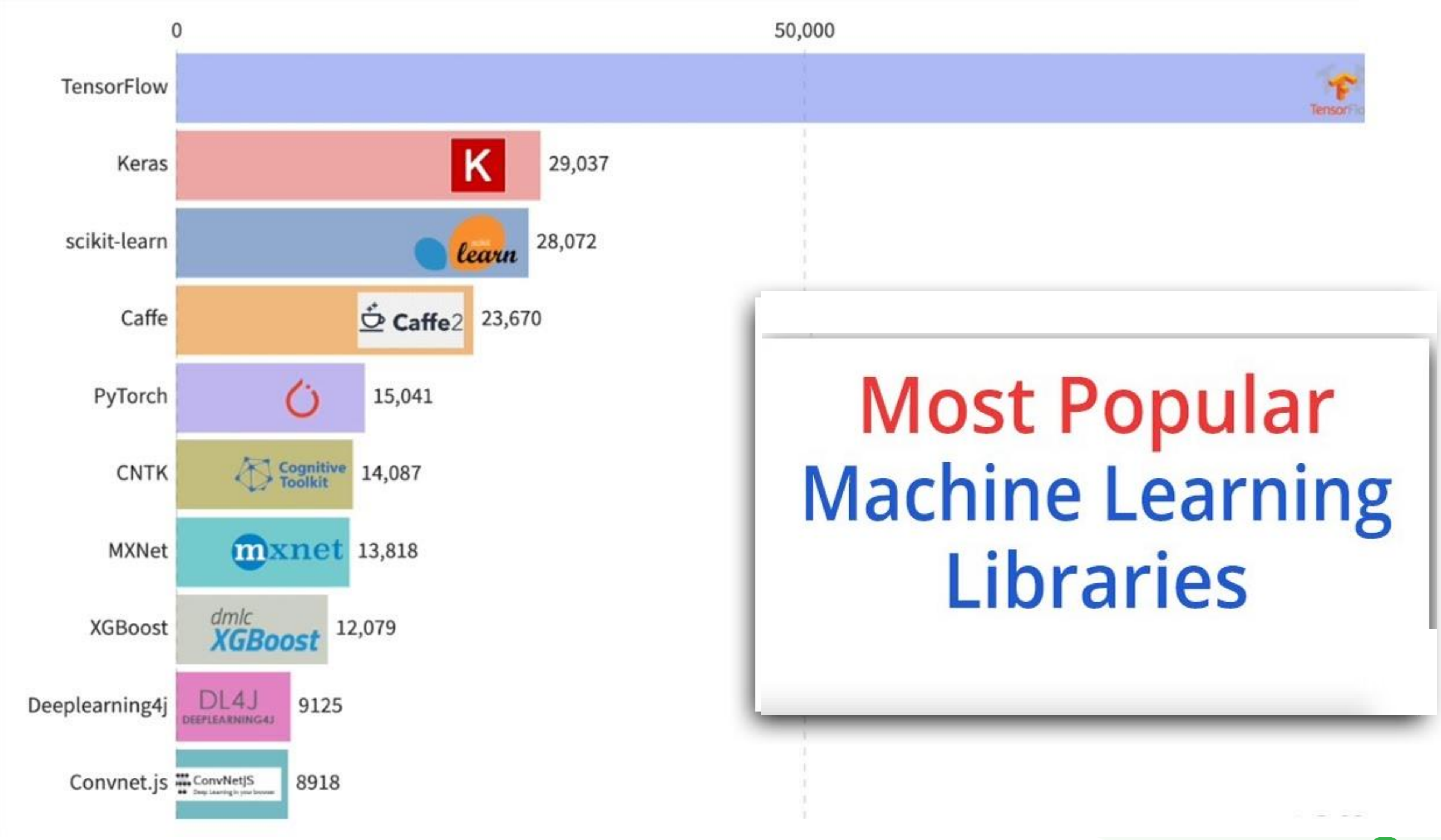
DATA SCIENCE

DATE	:	08.10.2024
DT/NT	:	DT
LESSON	:	DEEP LEARNING
SUBJECT	:	ANN BACKPROPAGATION
BATCH	:	250

TECHPRO EDUCATION

MOST POPULAR DEEP LEARNING LIBRARIES & PLATFORMS

DEEP LEARNING LIBRARIES



DEEP LEARNING LIBRARIES

TECHPRO EDUCATION

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.

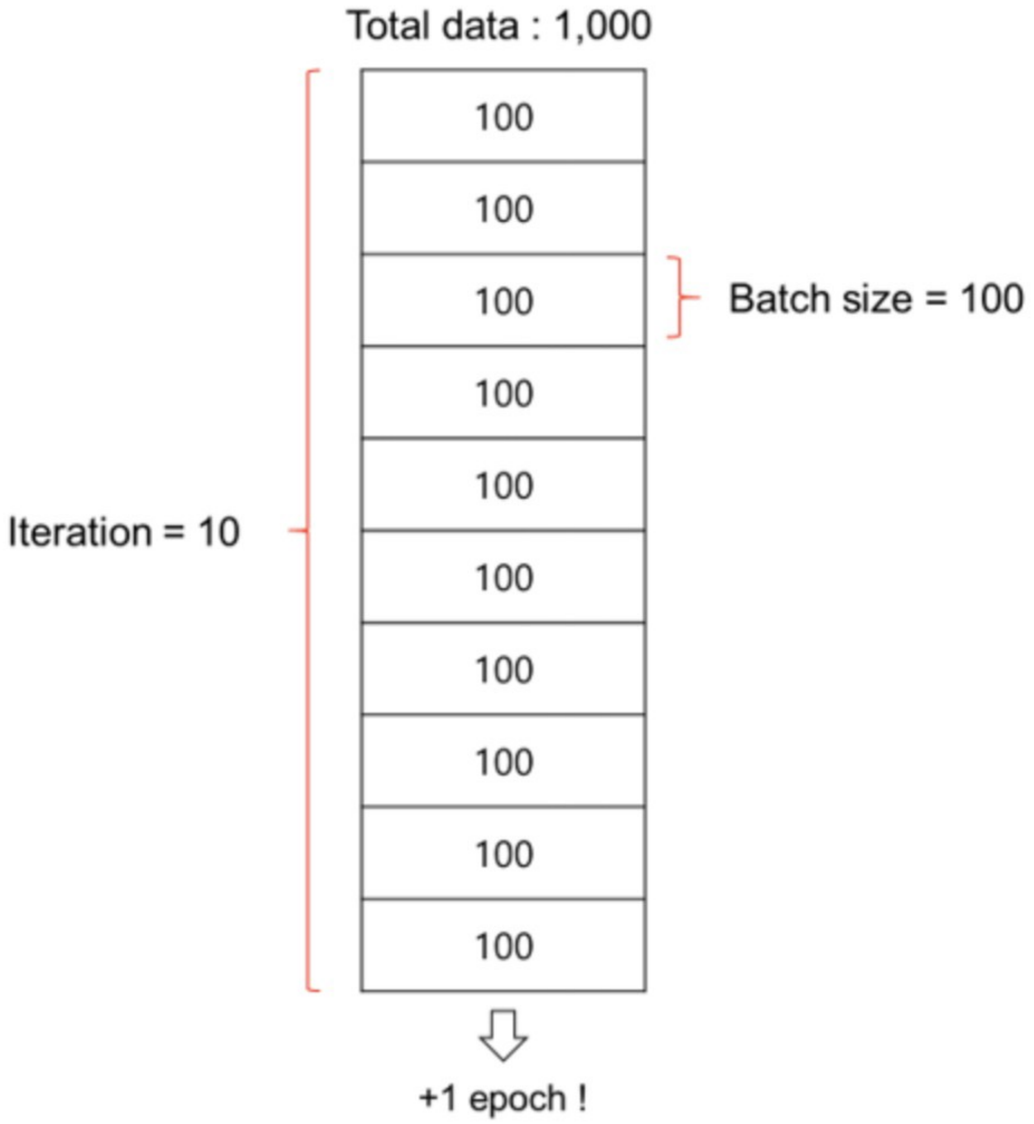
DEEP LEARNING LIBRARIES

	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

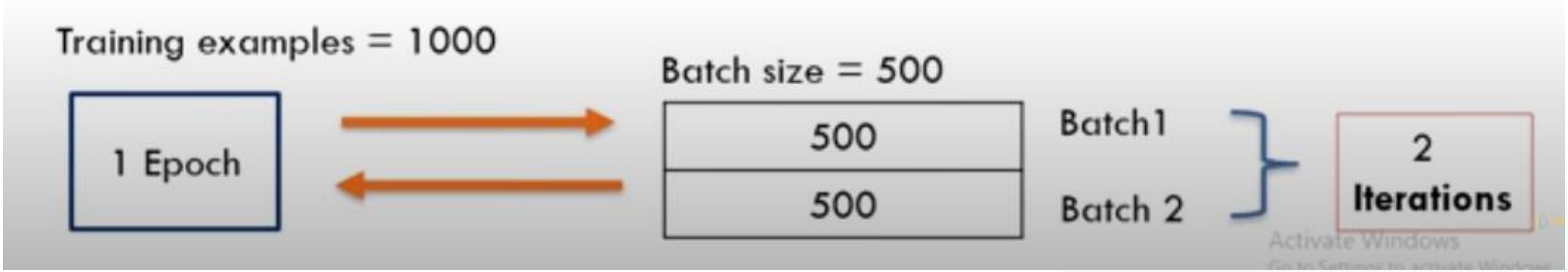
BATCHSIZE

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



DIFFERENCE BETWEEN BATCH & 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

```
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)
```

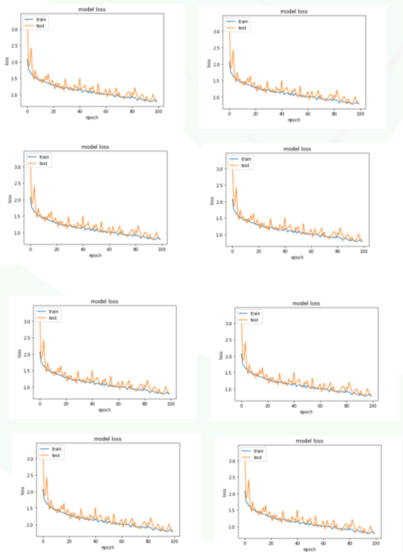
```
Epoch 1/300
22/22 [==============================] - 4s 2ms/step - loss: 256629.3281
Epoch 2/300
22/22 [==============================] - 0s 2ms/step - loss: 256489.2812
Epoch 3/300
22/22 [==============================] - 0s 3ms/step - loss: 256325.5469
```

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

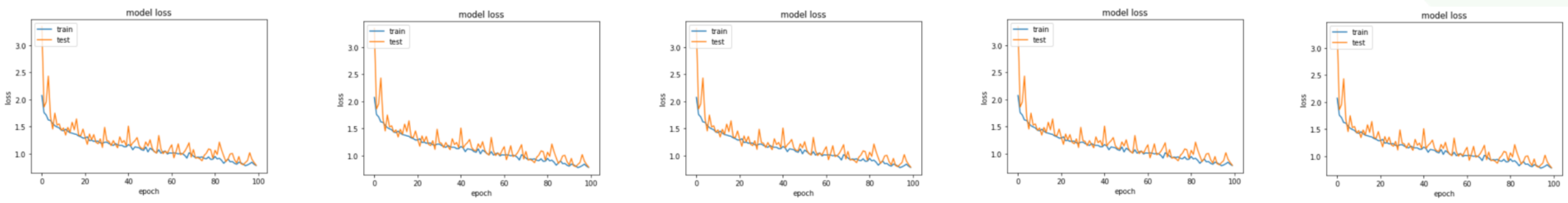
700 ÷ 32 =

21,875

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



iteration



DIFFERENCE BETWEEN BATCH & EPOCH

TECHPRO EDUCATION

Epoch :

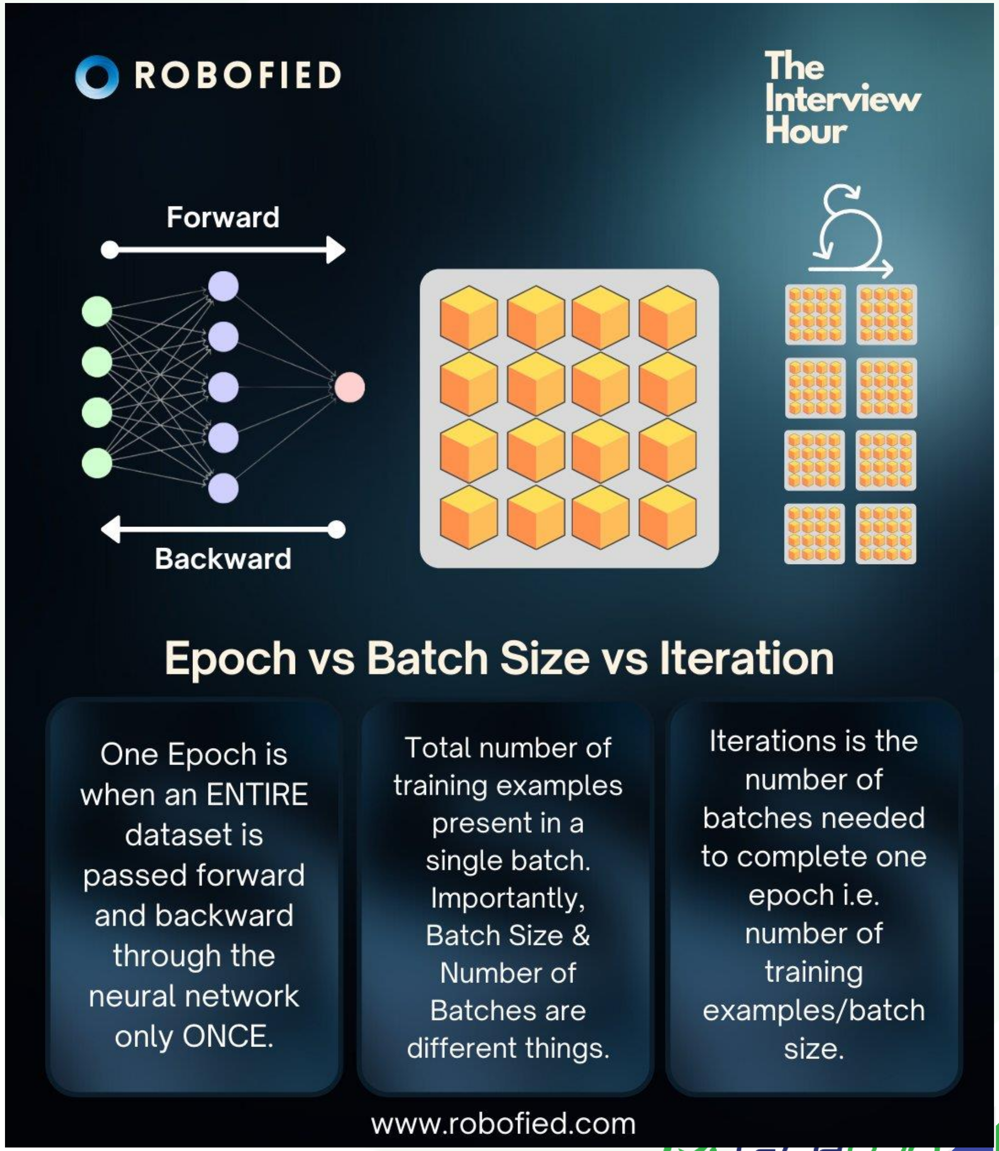
An Epoch represent one iteration over the entire dataset.

