



# **Introduction to Neural Networks**

Prof Cengiz Öztireli

# Parametric approach

Image



W \* H \* 3 numbers

$$f(x, W)$$

Cat	0.1
Dog	0.8
Person	0.02
Road	0.01
Tree	0.01
Airplane	0.01
Apple	0.01
Cell phone	0.03
Cup	0
Table	0.01

10 numbers for class scores

# Parametric approach

If  $W = 32, H = 32,$   
 $x = 32 * 32 * 3 = 3072$   $\longrightarrow f(x, W) = Wx$

Image



$W * H * 3$  numbers

$f(x, W)$

Cat	0.1
Dog	0.8
Person	0.02
Road	0.01
Tree	0.01
Airplane	0.01
Apple	0.01
Cell phone	0.03
Cup	0
Table	0.01

10 numbers for class scores

# Parametric approach

If  $W = 32, H = 32,$   
 $x = 32 * 32 * 3 = 3072$

$$\longrightarrow f(x, W) = Wx \quad 3072 \times 1$$

$10 \times 1 \quad 10 \times 3072$

Image



$W * H * 3$  numbers

$$f(x, W) \longrightarrow$$

Cat	0.1
Dog	0.8
Person	0.02
Road	0.01
Tree	0.01
Airplane	0.01
Apple	0.01
Cell phone	0.03
Cup	0
Table	0.01

10 numbers for class scores

# Parametric approach

If  $W = 32, H = 32,$   
 $x = 32 * 32 * 3 = 3072$

$$\longrightarrow f(x, W) = Wx + b$$

$10 \times 1$                        $10 \times 1$

Image



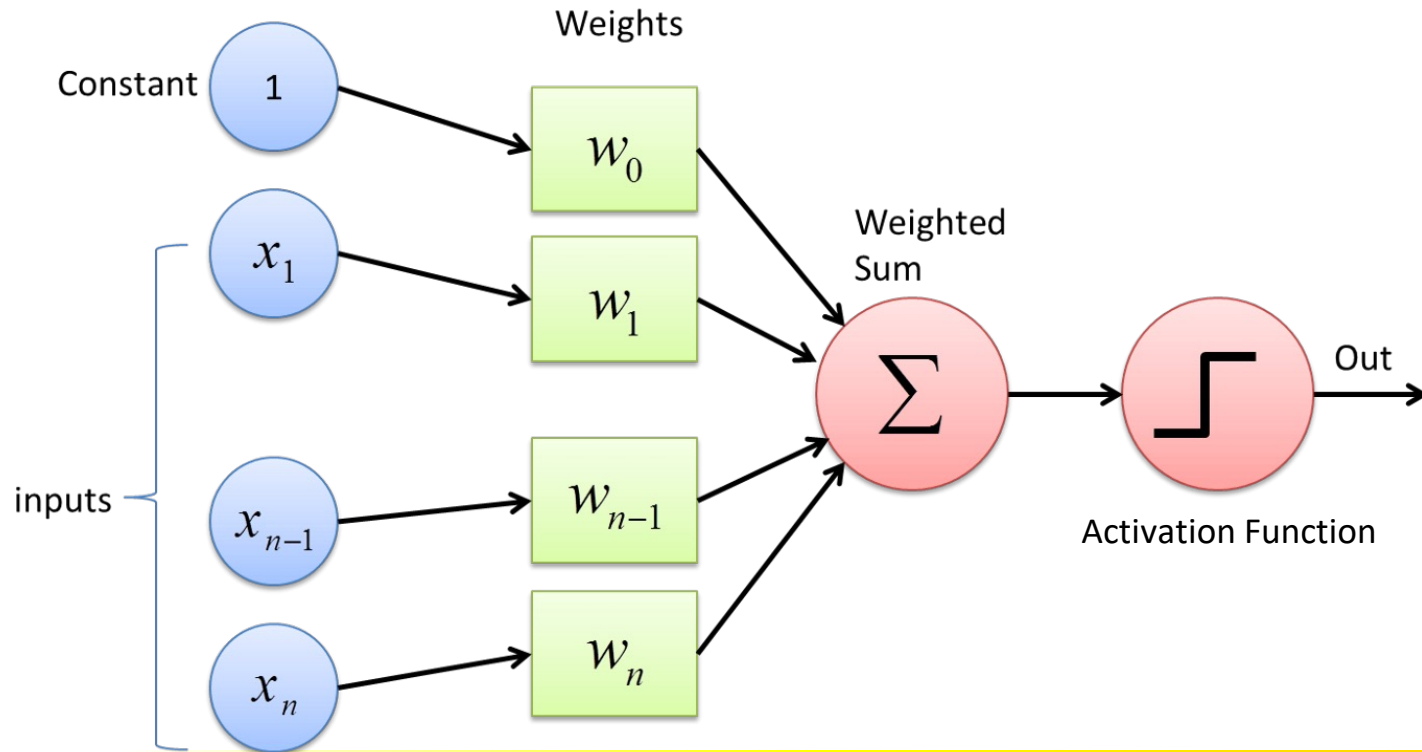
$W * H * 3$  numbers

$$f(x, W) \longrightarrow$$

Cat	0.1
Dog	0.8
Person	0.02
Road	0.01
Tree	0.01
Airplane	0.01
Apple	0.01
Cell phone	0.03
Cup	0
Table	0.01

