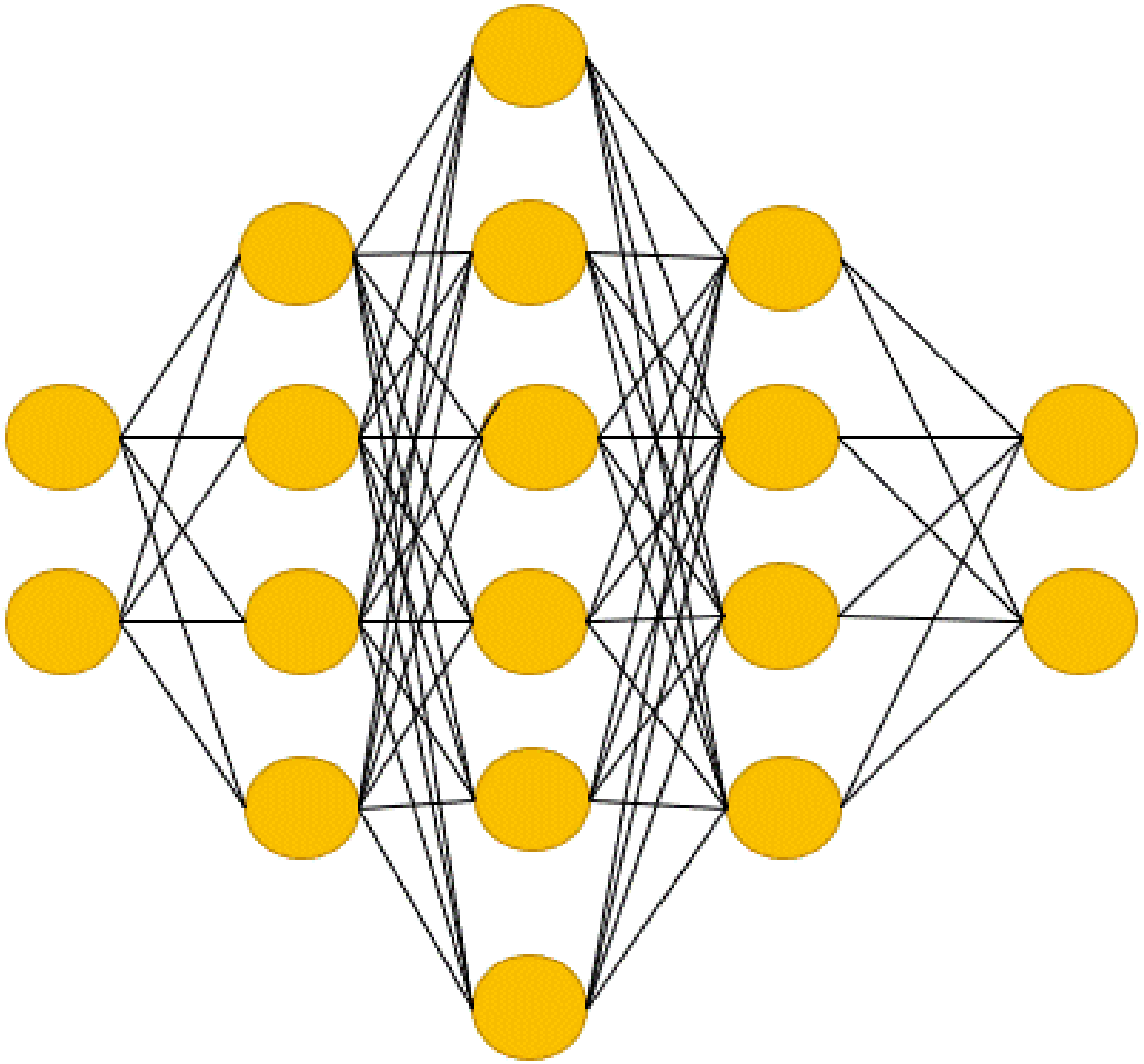
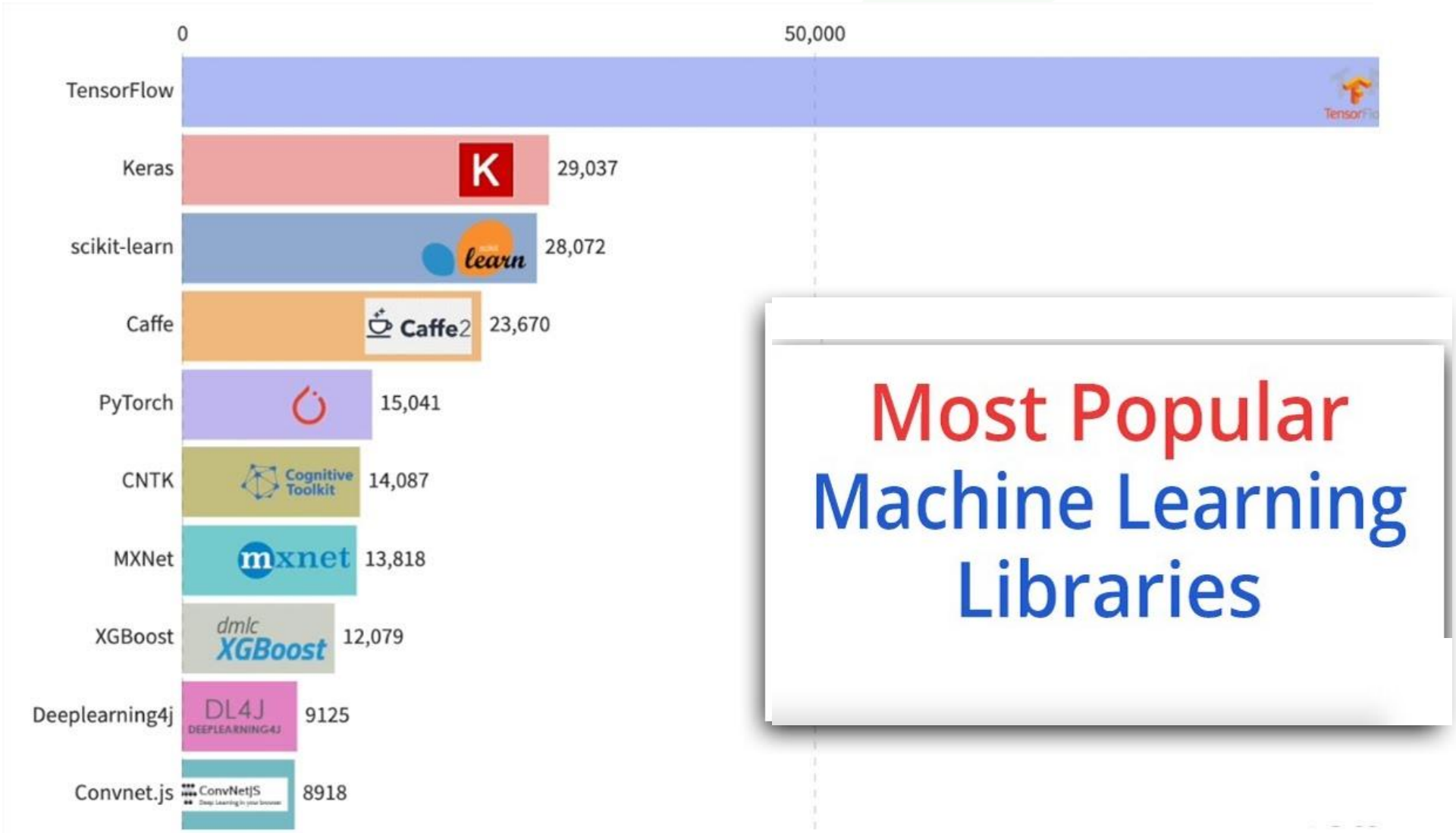
Let me ask, Gru

What is the Price of the Jet, Bob? It must be 50K



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++	+	+	+++	+	++	+	

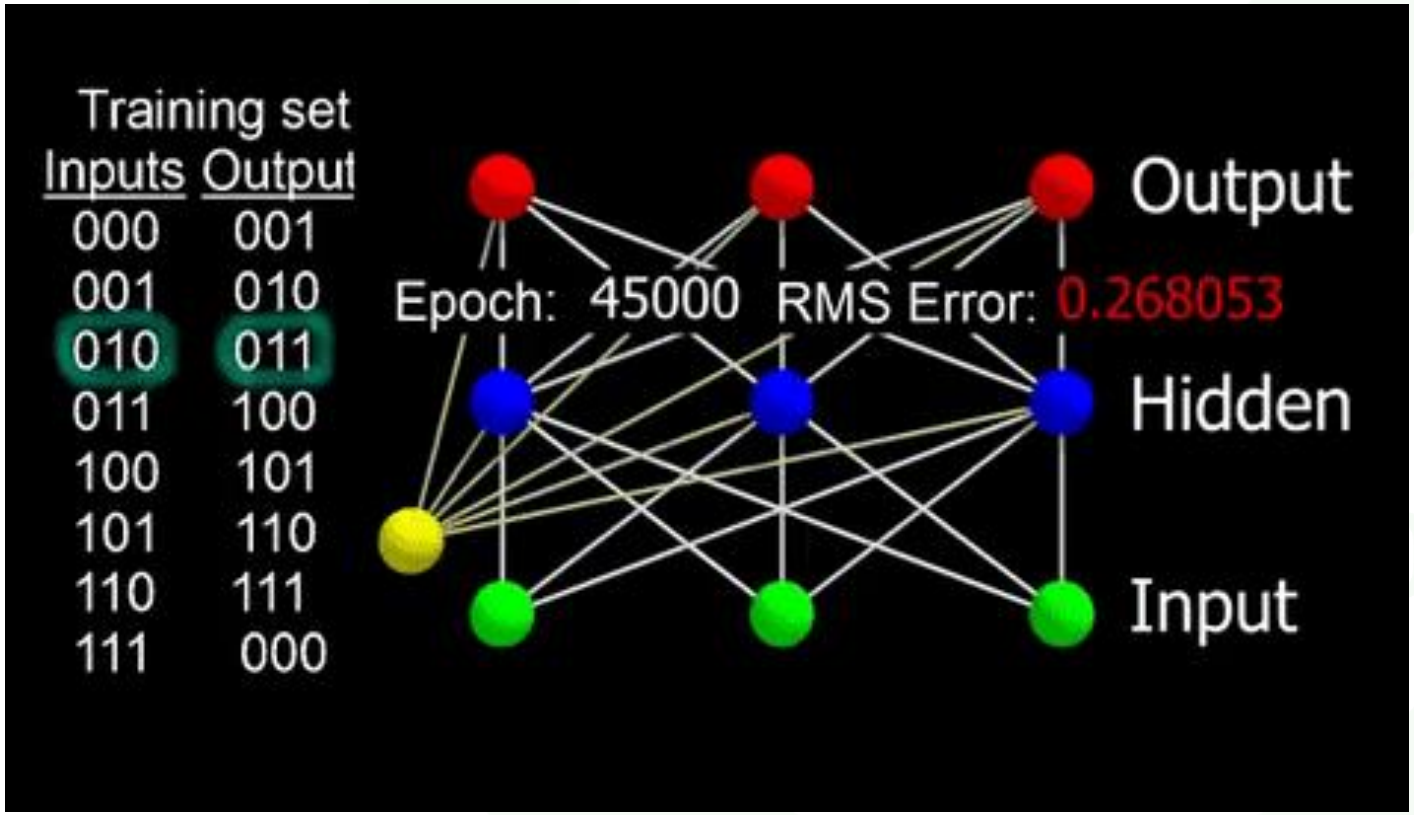
EPOCH, BATCHSIZE


ER
EFUL
BELT
TION.
KITCHEN

EPOCH

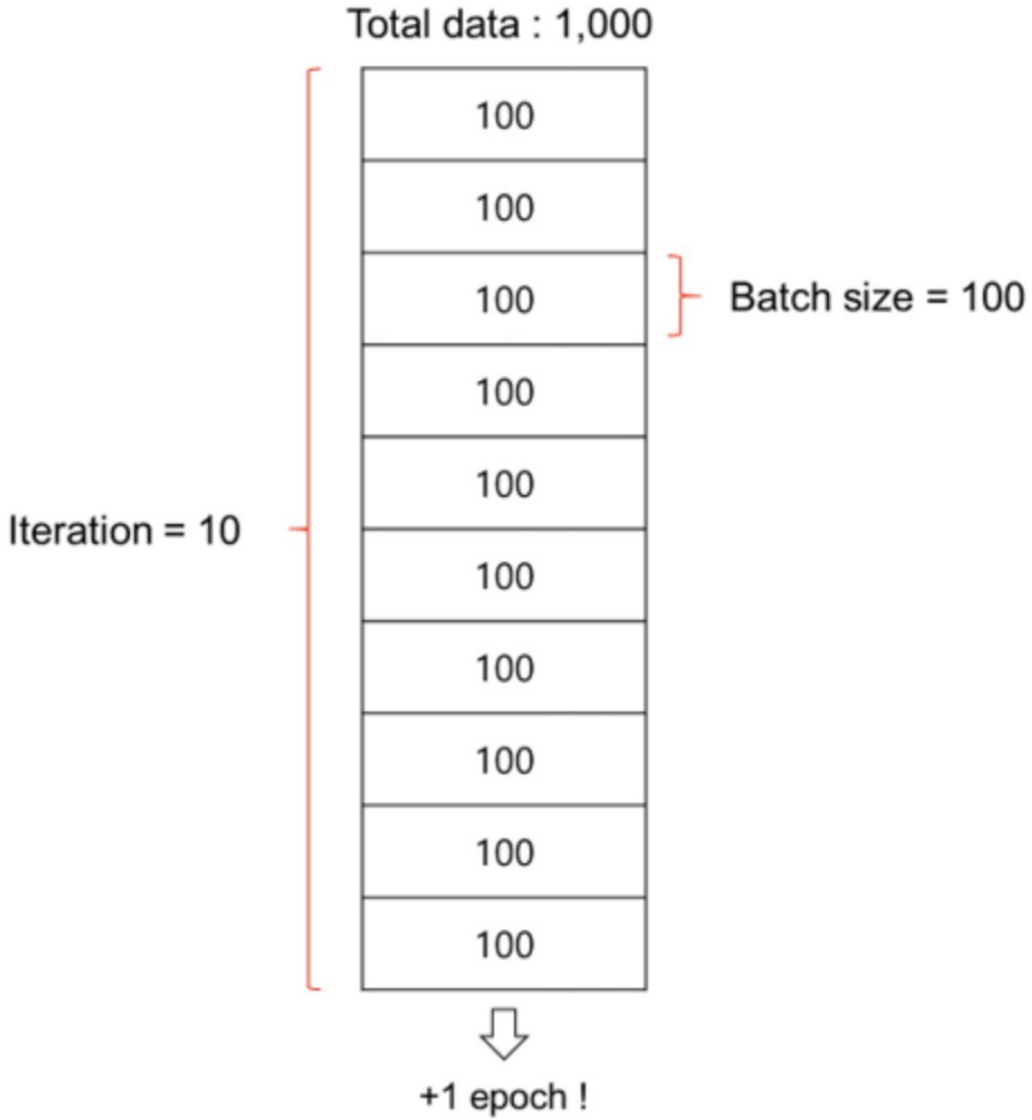
EPOCH

An **epoch** is a term used in machine learning and indicates the number of passes of the entire training dataset the machine learning algorithm has completed. Datasets are usually grouped into batches (especially when the amount of data is very large).



