1 Machine Learning

A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E

- **What is Machine Learning:**
  - experience E → dataset, quality, enough data
  - task T → algorithms, problems, optimizing
  - measure P → metrics, direction the optimum
  - models → solve the task, simplified representation
- **Types of Machine Learning algorithms:**
  - Supervised: $\boldsymbol{X}$ and $\boldsymbol{Y}$, distance $y - \hat{y}$, regression, classification
  - Unsupervised: $\boldsymbol{X}$, look for patterns in the data, clustering, anomaly d.
  - Semi-supervised: label/un-label $\boldsymbol{X}$ and $\boldsymbol{Y}$
  - Reinforcement: agent iteract with an environment, the interaction is driven by the features, trained for max. the reward
- Further distinction:
  - ***Model Based or parametric:*** algorithm builds a model based on the data, learning process based on getting the right paramters
  - ***Instance Based or Non-parametric:*** algorithms that do not learn parameters but exploit the data itself to compare a new example using similarity metrics

2 Regression Models

Regression: purpose of regression is to identify the parameters, continuous supervised learning problems, all data is plagued by different types of errors: measurement, quantization, representation, human errors. Those error terms are summed together into an ***irriducible error term***, the objective is to leave only these contributions in the error term correctly identifying the other parameters, check if there is any information left in the error term is to do a **whiteness test** or **correlation** between the error and y is 0, optimal solution in one go with ***OLS*** minimizing MSE

Polynomial regression: regress the model only between an x and a y, SISO (Single Input Single Output), we can apply a series of ***basis functions*** to our x diff behaviours, as features we can use the ***polynomial expansion*** of our input variable, ***Metrics MSE***, what ***degree*** of x do we get to describe y? ⇒ ***overfitting or underfitting***, choosing the right features

Solution space / parameters: depending of the variations of $\boldsymbol{\theta}$ - local or global optimum, optimal solution has the lowest absolute error value, optimal parameters are searched iterative algorithms (gradient descent) or closed form, we cant always solve a problem exactly: too much noise in the data, we don't have enough information, no numerical solution

3 Regularization & Model selection

Overfitting: the goal is minimize the error as much as we can, unknow model data complexity, we try to optimize as much as he can, ↑ feature ↓ data per feature ⇒ ↑ risk of overfitting, we can: collect more data, select our features better only the most important ones has to be included, limit model learning

Data: it is better to collect as much as possible as long as they are **IID**, collect no IID data no new information added, collect data its expensive → not enough data, algorithms problem related to data: insufficient data, poor quality data, non-representative data, sampling (selection) bias → portion selection, irrelevant features, confounding features, ***Rule of thumb:*** 20 examples per parameter

Features selection: select only the features that are most significant, ↓ features ↓ expressive capacity of the model, select feature with the highest correlation with y, another way is to use **regularization techniques:**

- **Lasso regression L1:** irrelevant feature to 0
- **Ridge regression L2:** irrelevant feature ≈ 0
- **Elastic net:** combines L1 and L2 by weighting their usage

adding a ***penalty term*** to our cost function, penalty term that measure the compexity (parameters explosion), the penalization is controlled by λ ***hyperparameter***, ***regularizer*** $\Omega(\boldsymbol{w})$ how complex the model is, ***Lasso*** can't be used into OLS uses iterations

Other penalizations: another method to penalize models is to evaluate the gain of adding an extra feature to the model, assuming that we had d parameters and N data the dependence on the number of features is added to the cost function, $E(\hat{\boldsymbol{\theta}}_N; d)$ is our performance measure, ↑ d parameters ↑ cost function

- **Akaike Information Criterion: AIC(d)**
- **Bayesian Information Criterion: BIC(d)**
- **Minimum Description Length: MDL(d)**

Train, validation, test: validation is the operation of measuring the performance of our model on a set of data on which it has not optimized, if we have enough data we should divide into:

- **Train** the dataset on which our model learns - 60% - 70% - 70%
- **Validation:** dataset for model selection - 20% - 15% - 20%
- **Test:** measure our final performance - 20% - 15% - 10%

the datasets must be sampled by the same process, the training dataset has the largest quantity, model selection is based on validation dataset, once the model is selected we can re-train the model on training + validation, validation dataset: sometimes called out-of-sample its performance are MSE_{val} *or* MSE_{out}, in iterative learning by gradient descent we can measure the performance on the validation dataset even during learning → check **learning curves** graph method to check overfitting

Selection during training: several techniques for taking a good model during training, ***Early stop:*** we stop training when we observe train and validation diverging, ***Best validation*****:** we save the model that performs best in validation, both techniques are robust to noise in the data, check **learning curves** during training

Cross-validation: we do not have enough data to divide between train validation and test, reusing a piece of the training set itself to validate, requires carrying out N training sessions, expensive in terms of computation:

- ***k-fold:*** K groups, K iterations, performance average the k tests
- ***Leave-one-out (LOOCV):*** $K = N$, too small dataset, cross-validation performance will be the average of all prediction errors on the individual examples excluded from training, high variance result

