Dispensa Intro to Machine Learning

Bonmassar Ivan

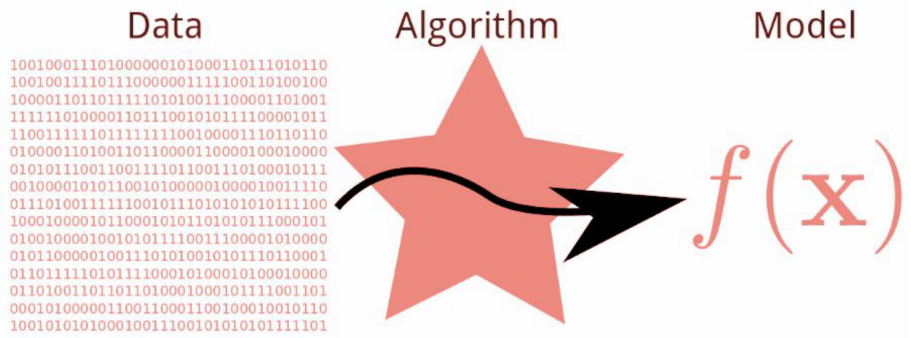
July 4, 2022

Contents

Chapter 1

ML Basics

The main notion to get from this is the following. ML allows computers to gain **knowledge** acquired through **algorithms** by learning from data. This knowledge is represented through a **model** which is then used on future data.



The training data produces a model or predictor, whereas the testing data produce a prediction.

1.1 Data

Data can be a list of movies from IMDB which is easily representable. Although we need to use **features** when taking into consideration other examples. Features is how an algorithm view data and is generally represented with vectors.

For example, classifying different apples, features could be the shape and colour of them.

The general problem with data for classification for example is that not all data is the same. For instance a banana can either be green or yellow. Although this is true we cannot go to deep with this so we use a probabilistic model called **data generating distribution**. Both training and test data are based on this.

So in our previous example, we will generalize and say that bananas are yellow.

Definition 1.1.1 (Probability distribution)**.** Describes how likely certain events are.

High probability: round apples
Low probability: square apples

1.2 Types of Learning

SUPERVISED LEARNING
Supervised learning is when the algorithm is given labeled examples and the predictor should output a label. A further example of this is **classification**, where the model classifies from a pool of categories.

Given a training set $T = (x_i, y_i)$ learn a function that predicts y given x. x is multi-dimensional.

Some real world examples can be facial recognition, spam detection and character recognition.

Regression is similar to classification only with real values (i.e. numbers).

Ranking the label is a ranking (most similar, most popular web pages etc.)

UNSUPERVISED LEARNING The given data is without labels. Some examples are:

Clustering, where the output is the general structure of the data set (clusters of data). Real world examples are image segmentation, social network analysis

Anomaly detection

Dimensionality reduction

REINFORCEMENT LEARNING

The idea is that the agent interacts with the environment and receives rewards based on behavior.

1.3 ML Ingredients

TASK

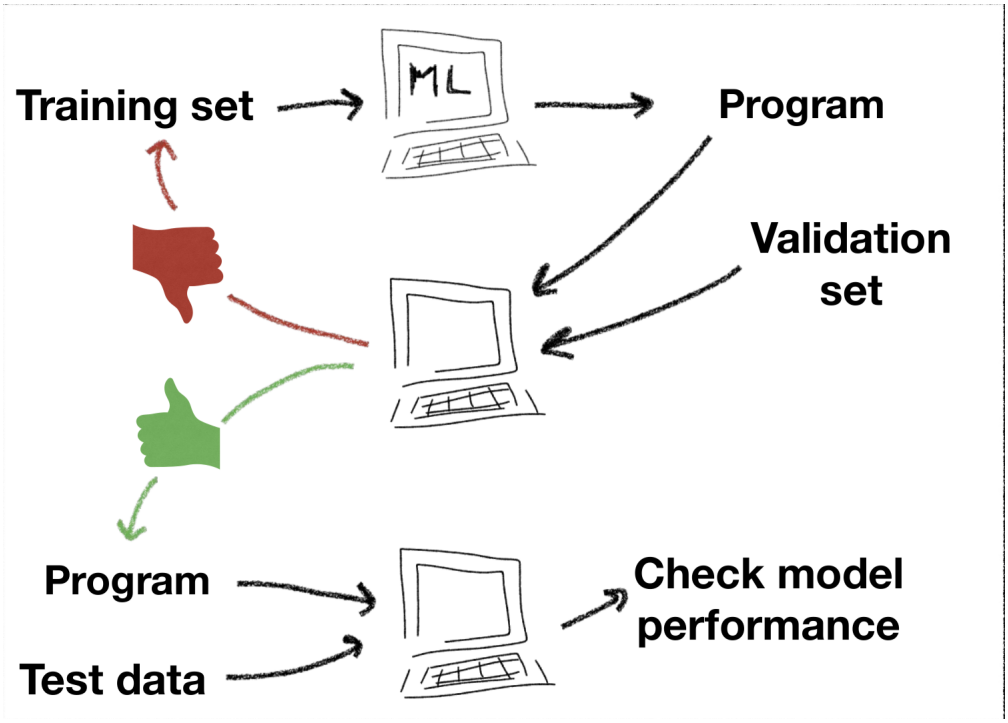
A task represents the type of prediction being made to solve a problem.

