Deep Learning

Session-2

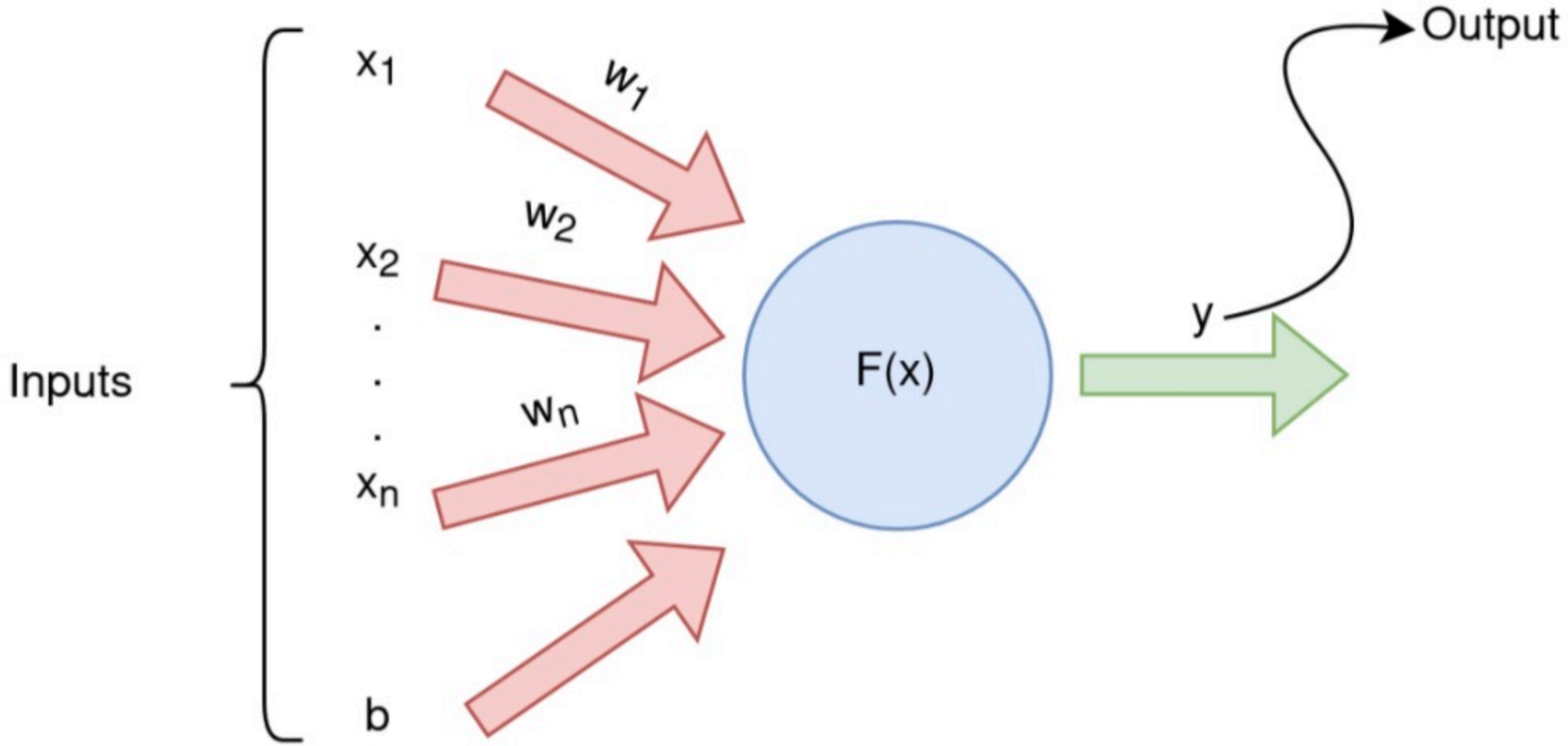
Table of Contents

Recap

Perceptron Models

Generalization of the formula:

$$y = x_1 \cdot w_1 + x_2 \cdot w_2 + \ldots + x_n \cdot w_n + b$$



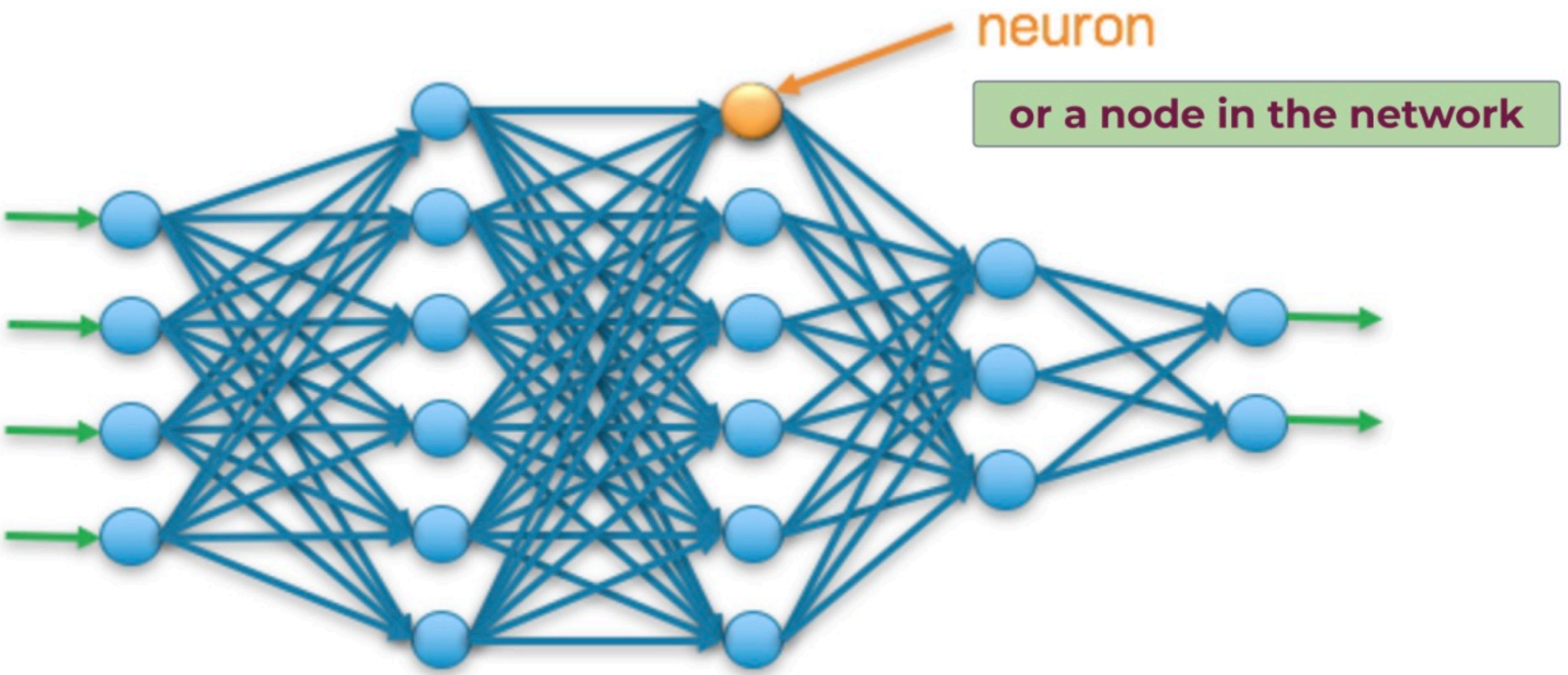
Perceptron Models

input x weight + bias = output

The formula is: $z = \sum x_i \cdot w_i + b$ b: *bias*

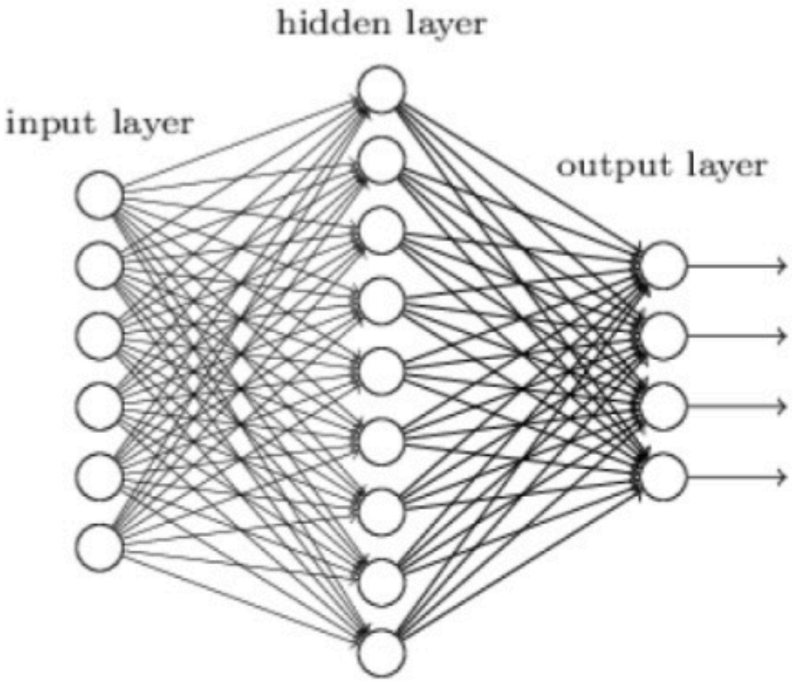
$$z = x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 + \cdots + x_n \cdot w_n + b$$

Neurons in a Network

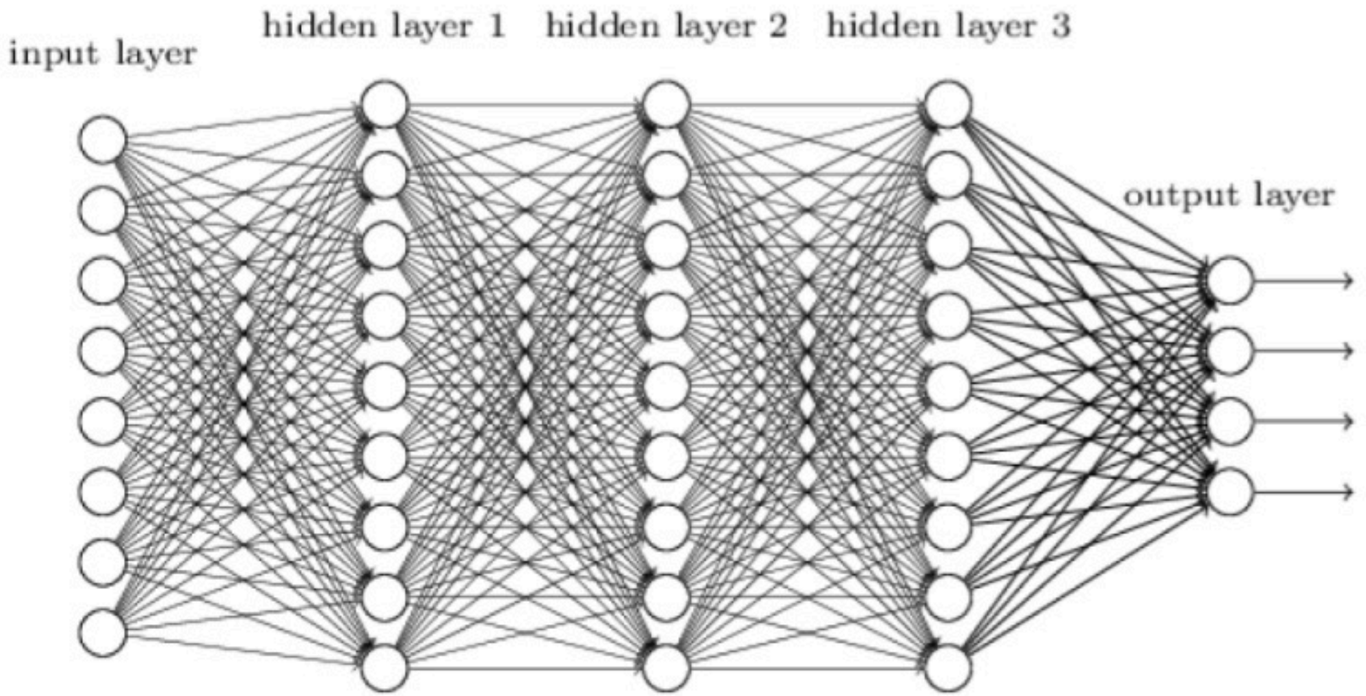


Deep Neural Networks

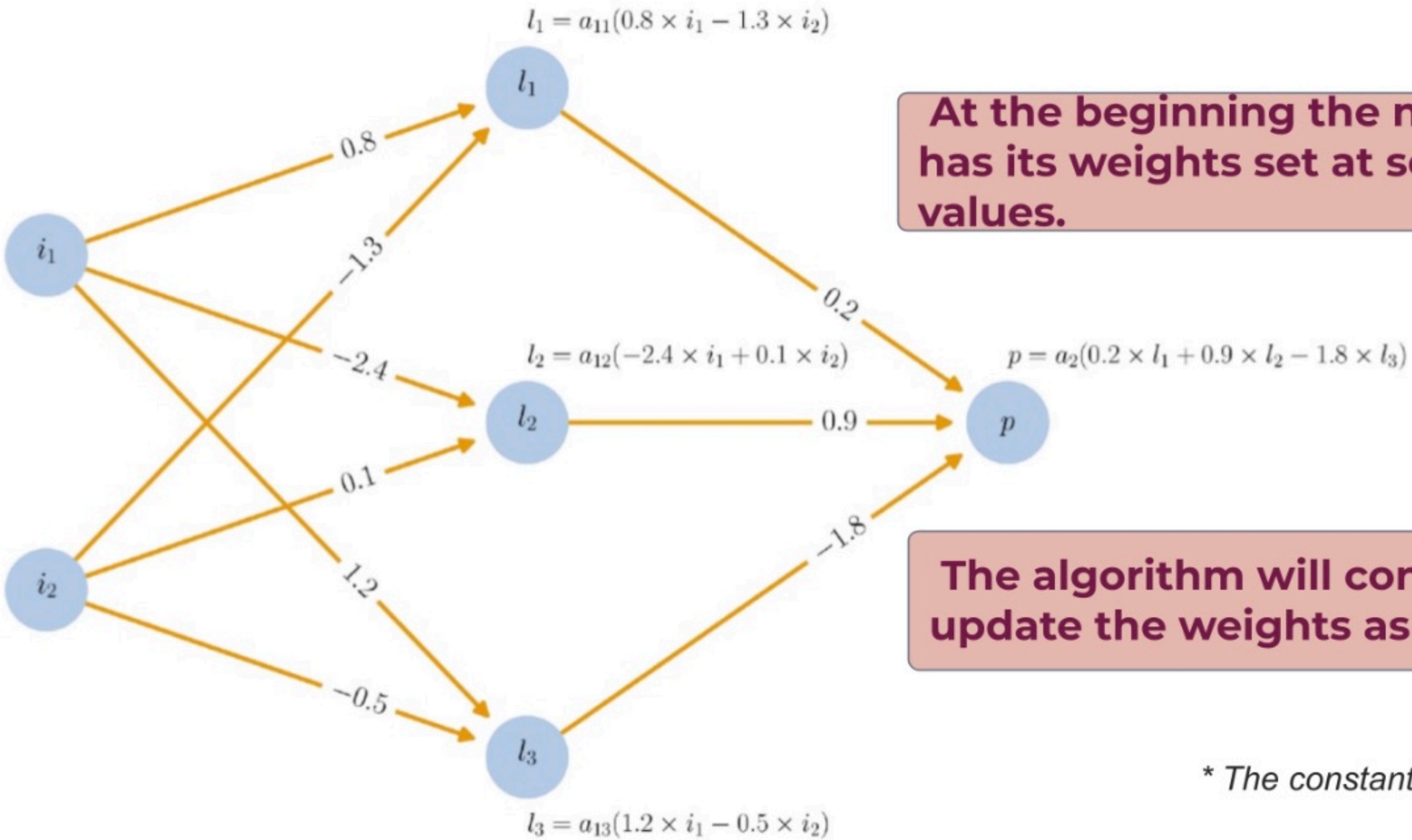
Neural Network



Deep Neural Network



Deep Learning vs Linear Regression



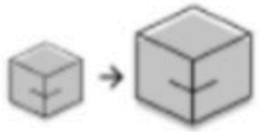
At the beginning the network has its weights set at some values.