10 numbers for class scores

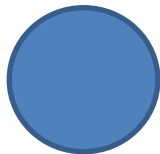
# Perceptron



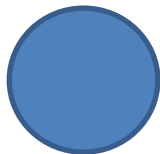
# Two-layer Perceptron

Input

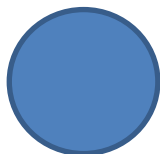
$a_1$



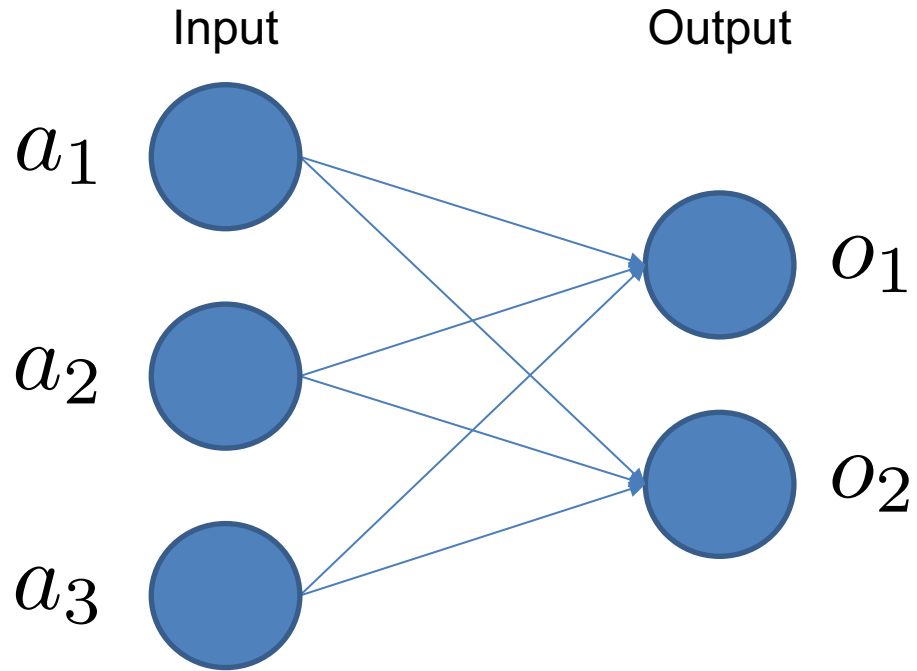