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.



TECHPRO EDUCATION

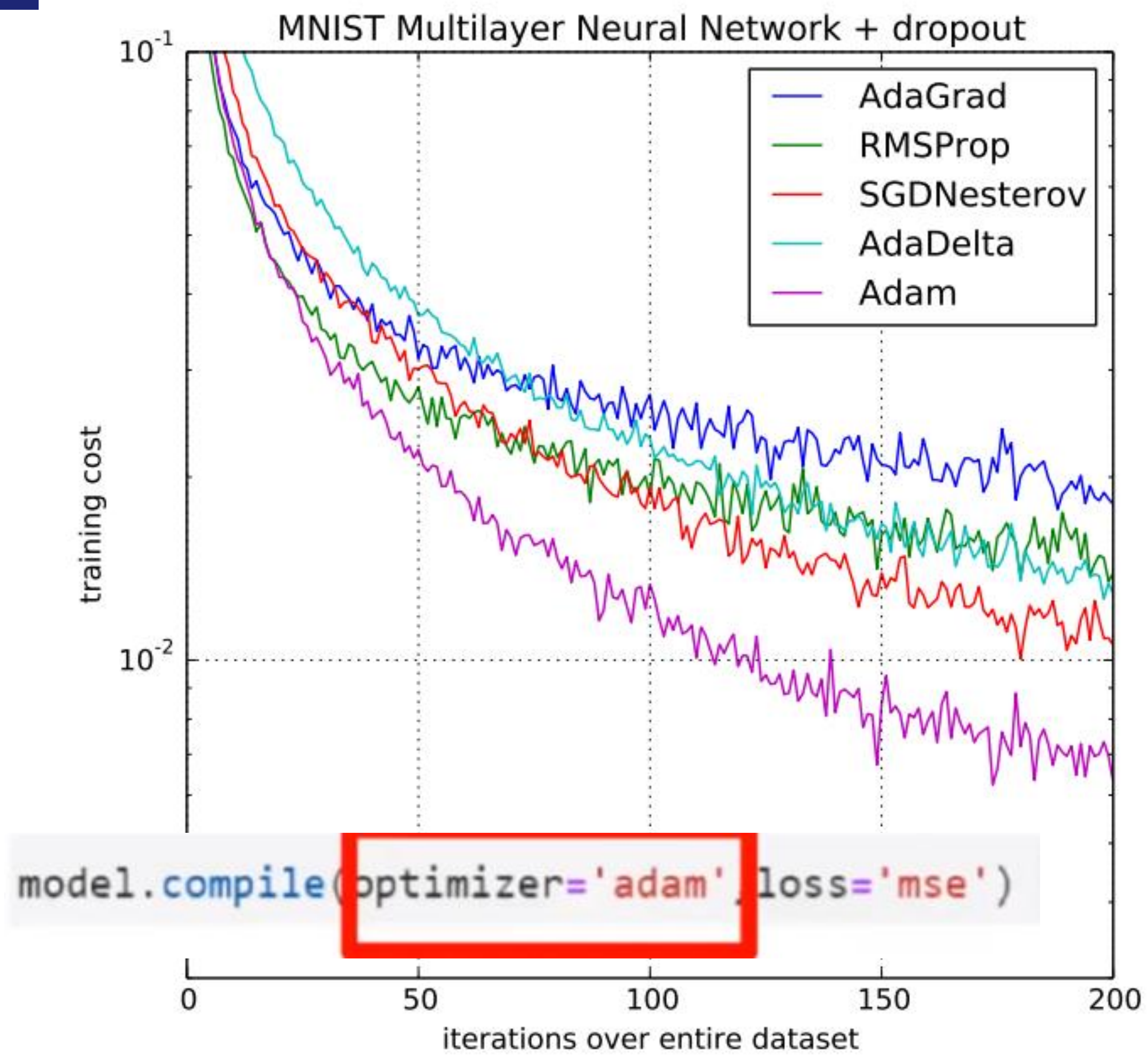
GRADIENT DESCENT

GRADIENT DESCENT

GRADIENT DESCENT

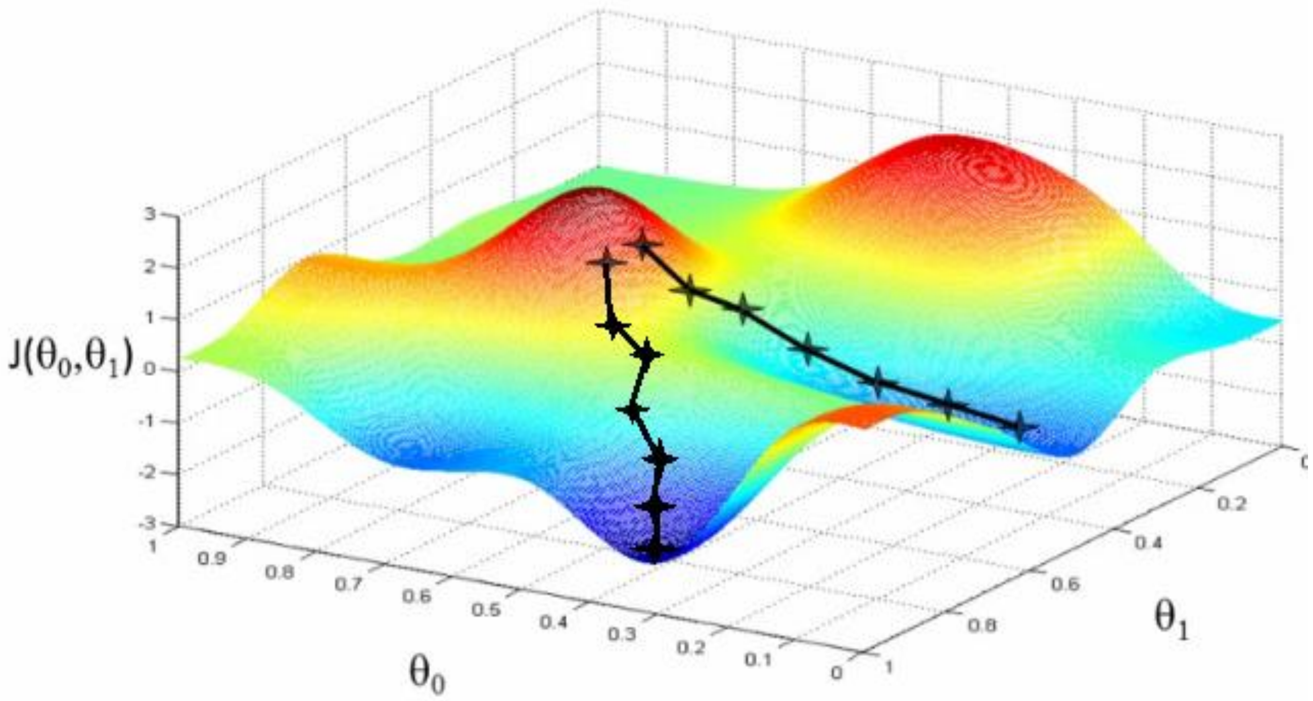
OPTIMIZER

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.

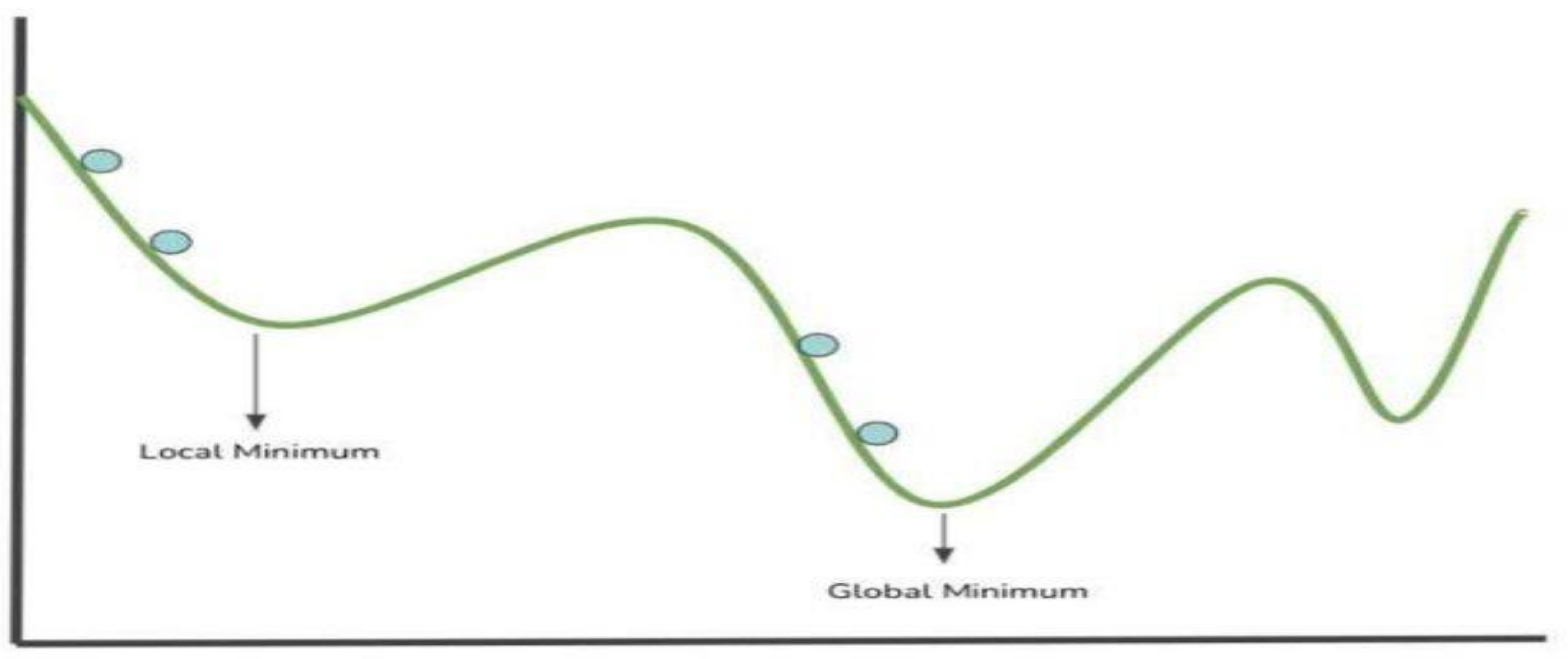


GRADIENT DESCENT

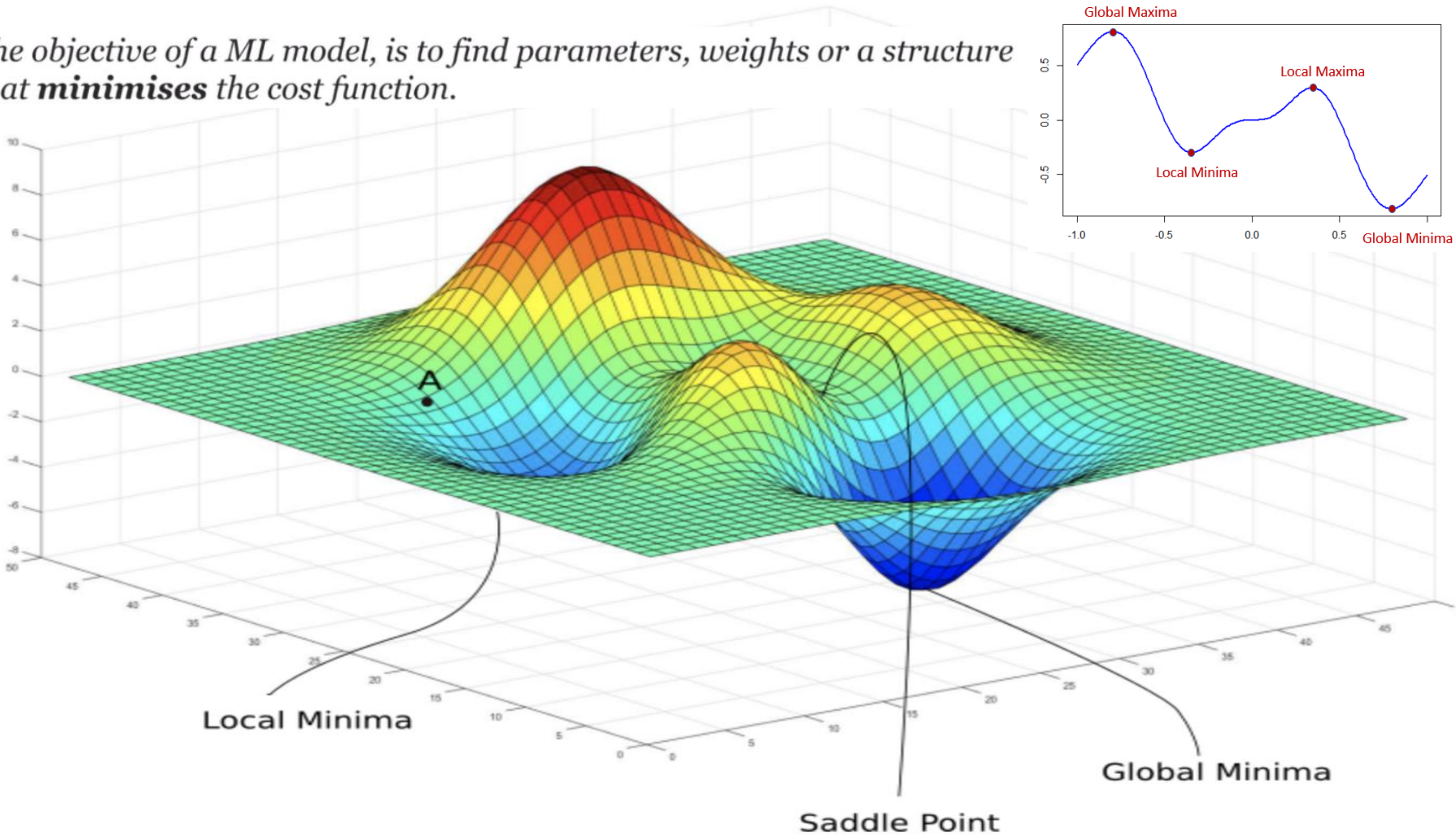
Gradient descent is the process of using gradients to find the minimum value of the cost function, while backpropagation is calculating those gradients by moving in a backward direction in the neural network.



