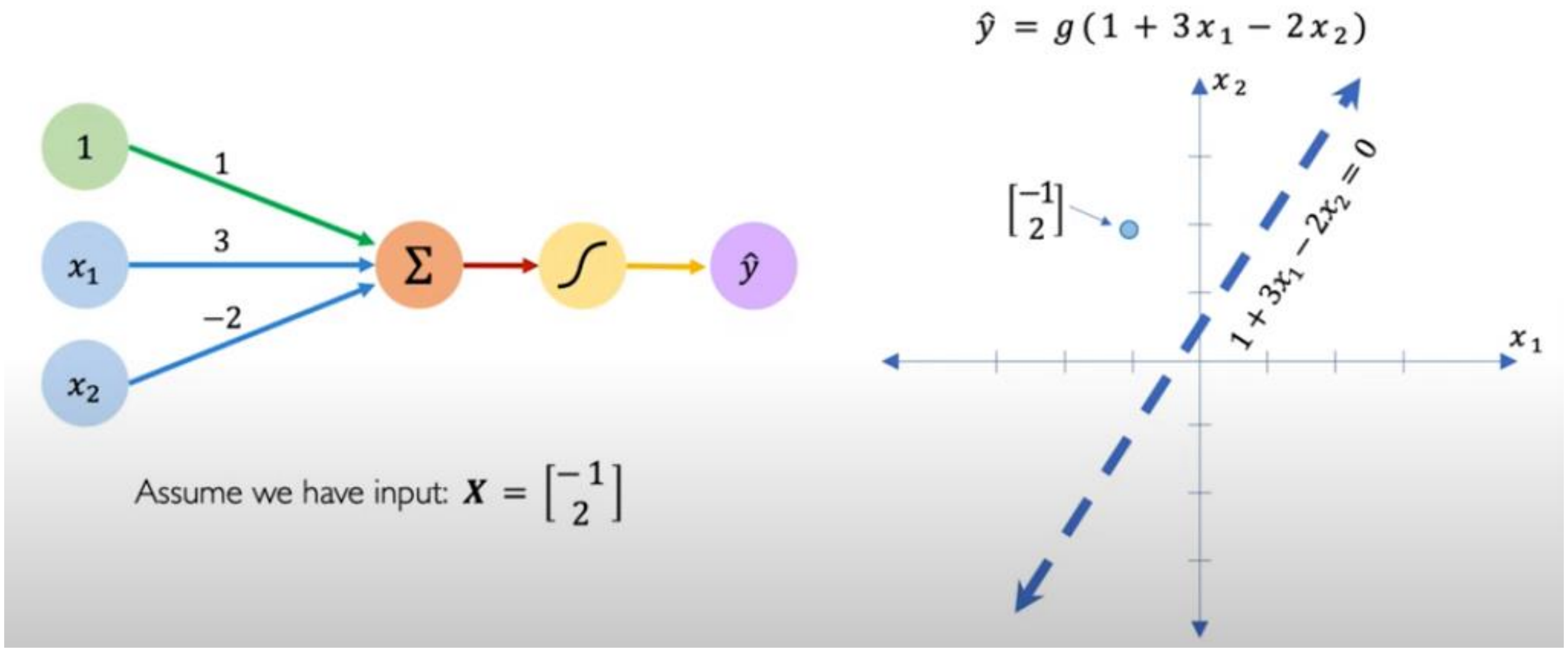
g is a general nonlinear function in most neuron network design and applications

## Exercise case 2



This is a straight line in the space. This line form the decision boundary.

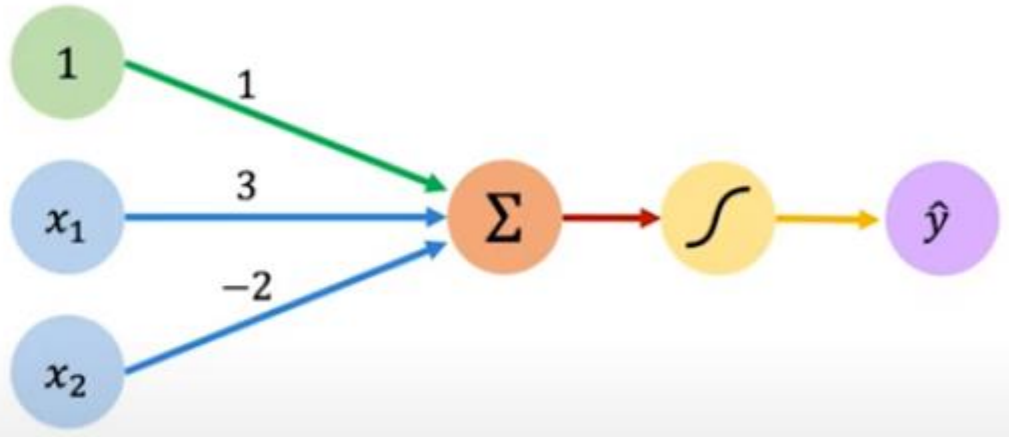
# Exercise case 2



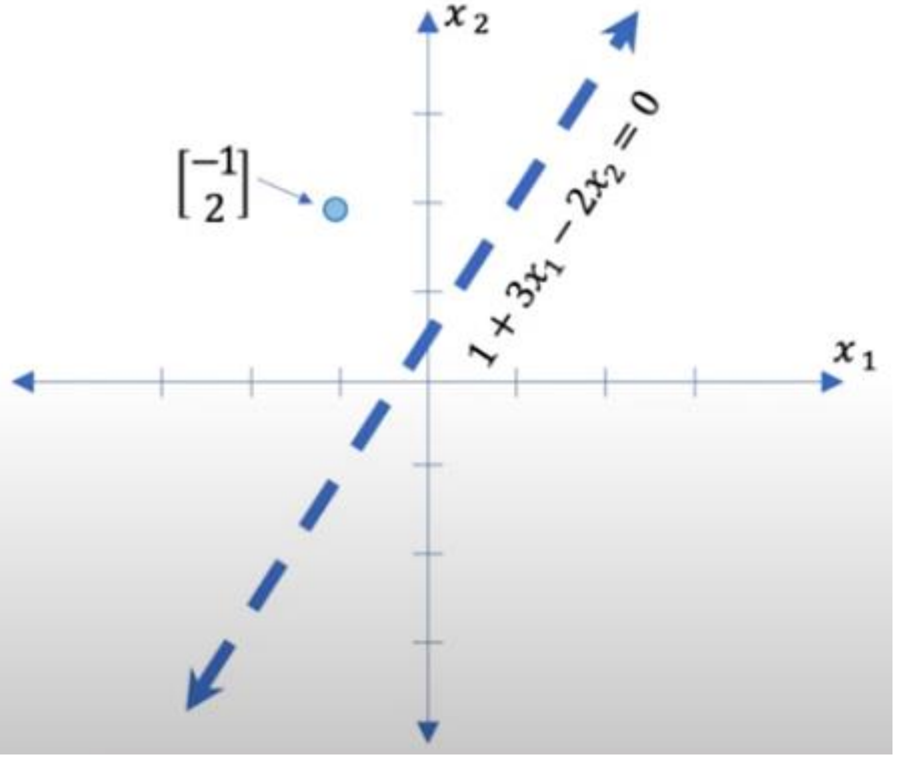
$\hat{y} = g(1 + 3x_1 - 2x_2)$
1
1
$x_1$
3
$x_2$
$-2$
$\Sigma$
$\hat{y}$
Assume we have input: $\boldsymbol{X} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$
$x_2$
$x_1$
$\begin{bmatrix} -1 \\ 2 \end{bmatrix}$
$1 + 3x_1 - 2x_2 = 0$

For $[x_1, x_2] = [-1, 2]$; the output lie somewhere in the space.