The algorithm will continuously update the weights as it learns.

** The constants are ignored.*

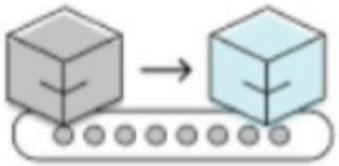
Activation Functions

How it works?



Take input and multiply by the neuron's weight.

Add bias*



Feed the result, x, to the activation function: f(x)

Take the output and transmit to the next layer of neurons.

Type of Activation Functions

Binary Step Function 1

Linear Activation Function 2

Non-Linear Activation Function 3

- **Sigmoid**
- **Softmax**
- **ReLU**
- **TanH**
- **Leaky ReLU**

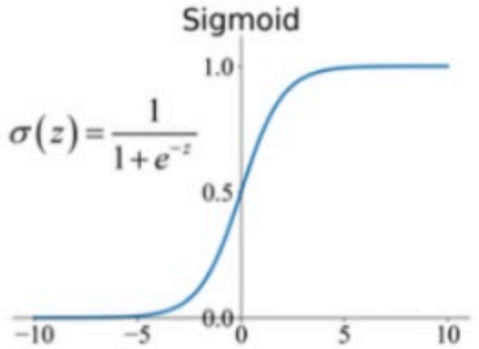
** Modern neural network models use non-linear activation functions in hidden layers*

Type of Activation Functions

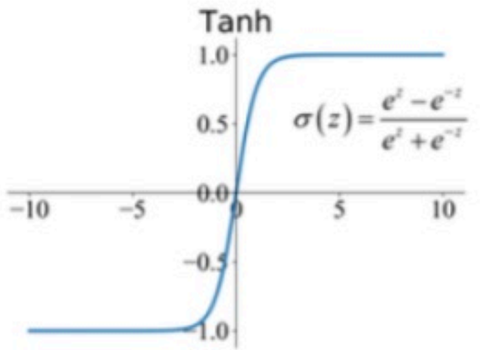
Non-Linear Activation Functions

Allow backpropagation

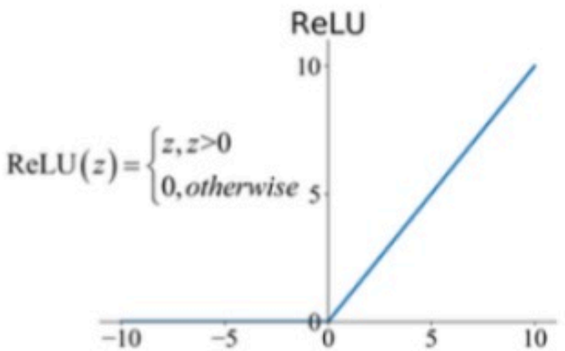
Allow multiple hidden layers



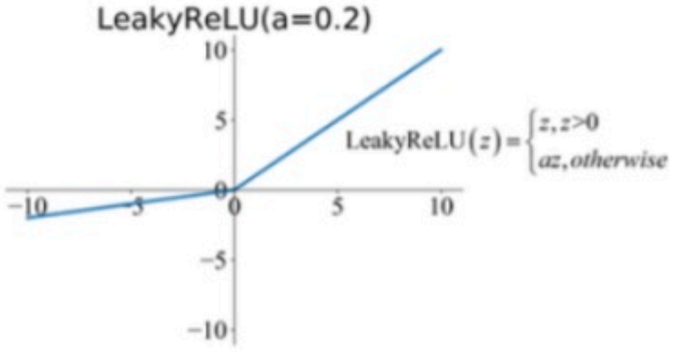
(a)



(b)



(c)

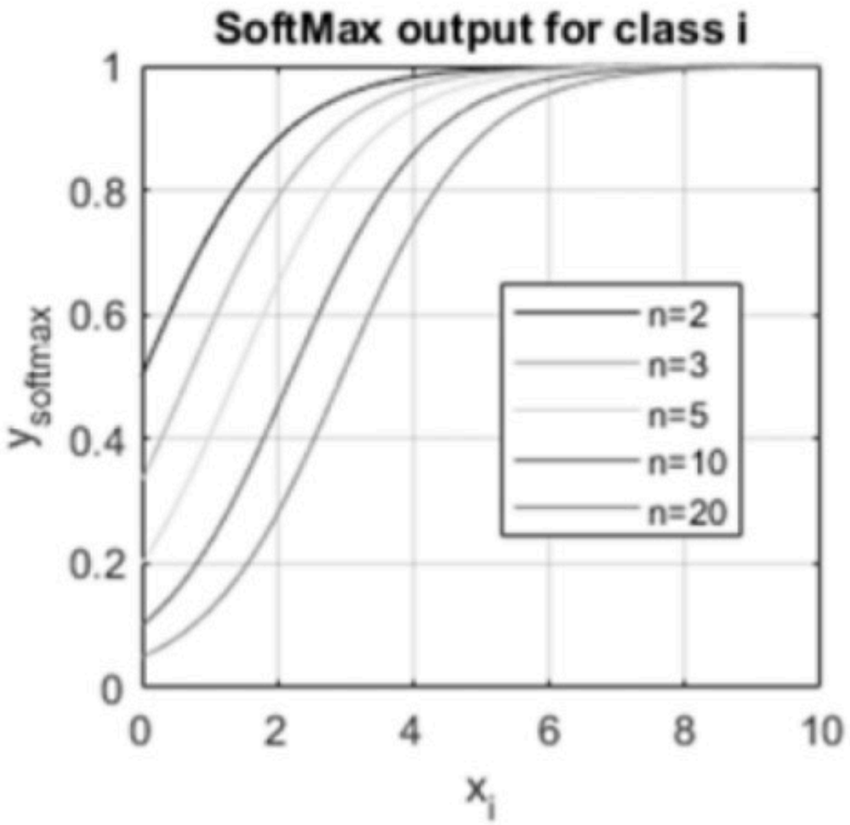


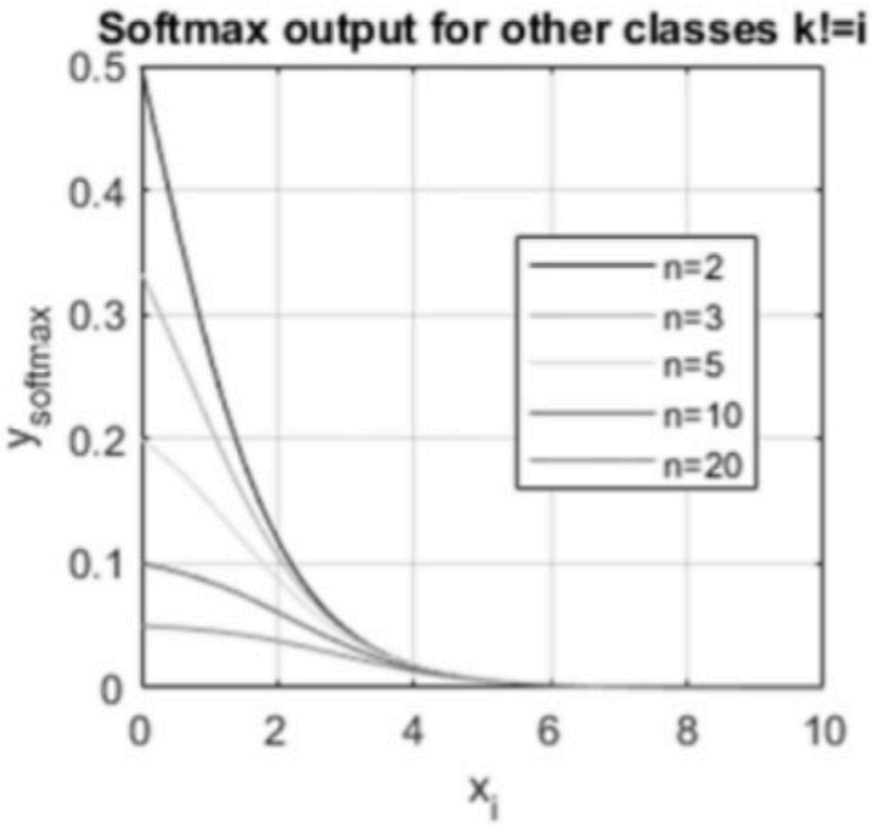
(d)

Modern neural network models use non-linear activation functions in hidden layers

Softmax Activation Functions

$$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



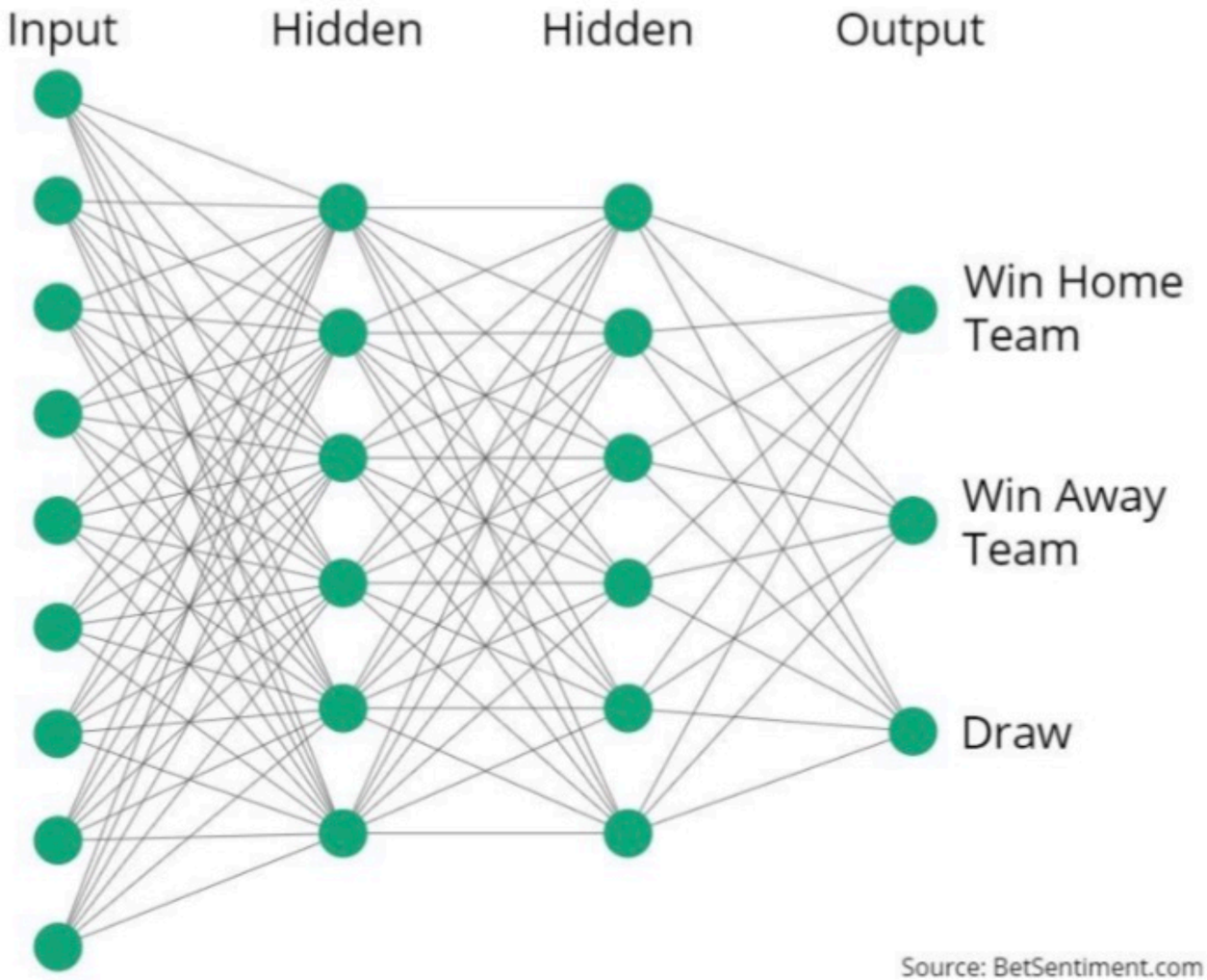


- **Able to handle multiple classes** only one class in other activation functions—normalizes the outputs for each class between 0 and 1

- **Useful for output neurons**—typically Softmax is used only for the output layer