Assigning each input $x \in \mathscr{X}$ to an output $y \in \mathscr{Y}$

Data

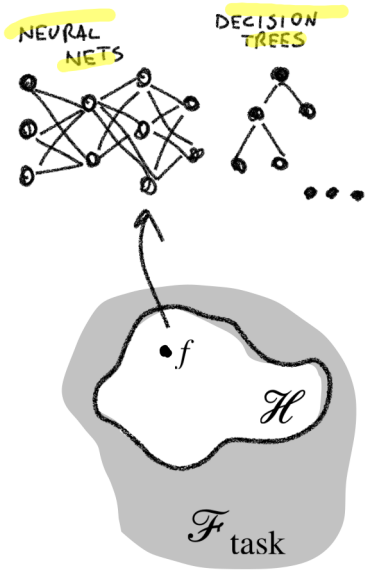
Data is basically the information required to solve a specific problem and as said previously is usually sampled from an unknown data generating distribution :

$\mathbf{p}_{data}$
For classification and regression $\mathbf{p}_{data} \in \triangle(\mathscr{X} \times \mathscr{Y})$



Model and hypothesis space

A model is like a program that solves the problem. There are various models (decision trees, neural networks..) and a set of them makes up the hypothesis space.



The objective

The objective is to minimize an error function $E(f, \mathbf{p}_{data})$ to find the optimal function

$f^\star = \arg\min E(f, \mathbf{p}_{data})$.

This however is really hard to do because of the too large search space.

The *feasible target* is the optimal one in a restricted hypothesis space $\mathscr{H}$

$f^\star_{\mathscr{H}} = \arg\min E(f, \mathbf{p}_{data})$.

This is also not doable because we do not have access to $\mathbf{p}_{data}$

The *actual target* is then the following:

$f^\star_{\mathscr{H}}(\mathscr{D}_n) = \arg\min E(f, \mathscr{D}_n)$. where $\mathscr{D}_n$ is a training set.

The error function is usually specified as a pointwise-loss (f, z) measuring the error of f on the training set z.

The learning algorithm solves the optimization problem which targets the actual target.

Chapter 2

K-Nearest Neighbor

Imagine assigning each feature a numeric value, and putting them as coordinates in a feature space. To find which label to give to an example d we can see the k-nearest neighbors. The correct label will be the one that has the majority in k neighbors.

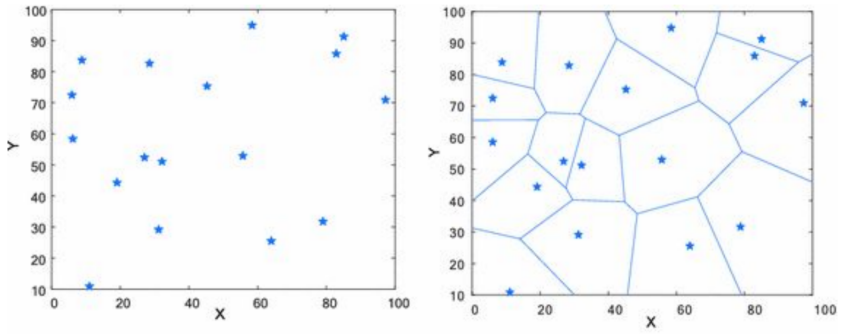
In cases where features are comparable (each feature has the same units) we will use the simple method of Euclidean Distance:

$$D(a, b) = \sqrt{(a_1 - b_1)^2 + (a_n - b_n)^2}$$

2.1 Decision boundaries

Definition 2.1.1 (Decision boundaries)**.** Decision boundaries are places where the the classification of an example changes.

These boundaries are a subset of the Voronoi diagram which divides the initial points distribution by equidistant lines.



(a) Initial points distribution **(b)** Voronoi diagram by initial points

Choosing K will have an effect on underfitting and overfitting.

Definition 2.1.2 (Underfitting)**.** Underfitting occurs when the model is not able to achieve a low error value on the training set

Definition 2.1.3 (Overfitting)**.** Overfitting occurs when the gap between training set error and test set error is large

Some heuristics come into play. It's important to choose an odd number to avoid ties. A general rule of thumb is that k should be lower than the square root of N training examples.
Weighted k-nn, uses the same mechanics only that the examples are weighted which means that they will weigh in more in a vote situation.

Lazy learning is when an algorithm simply stores data and operates when given a test example.

Eager learning on the other hand is when given a training set, the algorithm constructs a classification model which then reutilizes.

2.2 Summary

When is it useful:

- Few features per instance
- Lots of training data

Advantages:

- Training is very fast (lazy)
- Learn complex functions

Disadvantages:

- Slow at query time

The main issue has to do with dimensionality. Every algorithm requires a dense data set, although K-NN requires to have at least one neighbor in each dimension, which makes for a very heavy computational load.

Chapter 3

Linear Models

3.1 Model assumptions

Assumptions (when correct) are a great tool to improve an algorithm. They can however lead to overfitting. For K-NN the only assumption that's made is the fact that proximity relates to label.

