DATE : 25.03.2024

DT/NT : NT

LESSON : DEEP LEARNING

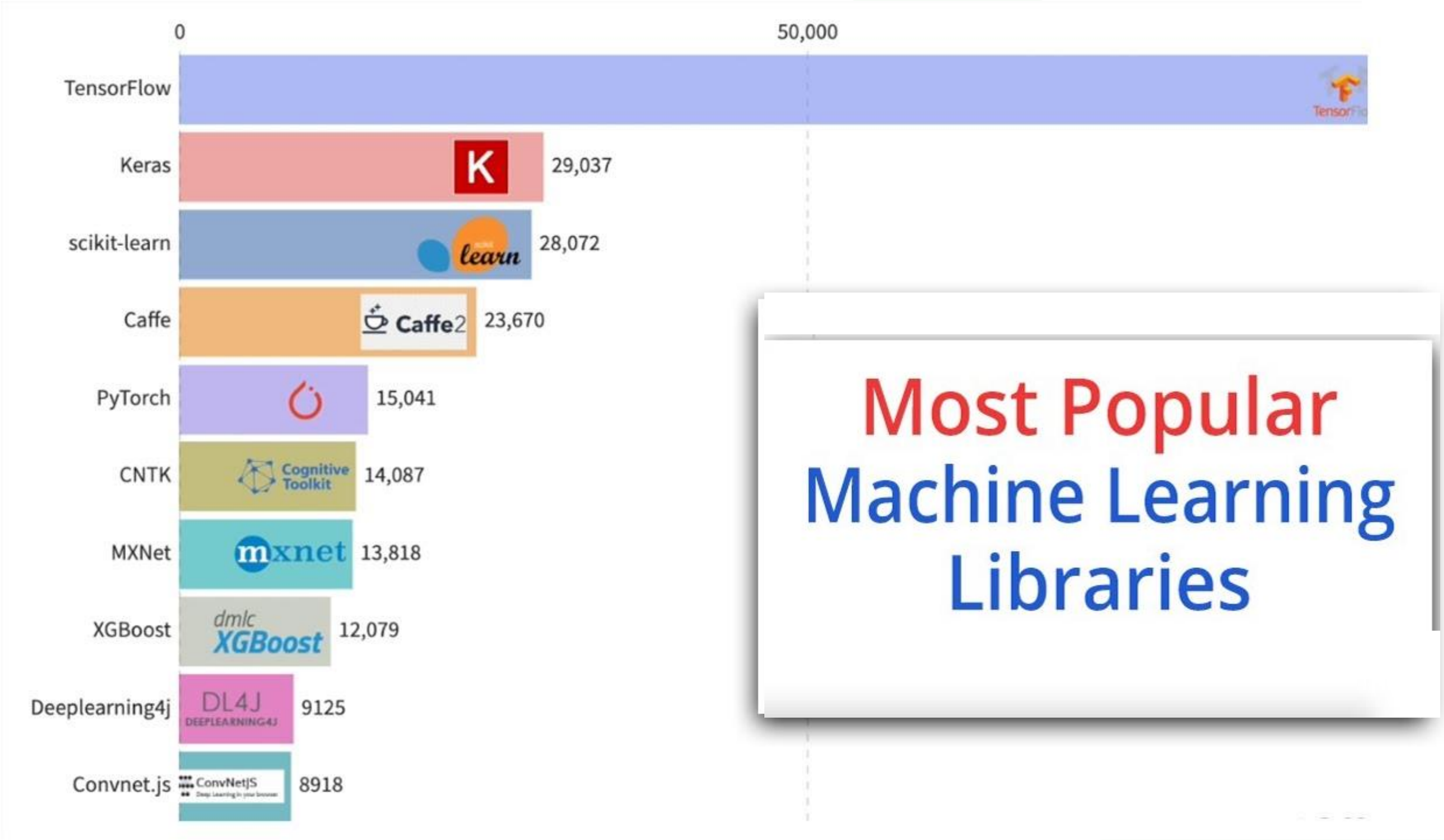
SUBJECT : BACKPROPAGATİON

BATCH : B223

TECHPRO EDUCATION

# MOST POPULAR
# DEEP LEARNING LIBRARIES&PLATFORMS

# DEEP LEARNING KÜTÜPHANELERİ



# DEEP LEARNING KÜTÜPHANELERİ

## Keras

Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow. It is designed to enable fast experimentation with **deep neural networks**.

## TensorFlow

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library that is used for **machine learning** applications like neural networks.

## PyTorch

PyTorch is an open source **machine learning** library for Python, based on Torch. It is used for applications such as **natural language processing** and was developed by Facebook's AI research group.

## Theano

## Caffe2

Caffe is a deep learning framework, originally developed at University of California, Berkeley. It is open source, under a BSD license. It is written in C++, with a Python interface.

# DEEP LEARNING KÜTÜPHANELERİ

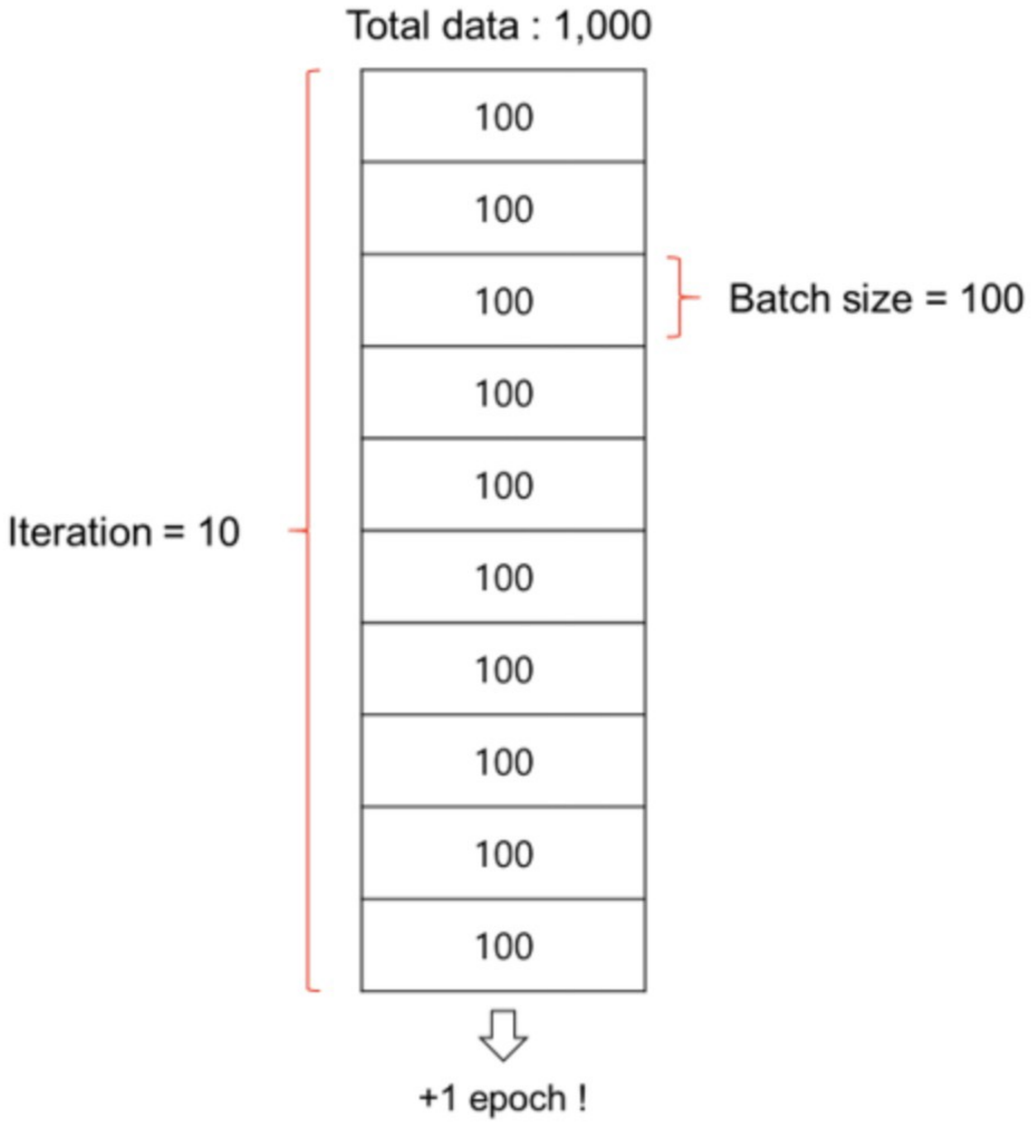
| | Languages | Tutorials and training materials | CNN modeling capability | RNN modeling capability | Architecture: easy-to-use and modular front end | Speed | Multiple GPU support | Keras compatible |
|---|---|---|---|---|---|---|---|---|
| Theano | Python, C++ | ++ | ++ | ++ | + | ++ | + | + |
| Tensor-Flow | Python | +++ | +++ | ++ | +++ | ++ | ++ | + |
| Torch | Lua, Python (new) | + | +++ | ++ | ++ | +++ | ++ | |
| Caffe | C++ | + | ++ | | + | + | + | |
| MXNet | R, Python, Julia, Scala | ++ | ++ | + | ++ | ++ | +++ | |
| Neon | Python | + | ++ | + | + | ++ | + | |
| CNTK | C++ | + | + | +++ | + | ++ | + | |

# EPOCH, BATCHSIZE

# BATCHSIZE

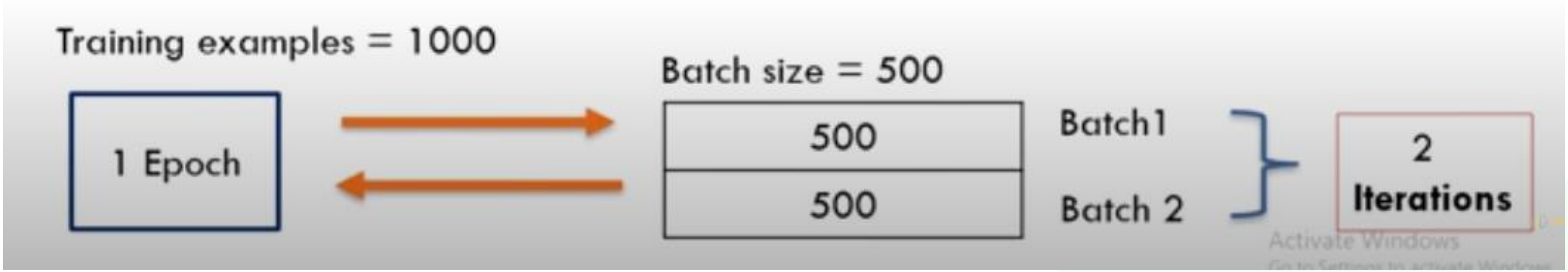
## BATCHSIZE

Batch size is a term used in machine learning and refers to **the number of training examples utilized in one iteration**.



# DIFFERENCE BETWEEN BATCH AND EPOCH

Example: if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.



