ANN
Artificial Neural Networks

STUDENT 3
Test: 7/10
Grades: 6/10

QUIZ

Does the student get Accepted?

- Yes
- No

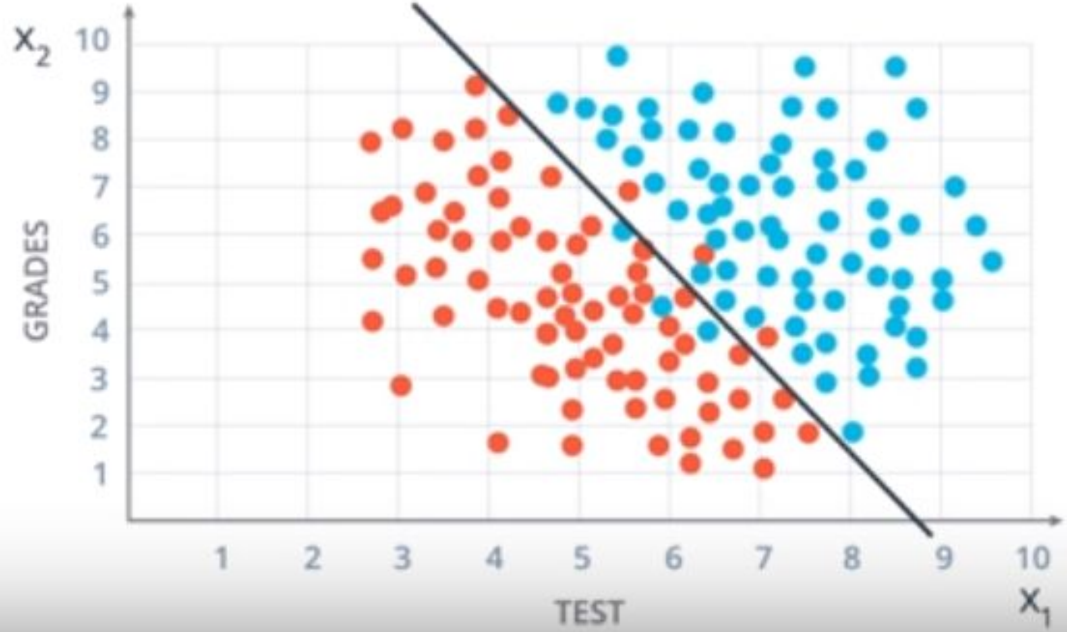
Acceptance at a University

QUIZ

Does the student get Accepted?

- Yes
- No

Acceptance at a University



BOUNDARY:

A LINE

$w_1x_1 + w_2x_2 + b = 0$

$Wx + b = 0$

$W = (w_1, w_2)$

$x = (x_1, x_2)$

y = label: 0 or 1

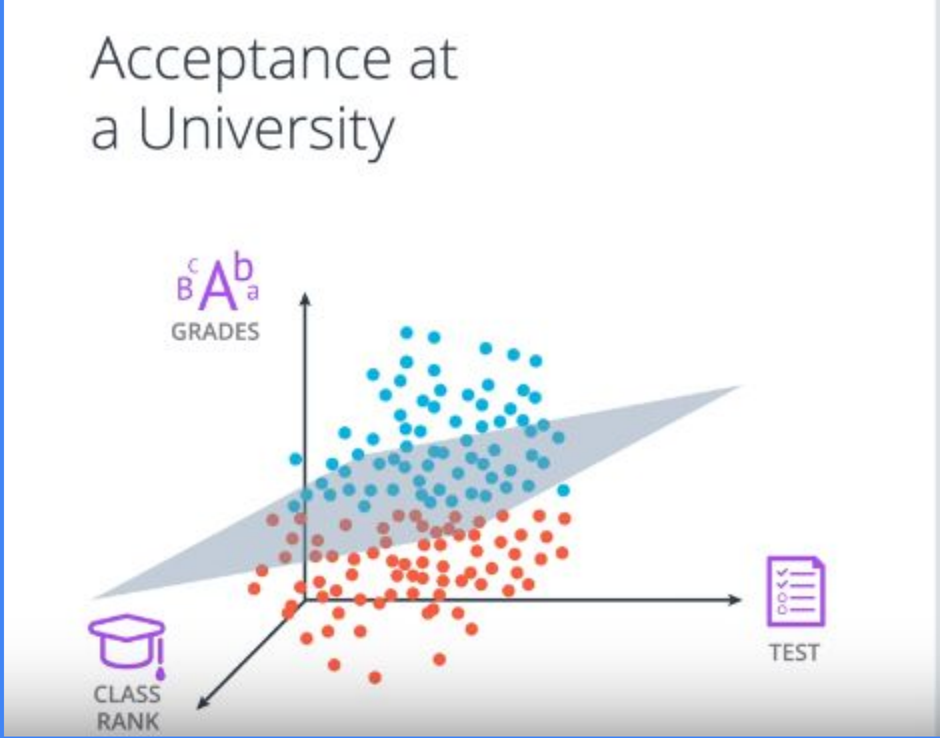
PREDICTION:

$$\hat{y} = \begin{cases} 1 \text{ if } Wx + b \geq 0 \\ 0 \text{ if } Wx + b < 0 \end{cases}$$

Applications of Deep Learning?

- Beating professional players at games like chess, checkers and go
- Detecting spam emails
- Predicting stock prices
- Classifying images
- Diagnosing illnesses
- Self-driving cars

What if we have more than 2 features?



BOUNDARY:
A PLANE

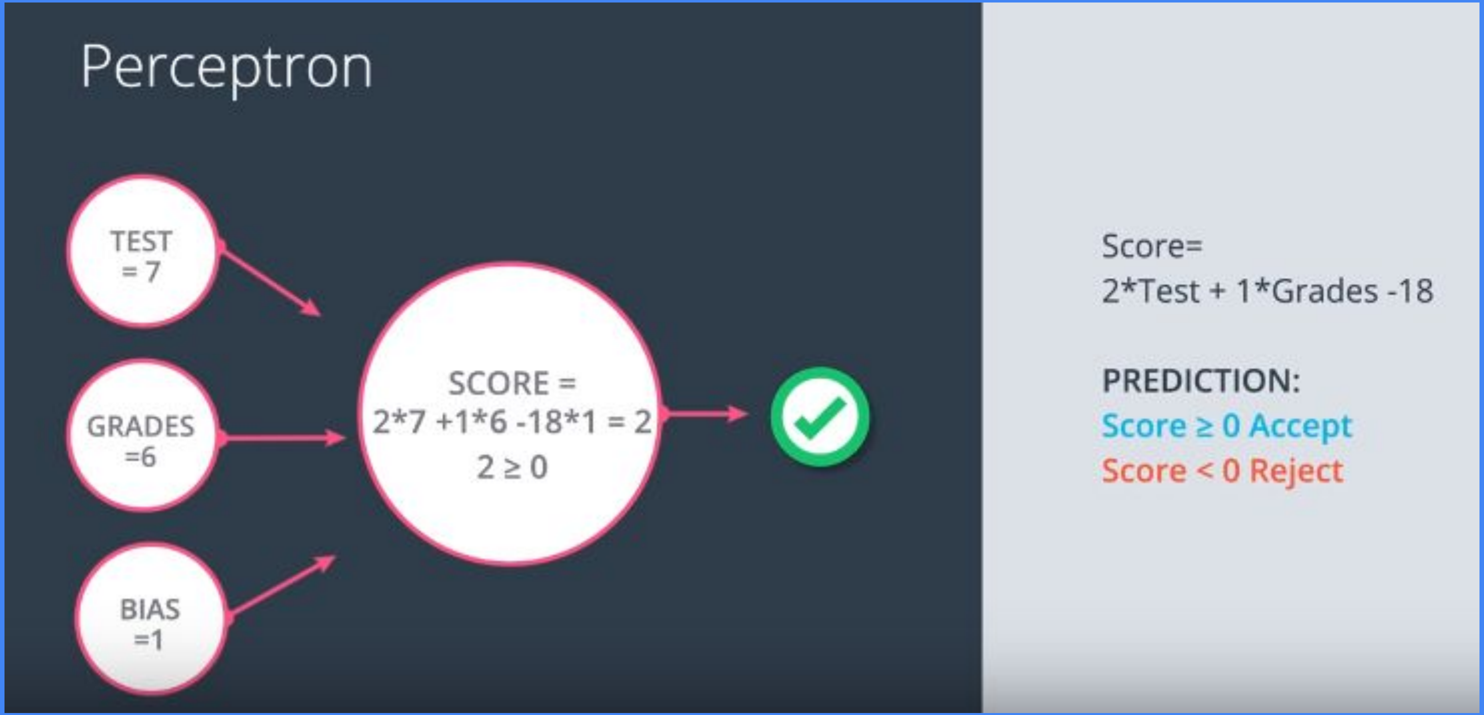
$w_1x_1 + w_2x_2 + w_3x_3 + b = 0$

$Wx + b = 0$

PREDICTION:

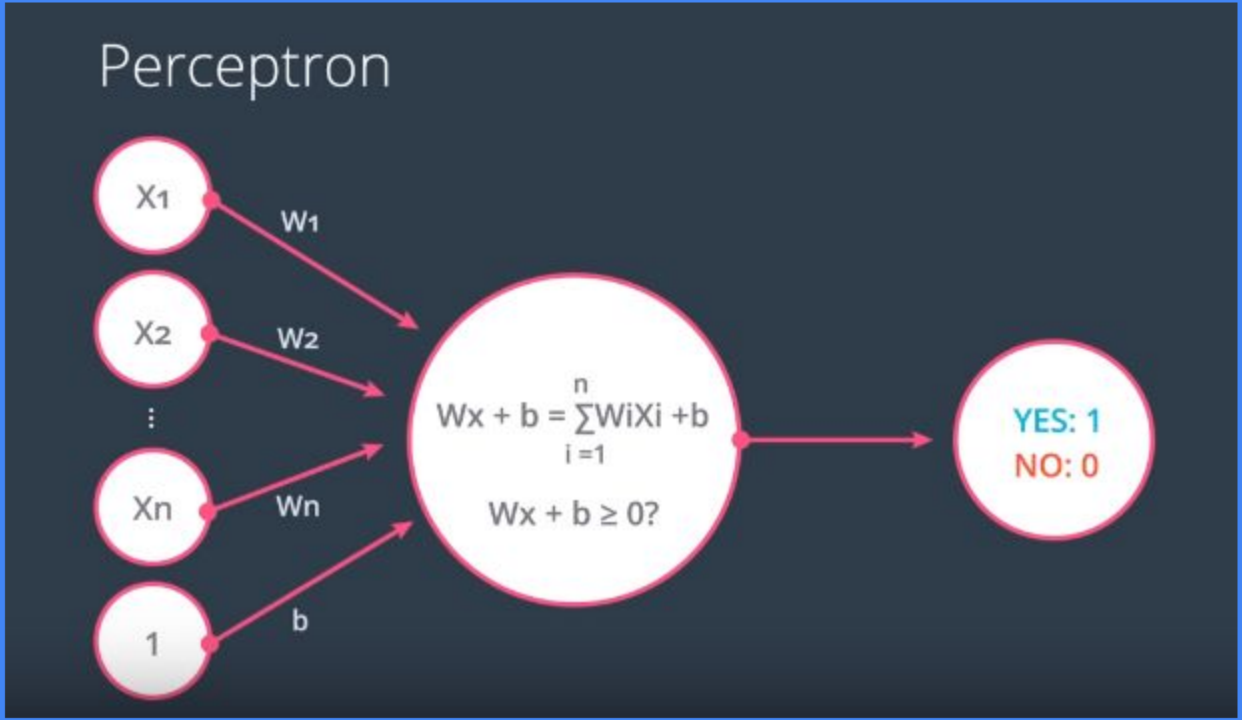
$$\hat{y} = \begin{cases} 1 \text{ if } Wx + b \geq 0 \\ 0 \text{ if } Wx + b < 0 \end{cases}$$