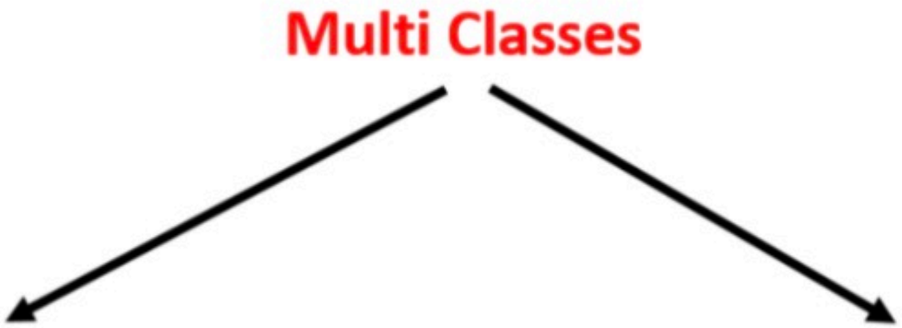
Multiclass Classification

Multiclass Classification



Source: BetSentiment.com

Multiclass Classification



Mutually Exclusive Classes

Data Point 1	RED
Data Point 2	GREEN
Data Point 3	BLUE
...	...
Data Point N	RED

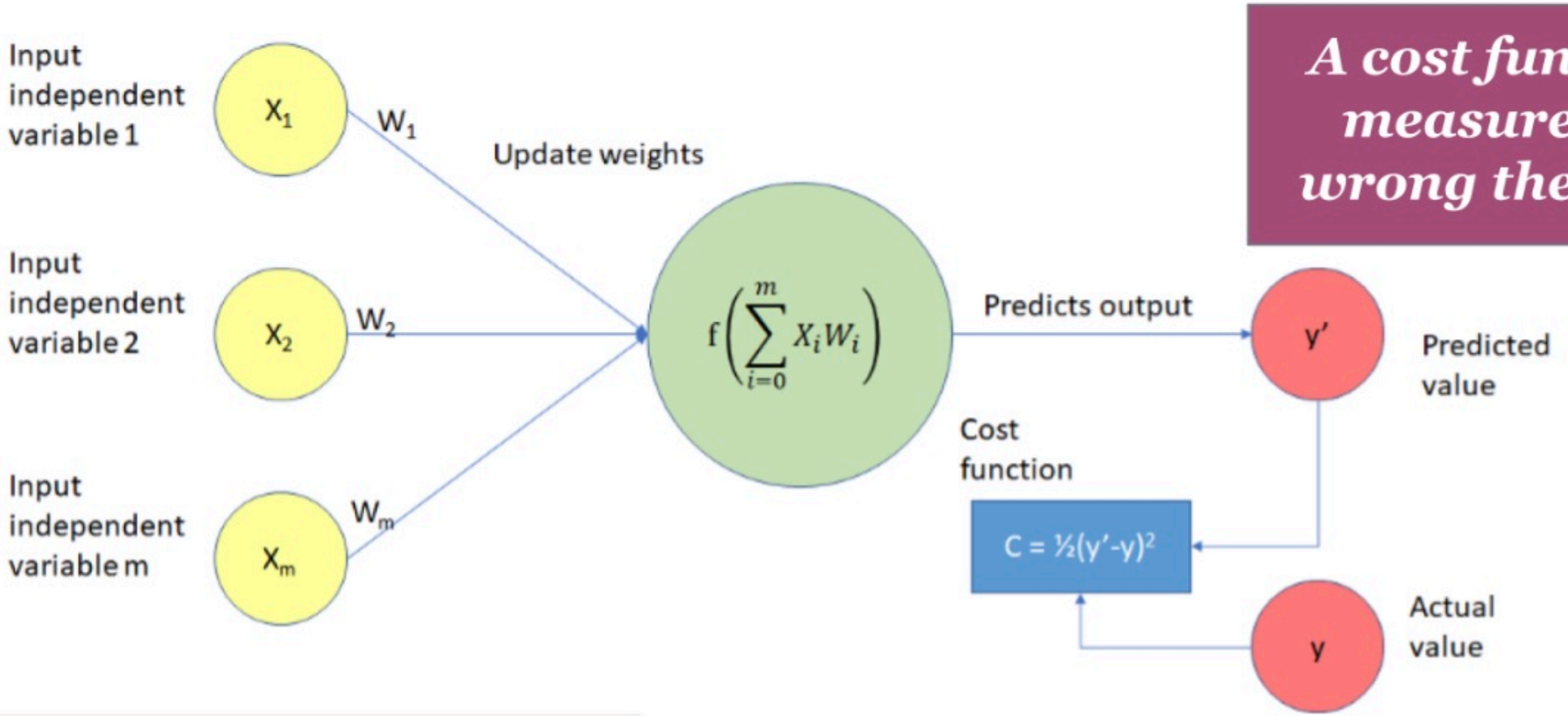
Softmax Activation Function

Non-Exclusive Classes

Data Point 1	A,B
Data Point 2	A
Data Point 3	C,B
...	...
Data Point N	B

Sigmoid Activation Function

Cost Function (Loss Function)



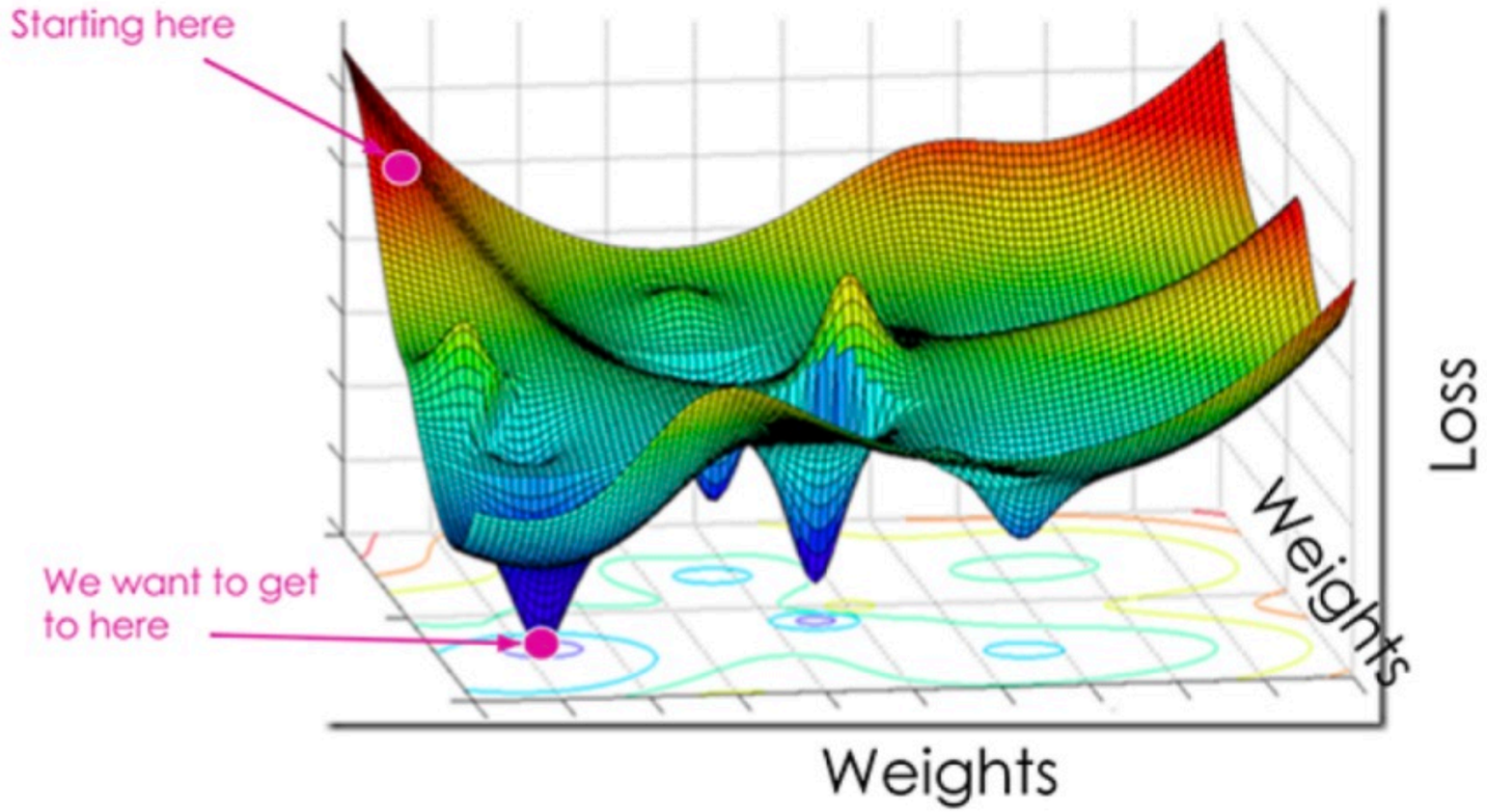
A cost function is a measure of how wrong the model is

Cost function $J = \sum_{1}^{m} \frac{1}{2}(y-\hat{y})^2$

Sum over all samples — true value — predicted value

Cost function (J) = 1/m (Sum of Loss error for 'm' examples)

Cost Function (Loss Function)



The objective of a ML model, is to find parameters, weights or a structure that ***minimises*** *the cost function.*

Types of Loss Function

Regression Loss Functions

```
model.compile(optimizer='adam',loss='mse')
```

1. Mean Squared Error Loss
2. Mean Squared Logarithmic Error Loss
3. Mean Absolute Error Loss

Multi-Class Classification Loss Functions

1. Multi-Class Cross-Entropy Loss
2. Sparse Multiclass Cross-Entropy Loss
3. Kullback Leibler Divergence Loss

```
1 model.compile(loss='categorical_crossentropy',
```

Binary Classification Loss Functions

1. Binary Cross-Entropy
2. Hinge Loss
3. Squared Hinge Loss

```
model.compile(loss='binary_crossentropy', optimizer='adam')
```

Cost Function (Loss Function)

```
Epoch 1/600
426/426 [==============================] - 0s 1ms/sample - loss: 0.6807 - val_loss: 0.6623
Epoch 2/600
426/426 [==============================] - 0s 108us/sample - loss: 0.6498 - val_loss: 0.6330
Epoch 3/600
426/426 [==============================] - 0s 80us/sample - loss: 0.6188 - val_loss: 0.6006
Epoch 4/600
```

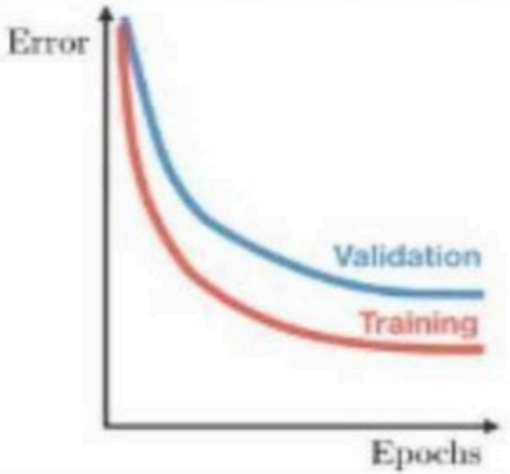
▪
▪
▪

```
Epoch 99/600
426/426 [==============================] - 0s 68us/sample - loss: 0.0485 - val_loss: 0.1003
Epoch 100/600
426/426 [==============================] - 0s 73us/sample - loss: 0.0483 - val_loss: 0.1060
Epoch 101/600
426/426 [==============================] - 0s 68us/sample - loss: 0.0500 - val_loss: 0.1036
Epoch 00101: early stopping
```

decreases as model learns

Cost Function (Loss Function)

UNDERFITTING VS OVERFITTING

	Underfitting	Just right	Overfitting
Deep learning illustration			
Possible remedies	• Complexify model • Add more features • Train longer		• Perform regularization • Get more data

Cost Function (Loss Function)

Problem Type	Output Type	Final Activation Function	Loss Function
Regression	Numerical value	Linear	Mean Squared Error (MSE)
Classification	Binary outcome	Sigmoid	Binary Cross Entropy
Classification	Single label, multiple classes	Softmax	Cross Entropy
Classification	Multiple labels, multiple classes	Sigmoid	Binary Cross Entropy

Gradient Descent

Epoch and Batchsize

```
model.fit(x=X_train,y=y_train.values,
          validation_data=(X_test,y_test.values),
          batch_size=128,epochs=400)
```

Epoch

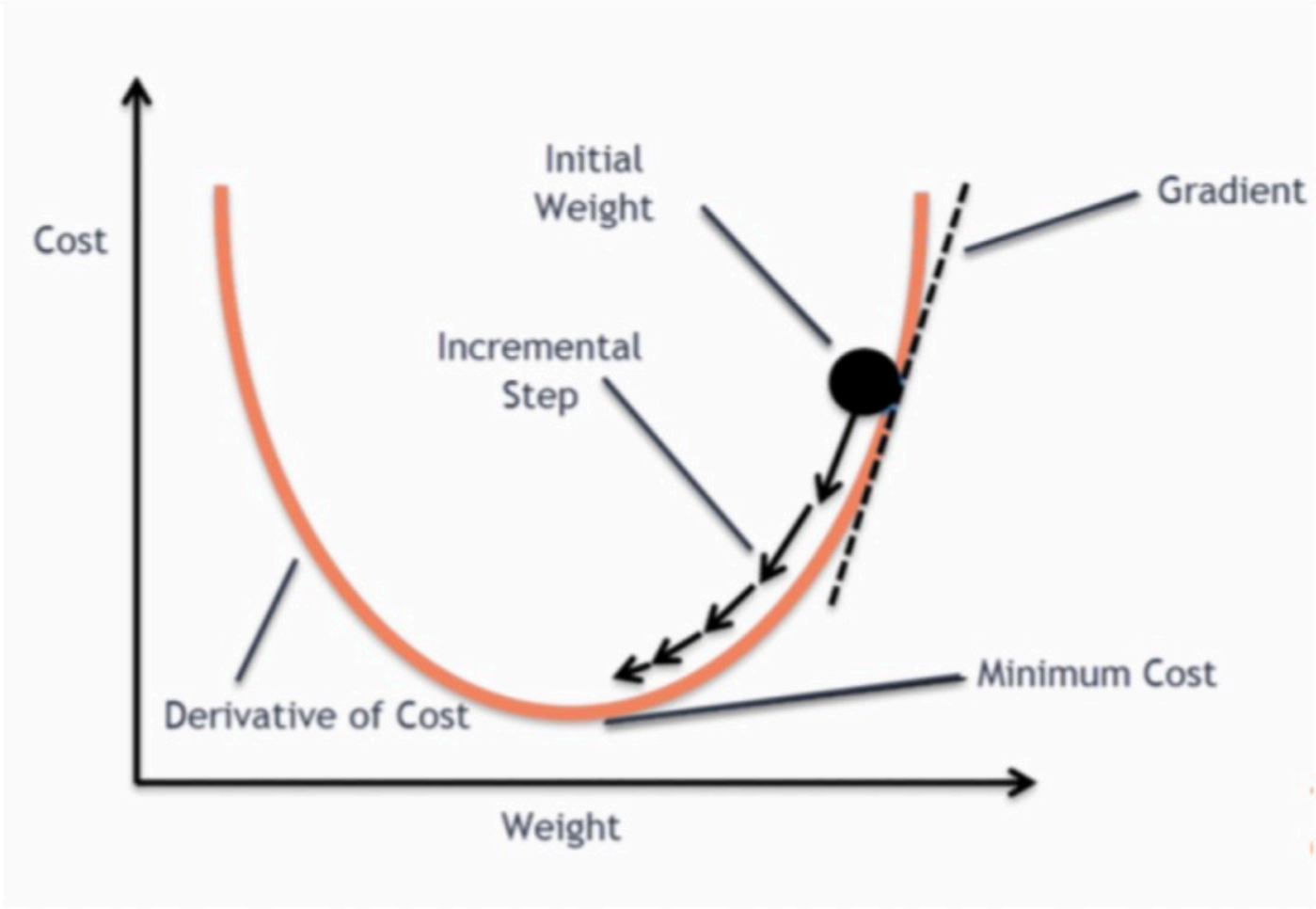
One epoch means, the entire dataset is passed forward and backward through the neural network once.

Batch Size

Batch size refers to the number of training examples utilized in one iteration.