# EPOCH

## EPOCH

```
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.***

# EPOCH

```
model.fit(x = X_train, y = y_train, batch_size = 32, epochs = 300)
```

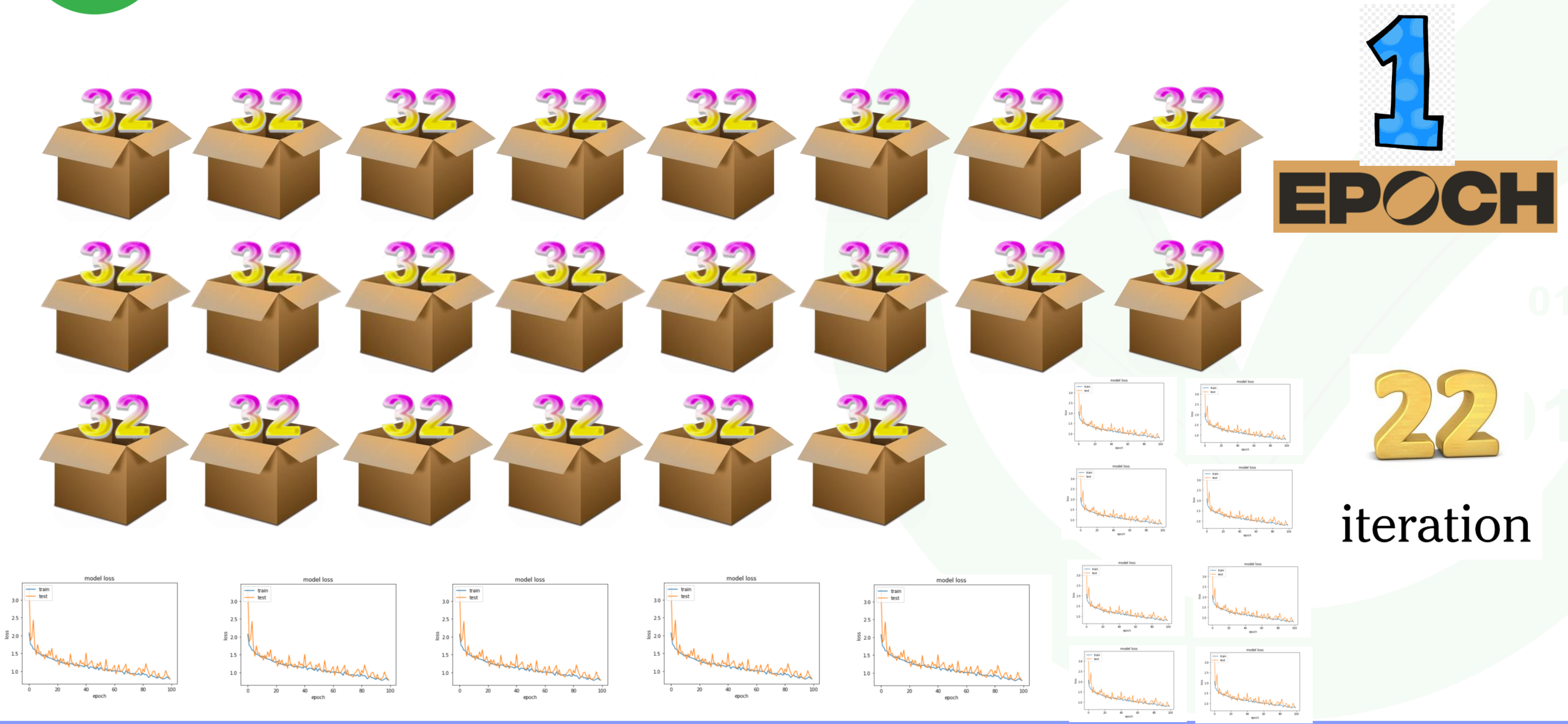
```
Epoch 1/300
22/22 [==============================] - 4s 2ms/step - loss: 256629.3281
Epoch 2/300
22/22 [==============================] - 0s 2ms/step - loss: 256489.2812
Epoch 3/300
22/22 [==============================] - 0s 3ms/step - loss: 256325.5469
```


700 (TRAIN DATASI) / 32 (BATCH SIZE) = 22

700 ÷ 32 =

21,875

# 700 (TRAIN DATASI) / 32 (BATCH SIZE) = 22



# DIFFERENCE BETWEEN BATCH AND EPOCH

## Epoch :

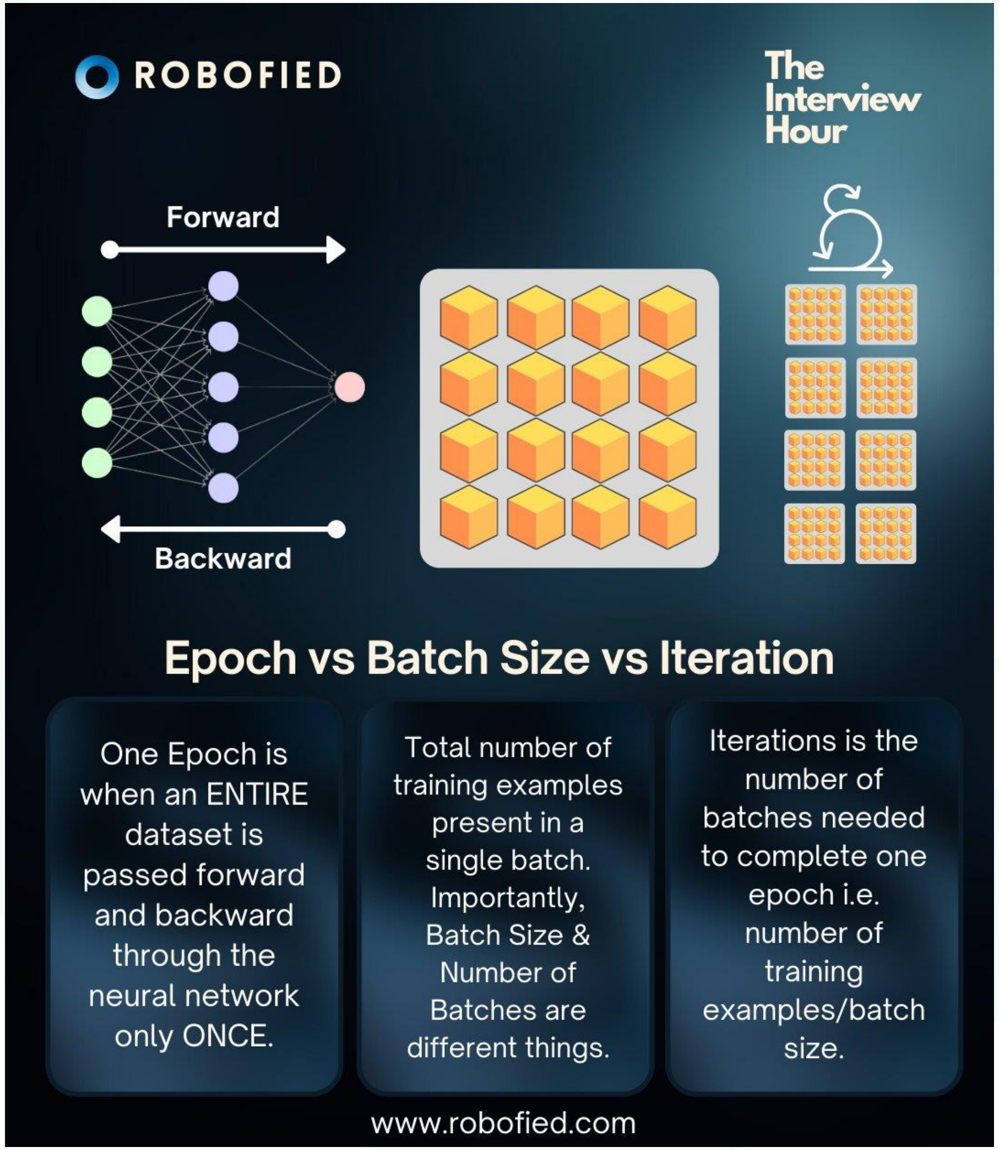
An Epoch represent one iteration over the entire dataset.

## Batch :

We cannot pass the entire dataset into the Neural Network at once. So, we divide the dataset into number of batches.

## Iteration :

If we have 1000 images as Data ane a batch size of 20, then an Epoch should run 1000/20 = 50 iteration.



