Fundamental of Deep learning

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Course Code: **CS3232**

Course Name: **Fundamentals of Deep Learning**

Semester: **5**

Area: **Specialization**

CIE : 70 (Marks)

SEE Type: **Theory (30 Marks)**

Credits: 3: 0 : 2

Contact Hours: 75

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Course Context & Overview

This course offers an in-depth exploration of deep learning techniques and their applications.

The syllabus includes foundational topics such as neural networks and deep learning principles, practical skills in implementing Convolutional Neural Networks (CNNs) for object recognition, and advanced techniques like autoencoders and Generative Adversarial Networks (GANs) for unsupervised learning.

Students will also gain experience with Recurrent Neural Networks (RNNs) for sequence modeling and text generation.

The course emphasizes practical skills in optimizing and deploying models using deep learning SDKs on cloud platforms like AWS.

Internal Assessment Plan: 70 Marks

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Sl#	Component	Marks	Type of Assessment	Timeline
CP 1 (20 Marks)				
1	CO1	10	Graded Component 1 (e.g., Quiz)	Week 3
2	CO2	10	Graded Component 2 (e.g., Quiz)	Week 5
CP 2 (25 Marks)				
3	CO1 and CO4	25	Mid Sem Examination (Theory)	Week 9
CP 3 (25 Marks)				
4	CO1 – CO6	25	Practical Component	Week 14

Unit 1: Introduction to Neural Networks, Deep Learning Fundamentals and Convolutional Neural Networks

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

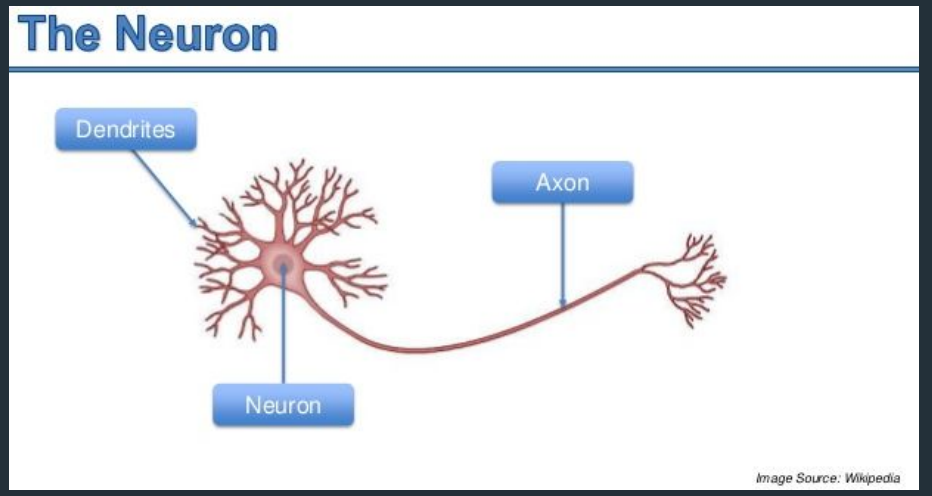
- Introduction to Neural networks: Perceptron, Multilayer Perceptron, Back propagation, Gradient Descent (GD), Momentum Based GD, Nesterov Accelerated GD, Stochastic GD, AdaGrad, RMSProp, Adam, Shallow neural networks, deep neural networks.
- Introduction to Deep Learning: Overview of machine learning and deep learning, History and Evolution of Deep Learning.
- Introduction to Convolutional Neural Networks, Layers in CNN, Types of convolutions,
- Regularization Techniques: L1/L2 regularization, dropout, Early stopping, Data augmentation, A typical CNN structure, Standard CNN models: AlexNet, ZF-Net, VGGNet, GoogLeNet, ResNet, Transfer learning and fine-tuning pre-trained models.

Neural networks

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Neural networks were inspired by the neural architecture of a human brain, and like in a human brain the basic building block is called a Neuron. Its functionality is similar to a human neuron, i.e. it takes in some inputs and fires an output.

A **Neural Network** is a biological representation of neurons in the brain it deals with all the connections and chemical reactions in the brain. Artificial Neural Networks are the computational models inspired by the human brain



Supervised Learning

Input(x)	Output (y)	Application
Home features	Price	Real Estate
Ad, user info	Click on ad? (0/1)	Online Advertising
Image	Object (1,...,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image, Radar info	Position of other cars	Autonomous driving

Standard NN

CNN

RNN

Custom/ Hybrid

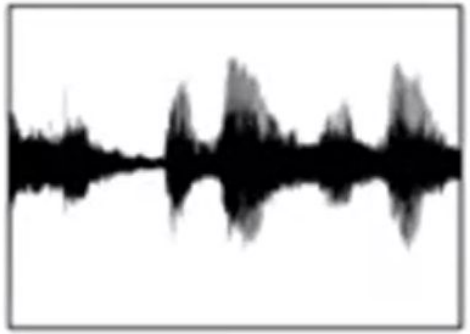
Supervised Learning

Structured Data

Size	#bedrooms	...	Price (1000$s)
2104	3		400
1600	3		330
2400	3		369
⋮	⋮		⋮
3000	4		540

User Age	Ad Id	...	Click
41	93242		1
80	93287		0
18	87312		1
⋮	⋮		⋮
27	71244		1

Unstructured Data



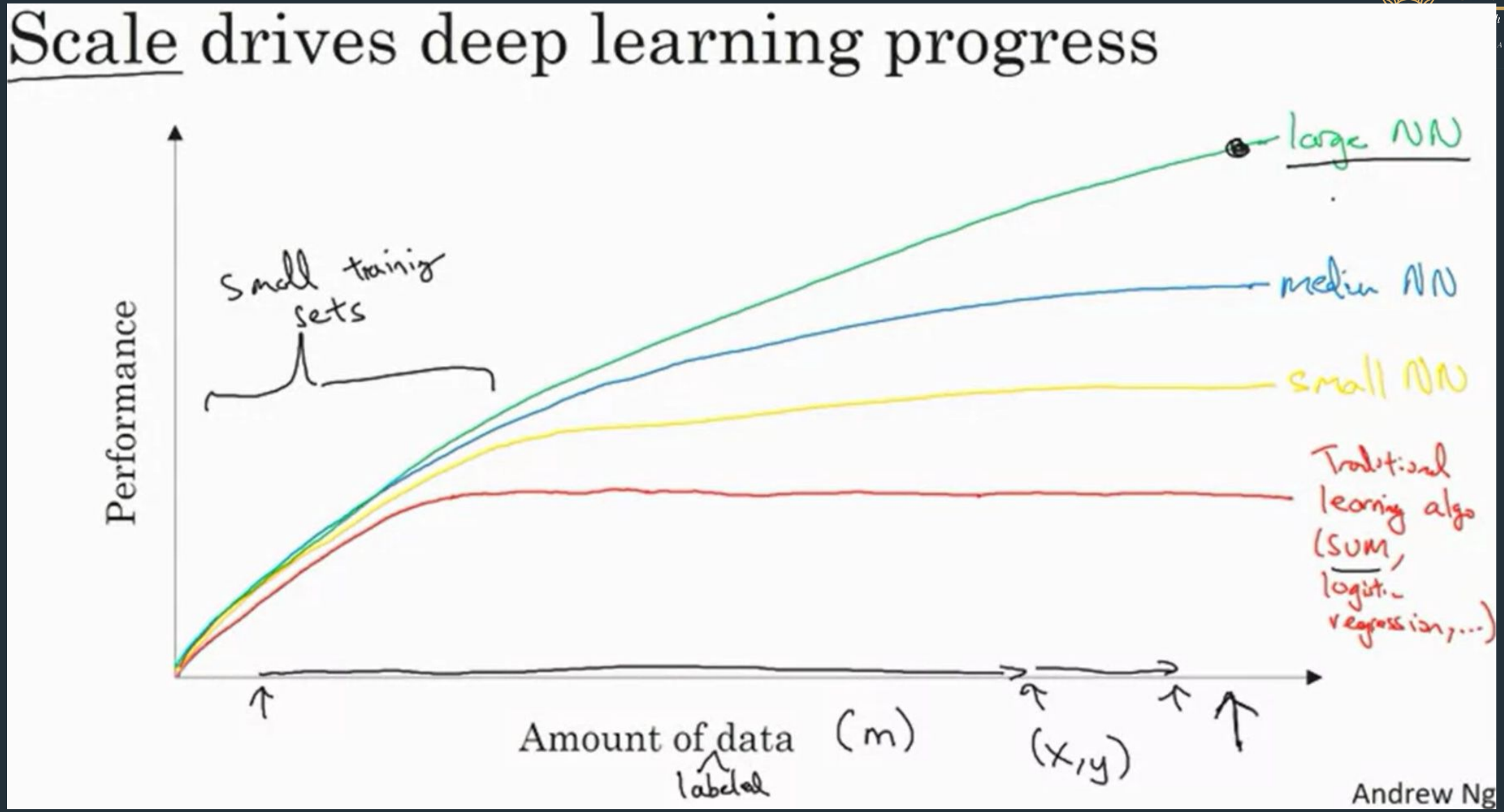
Audio

Image

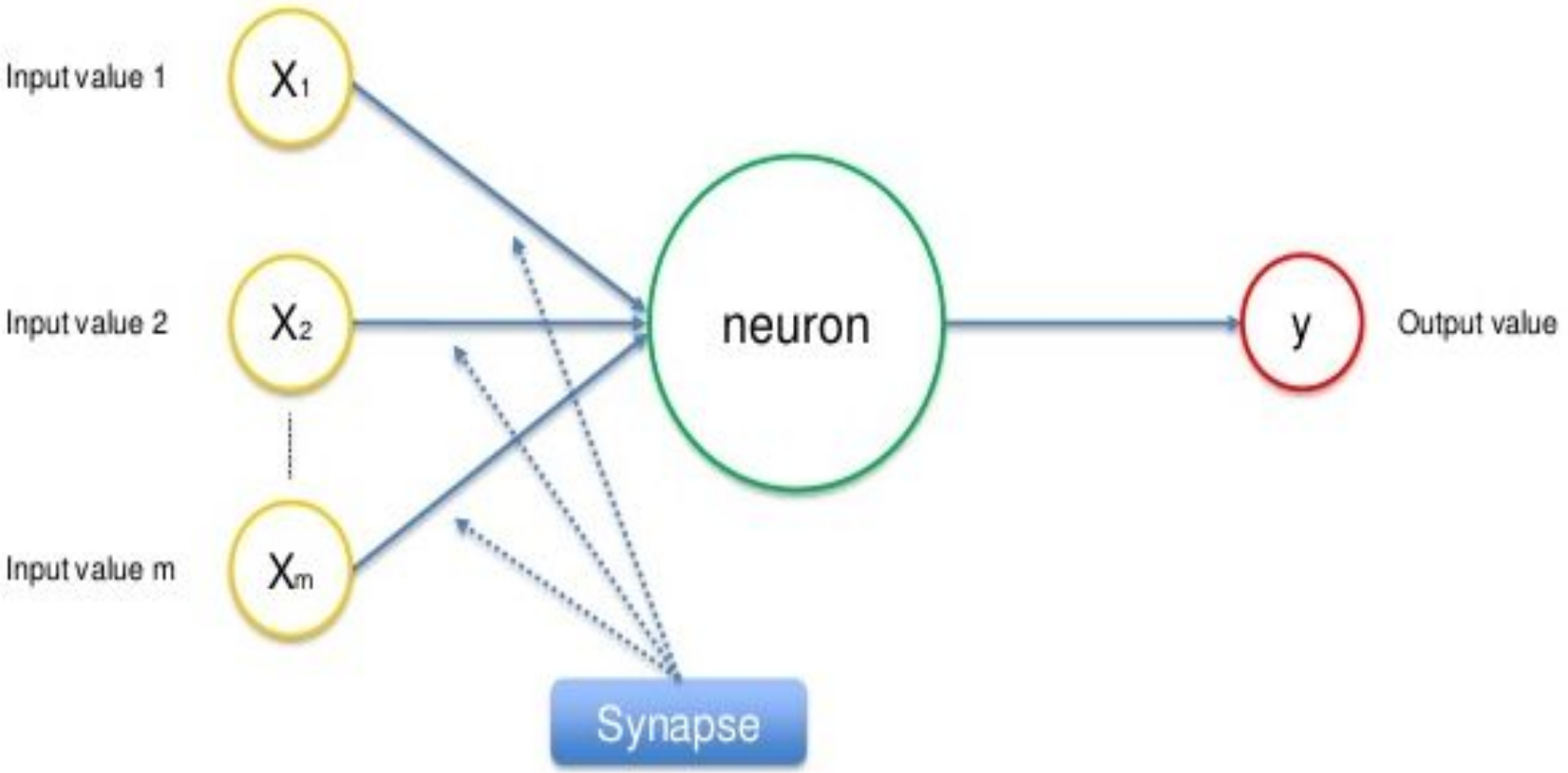
Four scores and seven years ago...

Text

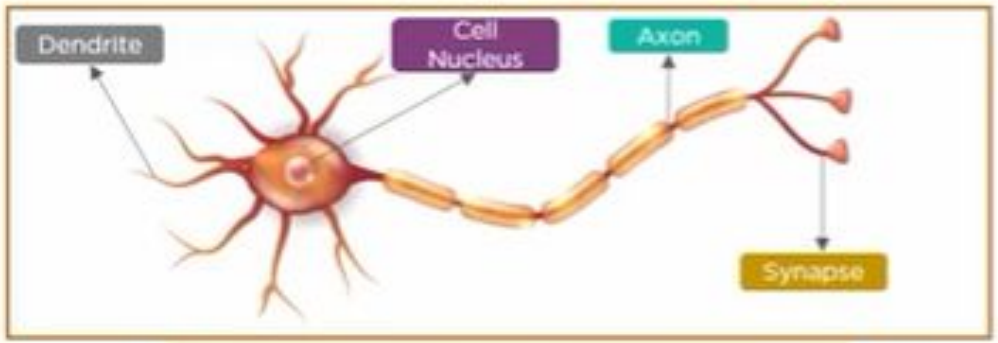
Scale drives deep learning progress

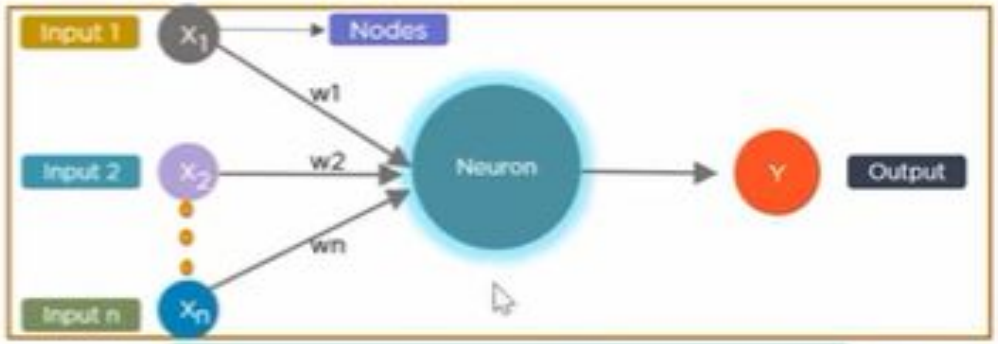


The Neuron


Input value 1
X1
Input value 2
X2
Input value m
Xm
neuron
y
Output value
Synapse

Biological Neuron vs Artificial Neuron

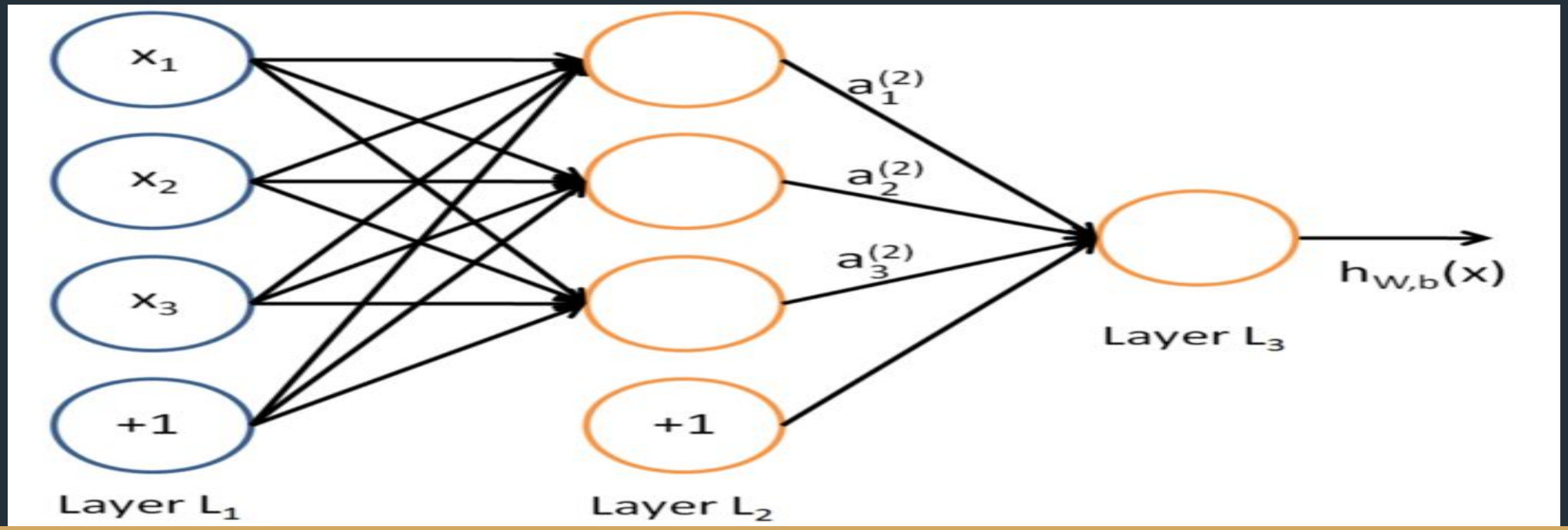




Biological Neuron	Artificial Neuron
Dendrites	Inputs
Cell Nucleus	Nodes
Synapse	Weights
Axon	Output

What is a layer in a Neural Network?

