CS-618
Deep Learning
Spring 2023

Lecture 2

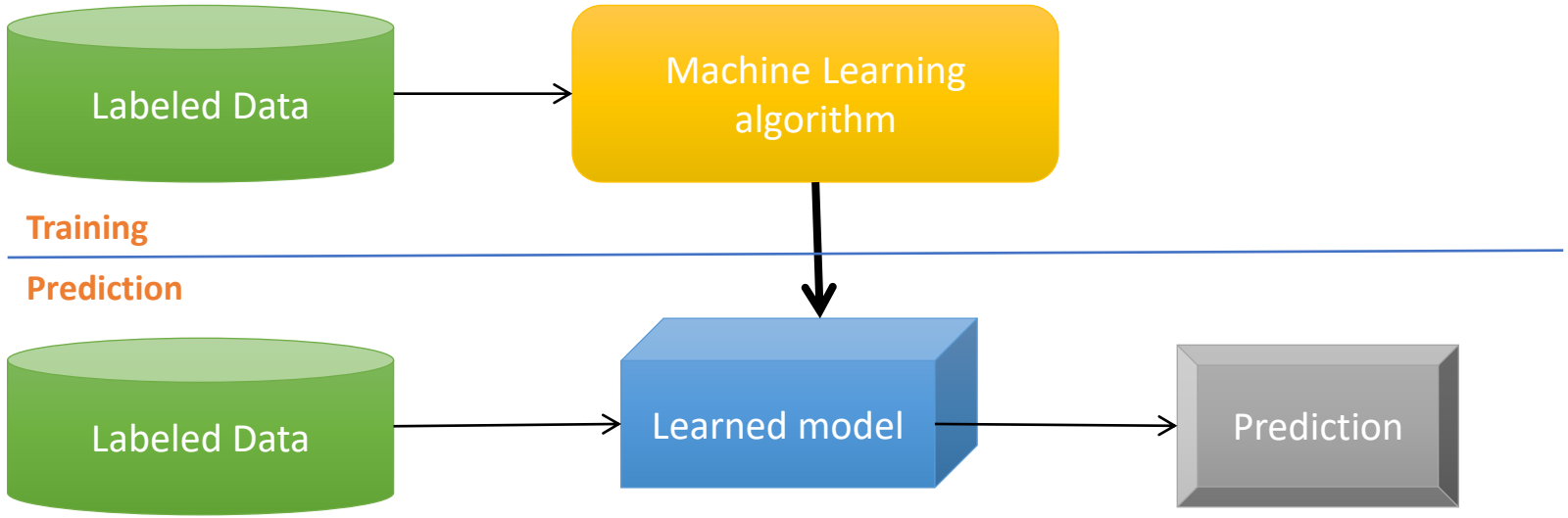
Outline

- Machine learning basics
  - Supervised and unsupervised learning
  - Linear and non-linear classification methods
- Introduction to deep learning
- Elements of neural networks (NNs)
  - Activation functions
- Training NNs
  - Gradient descent
  - Regularization methods
- NN architectures
  - Convolutional NNs
  - Recurrent NNs

Machine Learning Basics

Artificial Intelligence is a scientific field concerned with the development of algorithms that allow computers to learn without being explicitly programmed

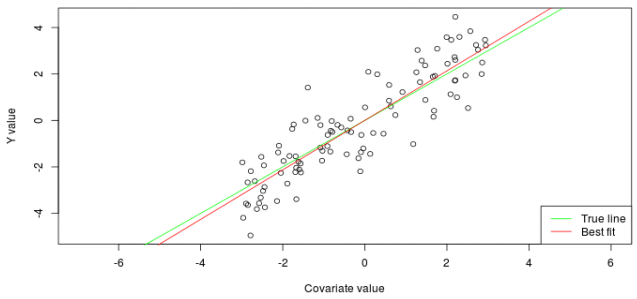
Machine Learning is a branch of Artificial Intelligence, which focuses on methods that learn from data and make predictions on unseen data



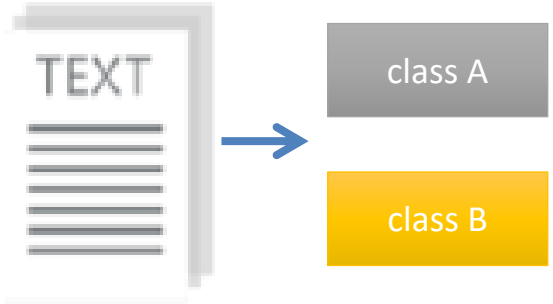
Main types of machine learning

Machine Learning Types

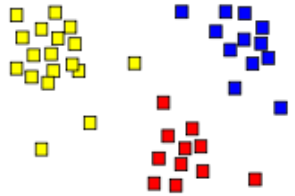
- ***Supervised***: learning with labeled data
  - Example: email classification, image classification
  - Example: regression for predicting real-valued outputs
- ***Unsupervised***: discover patterns in unlabeled data
  - Example: cluster similar data points
- ***Reinforcement learning***: learn to act based on feedback/reward
  - Example: learn to play Go



Regression

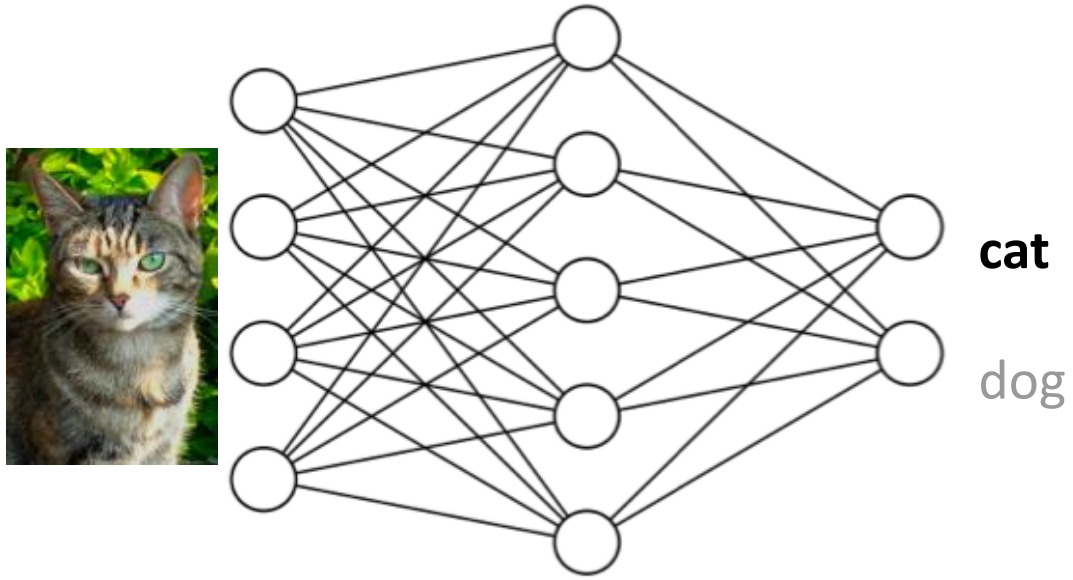


Classification


Clustering

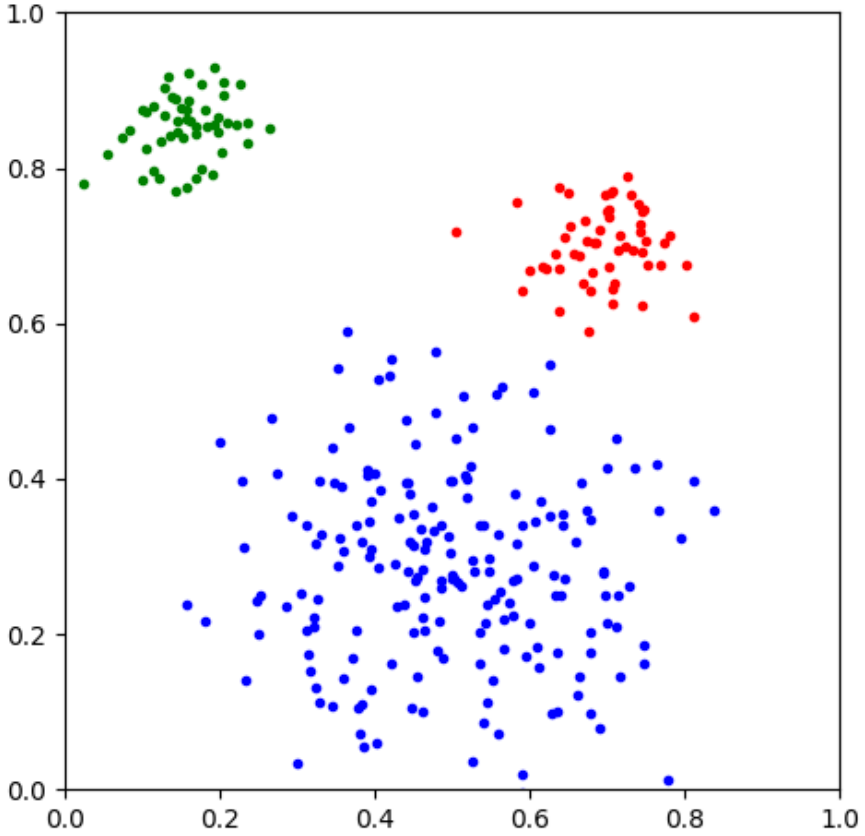
Main types of machine learning

- **Supervised learning**
- Unsupervised learning
- Self-supervised learning
- Reinforcement learning



Main types of machine learning

- Supervised learning
- **Unsupervised learning**
- Self-supervised learning
- Reinforcement learning



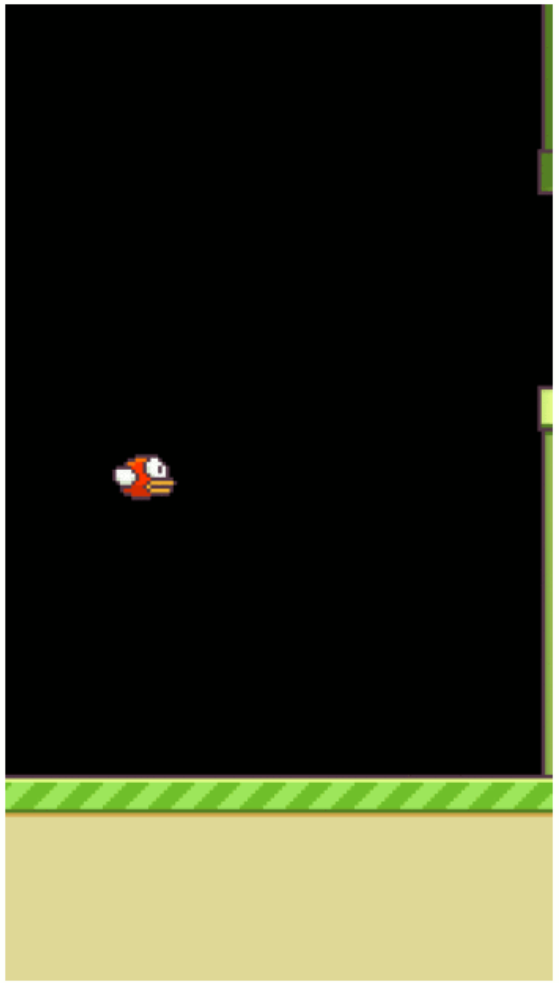
Main types of machine learning

- Supervised learning
- Unsupervised learning
- **Self-supervised learning**
- Reinforcement learning

Image from https://arxiv.org/abs/1710.10196

Main types of machine learning

- Supervised learning
- Unsupervised learning
- Self-supervised learning
- **Reinforcement learning**



Animation from https://yanpanlau.github.io/2016/07/10/FlappyBird-Keras.html

Supervised Learning

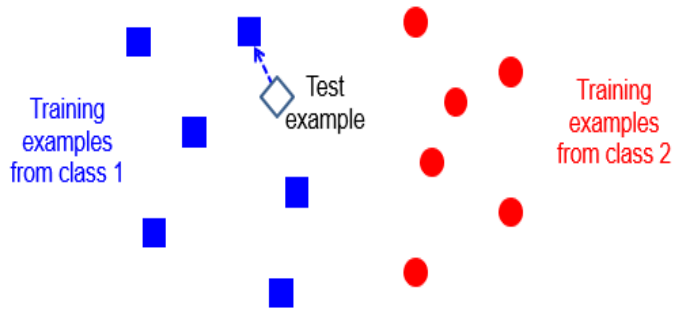
- ***Supervised learning*** categories and techniques
  - **Numerical classifier functions**
    - Linear classifier, perceptron, logistic regression, support vector machines (SVM), neural networks
  - **Parametric (probabilistic) functions**
    - Naïve Bayes, Gaussian discriminant analysis (GDA), hidden Markov models (HMM), probabilistic graphical models
  - **Non-parametric (instance-based) functions**
    - *k*-nearest neighbors, kernel regression, kernel density estimation, local regression
  - **Symbolic functions**
    - Decision trees, classification and regression trees (CART)
  - **Aggregation (ensemble) learning**
    - Bagging, boosting (Adaboost), random forest

Unsupervised Learning

- ***Unsupervised learning*** categories and techniques
  - **Clustering**
    - *k*-means clustering
    - Mean-shift clustering
    - Spectral clustering
  - **Density estimation**
    - Gaussian mixture model (GMM)
    - Graphical models
  - **Dimensionality reduction**
    - Principal component analysis (PCA)
    - Factor analysis

Nearest Neighbor Classifier

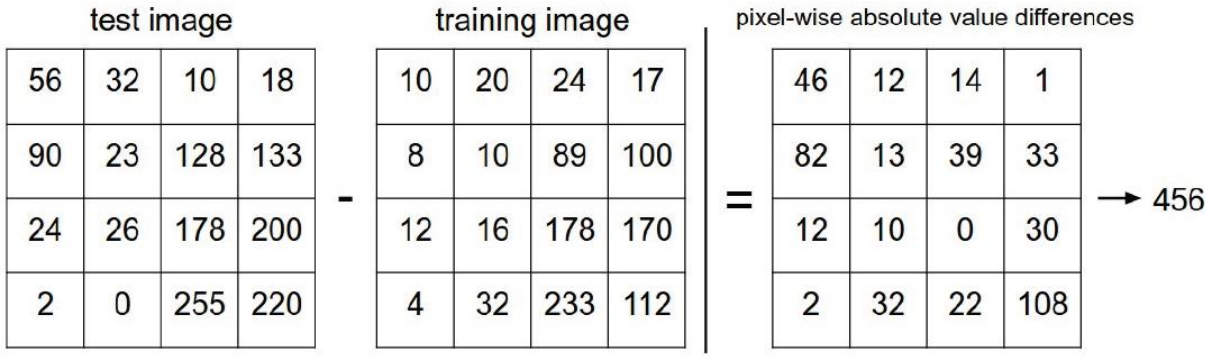
- ***Nearest Neighbor*** – for each test data point, assign the class label of the nearest training data point
  - Adopt a distance function to find the nearest neighbor
    - Calculate the distance to each data point in the training set, and assign the class of the nearest data point (minimum distance)
  - It does not require learning a set of weights



Picture from: James Hays – Machine Learning Overview

Nearest Neighbor Classifier

- For image classification, the distance between all pixels is calculated (e.g., using ℓ_1 norm, or ℓ_2 norm)
  - Accuracy on CIFAR-10: 38.6%
- Disadvantages:
  - The classifier must remember all training data and store it for future comparisons with the test data
  - Classif image



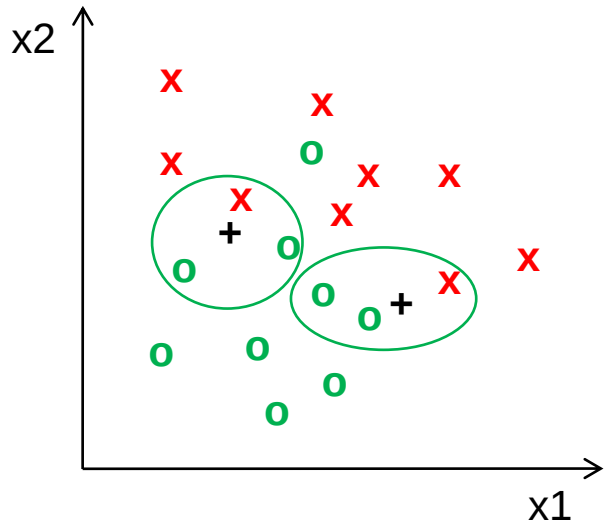
a comparison to all training

ℓ_1 norm
(Manhattan distance)

$$d_1(I_1, I_2) = \Sigma_p \, |I_1^p - I_2^p|$$

k-Nearest Neighbors Classifier

- ***k-Nearest Neighbors*** approach considers multiple neighboring data points to classify a test data point
  - E.g., 3-nearest neighbors
    - The test example in the figure is the + mark
    - The class of the test example is obtained by voting (based on the distance to the 3 closest points)