$a_2$



$a_3$

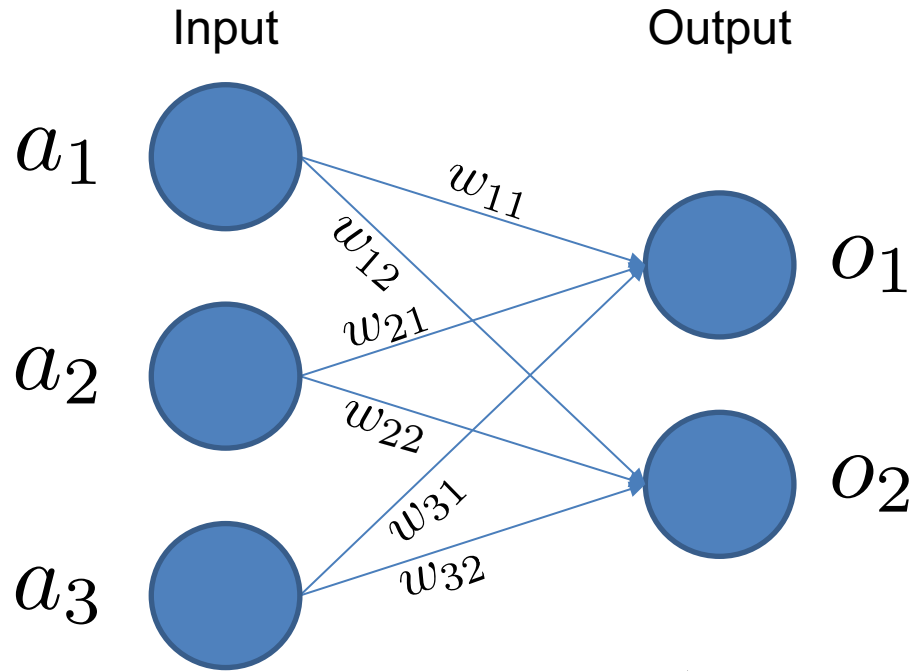


# Two-layer Perceptron



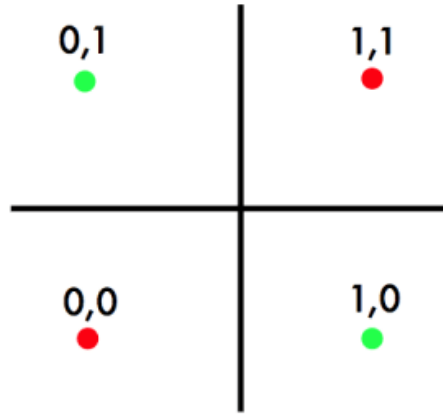


# Two-layer Perceptron



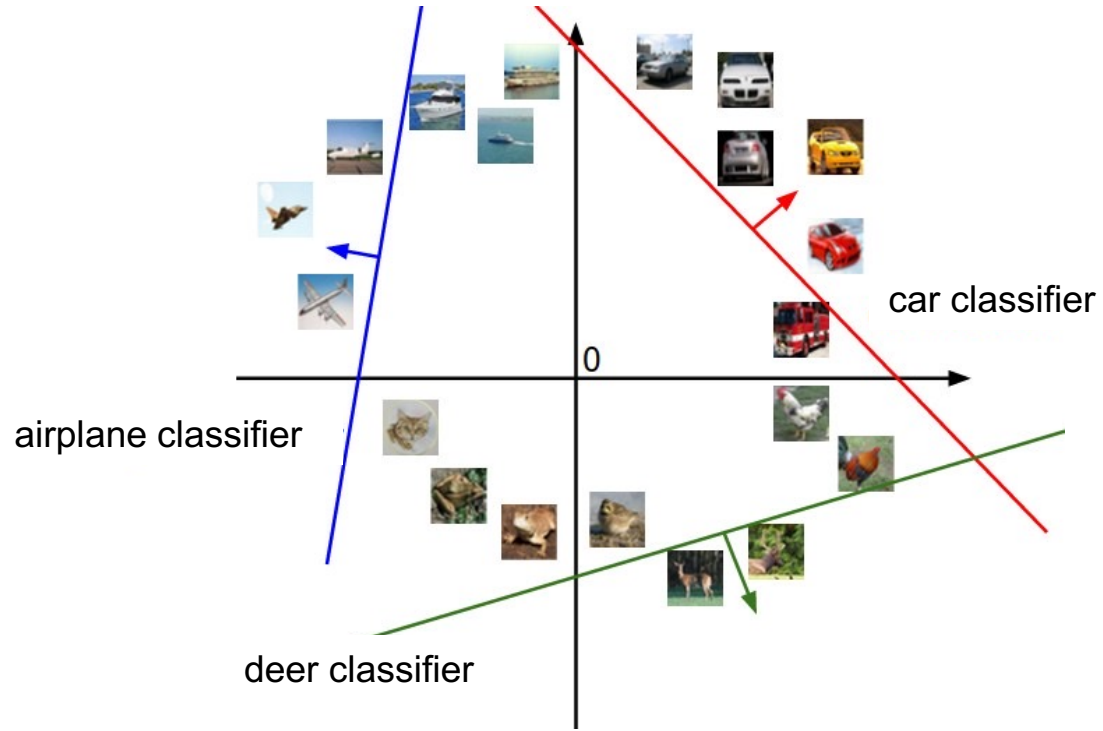
$$o_1 = g(w_{11} * a_1 + w_{12} * a_2 + w_{13} * a_3)$$