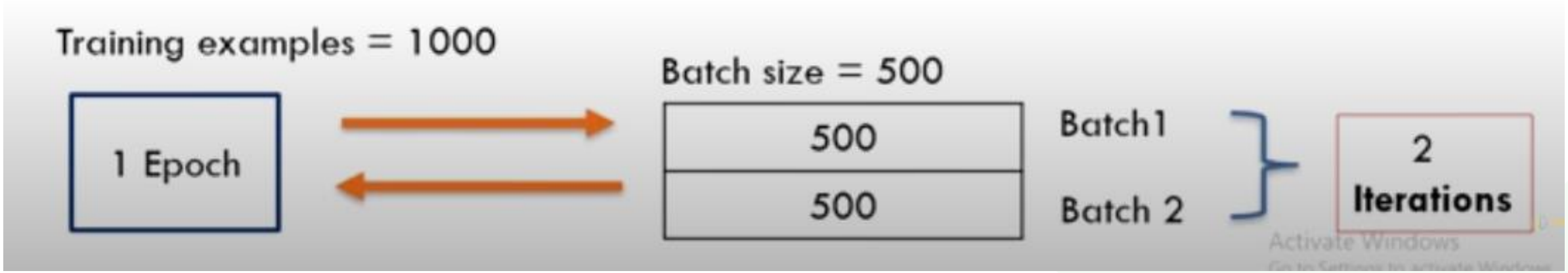
BATCHSIZE

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

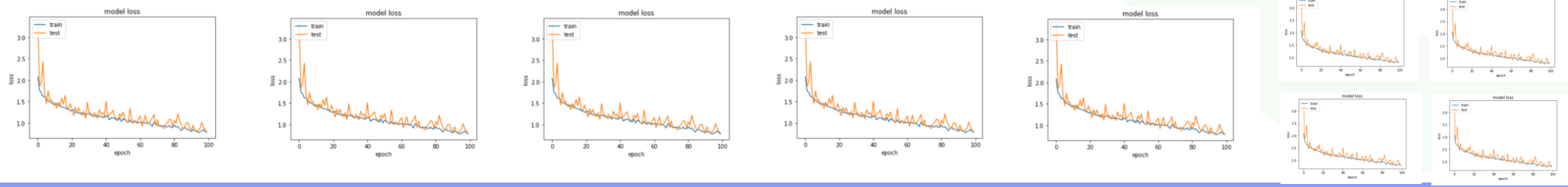
700 ÷ 32 =

21,875

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

1 EPOCH

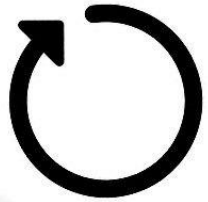
22 iteration



DIFFERENCE BETWEEN BATCH AND EPOCH

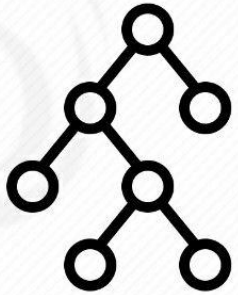
Epoch :

An Epoch represent one iteration over the entire dataset.



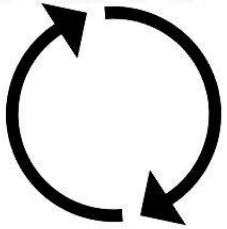
Batch :

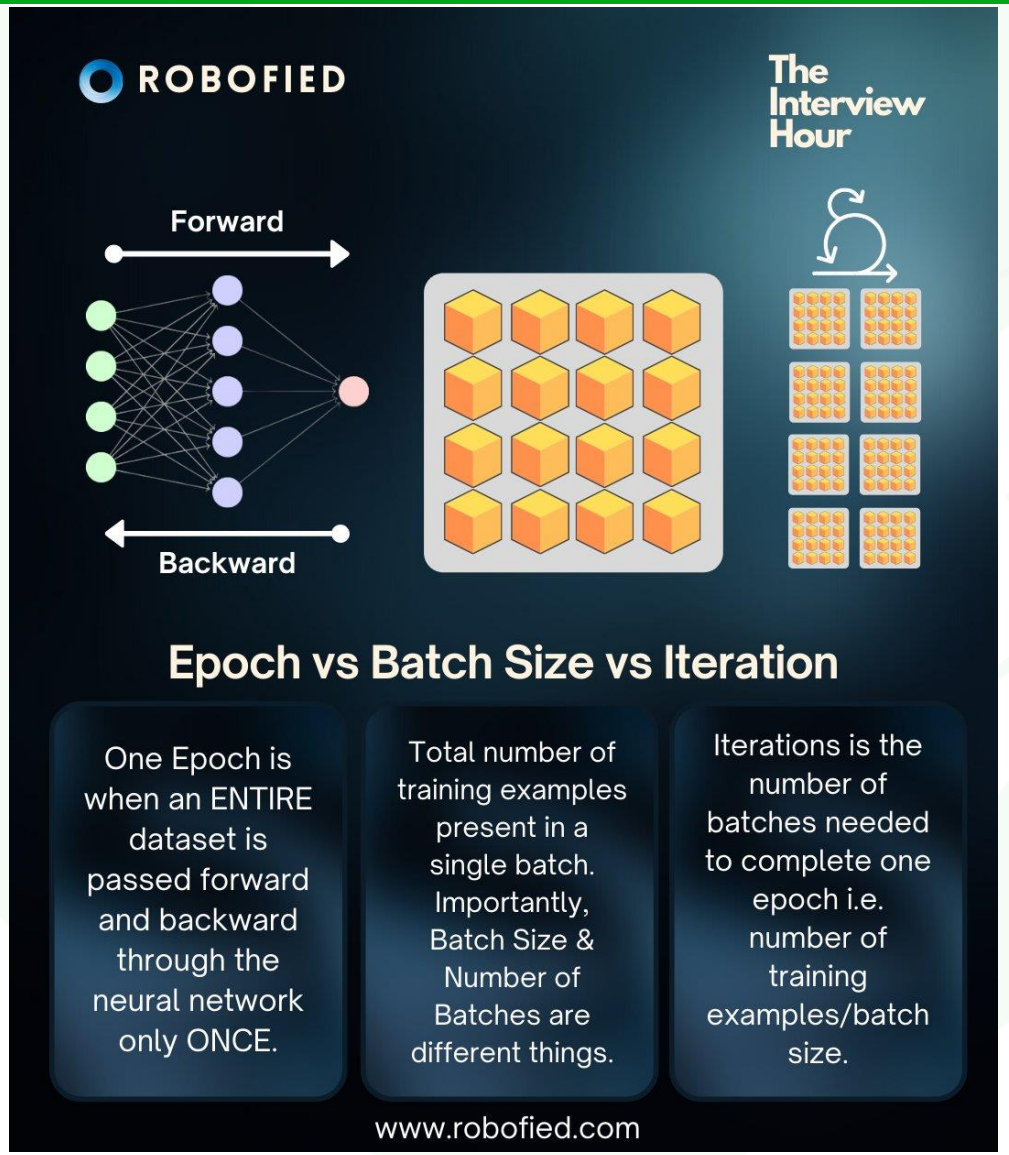
We cannot pass the entire dataset into the Neural Network at once. So, we divide the dataset into number of batches.



Iteration :