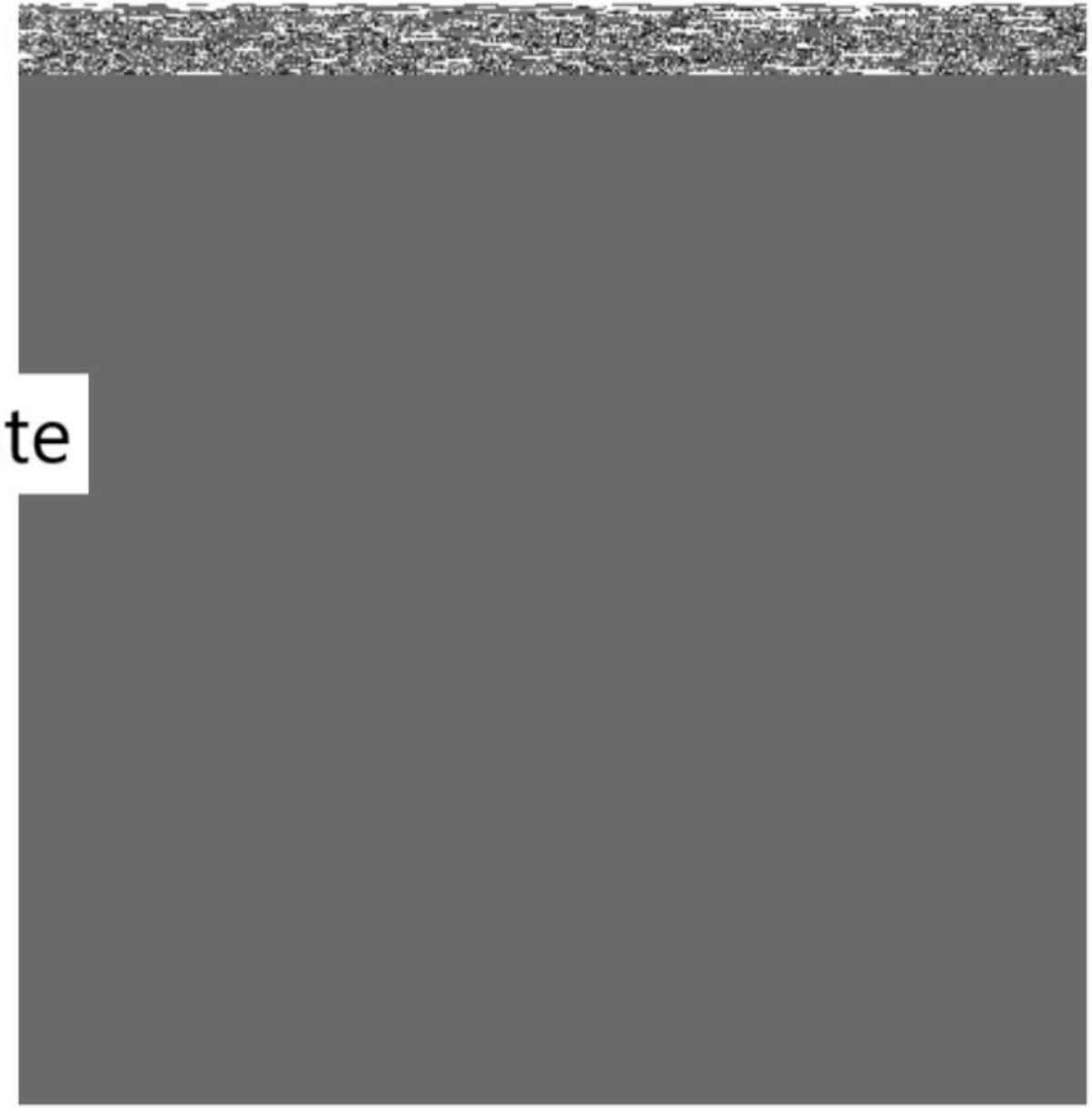
Gradient Descent

```
model.compile(optimizer='adam',loss='mse')
```



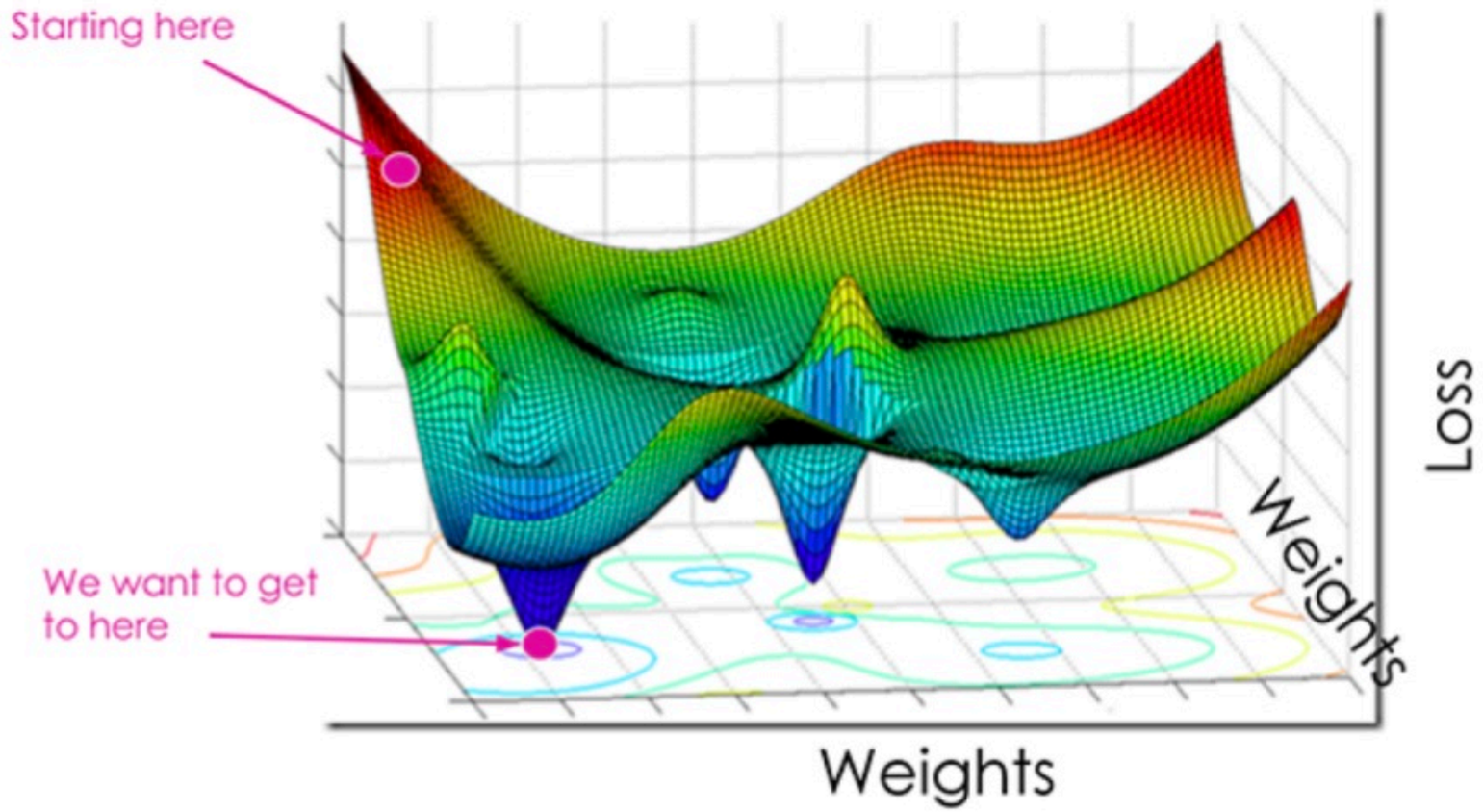
Gradient descent is an *optimization algorithm* used to minimize the cost function.

Gradient Descent (Learning Rate)



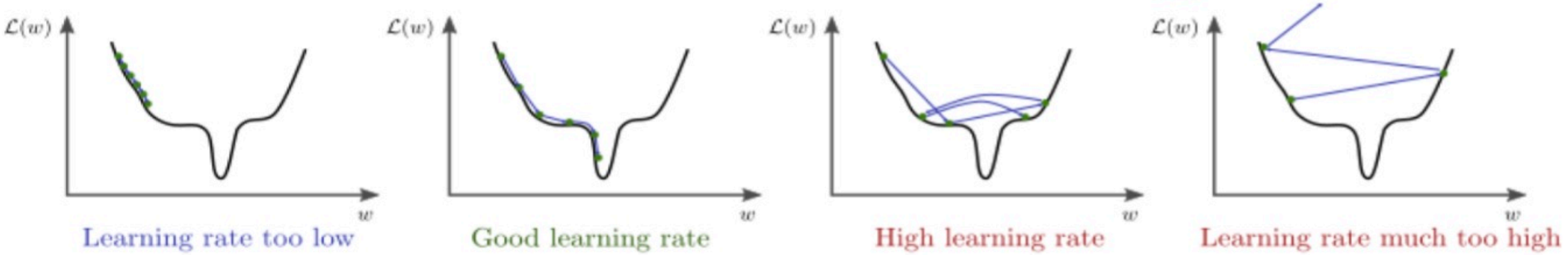
Step Size = slope x learning rate

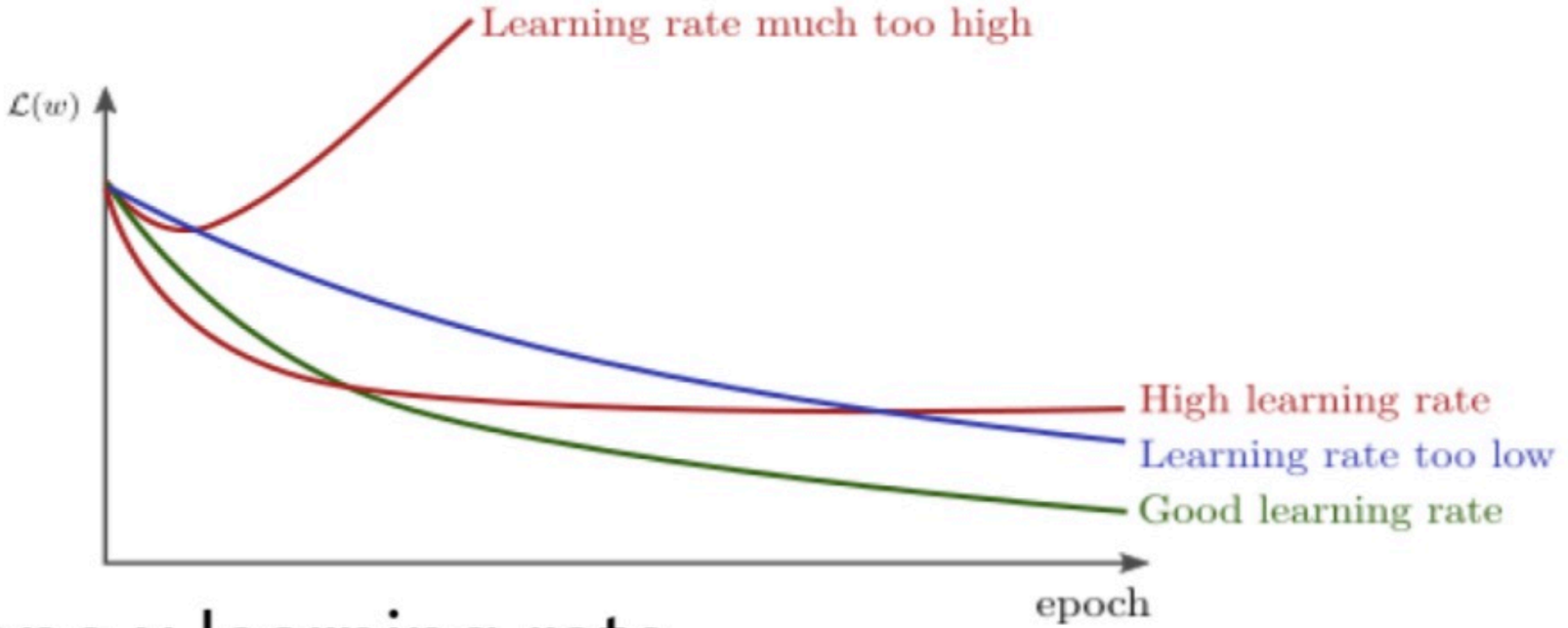
Gradient Descent (Learning Rate)



The objective of a DL model, is to find parameters, weights or a structure that ***minimises*** *the cost function.*

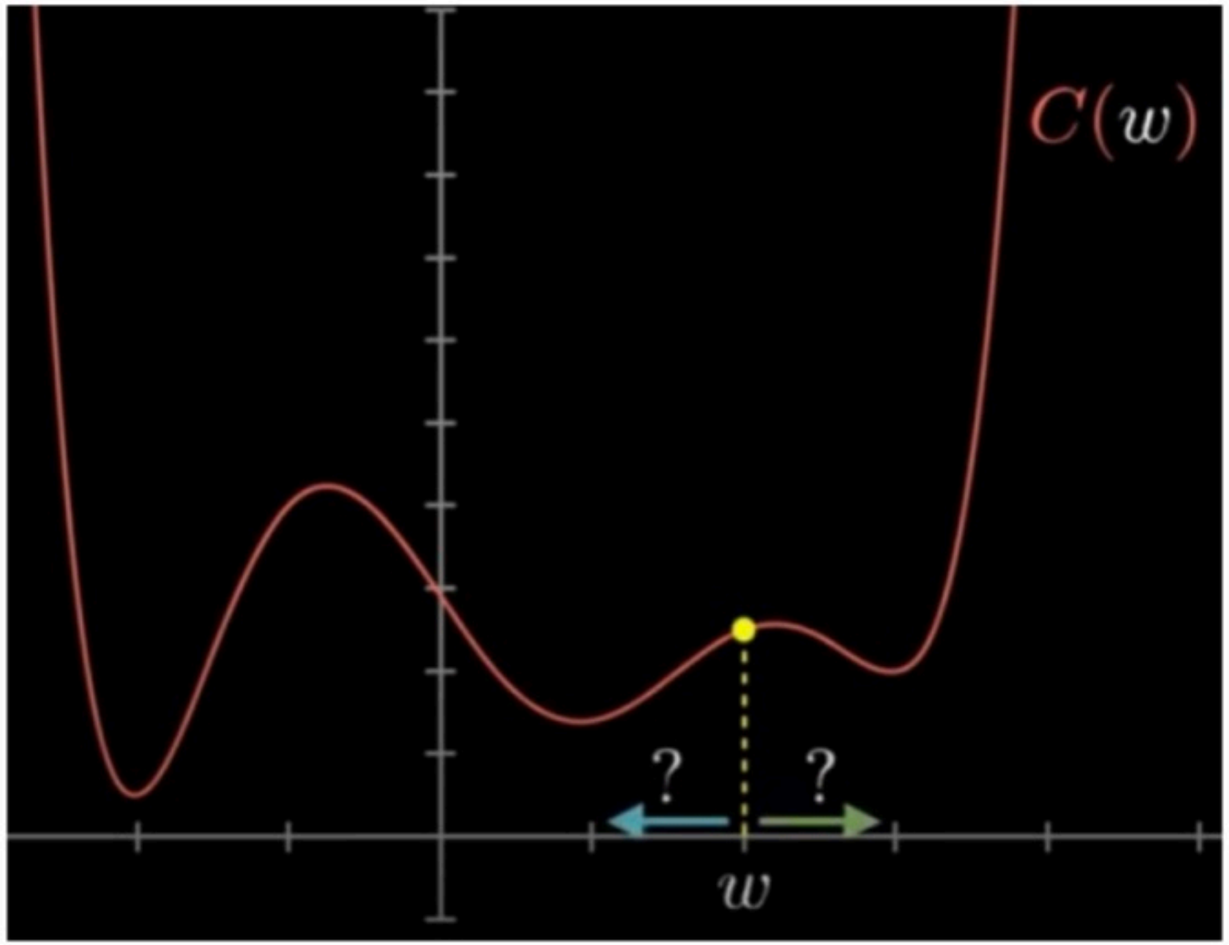
Gradient Descent (Learning Rate)





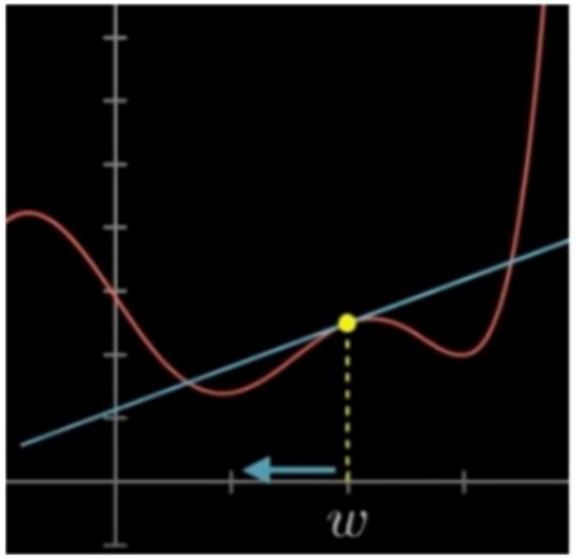
Step Size = slope x learning rate

Gradient Descent (Direction)

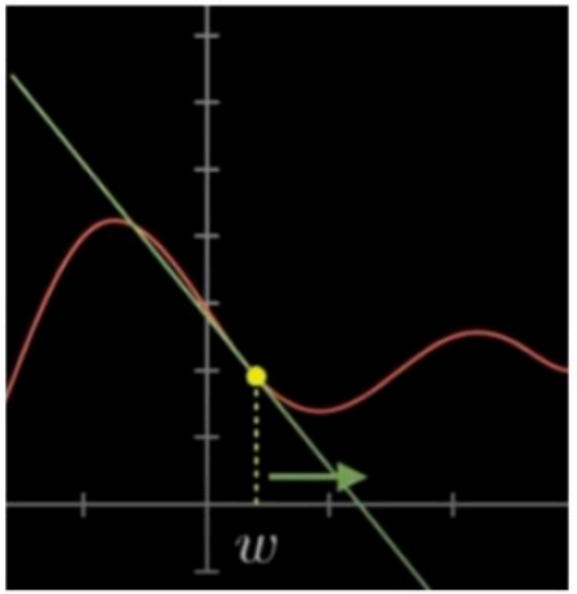


slope

positive



negative



Gradient Descent (Variants)

Batch (Vanilla) Gradient Descent

Calculates error for each example. Model is updated only after an epoch.

