

# Applied Machine Learning

## Regression: Recap

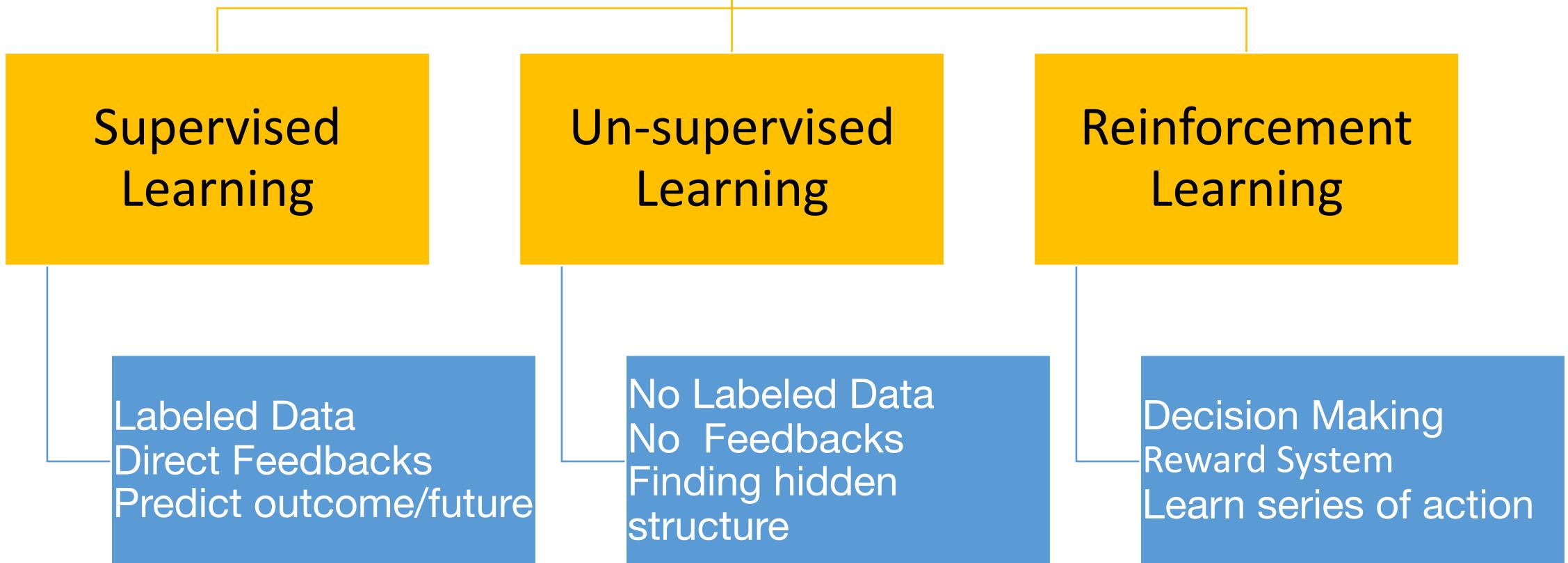
# Machine Learning

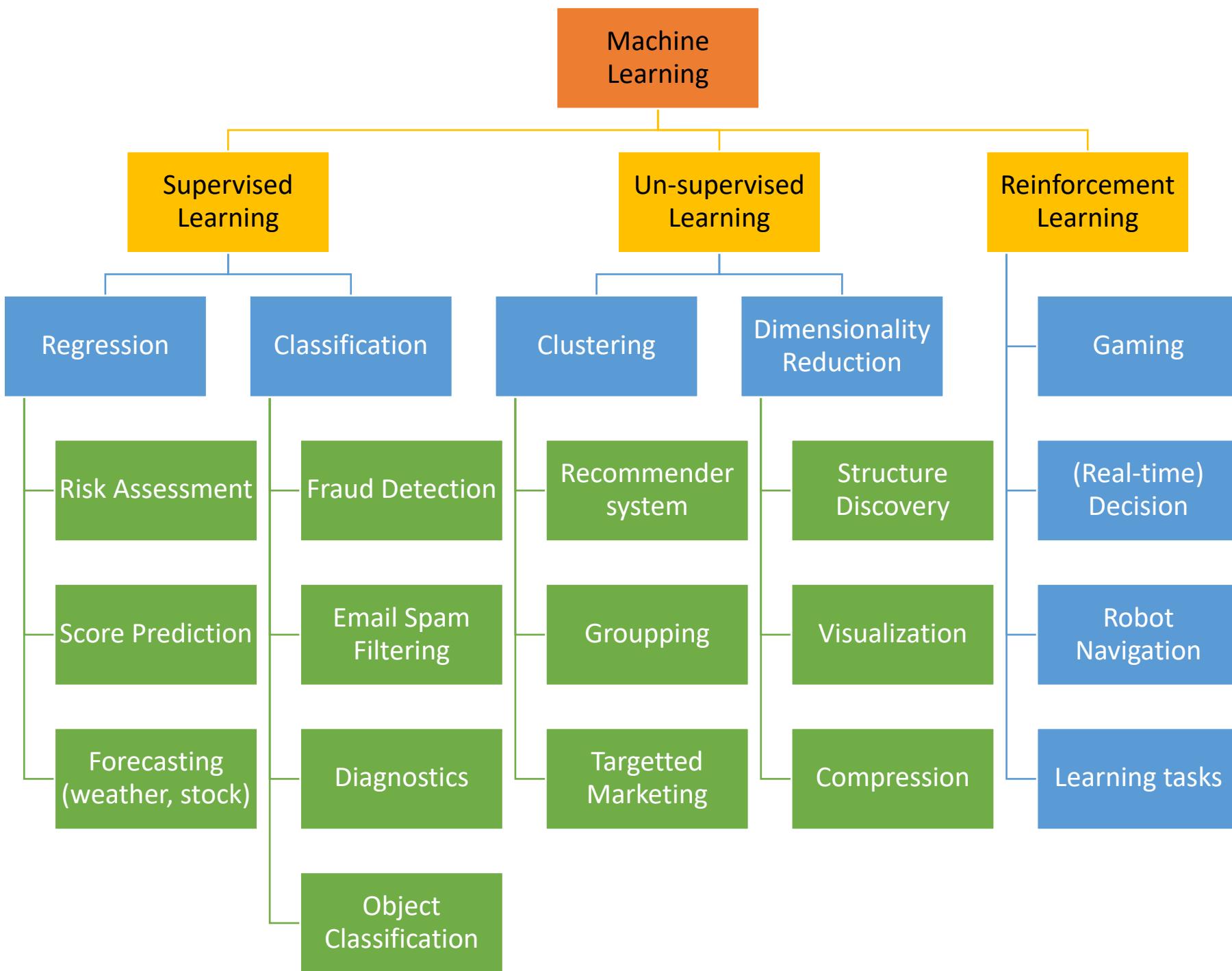
Supervised  
Learning

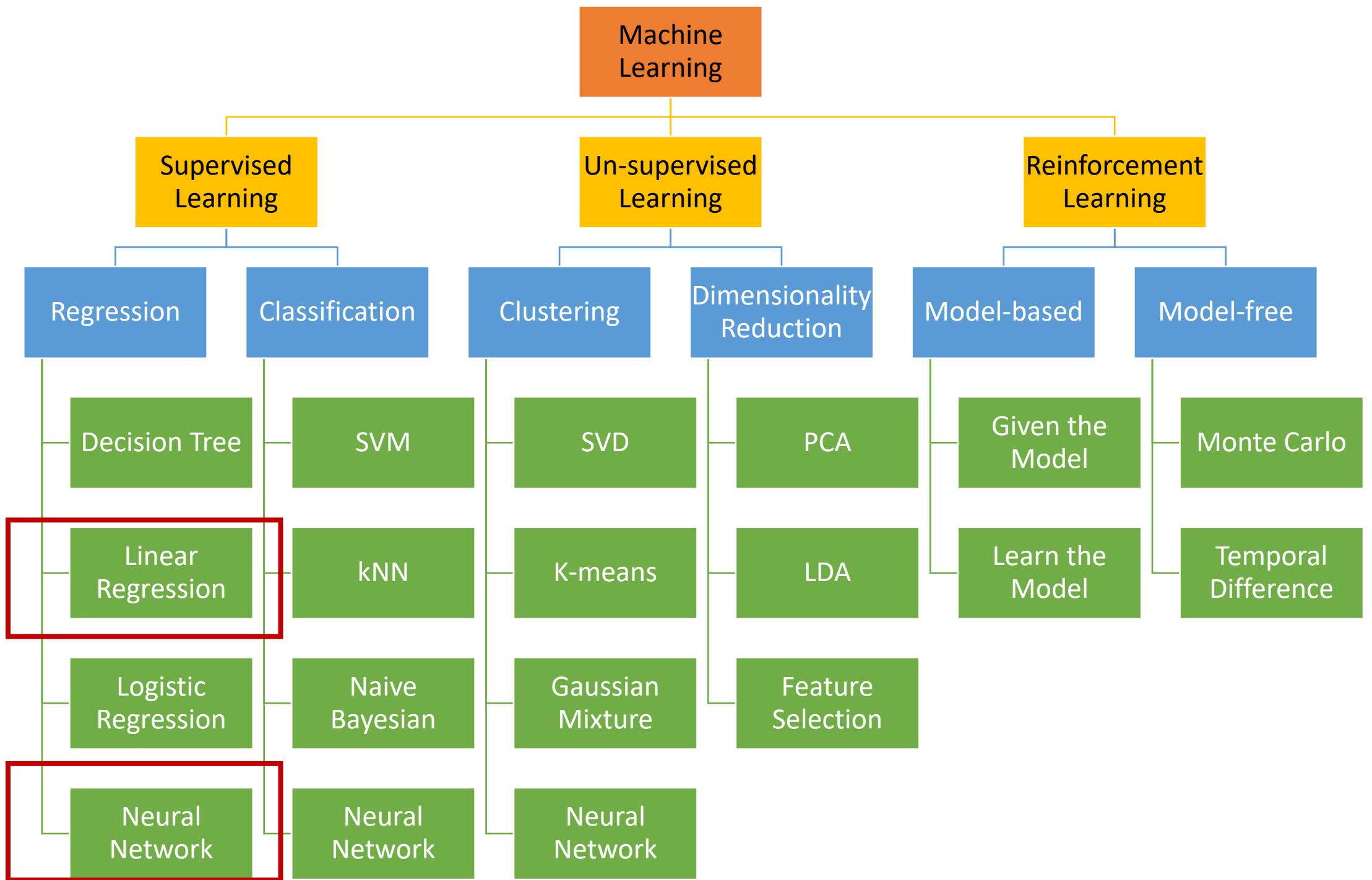
Un-supervised  
Learning

Reinforcement  
Learning

# Machine Learning







## Load data



Numpy.loadtxt

Panda.read\_csv

## Preprocessing data



sklearn.model\_selection.train\_test\_split

sklearn.preprocessing.StandardScaler

sklearn.preprocessing.Normalizer

## Model Selection



sklearn.linear\_model.LinearRegression

## Evaluate Performance



sklearn.metrics.mean\_absolute\_error

sklearn.metrics import  
mean\_squared\_error

sklearn.metrics import r2\_score

# Python For Data Science Cheat Sheet

## Scikit-Learn

Learn Python for data science interactively at [www.DataCamp.com](https://www.DataCamp.com)



### Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

#### A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

### Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

### Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
...                                                     y,
...                                                     random_state=0)
```

### Preprocessing The Data

#### Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

#### Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

#### Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

## Create Your Model

### Supervised Learning Estimators

#### Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

#### Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

#### Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

#### KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

### Unsupervised Learning Estimators

#### Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

#### K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

## Model Fitting

### Supervised learning

```
>>> lr.fit(X, y)
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```

### Unsupervised Learning

```
>>> k_means.fit(X_train)
>>> pca_model = pca.fit_transform(X_train)
```

### Fit the model to the data

Fit the model to the data  
Fit to data, then transform it

## Prediction

### Supervised Estimators

```
>>> y_pred = svc.predict(np.random.random((2,5)))
>>> y_pred = lr.predict(X_test)
>>> y_pred = knn.predict_proba(X_test)
```

Predict labels  
Predict labels  
