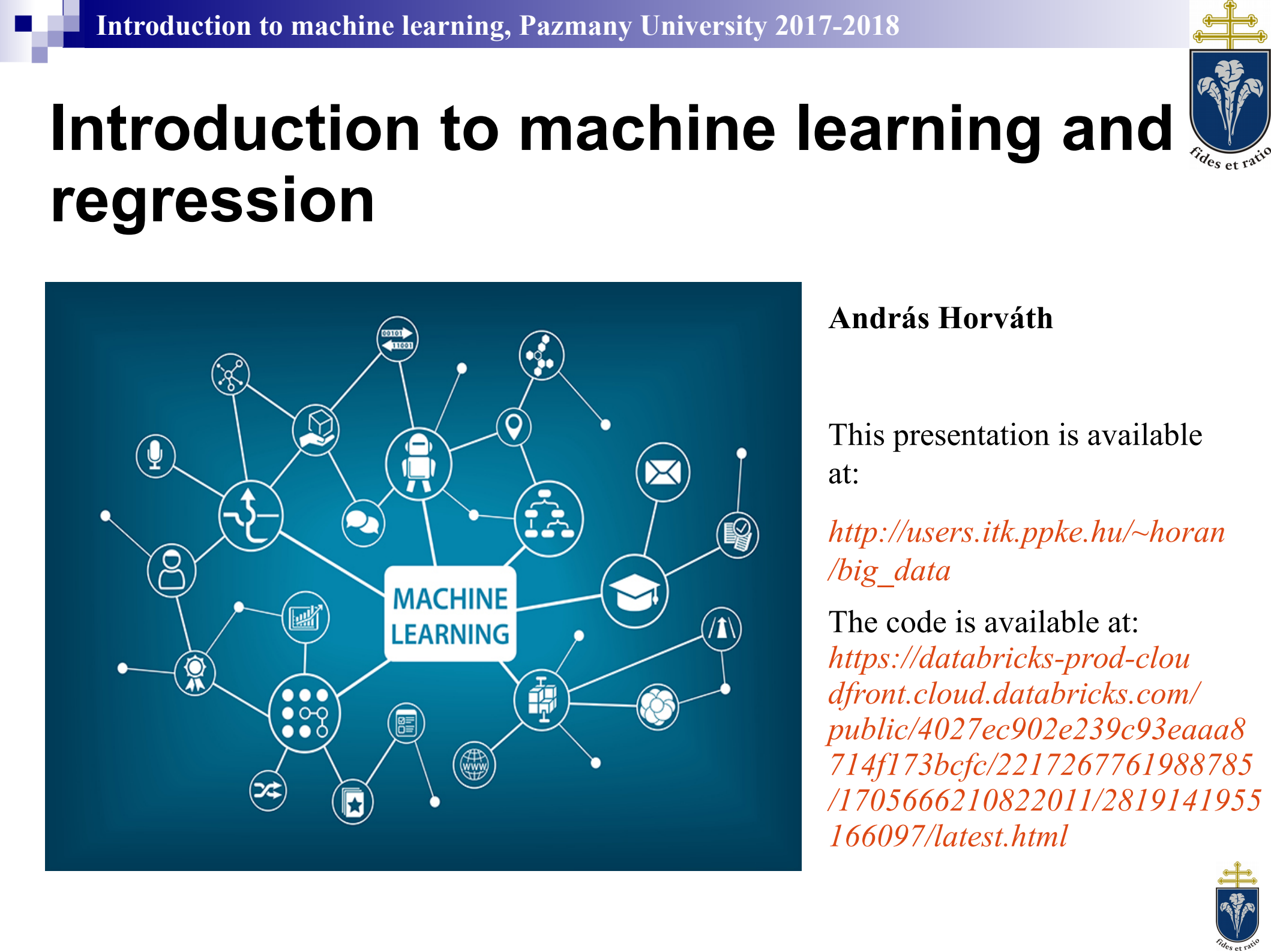


Introduction to machine learning and regression

András Horváth

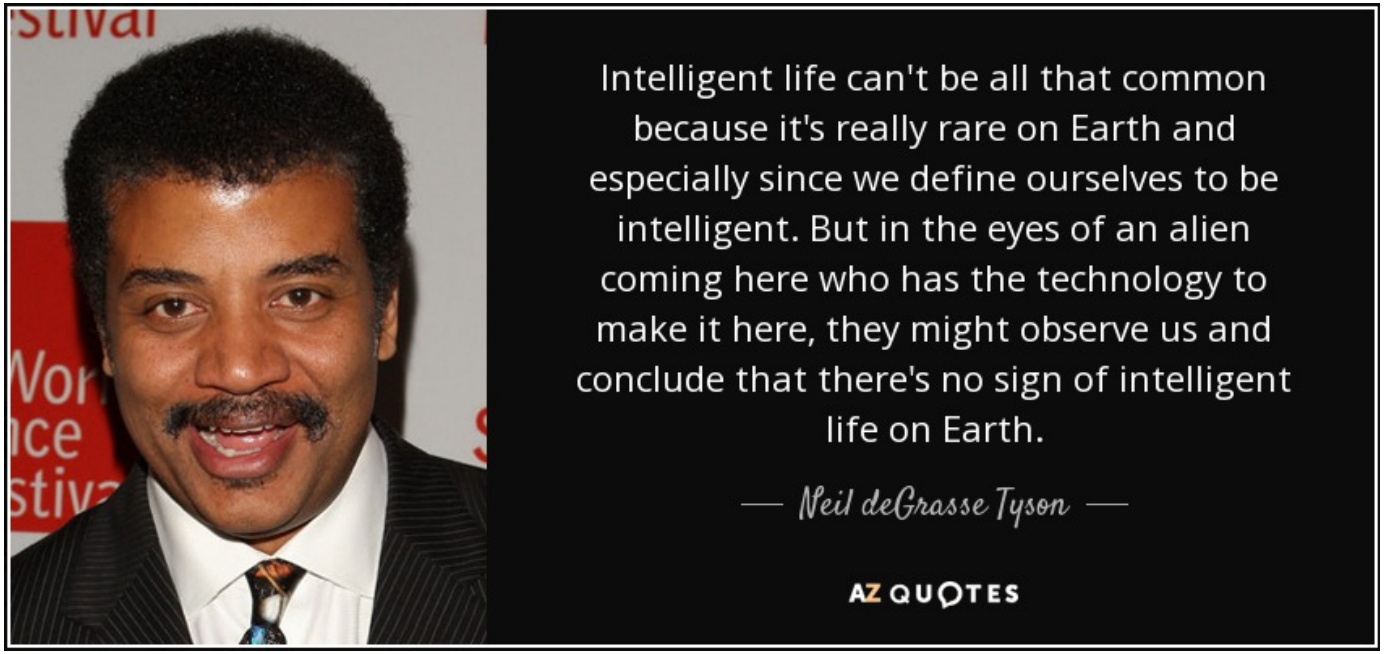
This presentation is available at:

http://users.itk.ppke.hu/~horan/big_data

The code is available at:
https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/2217267761988785/1705666210822011/2819141955166097/latest.html

Machine learning, machine lintelligence

What is intelligence?



Machine learning, machine lintelligence

What is intelligence?

The ability to acquire and apply knowledge and skills.

Machine learning, machine lintelligence

What is intelligence?

The ability to acquire and apply knowledge and skills.

Machine learning, machine lintelligence

What is intelligence?

The ability to acquire and apply knowledge and skills

Intelligence is the ability to adapt to change

Providing computers the ability to learn without being explicitly programmed:

Involves: programming, Computational statistics, mathematical optimization, image processing, natural language processing etc...

Machine learning, machine lintelligence

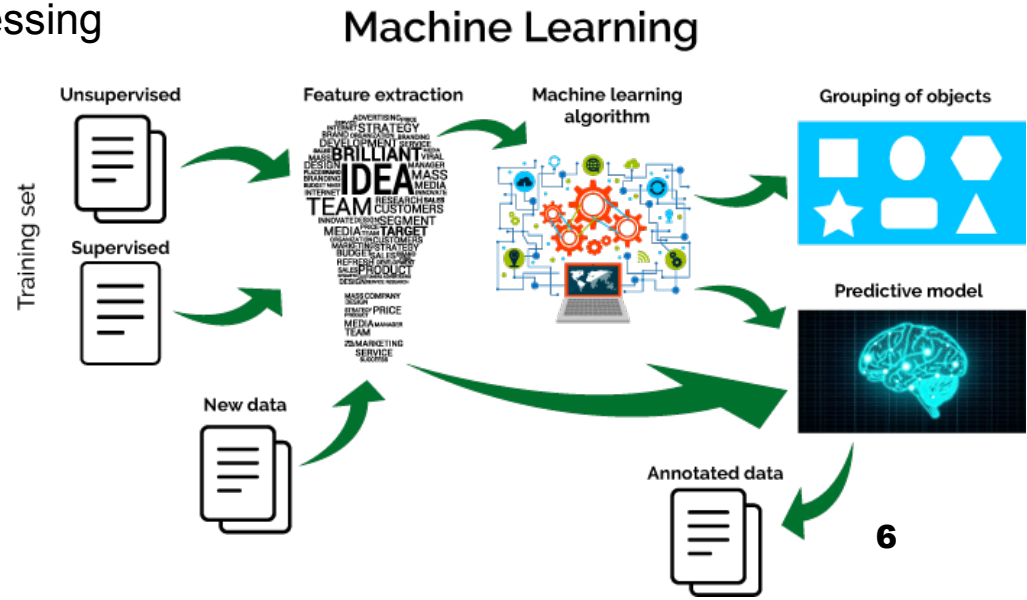
What is intelligence?

The ability to acquire and apply knowledge and skills

Intelligence is the ability to adapt to change

Providing computers the ability to learn without being explicitly programmed:

Involves: programming, Computational statistics, mathematical optimization, image processing, natural language processing etc...



