

# Data Science & ML Course

## Lesson #24 Deep Learning Fundamentals I

Ivanovitch Silva  
December, 2018



# Update from repository

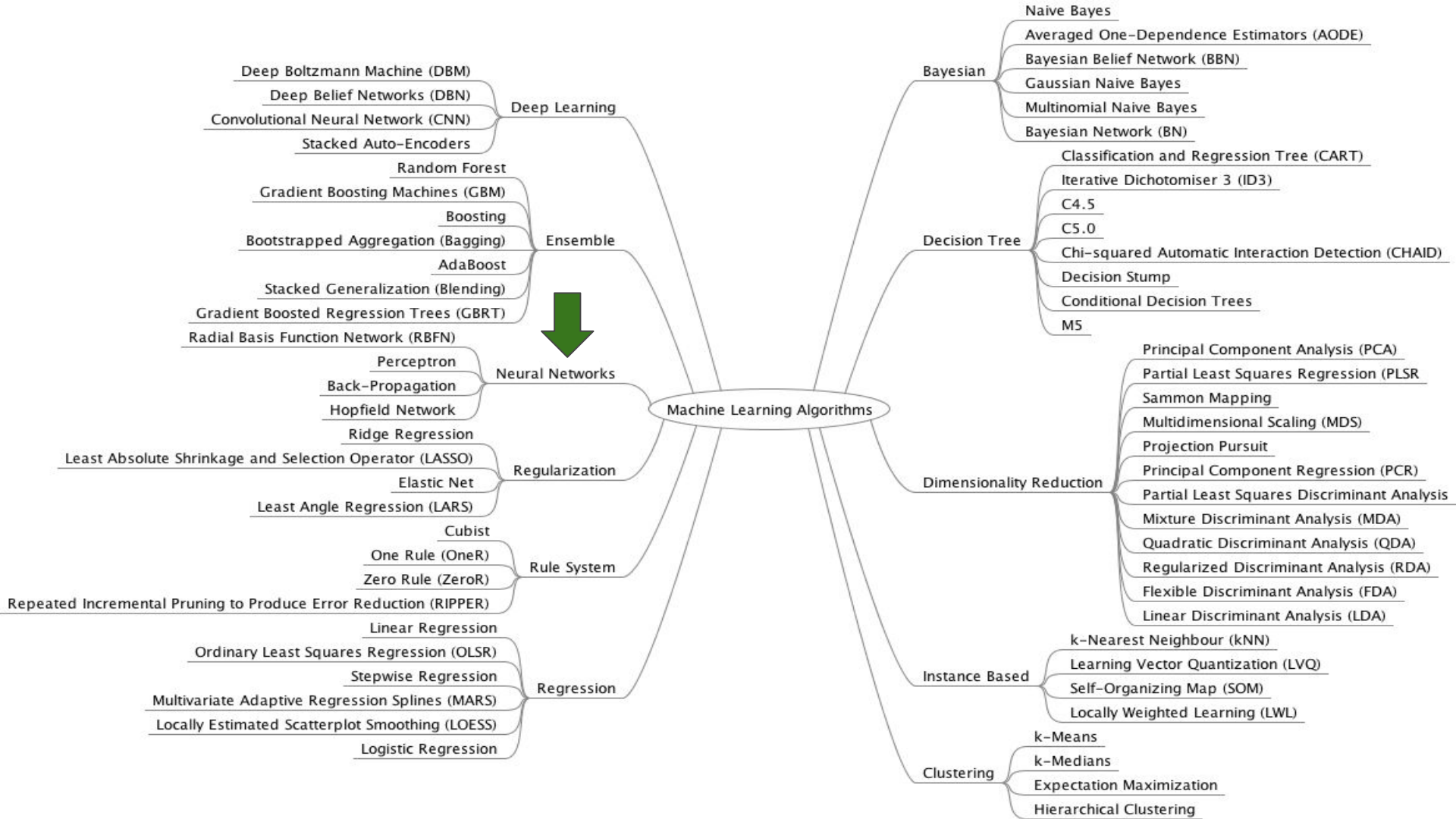
---

```
git clone https://github.com/ivanovitchm/datascience2machinelearning.git
```

Or ....

```
git pull
```





# Agenda

---

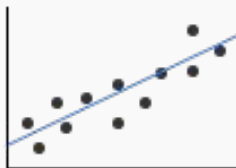
1. Representing neural network
2. Nonlinear activation functions
3. Hidden Layers
4. Case study: build a handwritten digit classifier

k-nearest neighbors



None  
(nonexistent  
training process)