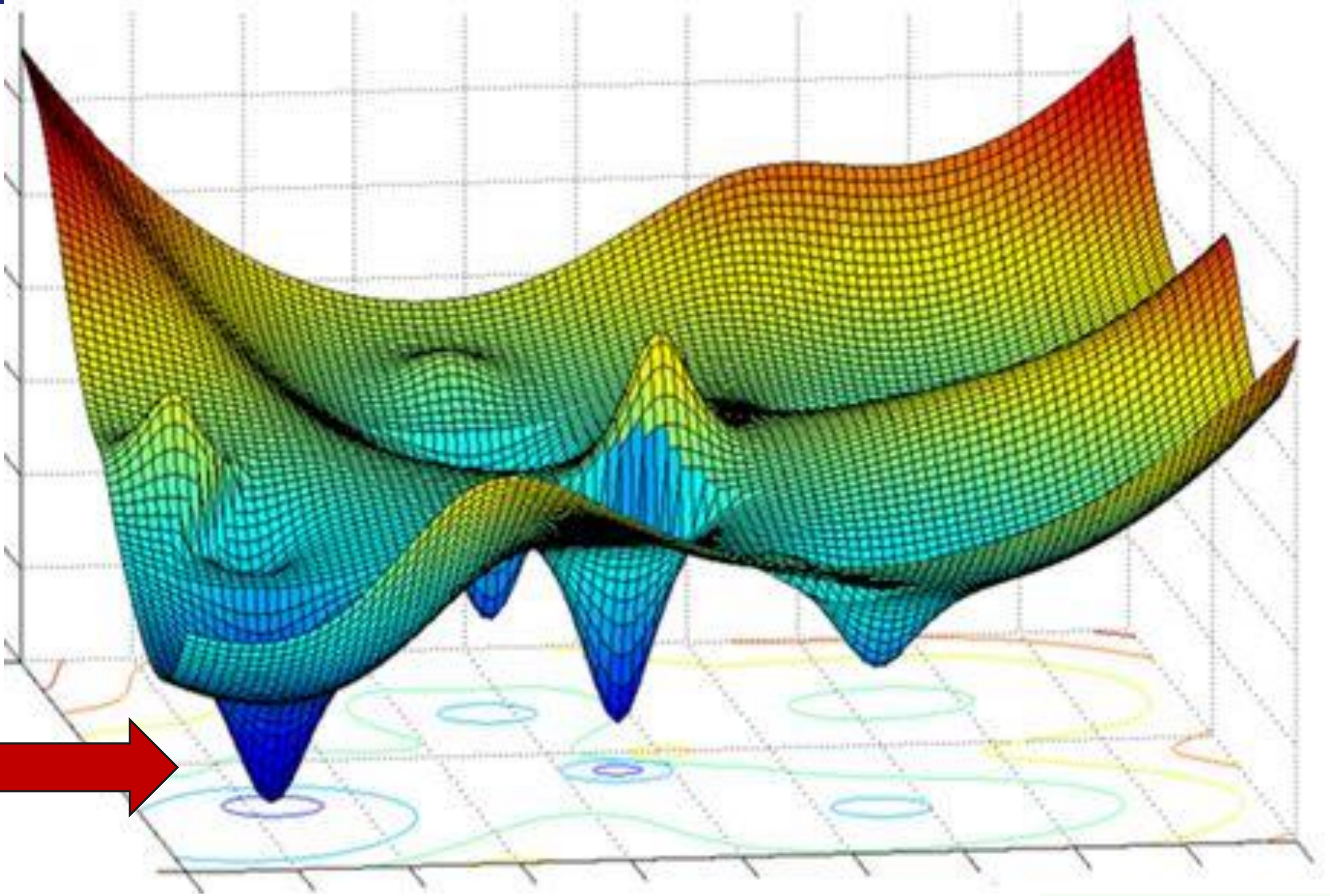
GRADIENT DESCENT

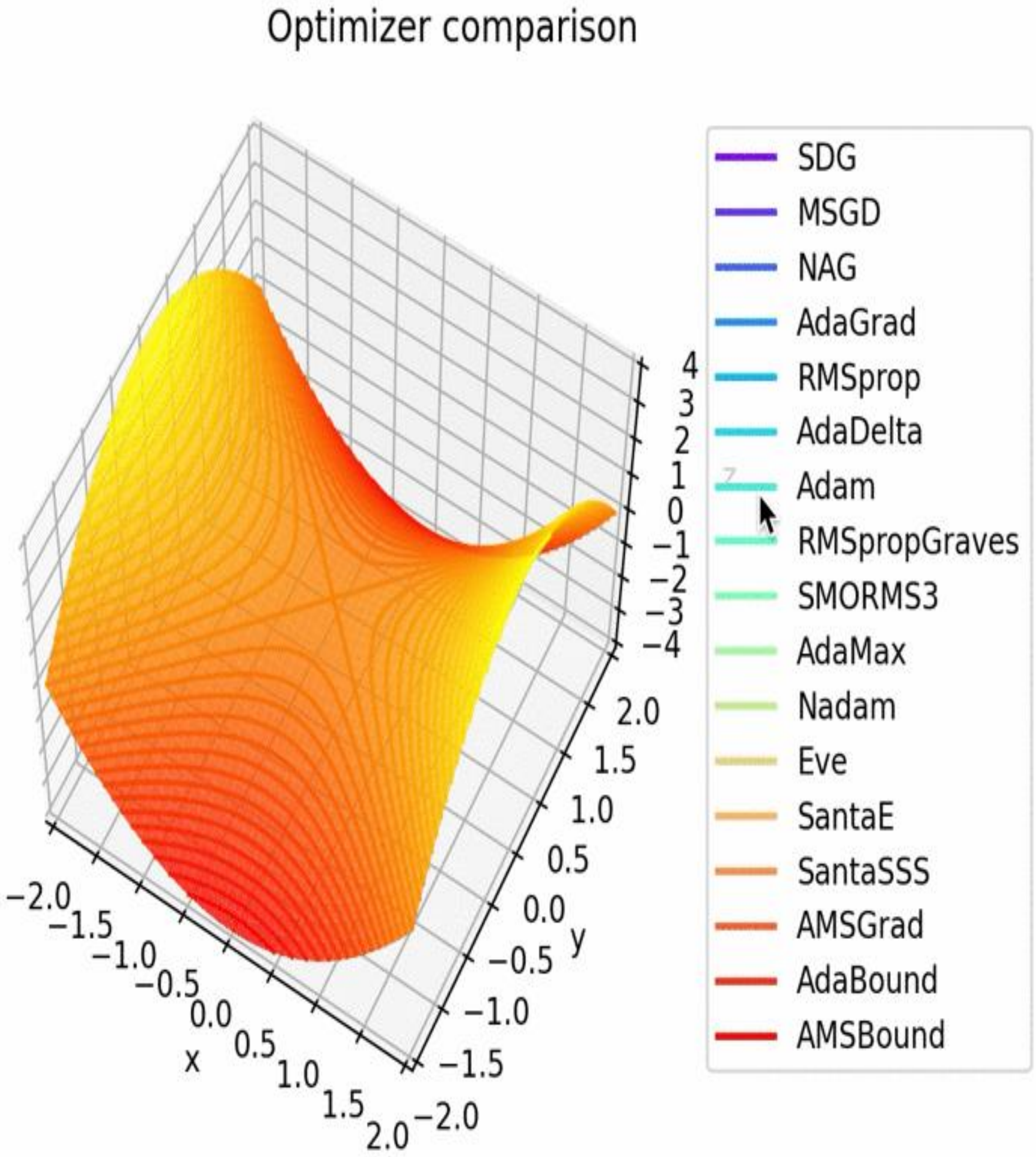


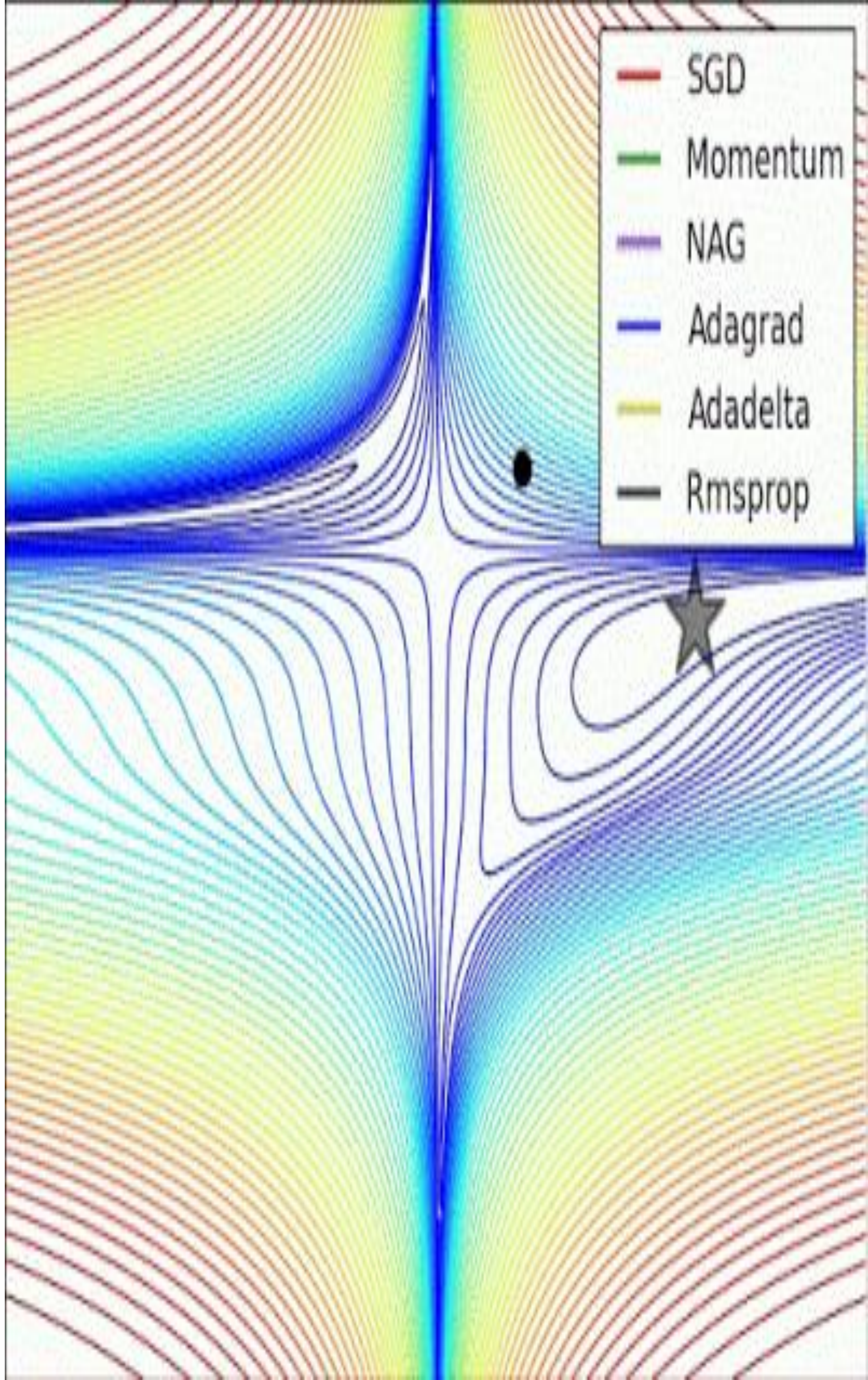
*The objective of a ML model, is to find parameters, weights or a structure that **minimises** the cost function.*



GRADIENT DESCENT

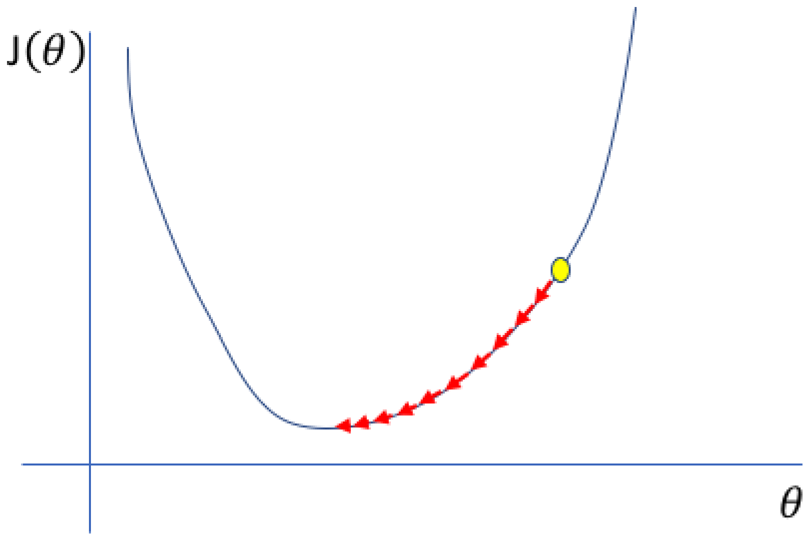



Optimizer comparison
SDG
MSGD
NAG
AdaGrad
RMSprop
AdaDelta
Adam
RMSpropGraves
SMORMS3
AdaMax
Nadam
Eve
SantaE
SantaSSS
AMSGrad
AdaBound
AMSBound
x
y
z
4
3
2
1
0
−1
−2
−3
−4
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

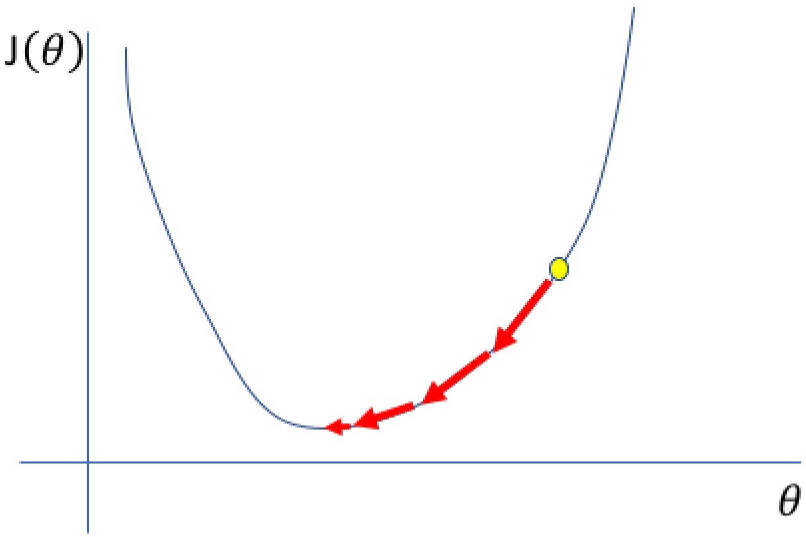
LEARNING RATE

Too low



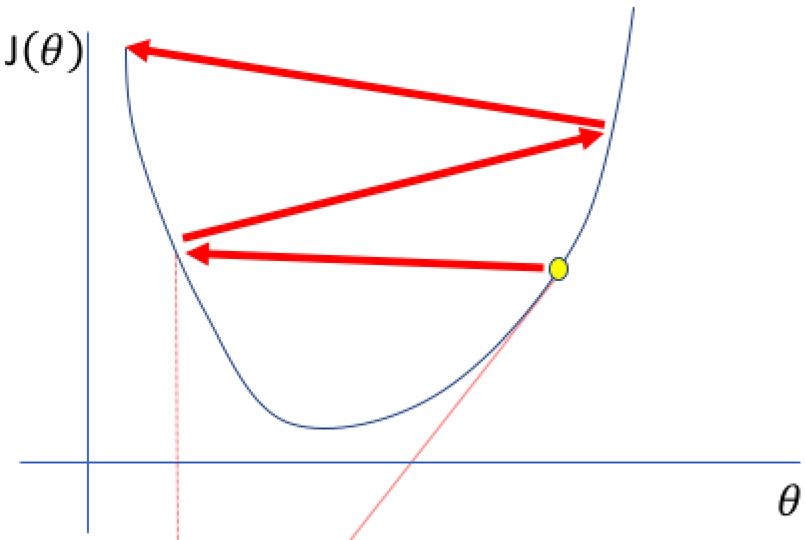
A small learning rate requires many updates before reaching the minimum point

