# Introduction

**C.M. Bishop**: Machine Learning is concerned with the automatic discovery of regularities in data through the use of omputer algorithms and with the use of these regularities to take actions.
**K.P. Murpy**: The goal of Machine Learning is to develop methods that can automatically detect patterns in data, and then to use the uncovered patterns to predict future data or other outcomes of interest.
**Hal Daume III**: Machine Learning is about predicting the futre based on the past.
**T. Mitchell**: A computer program is said to learn from experience $E$ with respect to some calss of tasks $T$ and performance $P$, if its performance at task in $T$, as measured by $P$, improves with experience $E$.

**Formally** Machine Learning is the study of algorithms that improve their performance $P$ at some task $T$ with experience $E$. A well-defined learning task is given by a triplet $\langle T, P, E\rangle$.

# Data, Features, and Models

## Learning process

1. Measuring devices — sensors, cameras, databases;
2. Preprocessing — noise filtering, feature extraction, normalization;
3. Dimensionality Reduction — feature selection and/or projection;
4. Model Learning — classification, regression, clustering, description;
5. Model Testing — cross-validation, bootstrap;

## Data

**Data** are individual facts, statistics, or items of information, often numeric. Data are set of values of qualitative or quantitative variables about one or more persons or objects.

- **Training data**: the part of data that we use to train our model. This is data that the model learns from.
- **Validation data**: the part of data that is used to do a frequent evaluation of the model, fit on the training dataset along with improving involved hyperparameters.
- **Test data** once the model is trained, test data provides an unbiased evaluation. After prediction, we evaluate our model by comparing it with the actual output present in the test data.

## Feature

A **feature** is an individual measurable property or characteristic of a phenomenon. A **feature-vector** is an $n$-dimensional vector of numerical features that represent some object, and they are often combined with weights using a dot product in order to construct a linear predictor function.

## Task

Tasks represent the type of prediction being made to solve a problem on some data. In general, it consists of functions assigning each input $x \in X$ an output $y \in Y$.

$$f : X \to Y \qquad F_{\text{task}} \subset Y^X \tag{1}$$

## Types of learning

### Supervised Learning

Build a mathematical model of a set of data that contains both the inputs and the desired outputs. Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the ouput associated with new inputs.

- **Classification**: problem of identifying which of a set of categories an observation belongs to.
  **Formally** given a training set $T = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, learn a function $f$ to predict $y$ given $x$.
  - *Binary*:
  $$f : \mathbb{R}^1 \to \{1, 2, \ldots, k\} \tag{2}$$
  - *Multiclass*:
  $$f : \mathbb{R}^d \to \{1, 2, \ldots, k\} \tag{3}$$

  **Task** find a function $f \in Y^X$ assigning each input $x \in X$ a discrete label
  $$f(x) \in Y = \{c_1, c_2, \ldots, c_k\} \tag{4}$$
- **Regression**: set of statistical processes for estimating the relationships between a dependent variable and one or more independent variables.
  **Formally** given a training set $T = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, learn a function $f$ to predict $y$ given $x$.
  $$f : \mathbb{R}^d \to \mathbb{R} \qquad \text{where } d = 1 \tag{5}$$
  **Task** find a function $f(x) \in Y$ assigning each input a continuous label.
- **Ranking**: Data transformation in which numerical or ordinal values are replaced by their rank when the data are sorted.

### Unsupervised Learning

Unsupervised learning algorithms take a set of data that contains only inputs, and find structure in the data, like grouping or clustering of data points.

- **Clustering**: assignment of observations into subsets so that observation within the same cluster are similar accorting to one or more predesignated criteria.
  **Formally** given $T = \{x_1, x_1, \ldots, x_m\}$ (without labels), the output is the hidden structure behind the $x$'s, that is the cluster.
  **Task** find a function $f \in \mathbb{N}^X$ that assigns each input $x \in X$ a cluster index $f(x) \in \mathbb{N}$.
- **Anomaly detection**: identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data.
- **Dimensionality reduction**: process of reducing the number of random variables under consideration by obtaining a set of principal variables — it is the process of reducing the feature set.
  **Task** find a function $f \in Y^X$ mapping each input $x \in X$ to a lower dimensional embedding $f(x) \in Y$, where $\dim(Y) \ll \dim(X)$ and $Y = \mathbb{R}^2$.
- **Density estimation**: construction of an estimate, based on observed data, of an unobservable underlying probability density function.
  **Task** find a probability distribution $f \in \Delta(x)$ that fits the data $x \in X$.

### Reinforcement Learning

Area of ML concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment.

# Models and Hypotesis space

A **model** in machine learning is the output of a machine learning algorithm run on data. A model represents what was learned by a machine learning algorithm.

We assume that the set of all possible models is a subset $H \subset F_{\text{task}}$, called **hypothesis space**.

### Ideal target

$$f^* \in \arg\min_{f \in F_{\text{task}}} E(f; p_{\text{data}}) \tag{6}$$

### Feasible target

$$f^*_H \in \arg\min_{f \in H} E(f; p_{\text{data}}) \tag{7}$$

### Actual target

$$f^*_H(D_n) \in \arg\min_{f \in H} E(f; D_n) \tag{8}$$

## Error function

Typically in machine learning problems, we seek to minimize the error between the predicted value vs the actual value. The word error represents the penalty of failing to achieve the expected output. If the loss is calculated for a single training example, it is called or error function.

Typically, the generalization and training error function can be written in terms of a pointwise loss l(f; z) measuring the error incurred by f on the training example z.

$$E(f; p_{\text{data}}) = E_{z\sim p_{\text{data}}}[l(f;z)] \quad E(f;D_n) = \frac{1}{n}\sum_{i=1}^{n} l(f;z_i) \tag{9}$$

### Overfitting

Production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations.

### Underfitting

Occurs when a ML algorithm cannot adequately capture the underlying structure of the data. It occurs when the algorithm shows low variance but high bias.

## Generalization

Generalization error cannot be computed for $p_{\text{data}}$, so we use data sample, and we assume that the probability distribution is the same for the training set, for the validation set, and for the test set.

In order to improve the generalization well, there are number of approaches:

- Avoid to obtain the minimum on training error;
- Reduce model capacity;
- Change the objective with a regularization term;
- Inject noise in the learning algorithm to smooht the data points;
- Stop the learning before the convergence.

**Regularization** Is the modification of the training error function with a term $\Omega(f)$ that typically penalizes complex solutions.

$$E_{\text{reg}}(f;D_n) = E(f;D_n) + \lambda_n \cdot \Omega(f) \tag{10}$$

# $k$-Nearest Neighbors

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, $k$ is a user-defined constant, and an unlabeled vector is classified by assigning the label which is most frequent amongh the k training samples nearest to that query point.

## Distance

### Euclidean

$$D(a,b) = \sqrt{\sum_{d=1}^{D}(a_d - b_d)^2} \tag{11}$$

### Minkowski

$$D(a,b) = \left[\sum_{k=1}^{p}|a_k - b_k|^r\right]^{\frac{1}{r}} \tag{12}$$

### Cosine similarity

$$S_C(a,b) = \cos(\theta) = \frac{a\cdot b}{||a||\ ||b||} \tag{13}$$

**Alg 1:** $k$-NN-classification(data, query, $k$, distance$_{fn}$)

```
neigDist ← [ ];
foreach (index, example) ∈ data do
    dist ← distance_fn(example.pop(), query);
    neigDist.append((dist,index));
end
sortedND ← sort(neigDist);
distList ← sortedND[k];
labels ← [distList.index];
return distList.counter(labels).mostCommon()
```

## Choosing $k$

The best choice of $k$ depends upon the data; generally, larger values of $k$ reduces effect of the noise on the classification, but make boundaries between classes less distinct. A good $k$ can be selected by various heuristic techniques.

The accuracy of the $k$-NN algorithm can be severely degraded by the presence of noisy or irrelevant features, or if the feature scales are not consistent with their importance

## Weighted $k$-NN

Give more weight to the points which are nearby and less weight to the points which are farther away. The $i$-th nearest neighbour is assigned a weight $w_{n_i}$ with

$$\sum_{i=1}^{n} w_{n_i} = 1 \tag{14}$$

The algorithm is the same, but in the end we use the distance-weighted voting, that use the following formula

$$y' = \arg\max_{v} \sum_{(x_i,y_i)\in D_z} w_i \times 1(v = y_i) \tag{15}$$

where $v$ represents the class labels.