A layer is nothing but a collection of neurons which take in an input and provide an output. Inputs to each of these neurons are processed through the activation functions assigned to the neurons.



A few properties of Neural Networks

- The leftmost layer of the network is called the input layer, and the rightmost layer the output layer
- The middle layer of nodes is called the hidden layer because its values are not observed in the training set.
- Any neural network has 1 input and 1 output layer. The number of hidden layers, for instance, differ between different networks depending upon the complexity of the problem to be solved.
- Each of the hidden layers can have a different activation function, for instance, hidden layer1 may use a sigmoid function and hidden layer2 may use a ReLU, followed by a Tanh in hidden layer3 all in the same neural network.
- For a neural network to make accurate predictions each of these neurons learn certain weights at every layer. The algorithm through which they learn the weights is called back propagation

A few properties of Neural Networks

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

- Weights(Parameters)—A weight represent the strength of the connection between units. If the weight from node 1 to node 2 has greater magnitude, it means that neuron 1 has greater influence over neuron 2.
- A weight affects the importance of the input value. Weights near zero means changing this input will not change the output. Negative weights mean increasing this input will decrease the output. A weight decides how much influence the input will have on the output.
- Bias(Offset)—It is an extra input to neurons and it is always 1, and has it's own connection weight. This makes sure that even when all the inputs are none (all 0's) there's gonna be an activation in the neuron.

Activation Functions

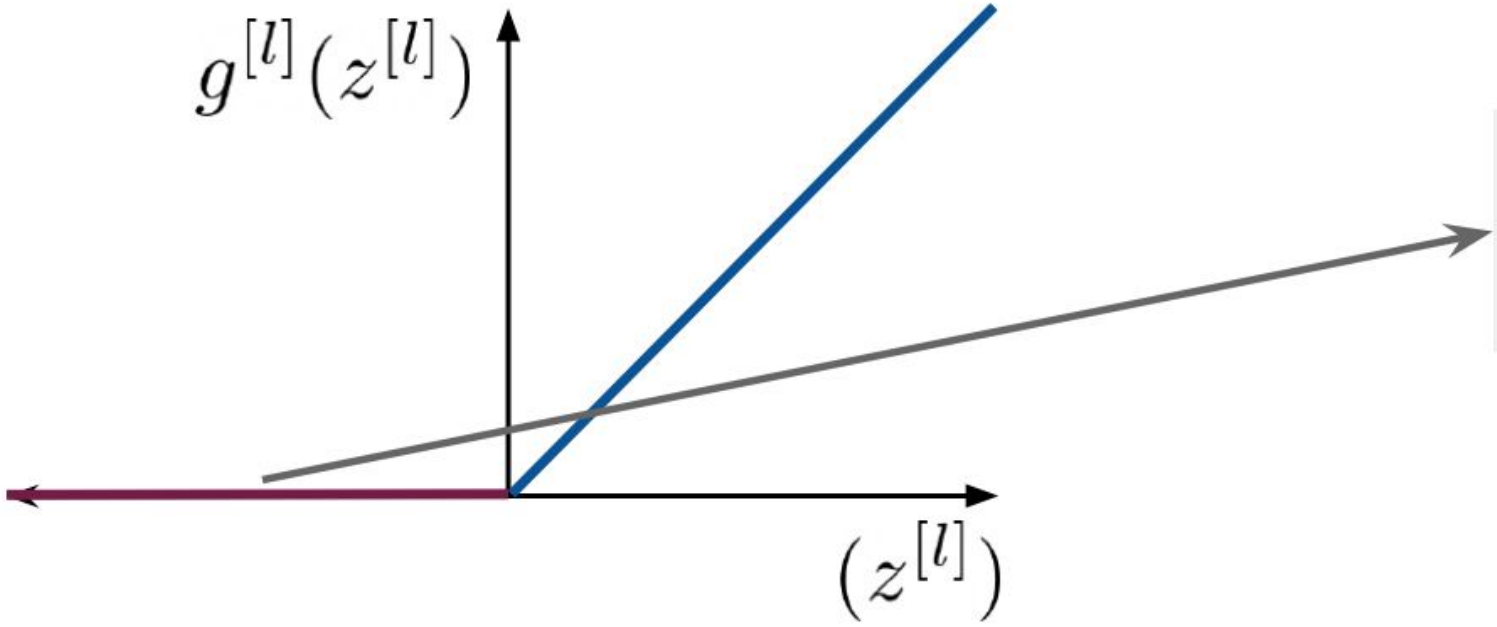
- Functions applied to the weighted sum of inputs and biases to introduce non-linearity into the network.
- **Sigmoid:** Maps inputs to a range between 0 and 1, often used in binary classification.
- **ReLU (Rectified Linear Unit):** Outputs the input if positive; otherwise, it outputs zero. Commonly used in hidden layers.
- **Tanh:** Maps inputs to a range between -1 and 1, often used for hidden layers.
- **Softmax:** Converts logits to probabilities, typically used in the output layer for multi-class classification.

- https://www.tinkershop.net/ml/sigmoid_calculator.html

Activations: ReLU

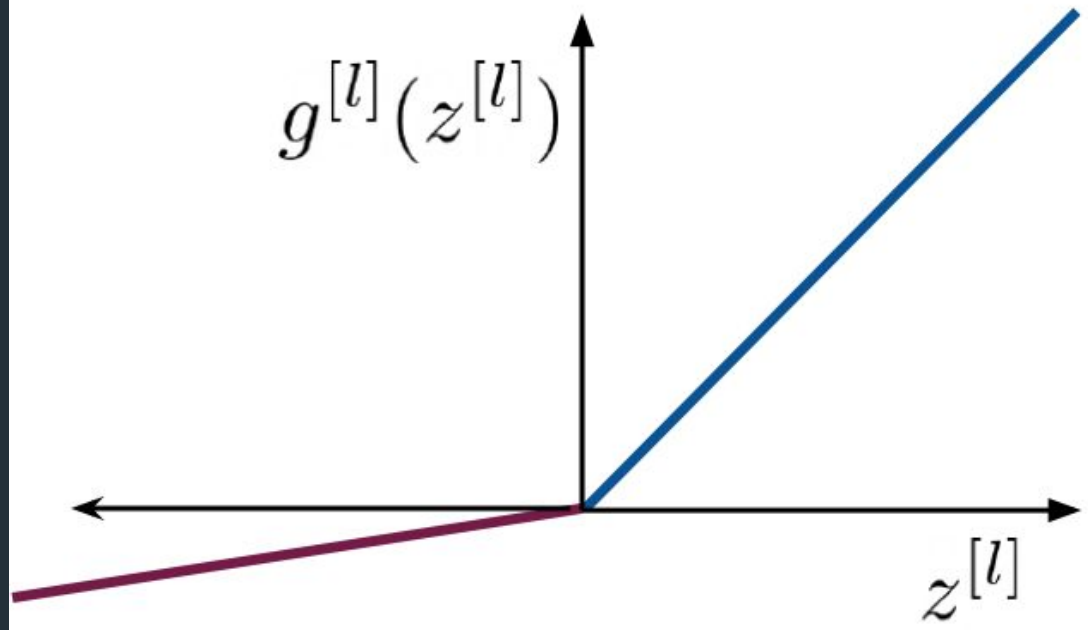
ReLU = Rectified Linear Unit

$$g^{[l]}(z^{[l]}) = \max(0, z^{[l]})$$



Dying ReLU problem

Activations: Leaky ReLU

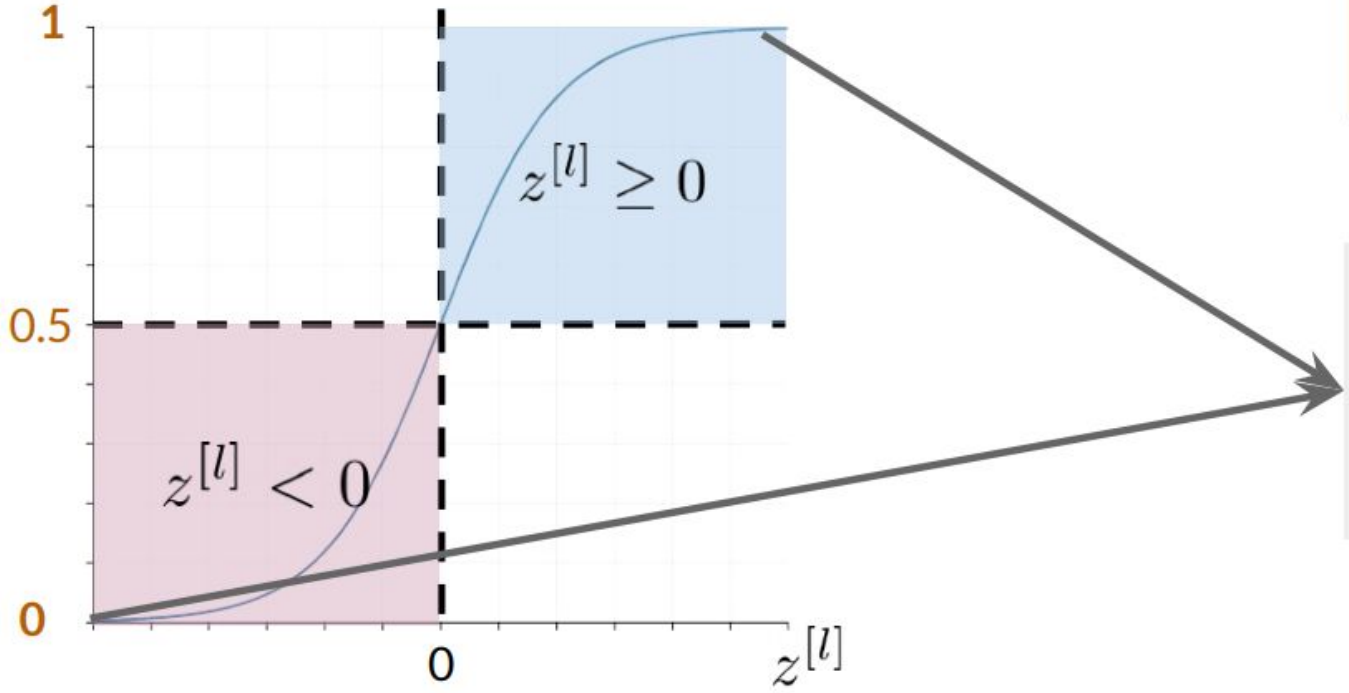


$$g^{[l]}(z^{[l]}) = max(az^{[l]}, z^{[l]})$$

Solves the dying ReLU problem

Activations: Sigmoid

$$g^{[l]}(z^{[l]}) = \frac{1}{1 + e^{-z^{[l]}}}$$



Values between 0 and 1

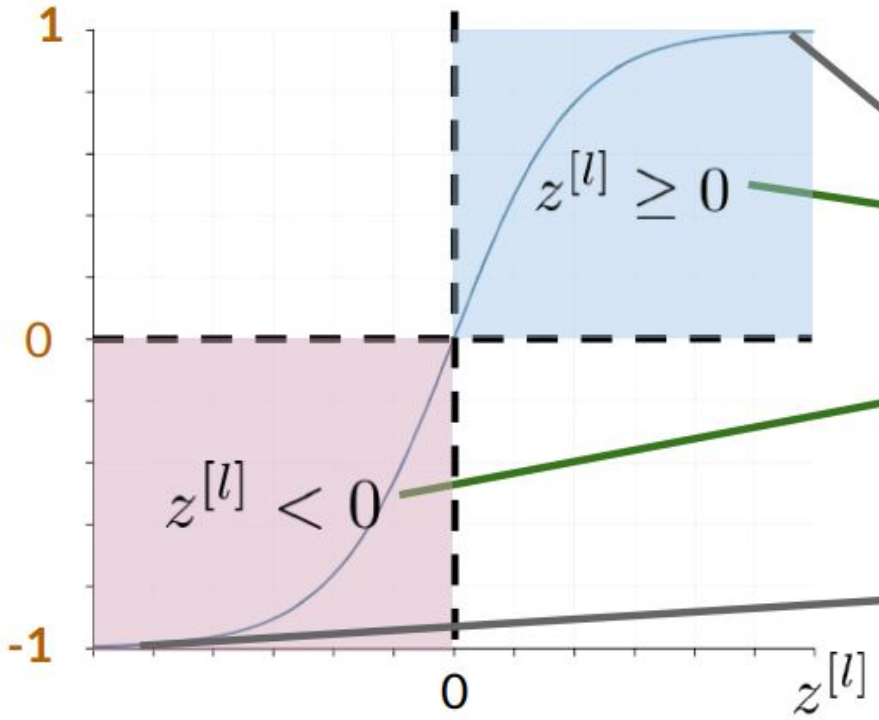
Vanishing gradient and saturation problems