Definition 3.1.1 (Bias)**.** The bias of a model is how strong its assumptions are.

K-NN and Decision Trees are considered to be low bias.

3.2 Linear models

A high-bias assumption is linear separability, which means that the classes can be simply divided by a line (or hyperplane in n dimensions).

A line is defined by a pair of values:

$$0 = w_1 f_1 + w_2 f_2$$

To classify we can input the values into the equation. If it's positive it will be above the line and viceversa.

In n-dimensions:

$$0 = b + \sum_{i=1}^{n} w_i f_i$$

3.3 Online learning

This linear model is different because it sees one example at a time. This is useful when taking in data streams (for example from online resources).

The algorithm receives an unlabeled example, it predicts and if it is not correct it updates its model.

3.4 Perceptron algorithm

This is a linear model that constantly updates when not correct.

repeat until convergence (or for some # of iterations):

for each training example (f_1, f_2, ..., f_n, label):

$$prediction = b + \sum_{i=1}^{n} w_i f_i$$

if not correct, update all the weights:

for each w_i:

$w_i = w_i + f_i$*label

$b = b$ + label

Chapter 4

Multiclass Classification

Definition 4.0.1 (Binary classification)**.** Given an input $\mathscr{X}$, an unknown distribution D over $\mathscr{X} \times \{-1, 1\}$ a training set D compute a function f.

Definition 4.0.2 (Multiclass classification)**.** Given an input $\mathscr{X}$ and a number K of classes, an unknown distribution D over $\mathscr{X} \times K$ a training set D compute a function f.

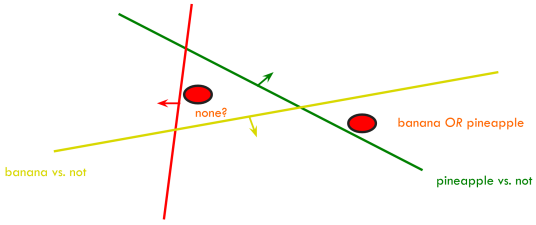
None of the current methods work.

The main idea is to modify the perceptron by adding multiple lines. There are two approaches to this.

4.1 One vs All (OVA)

For each label L define a binary problem. All examples with L are positive, the rest is negative.

This poses a problem related to some areas in which it can either be two classes or none at all.



How to classify?

Generally speaking the classifier should provide a level of confidence. To calculate this value we need to use decision boundaries and its distance from the hyperplane.

4.2 All vs All (AVA)

The idea behind AVA is to pair up each permutation of 1v1 pairs. Then, the classifier will receives all examples of i as a +1 and -1 for the j class. When a test point arrives we evaluate on all the F_{ij} classifiers.

4.3 Summary

AVA has faster training time but a slower test time. AVA has more chances of error at test time.

Evaluation comes when we need to evaluate a specific performance of an algorithm. Can be done through microaveraging, average over the examples, or macroaveraging, average of the average of each label.

Chapter 5

Gradient Descent

5.1 Model based machine learning

There are 3 steps to this model:

1. Pick a model (DT, perceptron)
2. Pick a criterion to optimize
3. Develop a learning algorithm

In linear model:

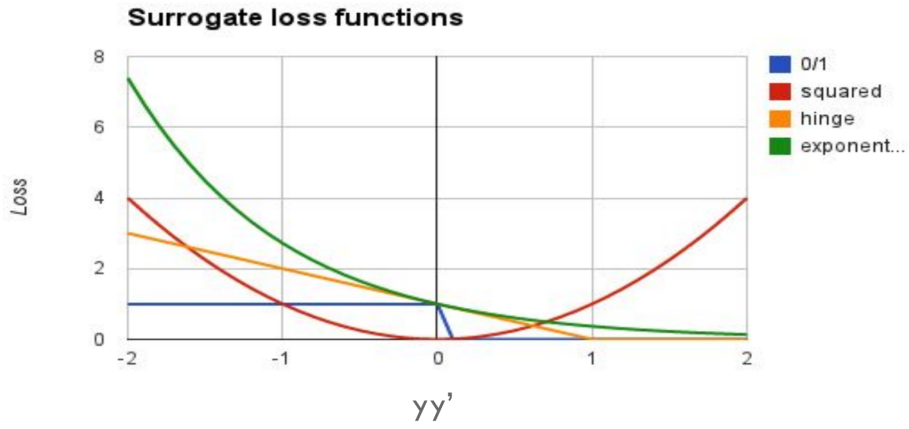
1. $0 = \text{b} + \sum_{j=1}^{n} w_j f_j$
2. $\sum_{i=1}^{n} 1[y_y(w \cdot x_i + b) \leq 0]$ (0/1 Loss function aka the total number of mistakes)
3. $argmin_{w,b} \sum_{i=1}^{n} 1[y_y(w \cdot x_i + b) \leq 0]$

5.2 Surrogate loss functions

Minimizing the 0/1 Loss function is an NP-hard problem, because of the many local minima and any change in w can change drastically the result. Ideally we would want a convex function so to have at least one minimum.

By **surrogate loss function** we mean a loss function that provides an upper bound to the 0/1 loss function.