Stochastic Gradient Descent

Iterates over each example while updating the model.

Mini-Batch Gradient Descent

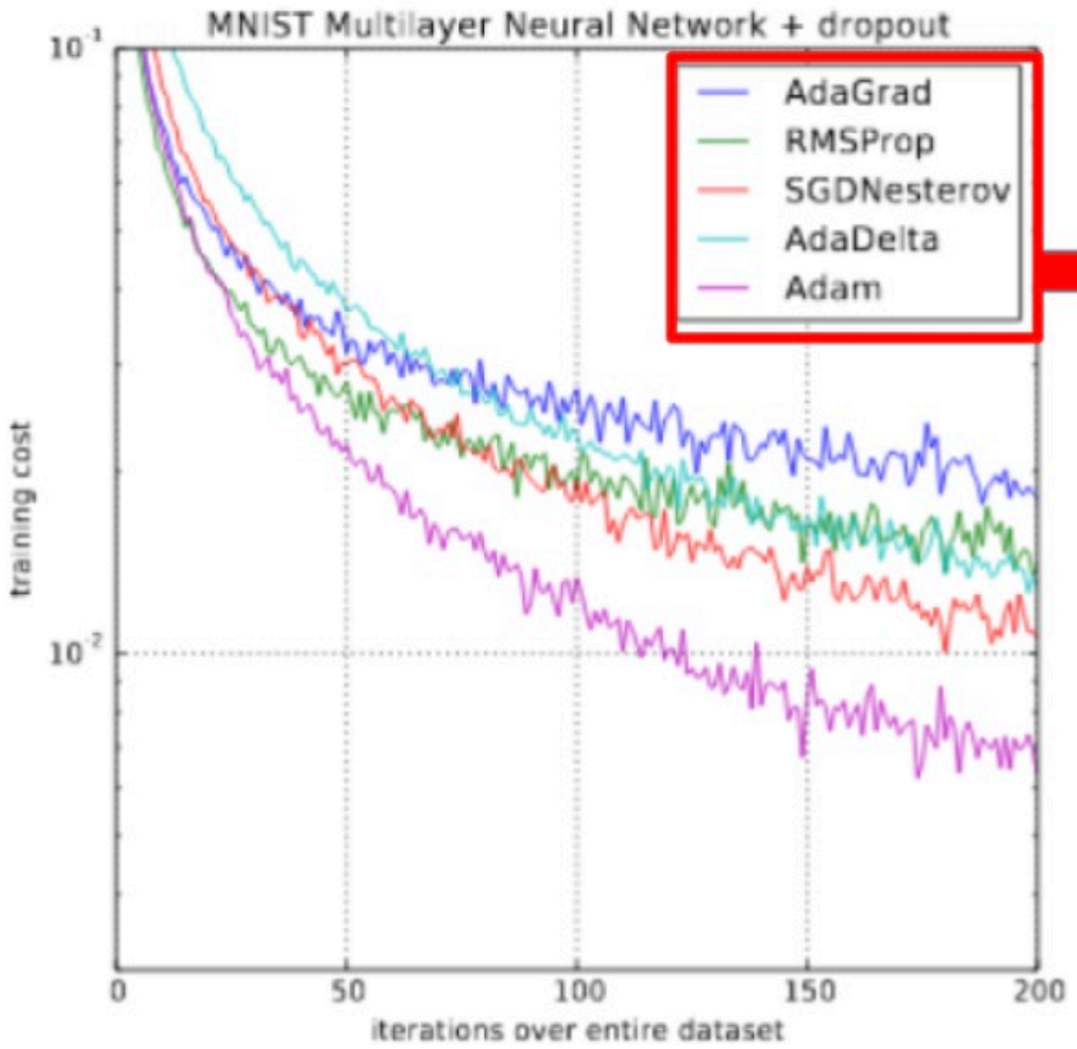
A combination of concepts of both SGD and Batch Gradient Descent. Splits data into batches balancing between the efficiency of batch gradient descent and the robustness of SGD.

Gradient Descent (Variants)

PARAMETERS	BATCH GD ALGORITHM	MINI BATCH ALGORITHM	STOCHASTIC GD ALGORITHM
ACCURACY	HIGH	MODERATE	LOW
TIME CONSUMING	MORE	MODERATE	LESS

Gradient Descent (Optimizer)

Gradient Descent (Optimizer)



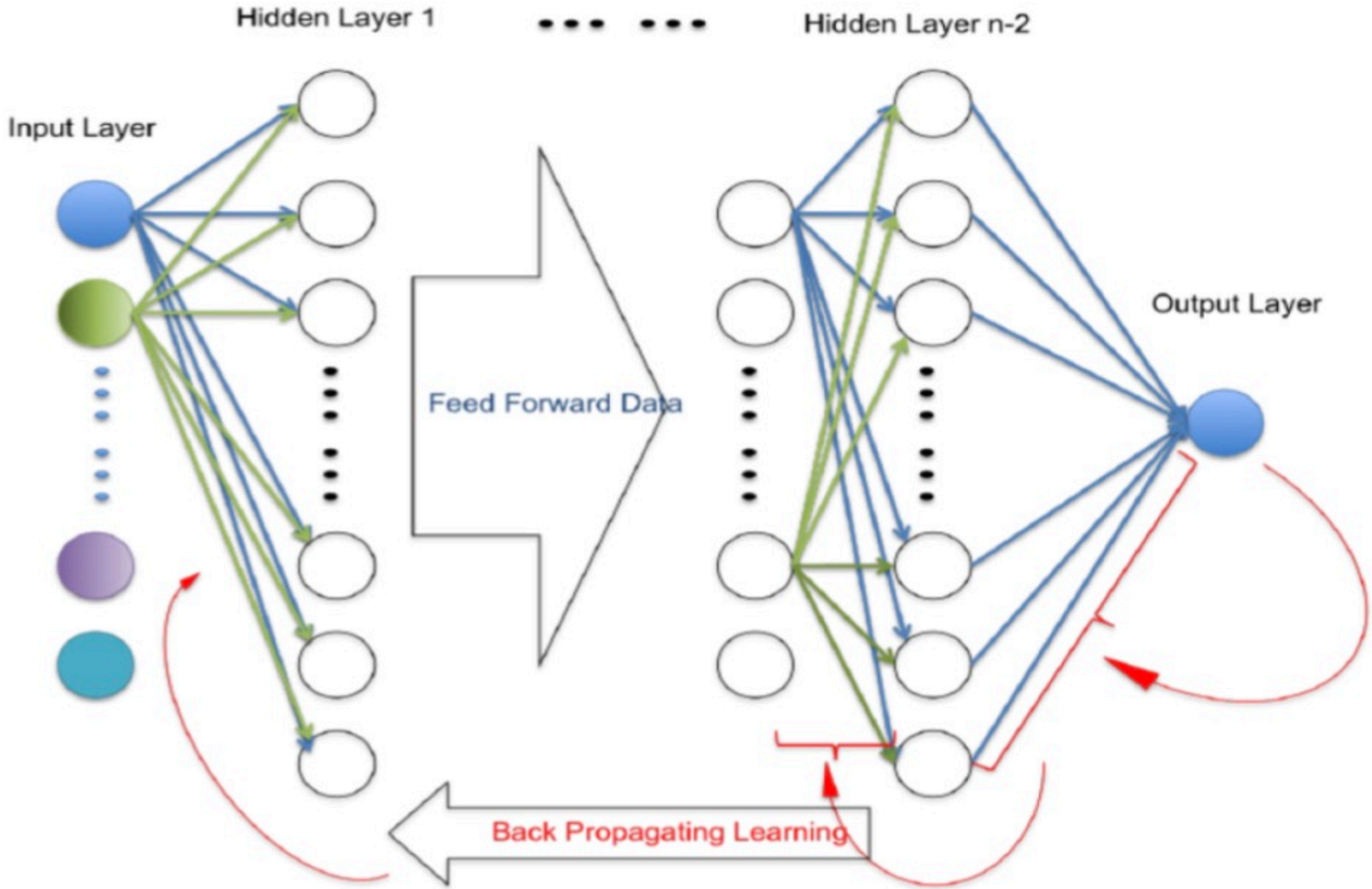
source: original Adam paper

```
model.compile(optimizer='adam',loss='mse')
```

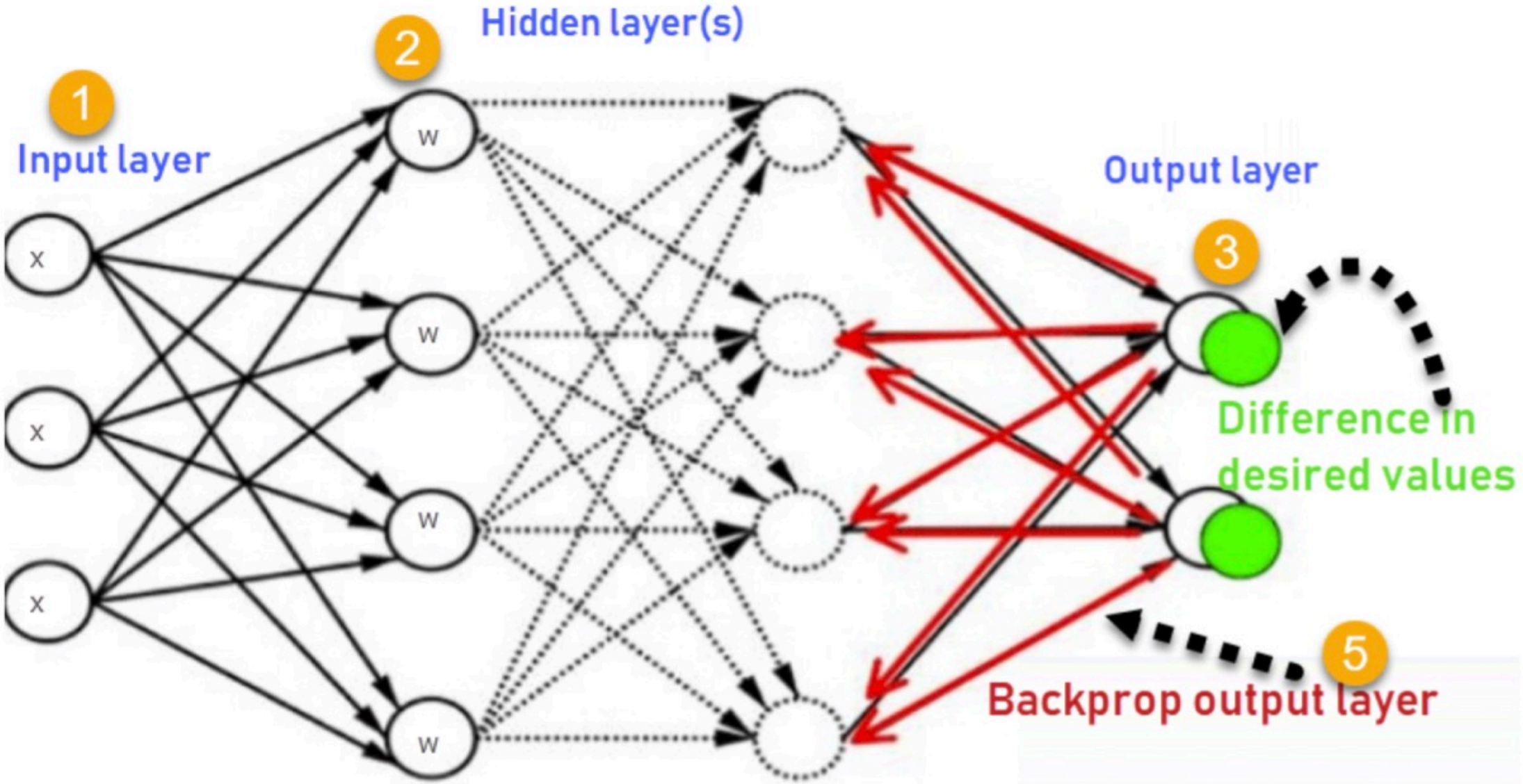
Different options for choosing *optimizers* besides *adam*.