Let’s suppose that the line demarcating the 2 classes is 2*test+1*grades-18

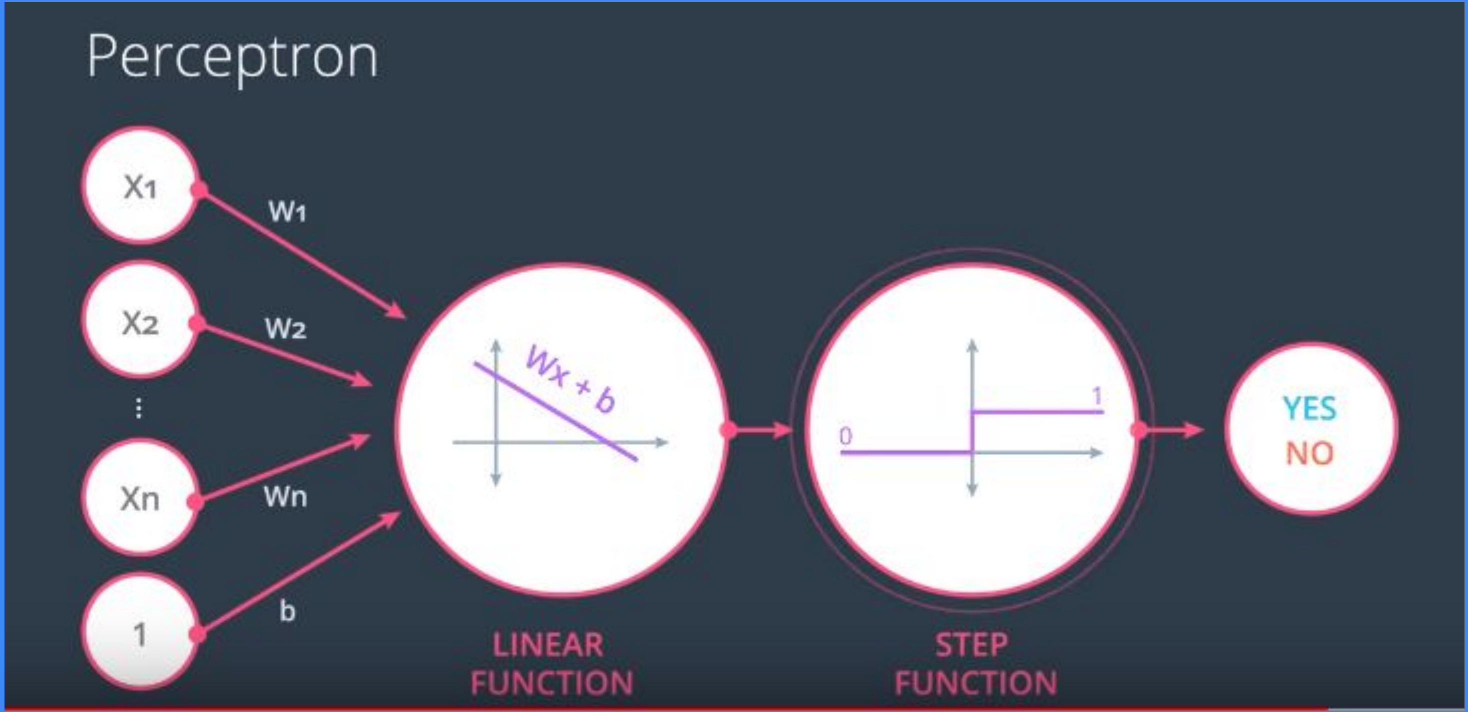


The weights for the edges are 2,1,-18 respectively.

In general, this is how a perceptron model looks like!



We can separate the step of checking Wx+b>0

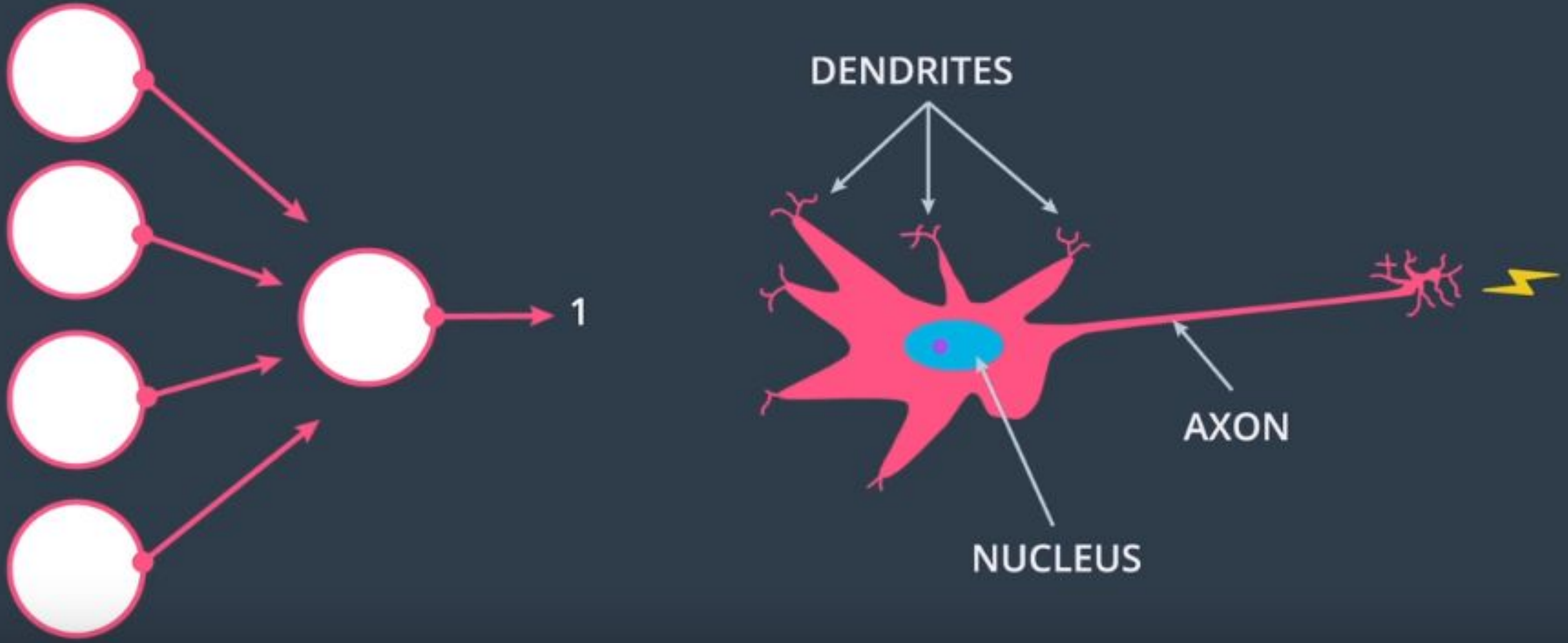


QUIZ QUESTION

Given Score = 2*Test + 1*Grade - 18, suppose w1 is 1.5 instead of 2. Would the student who got 7 on the test and 6 on the grades be accepted or rejected?

ANSWER : REJECTED!

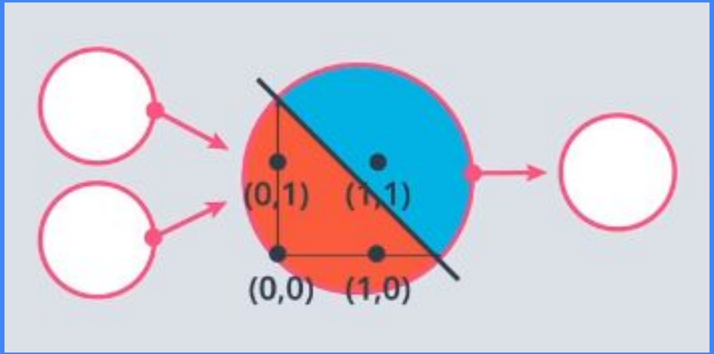
Perceptron



Perceptron as logical operators

FOR AND OPERATION

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

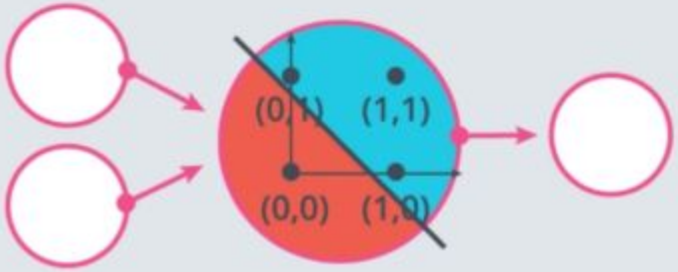


What should be the weights for the perceptron?