- 0/1 loss : $l(y, y') = 1[yy' \leq 0]$
- Hinge loss: $l(y, y') = max(0, 1 - yy')$
- Exponential loss: $l(y, y') = exp(yy')$
- Squared loss: $l(y, y') = (y - y')^2$



5.3 Gradient descent

We use gradient descent to find a minimum in the function.

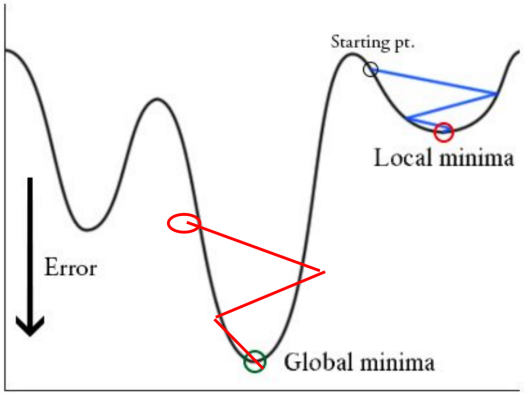
The general approach is to choose a starting point w, calculate the derivative and move in that direction.

$$w_j = w_j - \eta \frac{d}{dw_i} loss(w)$$

η is the rate at which we move and will change over time.

Since it's changing over time it can be implemented through a perceptron algorithm.

The main problem with gradient descent is local minima:



Chapter 6

Regularization

The problem with gradient descent remains the fact that since we're using it on the training set it can lead to overfitting. A solution to this is to use a regulalrizer.

Definition 6.0.1 (Regularizer)**.** An additional criterion to the loss function to avoid overfitting.

More generally it regularizes the way we handle certain weights.

Generally, we do not want huge weights: if weights are large, a small change in a feature can result in a large change in the prediction.

Two common regularizers are:

- Sum of the weights: $\sum |w_j|$
- Sum of the squared weights: $\sqrt{\sum |w_j|^2}$

Sum of weights penalizes small values whereas square sum penalizes large values.

P-norm: $\sqrt[p]{\sum |w_j|^p}$

Chapter 7

Support Vector Machines

In linear classifiers there are two variations:

- which hyperplane to choose
- how does it handle non separable data

7.1 Large margin classifiers

Definition 7.1.1 (Margin)**.** The margin is the distance from the classifier to the **closest** point of either class.

These points are called **support vectors** and for n dimensions there are $n+1$.
Maximizing the margin is usually good because it means that we can only consider SVs.
To calculate the margin:

$$\frac{w \cdot x_i + b}{||w||} = \frac{1}{||w||}$$

To maximize the margin we need to setup a constrained optimization problem:

$$max_{w,b}(\frac{1}{||w||})$$

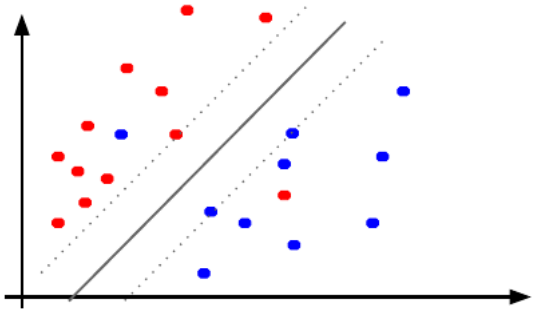
subject to: $y_i(w \cdot x_i + b \geq 1) \forall i$

We can also see it as the min of $||w||$

A **support vector machine problem** is maximize or minimize a quadratic function subject to a set of linear constraints.

7.2 Soft margin

The problem arises when the data is not linearly separable.



In this case we use **slack variables** which allow for some margin of error.

$$min_{w,b}||w||^2 + C\sum_i \varsigma_i$$

subject to: $y_i(w \cdot x_i + b \geq 1 - \varsigma_i)\forall i$

C is a regularization parameter to keep overfitting under control.
Slack values can also be calculated:

$$Slackvalues : \begin{cases} 0 & y_i(w \cdot x_i + b) \geq 1 \\ 1 - y_i(w \cdot x_i + b) & otherwise \end{cases} \tag{7.1}$$

From this we can derive an unconstrained problem:

$$min_{w,b}||w||^2 + C\sum_i (max(0, 1 - y_i(w \cdot x_i + b))$$

which is similar to a loss function with a regularizer.
An application of SVMs is for example pedestrian detection.

Chapter 8

Ranking

8.1 Multiclass vs Multilabel

Remember: multiclass classification is assigning a discrete label to a number of examples.