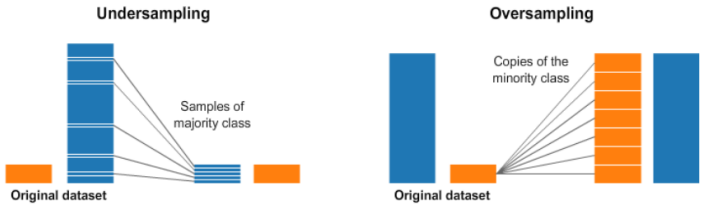
4 Classification

Classification problems determine what class an example belongs to

- **Classification:**
  - set of features and one or more classes as output
  - *Binary classification:* two classes (0/1)
  - *Multi-class classification:* N classes to choose from
  - *Multi-label problems:* example can belong to multiple classes
- **Regression performance metrics**
  - *Mean Squared Error:* robust, allows us to use OLS, RMSE
  - *Mean Absolute Error:* dirtier data, higher level of detail in generative models
  - *Mean Absolute Percentage Error:* more difficult to converge
- **Classification performance metrics**
  - ***Accuracy:*** measures the good predictions on the total predictions, the most important metric but it errors the more the classes are unbalanced
  - ***F1-score:*** harmonic mean of precision and recall
  - ***Precision:*** accuracy of your positive predictions, TP/(TP + FP)
  - ***Recall:*** completeness of your positive predictions, TP/(TP + FN)
  - ***Confusion Matrix:*** prediction mistakes, comparing predicted vs actual value
  - ***ROC (Receiver Operating Curves):*** compare different classifiers, classifier performs well moves towards the top left corner, point is the perfect classifiers (indipendently of the thresholds), Lower Threshold (Right X-axis)
  - ***AUC (Area Under Curve):*** summarizing the ROC, ↑ area ↑ performance
- **Binary classification:**
  - all classif. problems as *one-vs-rest* ⇒ ***binary classification***, a parametric model searches for the *hyperplane* that allows classes to be divided between each other ⇒ max. the decision surface, N classes have to train N classifiers
  - ***Perceptron:*** binary classification model, $\boldsymbol{x}$ as input to a nonlinear transformation function and produce linear value a, **activation function f(a)**, weights the input values to generate an output, non linear result (stepped function), we would like to minimize the number of misclassified examples ⇒ gradient is often zero and difficult to optimize, we optimize the error $E_p(\boldsymbol{w})$ and find $\boldsymbol{w}$ class. well
  - ***Logistics Regression:*** evolution of the perceptron, activation function to discontinous to continuos ⇒ **sigmoid** $\sigma(\cdot)$, probability of belonging to class is set ⇒ **threshold value**, **Log Loss/Cross Entropy** (new cost function to minimize) has no close form ⇒ but convex, gradient descent (iterative method)
    - ***gradient descent:*** find the optimal parameters, randomize θ initial parameters, get best **learning rate** suit to the problem (local - global minimum - convergence problem), reupdate parameter by learning rate, an entire optimization cycle on all data ⇒ **an epoch**, to speed up we reupdate after a **batch** of data and not after each obs, derivatives reupdate iterate
- **Feature scaling:** scaling the features, no scaling the gradient has convergence problems, scaling is important ⇒ better convergence, coefficients as indicator

5 Regression & Classification methods

- **1-vs-all:** multiclass classification problem, train N classifiers 1-vs-all
- **Threshold values:** more tollerence in incorrect pred, threshold inf. FN and FP
- **Unbalanced dataset:** few observations of the phenomenon vs the other one, take into account different techniques to solve:
  - ***Resample the dataset:*** subsampling/oversampling the larghest/poorest class
  - Use ***correct metrics in validation*** to choose more balanced models
  - ***Weighting*** the classes differently during the training phase, when update the $\boldsymbol{w}$ by gradient add multiplicative factor to rappresentative class, compensate for the fact that the model optimizes the parameters N more times on the most represented class than on the undersampled one



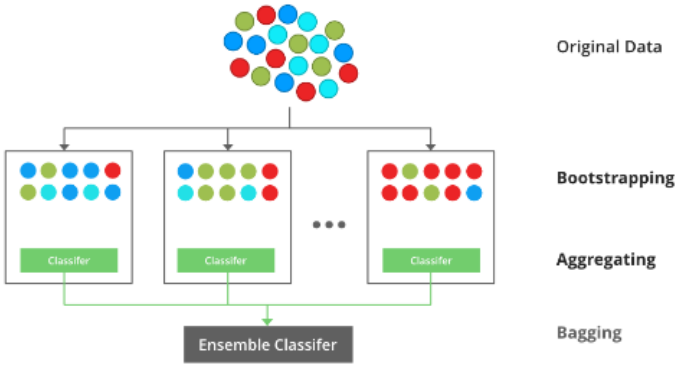
Classification algorithms:

- ***Naïve Bayes:*** simple binary classification algorithm, assumes that each variable independently of the others carries a probability component that the example belongs to the positive class $P(x_i|y)$ based on Bayes' rule of conditional probability + determinate $P(y|x_1, \ldots, x_n)$ as production likelihood ⇒ **Naive**, variables are often not independent, no inaccurate classifications, longer used
  - ***Likelihood:*** pdf congiunta dei dati, indica la probabilitá che si realizzi il vettore di dati osservato ⇒ produttoria osservazioni IID, applicado il teorema di Bayes determino la posterior $P(y|x_1, \ldots, x_n)$ dove voglio sapere y dato i dati che osservo, impiegando la likelihood $P(x_1, \ldots, x_n|y)$
- ***Decision tree:*** most interpretable classification and regression algorithms, iteratively at each step it is selected the most important ***feature or dimension*** to create discriminating between two groups based on a ***threshold***, from the root a decision tree is created, feature importance for splitting are selected first, regressor space (possible values) divided into M **regions**, **no complex separations** simple regions, the regions are separted iteratively ⇒ cutted the region into smaller ones and ***not groups***, the prediction per each region is the average of the values inside, for each region we calculate the best threshold and a degree of **node impurity**:
  - ***node impurity:*** is a mathematical value that tells you how mixed up the categories are within a node, ***high impurity*** means the data points at that node belong to various categories and the split isn't very effective. ***A low impurity*** indicates a good separation, with most data points belonging to the same class
  - ***Gini Impurity:*** a way to calculate the node impurity, measures how likely a randomly chosen data point from the node would be incorrectly labeled if its category were assigned based on the most frequent category at that node.