Activations: Tanh

$$g^{[l]}(z^{[l]}) = tanh(z^{[l]})$$

$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

Values between -1 and 1

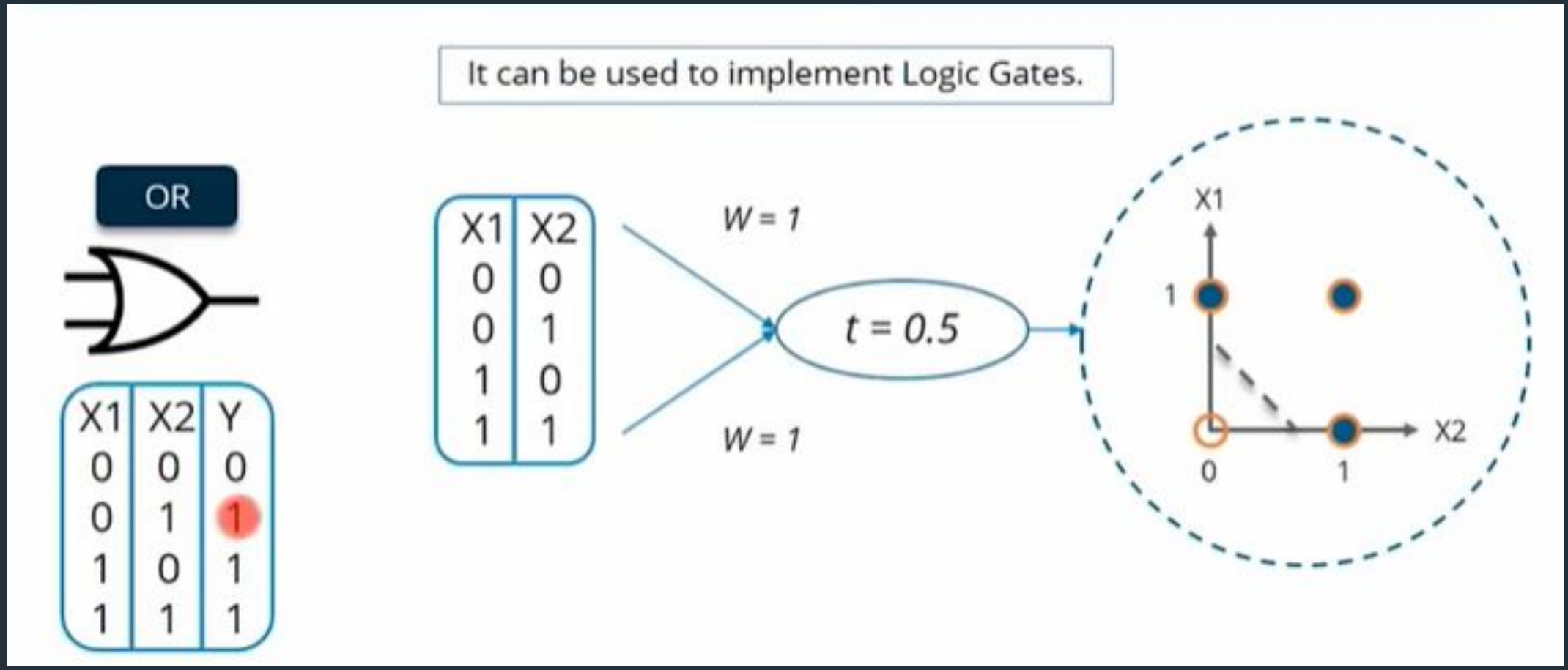


Keeps the sign of the input

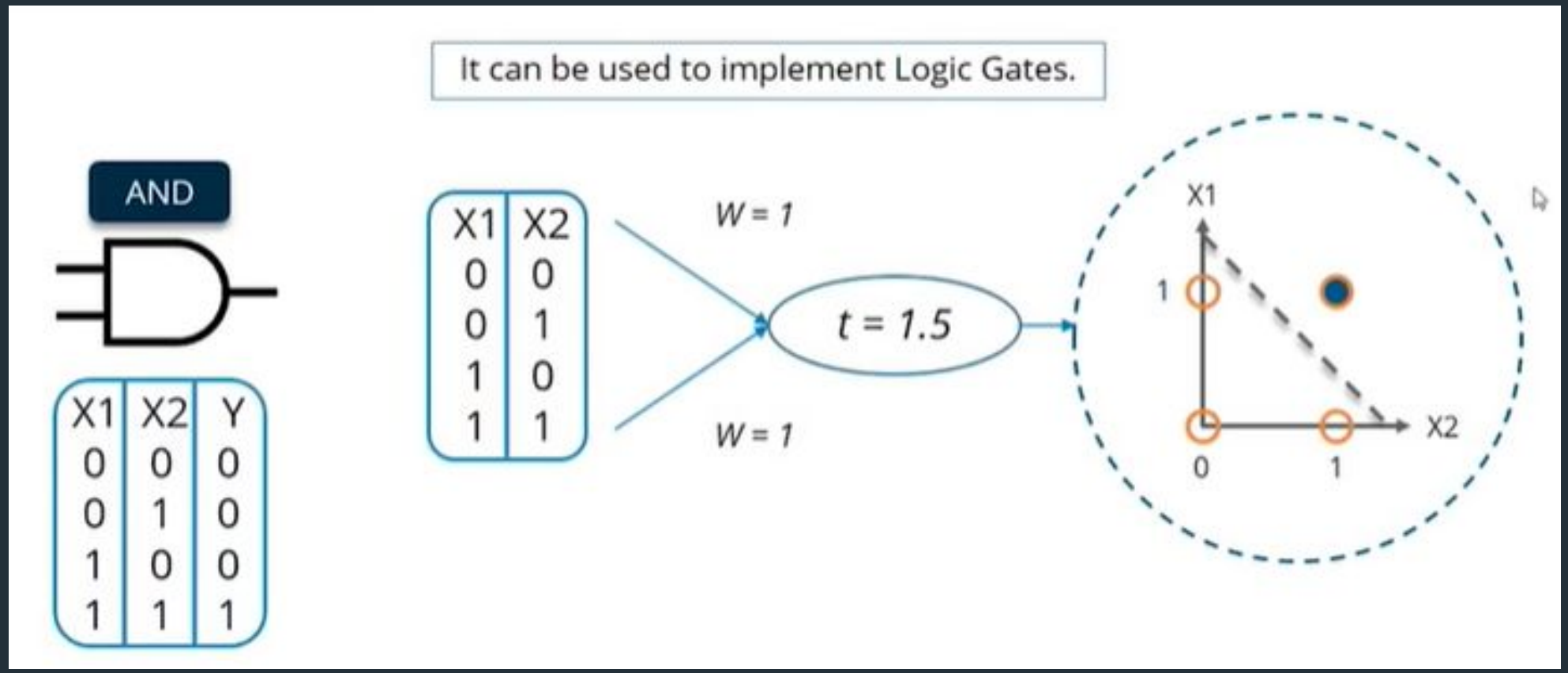
Same issues as Sigmoid

Activation Function	Formula	Range	Characteristics	Advantages	Disadvantages
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	(0, 1)	Squashes values to a range between 0 and 1.	Easy to implement. Outputs probabilities.	Can suffer from vanishing gradients. Output not zero-centered.
Tanh	$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	(-1, 1)	Squashes values to a range between -1 and 1.	Zero-centered. Mitigates vanishing gradient problem better than sigmoid.	Can suffer from vanishing gradients.
ReLU (Rectified Linear Unit)	$\mathrm{ReLU}(x) = \max(0, x)$	[0, ∞)	Outputs zero for negative inputs and the input itself for positive inputs.	Simple and computationally efficient. Helps mitigate vanishing gradients.	Can suffer from dying ReLU problem (neurons becoming inactive).

OR GATE


It can be used to implement Logic Gates.
OR
X1 X2 Y
0 0 0
0 1 1
1 0 1
1 1 1
X1 X2
0 0
0 1
1 0
1 1
W = 1
W = 1
t = 0.5
X1
1
0
1
X2

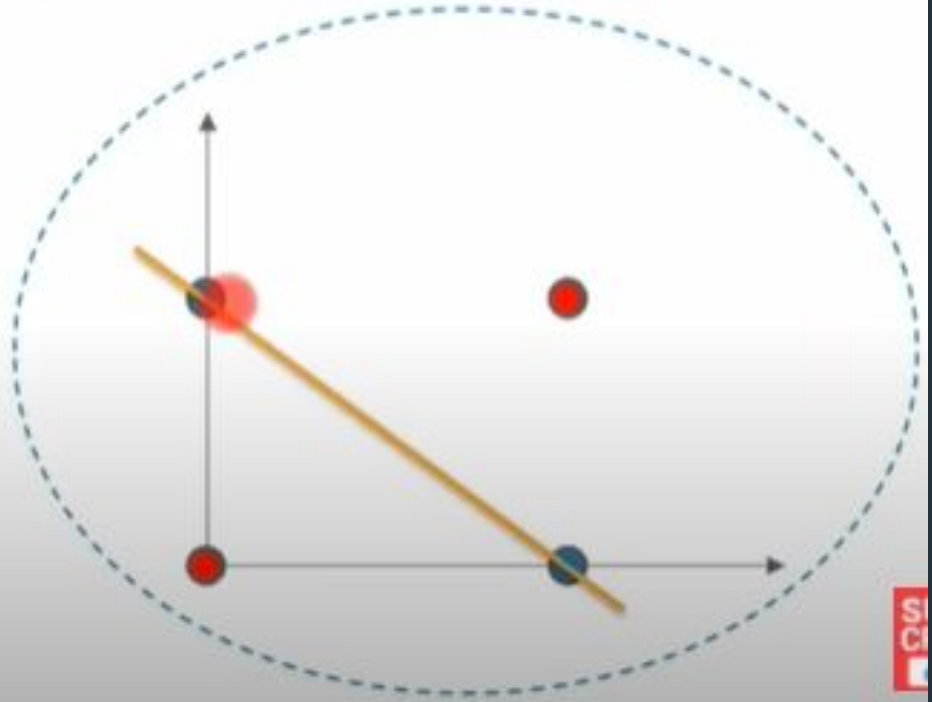
AND GATE


It can be used to implement Logic Gates.
AND
X1 X2 Y
0 0 0
0 1 0
1 0 0
1 1 1
X1 X2
0 0
0 1
1 0
1 1
W = 1
W = 1
t = 1.5
X1
1
0
1
X2

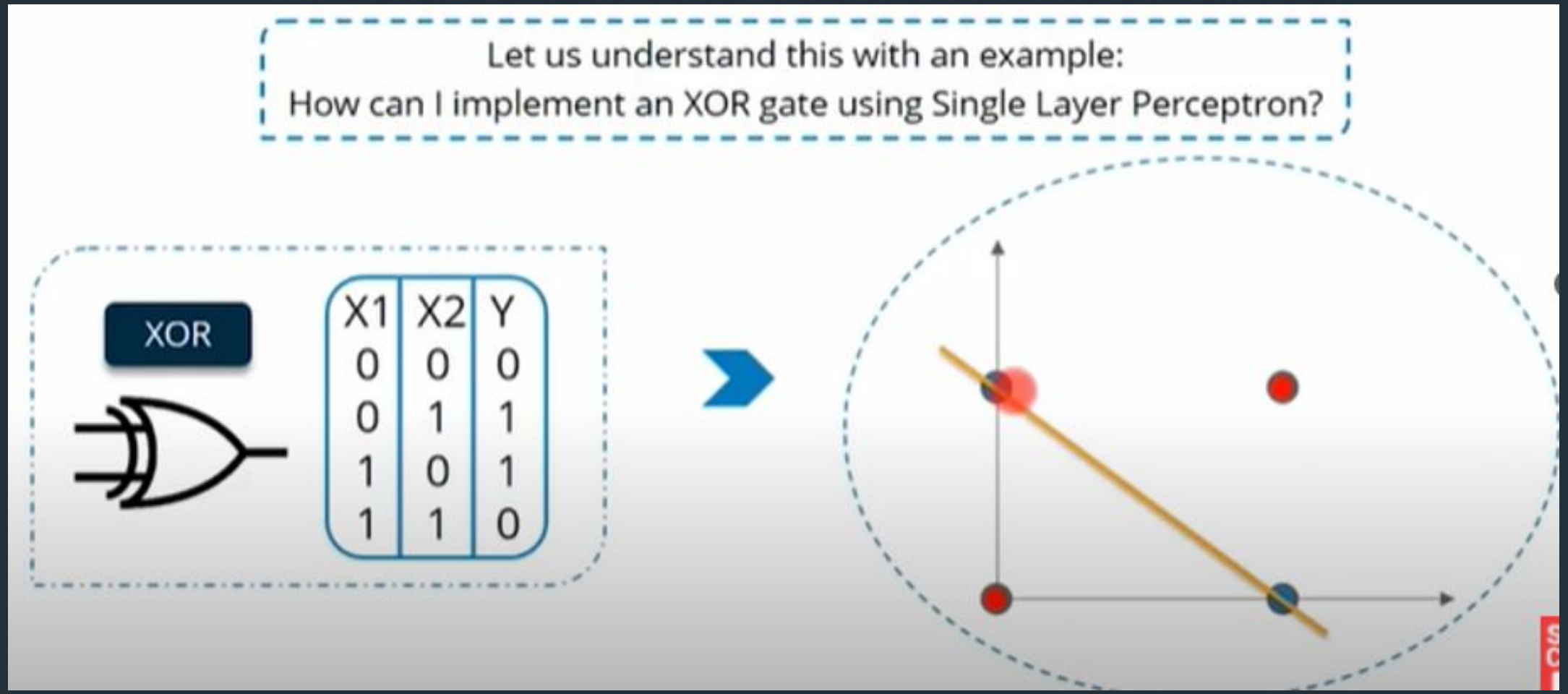
Limitations Of Single Layer Perceptron

Let us understand this with an example:
How can I implement an XOR gate using Single Layer Perceptron?

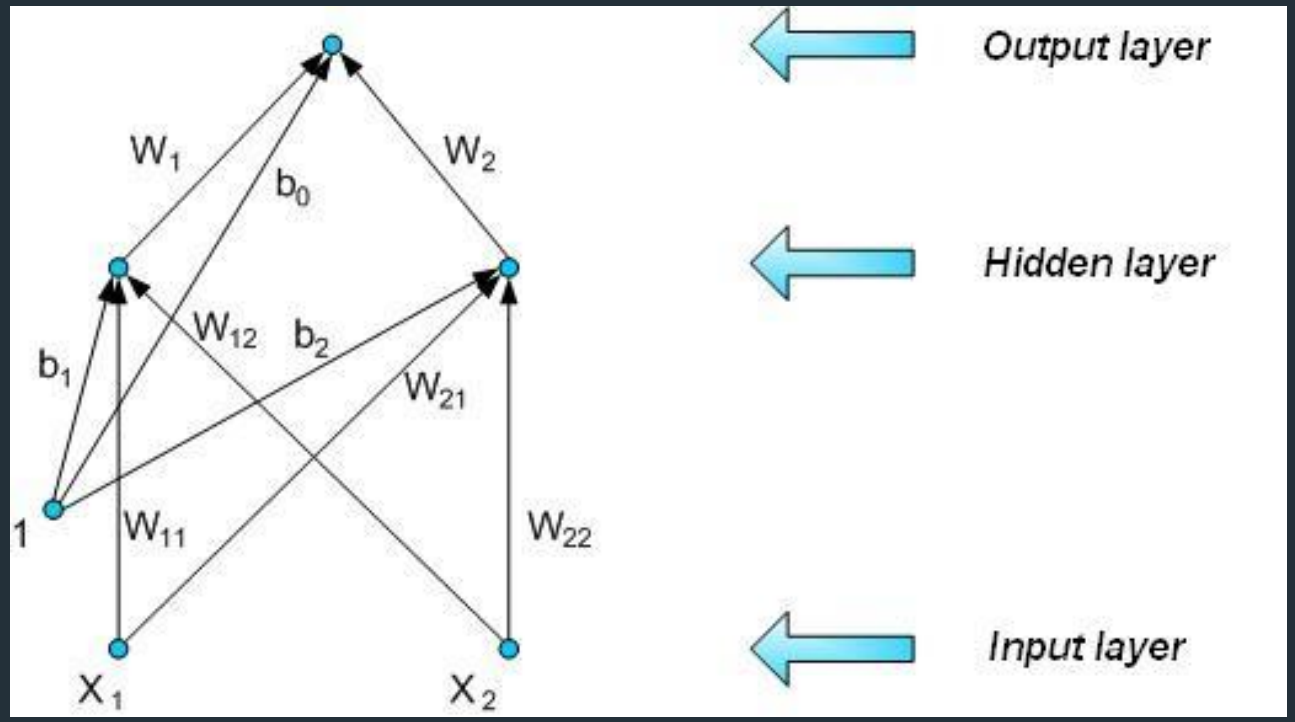
X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0



