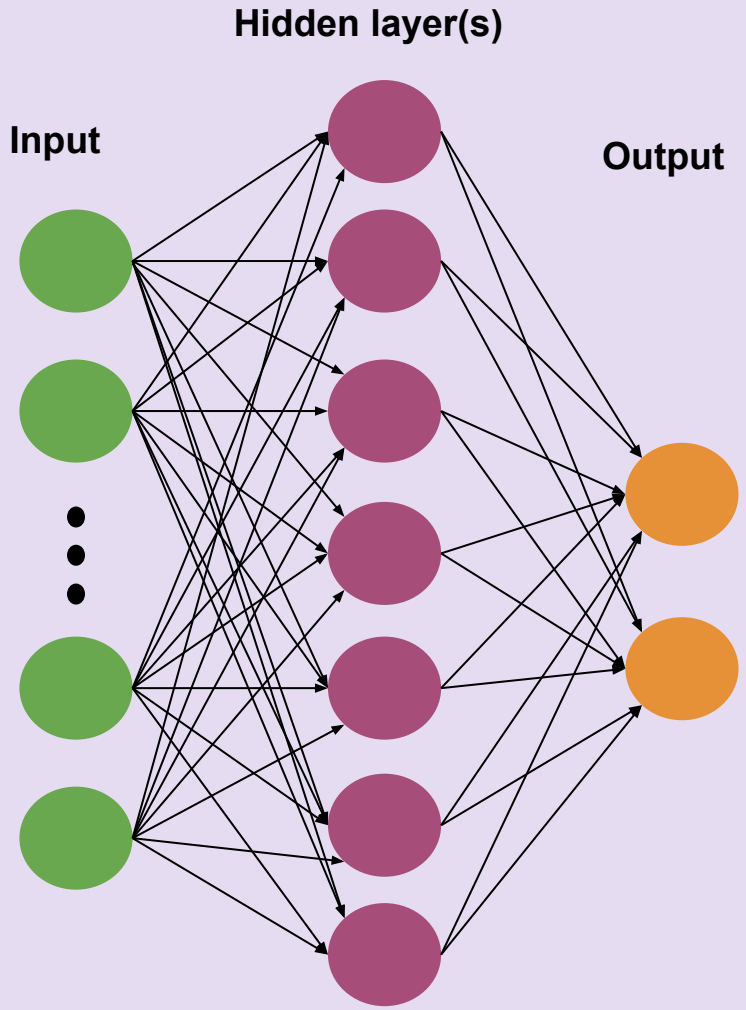
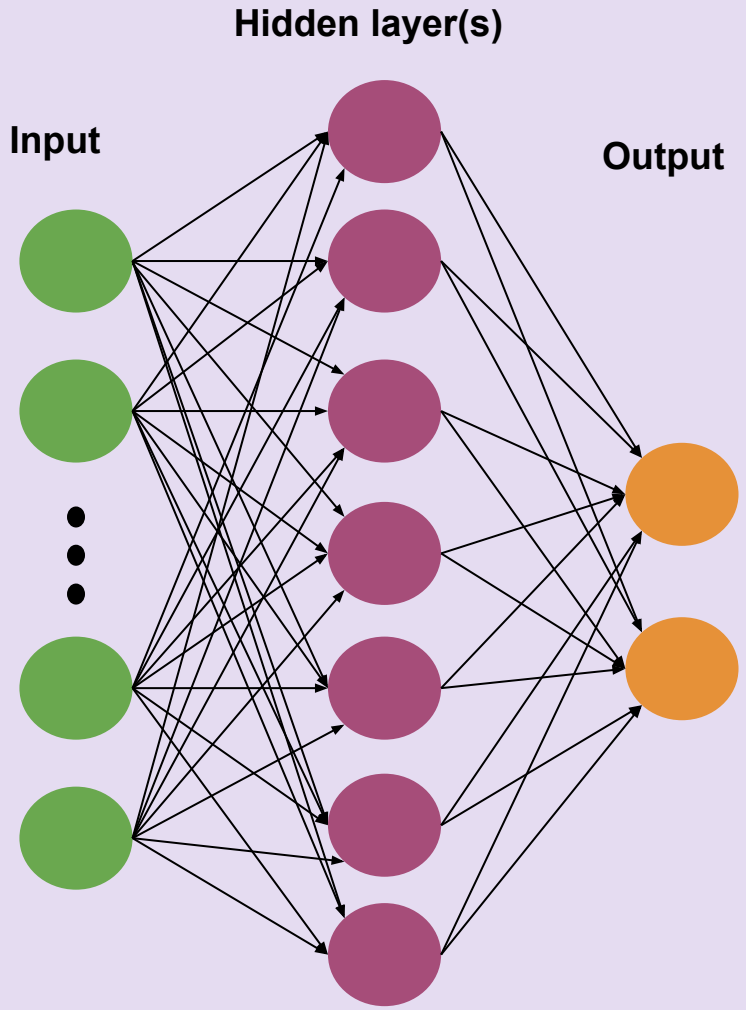
Introduction to Neural Networks

What is a Neural Network & how does it work?

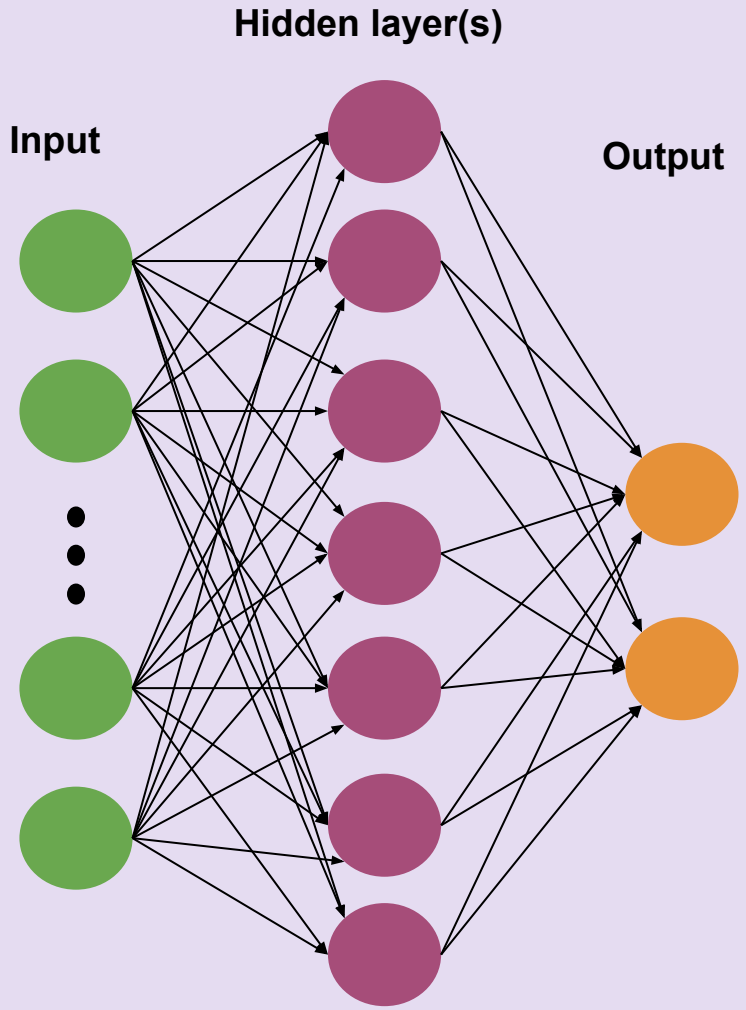


What is a Neural Network & how does it work?



This is how your brain works!

What is a neural network & how does it work?

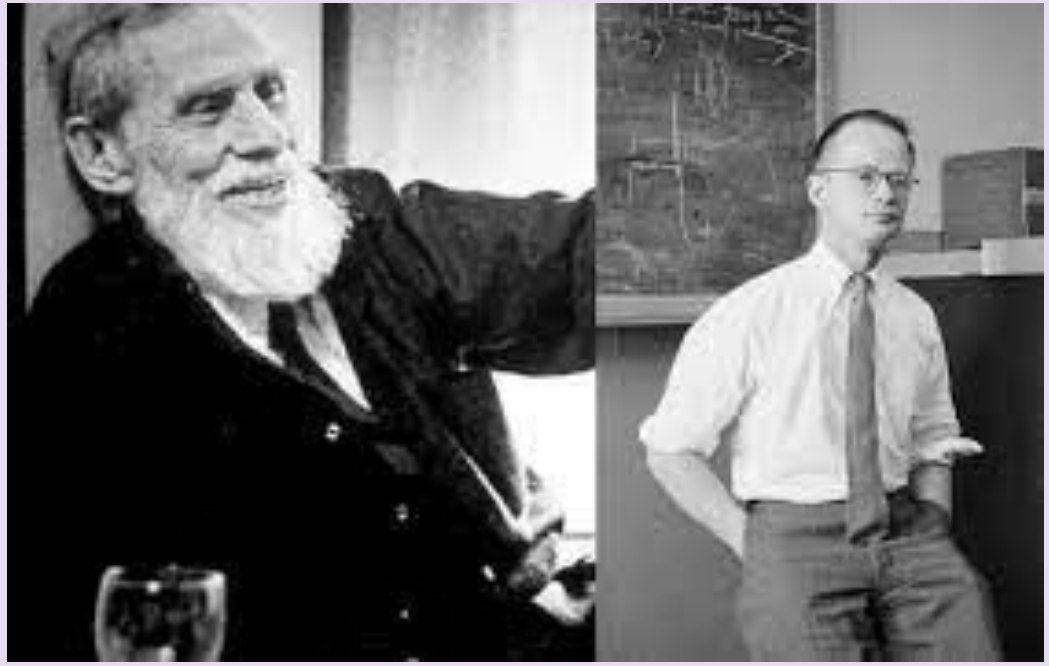


This is how your brain works!

Neural Network History

1943

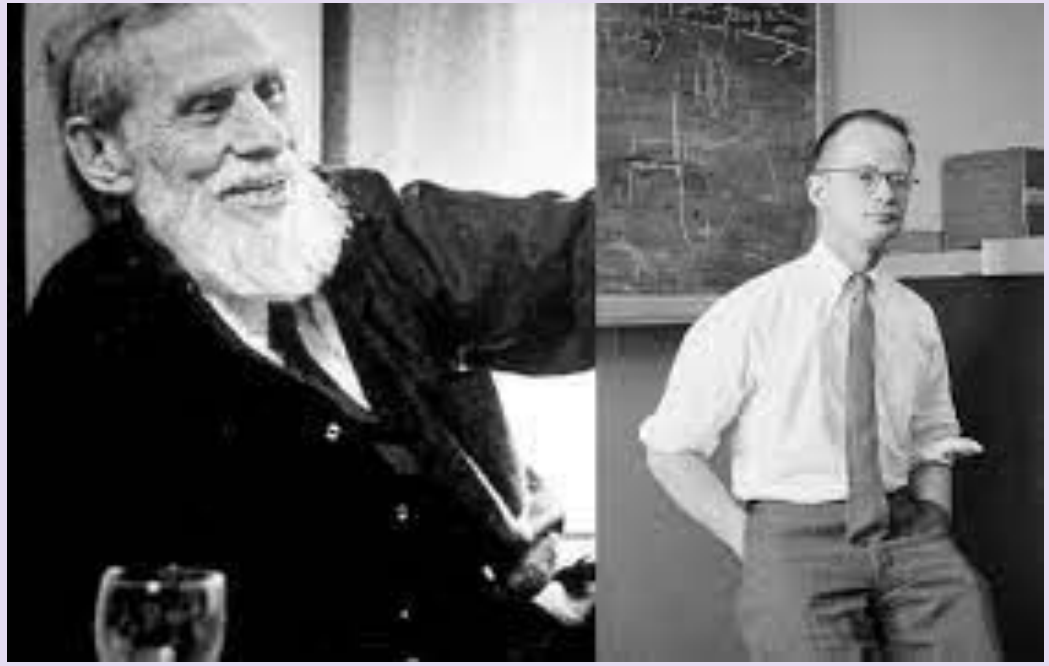
Electronic Brain → neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper on how neurons might work. In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits **to simulate intelligent behaviour termed 'connectionism'.**



Neural Networks History

1943

Electronic Brain → neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper on how neurons might work. In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits **to simulate intelligent behaviour termed 'connectionism'.**



1958

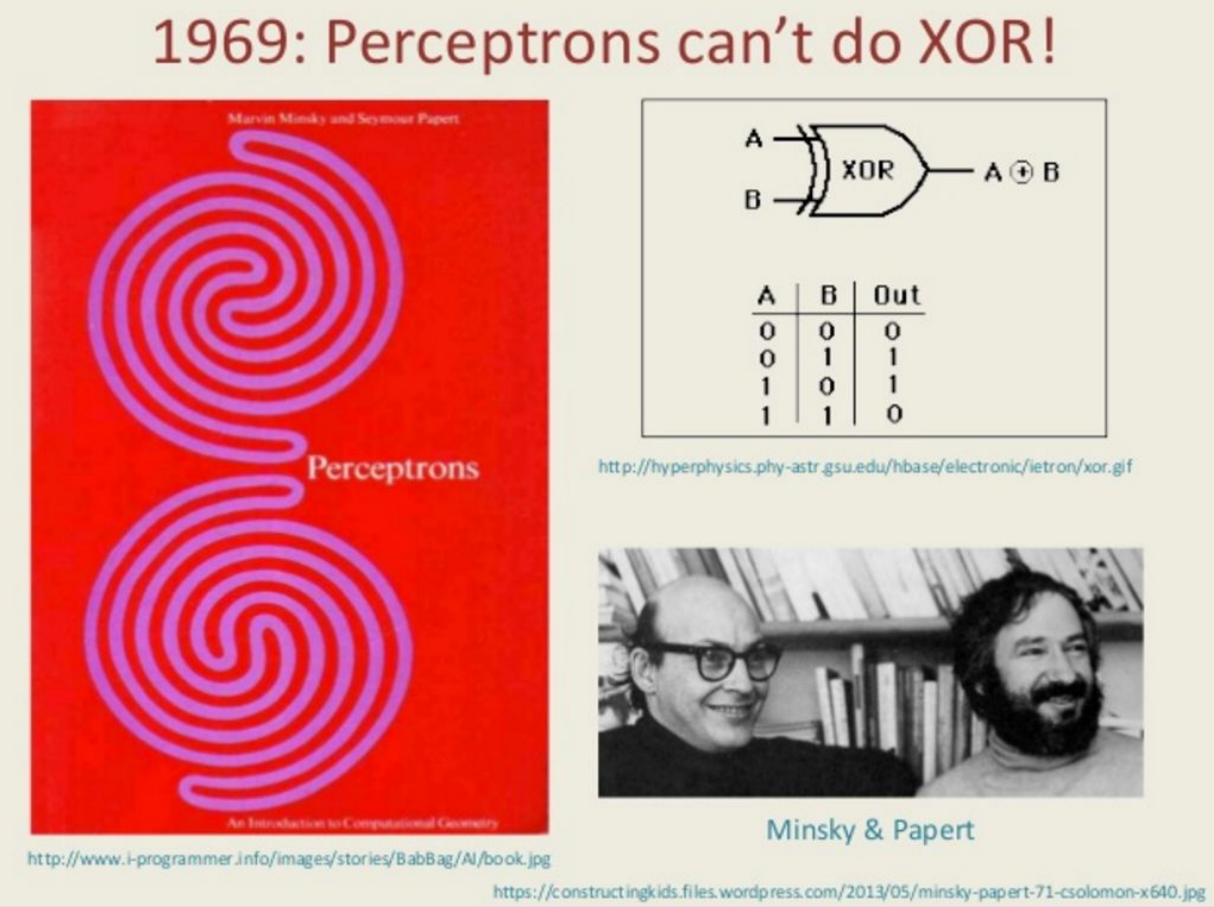
In 1958, Frank Rosenblatt inspired by their work invented the **perceptron model.**

Amazingly, even back then he saw huge potential:
“ a perceptron may eventually be able to learn, make decisions and translate languages.”

Neural Networks History

1969

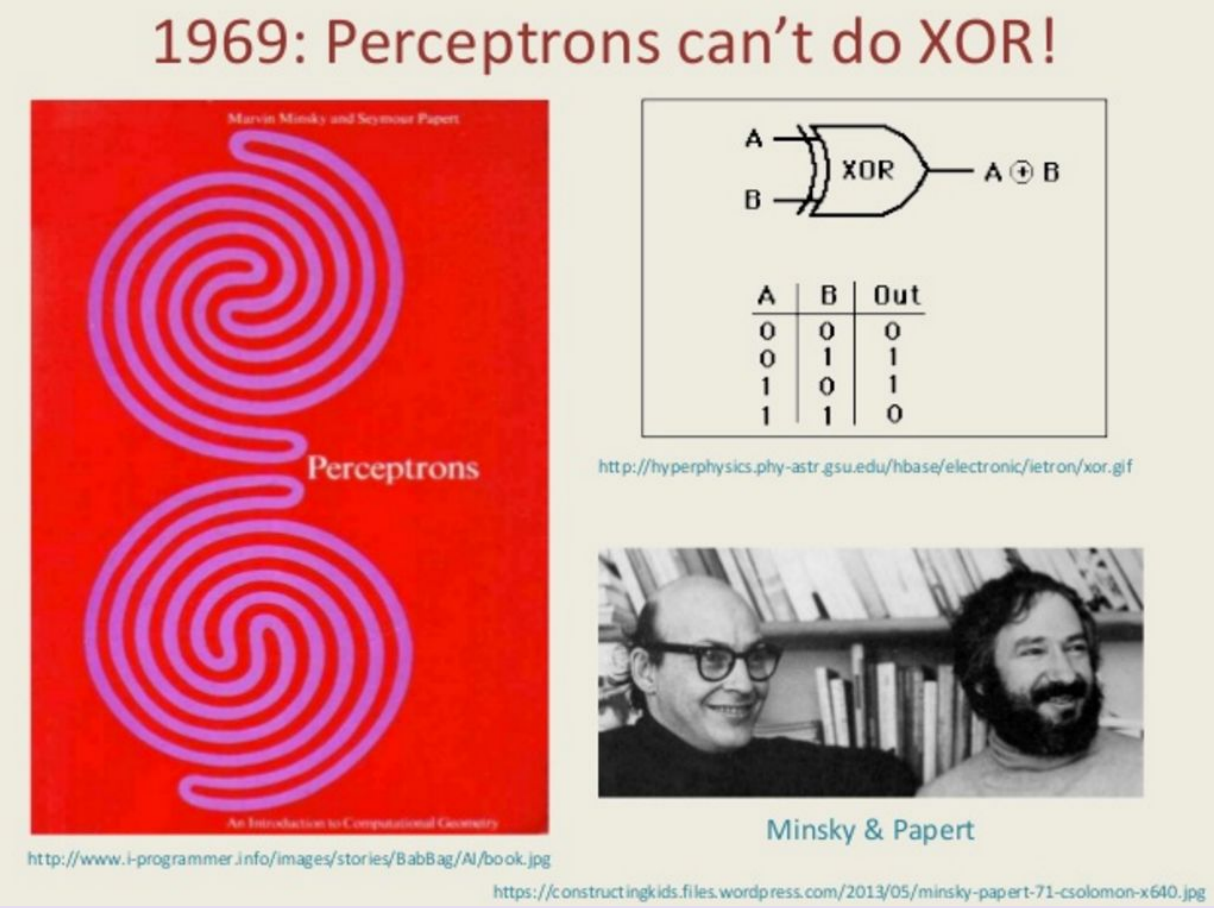
This lead to 1969 Minsky and Papert to publish a book “ Perceptrons” describing **the severe limitations to what perceptrons could do**!!!



Neural Networks History

1969

This lead to 1969 Minsky and Papert to publish a book “ Perceptrons” describing **the severe limitations to what perceptrons could do**!!!

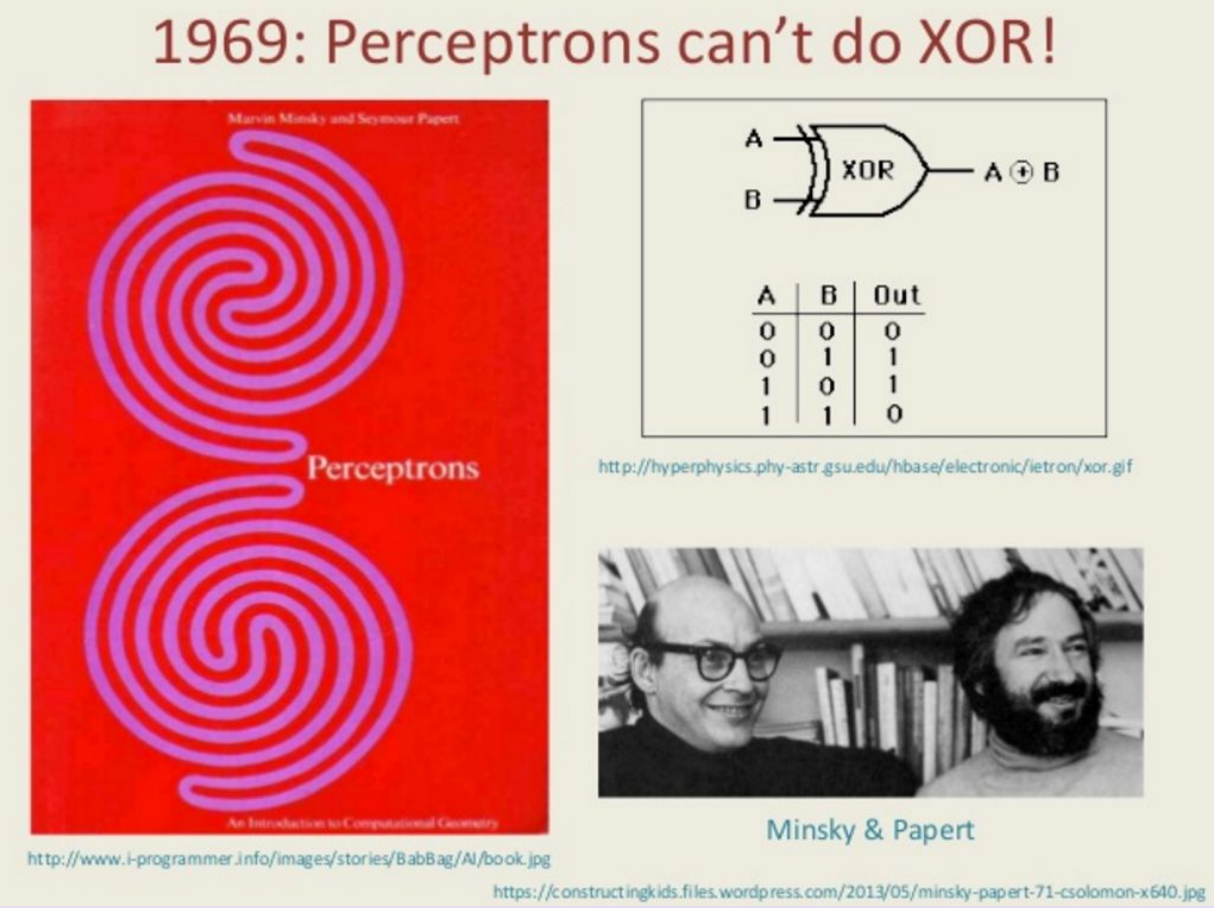


This marked the beginning of what is known as the **AI winter.**

Neural Networks History

1969

This lead to 1969 Minsky and Papert to publish a book “ Perceptrons” describing **the severe limitations to what perceptrons could do**!!!