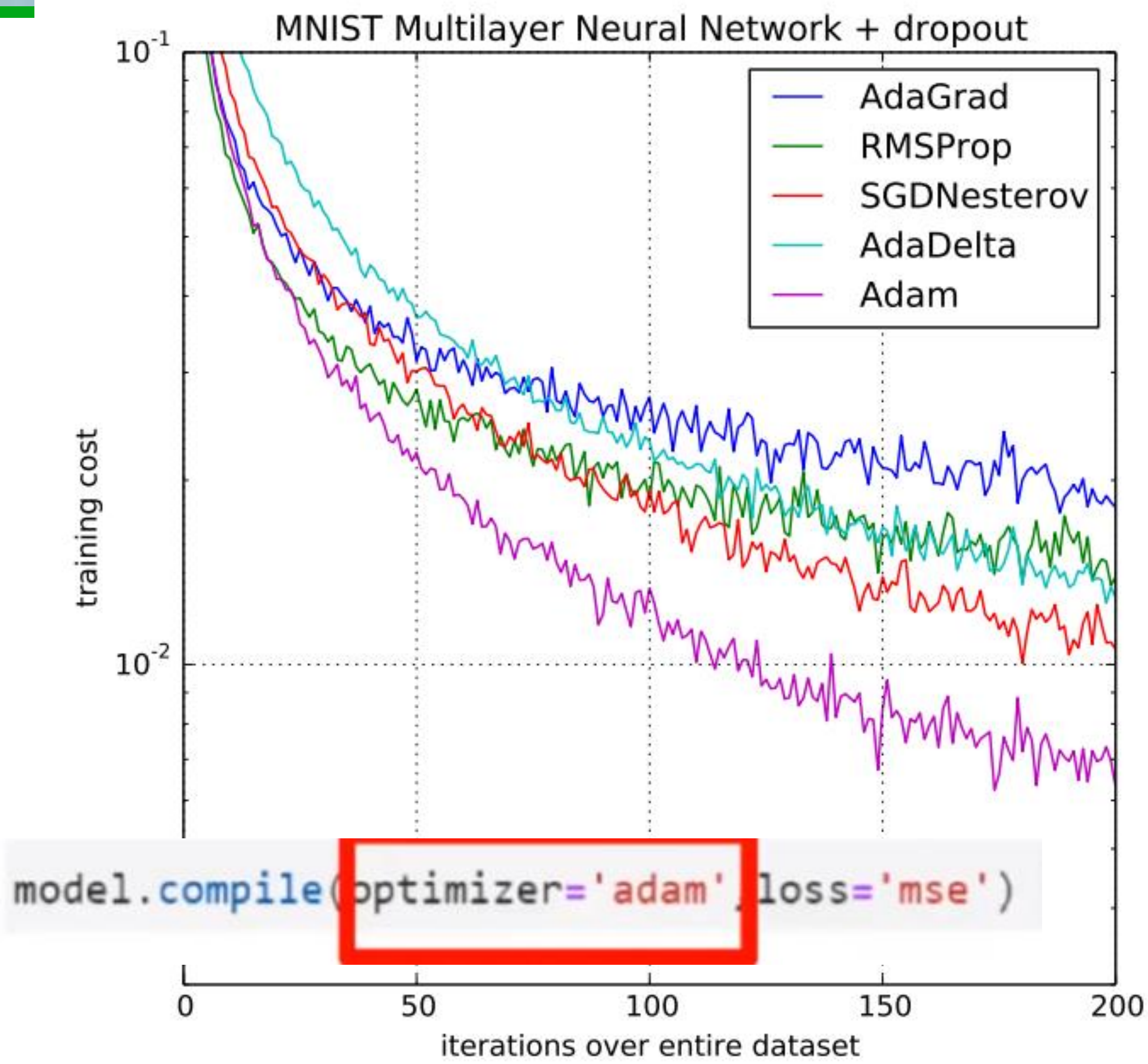
# GRADIENT DESCENT

# GRADIENT DESCENT

# GRADIENT DESCENT

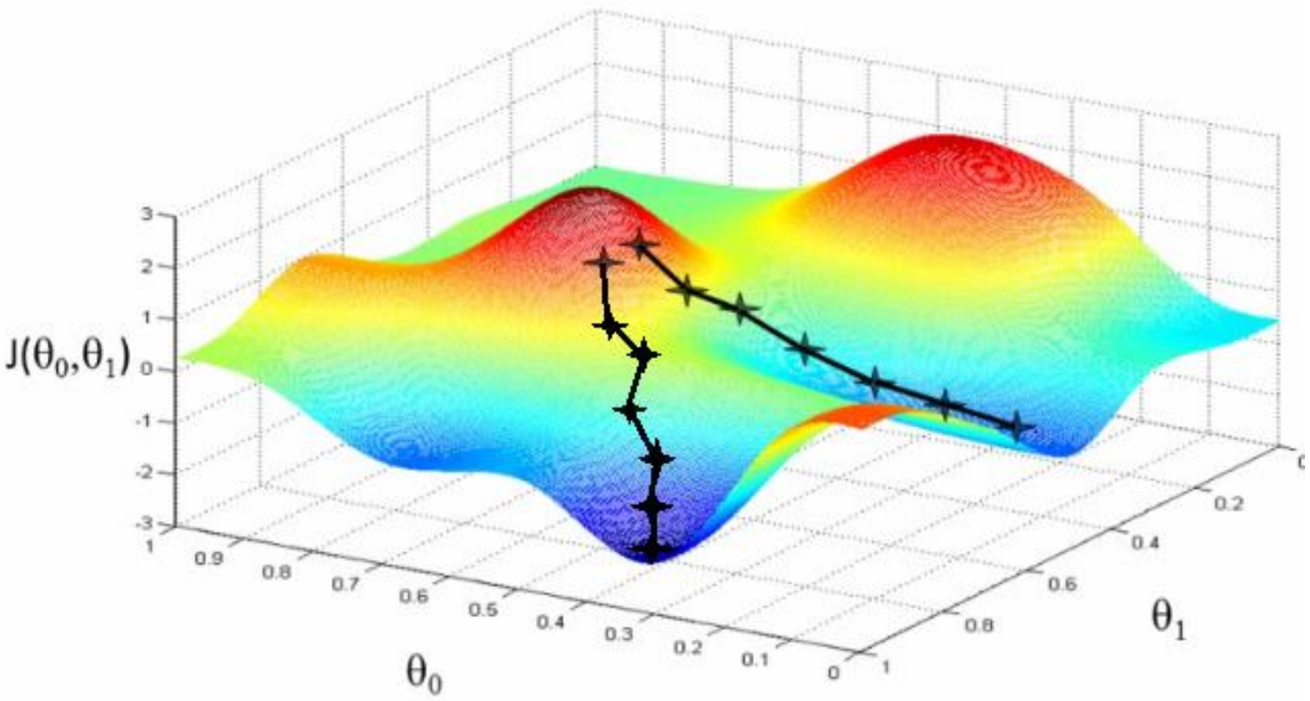
## OPTIMIZER

Gradient descent is an optimization algorithm that uses the gradient of the objective function to navigate the search space. Optimization is a mathematical discipline that determines the “best” solution in a quantitatively well-defined sense.

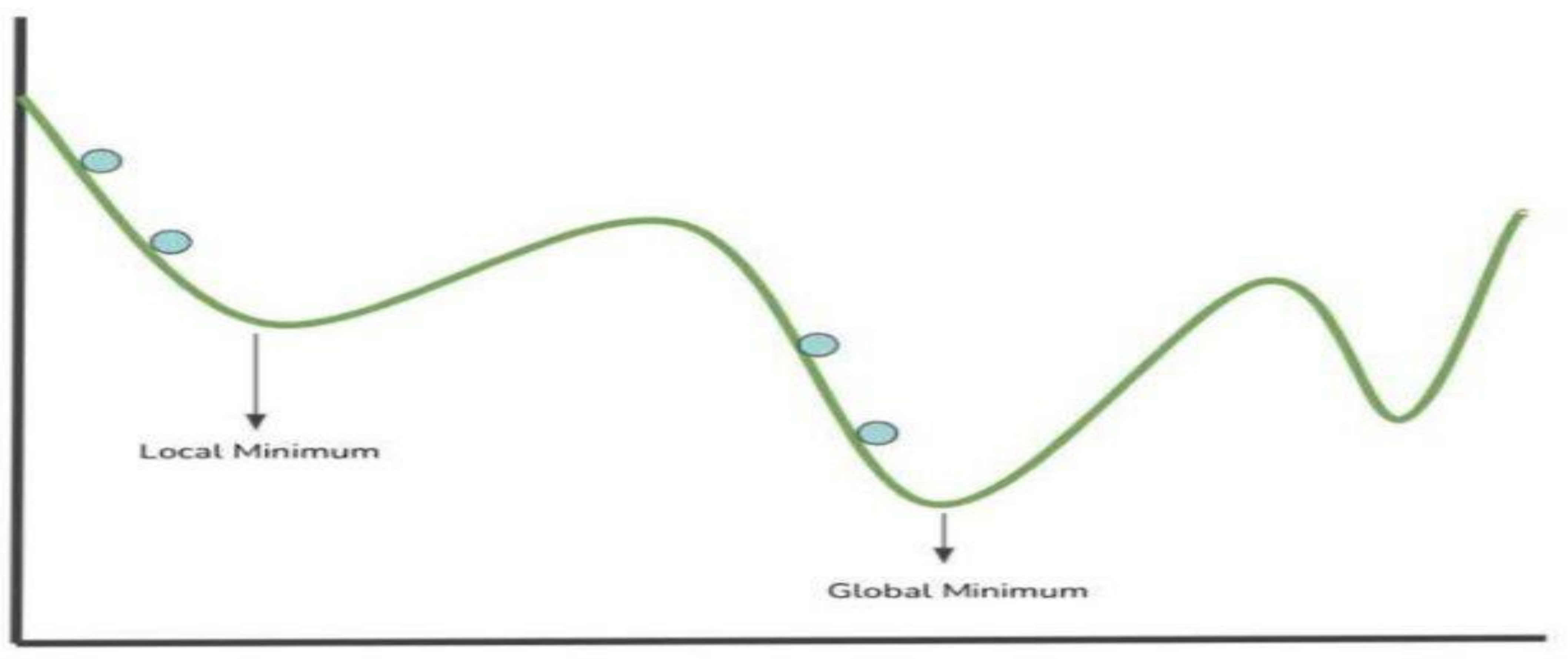


# GRADIENT DESCENT

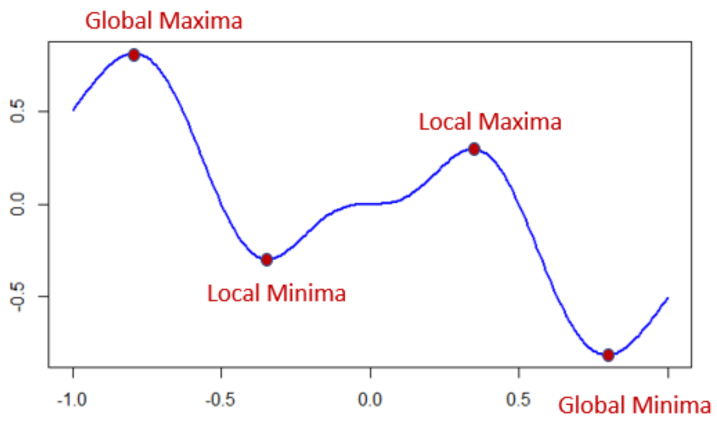
Gradient descent is the process of using gradients to find the minimum value of the cost function, while backpropagation is calculating those gradients by moving in a backward direction in the neural network.

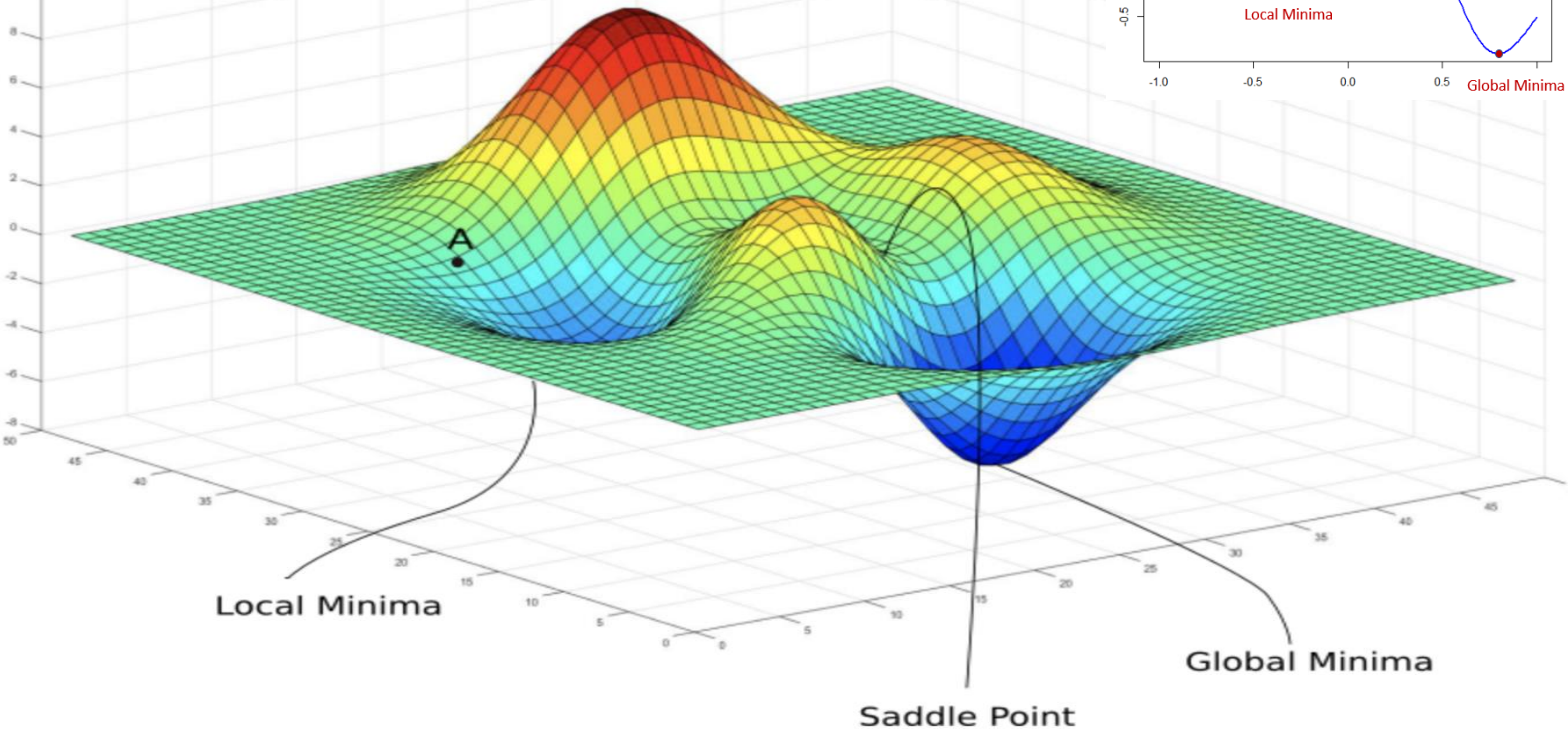



# GRADIENT DESCENT

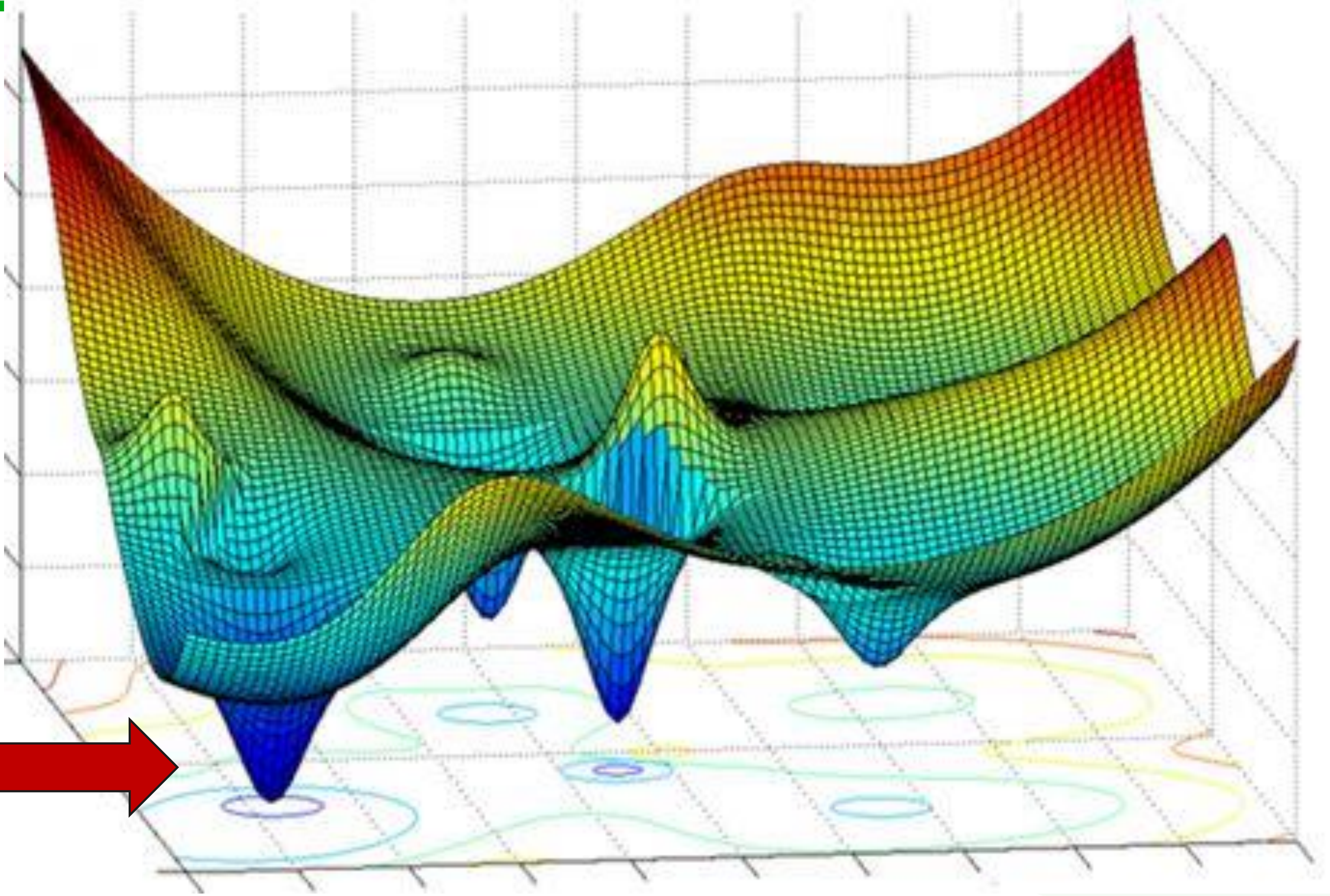


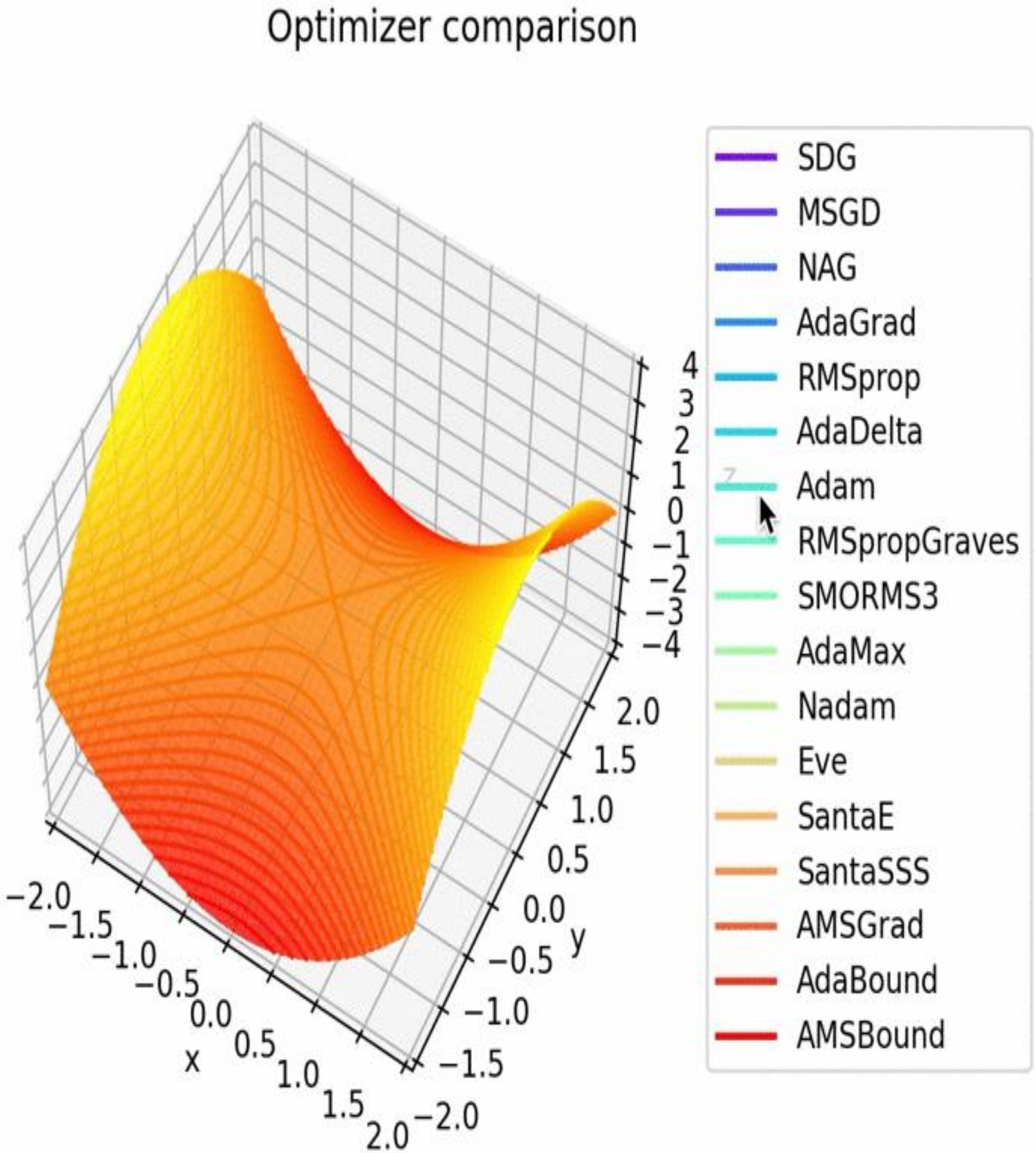
*The objective of a ML model, is to find parameters, weights or a structure that* ***minimises*** *the cost function.*