ANSWER : weight1 = 1.0, weight2 = 1.0, bias = -1.5

FOR OR PERCEPTRON

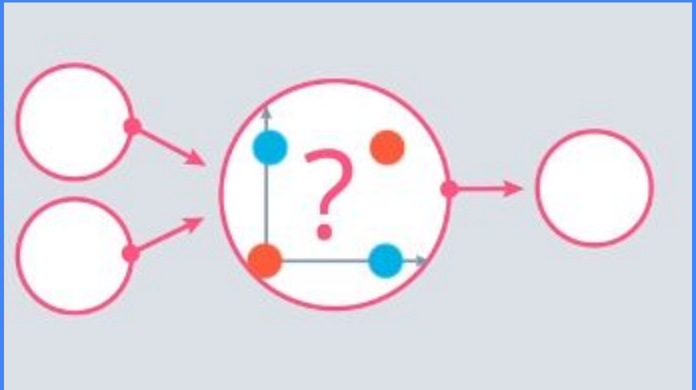
OR Perceptron

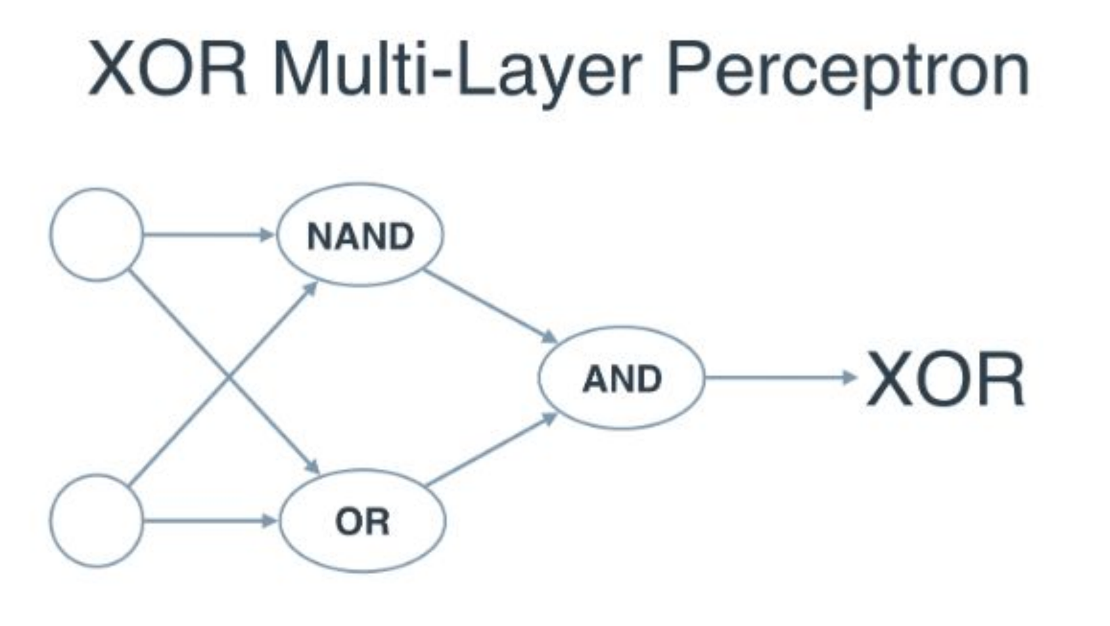


IN	IN	OUT
1	1	1
1	0	1
0	1	1
0	0	0

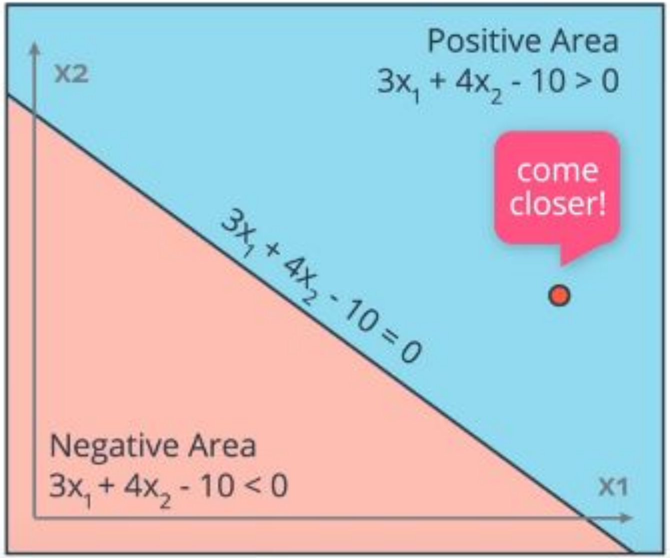
weight1 = 1.0, weight2 = 1.0, bias = -0.5

XOR perceptron





$Y = (AB)'(A+B)$



LINE: $3x_1 + 4x_2 - 10 = 0$

POINT: (4,5)

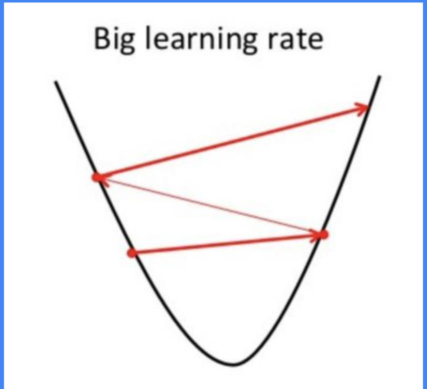
LEARNING RATE: 0.1

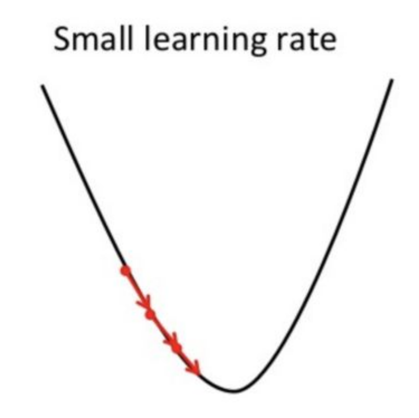
$$
\begin{array}{rccc}
 & 3 & 4 & -10 \\
- & 0.4 & 0.5 & 0.1 \\
\hline
 & 2.6 & 3.5 & -10.1
\end{array}
$$

NEW LINE

$2.6x_1 + 3.5x_2 - 10.1 = 0$

Learning rate





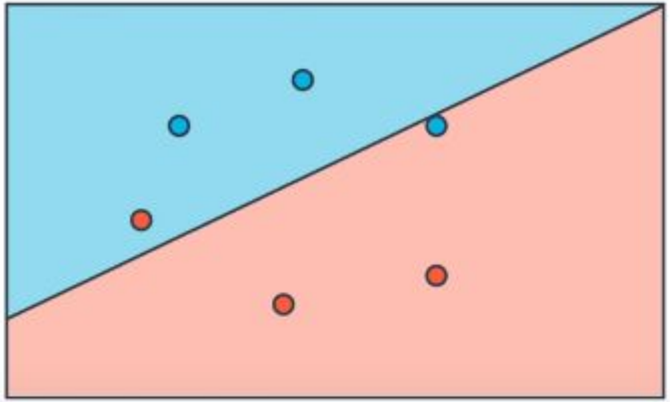
It determines how fast or slow we will move towards the optimal weights.

- If it is **too big**, it maybe will not reach the local minimum because it just bounces back and forth.

- If it is **very small**, gradient descent will eventually reach the local minimum in a long time.

Keep learning rate :neither too low nor too high.

Perceptron Algorithm



1. Start with random weights: $w_1, \ldots, w_n, b$

2. For every misclassified point $(x_1,\ldots,x_n)$:

   2.1. If prediction = 0:
   - For i = 1 ...n
     - Change $w_i + \alpha x_i$
   - Change b to $b + \alpha$

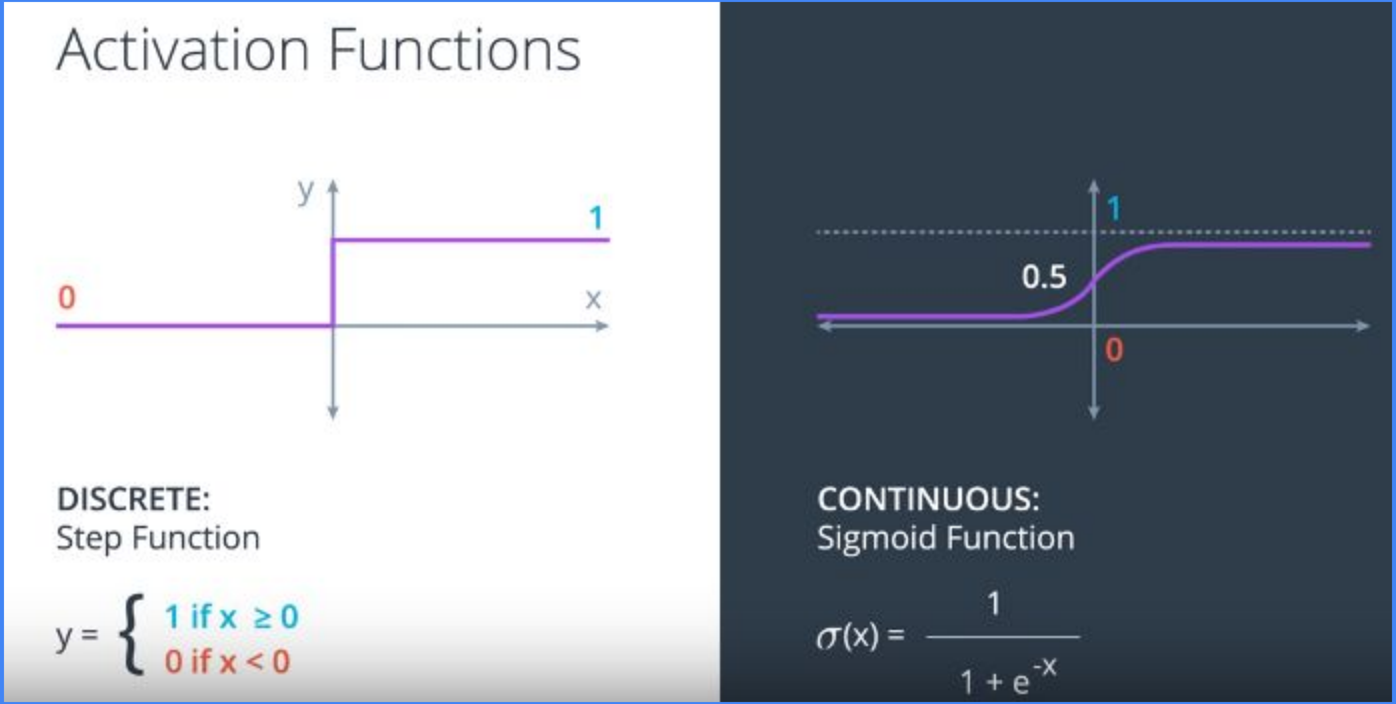
   2.2. If prediction = 1:
   - For i = 1 ...n
     - Change $w_i - \alpha x_i$
   - Change b to $b - \alpha$

Error Function

The deviation of the actual output from the predicted output

Tells the model how badly it is performing

Predictions


Activation Functions
y
1
0
x
DISCRETE:
Step Function
$y = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$
1
0.5
0
CONTINUOUS:
Sigmoid Function
$\sigma(x) = \frac{1}{1 + e^{-x}}$

The sigmoid function is defined as sigmoid(x) = 1/(1+e-x). If the score is defined by 4x1 + 5x2 - 9 = score, then which of the following points has exactly a 50% probability of being blue or red?

a) (1,1)
b) (2,4)
c) (5,-5)

ANSWER: (1,1)

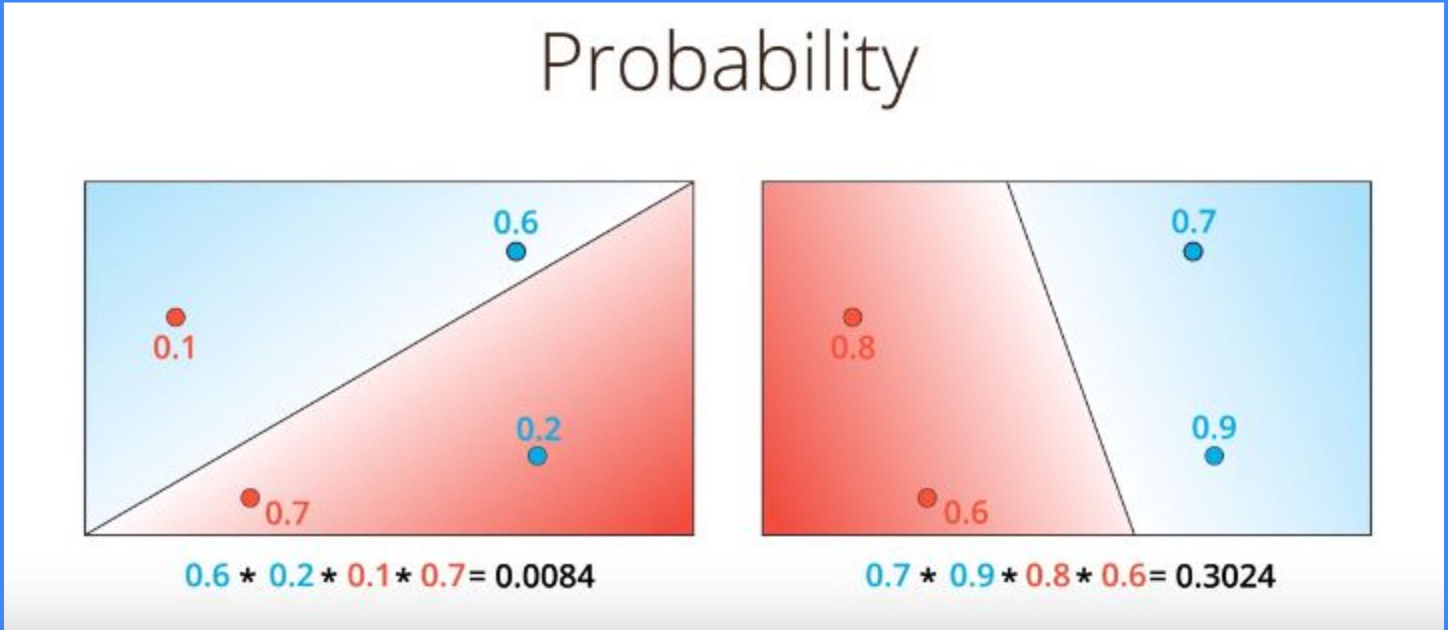
Multi class classification

Softmax Function

LINEAR FUNCTION SCORES:

$Z_1, \ldots, Z_n$

$$P(\text{class } i) = \frac{e^{Z_i}}{e^{Z_1} + \ldots + e^{Z_n}}$$

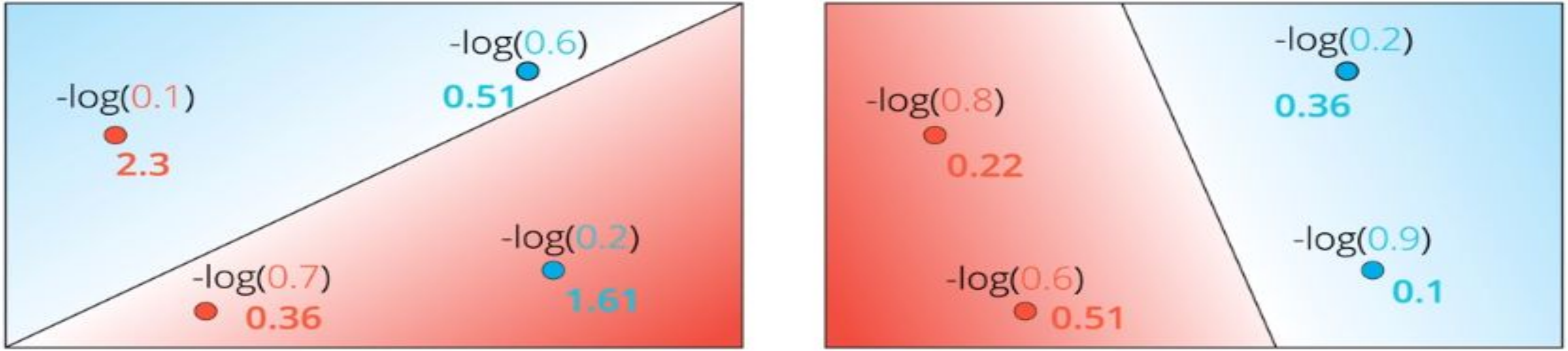

Probability
0.6
0.1
0.2
0.7
0.6 * 0.2 * 0.1 * 0.7 = 0.0084
0.7
0.8
0.9
0.6
0.7 * 0.9 * 0.8 * 0.6 = 0.3024

What function turns products into sums?

a) Sin
b) Cos
c) Log
d) exp

ANSWER : LOG!

Cross Entropy



$0.6 * 0.2 * 0.1 * 0.7 = 0.0084$

$-\log(0.6) - \log(0.2) - \log(0.1) - \log(0.7) = 4.8$

$0.7 * 0.9 * 0.8 * 0.6 = 0.3024$

$-\log(0.7) - \log(0.9) - \log(0.8) - \log(0.6) = 1.2$

$$\text{Cross-Entropy} = -\sum_{i=1}^{m} y_i \ln(p_i) + (1 - y_i)\ln(1 - p_i)$$

Logistic Regression

It is one of the most popular and useful algorithms in Machine Learning, and the building block of all that constitutes Deep Learning. It basically goes like this:

- Take your data
- Pick a random model
- Calculate the error
- Minimize the error, and obtain a better model

Gradient Descent

1. Start with random weights:

   $w_1, \ldots, w_n, b$

2. For every point $(x_1, \ldots, x_n)$:

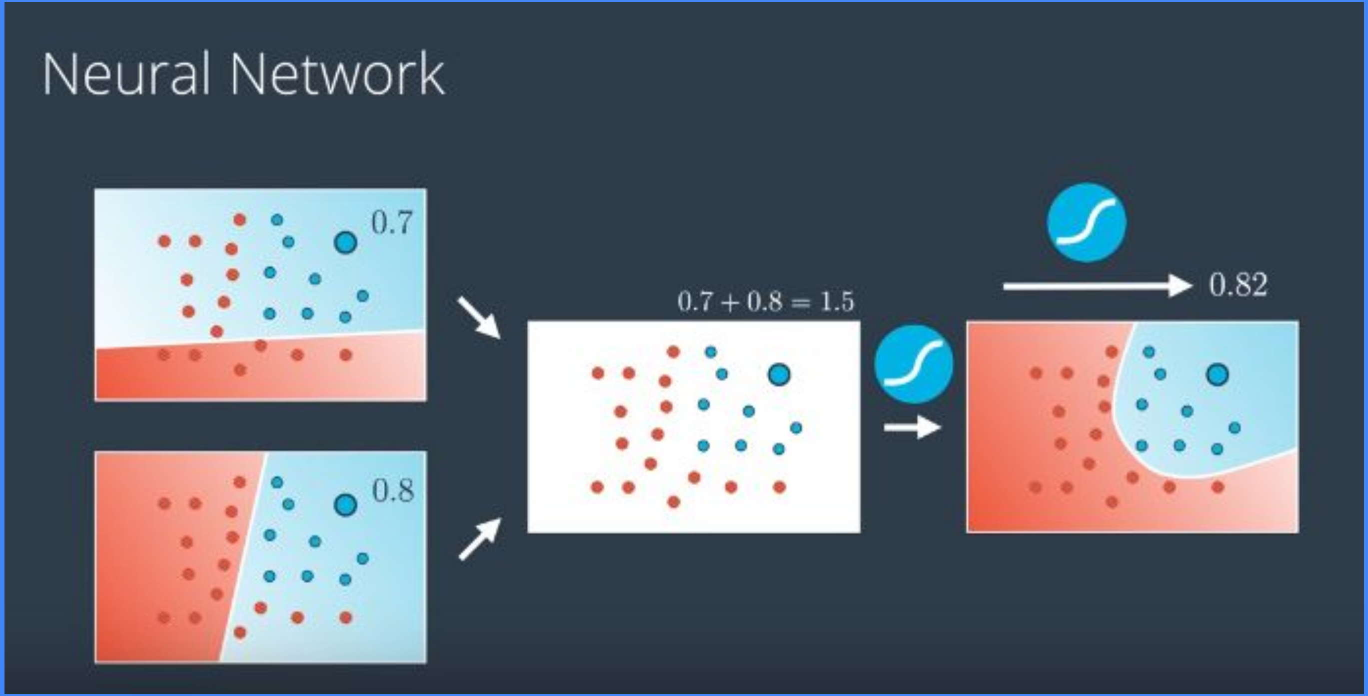
   2.1. For $i = 1 \ldots n$

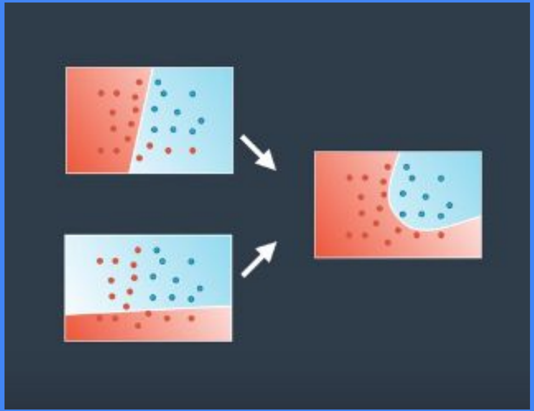
   2.1.1. Update $w_i' \longleftarrow w_i - \alpha(\hat{y}-y)x_i$

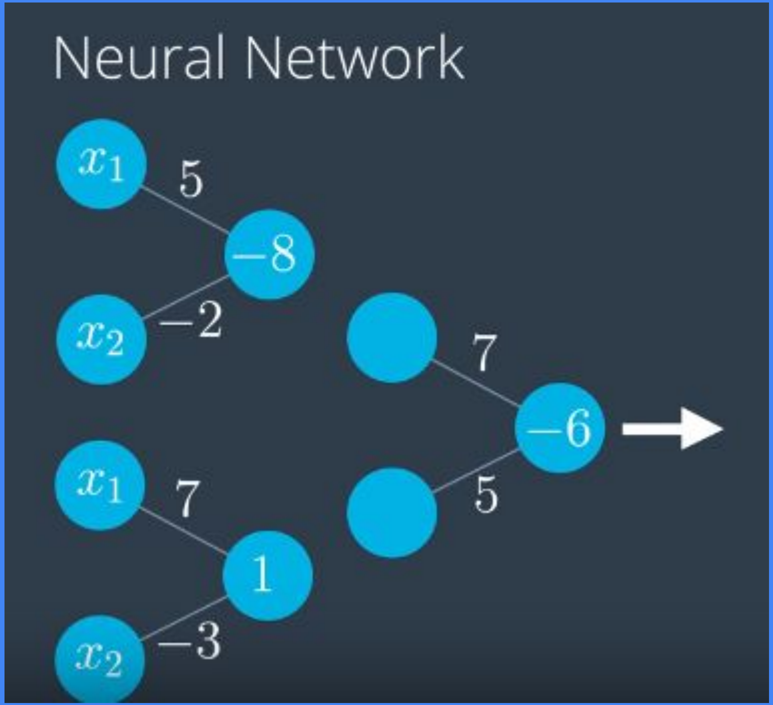
   2.1.2. Update $b' \longleftarrow b - \alpha(\hat{y}-y)$

3. Repeat until error is small

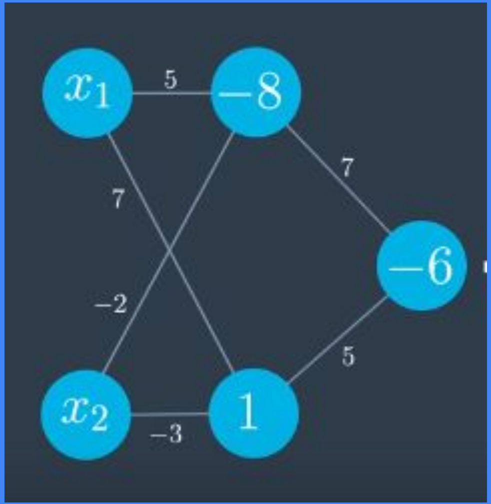
What if the data is not linearly separable?