For multiclass, each example has one and exactly one label, whereas with multilabel each example has **zero** or multiple labels.
Some applications are:

- Reidentification
- Medical diagnosis
- Image annotation

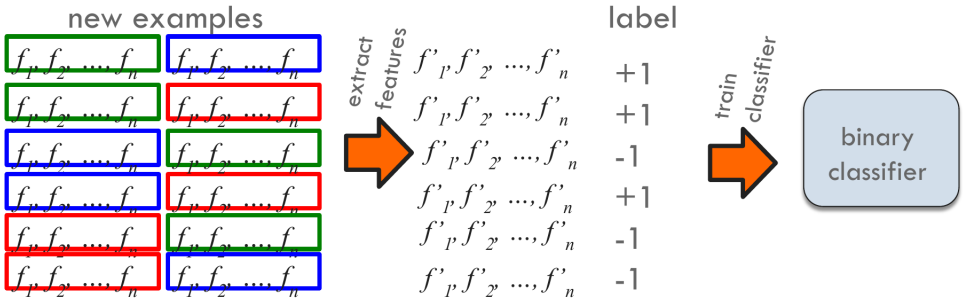
8.2 Ranking

In general the training data consists of a set of ranked examples. The **ranker** will have to rank and order the test data.
Applications:

- google search
- favourite movies
- loop closing in robotics

An approach could be using a binary classifier on pairings of examples and predict the better and worse between the two. The problem with this is that binary classifiers only take one example at the time.
The solution to this is to turn any pairings into one, comparing their features:

$$\begin{cases} 1 & a_i \geq b_i \\ 0 & otherwise \end{cases} \tag{8.1}$$



The final part of this algorithm is how to actually rank them and that's pretty straightforward: the ranking score will be equal to the sum of the rankings in the pairings.

An application of this is if a document is relevant or not and in this case we would use a bipartite ranking system which is more similar to binary classification: is this document relevant or not?

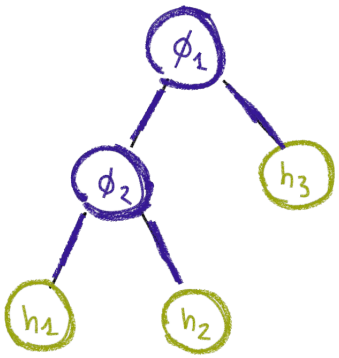
Chapter 9

Decision trees

In decision trees each internal node corresponds to a feature, the branches correspond to a value of said feature and the leaves perform a prediction.

It is composed of non-terminal nodes that have 2+ children and implement a **routing function** and of leaf nodes which implement a prediction function. A decision tree takes an input $x \in \mathscr{X}$ and routes it through the nodes until finding the final leaf, in which the prediction will take place.

Each non-terminal node is structured like so:
Node(ϕ, T_L, T_R), with a routing function *phi* and the left and right trees underneath.
On the other hand leaf nodes are Leaf(h), with h being a prediction function.



9.1 Growing the tree

The main problem with growing the tree is whether we need to grow a node or a leaf.

If a training set $\mathscr{D}$ is impure we will need to grow a node. On the other hand

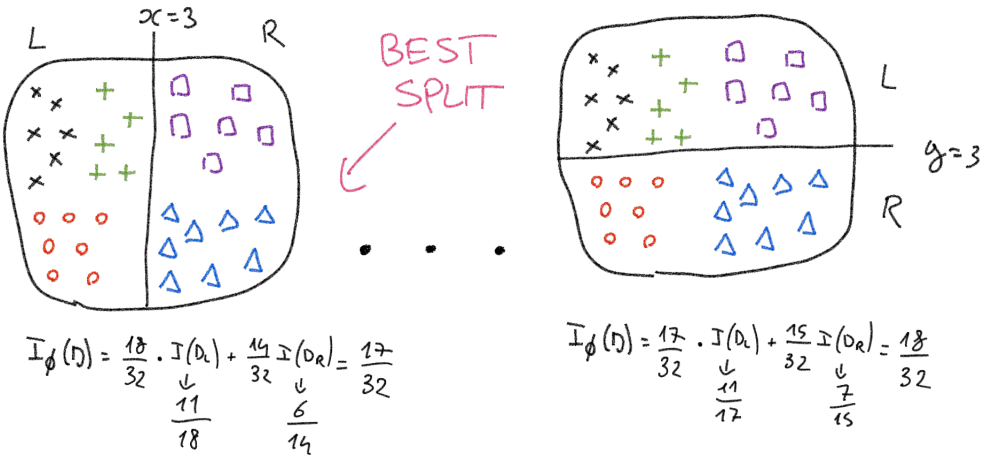
we will build a leaf if:

$$I(\mathscr{D}) = E(h^\star_{\mathscr{D}}, \mathscr{D})$$

As said previously if this stopping criterion is not met a node will be built with a routing function:

$$\phi^\star_{\mathscr{D}} \in argmin I_\phi(\mathscr{D})$$

The impurity is computed in terms of the impurity of the split data.



9.2 Overfitting

Because of the fact that they're determined by data, DT are particularly susceptible to overfitting. The usual techniques (regularization etc.) can be employed to avoid this, but there's also another called pruning.
This consists of taking a subtree and making it into a single node. This is usually done with subtress that are comprised of noisy data.
A **random forest** is a collection of decision trees

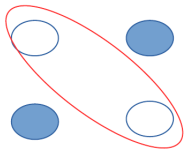
Chapter 10

Introduction to Neural Networks

Definition 10.0.1 (Artificial neuron)**.** An artificial neuron (i.e. a perceptron) is a non linear parametrized function with restricted output range

One example of perceptron is Rosemblatt (1958).