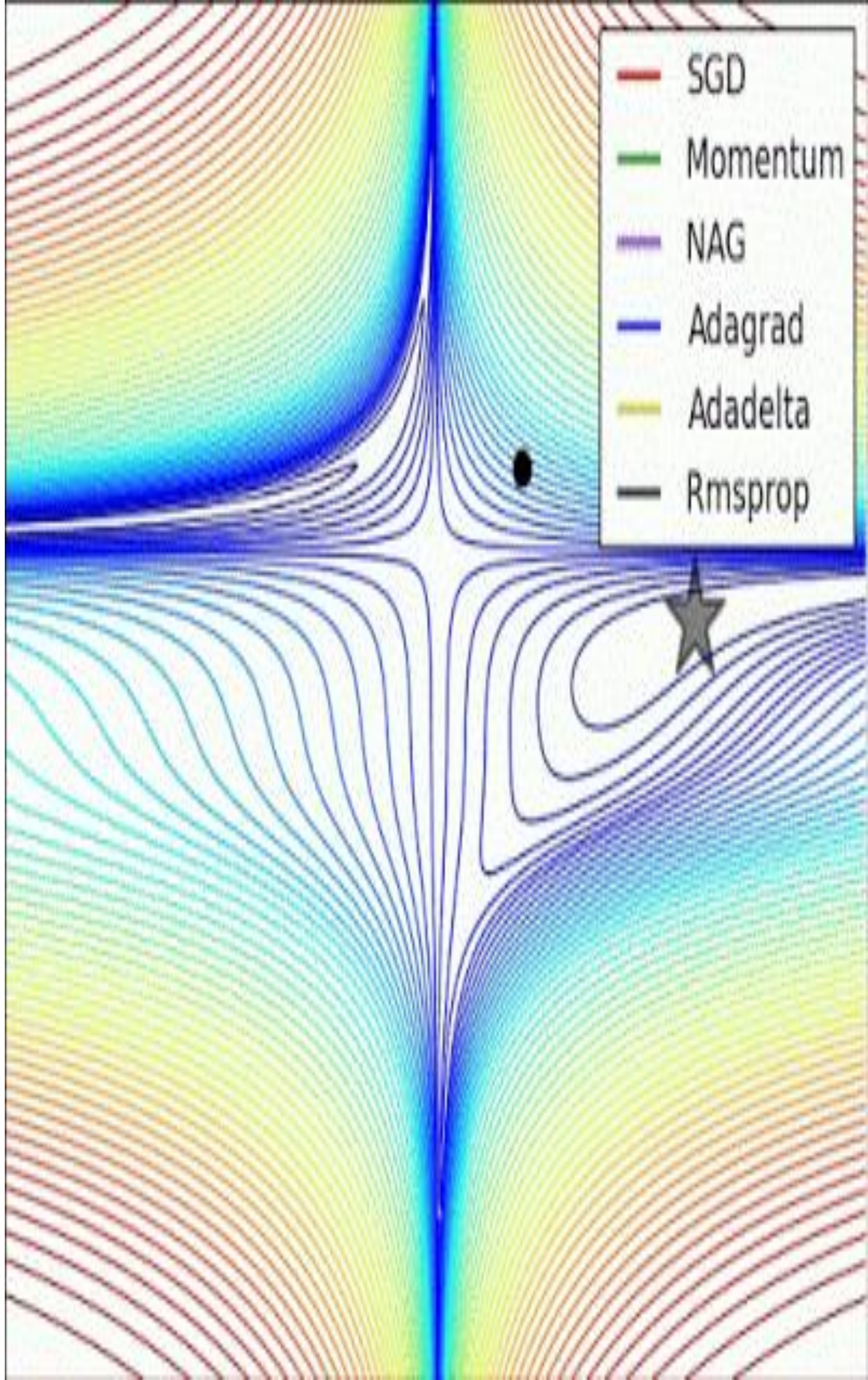


# GRADIENT DESCENT

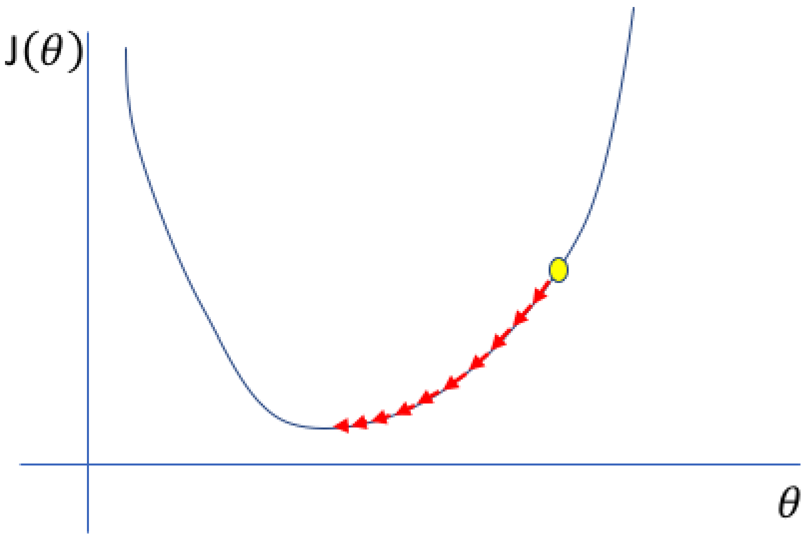



Optimizer comparison
SDG
MSGD
NAG
AdaGrad
RMSprop
AdaDelta
Adam
RMSpropGraves
SMORMS3
AdaMax
Nadam
Eve
SantaE
SantaSSS
AMSGrad
AdaBound
AMSBound
x
y


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

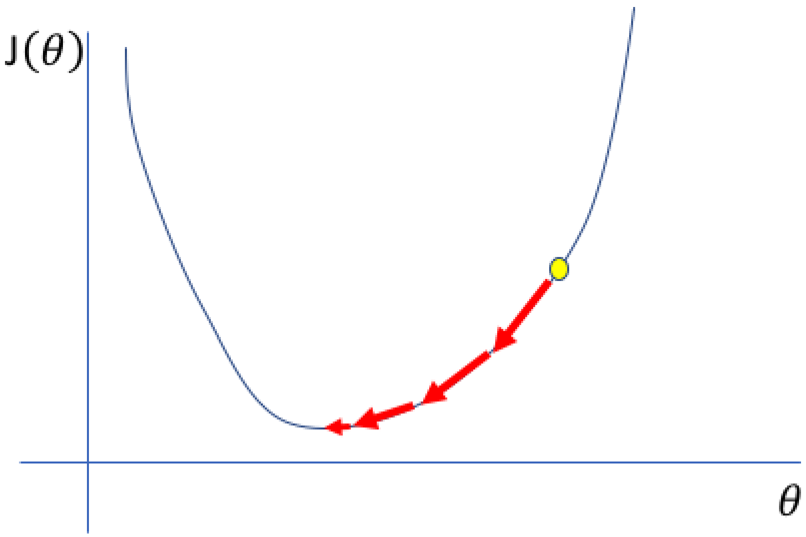
# LEARNING RATE

**Too low**



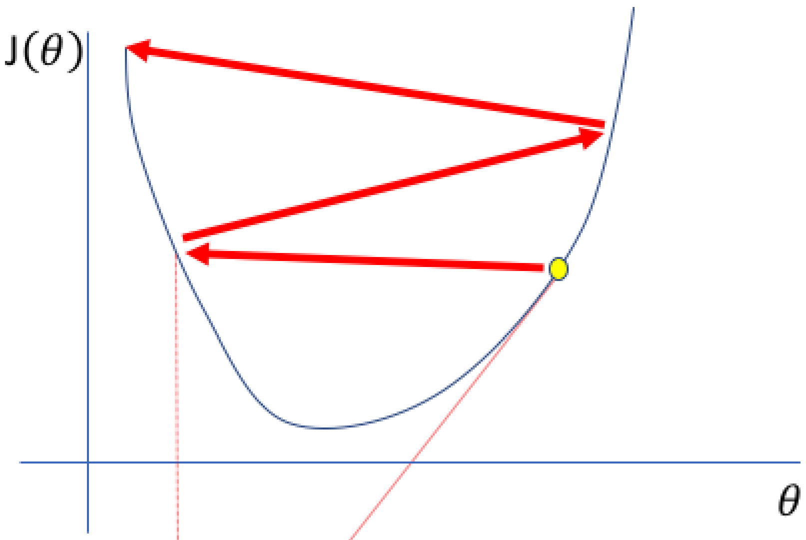

A small learning rate requires many updates before reaching the minimum point

**Just right**




The optimal learning rate swiftly reaches the minimum point

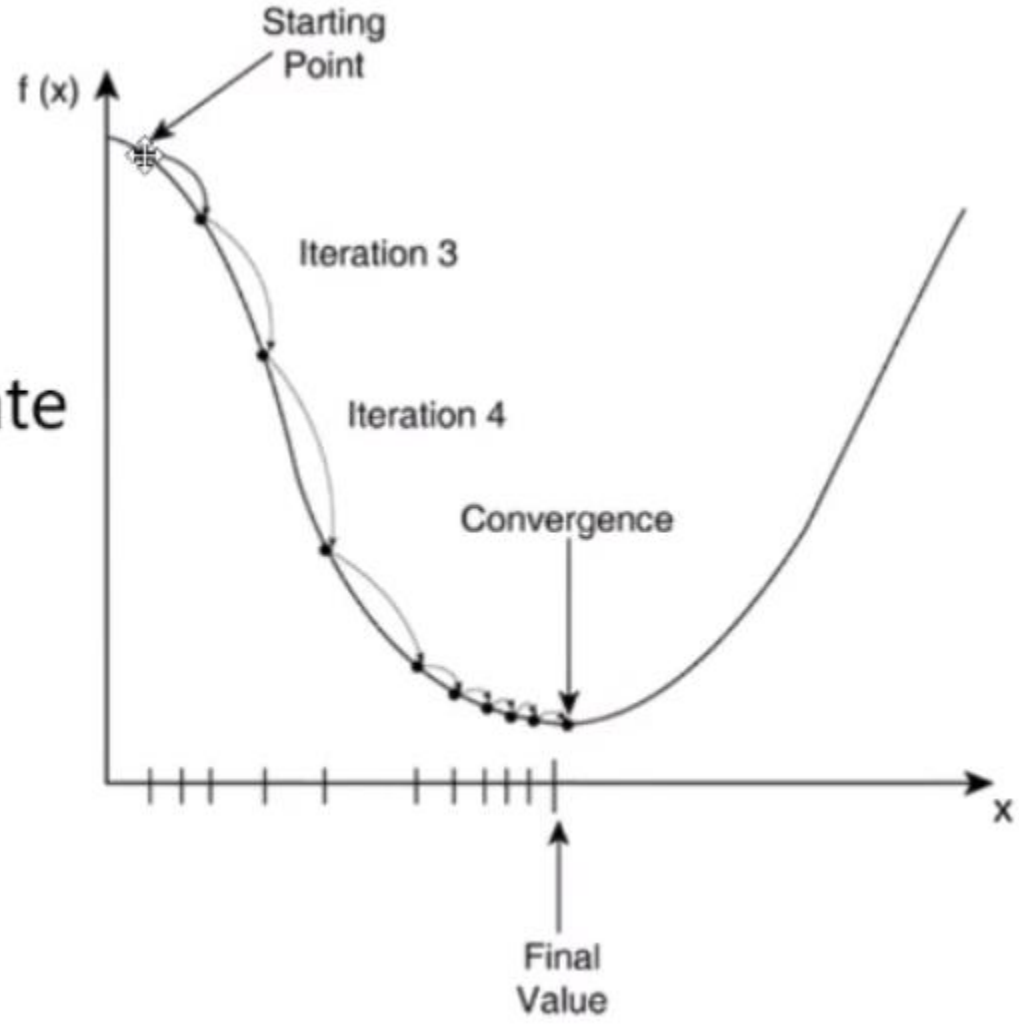
**Too high**




Too large of a learning rate causes drastic updates which lead to divergent behaviors

# LEARNING RATE

Step Size = slope x learning rate

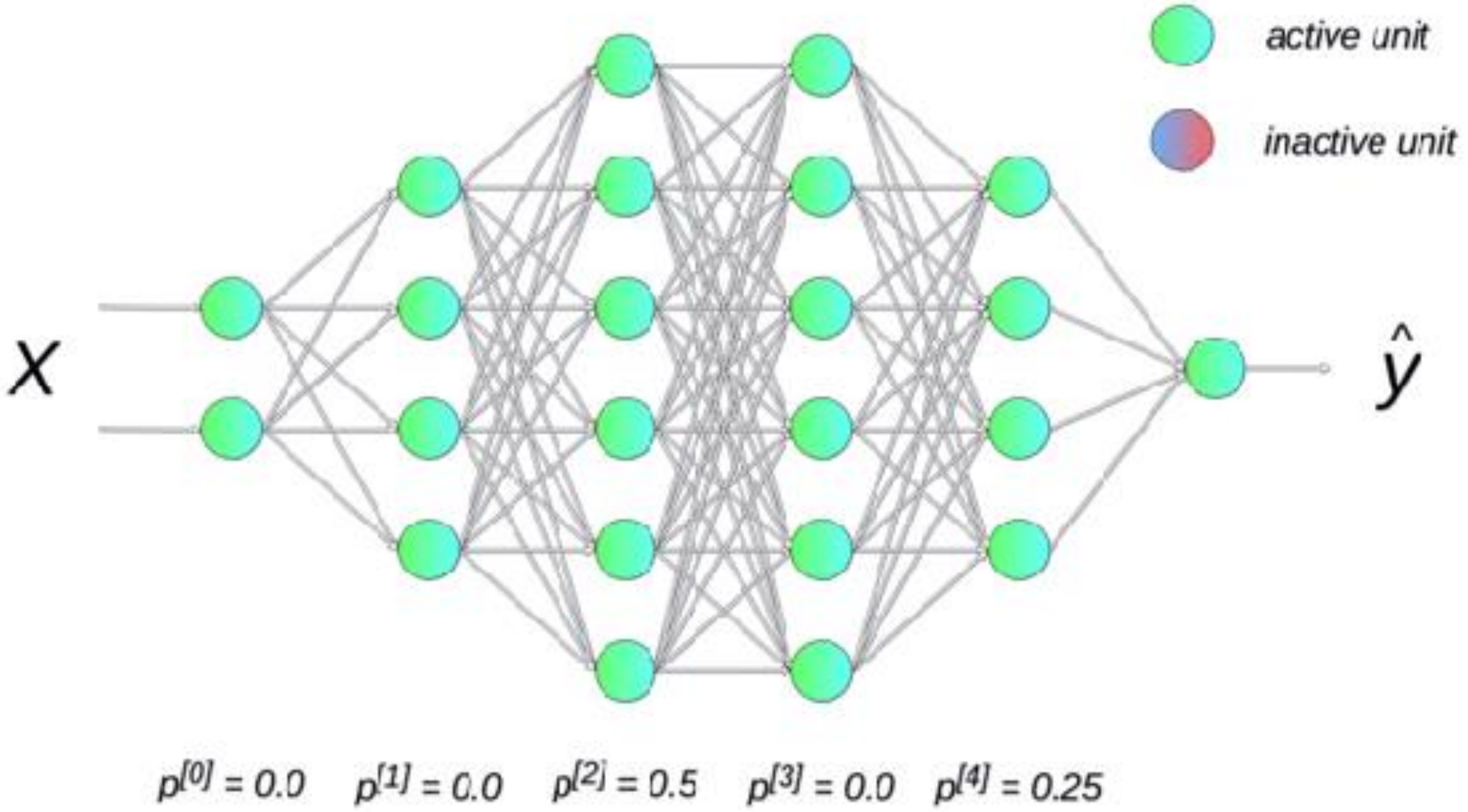


# REGULARIZATION

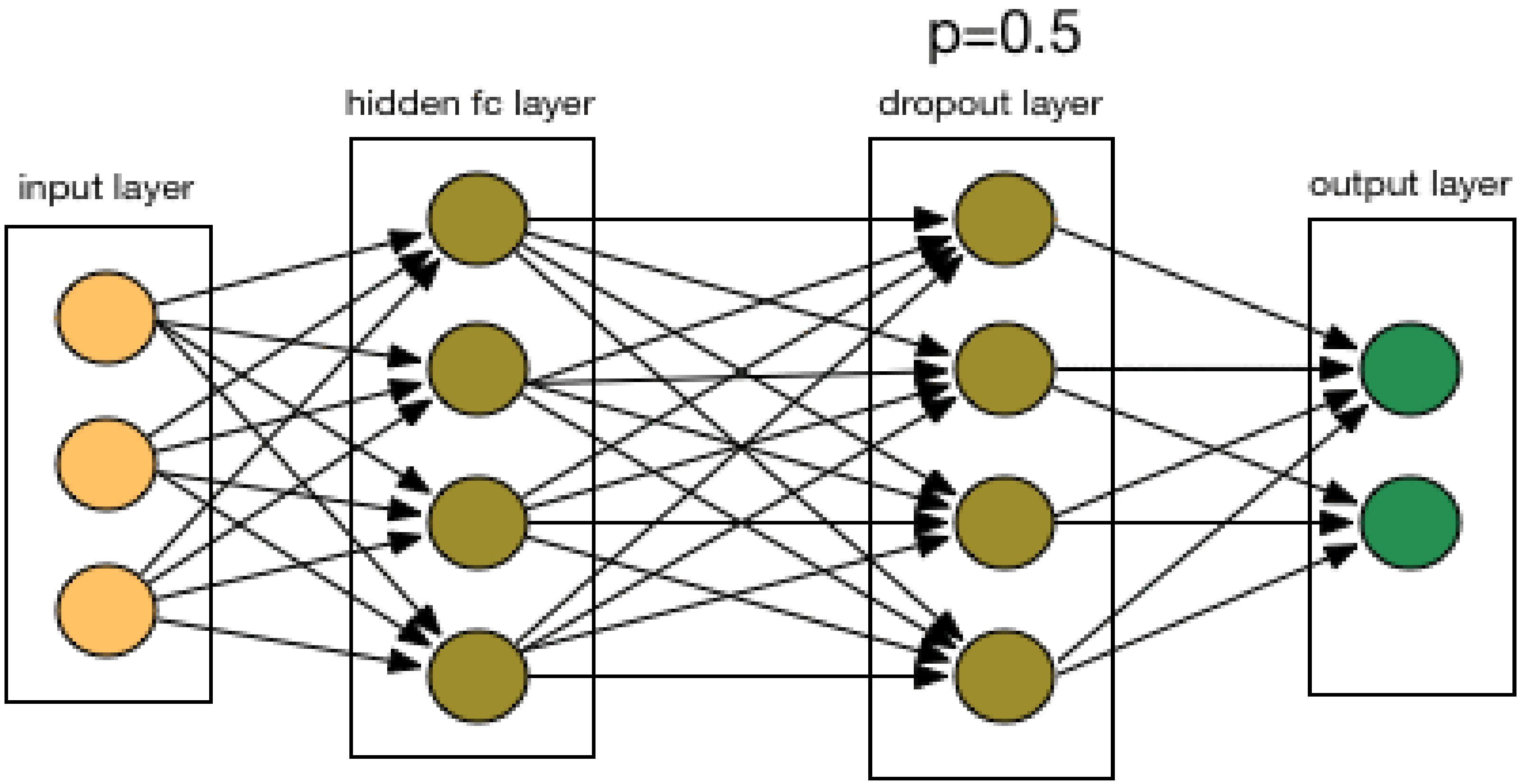
DROPOUT
EARLYSTOPPING
BATCHNORMALIZATION

L1
L2

# DROPOUT

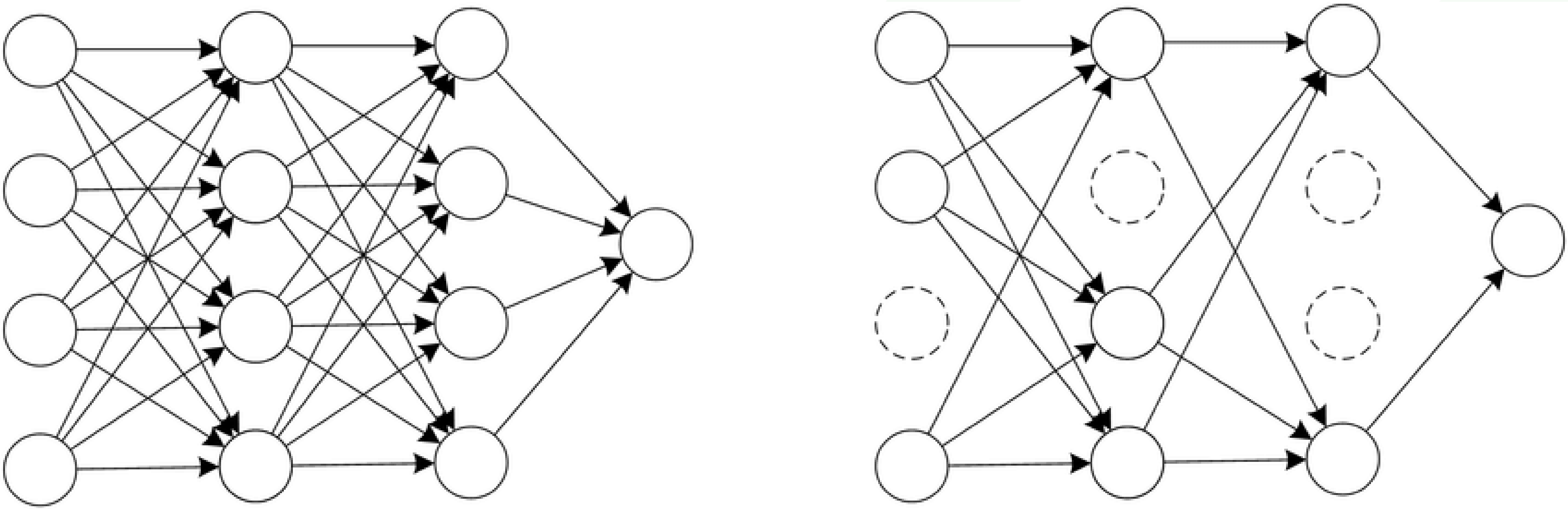


# DROPOUT




Training time