linear regression



$$\hat{y} = 3x_1 + 10$$

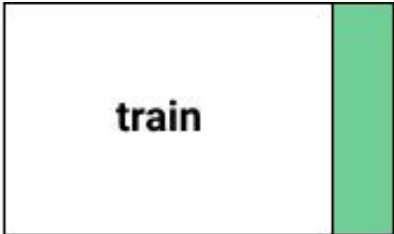
logistic regression

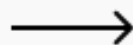


$$\hat{p} = \frac{e^{3x}}{1 + e^{3x}}$$

decision trees &  
random forests



`model.fit(`  `)`

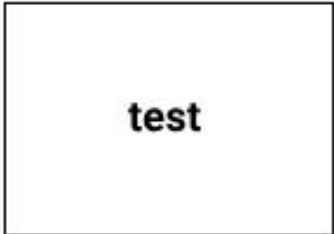


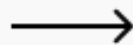
regression

$$y = 3x_1 + 4x_2$$

decision tree





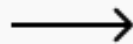
`model.predict(`  `)`



predictions



`error(`   `)`



MSE

600.25

RMSE

24.5

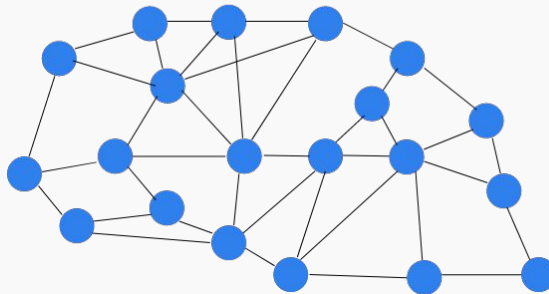
AUC

0.7

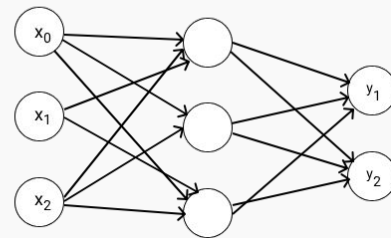
# Representing a Neural Network

---

Biological Neural Network



Artificial Neural Network

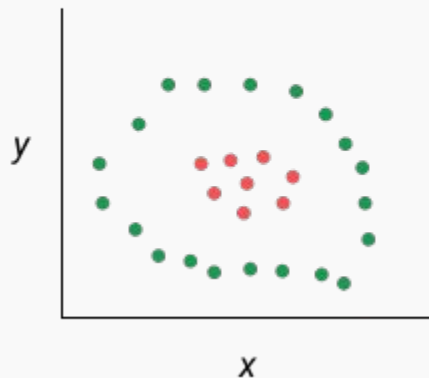
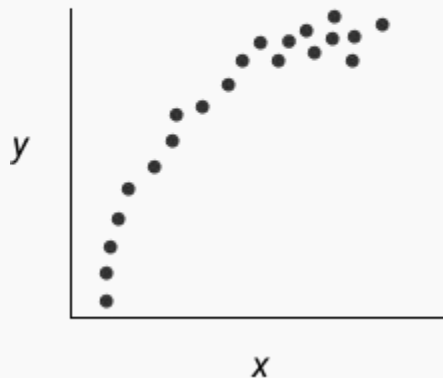


Neuron

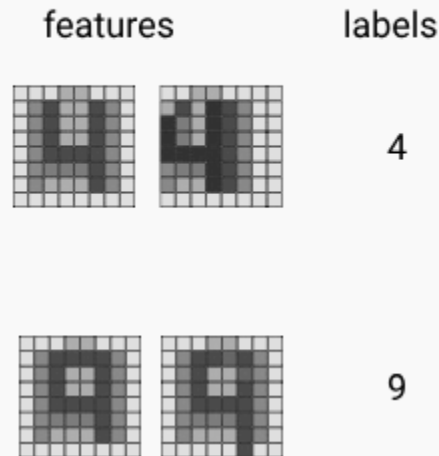
Neural network models were inspired by the structure of neurons in our brain and message passing between neurons

# Deep Neural Network

*nonlinear relationship between  $x$  and  $y$*



*no obvious relationship between pixel values and labels*



A **deep neural network** is a specific type of **neural network** that excels at capturing nonlinear relationships in data



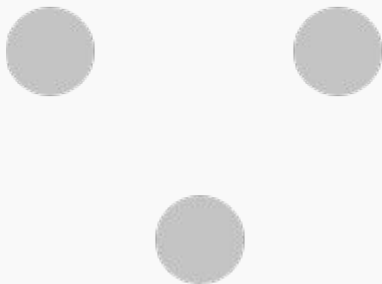
How **neural networks** are represented and how to represent **linear regression** and **logistic regression** models in that representation

# Introduction to Graphs

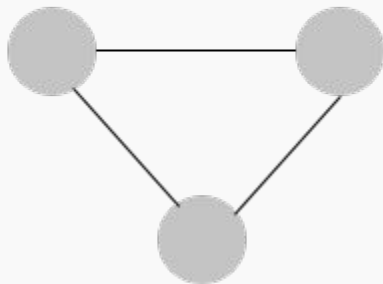
---

Neural networks are usually represented as graphs.

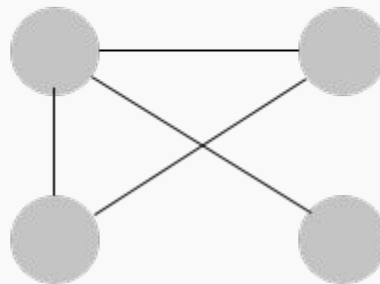
A graph with  
3 nodes ● and  
0 edges —



A graph with  
3 nodes ● and  
3 edges —



A graph with  
4 nodes ● and  
4 edges —



# Computational Graphs

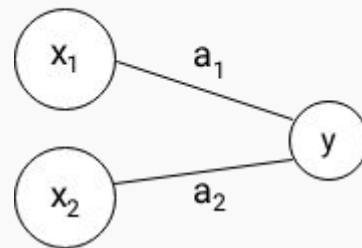
In the context of neural networks, graphs let us compactly express a pipeline of functions that we want to be executed in succession

$$\text{stage 1} \quad \sigma \left( \underset{X}{\begin{bmatrix} 100 \times 3 \end{bmatrix}} \underset{a_1^T}{\begin{bmatrix} 3 \times 6 \end{bmatrix}} \right) = \underset{L_1}{\begin{bmatrix} 100 \times 6 \end{bmatrix}}$$

↓

$$\text{stage 2} \quad \sigma \left( \underset{L_1}{\begin{bmatrix} 100 \times 6 \end{bmatrix}} \underset{a_2^T}{\begin{bmatrix} 6 \times 1 \end{bmatrix}} \right) = \underset{L_2}{\begin{bmatrix} 100 \times 1 \end{bmatrix}}$$

$$y = a_1 x_1 + a_2 x_2$$



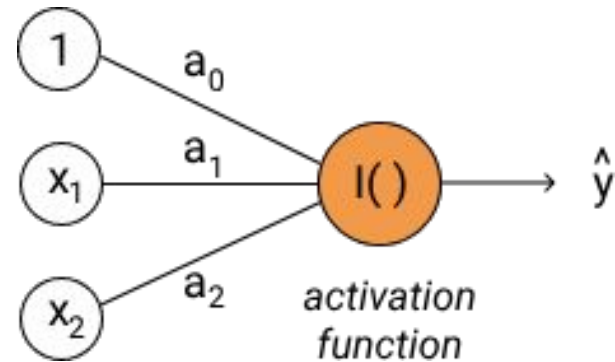
# A neural network that performs a linear regression

$$\hat{y} = a_0 + a_1x_1 + a_2x_2 + \dots + a_nx_n$$

$$Xa^T = \hat{y}$$

$$\begin{bmatrix} 1 & 2.3 & 0.2 \\ 1 & 3.1 & 0.9 \\ 1 & 1.1 & 0.5 \end{bmatrix} \begin{bmatrix} 0.9 \\ 0.7 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 2.61 \\ 3.52 \\ 1.92 \end{bmatrix}$$