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





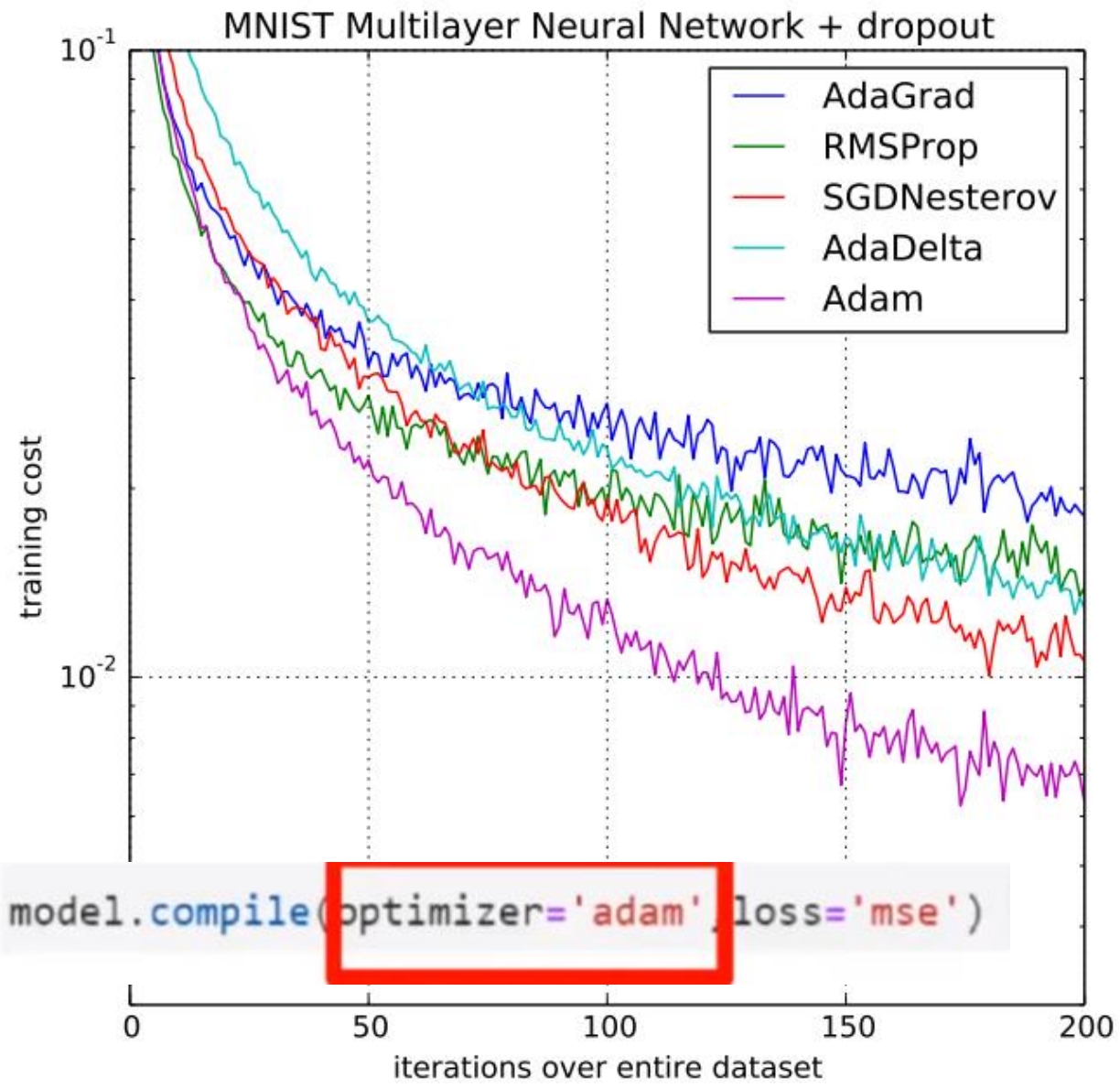
GRADIENT DESCENT

GRADIENT DESCENT

GRADIENT DESCENT

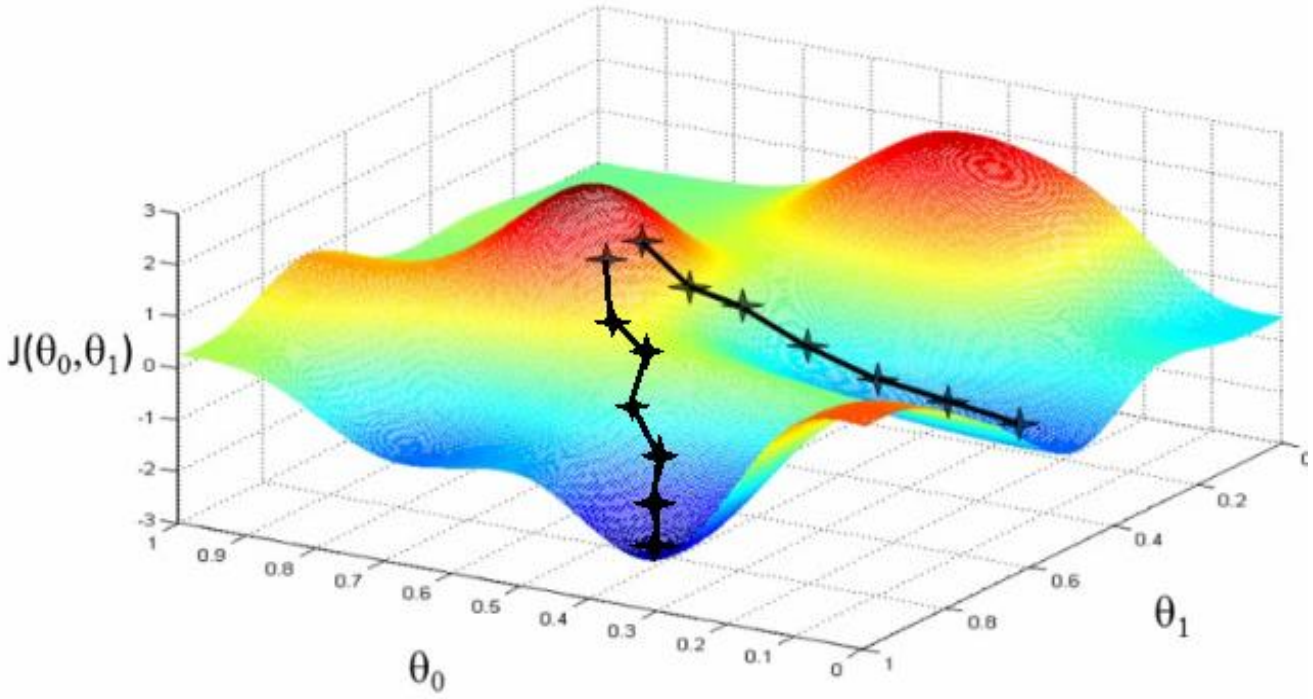
OPTIMIZER

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

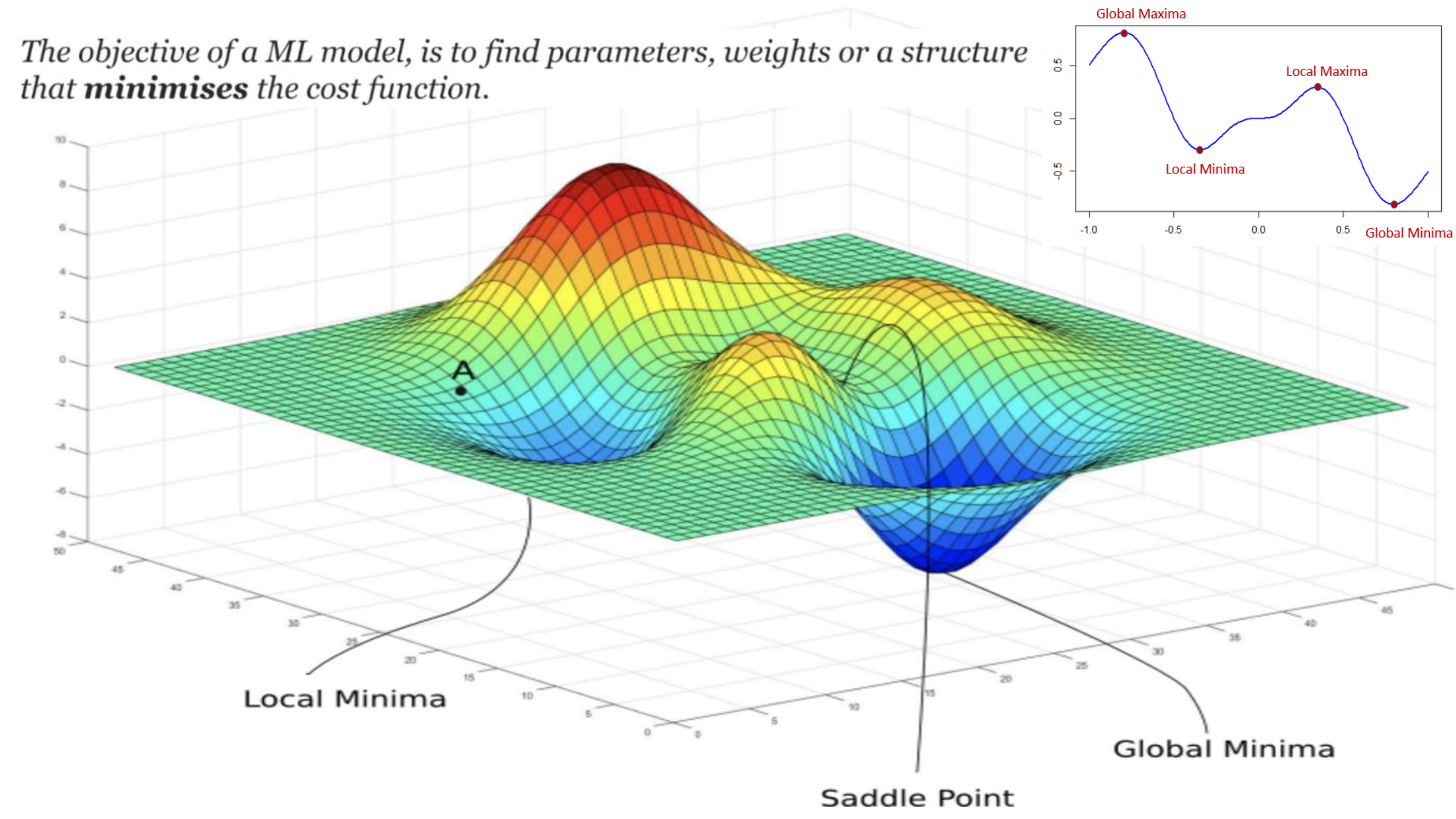
Gradient descent is the process of using gradients to find the minimum value of the cost function, while backpropagation is calculating those gradients by moving in a backward direction in the neural network.



