Data Science & ML Course
Lesson #24 Deep Learning Fundamentals I

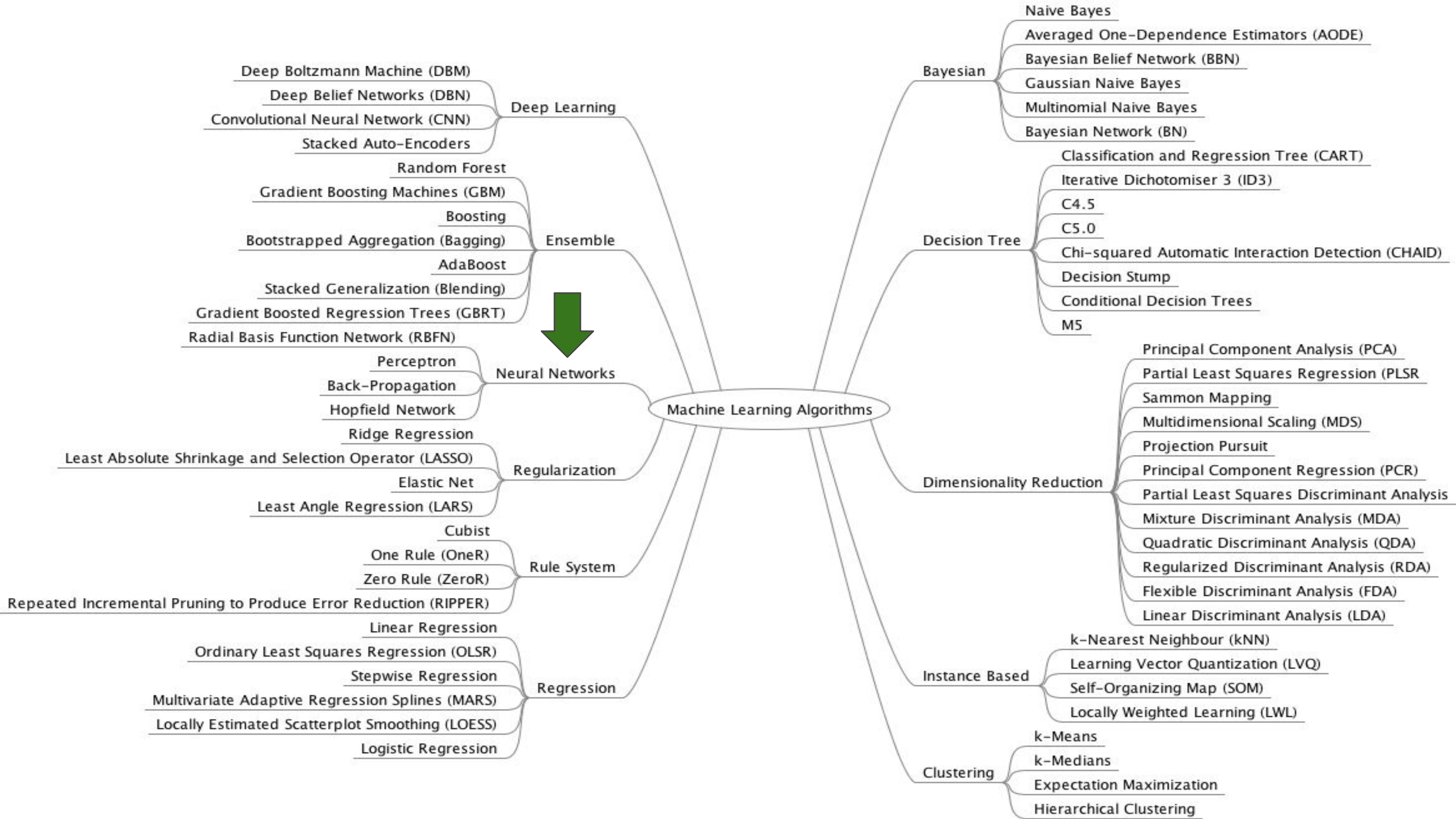
Ivanovitch Silva
December, 2018

Update from repository

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

Or ....

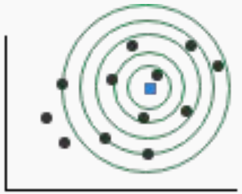
```
git pull
```


Machine Learning Algorithms
Deep Learning
Deep Boltzmann Machine (DBM)
Deep Belief Networks (DBN)
Convolutional Neural Network (CNN)
Stacked Auto-Encoders
Ensemble
Random Forest
Gradient Boosting Machines (GBM)
Boosting
Bootstrapped Aggregation (Bagging)
AdaBoost
Stacked Generalization (Blending)
Gradient Boosted Regression Trees (GBRT)
Neural Networks
Radial Basis Function Network (RBFN)
Perceptron
Back-Propagation
Hopfield Network
Regularization
Ridge Regression
Least Absolute Shrinkage and Selection Operator (LASSO)
Elastic Net
Least Angle Regression (LARS)
Rule System
Cubist
One Rule (OneR)
Zero Rule (ZeroR)
Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
Regression
Linear Regression
Ordinary Least Squares Regression (OLSR)
Stepwise Regression
Multivariate Adaptive Regression Splines (MARS)
Locally Estimated Scatterplot Smoothing (LOESS)
Logistic Regression
Bayesian
Naive Bayes
Averaged One-Dependence Estimators (AODE)
Bayesian Belief Network (BBN)
Gaussian Naive Bayes
Multinomial Naive Bayes
Bayesian Network (BN)
Decision Tree
Classification and Regression Tree (CART)
Iterative Dichotomiser 3 (ID3)
C4.5
C5.0
Chi-squared Automatic Interaction Detection (CHAID)
Decision Stump
Conditional Decision Trees
M5
Dimensionality Reduction
Principal Component Analysis (PCA)
Partial Least Squares Regression (PLSR
Sammon Mapping
Multidimensional Scaling (MDS)
Projection Pursuit
Principal Component Regression (PCR)
Partial Least Squares Discriminant Analysis
Mixture Discriminant Analysis (MDA)
Quadratic Discriminant Analysis (QDA)
Regularized Discriminant Analysis (RDA)
Flexible Discriminant Analysis (FDA)
Linear Discriminant Analysis (LDA)
Instance Based
k-Nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organizing Map (SOM)
Locally Weighted Learning (LWL)
Clustering
k-Means
k-Medians
Expectation Maximization
Hierarchical Clustering

Agenda

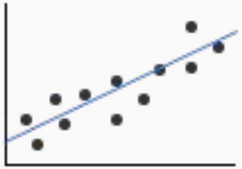
1. Representing neural network
2. Nonlinear activation functions
3. Hidden Layers
4. Case study: build a handwritten digit classifier

k-nearest neighbors



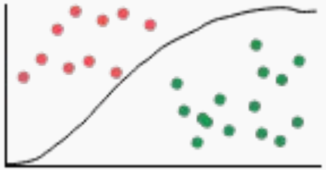
None
(nonexistent
training process)