Estimate probability of a label

### Unsupervised Estimators

```
>>> y_pred = k_means.predict(X_test)
```

Predict labels in clustering algos

### Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

### Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

### Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

## Evaluate Your Model's Performance

### Classification Metrics

#### Accuracy Score

```
>>> knn.score(X_test, y_test)
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Estimator score method  
Metric scoring functions

#### Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

Precision, recall, f1-score  
and support

#### Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

### Regression Metrics

#### Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

#### Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

#### R<sup>2</sup> Score

```
>>> from sklearn.metrics import r2_score
```

```
>>> r2_score(y_true, y_pred)
```

### Clustering Metrics

#### Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

#### Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

#### V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

### Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

## Tune Your Model

### Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
...            "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
...                      param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

### Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
...            "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knk,
...                               param_distributions=params,
...                               cv=4,
...                               n_iter=8,
...                               random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

Source:  
<https://www.datacamp.com/community/blog/scikit-learn-cheat-sheet>



# Recap: Regression

**Regression:** Estimating the relationships between a dependent variable and one or more independent variables. (e.g. we estimated the height from the foot size)

- Linear Regression

$$\text{straight line: } y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad i = 1, \dots, n.$$

- Non-linear Regression

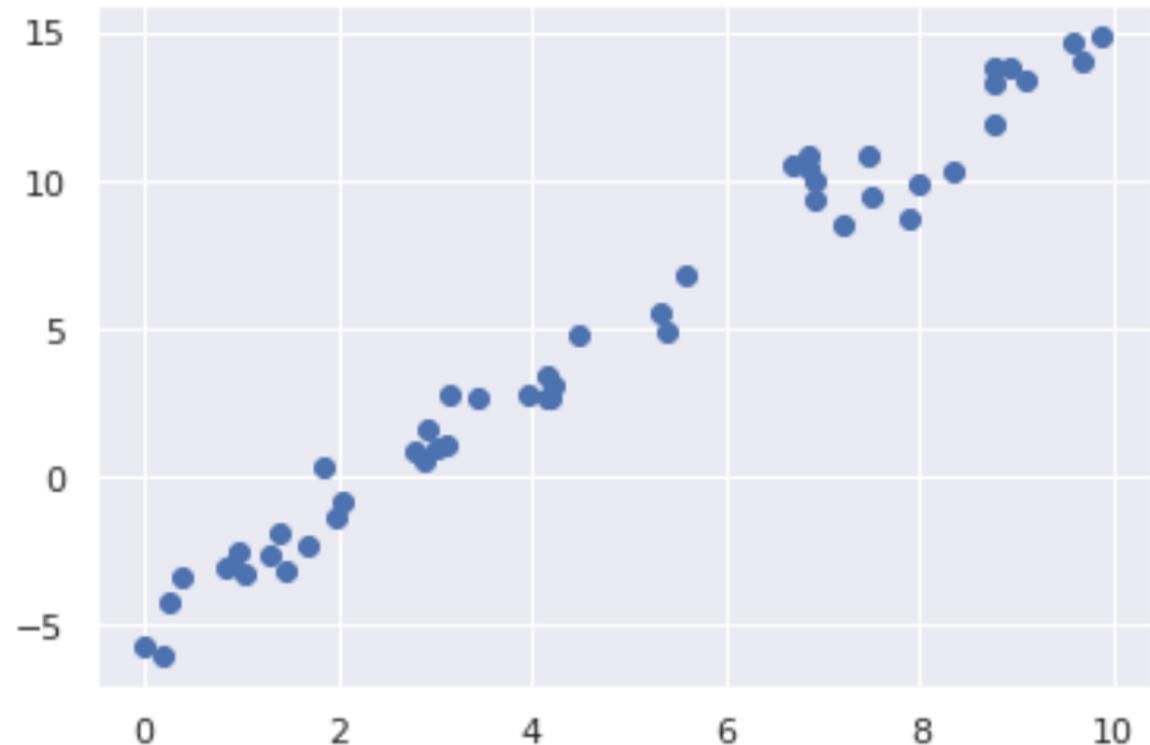
- Polynomial Regression
- Regression ANNs

# Linear Regression

We will start with the most familiar linear regression, a straight-line fit to data.