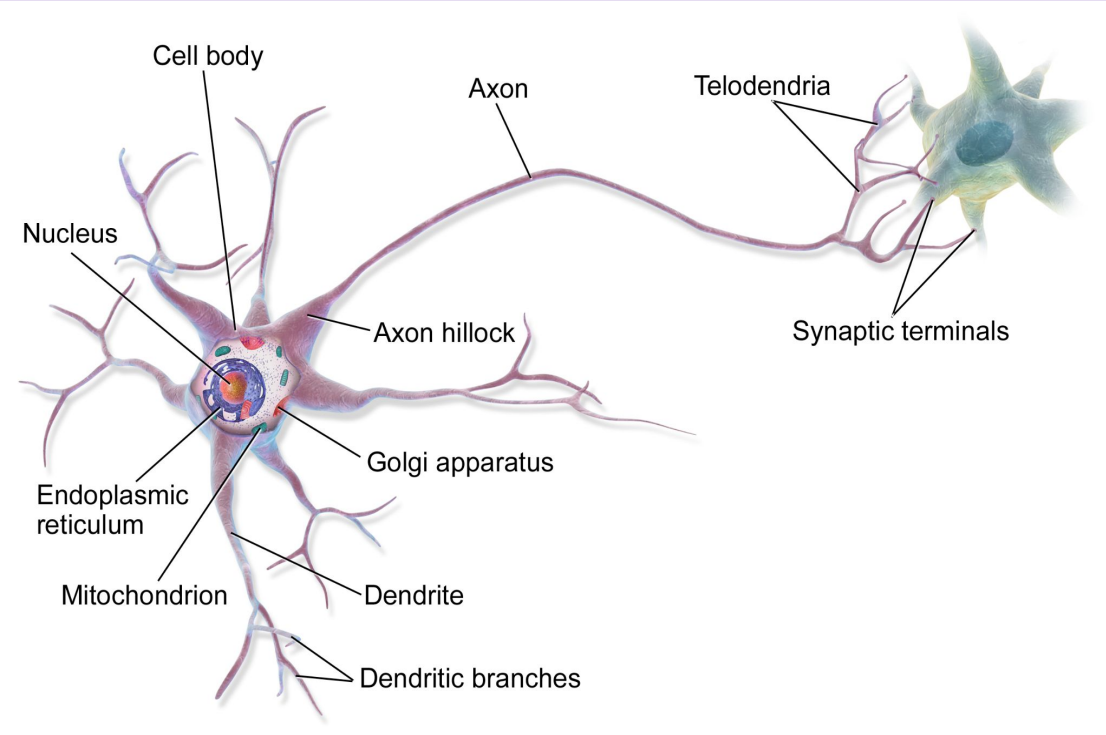


This marked the beginning of what is known as the **AI winter.**

Perceptron Model

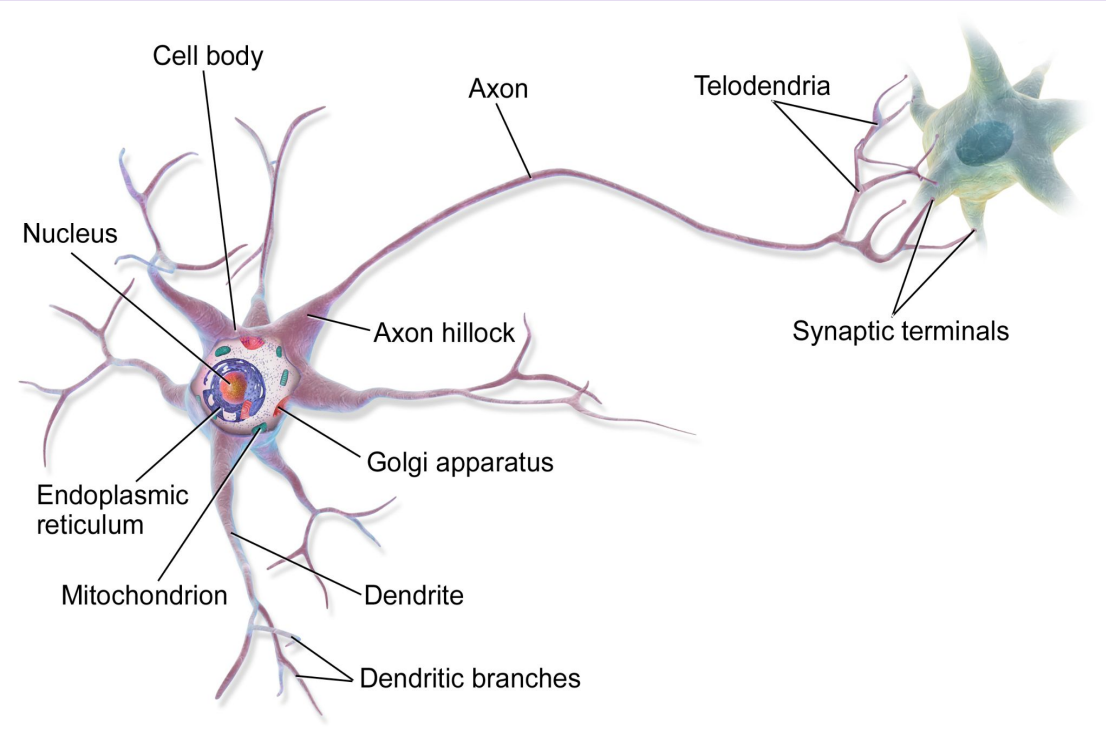
Perceptron Model

Illustration of biological neurons.

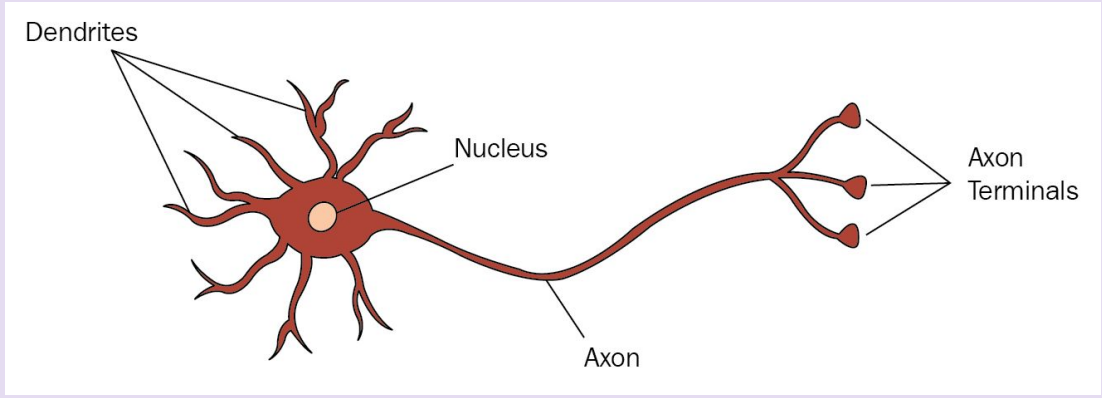


Perceptron Model

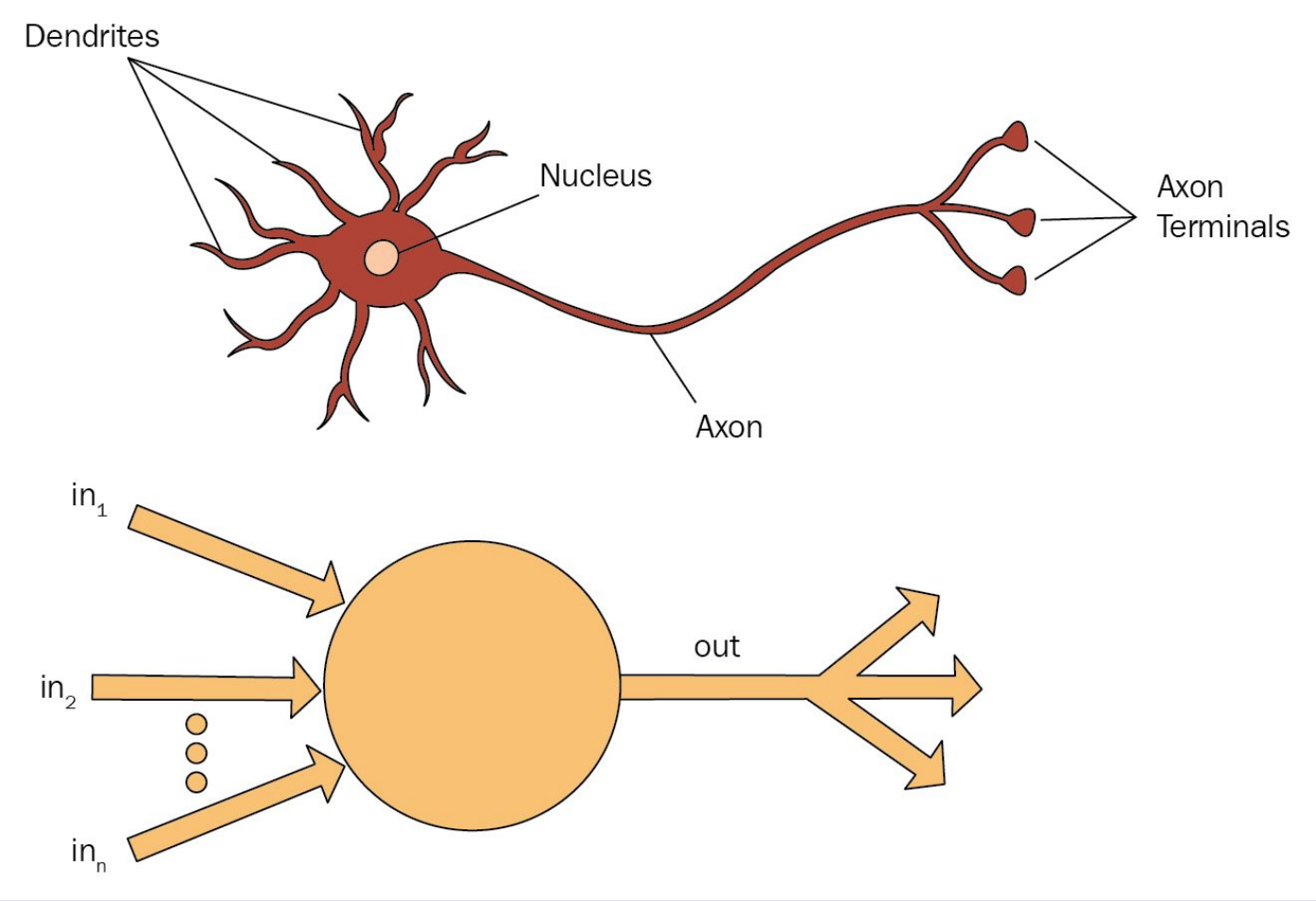
Illustration of biological neurons.



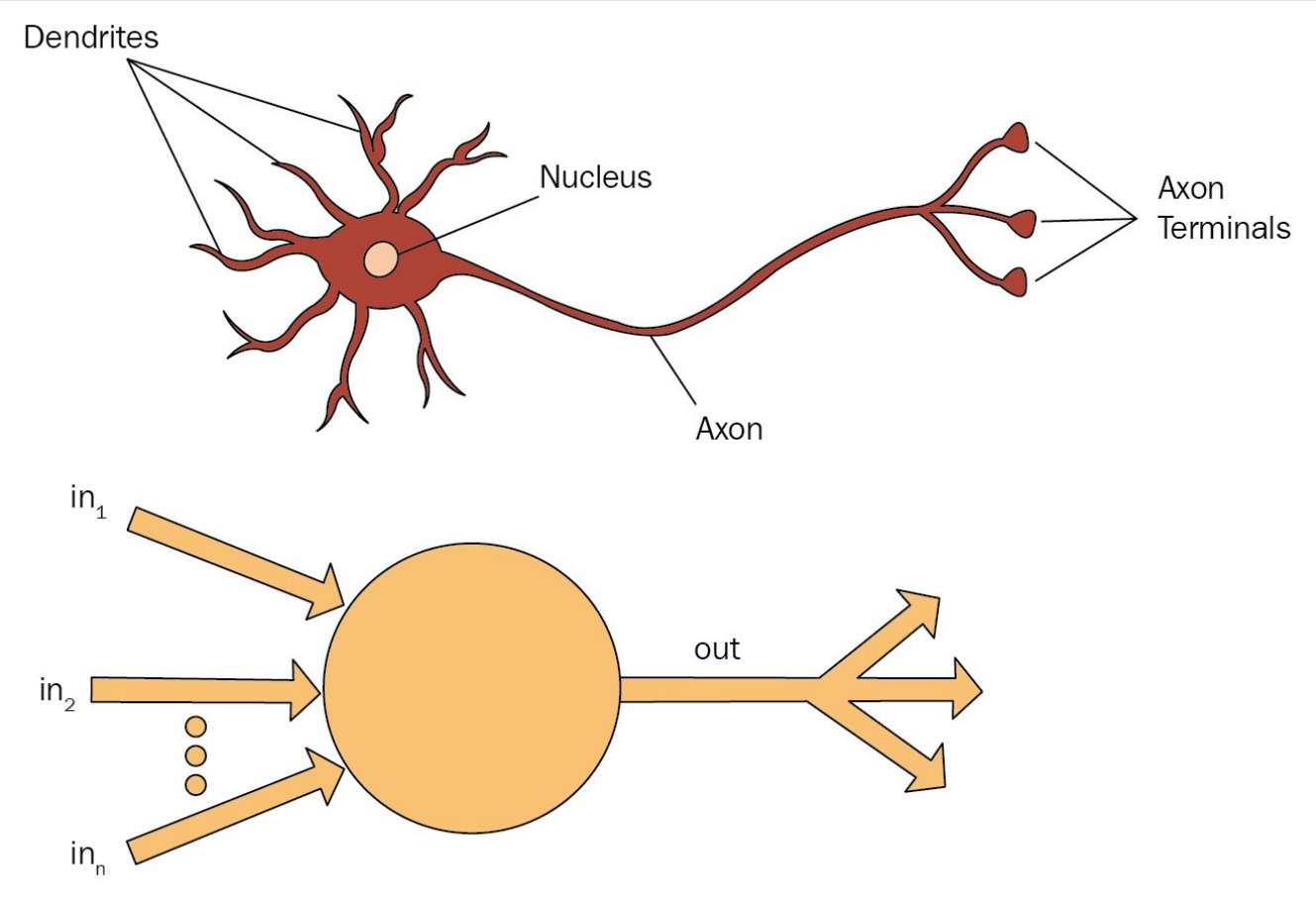
Simplified illustration of biological neurons.

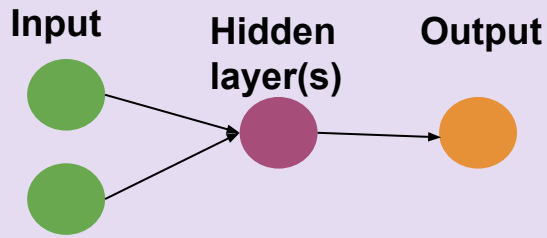


Perceptron Model

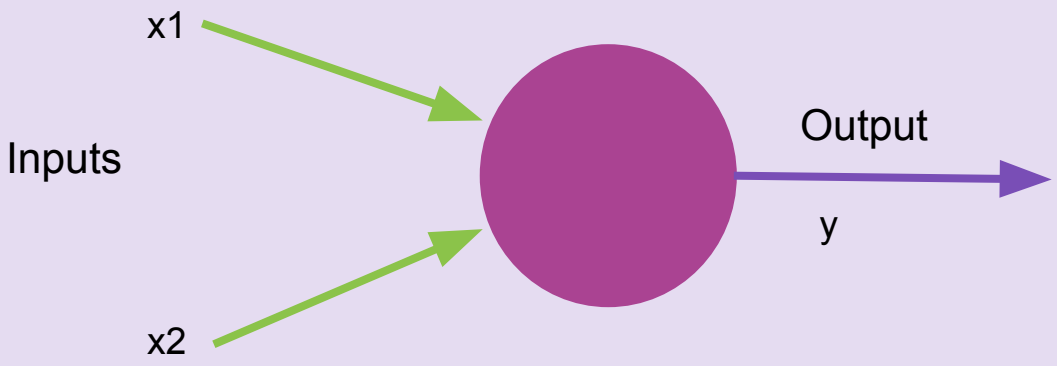


Perceptron Model

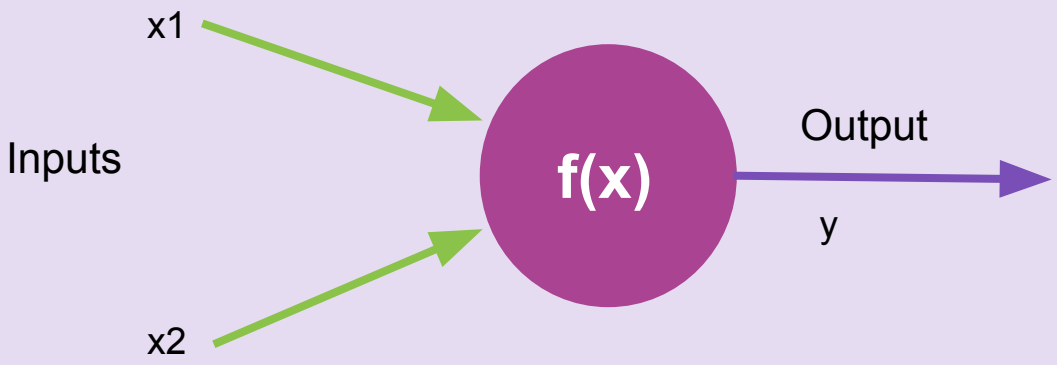




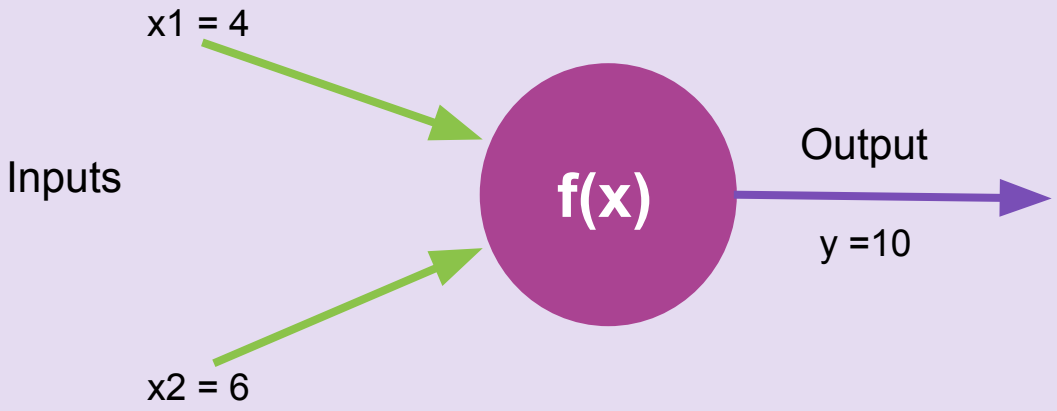
Perceptron Model



Perceptron Model

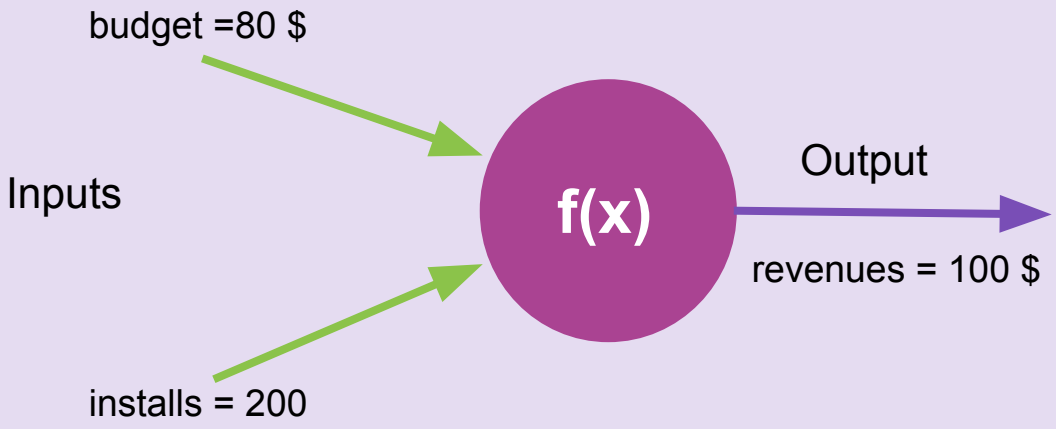


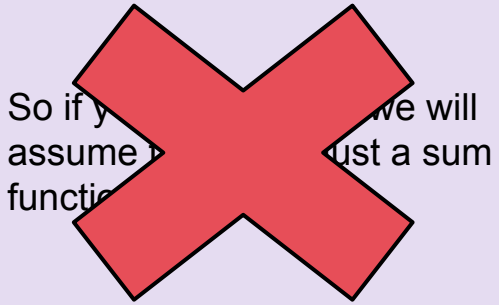
Perceptron Model



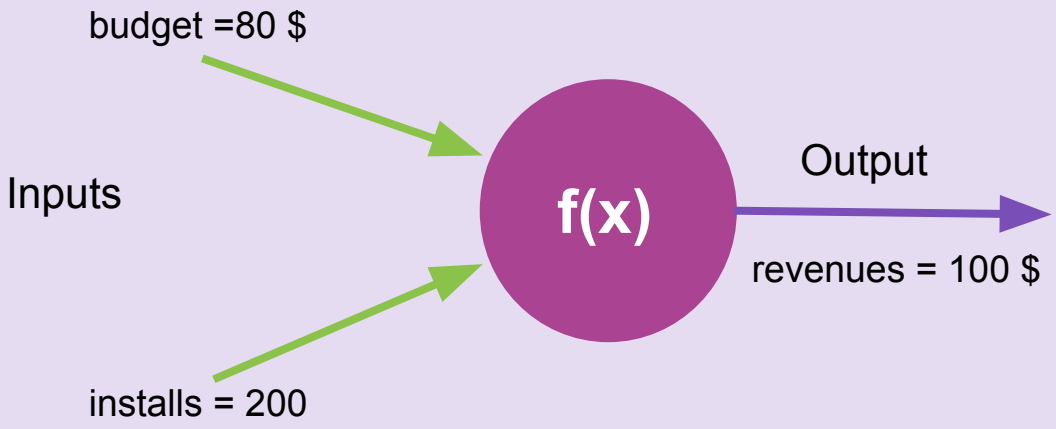
So if y = x1 + x2 then we will assume that f(x) is just a sum function.