linear regression



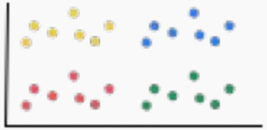
$$\hat{y} = 3x_1 + 10$$

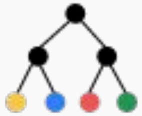
logistic regression

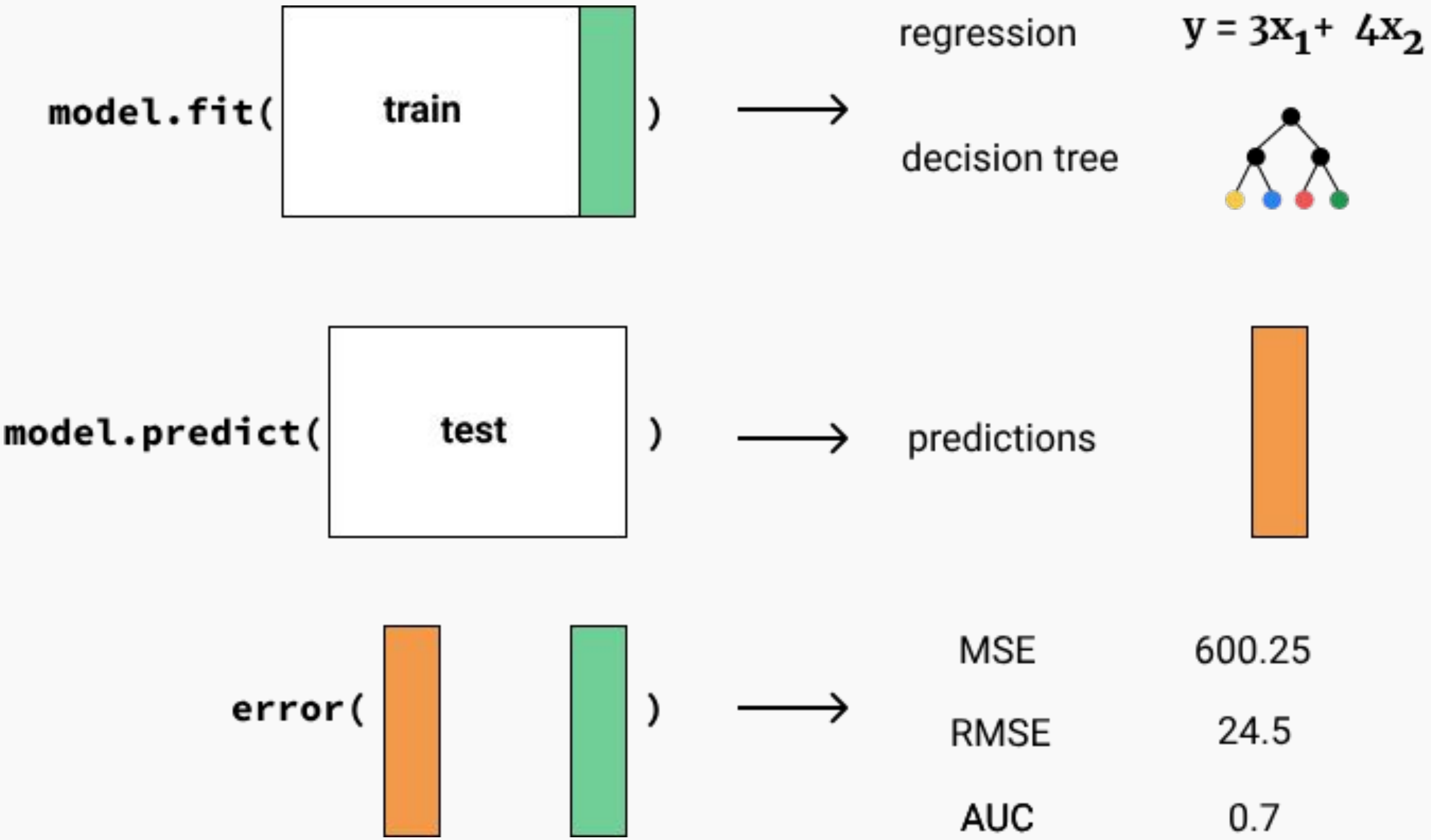


$$\hat{p} = \frac{e^{3x}}{1 + e^{3x}}$$

decision trees &
random forests

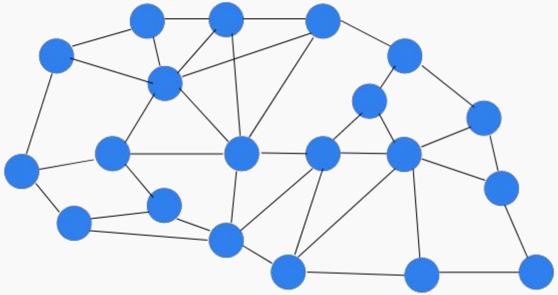




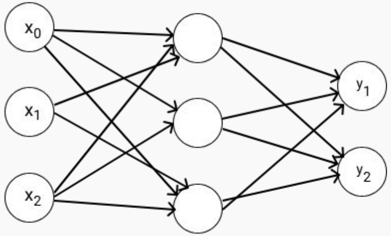

model.fit(
train
)
regression
y = 3x1+ 4x2
decision tree
model.predict(
test
)
predictions
error(
)
MSE
600.25
RMSE
24.5
AUC
0.7

Representing a Neural Network

Bioligical Neural Network



Artificial Neural Network

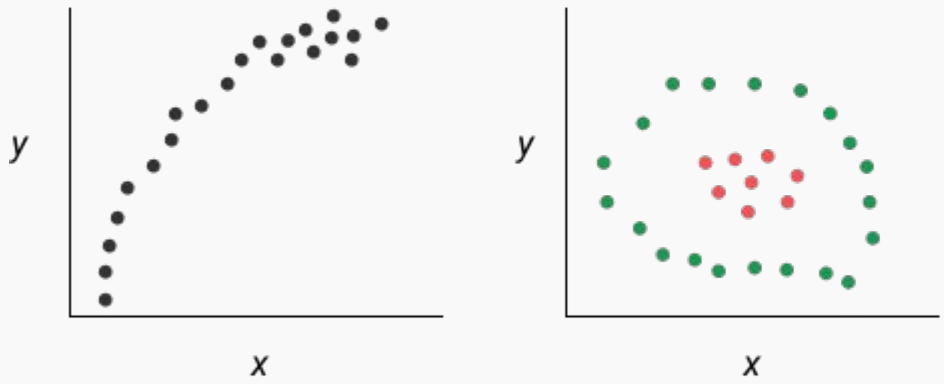


Neuron

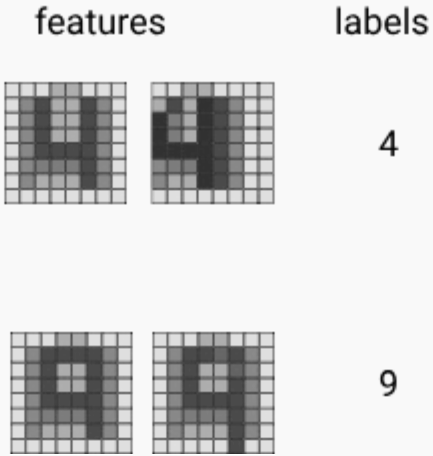
Neural network models were inspired by the structure of neurons in our brain and message passing between neurons

Deep Neural Network

nonlinear relationship between x and y



no obvious relationship between pixel values and labels

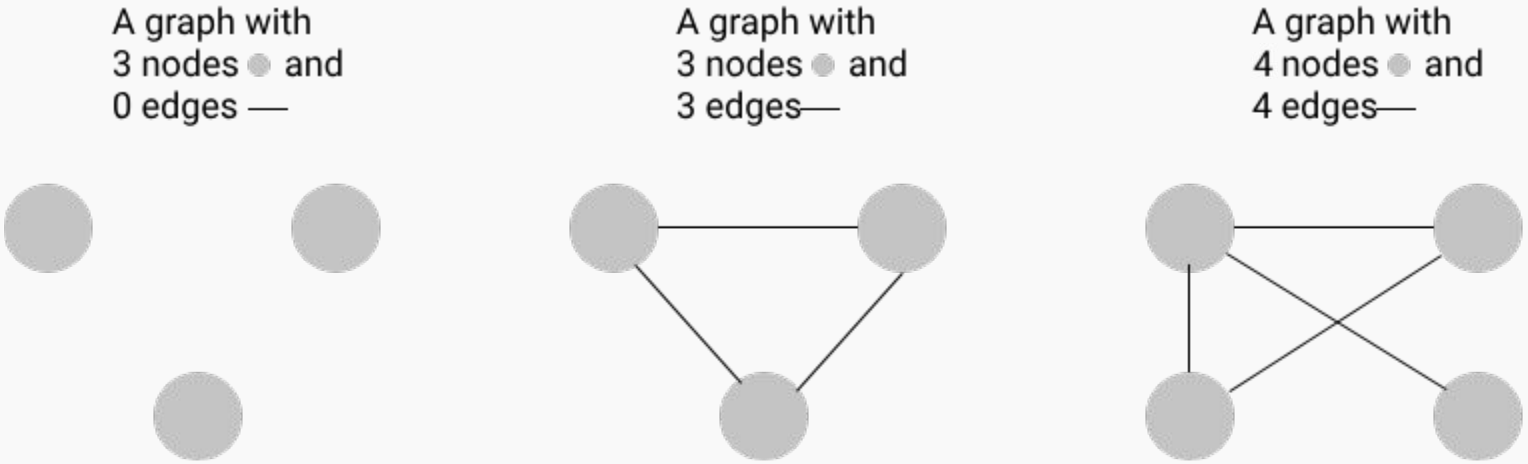


A deep neural network is a specific type of neural network that excels at capturing nonlinear relationships in data

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.