# Exercise case 2



$$\hat{y} = g(1 + 3x_1 - 2x_2)$$

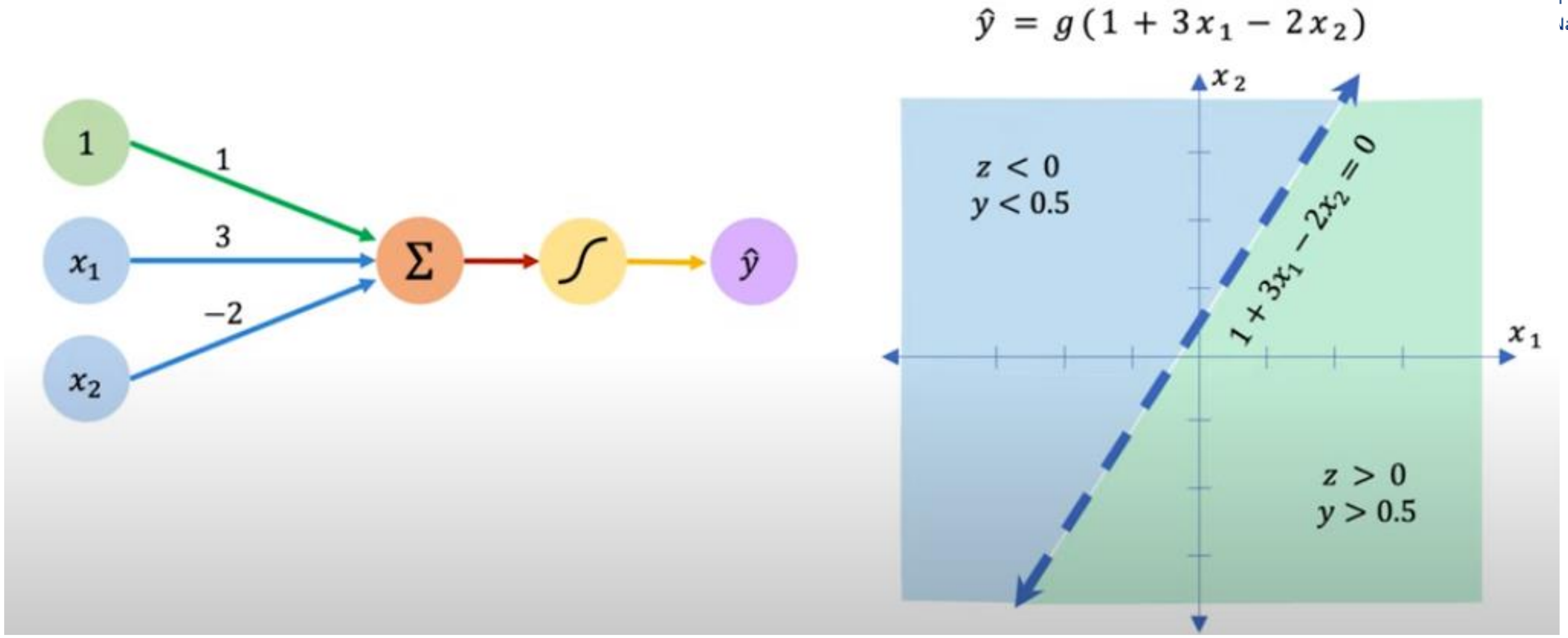


Assume we have input: $\boldsymbol{X} = \begin{bmatrix}-1\\2\end{bmatrix}$

$$\hat{y} = g\big(1 + (3*-1) - (2*2)\big)$$
$$= g(-6) \approx 0.002$$

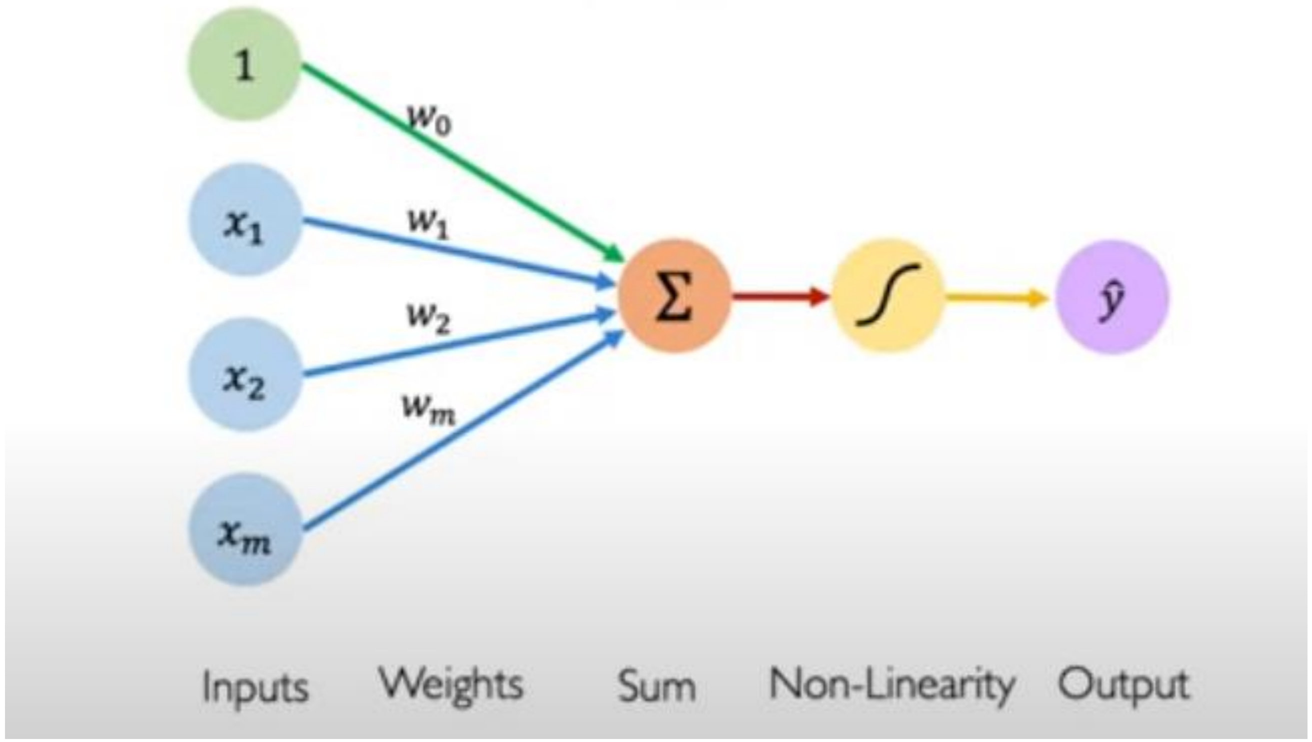
Assuming g is the sigmoid function then you get the output of 0.002.

# Exercise case 2



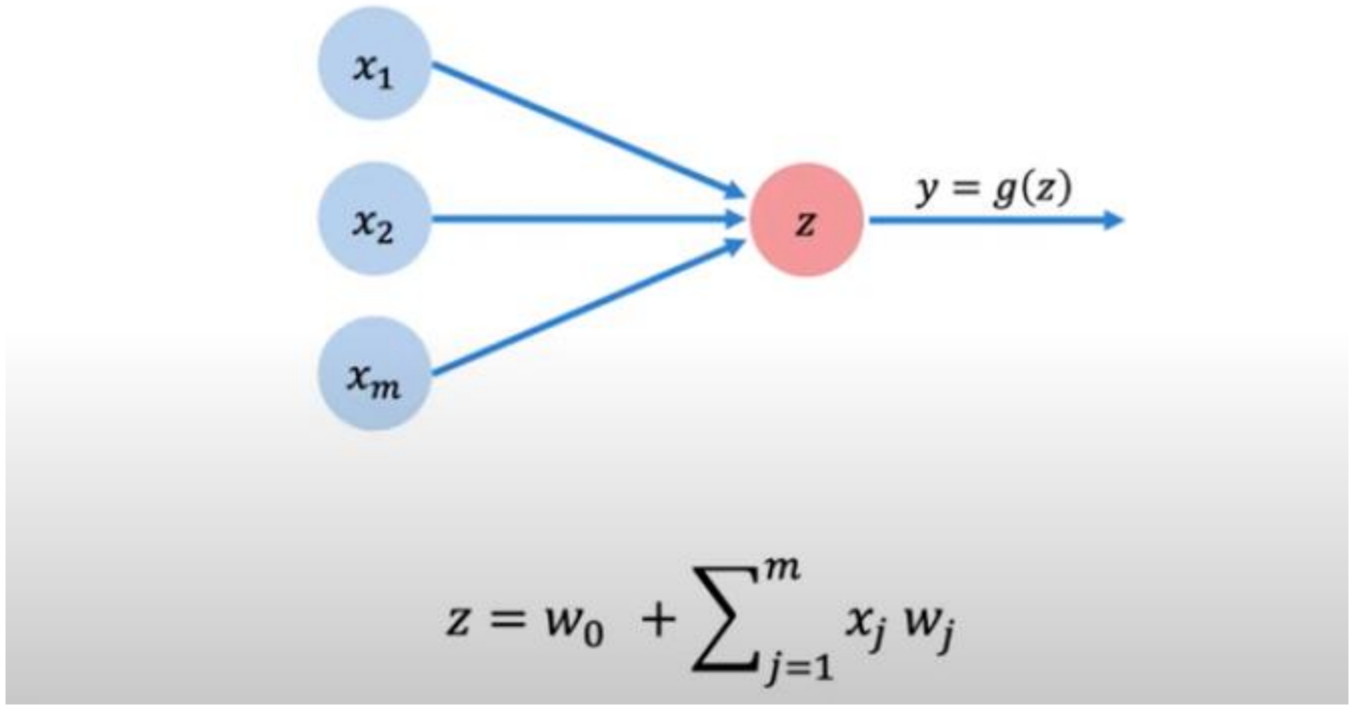
# Revisit: The Perceptron Equation

$$\hat{y} = g\left( w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$



1. Dot product of the input and the weight $X^T.W$
2. Add the bias term $w_0$
3. Apply the function. In this case the function is a nonlinear function such as sigmoid function

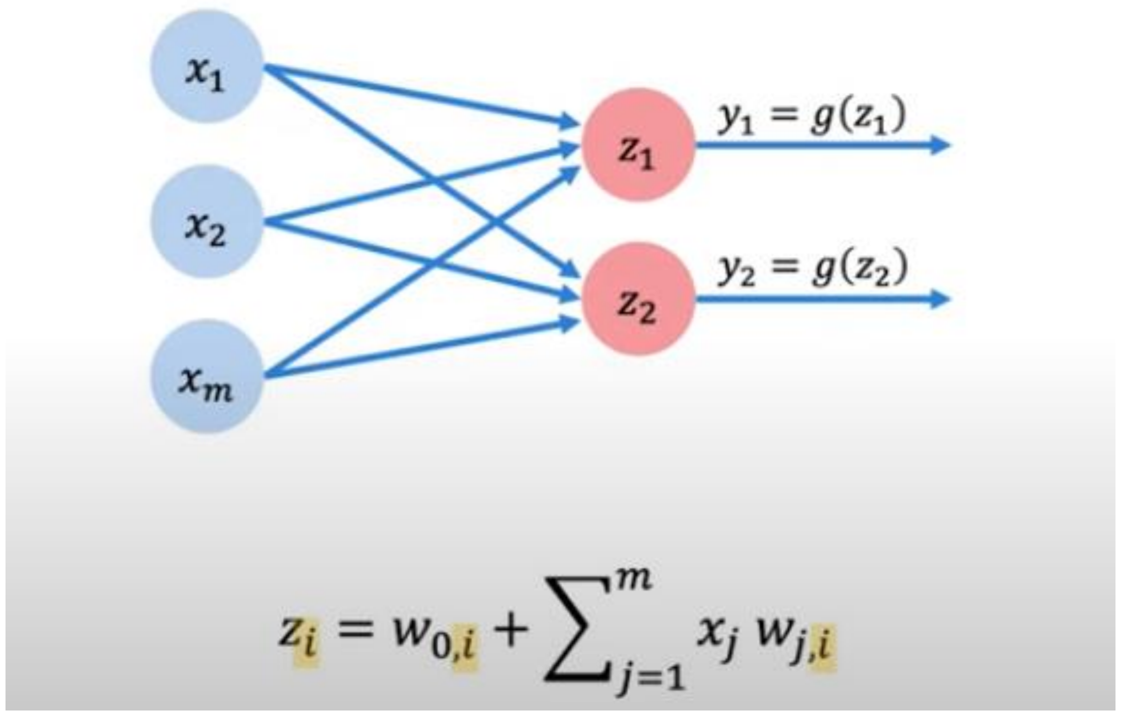
# The Perceptron: Simplified



$$z = w_0 + \sum_{j=1}^{m} x_j \, w_j$$

1. We can further simplified the equation and diagram as above. The bias term $w_0$
2. Apply the function g(z). In this case the function, g, is a nonlinear function such as sigmoid function