Just right



The optimal learning rate swiftly reaches the minimum point

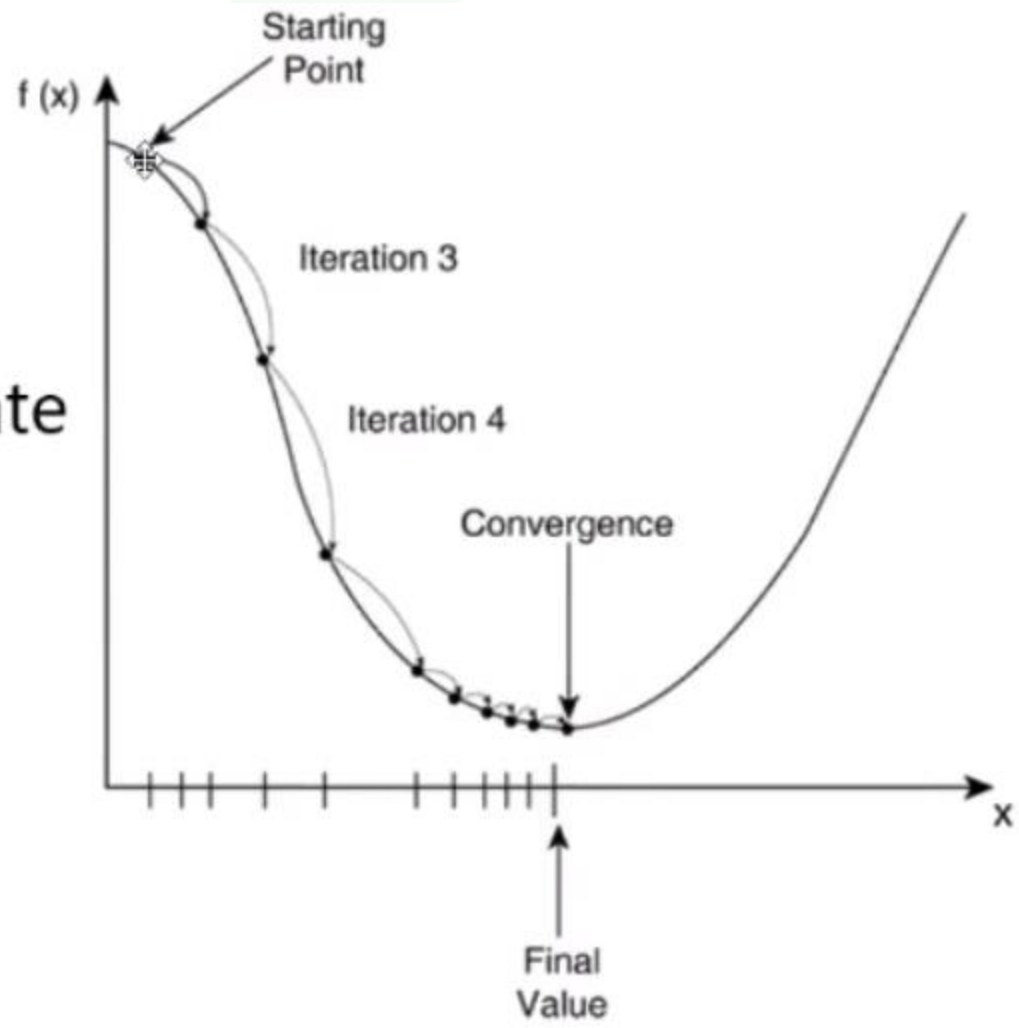
Too high



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

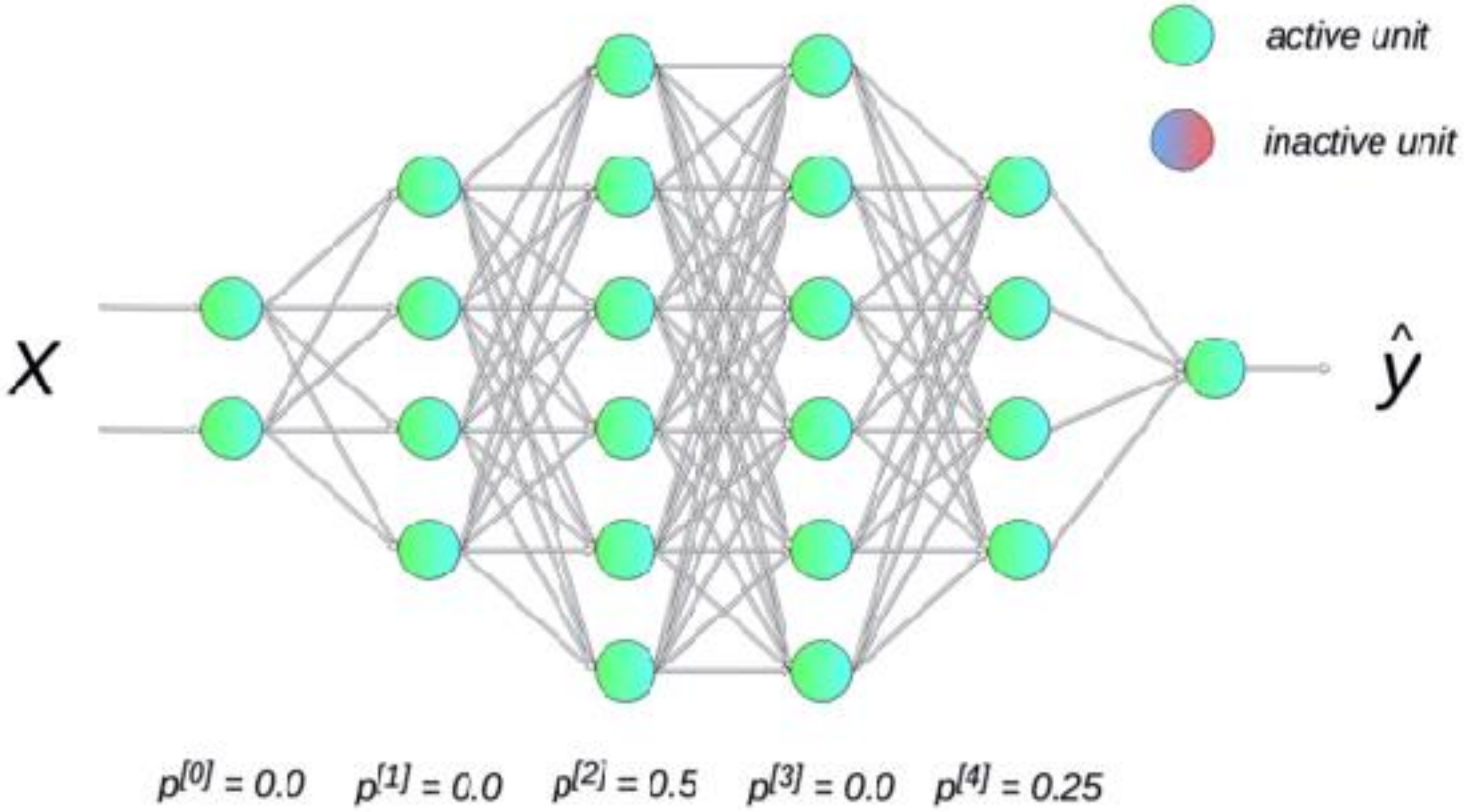
LEARNING RATE

Step Size = slope x learning rate

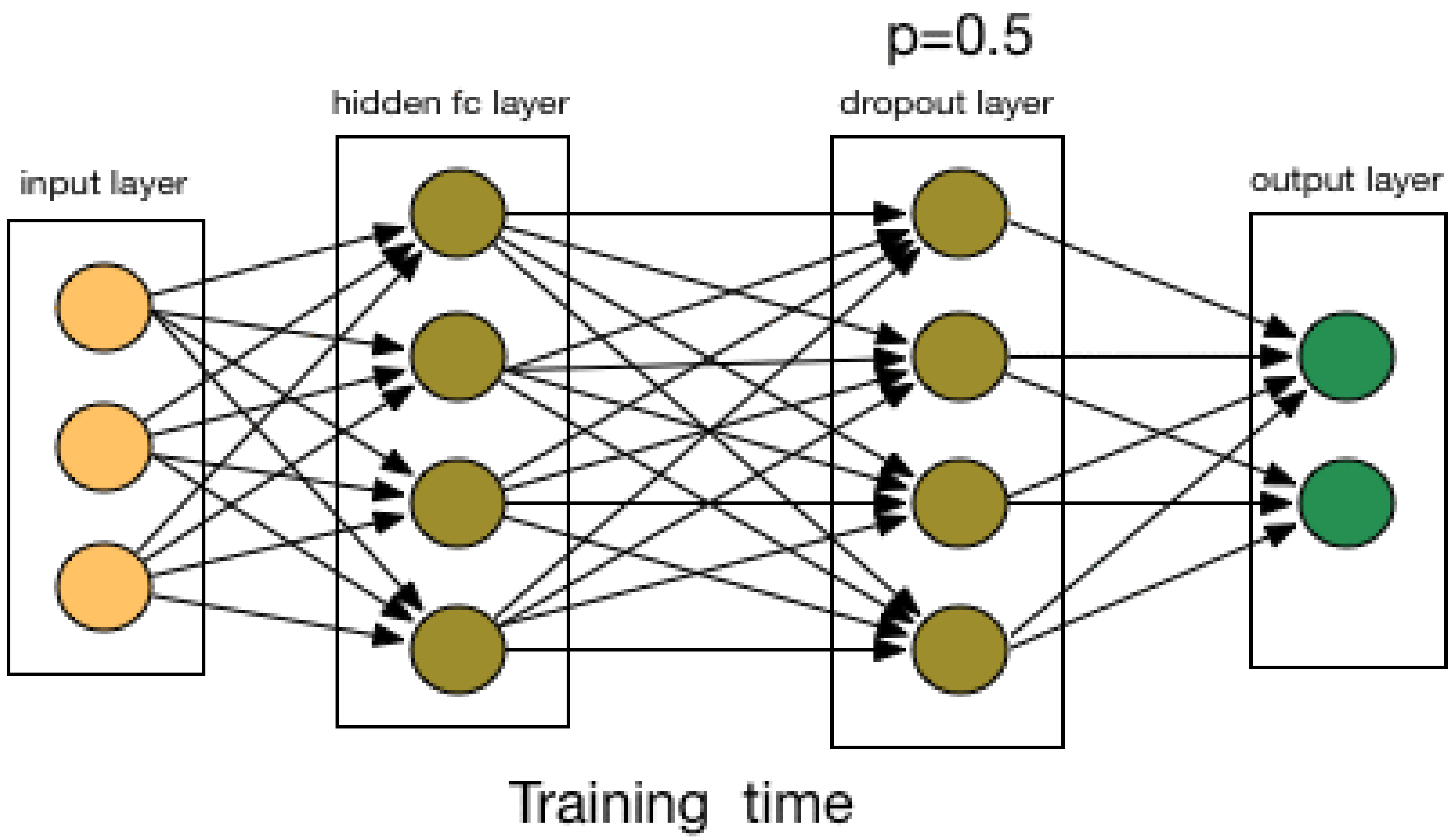


REGULARIZATION:
- DROPOUT L1
- EARLYSTOPPING L2
- BATCH NORMALIZATION

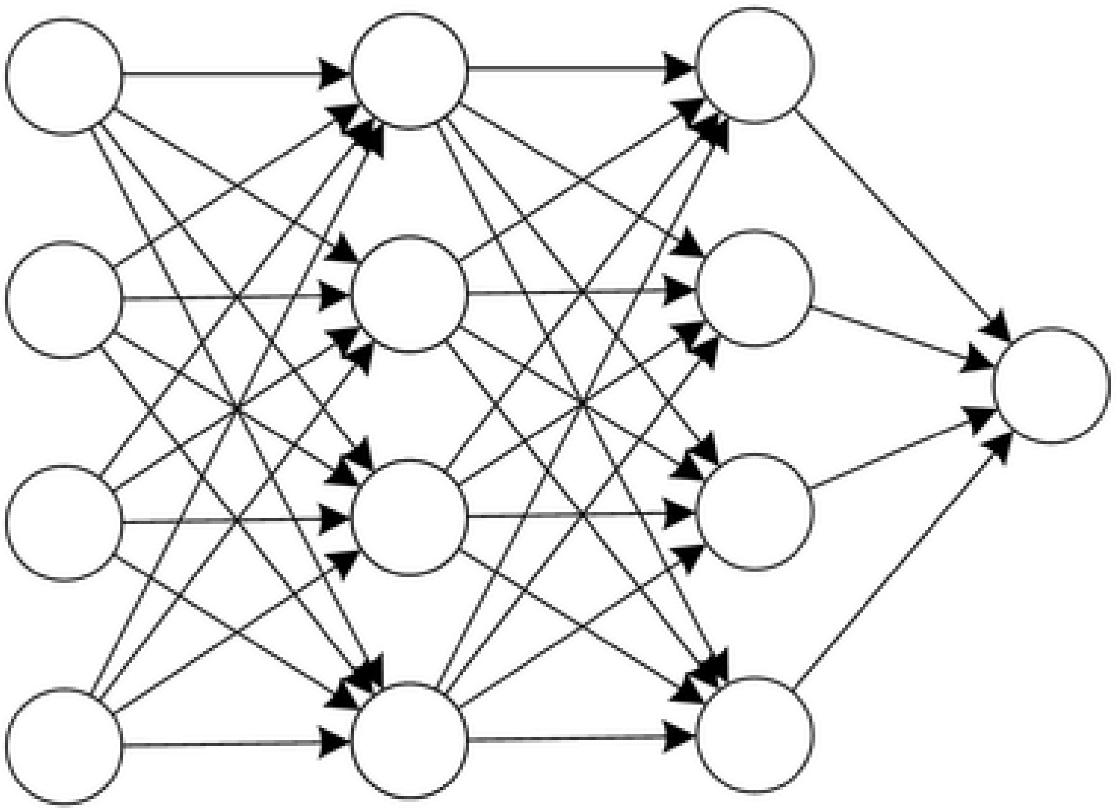
DROPOUT



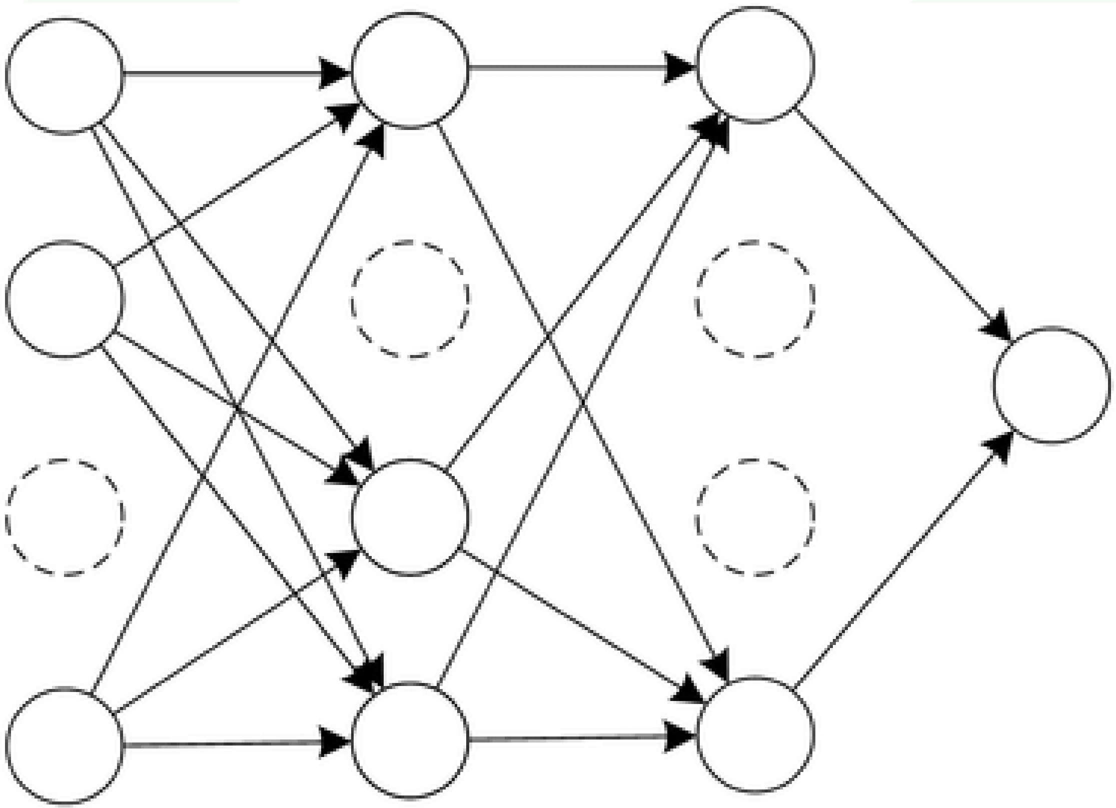