# AI Winter

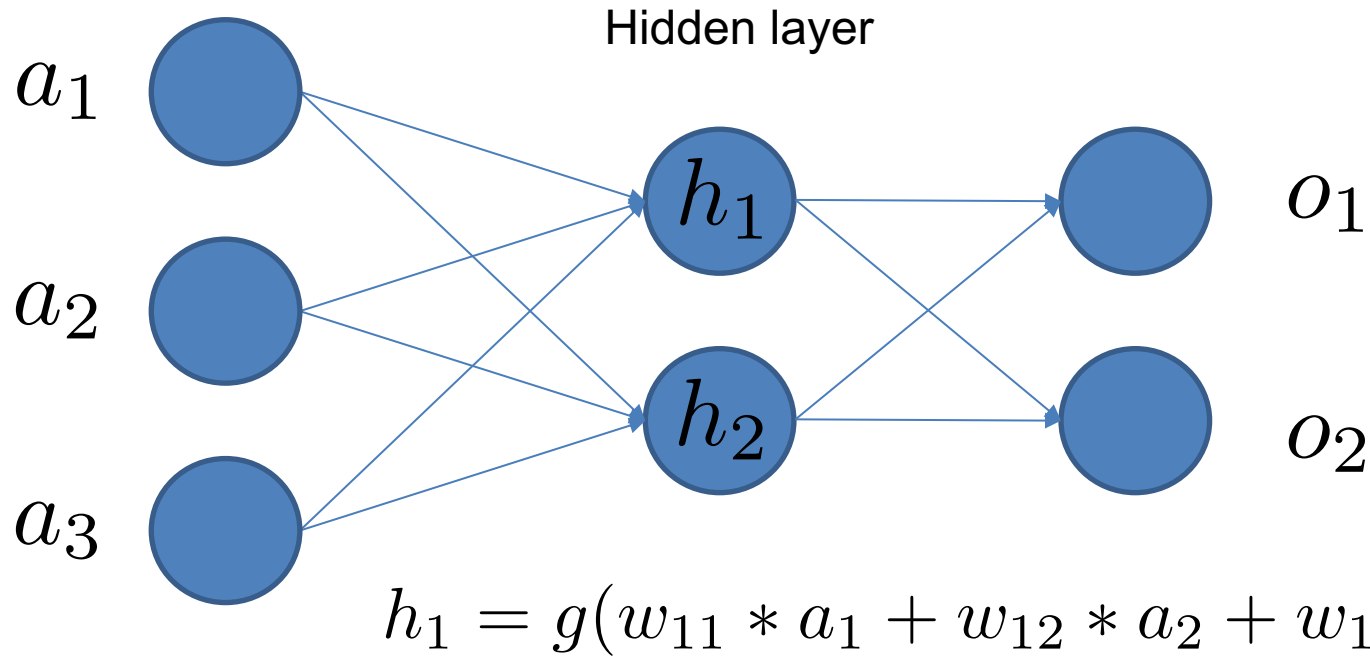


**XOR**

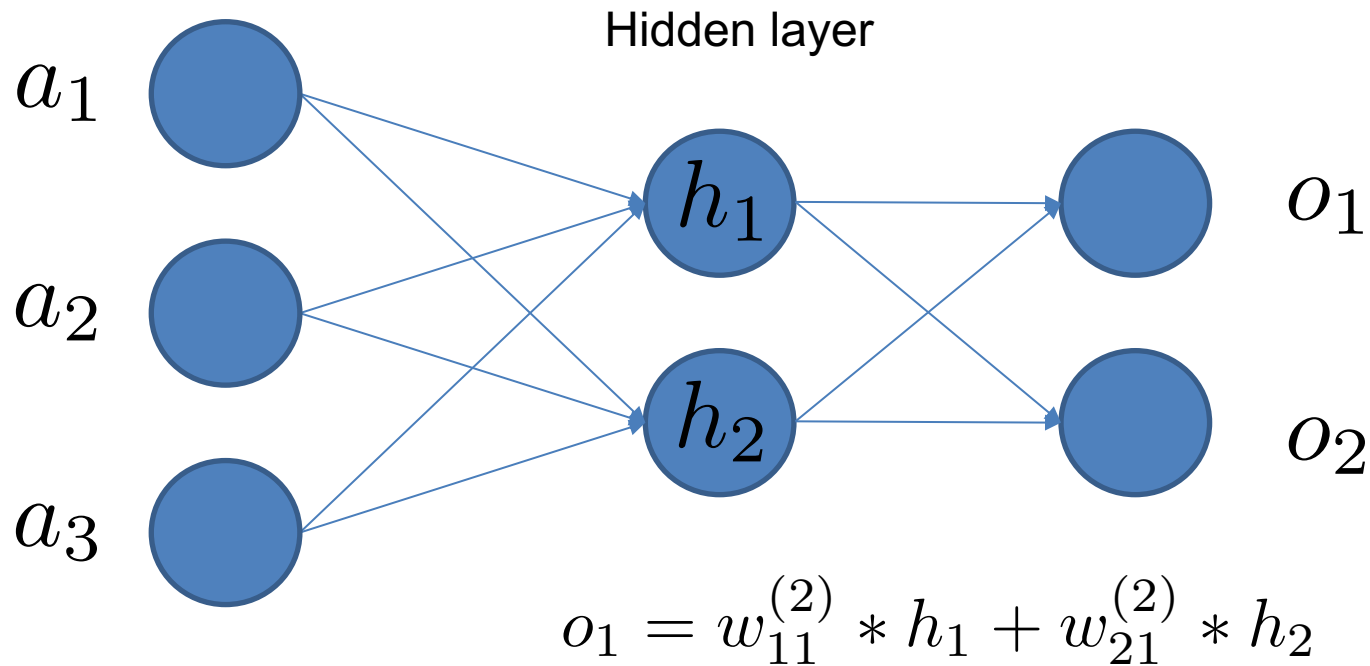
Can not solve!