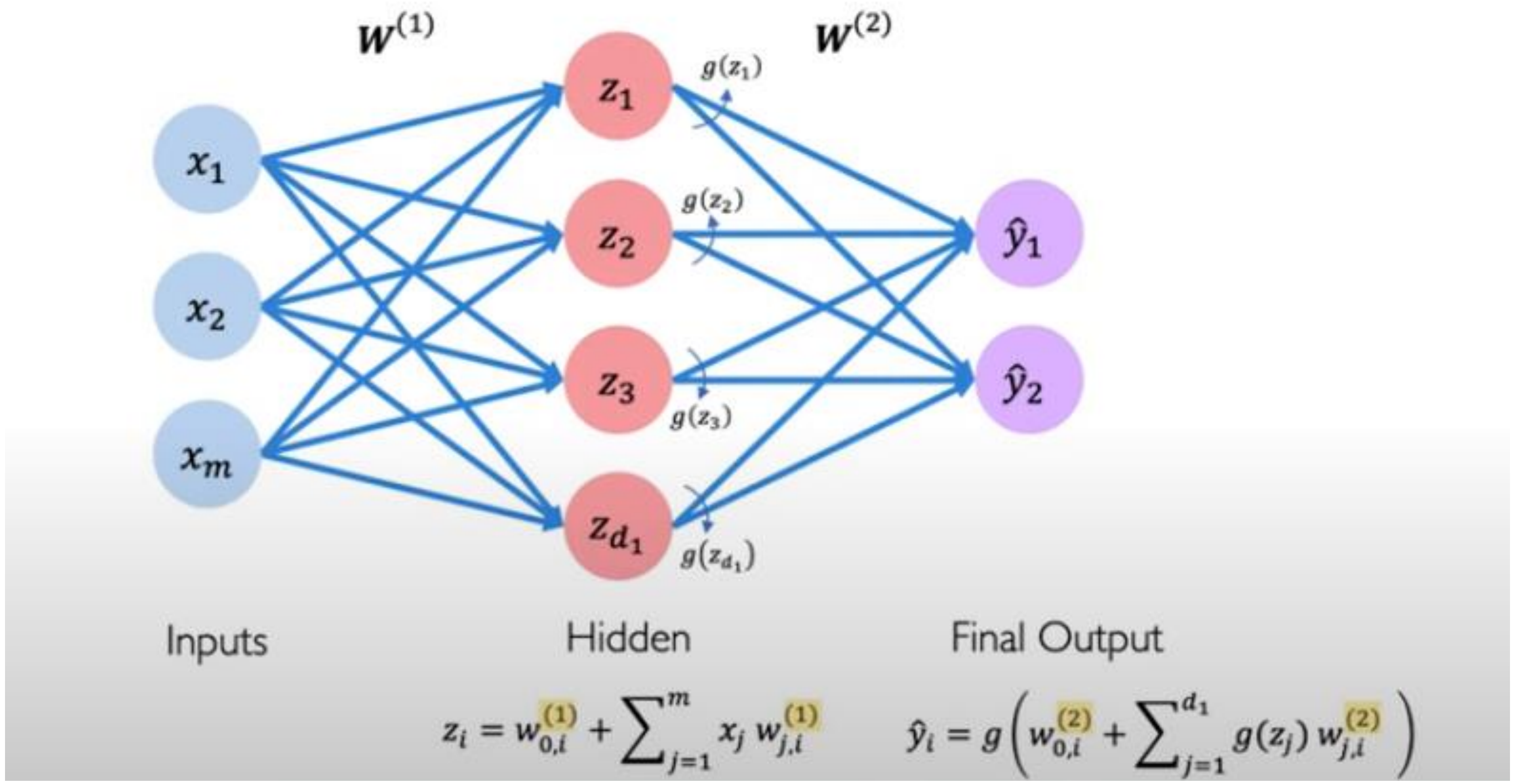
## Multiple Output Perceptron



$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j\, w_{j,i}$$

I will cover this again at Part 2 of this lecture on multiple layers and multiple output.

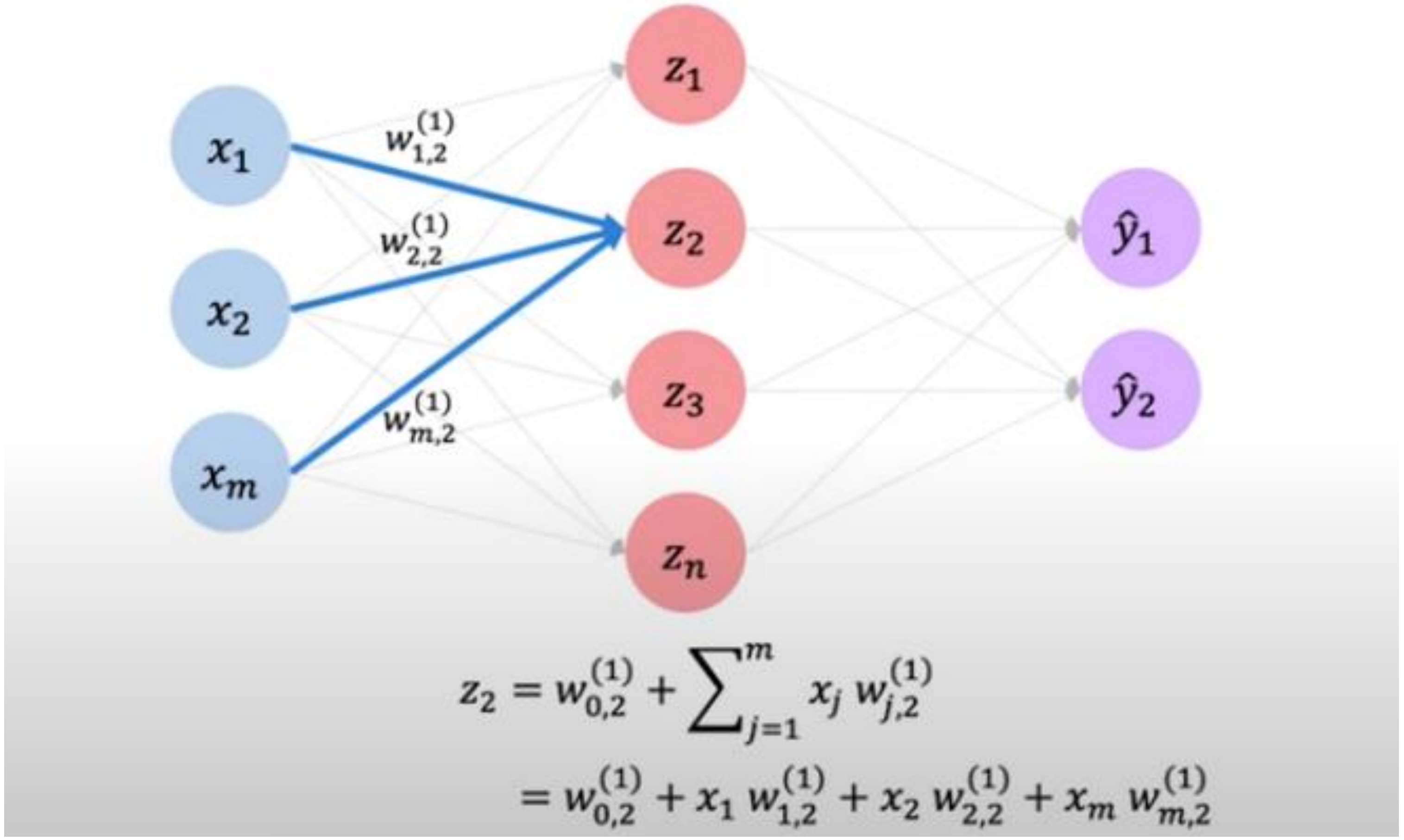
# A single hidden layer ANN



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j) \, w_{j,i}^{(2)}\right)$$

1. The layer in between the input and output layer is called the hidden layer
2. The state of the hidden layers are typically unobserved or hidden.