# DROPOUT

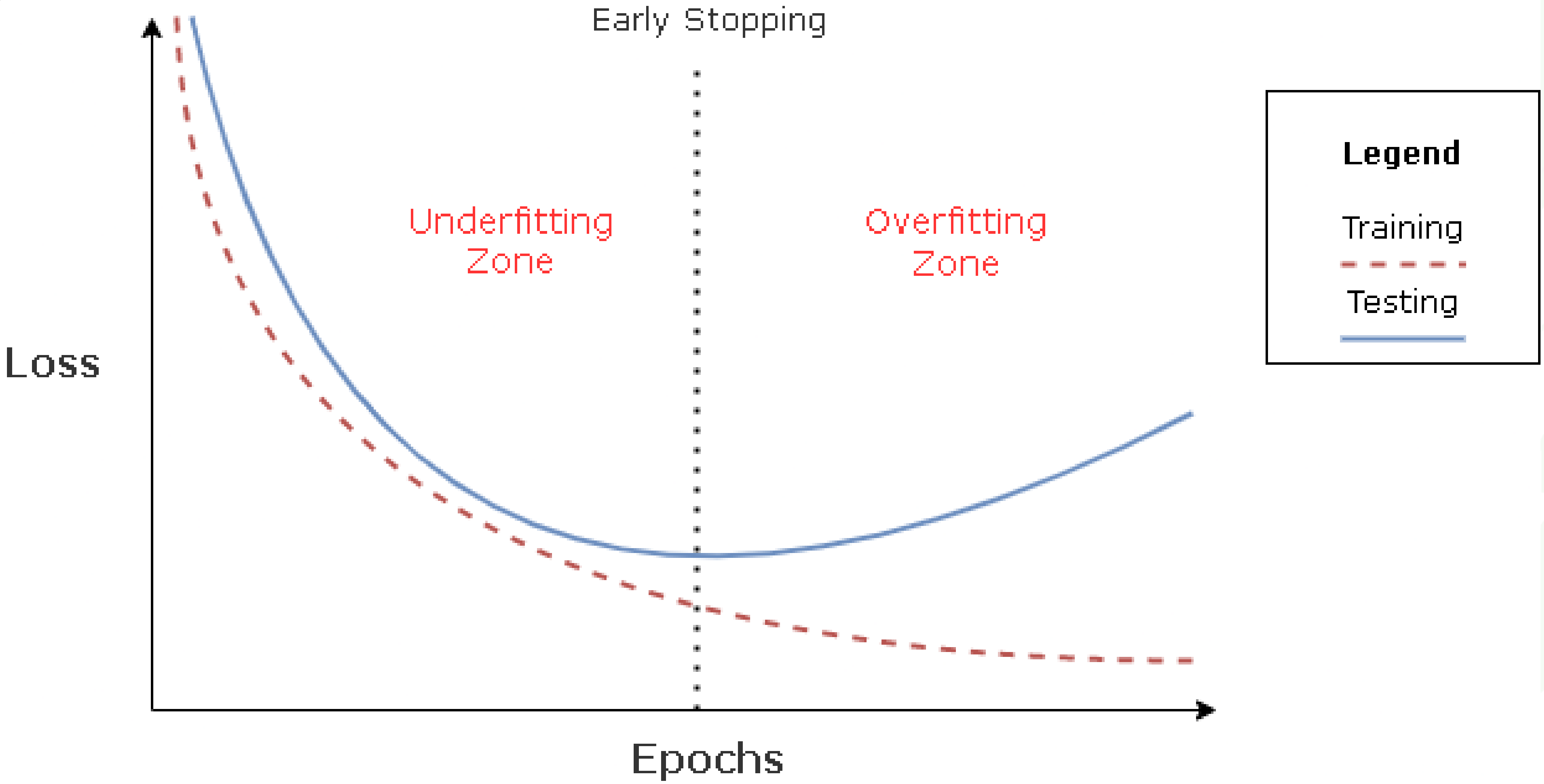


(a) Standard Neural Network

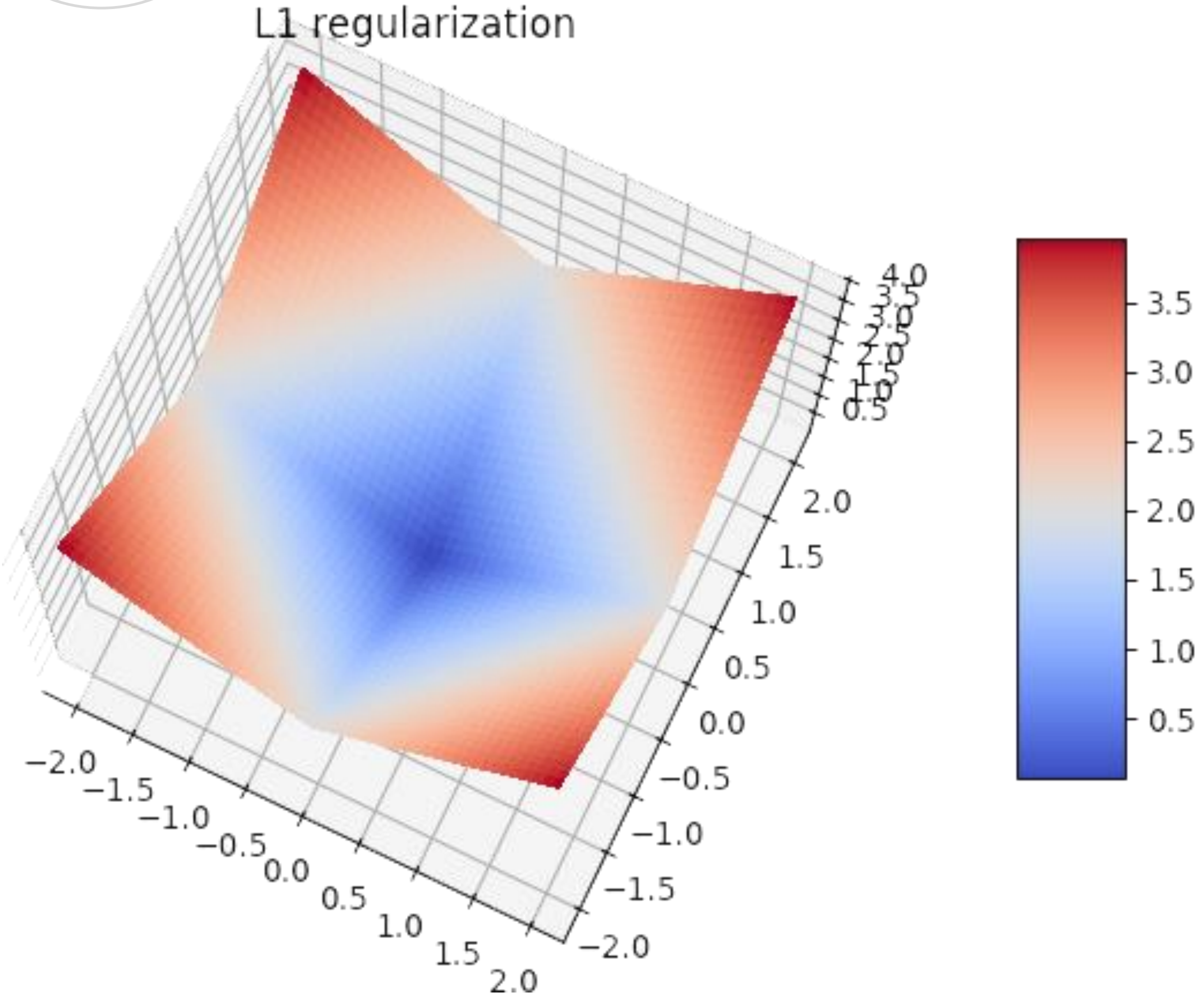
(b) Network after Dropout

# EARLYSTOPPING

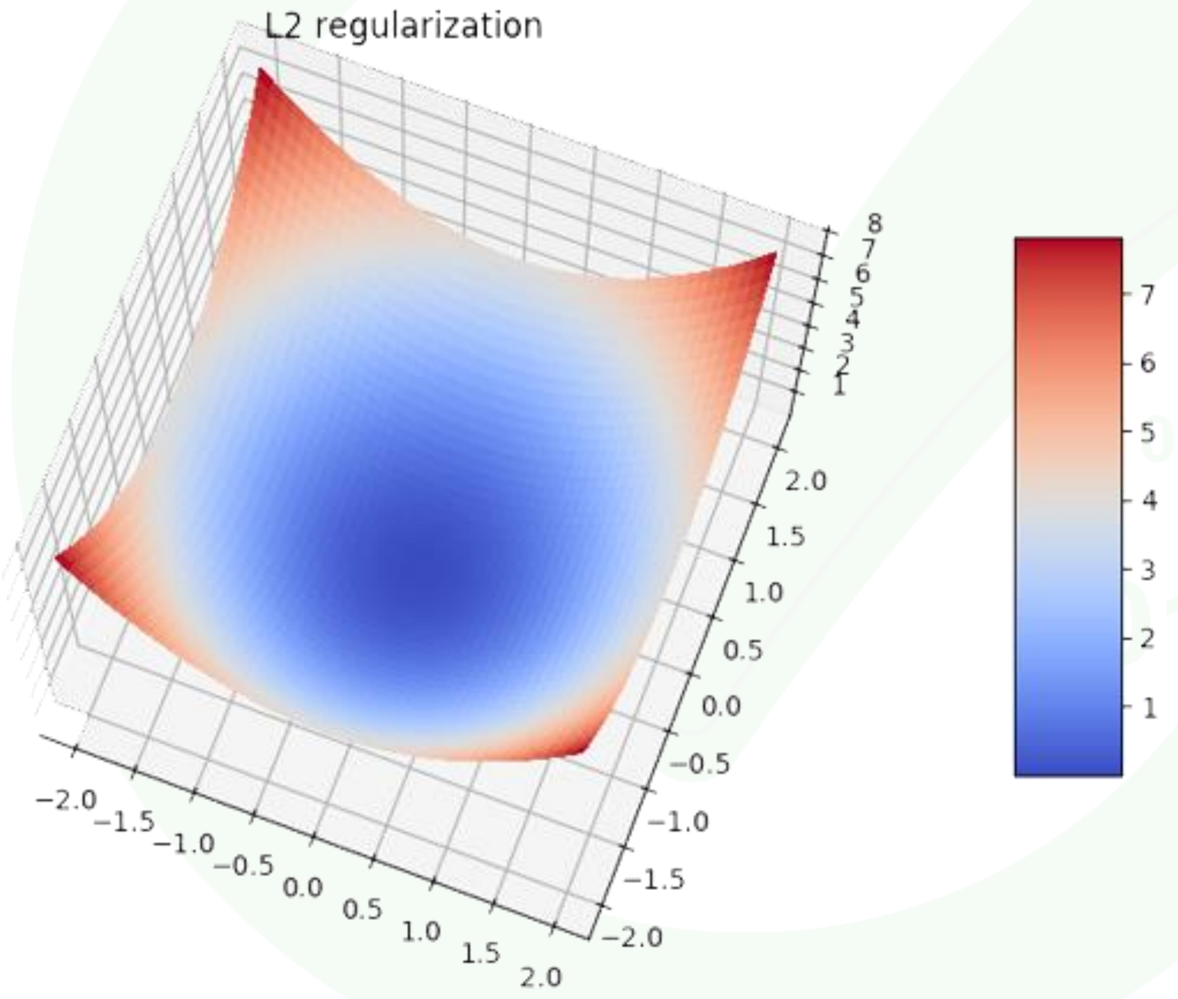
# EARLYSTOPPING



# L1 and L2



L1 regularization
3.5
3.0
2.5
2.0
1.5
1.0
0.5

L2 regularization
7
6
5
4
3
2
1

# L1 and L2

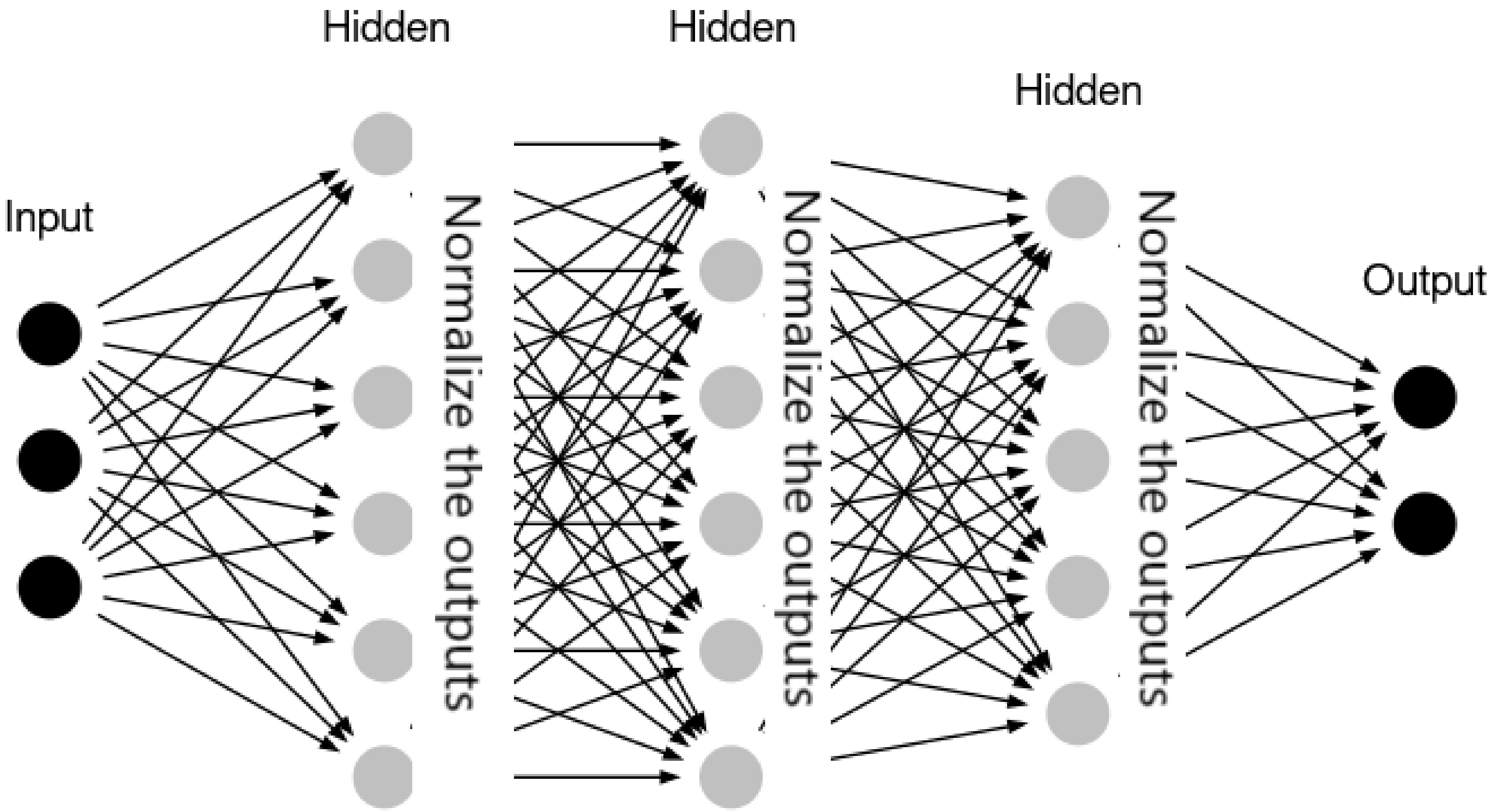
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import regularizers


tf.random.set_seed(seed)

# Modeli oluştur
model = Sequential()

# Katmanları ekle ve sadece L1 düzenlemesini uygula
model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l1(0.01)))
model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l1(0.01)))
model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l1(0.01)))
model.add(Dense(16, activation='relu', kernel_regularizer=regularizers.l1(0.01)))
model.add(Dense(8, activation='relu', kernel_regularizer=regularizers.l1(0.01)))
model.add(Dense(1))  # Düzenleme uygulanmadı
```