Computational Graphs

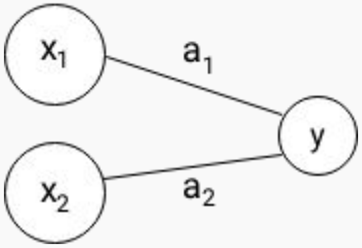
In the context of neural networks, graphs let us compactly express a pipeline of functions that we want to be executed in succession

stage 1 $\sigma\left(\underset{X}{\left[\,100 \times 3\,\right]}\ \underset{a_1^T}{\left[\,3 \times 6\,\right]}\right) = \underset{L_1}{\left[\,100 \times 6\,\right]}$

↓

stage 2 $\sigma\left(\underset{L_1}{\left[\,100 \times 6\,\right]}\ \underset{a_2^T}{\left[\,6 \times 1\,\right]}\right) = \underset{L_2}{\left[\,100 \times 1\,\right]}$

$y = a_1 x_1 + a_2 x_2$

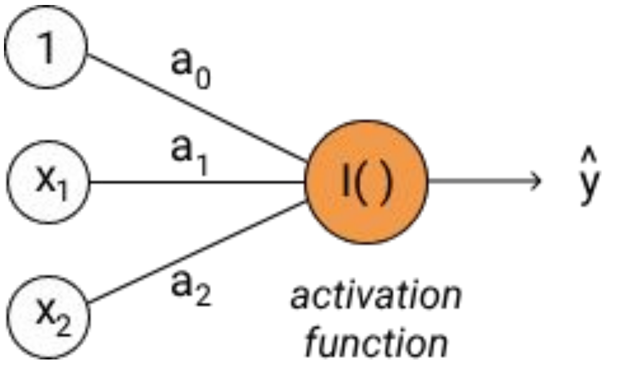


A neural network that performs a linear regression

$$\hat{y} = a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_n x_n$$

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

$$\underbrace{\begin{bmatrix} 1 & 2.3 & 0.2 \\ 1 & 3.1 & 0.9 \\ 1 & 1.1 & 0.5 \end{bmatrix}}_{X} \underbrace{\begin{bmatrix} 0.9 \\ 0.7 \\ 0.5 \end{bmatrix}}_{a^T} = \underbrace{\begin{bmatrix} 2.61 \\ 3.52 \\ 1.92 \end{bmatrix}}_{\hat{y}}$$

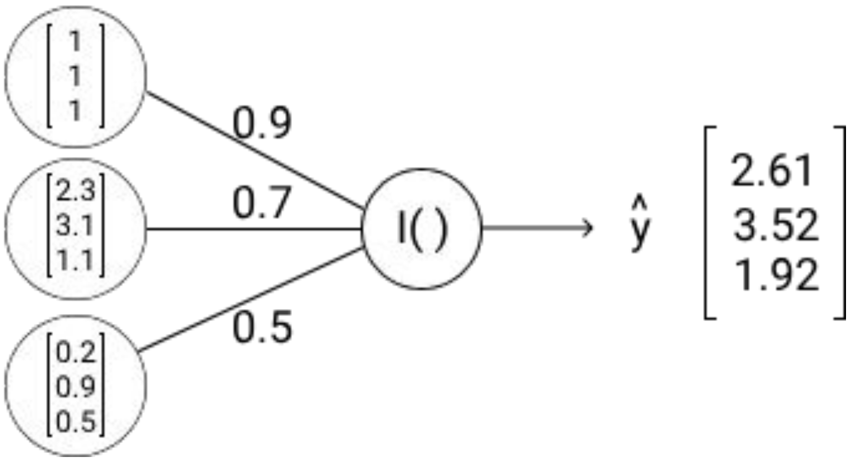


A neural network that performs a linear regression

Linear Algebra

$$l\left(\underset{X}{\begin{bmatrix}1 & 2.3 & 0.2\\ 1 & 3.1 & 0.9\\ 1 & 1.1 & 0.5\end{bmatrix}}\underset{a^T}{\begin{bmatrix}0.9\\ 0.7\\ 0.5\end{bmatrix}}\right)=\underset{\hat{y}}{\begin{bmatrix}2.61\\ 3.52\\ 1.92\end{bmatrix}}$$

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()

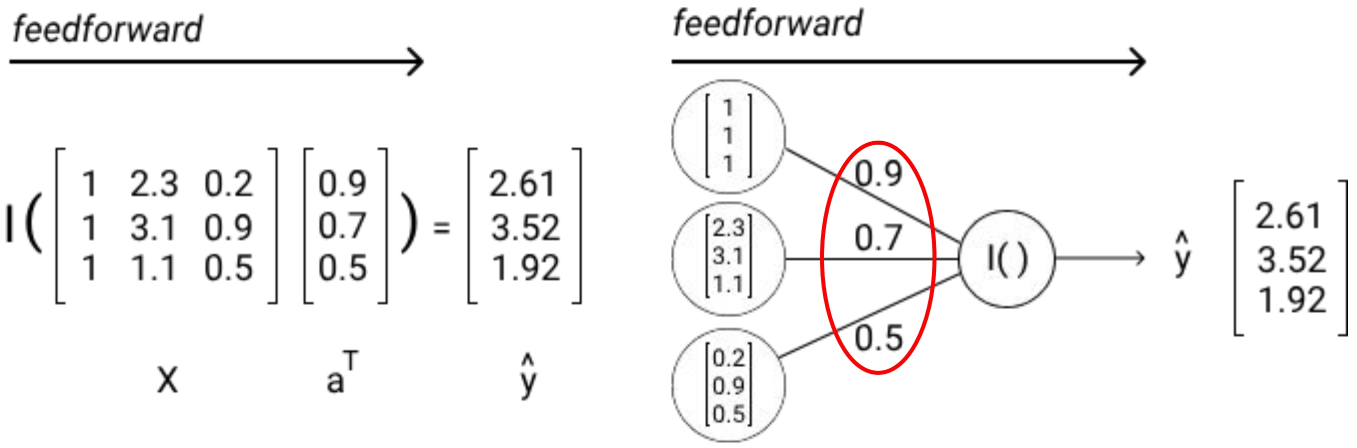
Generating regression data

```python
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



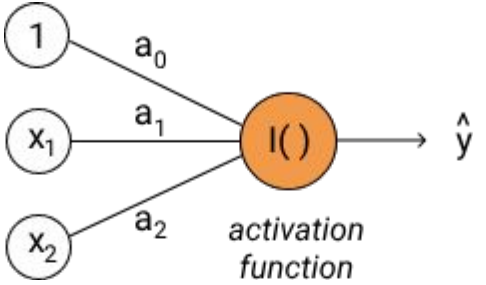

```python
from sklearn.linear_model import SGDRegressor
lr = linear_model.SGDRegressor()
lr.fit(X,y)
```

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
```

```
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
```

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



Generating a classification data

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

```python
# Features
class_features = class_data[0]
class_features[:5]
```

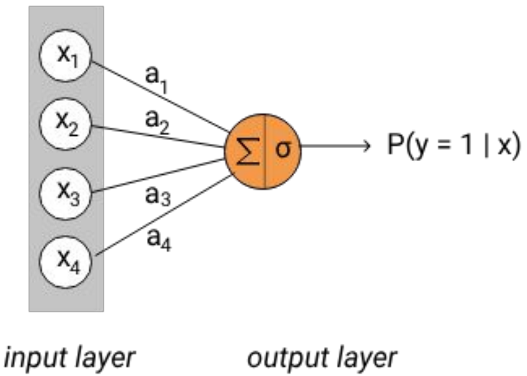
```
array([[ 1.91518414,  1.14995454, -1.52847073,  0.79430654],
       [ 1.4685668 ,  0.80644722, -1.04912964,  0.74652026],
       [ 1.47102089,  0.90060386, -1.20228498,  0.57845433],
       [ 1.07642824, -0.1813636 ,  0.49116807,  1.95642108],
       [-5.34139911, -2.29763222,  2.77907005, -3.87463248]])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

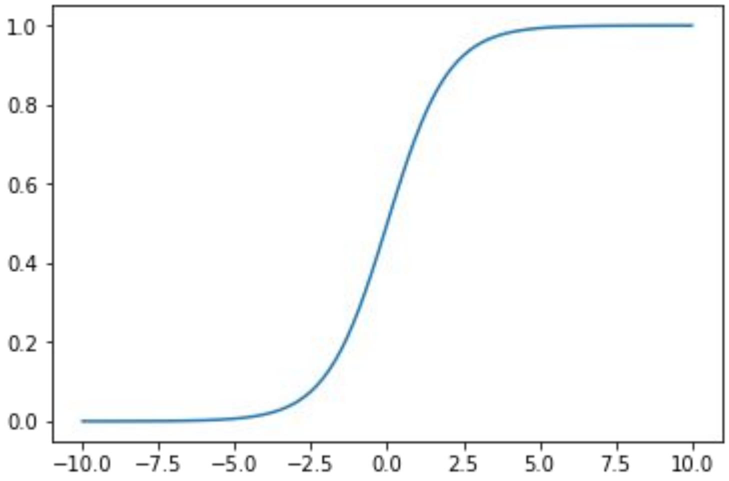
```python
# Labels
class_labels = class_data[1]
class_labels[:5]
```

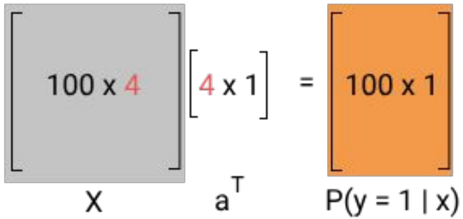
```
array([1, 1, 1, 1, 0])
```

Implementing a neural network that performs classification

Binary Classification







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

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

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

Implementing a Logistic Regression Model