Limitation of Single Layer Perceptron

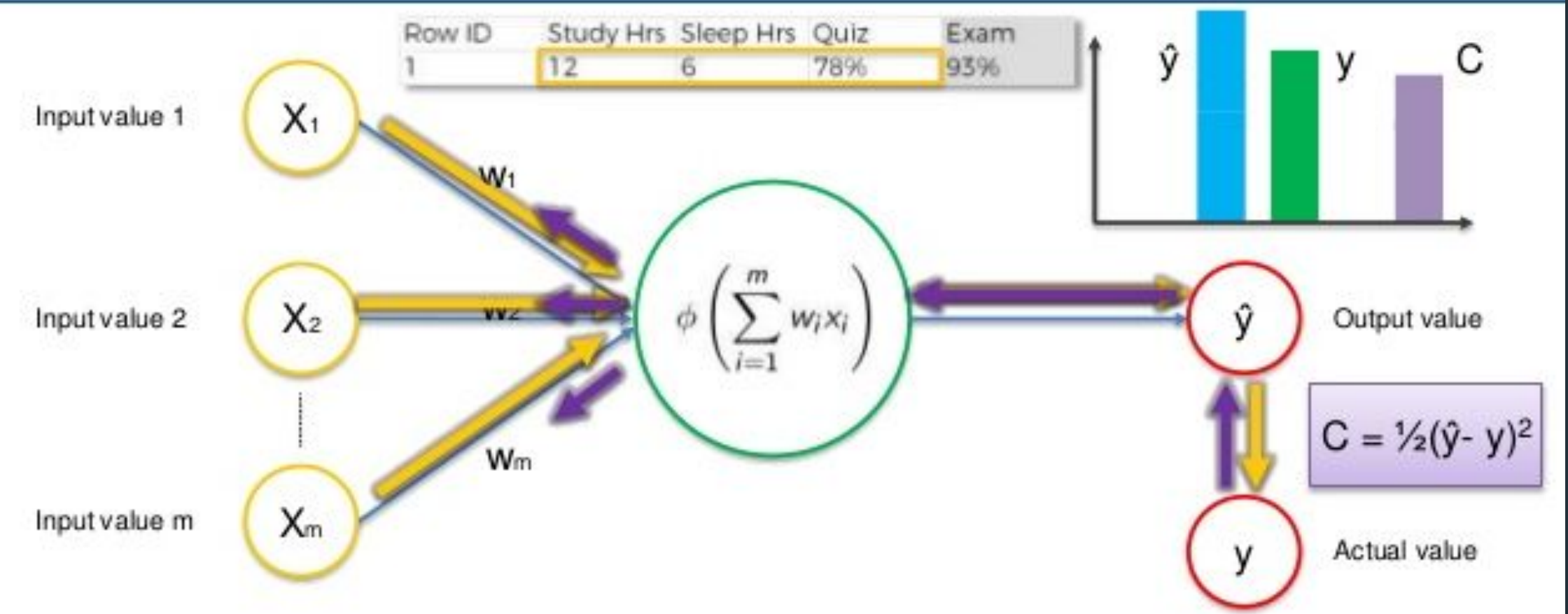


Structure of a network that has ability to implement XOR function

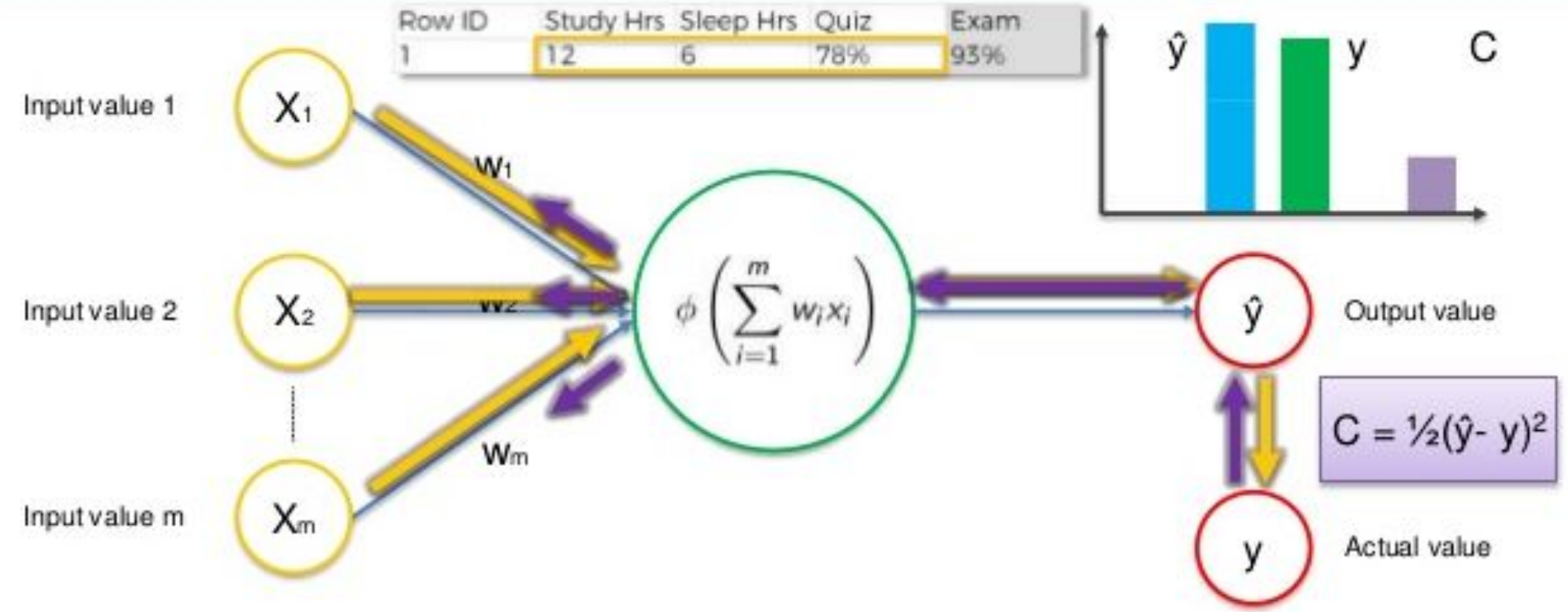


https://colab.research.google.com/drive/1uAeKetBR-ql_rqLXf942ziZXiZwih7ue?authuser=1

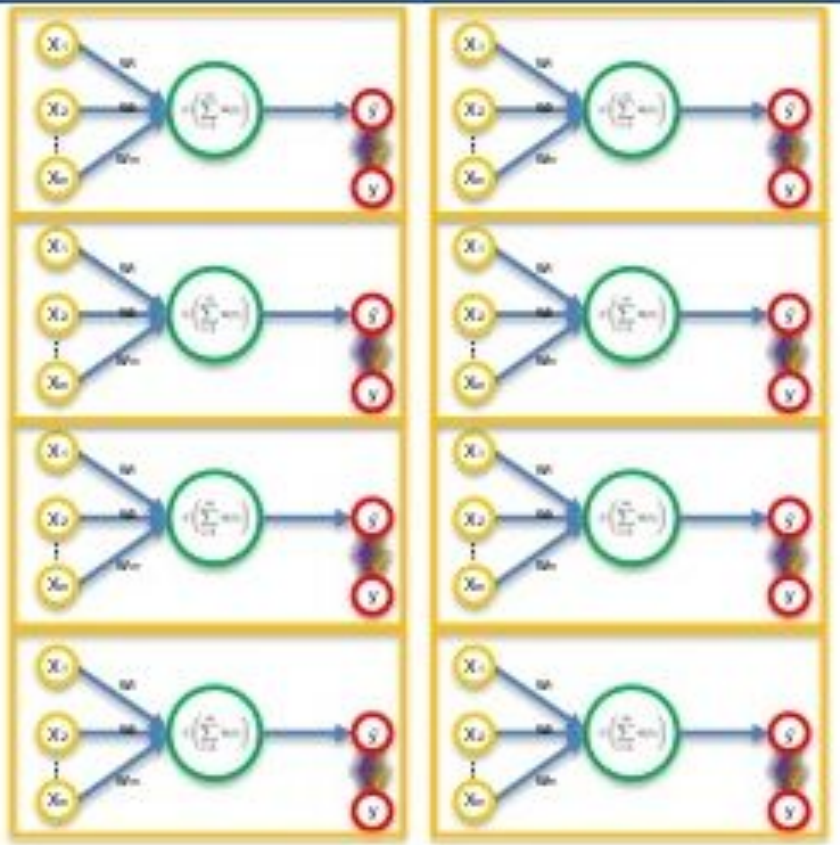
How do Neural Networks learn?



How do Neural Networks learn?



How do Neural Networks learn?



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

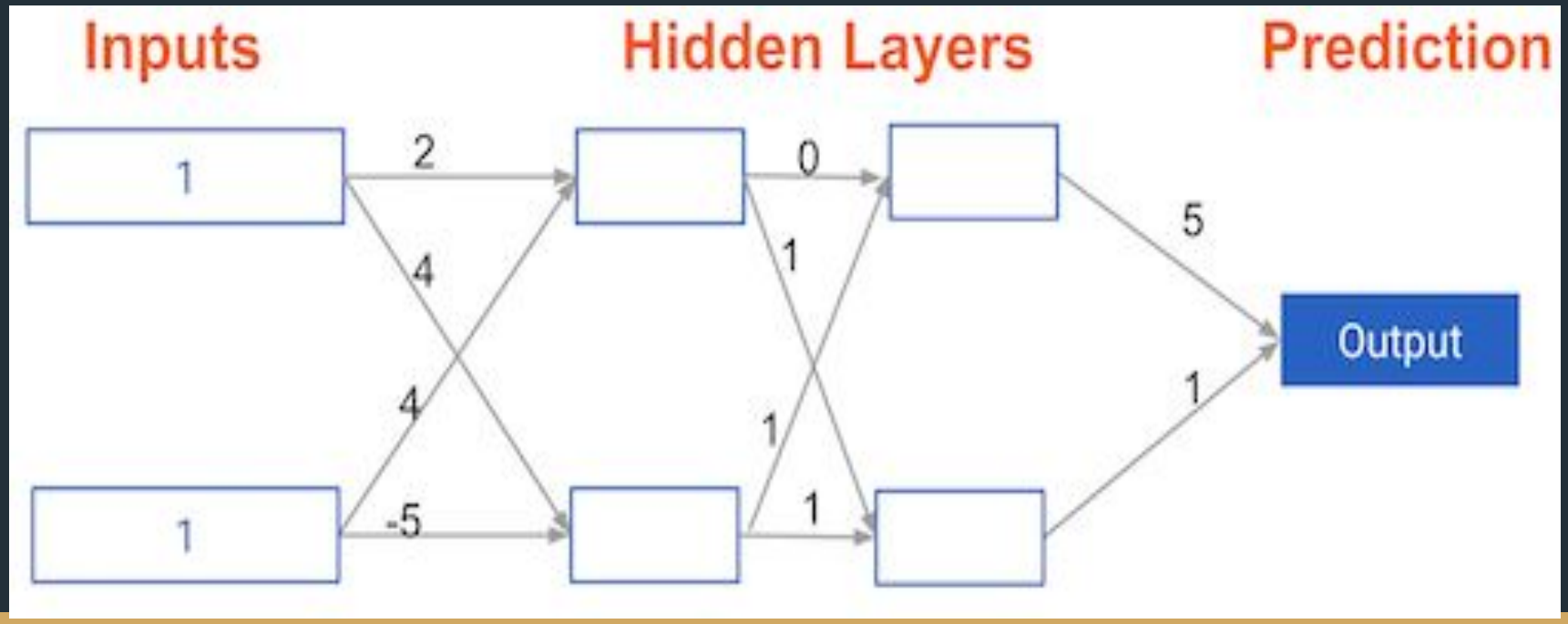
$$C = \sum \tfrac{1}{2}(\hat{y} - y)^2$$

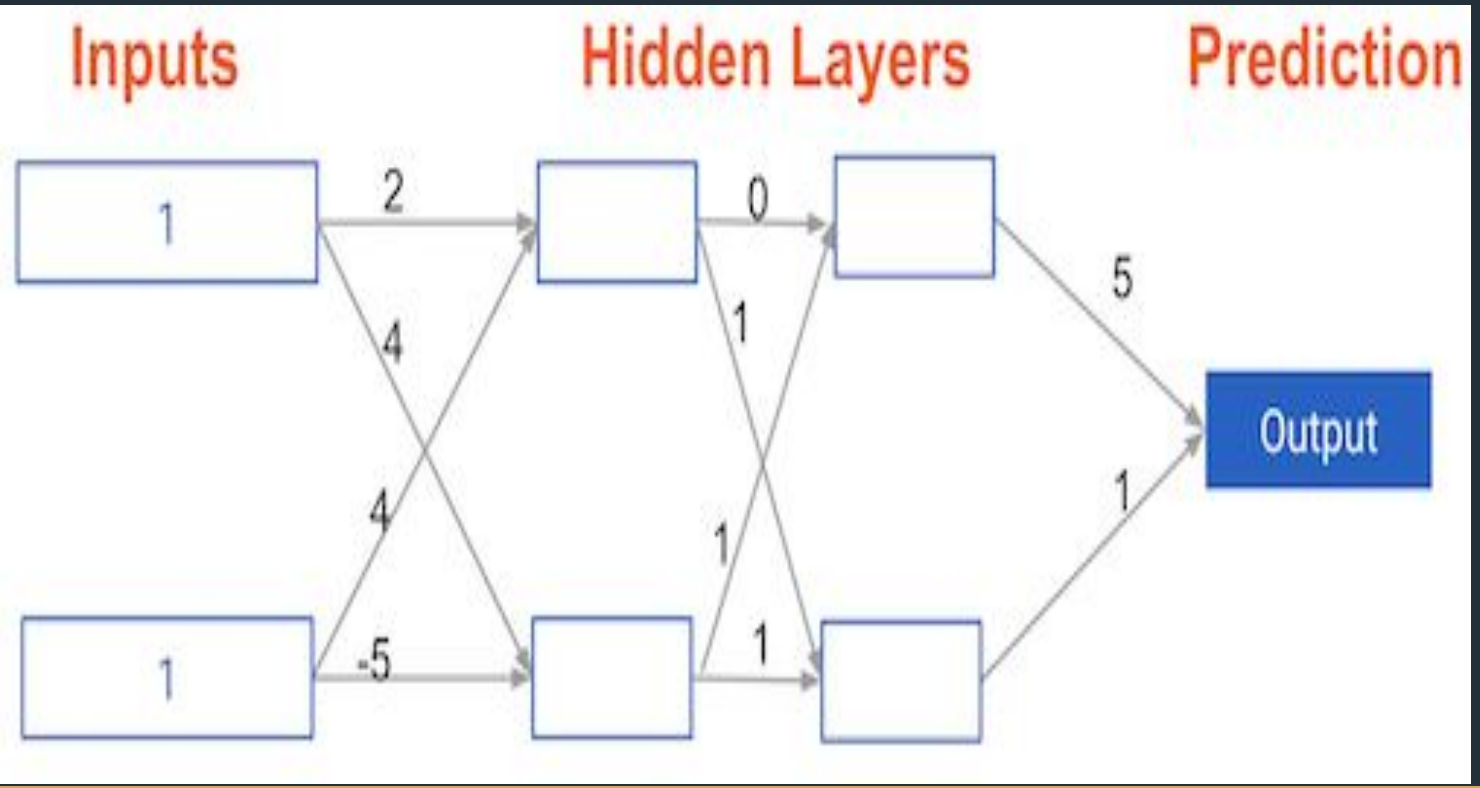
Handson Session-1

Use vector and matrix-based functions rather than for loops, whenever possible

- np.exp(v)
- np.log(v)
- np.abs(v)
- np.maximum(v,0)
- v**2
- 1/v: element-wise inverse
- . . .

- You now have a model with 2 hidden layers. The values for an input data point are shown inside the input nodes. The weights are shown on the edges/lines. What prediction would this model make on this data point?
- Assume the activation function at each node is the identity function. That is, each node's output will be the same as its input. So the value of the bottom node in the first hidden layer is -1,.





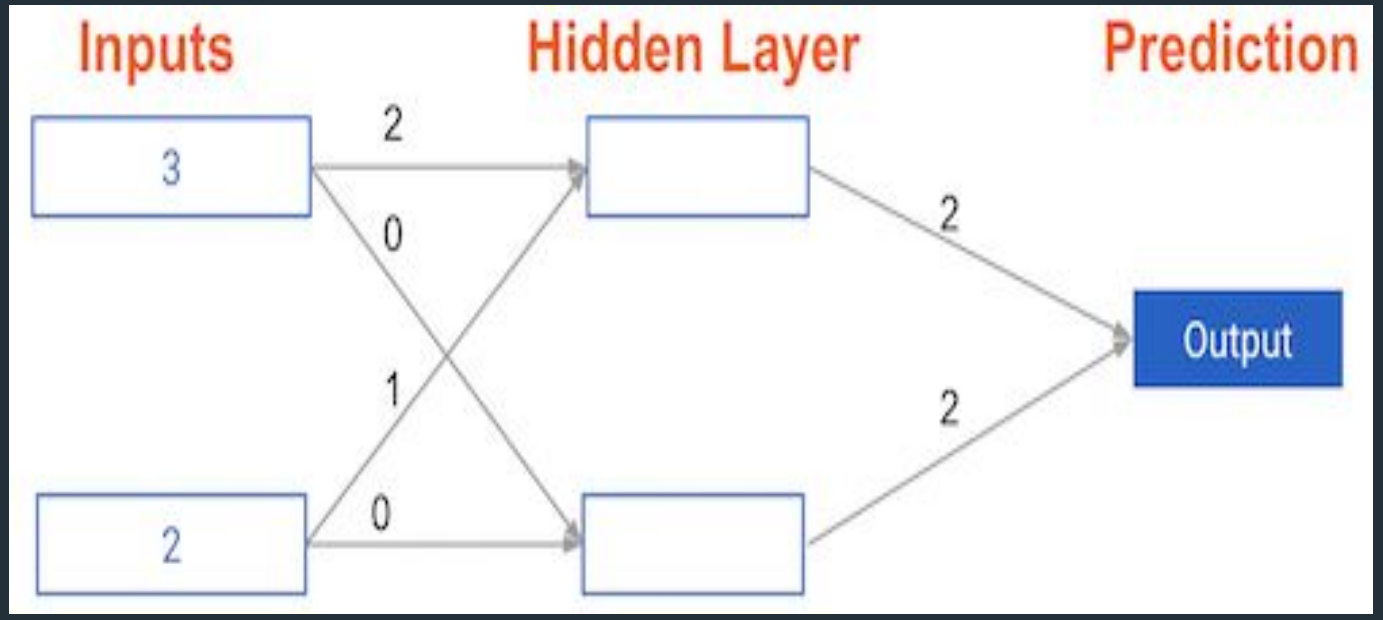
Hidden layer-1
1*2+1*4=6
1*-5+1*4=-1

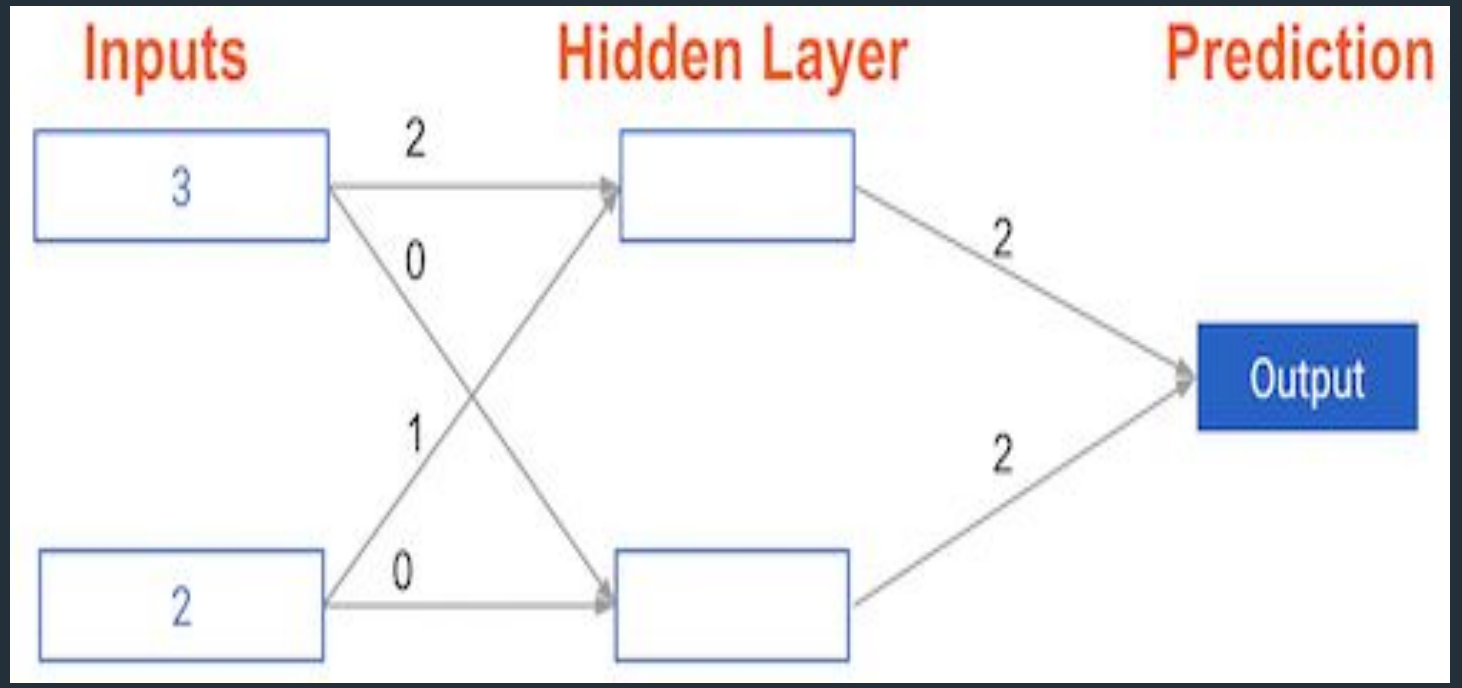
Hidden layer-2
6*0+-1*1=-1
6*1+-1*1=5

Output
-1*5+5*1

Solution:
0

What is the error (predicted – actual) for the following network using the ReLU activation function when the input data is [3, 2] and the actual value of the target (what you are trying to predict) is 5?