# Linear Models

Linear models generate a formula to create a best-fit line to predict unknown values.

## Bias

The bias of a model is how strong the model assumptions are.

- Low bias: minimal assumptions ($k$-NN, decision tree),
- High bias: strong assumptions about the data.

A strong high bias assumption is **linear separability**: can separate classes by a hyperplane. A linear model in $n$-dimensional space is defined by $n+1$ weights. The hyperplane can be used for classification by simply looking at the sign of the equation:

$$b + \sum_{i=1}^{n} w_f \cdot f_i \tag{16}$$

## Online Learning

Method in which data becomes available in a sequential order and is used to update the best predictor for future data at each step. Predictive model can be updated instantly for any new data instances, thus are more efficient and scalable for large-scale ML tasks.

## Perceptron

The perceptron is an algorithm for learning a binary classifier called threshold function, that maps its input $x$ to a value $f(x)$:

$$f(x) = \begin{cases} 1 & \text{if } w\cdot x + b > 0, \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

The value of $f(x)$ is used to classify $x$ as either positive or negative instance. Spatically, the bias alters the position of the decision boundary.

The perceptron algorithm does not terminate if the learning set is not linearly separable. In order to prevent infinite loop we can use a maximum number of iteration.

The perceptron algorithm guarantees that find some line, but not a special line.

**Alg 2:** Perceptron learning

```
while convergence not reached do
    foreach training example $f_i$, label do
        prediction $\leftarrow b + \sum_{i=1}^{n} w_i f_i$;
        if not correct then
            foreach $w_i$ do
                $w_i \leftarrow w_i + f_i \cdot$ label;
            end
            $b \leftarrow b +$ label;
        end
    end
end
```

# Decision trees

Non-parametric supervised learning algorithm. The concept is to select splits that decrease the impurity of classes distribution in the resulting subsets of intstances.

Decision tree takes an inptu $x \in X$. Each non-terminal node $N(\phi, t_L, t_R)$ holds a routing function $\phi \in L, R^X$, a left child $t_L$ and a right child $t_R$, that will be routed on based on the value of $\phi(x)$.

When you do a split, each of the created descendant nodes corresponds to the applicable subset of the training data set. Nodes that are not split further become leaves, each leaf node $L(h)$ holds a predition function $h \in F_{\text{task}}$.

Given a training set $D_n = \{z_1, z_2, \dots, z_n\}$, the set $T$ of all the possible decision trees, find $f_{t^*}$, that corresponds to the optimal tree, where

$$t^* \in \arg\min_{t \in T} E(f_t; D_n) \tag{18}$$

We assume

$$E(f_t; D) = \frac{1}{|D|} \sum_{z \in D} l(f; z) \tag{19}$$

where $l(f; z)$ is the pair-wise loss.

**Growing a leaf** Given $D = \{z_1, z_2, \dots, z_n\}$, the optimal leaf predictor that minimizes the local error can be computed as

$$h_D^* \in \arg \min_{h \in H_{\text{leaf}} E(h; D)} \tag{20}$$

The optimal error value is called impurity measure, and tells us what is the probability of misclassifying an observation — the lower the impurity, the better the split.

**Growing a node** Given the set of all the possible routing functions $\Phi$, and the training samples $D$

$$\Phi_D^* \in \arg\min_{\phi \in \Phi} I_\phi(D) \tag{21}$$

The impurity $I_\phi(D)$ is computed in terms of the impurity of the split data

$$I_\phi(D) = \sum_{d \in \{L,R}\} \frac{|D_d^\phi|}{|D|} I(D_d^\phi)\, D_d^\phi = \{(x,y) \in D;\ \phi(x) = d\} \tag{22}$$

$$\text{Grow}(D) = \begin{cases} L(h_D^*)\text{if stopping criterion met} \\ N(\phi_D^*, \text{Grow}(D_L^*), \text{Grow}(D_R^*))\text{otherwise} \end{cases} \tag{23}$$

## Impurity measures

The error function can have different loss function that can lead to completely different error functions.

### Classification

### Classification error

$$I(D) = 1 - \max_{y \in Y} \frac{|D^y|}{|D|} \tag{24}$$

### Gini impurity

$$I(D) = 1 - \sum_{y \in Y} \left(\frac{|D^y|}{|D|}\right)^2 \tag{25}$$

### Entropy

$$I(D) = -\sum_{y \in Y} \frac{|D^y|}{|D|} \log\left(\frac{|D^y|}{|D|}\right) \tag{26}$$

### Regression

### Variance

$$I(D) = \frac{1}{|D|} \sum_{(x,y) \in D} ||x - \mu_D||^2 \tag{27}$$

## Split functions

### Discrete nominal features

$$\phi(x) = \begin{cases} L & \text{if } x \in K_L \\ R & \text{if } x \in K_R \end{cases} \tag{28}$$

### Ordinal features

$$\phi(x) = \begin{cases} L & \text{if } x \le r \\ R & \text{if } x > r \end{cases} \tag{29}$$

### Oblique

$$\phi(x) = \begin{cases} L & \text{if } w^T \cdot x \le r \\ R & \text{otherwise} \end{cases} \tag{30}$$

## Pruning

Data compression technique that reduces the size of decision trees by removing sections in the tree that are non-critical and redundant to classify instances. Remove the nodes that do not provide additional information.

## Random Forests

We inject randomness into the tree building to ensure each tree is different.

Split functions are optimized on randomly samples features. The final prediction of the forest is obtained by averaing the prediction of each tree in the ensemble $Q = \{t_1, t_2, \dots, t_T\}$

$$f_Q(x) = \frac{1}{T} \sum_{j=1}^{T} f_t(x) \tag{31}$$

**Alg 3:** Random Forest Learning

```
foreach $b \leftarrow 1$ to $B$ do
    Draw a boostrap sample $\vec{Z}^*$ of size $N$ from the training
      data;
    Grow a random-forest tree $T_b$ to the bootstrapped data;
    foreach terminal node $\in T_b$ do
        repeat
            Select $m$ variables at random from the $p$ variables;
            Pick the best variable/split-point among the $m$;
            Split the node into two daughter nodes;
        until minimum node size $n_{min}$ is reached;
    end
end
return Output the ensemble of trees $T_b$
```

# Multiclass Classification

Problem of classifying instances into one of three or more classes.

It is easy to think of a binary classifier as a black box, which can be reused for solving more complex problems.

$k$-NN and decision tree algorithms can by nature handle multiclass classification. Perceptron algorithm need some changes to fit the multiclass problem.

## One vs All (OVA)

You train $K$-many binary classifiers $f_1, f_2, \dots, f_K$. Classifier $f_i$ receives all examples labeled class $i$ as positives and all other examples as negatives.

```
Alg 4: OneVersusAllTrain($D^{\text{multiclass}}$, $BinaryTrain$)
for $i \leftarrow 1, ..., K$ do
    $D^{\text{bin}} \leftarrow$ relabel $D^{\text{multiclass}}$ so class $i$ is positive, and class $\neg i$ is negative;
    $f_i \leftarrow$ BinaryTrain($D^{\text{bin}}$);
end
return $f_1, ..., f_K$
```

```
Alg 5: OneVersusAllTest($f_1, ..., f_K$, $x$)
$score \leftarrow \langle 0, 0, ..., 0 \rangle$;
for $i \leftarrow 1, ..., K$ do
    $y \leftarrow f_i(x)$;
    $score_i \leftarrow score_i + y$;
end
return arg max$_k$ $score_k$
```

## All vs All (AVA)

$f_{ij}$ for $1 \le i < j \le K$ is the classifier that pits class $i$ againts class $j$.

```
Alg 6: AllVersusAllTrain($D^{\text{multiclass}}$, $BinaryTrain$)
$f_{ij} \leftarrow \emptyset, \forall 1 \le i < j \le K$;
for $i \leftarrow 1, ..., K-1$ do
    $D^{\text{pos}} \leftarrow$ all $x \in D^{\text{multiclass}}$ labeled $i$;
    for $j \leftarrow i+1, ..., K$ do
        $D^{\text{neg}} \leftarrow$ all $x \in D^{\text{pos}}$ labeled $j$;
        $D^{\text{bin}} \leftarrow \{(x, +1) : x \in D^{\text{pos}}\} \cup \{(x, -1) : x \in D^{\text{neg}}\}$;
        $f_{ij} \leftarrow BinaryTrain(D^{\text{bin}})$;
    end
end
return $all\ f_{ij}$
```

## Evaluation

- **micro-average** calculates the metric from the aggregate contributions of all classes. This is used for unbalanced datasets.
- **macro-average** calculates the metric autonomously for each class to calculate the average;

### Confusion matrix

Shows the combination of the actual and predicted classes. The rows represent the predicted class and the columns represent the instances of the actual class.