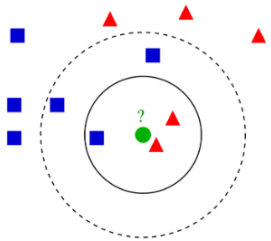
  In learning phase understand when **to stop dividing the space:**
  - put a **tolerated error/impurity threshold** value inside the leaf node
  - build entire tree + **pruning** ⇒ balancing, reduce variance and complexity

limited expressiveness ⇒ **non robust**, bad performances, good performance in easy problems ⇒ improve by creating N models and average their predictions, output using N predictors → regression: average predicted values - classification: majority voting

- ***Ensemble methods*** combining different models to obtain more accurate predictions, **weak learners** combined get **strong learner**:
  - ***Bagging:*** train N models on N different samples of data **(bootstrapping)** + get different predictions ⇒ average them and optain a single value
  - ***Boosting:*** trained N models sequentially has different **weights** depending on their bad performance, each model is created using information from the previous one, ↑ mistakes ↑ weight, next trained model correct those mistakes, the learning rate is controlled by λ shrinkage parameter, low λ
  - ***Stacking:*** predictions from N models comes into other models for refinement

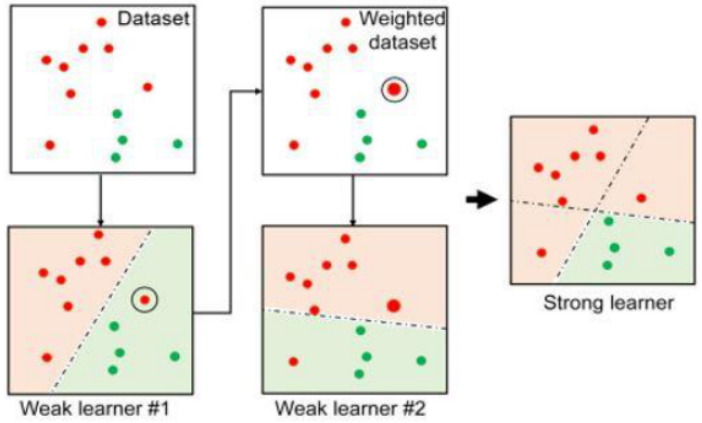


- ***Random forest:*** train N decisions tree ⇒ applying bagging, **decorrelate** trees to each other: *randomly* selects only q regressors among the total and *limits the splits* to those q regressors ⇒ reduce variance, importance of each feature by measuring how much variance has captured creating a ranking
- ***k-Nearest Neighbor:*** non parametric classification and regression algorithm, compare observation $\boldsymbol{x}$ with the vectors of other k closest examples using a **metric**, he decision is made by **majority voting**:classification or **average**:regression, all examples in the dataset must be comparable according to a metric called the **kernel**, the kernel apply a transformation to an observation an project it into an m-dimensional space normally larger than the orginal dimension of the vector, main disadvantage of k-NN: is its poor scalability, no training phase, computational heavy as dataset grows, a lot of comperisons, k comperisons for each observation

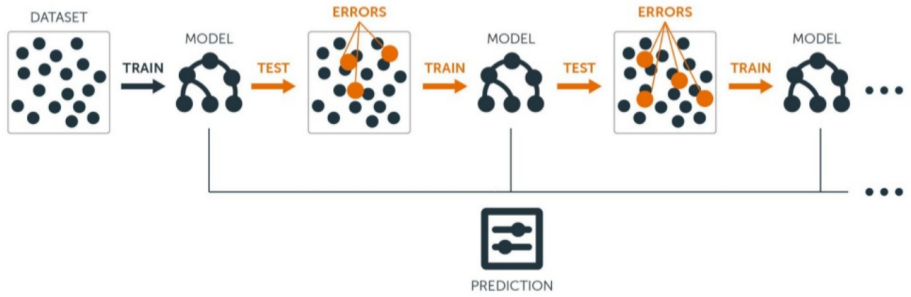


6 Regression & Classification methods 2

- **AdaBoost:** implementation of the classic Boosting algorithm, estimator by default a decision tree, assigns weights to each ***data point*** in your training set, all points have equal weight, ***iteratively trains*** multiple weak learners on the data and the ***weights*** of the data points are adjusted based on the performance of the previous weak learner, misclassified points has higher weight ⇒ forces the next weak learner to focus on these points, final AdaBoost classifier combines the predictions from all the weak learners based on their weight, sensitive to outliers

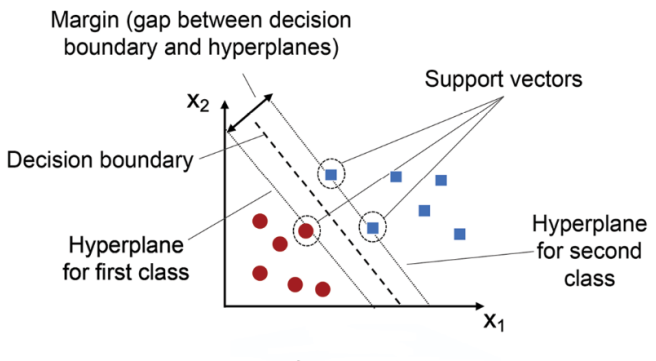


- **Gradient Boosting:** sequence of weak learners that learn only from the residuals of the previous estimators, starting point first general model (classification) then learning and predicting the residuals ⇒ increasingly refined estimator. ***Adv/dis:*** high accuracy, flexible model type, handles different data dypes, computationally expensive, ***prone to overfitting***