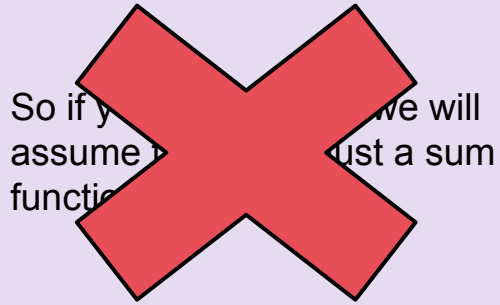
Perceptron Model





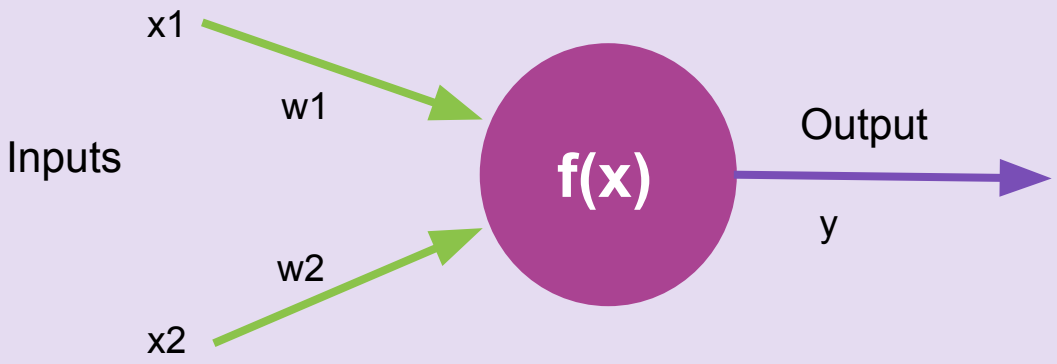
Perceptron Model





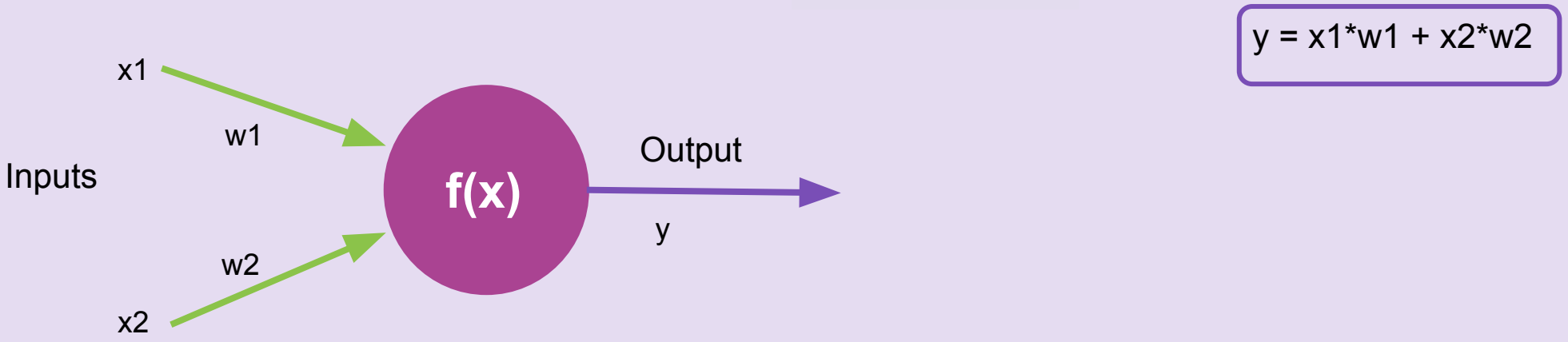
Realistically, we would want to be able to **adjust some parameters** in order for the perceptron to “ **learn**” so that it can correct the output of y.

Perceptron Model



We can add an adjustable weight which we multiply to the input.

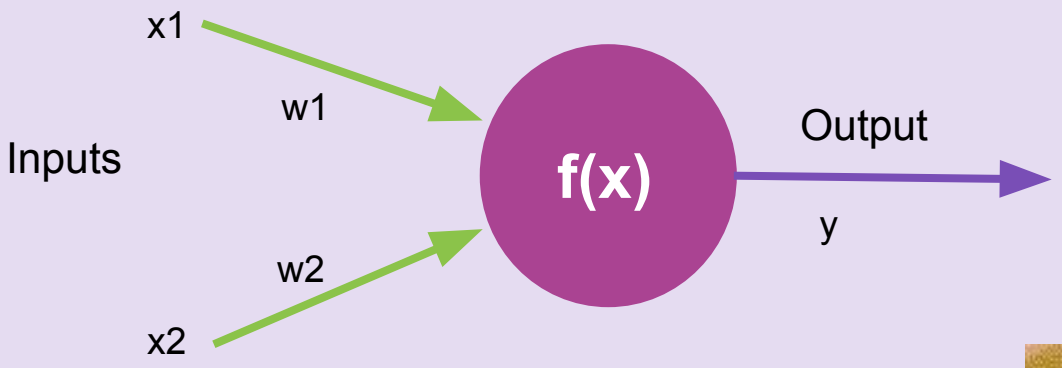
Perceptron Model



$$y = x1*w1 + x2*w2$$

We can add an adjustable weight which we multiply to the input.

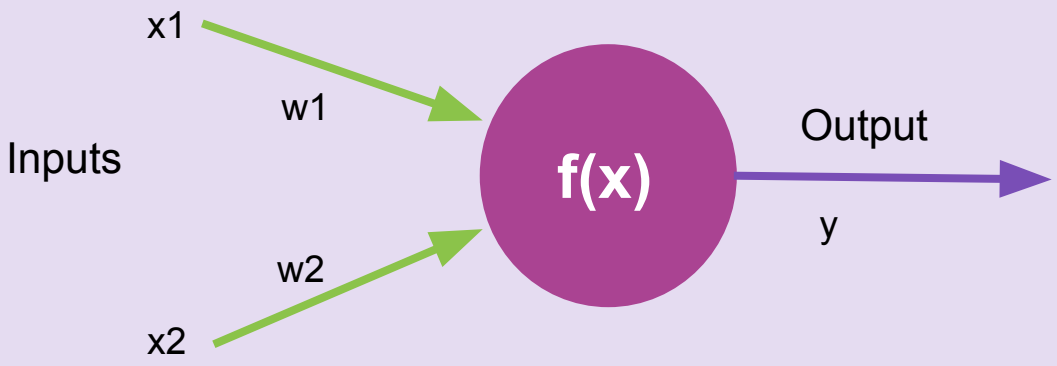
Perceptron Model



y = x1*w1 + x2*w2

And we will need to adjust it somehow (the weights) in order to get the correct value of y.

Perceptron Model

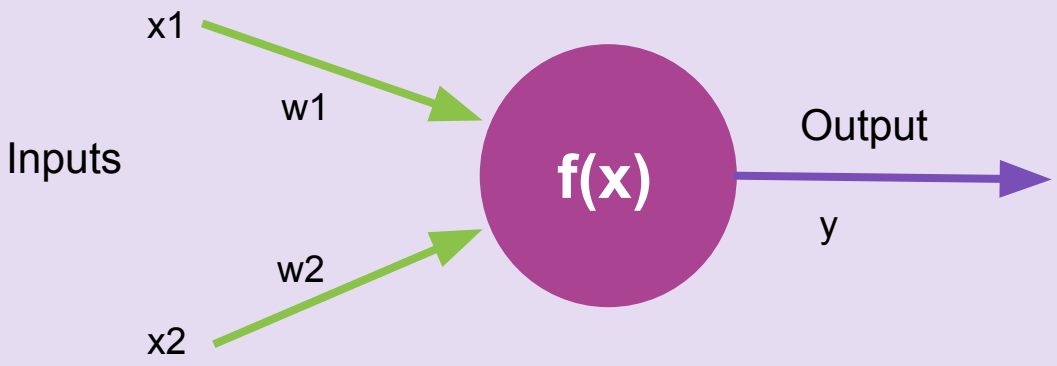


$$y = x1*w1 + x2*w2$$

And we will need to adjust it somehow (the weights) in order to get the correct value of y.

But what happen if an **x** is zero? → **w** won't change anything!

Perceptron Model



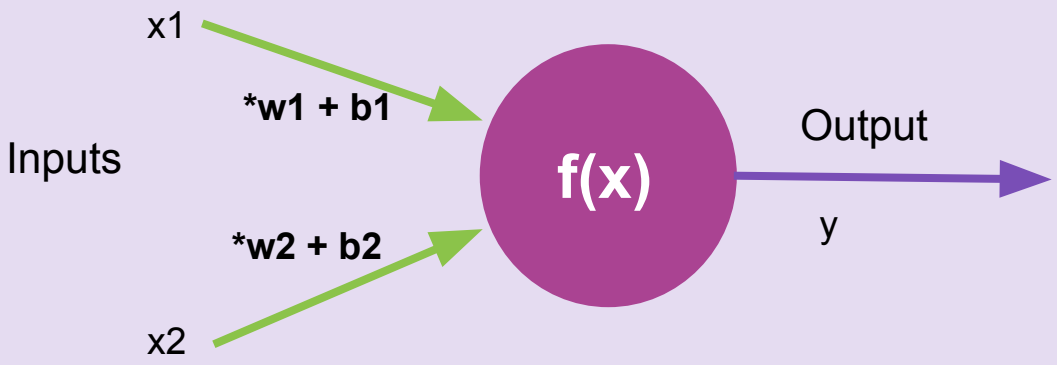
$$y = x1*w1 + x2*w2$$

And we will need to adjust it somehow (the weights) in order to get the correct value of y.

But what happen if an **x** is zero? → **w** won't change anything!

To fix this problem let's add in a **bias** term **b** to the inputs.

Perceptron Model

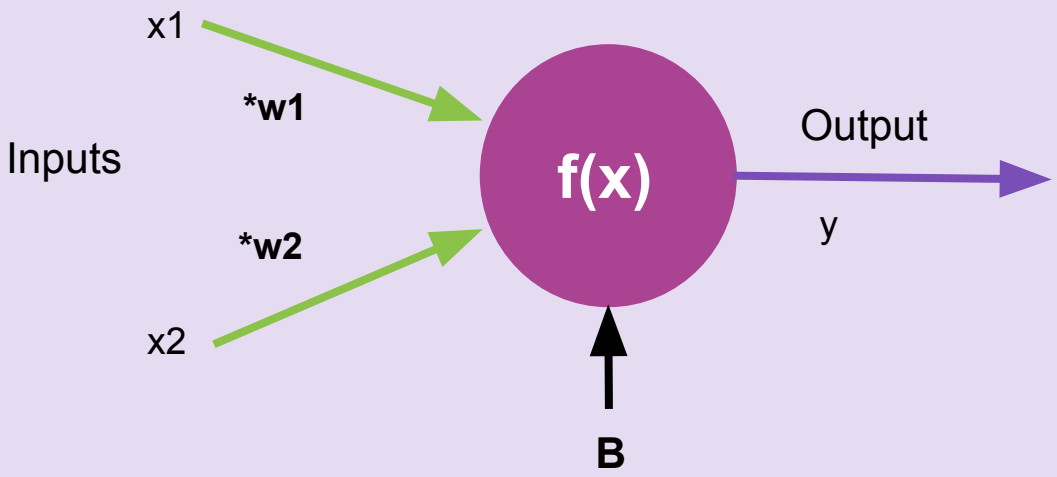


$y = (x1*w1 + b1) + (x2*w2 + b2)$

$y = (x1*w1 + x2*w2) + (b1 + b2)$

A way to think about bias is that the **x*w** has to **overcome the bias value** in order to start having an effect on the output y.

Perceptron Model



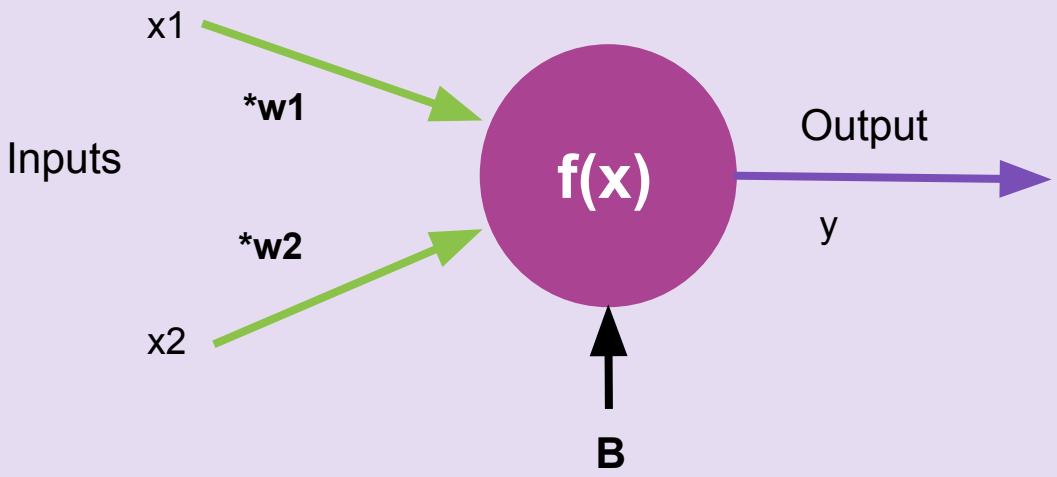
y = (x1*w1 +b1) + (x2*w2 +b2)

y = (x1*w1+ x2*w2) + (b1 + b2)

y = (x1*w1+ x2*w2) + B

W → weight tells us how important is each input
B → we can think of it as an offset value making x*w have to reach a certain threshold before having an effect.

Perceptron Model



$$y = (x1*w1 + b1) + (x2*w2 + b2)$$

$$y = (x1*w1 + x2*w2) + (b1 + b2)$$

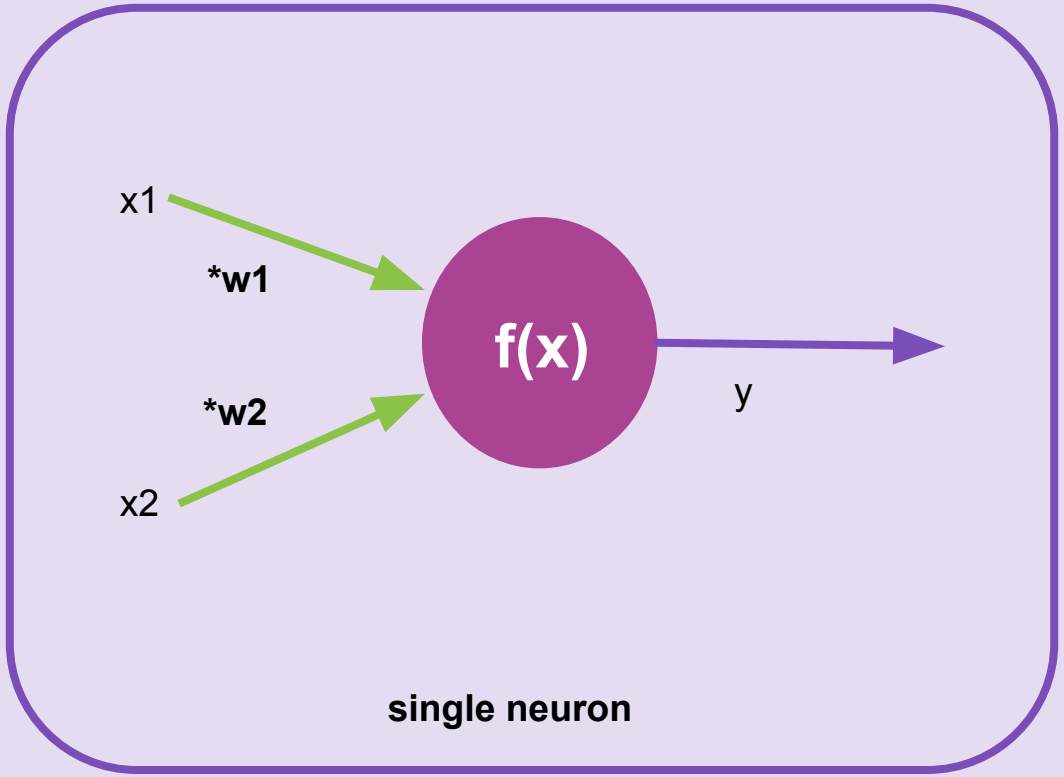
$$y = (x1*w1 + x2*w2) + B$$

Training/ Learning → finding the right setting of weights and biases of the model means we are tuning the model to predict the y as close to reality as possible.

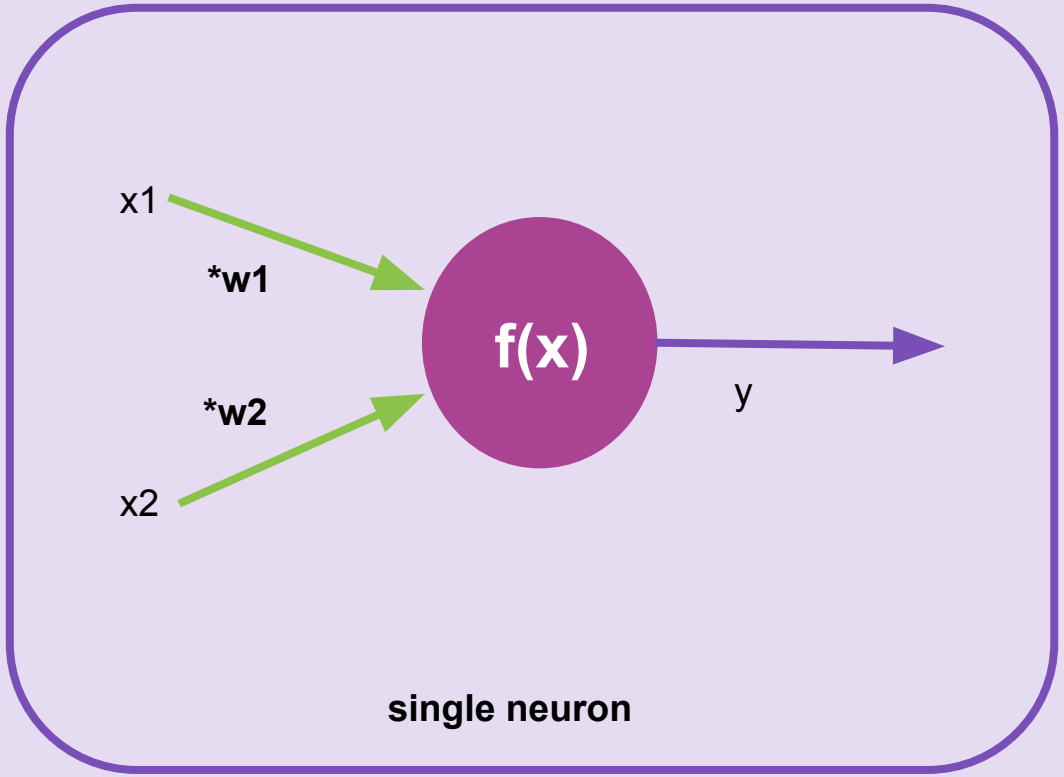
We want to set boundaries for the overall output value y.
And then pass y through some **activation function** to limit its value.

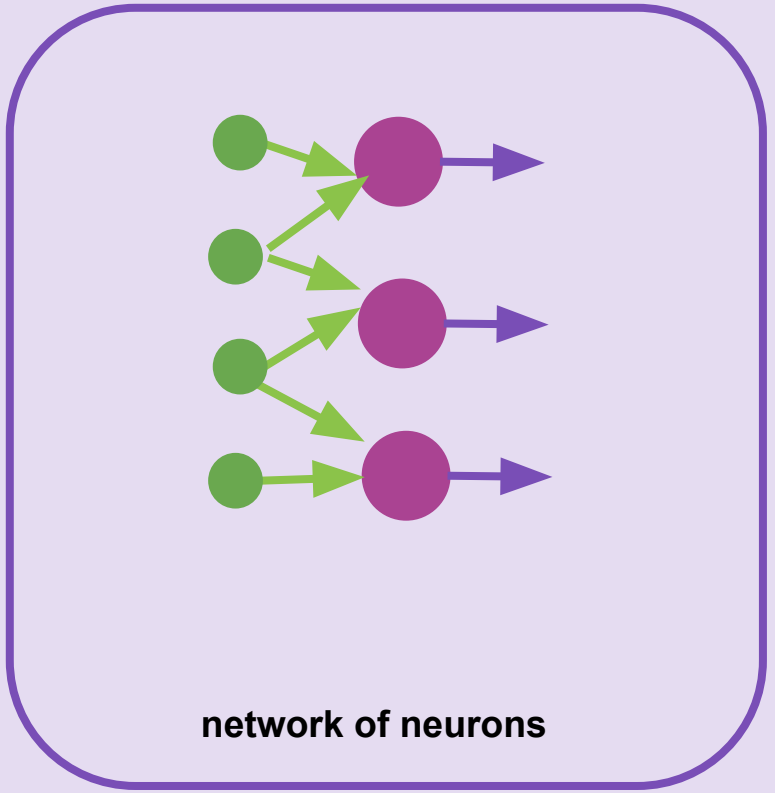
Activations function -> transforms the y into a value between (0 and 1).