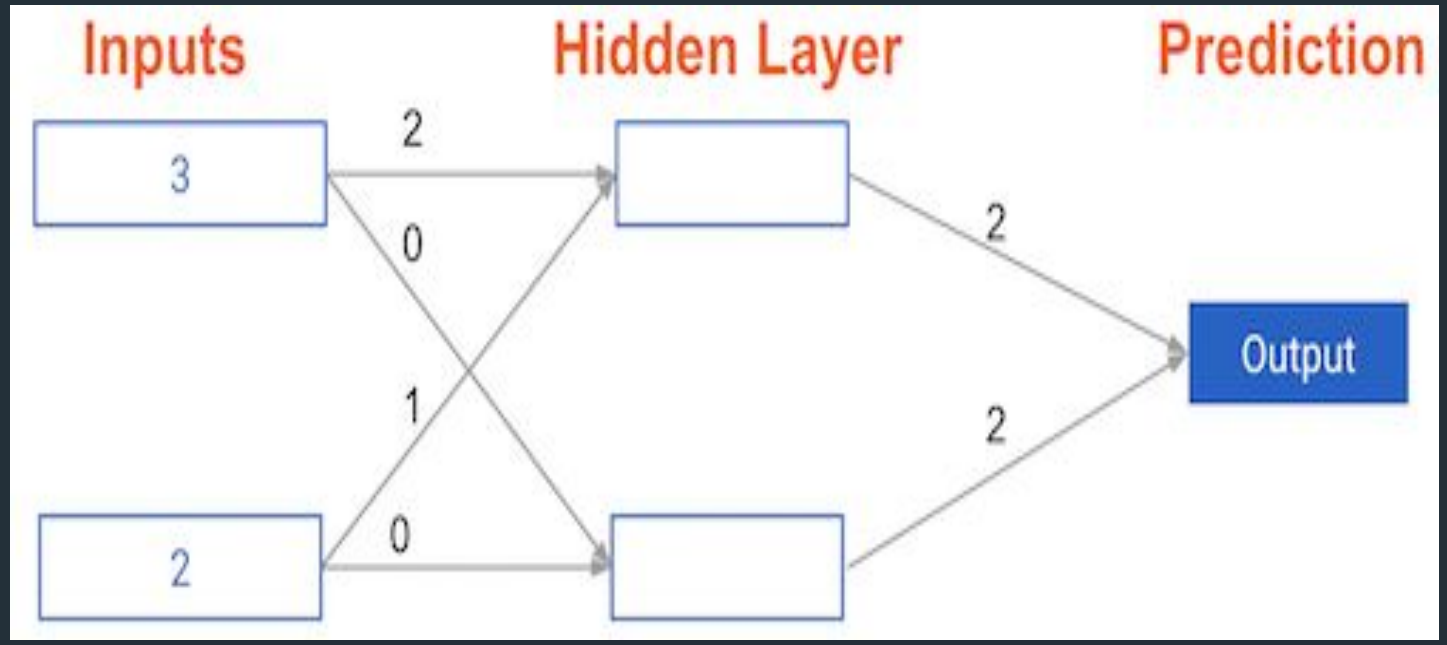




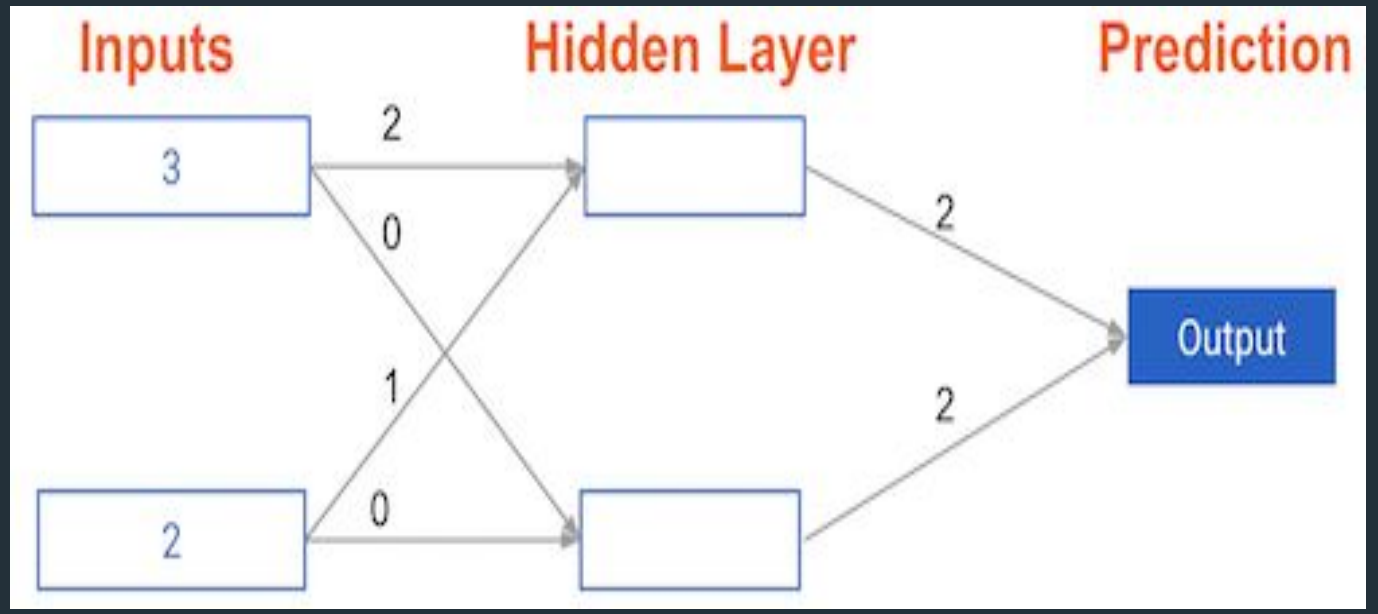
Hidden layer
3*2+2*1=8
3*0+2*0=0
Output layer
8*2+ 0*2=16

error =predicted – actual
16–5
=11

Imagine you have to make a prediction for a single data point. The actual value of the target is 7. The weight going from node_0 to the output is 2, as shown below. If you increased it slightly, changing it to 2.01, would the predictions become more accurate, less accurate, or stay the same?



Imagine you have to make a prediction for a single data point. The actual value of the target is 7. The weight going from node_0 to the output is 2, as shown below. If you increased it slightly, changing it to 2.01, would the predictions become more accurate, less accurate, or stay the same?



Hidden layer

3*2+2*1=8

3*0+2*0=0

Output layer

8*2+ 0*2=16

error =predicted – actual

=16-7

=9

Hidden layer

3*2+2*1=8

3*0+2*0=0

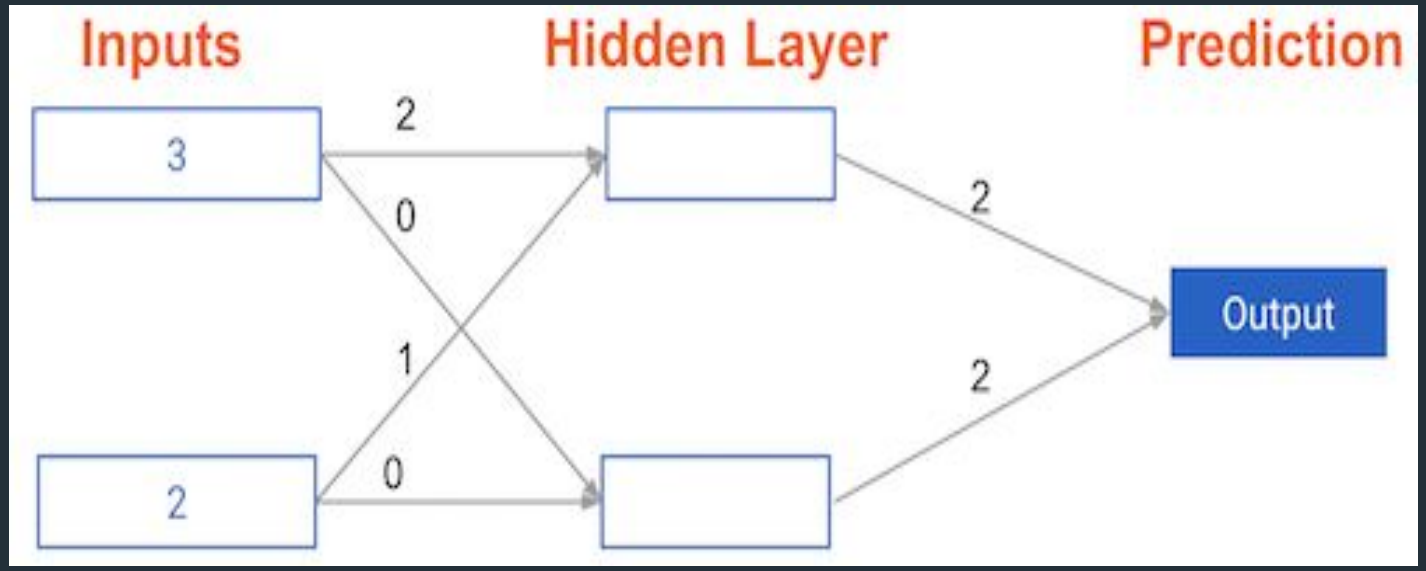
Output layer

8*2.01+ 0*2=16.08

error =predicted – actual

=16.08-7

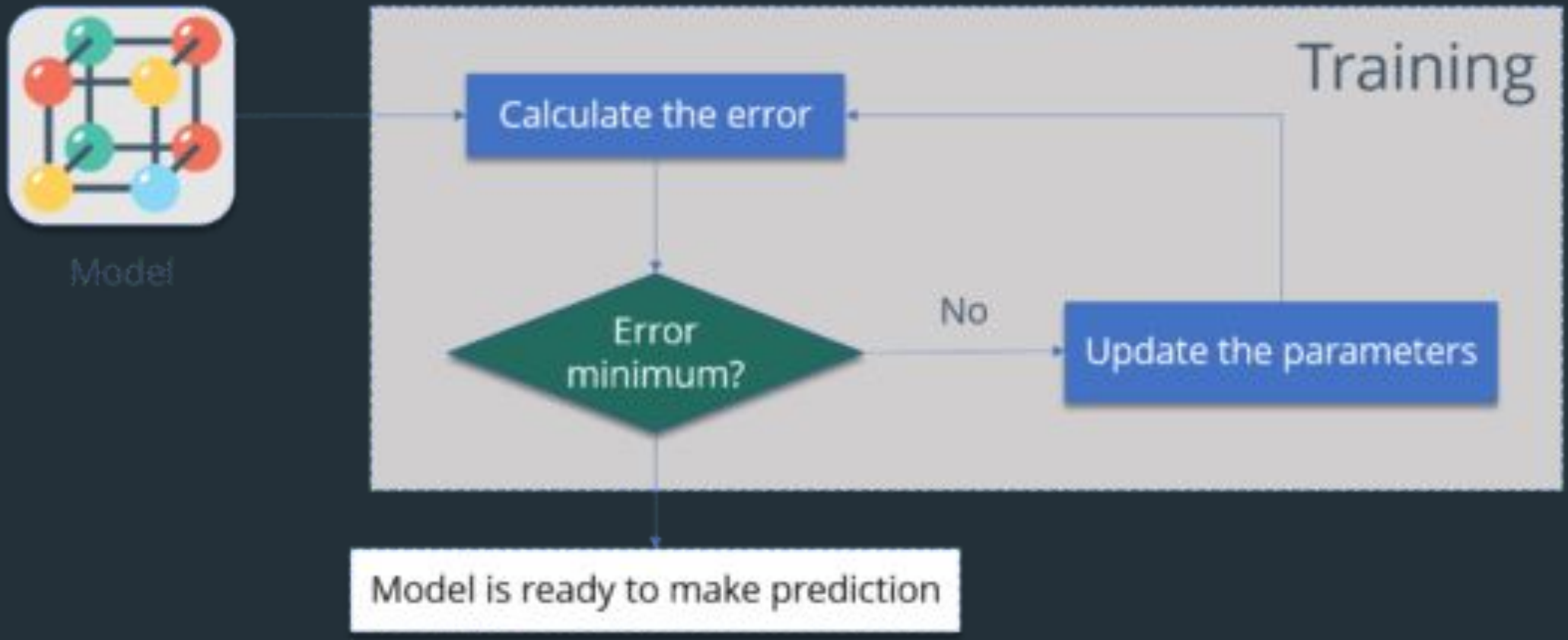
=9.08



Less accurate

Increasing the weight to 2.01 would increase the resulting error from 9 to 9.06, making the predictions less accurate.

Why We Need Backpropagation?

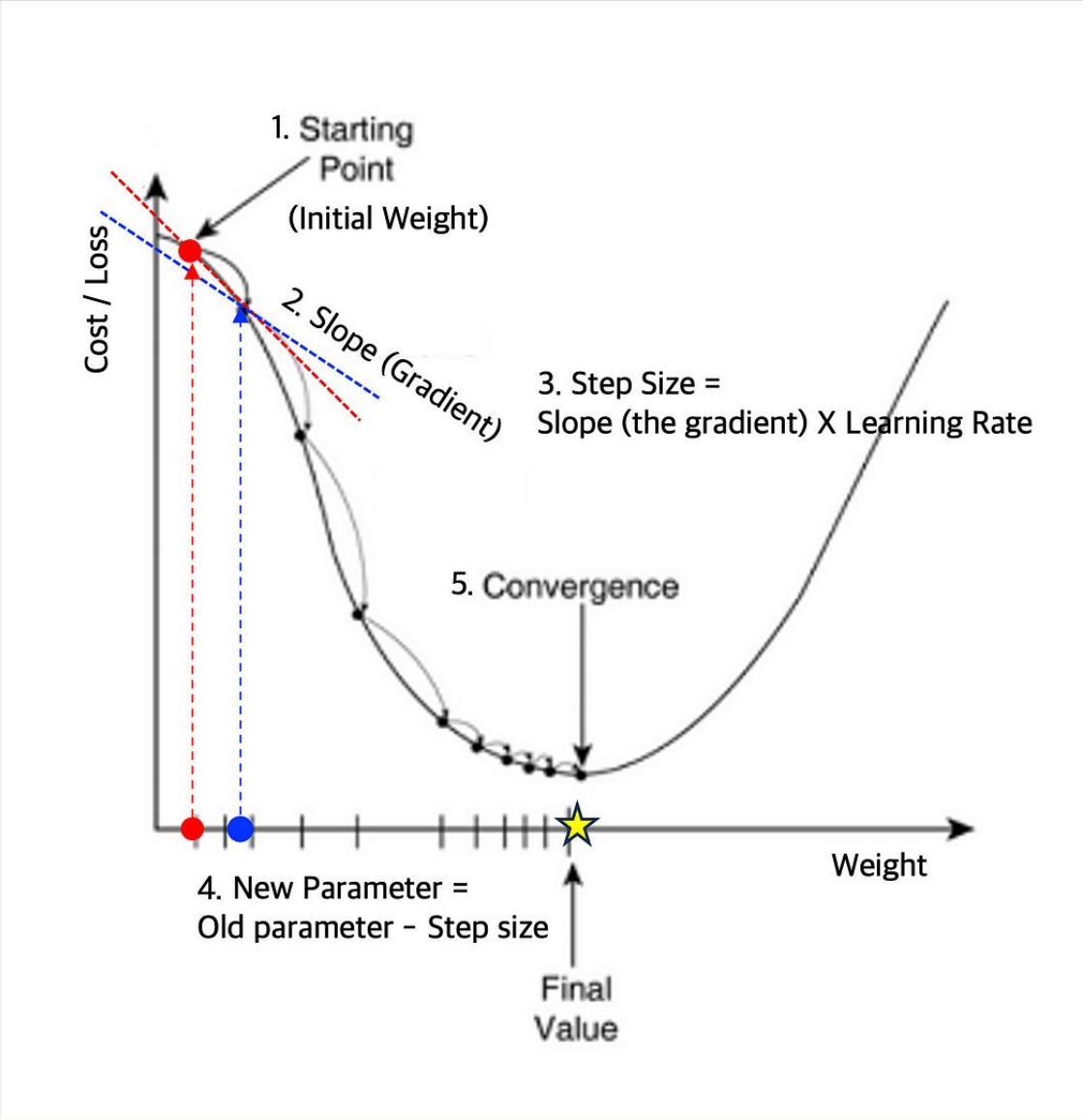


Gradient descent

Gradient Descent (GD) is a fundamental optimization algorithm used in machine learning and neural networks to minimize the cost function (or loss function).