# Gradient Descent

First-order iterative optimization algorithm for finding a local minimum of a differentiable function.

## Optimization framework for linear models

In the case that your training data isn't linearly separable, you want to find the hyperplane that makes the fewest errors on the training data.

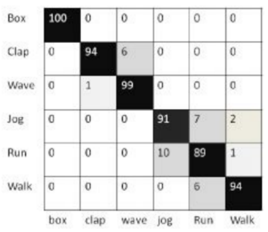
$$\min_{w,b} \sum_n 1\left[y_n(\vec{w} \cdot \vec{x} + b) > 0\right] \tag{32}$$

If the optimum is zero, then the perceptron will efficiently find parameters for this model.

## Convex Surrogate Loss Functions

Convex functions are easy to minimize. We want to approximate zero/one loss with a convex function, called **surrogate loss** — it

```
Alg 7: AllVersusAllTest(all $f_{ij}$, $x$)
$score \leftarrow \langle 0, 0, ..., 0 \rangle$;
for $i \leftarrow 1, ..., K-1$ do
    for $j \leftarrow i+1, ..., K$ do
        $y \leftarrow f_{ij}(x)$;
        $score_i \leftarrow score_i + y$;
        $score_j \leftarrow score_j - y$;
    end
end
return arg max$_k$ $score_k$
```



| | box | clap | wave | jog | Run | Walk |
|---|---|---|---|---|---|---|
| Box | 100 | 0 | 0 | 0 | 0 | 0 |
| Clap | 0 | 94 | 6 | 0 | 0 | 0 |
| Wave | 0 | 1 | 99 | 0 | 0 | 0 |
| Jog | 0 | 0 | 0 | 91 | 7 | 2 |
| Run | 0 | 0 | 0 | 10 | 89 | 1 |
| Walk | 0 | 0 | 0 | 0 | 6 | 94 |

will be the uppor bound on the true loss function.

$$\textbf{Zero/one:}\ l^{(0/1)}(y, \hat{y}) = 1[y\hat{y} \le 0] \tag{33}$$

$$\textbf{Hinge:}\ l^{(\text{hin})}(y, \hat{y}) = \max\{0, 1 - y\hat{y}\} \tag{34}$$

$$\textbf{Logistic:}\ l^{(\log)}(y, \hat{y}) = \frac{1}{\log 2}\log(1 + \exp[-y\hat{y}]) \tag{35}$$

$$\textbf{Exponential:}\ l^{(\exp)}(y, \hat{y}) = \exp[-y\hat{y}] \tag{36}$$

$$\textbf{Squared:}\ l^{(\text{hin})}(y, \hat{y}) = (y - \hat{y})^2 \tag{37}$$

$$\tag{38}$$

## Gradients

Multidimensional generalization of a derivative. The radient of $f$ is just the vector consisting of the derivative $f$ with respect to each of its input coordinates independently.

$$\nabla_x f = \langle \frac{\delta f}{\delta x_1}, \frac{\delta f}{\delta x_2}, \dots \frac{\delta f}{\delta x_D} \rangle \tag{39}$$

## Optimization with gradient descent

At each step, it measures the gradient of the function that it is trying to optimize. It then takes a step in the direction of the gradient. The complete step is $\vec{x} \leftarrow \vec{x} + \eta g$.

```
Alg 8: GradientDescent($F$, $K$, $\eta_1, ..., \eta_K$)
$\vec{z}^{(0)} = \langle 0, 0, ..., 0 \rangle$;
for $k \leftarrow 1, ..., K$ do
    $\vec{g}^{(k)} \leftarrow \nabla_{\vec{z}} F|_{\vec{z}^{(k-1)}}$;
    $\vec{z}^{(k)} \leftarrow \vec{z}^{(k-1)} - \eta^{(k-1)}\vec{g}^{(k)}$;
end
return $\vec{z}^{(K)}$
```

$$\frac{\delta l}{\delta w_j} = \frac{\delta}{\delta w_j}\sum_{i=1}^{n}\exp(-y_i(\vec{w} \cdot x_i + b)) \tag{40}$$

$$= \sum i = 1^n - y_i \cdot x_{ij} \cdot \exp(-y_i(\vec{w} \cdot \vec{x}_i + b)) \tag{41}$$

# Regularization

The goal is to ensure that the learned function does not overfit.

If you want the funcion to change slowly, you want to ensure that the weights stay small.

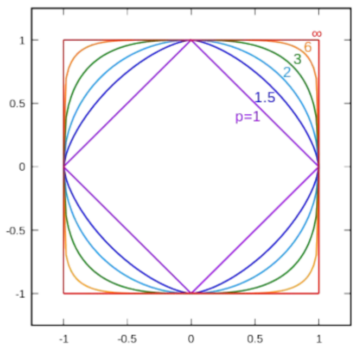
$$R^{(\text{norm})}(\vec{w}, b) = ||\vec{v}|| = \sqrt{\sum_d w_d^2} \tag{42}$$

$$R^{(\text{abs})}(\vec{w}, b) = \sum_d |w_d| \tag{43}$$

## $p$-norm

This is a family of norms that have the same general flavor. $||\vec{w}||_p$ denote the $p$-norm of $\vec{w}$.

$$||\vec{w}||_p = R(\vec{w}, b) = \left[\sum_{w_j} |w_j|^p\right]^{\frac{1}{p}} \tag{44}$$



## L1 and L2 norms

L1 and L2 norms are common norms, and both push the weights to have smaller values close to zero.

Given the model

$$\hat{y} = wx + b \tag{45}$$

L1 regularizer gives a diamond, this means that one of the coordinates is different from zero, and the other is exactly zero.

$$L_1 = (wx + b - y)^2 + \lambda|w| \tag{46}$$

$$\frac{d|w|}{dw} = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \end{cases} \tag{47}$$

L2 regularizer gives a circle, that has no corner, it is very unlikely that the meet-point is on any of axes, but it is very close to zero.

$$L_2 = (wx + b - y)^2 + \lambda w^2 \tag{48}$$

## Gradient descent

$$\min_{\vec{w}, b} \sum_{i=1}^{n} l(y\hat{y}) + \lambda R(\vec{w}) \tag{49}$$

is convex as long as both the loss function and the regularizer are convex.

$$\text{L1: } w_{\text{new}} = \begin{cases} w - \eta \cdot [2x(wx + b - y) + \lambda] & w > 0 \\ w - \eta \cdot [2x(wx + b - y) - \lambda] & w < 0 \end{cases} \tag{50}$$

$$\text{L2: } w_{\text{new}} = w - \eta \cdot [2x(wx + b - y) + 2\lambda w] \tag{51}$$

## Model-based ML

- Pick a model

$$b + \sum_{j=1}^{n} w_j \cdot f_j = 0 \tag{52}$$

- Pick the objective function

$$\sum_{j=1}^{n} \exp(-y_i \cdot (\vec{w}x_i + b)) + \frac{\lambda}{2}||\vec{w}||^2 \tag{53}$$

- Develop a learning algorithm

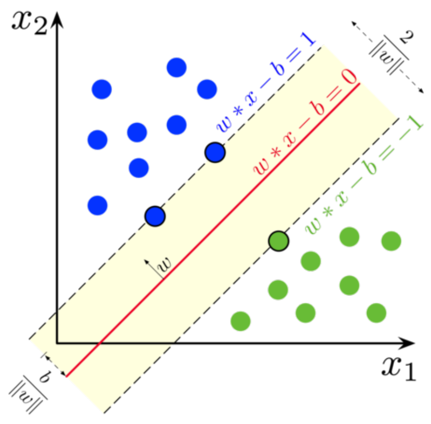
$$\min_{\vec{w}, b} \sum_{j=1}^{n} \exp(-y_i \cdot (\vec{w}x_i + b)) + \frac{\lambda}{2}||\vec{w}||^2 \tag{54}$$

## SVM

Optimization problem that attempts to find a separating hyperplane with as large margin as possible.