Multi-layer perceptron model / Basic ANN

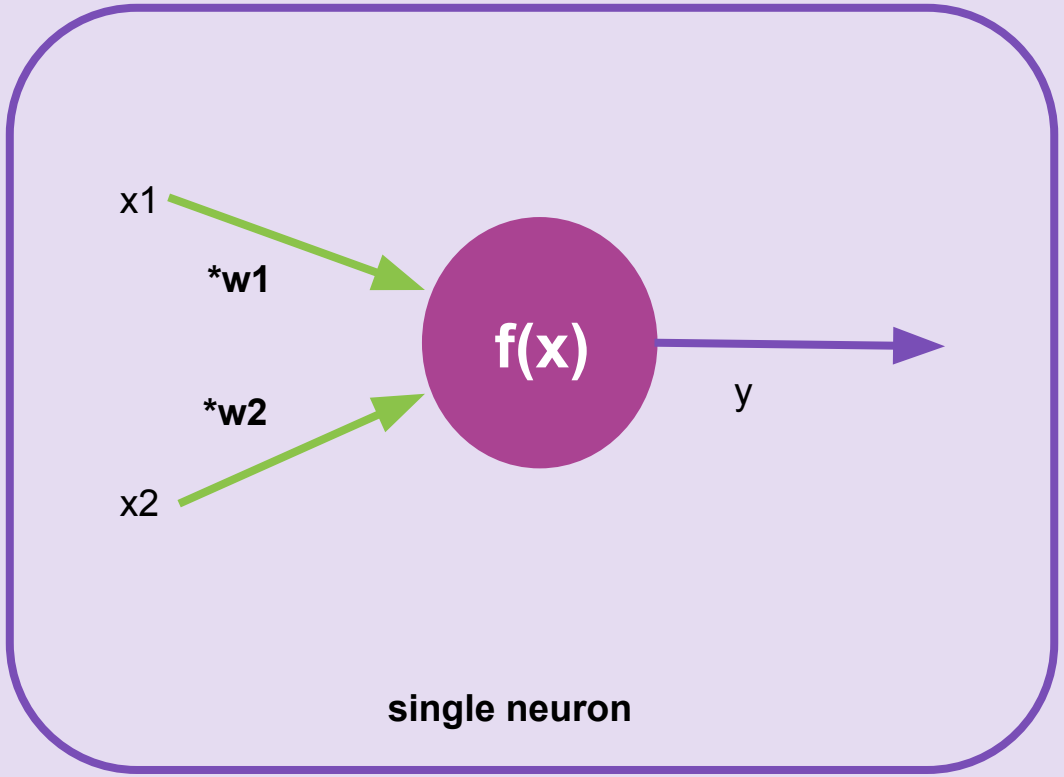


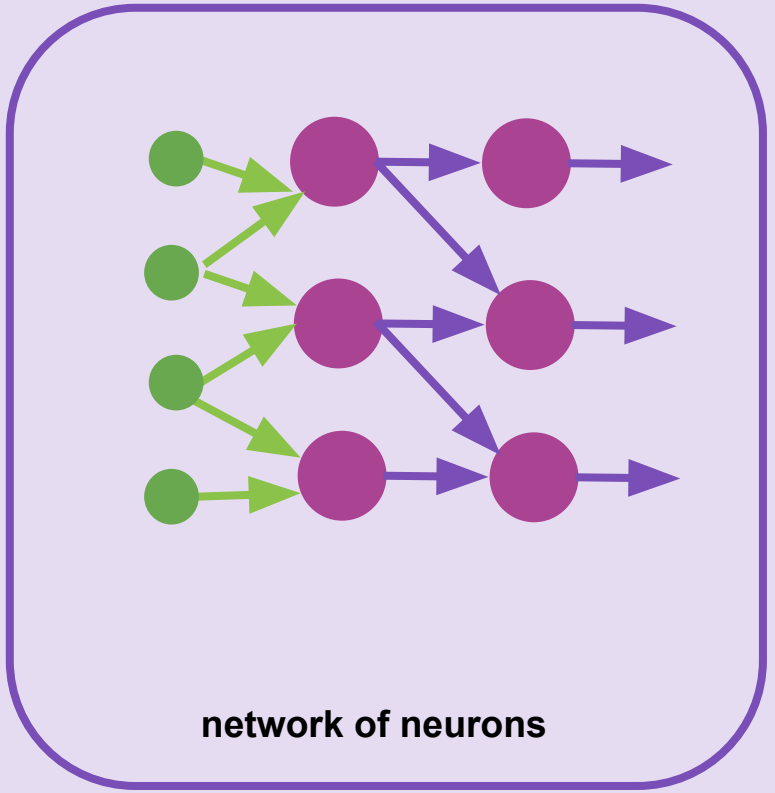
Multi-layer perceptron model / Basic ANN

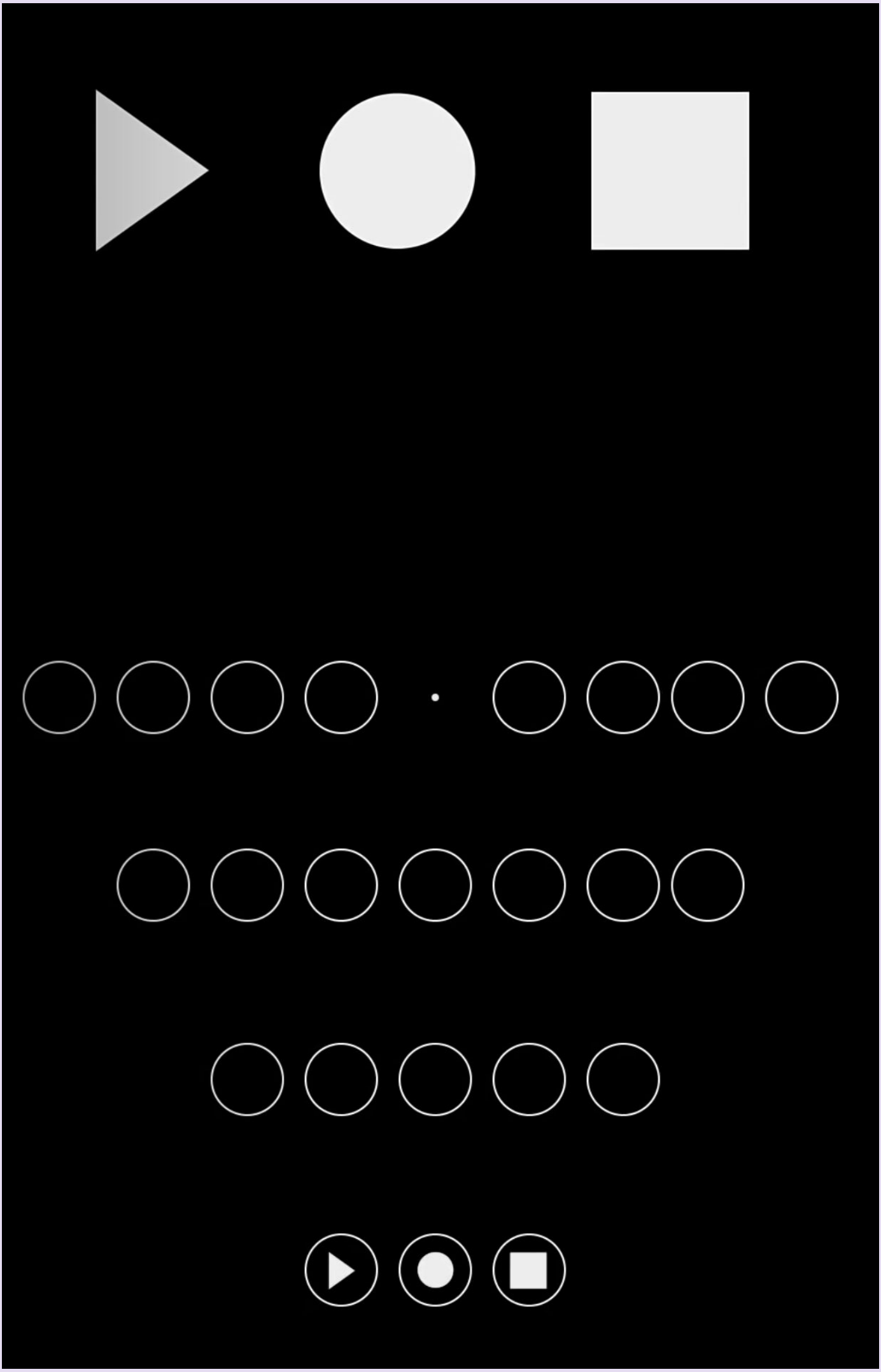


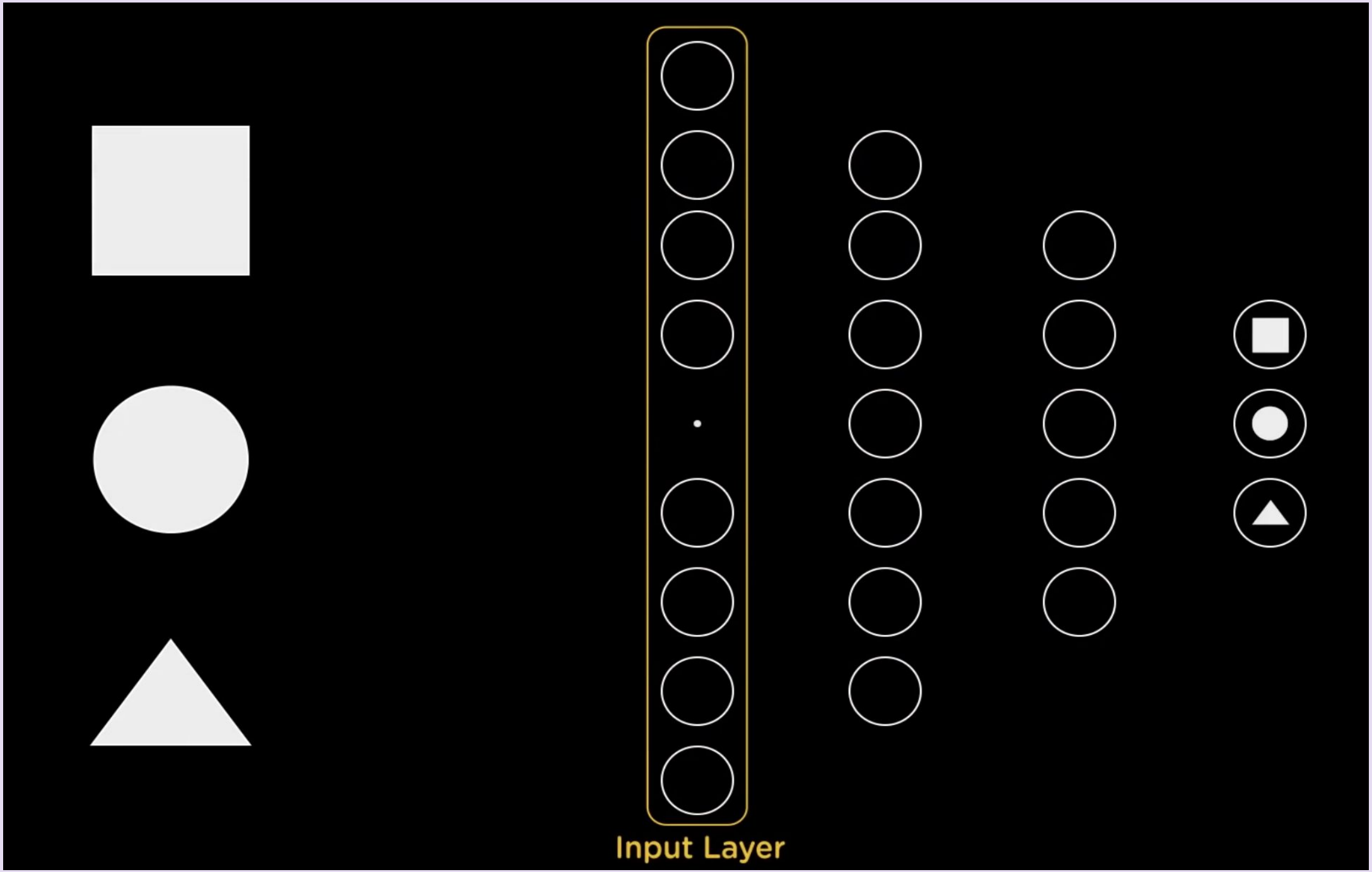


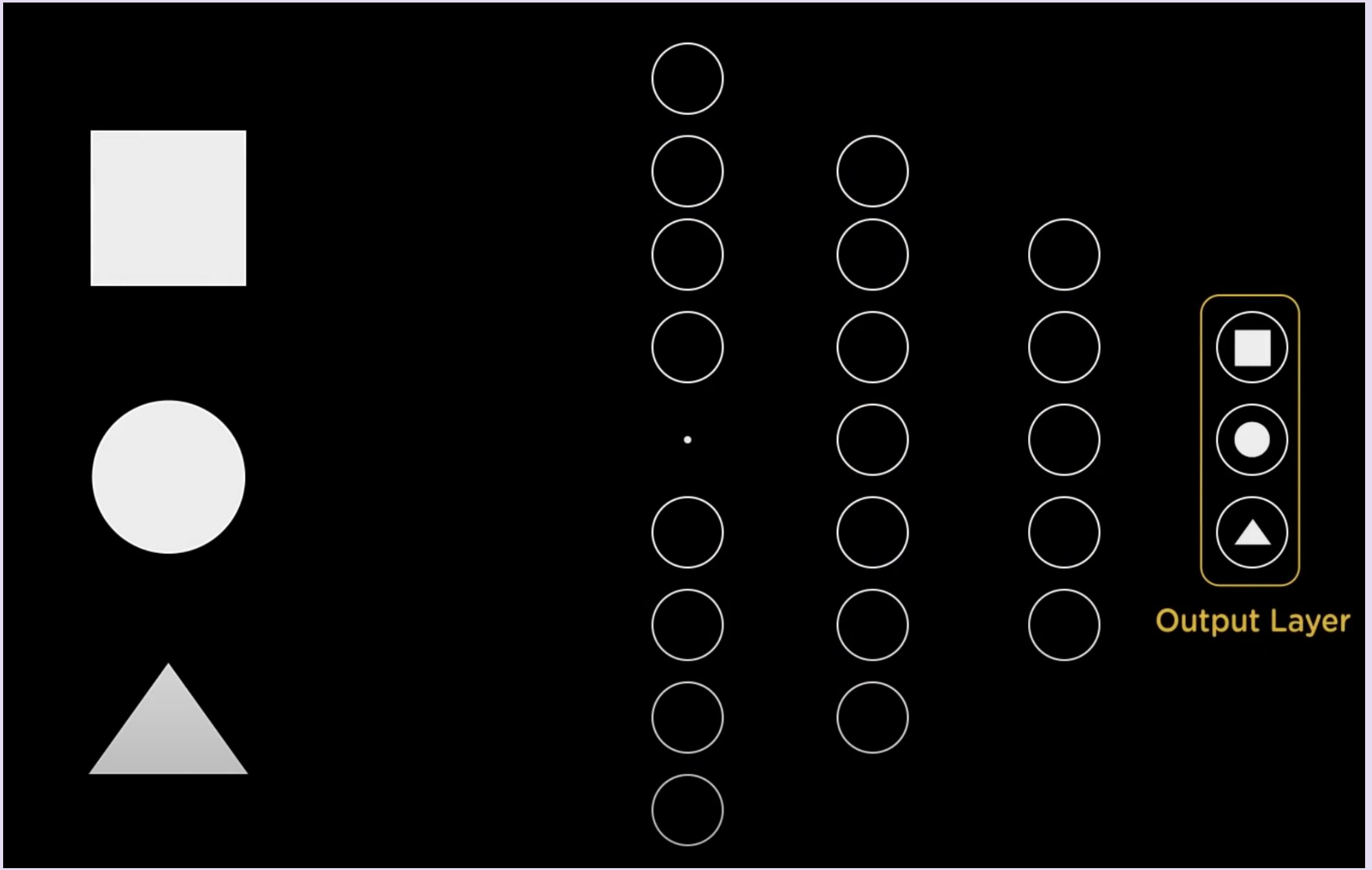
Multi-layer perceptron model / Basic ANN