The goal is to find the optimal parameters (weights) of the model that result in the lowest possible value of the loss function.

Gradient descent



Optimization Algorithms

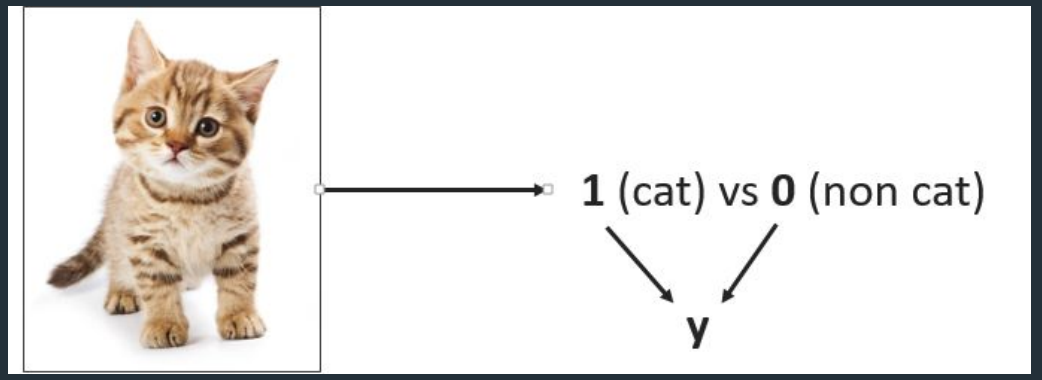
- Gradient Descent (GD): Standard method, suitable for small datasets.
- Stochastic Gradient Descent (SGD): Faster updates, good for large datasets but can be noisy.
- Mini-Batch Gradient Descent: Compromise between GD and SGD, commonly used in practice.
- Momentum-Based GD: Helps with convergence speed and escaping local minima.
- Nesterov Accelerated Gradient (NAG): Provides better performance by considering future gradients.
- AdaGrad: Adaptive learning rates, good for sparse datasets but can slow down training.
- RMSProp: Addresses AdaGrad's shortcomings with a moving average, stable learning rates.
- Adam: Combines benefits of momentum and RMSProp, widely used with good performance.
- AdamW: Enhances Adam with better regularization by decoupling weight decay.

Gradient descent

- Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function.
- The goal of the gradient descent is to minimize a given function which, in our case, is the loss function of the neural network.
- Compute the slope (gradient) that is the first-order derivative of the function at the current point
- Move-in the opposite direction of the slope increase, from the current point, by the computed amount

Binary Classification

- Binary classification is the task of classifying elements of a given set into two groups. Logistic regression is an algorithm for binary classification.
- We have an input image x and the output y is a label to recognize the image. 1 means cat is on an image, 0 means that a non-cat object is in an image.



Logistic regression

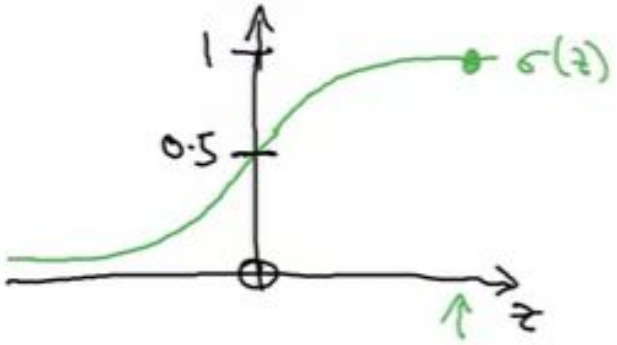
Given x, want $\hat{y} = P(y=1|x)$

$x \in \mathbb{R}^{n_x}$

$0 \le \hat{y} \le 1$

Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$.

Output $\hat{y} = \sigma(\underbrace{w^T x + b}_{z})$



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

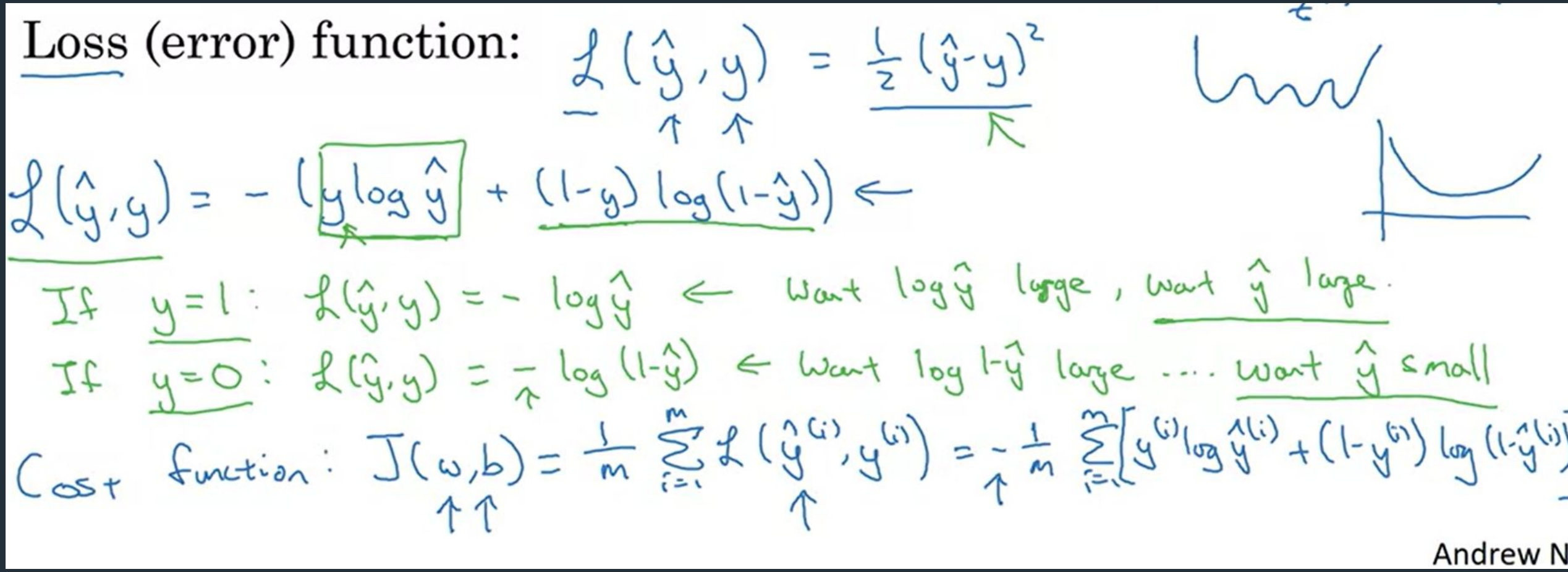
If z large $\sigma(z) \approx \frac{1}{1+0} = 1$

If z large negative number

$$\sigma(z) = \frac{1}{1+e^{-z}} \approx \frac{1}{1+\text{Bignum}} \approx 0$$

Andrew Ng

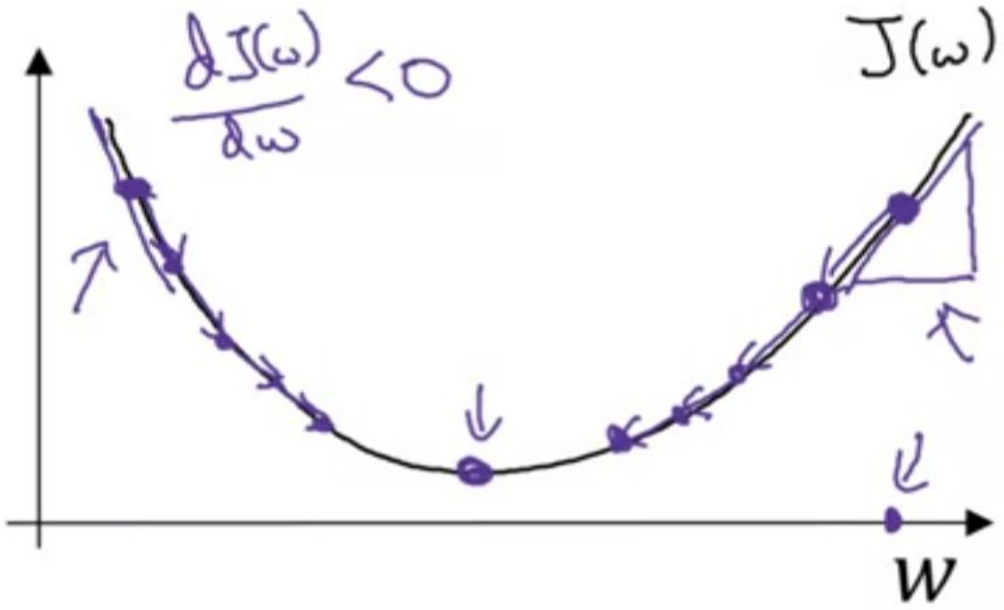
Logistic regression – loss function



Loss function is over 1 training example; cost function, over all the examples

Log loss or binary cross-entropy function

Gradient Descent



Repeat {

$w := w - \alpha \frac{dJ(w)}{dw}$

}

learning rate

"dw"

$w := w - \alpha dw$

$\frac{dJ(w)}{dw} = ?$

$J(w,b)$

$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$

$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$

$\frac{\partial J(w,b)}{\partial w}$

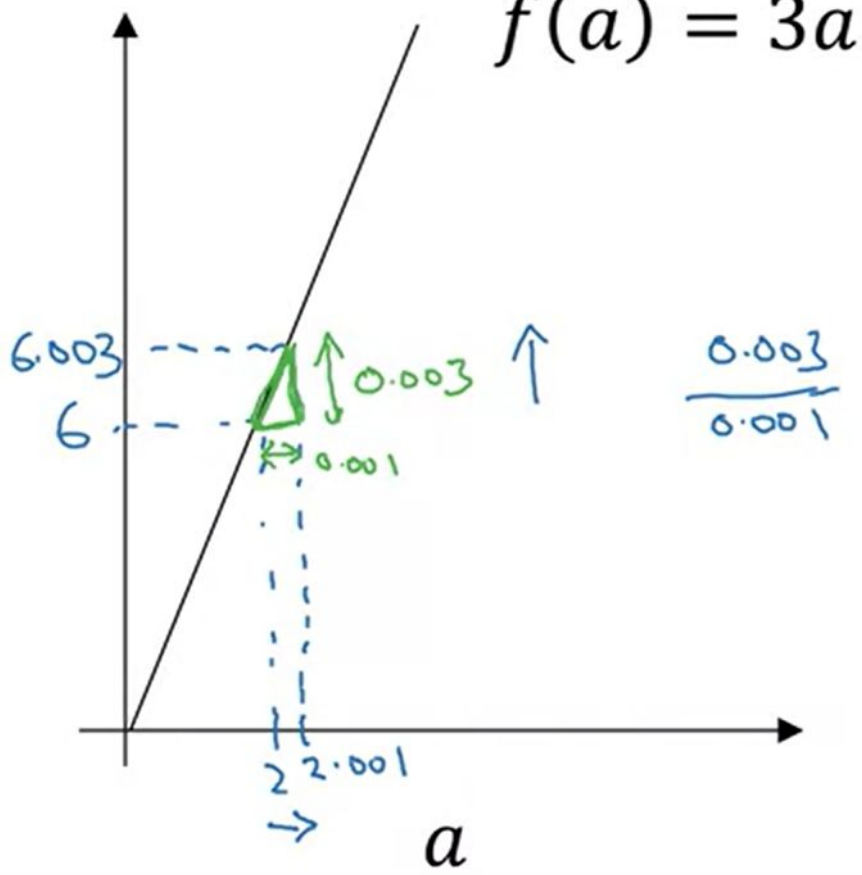
$\frac{\partial J(w,b)}{\partial b}$

∂ "partial derivative"