$X \qquad a^T \qquad \hat{y}$



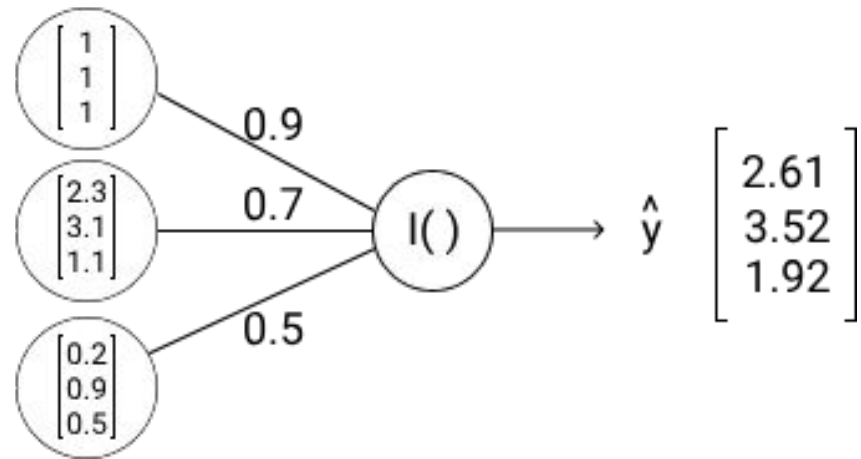
# A neural network that performs a linear regression

Linear Algebra

$$I\left(\begin{bmatrix} 1 & 2.3 & 0.2 \\ 1 & 3.1 & 0.9 \\ 1 & 1.1 & 0.5 \end{bmatrix} \begin{bmatrix} 0.9 \\ 0.7 \\ 0.5 \end{bmatrix}\right) = \begin{bmatrix} 2.61 \\ 3.52 \\ 1.92 \end{bmatrix}$$

$X$                        $a^T$                        $\hat{y}$

Neural Network



# Generate yourself dataset

---

Scikit-learn contains the following convenience functions for generating data:

- [sklearn.datasets.make\\_regression\(\)](#)
- [sklearn.datasets.make\\_classification\(\)](#)
- [sklearn.datasets.make\\_moons\(\)](#)

# Generating regression data

```
from sklearn.datasets import make_regression
import pandas as pd

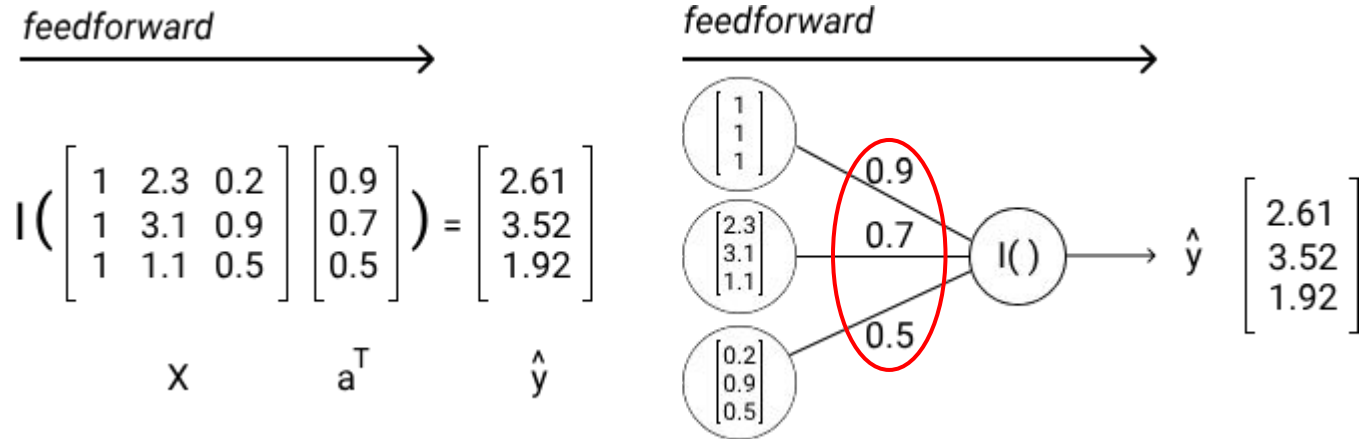
# make_regression return a tuple
# data[0].shape (1000,3) -> features
# data[1].shape (1000,) -> target
data = make_regression(n_samples=100,
                      n_features=3, random_state=1)

features = pd.DataFrame(data[0])
labels = pd.Series(data[1])
```

```
0    -10.378660
1     25.512450
2     19.677056
3    149.502054
4    -121.652109
dtype: float64
```

	0	1	2
0	1.293226	-0.617362	-0.110447
1	-2.793085	0.366332	1.937529
2	0.801861	-0.186570	0.046567
3	0.129102	0.502741	1.616950
4	-0.691661	-0.687173	-0.396754

# Fitting a linear regression neural network



```
from sklearn.linear_model import SGDRegressor
lr = linear_model.SGDRegressor()
lr.fit(X,y)
```



# Fitting a linear regression neural network

```
# generate the dataset
data = make_regression(n_samples=100,
                      n_features=3,
                      random_state=1)

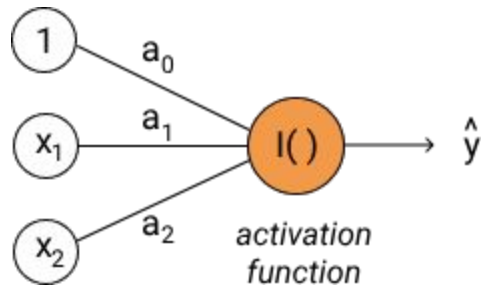
features = pd.DataFrame(data[0])
labels = pd.Series(data[1])
```

```
# configure the bias
features["bias"] = 1
```

```
train_weights = train(features, labels)
linear_predictions = feedforward(features,
                                  train_weights)
```

```
def train(features, labels):
    lr = SGDRegressor()
    lr.fit(features, labels)
    # Returns a nested NumPy array of weights.
    weights = lr.coef_
    return weights

def feedforward(features, weights):
    predictions = np.dot(features, weights.T)
    return predictions
```



# Generating a classification data

---

```
from sklearn.datasets import make_classification
class_data = make_classification(n_samples=1000,
                                n_features=4,
                                random_state=1)
```