```python
# 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]
```

```python
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
```

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

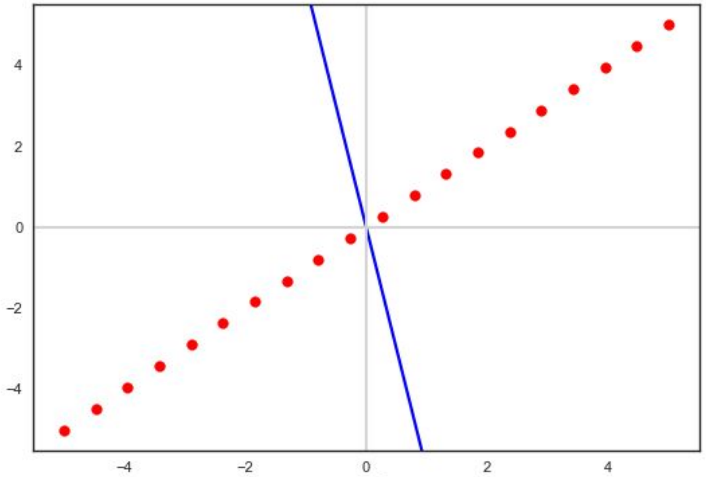
```python
log_train_weights = log_train(class_features,
                              class_labels)
log_predictions = log_feedforward(class_features,
                                  log_train_weights)
```

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

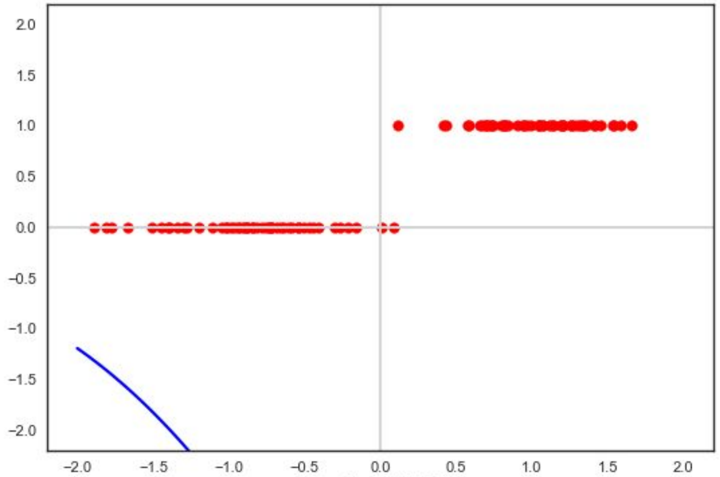
$$P(y = 0|x) < 0.5$$

Activation functions

Nonlinear Activation Functions



y = -6x



y = sigmoid(-10x)

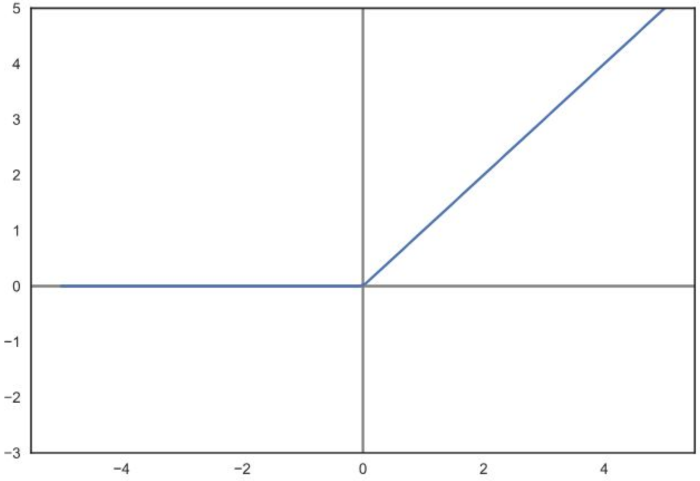
Neural Networks - Activation Functions

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

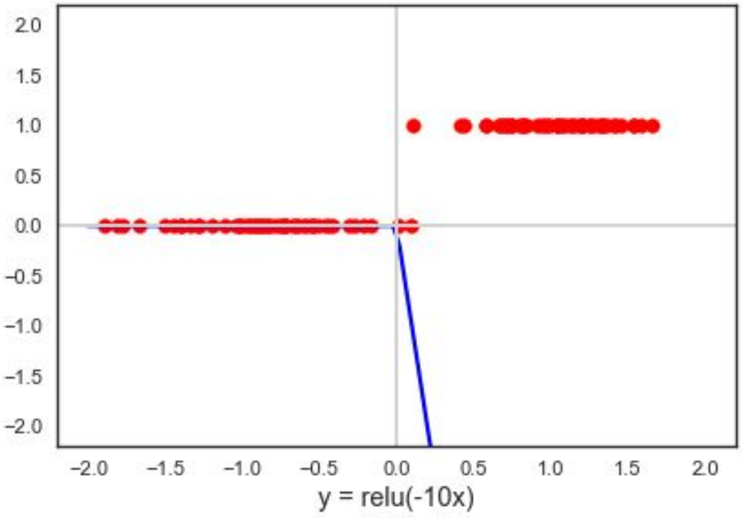
- the sigmoid function
- the ReLU function
- the tanh function

ReLU Function - Rectifier Linear Unit

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

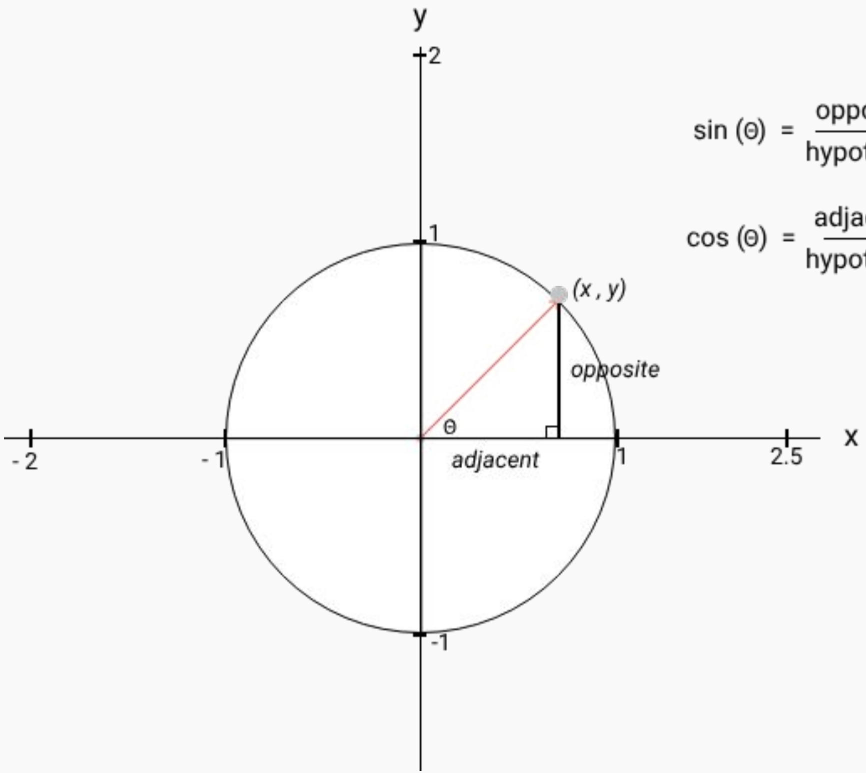


$$ReLU = max(0, x)$$



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

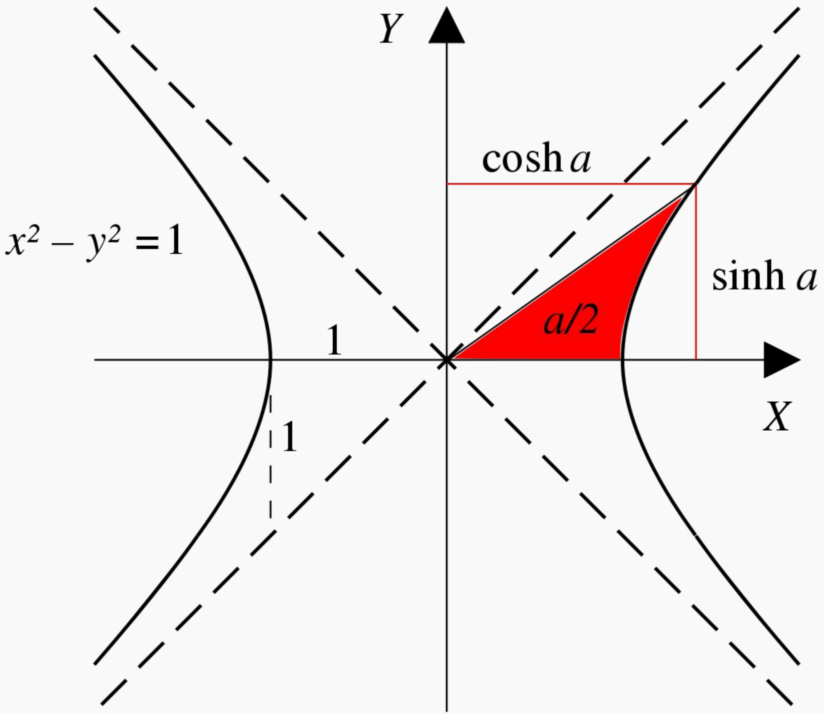
Trigonometric Functions