Backpropagation

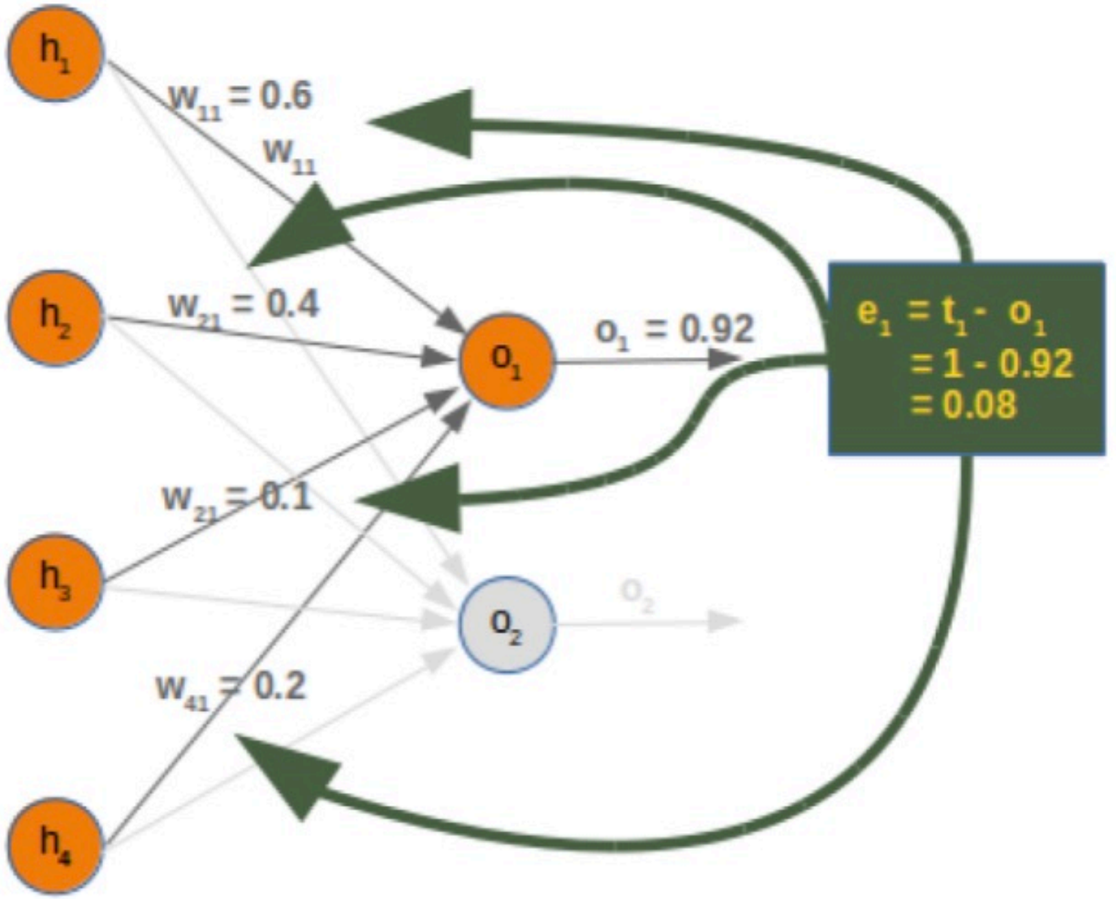
Backpropagation



Backpropagation



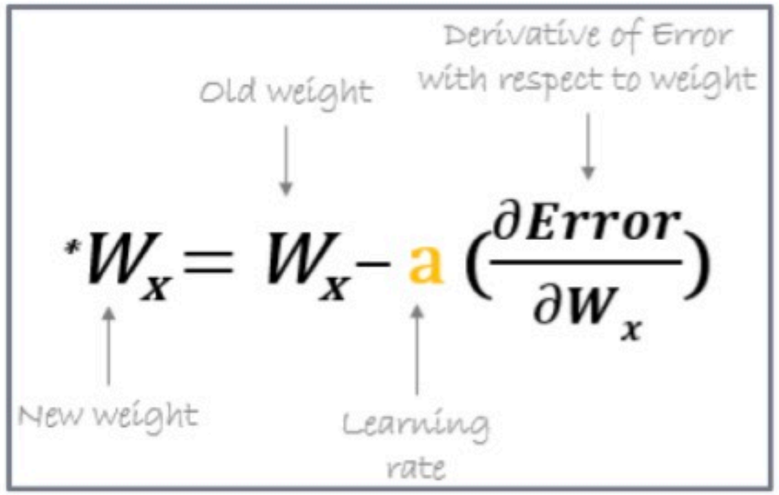
Backpropagation



Error is calculated between the expected outputs and the outputs forward propagated from the network.

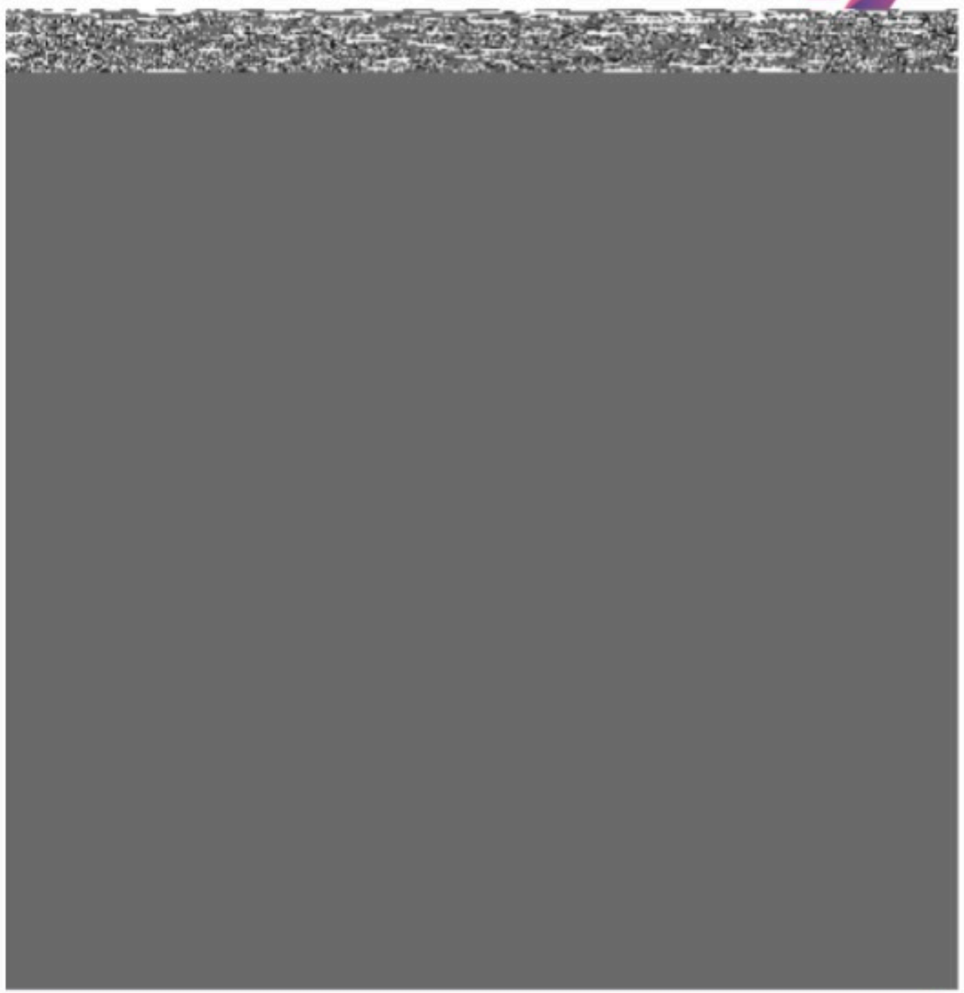
Errors (loss) are then propagated backward through the network from the output layer to the hidden layer, assigning blame for the error and updating weights as they go.

Backpropagation



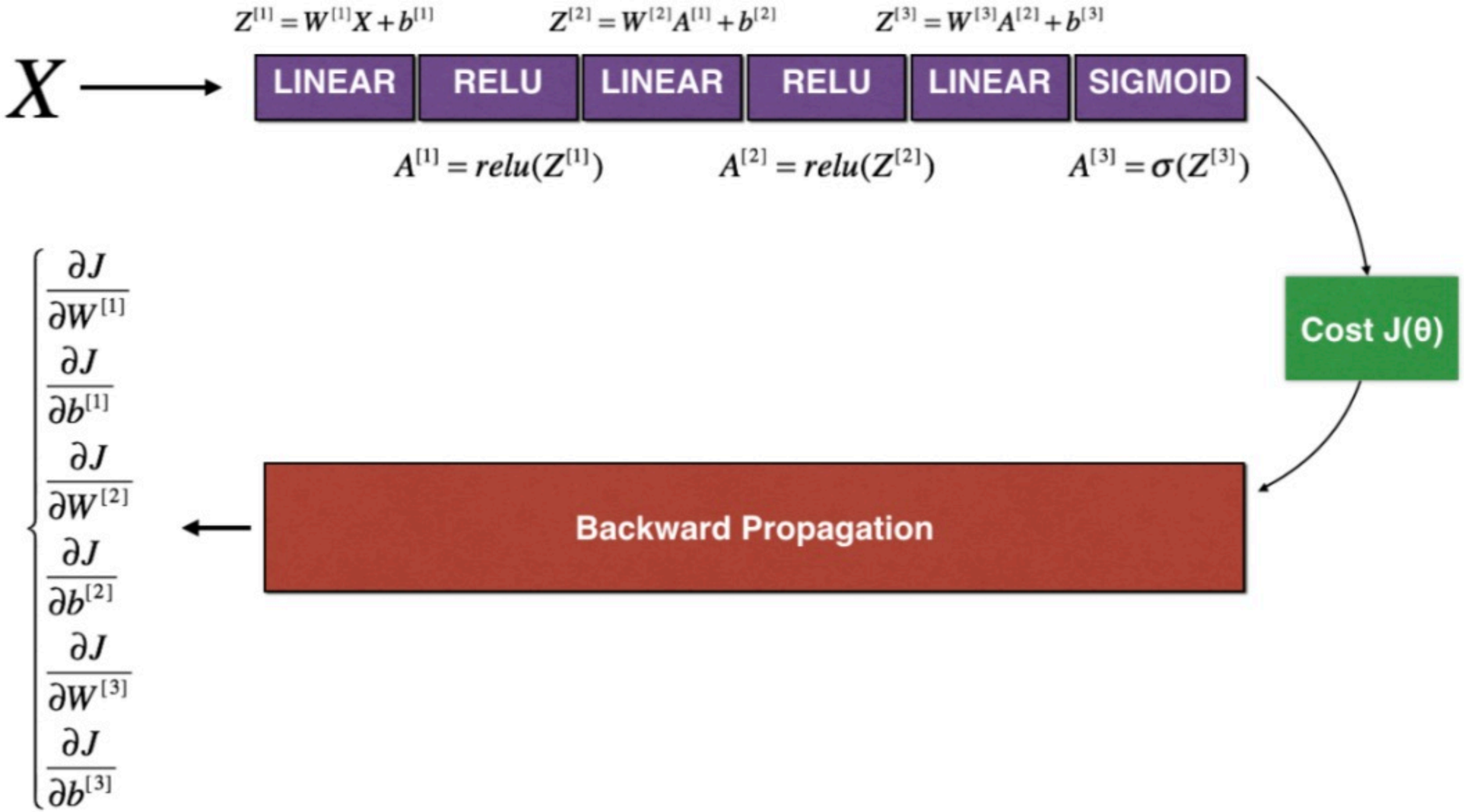
Step Size = slope x learning rate

New Parameter = Old Parameter - Step Size

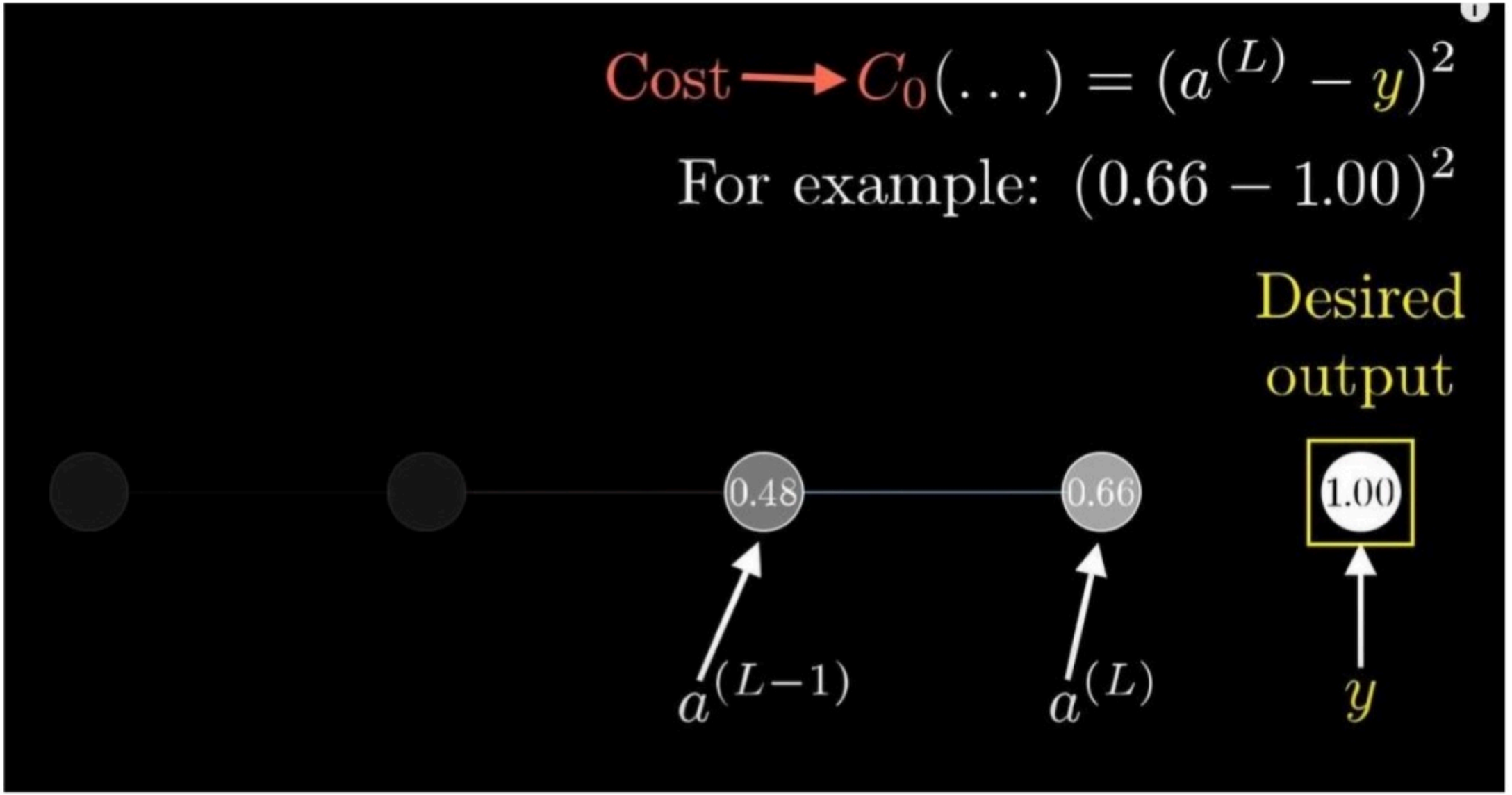


The change in each parameter changes according to the effect of that parameter on the output value. This effect is calculated by taking derivatives in the opposite direction.