# L1 and L2

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import regularizers

tf.random.set_seed(seed)

# Modeli oluştur
model = Sequential()

# Katmanları ekle ve sadece L2 düzenlemesini uygula
model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(16, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(8, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(1))  # Düzenleme uygulanmadı

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

# BATCH NORMALIZATION

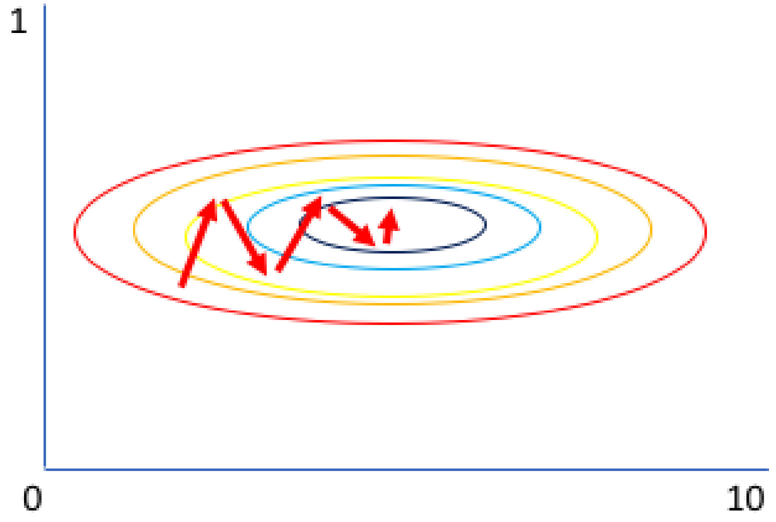
# Batch Normalization



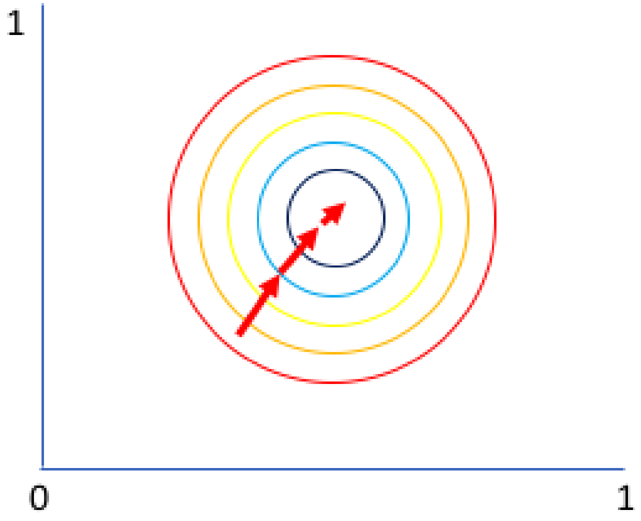
| Kullanım Durumu | Açıklama |
|---|---|
| Derin Sinir Ağları (DNN) | DNN'lerde Batch Normalization, daha hızlı ve istikrarlı bir eğitim süreci sağlayabilir ve aşırı uyumu azaltabilir. |
| Evrişimli Sinir Ağları (CNN) | CNN'lerde özellikle büyük ve karmaşık modellerde kullanılabilir. Evreli ve tam bağlantılı katmanlar arasında eklenerek özellik haritalarının daha iyi öğrenilmesine yardımcı olabilir. |
| Rekürrent Sinir Ağları (RNN) | RNN'lerde Batch Normalization, özellikle büyük zaman serisi verileri işlerken kullanılabilir. Ancak, dikkatli bir şekilde yapılandırılması gerekebilir. |
| Uzun Kısa Süreli Bellek (LSTM) ve GRU | LSTM ve GRU gibi özel RNN türleri, Batch Normalization ile kullanılabilir. Özellikle bu hücre tiplerinin daha istikrarlı eğitim sağlamasına yardımcı olabilir. |
| Doğal Dil İşleme (NLP) | Metin verilerini işleyen NLP modellerinde özellik çıkarma veya tam bağlantılı katmanlar gibi yerlerde kullanılabilir. |
| Genel Veri Normalizasyonu | Veri normalizasyonu işlemini daha hızlı ve daha kararlı hale getirir. Bu, giriş verilerini aynı ölçekleme düzeyine getirerek eğitim sürecini iyileştirebilir. |