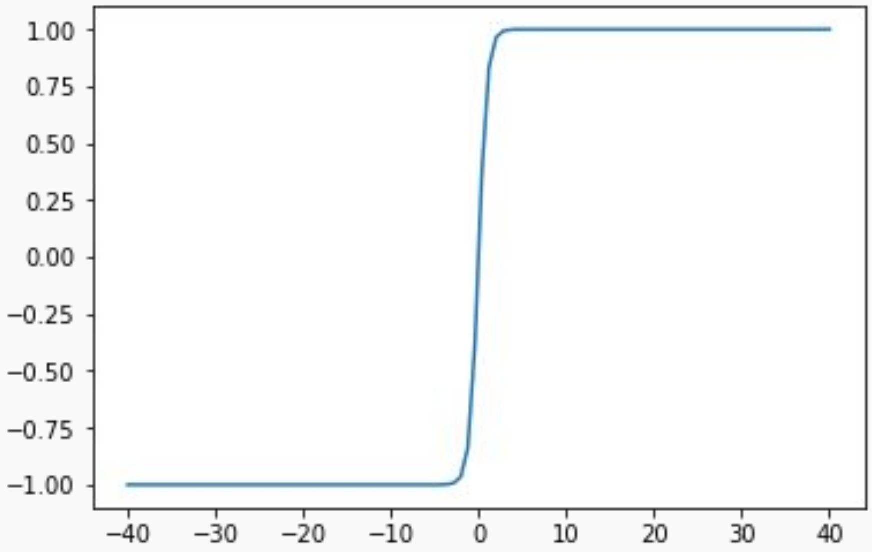


$$\sin(\theta) = \frac{\text{opposite}}{\text{hypotenuse}} = \text{opposite} = y$$

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

Hyperbolic Tangent Function (tanh)

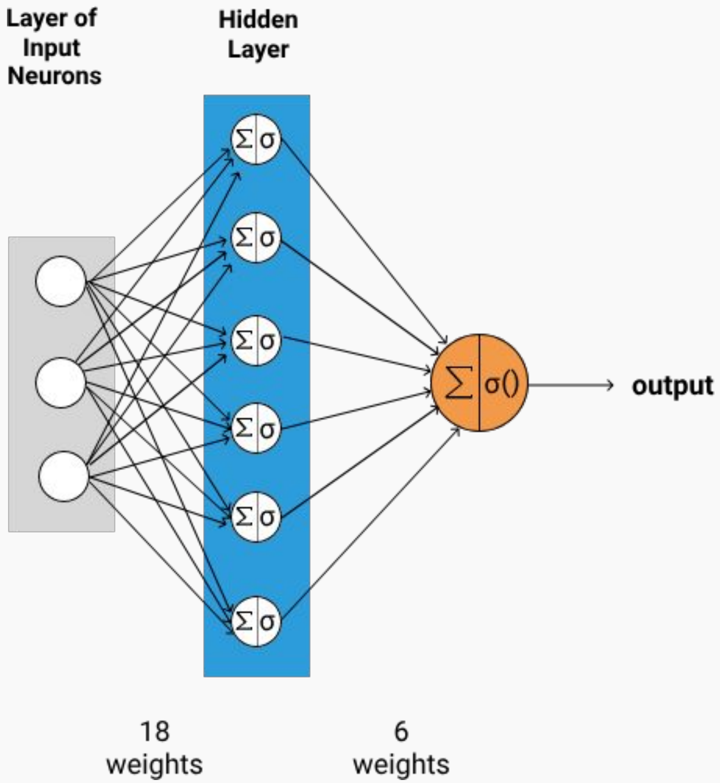




It is commonly used in neural networks for classification tasks.

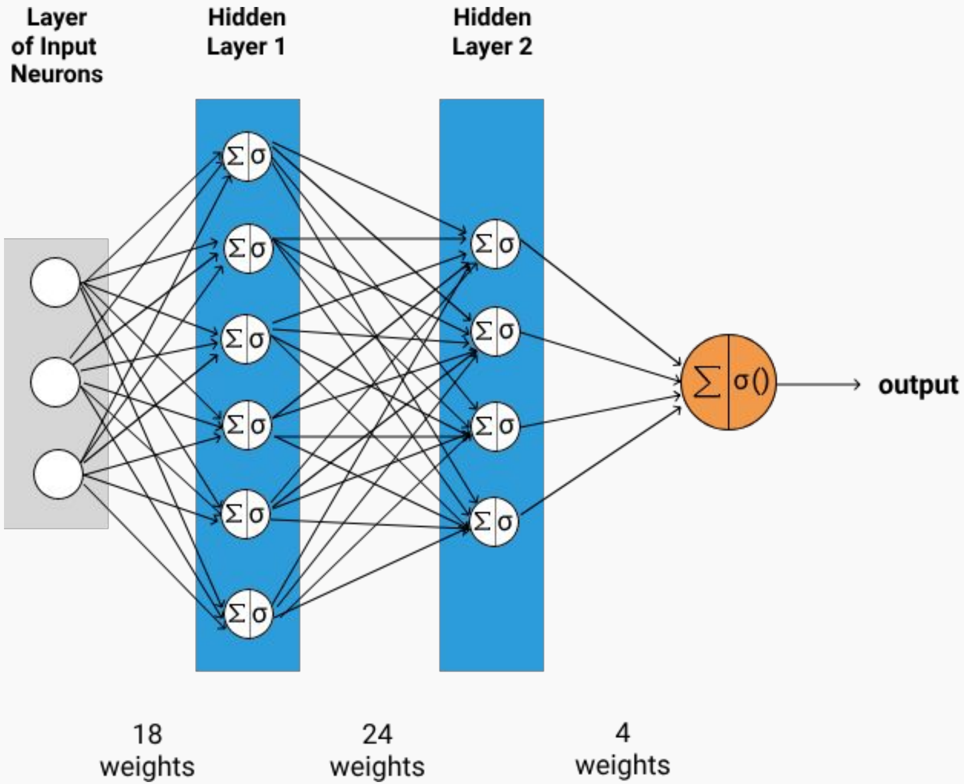
Hidden Layers

Multi-layers networks (deep neural networks)



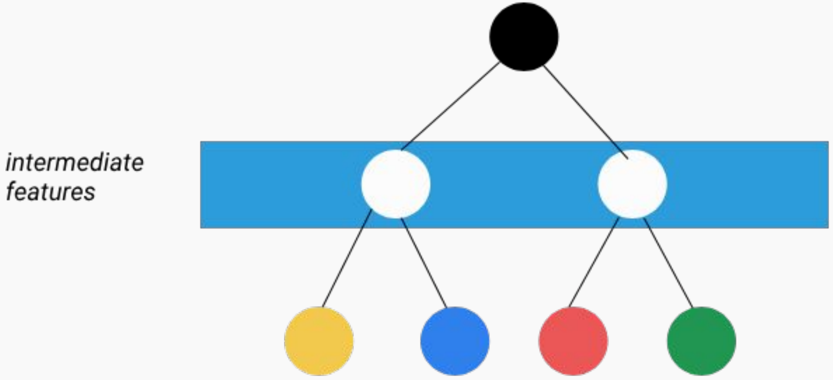
- This kind of models are able to better capture nonlinearity in the data
- Choosing the number of neurons in this layer is a bit of an art
- We can think of each hidden layer as intermediate features that are learned during the training process.

Multi-layers networks (deep neural networks)

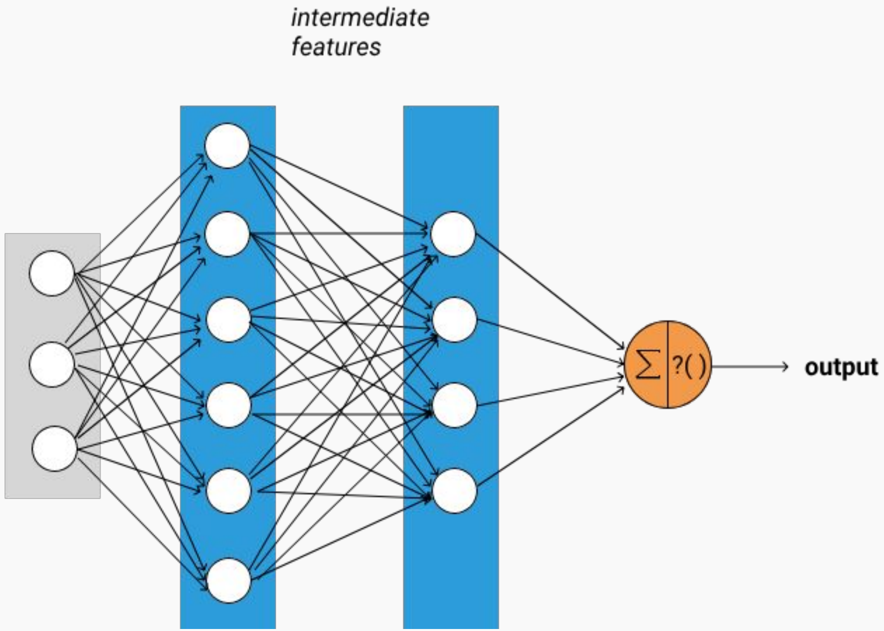


Comparison with Decision Tree Models

Decision Tree

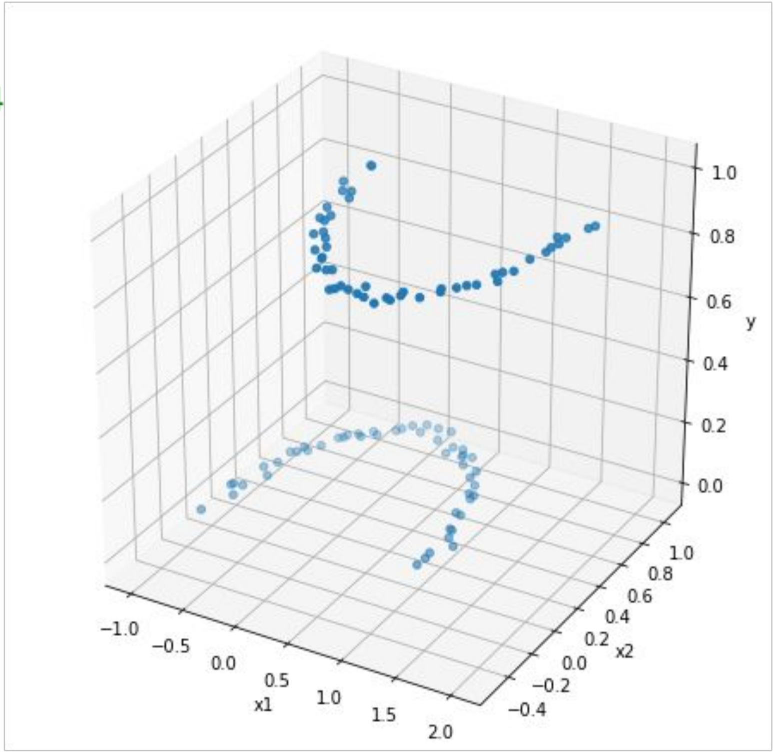


Deep Neural Network



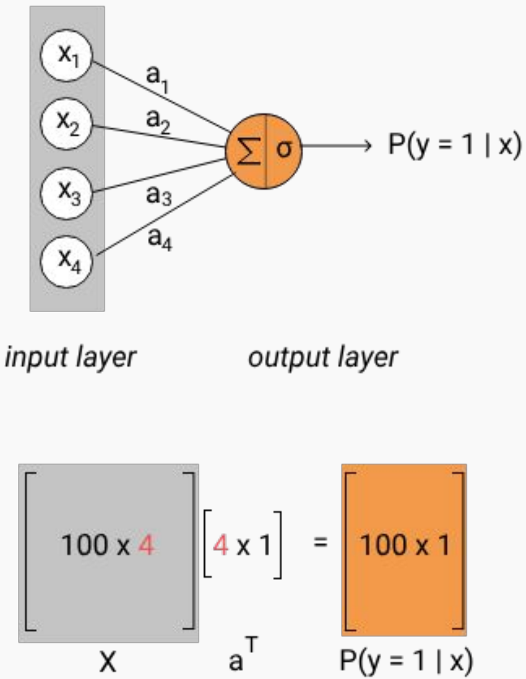
Generating data that contains nonlinearity

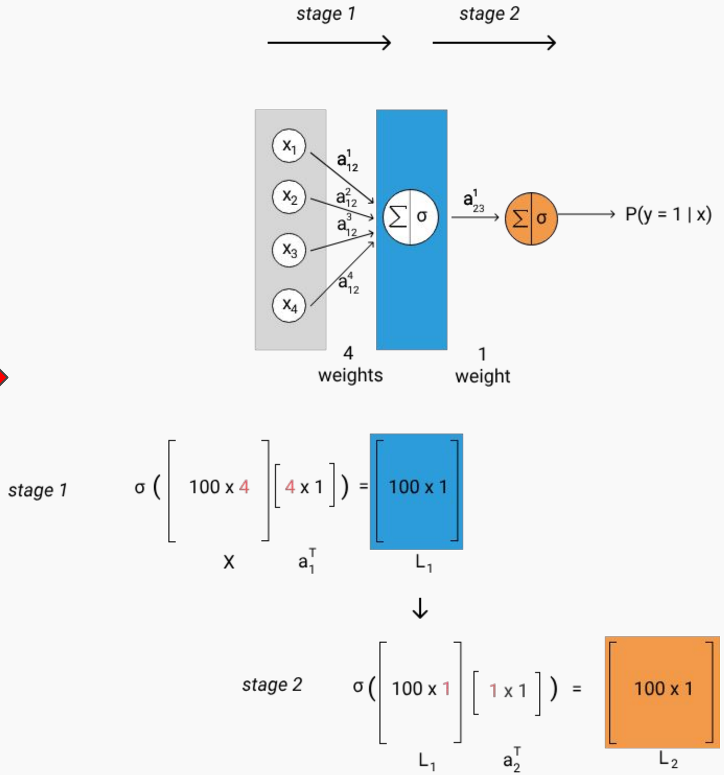
```
data = make_moons(100, random_state=3, noise=0.04)
features = pd.DataFrame(data[0])
labels = pd.Series(data[1])