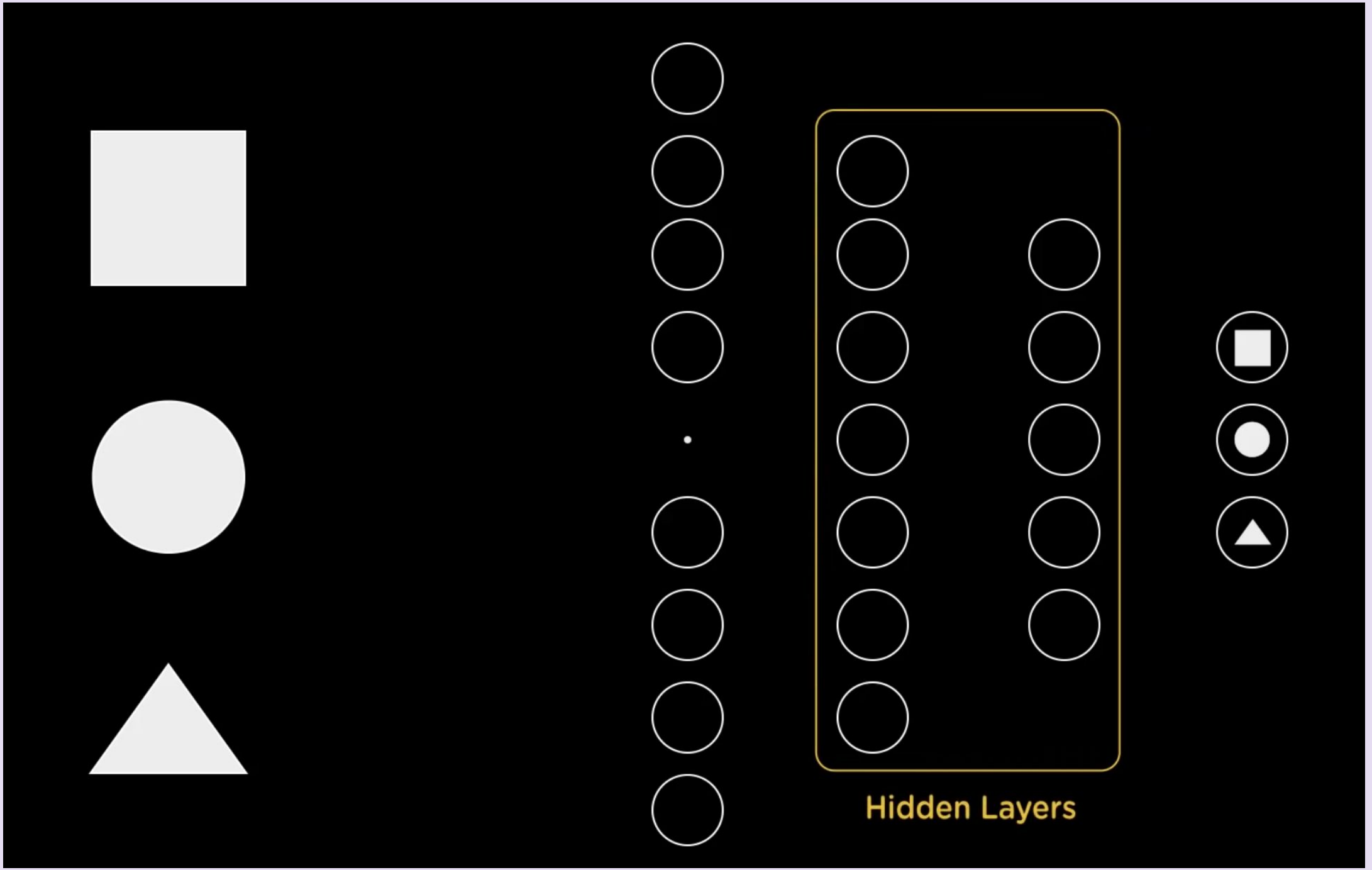


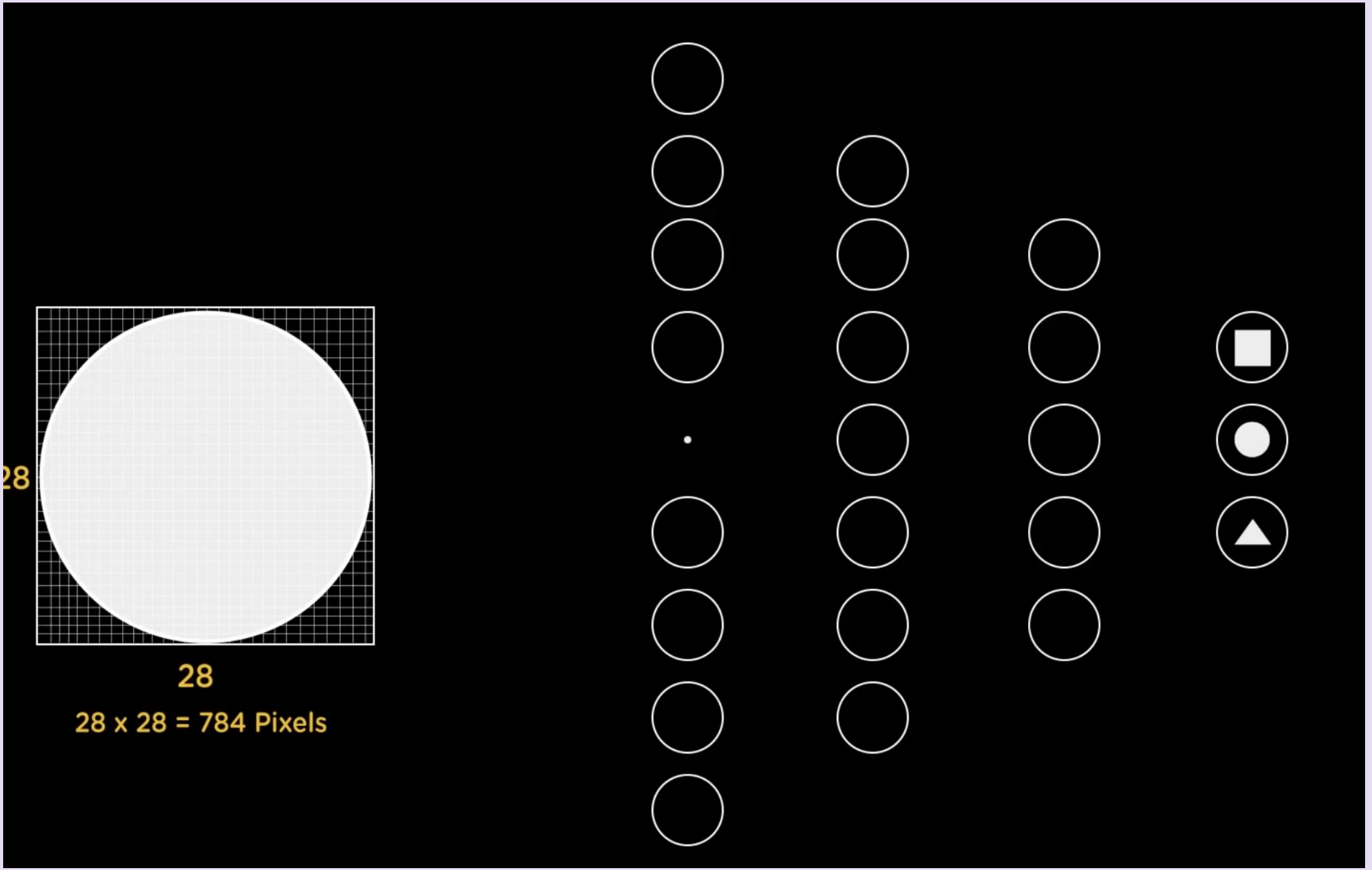


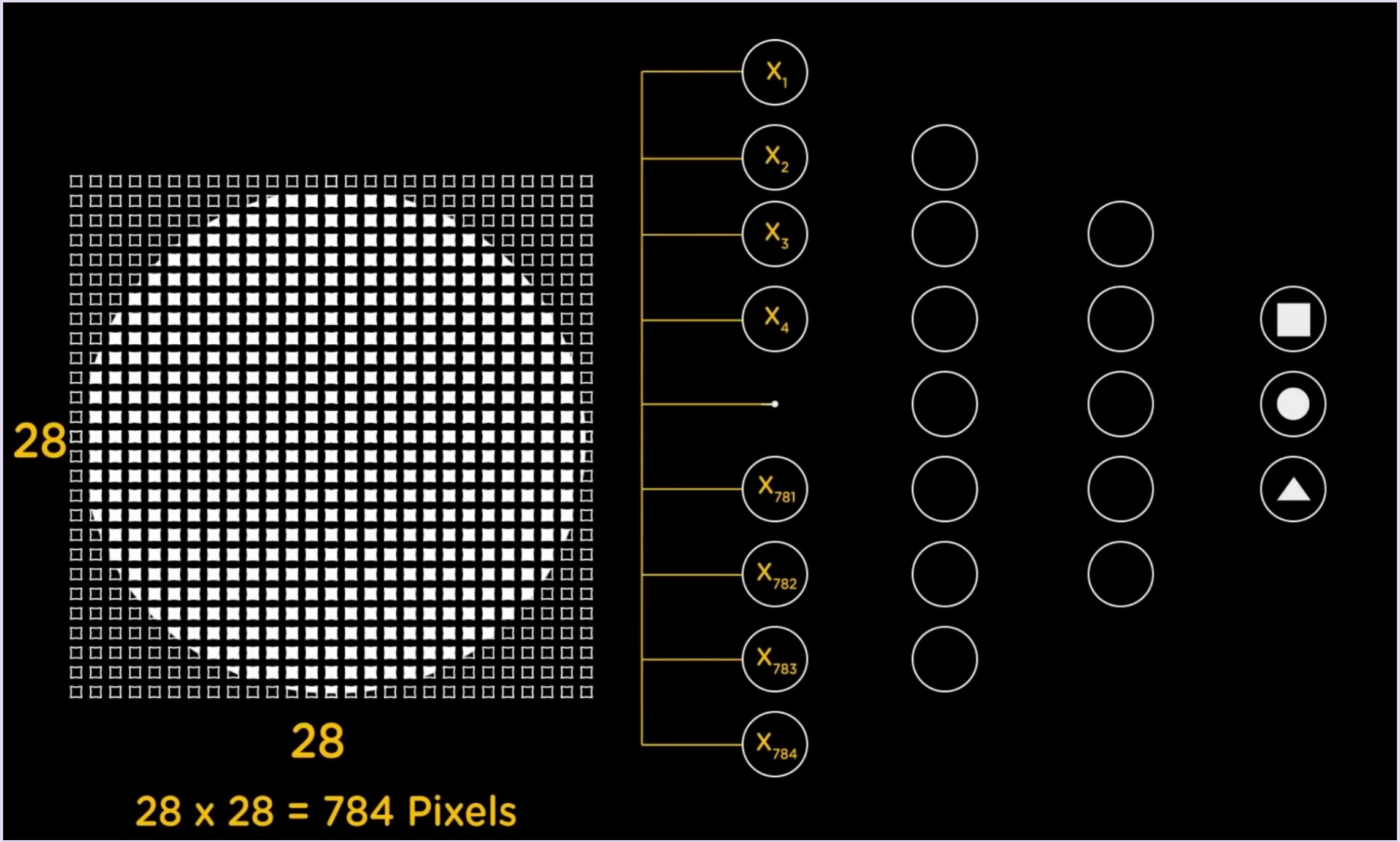


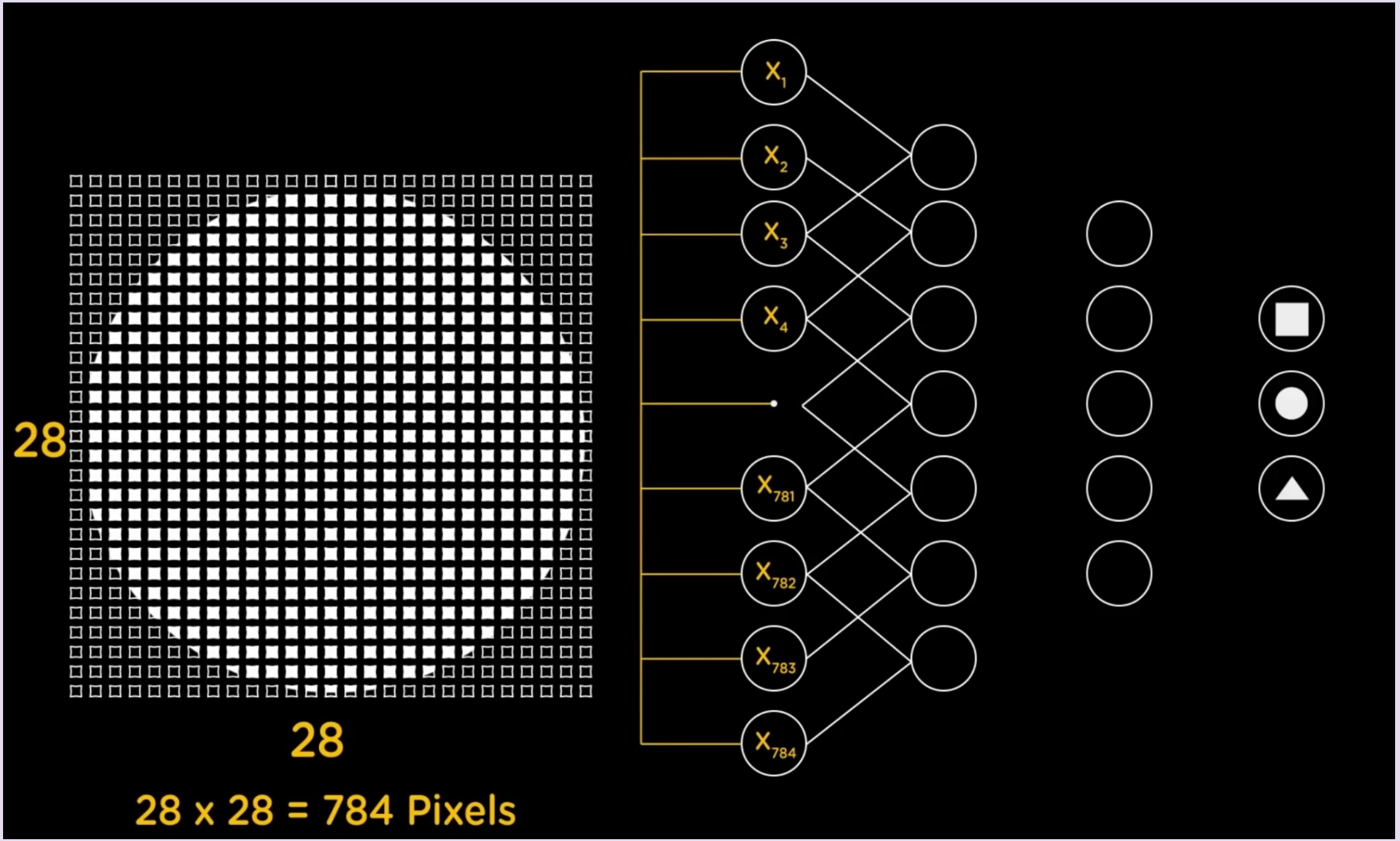

Input Layer

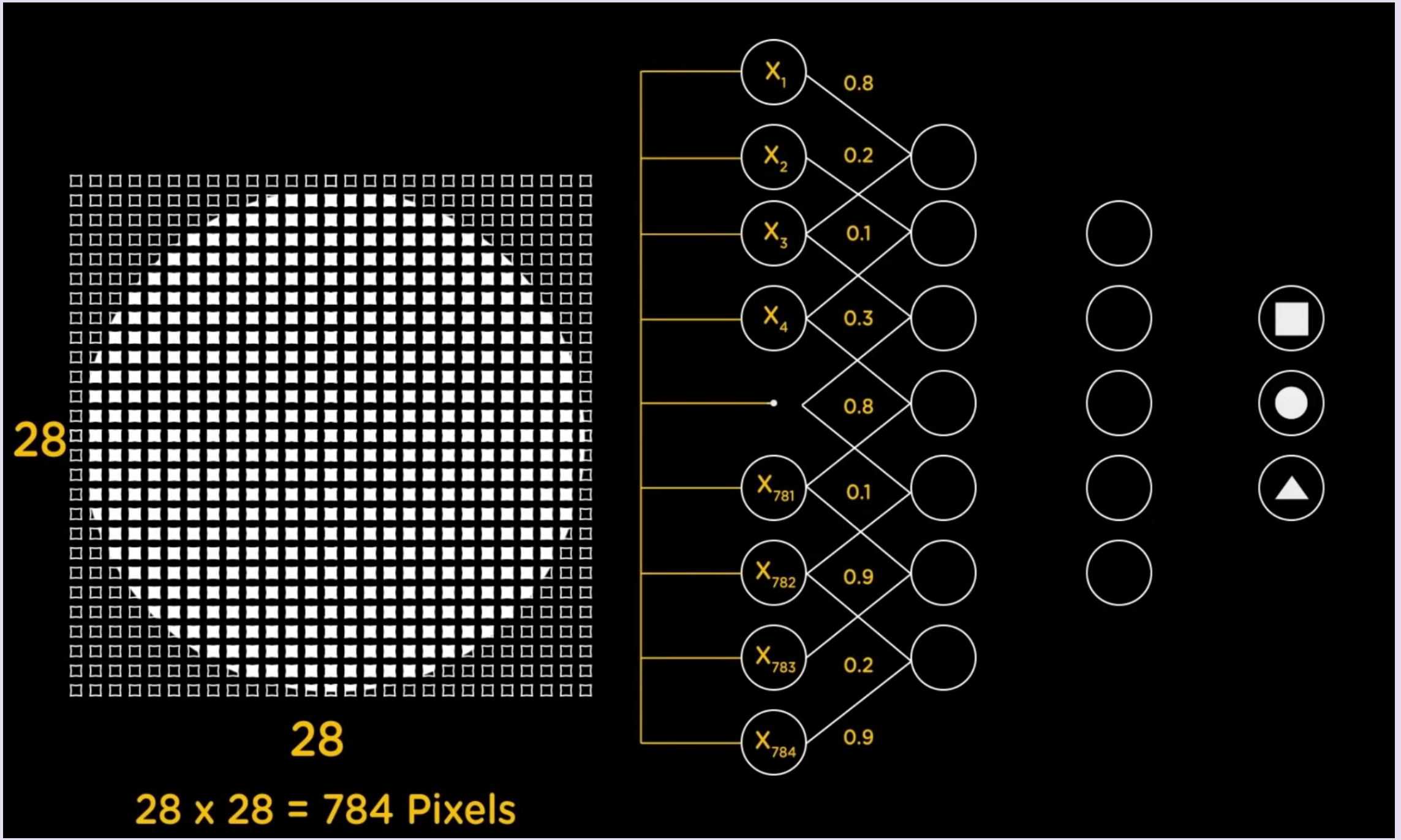

Output Layer

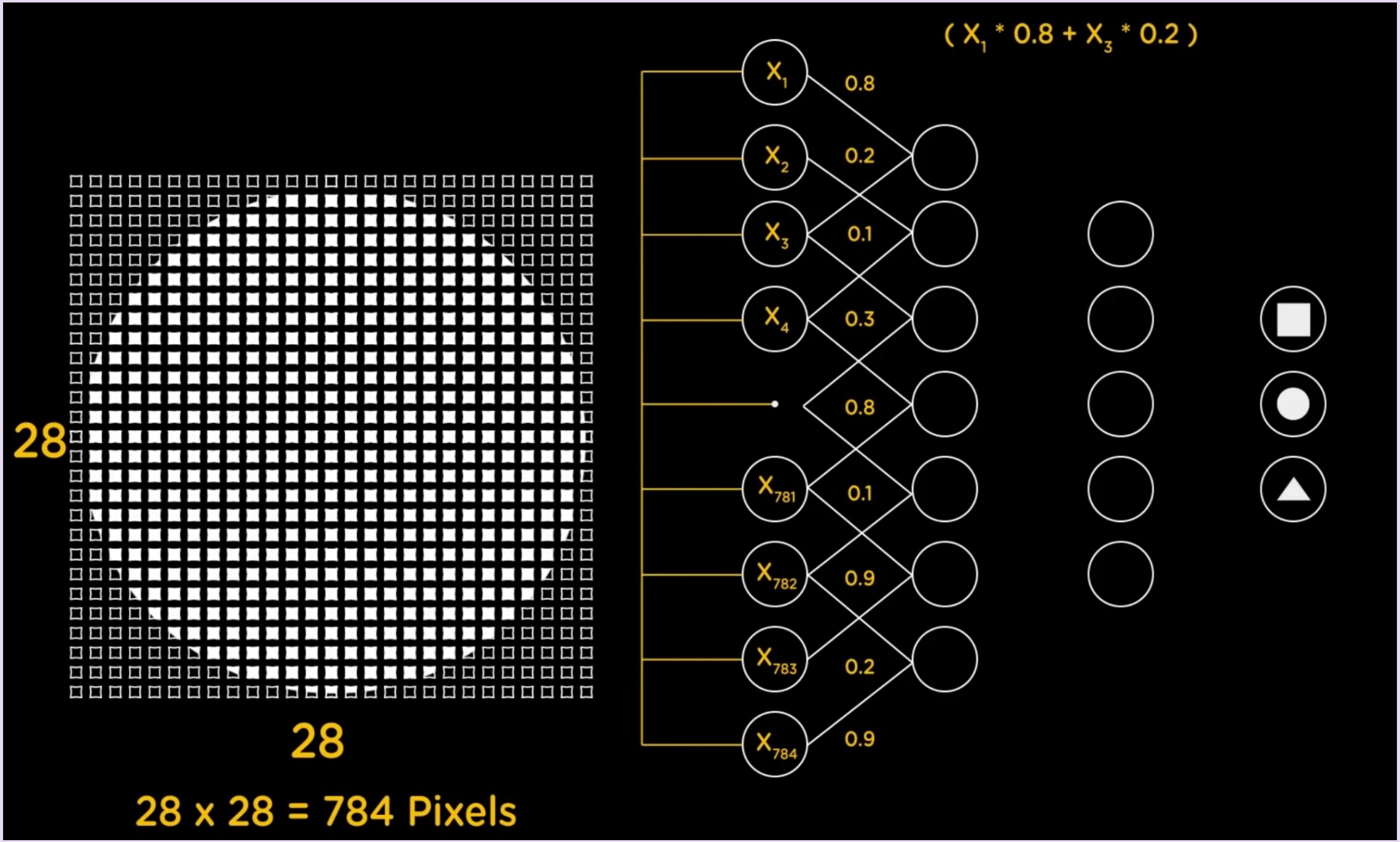

Hidden Layers

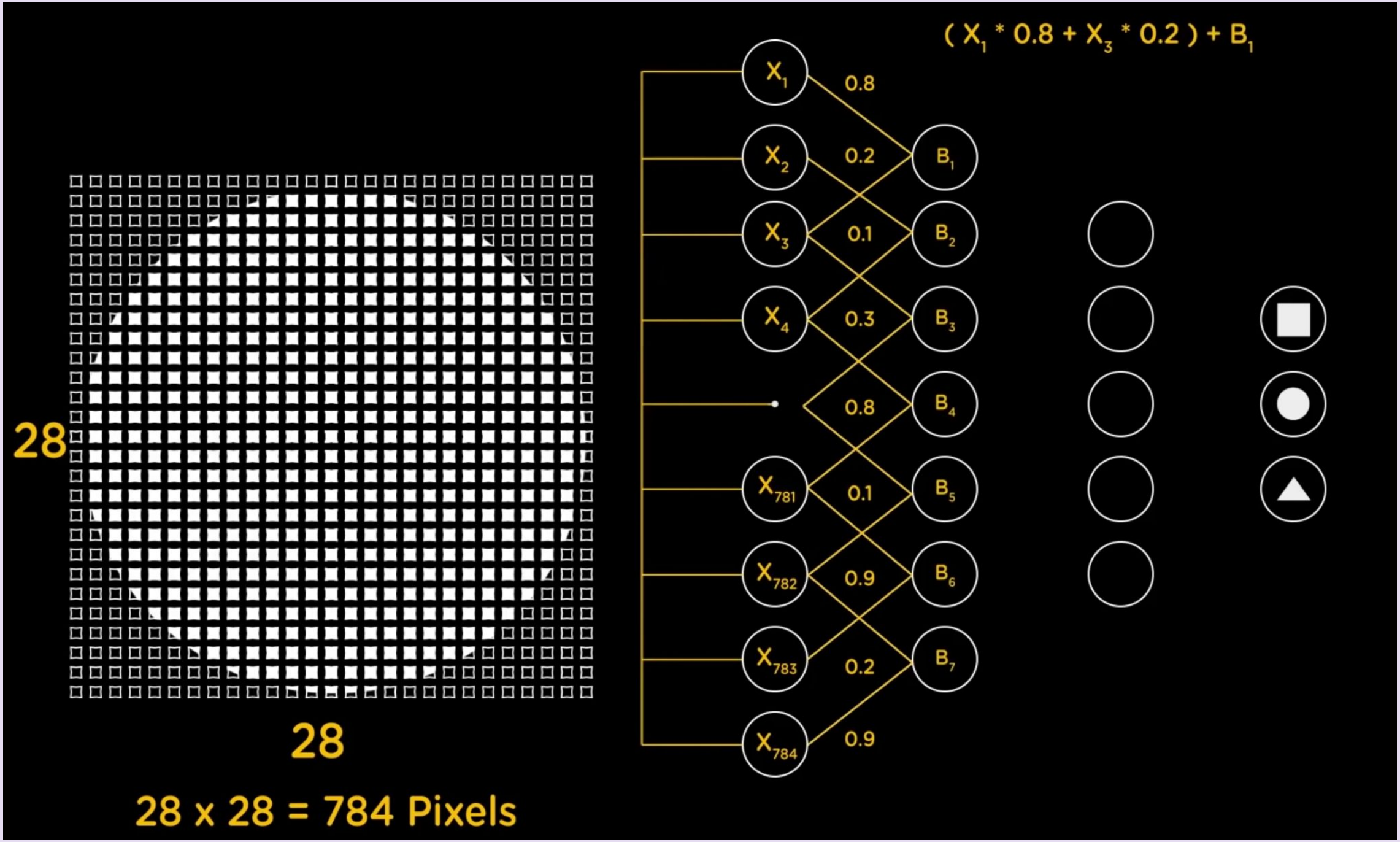

28
28
28 x 28 = 784 Pixels

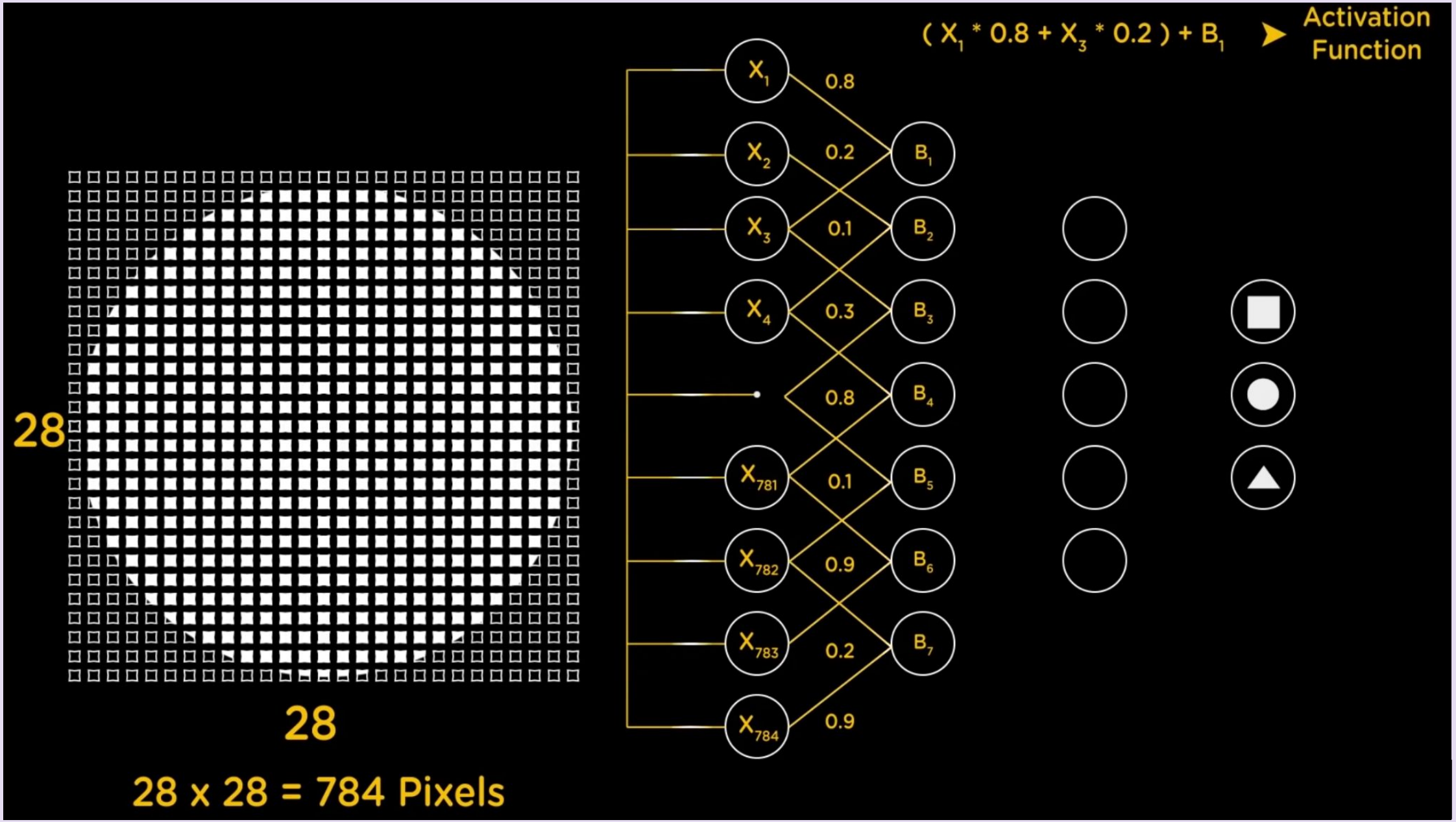

28
28
28 x 28 = 784 Pixels
X1
X2
X3
X4
X781
X782
X783
X784

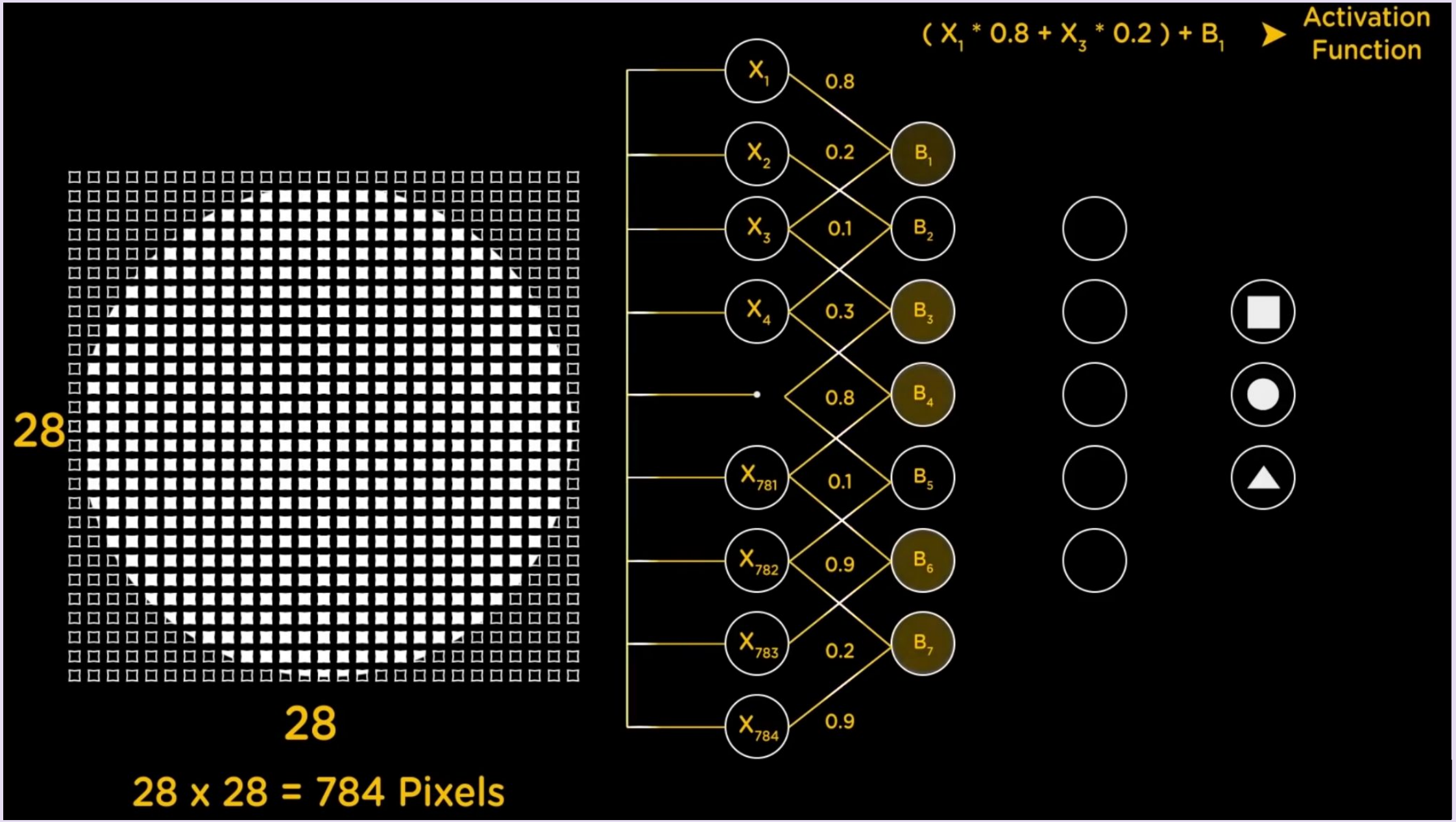

28
28
28 x 28 = 784 Pixels
X1
X2
X3
X4
X781
X782
X783
X784

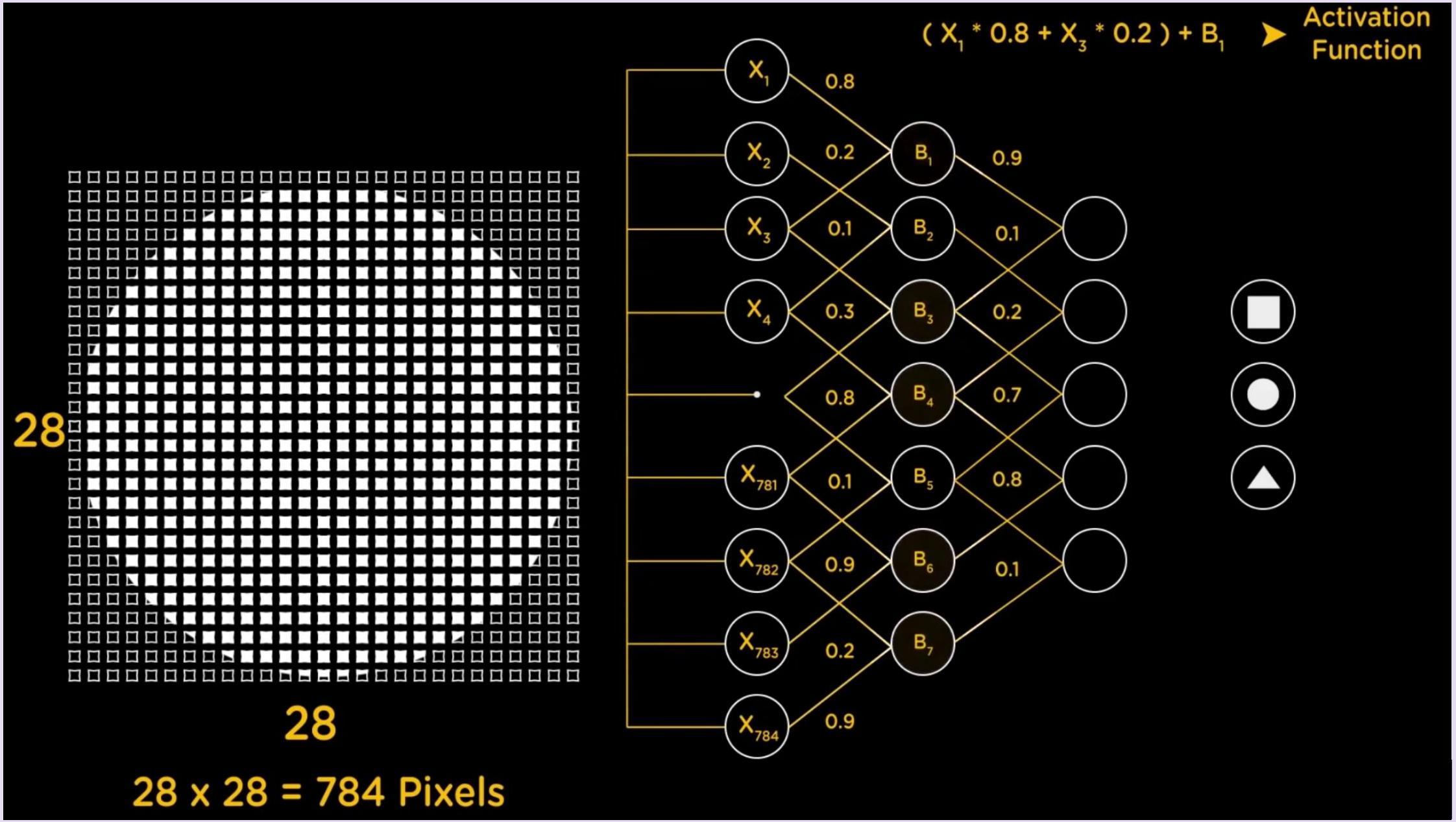

28
28
28 x 28 = 784 Pixels