Conquests of machine learning

1952 Arthur Samuel (IBM): First machine learning program playing checkers

Arthur Samuel coined the term „machine learning”

Conquests of machine learning

1952 Arthur Samuel (IBM): First machine learning program playing checkers

1997 IBM Deep Blue Beats Kasparov:

First match (1996 Nov): Kasparov–Deep Blue (4–2)

Second Match (1997 May): Deep Blue–Kasparov (3½–2½)

Conquests of machine learning

1952 Arthur Samuel (IBM): First machine learning program playing checkers

1997 IBM Deep Blue Beats Kasparov:

2011 IBM Watson: Beating human champions in Jeopardy

It's a 4-letter term for a summit; the first 3 letters mean a type of simian : Apex

4-letter word for a vantage point or a belief : View

Music fans wax rhapsodic about this Hungarian's "Transcendental Etudes" : Franz Liszt

While Maltese borrows many words from Italian, it developed from a dialect of this semitic language : Arabic

Conquests of machine learning

1952 Arthur Samuel (IBM): First machine learning program playing checkers

1997 IBM Deep Blue Beats Kasparov:

2011 IBM Watson: Beating human champions in Jeopardy

2014 Deep face algorithm Facebook

Reached 97.35% accuracy

Human performance is around 97%

Conquests of machine learning

1952 Arthur Samuel (IBM): First machine learning program playing checkers

1997 IBM Deep Blue Beats Kasparov:

2011 IBM Watson: Beating human champions in Jeopardy

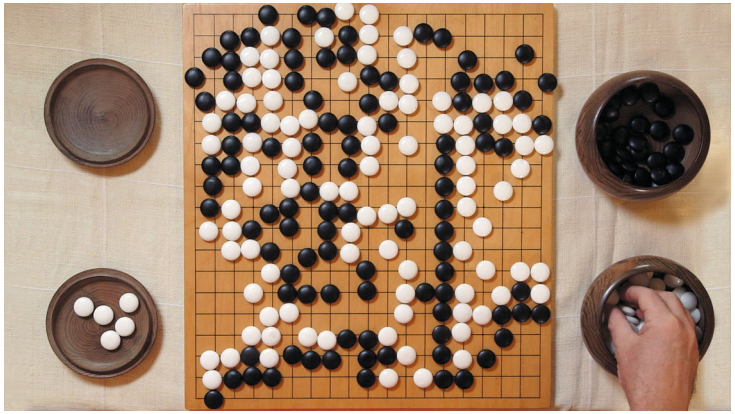
2014 Deep face algorithm Facebook

2016 Alpha go: deep learning

Fan Hui (5-0)

Lee Sedol (4-1)

99.8% win rate against other Go programs



Types of machine learning

Supervised learning and unsupervised learning

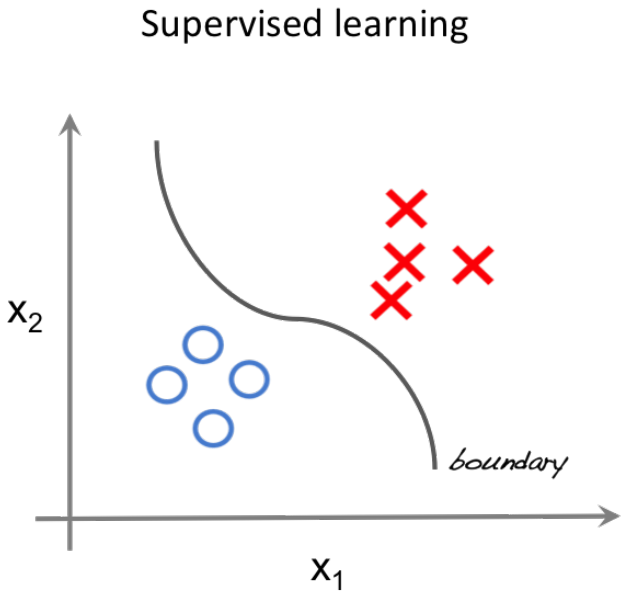
Types of machine learning

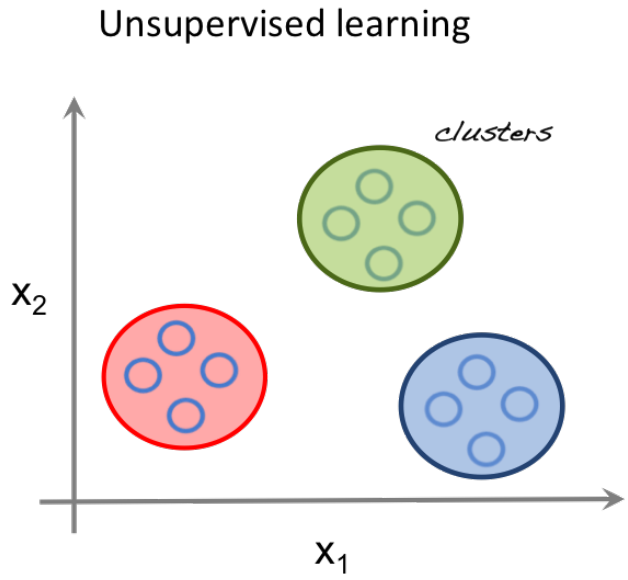
Supervised learning and unsupervised learning

Supervised:

We know the expected output we know the perfect, desired output on the training set

Classification and regression





Unsupervised:

All we have is data and no labels. We have to identify rules in the structure of the data

Clustering and association

Types of machine learning

Supervised learning and unsupervised learning

Supervised:

Classification and regression

Classification: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.

Regression: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

Unsupervised:

Clustering and association

Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.

Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Semi-supervised: Some data is labeled but most of it is unlabeled and a mixture of supervised and unsupervised techniques can be used.

Supervised Classification

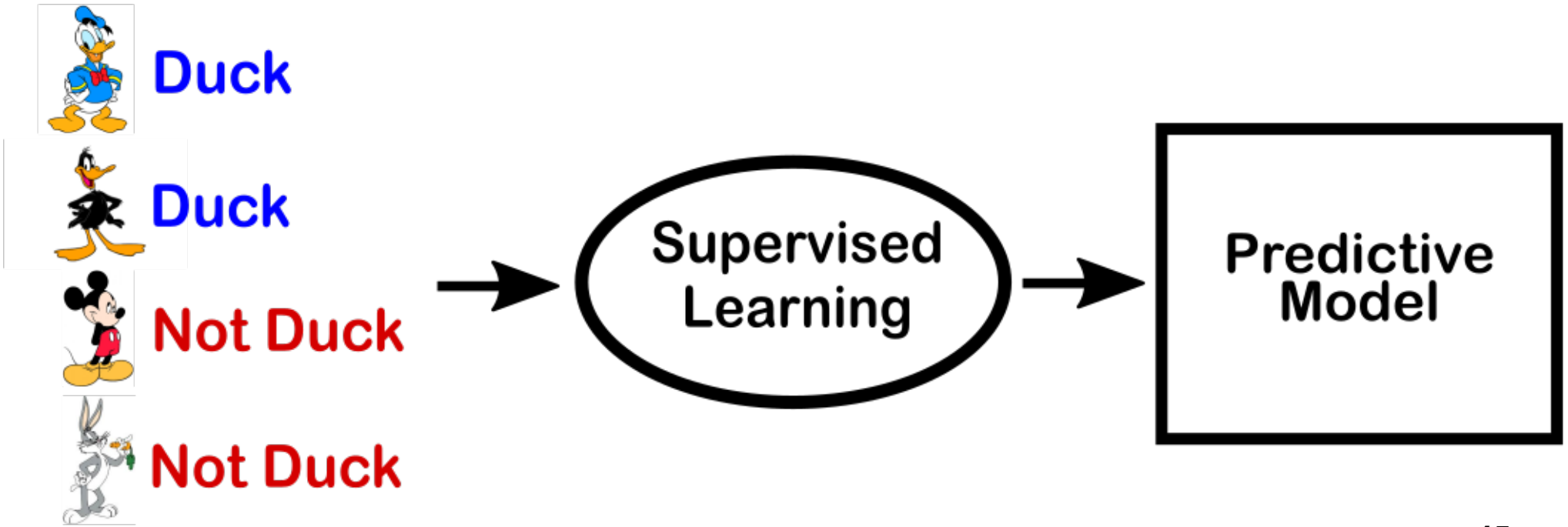
The most important conquests of deep learning comes from supervised calssifcation.

There are three major improvements in this:

- New methods and technologies

- Powerful hardware for training (GPGPUs)

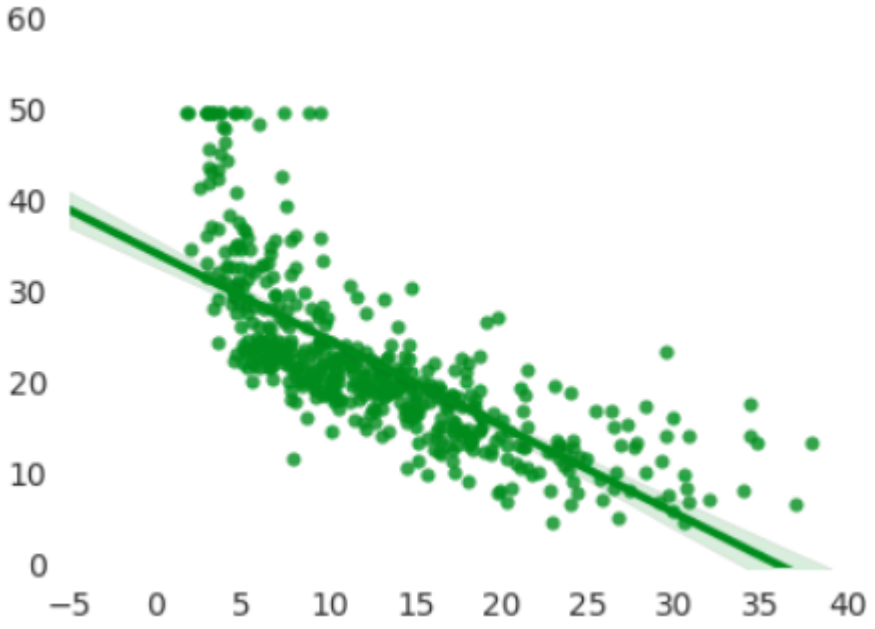
- Vast amount of data is available



Supervised Regression

Our aim is to predict a target value (Y) based on our data (X)

We have to find a model that explains the rules how Y can be derived from X m(X)=Y



The prefect fit is usually not possible, because our world is not perfect:

We have a model which has flaws

There is noise in our observation

We have to fit a model which minimizes the error on the dataset:

L1 error $$\sum_i |m(X_i) - Y_i|$$

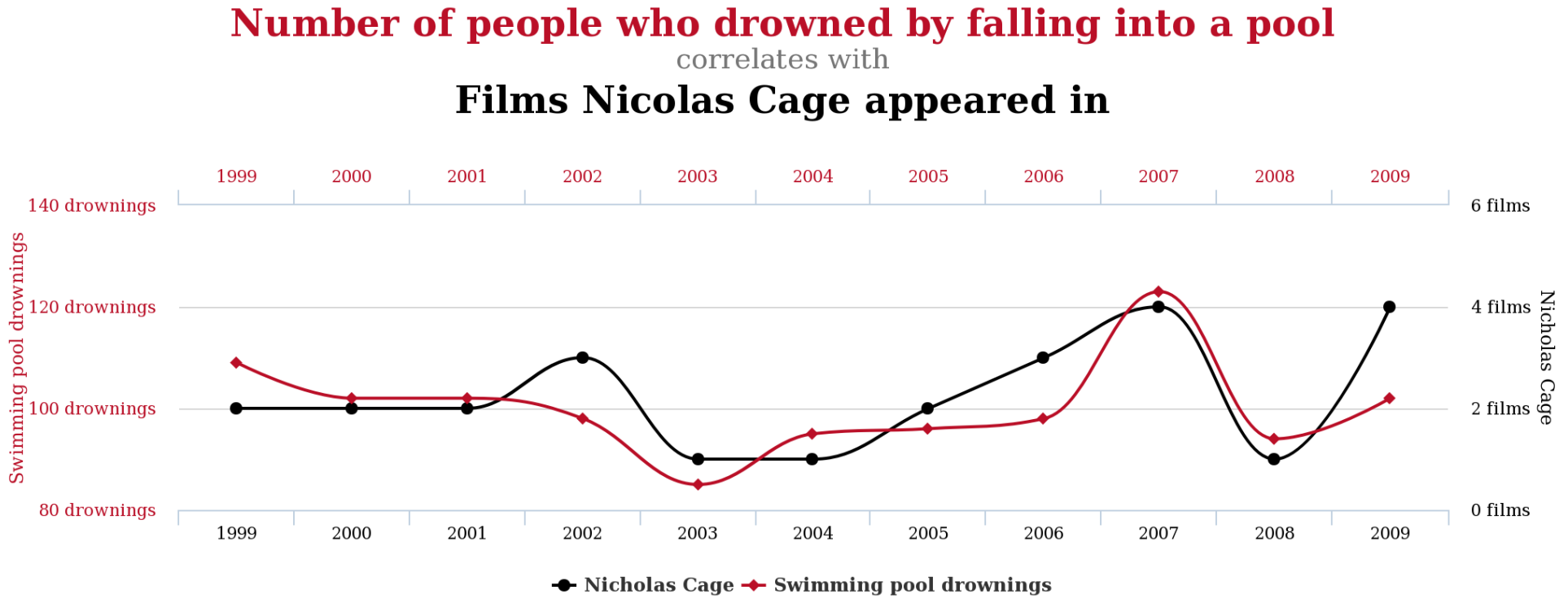
L2 error $$\sum_i (m(X_i) - Y_i)^2$$

Correlation and cause effect relationship

Large correlation between the data does not necessary mean that one is caused by the other

One has to gather the large amount of data to ensure the coincidence by luck

One has to understand the data

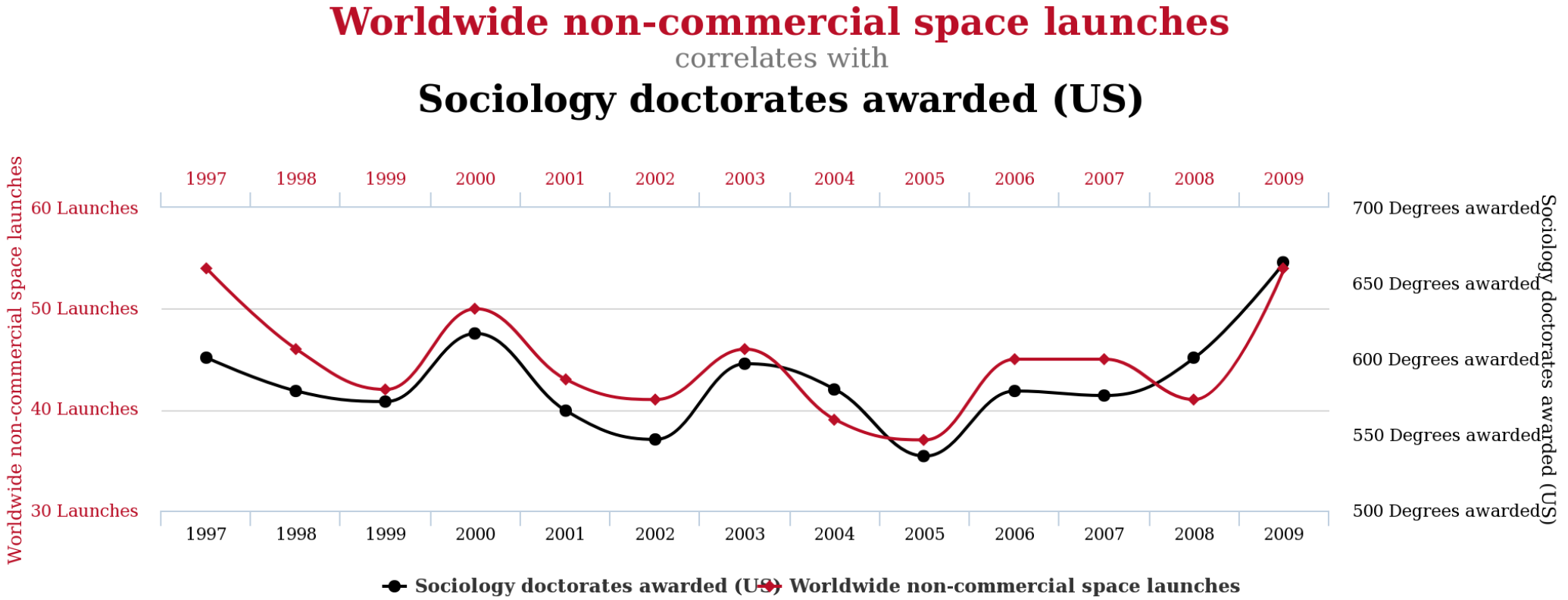


Correlation and cause effect relationship

Large correlation between the data does not necessary mean that one is caused by the other

One has to gather the large amount of data to ensure the coincidence by luck

One has to understand the data



Steps of machine learning

Supervised learning and unsupervised learning:

1, Gather Data

2, Understand your data

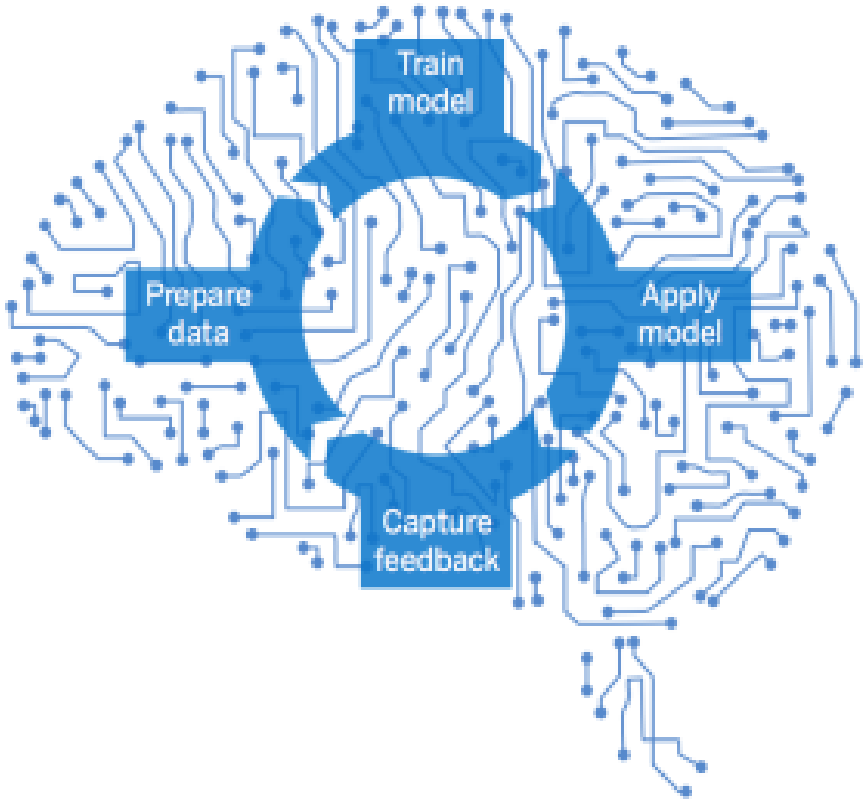
3, Data preparation

4, Choosing a model

5, Training and evaluation

6, Parameter tuning

7, Go back to point 1



Gathreing data

Lets have an example dataset:

Boston house prices dataset:

A typical regression problem, good for trying out different machine learning models

There are many available solutions on this dataset

Standard dataset for model evaluation contained by Python libraries

Easy to load the data:

```
#gather data
# we will import modules which already contain the dataset
import sklearn
from sklearn.datasets import load_boston
#load the database
boston = load_boston()
```

Understanding the data

```
#understand the data
print boston.DESCR #contains the description of the dataset
print boston.data.shape # the shape of the dataset (rows and columns)
print boston.keys()
X, Y = boston.data, boston.target
import pandas as pd
boston_df = pd.DataFrame(boston.data)
boston_df.columns = boston.feature_names
boston_df.describe()
```

The dataset contains 506 rows (different data points) and 14 columns

13 features describing the properties (X):

per capita crime rate by town, proportion of residential land zoned for lots over 25,000 sq.ft., proportion of non-retail business acres per town, Charles River dummy variable (= 1 if tract bounds river; 0 otherwise), nitric oxides concentration (parts per 10 million), average number of rooms per dwelling, proportion of owner-occupied units built prior to 1940, weighted distances to five Boston employment centers, index of accessibility to radial highways, full-value property-tax rate per $10,000, pupil-teacher ratio by town, 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town, % lower status of the population

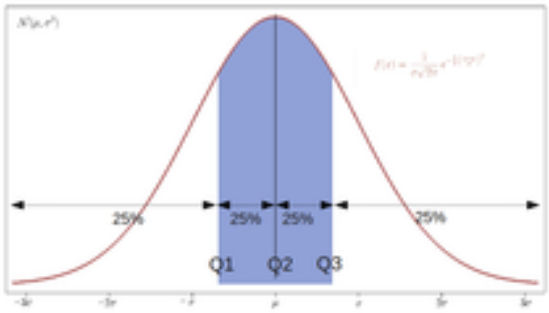
14th column: the target value (Y)

Our aim is to predict Y from X

Data preparation

Write out some values:

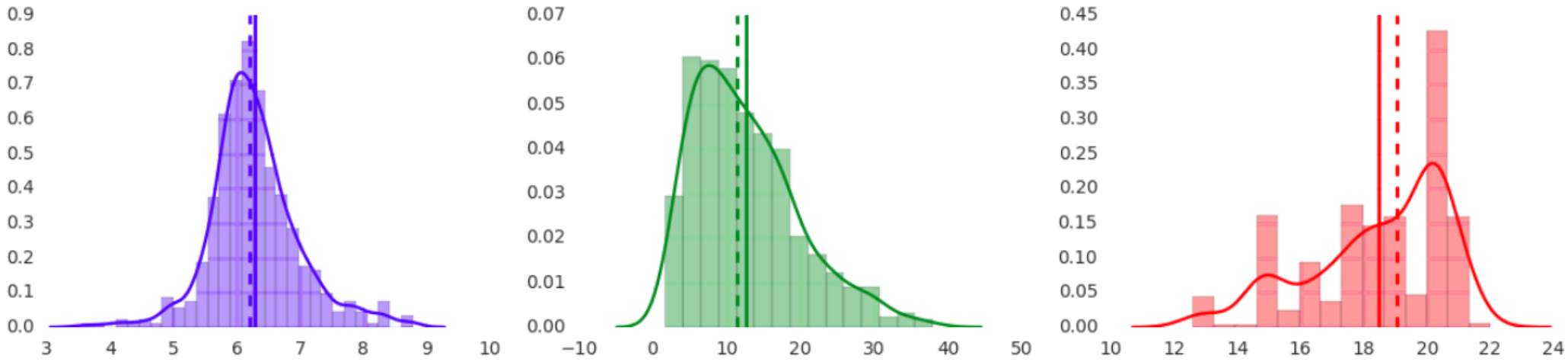
```
#check values
boston_df.min() #minimum values of the features
boston_df.max() #maximum values of the features
boston_df.median()#median values of the features
boston_df.quantile(0.5)  #50 percent quantile of the features
```



Data preparation

Plot some values as histograms:

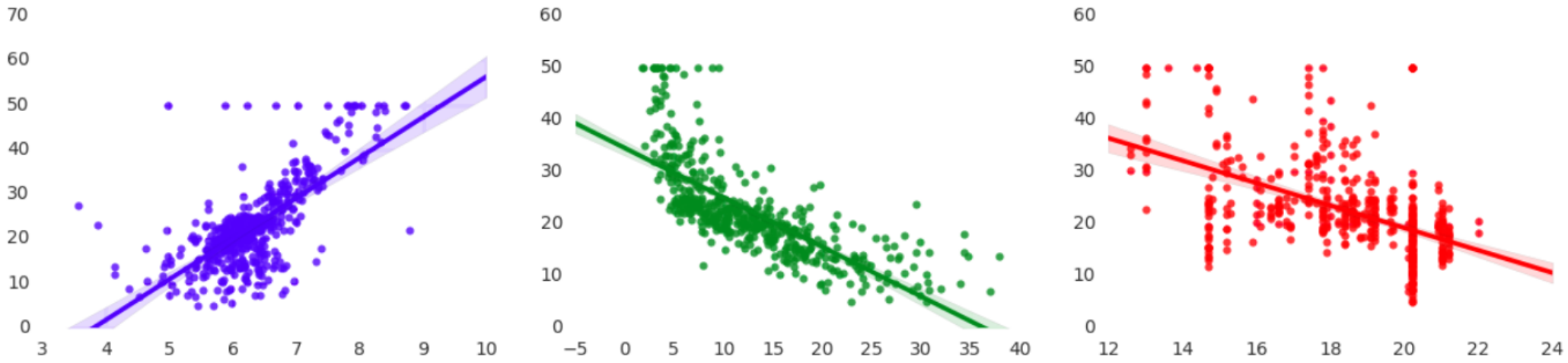
```
1  import matplotlib.pyplot as plt
2  import seaborn as sns
3  clr = ['blue', 'green', 'red']
4  fig,  ax = plt.subplots(ncols=3,figsize=(15,3))
5
6  for i, var in enumerate(['RM', 'LSTAT', 'PTRATIO']):
7      sns.distplot(boston_df[var],  color = clr[i], ax=ax[i])
8      ax[i].axvline(boston_df[var].mean(), color=clr[i], linestyle='solid', linewidth=2)
9      ax[i].axvline(boston_df[var].median(), color=clr[i], linestyle='dashed', linewidth=2)
10
11 display(fig)
```



Data preparation

Plot some values related to the average price:

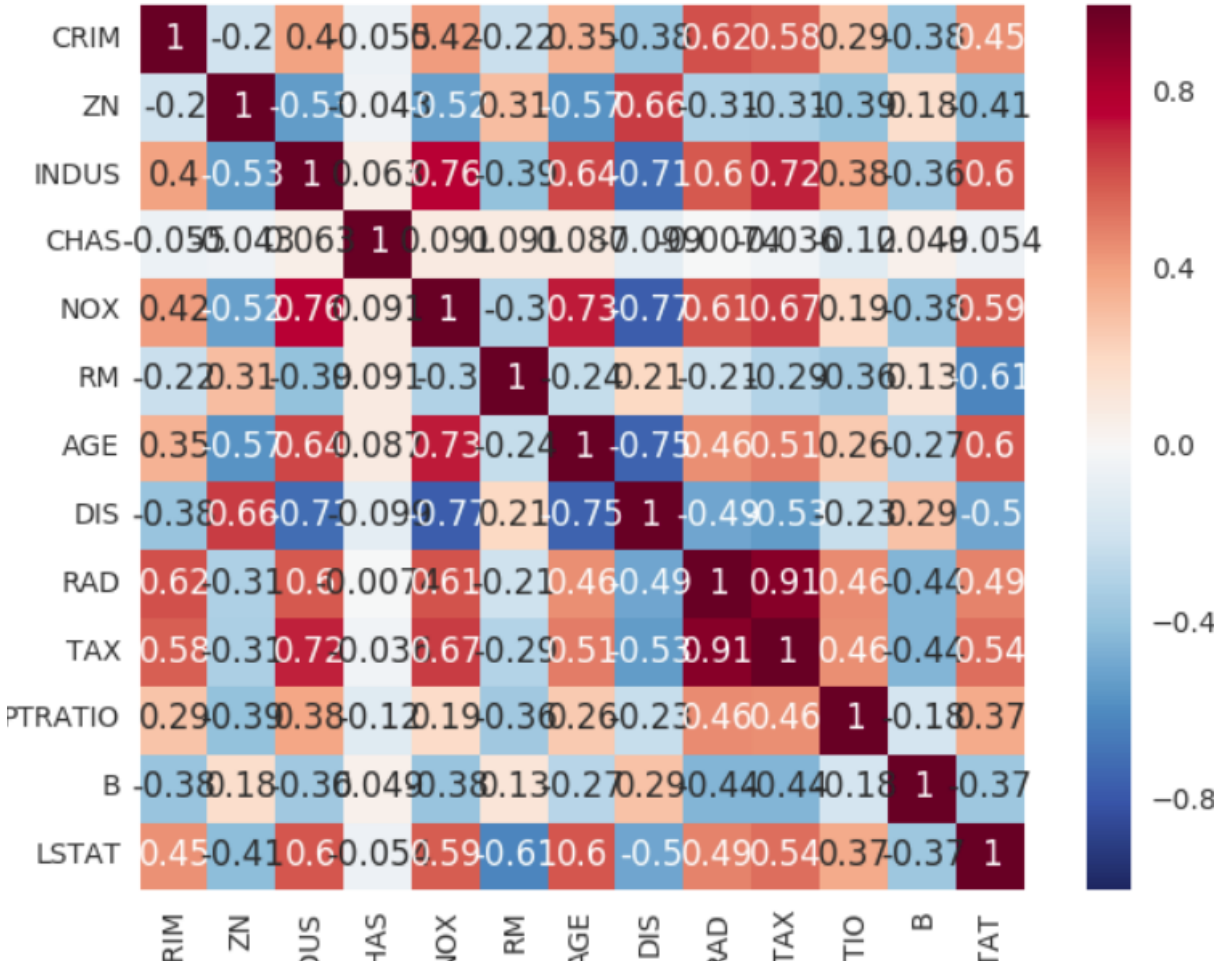
```
fig, ax = plt.subplots(ncols=3,figsize=(15,3))

for i, var in enumerate(['RM', 'LSTAT', 'PTRATIO']):
    sns.regplot(boston_df[var], boston.target, color=clr[i], ax=ax[i])
    ax[i].set(ylim=(0, None))
display(fig)
```



Data preparation

Plot correlation between the different features:

```
1  fig,ax= plt.subplots()
2  sns.heatmap(boston_df.corr(), square=True,annot=True)
3  display(fig)
```



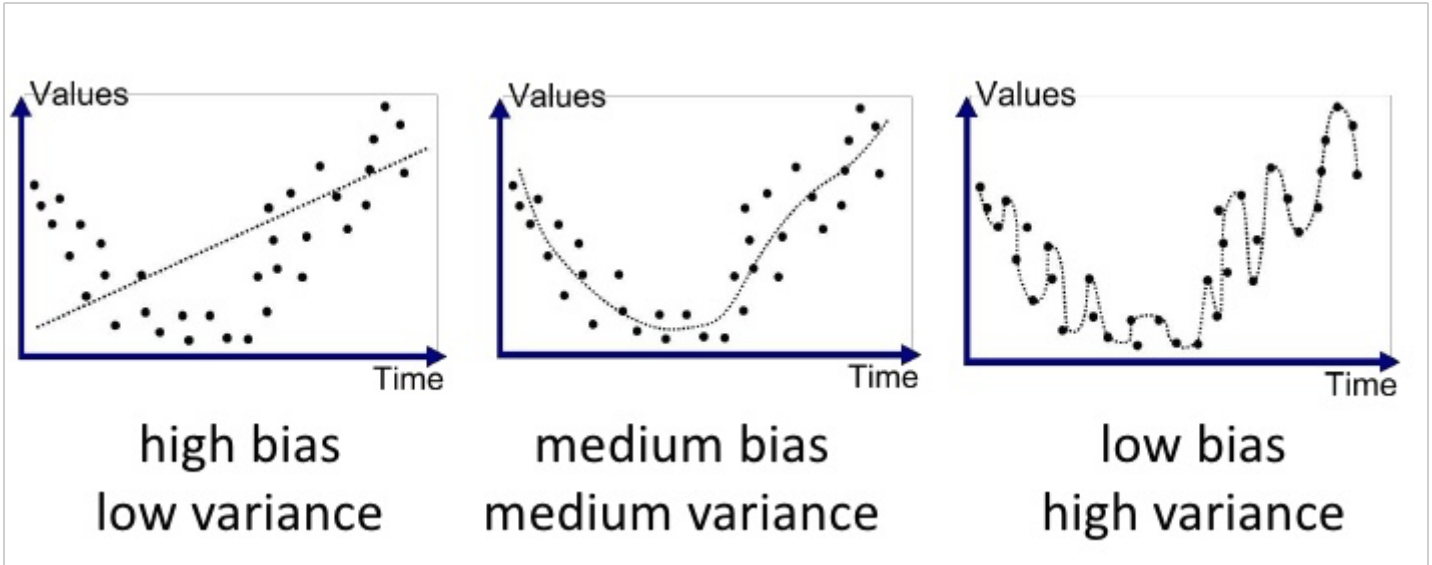
Evaluation of our method: the bias variance problem

We have to simultanously minimize two different errors

The bias is error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting)

The variance is error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting).