```
import matplotlib.pyplot as plt
import numpy as np

rng = np.random.RandomState(1)
x = 10 * rng.rand(50)
y = 2 * x - 5 + rng.randn(50)
plt.scatter(x, y);
```



# Linear Regression

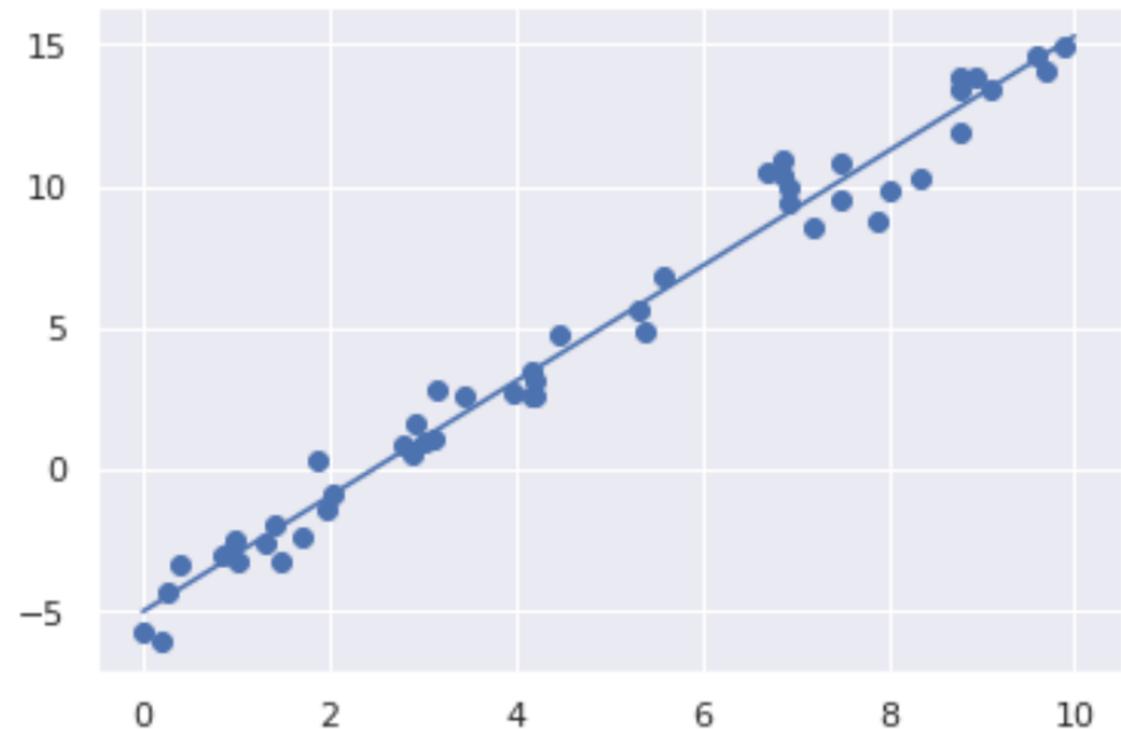
We can use Scikit-learn's *LinearRegression* estimator to fit this data and construct the best-fit line:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)

model.fit(x[:, np.newaxis], y)

xfit = np.linspace(0, 10, 1000)
yfit = model.predict(xfit[:, np.newaxis])

plt.scatter(x, y)
plt.plot(xfit, yfit);
```



# Linear Regression

We can also use tensorflow to fit this data and construct the best-fit line:

```
import tensorflow as tf
from tensorflow.keras import layers

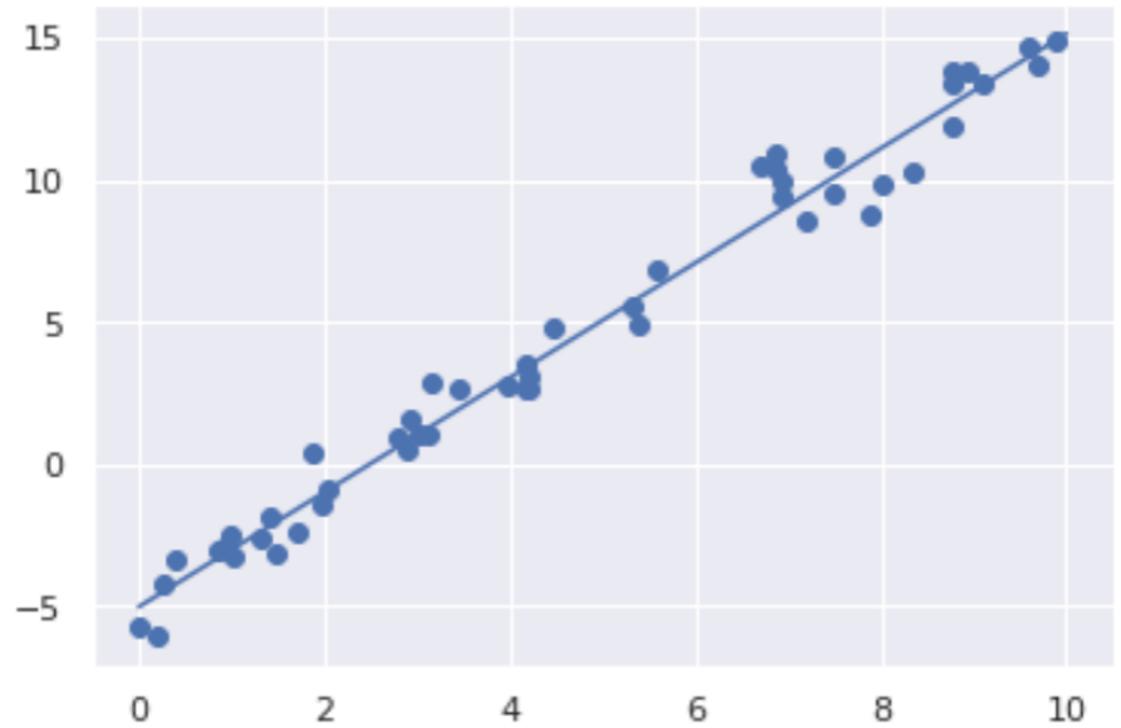
model = tf.keras.Sequential([
    layers.Dense(units=1)
])

model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.1),
    loss='mean_squared_error'
)

model.fit(x, y, epochs=100)

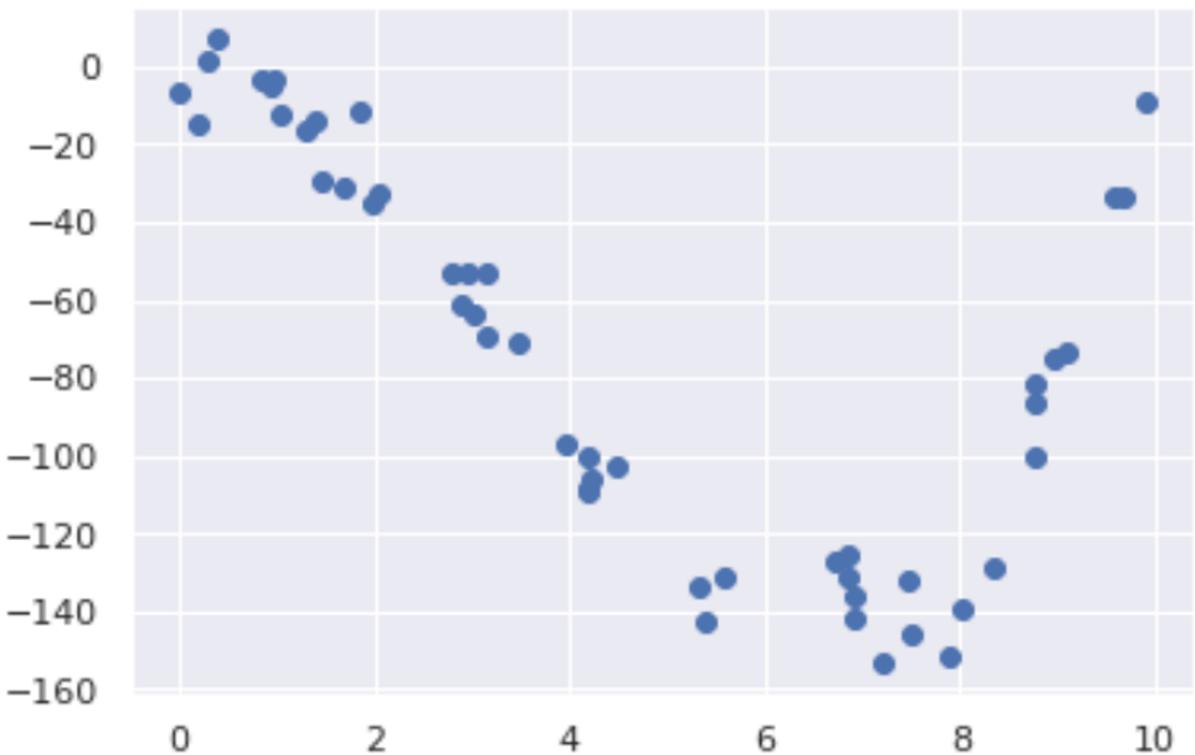
xfit = np.linspace(0, 10, 1000)
yfit = model.predict(xfit)

plt.scatter(x, y)
plt.plot(xfit, yfit);
```