- **XGBoost:** gradient Boosting including regularization (L1 and L2), very high performance, convenient for building models for huge datasets ⇒ Scalability
- **Support Vector Machines (SVM):** non parametric technique, find the ***decision boundary*** that best separates different classes of data, goal is to ***maximize the margin***, which is the distance between the decision boundary and the nearest data points from each class ⇒ ***large margin classification***, adding points in the groups does not change the decision boundary, only the data points on the edge of the margin (***support vectors***) influence the decision boundary, computationally heavy

- ***Hard margin:*** training points must be correctly classified, decision boundary follows a complex path to ensure no misclassifications, this approach is only possible if there are **no outliers** ⇒ ***large margin classification***, infinite hyperplanes (decision boundary) but we want the one that maximize the margin, it works well when the data is linearly separable
- ***Soft margin:*** allows some points to be misclassified ⇒ **regularization term C** which controls the trade-off between maximizing the margin and allowing misclassifications ⇒ determines the weight and severity of the violations, ↑ C prioritizes a larger margin + risk overfitting, ↓ C allows for more misclassifications reducing training errors and is less strict about the margin + risk underfitting ⇒ ***Linear Support Vector Classifier***



- ***Linearly Separable dataset:*** we can transform our features so that they are linearly separable, easily separate the classes, applying basis functions we can project our feature space into a larger one ⇒ **kernel is a function**
- ***Linear Support Vector Classifier:*** generalization of the maximal margin classifier, regularization term C, classification in the two-class setting, if the boundary between the two classes is **linear**
- ***Non linear SVM:*** classes not linerly separable, project into a large space, **kern**
- ***Kernel trick:*** what the kernel trick does for us is to offer a more efficient and less expensive way to transform data into higher dimensions without explicitly computing the transformation ⇒ this enables the SVM to find a linear decision boundary in the transformed space, which corresponds to a non-linear boundary in the original space.
- ***SVM for regressions:*** instead of looking for the decision boundary that maximizes the distance, searches for the points that allow the greatest number of points to fit within the margin
- ***Training:*** consists of finding the support vectors and the **weights** that make the margin as large as possible while limiting the number of violations, ↓ weights result in larger margins, a cost function that is often used for training SVMs is **Hinge Loss:** correctly classified point does not carry any information (**loss** = 0) instead the incorrect ones contribute linearly to the error
- **One Class SVM:** unsupervised, goal is to determine the minimum area tolerating a number of violations

- **Gaussian Processes (GP):** non-parametric technique used for regression and classification, GPs assume the data is generated by a giant multivariate normal distribution characterized by mean vector and a covariance matrix, the covariance matrix captures the relationships and covariances between all pairs of input point ⇒ matrix is defined by the kernel function, kernel function defines how similar two input points are and how this similarity translates to the covariance between their corresponding function values, a higher kernel value for two points indicates greater similarity and a stronger correlation between their function values, very expensive technique from a computational point of view, loses expressiveness in N-dimensional problems
- **Hyperparameter tuning:** all configuration parameters are tuned based on experince ⇒ right performance, there are techniques that allow us to test different hyperparameters automatically, example **Grid Search** creates a grid of values and performs an exhaustive search trying all combinations of set hyperparameters

7 Unsupervised Learning

- **Features:** ↓ features: greater data-parameter ratio, greater interpretability of the results, greater robustness of the model, ***regularization techniques*** select some more significant features automatically ⇒ there are techniques that allow us to ***reduce the dimensionality*** of our problem
- **Principal Component Analysis (PCA):** is a technique that is used to:
  - ***Data visualization*** reporting a problem with N dimensions in 2D/3D
  - ***Data compression*** projecting our points into a space with fewer dimensions losing some information (lossy compression)

  input we have a ***linear combination of the feature***, uncorrelated variables as output, goal is to find features that are highly correlated with each other and summarize them in a smaller space with a reduced number of variables ⇒ throw others variable lose some info/variability. ***Process to obtain PCA*:**
  - identify direction, the one where data variability is higher
  - form that minimizes the projection error is found
  - data is projected and the procedure is reexecuted
  - NB: the new direction will be orthogonal to the one previously identified

  PCA - Implementation:
  - we have the features in $\boldsymbol{X} \in \mathbb{R}^{N \times d}$
  - standardize the features (remove mean, divide standard deviation)
  - Define the normalized data matrix as $\widetilde{\boldsymbol{X}} \in \mathbb{R}^{N \times d}$
  - calculate the **Singular Value Decomposition SVD** of $\widetilde{\boldsymbol{X}} = \boldsymbol{U} \cdot \boldsymbol{\Sigma} \cdot \boldsymbol{V^T}$
  - Select the first $q \leq d$ columns of $\boldsymbol{V}$ obtaining the reduced matrix $\boldsymbol{V}_q \in \mathbb{R}^{d \times q}$
  - Compute the projected data $\boldsymbol{Z} = \widetilde{\boldsymbol{X}}\boldsymbol{V_q} \in \mathbb{R}^{N \times q}$ ***principal component scores***