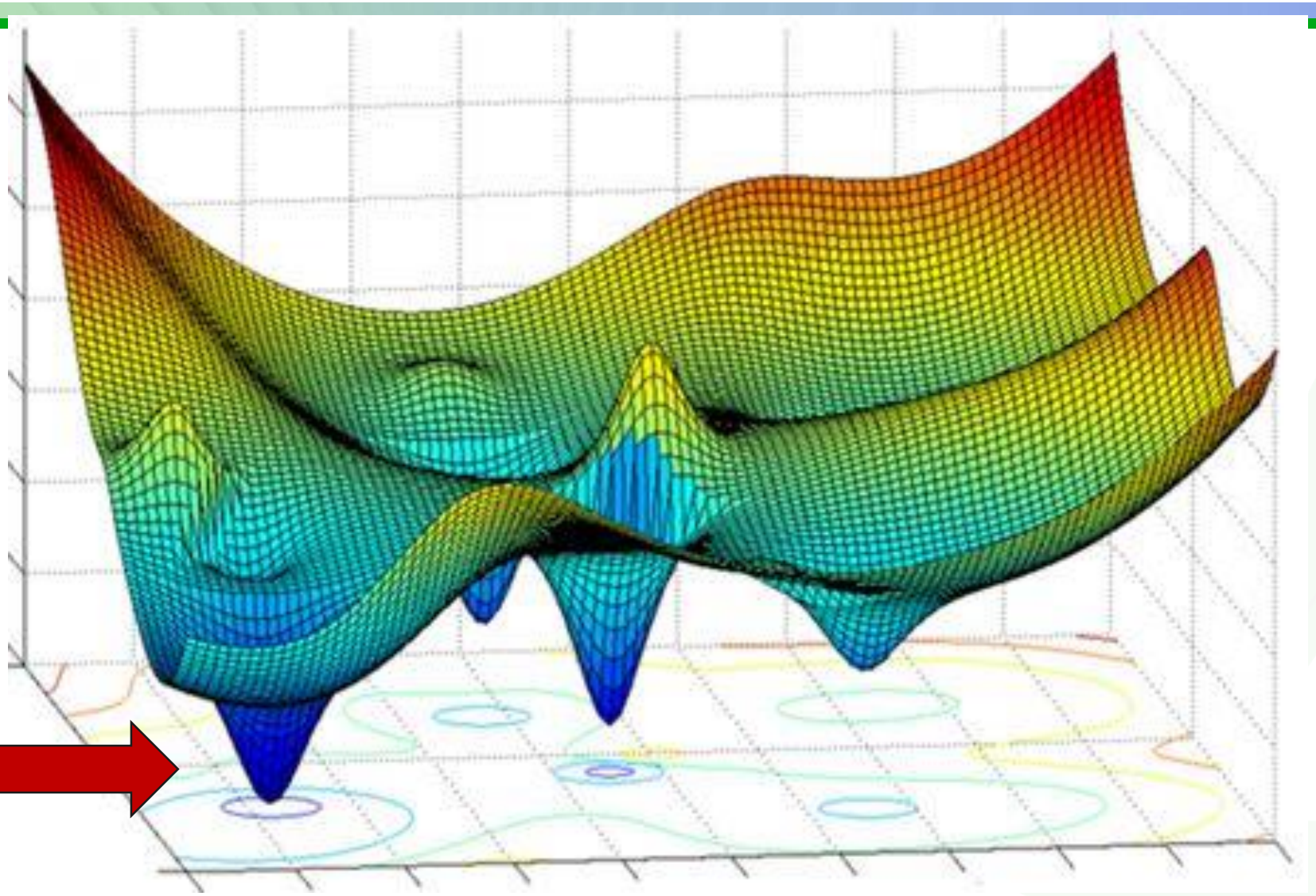
GRADIENT DESCENT

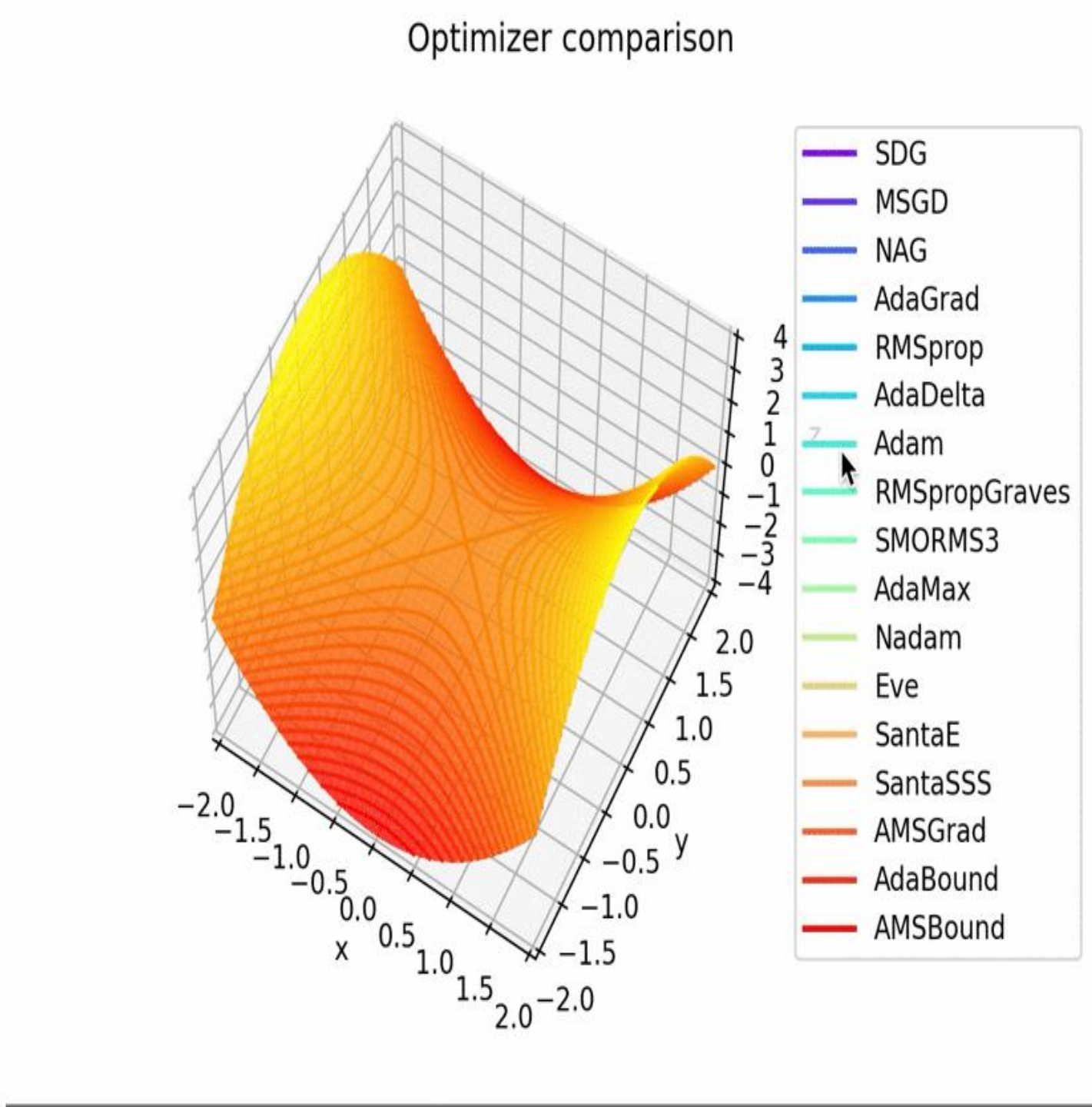

Leo Learns the Ropes
at Project Chimps

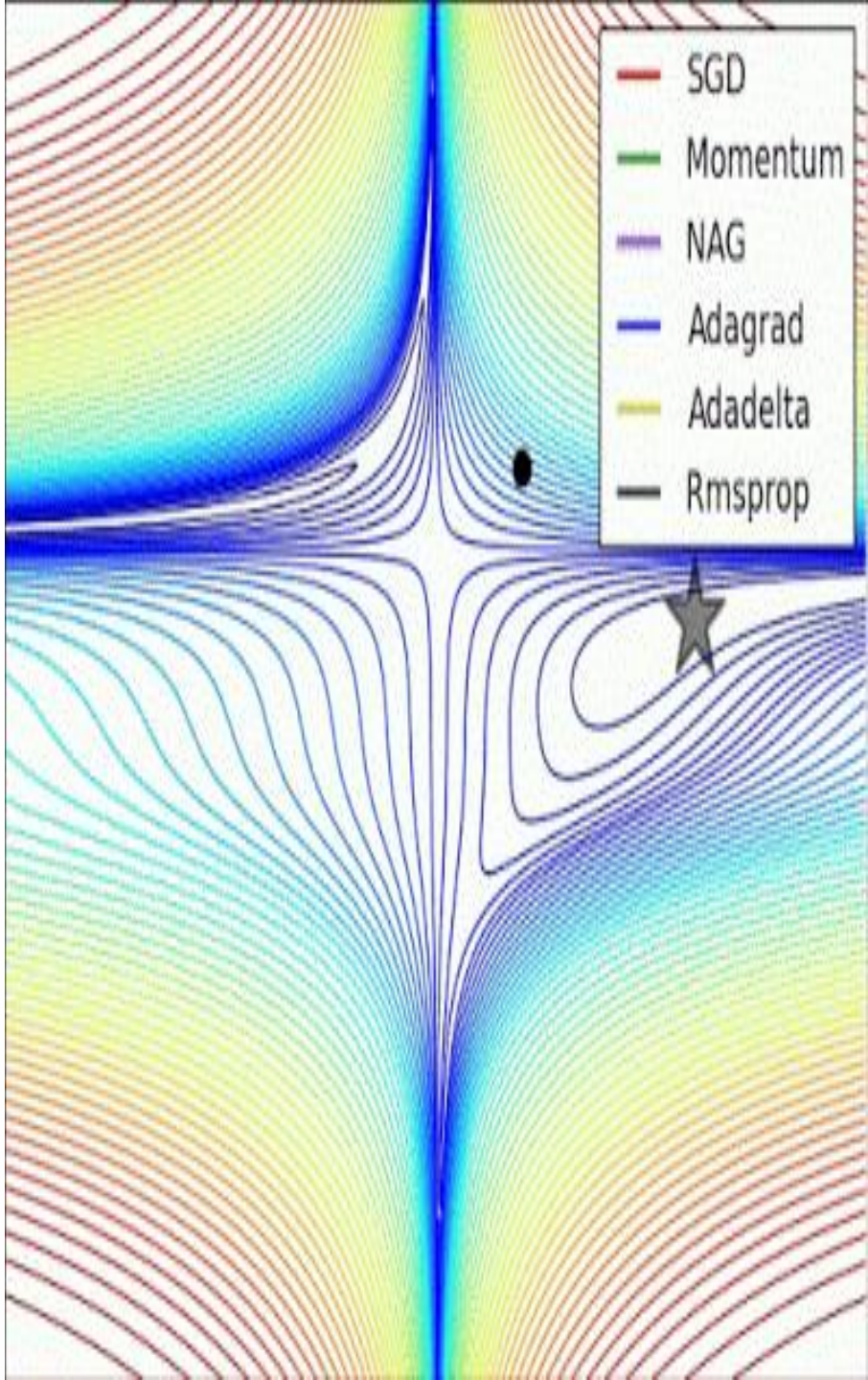
*The objective of a ML model, is to find parameters, weights or a structure that **minimises** the cost function.*



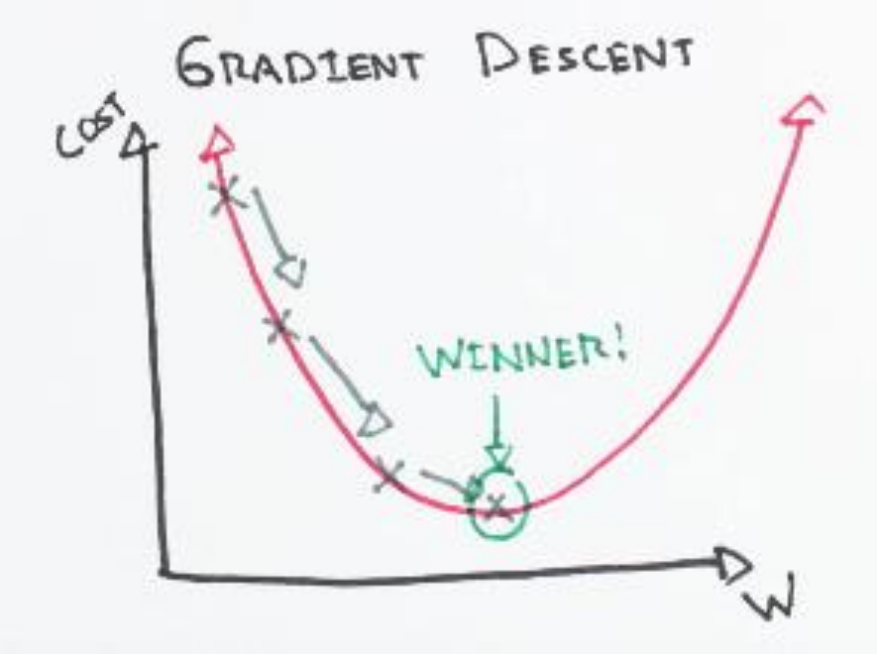
GRADIENT DESCENT



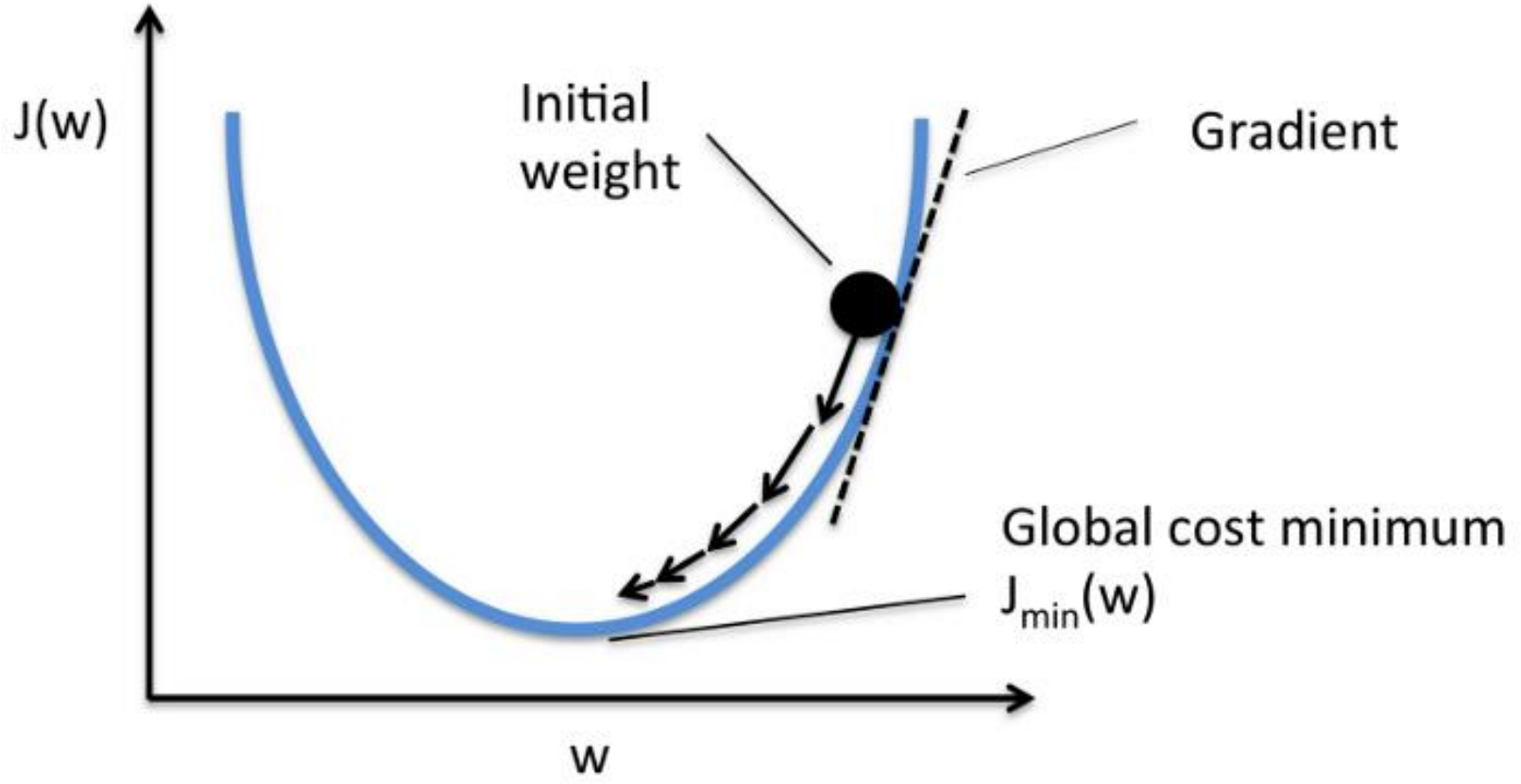

Optimizer comparison
SDG
MSGD
NAG
AdaGrad
RMSprop
AdaDelta
Adam
RMSpropGraves
SMORMS3
AdaMax
Nadam
Eve
SantaE
SantaSSS
AMSGrad
AdaBound
AMSBound
x
y
z


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

GRADIENT DESCENT

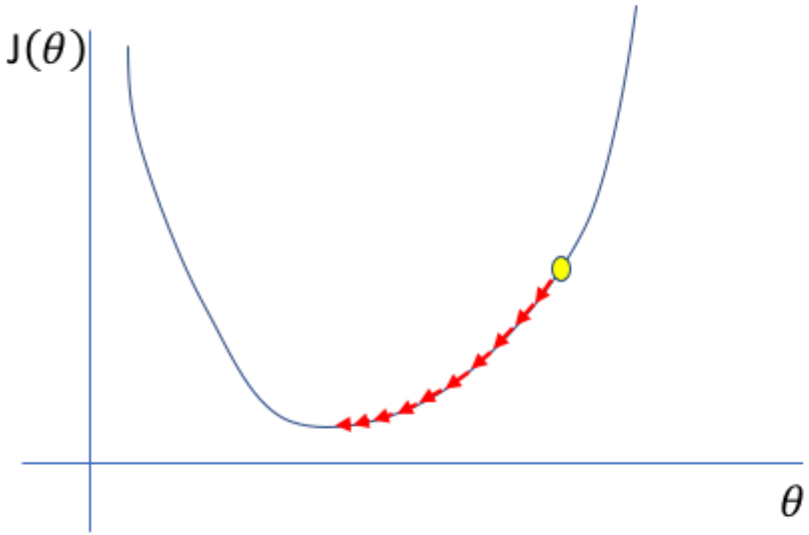


Step Size = slope x learning rate



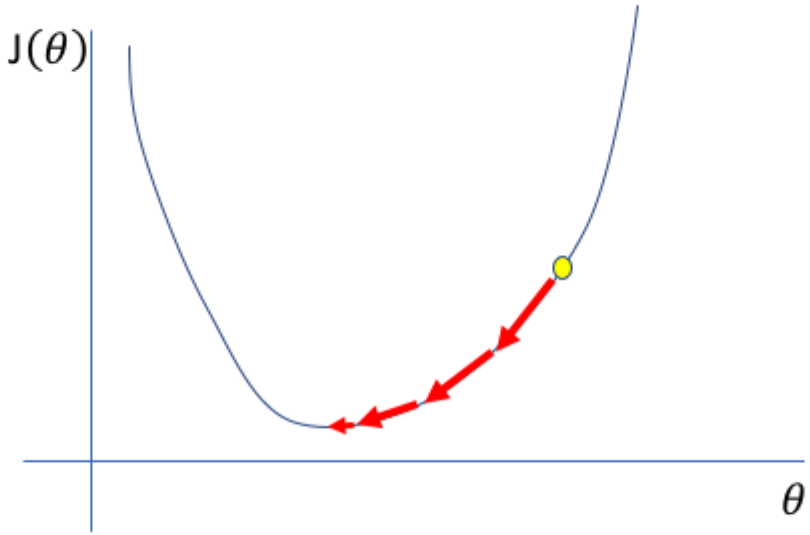
LEARNING RATE

Too low



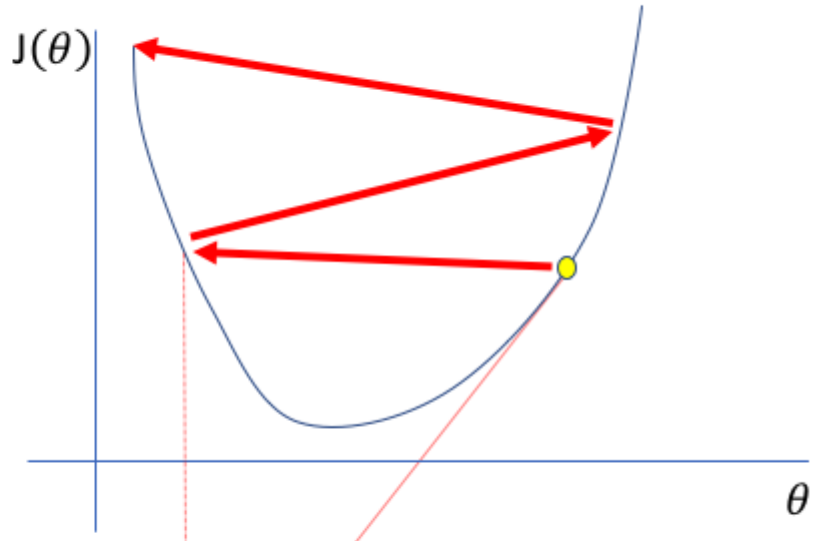
A small learning rate requires many updates before reaching the minimum point