**Margin** of a data point: distance from the data point to a decision boundary.

**Margin classifier**: gives an associated sitance from the decision boundary for each example.



**Support vectors**: elemtns of the training set that would change the position of the dividing hyperplane if removed. For $n$ dimension we have $n + 1$ suppor vectors.

The measure for the margin is defined by

$$\frac{\vec{w} \cdot \vec{x}_i + b}{||\vec{w}||} = \frac{1}{||\vec{w}||} \tag{55}$$

Our goal is to **maximize the margin**

$$\max_{a,b} \text{margin}(\vec{w}, b) = \max_{\vec{w}, b} \frac{1}{||\vec{w}||} \tag{56}$$

$$= \min_{\vec{w}, b} ||\vec{w}|| \tag{57}$$

The support vetor machine problem is a version of a quadratic optimization problem

$$\min_{\vec{w}, b} ||\vec{w}||^2 \text{subject to: } y_i \cdot (\vec{w} \cdot \vec{x}_i + b) \geq 1 \qquad \forall i \tag{58}$$

## Soft margin classification

If the training set $D$ is not linearly separable, we allow the fat decision margin to make some mistake. We pay a cost for each misclassified example. To implement this we introduce **slack variables** $\zeta_i$.

A non-zero value for $\zeta_i$ allows $\vec{x}_i$ to not meet the margin requirement at a cost proportional to the value of $\zeta_i$.

The formulation of the SVM optimization problem with slack variable is:

Find $\vec{w}$, $b$ and $\zeta_i > 0$ such that:

$$\min_{\vec{w}, b} ||\vec{w}||^2 + C \sum_i \zeta_i \tag{59}$$

$$\text{subject to: } y_i \cdot (\vec{w} \cdot \vec{x}_i + b) \geq 1 - \zeta_i \qquad \forall i, \zeta \geq 0 \tag{60}$$

The sum of the $\zeta_i$ gives an upper bound on the number of training errors.

$$\zeta_i = \begin{cases} 0 & \text{if } y_i \cdot (\vec{w} \cdot \vec{x}_i + b) \geq 1 \\ 1 - y_i \cdot (\vec{w} \cdot \vec{x}_i + b) & \text{otherwise} \end{cases} \tag{61}$$

$$= \max\{0, 1 - y_i \cdot (\vec{w} \cdot \vec{x}_i + b)\} \tag{62}$$

$$= \max\{0, 1 - y\hat{y}\} \tag{63}$$

$$\tag{64}$$

We can rewrite our optimization problem as

$$\min_{\vec{w}, b} ||\vec{w}||^2 + C \sum_i \max\{0, 1 - y\hat{y}\} \tag{65}$$

## Non linearly separable data

**Dual problem** Find $\alpha_1, \alpha_2, \ldots, \alpha_n$ such that $\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \vec{x}_i^T s_j$ is maximized and

- $\sum_i \alpha_i y_i = 0$
- $\alpha_i \geq 0 \qquad \forall \alpha_i$

Given a solution of the dual problem, the solution of the primal is

$$\vec{w} = \sum_i \alpha_i y_i \vec{x}_i \tag{66}$$

$$b = y_k - \sum_i \alpha_i y_i \vec{x}_i^T \vec{x}_k \qquad \text{for any } \alpha_k > 0 \tag{67}$$

The classifying function is

$$f(\vec{x}) = \sum_i \alpha_i y_i \vec{x}_i^T \vec{x} + b \tag{68}$$

## Kernel trick

The linear classifier relies on inner product between vectors

$$K(\vec{x}_i, \vec{y}_i) = \vec{x}_i^T \cdot \vec{y}_i \tag{69}$$

If every datapoint is mapped into high-dimensional space via some transfomration $\Phi : \vec{x} \to \phi(\vec{x})$ the inner product becomes

$$K(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i)^T \cdot \phi(\vec{x}_j) \tag{70}$$

A **kernel function** is a function that is equivalent to an inner product in some feature space.

**Mercer's theorem** Every positive semidefinite symmetric function is a kernel. A positive semidefinite symmetric function corresponds to a positive emidefinite symmetric Gram matrix.

$$K = \begin{bmatrix} K(\vec{x}_1,\vec{x}_1) & K(\vec{x}_1,\vec{x}_2) & ... & K(\vec{x}_1,\vec{x}_n) \\ K(\vec{x}_2,\vec{x}_1) & K(\vec{x}_2,\vec{x}_2) & ... & K(\vec{x}_2,\vec{x}_n) \\ ... & ... & ... & ... \\ K(\vec{x}_n,\vec{x}_1) & K(\vec{x}_n,\vec{x}_2) & ... & K(\vec{x}_n,\vec{x}_n) \end{bmatrix} \tag{71}$$

### Kernels

- Linear: $K(\vec{x}_i, \vec{x}_j) = \vec{x}_i^T \vec{x}_j$
- Polynomial of power $p$: $K(\vec{x}_i, \vec{x}_j) = (1 + \vec{x}_i^T \vec{x}_j)^p$
- Gaussian (radial-basis function): $K(\vec{x}_i, \vec{x}_j) = e^{\frac{||\vec{x}_i - \vec{x}_j||^2}{2\sigma^2}}$

With the kernels, the dual problem formulation becomes

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\vec{x}_i, \vec{x}_j)$$
$$\text{s.t.} \sum_i \sum_j \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad \forall i$$

And finally the solution is

$$f(\vec{x}) = \sum_i \alpha_i y_i K(\vec{x}_i; \vec{x}) + b \tag{72}$$

# Ranking

Produce a permutation of items in new, unseen lists in a similar way to rankings in the training data.

**Multilabel classification** multiple labels may be assigned to each instance, each example has zero or more labels. Formally, is the problem of finding a model that maps inputs $x$ to binary vectors $y$.

**Bipartite ranking** predict a binary response, for instance "is this document relevant or not?".

## Naive algorithm

**Alg 9:** NaiveRankTrain(RankingData, BinaryTrain)

```
$D \leftarrow [\,]$;
for $n \leftarrow 1$ to $N$ do
    for $i, j \leftarrow 1$ to $M$ and $i \neq j$ do
        if $i$ is preferred to $j$ on query $n$ then
            $D \leftarrow D.append(\vec{x}_{nij}, +1)$;
        else
            if $j$ is preferred to $i$ on query $n$ then
                $D \leftarrow D.append(\vec{x}_{nij}, -1)$;
            end
        end
    end
end
return BinaryTrain(D)
```

These algorithmswork well in case of bipartitte ranking problems.

**Alg 10:** NaiveRankTest($f$, $\hat{x}$)

```
$score \leftarrow \langle 0, 0, ..., 0 \rangle$;
for $i, j \leftarrow 1$ to $M$ and $i \neq j$ do
    $y \leftarrow f(\hat{x}_{ij})$;
    $score_i \leftarrow score_i + y$;
    $score_j \leftarrow score_j - y$;
end
return score.sort()
```

## Weighted binary classification

**$\omega$-ranking** Given an input space $\mathcal{X}$, an unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \Sigma_M$, and a training set $D$ sampled from $\mathcal{D}$. We want to compute a function $f : \mathcal{X} \to \Sigma_M$ minimizing:

$$\mathbb{E}_{(\vec{x},\sigma)\sim\mathcal{D}} \left[ \sum_{u \neq v} [\sigma_u < \sigma_v] \cdot [\hat{\sigma}_v < \hat{\sigma}_u] \cdot \omega(\sigma_u, \sigma_v) \right] \tag{73}$$

where $\hat{\sigma} = f(\vec{x})$.

$$\omega(i,j) = \begin{cases} 1 & \text{if } \min\{i,j\} \leq K \text{ and } i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{74}$$

**Alg 11:** RankTrain($D^{\text{rank}}$, $\omega$, BinaryTrain)

```
$D^{\text{bin}} \leftarrow [\,]$;
for $(\vec{x}, \sigma) \in D^{rank}$ do
    for $u \neq v$ do
        $y \leftarrow \text{sign}(\sigma_v - \sigma_u))$;
        $w \leftarrow \omega(\sigma_u - \sigma_v)$;
        $D^{\text{bin}} \leftarrow D^{\text{bin}} \oplus (y, w, \vec{x}_{uv})$;
    end
end
return BinaryTrain($D^{bin}$)
```

This algorithm is better than the naive algorithms because it make $O(M \log_2 M)$ calls to $f$ rather than $O(M^2)$ class, and it achieves a better error bound.