Linear Classifier

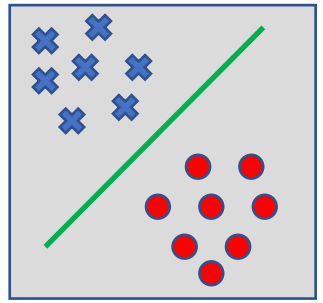
- ***Linear classifier***
  - Find a linear function *f* of the inputs x_i that separates the classes

$$f(x_i, W, b) = Wx_i + b$$

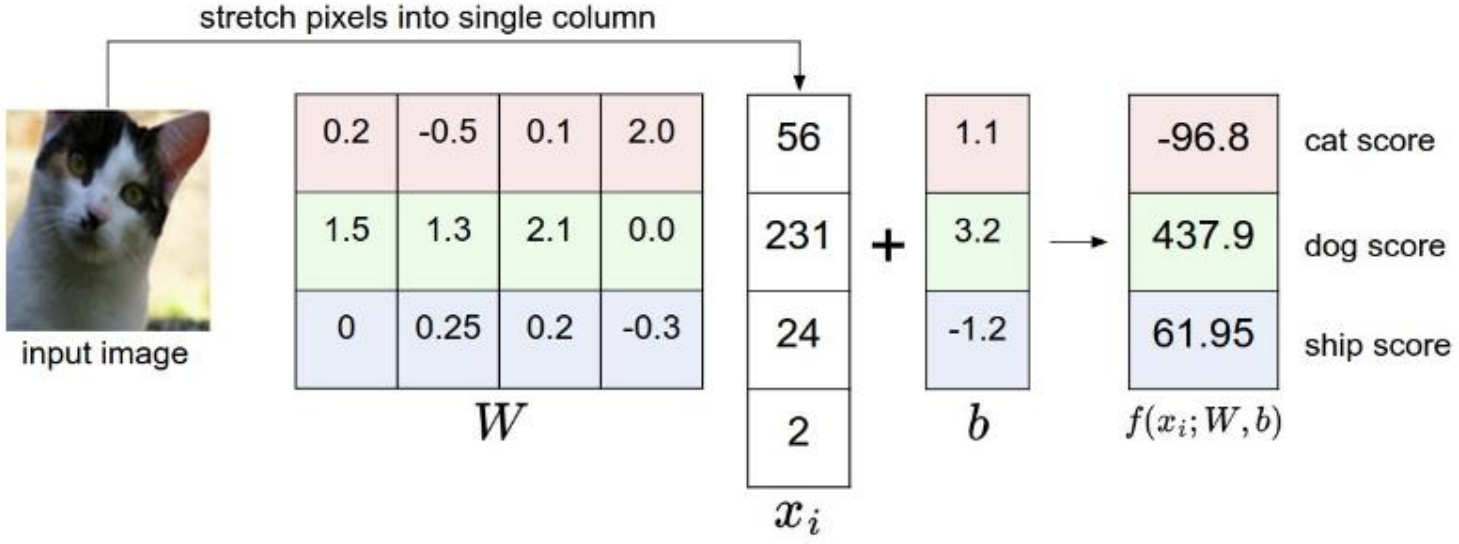
  - Use pairs of inputs and labels to find the weights matrix *W* and the bias vector *b*
    - The weights and biases are the parameters of the function *f*
  - Several methods have been used to find the optimal set of parameters of a linear classifier
    - A common method of choice is the ***Perceptron*** algorithm, where the parameters are updated until a minimal error is reached (single layer, does not use backpropagation)
  - Linear classifier is a simple approach, but it is a building block of advanced classification algorithms, such as SVM and neural networks
    - Earlier multi-layer neural networks were referred to as multi-layer perceptrons (MLPs)

Linear Classifier

- The decision boundary is linear
  - A straight line in 2D, a flat plane in 3D, a hyperplane in 3D and higher dimensional space

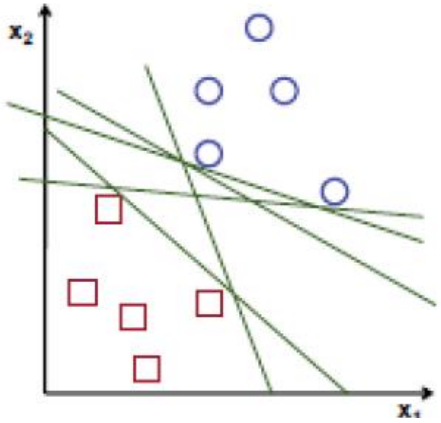


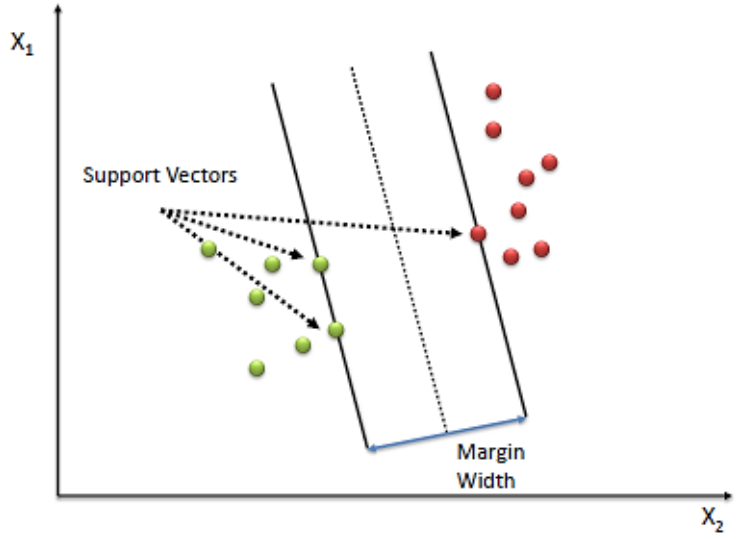
- Example: classify an input image
  -



Support Vector Machines

- ***Support vector machines (SVM)***
  - How to find the best decision boundary?
    - All lines in the figure correctly separate the 2 classes
    - The line that is farthest from all training examples will have better generalization capabilities
  - SVM solves an optimization problem:
    - First, identify a decision boundary that correctly classifies the examples



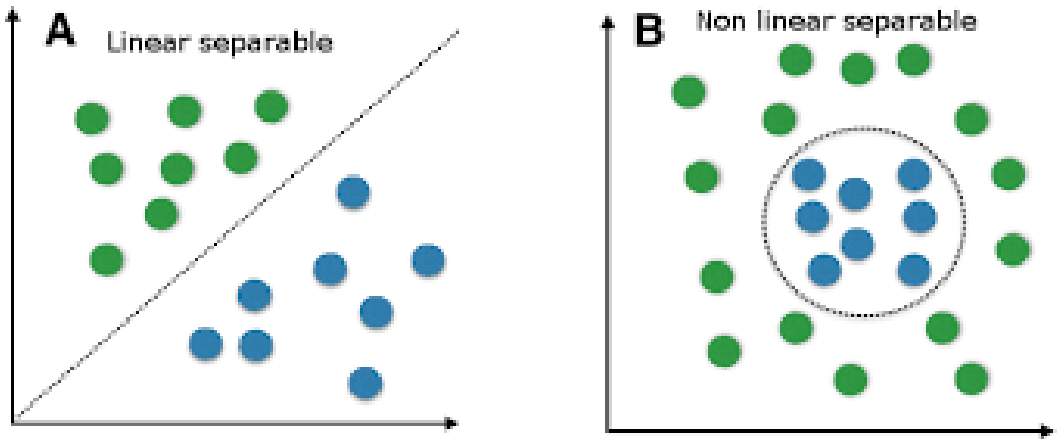


Linear vs Non-linear Techniques

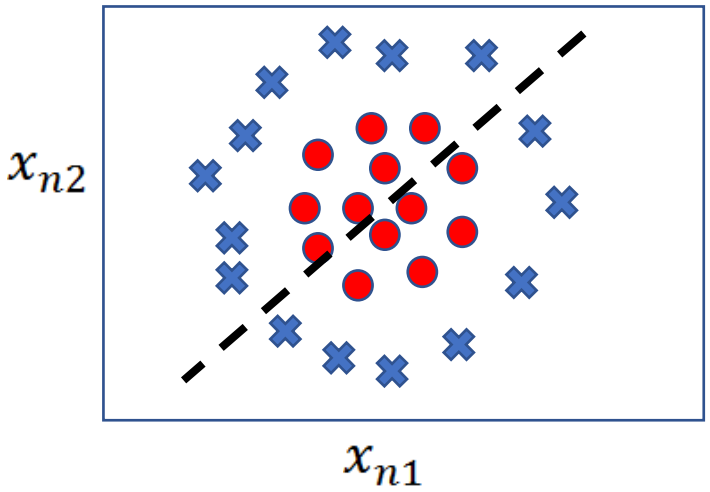
- Linear classification techniques
  - Linear classifier
  - Perceptron
  - Logistic regression
  - Linear SVM
  - Naïve Bayes
- Non-linear classification techniques
  - *k*-nearest neighbors
  - Non-linear SVM
  - Neural networks
  - Decision trees
  - Random forest

Linear vs Non-linear Techniques

- For some tasks, input data can be linearly separable, and linear classifiers can be suitably applied

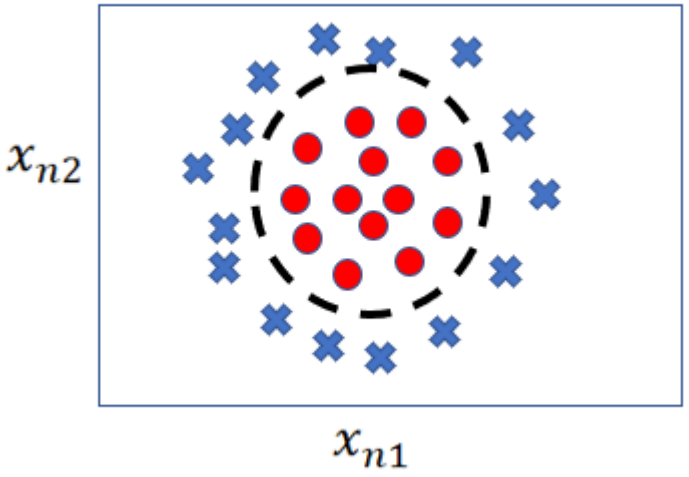


- For other tasks, linear classifiers may have difficulties to produce adequate decision boundaries



Non-linear Techniques

- Non-linear classification
  - Features z_i are obtained as non-linear functions of the inputs x_i
  - It results in non-linear decision boundaries
  - Can deal with non-linearly separable data



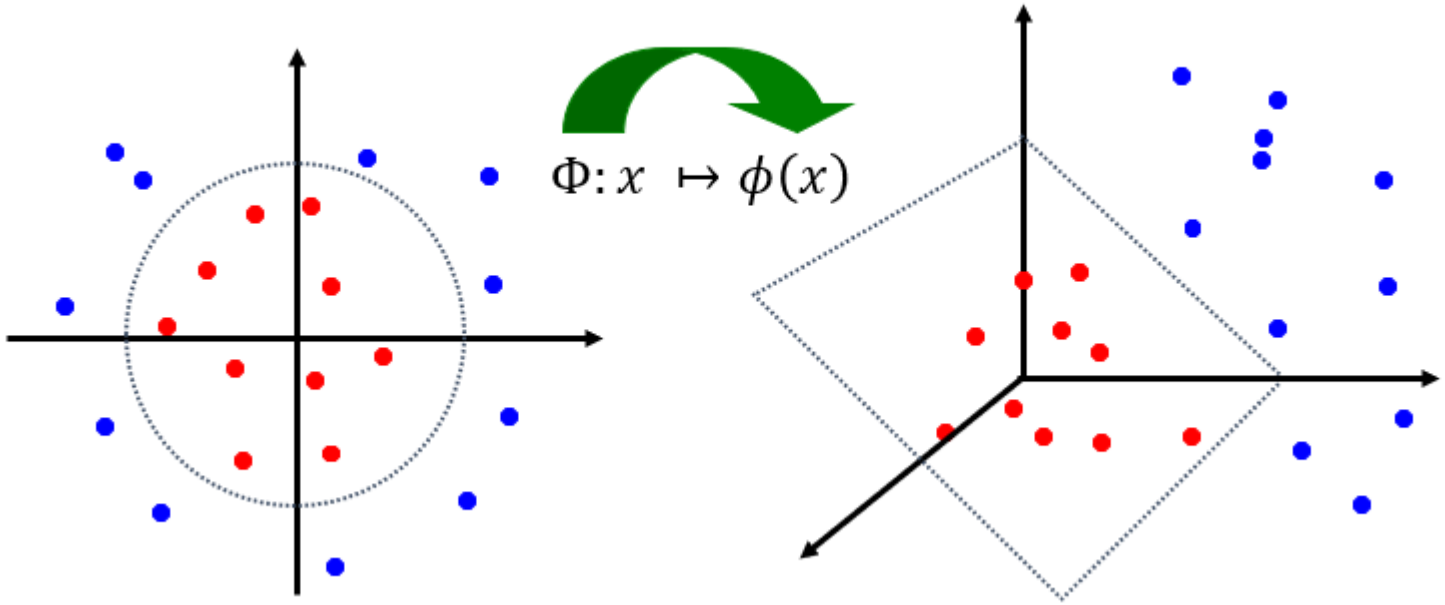
Inputs: $x_i = [x_{n1} \quad x_{n2}]$

Features: $z_i = [x_{n1} \quad x_{n2} \quad x_{n1} \cdot x_{n2} \quad x_{n1}^2 \quad x_{n2}^2]$

Outputs: $f(x_i, W, b) = Wz_i + b$

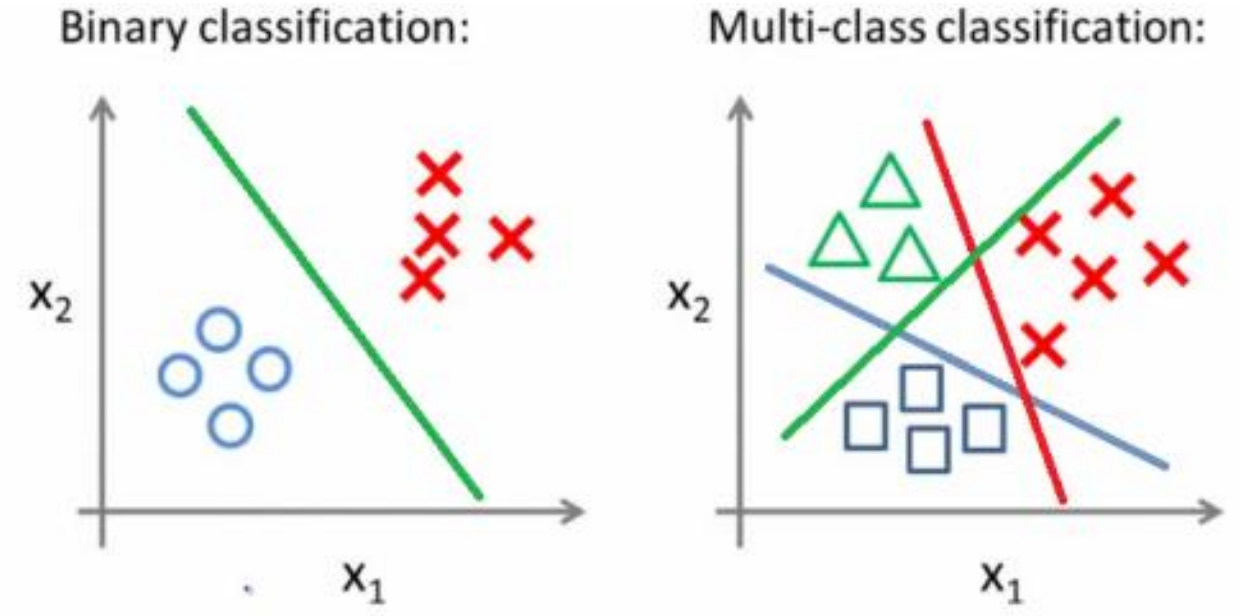
Non-linear Support Vector Machines

- ***Non-linear SVM***
  - The original input space is mapped to a higher-dimensional feature space where the training set is linearly separable
  - Define a non-linear kernel function to calculate a non-linear decision boundary in the original feature space