DROPOUT


p=0.5
hidden fc layer
dropout layer
input layer
output layer
Training time

DROPOUT



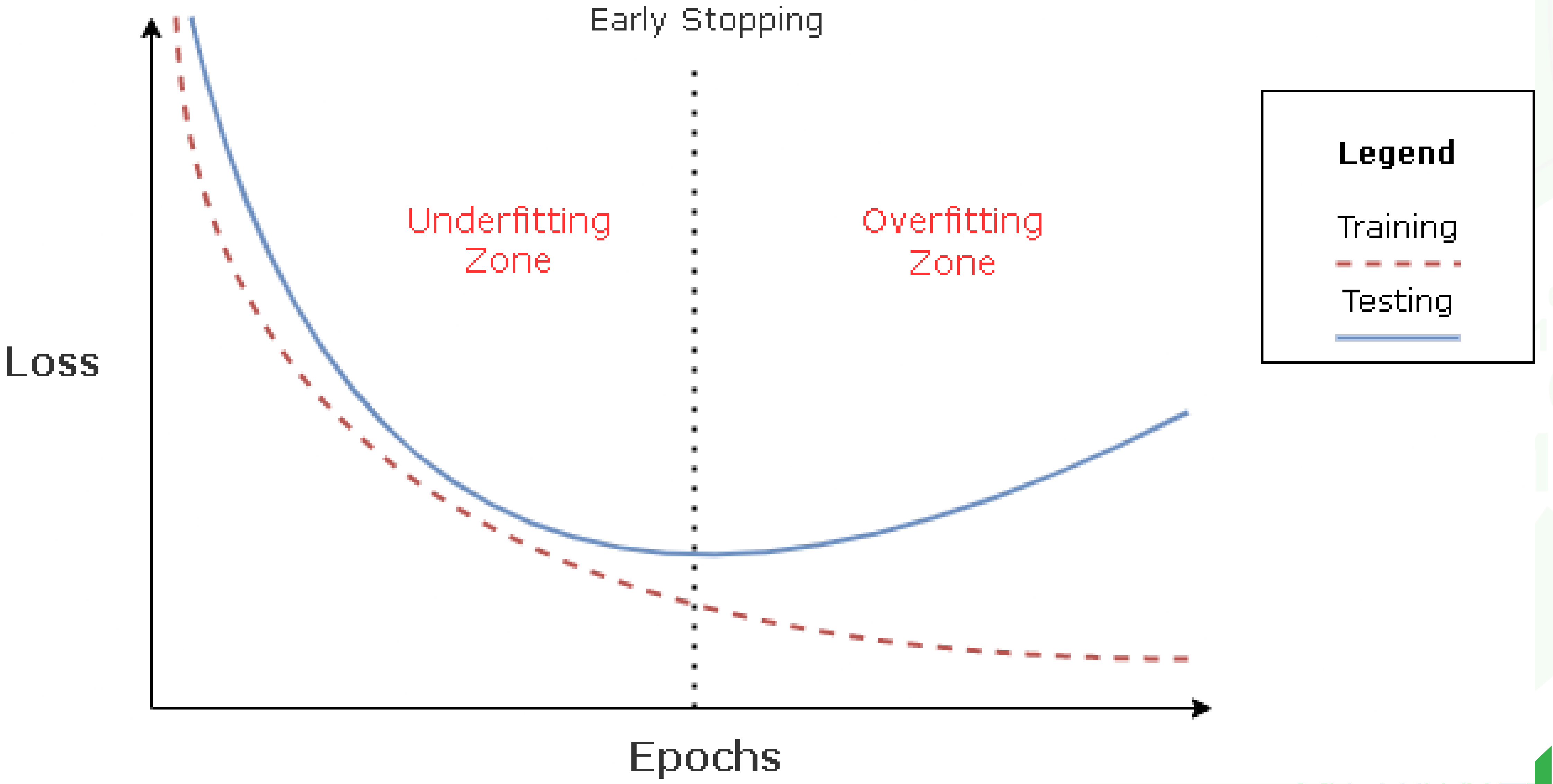
(a) Standard Neural Network



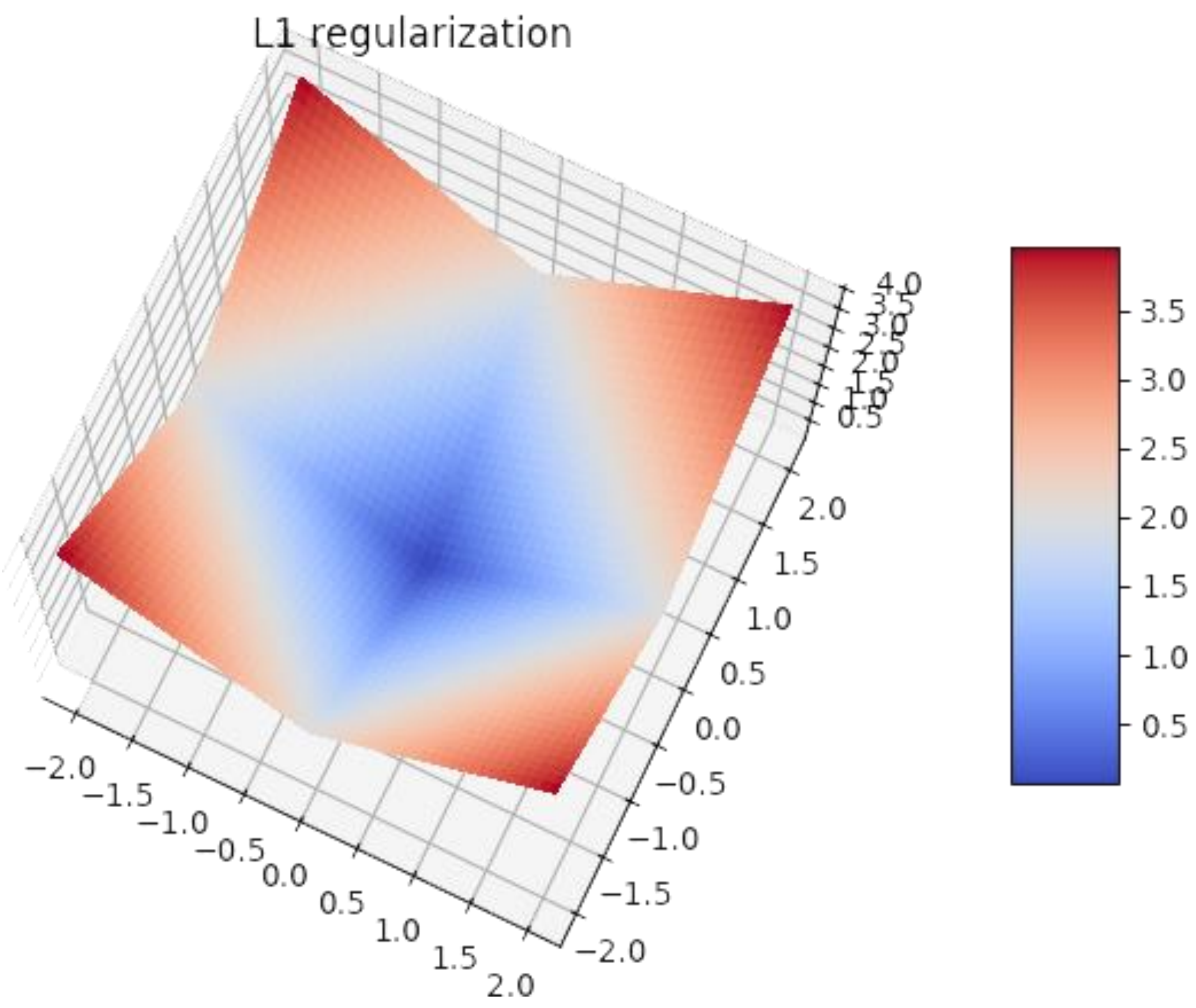
(b) Network after Dropout

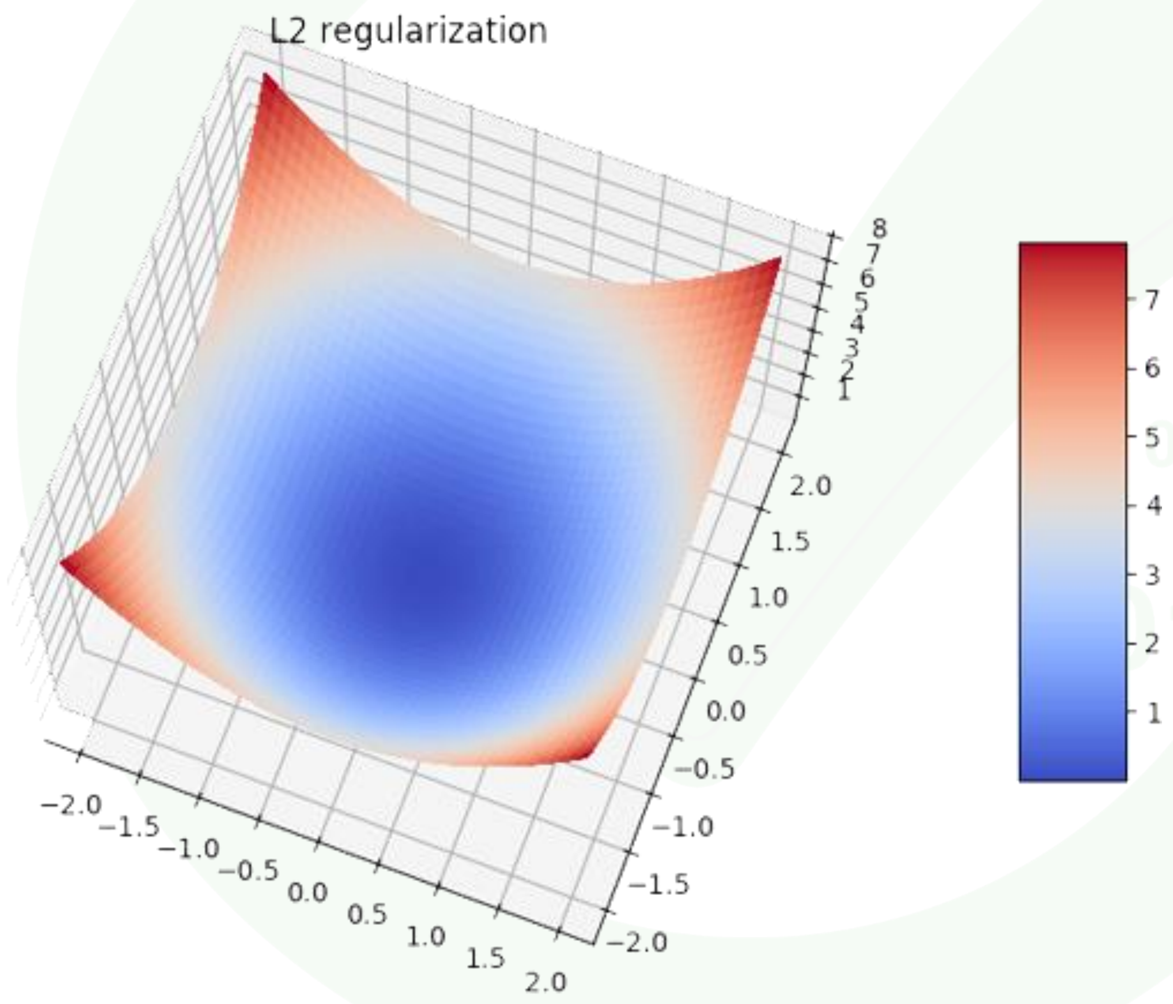
EARLYSTOPPING

EARLYSTOPPING



L1 & L2


L1 regularization
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


L2 regularization
8
7
6
5
4
3
2
1
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