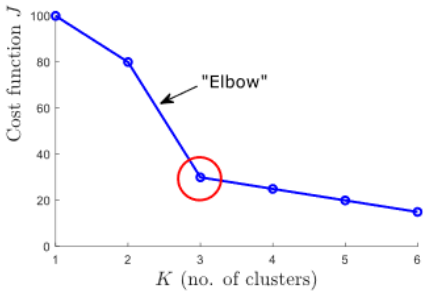
  PCA has some disadvantages: assumes that the variables are linearly correlated, only the axes with the greatest variance are considered ⇒ this leads to enormous errors
- **Kernel PCA:** we can use a Kernel to find a linearly separable feature space and then apply PCA to it
- **Independent Component Analysis (ICA):** unsupervised learning technique, does not serve to reduce the number of features but to ***make the components independent***, dataset needs to be transformed to separate different sources of information, more robust and potentially interpretable input features to the model ⇒ ***reducing the correlation terms***
- **t-distributed stochastic neighbor embedding (t-SNE):** technique used for

projecting problems with a very large number of features into a 2D or 3D space for visualization, it is based on **comparing the distance of points (examples)**, points that are close in N-dimensional space are also close in 2D space, 2 neighborhood probability distributions are constructed one in N-dimensional space and one in reduced space, the algorithm iteratively tries to move the points of the second one so that the distance (**KL Distance**) between the two distributions is minimized

- **Multi Dimensional Scaling (MDS):** technique used to do dimensionality reduction, similar to that of t-SNE, distances between elements are calculated in a ***dissimilarity matrix***, tries to keep the distances of the points unchanged by optimizing a function called ***Strain*** starting from the dissimilarity matrix ***D***
- **Isomap:** is a non-linear dimensionality reduction technique used to visualize high-dimensional data by embedding it into a lower-dimensional space while preserving the ***geodesic distances*** between data points. **Iteratively create the *geodesic distances*:**
  - determines the closest points to each candidate within a radius or its k-NNs
  - constructs a graph of neighbors with edges weighted as a function of distance
  - calculate the minimum distance between two points using Dijkstra or other

  The geodesic distances from the neighborhood graph are used to construct a low-dimensional embedding using multi dimensional scaling (MDS) techniques
- **Clustering:** we find **clusters** or groups of points similar to each other within the data without having a label, understand patterns within the data, check for similarities, define policies to clusters
  - ***K-means:*** simplest clustering technique, need to know **how many clusters** K, choose **randomly** K centroids, minimize the total sum of the squares of the distances of each of its points from its centroid. **Iteratively:**
    * We associate each point with the closest centroid
    * recalculate the position of the centroid as the average of the distances from of the points beloning to that centroid
    * when we no longer move any points we have finished the search

    K-means is a **greedy algorithm** that converges to a **local minimum**, we can search for clusters with random centroids multiple times to improve performance.
    Number of clusters: we do not know a priori how many clusters K, rely on data problem, rely on data distribution/visualization, **elbow rule**
    K-means errors: tends to give each cluster the same number of points, randomizing the initial state is very subject to different results



8 Unsupervised Learning

- ***DBSCAN*** clustering technique that is based on the concept of **density**, identifies clusters of **arbitrary shape** in a dataset that may contain **noise**. **Algorithm:**
  - Start with an arbitrary unvisited point
  - expand the cluster including the points that have distance less then ε fixed
  - so we are collecting all points inside the ε radius
  - we will get the first cluster with N points and the other outside
  - we evaluate the other points, if they are above a certain **threshold** we will consider it part of a new cluster and we will start to expand it, if the point is sparse with nothing or few points in its surroundings it is **classified as noise**

  Disadtanges: the fixed radius severely limits its capabilities when there are very close points, close clusters they will be belong to the same one
- ***OPTICS:*** clustering technique that wants to **overcome DBSCAN limits** $\Rightarrow$ the problem of detecting meaningful clusters in data of ***varying density***, create an empty ordering list to store the cluster order, the points are of the database are processed and ***ordered*** based on a ***reachability metric*** such that spatially closest points become **neighbors** in the ordering, goal is to order points in such a way that the clustering structure can be identified via a reachability plot/dendogram, ***valleys*** represent the dense cluster while ***peaks*** the separation between clusters, noise is identified as strange peak in the dendogram
- **Hierarchical clustering:** data with a different level of detail, is a technique that allows us to build a ***tree of relationships*** following one approach:
  - ***Bottom-up***, each point has its own cluster, iteratively merges the closest clusters until only one cluster remains or a stopping criterion is met
  - ***Top-down***, starts with all points in one cluster, iteratively splits the clusters until each point is its own cluster or a stopping criterion is met

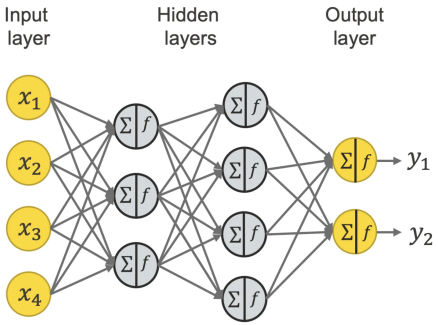
  there are several metrics with which to proceed to ***divide*** or ***aggregate*** clusters, the ***ward*** metric minimizes the sum of the squared distances within all clusters trying to minimize the variance within the cluster
- **Recommender systems:** algorithms that try to suggest a new item to a user, can be seen both as **unsupervised problems** (I find users more similar to me) and as **regression problems** (what could be the rating I will give to that film)
  - ***Content Based:*** recommend items that are similar liked in the past
  - ***Collaborative Filtering:*** recommend items based on similar users target, more popular one, similar users may have explored new items that I didn't know existed, items that I have already liked I may no longer need, the idea that information about users' tastes is widespread, similar users who share the same tastes will allow you to propose new items to others
    * **Item-based:** comparison between items, calculate a distance metric between all the items in order to find films that are similar to each other
    * **User-based:** user similarity, propose items to users by selecting them from those appreciated by a sampling of other users with similar items