# Polynomial Regression

```
import matplotlib.pyplot as plt  
import numpy as np  
  
rng = np.random.RandomState(1)  
x = 10 * rng.rand(50)  
y = x**3 - 10*x**2 + 10 * rng.randn(50)  
plt.scatter(x, y);
```



# Polynomial Regression

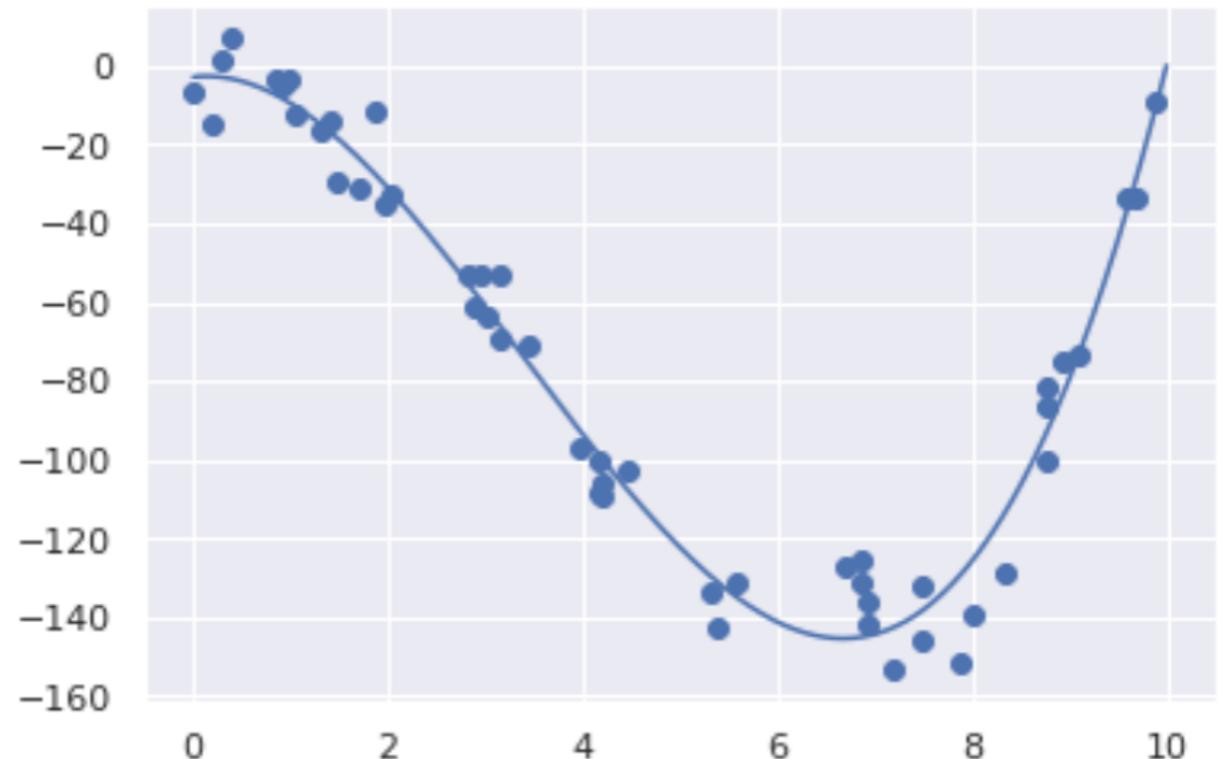
You can use Sklearn's PolynomialFeatures for the polynomial regression

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

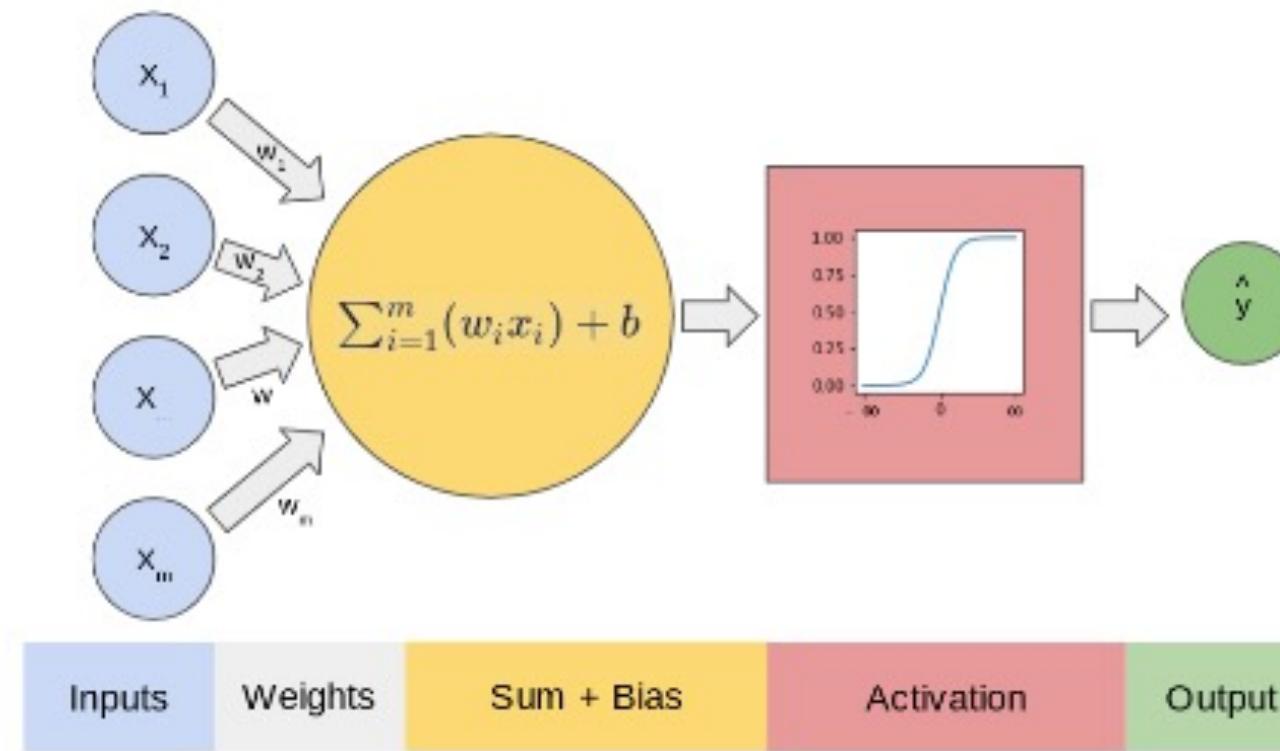
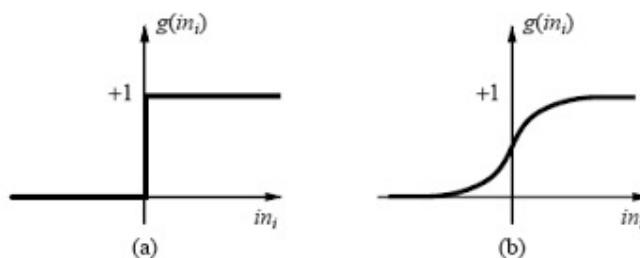
model = make_pipeline(
    PolynomialFeatures(degree=4),
    LinearRegression()
)
model.fit(x[:, np.newaxis], y)

xfit = np.linspace(0, 10, 1000)
yfit = model.predict(xfit[:, np.newaxis])