**Alg 12:** RankTest($f$, $\hat{x}$, $obj$)

```
if obj contains 0 or 1 elements then
    return obj
else
    $p \leftarrow$ randomly chosen object in $obj$;
    $left \leftarrow [\,]$;
    $right \leftarrow [\,]$;
    for $u \in obj \setminus \{p\}$ do
        $\hat{y} \leftarrow f(\vec{x}_{up})$;
        if uniform random variable ¡ $\hat{y}$ then
            $left \leftarrow left \oplus u$;
        else
            $right \leftarrow right \oplus u$;
        end
    end
    $left \leftarrow \text{RankTest}(f, \hat{x}, left)$;
    $right \leftarrow \text{RankTest}(f, \hat{x}, right)$;
end
return $left \oplus \langle p \rangle \oplus right$
```

# Neural Networks

The idea of the multi-layer perceptron is to densely connect artifical neurons to realize compositions of non-linear functions. The information is propagated from the inputs to the outputs. No cycles between outputs and inputs (DAG). Compute one or more non-linear function. Computation is carried out by composition of some number of algebraic functions implemented by the connections, weights and biases of the hidden and output layers. Hidden layers compute intermediate representations.

**Backpropagation** Evaluates the gradient of the cost w.r.t. weights and biases.

- Forward propagation: sum inputs, produce activations, feed-forward.
- Error estimation.
- Backpropagate the error signal and use it to update weights.

## Feedforward Networks

Information moves in only one direction, from the input nodes, through the hidden and to the output nodes.

The goal is to approximate some function $f^* : \mathcal{X} \to \mathcal{Y}$.

### Training

The training examples specify directly what the output layer must do at each point $\vec{x}$: it must produce a value that is close to $y$. The other layers are not directly specified by the training data. The learning algorithm must decide how to use those layers to produce the desired output.

The nonlinearity of a NN causes most interesting loss functions to become non-convex. So NN are trained by iterative, gradient-based optimizers.

**Cost function** The most common cost function is the **cross-entropy loss**

$$\mathcal{L}_i = -\sum_k y_k \cdot \log(S(l_k)) = -\log(S(l)) \tag{75}$$

**Activation function** Implemented inside the hidden units. The most popular is the **ReLU**, where $h(x) = \max\{0, x\}$. It gives 1 if the gradient unit is active.

**Dying ReLU problem**: neurons can be pushed into states in which they become inactive for essentially all inputs. No gradients flow backward through the neuron, and so the neurom becomes stuck in an inactive state and *dies*.

In order to mitigate these problems, there are some variants, for examples:

- Leaky ReLU: $y_i = a_i \cdot x_i$
- Randomized Leaky ReLu: $y_{ji} = aji \cdot x_{ji}$

Other activation function

- **Sigmoid**: $h(x) = \frac{1}{1+e^{-x}}$
- **Hyperbolic tangent**: $h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

## Backpropagation

Algorithm for training FFNN. Computes the gradient of the loss function w.r.t. the weights of the network for a single input-output example, one layer at a time, iterating backword from the last layer to avoid redundant calculations in the chain rule.

1. Feedforward propagation: accept input $x$, pass through intermediate stages and optain output.

$$\hat{y}(\vec{x}; \vec{w}) = f\left(\sum_{j=1}^{m} w_j^{(2)} h\left(\sum_{i=1}^{d} w_{ij}^{(1)} x_i + w_{0j}^{(1)}\right) + w_0^{(2)}\right) \tag{76}$$

2. Use the computed output to compute a scalar cost depending on the loss function.

$$L(X; \vec{w}) = \sum_{i=1}^{N} \frac{1}{2}(y_i - \hat{y}(\vec{x}_i; \vec{w}))^2 \tag{77}$$

3. Backpropagation: allows information to flow backwards from cost to compute the gradient. The general unit activation in a multilayer network is

$$z_t = h\left(\sum_j w_{jt} z_j\right) \tag{78}$$

In forward propagation we calculate for each unit $a_t = \sum_j w_{jt} z_j$.

$$\frac{\delta L}{\delta w_{jt}} = \frac{\delta L}{\delta a_t} \frac{\delta a_t}{\delta w_{jt}} = \frac{\delta L}{\delta a_t} z_j \tag{79}$$

Key idea: it can be computed recursively starting from the final layer to the earlies layer.

## Training

Adjust all the weights of the network $\Theta$ such that a cost function is minimized

$$\min_{\Theta} \sum_i L(y_i, f(x_i, \Theta)) \tag{80}$$

### Gradient Descent

#### Batch Gradient Descent

**Alg 13:** BatchGradientDescent($k$)

```
Data: Learning rate $\epsilon_k$
Data: Initial parameter $\Theta$
while stopping criteria is not met do
    $\hat{g} \leftarrow \frac{1}{N} \nabla_\Theta \sum_i L\left(f(\vec{x}^{(i)}; \Theta), \vec{y}^{(i)}\right)$;
    Apply Update: $\Theta \leftarrow \Theta - \epsilon\hat{g}$;
end
```

#### Stochastic Gradient Descent

**Alg 14:** StochasticGradientDescent($k$)

```
Data: Learning rate $\epsilon_k$
Data: Initial parameter $\Theta$
while stopping criteria is not met do
    Sample example $(\vec{x}^{(i)}, \vec{y}^{(i)})$ from training set;
    $\hat{g} \leftarrow \nabla_\Theta L\left(f(\vec{x}^{(i)}; \Theta), \vec{y}^{(i)}\right)$;
    Apply Update: $\Theta \leftarrow \Theta - \epsilon\hat{g}$;
end
```

#### Momentum

**Alg 15:** StochasticGradientDescent with Momentum

```
Data: Learning rate $\epsilon_k$
Data: Momentum parameter $\alpha$
Data: Initial parameter $\Theta$
Data: Learning velocity $\vec{v}$
while stopping criteria is not met do
    Sample example $(\vec{x}^{(i)}, \vec{y}^{(i)})$ from training set;
    $\hat{g} \leftarrow \nabla_\Theta L\left(f(\vec{x}^{(i)}; \Theta), \vec{y}^{(i)}\right)$;
    $\vec{v} \leftarrow \alpha\vec{v} - \epsilon\hat{g}$;
    Apply Update: $\Theta \leftarrow \Theta - \epsilon\hat{g}$;
end
```

## Convolutional Neural Networks

The input is a tensof rwith a shape, given the number of inputs $n$, the input height $h$, the input width $w$, and the input channels $c$, the shape $s$ is defined as

$$s = n \cdot h \cdot w \cdot c \tag{81}$$

After passing through a convolutional layer, the image becomes abstracted to a feature map, called activation map.

### Convolution

Mathematical operation on two functions $f$ and $g$ that produces a third function $f \cdot g$ that expressed how the shape of one is modified by the other. A kernel matrix is applied to an image, and it works by determining the value of a central pixel by adding the weighted values of all its neighbors together.

$$S(i,j) = (I \cdot K)(i,j) \tag{82}$$
$$= \sum_m \sum_n I(m,n) \cdot K(i-m, j-n) \tag{83}$$

The intuition is that the network will learn filters that activate when they see some specific type of feature at some spatial position in the input.

### Non-linearity

The result of the convolution is passed to an activation function

$$y_{i,j} = f(a_{i,j}) \tag{84}$$
$$\text{e.g. } f(a) = [a]_+ \tag{85}$$
$$f(a) = \text{sigmoid}(a) \tag{86}$$

We are using non-linearity in order to have a final non-linear representation for our prediction model, and it doesn't affect the receptive field of the NN.

### Spatial Pooling

After the application of the non-linearity, spatial pooling is used to provide invariance to small traslation of the input.

It partitions the input image into a set of rectangles and, for each sub-region, outputs the maximum.

The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters, memory footprint and amount of computation in the network, and hence to control overfitting.

$$x_{i,j} = \max_{|k|<\tau,\ |l|<\tau} y_{i-k,\ j-l} \tag{87}$$

It is common to periodically insert pooling layer between successive convolutional layers.

## Other Neural Networks

### Recurrent Neural Networks

Recurrent NN has been designed in problems in which we have multiple object as input and we want to predict multiple object as output.