# Batch Normalization

## Why normalize?



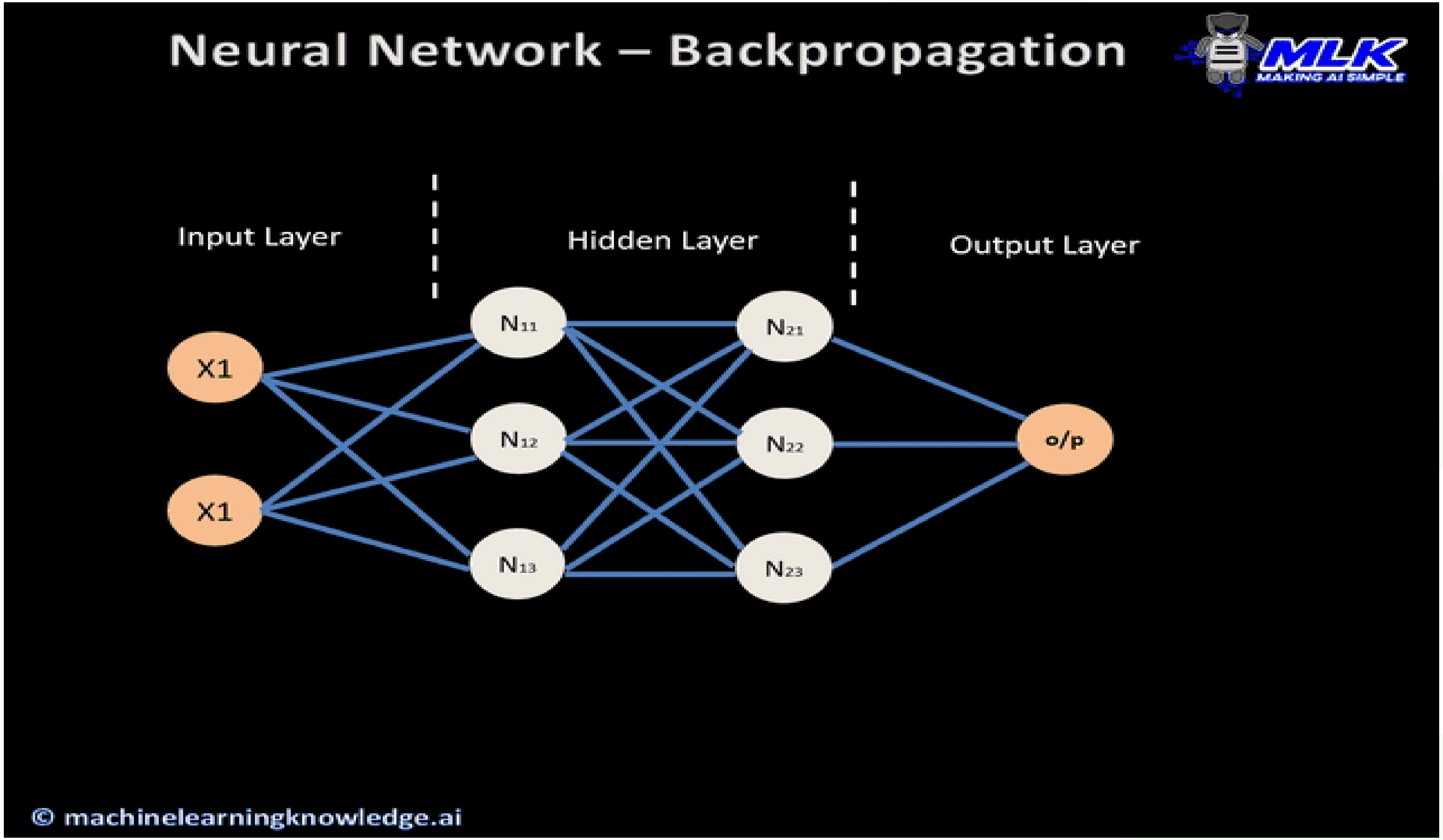

Gradient of larger parameter dominates the update



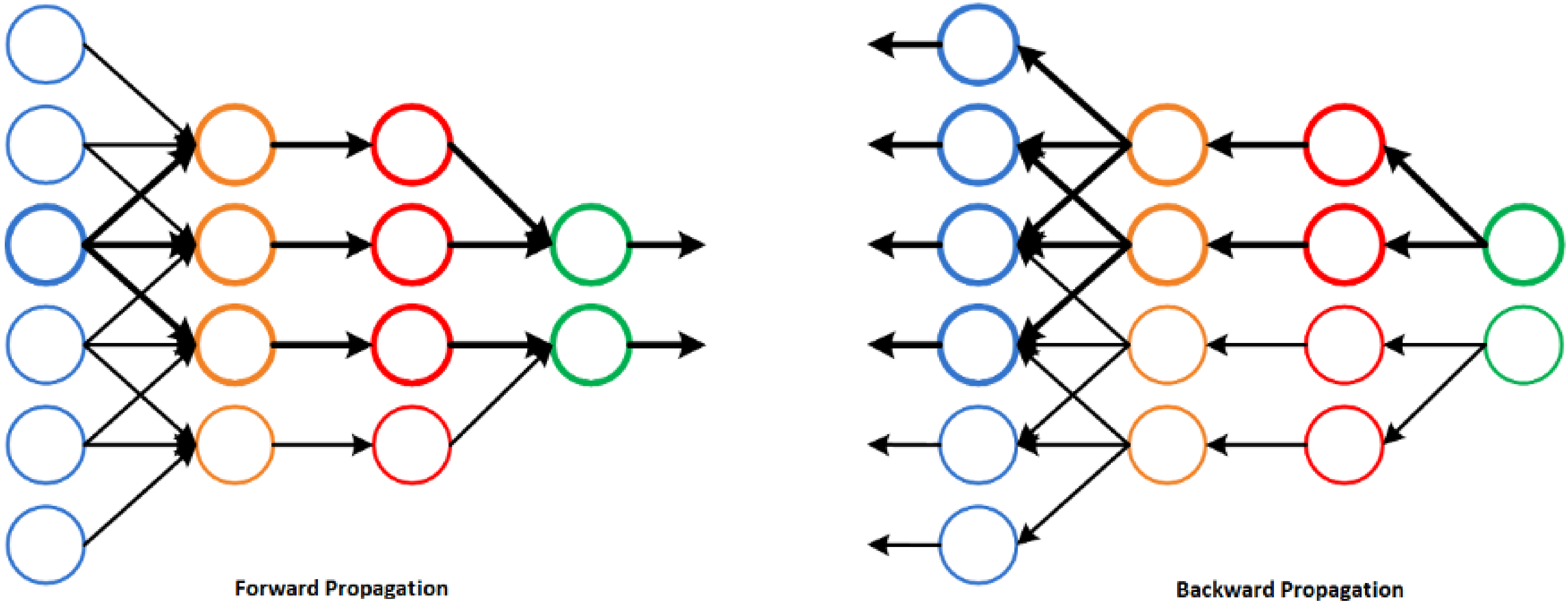

Both parameters can be updated in equal proportions

# BACKPROPAGATION

# BACKPROPAGATION



# BACKPROPAGATION



# BACKPROPAGATION

## Forward propagation

### Compute of $j_1$

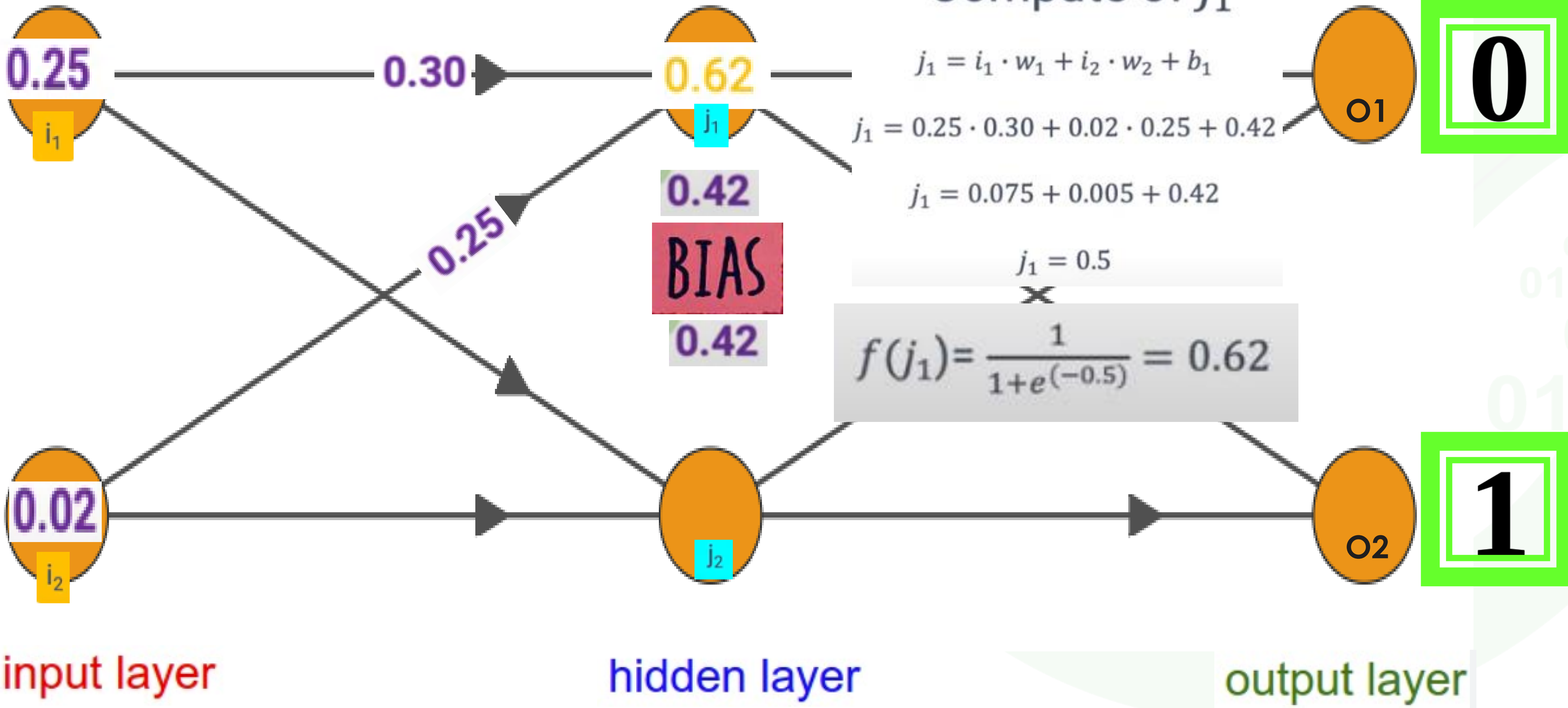
$$j_1 = i_1 \cdot w_1 + i_2 \cdot w_2 + b_1$$

$$j_1 = 0.25 \cdot 0.30 + 0.02 \cdot 0.25 + 0.42$$

$$j_1 = 0.075 + 0.005 + 0.42$$

$$j_1 = 0.5$$

$$f(j_1) = \frac{1}{1+e^{(-0.5)}} = 0.62$$



# BACKPROPAGATION

## Forward propagation

### Compute of $j_2$

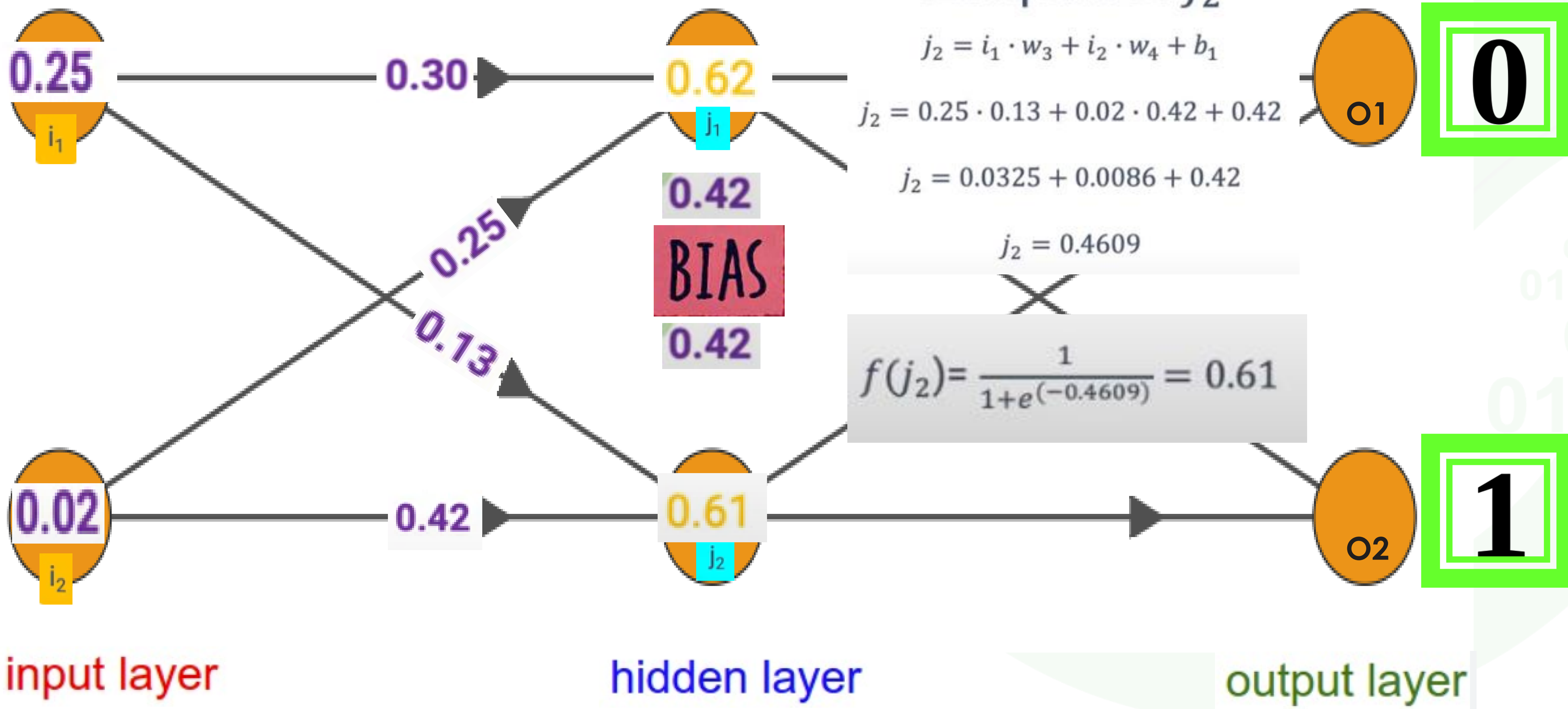
$$j_2 = i_1 \cdot w_3 + i_2 \cdot w_4 + b_1$$

$$j_2 = 0.25 \cdot 0.13 + 0.02 \cdot 0.42 + 0.42$$

$$j_2 = 0.0325 + 0.0086 + 0.42$$

$$j_2 = 0.4609$$

$$f(j_2) = \frac{1}{1+e^{(-0.4609)}} = 0.61$$



# BACKPROPAGATION

## Forward propagation

### Compute of $o_1$

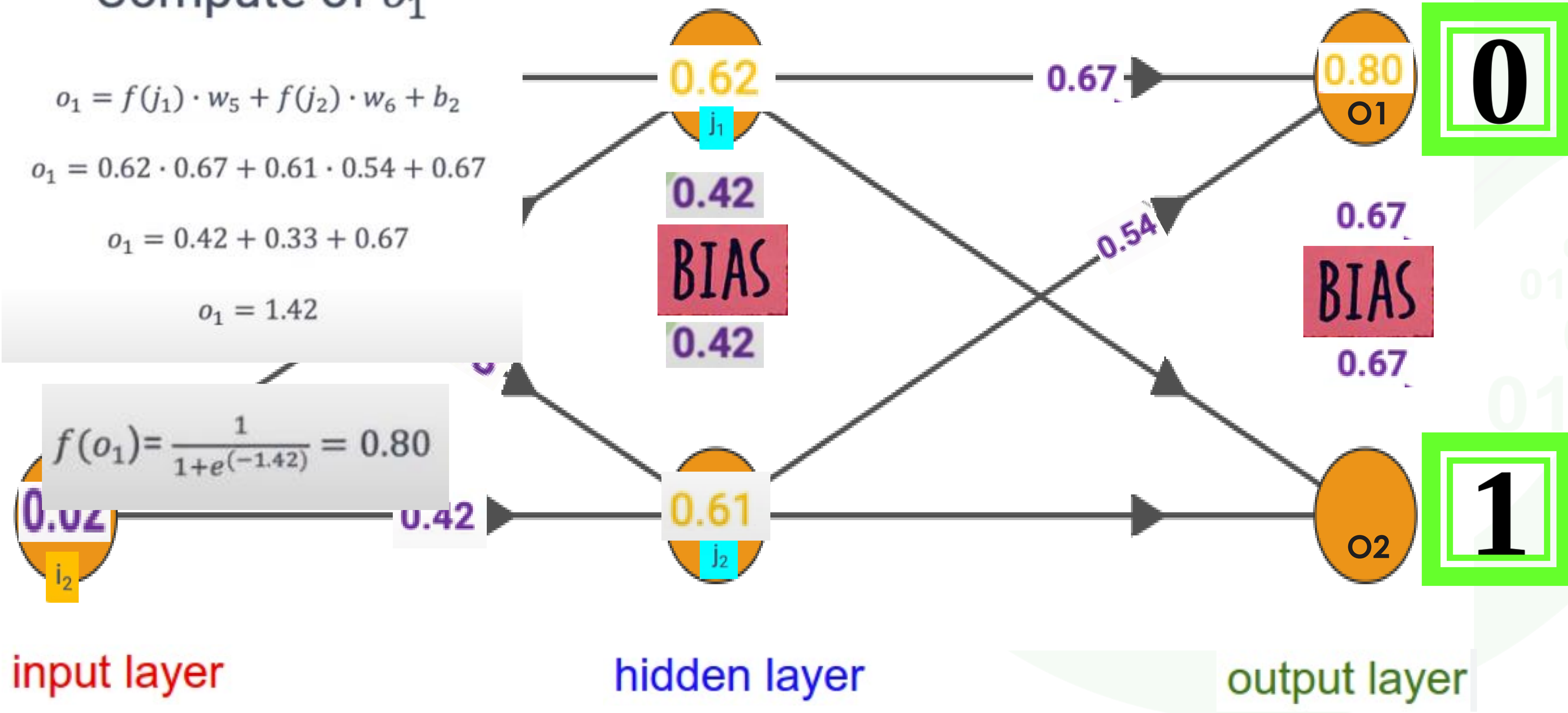
$$o_1 = f(j_1) \cdot w_5 + f(j_2) \cdot w_6 + b_2$$

$$o_1 = 0.62 \cdot 0.67 + 0.61 \cdot 0.54 + 0.67$$

$$o_1 = 0.42 + 0.33 + 0.67$$

$$o_1 = 1.42$$

$$f(o_1) = \frac{1}{1+e^{(-1.42)}} = 0.80$$



# BACKPROPAGATION

## Forward propagation

### Compute of $o_2$

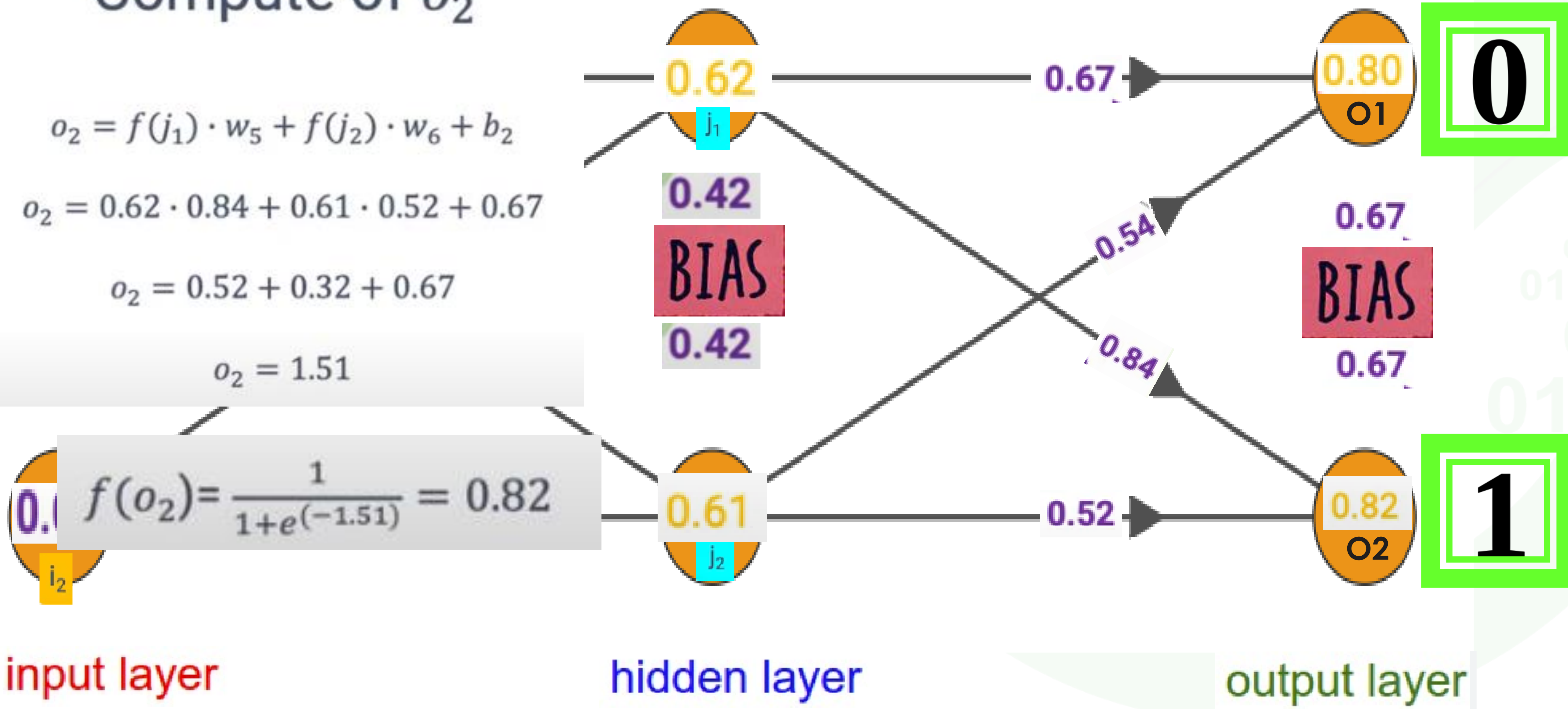
$$o_2 = f(j_1) \cdot w_5 + f(j_2) \cdot w_6 + b_2$$