# The single layer ANN



$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j\, w_{j,2}^{(1)}$$

$$= w_{0,2}^{(1)} + x_1\, w_{1,2}^{(1)} + x_2\, w_{2,2}^{(1)} + x_m\, w_{m,2}^{(1)}$$

An example of computing the hidden layer second neuron ($Z_2$)

## How do you train the ANN?

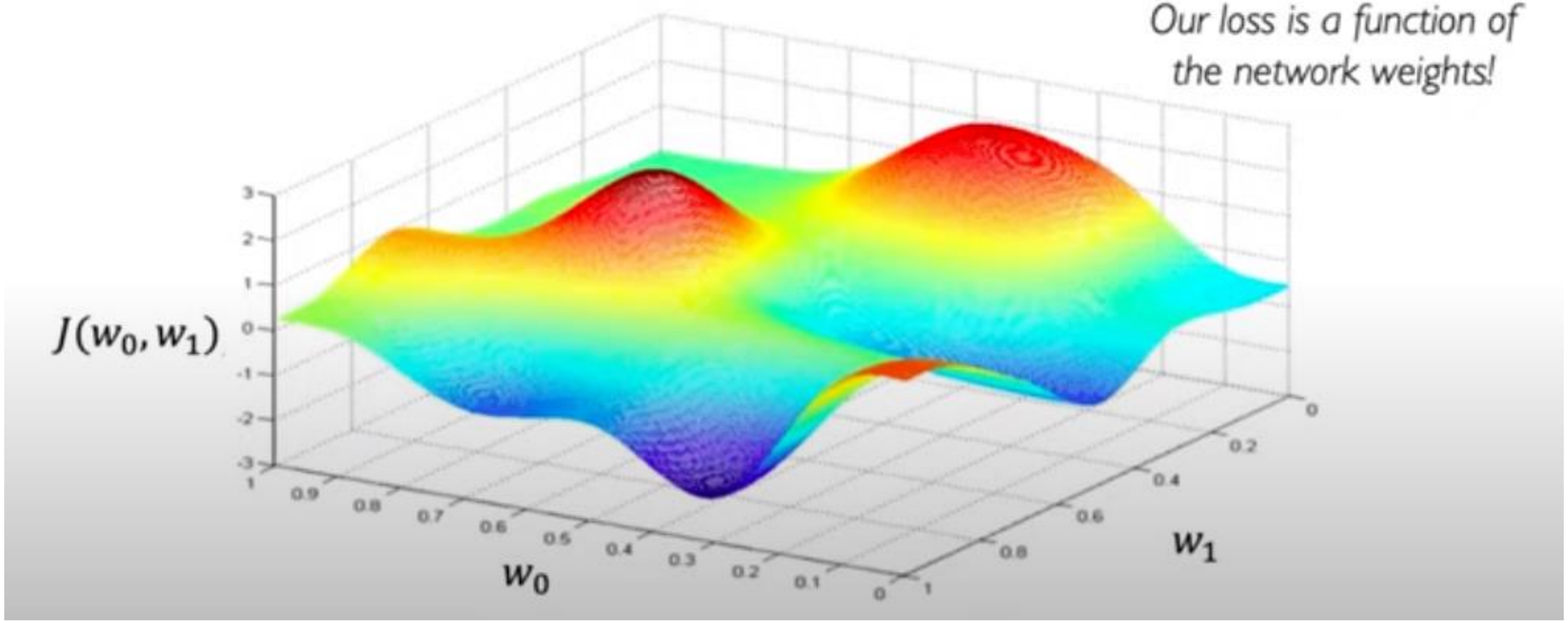
ANN are trained using gradient descent and require that you use a loss function when designing and configuring your  ANN model. Note: Loss function are sometimes known as Objective Function or Cost Function or Error Function.

Loss function plays a critical role in training ANN as it provide a means to calculate the model error.  Typically one would like to minimise the error hence minimising the loss function

Mean squared error (MSE) is the main type of loss functions to use when training neural network models.

# Loss Function and Optimisation

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



Remember:
*Our loss is a function of the network weights!*

Our loss function J (W) is to optimise all the weights.   Loss function here is minimising the error of the network, and to do so, is to find the lowest points in this landscape.  We do that by computing the gradient of the loss function J (W) with respect to the weight.

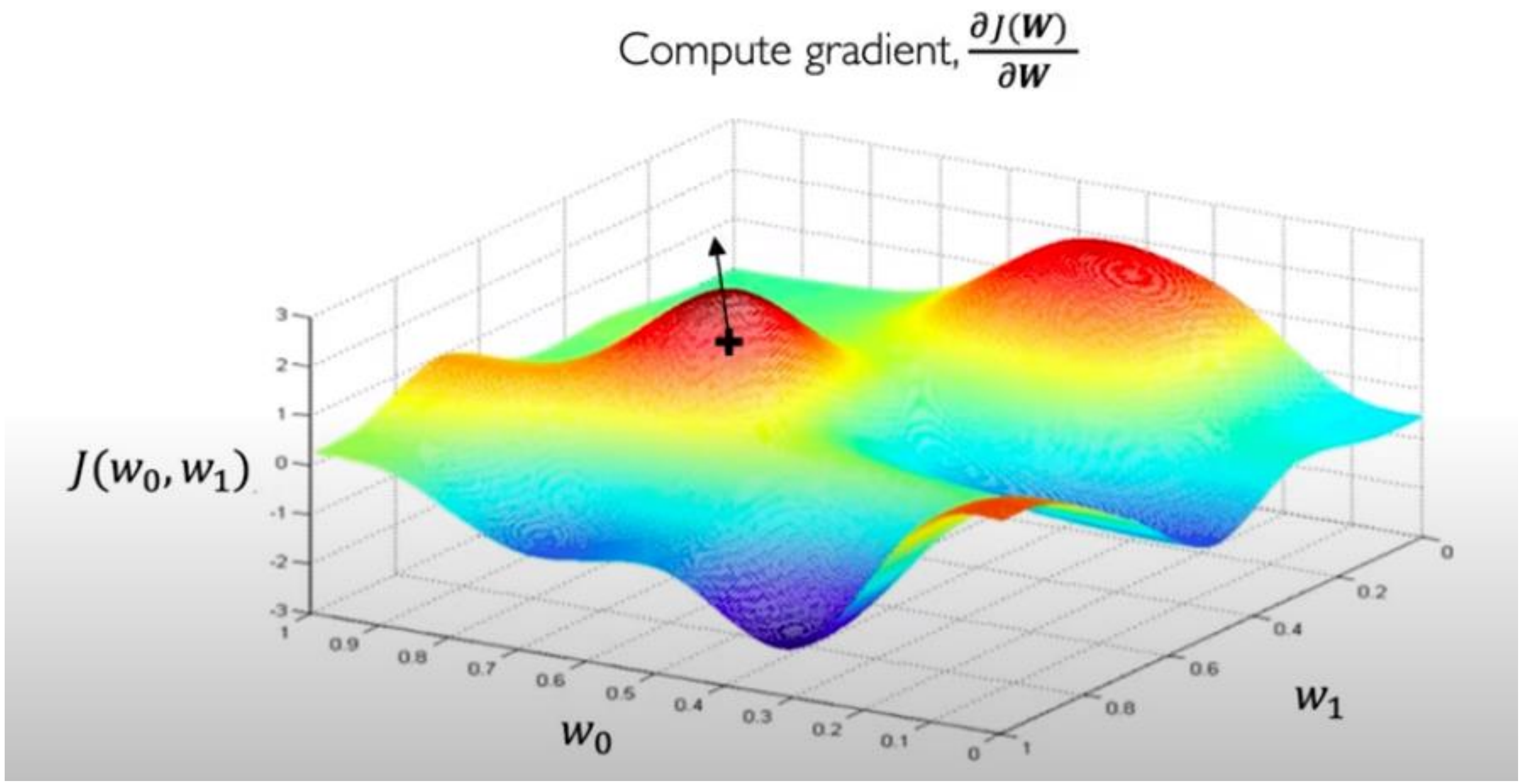
## Loss Function using Mean Squared Error (MSE)

This is calculated as the average of the squared differences between the predicted and actual values. The result is always positive regardless of the sign of the predicted and actual values and a perfect value is 0.0. The loss value is minimized

The Python function below provides a pseudocode-like working implementation of a function for calculating the mean squared error for a list of actual and a list of predicted real-valued quantities.

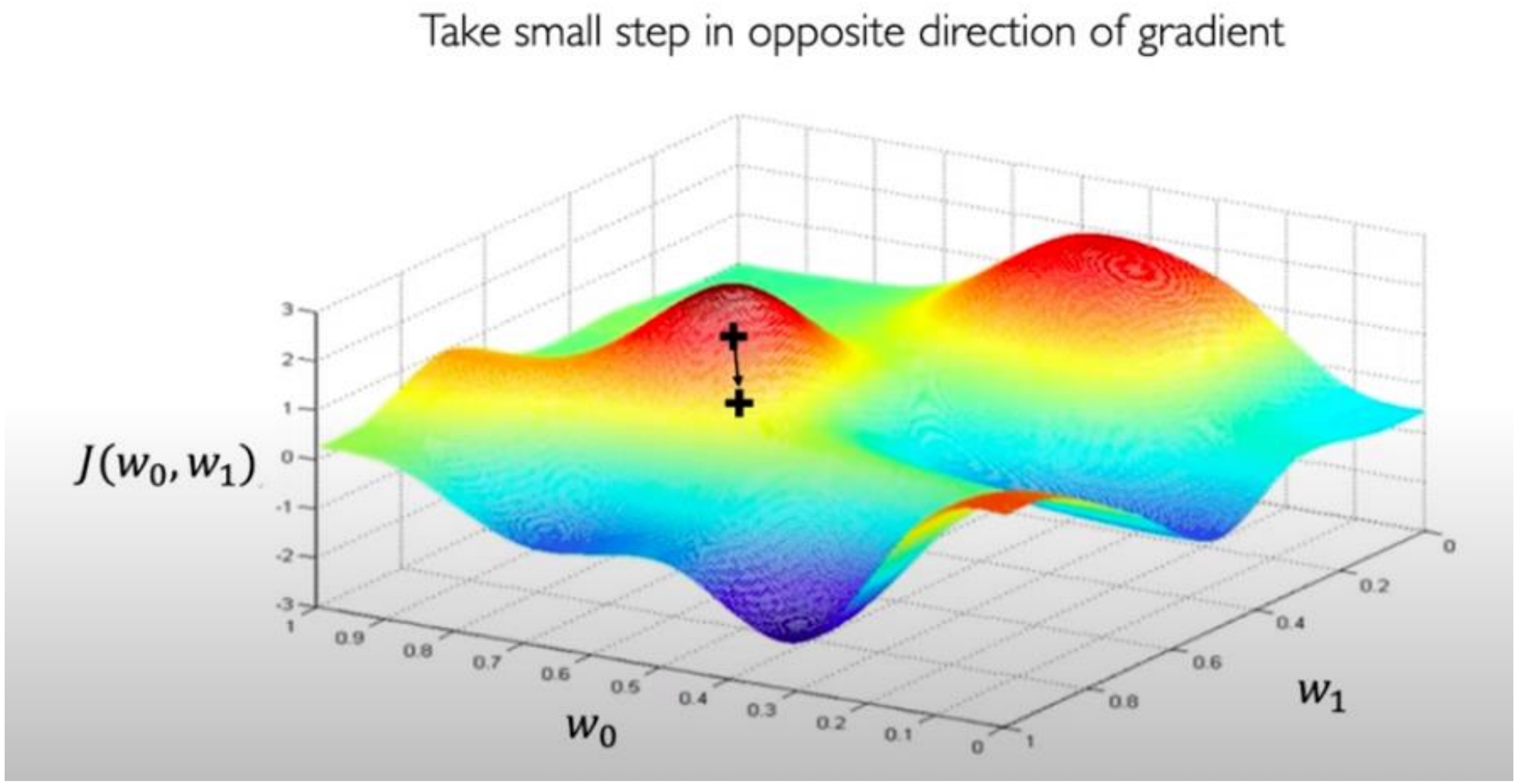
```
# calculate mean squared error
def mean_squared_error(actual, predicted):
    sum_square_error = 0.0
    for i in range(len(actual)):
        sum_square_error += (actual[i] - predicted[i])**2.0
    mean_square_error = 1.0 / len(actual) * sum_square_error
    return mean_square_error
```