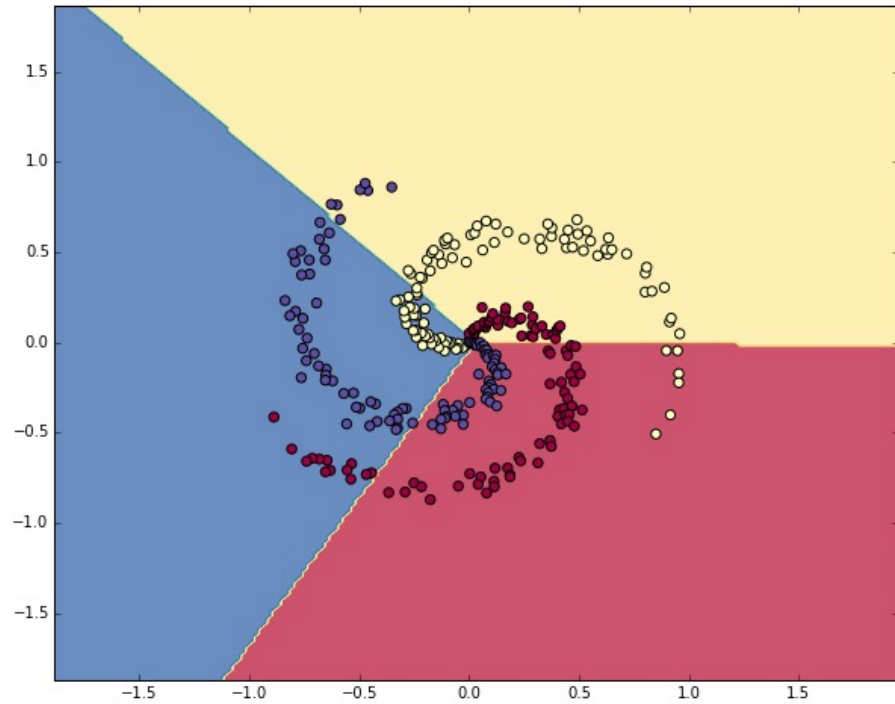
# Multi-layer Perceptron



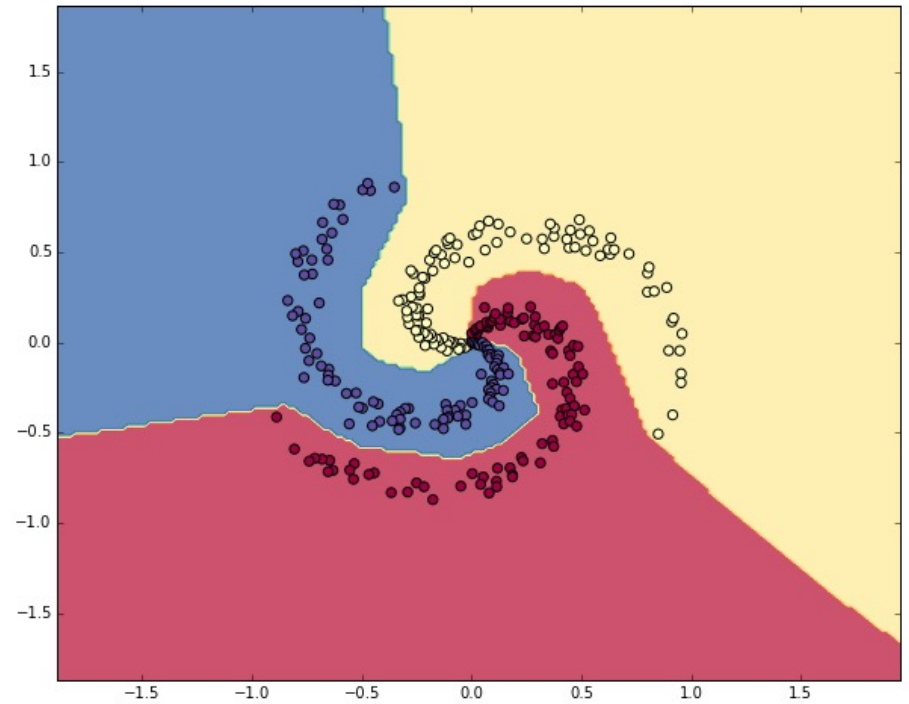
# Multi-layer Perceptron



# Comparison

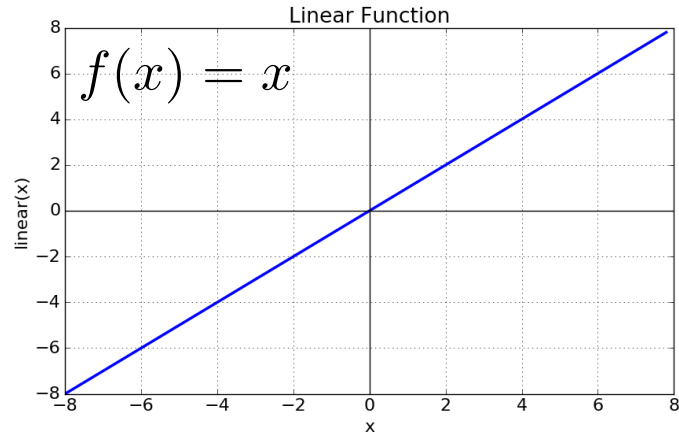


Without hidden layers

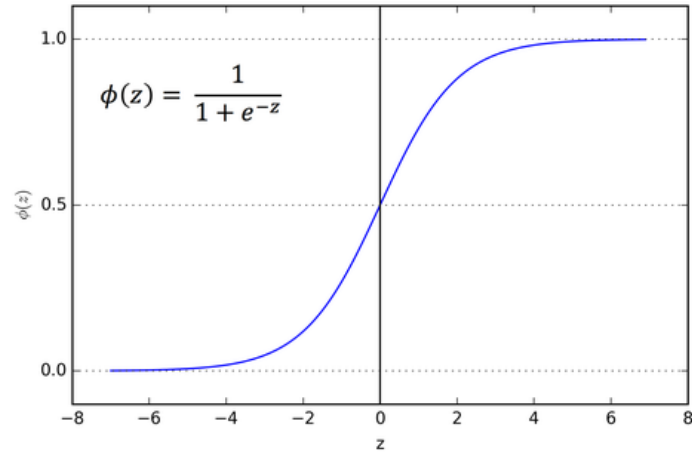


With hidden layers

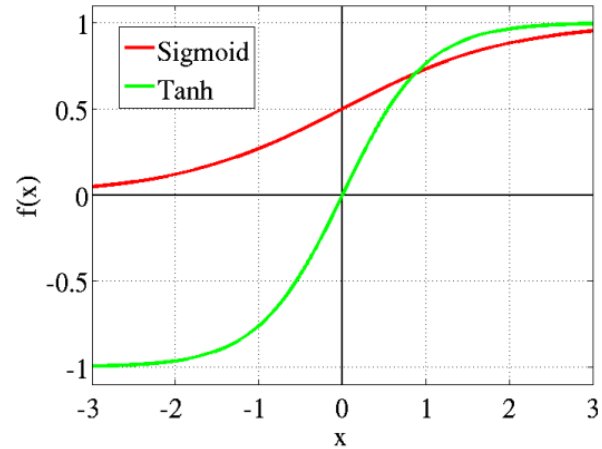
# Activation Functions



# Sigmoid



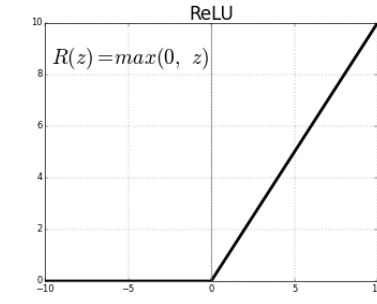
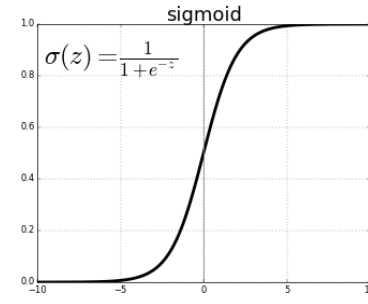
# Tanh



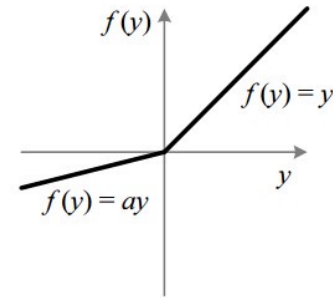
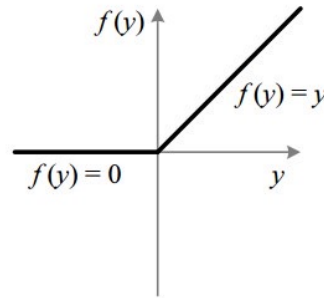


# Activation Functions

Sigmoid and ReLU



ReLU and Leaky ReLU



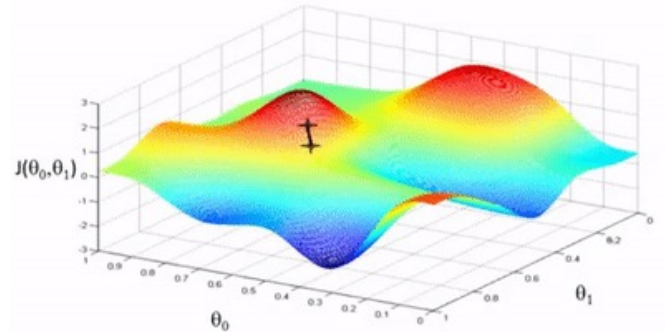
# Gradient Decent

Hypothesis:  $h_{\theta}(x) = \theta_0 + \theta_1 x$

Parameters:  $\theta_0, \theta_1$

Cost function:  $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$

Goal: minimize  $J(\theta_0, \theta_1)$   
 $\theta_0, \theta_1$



# Gradient Descent

$$\theta_i = \theta_i - \alpha * \frac{\partial L}{\partial \theta_i} \rightarrow \text{Gradient}$$

Learning rate

# Momentum & RMSProp

Momentum: accelerates the training process

$$v_i = \gamma v_i + \alpha * \frac{\partial L}{\partial \theta_i}$$

$$\theta_i = \theta_i - v_i$$

RMSProp: adjust the learning rate

$$v_i = \beta v_i + (1 - \beta) * \frac{\partial L}{\partial \theta_i}$$

$$\theta_i = \theta_i - \alpha \frac{\frac{\partial L}{\partial \theta_i}}{\sqrt{v_i} + \epsilon}$$

[Source](#)

# Adam

**Require:**  $\alpha$ : Stepsize

**Require:**  $\beta_1, \beta_2 \in [0, 1)$ : Exponential decay rates for the moment estimates

**Require:**  $f(\theta)$ : Stochastic objective function with parameters  $\theta$

**Require:**  $\theta_0$ : Initial parameter vector

$m_0 \leftarrow 0$  (Initialize 1<sup>st</sup> moment vector)

$v_0 \leftarrow 0$  (Initialize 2<sup>nd</sup> moment vector)

$t \leftarrow 0$  (Initialize timestep)

**while**  $\theta_t$  not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$  (Get gradients w.r.t. stochastic objective at timestep  $t$ )

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$  (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$  (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  (Compute bias-corrected second raw moment estimate)