*# Features*

```
class_features = class_data[0]
class_features[:5]
```

*# Labels*

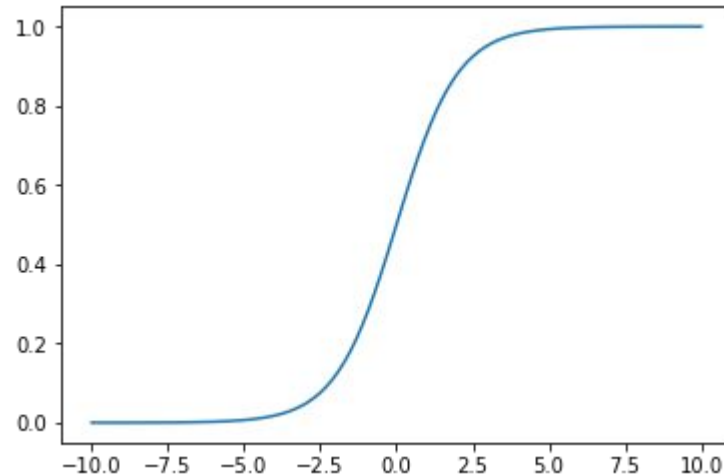
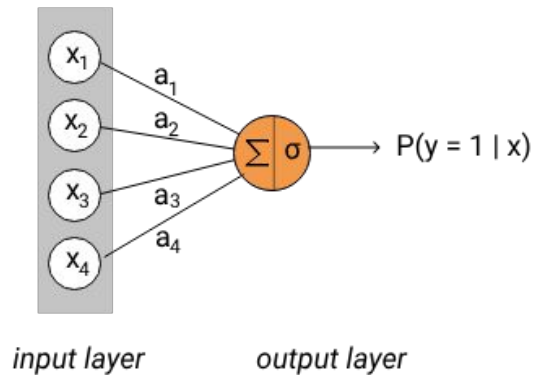
```
class_labels = class_data[1]
class_labels[:5]
```

```
array([[ 1.91518414,  1.14995454, -1.52847073,  0.79430654],
       [ 1.4685668 ,  0.80644722, -1.04912964,  0.74652026],
       [ 1.47102089,  0.90060386, -1.20228498,  0.57845433],
       [ 1.07642824, -0.1813636 ,  0.49116807,  1.95642108],
       [-5.34139911, -2.29763222,  2.77907005, -3.87463248]])
```

```
array([1, 1, 1, 1, 0])
```

# Implementing a neural network that performs classification

## Binary Classification



$$\begin{bmatrix} 100 \times 4 \\ X \end{bmatrix} \begin{bmatrix} 4 \times 1 \\ a^T \end{bmatrix} = \begin{bmatrix} 100 \times 1 \\ P(y = 1 | x) \end{bmatrix}$$

$$\hat{y} = \sigma(Xa^T)$$

$$P(y = 1 | x) > 0.5$$


$$P(y = 0 | x) < 0.5$$

# Implementing a Logistic Regression Model

```
# generate classification dataset
```

```
class_data = make_classification(n_samples=100,  
                                n_features=4,  
                                random_state=1)
```

```
class_features = class_data[0]  
class_labels = class_data[1]
```


$$\hat{y} = \sigma(Xa^T)$$


```
log_train_weights = log_train(class_features,  
                               class_labels)  
log_predictions = log_feedforward(class_features,  
                                   log_train_weights)
```

```
def log_train(class_features, class_labels):  
    sg = SGDClassifier()  
    sg.fit(class_features, class_labels)  
    return sg.coef_
```

```
def sigmoid(linear_combination):  
    return 1/(1+np.exp(-linear_combination))
```

```
def log_feedforward(class_features, log_train_weights):  
    linear_combination = np.dot(class_features,  
                                 log_train_weights.T)  
    log_predictions = sigmoid(linear_combination)  
    log_predictions[log_predictions >= 0.5] = 1  
    log_predictions[log_predictions < 0.5] = 0  
    return log_predictions
```



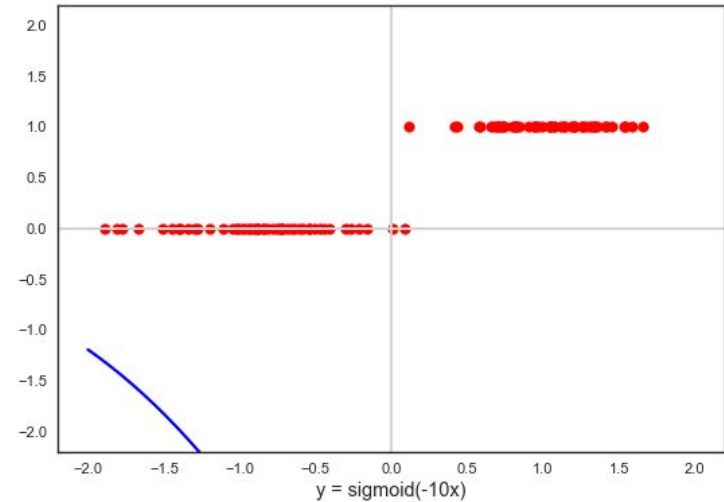
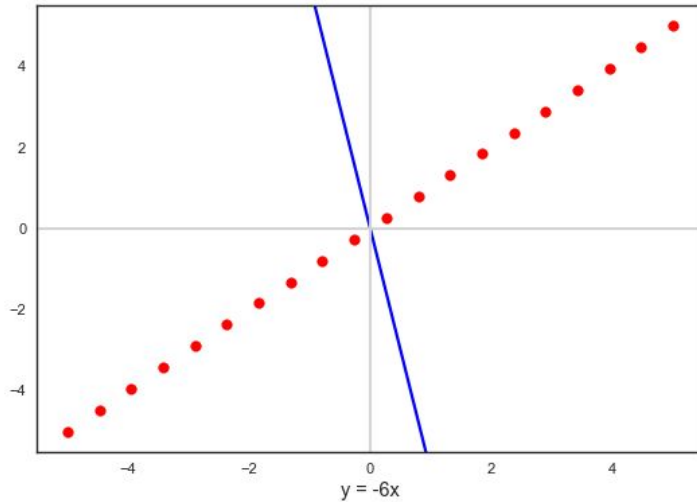
$$P(y = 1|x) > 0.5$$

$$P(y = 0|x) < 0.5$$

# Activation functions

# Nonlinear Activation Functions

---



# Neural Networks - Activation Functions

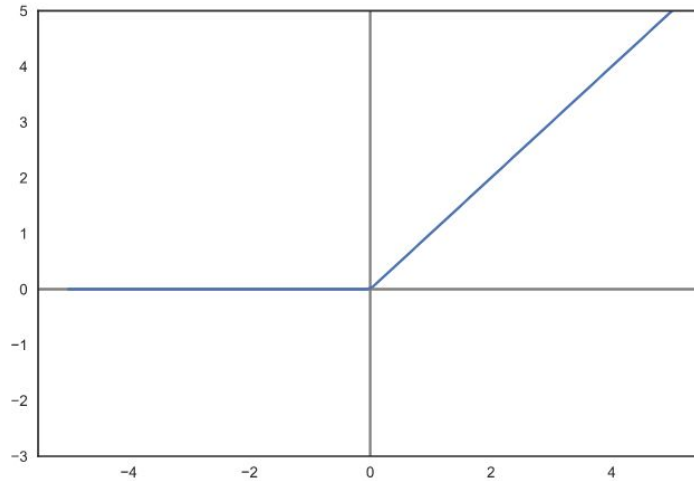
---

The three most commonly used activation functions in neural networks are:

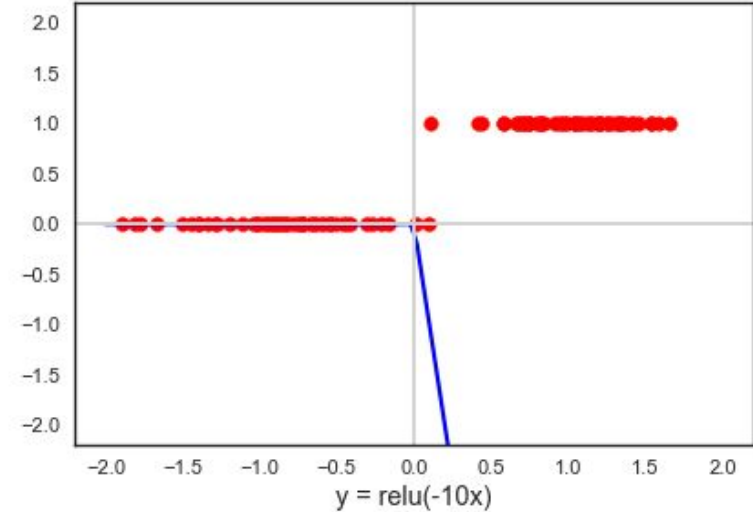
- the sigmoid function
- the ReLU function
- the tanh function

# ReLU Function - Rectifier Linear Unit

```
relu = lambda x: np.maximum(0,x)  
x=np.linspace(-10,10,100)  
plt.plot(x,relu(x))
```



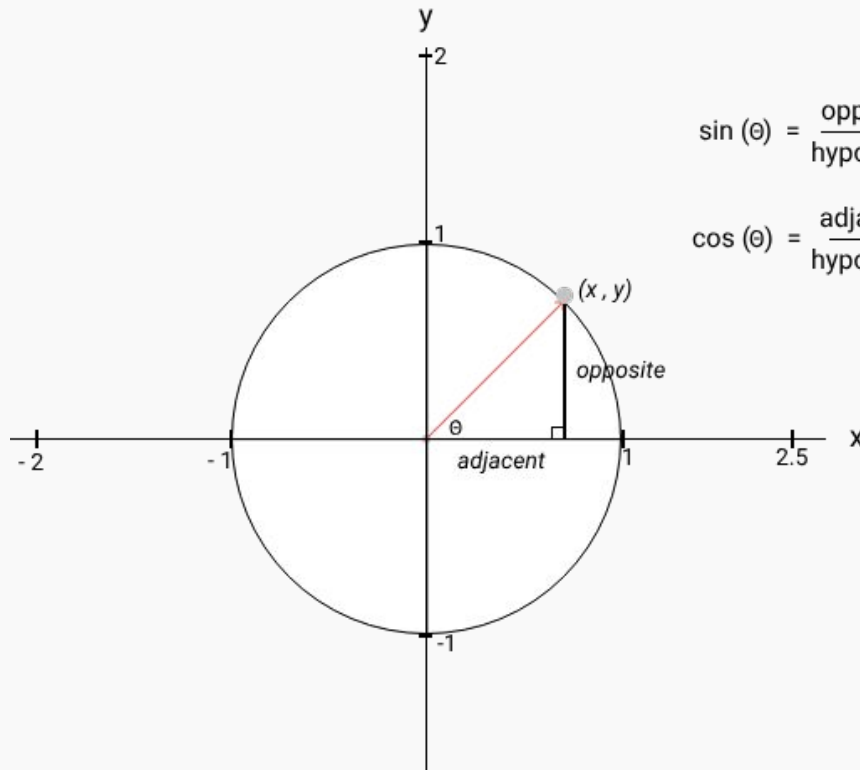
$$\text{ReLU} = \max(0, x)$$



ReLU is a commonly used activation function in neural networks for solving regression problems.



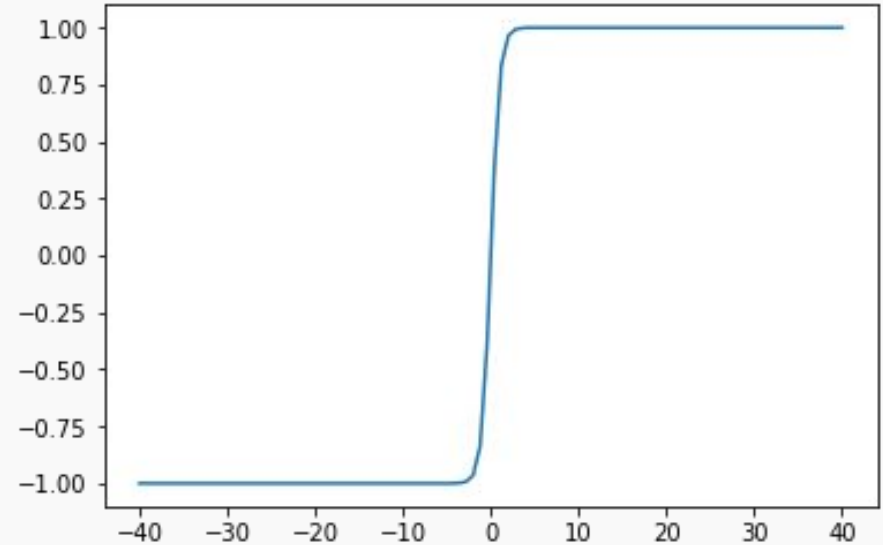
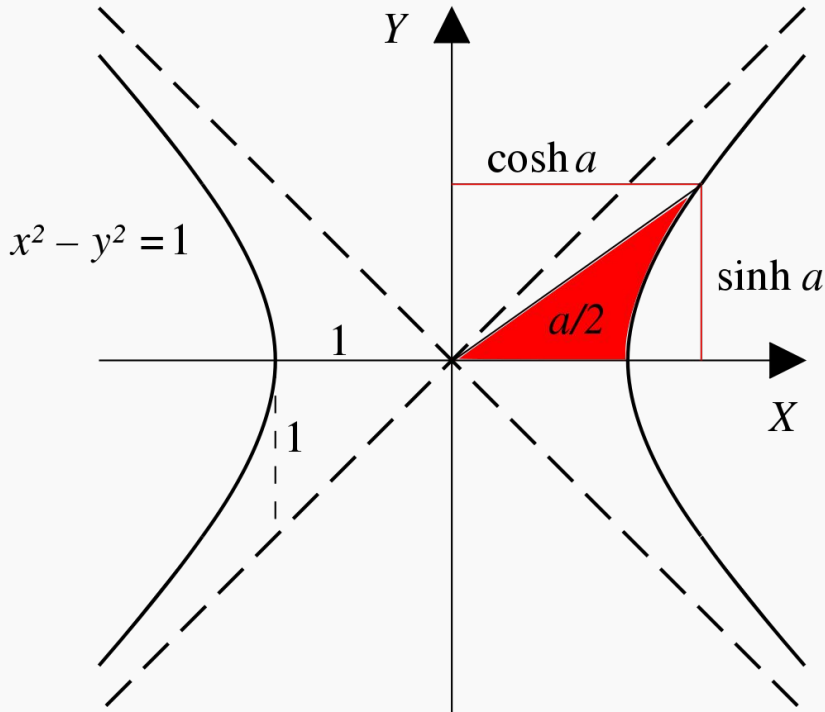
# Trigonometric Functions