d ← J

dw

Derivative (slope) of a line



$\frac{0.003}{0.001}$ $\frac{\text{height}}{\text{width}}$

$a = 2$ $f(a) = 6$

$a = 2.001$ $f(a) = 6.003$

slope (derivative) of $f(a)$ at $a = 2$ is 3

$a = 5$ $f(a) = 15$

$a = 5.001$ $f(a) = 15.003$

slope at $a = 5$ is also 3

$$\frac{d\,f(a)}{da} = 3 = \frac{d}{da} f(a)$$

0.001
0.000000001
0.000000000001

Derivative (slope) of a curve

$f(a) = a^2$

$\frac{d}{da} f(a) = \underbrace{2a}_{4}$

$a = 2$ $\quad f(a) = 4$

$a = 2.001$ $\quad f(a) \approx 4.004$

$f(a) = a^3$

$\frac{d}{da} f(a) = \underbrace{3a^2}_{3 \times 2^2 = 12}$

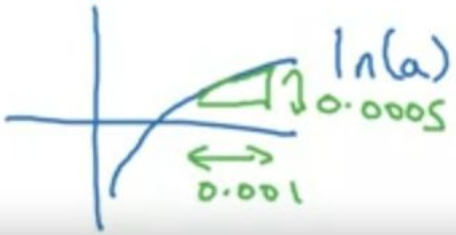
$a = 2$ $\quad f(a) = 8$

$a = 2.001$ $\quad f(a) \approx 8.012$

$f(a) = \log_e(a)$

$\ln(a)$

$\frac{d}{da} f(a) = \frac{1}{a}$



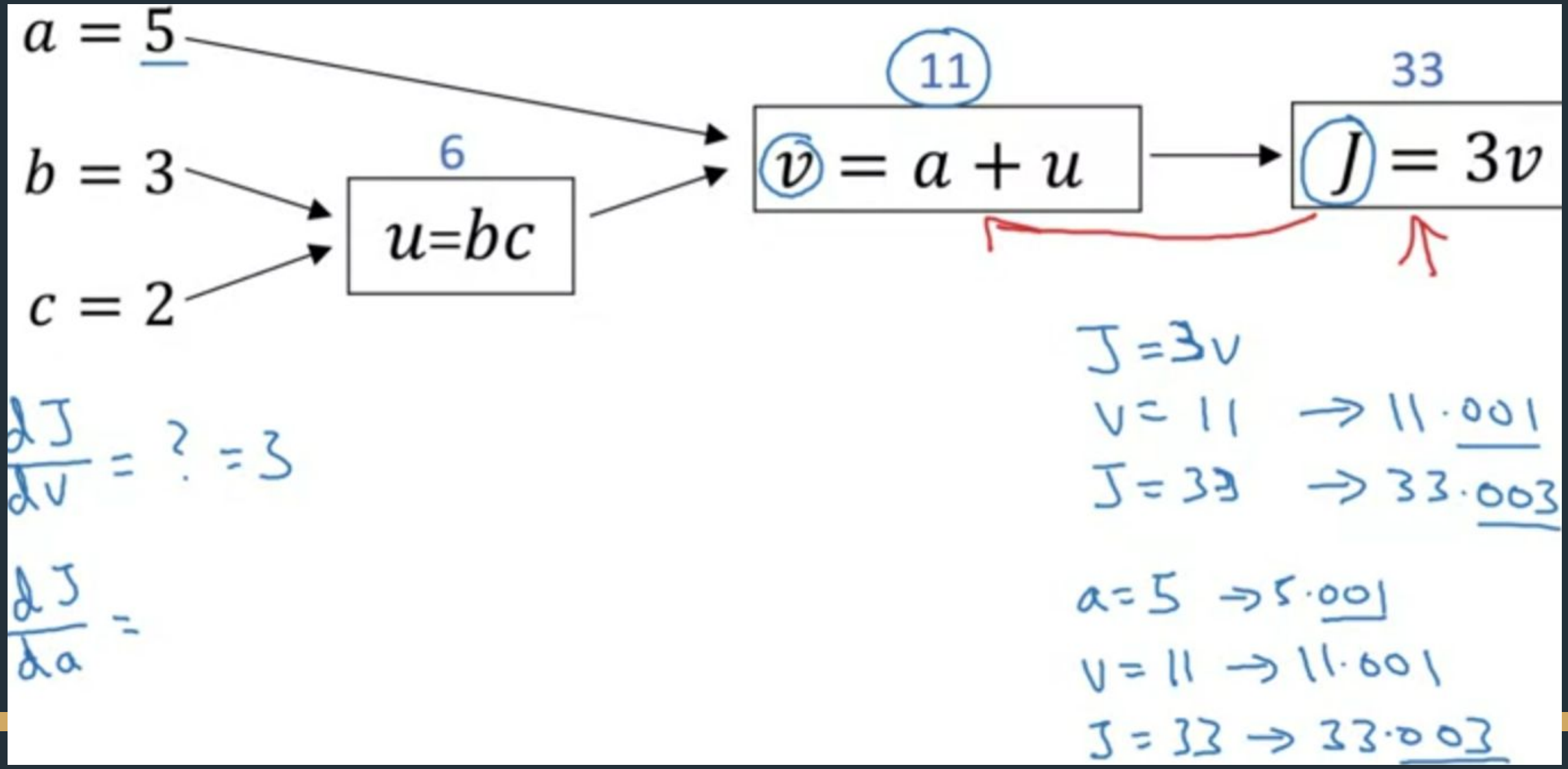
$\frac{d}{da} f(a) = \frac{1}{2}$

$a = 2$ $\quad f(a) \approx 0.69315$

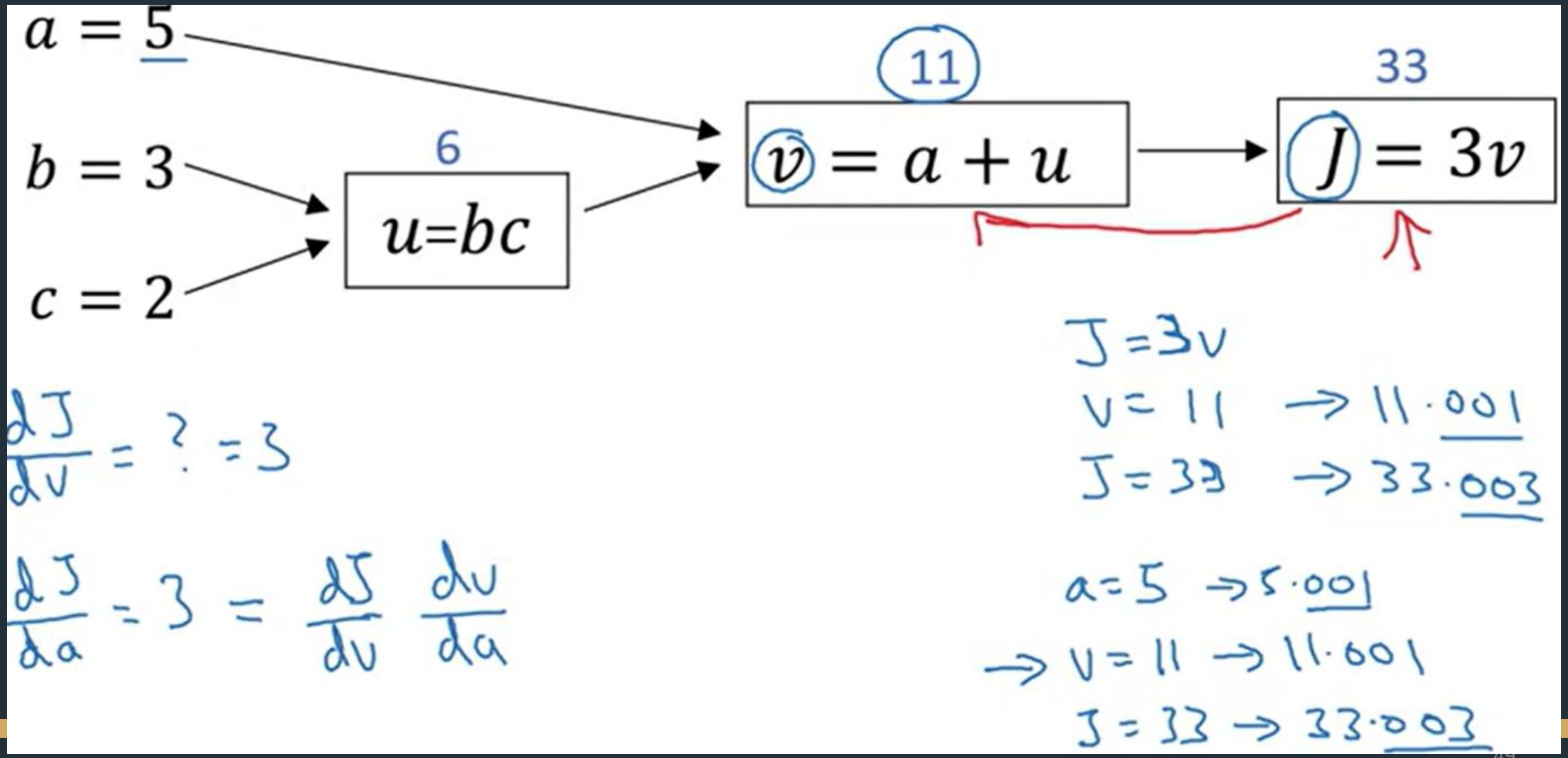
$a = 2.001$ $\quad f(a) \approx 0.69365$

0.0005 $\quad 0.0005$

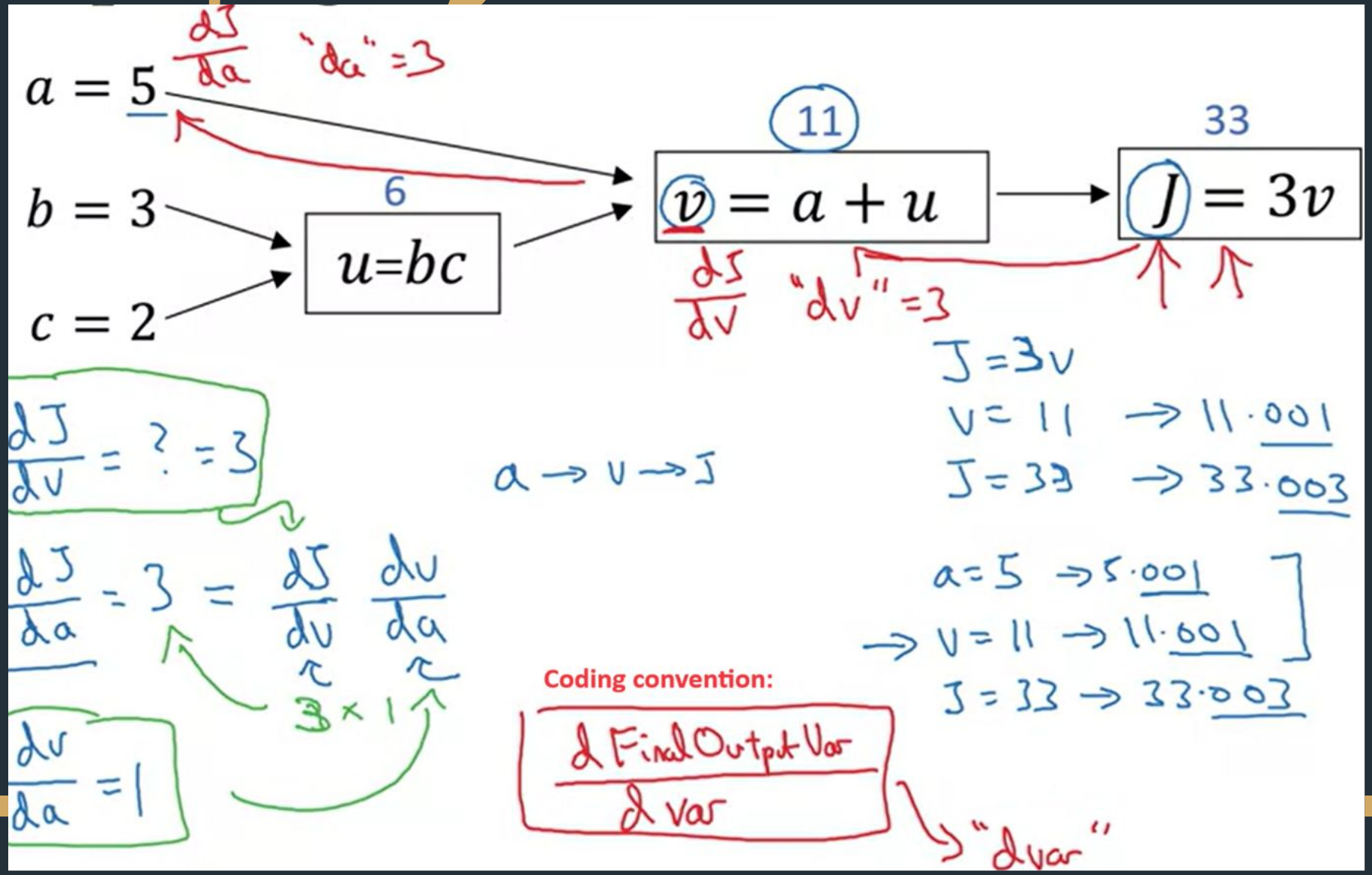
Computing derivatives



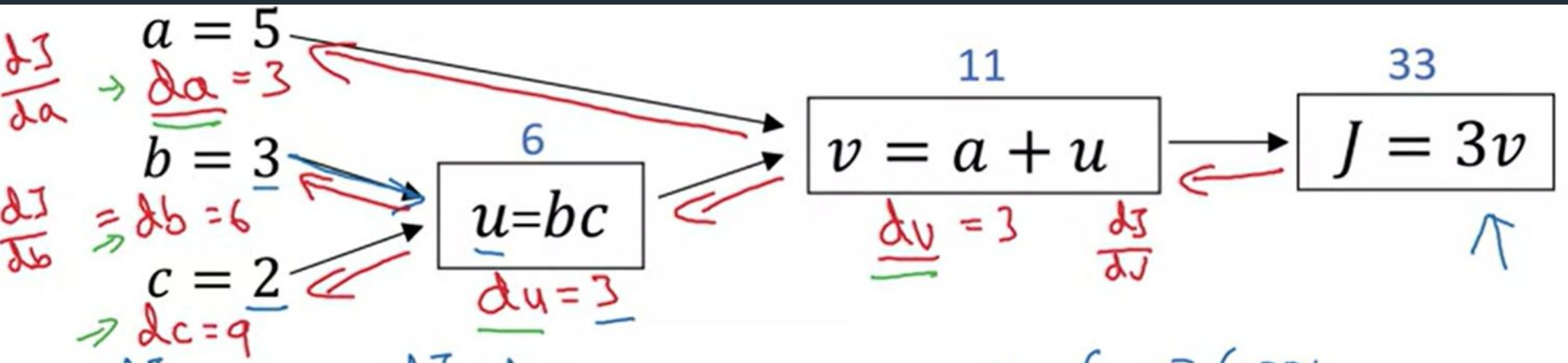
Computing derivatives



Computing derivatives



Computing derivatives cont.



$$\frac{dJ}{du} = 3 = \frac{dJ}{dv}\cdot\frac{dv}{du} \quad (3)\cdot(1)$$

$u = 6 \rightarrow 6.001$

$v = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

$$\frac{dJ}{db} = \frac{dJ}{du}\cdot\frac{du}{db} = 6 \quad (3)\cdot(2)$$

$b = 3 \rightarrow 3.001$

$u = b\cdot c = 6 \rightarrow 6.002$

$J = 33.006$

$c = 2$

$.006$

$$\frac{dJ}{dc} = \frac{dJ}{du}\cdot\frac{du}{dc} = 9 \quad (3)\times(3)$$

$V = 11.002$

$J = 3V$

Andrew Ng

Logistic regression – gradient descent on m examples

$J = 0,\ dw1 = 0,\ dw2 = 0,\ db = 0$

for i = 1 to m:

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

for j = 1 ... n_x:

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$ | $n_x = 2$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

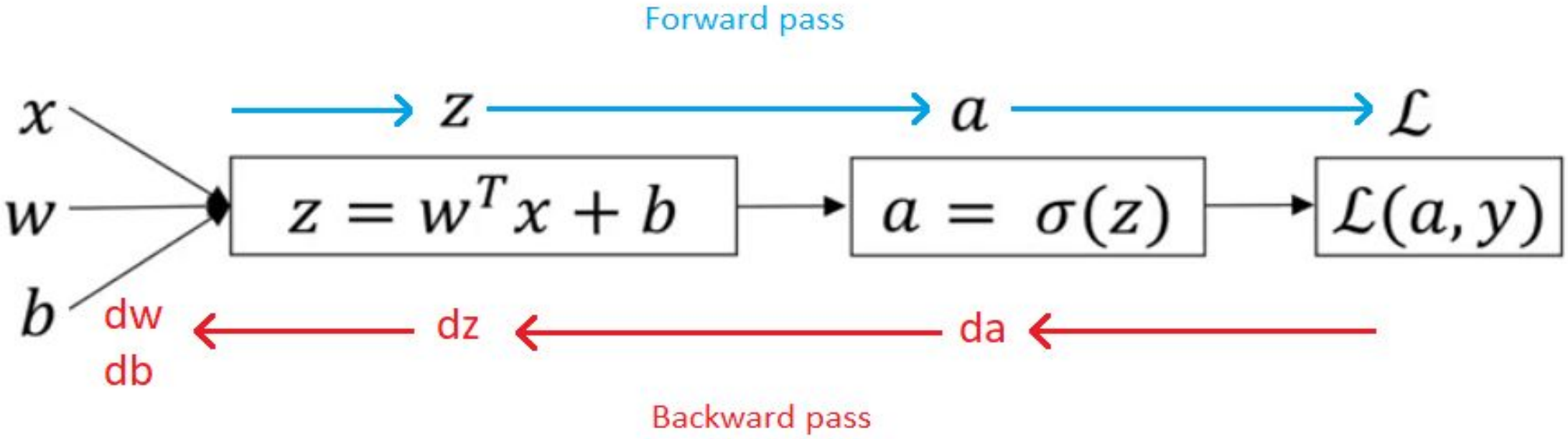
$J = J/m,\ dw_1 = dw_1/m,\ dw_2 = dw_2/m,\ db = db/m$

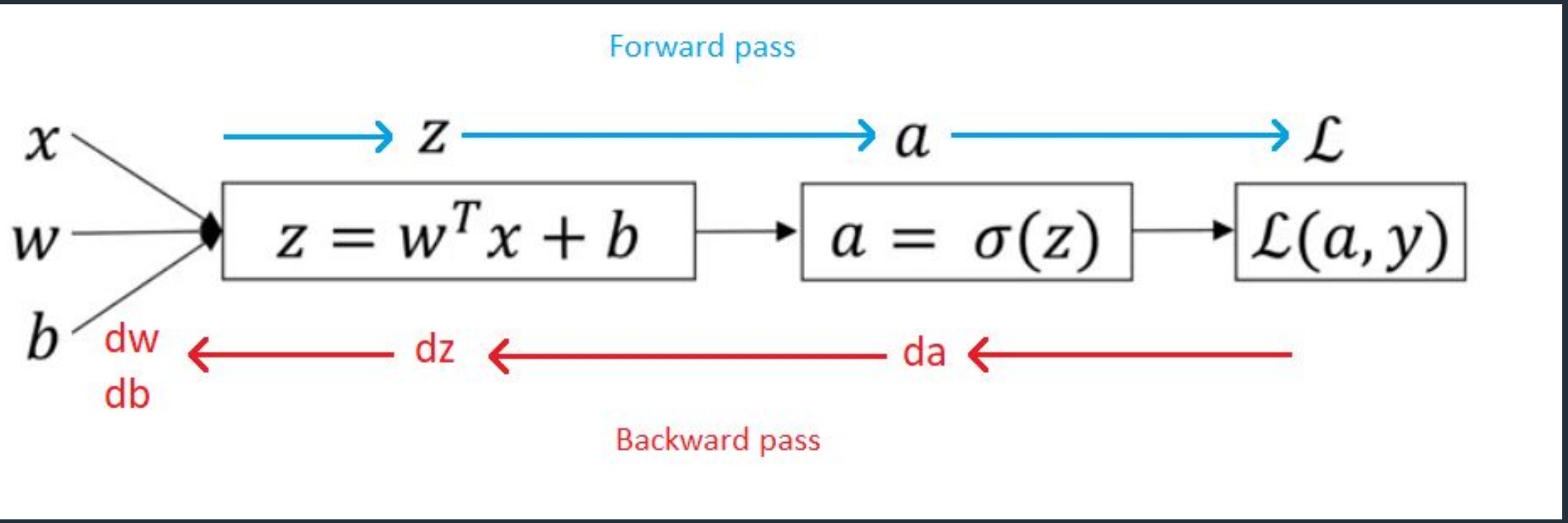
$dw_1 = \frac{\partial J}{\partial w_1}$