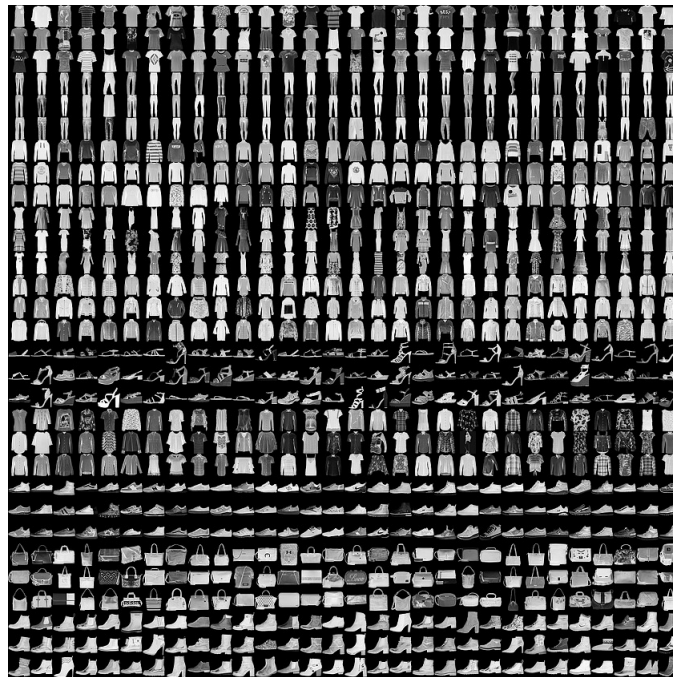
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (Update parameters)

**end while**

**return**  $\theta_t$  (Resulting parameters)

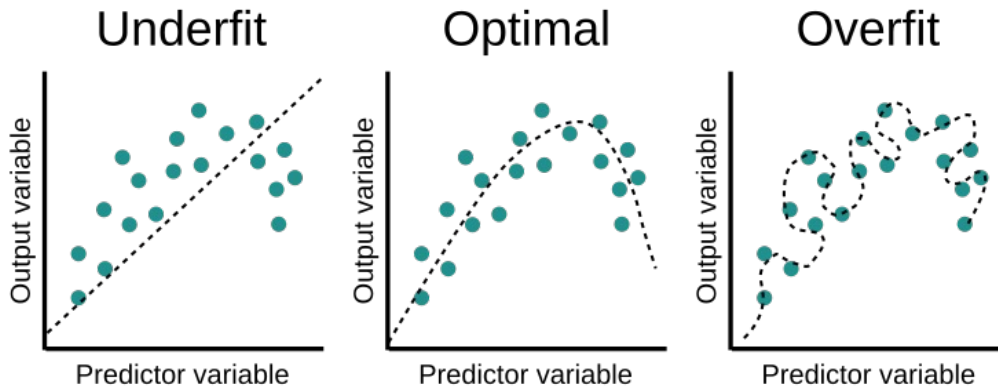
# Example: Training a Classifier

- Data
  - MNIST Fashion
  - 70,000 grayscale images
  - 10 categories
  - 28 \* 28 pixels



Code: <https://colab.research.google.com/drive/1ETGBYYFFSp3RAerKE9WZkBXFFxPa5GfYv?usp=sharing>