# Loss Function and Optimisation

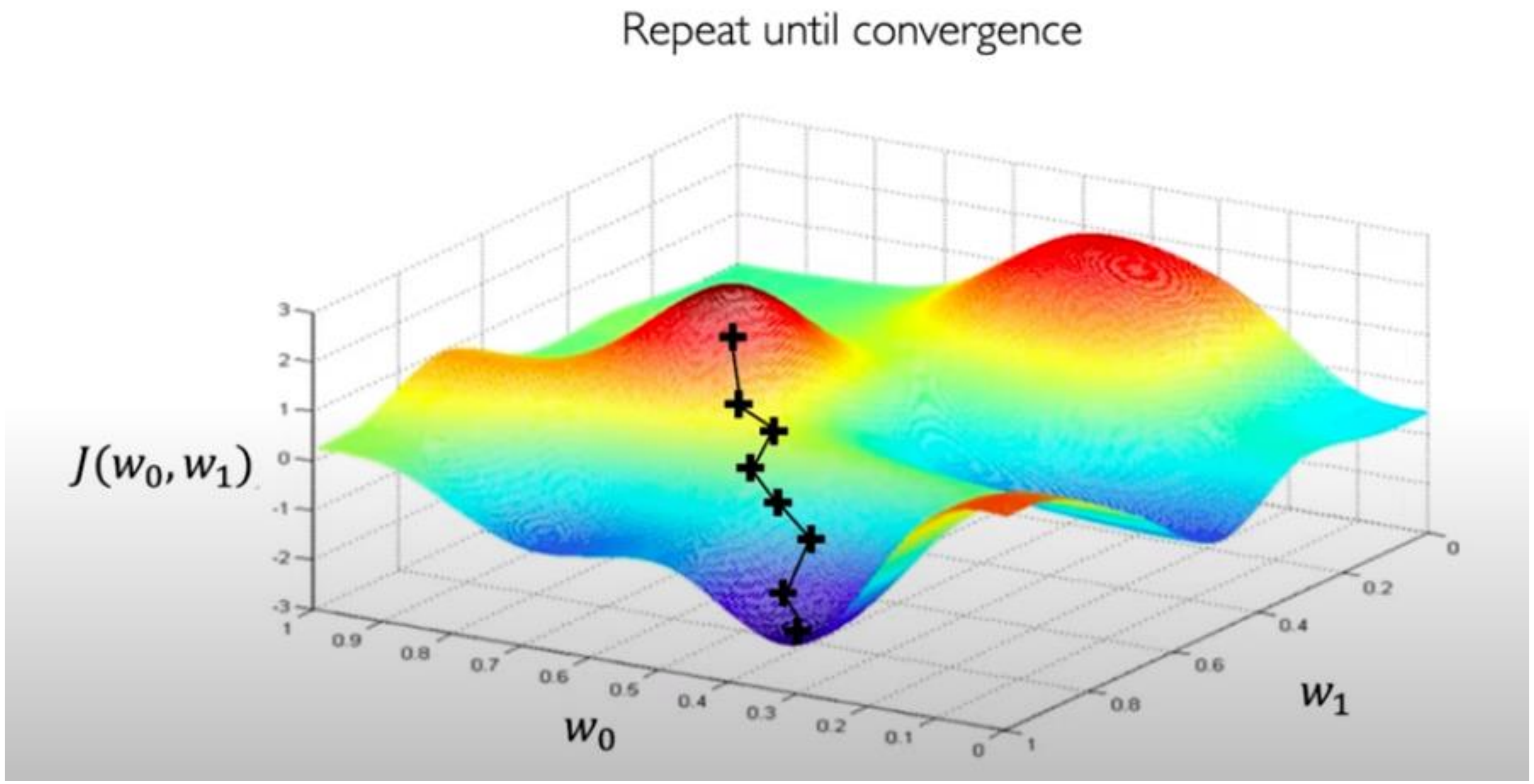


How do we start?  First pick a point for $w_0$ and $w_1$; then find the lowest point by computing the gradient of that point. Take the negative gradient, as we want to find the lowest point in the landscape.  Lowest point here is the minimum error.

# Loss Function and Optimisation

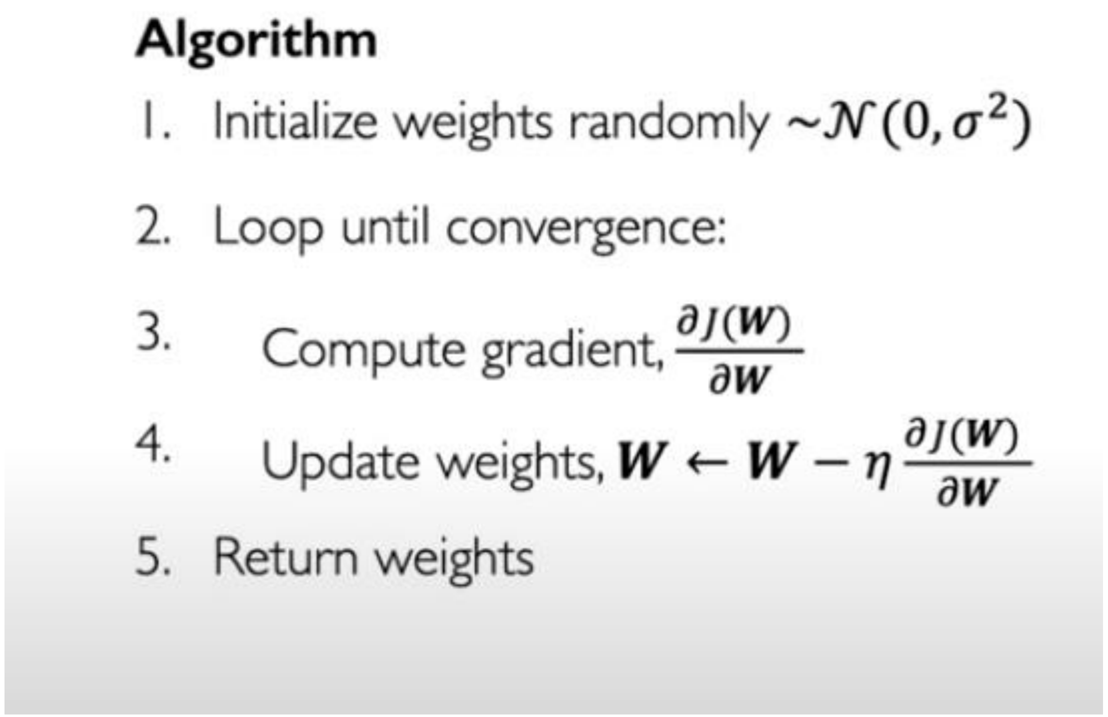


# Gradient Descent



Repeat until it converge to the lowest point.

# Gradient Descent algorithm

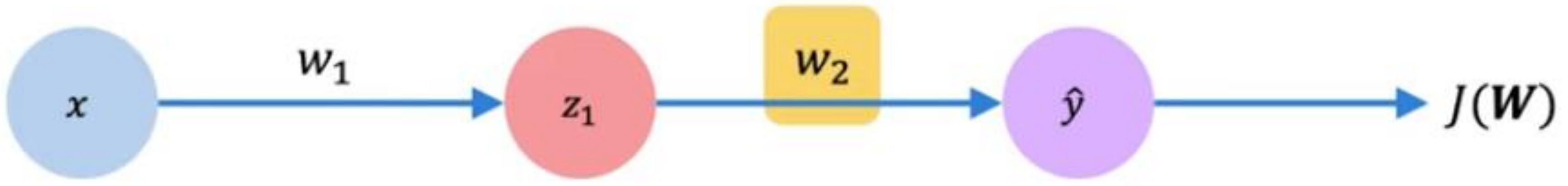


**Algorithm**

```
1. Initialize weights randomly $\sim\mathcal{N}(0,\sigma^2)$
2. Loop until convergence:
3.     Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4.     Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights
```

The small learning rate value lead to a small step.