If we fit our data perfectly on the training set, it might not be general. Our aim is to have a general model



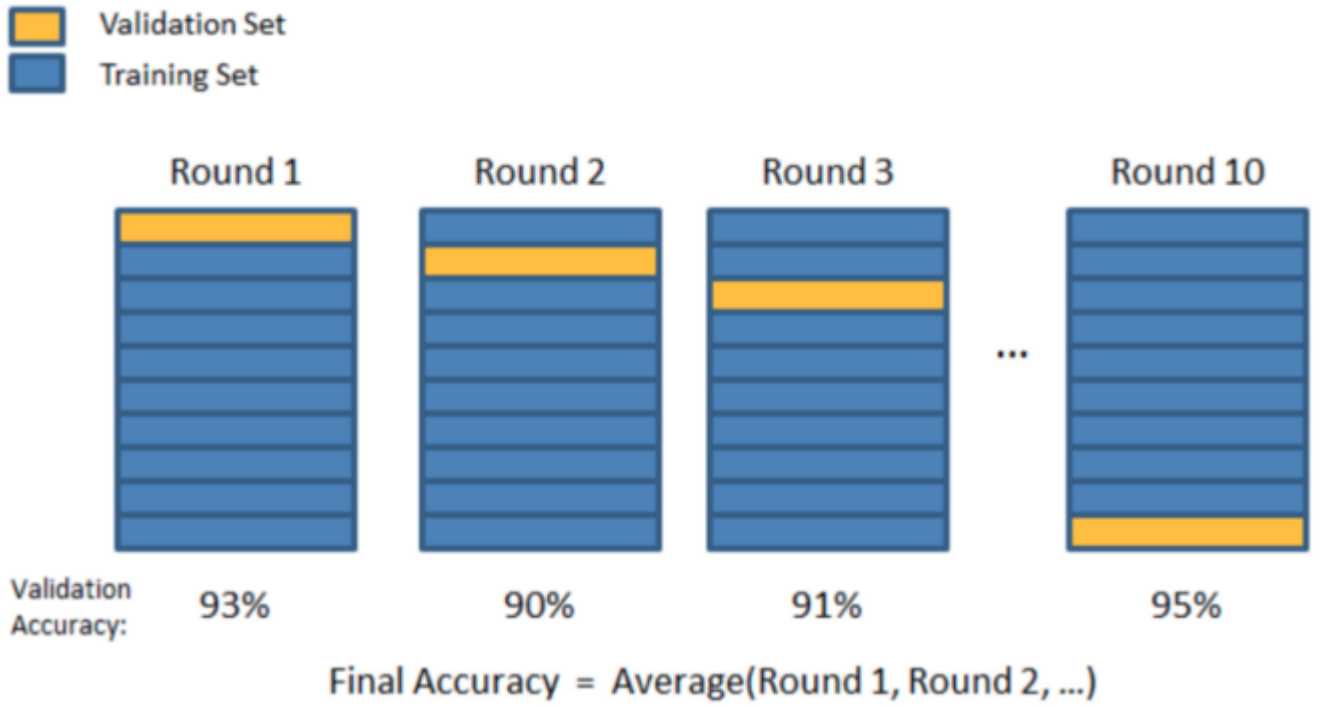
Evaluation of our method: the bias variance problem

We have to simultanously minimize two different errors

The bias is error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting)

The variance is error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting).

If we fit our data perfectly on the training set, it might not be general. Our aim is to have a general model



Evaluation of our method: the bias variance problem

We have to simultanously minimize two different errors

The bias is error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting)

The variance is error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting).

If we fit our data perfectly on the training set, it might not be general. Our aim is to have a general model

Let's reserve some of our data to see how general our model is:

```
#create train and test data
from sklearn.cross_validation import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

Once we have a good model we should re-run the training and evaluation on differently divided dataset and the accuracy should be consistent. This is called cross-validation

Scaling feautres

The features in our data can have orders of different magnitudes

Some algorithms can handle this fact, some can not

Even those which can learn these differences work faster and better with scaled data

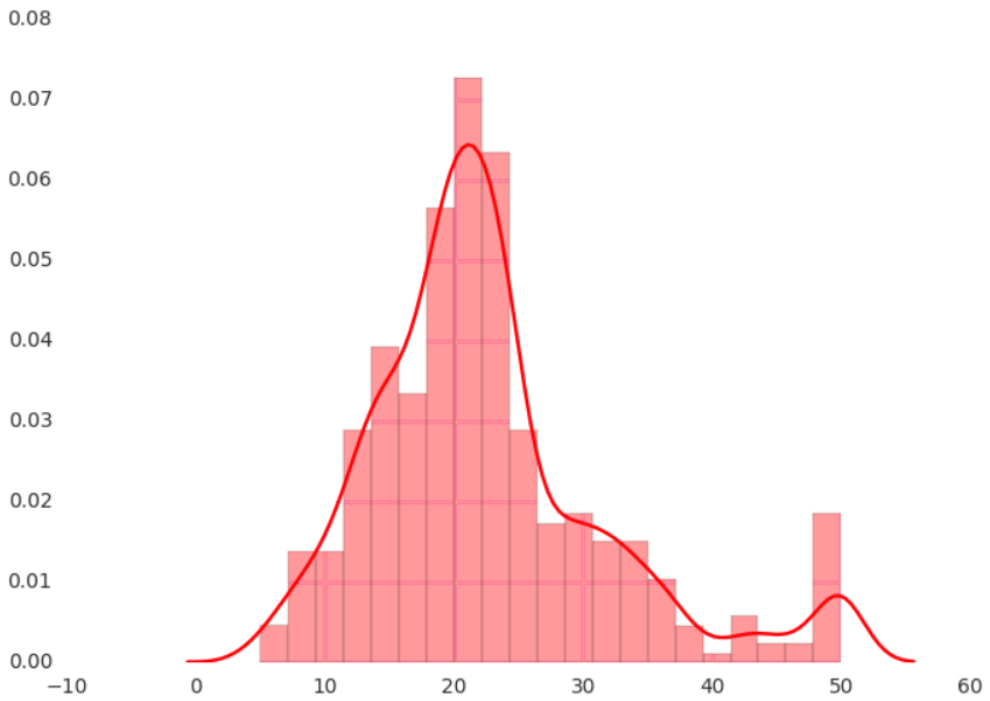
Let's change all features to ensure that all of them has zero mean and variance of 1

```
import numpy as np
#scaling features
X_train=X_train-np.mean(X_train,0)
X_train=X_train/np.var(X_train,0)
X_test=X_test-np.mean(X_train,0)
X_test=X_test/np.var(X_train,0)
#from sklearn import preprocessing
#scaler = preprocessing.StandardScaler( )
#X_train = scaler.fit_transform(X_train)
```

Detecting outliers

It really helps to have a look at the data:

```
print(X_test.shape)
fig,ax= plt.subplots()
sns.distplot(Y_train,  color = 'red')
display(fig)
```



One could see that prices are limited at 50K$

These elements will not fit into our model

They are outliers and it is not good to test the model on them in real life

The values are also thresholded in the test data in this case, so it is good if the model could learn this fact

Linear regression

We assume that there is a correlation in our data (x and y) which can be described as: Y=wX+b

We want to minimize the error between:

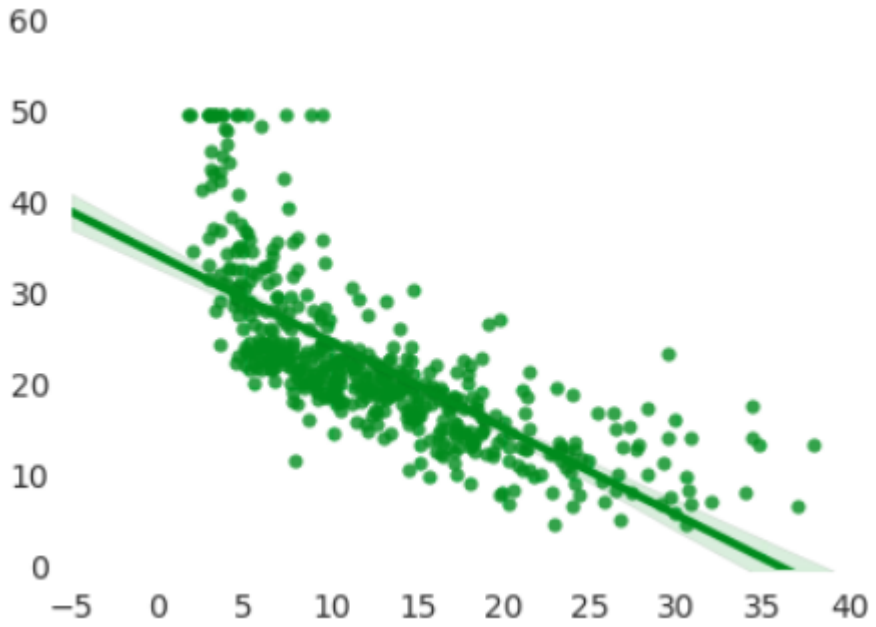
$$\sum_i |(wX_i + b) - Y_i|$$

According to w and b

There is an analytical formula to find the best w and b

We have to find the local extremum of the error function

$$\frac{\partial}{\partial w} \sum_i |(wX_i + b) - Y_i| = 0$$



$$\frac{\partial}{\partial b} \sum_i |(wX_i + b) - Y_i| = 0$$

Linear regression

We assume that there is a correlation in our data (x and y) which can be described as: Y=wX+b

Linear regression in SK learn

```
#use linear regression- we will fit the following model:
# y = wx + b
from sklearn.linear_model import LinearRegression

lm = LinearRegression()
lm.fit(X_train, Y_train)

Y_pred = lm.predict(X_test)

mse = sklearn.metrics.mean_squared_error(Y_test, Y_pred)
fig,ax= plt.subplots()

plt.scatter(Y_test, Y_pred)
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Prices vs Predicted prices, error: "+str(mse))
display(fig)
```

Linear regression

We assume that there is a correlation in our data (x and y) which can be described as: Y=wX+b

Linear regression in SK learn