The particularity of the RNN is the recurrence formula, that is a function of certain parameters of the network that relates the input at time $t$ ad the latent representation of the previous state with the latent representation of the current state:

$$h_t = f_W(x_t, h_{t-1}) \tag{88}$$

### Autoencoders

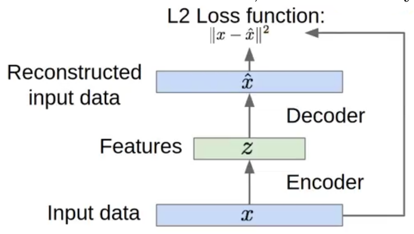
Can be considered as an unsupervised approach for learning a lower-dimensional feature representation from unlabeled data. There is a network that maps the original input data into low-dimensional latent representations.

The main idea is that the representation of the output $z$ should be smaller than the input $x$. The idea is to train the autoencoder so that features can be used to reconstruct original data.

Once we decided the architecture of the encoder and decoder, we need a loss function, and the loss function is a reconstruction function that minimizes the difference between the original data and the reconstructed version of the input data

$$||x - \hat{x}||^2 \tag{89}$$

Once we have trained the autoencoder, we throw away the decoder,



and we only take the latent representation $z$ and we append the loss function $\hat{y}$ and the encoder can be used to have a bettere initialization of a supervised model.

## Unsupervised Learning

## Tasks

### Dimensionality Reduction

Transformation of data from high-dimensional space to low-dimensional space so that the low-dimensional space retain some meaningful properties of the original data.

Find a function $f \in \mathcal{Y}^{\mathcal{X}}$ mapping each high-dimensional input $x \in \mathcal{X}$ to a lower dimensional embedding $f(x) \in \mathcal{Y}$ where $dim(\mathcal{Y}) \ll dim(\mathcal{X})$.

### Clustering

Grouping a set of objects in the same cluster that are more similar to each othern than to those in other groups.

Find a function $f \in \mathbb{N}^{\mathcal{X}}$ that assigns each inptu $x \in \mathcal{X}$ a cluster index $f(x) \in \mathbb{N}$.

### Density Estimation

Construction of an estimate, based on observed data, of an unobservable underlying probability density function.

Find a probability distribution $f \in \Delta(\mathcal{X})$ that fits the data $x \in \mathcal{X}$.

## Principal Component Analysis (PCA)

**Principal Component** sequence of $p$ unit vectors, where the $i$-th vector is the direction of a line that best fits the data while being orthogonal to the first $i-1$ vectors.

PCA is the process of computing the principal components and using them to perform a change of basis on the data, sometimes using only the first few PC.

We want to find the direction that explains the maximum variance of the data. Our second principal component is orthogonal to the first PC.

The way PCA allows us to do dimensionality reduction is by dropping the dimensions of least variance.

### Variance along unit direction

$$\mathbb{E}[t] = \frac{1}{n}\sum_{i=1}^{n} t_i \tag{90}$$
$$= \bar{\vec{x}}^T\vec{w} - \vec{c}^T\vec{w} \tag{91}$$
$$\mathbb{V}ar[t] = \frac{1}{n}\sum_{i=1}^{n}(t_i - \mathbb{E}[t])^2 \tag{92}$$
$$= \vec{w}^T C \vec{w} \tag{93}$$

Where $t_i = (\vec{x}_i - \vec{c})^T\vec{w}$, $\bar{X} = [\bar{\vec{x}}_1, \bar{\vec{x}}_2, ..., \bar{\vec{x}}_n]$, and $C = \left[\frac{1}{n}\bar{X}\bar{X}^T\right]$ is the covariance matrix.

### Eigenvalue decomposition

Let $A \in \mathbb{R}^{m\times m}$, symmetric. There exists $U = [\vec{u}_1, \vec{u}_2, ..., \vec{u}_m] \in \mathbb{R}^{m\times m} and \vec{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_m)^T \in \mathbb{R}^m$ such that:

$$A = U\Lambda U^T = \sum_{j=1}^{m}\lambda_j \vec{u}_j \vec{u}_j^T \qquad \text{and} \qquad U^TU = UU^T = I \tag{94}$$

where $U$ is the matrix of the eigenvectors, $\Lambda$ is a square matrix with values all 0 except of the principal diagonal, that is composed by $\lambda_1, \lambda_2, ..., \lambda_m$, $I$ is the identity matrix, $\vec{u}_j$ is an eigenvector and $\lambda_j$ is the corresponding eigenvalue.

### First PC

$$\vec{w}_1 \in \arg\max\left\{\vec{w}^T C\vec{w} : \vec{w}^T\vec{w} = 1\right\} \tag{95}$$

### Second PC

$$\vec{w}_2 \in \arg\max\left\{\vec{w}^T C\vec{w} : \vec{w}^T\vec{w} = 1, \vec{w} \perp \vec{w}_1\right\} \tag{96}$$

### PCA using Eigenvalue decomposition

**Alg 16:** PCA using ED

```
Data: $X = [x_1, x_2, \dots, n_n]$
$\bar{X} = X - \frac{1}{n}X1_n1_n^T$;
$C = \frac{1}{n}\bar{X}\bar{X}^T$;
$U, \lambda = \text{eig}(C)$;
return Principal components $W = U = [u_1, u_2, \dots, u_m]$ and
    variances $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_m)$
```

### Singular Value Decomposition

Let $A \in \mathbb{R}^{m\times n}$. There exist $U \in \mathbb{R}^{m\times k}, s \in \mathbb{R}^k$ with $s_1 \geq s_2 \geq ... \geq s_k > 0$ and $V \in \mathbb{R}^{n\times k}$ such that:

$$A = USV^T \qquad \text{and} \qquad U^TU = V^TV = I \tag{97}$$

**Alg 17:** PCA using SVD

```
Data: $X = [x_1, x_2, \dots, n_n]$
$\bar{X} = X - \frac{1}{n} X 1_n 1_n^T$;
$U, s, V = \mathrm{SVD}(\bar{X})$;
return Principal components $W = U = [u_1, u_2, \dots, u_k]$ and
  variances $\lambda = \left(\frac{s_1^2}{n}, \frac{s_2^2}{n}, \dots, \frac{s_k^2}{n}\right)$
```

## PCA using SVD

### Dimensionality Reduction with PCA

If we compute

$$T = \hat{W}^T \bar{X} \in \mathbb{R}^{k \times n} \tag{98}$$

will induce a change of the coordinate system w.r.t. the $k$ principal components in $\hat{W}^T$, and this will reduce the dimension of the features to $k$.

### How Many PC?

- depends on the goal
- no means of validating it, unless supervised learning
- we can compute the cumulative proportion of expaired variance

$$\text{ED: } \frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{m} C_{jj}} \qquad \text{SVD: } \frac{\sum_{j=1}^{k} s_j^2}{\sum_{ij} \bar{X}_{ji}^2} \tag{99}$$

  - allows to estimate the amount of information loss

# Clustering

## $k$-means Clustering

Given data points $X = [x_1, x_2, \dots, x_n] \in \mathbb{R}^{\{} d \times n\}$ and $k$ a fixed number of clusters. Our goal is to find a partition of data points into $k$ sets $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k$ minimizing the variation $V(\mathcal{C}_j)$ within eahc set:

$$\min_{\mathcal{C}_1, \dots, \mathcal{C}_k} \sum_{j=1}^{k} V(\mathcal{C}_j) \qquad V(\mathcal{C}_j) = \sum_{i \in \mathcal{C}_j} ||\vec{x}_i - \vec{\mu}_j||^2 \tag{100}$$

where $\mu_j$ is the centroid of $\mathcal{C}_j$.

**Alg 18:** $k$-means

```
Use some initialization strategy to get some initial cluster
  centroids $\mu_1, \dots, \mu_k$;
while clusters change do
    Assign each datapoint to the closest centroid forming new
      clusters;
    $\mathcal{C}_j = \{i \in \{1, \dots, n\} : j = \arg\min_l ||\vec{x}_i - \mu_l\}$;
    Compute the cluster centroids $\mu_1, \dots, \mu_k$;
end
```

### Properties

- guarantees to converge,
- it is not guaranteed to find the global minimum, but a local one
- centroids selection plays an important role with the result.