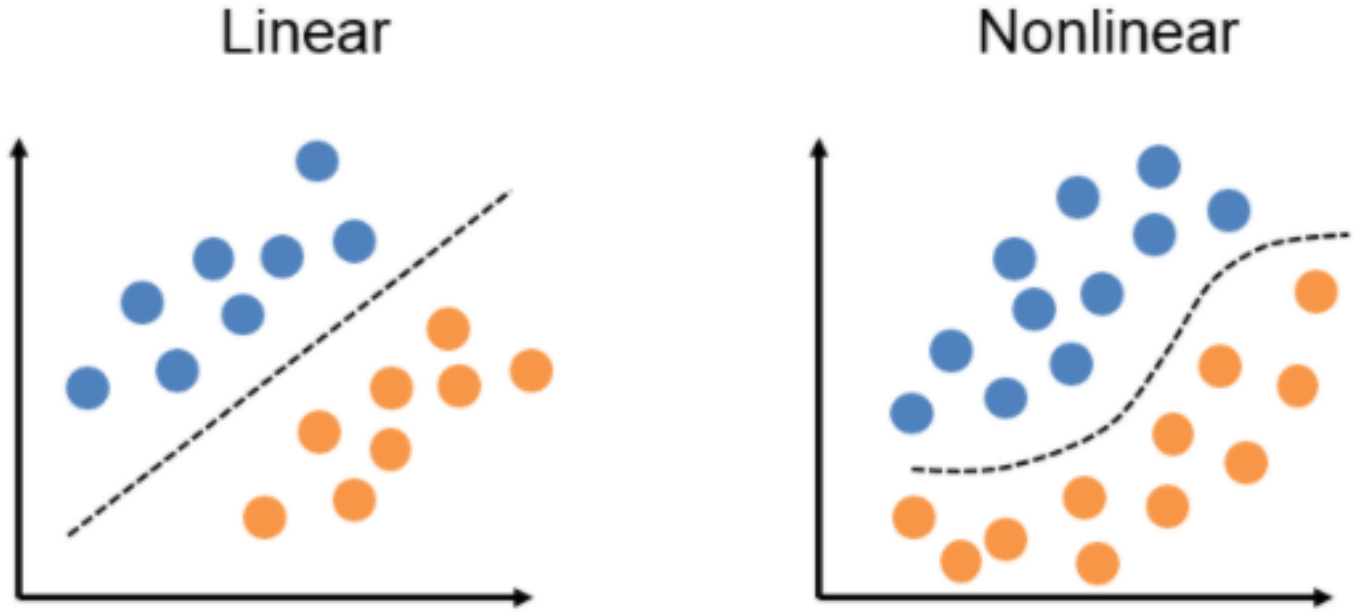
Binary vs Multi-class Classification

- A classification problem with only 2 classes is referred to as ***binary classification***
  - The output labels are 0 or 1
  - E.g., benign or malignant tumor, spam or no-spam email
- A problem with 3 or more classes is referred to as ***multi-class classification***



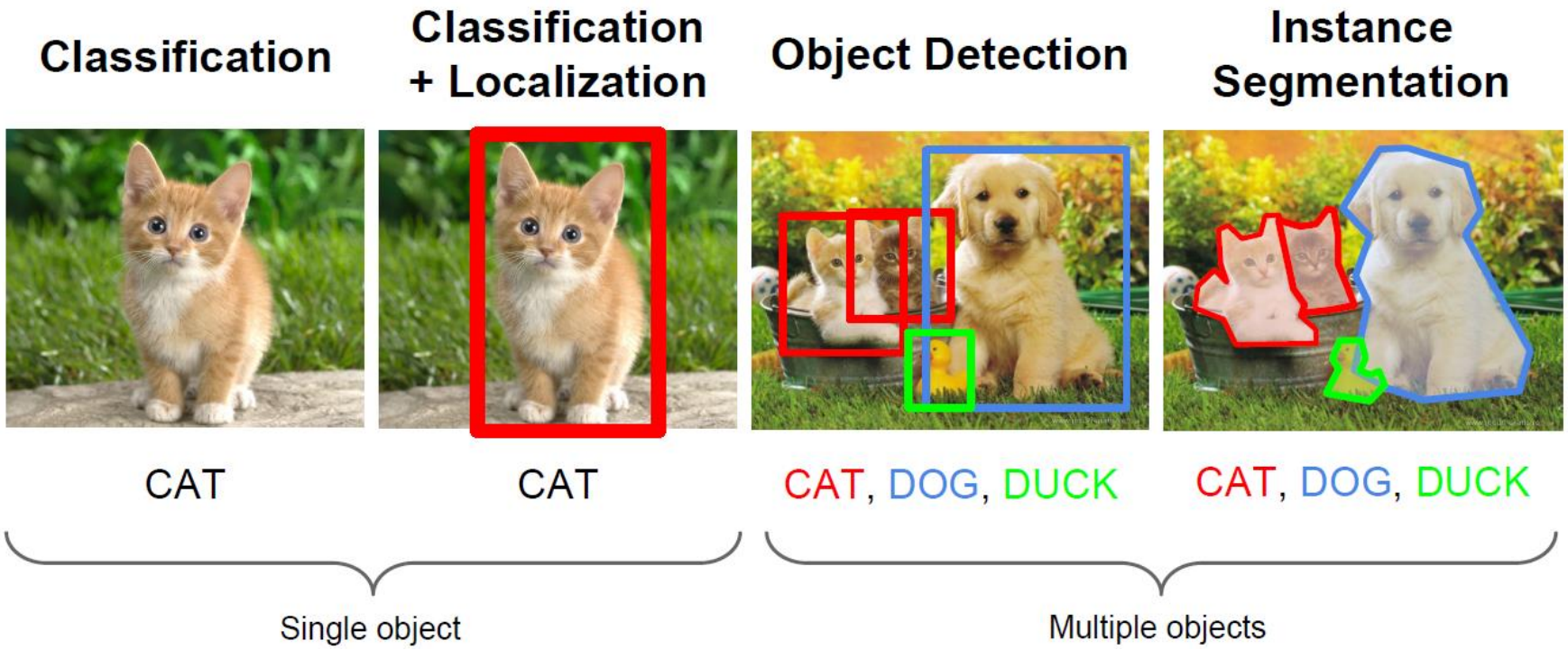
Binary vs Multi-class Classification

- Both the binary and multi-class classification problems can be linearly or non-linearly separated
  - Figure: linearly and non-linearly separated data for binary classification problem



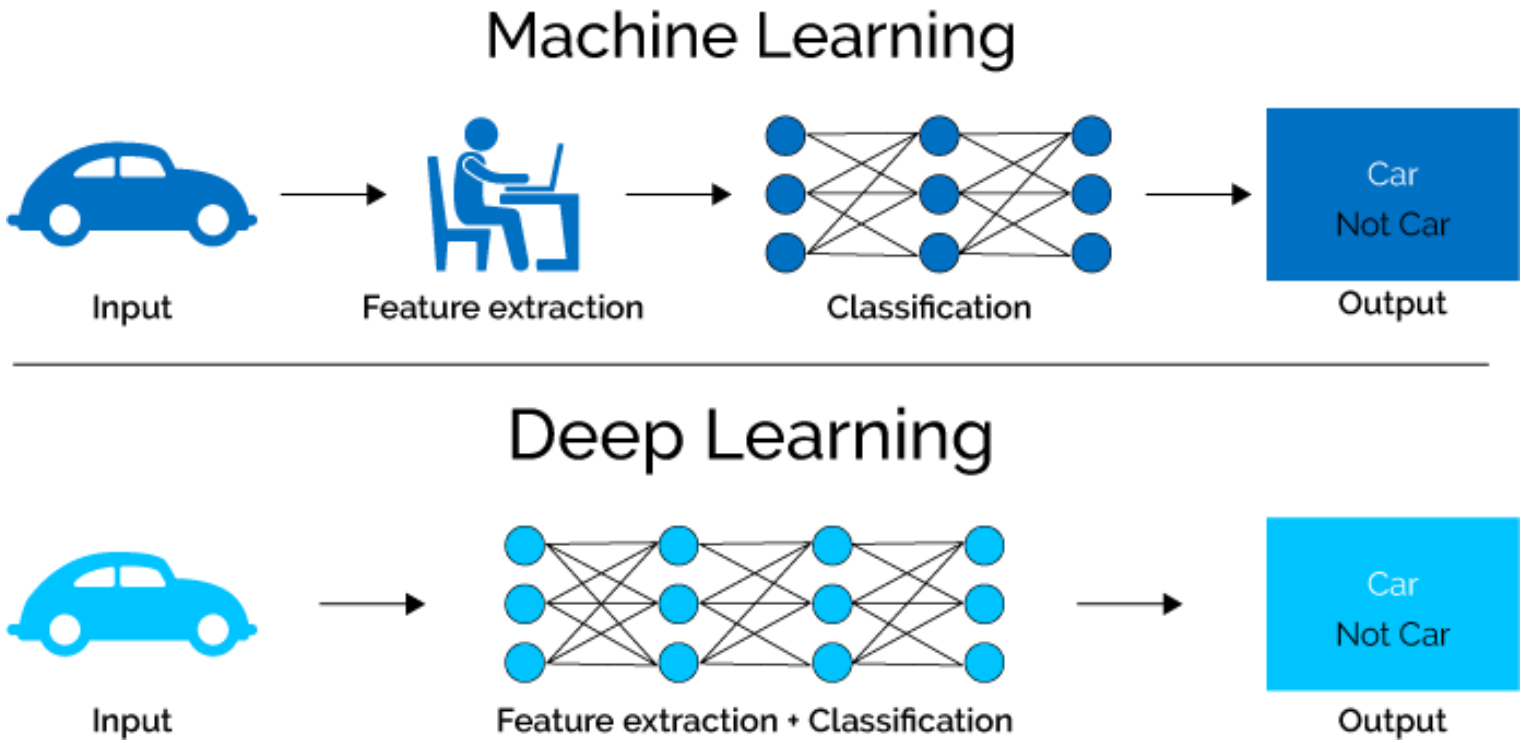
Computer Vision Tasks

- Computer vision has been the primary area of interest for ML
- The tasks include: classification, localization, object detection, instance segmentation



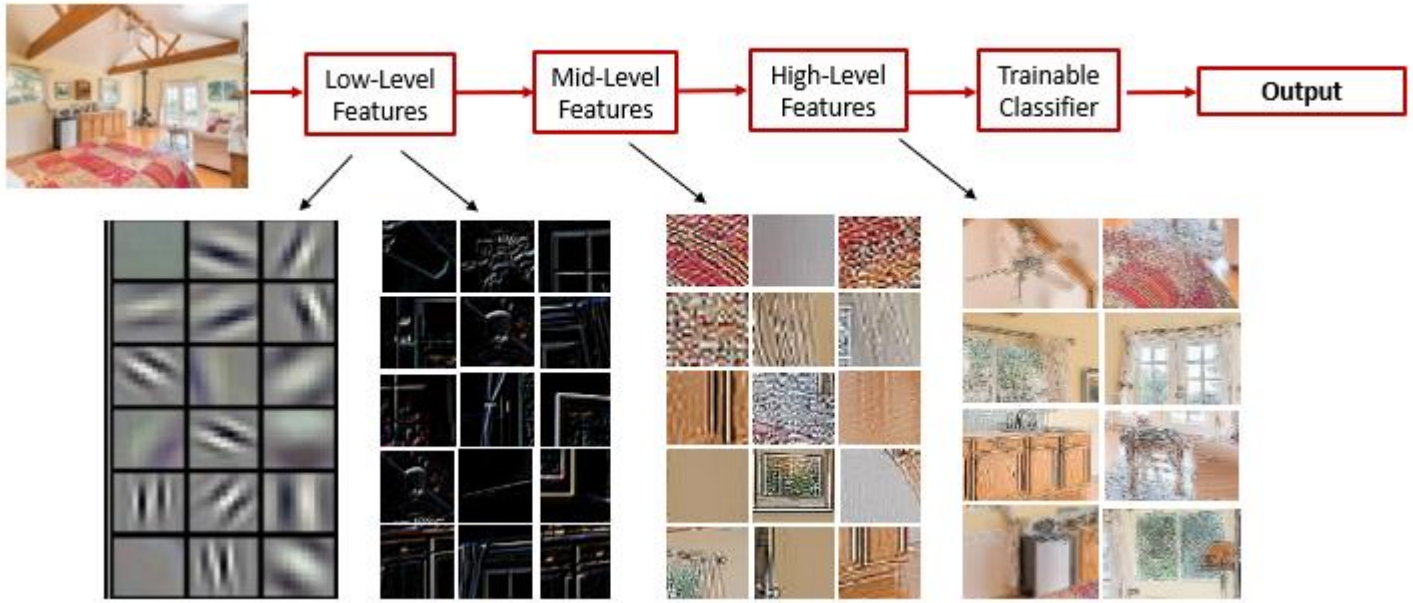
ML vs. Deep Learning

- ***Deep learning*** (DL) is a machine learning subfield that uses multiple layers for learning data representations
  - DL is exceptionally effective at learning patterns



ML vs. Deep Learning

- DL applies a multi-layer process for learning rich hierarchical features (i.e., data representations)
  - Input image pixels → Edges → Textures → Parts → Objects



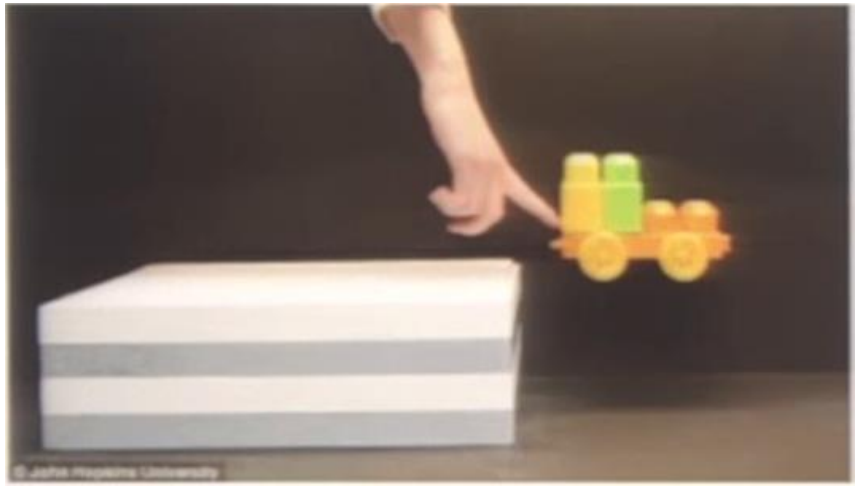
Why is DL Useful?

- DL provides a flexible, learnable framework for representing visual, text, linguistic information
  - Can learn in supervised and unsupervised manner
- DL represents an effective end-to-end learning system
- Requires large amounts of training data
- Since about 2010, DL has outperformed other ML techniques
  - First in vision and speech, then NLP, and other applications

Fundamentals of machine learning

Data

- Humans learn by observation and unsupervised learning
  - model of the world / common sense reasoning
- Machine learning needs lots of (labeled) data to compensate



Data

- Tensors: generalization of matrices to *n* dimensions (or rank, order, degree)
  - 1D tensor: vector
  - 2D tensor: matrix
  - 3D, 4D, 5D tensors
  - `numpy.ndarray(shape, dtype)`
- Training – validation – test split (+ adversarial test)
- Minibatches
  - small sets of input data used at a time
  - usually processed independently