```
#crossvalidation with simple linear regression
from sklearn.linear_model import LinearRegression
from sklearn. cross_validation import cross_val_score
linear = LinearRegression()
linear_scores = cross_val_score(linear, boston.data, boston.target, cv=5, scoring='neg_mean_squared_error')
print(linear_scores)
print(linear_scores.mean())
```

Linear regression - Lasso

We assume that there is a correlation in our data (x and y) which can be described as:
Y=wX+b

One can easily overfit the data:
There is noise on our input data (some variables might be more reliable than the other)

How could we select the important variables.
Lasso (Least Absolute Shrinkage and Selection Operator) penalizes large values in the regressor (the same output accuracy where the values in w are more uniform is better)

The parameter of the algorithm is the penalty constant

```
#cross validation with lasso regression
from sklearn.linear_model import Lasso
lasso = Lasso(0.3) #score 0.1 to 0.5
lasso_scores = cross_val_score(lasso, boston.data, boston.target, cv=5, scoring='neg_mean_squared_error')
print(lasso_scores.mean())
print(lasso_scores)
```

Linear regression – Ridge, Elastic Net

We assume that there is a correlation in our data (x and y) which can be described as:
Y=wX+b

One can easily overfit the data:
There is noise on our input data (some variables might be more reliable than the other)

Ridge and ElasticNet uses more complex penalization on the parameters depending on the data (Thikonov regularization)

```
#cross validation with Ridge regression
from sklearn.linear_model import Ridge
ridge = Ridge(1) #score 1 to 5
ridge_scores =cross_val_score(ridge, boston.data, boston.target, cv = 5, scoring='neg_mean_squared_error')
print(ridge_scores.mean())
print(ridge_scores)
```

```
#cross validation using ElasticNet
from sklearn.linear_model import ElasticNet
elastic_net = ElasticNet(1)  #score 1 to 5
elastic_net_scores =cross_val_score(elastic_net, boston.data, boston.target, cv = 5, scoring='neg_mean_absolute_error' ) #'neg_mean_absolute_error'
print(elastic_net_scores.mean())
print(elastic_net_scores)
```

Non-linear regression

How could we fit a more complex mode?
We could fit any model we found according to the previous formula:
Y=m(x)

Where we can derivate the model according to all the parameters and we will have an equation for every parameter

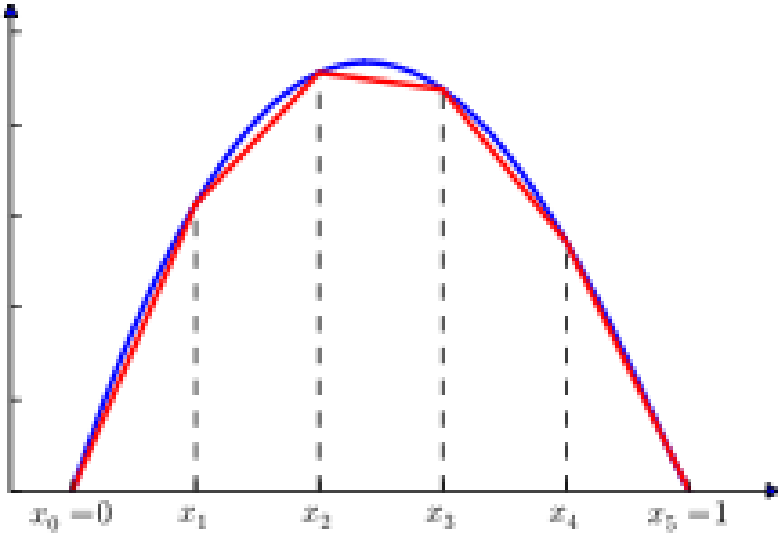
$$\frac{\partial}{\partial m_p} \sum_i |m(X_i) - Y_i| = 0$$

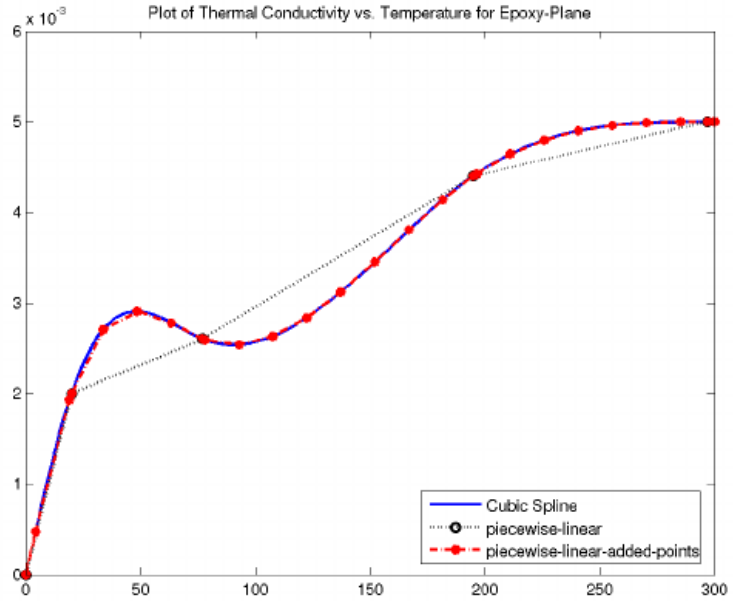
We could fit any model if it is known, but unfortunately in practice the problem is that we do not know the model

How could we create a general model, which could approximate any possible models

Non-linear regression

We can approximate any function with partially linear functions

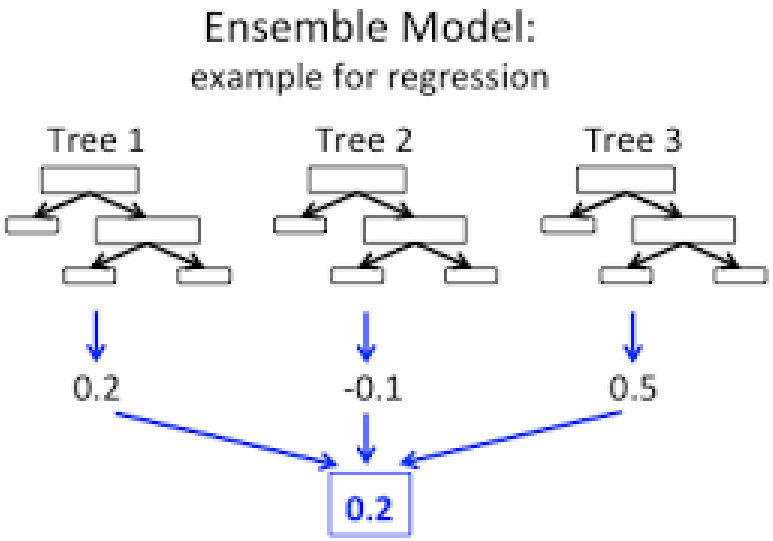


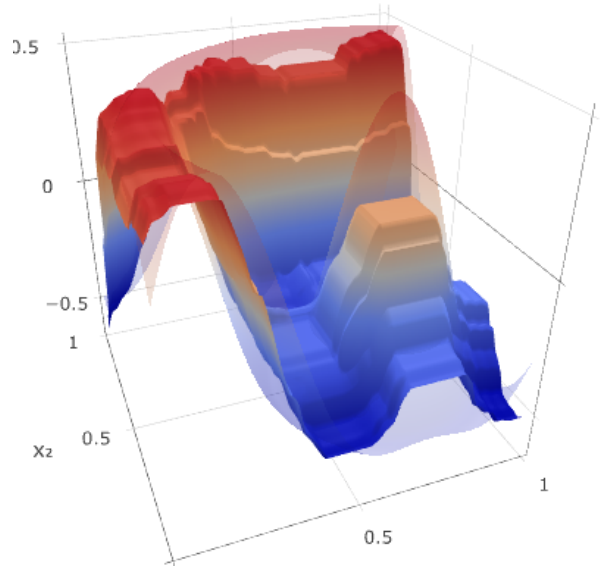


We can divide the problem into sub-domains depending on the input values and use separated (or combined) linear regressors to approximate the original function.

Ensemble regression

We can approximate any function with partially linear functions





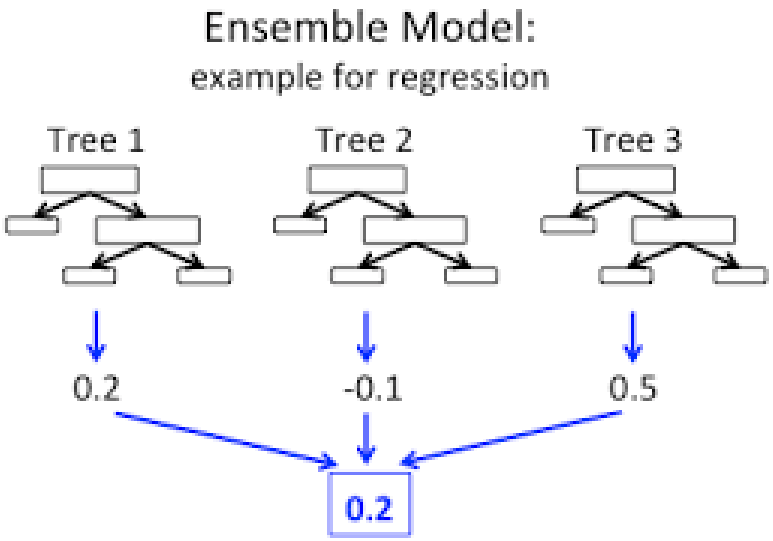
The approximation will be the weighted summation of linear models

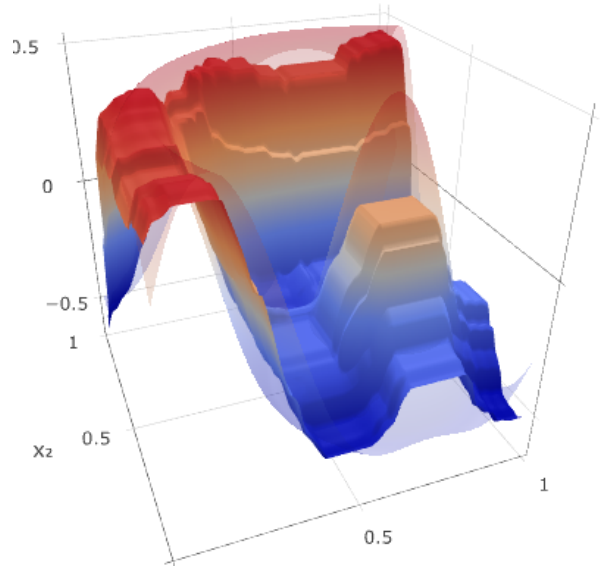
The weights are determined by the values of the data

http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html

Ensemble regression

We can approximate any function with partially linear functions





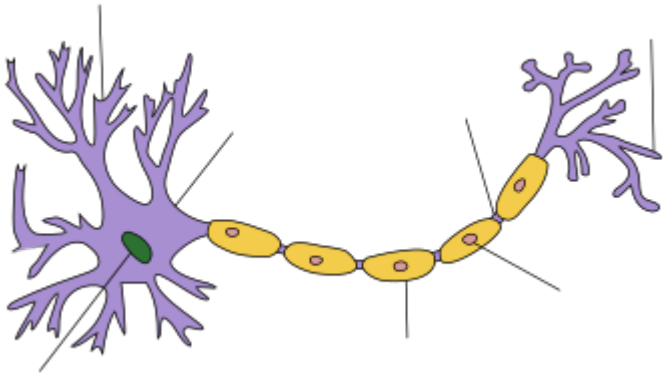