One problem with traditional perceptron is that it cannot solve the XOR:



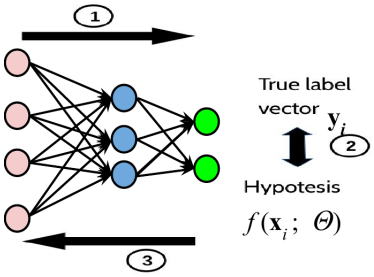
XOR

because it is not linearly separable.

An idea to solve this is an **MLP** (Multi Layered Perceptron), where the layers in between are intermediate results.

The problem now though is how to train an MLP. For perceptron we need to know the desired target and for hidden layers that is not the case. There are some ideas as follow.

Backpropagation was an idea to solve this problem, basically backpropagating the estimated error and updating the perceptron weights accordingly.



Now the problem was linked to the fact that neural networks cannot exploit many layers due to lack of computational power and overfitting.
Now neural networks reign supreme because of new findings. Raw data is always better than features and studies show that error rate is at an all time low.

Definition 10.0.2 (Neural Network)**.** It's a composition of modules and a series of hierarchically connected functions, each with its own parameters.

10.1 Feedforward networks: basic elements

The goal of a feedforward network is to approximate an idea function f. Information flows throughout the intermediate layers (hidden) to end on the last layer which is called output layer.
The function f is a composite of the functions that comprise all the layers.
The **training** is no different from other ML algorithms.
There are several modeling choices to be made:
COST FUNCTION: for example cross-entropy or square loss
OUTPUT UNITS: it's preferable to output probabilities (softmax) rather than linear outputs which could generate difficult gradient descent attributes.
The hidden unit accepts the input x and produces the output $h(z)$. The choice of h (called **ACTIVATION FUNCTION**) varies greatly.
ARCHITECTURE
OPTIMIZATION (see later)

10.2 Backpropagation

There are three steps to this:

1. Feedforward propagation
2. Use the computed output to calculate a scalar cost depending on the loss function
3. Backpropagation

The main idea of backpropagation is to use gradient descent. We do not know how or which of the hidden units are wrong but we can infer with gradient descent the speed at which we're getting farther from the correct output.

Chapter 11

Optimization for Neural Networks and CNNs

The optimization of a NN comes with more learning.
The main idea is to adjust all the weights of the network θ such that the cost function is minimized. Formally:

$$min_\theta \sum_i L(y_i, f(x_i, \theta))$$

To do this, usually we use gradient descent and back propagation, although there are different methods.

11.1 Batch (Stochastic) Gradient Descent (BGD) and (SGD)

The main idea is that the learning rate η changes linearly.
The update to the weights (w) is:
$w = w - \eta_k g$

Pros: estimates are stable
Cons: needs to estimate the GD throughout the whole training.

BGD

$$\mathbf{g} \leftarrow \frac{1}{N} \nabla_{\mathbf{w}} \sum_i L\left(f(\mathbf{x}_i; \mathbf{w}), \mathbf{y}_i\right)$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta_k \, \mathbf{g}$$

SGD

$$\mathbf{g} \leftarrow \nabla_{\mathbf{w}} L\left(f(\mathbf{x}_i; \mathbf{w}), \mathbf{y}_i\right)$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta_k \, \mathbf{g}$$

Gradient descent estimates can be very noisy so a solution is to reduce the number of batches (minibatches).
Another problem has to do with momentum: SGD can be very slow, so we introduce a new variable: velocity (v)

11.2 Convolutional Neural Networks (CNNs)

Neural networks are particularly effective when the data is spatial (i.e. language and images).

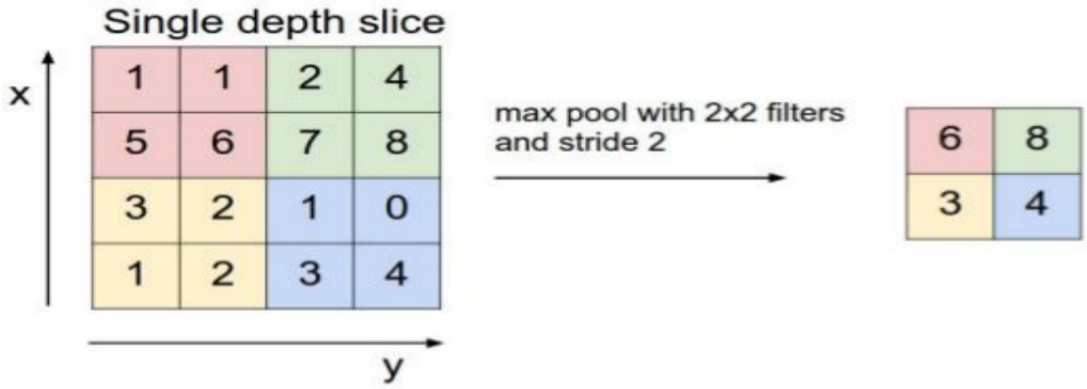
For images for instance each layer extract features from the previous output.

Definition 11.2.1 (Convolutional Neural Networks)**.** A CNN is a neural network in which it uses convolution in at least one of its layers

Okay, cool, but what is convolution?
Convolution is a general purpose filter operation for images. A kernel matrix is applied to images and it works by determining the central value of a pixel and adding the weights of all its neighbors.
In CNN's architecture there are 3 main operations:

- Convolution
- Non-linearity
- Pooling

Convolution consist of learned filters. The idea is that each filter is supposed to give insight on a specific type of feature.
Pooling consists of reducing the spatial size of the representation: this reduces the number of parameters and controls overfitting.