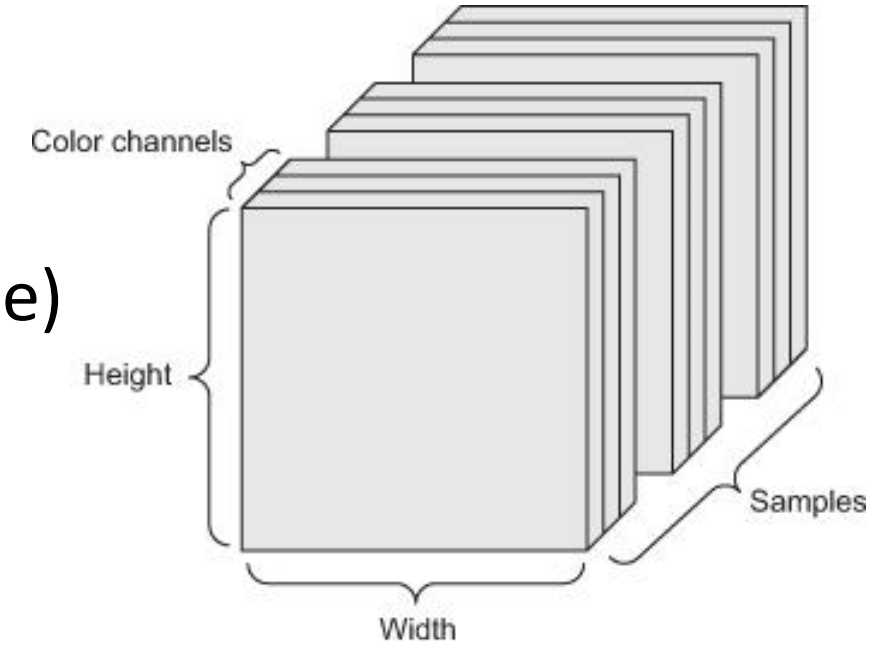
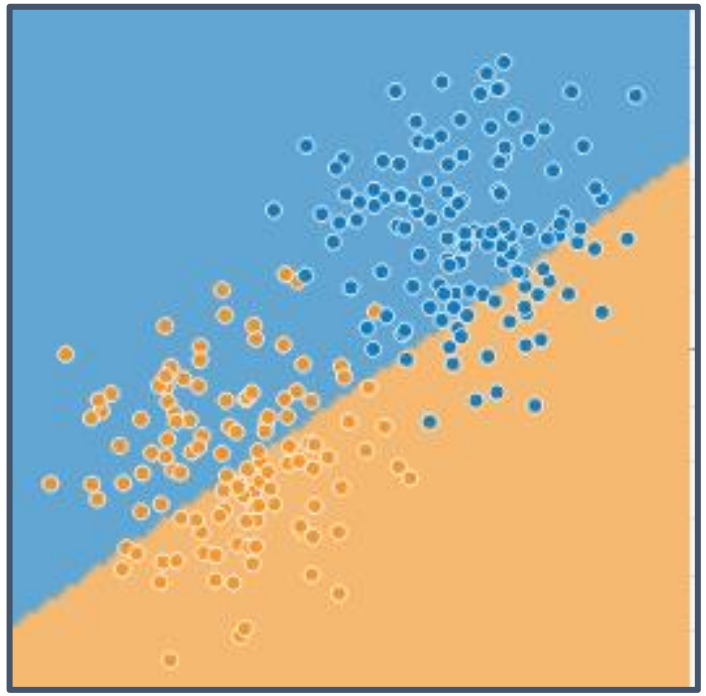
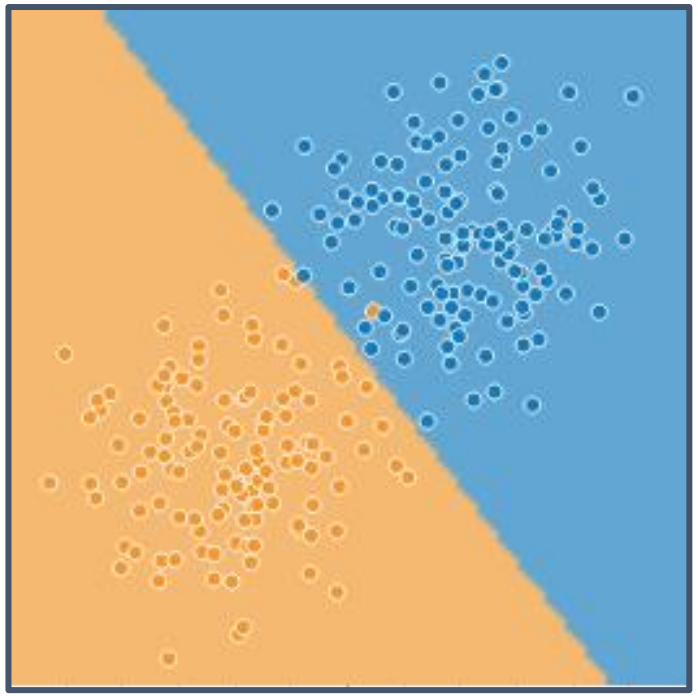
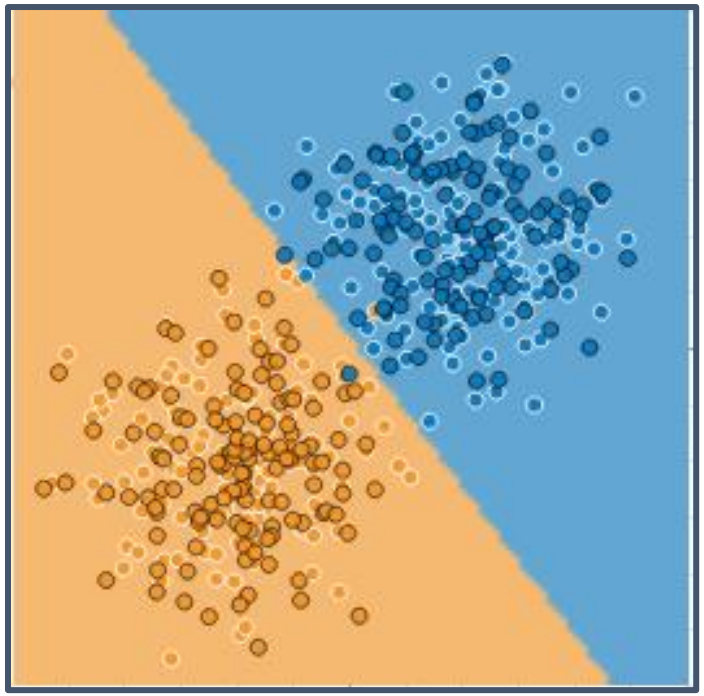


Image from: https://arxiv.org/abs/1707.08945

Model – learning/training – inference







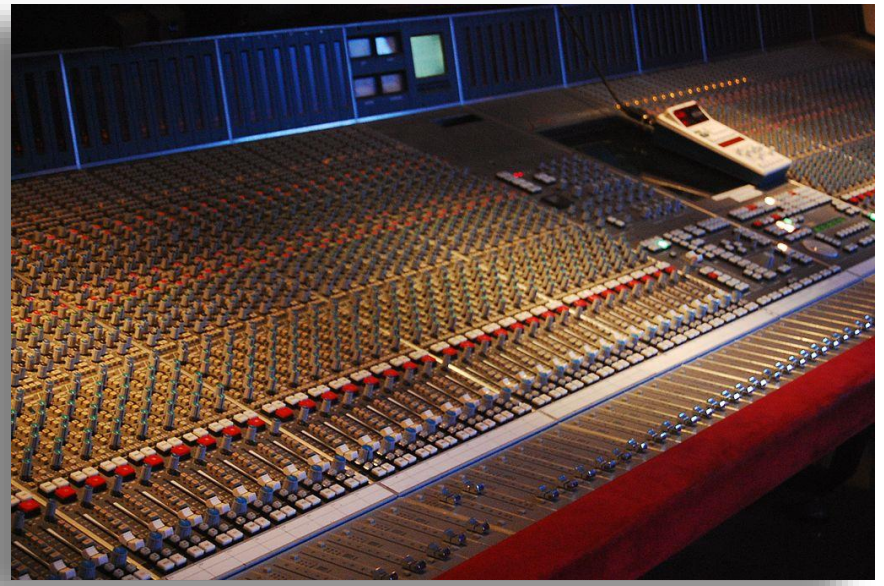
http://playground.tensorflow.org/

$$\hat{y} = f(\mathbf{x}; \theta)$$

- parameters θ and hyperparameters

Optimization

- Mathematical optimization:
  "the selection of a best element (with regard to some criterion) from some set of available alternatives" (Wikipedia)
- Main types:
  finite-step, iterative, heuristic
- Learning as an optimization problem
  - cost function:



loss

regularization

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} L(f(\mathbf{x}_i;\theta), y_i) + R(\theta)$$

Optimization

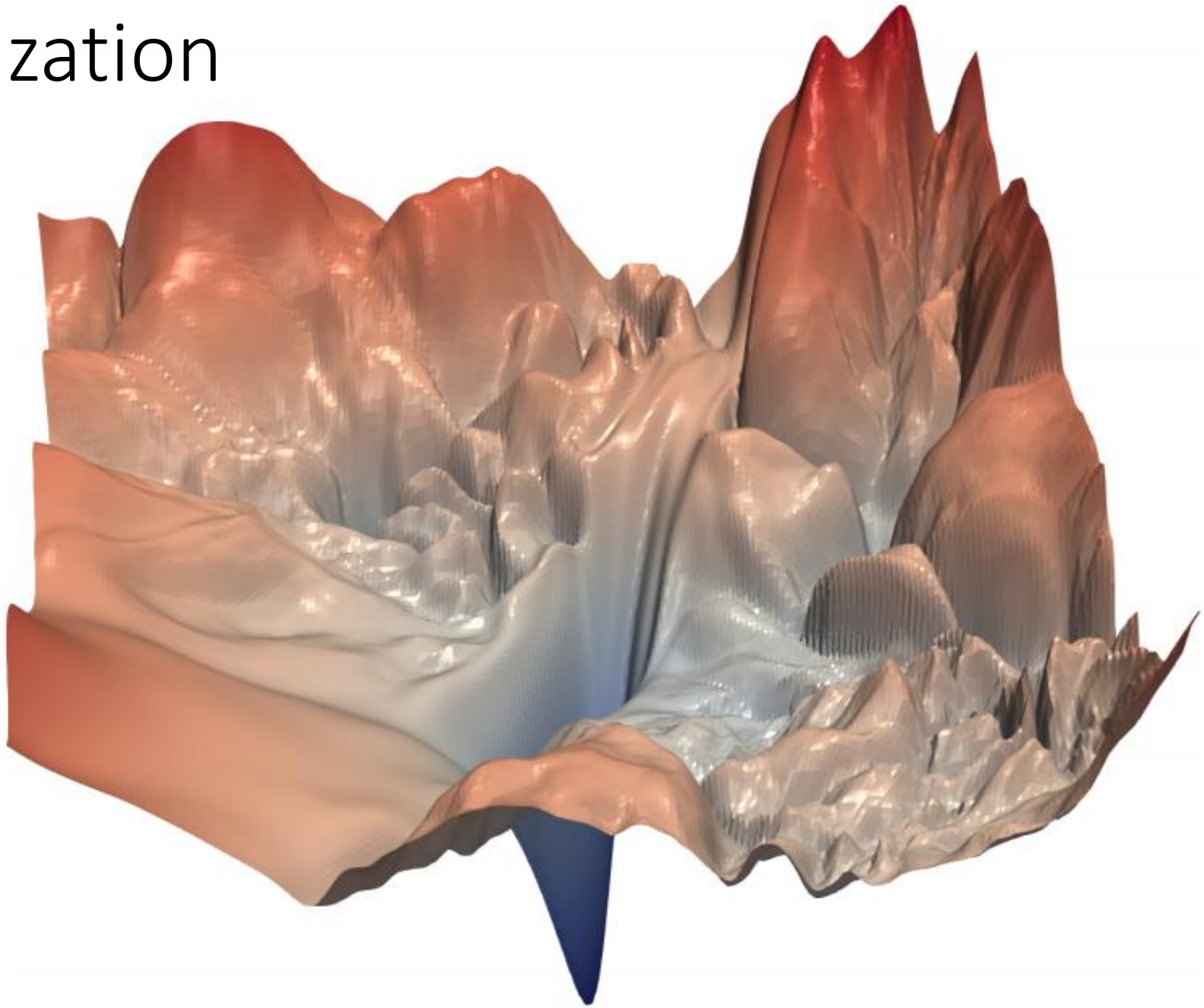
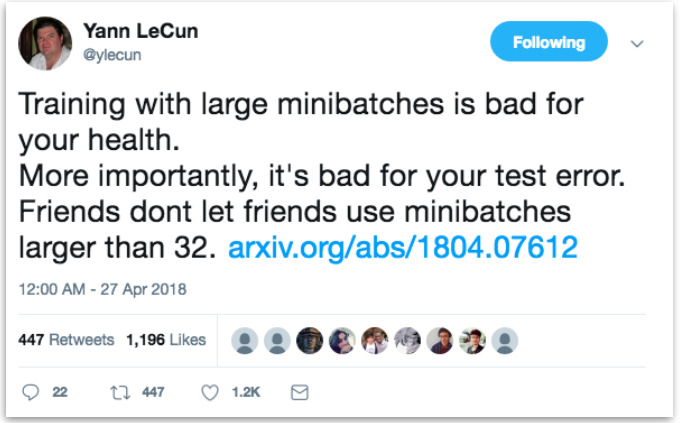


Image from: Li et al. “Visualizing the Loss Landscape of Neural Nets”, arXiv:1712.09913

Gradient descent



- Derivative and minima/maxima of functions
- Gradient: the derivative of a multivariable function
- Gradient descent:

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial J(\theta)}{\partial \theta}$$

- (Mini-batch) stochastic gradient descent (and its variants)

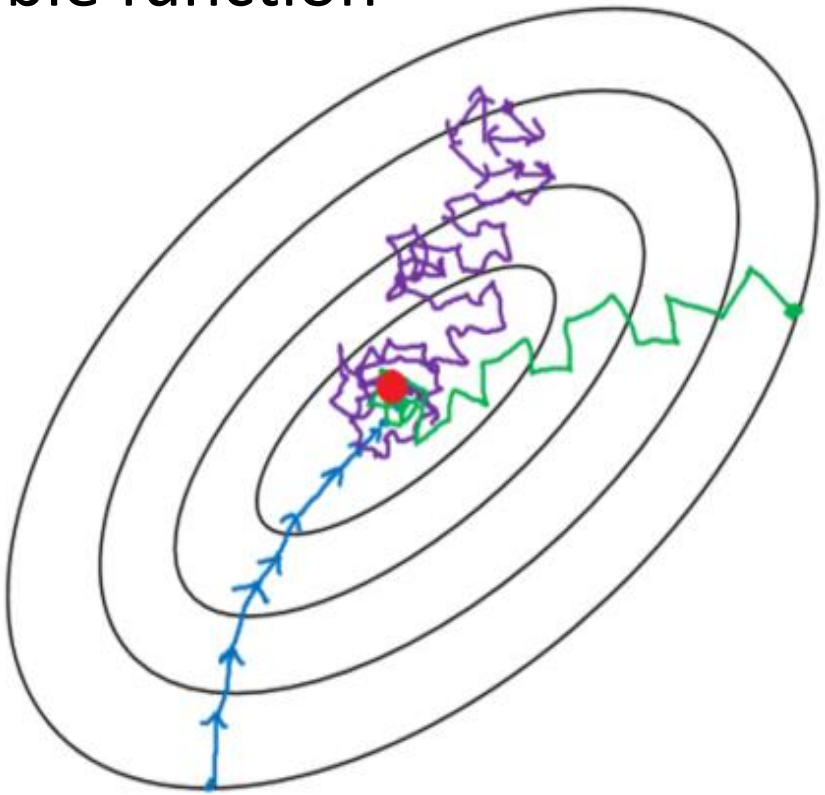
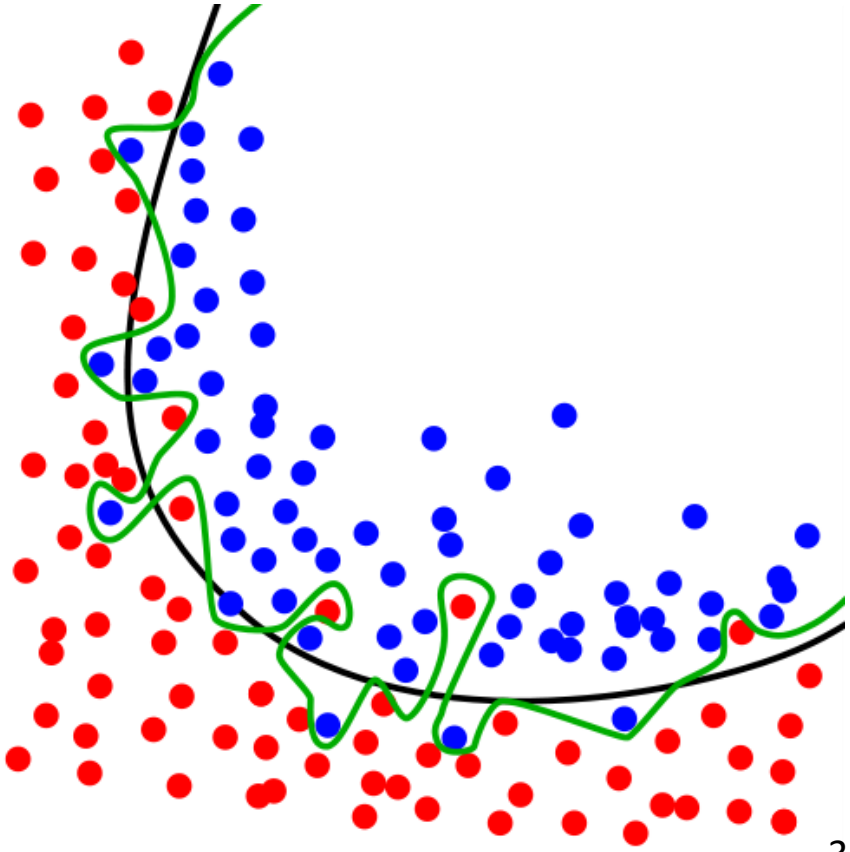


Image from: https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

Over- and underfitting, generalization, regularization

- Models with lots of parameters can easily overfit to training data
- **Generalization:** the quality of ML model is measured on new, unseen samples
- **Regularization:** any method* to prevent overfitting
  - simplicity, sparsity, dropout, early stopping
  - *) other than adding more data