$w_1 := w_1 - \alpha\, dw_1$

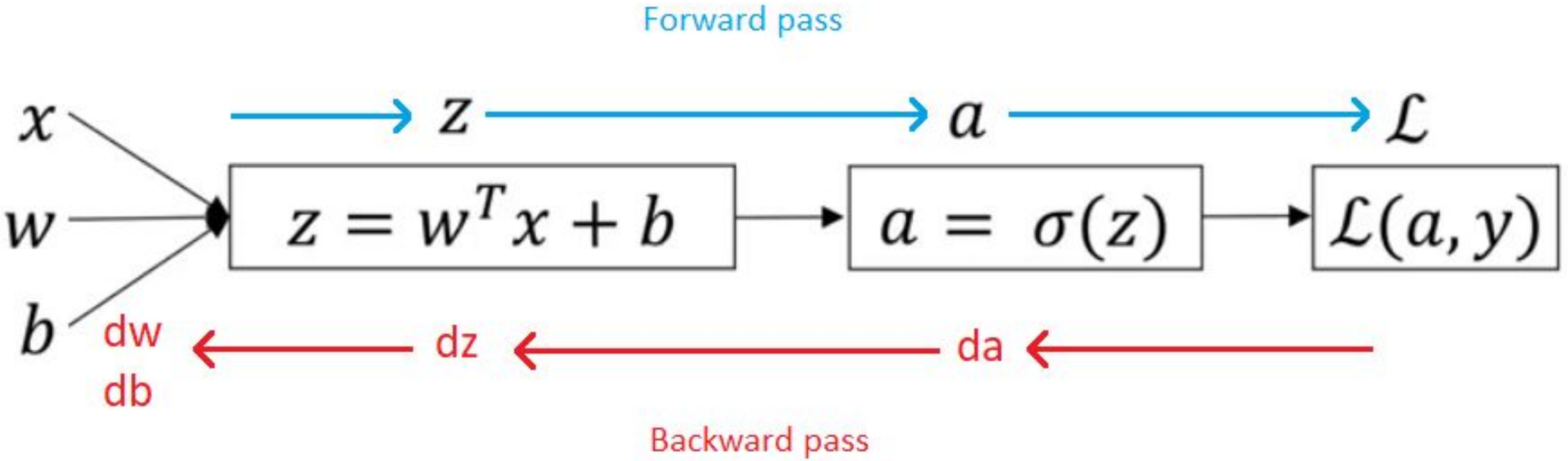
$w_2 := w_2 - \alpha\, dw_2$

$b := b - \alpha\, db$

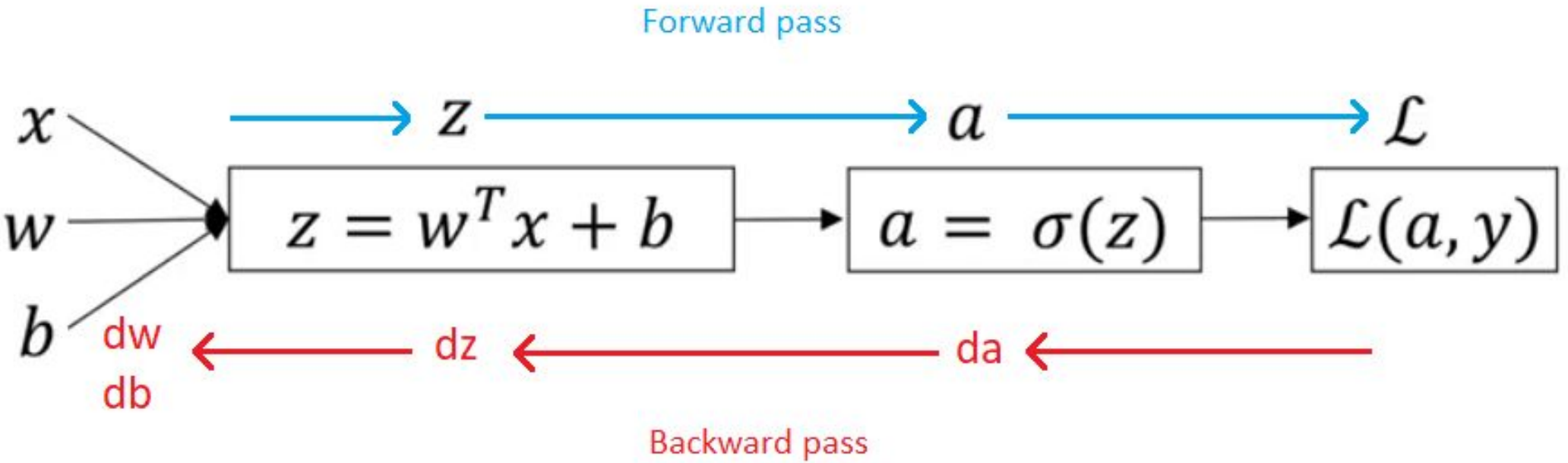

Forward pass
x
w
b
z
a
ℒ
z = w^T x + b
a = σ(z)
ℒ(a, y)
dw
db
dz
da
Backward pass



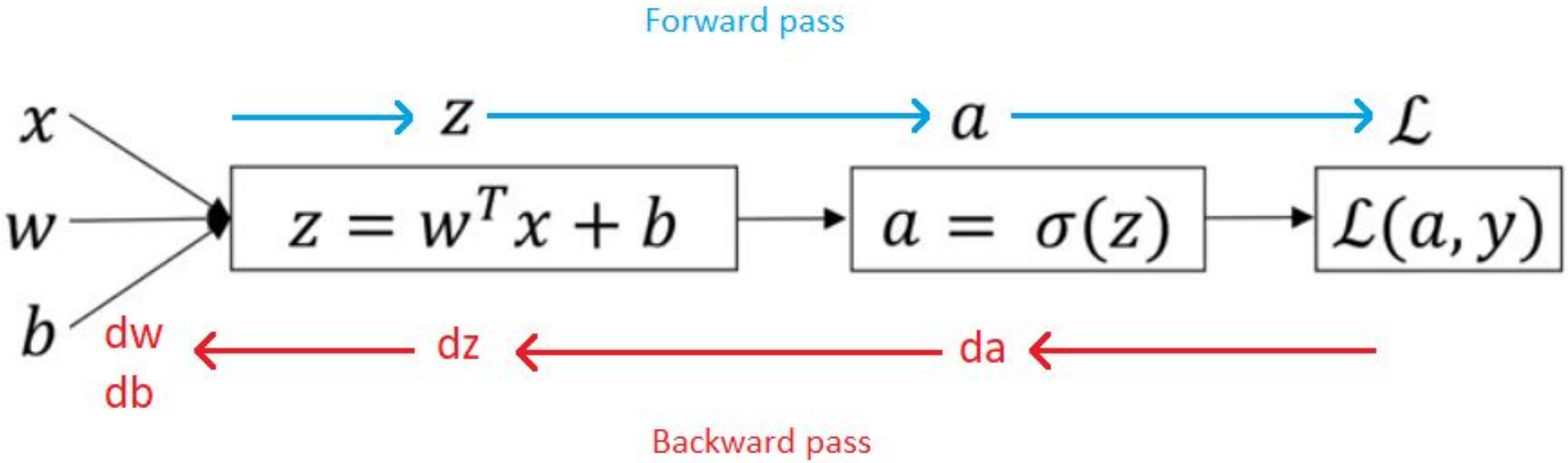
$$\mathcal{L}(a = \hat{y}, y) = - \; yloga - (1 - y) \; log(1 - a)$$



$$da = -\frac{y}{a} + \frac{1-y}{1-a}$$

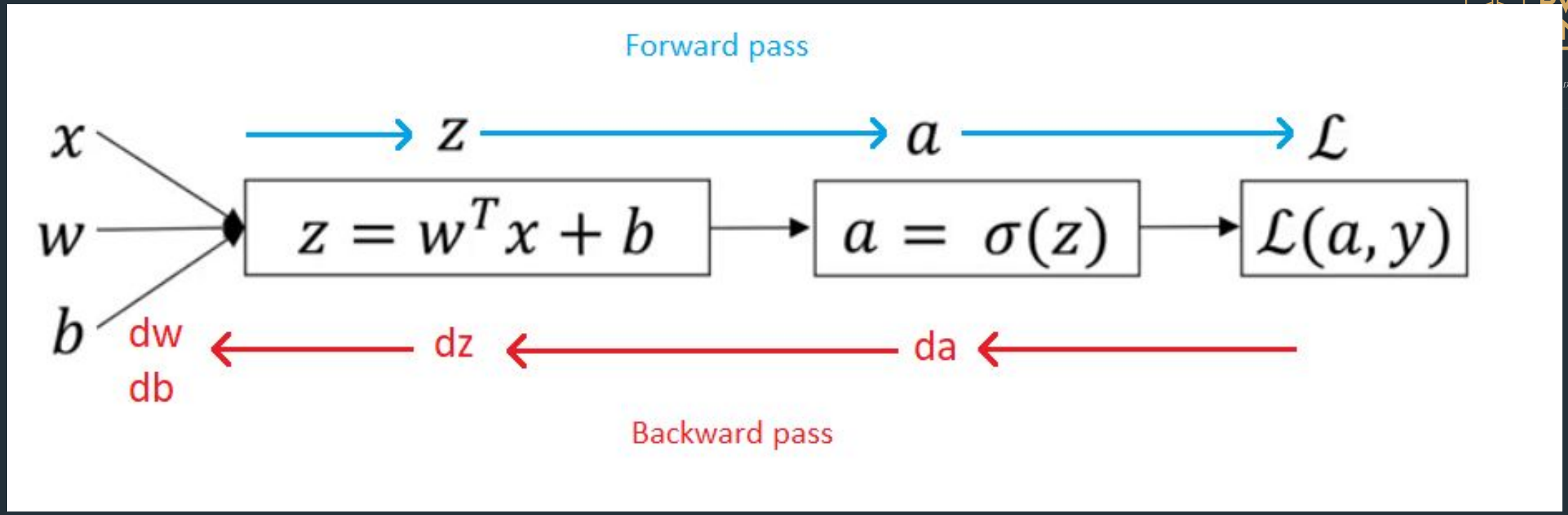


$$dz = (-\frac{y}{a} + \frac{1-y}{1-a})a(1-a) = a - y$$


Forward pass
x
w
b
z
a
$\mathcal{L}$
$z = w^T x + b$
$a = \sigma(z)$
$\mathcal{L}(a, y)$
dw
db
dz
da
Backward pass

$$dw = dz \; x$$

$$db = dz$$



w= w-learning_rate * dw
b = b- learning_rate * db

Handson Session-2

Logistics regression (Gradient Descent)

Stochastic Gradient Descent (SGD)

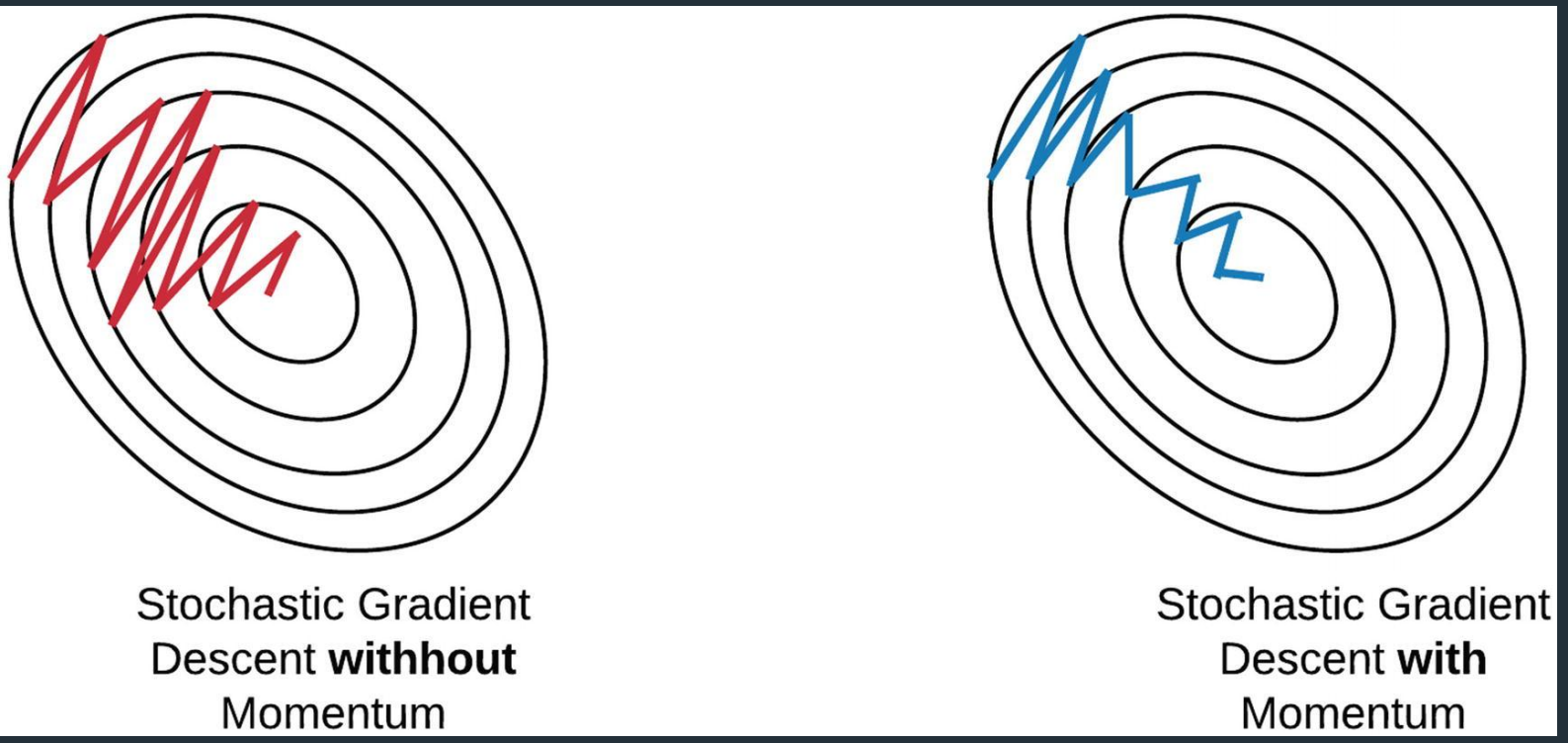
Updates weights based on the gradient of a single training example.

Mini-Batch Gradient Descent

Updates weights based on the gradient of a subset of the training dataset (mini-batch).

Momentum-Based GD

- Enhances SGD by adding a momentum term that accelerates updates in the relevant direction and reduce oscillations.



Optimization Algorithm	Description	Advantages	Disadvantages
Gradient Descent (GD)	Updates weights based on the average gradient of the entire dataset.	Simple and easy to implement.	Can be very slow to converge for large datasets. Requires the entire dataset for each update.
Stochastic Gradient Descent (SGD)	Updates weights based on the gradient of a single training example.	Faster updates due to smaller computations per update. Can escape local minima better.	Noisy updates can lead to unstable convergence. Requires careful tuning of learning rates.
Mini-Batch Gradient Descent	Updates weights based on the gradient of a subset of the training dataset (mini-batch).	Balances the efficiency of GD and the fast updates of SGD. Better convergence than full-batch GD.	Mini-batch size needs to be tuned. Performance can vary based on batch size and learning rate.
Momentum-Based GD	Enhances SGD by adding a momentum term that accelerates updates in the relevant direction and dampens oscillations.	Helps accelerate gradients vectors in the right directions, thus leading to faster converging.	Can overshoot minima if the momentum term is too high. Requires tuning of both learning rate and momentum parameter.
Nesterov Accelerated Gradient (NAG)	A variant of momentum where the gradient is calculated after a "look-ahead" step.	Provides a more accurate update by taking future gradients into account. Often leads to faster convergence.	More complex to implement. Requires careful tuning of learning rate and momentum.
AdaGrad	Adjusts learning rates for each parameter based on historical gradients.	Adaptive learning rates improve performance on sparse data.	Learning rates can become too small and stop updating early in training.
RMSProp	A variant of AdaGrad that uses a moving average of squared gradients to normalize the gradient.	Maintains a more stable learning rate by addressing AdaGrad's diminishing learning rate problem.	Requires tuning of decay rate and learning rate.
Adam	Combines ideas from Momentum and RMSProp. Uses estimates of first and second moments of gradients to adapt learning rates.	Often works well out-of-the-box with minimal hyperparameter tuning. Fast convergence and robust to noisy data.	Can have issues with very noisy gradients. Requires careful tuning of hyperparameters.
AdamW	A variant of Adam that decouples weight decay from the optimization steps.	Provides better regularization by separating weight decay from the learning rate updates.	Slightly more complex to implement compared to Adam.

Different Optimization Algorithms

https://colab.research.google.com/drive/1hapEpLWVeWfMFPCxAH5Kzn3eGMbxG7Y0?authuser=1