Chapter 12

Unsupervised Learning

whiUnsupervised learning works by observing data distribution $\mathbf{P}_data$, while lacking a target variable.
There are several applications to this:
DIMENSIONALITY REDUCTION, find a function f that maps each input $x \in \mathscr{X}$ to a lower dimensional embedding, where $\dim(\mathscr{Y}) < \dim(\mathscr{X})$.
Why would one want to do this? By compressing the input data we also reduce the feature dimensionality making the input more lightweight thus reducing the time for data elaboration.
CLUSTERING finds a function f that assigns each input $x \in \mathscr{X}$ to a cluster.
Why?
It allows to analyze data by grouping it together and can also be used to compress data with similar patterns.
DENSITY ESTIMATION find a probability distribution $f \in \triangle\mathscr{X}$ that fits the data $x \in \mathscr{X}$. Why? Allows for explicit estimate of an unknown probability distribution, generating new data and detecting anomalies.

12.1 Principal component analysis

The main idea is to fidn the direction of maximum variance, change the coordinate system and then dropping dimensions of least variance.

Definition 12.1.1 (Variance and Covariance)**.** Measure of the spread of a set of points around their center of mass (the mean)

I don't really know how to explain easily eigenvalues and eigenvectors so I'm just gonna put a picture of the example.

$$\textit{Show } x = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \textit{is an eigenvector for } A = \begin{bmatrix} 2 & -4 \\ 3 & -6 \end{bmatrix}$$

$$\textit{Solution}: Ax = \begin{bmatrix} 2 & -4 \\ 3 & -6 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\textit{But for } \lambda = 0, \ \lambda x = 0 \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\textit{Thus, } x \textit{ is an eigenvector of } A, \textit{and } \lambda = 0 \textit{ is an eigenvalue.}$$

It can be shown that the largest eigenvalue is the variance along the first principal component.

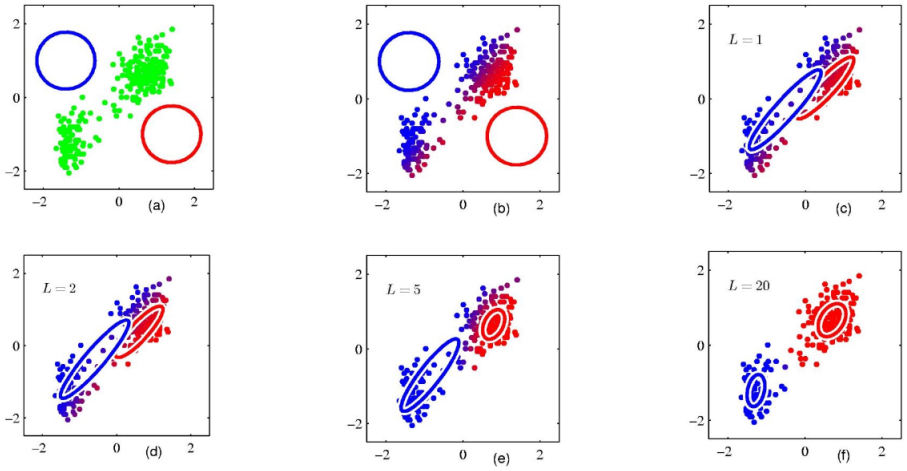
12.2 Clustering

The goal is to divide data into clusters. What K-means clustering does is fix a number of k clusters and divides the data with the least amount of variance inside the clusters.

$$min \sum_{j=1}^{k} V(\mathscr{C})$$

Some properties are that it is sensitive to the scale of features, it is guaranteed to converge although it's not guaranteed to find a global minimum.
The problem with K-means clustering is that it requires spherical clusters. It is also a **hard cluster**, meaning that example belong exactly to one cluster only. **EM-Clustering** (Expectations Maximization) on the other hand is a soft clustering technique that takes data as a mixture of gaussian.

With EM clustering you start with initial cluster centers, you soft assign points to each cluster and then iterate.



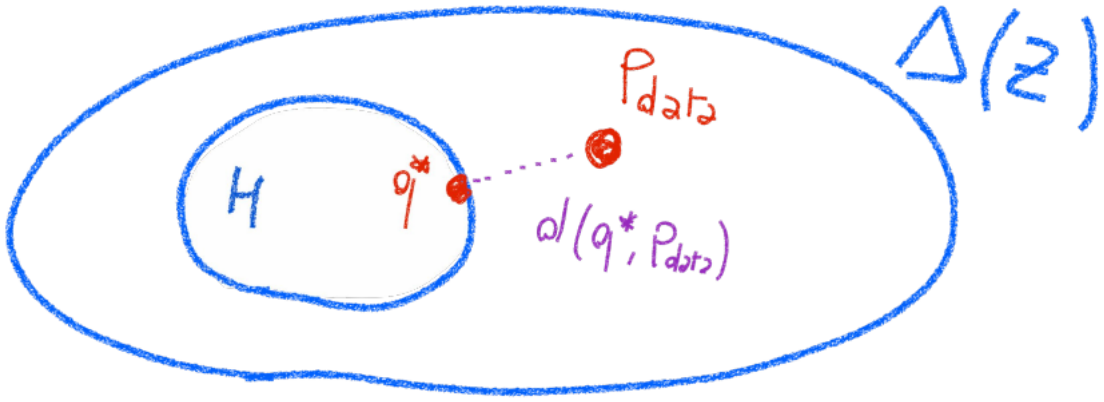
To recalculate the centers we need them to fit a gaussian.