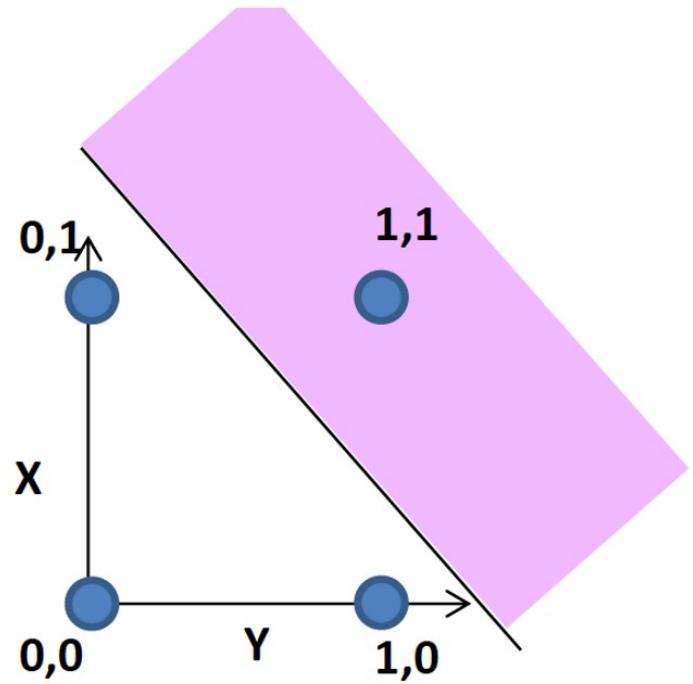
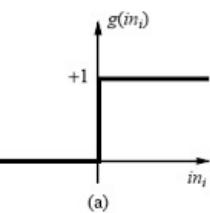
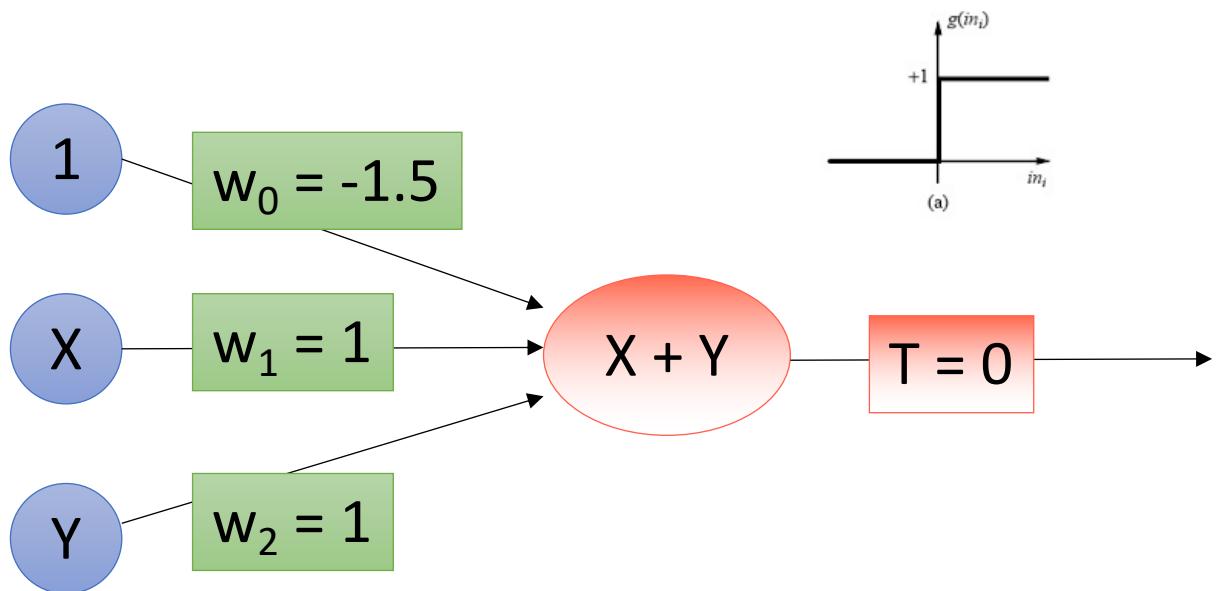
plt.scatter(x, y)
plt.plot(xfit, yfit);
```



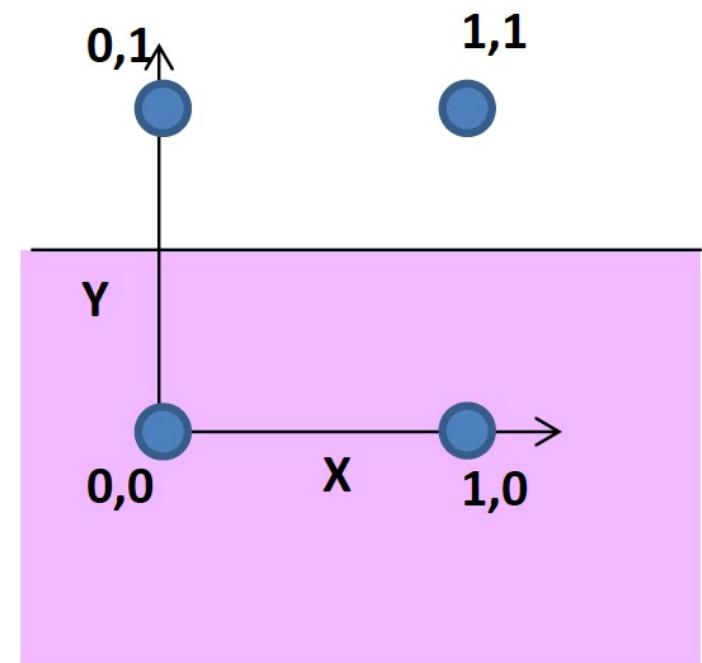
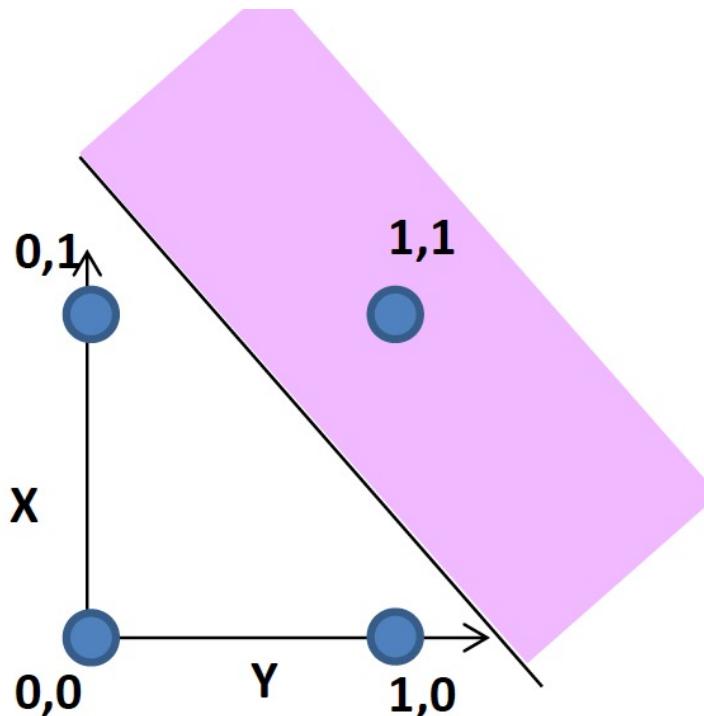