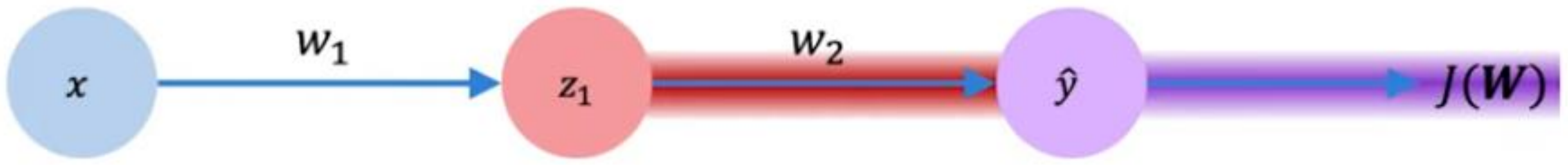
# Computing Gradient - Backpropagation



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

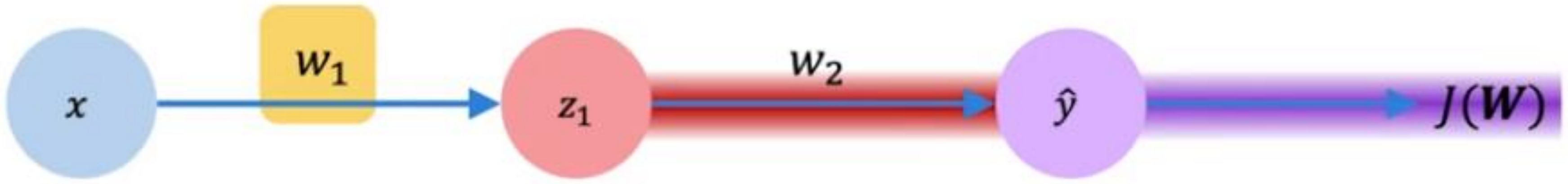
Start with a simple neural networks that have one input, one hidden layer with one neuron and one output. Compute the gradient of the loss function J(W)

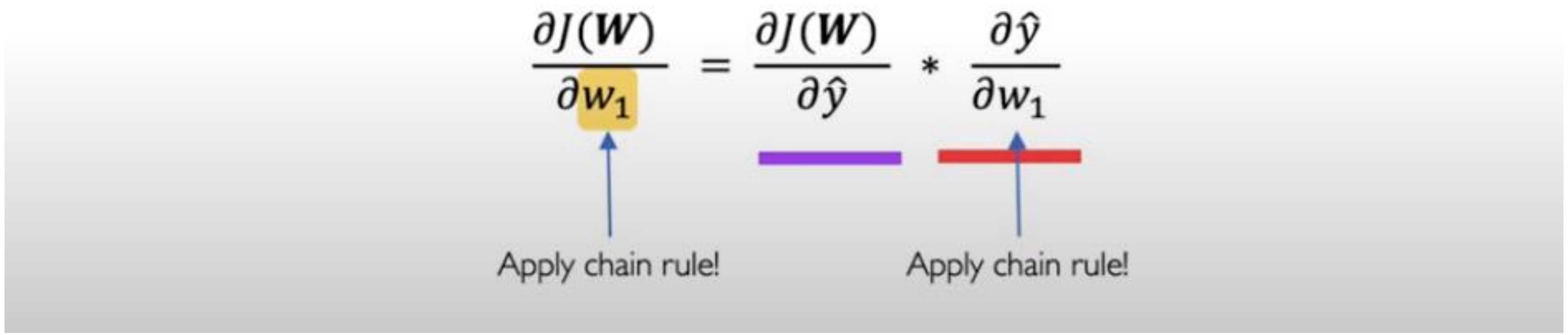
## Computing Gradient - Backpropagation



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

## Computing Gradient - Backpropagation

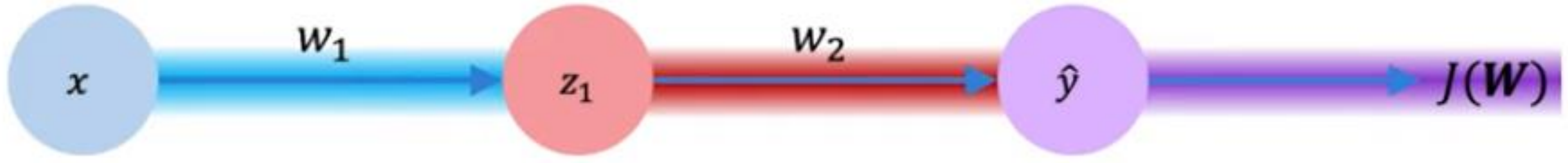




$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Question: How do we compute the gradient of the loss function with respect to $w_1$. Apply the chain rule.

# Computing Gradient - Backpropagation

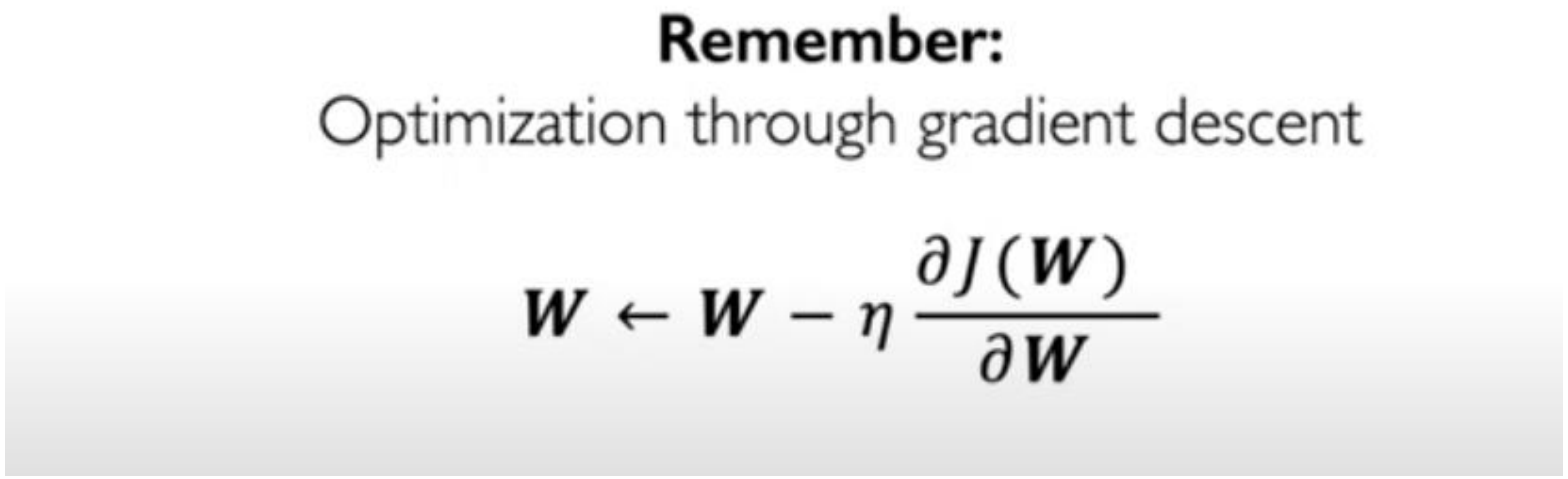


$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \underbrace{\frac{\partial J(\boldsymbol{W})}{\partial \hat{y}}} * \underbrace{\frac{\partial \hat{y}}{\partial z_1}} * \underbrace{\frac{\partial z_1}{\partial w_1}}$$

Small change in the weight will affect our loss.

Repeat this chain rule steps for all the weights in the subsequence layers (for deep neural network you have many layers)