We have seen:
- **Supervised:** regression, classification
- **Unsupervised:** dimensionality reduction, reccomendation

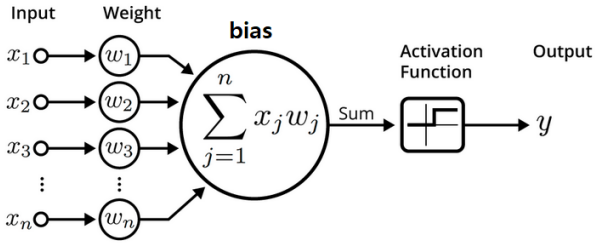
9 Neural Networks

Theory:

- neural networks are universal approximator
- connect enough neurons together and have enough data ⇒ solve any problem
- connecting the output of a group of neurons with the input of other neurons we obtain a **neural network**, each group of neurons on the same level is called a **layer**
- First layer **input layer** where we pass $\boldsymbol{x}$, last layer **output layer** we have $\hat{\boldsymbol{y}}$



- Every neuron accept inputs, every input has a **weight** w, all the input results will be summed $\sum$ linear combination of the input, after that an **activation function** f will be applied, and we will get an ouput value

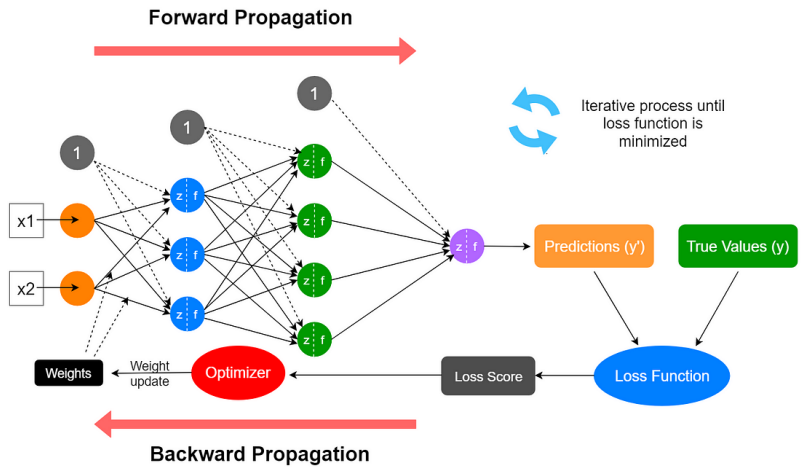


- The activation function transforms the input data weighted to generate an output
- **Some activation function:** linear, sigmoid, hyperbloc tangent, Relu, ..
- Neural networks has to to deal with no linearity pattern within the data ⇒ **non-linear activation functions**, but activation function ***must*** be differentiable and that its **derivative** somehow favors the learning of the network (never be zero), so that can be a problem with the non-linearity area from some activation function
- ***The linear activation function:*** nothing more than the linear combination of the inputs, using multiple consecutive layers of linear activations does not add any information to the model, it can be immediately approximated with a single layer

Multi Layer Perceptron (MLP): this is the type of neural network we have seen so far, all neurons from the previous layer are connected to the next layer, this formulation is very convenient because: easily define our layers, address very nonlinear problems by simply adding more layers, parallelize the execution/computation of each layer as each node is independent. **NB:** the number of parameters w within the model **grows very rapidly**

Learning: the learning process of a neural network is called **back-propagation** and is made up of these steps:

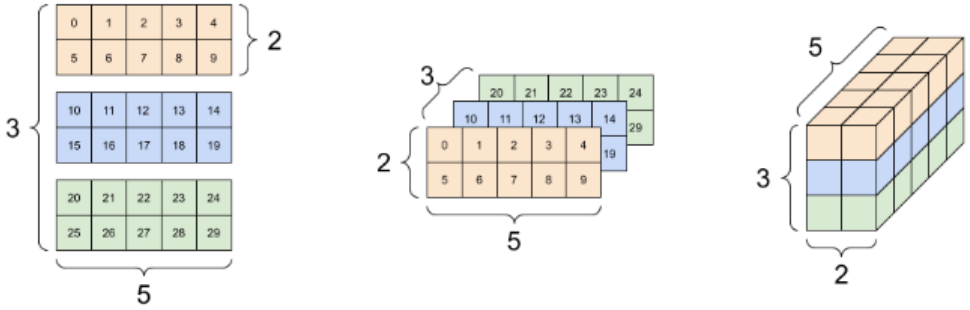
- A **batch** of data is selected from the training dataset, a batch is a small subset of the data, batches helps to **stabilize and speed up** the learning process
- The batch of data is fed into the neural network through the input layer. The data passes through the network layer by layer, with each neuron applying its weights and activation function to produce outputs ⇒ **Forward propagation**
- As the data moves through the network, **the intermediate results** (the outputs of **each layer**) are stored. These intermediate values are necessary for calculating the gradients during the back-propagation step
- Once the network has produced its predictions, the next step is to **evaluate how well these predictions** match the true labels of the data. This is done using a loss function (also called a cost function or error function). The loss function quantifies the difference between the predicted values and the actual values.
- **Back-propagation** starts by calculating how much each neuron in the output layer contributes to the overall error. This is done by computing the gradient of the loss function with respect to each neuron's output.
- The process of calculating gradients continues backward through the network, layer by layer. For each layer, the algorithm calculates how much each neuron in that layer contributes to the error in the subsequent layer. This involves using the **chain rule of calculus** to propagate the gradients backward from the output layer to the input layer.
- Once the gradients for all the neurons have been calculated, the network's parameters (weights and biases) are updated
- Steps are repeated iteratively for many **epochs**. An epoch is one complete pass through the entire training dataset. By continuously updating the parameters in this way, the network gradually improves its performance, reducing the loss and improving its predictions.