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





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)
```

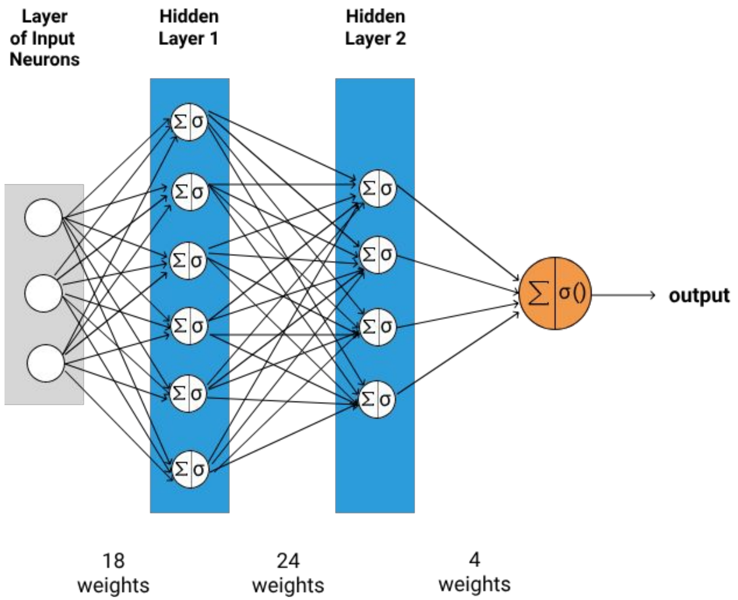
```
mlp = MLPClassifier(hidden_layer_sizes=(6,), activation='logistic')
```

```
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)
```

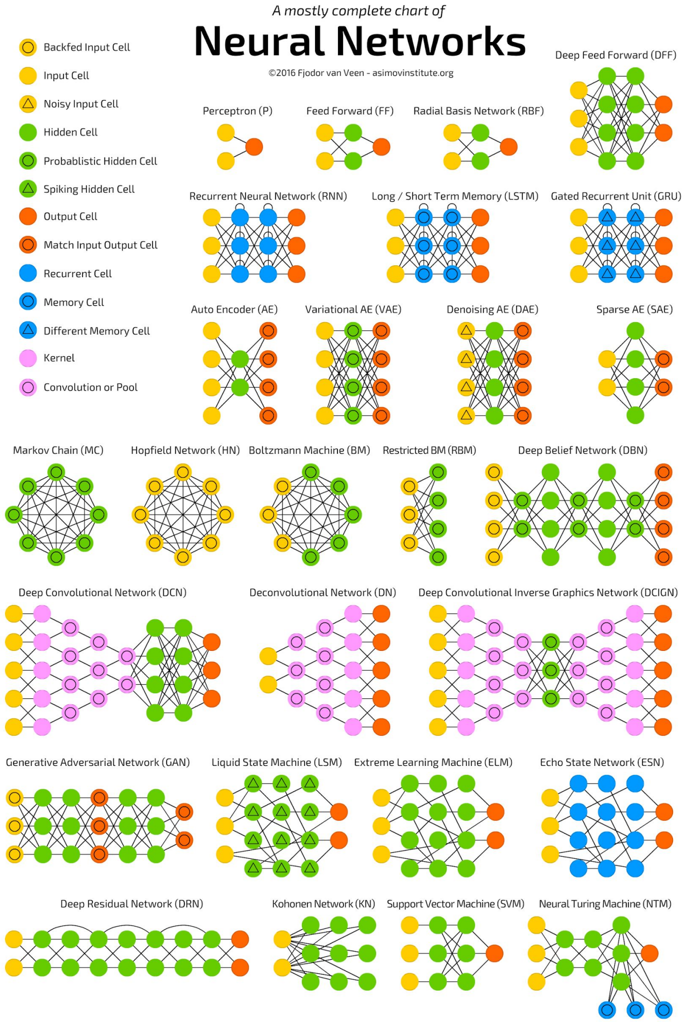
```
mlp = MLPClassifier(hidden_layer_sizes=(4,),
                    activation='logistic',max_iter=10000)
mlp.fit(train_x, train_y)
nn_predictions = mlp.predict(test_x)