### Centroids selection

- random points (not examples) in the space;
- random examples;
- points least similar to any existing center;
- try multiple starting points;
- initialize with the results of another clustering method;

## Issues for Clustering

- how to represent data points?
- the way we choose to measure the distance;
- does the problem necessitate of flat or hierarchical clustering?
- how to define the number of cluster.

### Types of clustering

- **Flat algorithms**: start with a random partitioning and refines it iteratively.
- **Hierarchical algorithms**: bottom-up agglomerative or top-dow divisive.
- **Hard clustering**: each example belongs to exactly one cluster.
- **Soft clustering**: an example can belong to more than one cluster.

## EM Clustering

Assumes that data came from mixture of gaussians. Assigns data to cluster with a certain probability. Follows an iterative scheme as $k$-means, iterates between assigning points and recalculating cluter centers. EM stands for

**Alg 19:** EM Clustering

```
Initialize clusters;
while clusters does not change do
    Calculate $p(\Theta_c|x)$ the probability of each point belonging
      to each cluster;
    Recalculate the new cluster parameters $\Theta_c$, the maximum
      likelihood cluster centers given the current soft
      clustering;
end
```

- **Expectation**: given the current model, figure out the expected probabilities of the daa points to each cluster $p(\Theta_c|x)$.
- **Maximization**: given the probabilistic assignment of all the points, estimate a new model $\Theta_c$.

## Other Clustering Algorithms

**Spectral clustering** We form a matrix that indicate the similarity of each data point w.r.t. each other. In can be represented in terms of graphs, where each edge indicates the similarity between two points.

We can create a fully connected graph, or a $k$-nearest neighbor graph. We can create a graph with some notion of similarity among nodes, gaussian kernel:

$$W(i,j) = \exp\left\{\frac{-|x_i - x_j|^2}{\sigma^2}\right\} \tag{101}$$

$Cut(A, B)$ is the sum of the weights of the set of edges that connect two groups. An intuitive goal is to find the partition that minimizes the cut.

**Hierarchical clustering** Produce a set of nested clusters organized as a hierarchical tree (**dendogram**).

**Agglomerative clustering** First merge very similar instances and then incrementally build larger clusters out of smaller clusters.

**Alg 20:** Agglomerative clustering

```
Each instance in its own cluster;
repeat
    Pick the two closest clusters;
    Merge them into a new cluster;
until until there is only one cluster;
return Family of clusterings represented by a dendogram
```

# Deep generative models

- **Generative**: statistical models of the data distribution $p_X$ or $p_{XY}$ depending on the availability of target data.
- **Discriminative**: statistical models of the conditional distribution $p_{X|Y}$ of the target given the input.

A discriminative model can be constructed from a generative model via the Bayes rule, but not vice versa

$$p_{X|Y}(y, x) = \frac{p_{XY}(x, y)}{\sum_{\hat{y}} p_{XY}(x, \hat{y})} \tag{102}$$

## Density Estimation

We want to find the probability distribution $f \in \Delta(\mathcal{Z})$ that fits teh data $z \in \mathcal{Z}$, where $z$ is sampled from unknown data distribution $p_{\text{data}} \in \Delta(\mathcal{Z})$, and $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$.

Objectives:

- define an hypothesis space $\mathcal{H} \subset \Delta(\mathcal{Z})$ of models that can represent probability distributions
- define a divergence measure $d \in \mathbb{R}^{\Delta(\mathcal{Z}) \times \Delta(\mathcal{Z})}$ between probability distributions $\Delta(\mathcal{Z})$
- find an hypothesis $q^* \in \mathcal{H}$ that best fits the data distributed according to $p_{\text{data}}$.

$$q^* \in \arg\min_{q \in \mathcal{H}} d(p_{\text{data}}, q) \tag{103}$$

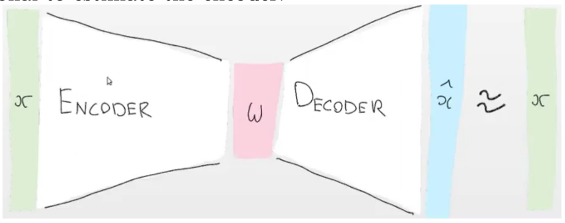
## Variational AutoEncoder (VAE)

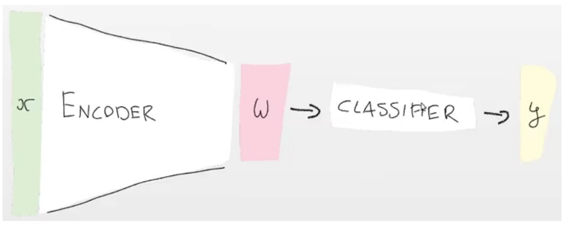
Compressing high-dimensional data into a lower dimensional representation. **Encoder**: maps input data $x$ to a compressed representation $\omega$ that preserves meaningful factors of variations in the data.

Encoder is trained by leveraging a decoder mapping the representation $\omega$ back to the input domain yielding $\hat{x}$ (**reconstruction**).

The objective is to minimize the divergence between input $x$ and its reconstruction $\hat{x}$.

After training the decoder is not needed anymore, since it was only functional to estimate the encoder.





### In generative models

Given a latent space $\Omega$ with a prior distribution $p(\omega)$, and an input space $\mathcal{X}$ with a data distribution $p_{\text{data}}(x)$. We want a decoder that produce a mapping from the latent space to the input space where the regions of low probability of the latent input space correspond to the region of the low probability of the latent space, and the same thing with the region of high probability.

$$q_\theta(x) = \mathcal{E}_{\omega \sim p_\omega}[q_\theta(x|\theta)] \tag{104}$$

$$\theta^* \in \arg\min_{\theta in \Theta} d(q_\theta, p_{\text{data}}) \tag{105}$$

### Kullback-Leibler divergence (distance)

$$d_{KL}(p, q) = \mathbb{E}x \sim p \left[\log \frac{p(x)}{q}(x)\right] \tag{106}$$

but we end up in an intractable problem.

### Variational Upper Bound

Let $q_\psi(\omega|x) \in \Delta(\Omega)$, where $\psi$ defines other parameters, denote an encoding probability distribution:

$$\log \mathbb{E}_{\omega \sim p_\omega}[q_\theta(x|\omega)] = \underbrace{\mathbb{E}_{\omega \sim p_\psi(\cdot|x)}[\log q_\theta(x|\omega)]}_{\text{Reconstruction}} - \underbrace{d_{KL}(q_\psi(\cdot|x), p_\omega)}_{\text{Regularizer}} \tag{107}$$

Now we can define a variational bound:

$$d_{KL}(p_{\text{data}}, q_\theta) \leq \mathbb{E}_{x \sim p_{\text{data}}}\left[-\mathbb{E}_{\omega \sim p_\psi(\cdot|x)}[\log q_\theta(x|\omega)] + d_{KL}(q_\psi(\cdot|x), p_\omega)\right] + const \tag{108}$$

The reconstruction is still intractable to compute, but is easy to get unbiased estimates of gradients with respect to $\theta$ and $\psi$. The regularizer might have closed-form solution, e.g. using gaussian distributions.

### Issues with VAE

- generator tends to produce blurry data;
- underfitting, in particular the balance between the regularization and the reconstruction terms must be managed carfully.

## Generative Adversarial Networks (GAN)

The idea is to have an implicit density model, so not requiring to estimate $q_\theta(x)$.

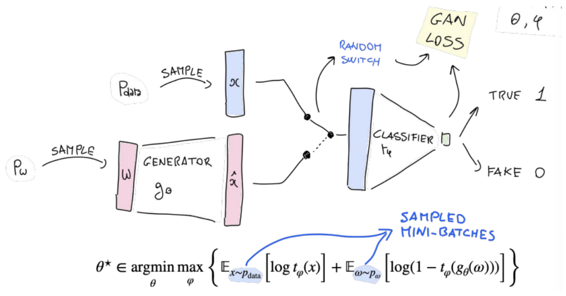
Enable the possibility of estimating implicit density. We assume to have a prior density $p_\omega \in \Delta(\Omega)$ and a generator (decoder) $g_\theta \in \mathcal{X}^\Omega$ that generates data points in $\mathcal{X}$ given a random element from $\Omega$.

$$q_\theta(x) = \mathbb{E}_{\omega \sim p_\omega}\delta[g_\theta(\omega) - x] \tag{109}$$

where $\delta$ is the Dirac delta function.