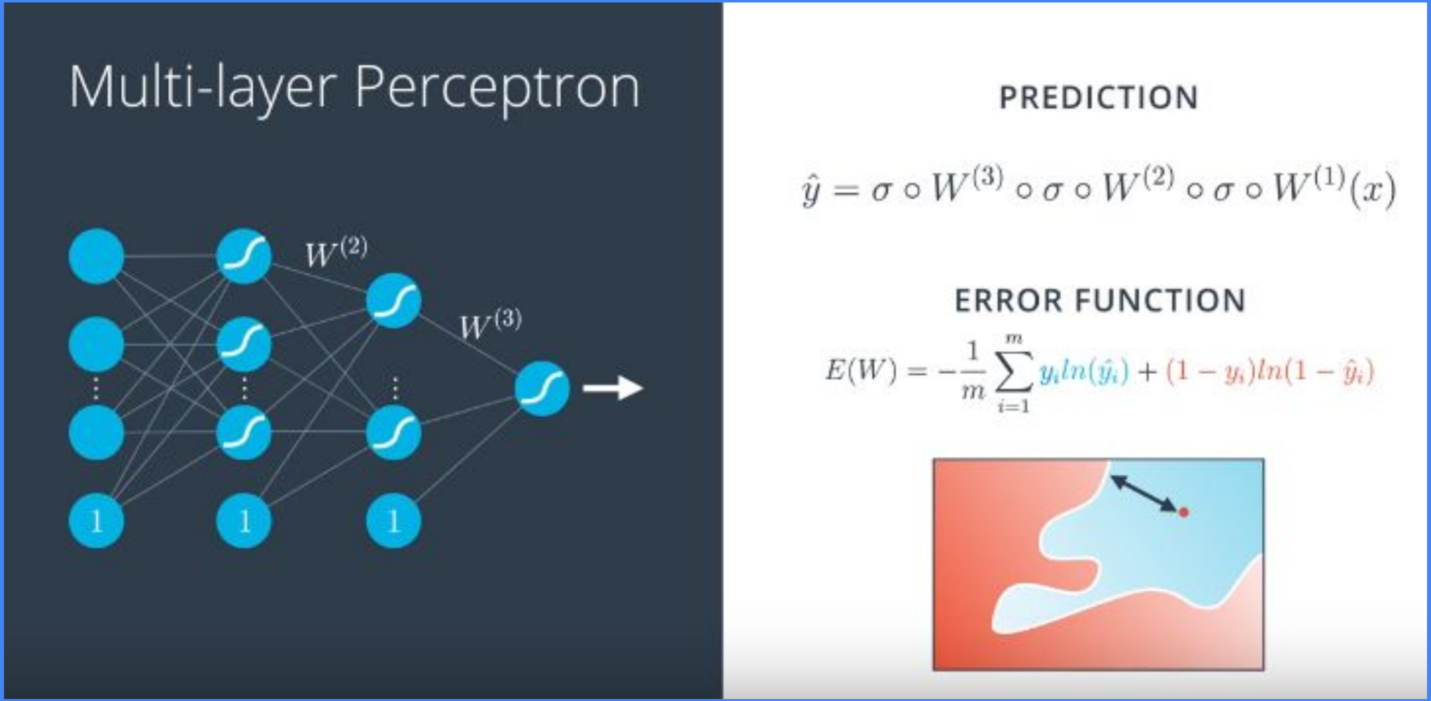


Neural Network
x_1
5
−8
x_2
−2
7
−6
x_1
7
1
x_2
−3
5

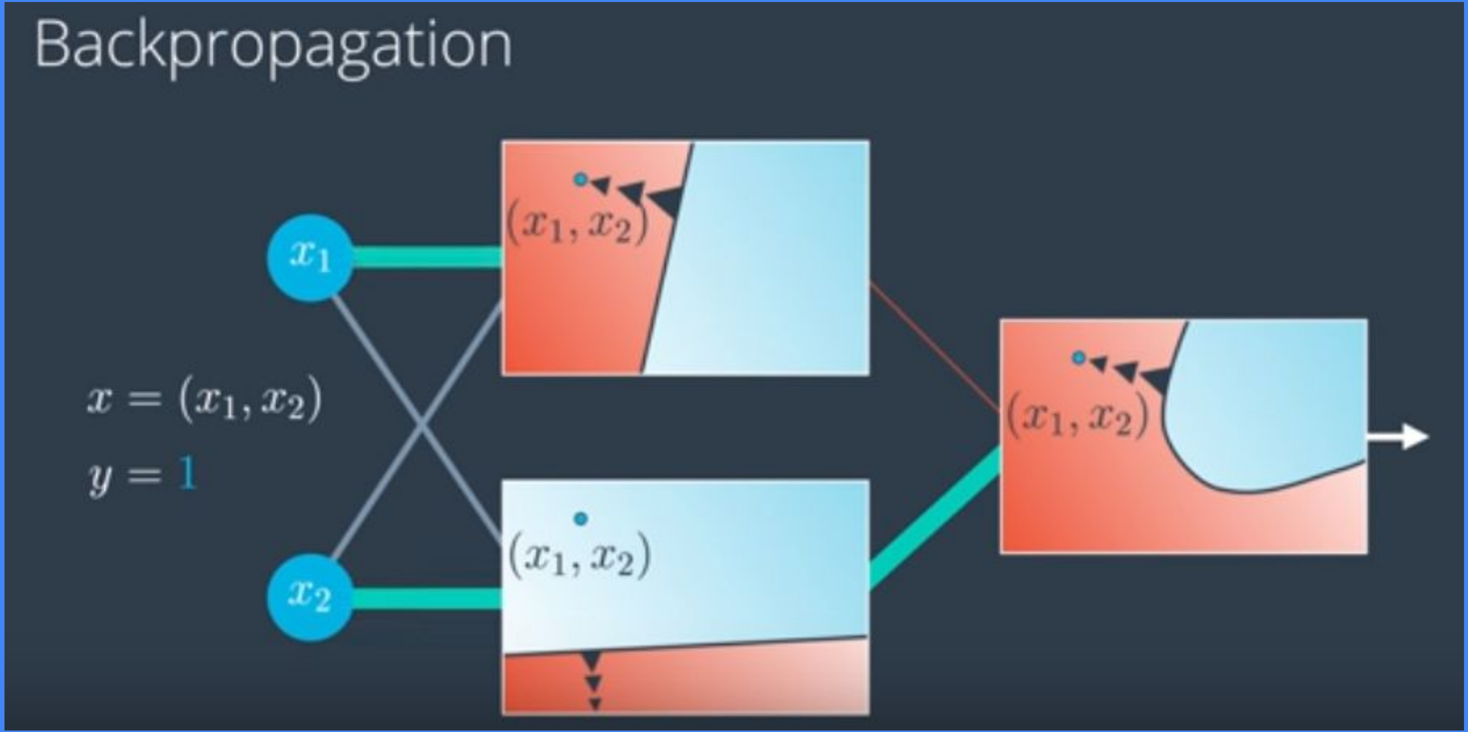


x_1
5
−8
7
7
−2
−6
x_2
−3
1
5

Feedforward Phase



Backpropagation



Multi-layer Perceptron

PREDICTION

$$\hat{y} = \sigma W^{(3)} \circ \sigma W^{(2)} \circ \sigma \circ W^{(1)}(x)$$

ERROR FUNCTION

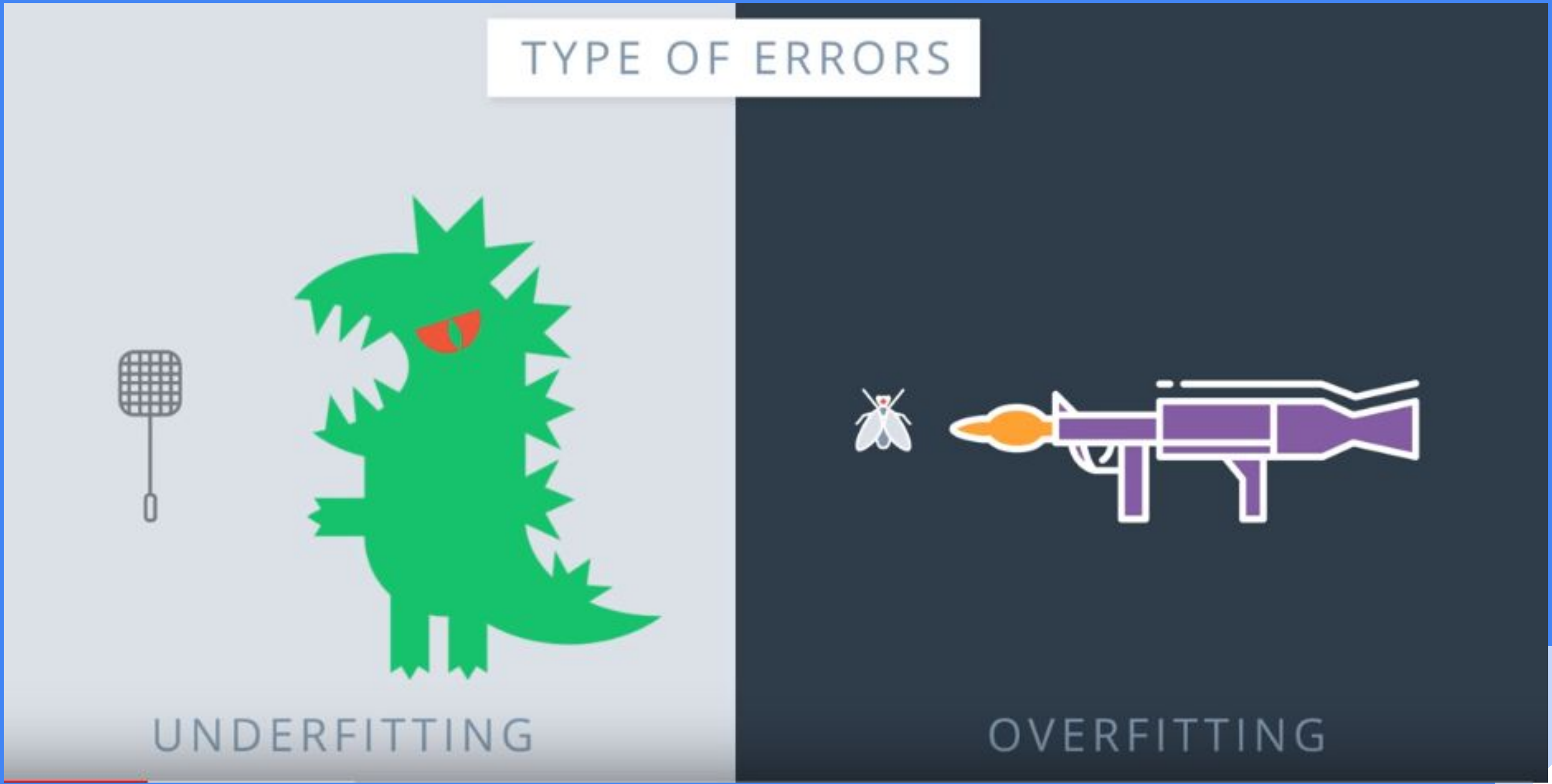
$$E(W) = -\frac{1}{m}\sum_{i=1}^{m} y_i ln(\hat{y}_i) + (1 - y_i) ln(1 - \hat{y}_i)$$