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
0.9

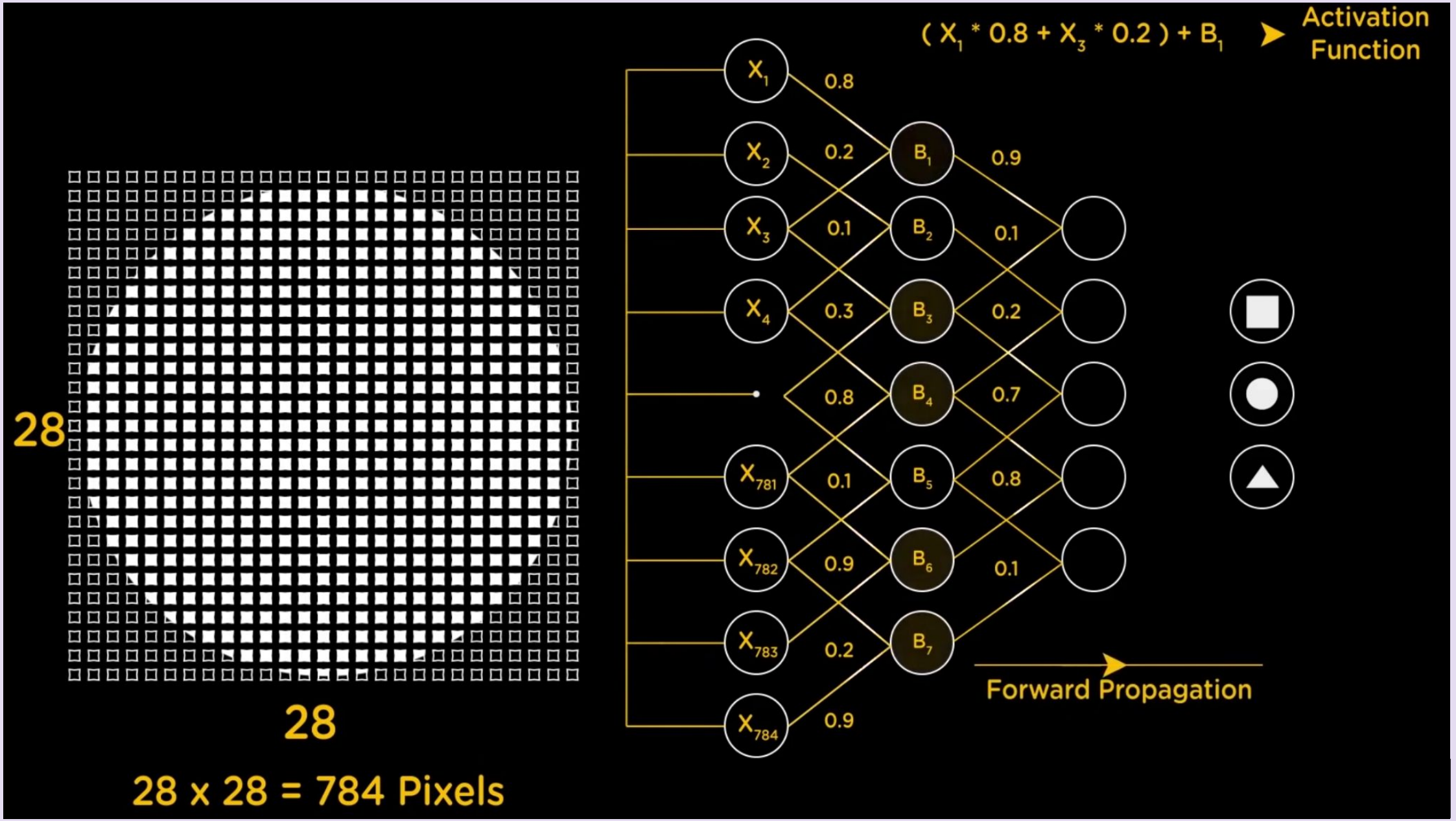

(X1 * 0.8 + X3 * 0.2)
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
0.9
28
28
28 x 28 = 784 Pixels

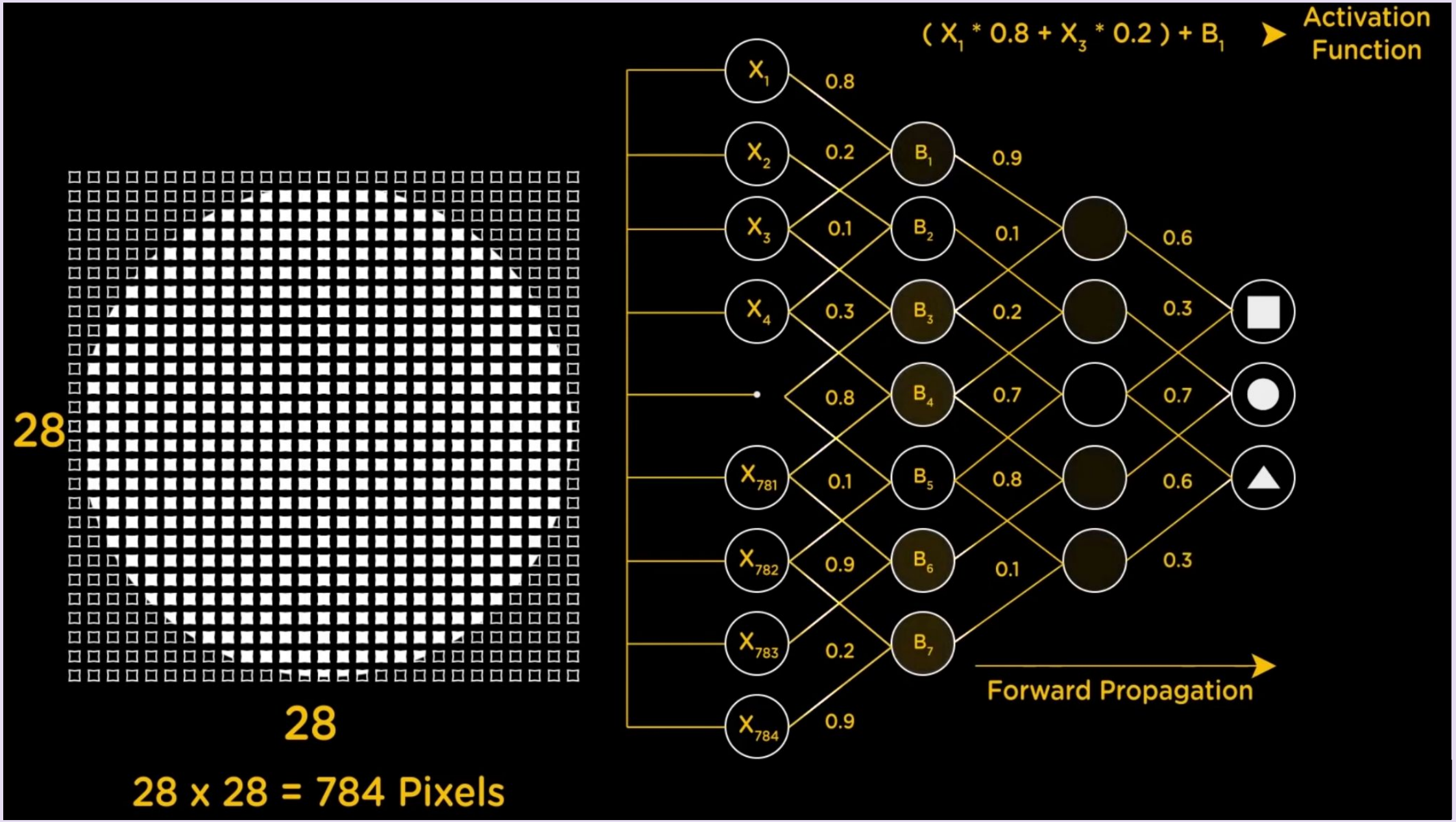

(X1 * 0.8 + X3 * 0.2) + B1
28
28
28 x 28 = 784 Pixels
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
0.9
B1
B2
B3
B4
B5
B6
B7

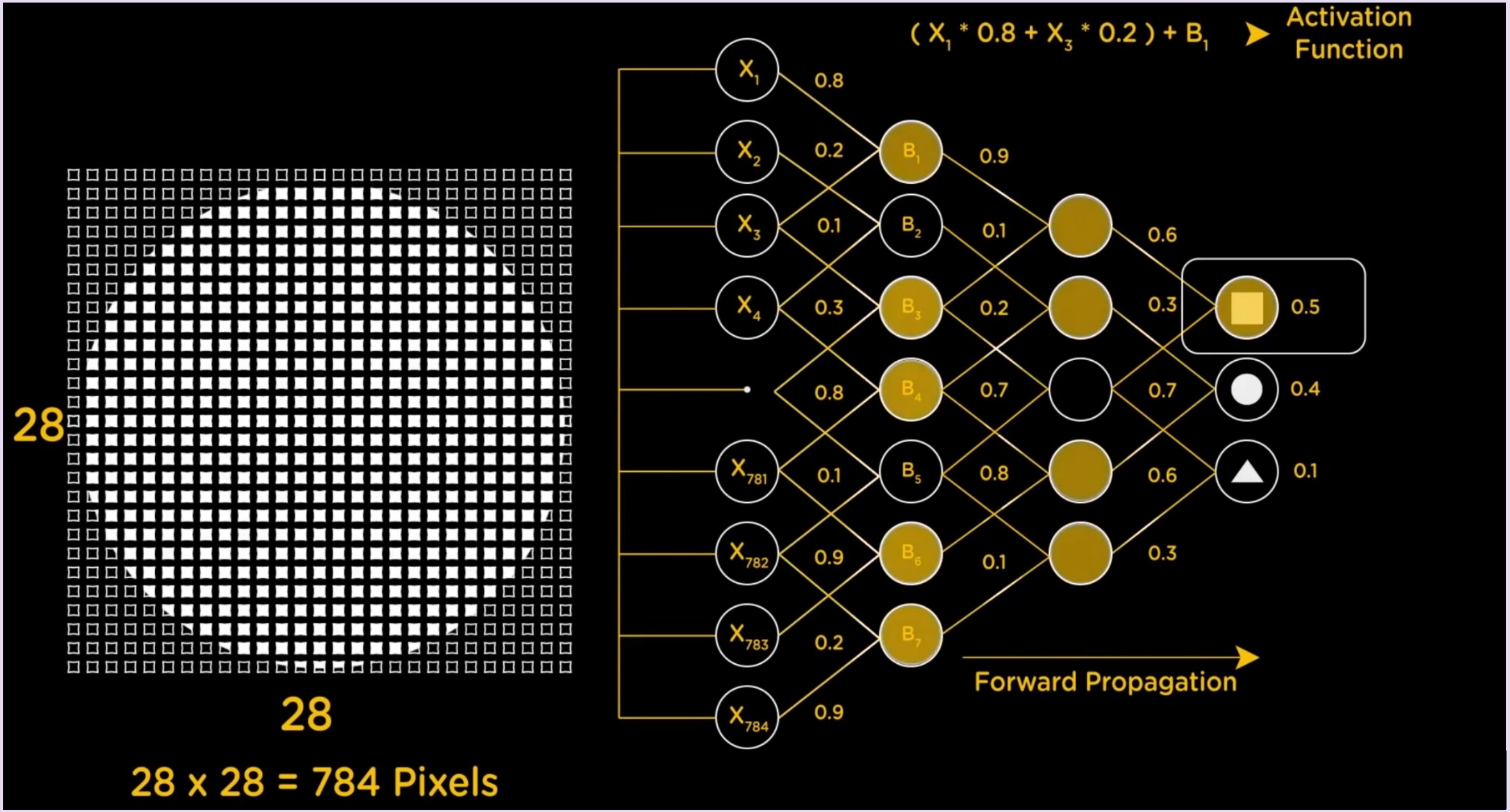

(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
0.9
B1
B2
B3
B4
B5
B6
B7
28
28
28 x 28 = 784 Pixels

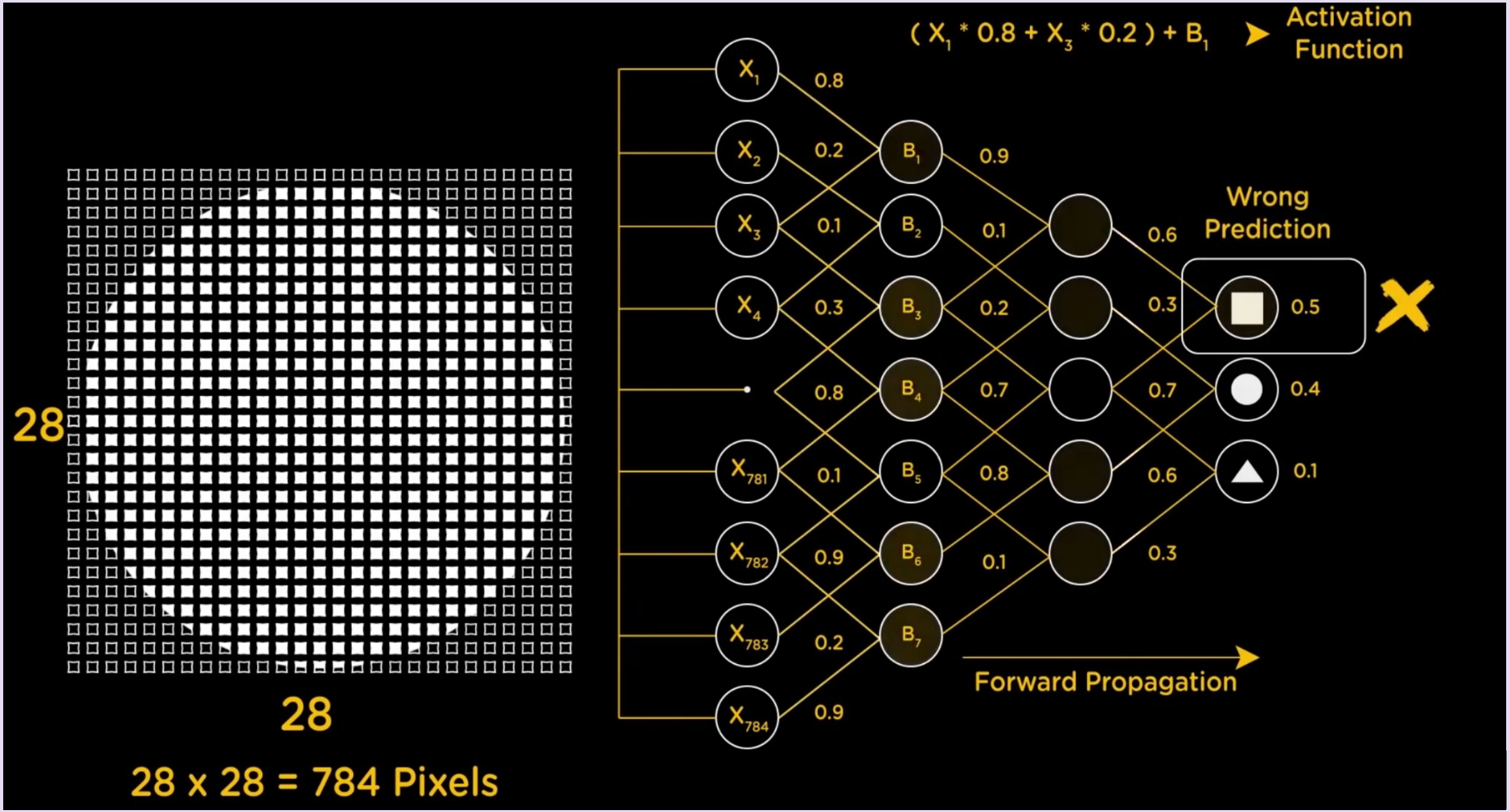

(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
0.9
B1
B2
B3
B4
B5
B6
B7
28
28
28 x 28 = 784 Pixels