### Objective

Find $\theta^*$ such that $q_{\theta^*}$ best fits the data distribution $p_{\text{data}}$.



GAN LOSS
θ, φ
RANDOM SWITCH
SAMPLE
$p_{data}$
x
TRUE 1
CLASSIFIER $t_\varphi$
SAMPLE
$p_\omega$
ω
GENERATOR $g_\theta$
x̂
FAKE 0
SAMPLED MINI-BATCHES
$\theta^\star \in \arg\min_\theta \max_\varphi \left\{\mathbb{E}_{x\sim p_{\text{data}}}\left[\log t_\varphi(x)\right] + \mathbb{E}_{\omega\sim p_\omega}\left[\log(1 - t_\varphi(g_\theta(\omega)))\right]\right\}$

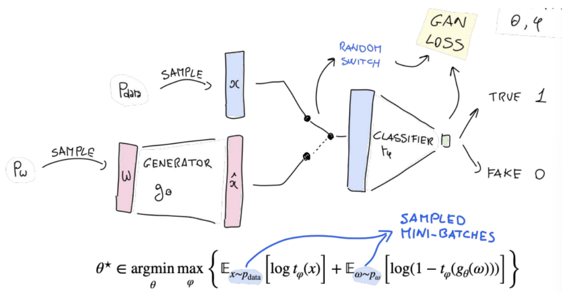
### Issues

- training stability: since we have the alternation between the generator and the discriminator, the parameters of the network may oscillate and never converge
- mode collapse: the generator might learn to perfectly generate few examples from the training set, but not cover all the variability that is in the training set
- vanishing gradient: if the discriminator is successful, then it will have a little gradient to learn, so it leaves the generator with little gradient to learn from.

# Reinforcement Learning

The problems involve an **agent** interacting with an **environment**, which provides numeric **rewards** signals.

The goal is to learn a **policy** to maximize the expected reward over time. **policy**: mapping from states to actions.



GAN LOSS
θ, φ
RANDOM SWITCH
SAMPLE
$p_{data}$
x
TRUE 1
CLASSIFIER $t_\varphi$
SAMPLE
$p_\omega$
ω
GENERATOR $g_\theta$
x̂
FAKE 0
SAMPLED MINI-BATCHES
$\theta^\star \in \arg\min_\theta \max_\varphi \left\{\mathbb{E}_{x\sim p_{\text{data}}}\left[\log t_\varphi(x)\right] + \mathbb{E}_{\omega\sim p_\omega}\left[\log(1 - t_\varphi(g_\theta(\omega)))\right]\right\}$

## Markov Decision Procedd

Framework used to help to make decisions on a stochastic environment. Goal si to find a policy, which is a map that gives us all optimal actions on each state in our environment.

Probability that the process moves into its new state $s'$ is influenced by the chosen action. It is given by the state transaction function $P_a(s, s')$. $s'$ depends on $s$ and the decision maker's action $a$.

**MDP**: 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

- $\mathcal{S}$ is the state space.
- $\mathcal{A}$ is the action space.

- $\mathcal{R}_a(s, s')$ is the reward received after transitioning from state $s$ to $s'$.
- $P_a(s, s')$ is the probability that action $a$ in state $s$ at time $t$ will lead to state $s'$ at time $t+1$.
- $\gamma$ discount factor.

A policy function $\pi$ is a mapping from state space $\mathcal{S}$ to an action space $\mathcal{A}$.

### Loop

- Agent selects action $a_t$;
- Environment samples reward $r_t \sim \mathcal{R}(\cdot|s_t, a_t)$;
- Environment samples next state $s_{t+1} \sim P(\cdot|s_t, a_t)$;
- Agent receives reward $r_t$ and next state $s_{t+1}$.

**Objective** find a good policy for the decision maker: a function $\pi$ that specifies the action $\pi(s)$ that the decision maker will choose when in state $s$.

Choose a policy $\pi$ that will maximize some cumulative function of the random rewards (called optimal policy $\pi^*$).

### Cumulative reward

$$r(s_0) = \gamma r(s_1) + \ldots = \sum_{t\geq 0} \gamma^t r(s_t) \qquad 0 < \gamma \leq 1 \tag{110}$$

## Reinforcement vs Supervised

Supervised:

- Get input $x_i$ samples from data distribution.
- Use model with parameters $w$ to predict output $y$.
- Observe target output $y_i$ and loss $l(w, x_i, y_i)$.
- Update $w$ to reduce loss with SGD:

$$w \leftarrow w - \eta \nabla l(w, x_i, y_i) \tag{111}$$

Reinforcement:

- From state $s$, take action $a$ determined by policy $\pi(s)$.
- Environment selects next state $s'$ based on transition model $P(s'|s, a)$.
- Observe $s'$ and reward $r(s)$, then update the policy.

In Supervised learning the next input does not depend on the previous inputs or agent predictions, there is a supervision signal at every step, and the loss is differentiable with respect to model parameters.

## Value Based Methods

Value function gives the toal amount of reward the agent can expect from a particular state to all possible states from that state.

$$V^\pi(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r(s_t) | s_0 = s.\pi\right] \tag{112}$$

$$V^*(s) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r(s_t) | s_0 = s, \pi\right] \tag{113}$$

### $Q$-Learning

Model-free reinforcement learning algoirhtm to learn the value of an action in a particular state, it can handle problems with sotchastic transitions and rewards without requiring adaptations.

### $Q$-Value Function

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r(s_t) | s_0 = s, a_0 = a, \pi\right] \tag{114}$$

$$Q^*(s, a) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r(s_t) | s_0 = s, a_0 = a, \pi\right] \tag{115}$$

$$\pi^*(s) = \arg\max_a Q^*(s, a) \tag{116}$$

**Alg 21:** $Q$-Learning

```
$Q \leftarrow [1...s][1...a] = \{0...0\}\{0...0\}$;
$s_0 \leftarrow random$;
for episode in E do
    while state $s_i \neq s_{goal}$ do
        Select a random possible action $a_r$ for $s_i$;
        Using $a_r$, consider going to this next state;
        Get maximum $Q$ value for this next state;
        $Q^*(s,a) \leftarrow \mathcal{R}(s,a) + \gamma \max_{a'}[Q^*(s',a')]$;
    end
end
```

### Bellman Equation

$$Q^*(s, a) = r(s) + \gamma \sum_{s'} P(s'|a, s) \cdot \max_{a'} Q^*(s', a') \tag{117}$$

$$= \mathbb{E}_{s' \sim P(\cdot|s,a)}[r(s) + \gamma \max_{a'} Q^*(s', a')] \tag{118}$$

## Policy Gradient Methods

It can be more efficient to parametrize $\pi$ and learn it directly. It is beneficial to learn a function giving the probability distribution over actions from current state:

$$\pi_\theta(s, a) = P(a|s) \tag{119}$$

where $\theta$ is the parameter of our NN.

### Objective

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right] \tag{120}$$

$$= \underbrace{\mathbb{E}_\tau[r(\tau)]}_{\text{Expectation over } \tau=(s_0,a_0,r_0,s_1,a_1,r_1,\ldots)} \tag{121}$$

$$= \int_\tau r(\tau) p(\tau; \theta) d\tau \tag{122}$$

$$p(\tau; \theta) = \prod_{t\geq 0} \pi_\theta(s_t, a_t) P(s_{t+1}|s_t, a_t) \tag{123}$$

### Optimization

$$\nabla_\theta J(\theta) = \mathbb{E}[r(\tau) \nabla_\theta \log p(\tau; \theta)] \tag{124}$$

$$p(\tau; \theta) = \prod_{t\geq 0} \pi_\theta(s_t, a_t) P(s_{t+1}|s_t, a_t) \tag{125}$$

$$\log p(\tau; \theta) = \sum_{t\geq 0} [\log \pi_\theta(s_t, a_t) + \log P(s_{t+1}|s_t, a_t)] \tag{126}$$

$$\nabla_\theta \log p(\tau; \theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(s_t, a_t) \tag{127}$$

**Alg 22:** Reinforce

Sample $N$ trajectories $\tau_i$ using current policy $\pi_\theta$;
Estimate the policy gradient:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} r(\tau_i) \left(\sum_{t=0}^{T_i} \nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t})\right) \tag{128}$$

Update the parameters by gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta) \tag{129}$$