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

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

```
0.88
0.88
```

Multiple Hidden Layers



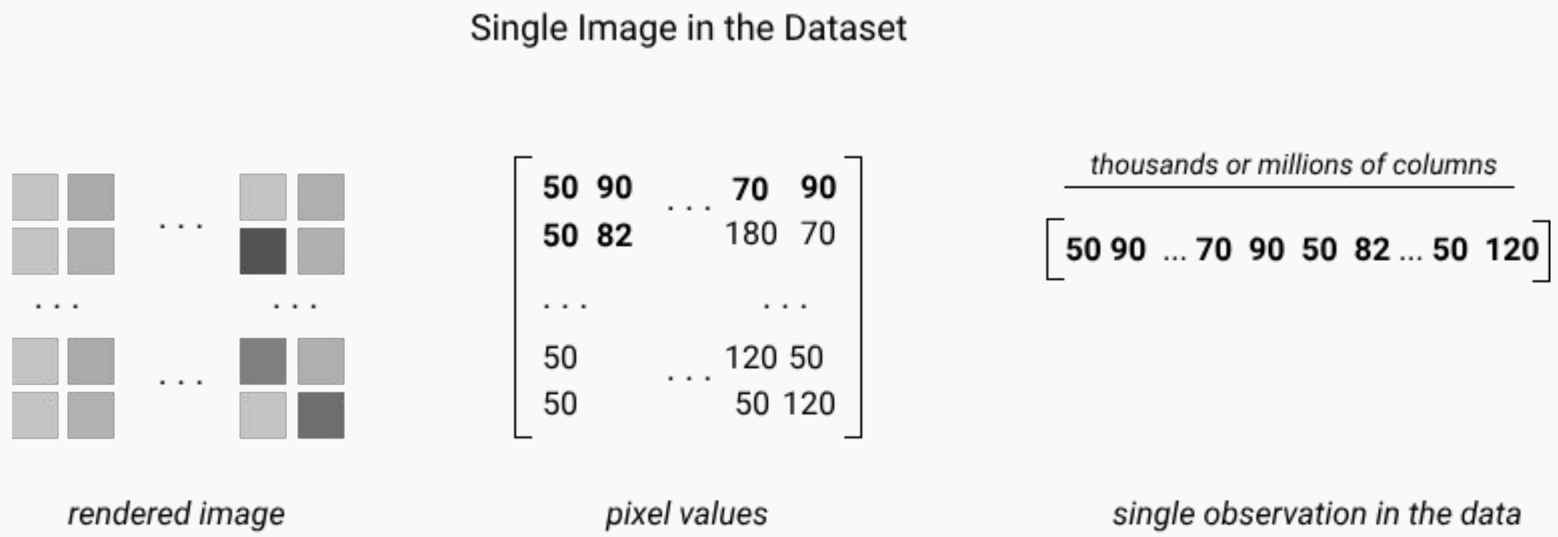

```
mlp = MLPClassifier(hidden_layer_sizes=(n,k))
```


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Lesson #24 - Deep Learning Fundamental I.ipynb

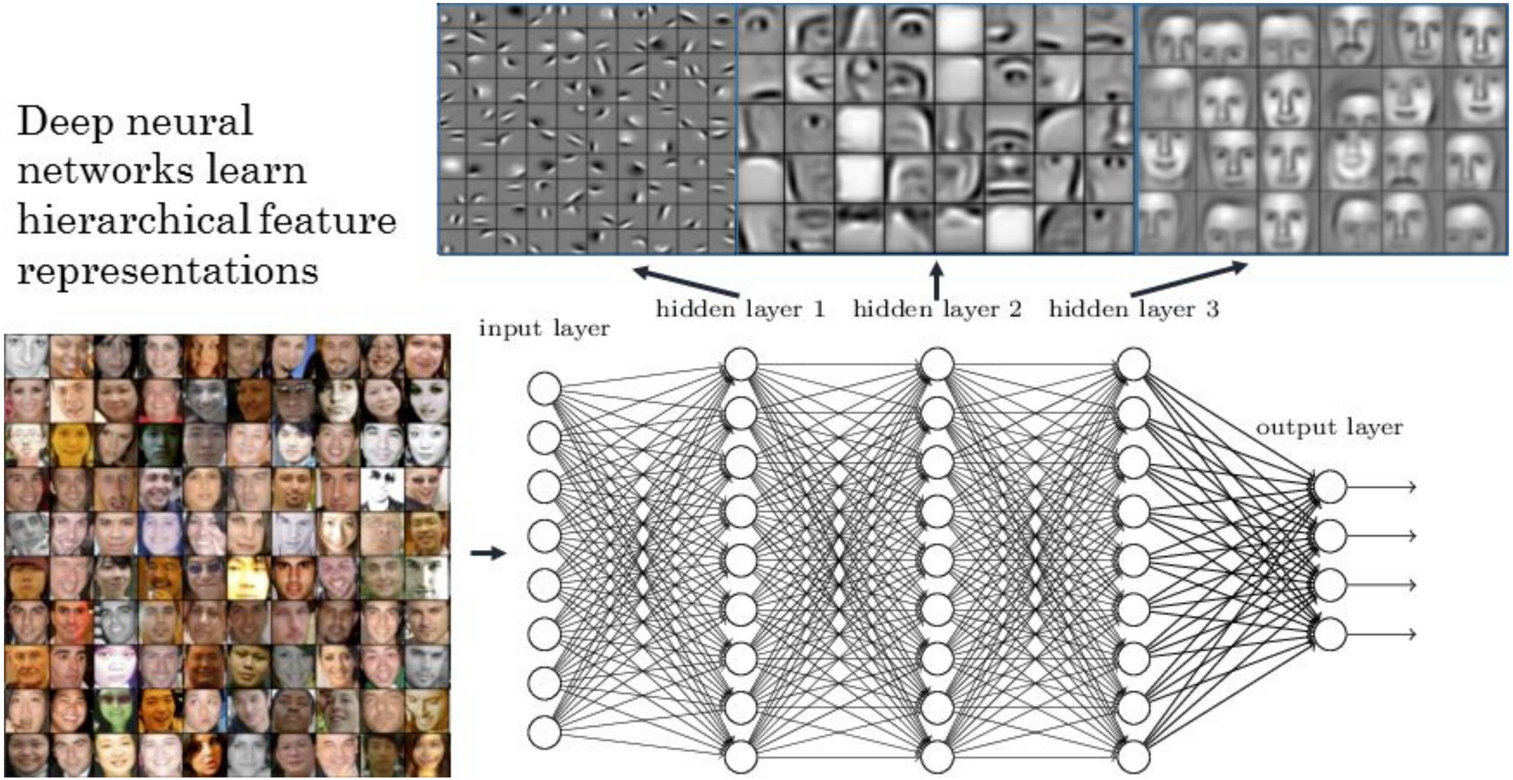
Case Study: building a handwritten digits classifier

Why is image classification a hard task?



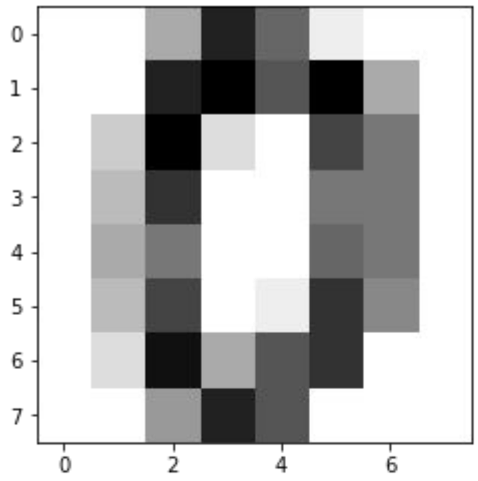
A 128 x 128 image has 16384 features

Why is deep learning effective in image classification?