GRADIENT OF THE ERROR FUNCTION

$$\nabla E = (\ldots, \frac{\partial E}{\partial w_j^{(i)}}, \ldots)$$

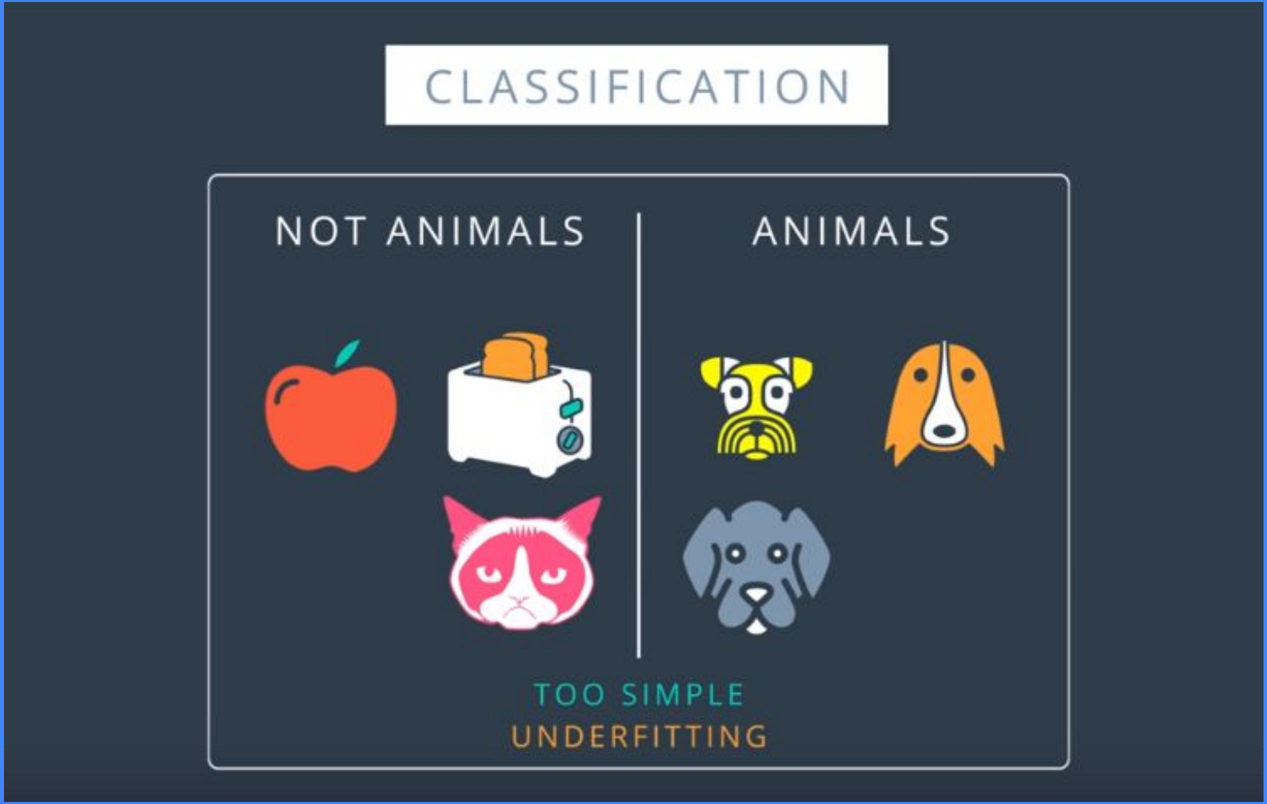
$$W_{ij}'^{(k)} \leftarrow W_{ij}^{(k)} - \alpha \frac{\partial E}{\partial W_{ij}^{(k)}}$$

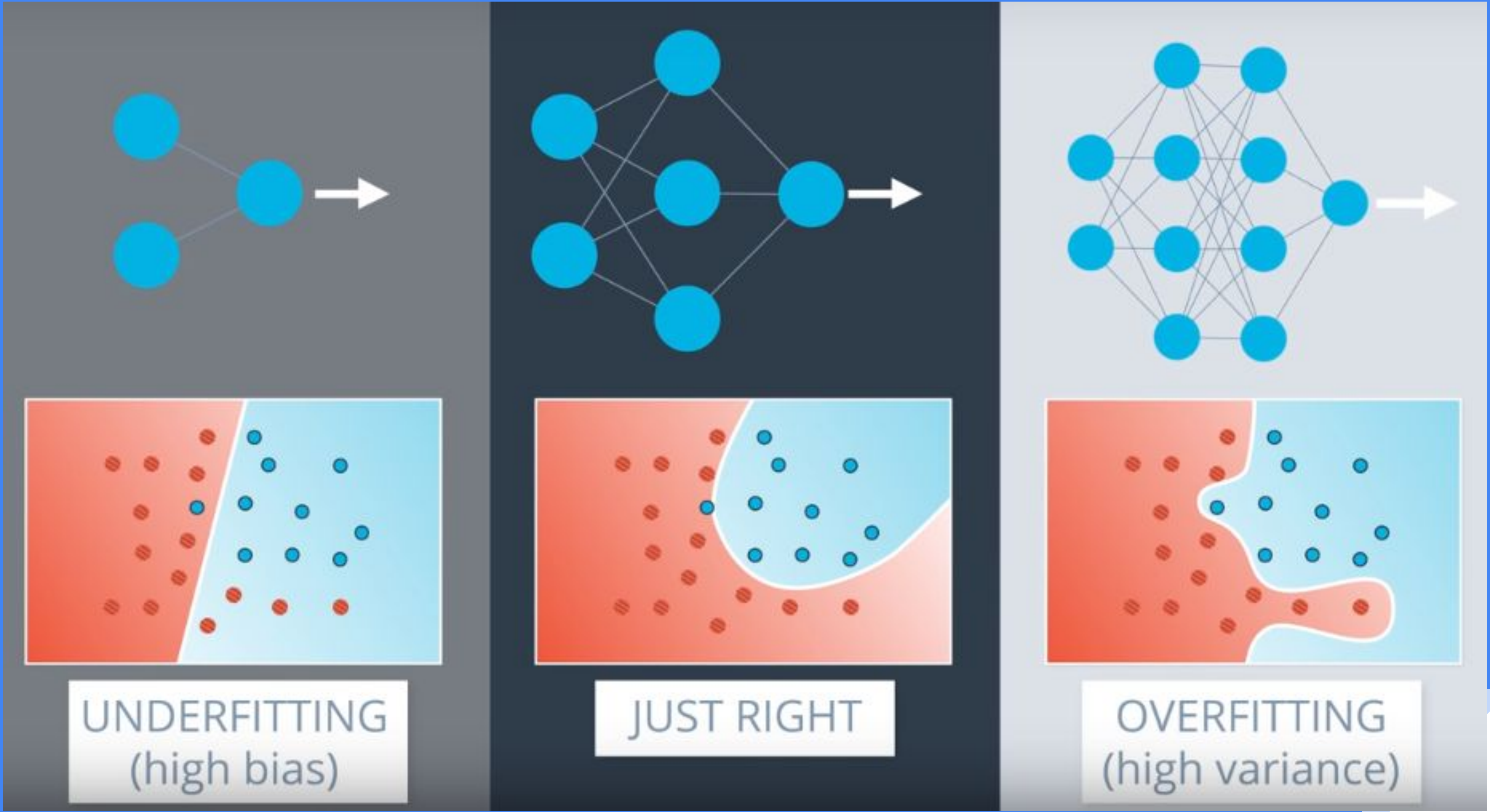
Overfitting vs Underfitting

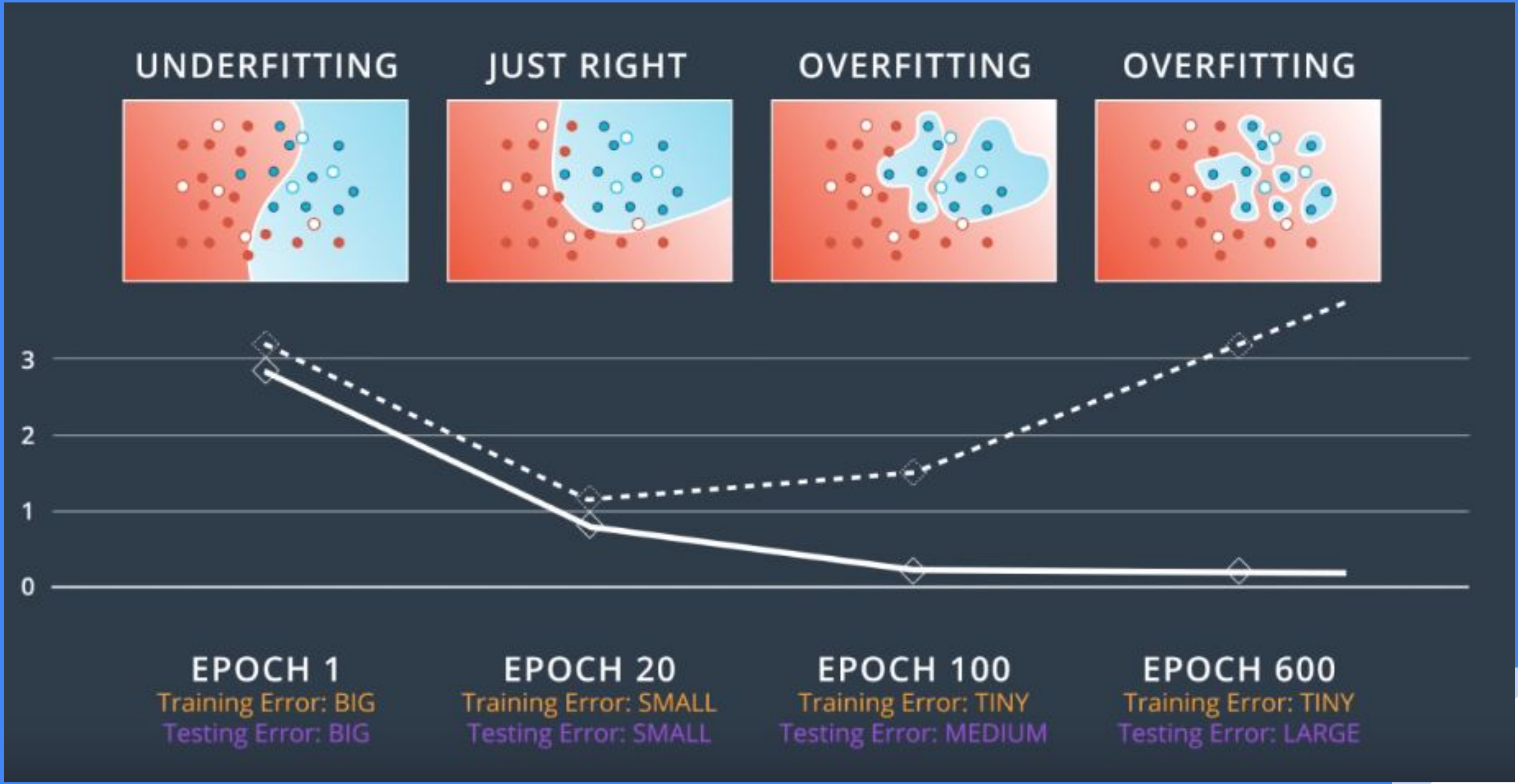


Overfitting

Underfitting




UNDERFITTING
(high bias)
JUST RIGHT
OVERFITTING
(high variance)

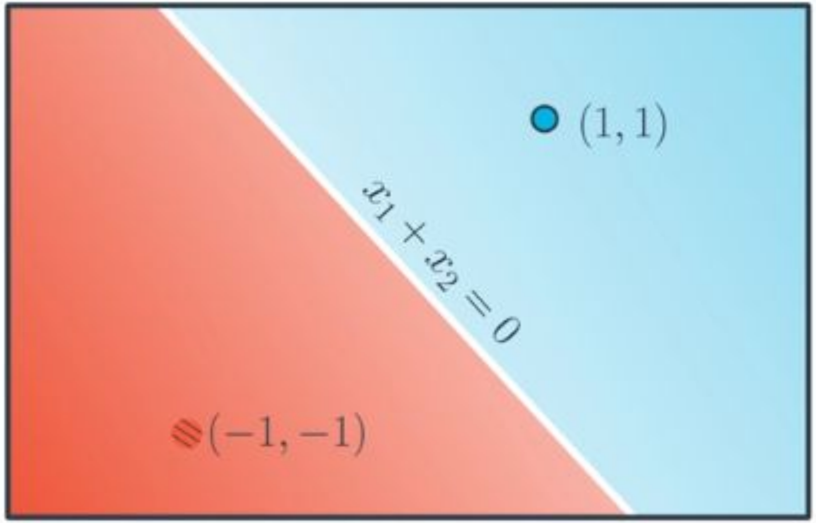

UNDERFITTING
JUST RIGHT
OVERFITTING
OVERFITTING
3
2
1
0
EPOCH 1
Training Error: BIG
Testing Error: BIG
EPOCH 20
Training Error: SMALL
Testing Error: SMALL
EPOCH 100
Training Error: TINY
Testing Error: MEDIUM
EPOCH 600
Training Error: TINY
Testing Error: LARGE

Goal: Split Two Points

QUIZ: WHICH GIVES A SMALLER ERROR?