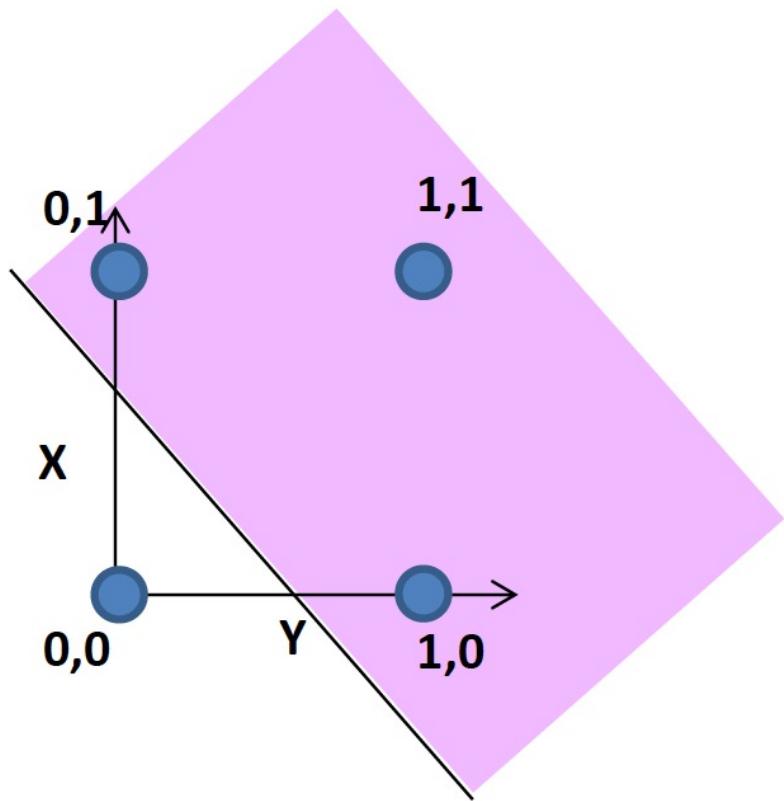
# Perceptron



# Perceptron

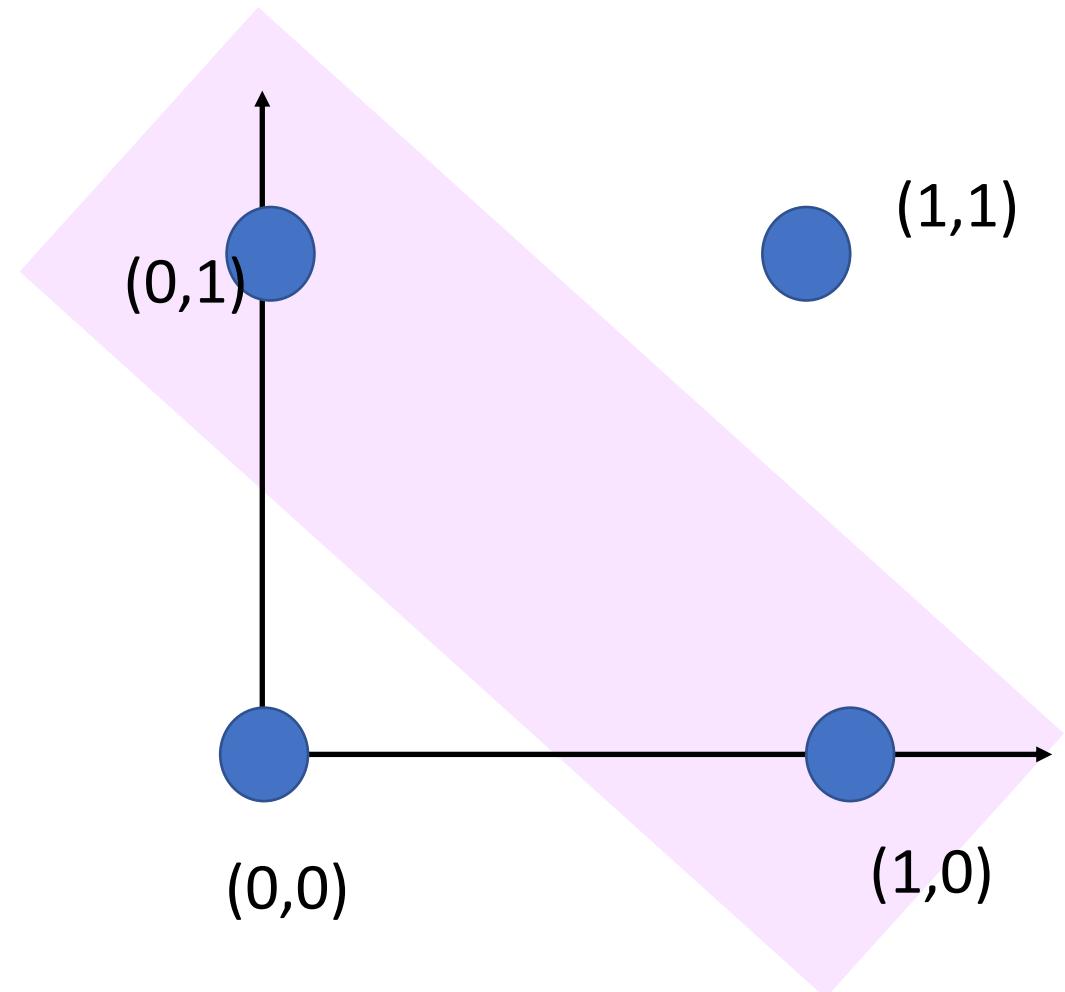


# Perceptron

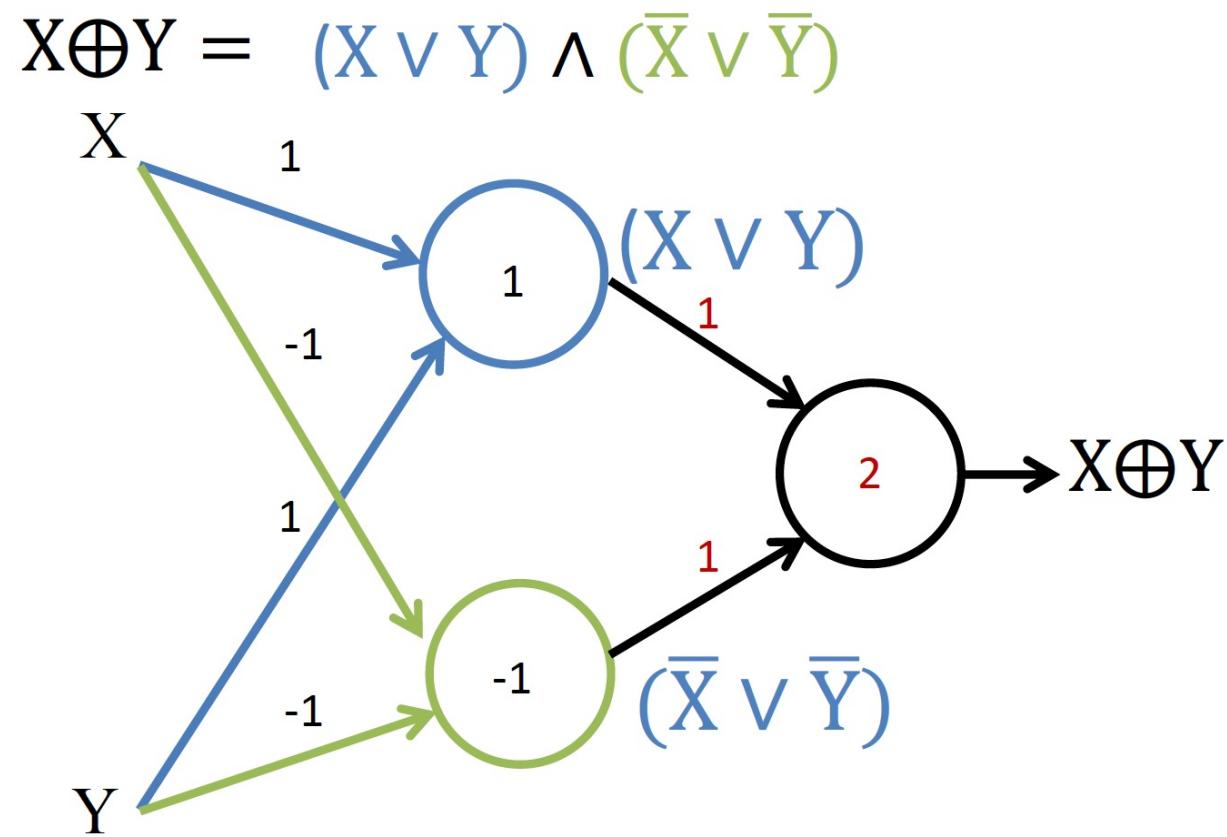


# Neuron - Perceptron

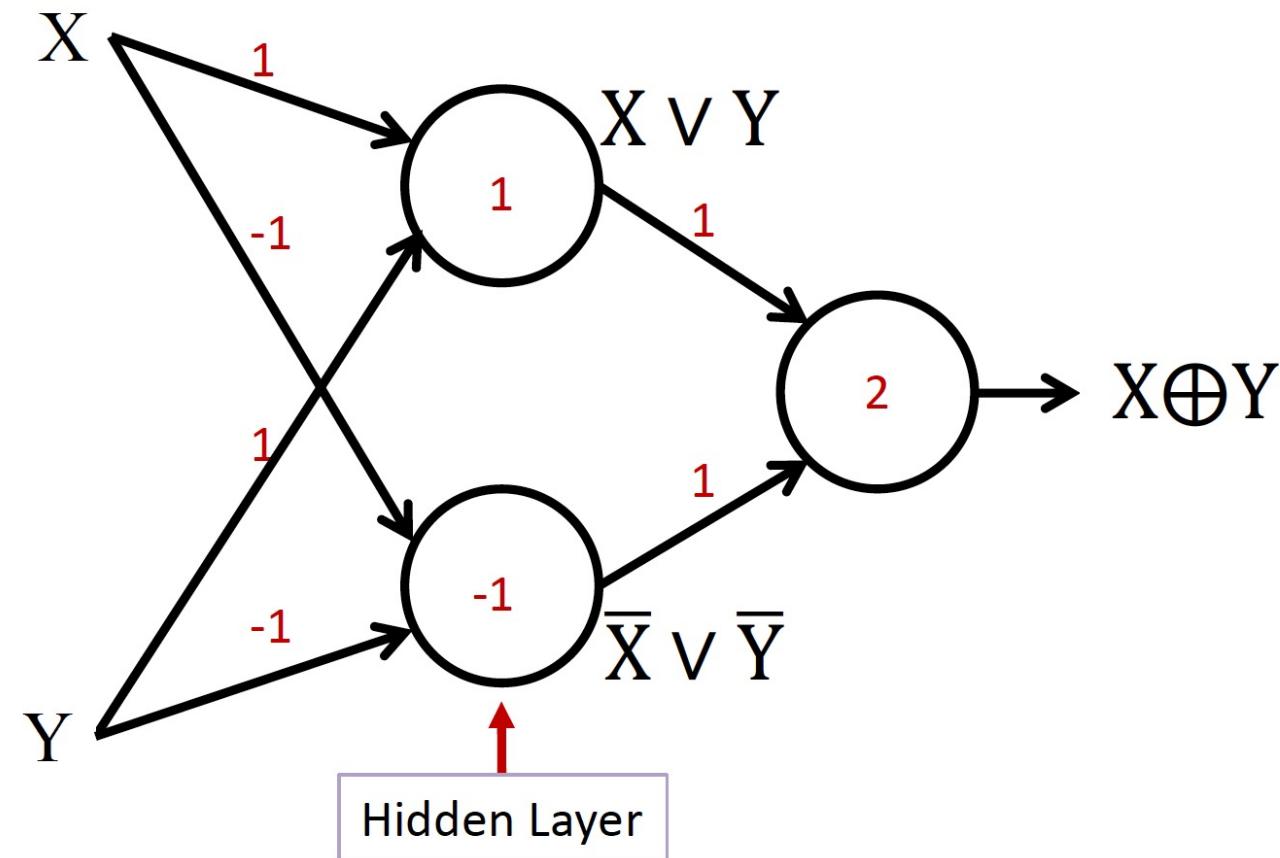
$x_1$	$x_2$	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0