NB: weights must be initialized **randomly** otherwise it would not learn anything, if two **weights** were exactly **identical** within the same layer their contribution would be identical and halved, it trains on **batches of data randomly sampled** according

to **stochastic gradient descent**, **scaling or normalizing** the input features helps enormously, we easily suffer from overfitting due to the high number of weight so we can apply ⇒ **regularization techniques**, number of epochs should not be too high ⇒ **overfitting**, dataset better be balanced appropriately, learning rate has a great impact and often needs to be varied during learning ⇒ **better low one**.

Tensorflow: more flexible and low-level frameworks than Scikit-Learn are used: **Tensorflow** (by Google) and **PyTorch** (ex. Facebook, now Linux Foundation) the basic concepts are identical the implementation changes slightly, with these frameworks we build the Machine Learning algorithm ourselves instead of finding it already implemented. The **tensor** is an in-memory representation of an N-dimensional immutable object:



Our algorithm will consist of three parts:

- **Network structure:** number of layers, how many neurons will each layer have, which activation function do we choose for each layer, any function layers, any regularizations, what each layer is connected to
- **The loss function:** define the cost function that we will have to optimize, the larger it is the further we will be from optimal, we can use standard cost functions:
  - MSE, MAE, MAPE, etc. for regression problems
  - BinaryCrossentropy, CategoricalCrossentropy, etc. for classification problems
  - others or define a custom one

  from the model we can retrive some metrics automatically calculate but we will not optimize those
- **The optimizer:** several optimizers have been developed that update the weights slightly differentl, all these optimizers in turn have **parameters** to look for such as the **learning rate**, tuning of the hyperparameters for the best choice
  - **Stochastic Gradient Descent**, **RMSProp**, **Adagrad**, **Adam**

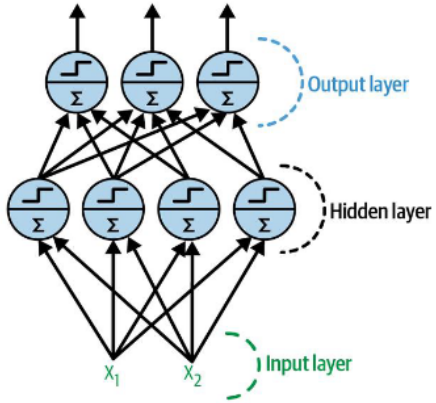
10 Neural Networks

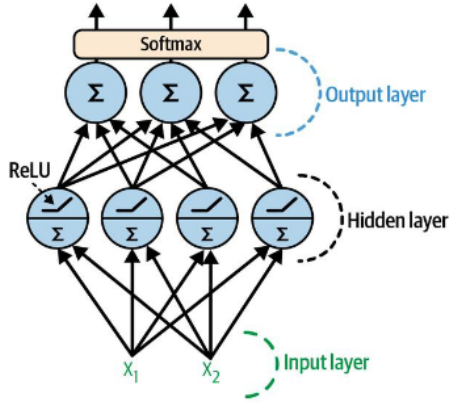
Structures:

- ***Sequential model:*** all the layers are connected one after the other (as MLP)
- ***Wide and deep:*** the model passes the input both through a deep neural network and directly to the output, output layer will be able to exploit information that arrives directly from the input and deeper information that arrives through the structure of the network
- ***Multiple inputs:*** we build the model based on two different source of inputs
  Multiple outputs: in this case the model has different outputs, we will need to specify a **loss function for each output** and consequently a target against which to optimize

Output layers:

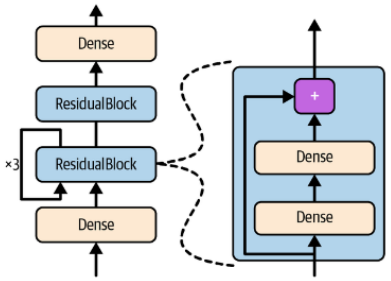
- In the case of **regression/forecasting** problems of a single variable we can use a **linear activation layer** output to directly predict the value or use other activation functions possibly rescaling the data.
- In the case of **single class classifications** we can use a single neuron with a **sigmoidal activation** function to represent its probability between 0 and 1.
- In the case of **multiclass classification** problems we must distinguish
  - ***Single label***, each exaple belong to a class, when we have N classes we can use the **softmax** which allows us to choose the class with the highest value as the output class **rescaling** the probabilities according to the relative values
  - ***Multi label***, example can belong to multiple classe, no **softmax** application ⇒ each neuron will instead have its own independent activation function