Just right



The optimal learning rate swiftly reaches the minimum point

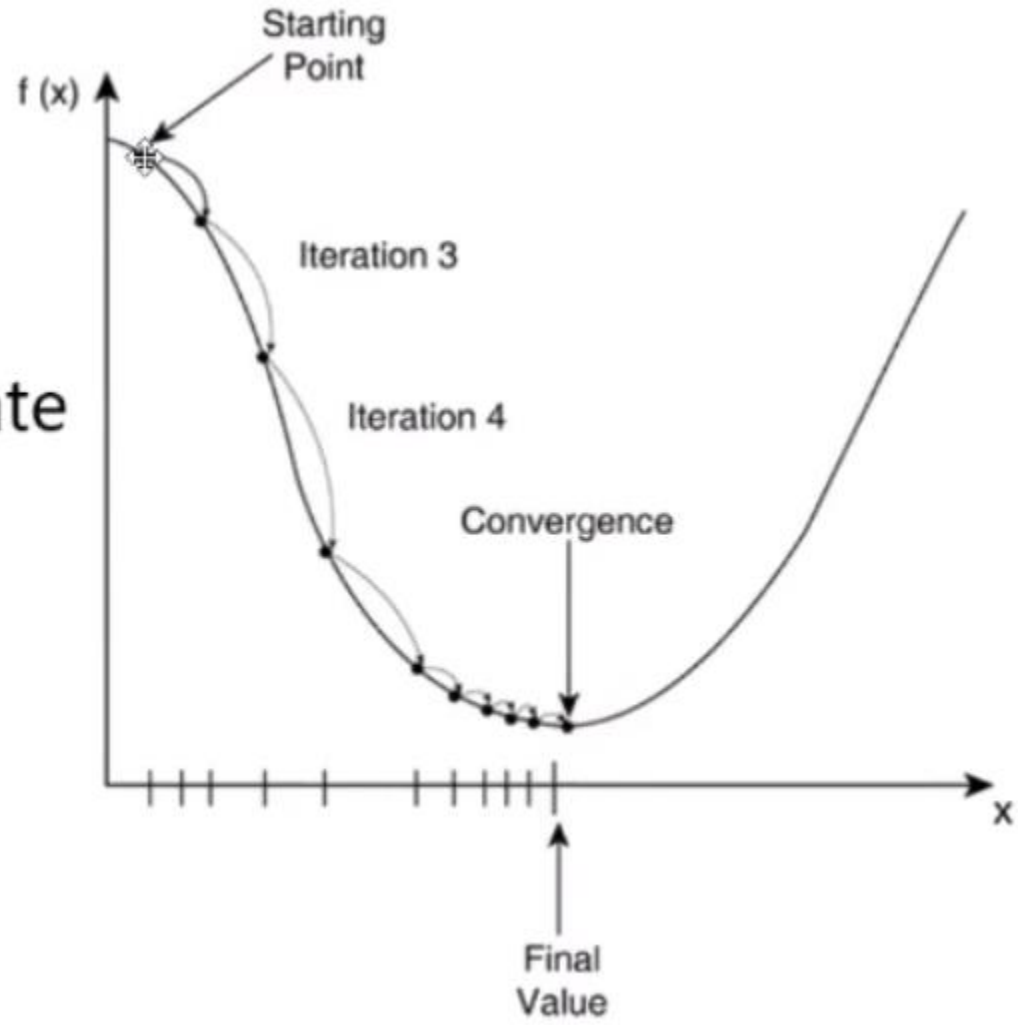
Too high



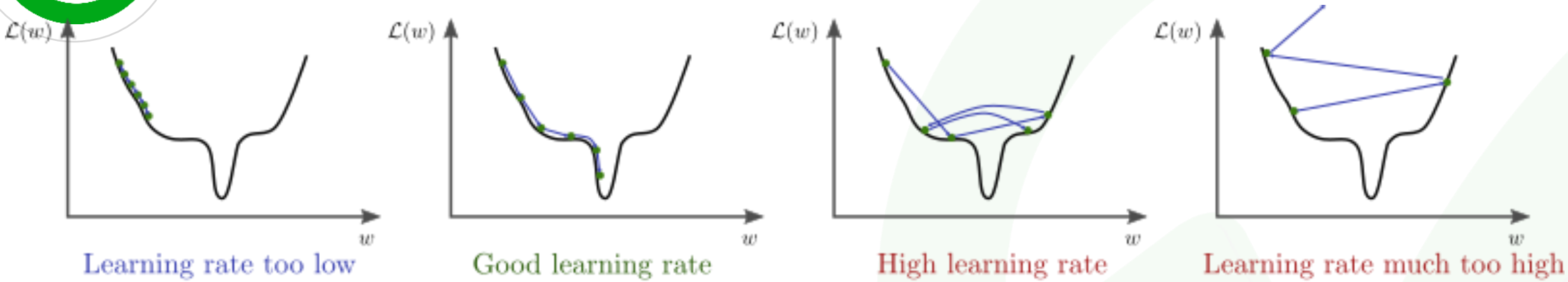
Too large of a learning rate causes drastic updates which lead to divergent behaviors

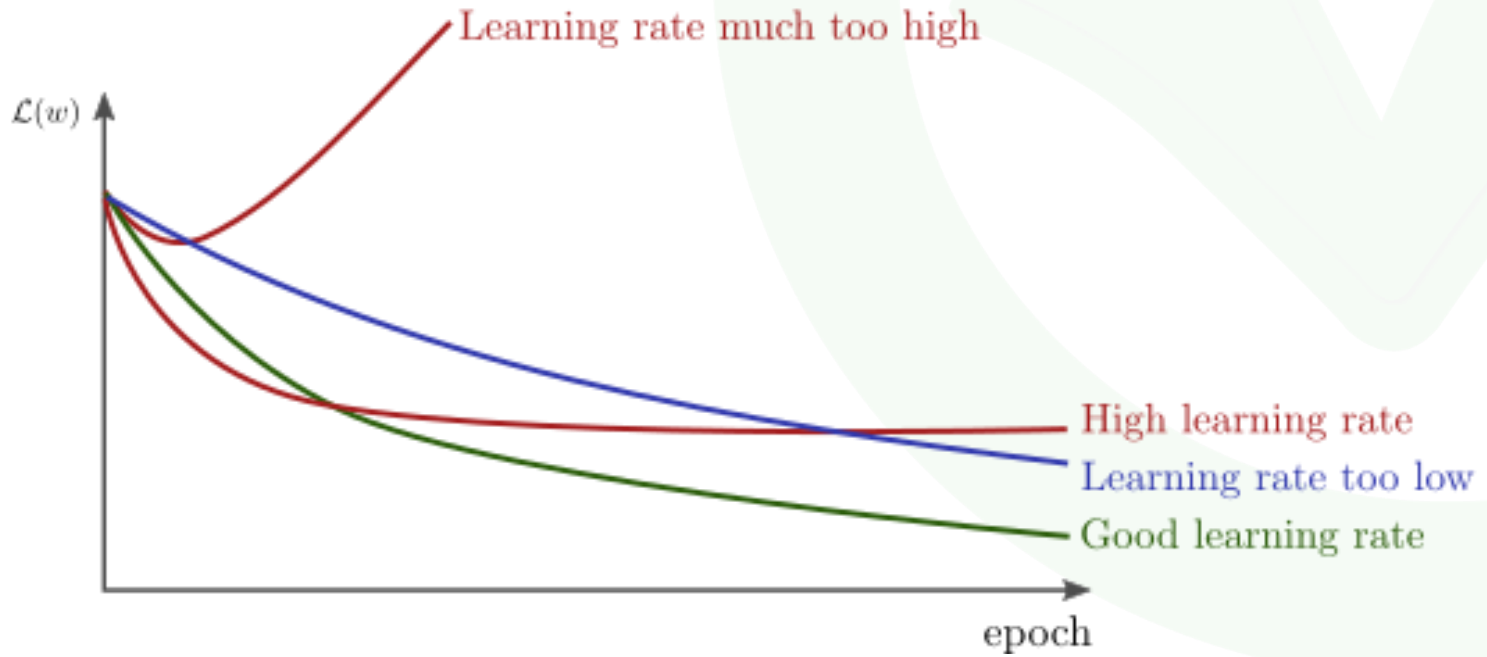
LEARNING RATE

Step Size = slope x learning rate



LEARNING RATE



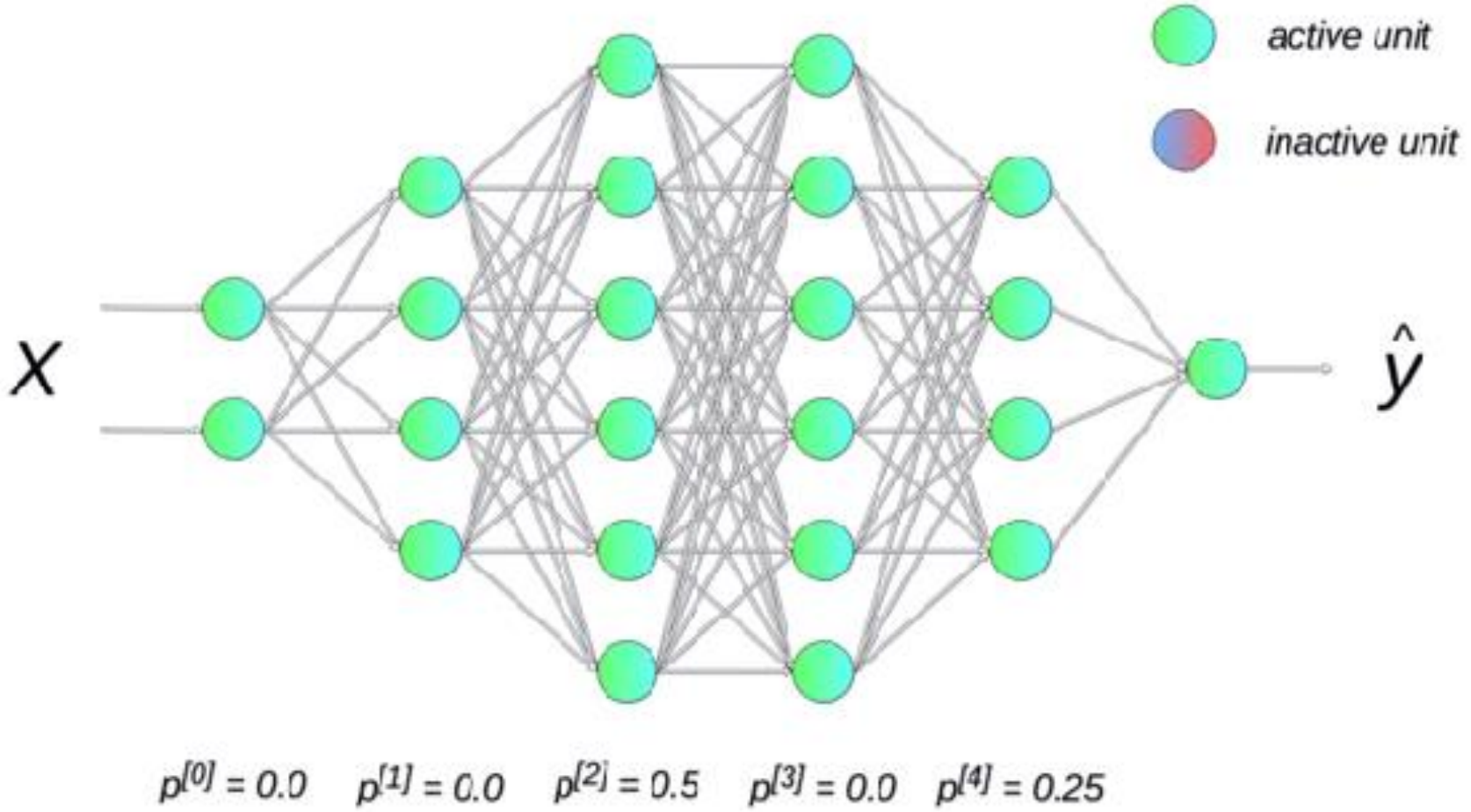


REGULARIZATION

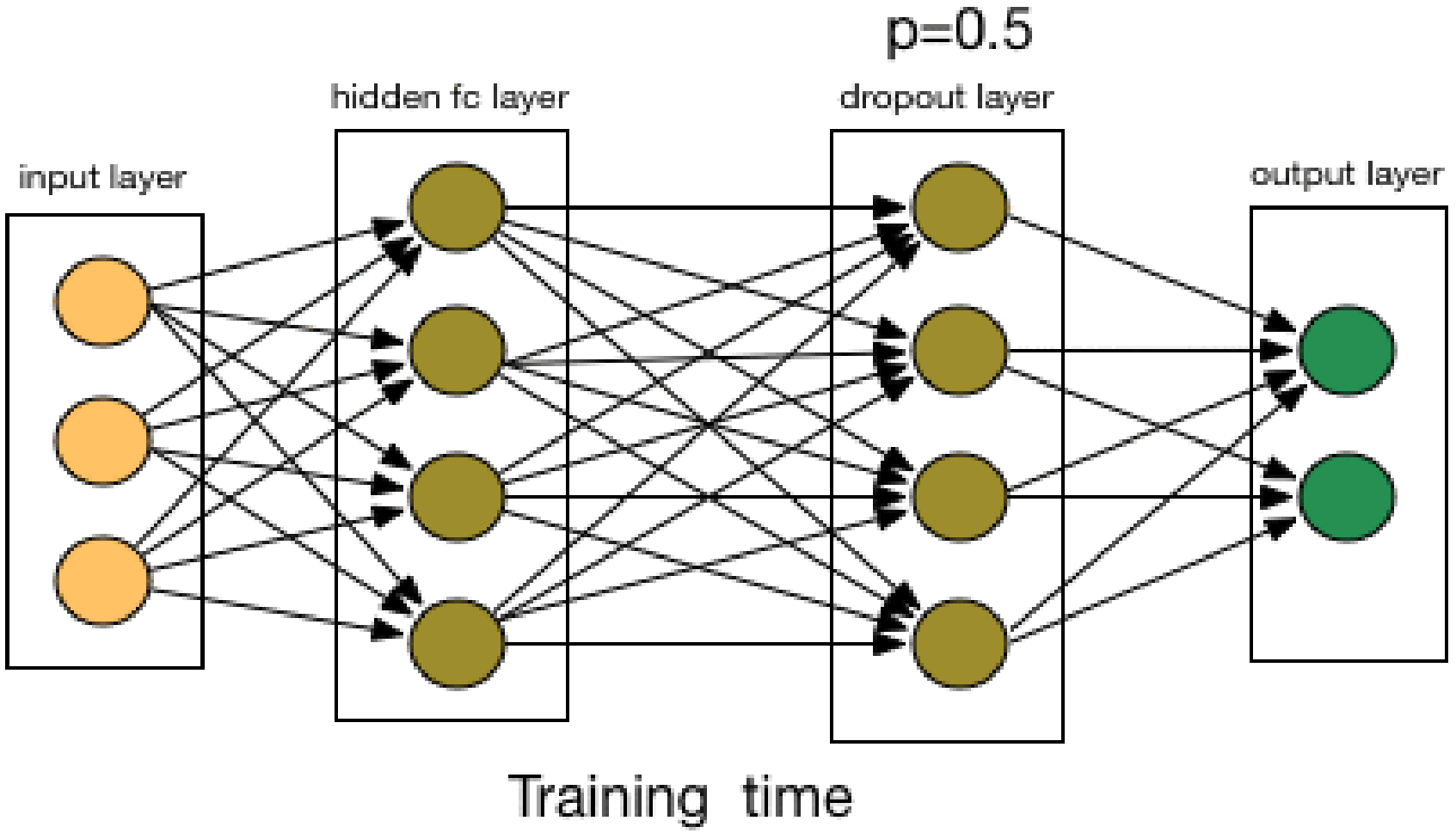
DROPOUT
EARLYSTOPPING
BATCHNORMALIZATION