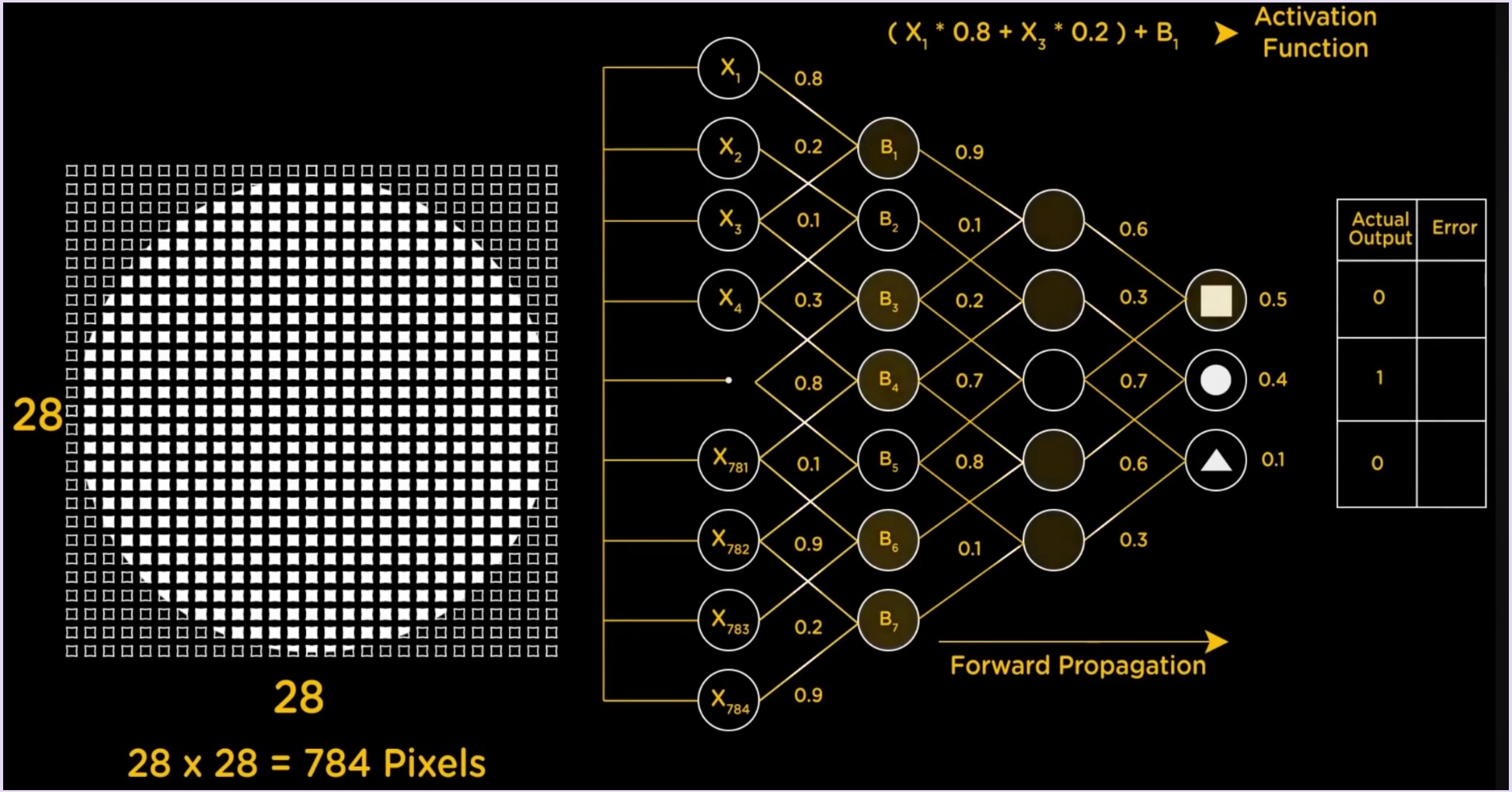

(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
B1
B2
B3
B4
B5
B6
B7
0.9
0.1
0.2
0.7
0.8
0.1
28
28
28 x 28 = 784 Pixels

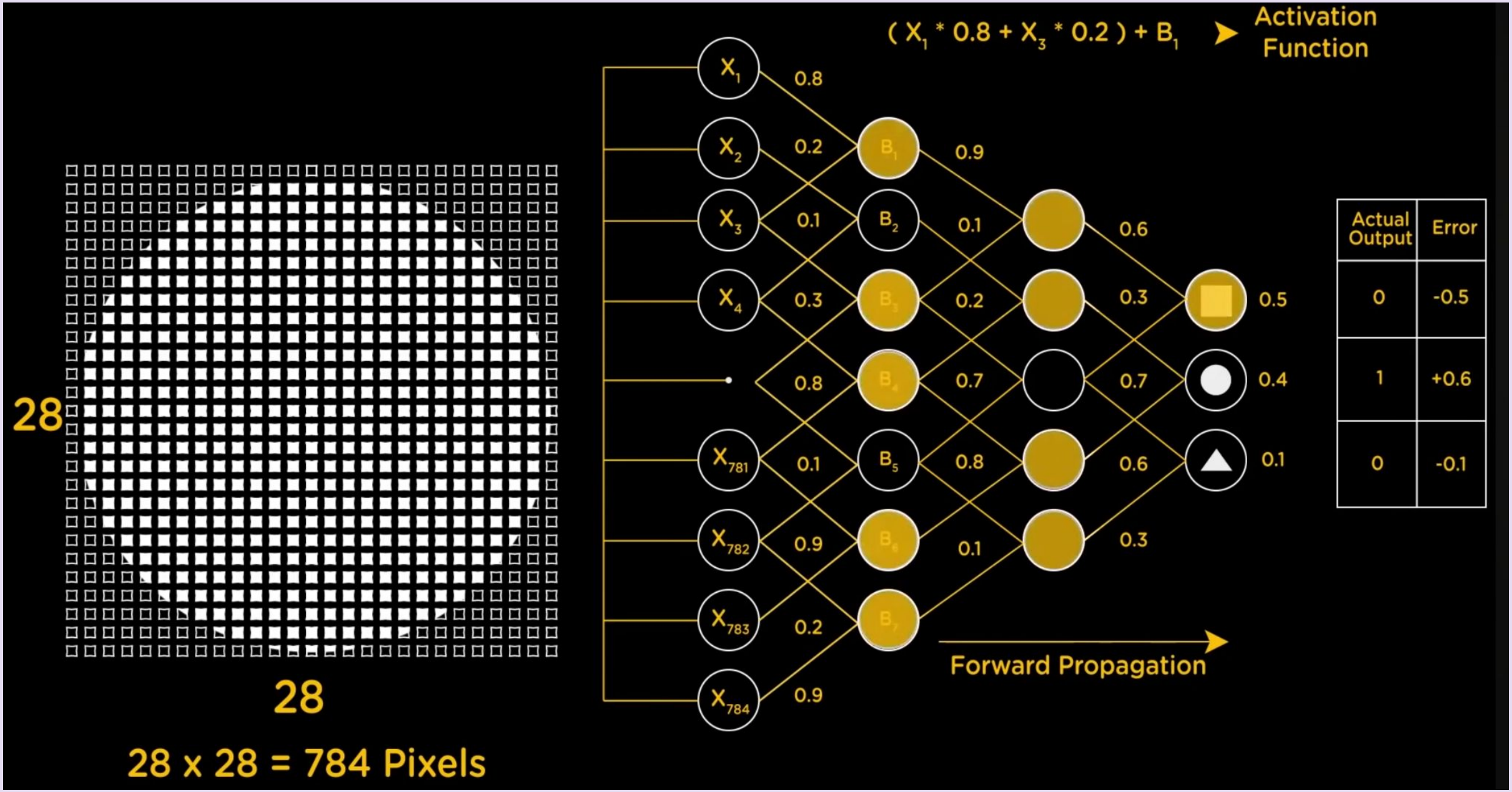

(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
B1
B2
B3
B4
B5
B6
B7
0.9
0.1
0.2
0.7
0.8
0.1
Forward Propagation
28
28
28 x 28 = 784 Pixels

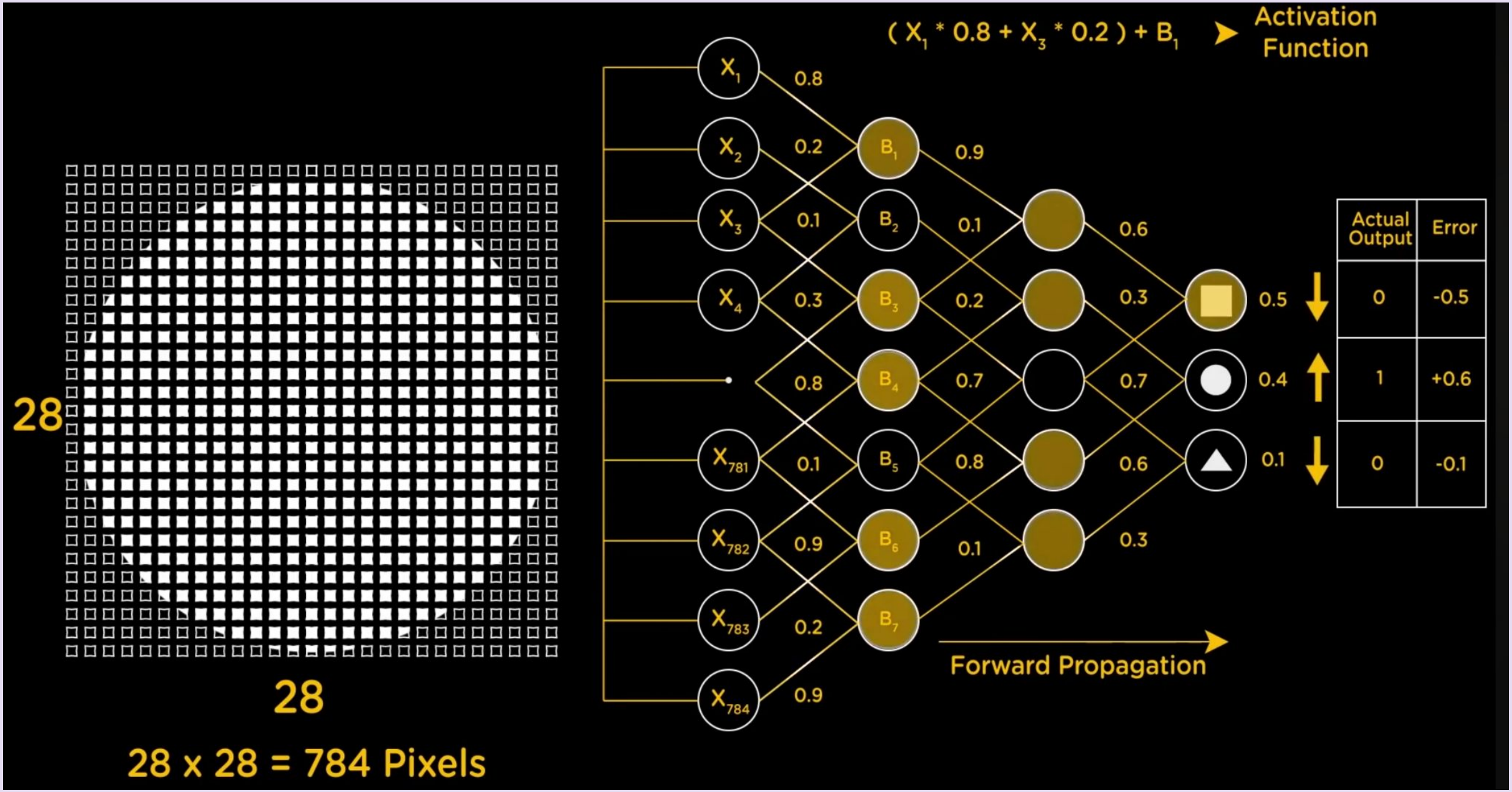

(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
28
28
28 x 28 = 784 Pixels
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
B1
B2
B3
B4
B5
B6
B7
0.9
0.1
0.2
0.7
0.8
0.1
0.6
0.3
0.7
0.6
0.3
Forward Propagation

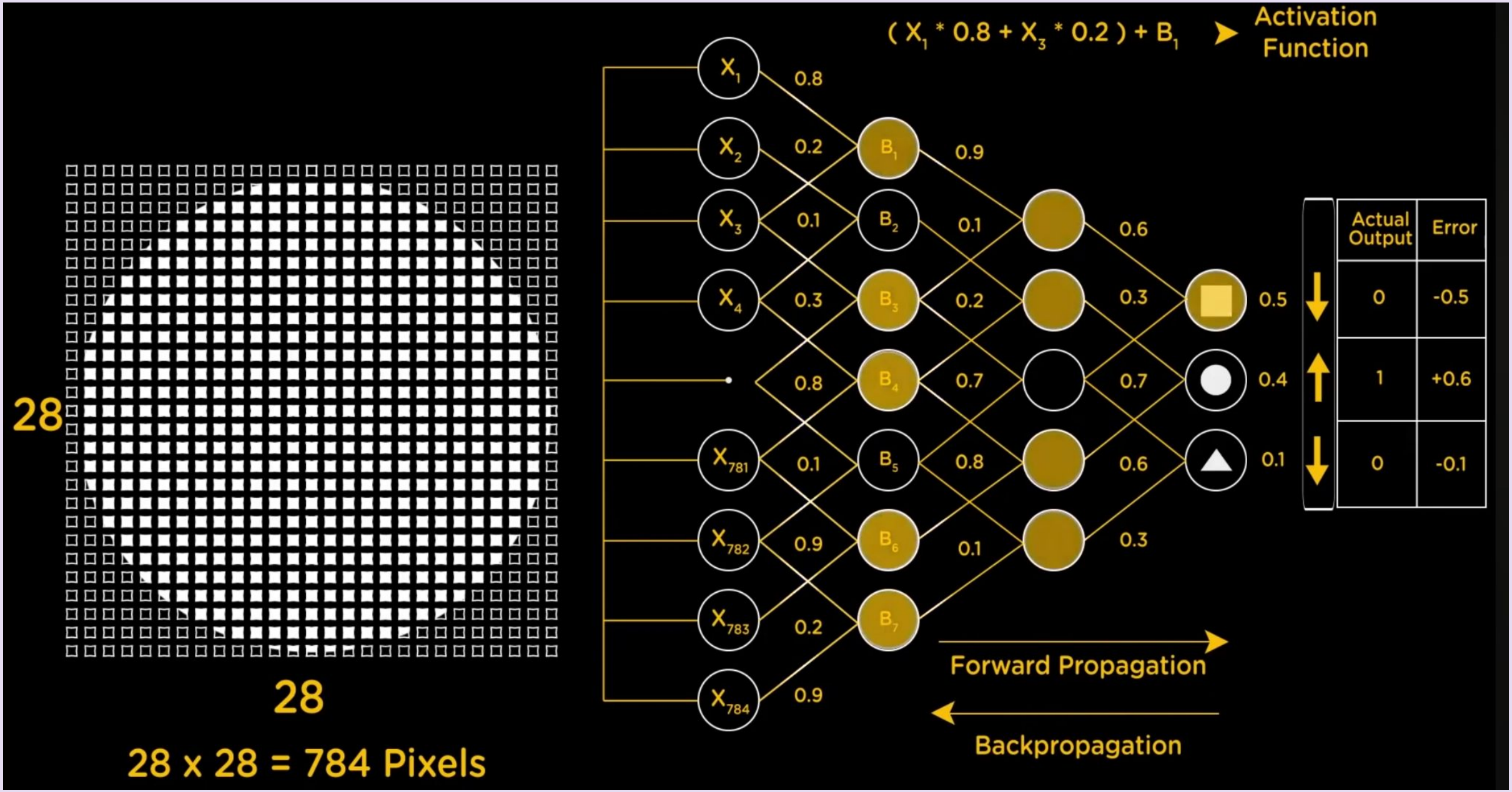

(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
B1
B2
B3
B4
B5
B6
B7
0.9
0.1
0.2
0.7
0.8
0.1
0.6
0.3
0.7
0.6
0.3
0.5
0.4
0.1
Forward Propagation
28
28
28 x 28 = 784 Pixels

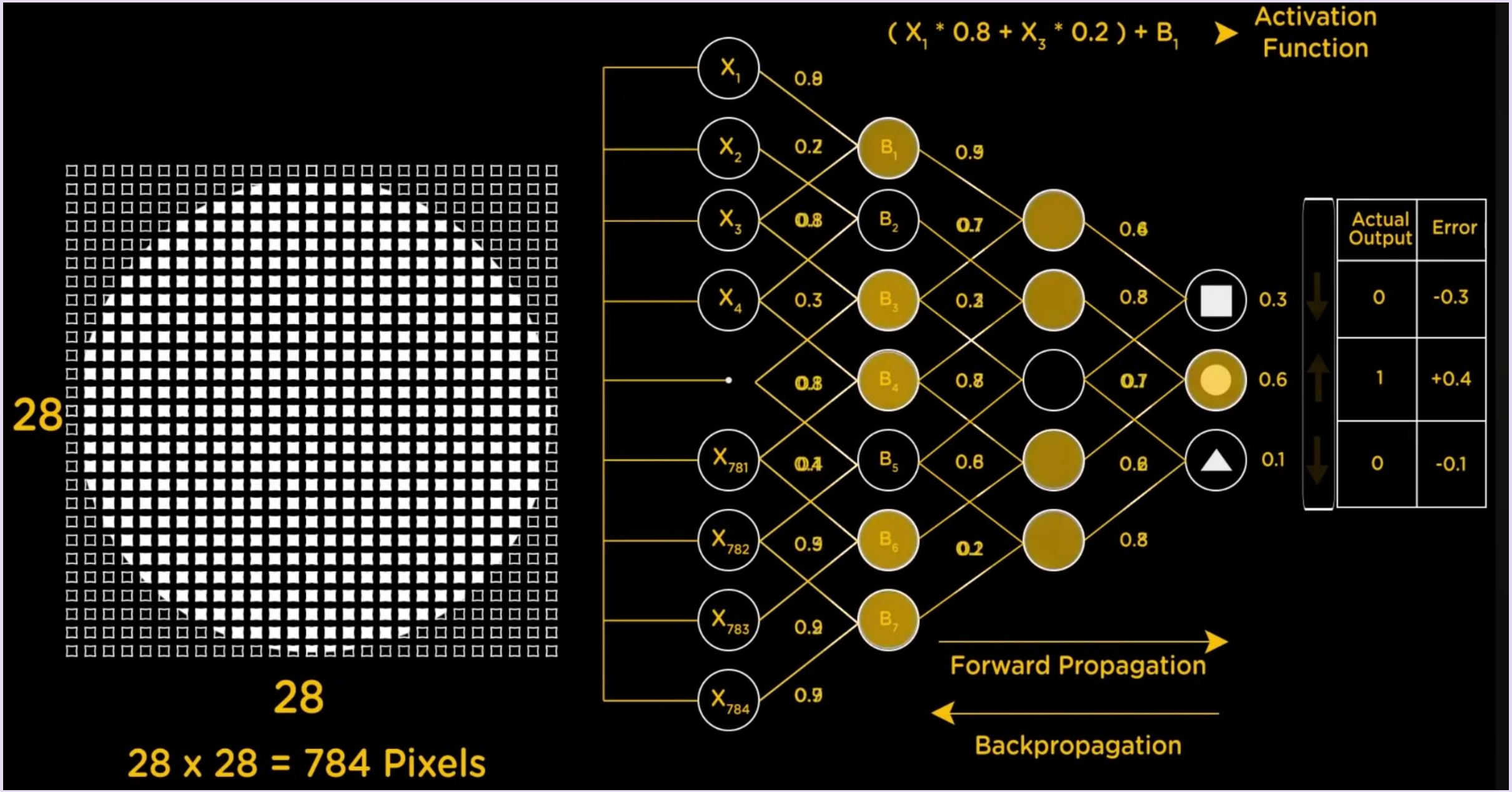

(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
B1
B2
B3
B4
B5
B6
B7
0.9
0.1
0.2
0.7
0.8
0.1
0.6
0.3
0.7
0.6
0.3
Wrong Prediction
0.5
0.4
0.1
Forward Propagation
28
28
28 x 28 = 784 Pixels

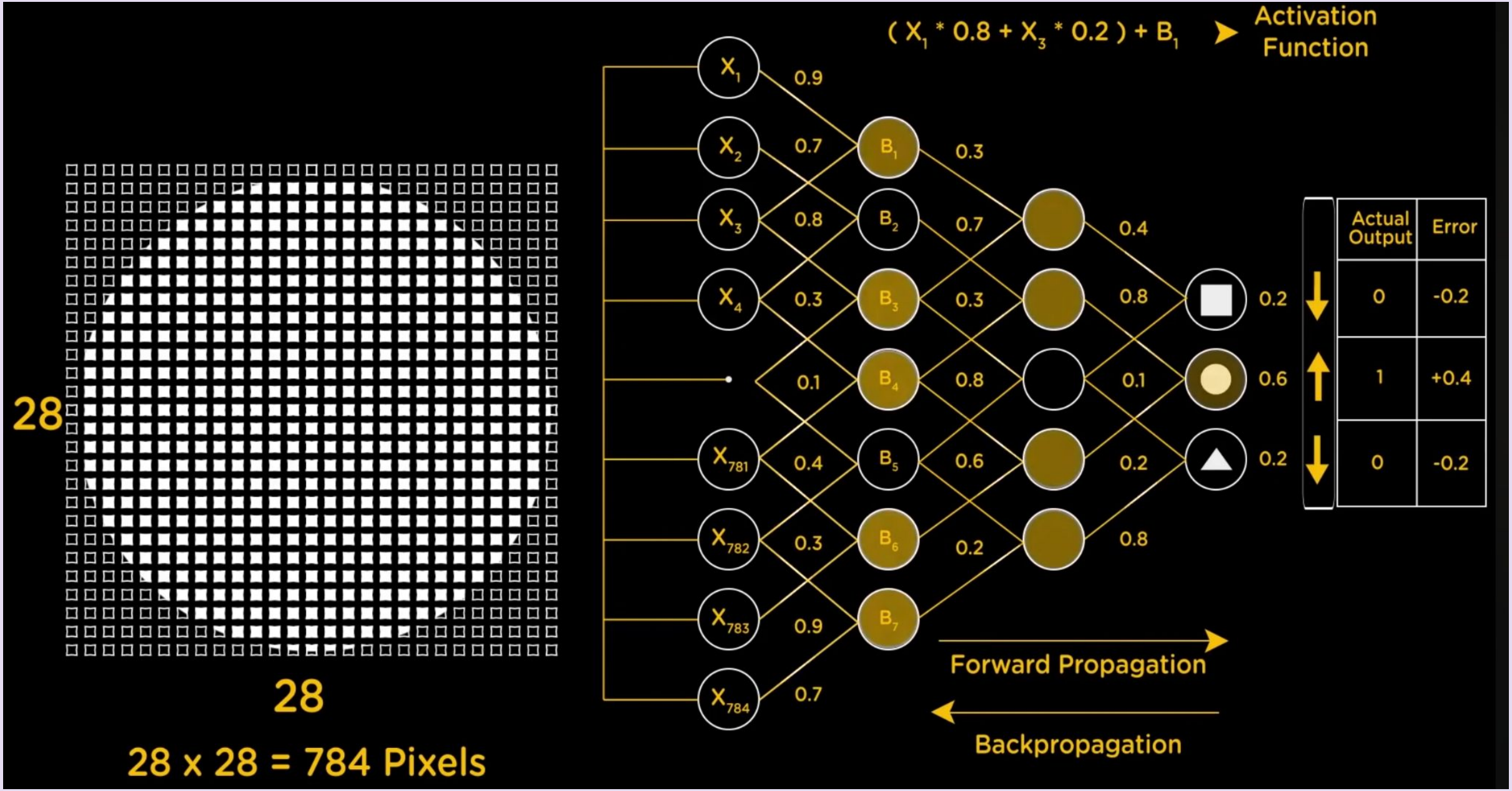

(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
28
28
28 x 28 = 784 Pixels
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
0.9
B1
B2
B3
B4
B5
B6
B7
0.6
0.3
0.7
0.8
0.1
0.2
0.1
0.3
0.7
0.6
0.3
0.5
0.4
0.1
Forward Propagation
Actual Output
Error
0
1
0