L1 & L2

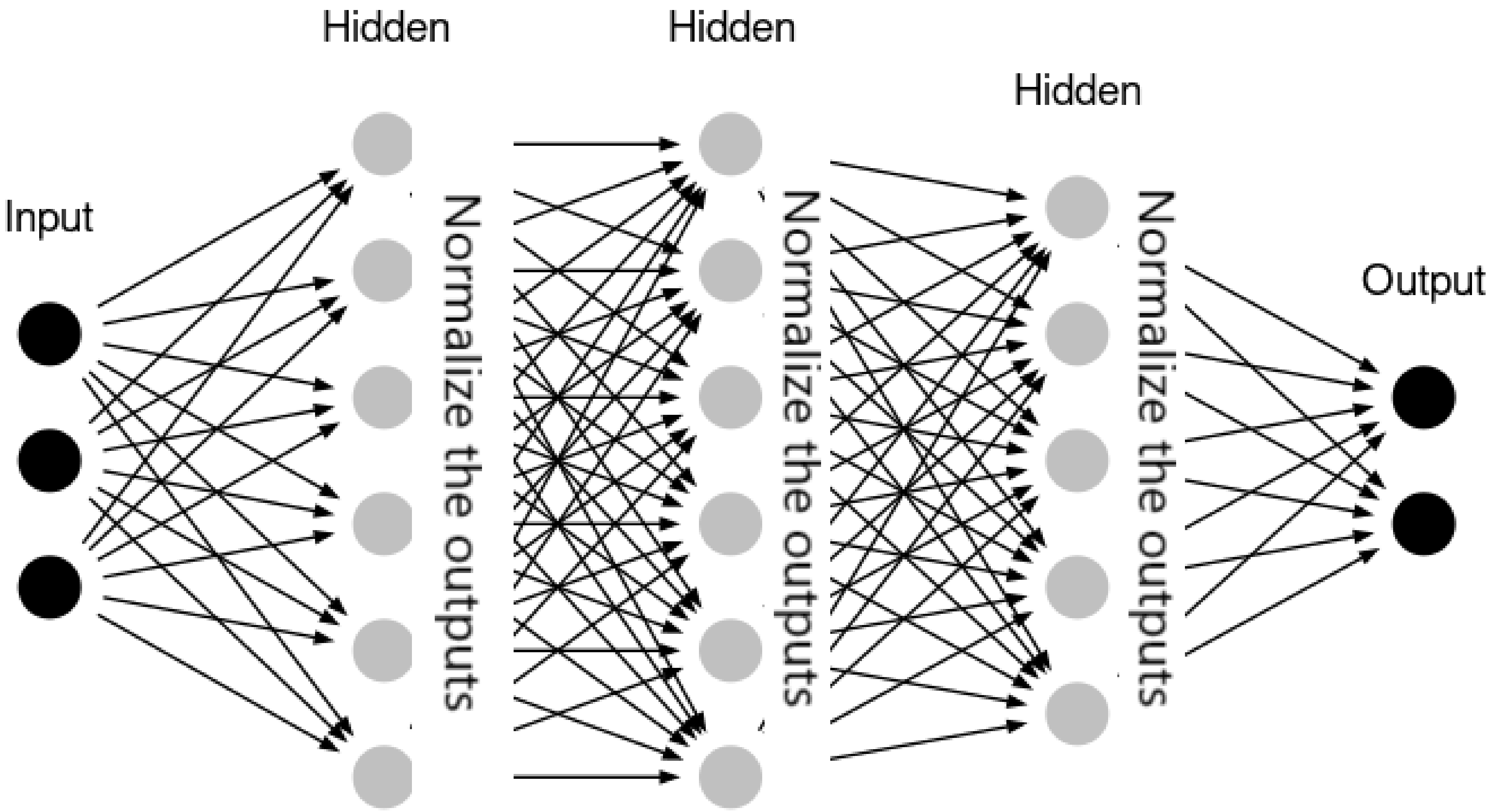
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import regularizers


tf.random.set_seed(seed)