L1
L2

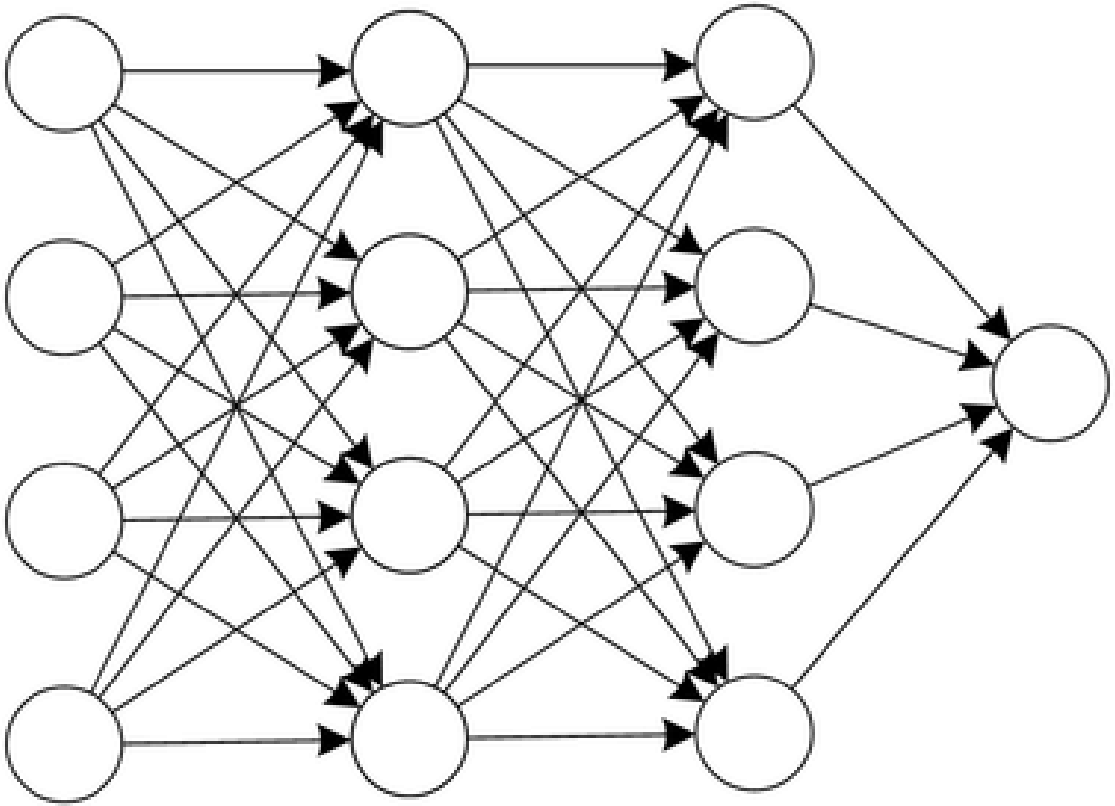
DROPOUT



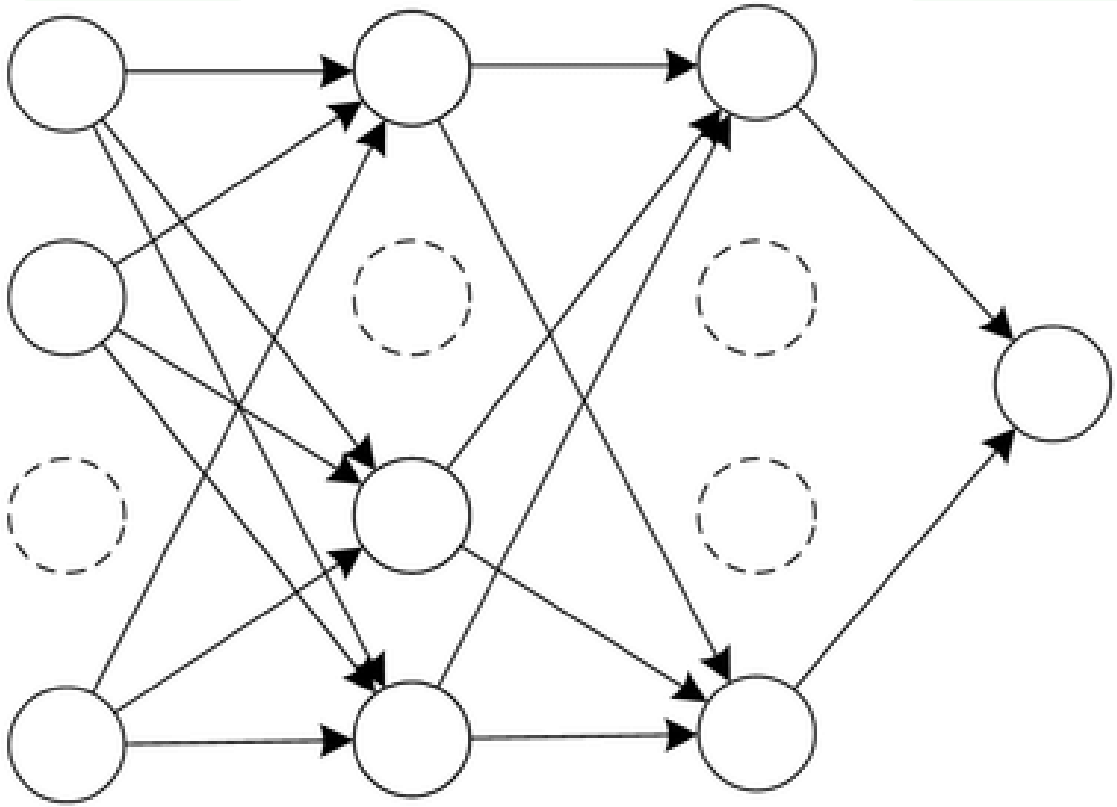
DROPOUT


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

DROPOUT

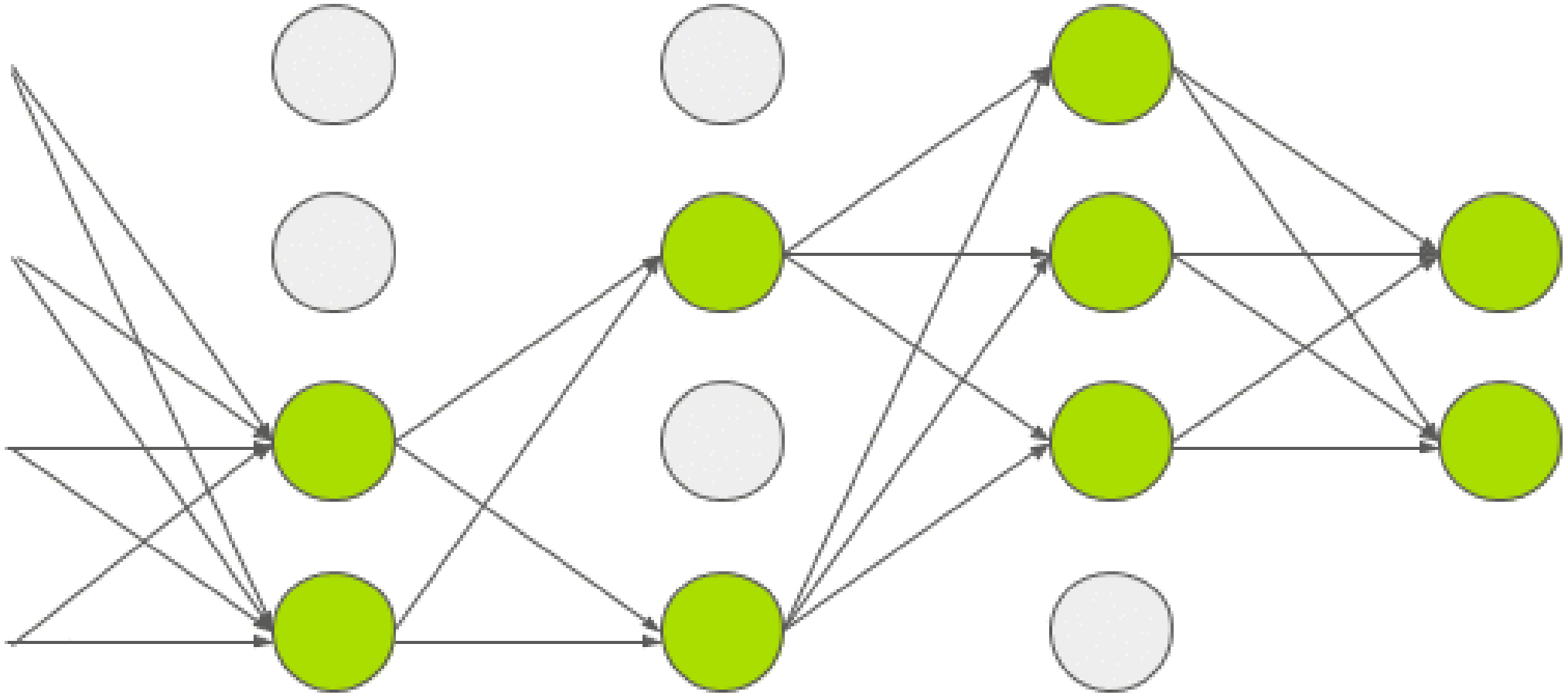


(a) Standard Neural Network



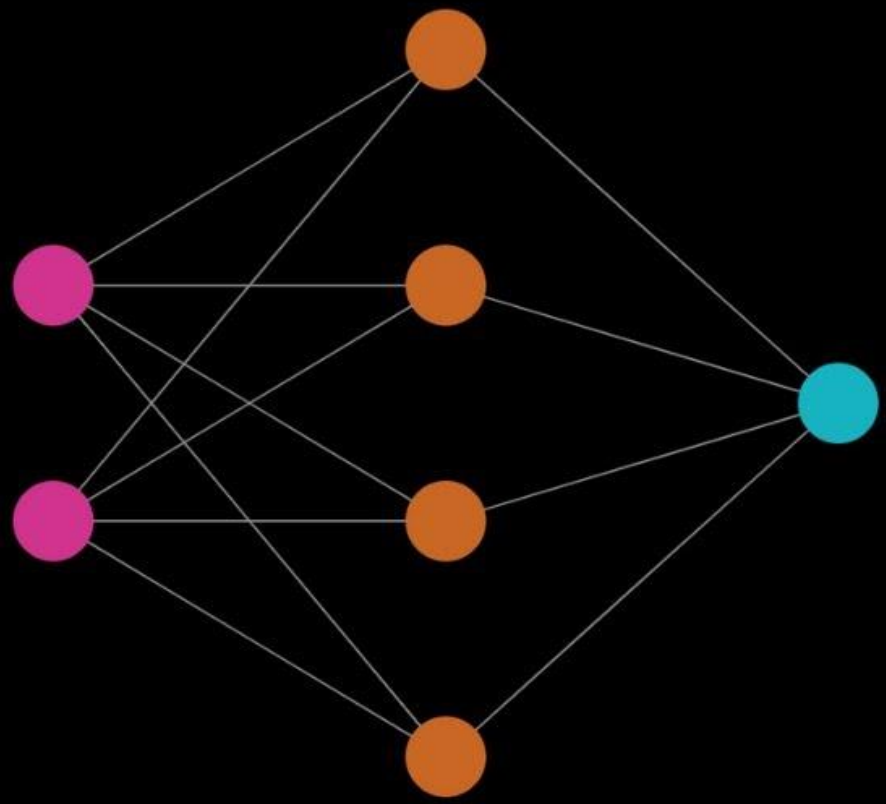
(b) Network after Dropout

DROPOUT

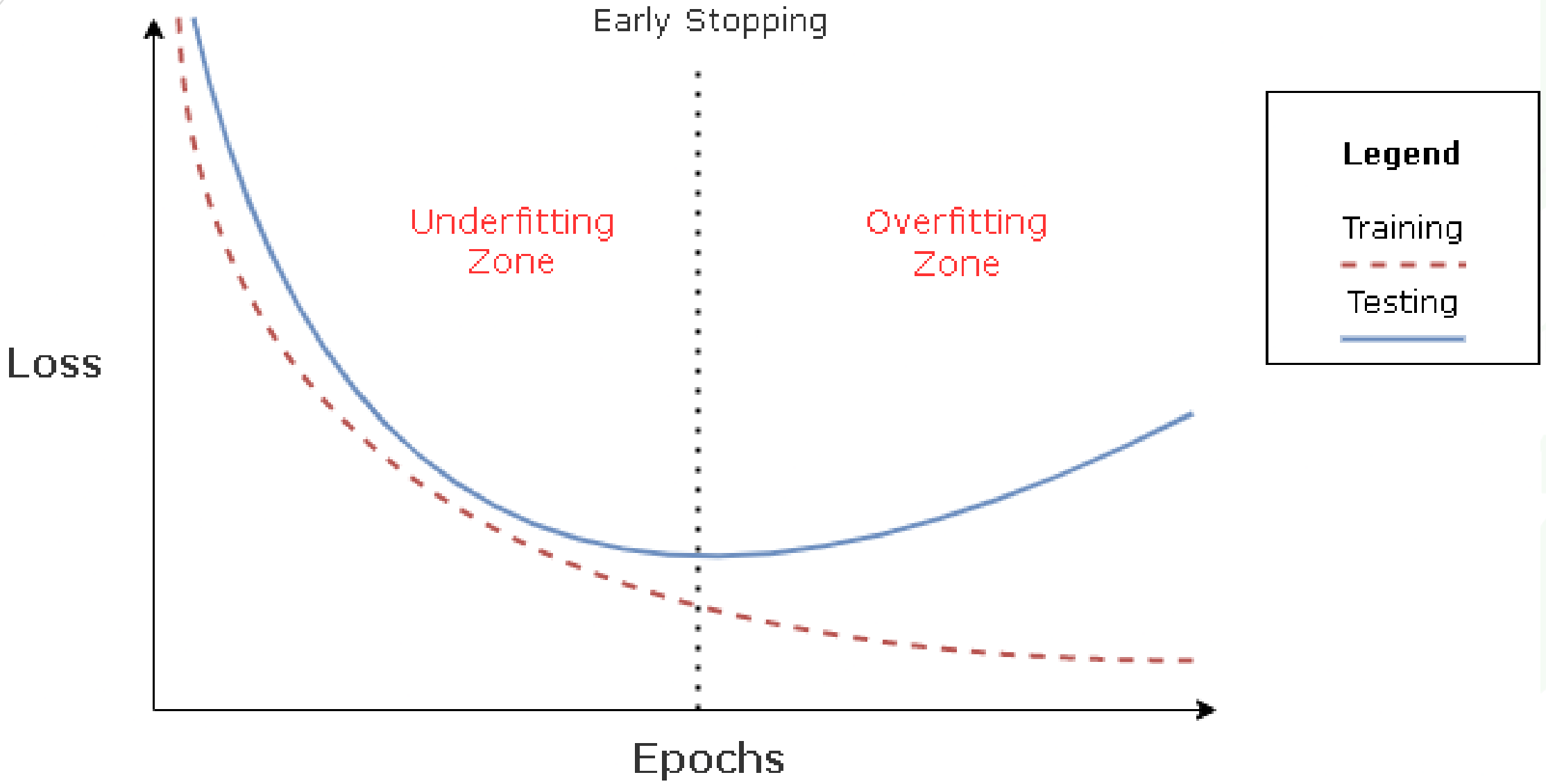


EARLYSTOPPING

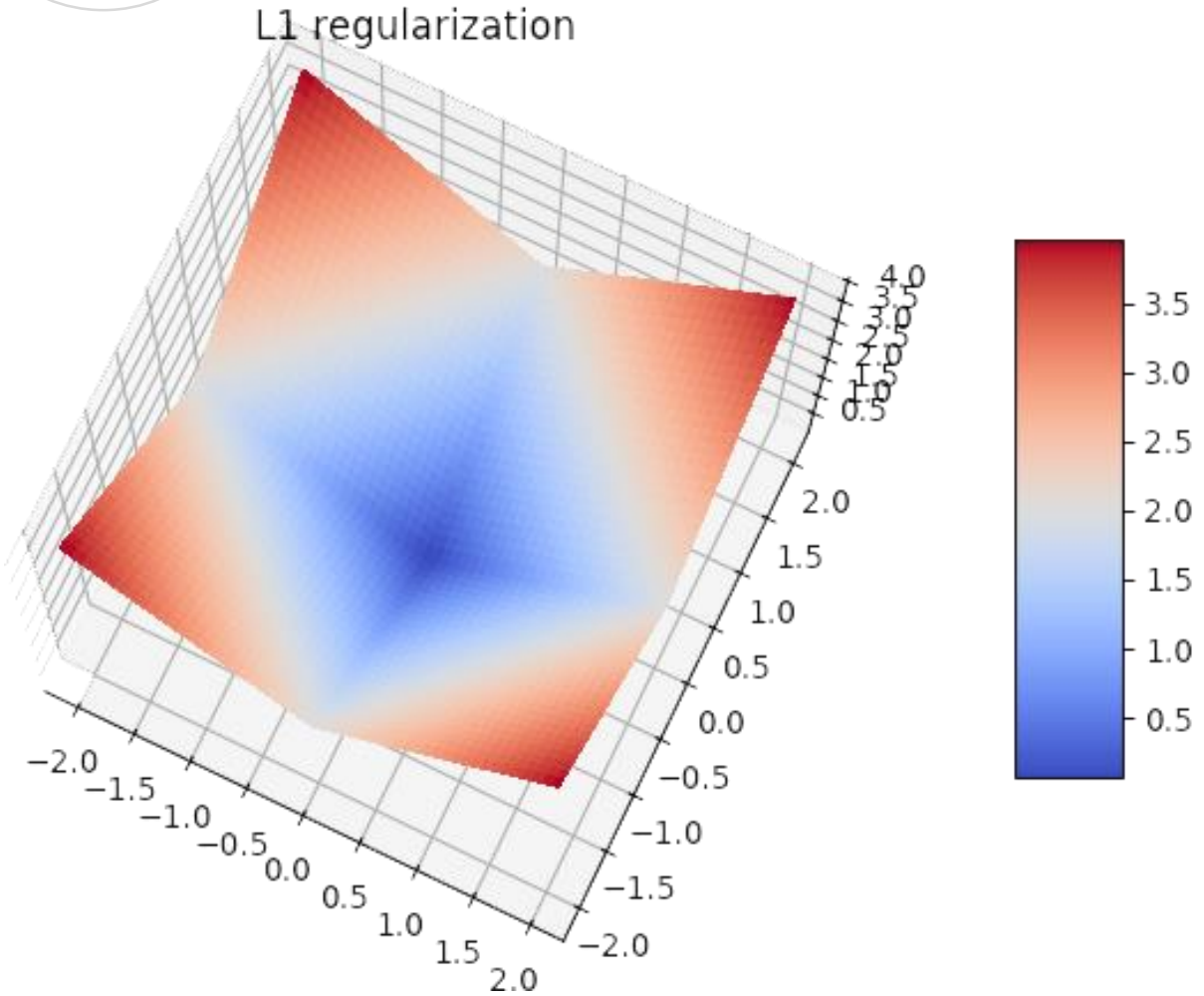
EARLY STOPPING TO PREVENT OVERFITTING

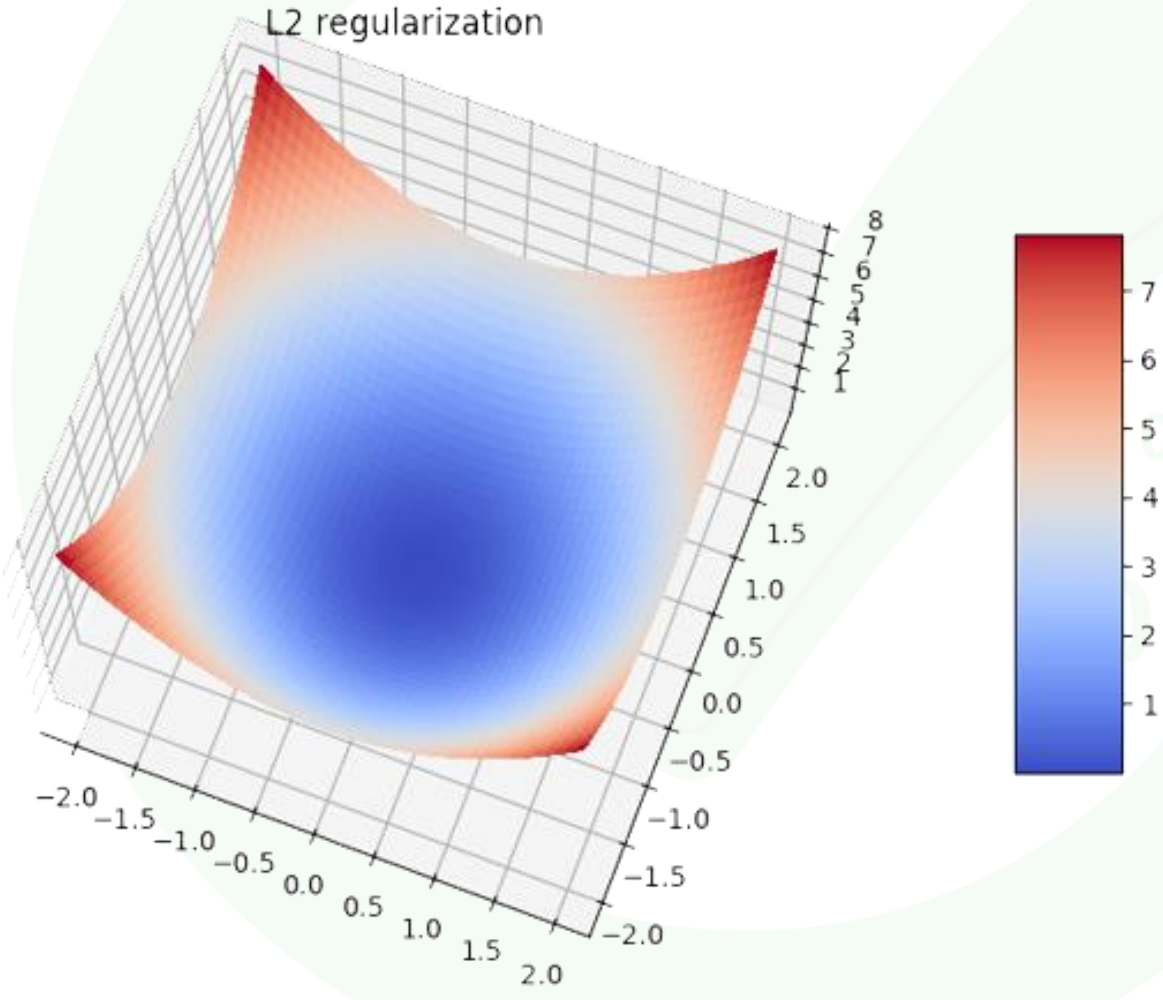


EARLYSTOPPING

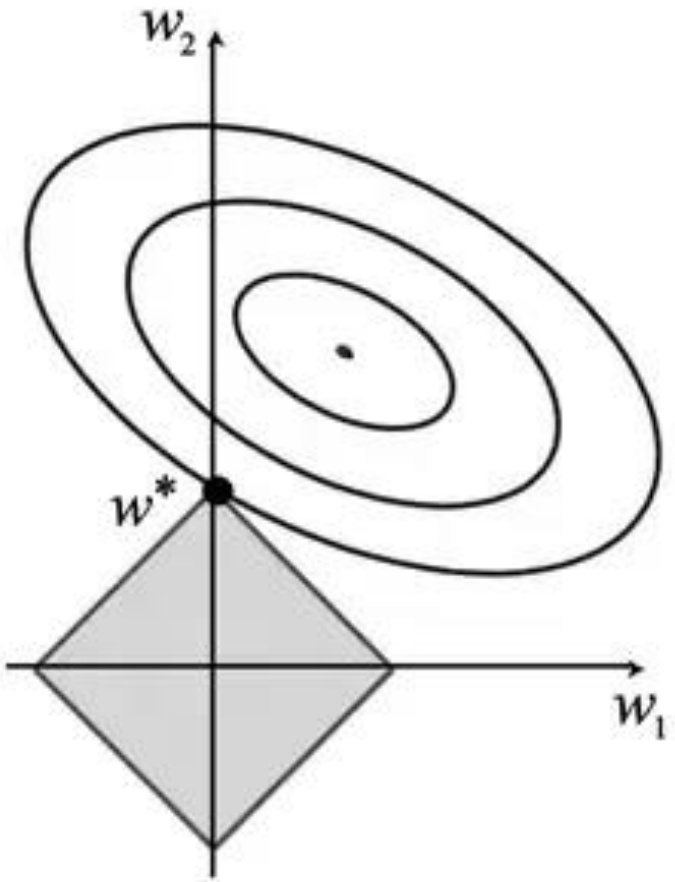


L1 and L2

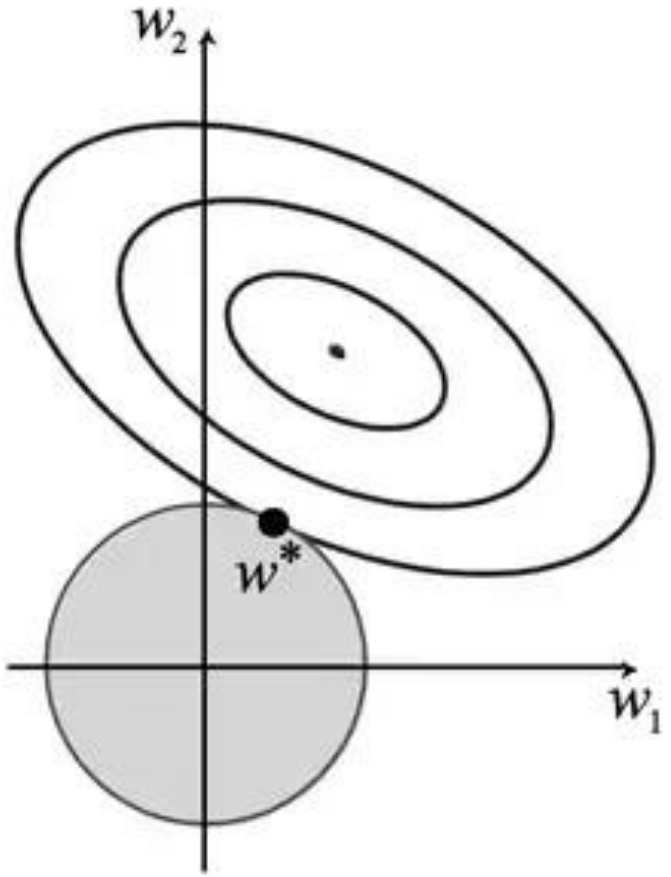

L1 regularization