## Loss Functions can be difficult to optimise

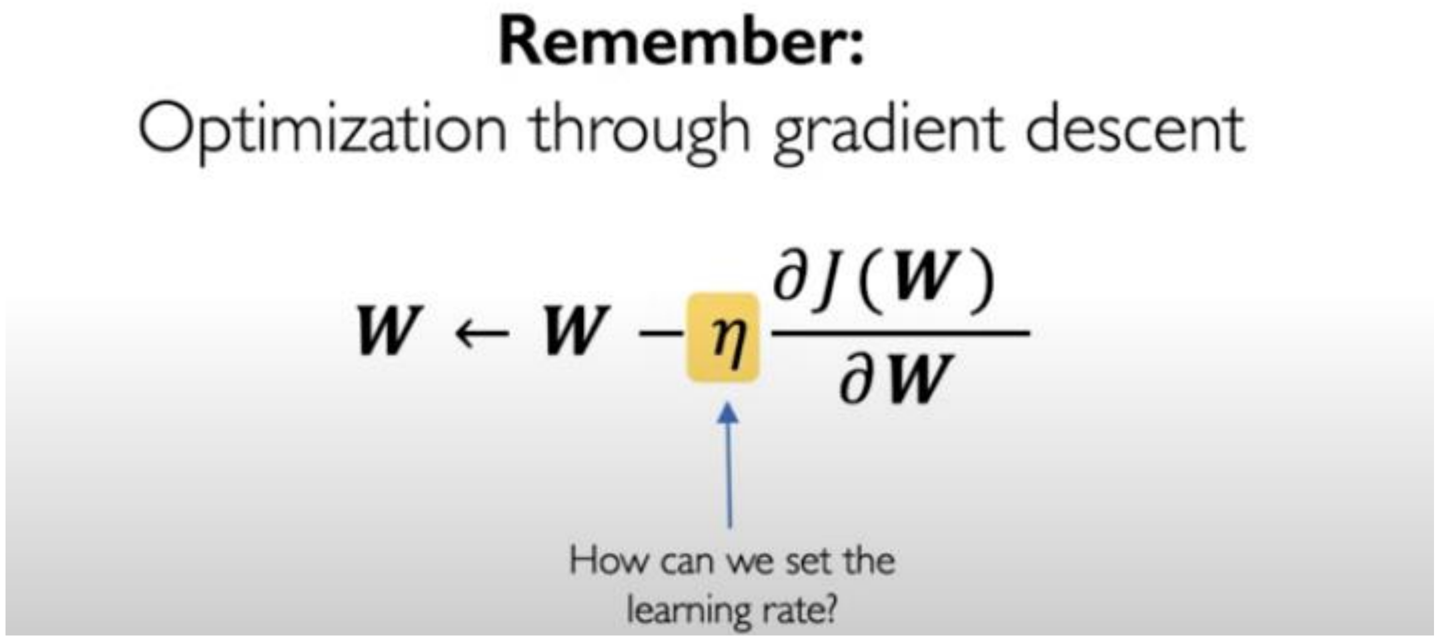


**Remember:**

Optimization through gradient descent

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

## How best to optimise the loss function?



**Remember:**

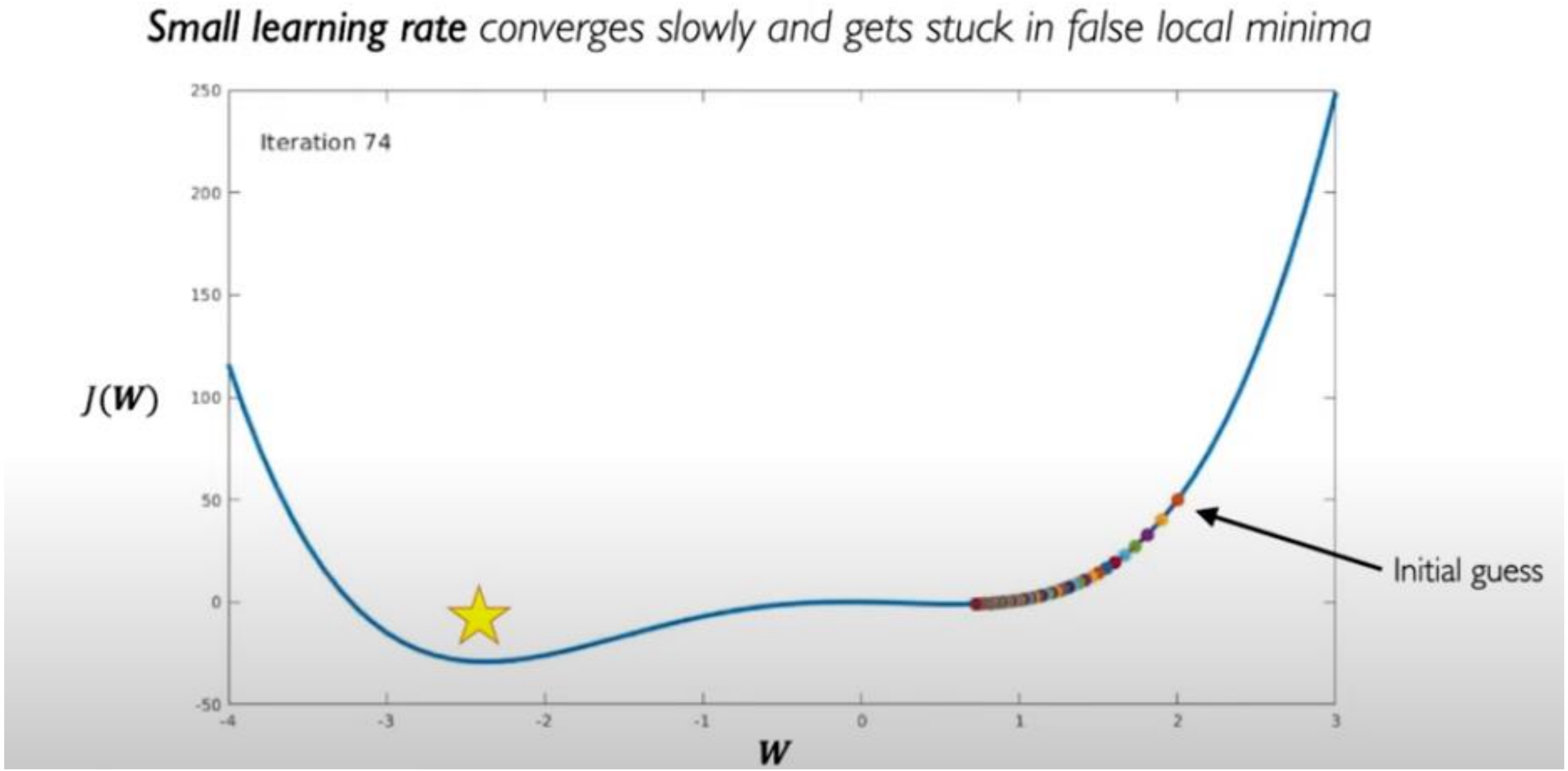
Optimization through gradient descent

$$W \leftarrow W - \boxed{\eta} \frac{\partial J(W)}{\partial W}$$

How can we set the learning rate?

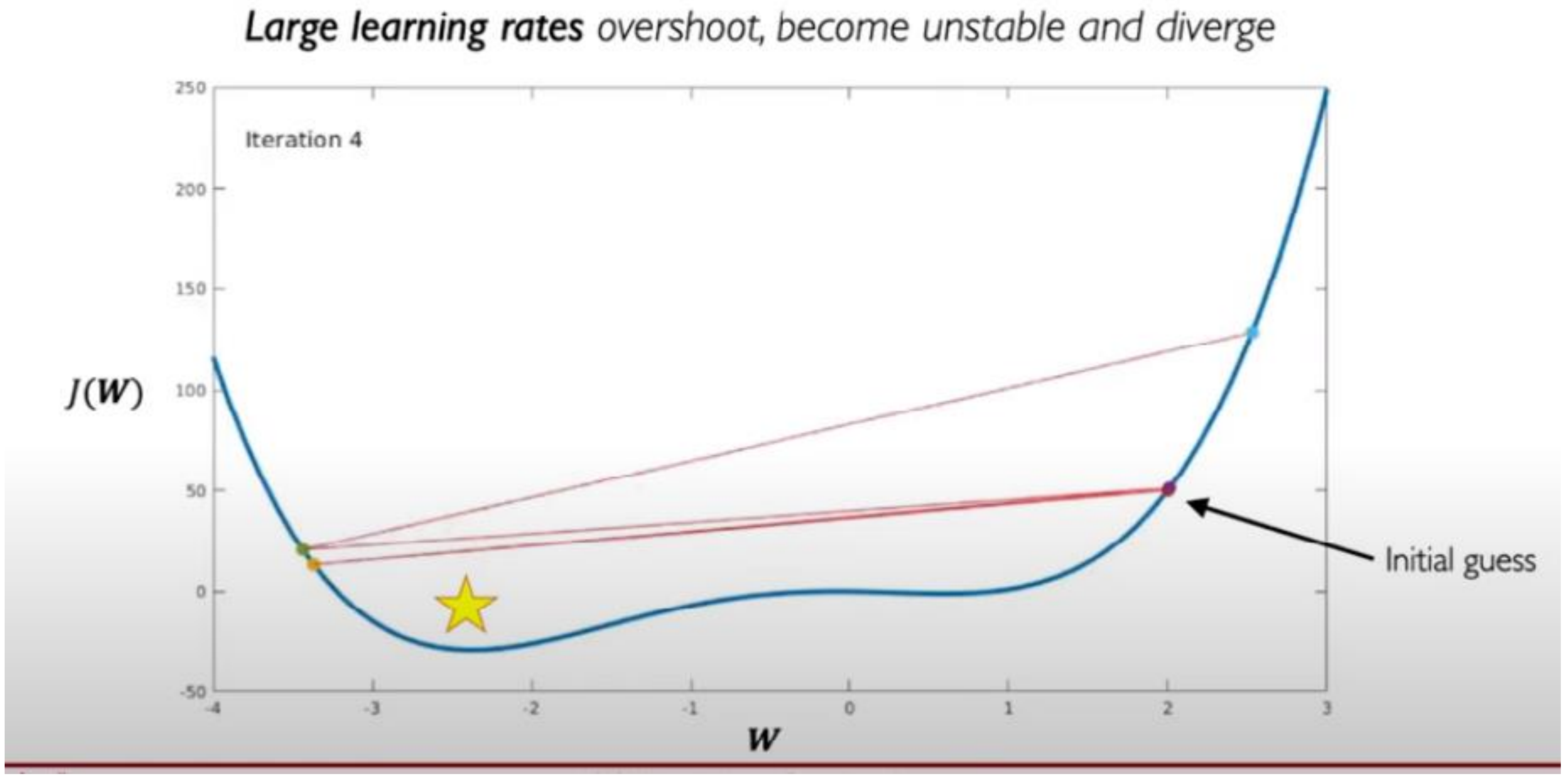
Eta (η) is the learning rate and is the step size of the moment or the update to the weight.