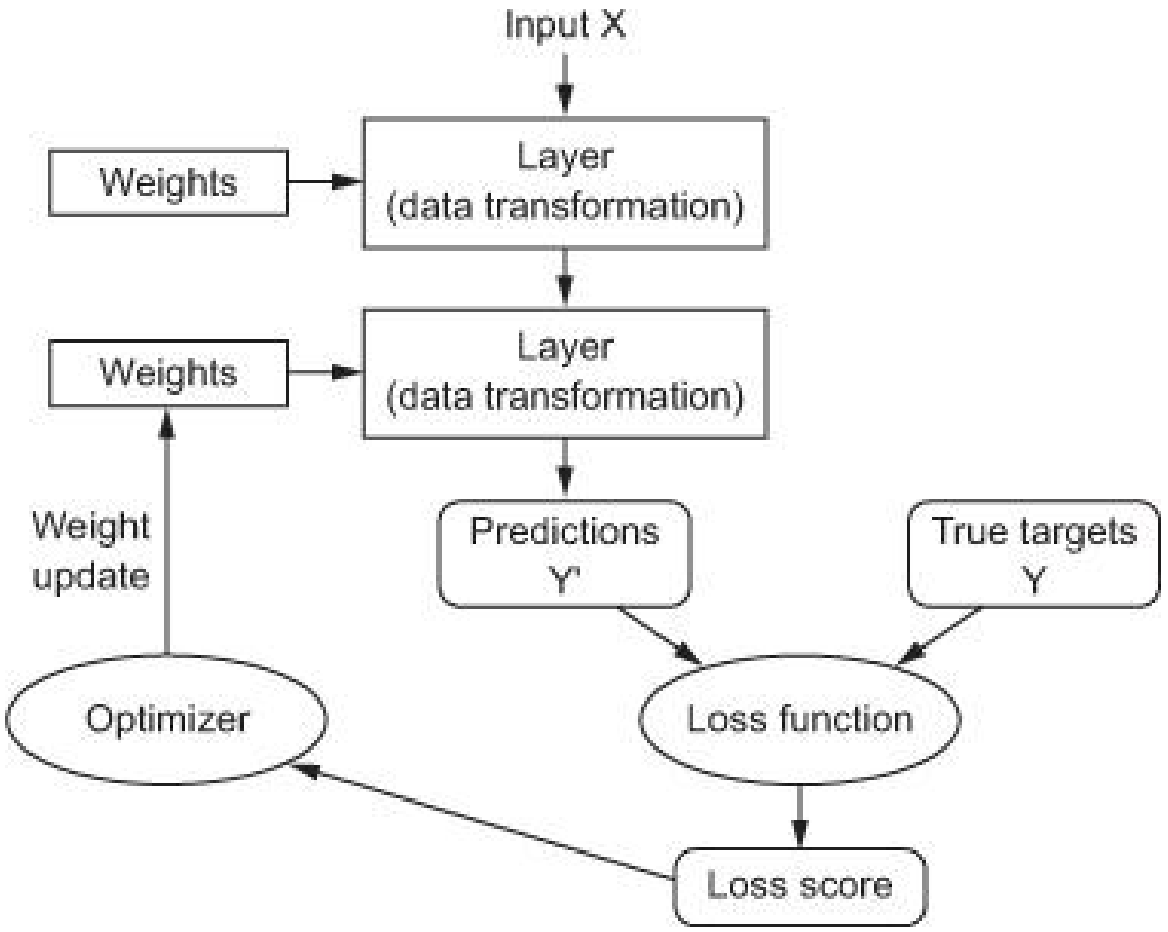


By Chabacano [GFDL or CC BY-SA 4.0], from Wikimedia Commons

Deep learning

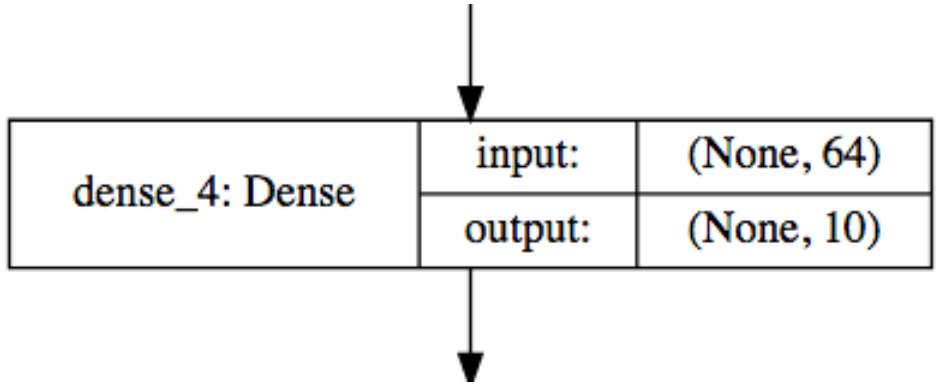
Anatomy of a deep neural network

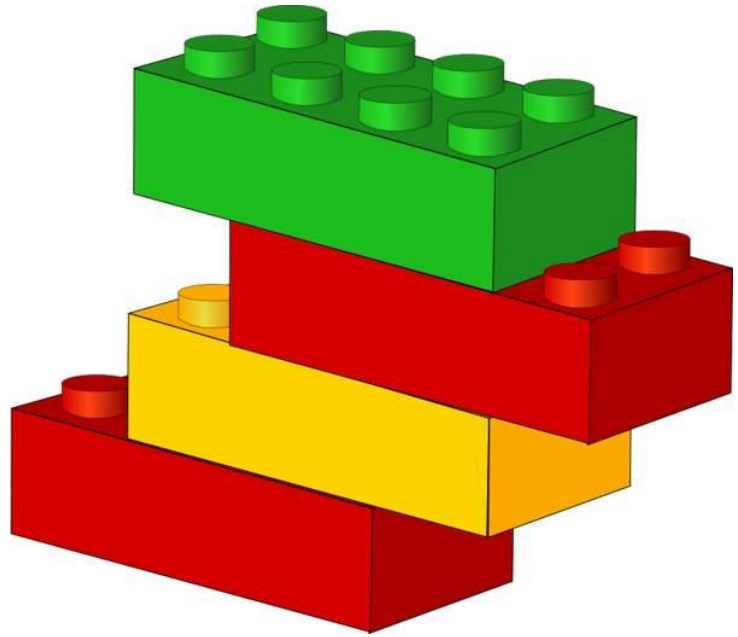
- Layers
- Input data and targets
- Loss function
- Optimizer



Layers

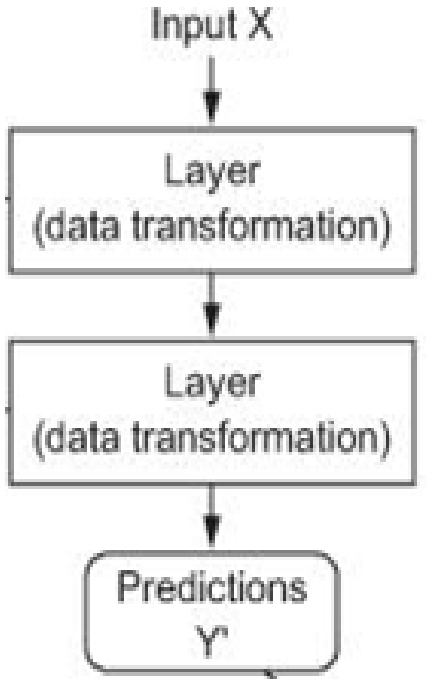
- Data processing modules
- Many different kinds exist
  - densely connected
  - convolutional
  - recurrent
  - pooling, flattening, merging, normalization, etc.
- Input: one or more tensors
  output: one or more tensors
- Usually have a state, encoded as **weights**
  - learned, initially random
- When combined, form a **network** or a **model**

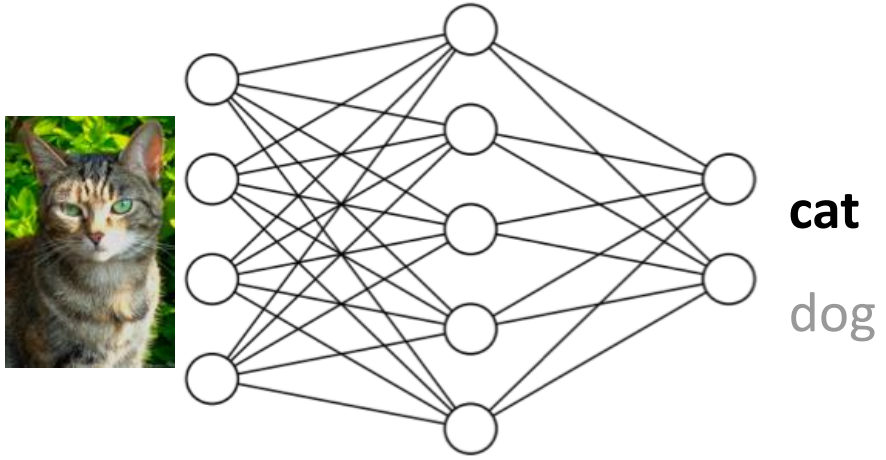




Input data and targets

- The network maps the input data X to predictions Y′
- During training, the predictions Y′ are compared to true targets Y using the loss function



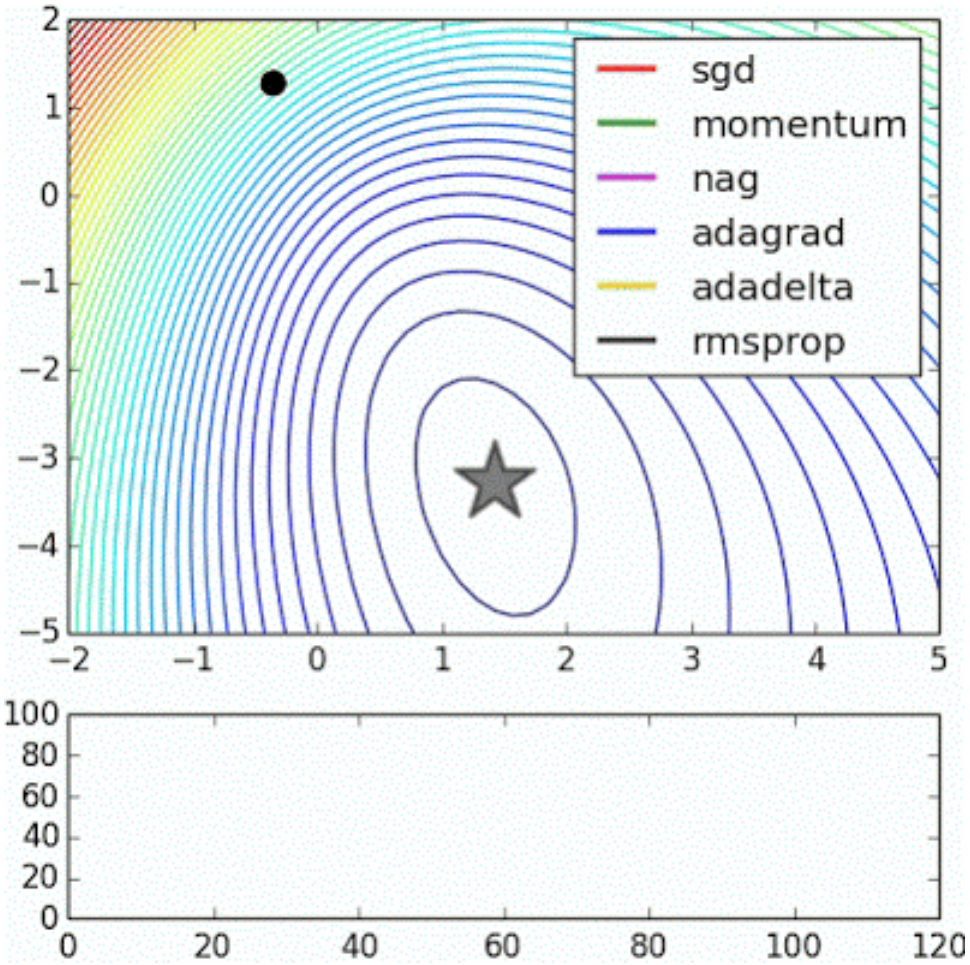


Loss function

- The quantity to be minimized (optimized) during training
  - the only thing **the network** cares about
  - there might also be other metrics **you** care about
- Common tasks have “standard” loss functions:
  - *mean squared error* for regression
  - *binary cross-entropy* for two-class classification
  - *categorical cross-entropy* for multi-class classification
  - etc.
- https://lossfunctions.tumblr.com/

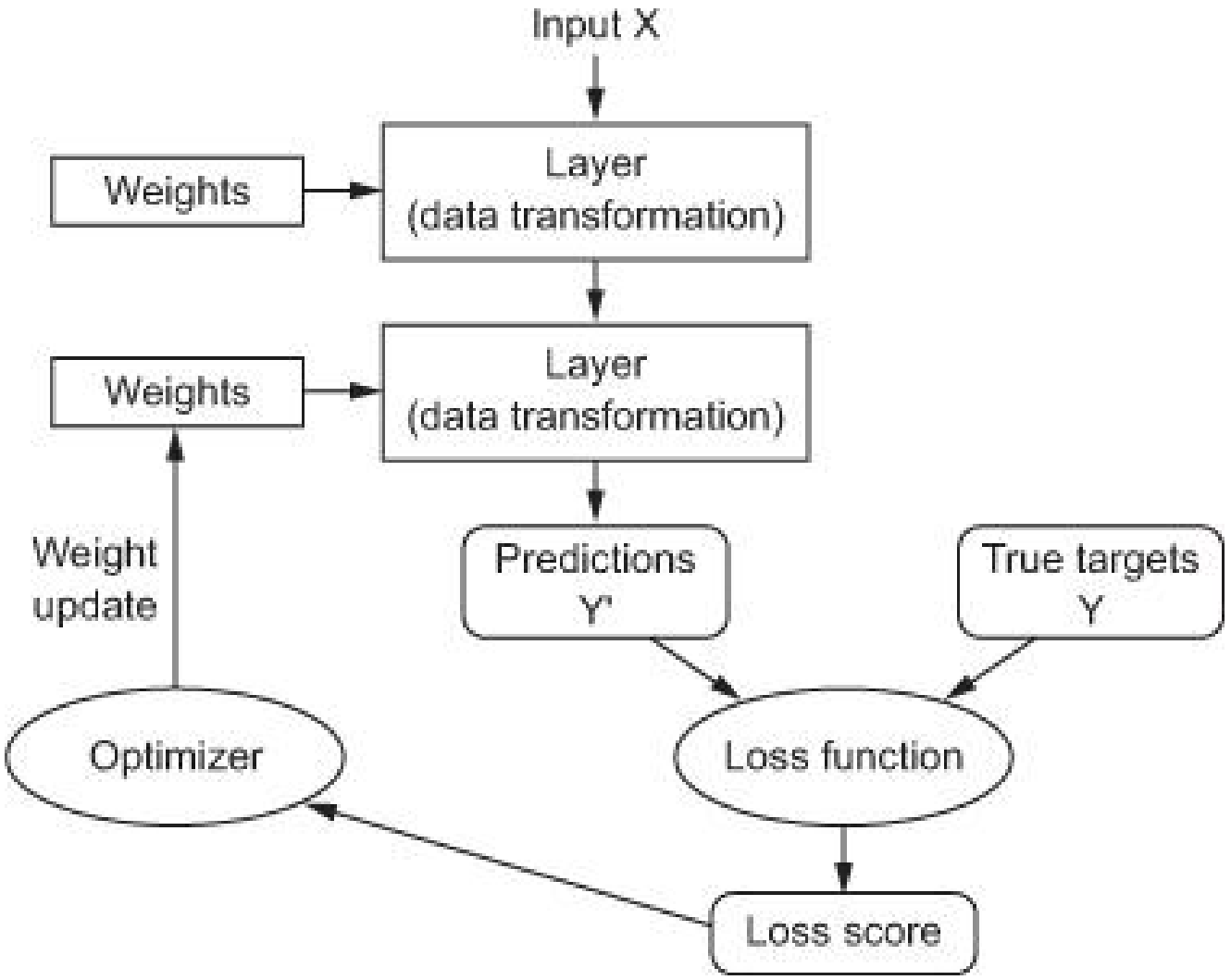
Optimizer

- How to update the weights based on the loss function
- *Learning rate (+scheduling)*
- Stochastic gradient descent, momentum, and their variants
  - RMSProp is usually a good first choice
  - more info: http://ruder.io/optimizing-gradient-descent/