There are other means of clustering namely spectral clustering (using graphs), hierarchical graphs, agglomerative clustering etc.

Chapter 13

Deep Generative Models

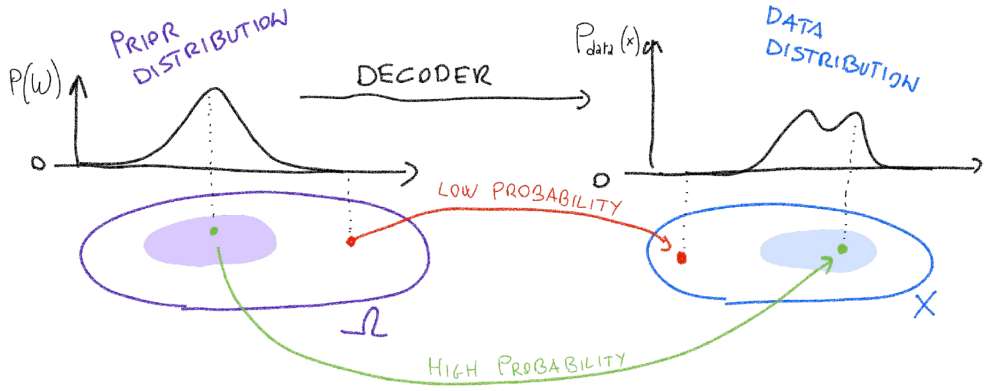
A generative model is a statistical model of the data distribution $\mathbf{p}$.
A discriminative model on the other hand are statistical models of conditional distributions.
To generate data there are two general ways: explicit and implicit density estimation.
In explicit we find a probability distribution $f \in \triangle\mathscr{Z}$ that fits the data, where z is sampled from $\mathbf{p}_{data}$.
In implicit we have to find a function f that generates data $f(\omega) \in \mathscr{Z}$
The objective is to find an hypothesis space that can represent probability distributions and that best fits the data.



13.1 Variational Autoencoders (VAE)

The main idea is to add a probability to traditional autoencoders.
Autoencoders are a way of compressing high-dimensional data into a lower dimensional representation (e.g. PCA).
An encoder is trained by leveraging a decoder mapping the representation back

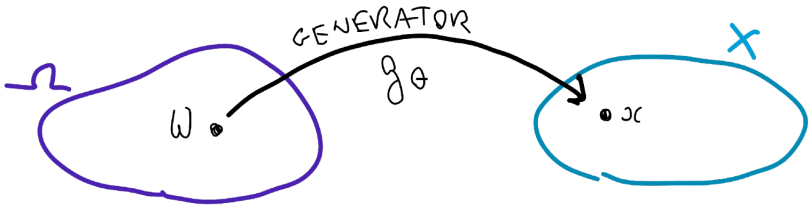
to the input domain, yielding a reconstruction.
This decoder can be used to generate new data although it will not be following the data distribution.



A variation of VAE is the conditional VAE that creates data considering other factors too (generate a person with glasses, for example.)
There are of course issues with VAE namely the risk of underfitting and blurry created data.

13.2 Generative Adversarial Networks (GAN)

GAN enables the possibility of estimating implicit densities.
We assume to have already a prior density $\mathbf{p}_\omega \in \triangle(\Omega)$ and a generator (or decoder) that takes $g \in \mathscr{X}^\Omega$ that generates points in $\mathscr{X}$ given random elements from Ω.



The objective is again to find the best fitting data.
The training for GAN works like a one versus one game. Player 1 generates data and player 2 needs to discern whether that's from the original p data or from the newly generated one. This is done with gradient descent
Issues are that parameters might not converge and that there might be a vanishing gradient.

Chapter 14

Reinforcement Learning

The idea is similar to behavioral psychology. It's good for problems where an agent is interacting with the environment.
The agent can take actions that affect the environment based on rewards, states and policies. Policies are a set of actions taken to change the state of the environment. An easy example is atari games ML.

14.1 Markov Decision Process

MDP is defined by the follwing:

- S set of possible states
- A set of actions
- R distribution of reward, given (state,action) pair
- P transition probability
- discount value γ

It uses dynamic programming (Bellman algorithm) to divide the problem in multiple sub problems. γ is used to calculate the rewards in the next states. Example: grid world
There are two main methods for RL: value based methods and policy based methods.

14.2 Value based methods

The value function is a function that calculates the value of the reward for a particular state. With this function we can find a policy by finding the max.
Another approach is the Q-function which takes as input a pair of states and actions. This is also called Q-Learning and it uses a reward table to explain the

actions taken and the respective reward.
At the start the agent has a Q matrix that through learning becomes the reward table R.