```
from sklearn import ensemble
ensemb = ensemble.GradientBoostingRegressor(
    n_estimators=100, #number of independent estimators
    learning_rate=0.1,
    max_depth=4,   #depth of the decision tree
    min_samples_leaf=5, #minimum number of samples required to be a leaf node
    max_features=0.8,  #maximum nubmers of features considered at each split
    loss= 'lad', #least absolute deviation
    random_state=700
)
ensemble_scores =cross_val_score(ensemb, boston.data, boston.target, cv = 5, scoring='neg_mean_absolute_error' ) #'neg_mean_absolute_error'
print(ensemble_scores.mean())
print(ensemble_scores)
```

Neural networks

Linear regression:

C1(wx+b)+C2(wx+b)+C3(wx+b)+C4(wx+b)+C5(wx+b)+C5(wx+b)

What does a neuron do?

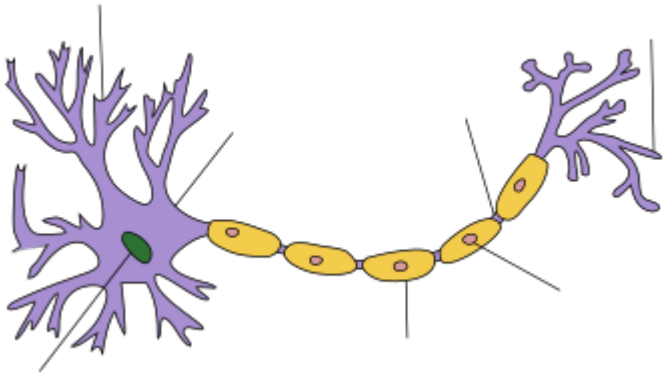


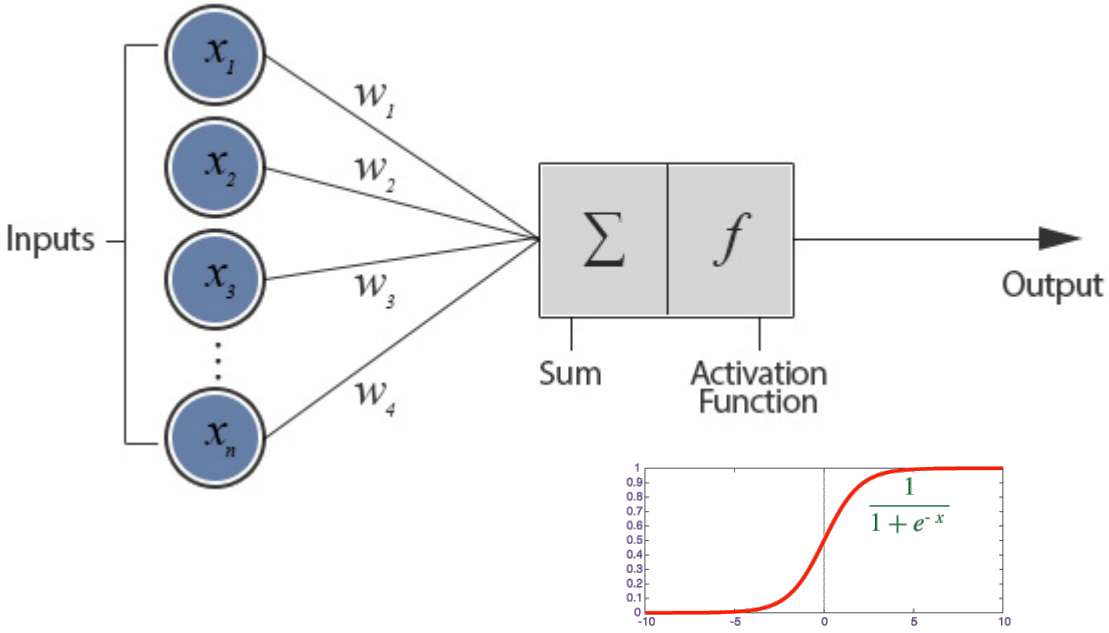
Neural networks

Linear regression:

C1(wx+b)+C2(wx+b)+C3(wx+b)+C4(wx+b)+C5(wx+b)+C5(wx+b)

What does a neuron do?





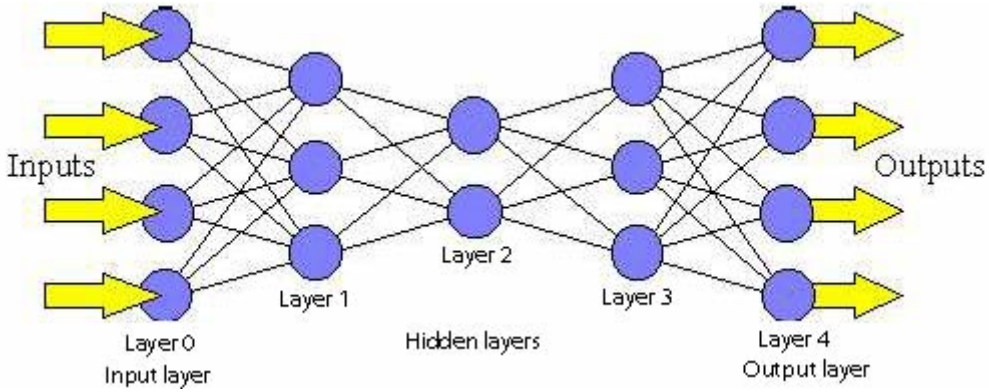
Neural networks

Linear regression:

C1(wx+b)+C2(wx+b)+C3(wx+b)+C4(wx+b)+C5(wx+b)+C5(wx+b)

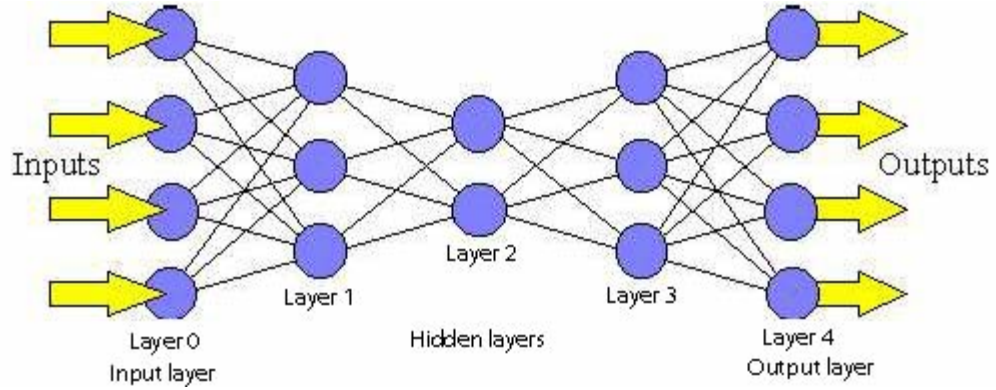
A feed forward neural network – layers of neurons:

f((w3f((w2f((w1x)+b1))+b2))+b3)



Neural networks

How can we define this as mathematical operations:

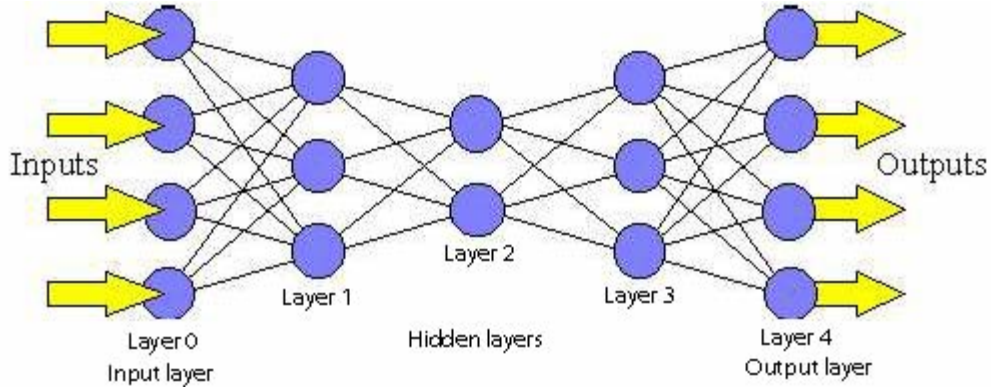


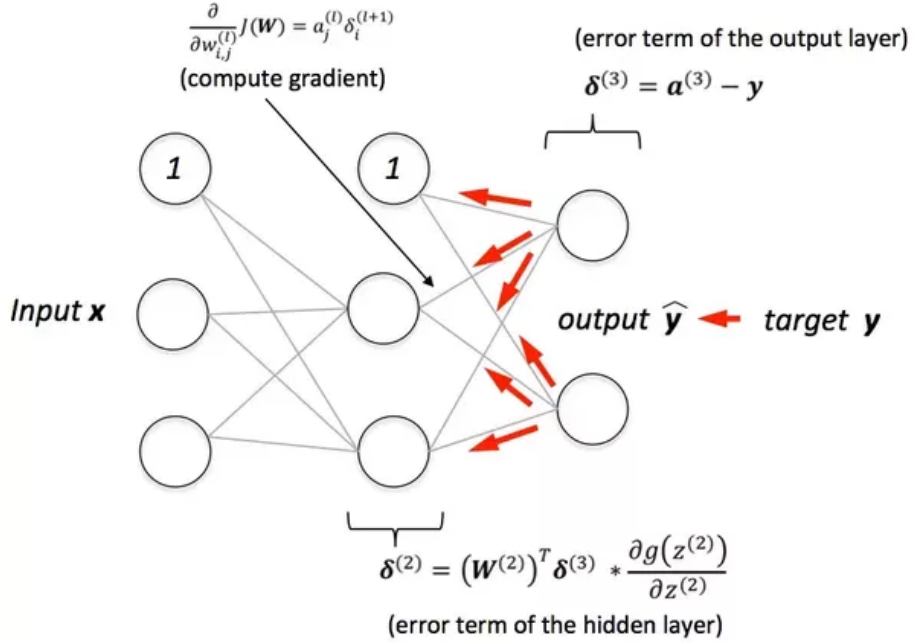
Neural networks

A feed-forward network is great

How could we teach such a network:

- Backpropagation algorithm





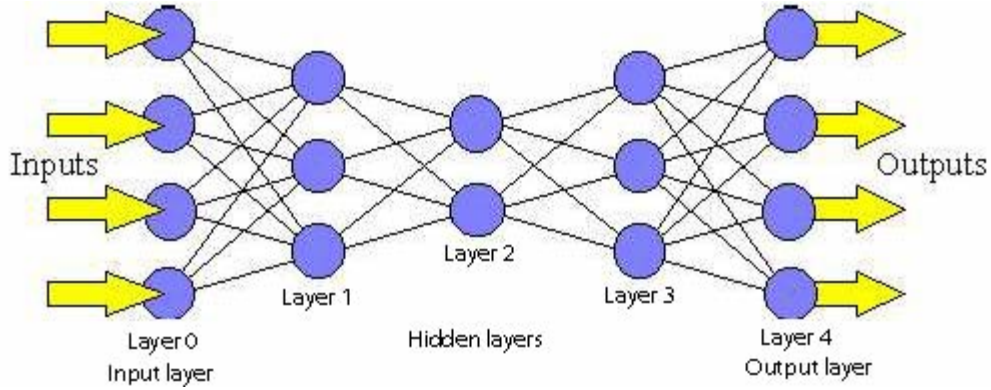
Neural networks

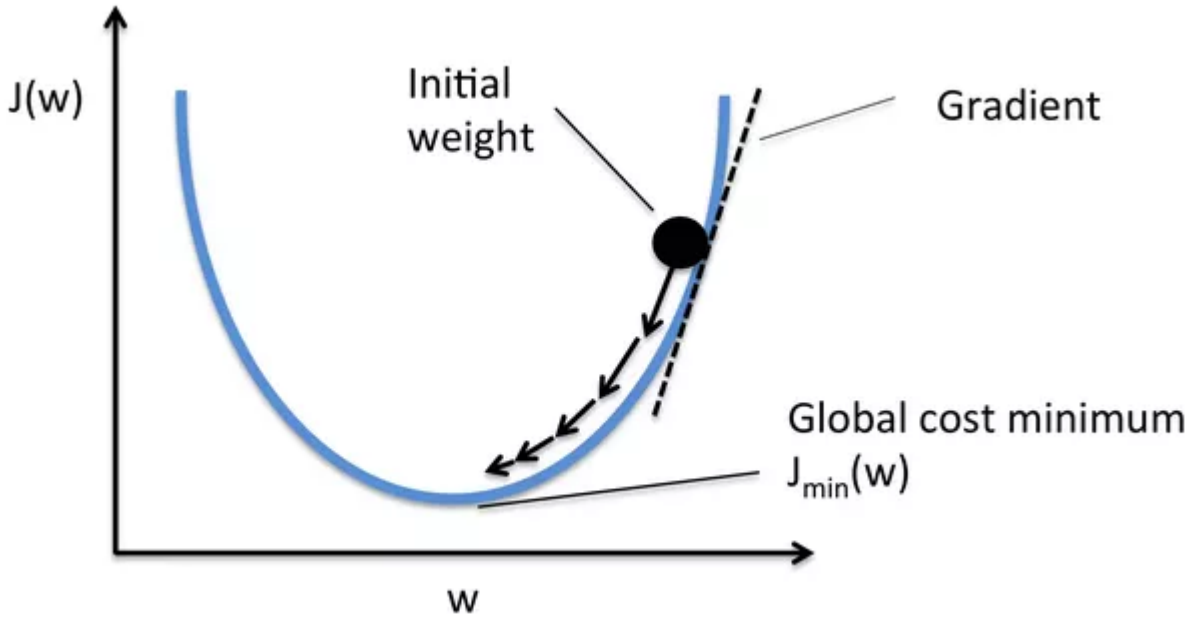
A feed-forward network is great

How could we teach such a network:

- Backpropagation algorithm

- Stochastic gradient descent





Neural networks

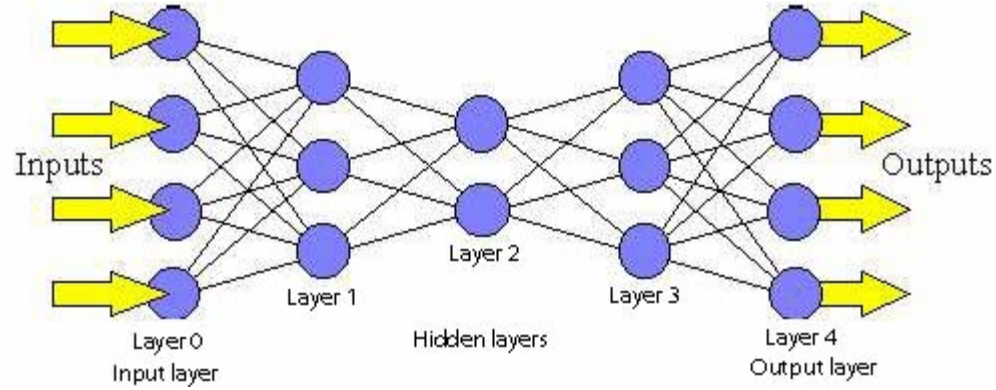
A feed-forward network is great

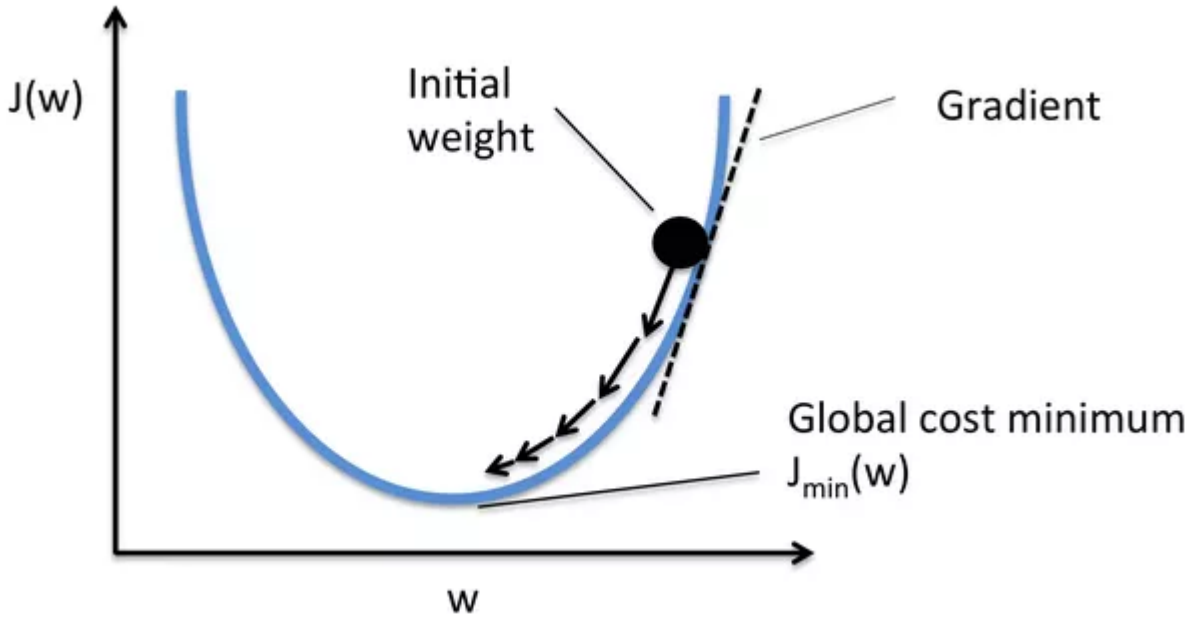
How could we teach such a network:

- Backpropagation algorithm

- Stochastic gradient descent

Tensorflow is here to help



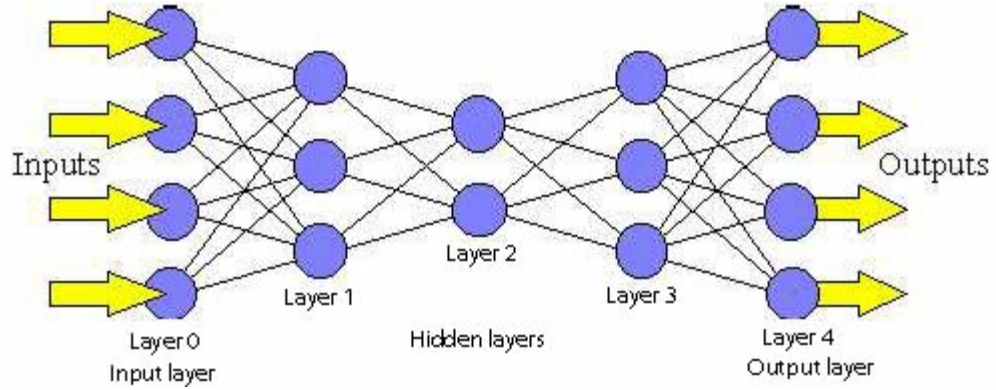


Neural networks

Regularization:

Using batches

Adding Dropout



Neural networks