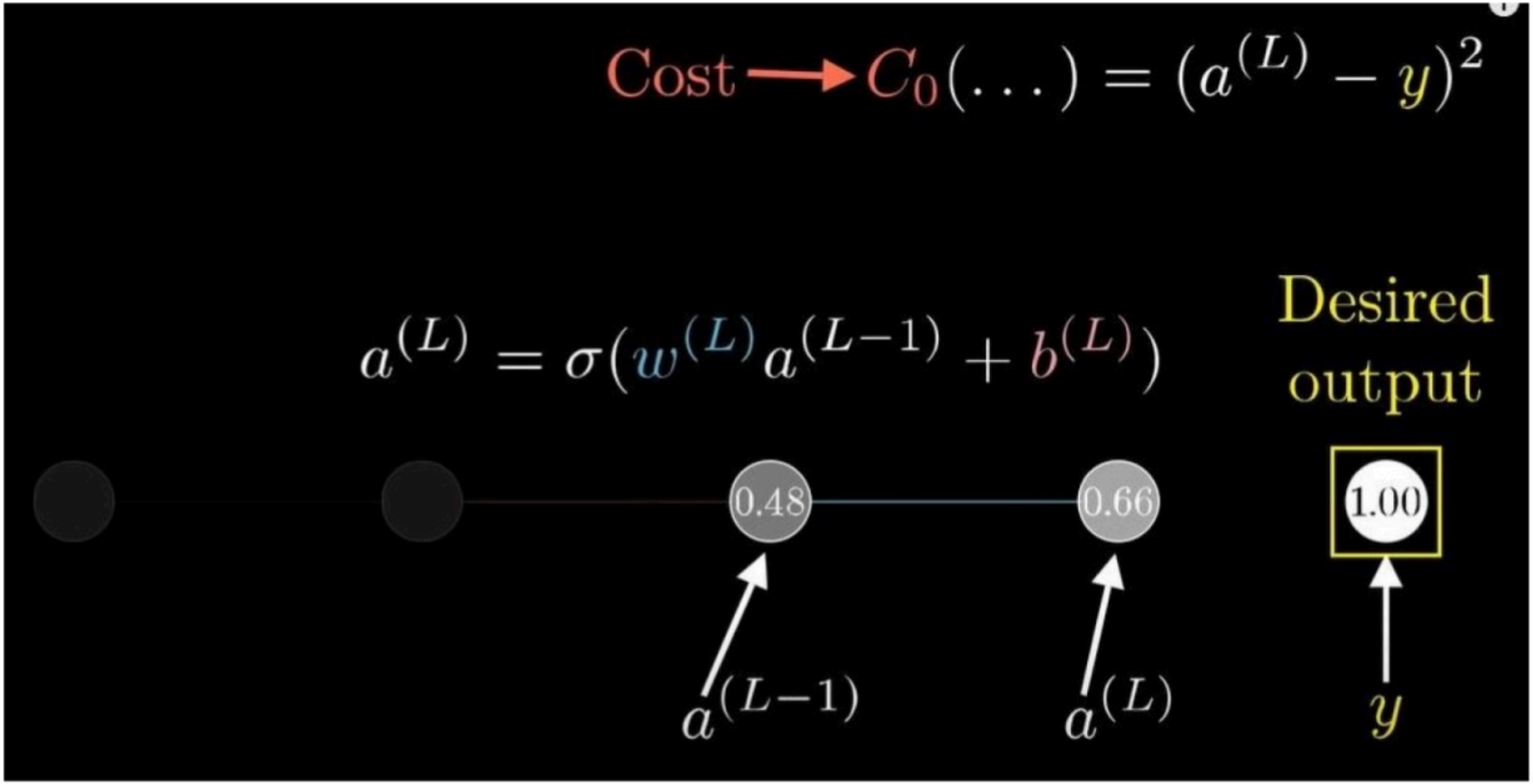
Backpropagation



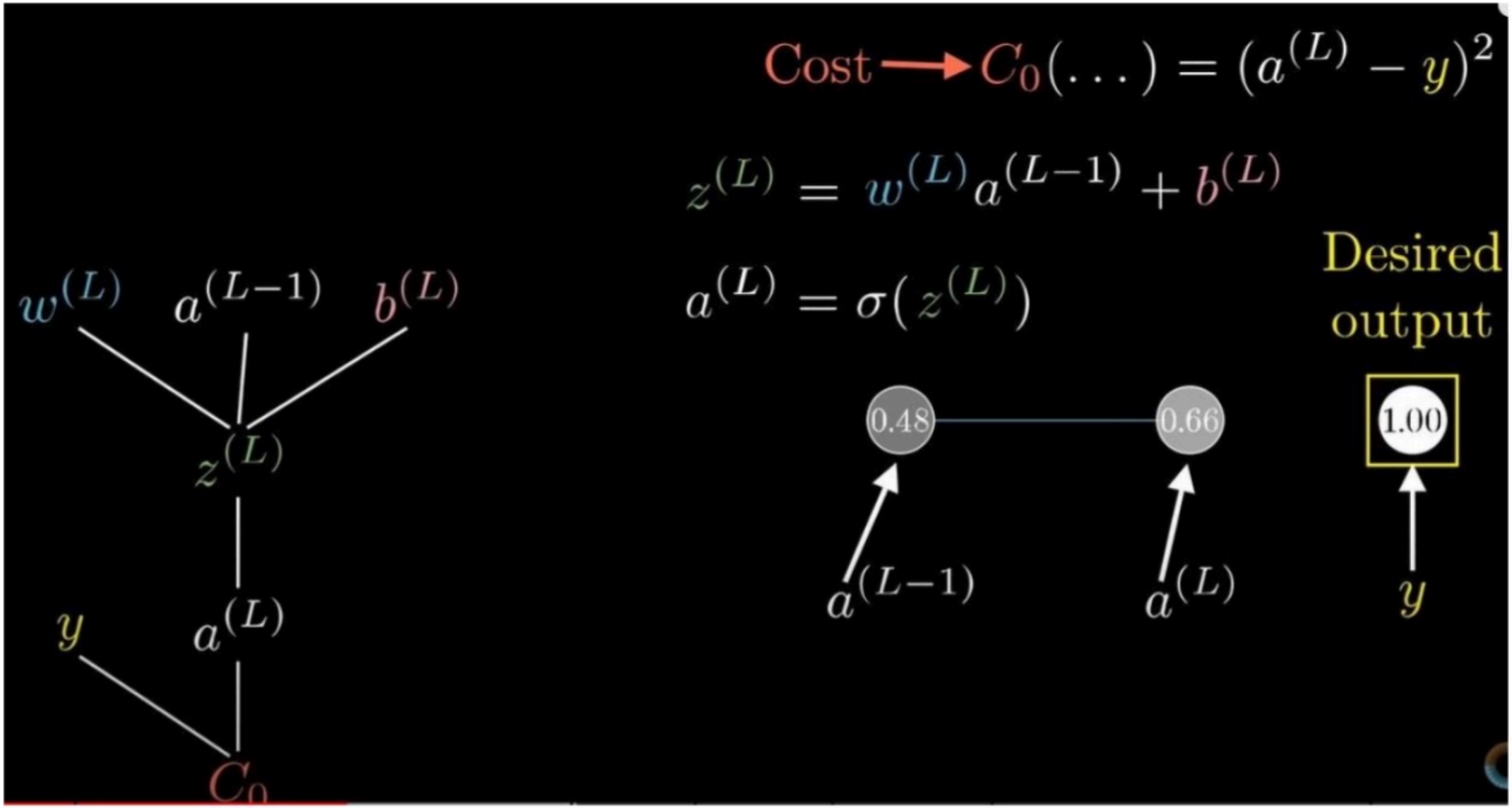
Backpropagation



Backpropagation



Backpropagation



Backpropagation

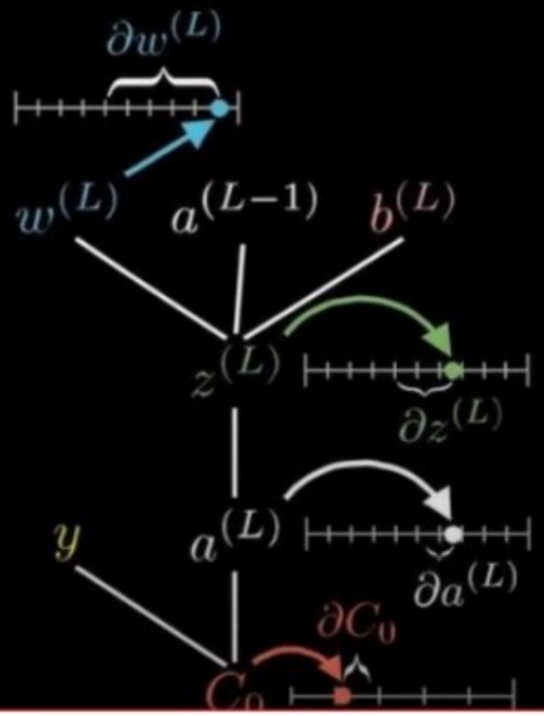
$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}}$$

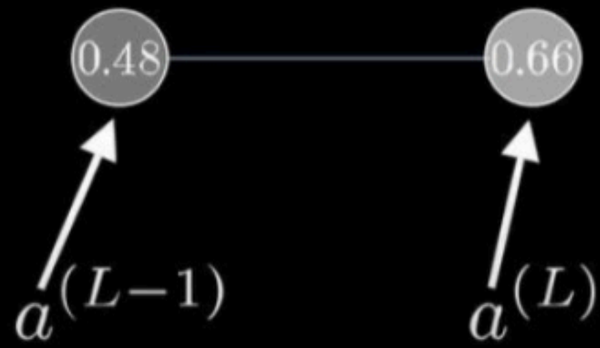
Chain rule

$$C_0(\dots) = (a^{(L)} - y)^2$$

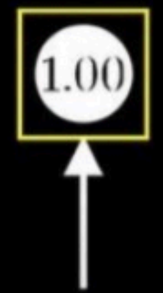
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