- SOLUTION 1: $x_1 + x_2$
- SOLUTION 2: $10x_1 + 10x_2$

Goal: Split Two Points



QUIZ: WHICH GIVES A SMALLER ERROR?

Prediction: $\hat{y} = \sigma(w_1 x_1 + w_2 x_2 + b)$

- SOLUTION 1: $x_1 + x_2$

  Predictions:

  $\sigma(1 + 1) = 0.88$

  $\sigma(-1 - 1) = 0.12$

- SOLUTION 2: $10x_1 + 10x_2$

  Predictions:

  $\sigma(10 + 10) = 0.9999999979$

  $\sigma(-10 - 10) = 0.0000000021$

Solution: Regularization

LARGE COEFFICIENTS → OVERFITTING

PENALIZE LARGE WEIGHTS

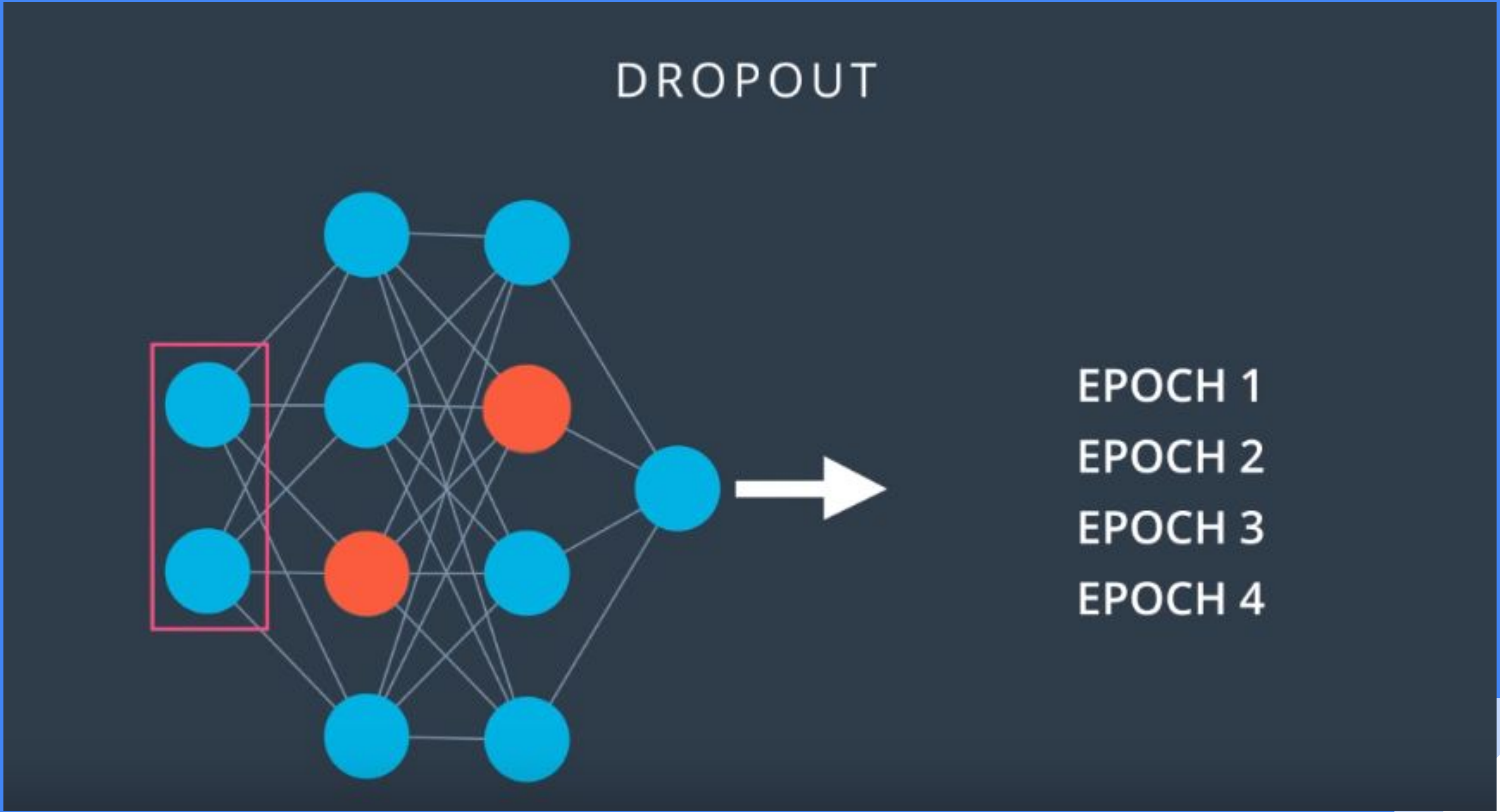
$$(w_1, \ldots, w_n)$$

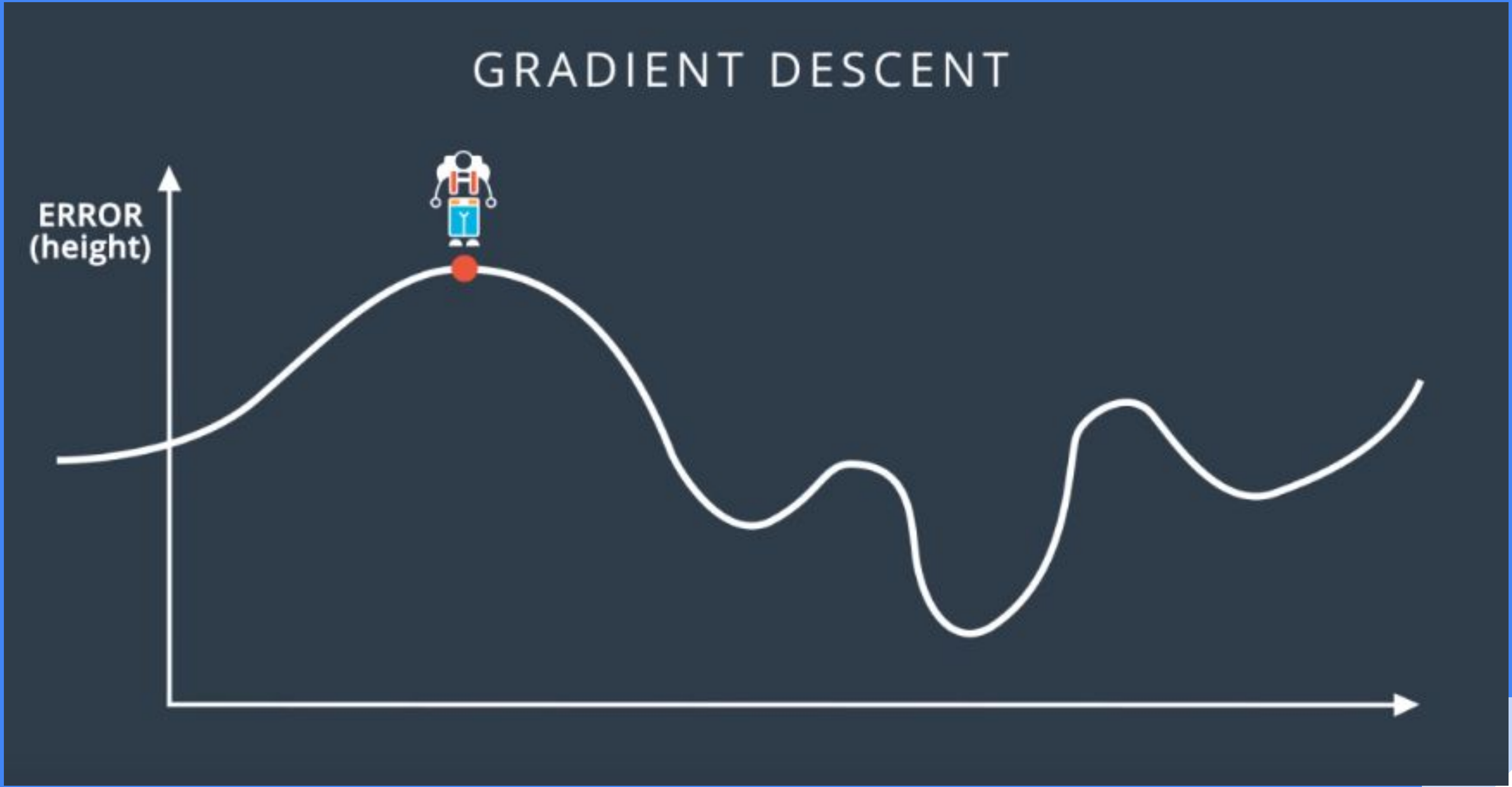
L1 ERROR FUNCTION $= -\frac{1}{m}\sum_{i=1}^{m}(1-y_i)ln(1-\hat{y}_i) + y_i ln(\hat{y}_i) + \boxed{\lambda(|w_1| + \ldots + |w_n|)}$

L2 ERROR FUNCTION $= -\frac{1}{m}\sum_{i=1}^{m}(1-y_i)ln(1-\hat{y}_i) + y_i ln(\hat{y}_i) + \boxed{\lambda({w_1}^2 + \ldots + {w_n}^2)}$