$$\sin(\theta) = \frac{\text{opposite}}{\text{hypotenuse}} = \text{opposite} = y$$

$$\cos(\theta) = \frac{\text{adjacent}}{\text{hypotenuse}} = \text{adjacent} = x$$

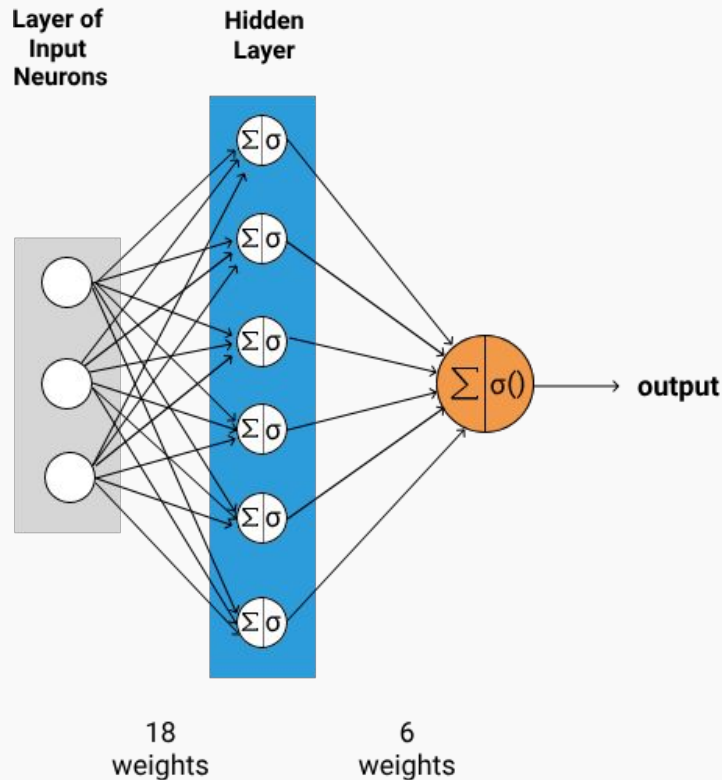
# Hyperbolic Tangent Function (tanh)



It is commonly used in neural networks for classification tasks.