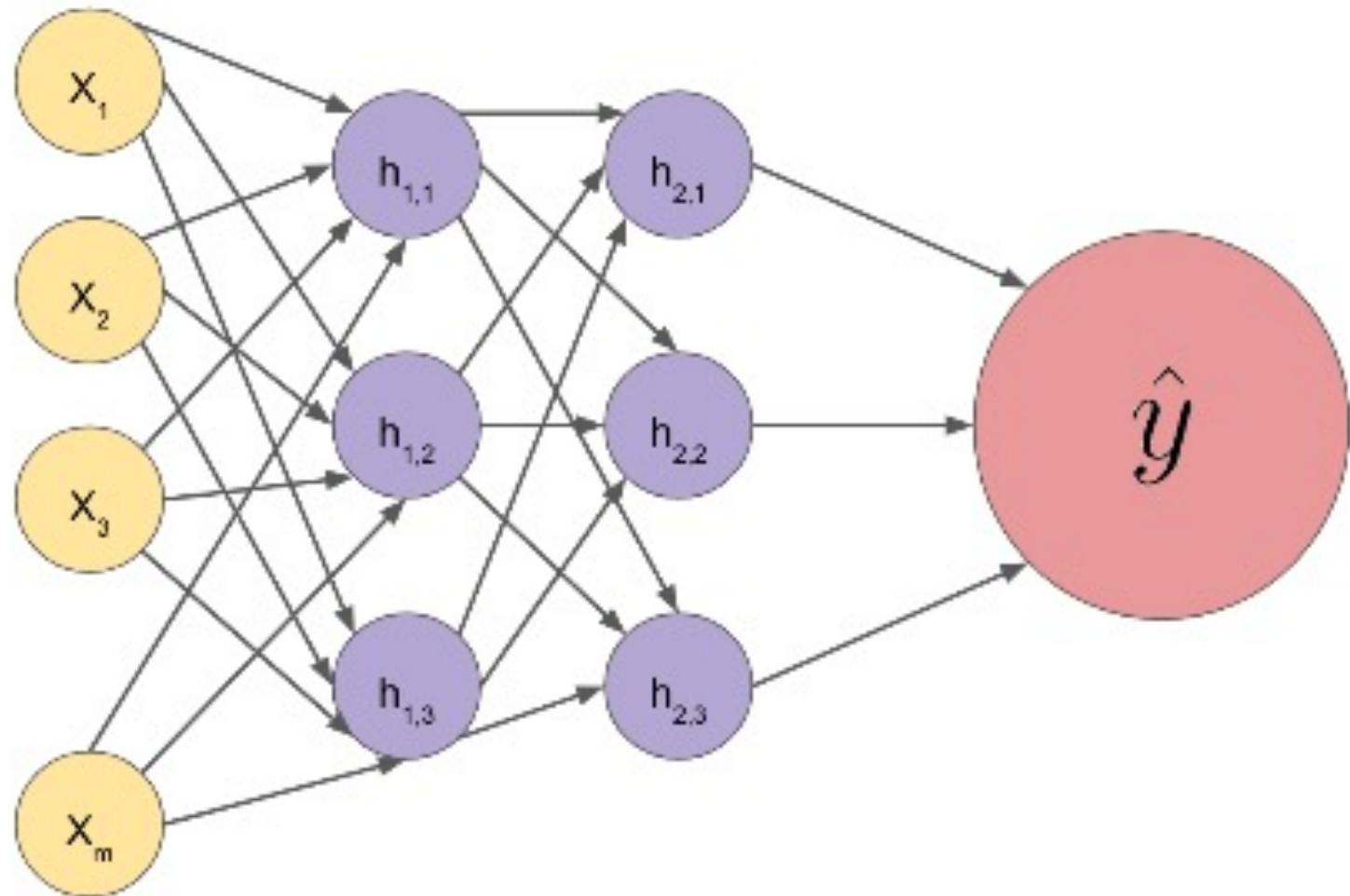
# Multilayer Perceptron (MLP)



# Multilayer Perceptron (MLP)



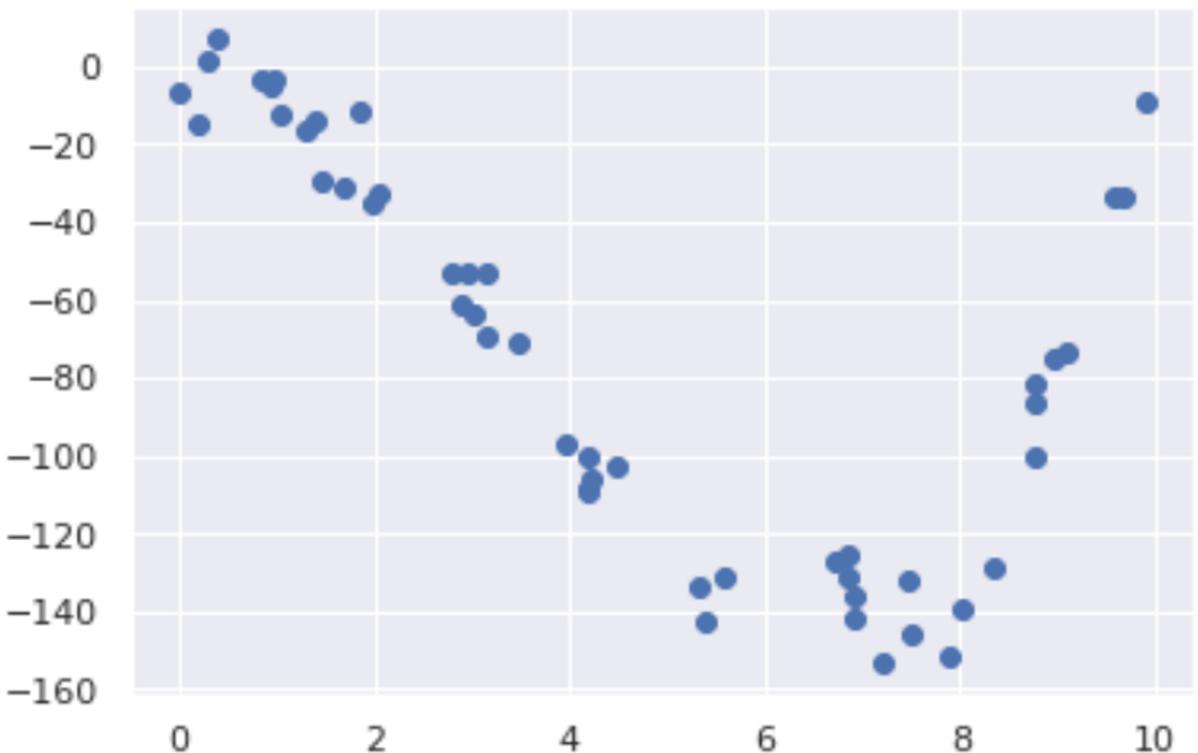
# Neuron with Hidden Layers



# ANN Regression

We will reuse the same data as polynomial example