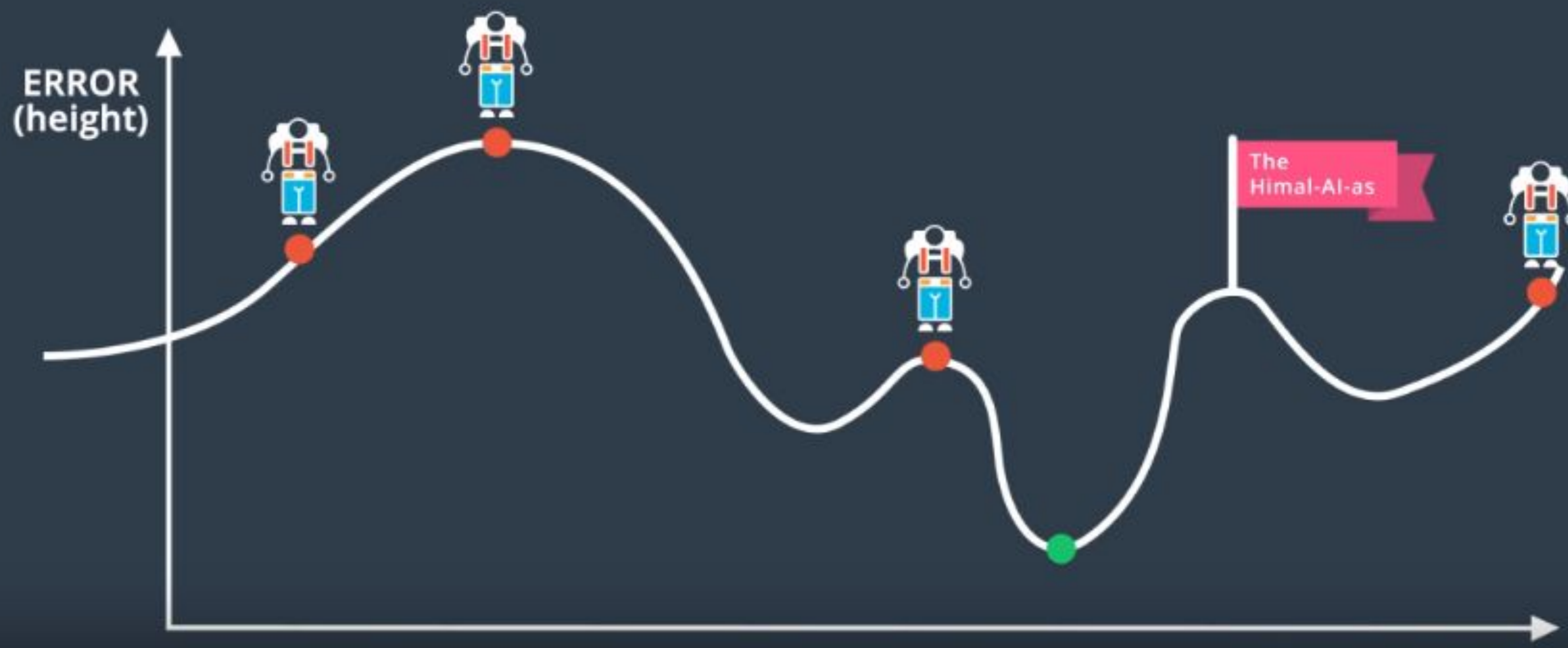

SPORTS

DROPOUT

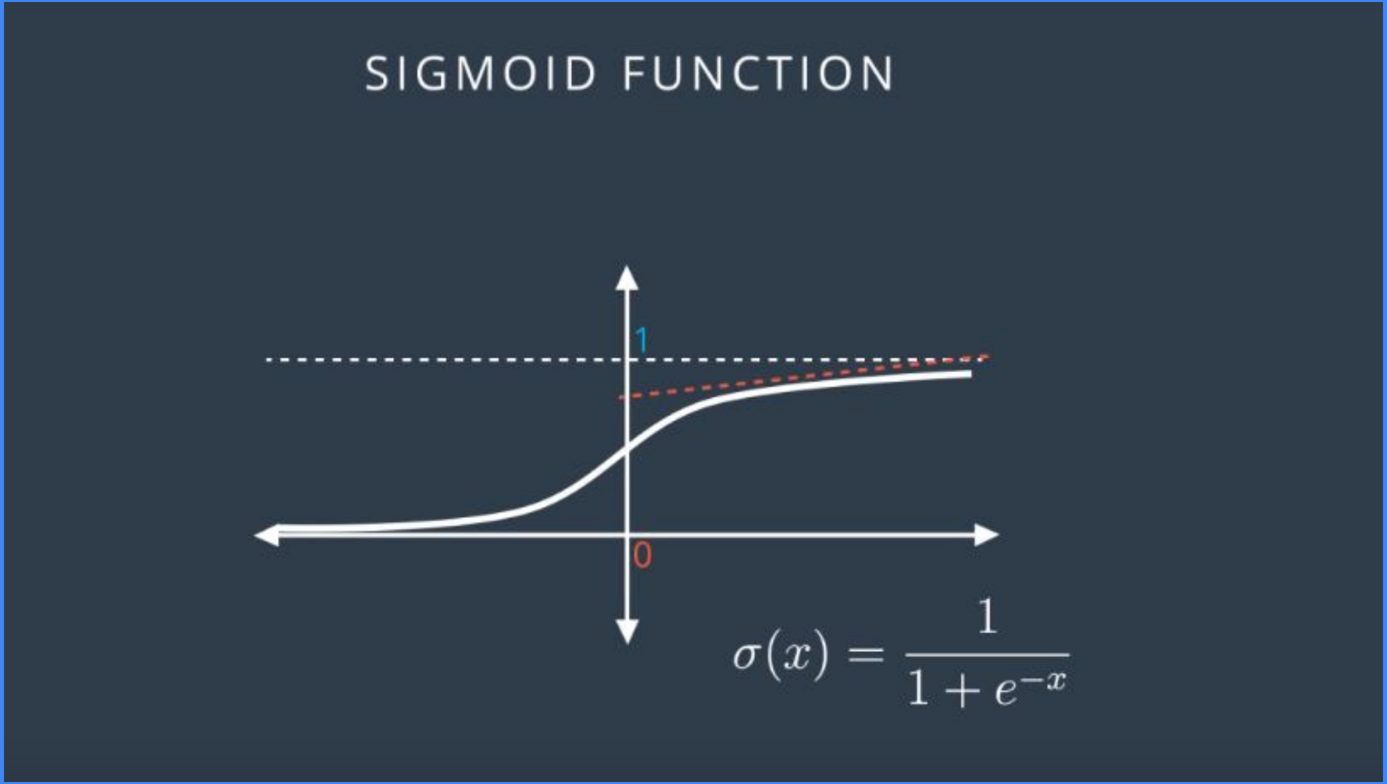

DROPOUT
EPOCH 1
EPOCH 2
EPOCH 3
EPOCH 4


GRADIENT DESCENT
ERROR
(height)

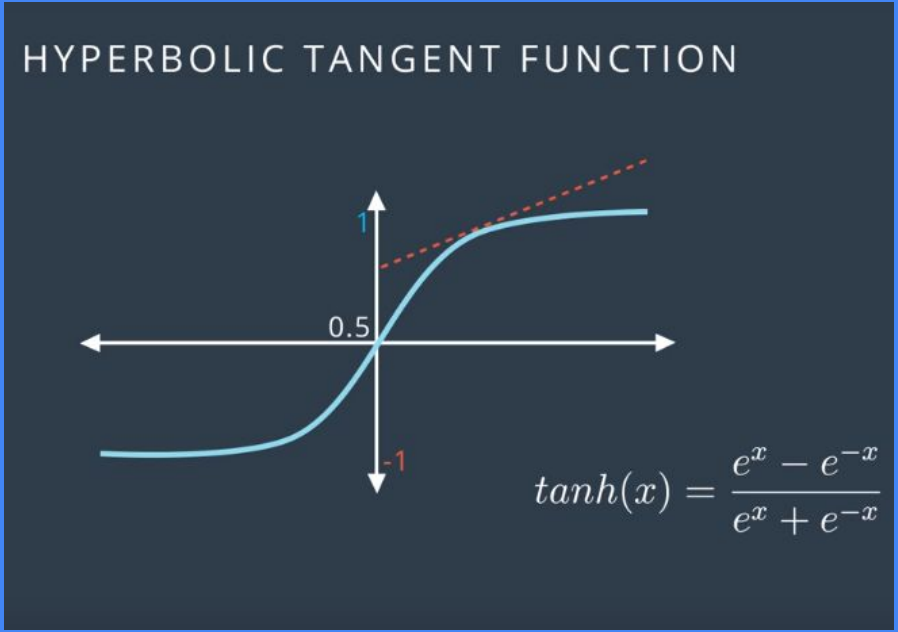
RANDOM RESTART



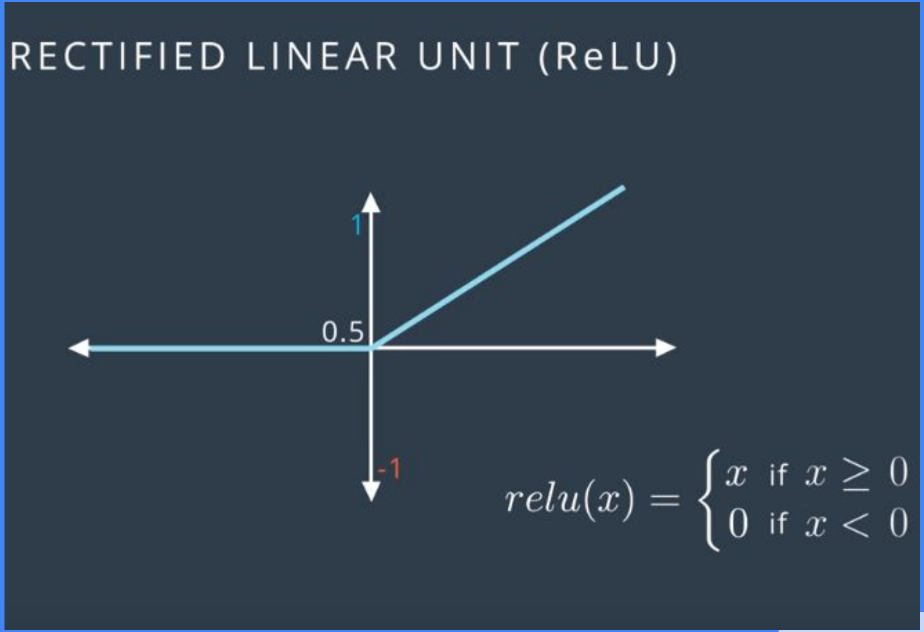
Vanishing Gradient



Solution :



HYPERBOLIC TANGENT FUNCTION
1
0.5
-1
$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



RECTIFIED LINEAR UNIT (ReLU)
1
0.5
-1
$relu(x) = \begin{cases} x \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$

Batch Gradient Descent

Batch Gradient Descent **calculates the error for each example** within the training dataset and after all training examples have been evaluated, the model gets updated. This whole process is like a cycle and called a training epoch.

Advantages

- Computational efficient
- It produces a stable error gradient
- A stable convergence

Disadvantage

- Requires a lot of training epochs
- It also requires that the entire training dataset is in memory

Stochastic Gradient Descent

Stochastic gradient descent **calculates error for each training example** within the dataset. It updates the parameters for each training example, one by one.

Advantage

- The frequent updates results in a detailed rate of improvement.
- Requires less number of training epochs.

Disadvantages

- Computationally expensive
- The frequency of those updates can also result in noisy gradients, which may cause the error rate to jump around, instead of slowly decreasing.
- It might not reach global minimum

Mini Batch Gradient Descent

It is a **combination of Stochastic Gradient Descent and Batch Gradient Descent**. It splits the training dataset into small batches and performs an update for each of these batches.

- It creates a balance between the robustness of Stochastic Gradient Descent and the efficiency of Batch Gradient Descent.
- Common mini-batch sizes range between 50 and 256.
- It is the most common type of gradient descent algorithm.