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5


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

L1 and L2



L1



L2

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

Batch Normalization (BN), bir derin öğrenme modelinin her bir eğitim mini-batch'indeki verilerin istatistiksel özelliklerini normalleştiren bir tekniktir.

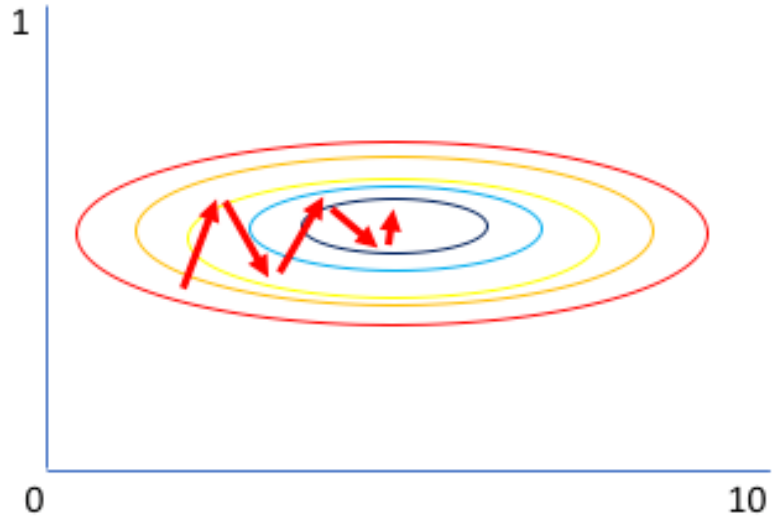
Daha basit bir şekilde, BN, eğitim sırasında her mini-batch verisinin ortalamasını ve varyansını hesaplar ve bu ortalamayı ve varyansı kullanarak verileri bir tür standart ölçekte yeniden ölçekler. Bu, ağın daha hızlı ve daha istikrarlı bir şekilde öğrenmesine yardımcı olabilir.