$$o_2 = 0.62 \cdot 0.84 + 0.61 \cdot 0.52 + 0.67$$

$$o_2 = 0.52 + 0.32 + 0.67$$

$$o_2 = 1.51$$

$$f(o_2) = \frac{1}{1+e^{(-1.51)}} = 0.82$$



For each neuron:

$$E_{total} = \sum \frac{1}{2}(target - actual)^2$$

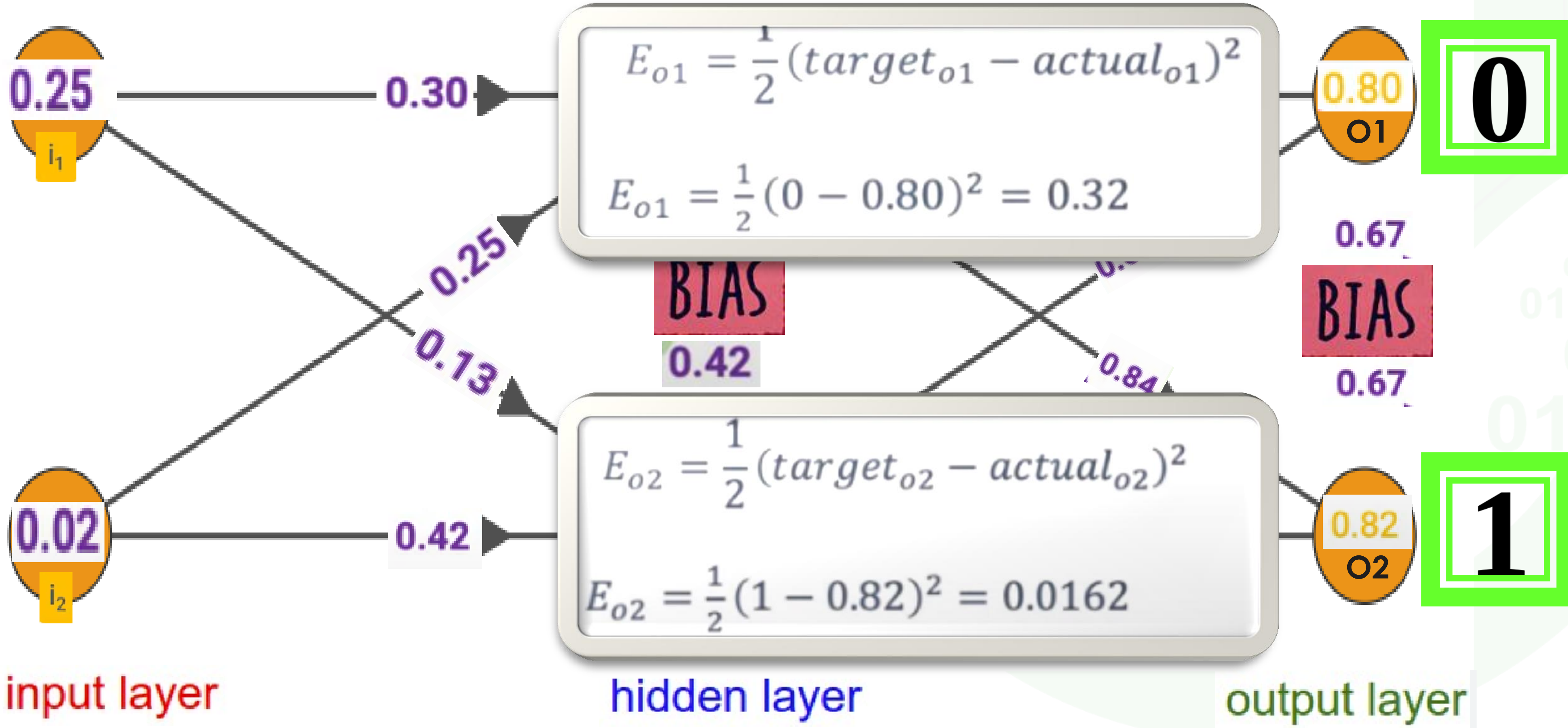
# Calculate the error

$$E_{o1} = \frac{1}{2}(target_{o1} - actual_{o1})^2$$

$$E_{o1} = \frac{1}{2}(0 - 0.80)^2 = 0.32$$

$$E_{o2} = \frac{1}{2}(target_{o2} - actual_{o2})^2$$

$$E_{o2} = \frac{1}{2}(1 - 0.82)^2 = 0.0162$$

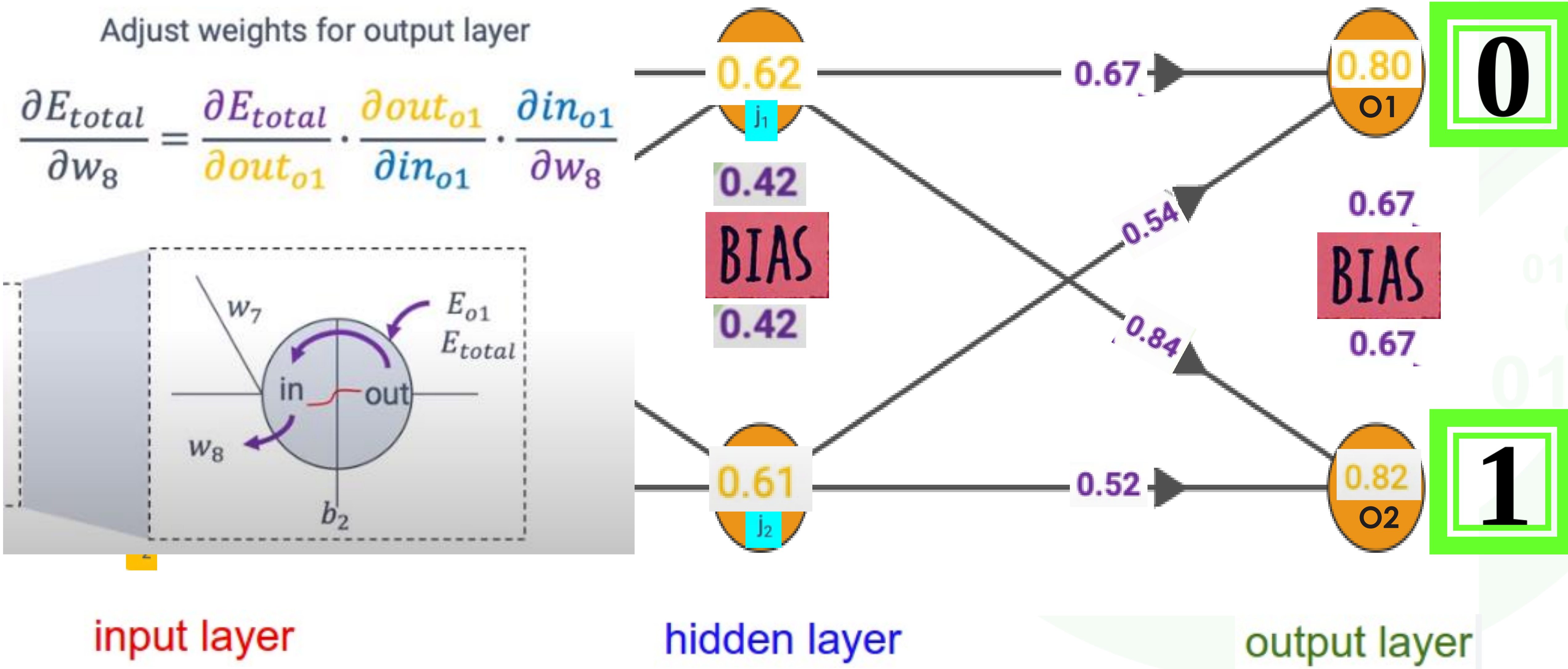


# BACKPROPAGATION

Backward propagation

Adjust weights for output layer

$$\frac{\partial E_{total}}{\partial w_8} = \frac{\partial E_{total}}{\partial out_{o1}} \cdot \frac{\partial out_{o1}}{\partial in_{o1}} \cdot \frac{\partial in_{o1}}{\partial w_8}$$



# Backward propagation

0.84 - (0.1) * (-0.027812) * (0.62) işlemini hesaplarsak:

0.84 + 0.00171787 = 0.84171787

$\delta$o2= -(O2-O2)çıkO2(1-çıkO2)

-(1-0.82)0.82(1-0.82)

-0.18 * 0.82 * 0.18

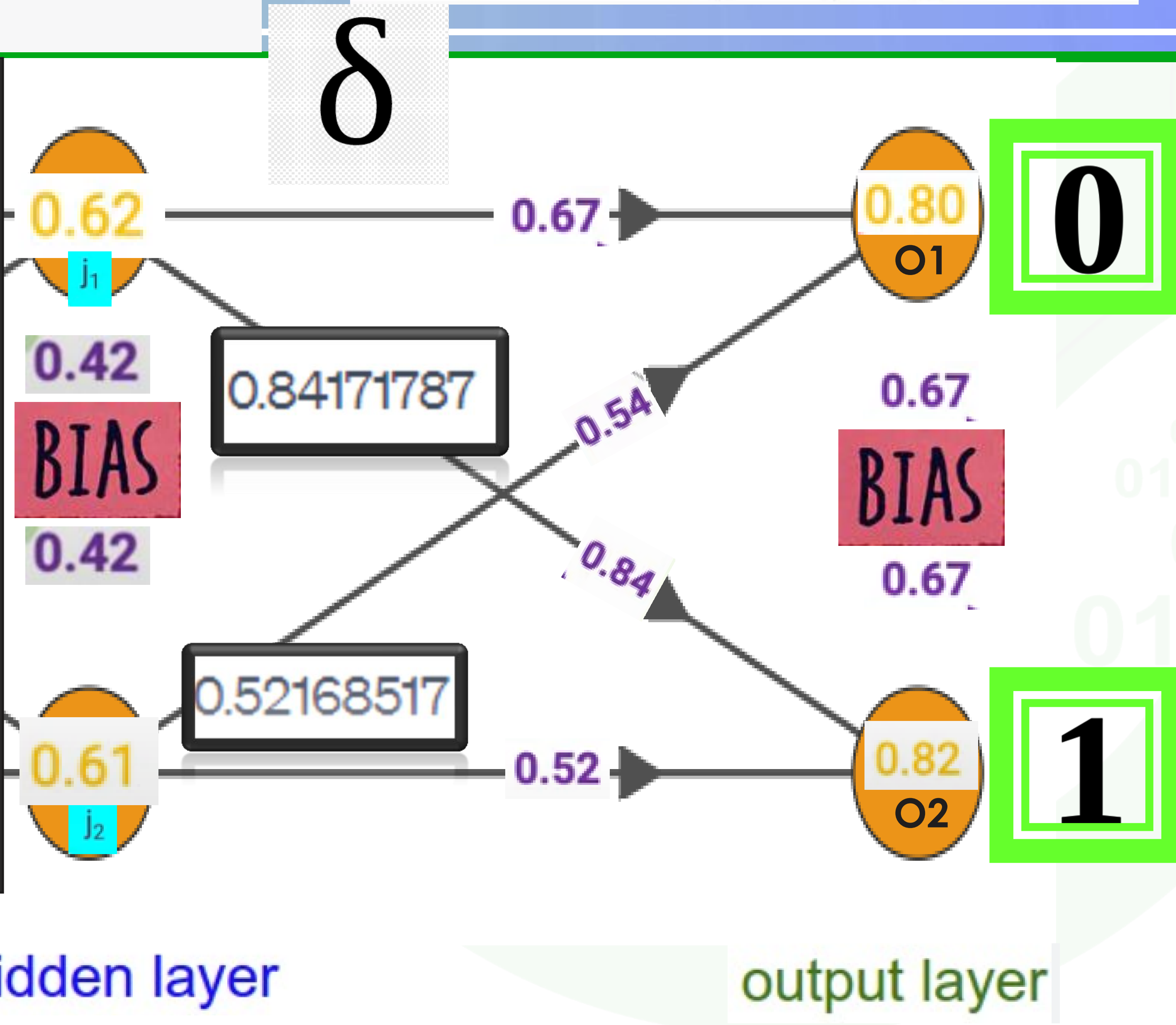
- 0.027812

ESKİ AĞIRLILIK (0.52)

Wyeni= Eski ağırlık- Learning rate $\delta$O2outputJ2

0.52-(0.1)(- 0.027812)(0.61)

0.52168517



# NOTEBOOK SAMPLES

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization

# Veri setini yüklemek ve hazırlamak için gerekli işlemleri yapabilirsiniz.

# Örnek bir CNN modeli oluşturma
model = Sequential()

# İlk evreli katman (Convolutional Layer)
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(BatchNormalization())  # Batch Normalization katmanı ekleniyor
model.add(MaxPooling2D((2, 2)))

# İkinci evreli katman
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))

# Düzleştirme katmanı
model.add(Flatten())

# Tam bağlantılı katmanlar
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(10, activation='softmax'))   # Örnek bir çıkış katmanı

# Modeli derleme
model.compile(optimizer='adam',
          loss='categorical_crossentropy',
          metrics=['accuracy'])

# Modeli öğrenme verileriyle eğitme
model.fit(x_train, y_train, epochs=10, batch_size=32)
```

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization

# Veri setini yüklemek ve hazırlamak için gerekli işlemleri yapabilirsiniz.

# Örnek bir ANN modeli oluşturma
model = Sequential()

model.add(Dense(64, input_dim=input_dim, activation='relu'))
model.add(BatchNormalization())  # Batch Normalization katmanı ekleniyor
model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(output_dim, activation='softmax'))

# Modeli derleme
model.compile(optimizer='adam',
          loss='categorical_crossentropy',
          metrics=['accuracy'])

# Modeli öğrenme verileriyle eğitme
model.fit(x_train, y_train, epochs=10, batch_size=32)
```