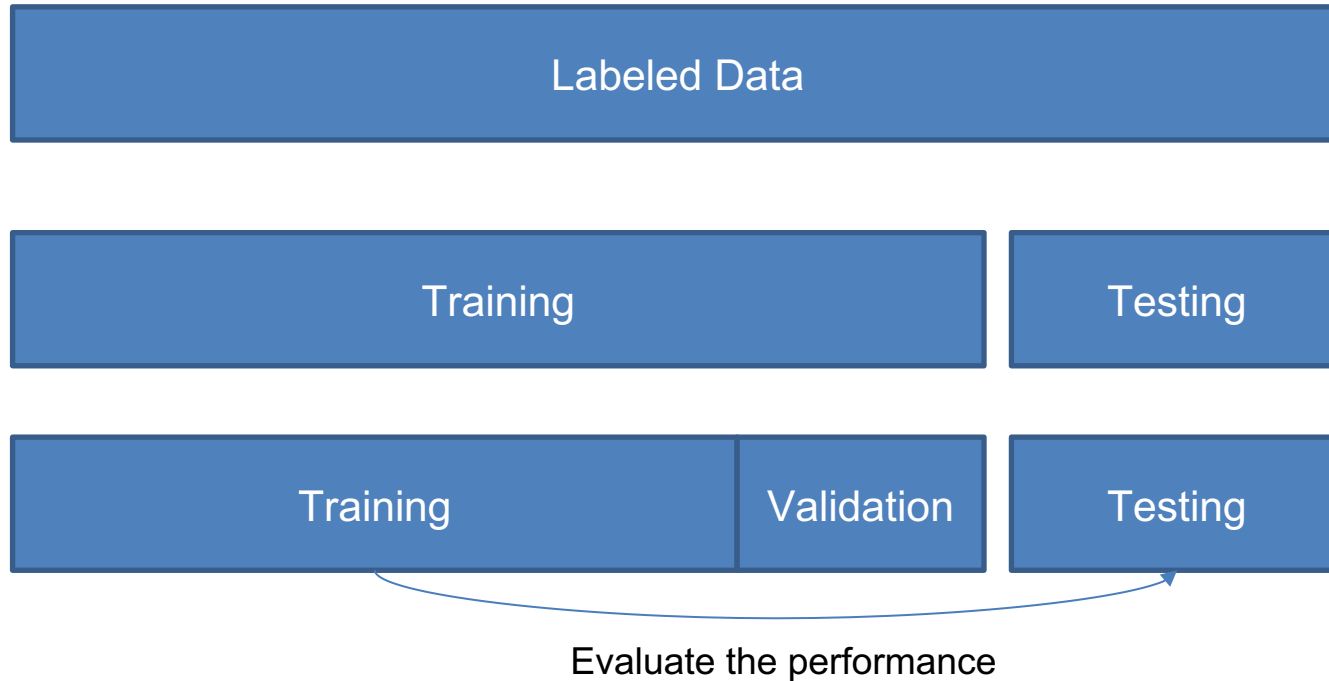
# Overfitting vs. Underfitting







# Train / Test / Val



# Example: Prepare the Data

```
fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) =
fashion_mnist.load_data()

train_images.shape
output: (60000, 28, 28)

test_images.shape
output: (10000, 28, 28)

train_labels
output: array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

# Example: Prepare the Data

```
fashion_mnist = tf
```

```
(train_images, tra  
fashion_mnist.load
```

```
train_images.shape
```

```
output: (60000, 28, 28)
```

```
test_images.shape
```

```
output: (10000, 28, 28)
```

```
train_labels
```

```
output: array([9, 0, 0, ...,
```

Label	Class
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

```
test_labels) =
```

# Example: Pre-process

---

```
train_images = train_images / 255.0
```

```
test_images = test_images / 255.0
```

# Example: Build the Model

```
model = tf.keras.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(128, activation='relu'),  
    tf.keras.layers.Dense(10)  
])
```

Input layer

# Example: Build the Model

```
model = tf.keras.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(128, activation='relu'),  
    tf.keras.layers.Dense(10)  
])
```

Hidden layer

# Example: Build the Model

```
model = tf.keras.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(128, activation='relu'),  
    tf.keras.layers.Dense(10)  
])
```

Output layer

# Example: Optimizer

```
model.compile(  
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),  
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
    metrics=[ 'accuracy' ])
```

## Learning rate:

- too large: unstable, hard to find the minimal
- too small: local minimal, training is too slow



# Example: Optimizer

```
model.compile(  
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),  
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
    metrics=[ 'accuracy' ])
```

Cross-entropy is usually used to train a classification model:

$$\ell = \sum_{c=1}^C y_c * \log(y'_c)$$

# Example: Train the Model

```
model.fit(train_images, train_labels, batch_size=128, epochs=10)
```

Batch size: the number of samples in one iteration

# Example: Train the Model

```
model.fit(train_images, train_labels, batch_size=128, epochs=10)
```

Batch size: the number of samples in one iteration

Epoch: the number of passes over the entire training dataset

# Example: Training Log

```
▶ model.fit(train_images, train_labels, batch_size = 128, epochs=10)

↳ Epoch 1/10
469/469 [=====] - 2s 3ms/step - loss: 0.2359 - accuracy: 0.9127
Epoch 2/10
469/469 [=====] - 1s 3ms/step - loss: 0.2282 - accuracy: 0.9161
Epoch 3/10
469/469 [=====] - 1s 3ms/step - loss: 0.2226 - accuracy: 0.9181
Epoch 4/10
469/469 [=====] - 1s 3ms/step - loss: 0.2158 - accuracy: 0.9213
Epoch 5/10
469/469 [=====] - 1s 3ms/step - loss: 0.2113 - accuracy: 0.9223
Epoch 6/10
469/469 [=====] - 1s 3ms/step - loss: 0.2103 - accuracy: 0.9224
Epoch 7/10
469/469 [=====] - 1s 3ms/step - loss: 0.2024 - accuracy: 0.9254
Epoch 8/10
469/469 [=====] - 2s 3ms/step - loss: 0.1983 - accuracy: 0.9265
Epoch 9/10
469/469 [=====] - 1s 3ms/step - loss: 0.1938 - accuracy: 0.9290
Epoch 10/10
469/469 [=====] - 1s 3ms/step - loss: 0.1885 - accuracy: 0.9301
<keras.callbacks.History at 0x7f40c8b42890>
```

# Example: Evaluation

```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('\nTest accuracy:', test_acc)
```

output: 313/313 - 0s - loss: 0.3369 - accuracy: 0.8880

Test accuracy: 0.8880000114440918

```
probability_model = tf.keras.Sequential(
    [model, tf.keras.layers.Softmax()])
```

```
predictions = probability_model.predict(test_images)
```

```
predictions[0]
```

output: array([9.8546373e-08, 5.0680104e-13, 6.7453760e-07, 1.8696349e-08, 5.5620589e-08, 8.5184647e-04, 1.2255837e-06, 4.7711013e-03, 9.1507090e-06, 9.9436593e-01], dtype=float32)

# Example: Get Predictions

```
np.argmax(predictions[0])
```

Output: 9



Ankle boot 100% (Ankle boot)