Batch Normalization, ağın daha hızlı ve daha güvenilir bir şekilde eğitilmesine yardımcı olan bir normalizasyon tekniğidir.

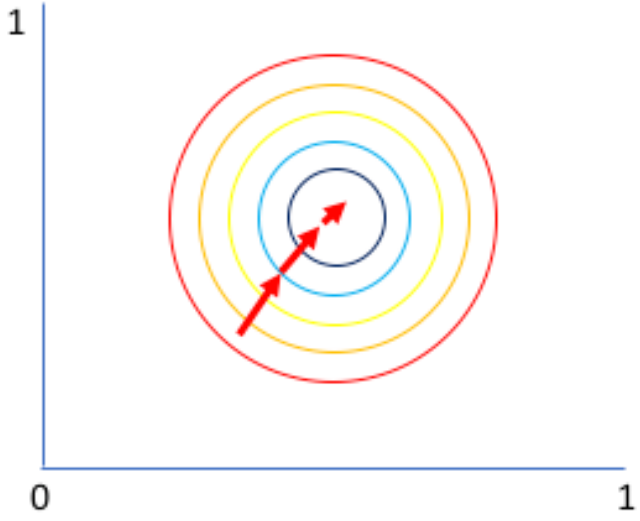
Kullanım Durumu	Açıklama
Derin Sinir Ağları (DNN)	DNN'lerde Batch Normalization, daha hızlı ve istikrarlı bir eğitim süreci sağlayabilir ve aşırı uyumu azaltabilir.
Evrişimli Sinir Ağları (CNN)	CNN'lerde özellikle büyük ve karmaşık modellerde kullanılabilir. Evreli ve tam bağlantılı katmanlar arasında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

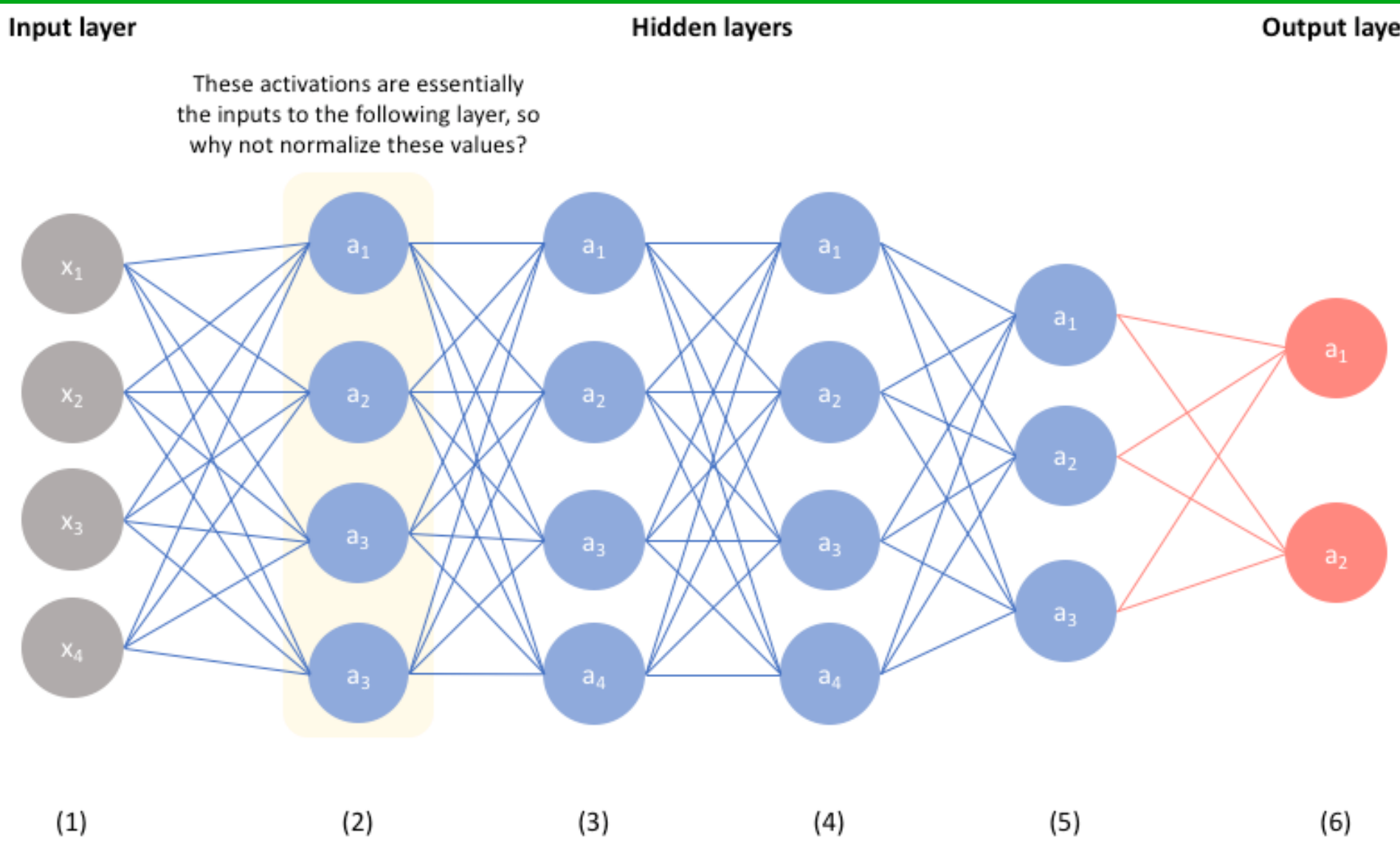


Gradient of larger parameter dominates the update

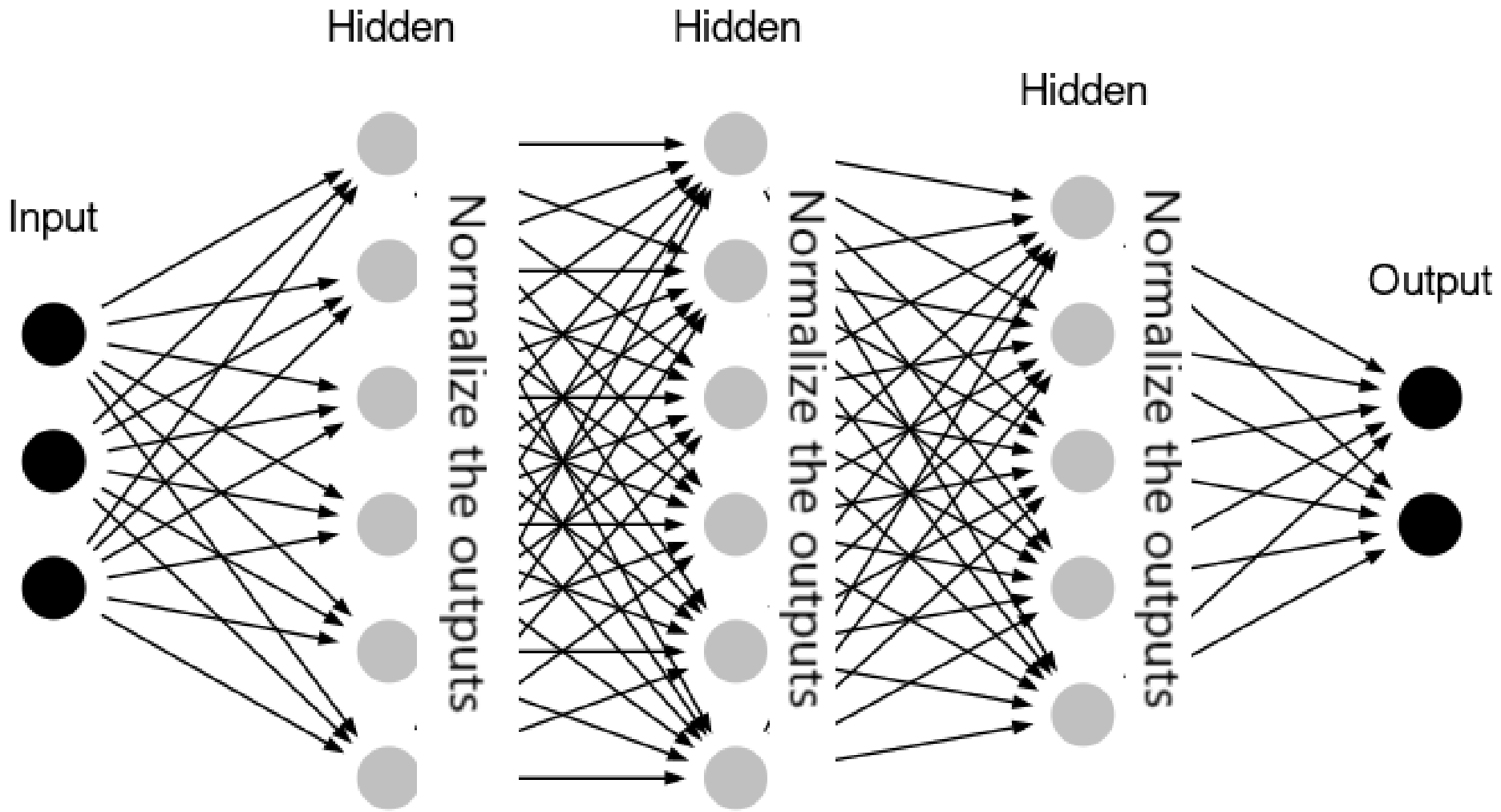


Both parameters can be updated in equal proportions

Batch Normalization

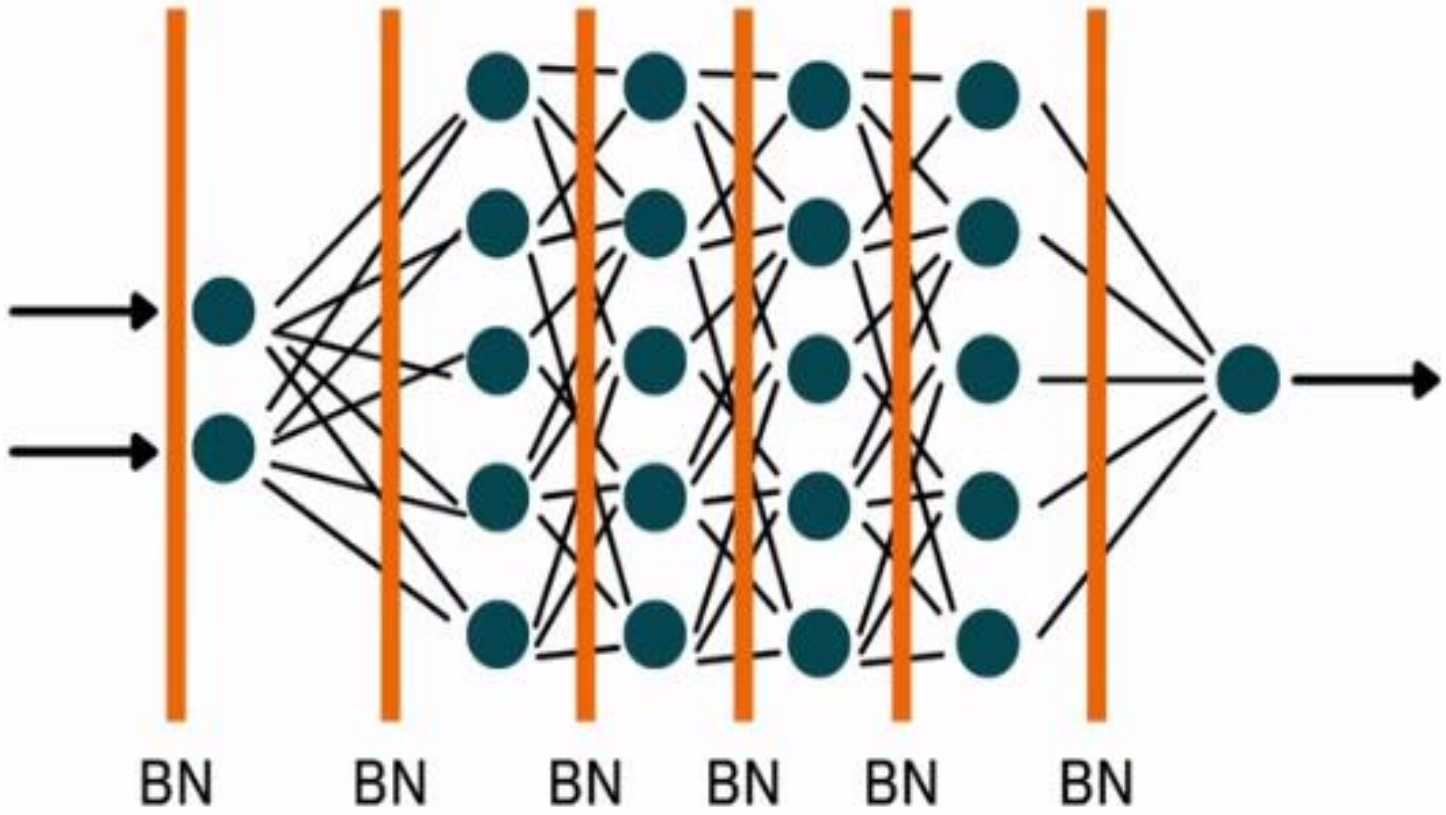


Batch Normalization



Batch Normalization

- Center around zero and normalize the inputs
- Achieves same accuracy faster
- Can lead to better performance
- No need to have a standardization layer
- Reduces the need for other regularization
- Epochs take longer due to the amount of computations but convergence will be faster



NOTEBOOK SAMPLES

```
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)
```

```
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)
```