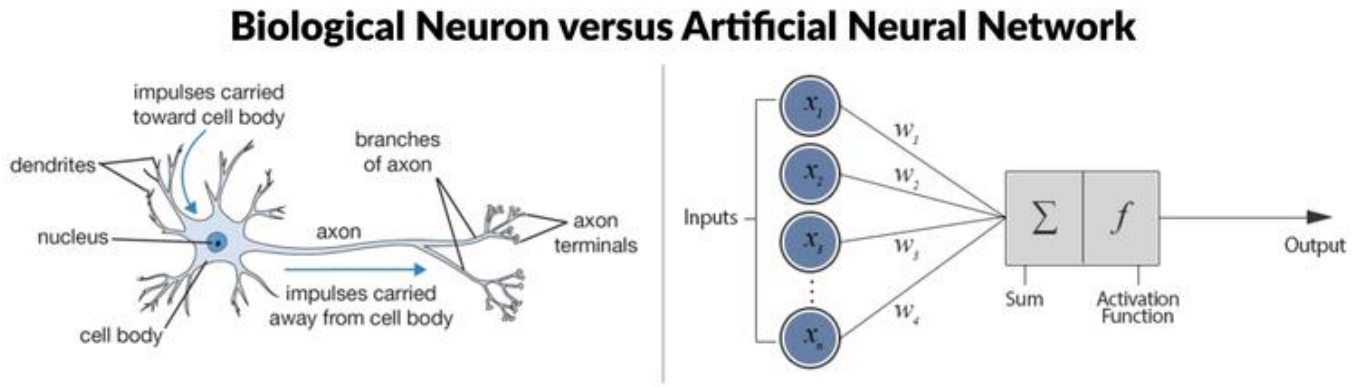
Animation from: https://imgur.com/s25RsOr

Anatomy of a deep neural network



Perceptron

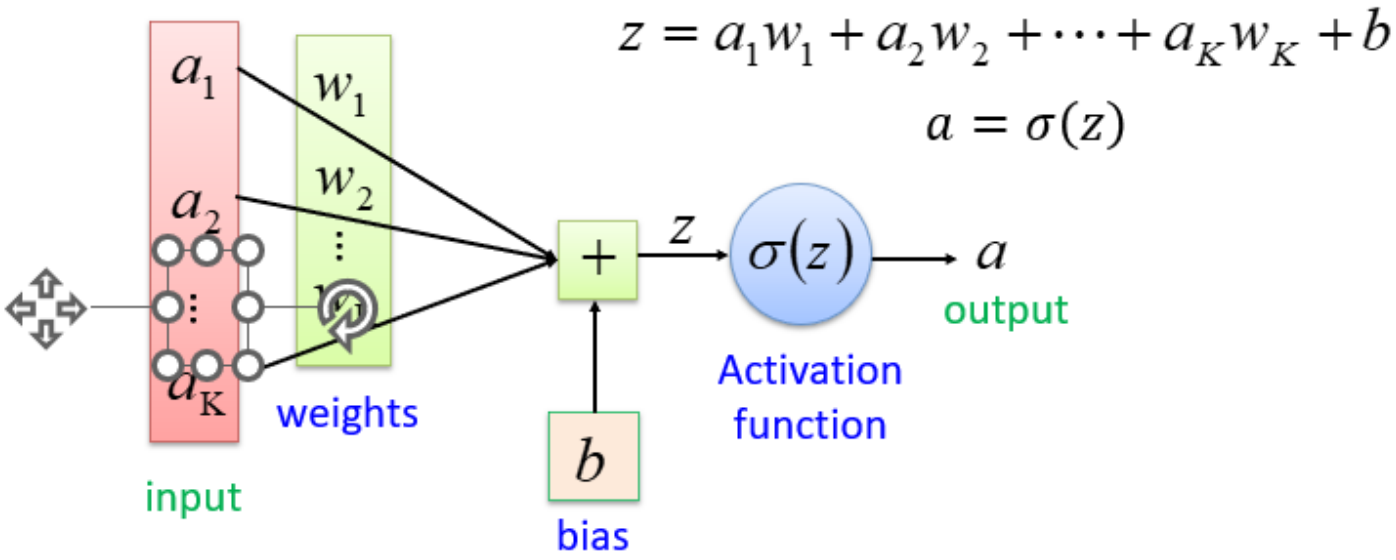
- The simplest neural network.
- Consists of a single neuron.
- Much like biological neurons
- The single artificial neuron is a simple tree structure
- Has input nodes and a single output node



Source: https://www.datacamp.com/community/tutorials/deep-learning-python

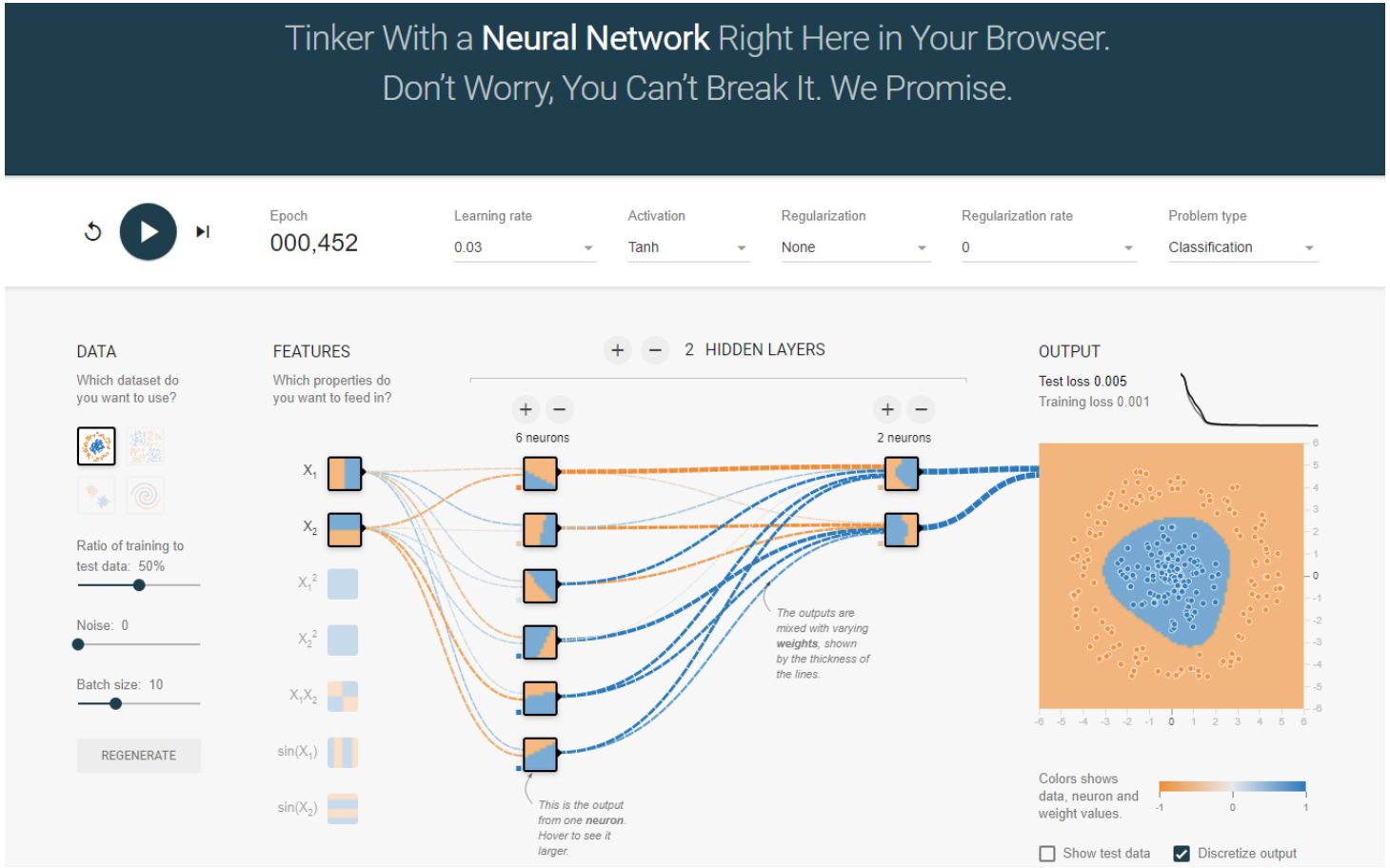
Elements of Neural Networks

- NNs consist of hidden layers with neurons (i.e., computational units)
- A single neuron maps a set of inputs into an output number, or $f: R^K \rightarrow R$



Elements of Neural Networks

- A neural network playground link



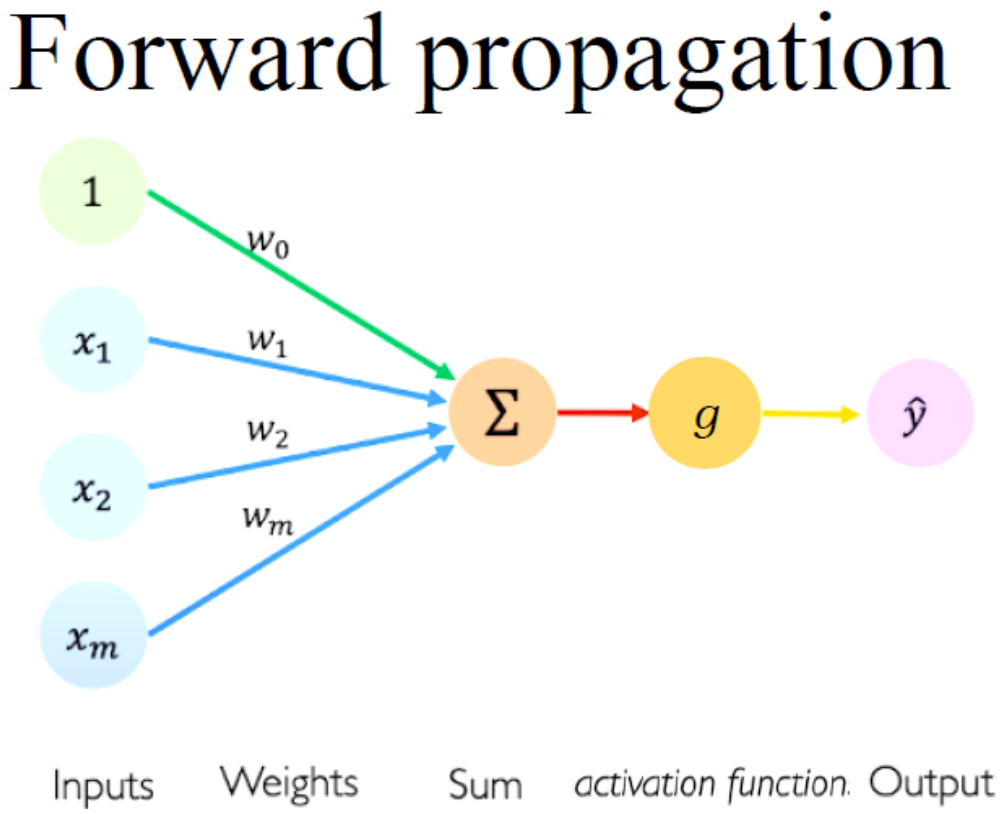
Artificial neurons components

- **Input nodes:** each input node is numerical value.
- **Connections**: each connection that departs from the input node has a weight associated with it.
- All the values of the input nodes and weights of the connections are brought together: they are used as inputs for a **weighted sum**
- This result will be the input for a **transfer or activation function**
- **Output node**: That associated with the function of the weighted sum of the input nodes.
- **Bias**: consider as the weight associated with an additional input node that is permanently set to 1.
  - The bias value is critical because it allows you to shift the activation function to the left or right, which can make a determine the success of your learning.

Source: https://www.datacamp.com/community/tutorials/deep-learning-python

Propagation

- Forward and backward propagation is very important to learn machine learning.
- Understand inside processing of model training
- Forward means moving forward with provided input and weights
- Backward is moving form output to input
- Each input in the feature vector is assigned its own relative weight
- Decides the impact that the particular input needs in the summation function



Source: http://introtodeeplearning.com/

Loss Functions and Optimization

- A loss function tells how good our current classifier
- Loss functions are one of the most important aspects of neural networks
- Directly responsible for fitting the model to the given training data.
- **Train** the process by the model maps the relationship between the training data and the outputs
- Each training input is loaded into the neural network in a process called **forward propagation**
- The model has produced an output, this predicted output is compared against the given target output in a process called **backpropagation**
- The hyperparameters of the model are then adjusted so that it now outputs a result closer to the target output.
- A **loss function** is a function that **compares** the target and predicted output values; measures how well the neural network models the training data.

Source: https://towardsdatascience.com/

Back Propagation and Optimization Function

For accurate predictions:

1- Need to minimize the calculated error

2- This is done using back propagation.

3-The current error is typically propagated backwards to a previous layer

4- Where it is used to modify the weights and bias

5- That will minimize error

6- The weights are modified using a function called Optimization Function

The components of a neural network model:

- the activation function

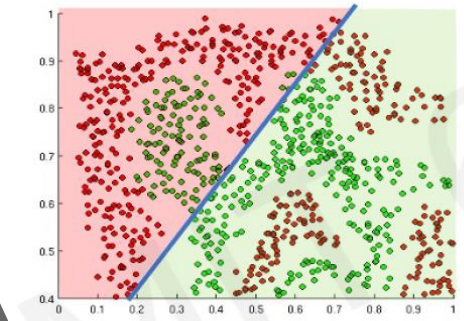
- loss function

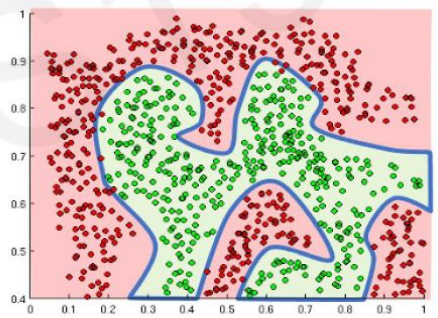
- optimization algorithm

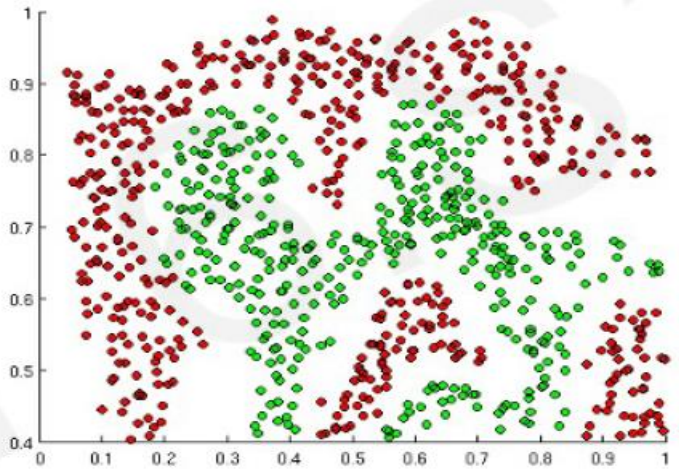
play a very important role in efficiently and effectively training a Model and produce accurate results.