# Setting the learning rate



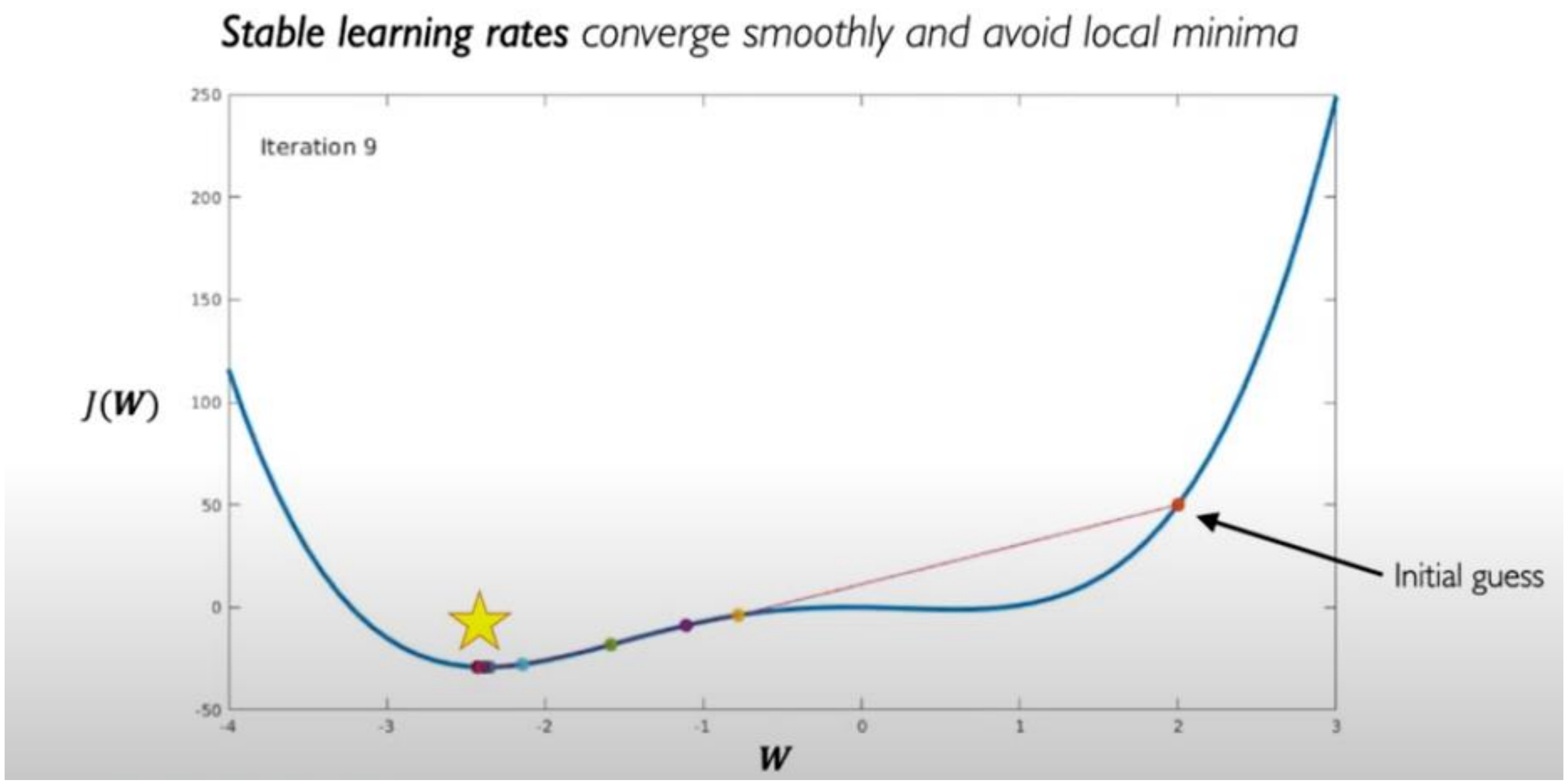
If the learning rate is too small it may stuck at local minimum or learn too slowly

# Setting the learning rate



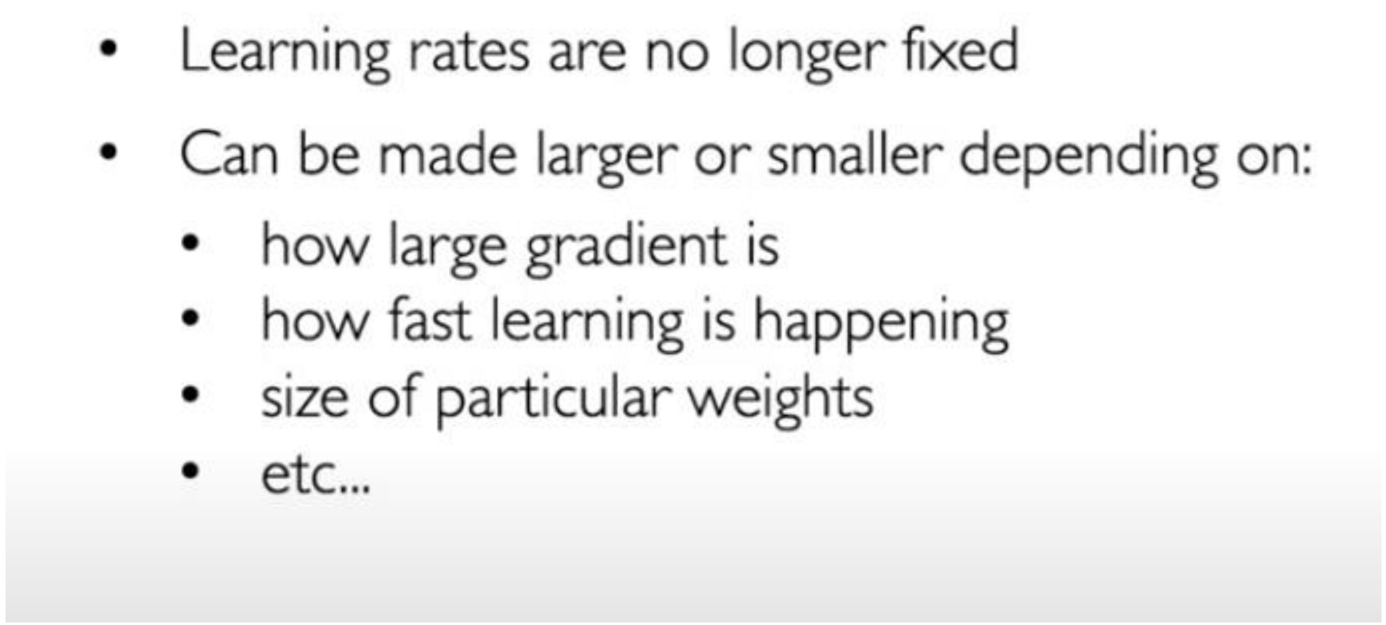
Large learning rate may cause overshoot and cannot converge.

## Setting the learning rate



Selecting the right learning rate and having a reasonable starting point.
What is a typical learning rate and what is a good starting point?

# Adaptive learning rates



- Learning rates are no longer fixed
- Can be made larger or smaller depending on:
  - how large gradient is
  - how fast learning is happening
  - size of particular weights
  - etc...

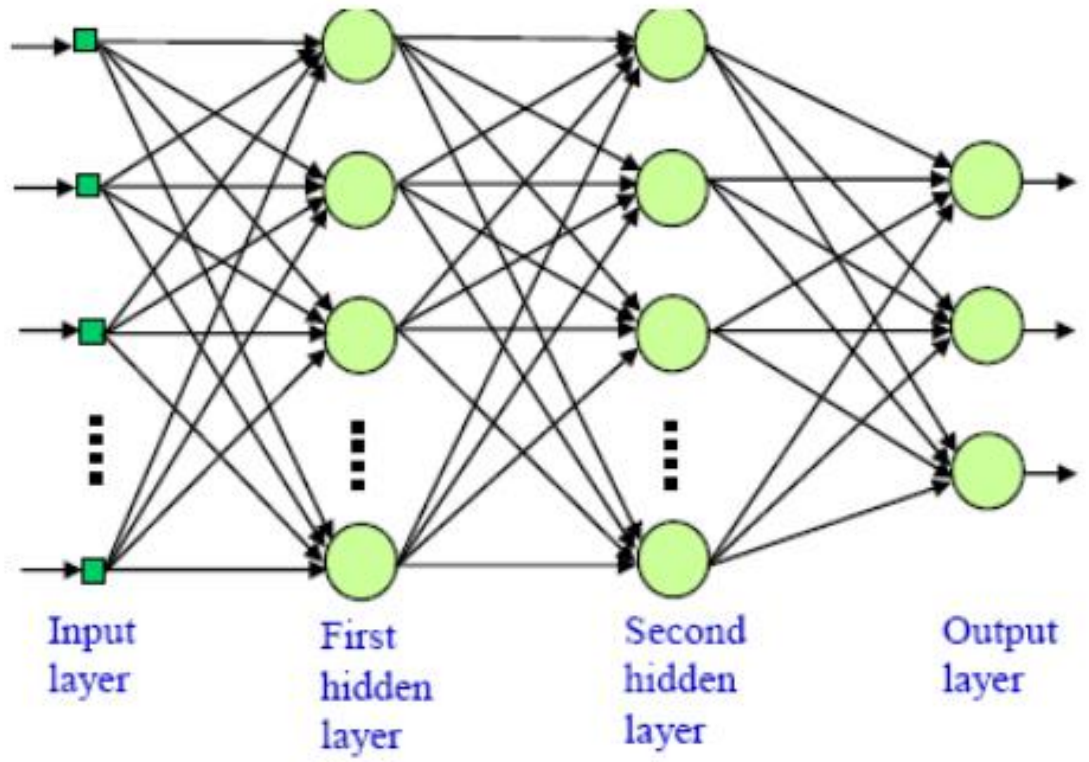
# Multiple Output Perceptron also known as Multi-Layer Perceptron (MLPs)

*Generalization of the single-layer perceptron*

Consists of:

- An input layer
- One or more hidden layers of computation nodes
- An output layer of computation nodes

Architectural graph of a multilayer perceptron with two hidden layers:



# Assignment on ANN

a.) Draw and label an ANN with one input layer, one hidden layer and one output layer; the number of neurons in the input layer is 2; and number of neuron in the hidden layer is 3. The output layer will have only 1 neuron. The activation function for the hidden neuron is sigmoid function. The activation function for output layer is any linear function. The input layer neurons have no activation function.

b.) List all the equations needed to calculate this ANN for both forward pass and backward pass.

c.) Demonstrate the training and learning of this ANN in doing classification or function approximation task. Note: To demonstrate the working of this simple ANN, you can generate your own data points.

d.) Show all diagrams, graphs and the results of the classification task.

National University of Singapore

Thank you for your attention
Q&A during tutorial time