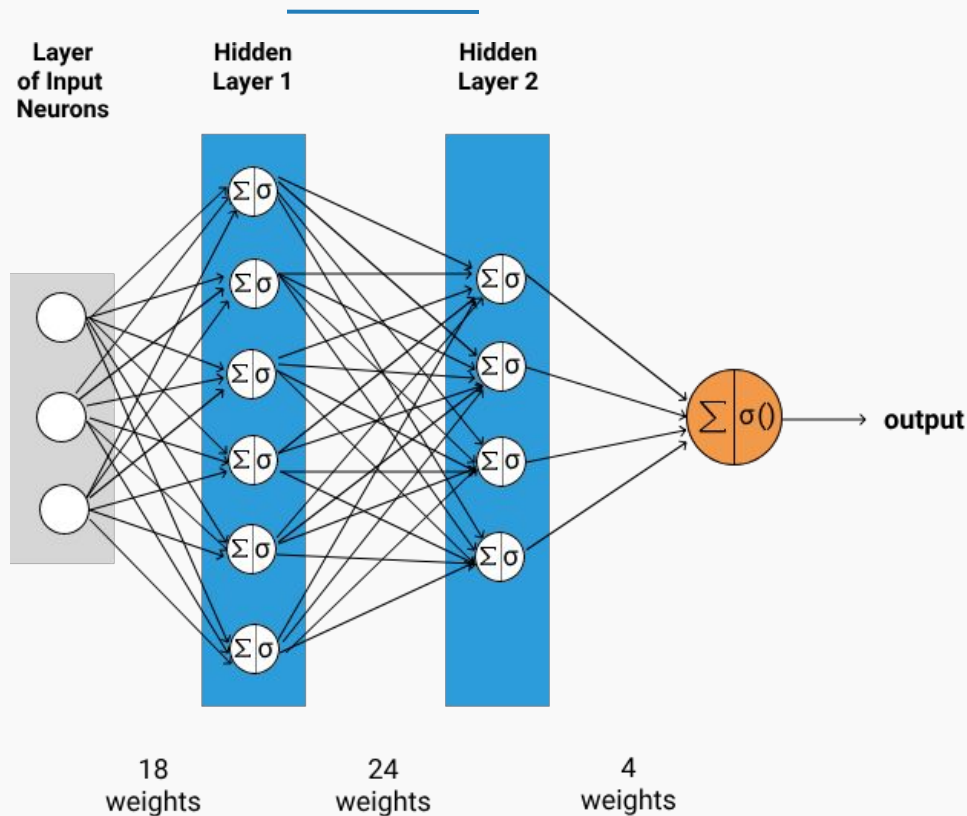
# Hidden Layers

# Multi-layers networks (deep neural networks)



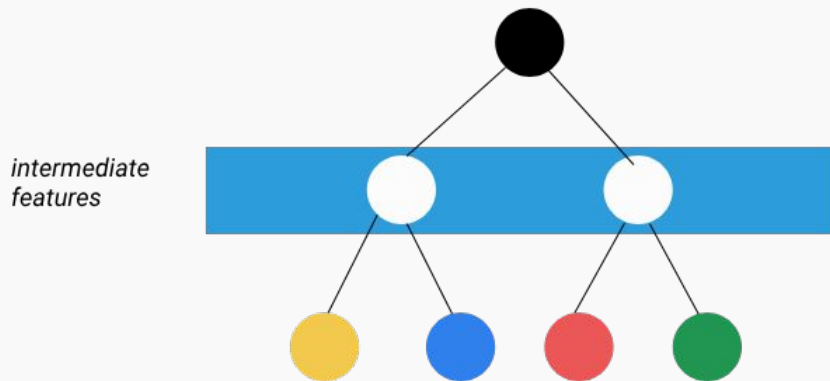
- This kind of models are able to better capture nonlinearity in the data
- Choosing the number of neurons in this layer is a bit of an art
- We can think of each hidden layer as intermediate features that are learned during the training process.

# Multi-layers networks (deep neural networks)

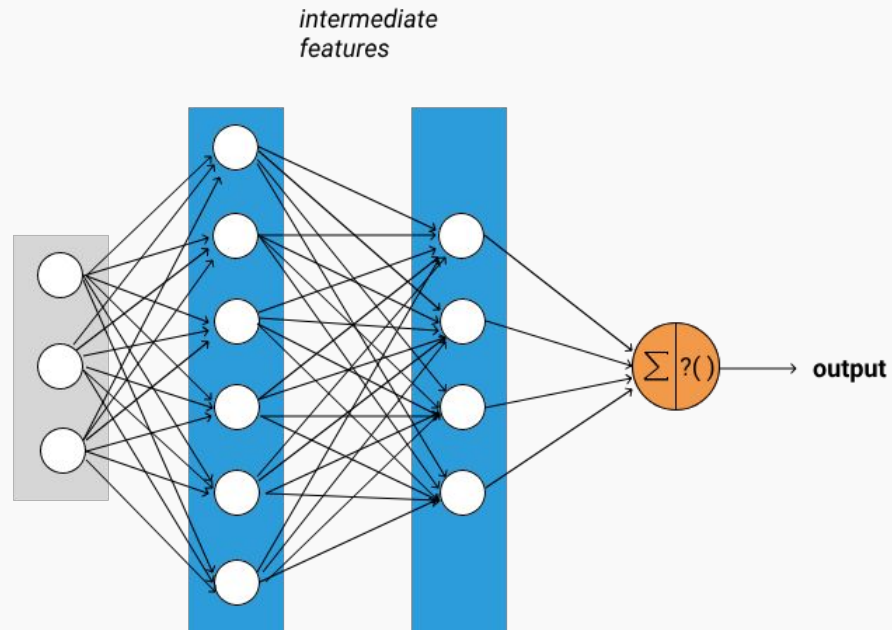


# Comparison with Decision Tree Models

**Decision Tree**



**Deep Neural Network**



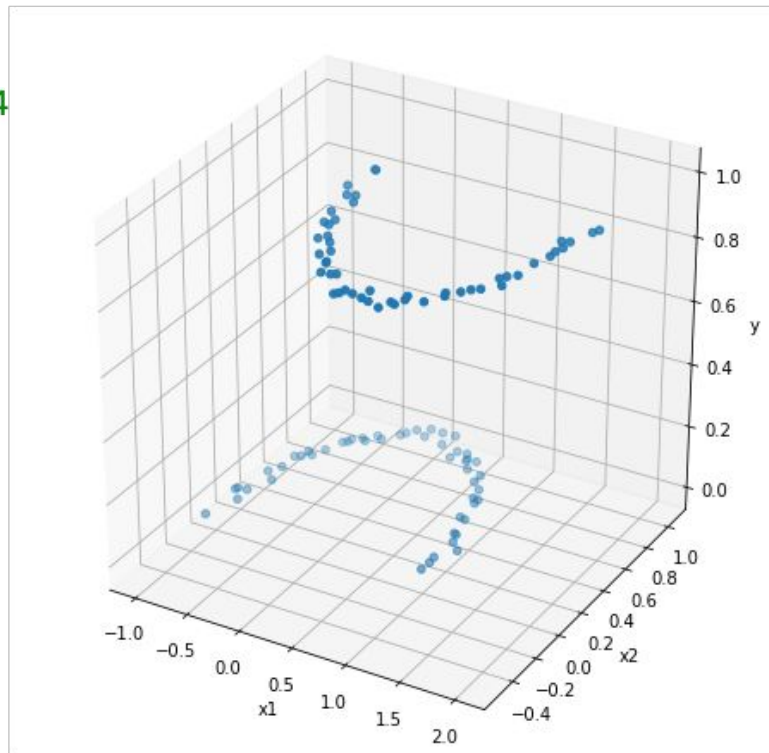
# Generating data that contains nonlinearity

---

```
data = make_moons(100, random_state=3, noise=0.04)
features = pd.DataFrame(data[0])
labels = pd.Series(data[1])
```

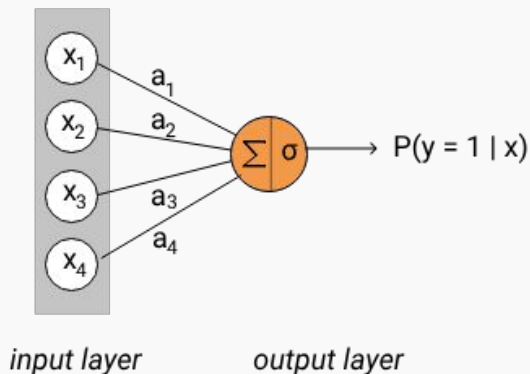
```
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')
```

```
ax.scatter(features[0], features[1], labels)
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
```