Working with image data

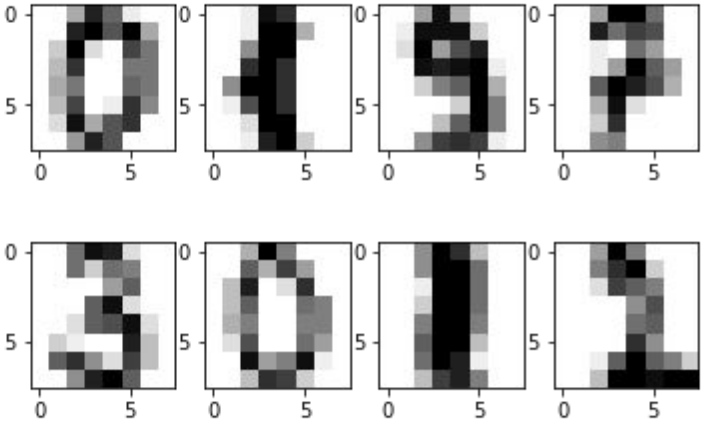
```
from sklearn.datasets import load_digits
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

digits = load_digits()
df_digits = pd.DataFrame(digits["data"])
labels = pd.Series(digits["target"])
```

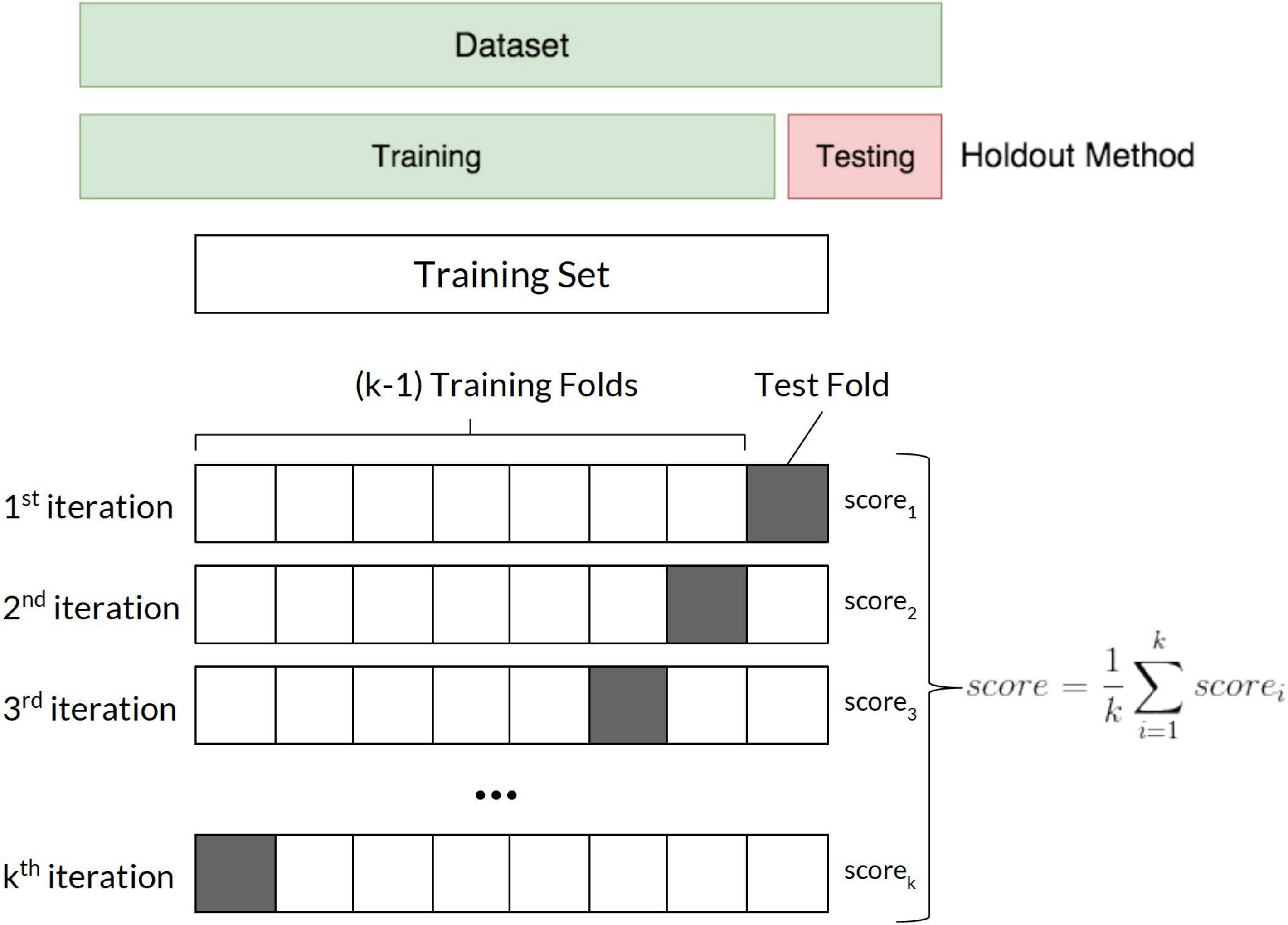



```
first_image = df_digits.iloc[0].values.reshape(8,8)
plt.imshow(first_image, cmap='gray_r')
```

Working with image data

```
fig, ax = plt.subplots(2, 4)

selected_rows = [0,99,199,299,999,1099,1199,1299]

for i,index in enumerate(selected_rows):
    ax[i//4, i%4].imshow(df_digits.iloc[index].values.reshape(8,8), cmap='gray_r')
```



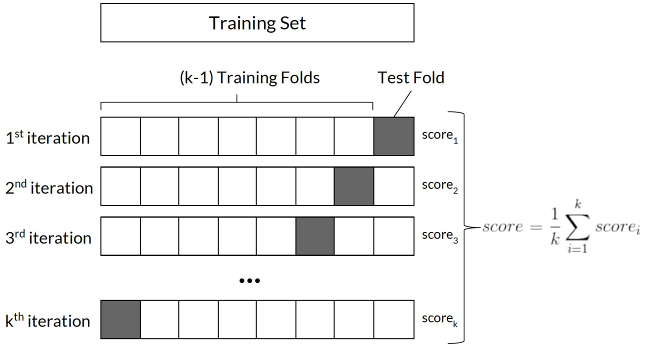

Dataset
Training
Testing
Holdout Method
Training Set
(k-1) Training Folds
Test Fold
1st iteration
2nd iteration
3rd iteration
...
kth iteration
$score_1$
$score_2$
$score_3$
$score_k$
$score = \frac{1}{k} \sum_{i=1}^{k} score_i$

Cross-Validation - Step #1


```python
# Split dataset into train and test
train_x, test_x, train_y, test_y = train_test_split(df_digits,
                                                     labels,
                                                     test_size=0.30,
                                                     random_state=42)
```

df_digits + labels = Dataset

Cross-Validation - Step #2




```
# k-fold validation
# df_digits is our original data
kf = KFold(n_splits = 4, random_state=2)

for train_index, test_index in kf.split(train_x):

    # split each fold into train and test

    # create a model

    # make predictions

    # evaluate accuracy

np.mean(fold_accuracies)
```

K-Nearest Neighbor Model

```
# number of neighbors
neighbors = 3

for train_index, test_index in kf.split(train_x):

    # split each fold into train and test
    train_features, test_features = train_x.iloc[train_index],train_x.iloc[test_index]
    train_labels, test_labels = train_y.iloc[train_index],train_y.iloc[test_index]

    # create a knn classifier model
    knn = KNeighborsClassifier(n_neighbors = neighbors)
    knn.fit(train_features, train_labels)

    # make predictions
    predictions = knn.predict(test_features)

    # evaluate accuracy
    overall_accuracy = accuracy_score(test_labels, predictions)
    fold_accuracies.append(overall_accuracy)

np.mean(fold_accuracies)
```

```
[0.9809523809523809,
 0.9840764331210191,
 0.9968152866242038,
 0.9745222929936306]

0.9840915984228086
```

K-Nearest Neighbor - Final Model

```
# number of neighbors
neighbors = 3

# create a knn classifier model
knn = KNeighborsClassifier(n_neighbors = neighbors)
knn.fit(train_x, train_y)

# make predictions
predictions = knn.predict(test_x)

# evaluate accuracy
overall_accuracy = accuracy_score(test_y, predictions)
overall_accuracy
```

```
0.9888888888888889
```

Neural Network with one hidden layer

```python
# architecture of network
# one hidden layer with 64 neurons
nn_one_neurons = (64,)
```

```python
for train_index, test_index in kf.split(train_x):

    # split each fold into train and test
    train_features, test_features = train_x.iloc[train_index], train_x.iloc[test_index]
    train_labels, test_labels = train_y.iloc[train_index], train_y.iloc[test_index]

    # create a MLP classifier model
    mlp = MLPClassifier(hidden_layer_sizes=nn_one_neurons,
                        max_iter=500,activation="logistic",
                        solver='adam')
    mlp.fit(train_features, train_labels)

    # make predictions
    predictions = mlp.predict(test_features)

    # evaluate accuracy
    overall_accuracy = accuracy_score(test_labels, predictions)
    fold_accuracies.append(overall_accuracy)
```

```
[0.9555555555555556,
 0.9585987261146497,
 0.9904458598726115,
 0.9426751592356688]
```

```
0.9618188251946214
```

NN with one hidden layer - Final Model

```python
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import KFold

# architecture of network
# one hidden layer with 64 neurons
nn_one_neurons = (64,)

# create a MLP classifier model
mlp = MLPClassifier(hidden_layer_sizes=nn_one_neurons,
                        max_iter=500,activation="logistic",
                        solver='adam')
mlp.fit(train_x, train_y)

# make predictions
predictions = mlp.predict(test_x)

# evaluate accuracy
overall_accuracy = accuracy_score(test_y, predictions)
```

```
0.9796296296296296
```

NN with two or three layers hidden layers

```
# architecture of network

# two hidden layers with 128 neurons
nn_two_neurons = (128,128)
# three hidden layers with 128 neurons
nn_three_neurons = (128,128,128)
```

?