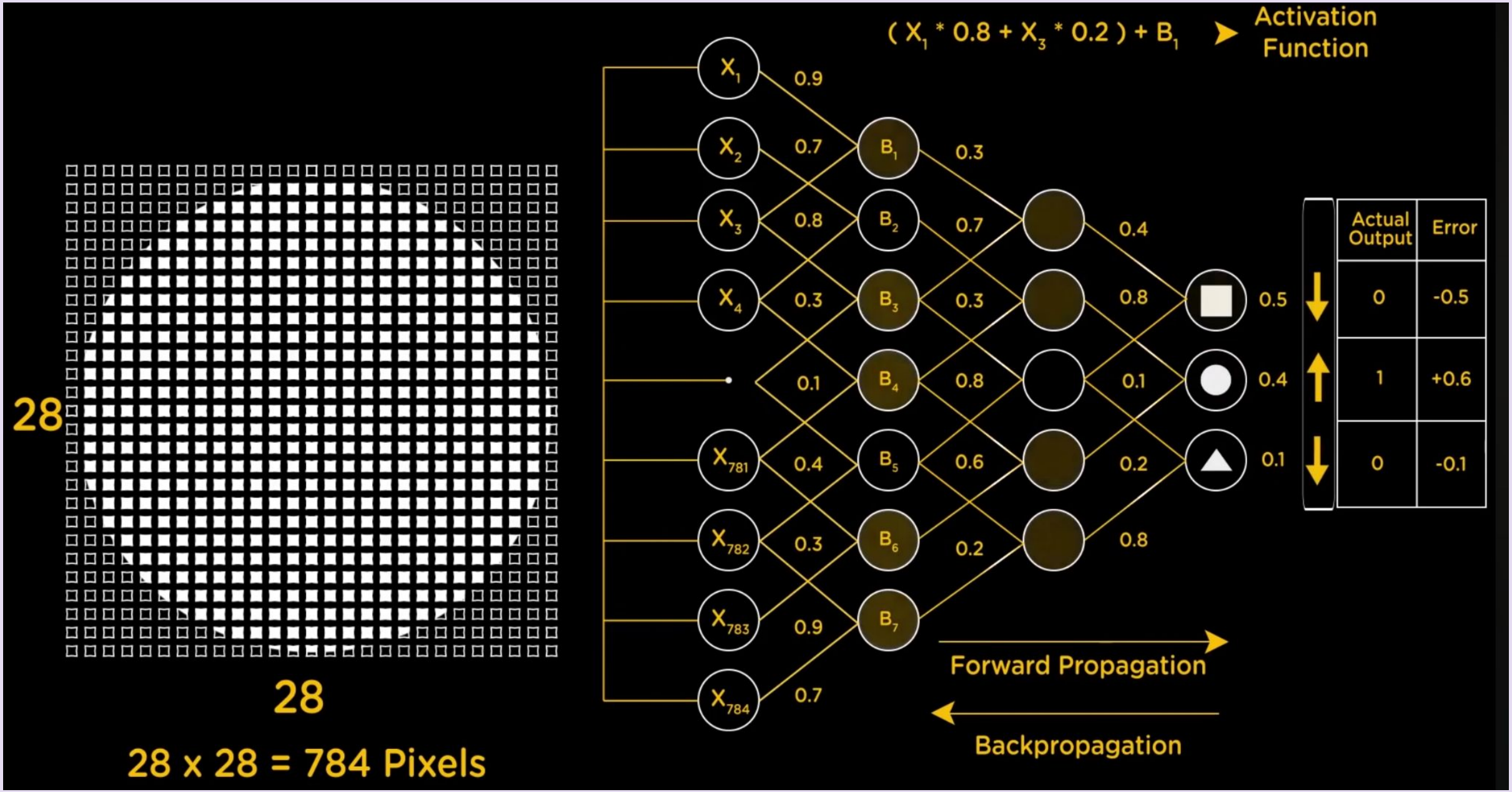

(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
28
28
28 x 28 = 784 Pixels
X1
X2
X3
X4
X781
X782
X783
X784
B1
B2
B3
B4
B5
B6
B7
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
0.9
0.1
0.2
0.7
0.8
0.1
0.6
0.3
0.7
0.6
0.3
0.5
0.4
0.1
Forward Propagation
Actual Output
Error
0
-0.5
1
+0.6
0
-0.1

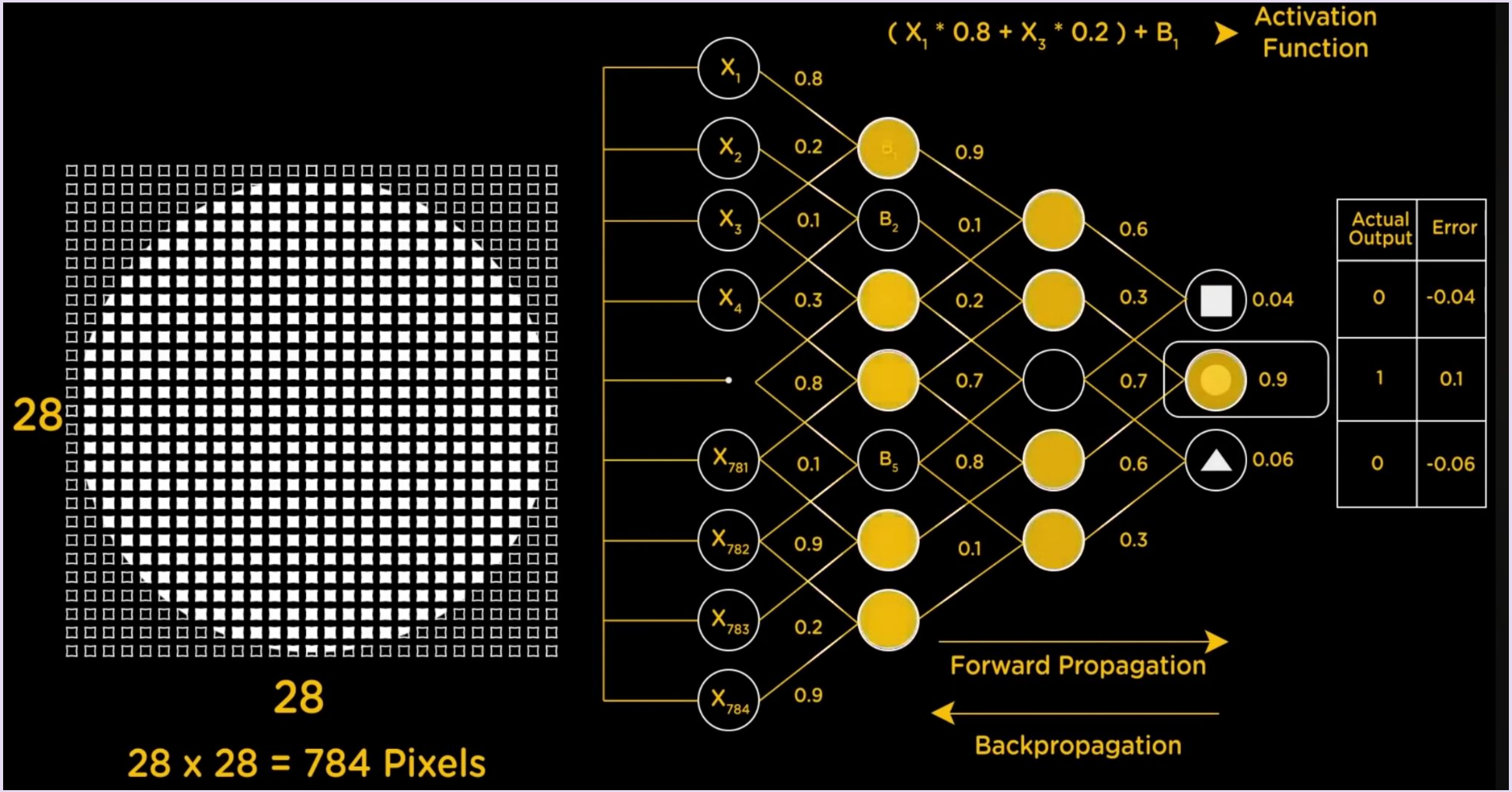

(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
28
28
28 x 28 = 784 Pixels
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
B1
B2
B3
B4
B5
B6
B7
0.9
0.1
0.2
0.7
0.8
0.1
0.6
0.3
0.7
0.6
0.3
0.5
0.4
0.1
Actual Output
Error
0
-0.5
1
+0.6
0
-0.1
Forward Propagation

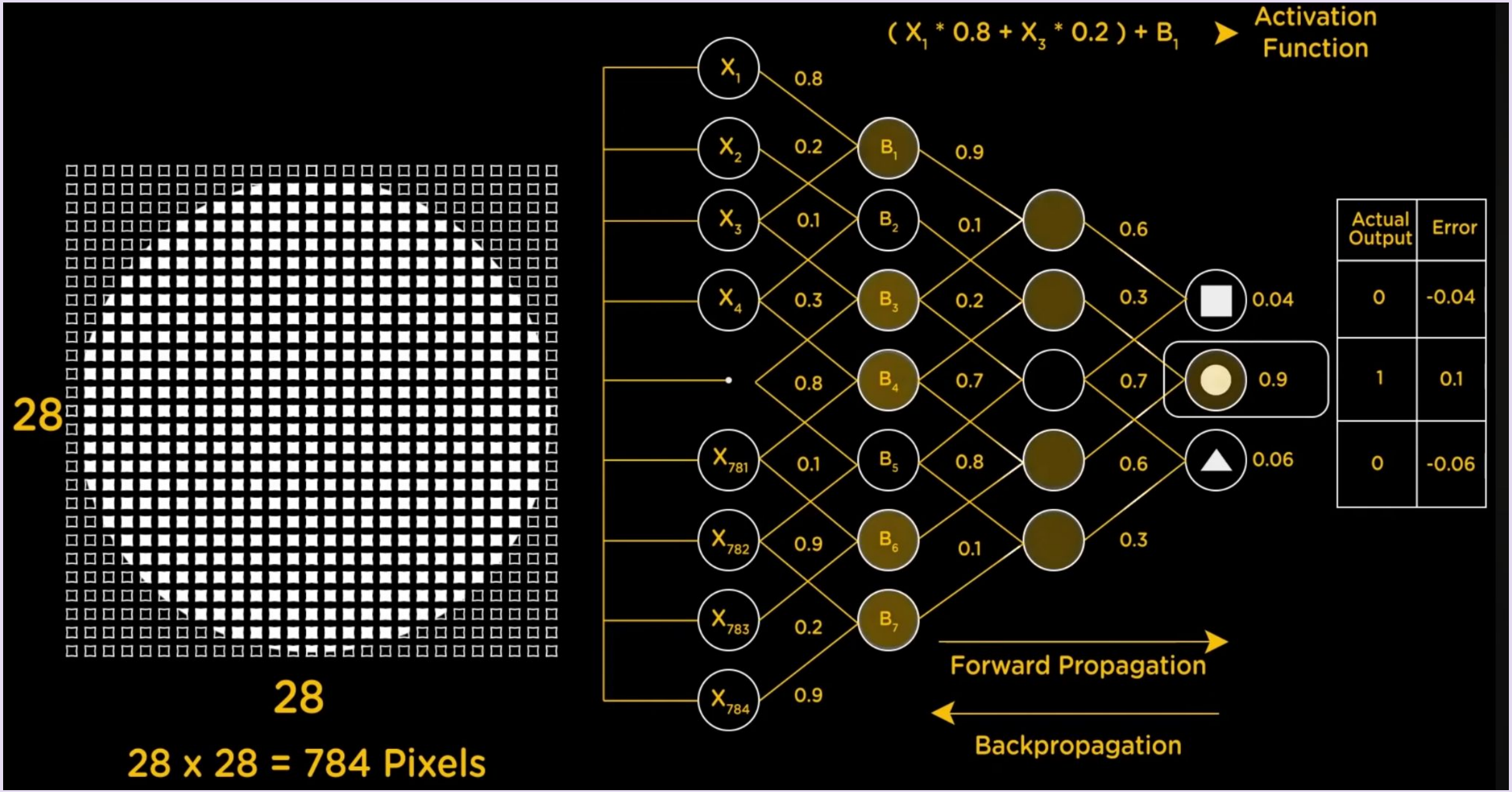

(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
28
28
28 x 28 = 784 Pixels
X1
X2
X3
X4
X781
X782
X783
X784
B1
B2
B3
B4
B5
B6
B7
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
0.9
0.1
0.2
0.3
0.7
0.8
0.1
0.6
0.3
0.7
0.6
0.3
0.5
0.4
0.1
Actual Output
Error
0
-0.5
1
+0.6
0
-0.1
Forward Propagation
Backpropagation


(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
28
28
28 x 28 = 784 Pixels
X1
X2
X3
X4
X781
X782
X783
X784
B1
B2
B3
B4
B5
B6
B7
0.3
0.6
0.1
Actual Output
Error
0
-0.3
1
+0.4
0
-0.1
Forward Propagation
Backpropagation

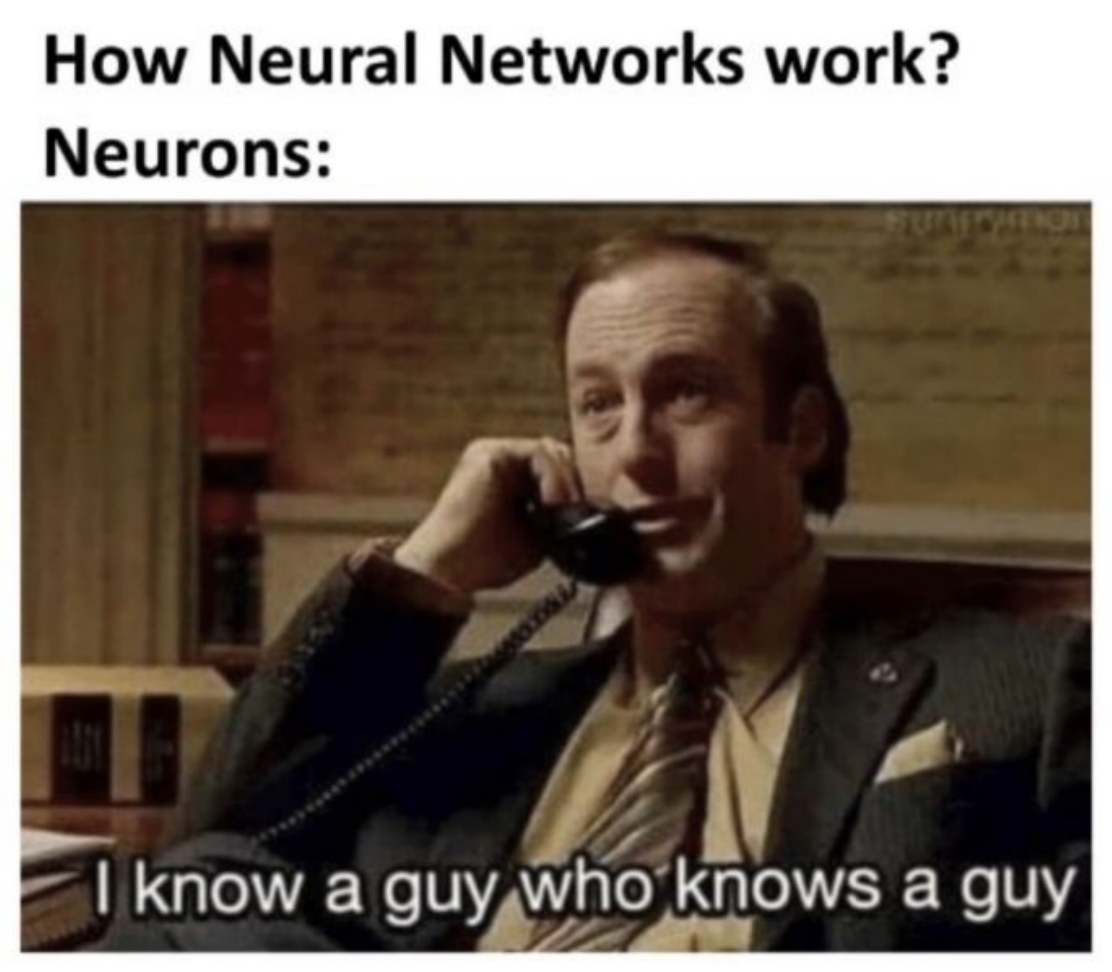

(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
28
28
28 x 28 = 784 Pixels
X1
X2
X3
X4
X781
X782
X783
X784
B1
B2
B3
B4
B5
B6
B7
0.9
0.7
0.8
0.3
0.1
0.4
0.3
0.9
0.7
0.3
0.7
0.3
0.8
0.6
0.2
0.4
0.8
0.1
0.2
0.8
0.2
0.6
0.2
Actual Output
Error
0
-0.2
1
+0.4
0
-0.2
Forward Propagation
Backpropagation


(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
28
28
28 x 28 = 784 Pixels
X1
X2
X3
X4
X781
X782
X783
X784
B1
B2
B3
B4
B5
B6
B7
0.9
0.7
0.8
0.3
0.1
0.4
0.3
0.9
0.7
0.3
0.7
0.3
0.8
0.6
0.2
0.4
0.8
0.1
0.2
0.8
0.5
0.4
0.1
Actual Output
Error
0
-0.5
1
+0.6
0
-0.1
Forward Propagation
Backpropagation

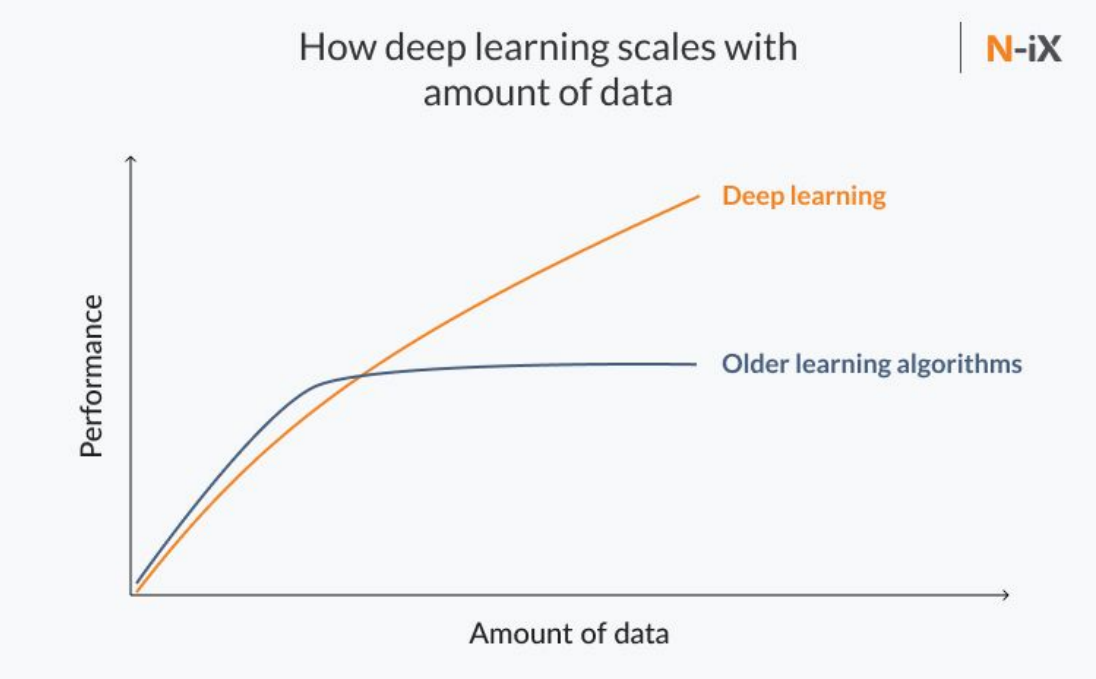

(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
B1
B2
B5
0.9
0.1
0.2
0.7
0.8
0.1
0.6
0.3
0.7
0.6
0.3
0.04
0.9
0.06
Actual Output
Error
0
-0.04
1
0.1
0
-0.06
Forward Propagation
Backpropagation
28
28
28 x 28 = 784 Pixels


(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
28
28
28 x 28 = 784 Pixels
X1
X2
X3
X4
X781
X782
X783
X784
B1
B2
B3
B4
B5
B6
B7
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
0.9
0.1
0.2
0.7
0.8
0.1
0.6
0.3
0.7
0.6
0.3
0.04
0.9
0.06
Actual Output
Error
0
-0.04
1
0.1
0
-0.06
Forward Propagation
Backpropagation

So how do actually NN work?



Why Neural Networks?



- Often, traditional ML models are conceptually simpler
- The performance of NN increases with the amount of data
- NN models try to learn high-level features from data in an incremental manner.
- They can learn and model the relationships between inputs and outputs that are nonlinear and complex

"The analogy to deep learning is that the rocket engine is the deep learning models and the fuel is the huge amounts of data we can feed to these algorithms."

Let's play

Playground:

https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.64300&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

Description:

https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks/playground-exercises