# create the model
model = Sequential()

# Add the layers and use only apply only L1 regularization
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))  # regylarization didn't apply
```

L1 & L2

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

tf.random.set_seed(seed)

# create the model
model = Sequential()

# add the layers and apply only L2 regularization
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))  # regularization didn't apply

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

BATCH NORMALIZATION

BATCH NORMALIZATION

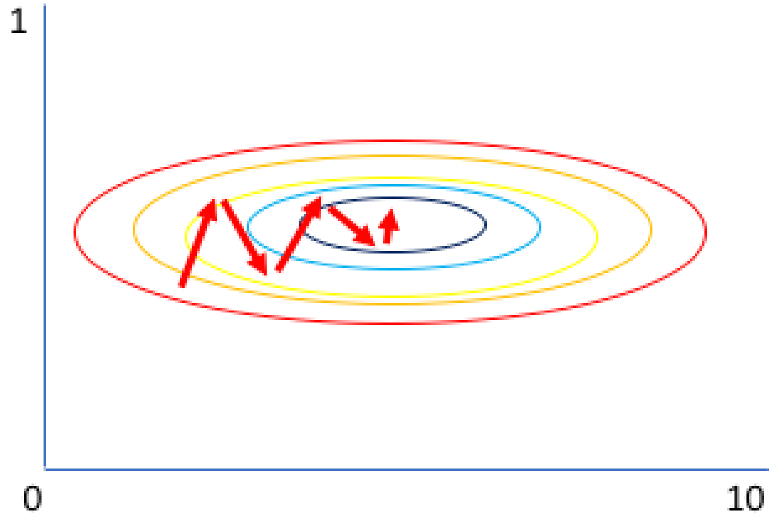


BATCH

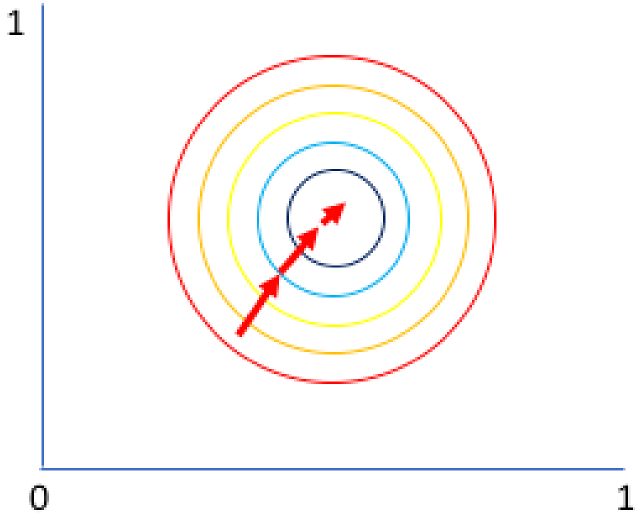
Kullanım Durumu	Açıklama
Derin Sinir Ağları (DNN)	DNN'lerde Batch Normalization, daha hızlı ve istikrarlı bir eğitim süreci sağlayabilir ve aşırı uyumu azaltabilir.
Evrişimli Sinir Ağları (CNN)	CNN'lerde özellikle büyük ve karmaşık modellerde kullanılabilir. Evreli ve tam bağlantılı katmanlar arasında 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?



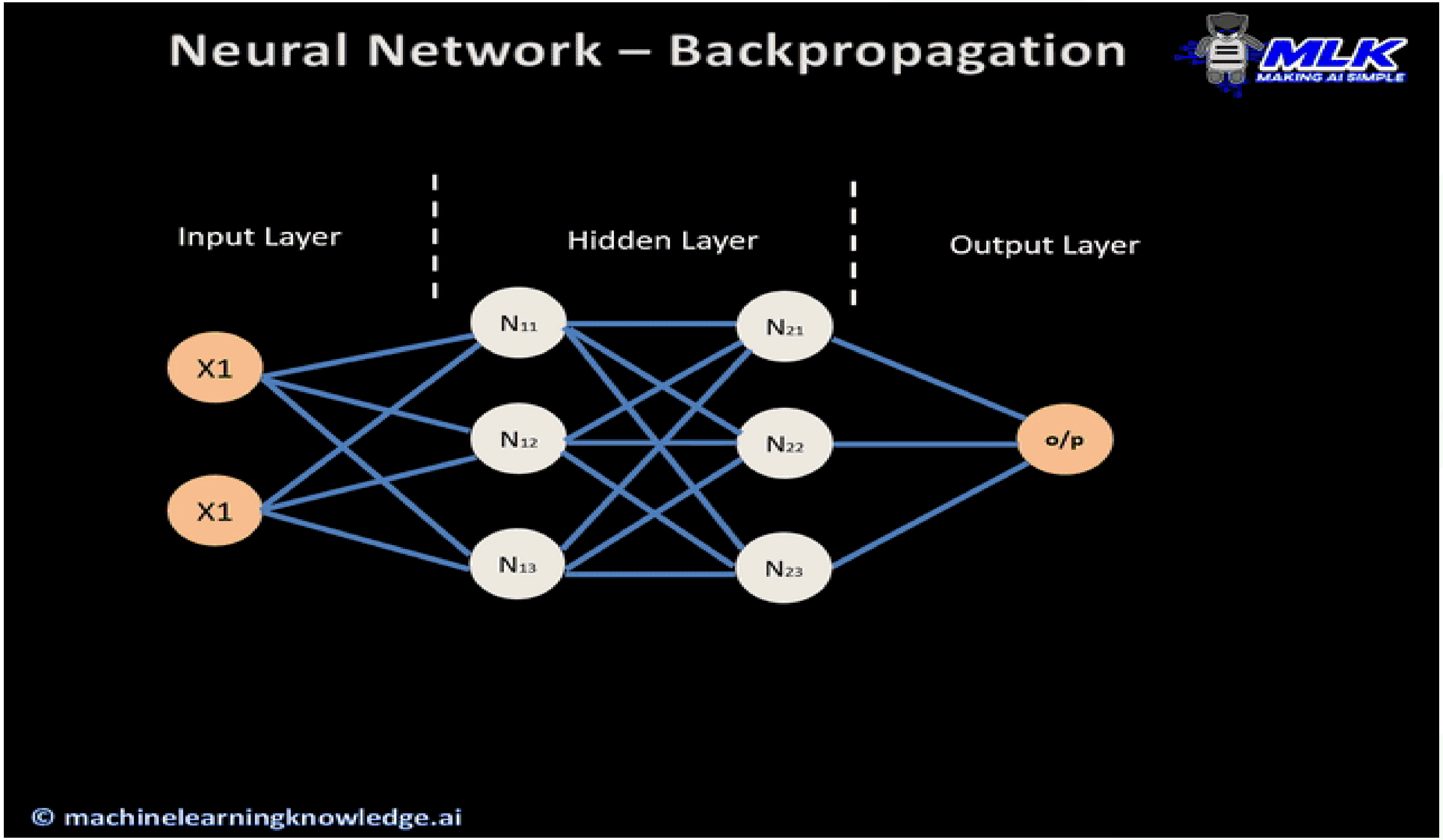
Gradient of larger parameter
dominates the update



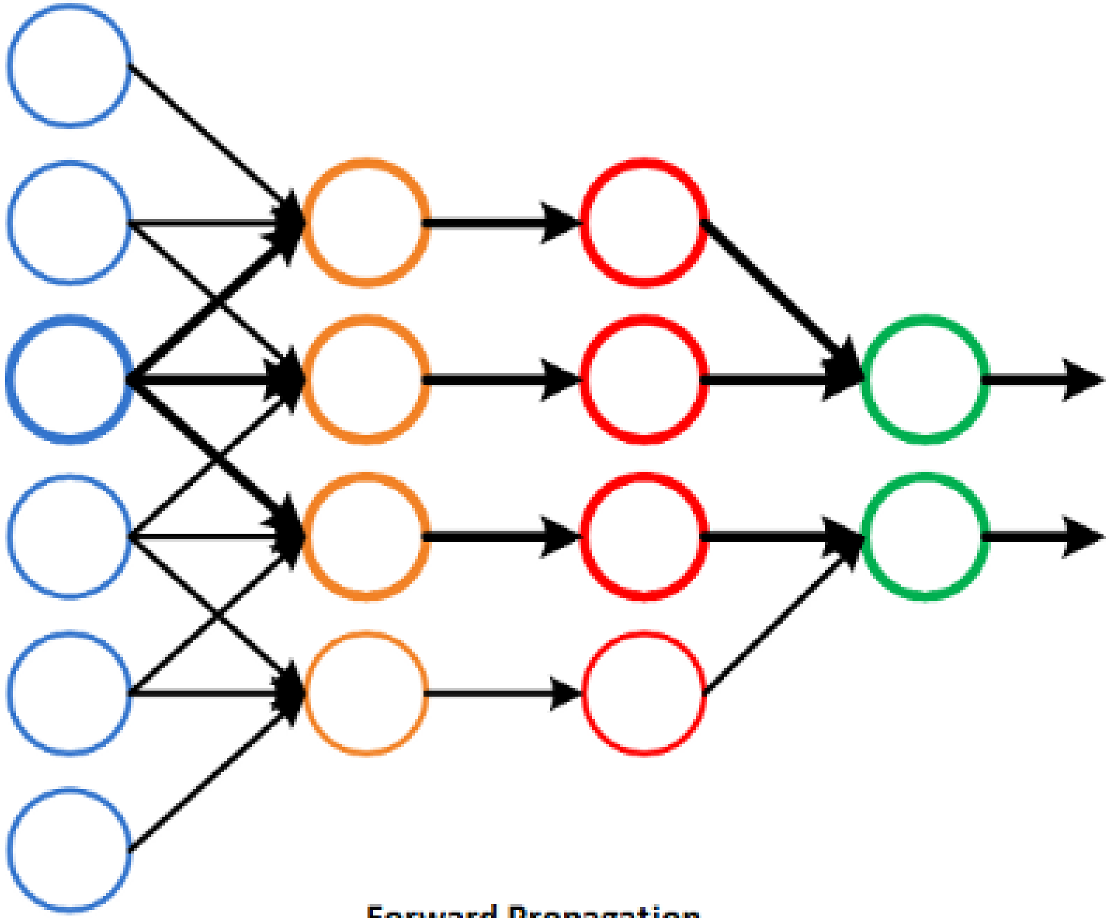
Both parameters can be
updated in equal proportions

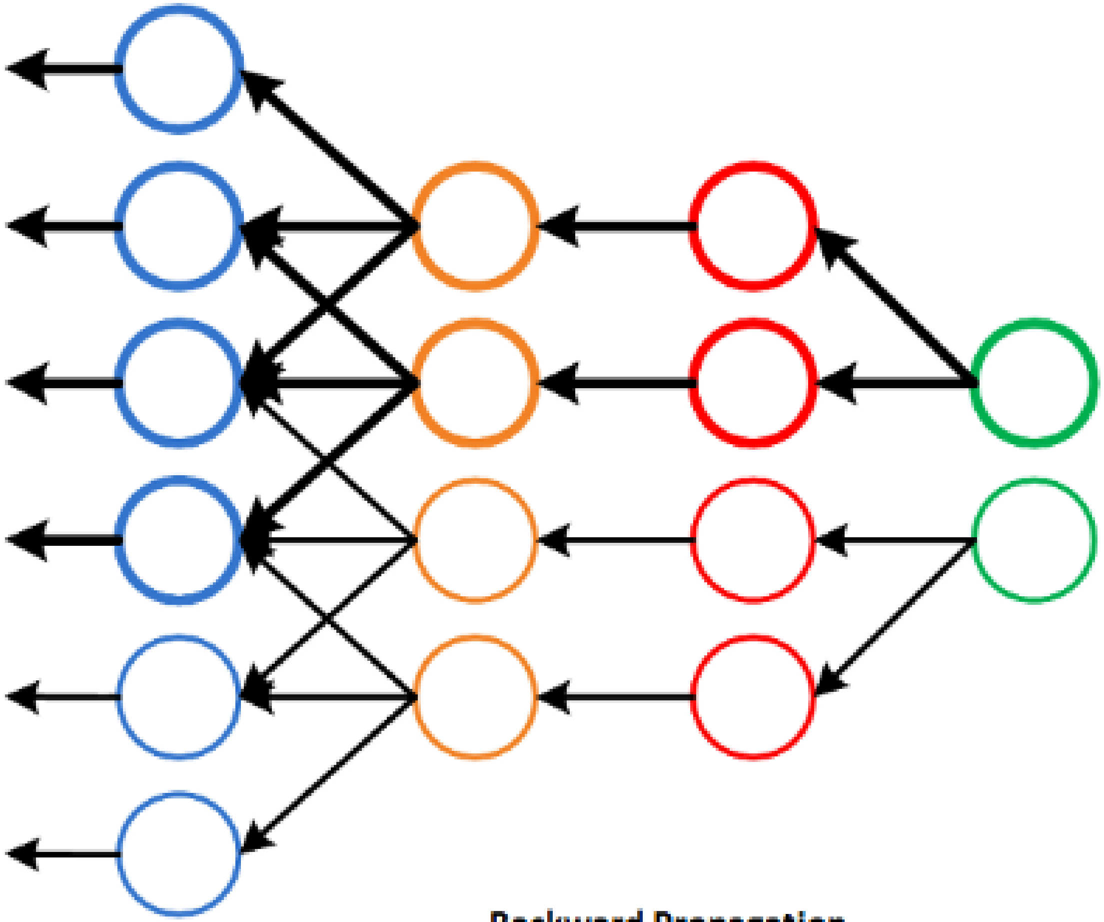
BACKPROPAGATION

BACK PROPAGATION

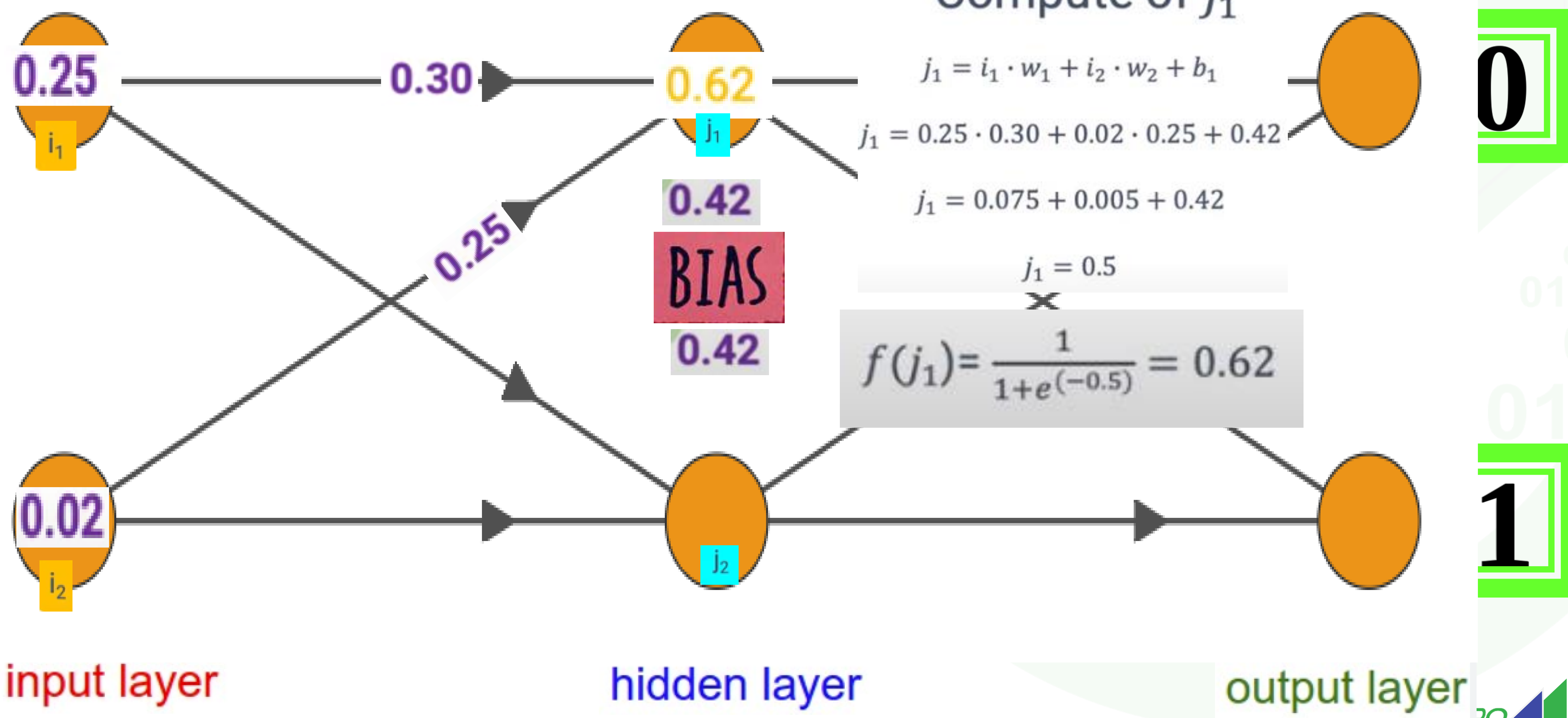


BACK PROPAGATION





FORWARD PROPAGATION



Compute of j_1

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

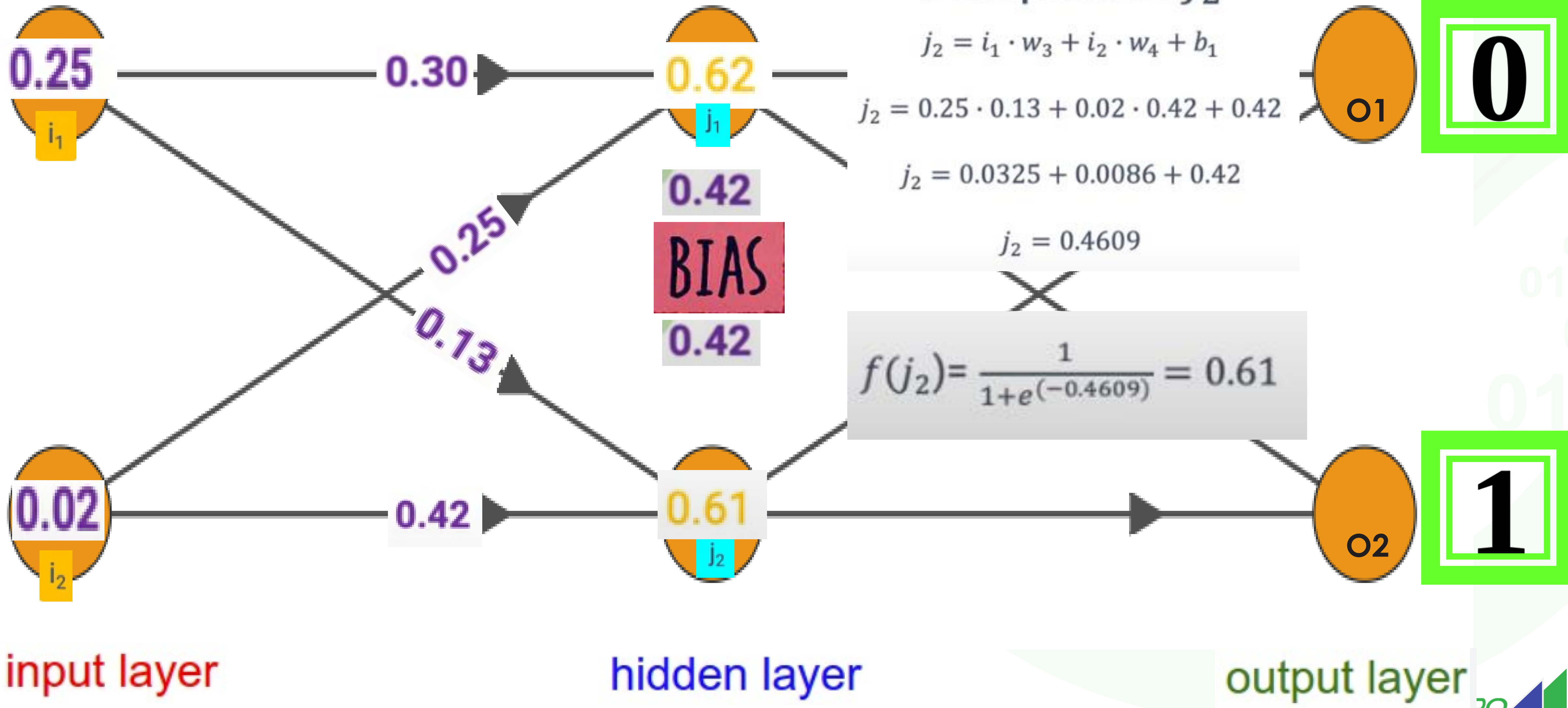
$$j_1 = 0.25 \cdot 0.30 + 0.02 \cdot 0.25 + 0.42$$

$$j_1 = 0.075 + 0.005 + 0.42$$

$$j_1 = 0.5$$

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

FORWARD PROPAGATION



Compute of j_2

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

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

$$j_2 = 0.0325 + 0.0086 + 0.42$$

$$j_2 = 0.4609$$

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

FORWARD PROPAGATION

Compute of o_1

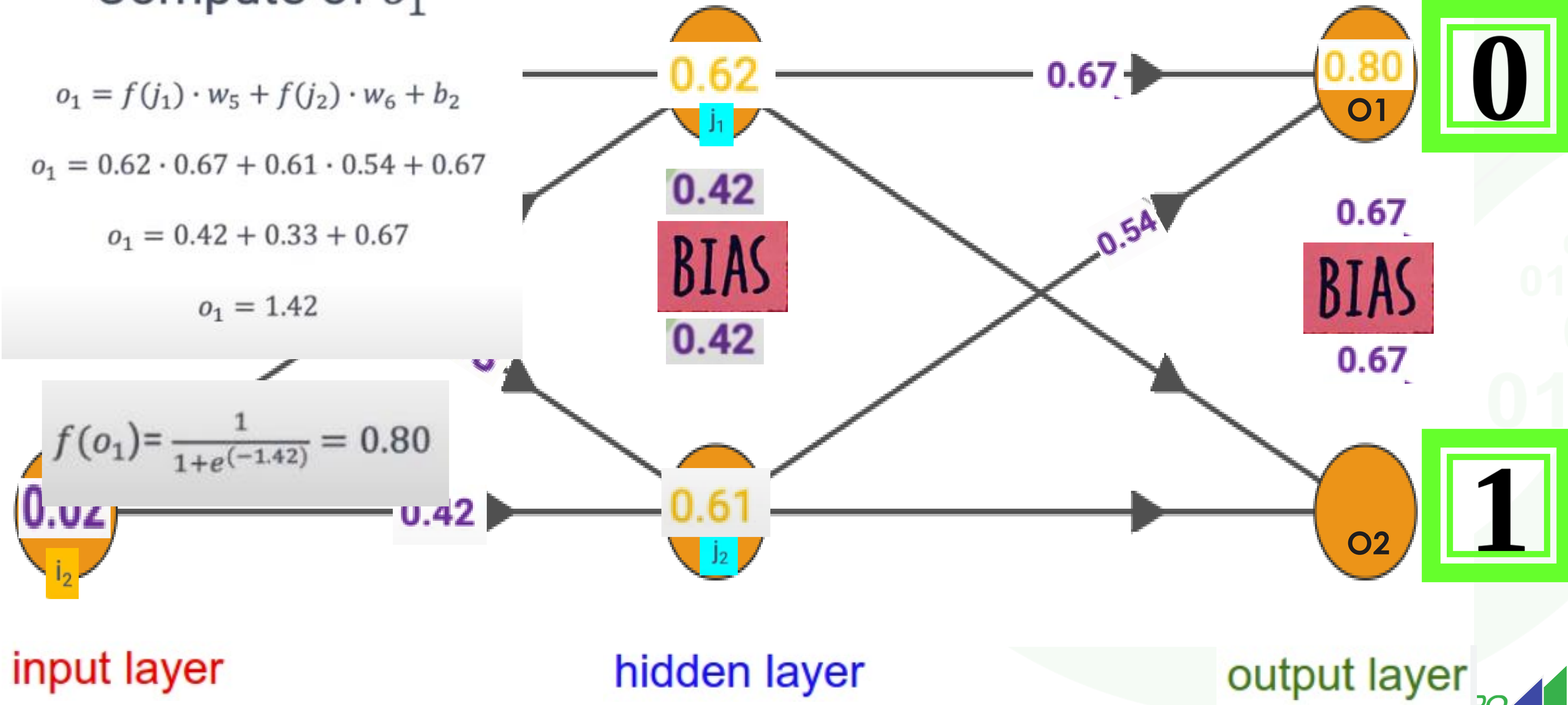
$$o_1 = f(j_1) \cdot w_5 + f(j_2) \cdot w_6 + b_2$$

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

$$o_1 = 0.42 + 0.33 + 0.67$$

$$o_1 = 1.42$$

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



FORWARD PROPAGATION

Compute of o_2

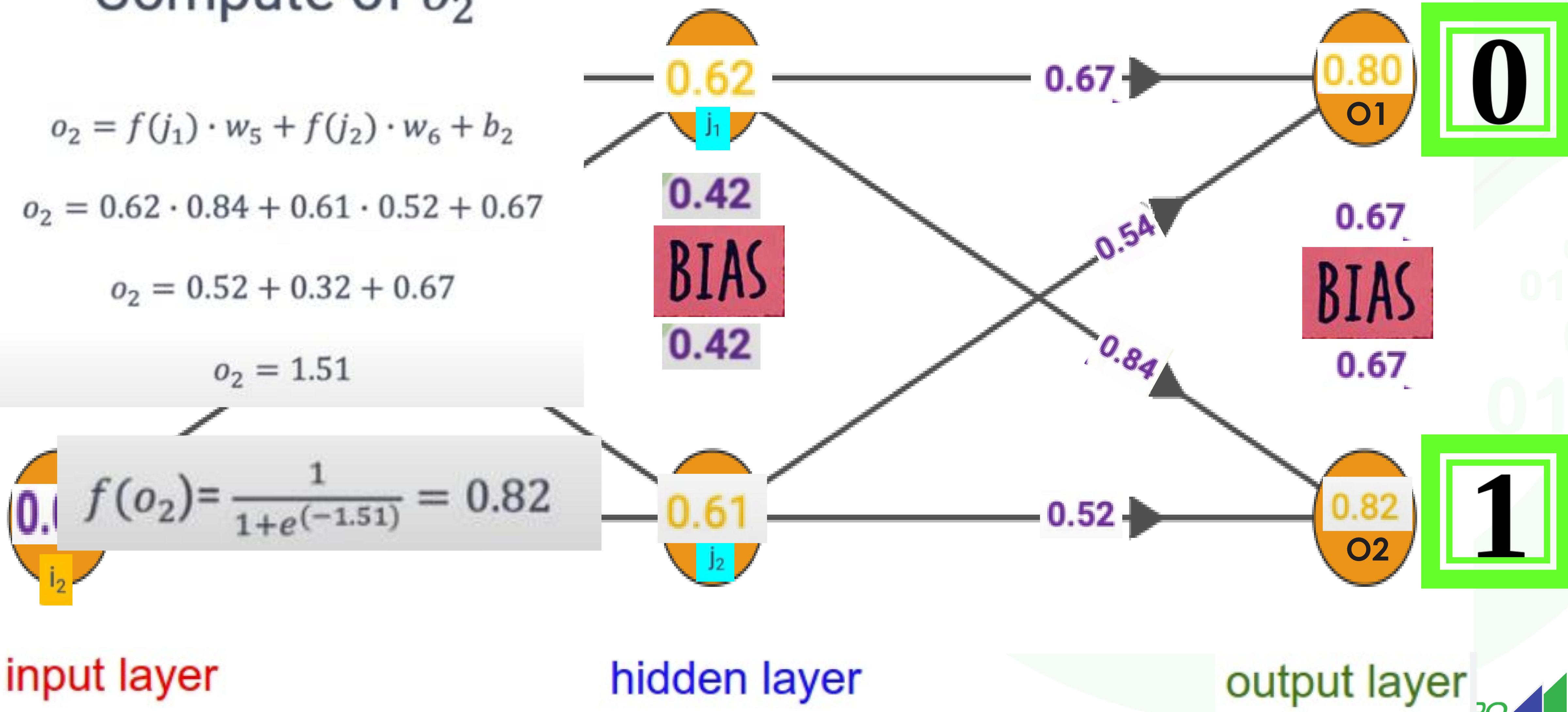
$$o_2 = f(j_1) \cdot w_5 + f(j_2) \cdot w_6 + b_2$$