```
tf.reset_default_graph()
# Parameters
learning_rate = 0.1
num_steps = 30000  #number of steps to train the network - one can stop training earlier
BatchLength = 52 #number of data fed paralell to the network

# Network Parameters
NumNeurons=[13,64,64,64,64,64,1] #the number of neruons in each layer

# tf Graph input -placeholders to feed data to the network
InputData = tf.placeholder(tf.float32, [None, NumNeurons[0]])
DesiredOutput = tf.placeholder(tf.float32, [None,1])
KeepProb = tf.placeholder(tf.float32)

CurrentInput=InputData
#this loop wil generate our network layer by layer
for i in range(len(NumNeurons)-1):
  with tf.variable_scope('layer'+str(i)):
    W=tf.Variable(tf.random_normal([NumNeurons[i], NumNeurons[i+1]], 0, 0.1))
    B=tf.Variable(tf.random_normal([NumNeurons[i+1]], 0, 0.1,))
    CurrentInput = tf.add(tf.matmul(CurrentInput, W), B)
    if i!=(len(NumNeurons)-2):
      #the nonlinearity of the network
      CurrentInput=tf.sigmoid(CurrentInput)
      #CurrentInput=tf.nn.relu(CurrentInput)
      #random dropout to keep variance
      CurrentInput= tf.nn.dropout(CurrentInput, KeepProb)
Output=CurrentInput

#definitaiton of the error of the network
loss = tf.reduce_mean( tf.abs( tf.subtract(DesiredOutput , Output)))
optimizer = tf.train.AdagradOptimizer(learning_rate).minimize(loss)
#optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
#optimizer = tf.train.AdamOptimizer(learning_rate).minimize(loss)
```