Activation function

- *Function that you use to get the output of node*
- *To determine the output of neural network like yes or no*
- *It maps the resulting values in between 0 to 1 or -1 to 1*
- *Introduce nonlinearities into the network*
- The Activation Functions can be basically divided into 2 types-
  1. Linear Activation Function
  2. Non-linear Activation Functions







Activation function criteria

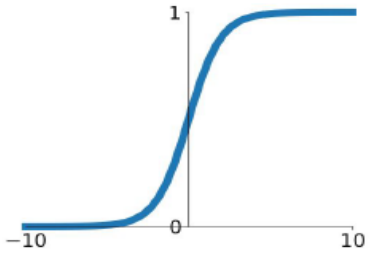
- **Vanishing Gradient problem:**
  - Neural Networks are trained using the process gradient descent.
  - The gradient descent consists of the backward propagation step which is basically chain rule to get the change in weights in order to reduce the loss after every epoch
  - The depth of the network and the activation shifting the value to zero.
- **Nonlinear function**
  - The composition of linear functions is just another linear function, gain nothing by having multiple layers
  - Use linear regression or classification.
- **Monotonic function**
  - The activation function should be either entirely non-increasing or non-decreasing.
  - If the activation function isn't monotonic then increasing the neuron's weight might cause it to have less influence on reducing the error of the cost function.
- **Quickly converging**
  - the activation function should swiftly reach its desired value.
- **Zero-centered**
  - Output of the activation function should be symmetrical at zero so that the gradients do not shift to a particular direction.
- **Computational Expense**
  - Activation functions are applied after every layer and need to be calculated millions of times in deep networks.
- **Differentiable**
  - The activation function should be differential because we want to calculate the change in error with respect to given weights at the time of gradient descent.
  - Neural networks are trained using the gradient descent process, hence the layers in the model need to differentiable

Source: https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253

Activation function

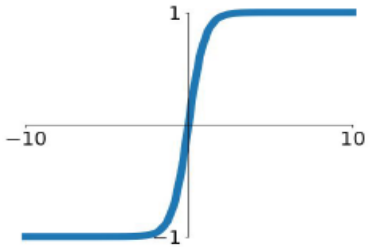
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



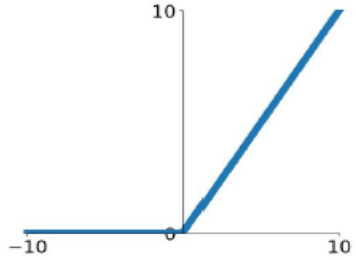
tanh

$$\tanh(x)$$



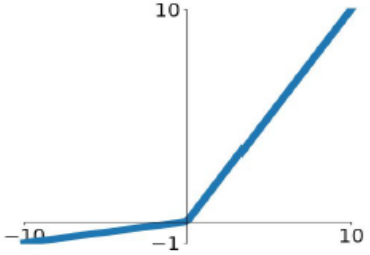
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

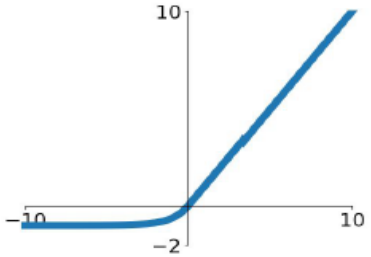


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

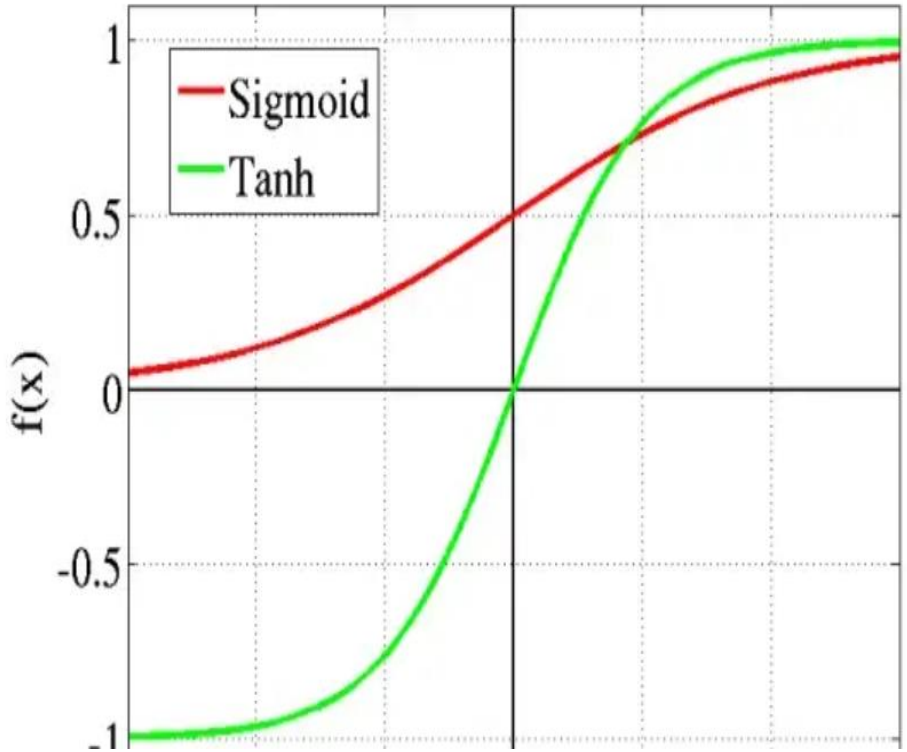
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Source: http://cs231n.stanford.edu/

Sigmoid vs Tanh



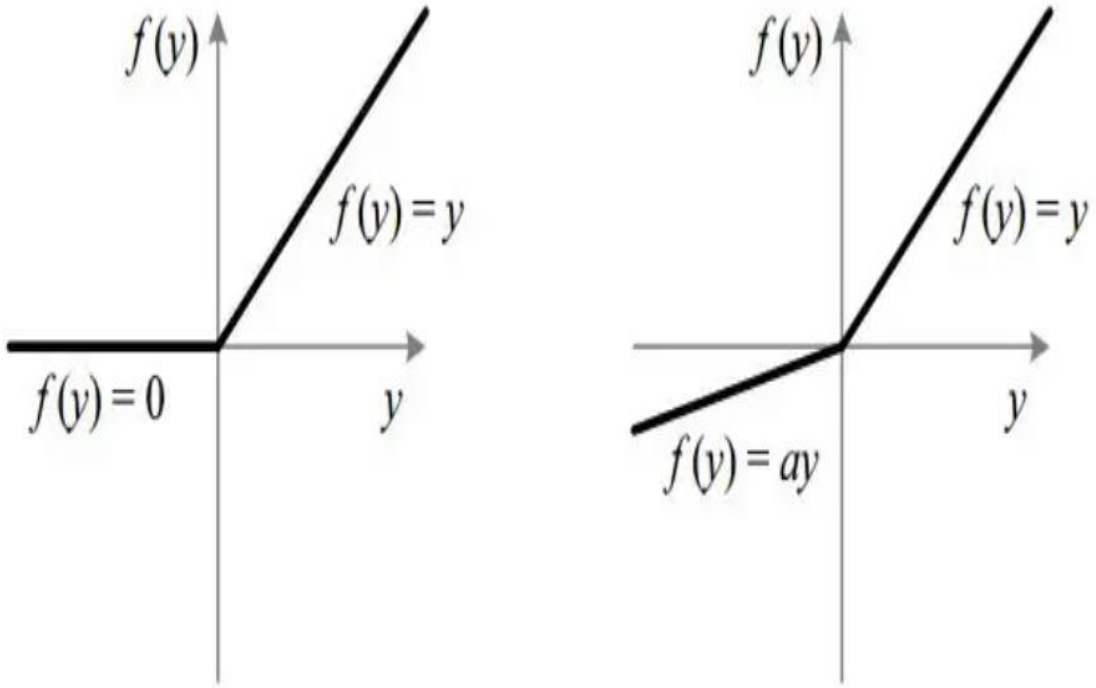
Sigmoid

- It is between (0 to 1)
- predict the probability
- The function is differentiable
- Sigmoid outputs are not zero-centered

Tanh

- Is like sigmoid
- Rang is (-1,1)
- the negative inputs will be mapped strongly negative
- The function is differentiable
- Used in classification between two classes
- zero centered

ReLU vs Leaky ReLU



ReLU

- The ReLU is the most used activation function
- is used in almost all the convolutional neural networks or deep learning
- Range: [0 to infinity)
- all the negative values become zero immediately
- decreases the ability of the model to fit or train from the data properly
- Converges much faster than sigmoid/tanh in practice
- Not zero-centered output

Leaky ReLU

- Solve ReLU issue for negative values
- Range: (-infinity to infinity)
- Computationally efficient
- Computation requires exp()

Loss function

The loss functions compute the quantity that a model should seek to minimize during training.

Loss functions are helpful to train a neural network
Given an input and a target, they calculate the loss

In supervised learning, there are two main types of loss functions

- Regression Loss Functions :
  used in regression neural networks
  given input value then the model predicts output value
  Ex. Mean Squared Error, Mean Absolute Error
- Classification Loss Functions:
  used in classification neural networks
  given an input, the neural network categories this input
  then select the category with the highest probability of belonging
  Ex. Binary Cross-Entropy, Categorical Cross-Entropy

```
from tensorflow.keras.losses import mean_squared_error
model.compiile(loss=mean_squared_error, optimizer='sgd')
```

Available losses

Probabilistic losses

- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- Poisson class
- binary_crossentropy function
- categorical_crossentropy function
- sparse_categorical_crossentropy function
- poisson function
- KLDivergence class
- kl_divergence function

Regression losses

- MeanSquaredError class
- MeanAbsoluteError class
- MeanAbsolutePercentageError class
- MeanSquaredLogarithmicError class
- CosineSimilarity class
- mean_squared_error function
- mean_absolute_error function
- mean_absolute_percentage_error function
- mean_squared_logarithmic_error function
- cosine_similarity function
- Huber class
- huber function
- LogCosh class
- log_cosh function

Source: https://keras.io/api/losses/

Mean Squared Error (MSE)

- One of the most popular loss functions
- Finds the average of the squared differences between the target and the predicted outputs
- The difference is squared, which means it does not matter whether the predicted value is above or below the target value
- MSE is highly sensitive to outliers

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

Mean Absolute Error (MAE)

- MAE finds the average of the absolute differences between the target and the predicted outputs.
- As an alternative to MSE in some cases
- MAE is used in cases when the training data has a large number of outliers

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y^{(i)} - \hat{y}^{(i)}|$$

Binary vs Categorical Cross-Entropy

- Loss function used in binary classification models
- The input needs to classify it into one of two pre-set categories.
- Categorical cross-entropy in multiclass classification

Optimization Algorithms

- Optimization Algorithms are used to update weights and biases
- The internal parameters of a model to reduce the error
- Optimization functions:
  1. Stochastic Gradient Decent
     - calculates gradient for the whole dataset and updates values in direction opposite to the gradients until we find a local minima.
     - Stochastic Gradient Descent, or SGD for short, is an optimization algorithm used to train machine learning algorithms, most notably artificial neural networks used in deep learning
     - much faster
  2. Adagrad
     - more preferrable for a sparse data set as it makes big updates for infrequent parameters and small updates for frequent parameters
     - uses a different learning Rate for every parameter
  3. Adam
     - stands for Adaptive Moment Estimation
     - also calculates different learning rate
     - faster, and outperforms other techniques

Available optimizers

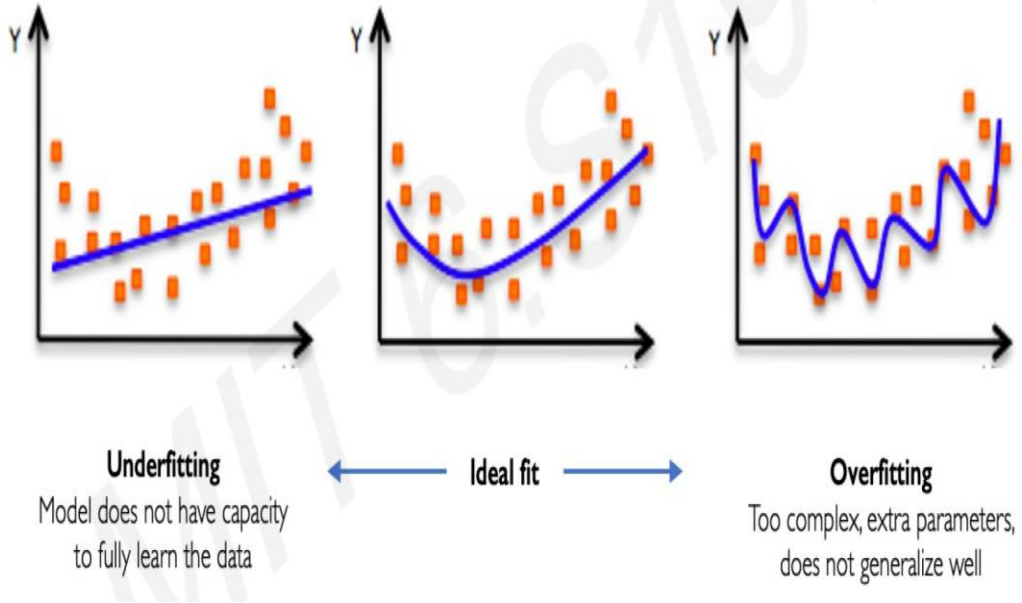
- **Available optimizers**
- SGD
- RMSprop
- Adam
- AdamW
- Adadelta
- Adagrad
- Adamax
- Adafactor
- Nadam
- Ftrl

- Source: https://keras.io/api/optimizers/

Sample, batch size, Epoch, iteration

- **A sample**
  - is a single row of data.
  - It contains inputs that are fed into the algorithm and an output that is used to compare to the prediction and calculate an error.
  - A training dataset is comprised of many rows of data (many samples).
  - A sample may also be called an instance, an observation, an input vector, or a feature vector.
- **The batch size**
  - is a hyperparameter that defines the number of samples to work through before updating the internal model parameters.
  - is a number of samples processed before the model is updated.
  - The size of a batch must be more than or equal to one and less than or equal to the number of samples in the training dataset
- **number of iterations**
  - number of passes, each pass using [batch size] number of examples
- **Epoch**
  - is comprised of one or more batches.
  - One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters.
  - The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.
  - The number of epochs can be set to an integer value between one and infinity
  - one **epoch**= one forward pass and one backward pass of all the training examples

Overfitting Vs Underfitting



- Underfitting is when the training error is high.

- Underfitting means that your model makes accurate, but initially incorrect predictions

- Train error is large and val/test error is large too.

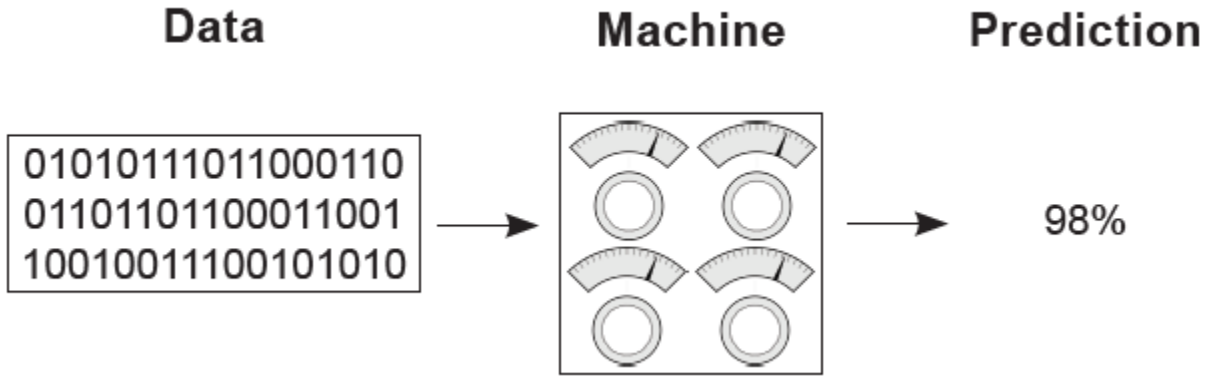
- Overfitting is when the testing error is high compared to the training error, or the gap between the two is large.

- Overfitting means that your model makes not accurate predictions.
- Train error is very small and val/test error is large

Source: https://towardsdatascience.com/overfitting-and-underfitting-principles-ea8964d9c45c

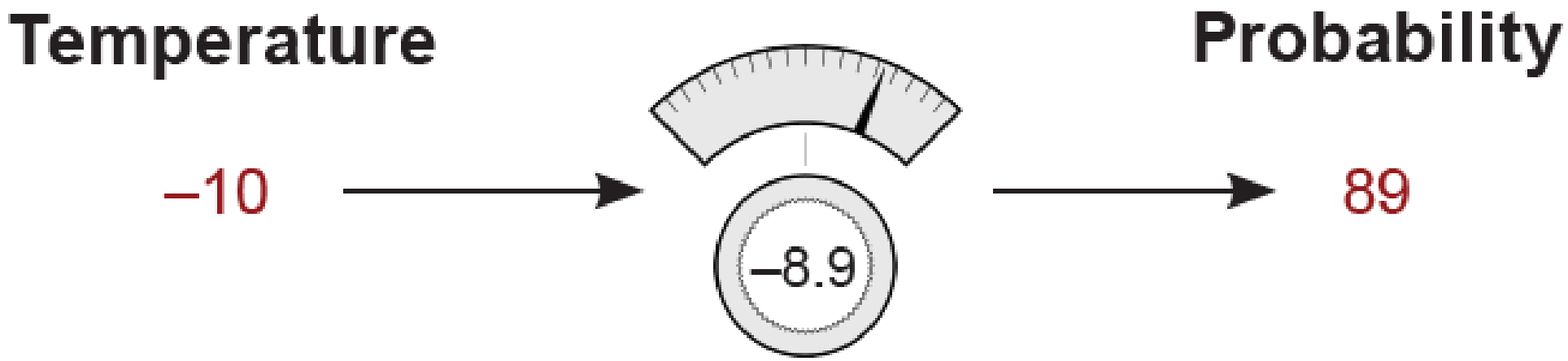
Supervised learning

- **Step 1: Predict**
- **Step 2: Compare to the truth pattern**
  Pred: 98% > **Truth**: 0%
- **Step 3: Learn the pattern**



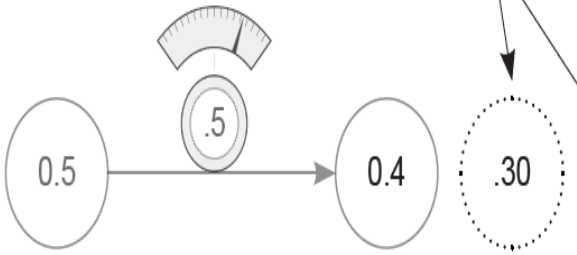
Source: Grokking Deep Learning First Edition by Andrew Trask

Negative number prediction



measure the error

- The goal of training a neural network is to make correct predictions
- Measuring error simplifies the problem




```
knob_weight = 0.5
input = 0.5
goal_pred = 0.8

pred = input * knob_weight

error = (pred - goal_pred) ** 2

print(error)
```

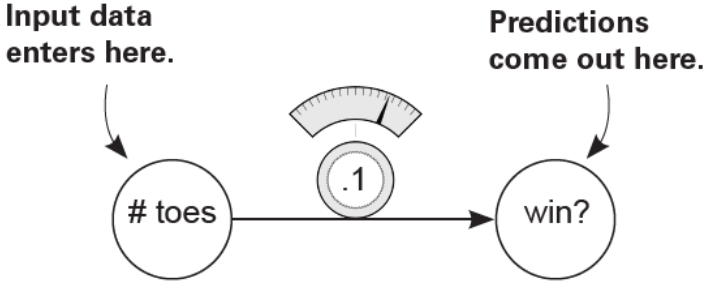
The error is a way to measure how much you missed. There are multiple ways to calculate error, as you'll learn later. This one is *mean squared error*.