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

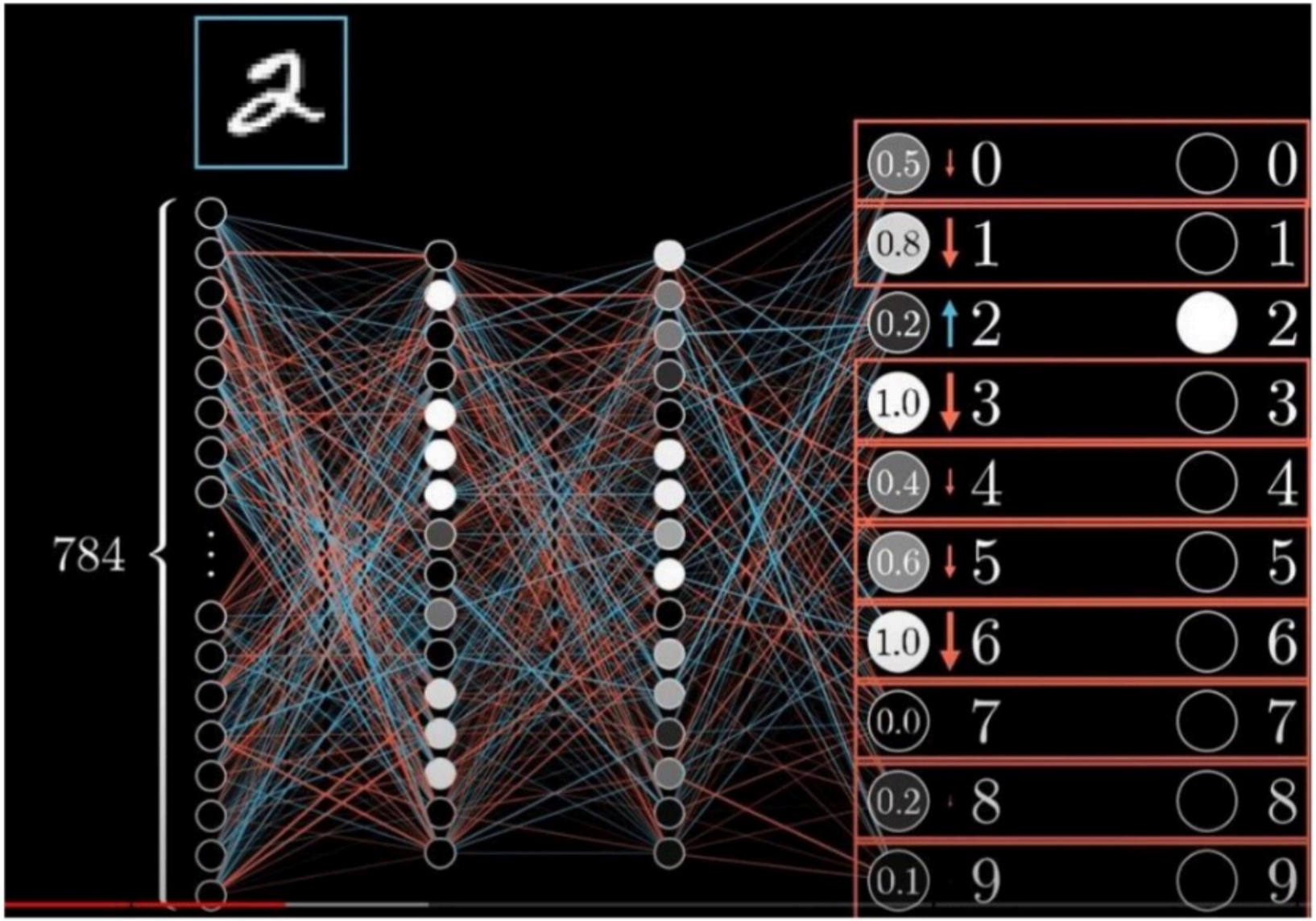




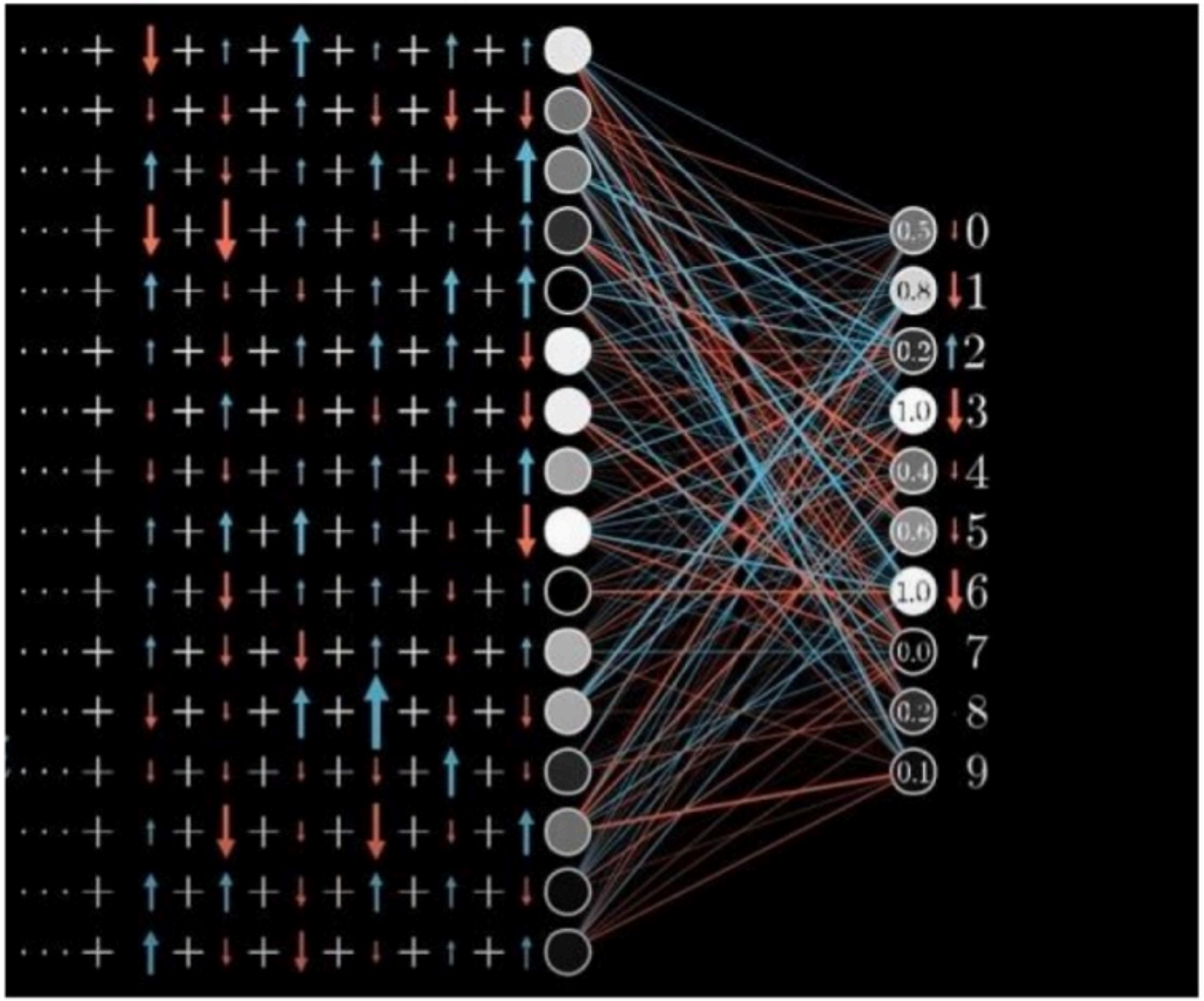
Desired output



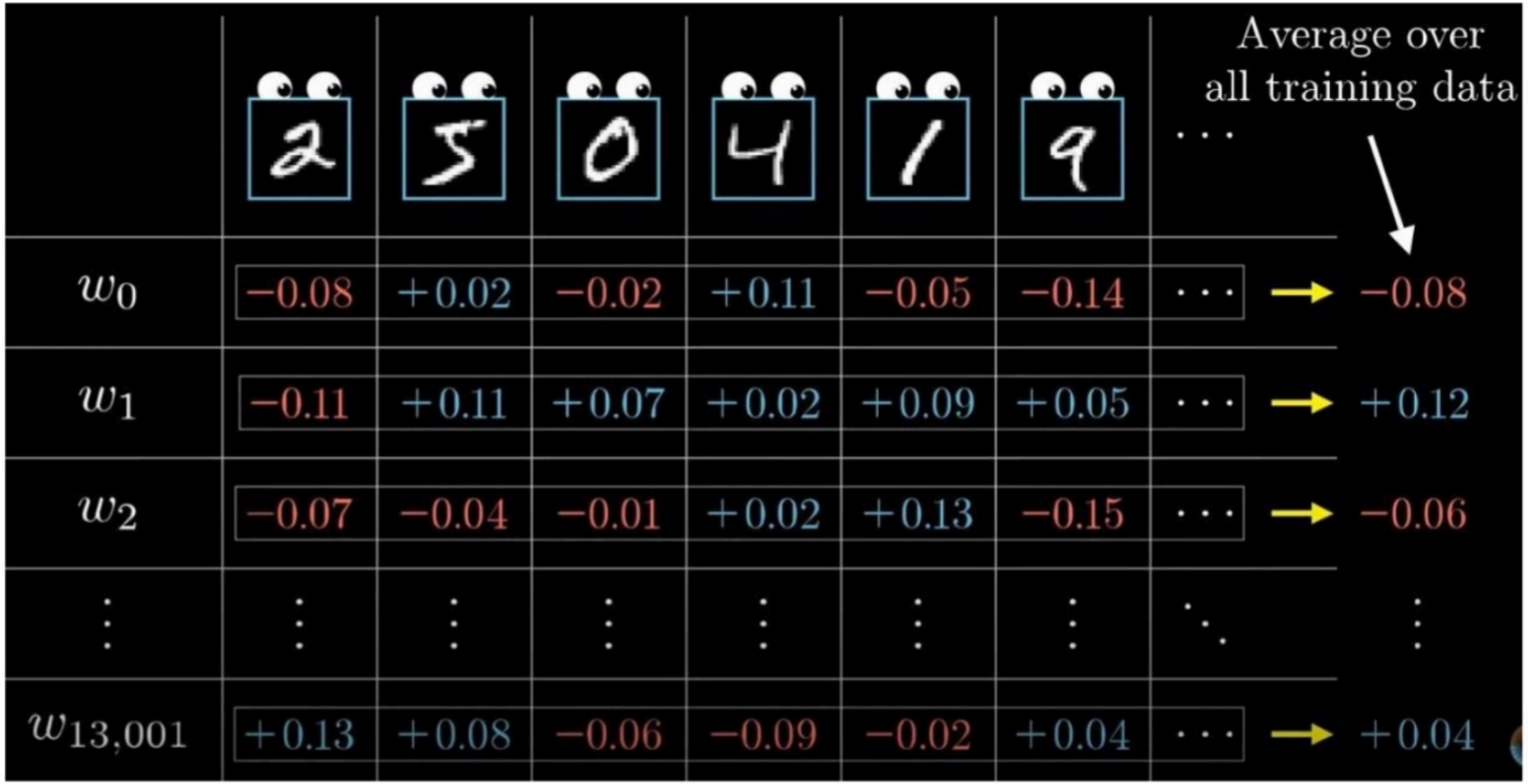
Backpropagation



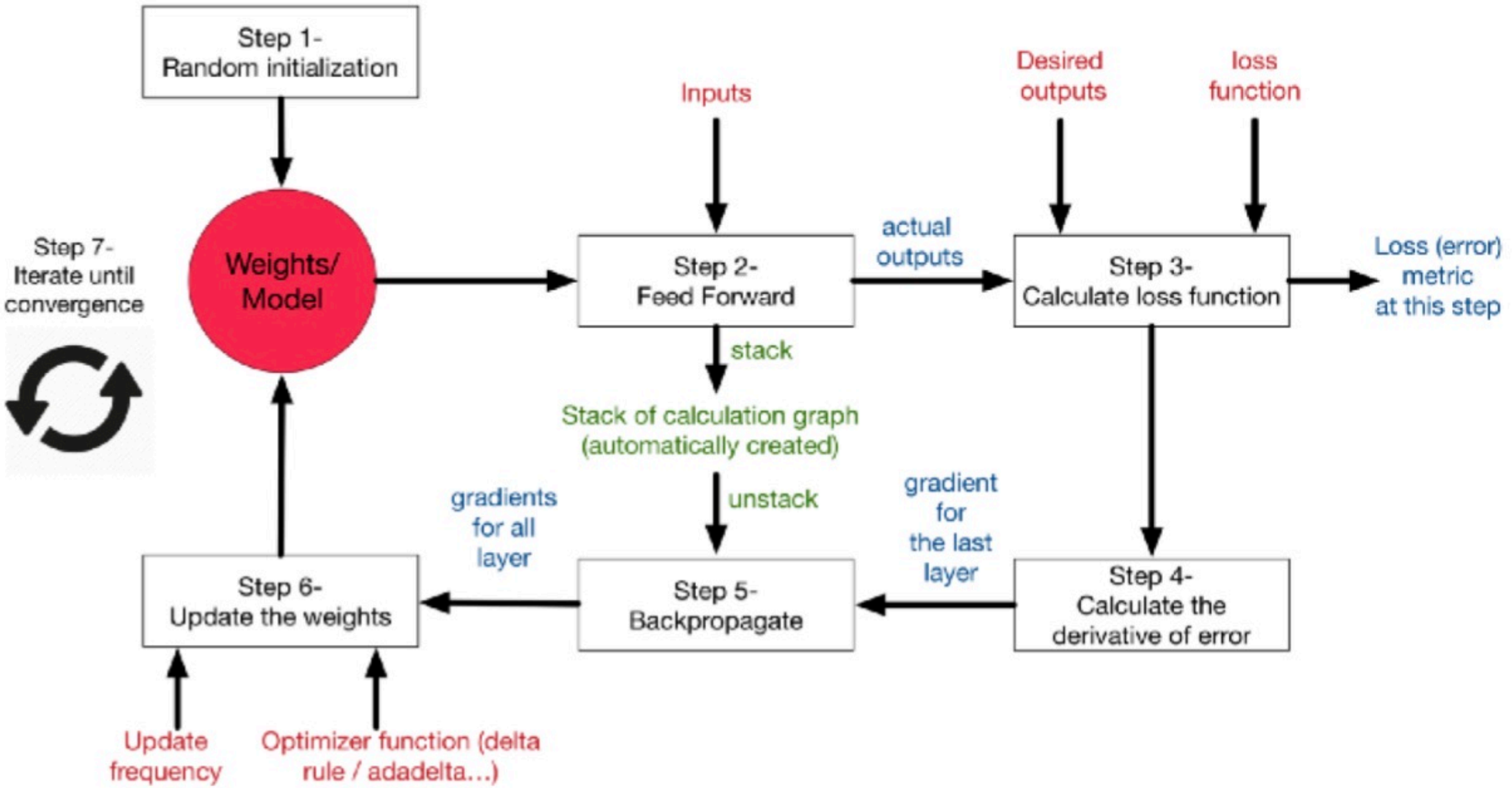
Backpropagation



Backpropagation

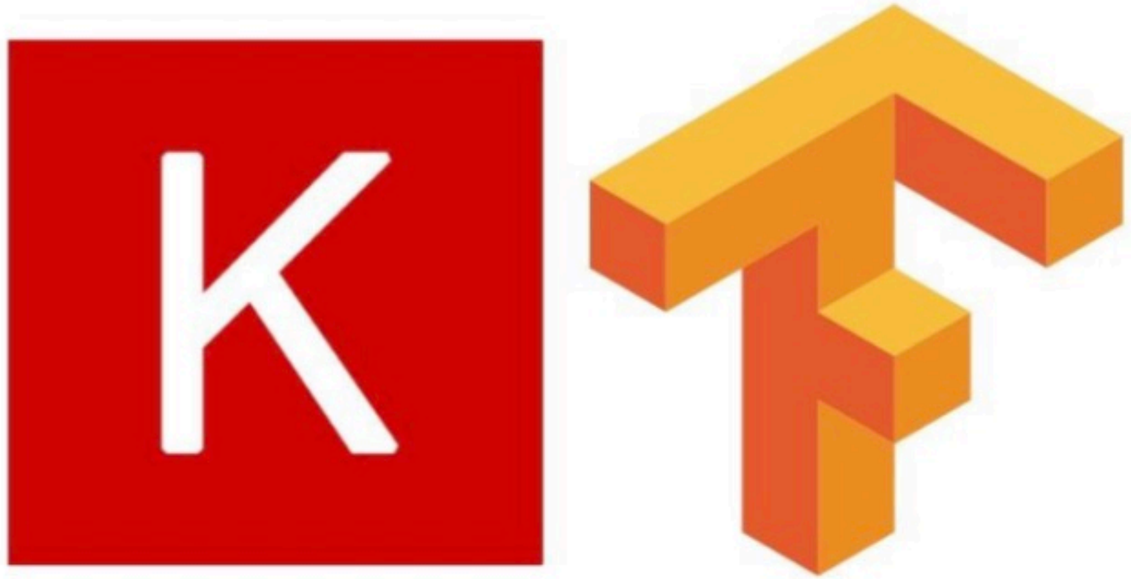


Overview


Step 1-
Random initialization
Inputs
Desired
outputs
loss
function
Step 7-
Iterate until
convergence
Weights/
Model
Step 2-
Feed Forward
actual
outputs
Step 3-
Calculate loss function
Loss (error)
metric
at this step
stack
Stack of calculation graph
(automatically created)
unstack
gradients
for all
layer
gradient
for
the last
layer
Step 6-
Update the weights
Step 5-
Backpropagate
Step 4-
Calculate the
derivative of error
Update
frequency
Optimizer function (delta
rule / adadelta...)

Keras vs TensorFlow

Keras vs TensorFlow



TensorFlow

The most popular library used to build Deep Learning models.

Keras

A high-level API that is built on top of TensorFlow.

```
from tensorflow.keras.models import Sequential
or
from tensorflow.keras.layers import Dense, Activation
```

Keras vs TensorFlow

Features of TensorFlow:

- GPU acceleration for faster training and inference.
- Has a high level API implemented within the ecosystem (Keras)
- Contains pre-trained models and datasets
- TFLite allows deployment on small edge devices.
- Tensorboard, a kit using TensorFlow's visualization toolkit
- Open Source

Keras vs TensorFlow

Why TensorFlow is popular?

- Backed by Google (lots of funding for active development)
- Lots of companies in the industry use it (not a lot of researchers tho)
- Convenient deployment & monitoring (TF Extended is great)

Keras vs TensorFlow

Library	Rank	Overall	Github	Stack Overflow	Google Results
tensorflow	1	10.87	4.25	4.37	2.24
keras	2	1.93	0.61	0.83	0.48
caffe	3	1.86	1.00	0.30	0.55
theano	4	0.76	-0.16	0.36	0.55
pytorch	5	0.48	-0.20	-0.30	0.98
sonnet	6	0.43	-0.33	-0.36	1.12
mxnet	7	0.10	0.12	-0.31	0.28
torch	8	0.01	-0.15	-0.01	0.17
cntk	9	-0.02	0.10	-0.28	0.17
dlib	10	-0.60	-0.40	-0.22	0.02
caffe2	11	-0.67	-0.27	-0.36	-0.04
chainer	12	-0.70	-0.40	-0.23	-0.07
paddlepaddle	13	-0.83	-0.27	-0.37	-0.20
deeplearning4j	14	-0.89	-0.06	-0.32	-0.51
lasagne	15	-1.11	-0.38	-0.29	-0.44
bigdl	16	-1.13	-0.46	-0.37	-0.30
dynet	17	-1.25	-0.47	-0.37	-0.42
apache singa	18	-1.34	-0.50	-0.37	-0.47
nvidia digits	19	-1.39	-0.41	-0.35	-0.64
matconvnet	20	-1.41	-0.49	-0.35	-0.58
tflearn	21	-1.45	-0.23	-0.28	-0.94
nervana neon	22	-1.65	-0.39	-0.37	-0.89
opennn	23	-1.97	-0.53	-0.37	-1.07

Source: The Data Incubator

TensorFlow key functionalities:

- TensorFlow provides an **accessible and readable syntax**.
- TensorFlow provides **excellent functionalities and services**.
- TensorFlow is a low-level library which provides **more flexibility**.
- TensorFlow provides **more network control**.

What is Tensor?

What is Tensor?

- Tensors are a type of data structure used in **linear algebra**, and **like vectors and matrices**, you can calculate **arithmetic operations** with tensors.

In the general case, an array of numbers arranged on a regular grid with a variable number of axes is known as a tensor

- Like vectors and matrices, tensors can be represented **in Python** using the **N-dimensional array (ndarray)**.

rank-0-tensor => scalar value
rank-1-tensor => vector / 1D array
rank-2-tensor => matrix / 2D array
rank-3-tensor => 3D array
…

What is Tensor?

Numpy	TensorFlow
`a = np.zeros((2,2)); b = np.ones((2,2))`	`a = tf.zeros((2,2)), b = tf.ones((2,2))`
`np.sum(b, axis=1)`	`tf.reduce_sum(a,reduction_indices=[1])`
`a.shape`	`a.get_shape()`
`np.reshape(a, (1,4))`	`tf.reshape(a, (1,4))`
`b * 5 + 1`	`b * 5 + 1`
`np.dot(a,b)`	`tf.matmul(a, b)`
`a[0,0], a[:,0], a[0,:]`	`a[0,0], a[:,0], a[0,:]`