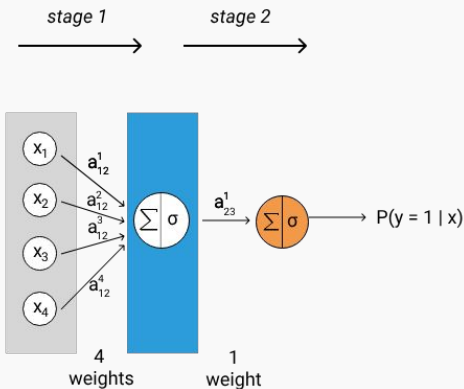


# Hidden Layer with a single neuron

## Binary Classification



$$\begin{bmatrix} 100 \times 4 \\ X \end{bmatrix} \begin{bmatrix} 4 \times 1 \\ a^T \end{bmatrix} = \begin{bmatrix} 100 \times 1 \\ P(y = 1 | x) \end{bmatrix}$$



$$\text{stage 1} \quad \sigma \left( \begin{bmatrix} 100 \times 4 \\ X \end{bmatrix} \begin{bmatrix} 4 \times 1 \\ a_1^T \end{bmatrix} \right) = \begin{bmatrix} 100 \times 1 \\ L_1 \end{bmatrix}$$

$$\text{stage 2} \quad \sigma \left( \begin{bmatrix} 100 \times 1 \\ L_1 \end{bmatrix} \begin{bmatrix} 1 \times 1 \\ a_2^T \end{bmatrix} \right) = \begin{bmatrix} 100 \times 1 \\ L_2 \end{bmatrix}$$



# Training a neural network using scikit-learn

---

Scikit-learn contains two classes for working with neural networks:

- [MLPClassifier](#)
- [MLPRegressor](#)

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier()
mlp.fit(X_train, y_train)
predictions = mlp.predict(X_test)
```

```
mlp = MLPClassifier(hidden_layer_sizes=(6,), activation='logistic')
```

```
data = make_moons(1000, random_state=3, noise=0.04)
features = pd.DataFrame(data[0])
labels = pd.Series(data[1])
features["bias"] = 1

train_x, test_x, train_y, test_y = train_test_split(features,
                                                    labels,
                                                    test_size=0.30,
                                                    random_state=42)
```

```
mlp = MLPClassifier(hidden_layer_sizes=(4,),
                    activation='logistic', max_iter=10000)
mlp.fit(train_x, train_y)
nn_predictions = mlp.predict(test_x)
```

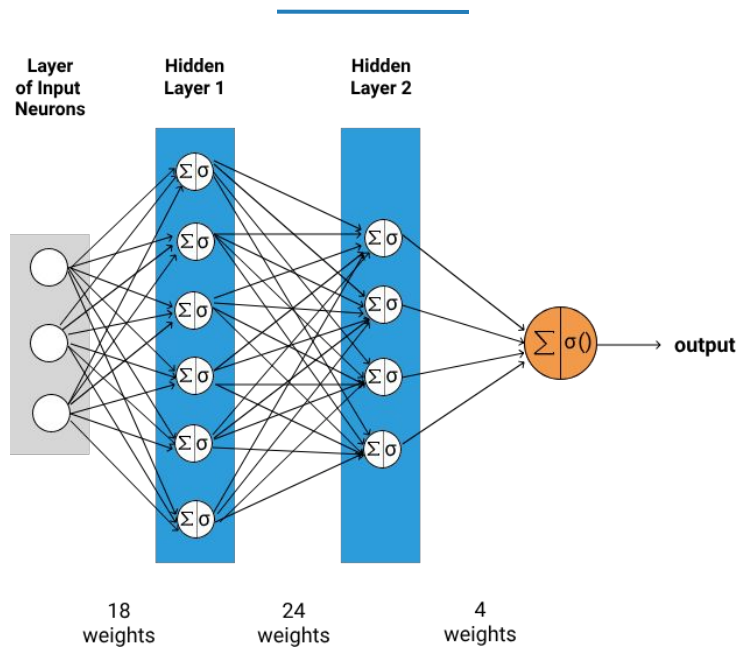
```
lr = LogisticRegression(solver='lbfgs')
lr.fit(train_x, train_y)
log_predictions = lr.predict(test_x)
```

0.88

0.88

```
nn_accuracy = accuracy_score(test_y, nn_predictions)
log_accuracy = accuracy_score(test_y, log_predictions)
```

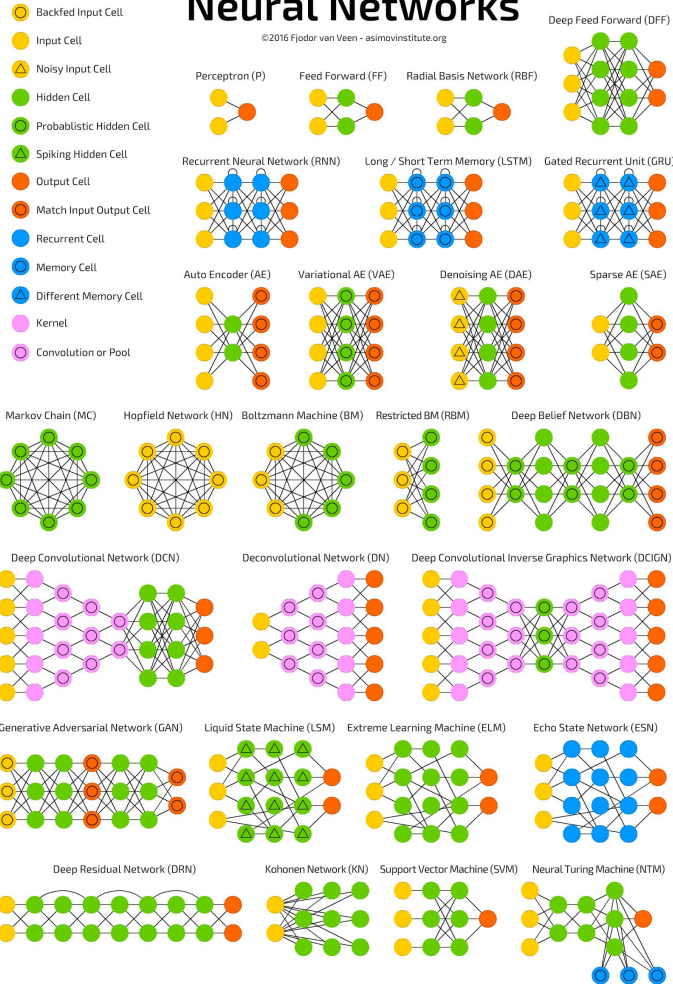
# Multiple Hidden Layers



```
mlp = MLPClassifier(hidden_layer_sizes=(n,k))
```

# Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org



# Lesson #24 - Deep Learning Fundamental I.ipynb