Raw error

Forces the raw error to be positive by multiplying it by itself. Negative error wouldn't make sense.

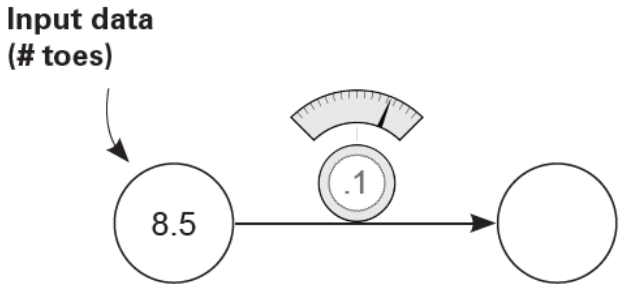
simplest neural network possible

❶ An empty network




```
weight = 0.1
def neural_network(input, weight):
    prediction = input * weight
    return prediction
```

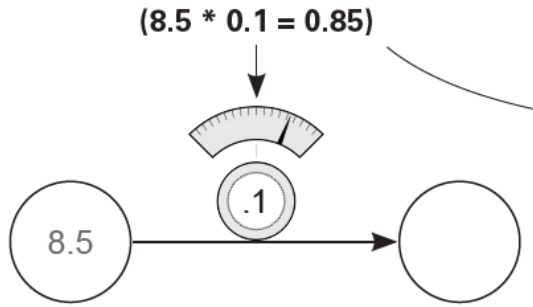
❷ Inserting one input datapoint




```
number_of_toes = [8.5, 9.5, 10, 9]
input = number_of_toes[0]
pred = neural_network(input,weight)
print(pred)
```

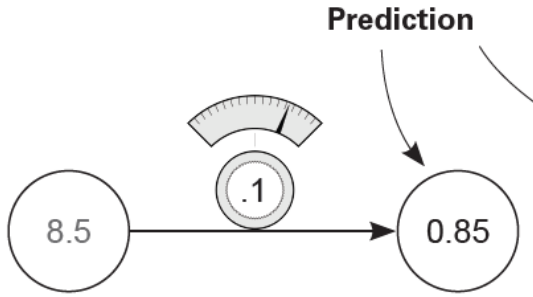
simplest neural network possible

❸ **Multiplying input by weight**




```
def neural_network(input, weight):
    prediction = input * weight
    return prediction
```

❹ **Depositing the prediction**




```
number_of_toes = [8.5, 9.5, 10, 9]
input = number_of_toes[0]
pred = neural_network(input,weight)
```

Supervised Learning Overview

- At a high level, machine learning can be described as building algorithms that can uncover or “learn” *relationships* in data.
- Supervised learning can be described as the subset of machine learning dedicated to finding relationships *between characteristics of the data that have already been measured.*
- *This done by* choosing one characteristic that we want to predict from the others;
- This characteristic called *target*

$$y_i = \beta_0 + \beta_1 \times x_1 + \ldots + \beta_n \times x_k + \epsilon$$

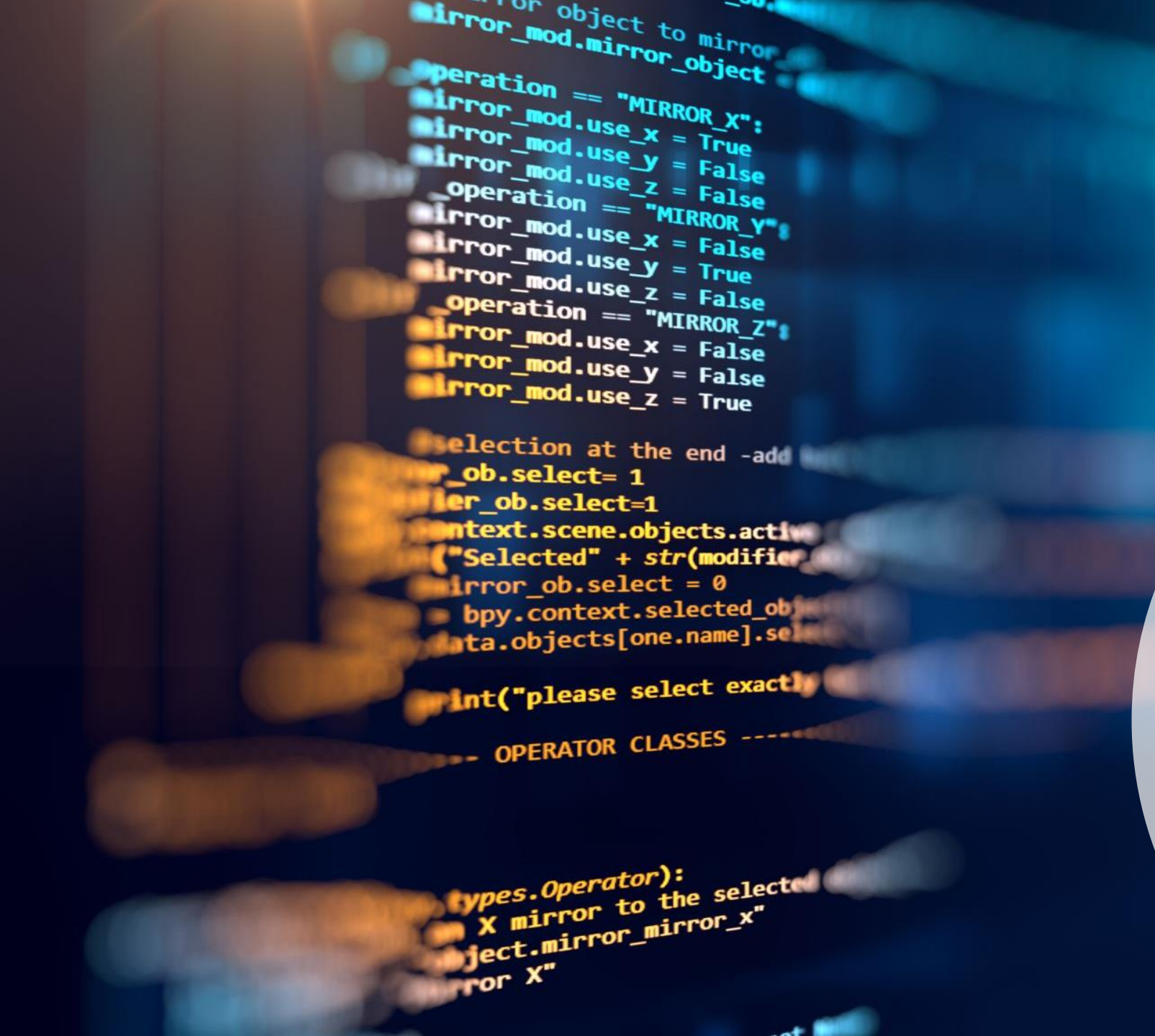
- Linear Regression
- Train a model At a high level, models take data, combine them with *parameters* in some way, and produce predictions

Train the Model

The following procedure over and over again:

1. Select a batch of data.
2. Run the forward pass of the model.
3. Run the backward pass of the model using the info computed on the forward pass.
4. Use the gradients computed on the backward pass to update the weights.

Coding time


mirror_mod.mirror_object
operation == "MIRROR_X":
irror_mod.use_x = True
irror_mod.use_y = False
irror_mod.use_z = False
operation == "MIRROR_Y":
irror_mod.use_x = False
irror_mod.use_y = True
irror_mod.use_z = False
operation == "MIRROR_Z":
irror_mod.use_x = False
irror_mod.use_y = False
irror_mod.use_z = True
selection at the end -add
_ob.select= 1
er_ob.select=1
ntext.scene.objects.acti
("Selected" + str(modifier
irror_ob.select = 0
bpy.context.selected_ob
ata.objects[one.name].se
int("please select exactly
OPERATOR CLASSES
types.Operator):
X mirror to the selected
ject.mirror_mirror_x"
ror X"

Importing Python library

```
import numpy as np
import pandas as pd
import tensorflow
from tensorflow import keras
from tensorflow.keras import models
from tensorflow.keras import layers
from numpy import ndarray
from typing import Callable, Dict, Tuple, List
import sklearn
```

Importing the file

- `from sklearn.datasets import load_boston`
- `b = load_boston()`
- `print(len(b.target))`
- `print(b.data.shape)`
- `data = np.hstack((b.data, b.target.reshape(-1, 1)))`
- `df = pd.DataFrame(data, columns=b.feature_names.tolist() + ['TARGET'])`
- `df.head(10)`

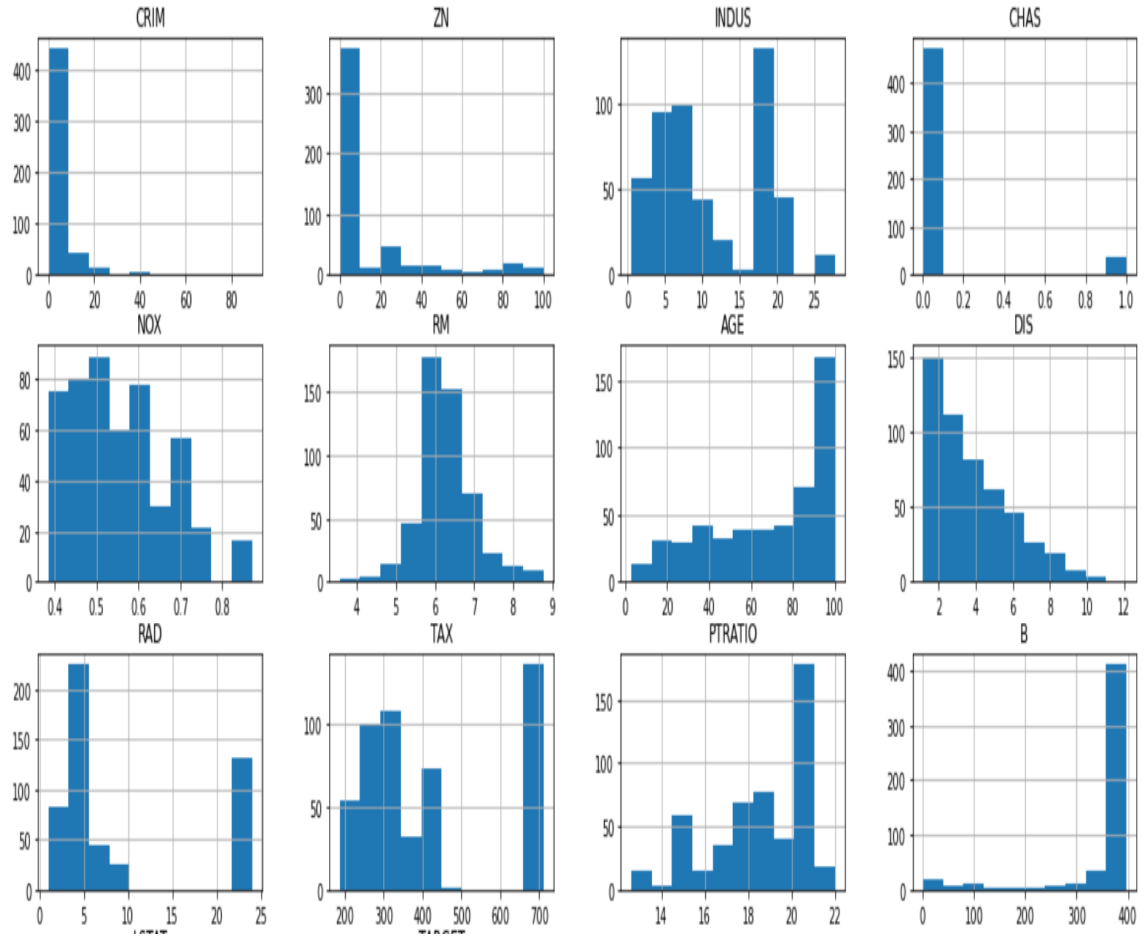
```
506
(506, 13)
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	TARGET
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21	28.7
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43	22.9
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15	27.1
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93	16.5
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	17.10	18.9

Check null values

- `df.isnull().sum()`
- `df.skew()`
- `df.hist(figsize=(20,10))`



Visualization

```
import seaborn as sns
%matplotlib inline
import matplotlib.pyplot as plt

for c in b.feature_names:
    plt.figure(figsize=(20,10))
    sns.boxplot(df[c], showfliers=True)
    plt.show()
```

Train the model

- ```
  from sklearn.model_selection import train_test_split
  ```
- ```
  from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler
  ```
- ```
  X, test, y, y_test = train_test_split(df[b.feature_names], df['TARGET'], test_size=.1)
  ```
  ```

scaling

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
test = scaler.transform(test)
X
```

- Scaling is a technique to make them closer to each
- Scaling is used for making data points generalized so that the distance between them will be lower
- Scale data for better performance of Machine Learning Model
- One of the most critical steps during the pre-processing of data before creating a machine learning model.
- Most common techniques of feature scaling are Normalization and Standardization.

Building the model

```
def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu', input_shape=(len(b.feature_names),)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model

model = build_model()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                896

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 5,121
Trainable params: 5,121
Non-trainable params: 0
_________________________________________________________________
```

Model Evaluation

- `from keras.wrappers.scikit_learn import KerasRegressor`
- `from sklearn.model_selection import KFold`
- `from sklearn.model_selection import cross_val_score`
- `model = KerasRegressor(build_fn=build_model, epochs=25, batch_size=8, verbose=1)`

execution

- `all_scores = []`
- `model.fit(X, y)`
- `scores = model.score(test, y_test)`
- `scores`

```
57/57 [==============================] - 0s 2ms/step - loss: 11.8026 - mae: 2.3289
Epoch 16/25
57/57 [==============================] - 0s 2ms/step - loss: 11.4420 - mae: 2.2813
Epoch 17/25
57/57 [==============================] - 0s 3ms/step - loss: 11.1750 - mae: 2.2740
Epoch 18/25
57/57 [==============================] - 0s 2ms/step - loss: 11.0956 - mae: 2.2679
Epoch 19/25
57/57 [==============================] - 0s 2ms/step - loss: 10.9232 - mae: 2.2200
Epoch 20/25
57/57 [==============================] - 0s 3ms/step - loss: 10.8361 - mae: 2.1856
Epoch 21/25
57/57 [==============================] - 0s 3ms/step - loss: 10.3575 - mae: 2.1875
Epoch 22/25
57/57 [==============================] - 0s 2ms/step - loss: 10.5158 - mae: 2.1499
Epoch 23/25
57/57 [==============================] - 0s 3ms/step - loss: 10.1584 - mae: 2.1578
Epoch 24/25
57/57 [==============================] - 0s 2ms/step - loss: 9.7839 - mae: 2.1246
Epoch 25/25
57/57 [==============================] - 0s 2ms/step - loss: 9.7088 - mae: 2.0710
7/7 [==============================] - 0s 2ms/step - loss: 6.7919 - mae: 2.1691

-6.791856288909912
```

prediction

- `predictions = model.predict(test)`
- `predictions`

```
array([23.748146, 28.633387, 15.670339, 26.745   , 23.92444 , 15.078281,
       20.488218, 25.405684, 13.691667, 24.615881, 48.448627, 20.901793,
       18.165346, 25.325342, 26.685385, 19.682386, 22.299126, 30.469011,
       36.16539 , 49.855328, 18.795685, 18.359104, 33.85078 , 13.499032,
       20.431002, 20.56697 , 29.047125, 12.253085, 26.042486, 21.013142,
       25.033255, 14.080876, 12.298603, 32.489273, 24.402313, 26.928602,
       24.55119 , 13.963896, 20.08138 , 39.75794 , 16.702135, 25.06964 ,
       40.741142, 19.66429 , 13.316799, 29.909609, 22.912773, 34.556763,
       16.448988, 21.457369,  9.156757], dtype=float32)
```

Prediction graph

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.scatter(y_test.tolist(), predictions)
plt.plot(range(75), range(75))
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Actual Rent vs Predicted Rent")

plt.show()
```

Thank You