```
import matplotlib.pyplot as plt  
import numpy as np  
  
rng = np.random.RandomState(1)  
x = 10 * rng.rand(50)  
y = x**3 - 10*x**2 + 10 * rng.randn(50)  
plt.scatter(x, y);
```



# ANN Regression

Create MLP (2 layers + ReLU) with tensorflow to fit the data.

```
import tensorflow as tf
from tensorflow.keras import layers

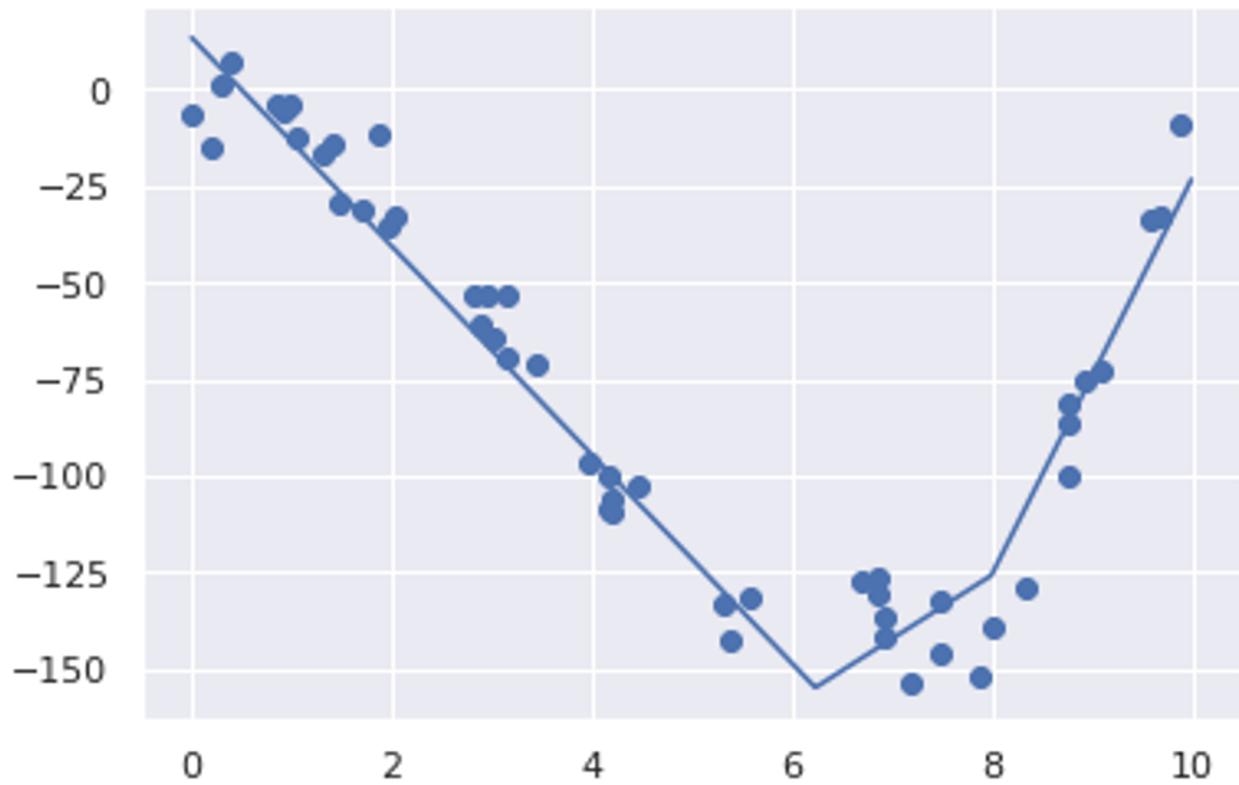
model = tf.keras.Sequential([
    layers.Dense(units=256, activation="relu"),
    layers.Dense(units=1)
])

model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.01),
    loss="mean_squared_error"
)

model.fit(x, y, epochs=3000)

xfit = np.linspace(0,10,1000)
yfit = model.predict(xfit)

plt.scatter(x, y)
plt.plot(xfit, yfit)
```



# ANN Regression

Create MLP (3 layers + ReLU) with tensorflow to fit the data.

```
import tensorflow as tf
from tensorflow.keras import layers

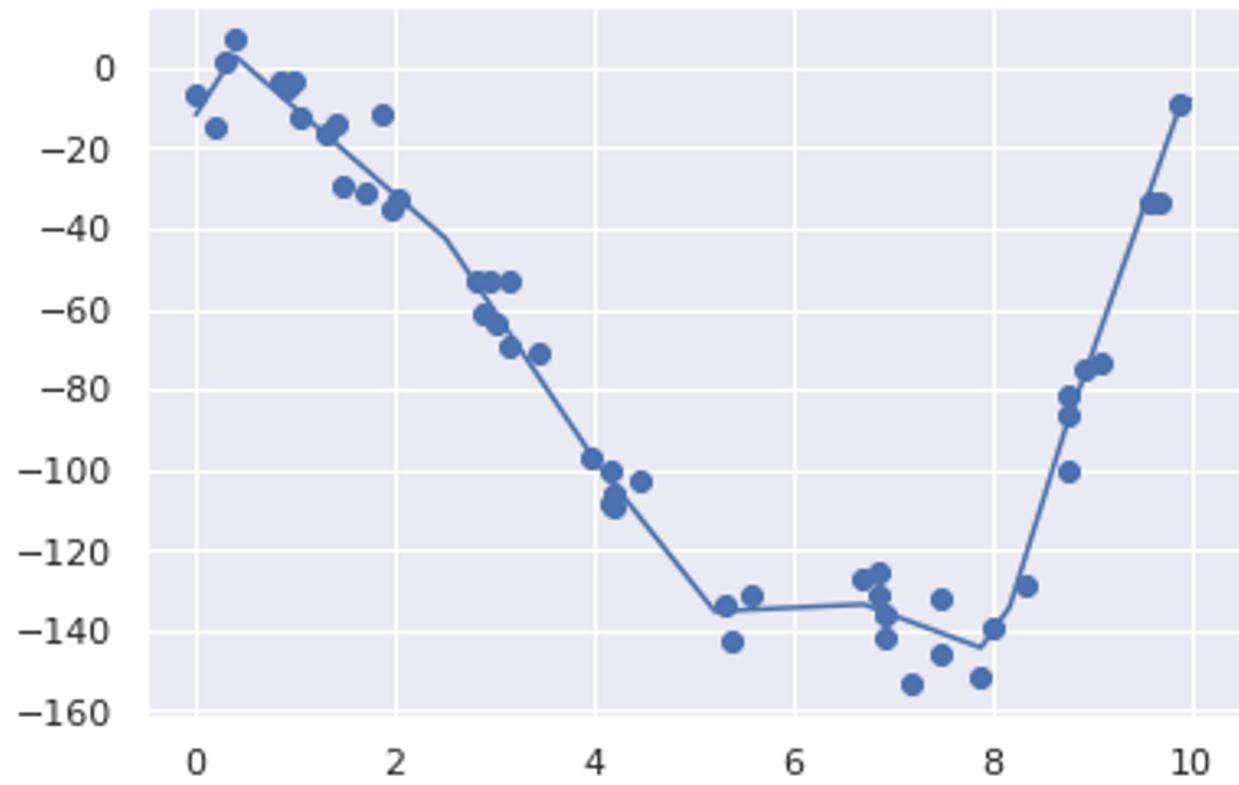
model = tf.keras.Sequential([
    layers.Dense(units=128, activation="relu"),
    layers.Dense(units=128, activation="relu"),
    layers.Dense(units=1)
])

model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.01),
    loss="mean_squared_error"
)

model.fit(x, y, epochs=3000)

xfit = np.linspace(0,10,1000)
yfit = model.predict(xfit)

plt.scatter(x, y)
plt.plot(xfit, yfit)
```



# ANN Regression

Create MLP (3 layers + Tanh) with tensorflow to fit the data.

```
import tensorflow as tf
from tensorflow.keras import layers

model = tf.keras.Sequential([
    layers.Dense(units=128, activation="tanh"),
    layers.Dense(units=128, activation="tanh"),
    layers.Dense(units=1)
])

model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.01),
    loss="mean_squared_error"
)

model.fit(x, y, epochs=3000)

xfit = np.linspace(0,10,1000)
yfit = model.predict(xfit)

plt.scatter(x, y)
plt.plot(xfit, yfit)
```