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

$$o_2 = 0.52 + 0.32 + 0.67$$

$$o_2 = 1.51$$

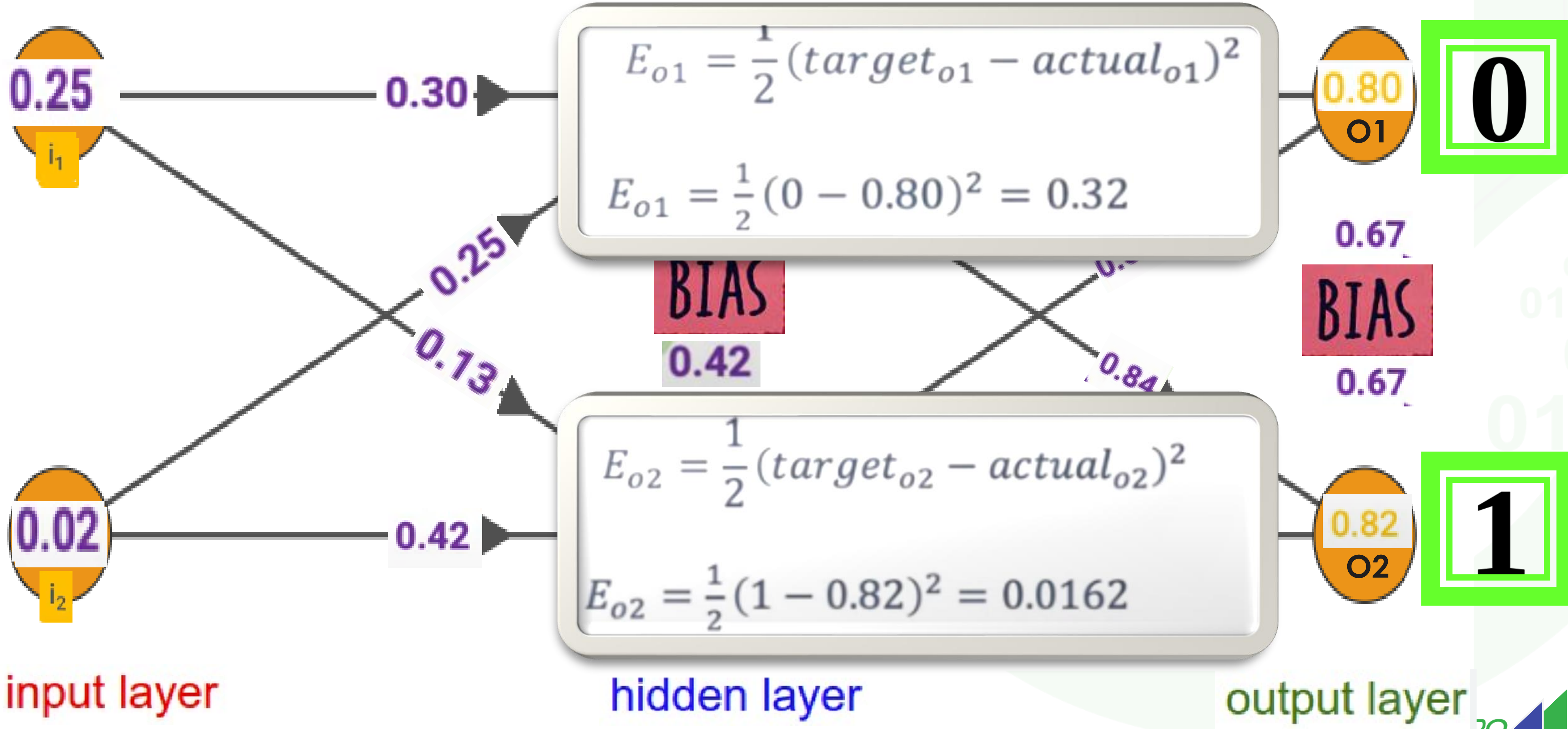
$$f(o_2) = \frac{1}{1+e^{(-1.51)}} = 0.82$$



Calculate the error

For each neuron:

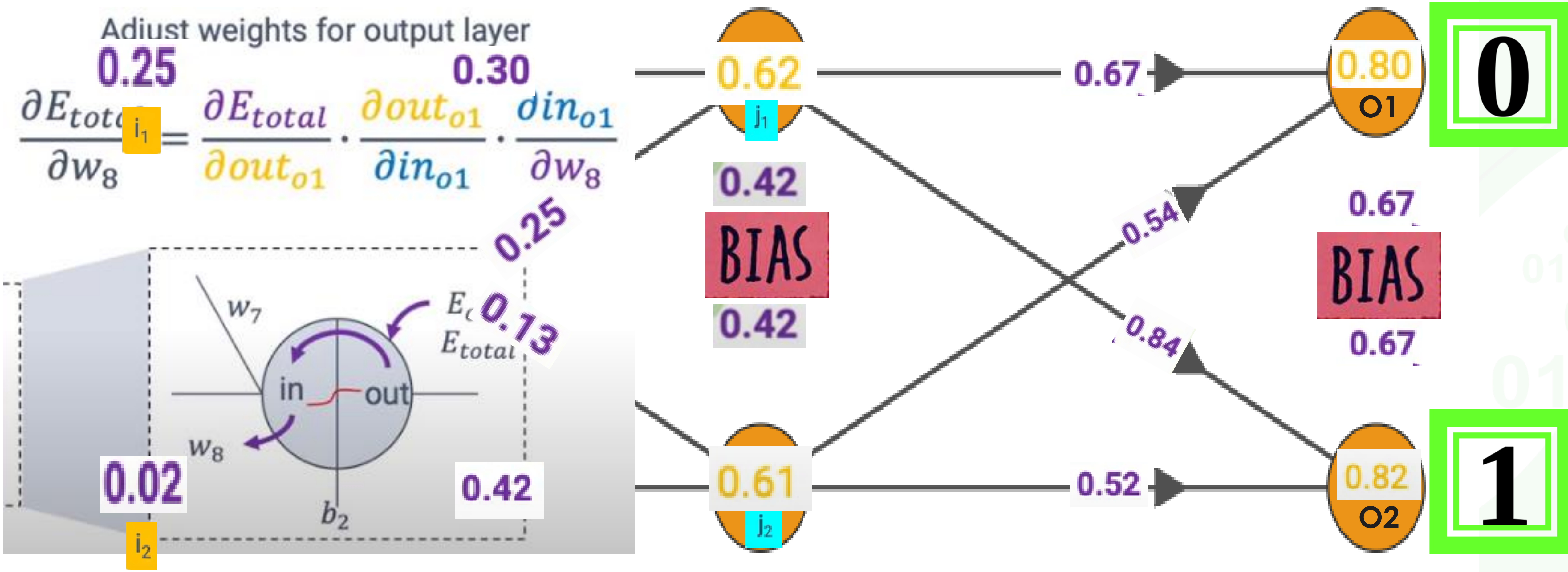
$$E_{total} = \sum \frac{1}{2}(target - actual)^2$$



BACK PROPAGATION

Adjust weights for output layer

$$\underset{0.25}{\frac{\partial E_{total}}{\partial w_8}} = \frac{\partial E_{total}}{\partial out_{o1}} \cdot \frac{\partial out_{o1}}{\partial in_{o1}} \cdot \overset{0.30}{\frac{\partial in_{o1}}{\partial w_8}}$$



input layer

hidden layer

output layer

BATCK PROPAGATION

δo2= -(O2-O2)otputO2(1-outputO2)

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

-0.18 * 0.82 * 0.18

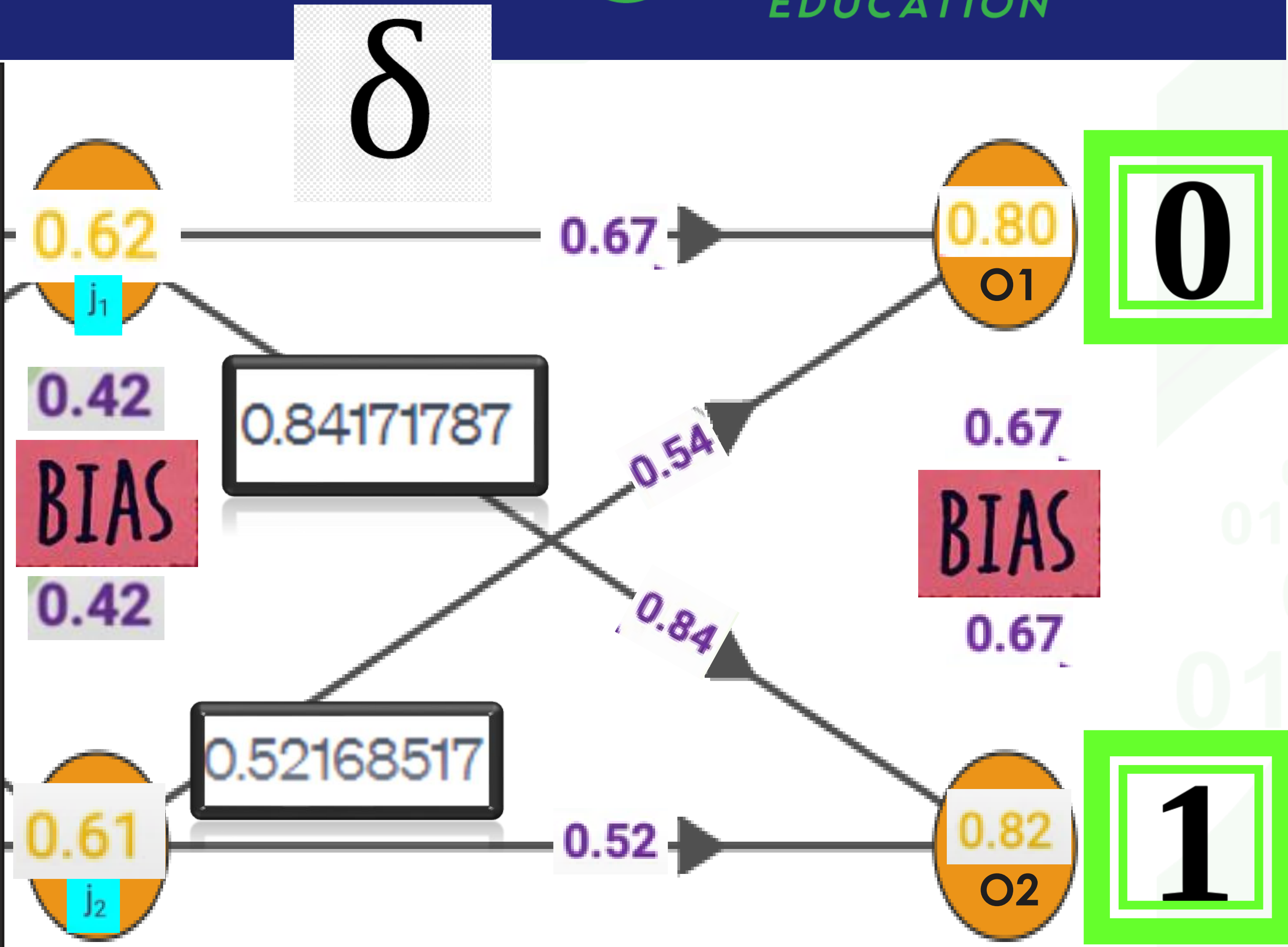
- 0.027812

PREVIOUS WEIGHT(0.52)

Wyeni= PREVIOUS WEIGHT-Learning rate δO2outputJ2

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

0.52168517



input layer

hidden layer

output layer

NOTEBOOK SAMPLES

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

# You can perform the necessary operations to load and prepare the dataset.

# Creating a sample CNN model
model = Sequential()

# First convolutional layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(BatchNormalization())  # Adding a Batch Normalization layer
model.add(MaxPooling2D((2, 2)))

# Second convolutional layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))

# Flattening layer
model.add(Flatten())

# Fully connected layers
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(10, activation='softmax'))  # Sample output layer

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

# Training the model with training data
model.fit(x_train, y_train, epochs=10, batch_size=32)
```

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

# You can perform the necessary operations to load and prepare the dataset.

# Creating a sample ANN model
model = Sequential()

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

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

# Training the model with training data
model.fit(x_train, y_train, epochs=10, batch_size=32)
```

Tea break...

10:00

Start | Stop | Reset | mins: 10 | secs: 0 | type: Tea

Breaktime for PowerPoint by Flow Simulation Ltd.

Pin controls when stopped