Vanishing gradient problem: disappearing gradient, when we use activation functions like ***sigmoid*** or ***hyperbolic tangent*** the derivative tends to be very small as we move away from the center, during the ***learning phase (back propagation)*** the chain rule will multiply the partial derivatives of each parameter together resulting in the product between them of very ***small values*** ⇒ in deep networks the gradient disappears so the first layers are ***unable to learn*** as they backpropagate through the network, we can ***skip connections*** as in the wide and deep network to bring the gradient directly to the deeper layers of the network

- ***Residual block:*** is a module of the network that we can exploit in cases of vanishing gradien, **based on the concept of skip-connection**, that allow information to flow directly from an earlier layer to a later layer bypassing some of the intervening layers, keeps the deeper features in the middle and skips the interested one, the skip connection ensures that the gradient can flow through the network without diminishing allowing the network to learn even when it has many layers ⇒ ***Residual neural network (ResNet)***, it is used to concatenate multiple residuals blocks in a row



Custom network: we start to define custom structures and custom loss functions we may also need to ***customize the training of our network***: we calculate the gradients pass them to the optimizer and update the weights

Pre-trainig: once the structure of our model has been defined we can generally carry out an initial train and then a ***fine tuning***, the first train has the objective of verifying the model's ***ability to converge*** and evaluate the hyperparameters inserted, pre-training is also performed on a larger and potentially dirty dataset with a higher learning rate, we can also use optimization methods such as ***GridSearch*** to tune some hyperparameters, hyperparameter tuning in the case of neural networks is ***extremely more onerous*** because we no longer have weak-learners but actual ***complex objects***.

Supervised pre-train: transfer pre-trained network or piece of network to solve tasks already faced, re-use knowledge already learned ⇒ ***transfer learning***

Unsupervised pre-train we use unsurvised learning techniques to train our layers to extract significant features on the same dataset as ours but with a different task:
- Compress information and decompress it (Autoencoders)
- Generate credible data (Generative Adversarial Networks)
- Reconstruct parts of data (self-supervised)
- Removing noise from data (de-noising)

Fine tuning: instead consists of perfecting the model so that it works best on our task, ***Fixed weight:*** fix the weights of some layers

Overfitting: we often have a very large number of parameters, pre-training and tuning the network by ***blocking some layers*** or with a much ***lower learning rate*** are just two of the different techniques we can apply to reduce overfitting, we can add ***regularization*** (which has to be integrated in the loss function) to the layers to keep the weights with low absolute values, we can also ***add a layer*** specifically to reduce the risk of overfitting

Dropouts: the dropout layer is a layer that ***deletes some random neurons*** during the training phase setting their output to zero, in this way the model is forced to ***distribute knowledge*** among multiple neurons not having the possibility of always exploiting the same features to make the decision, per each layer we will specify a percentage ***drop-rate of number*** of neurons at zero ⇒ forcing the neural network to adapt to the critical situation and see the performances

Data augmentation: data augmentation is the process by which we create new ***synthetic examples*** starting from real ones ***distorting them appropriately*** so that the information contained is still detectable, ***noises are often added*** which have a lower variance than the difference between the examples, in case of images we can apply other transformations (perspectives, rotations, noise, blur, exposure, ..), this way we can multiply our input data making the model ***more robust*** to new data ⇒ good for training and also testing

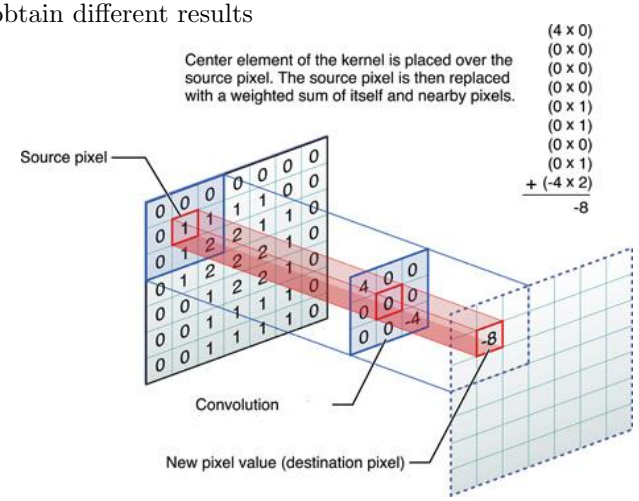
Batch normalization: usefull layer to add, ***normalizes*** the outputs with respect to the single training batch rescaling the values so that they are distributed like a normal, helps the learning time and the ability of the model to ***converge***

there are other layers that have no impact on training or model performance:
- **Flatten:** takes an N-dimensional array and reduces it to a vector
- **Concatenate:** concatenate arrays along an axis
- **Max, Min, Average:** layer that extracts a single value from all the features
- **Reshape:** change the structure of an N-dimensional array

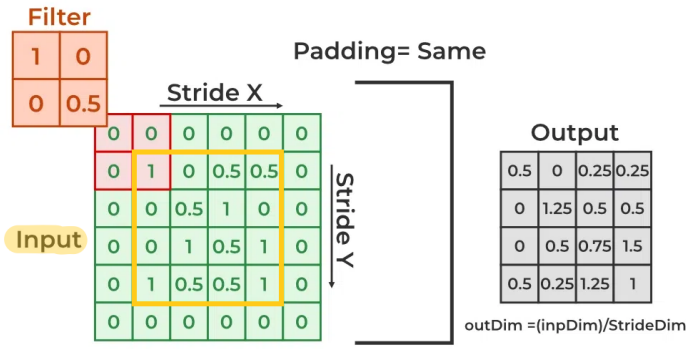
Callbacks: function executed at the end of each epoch, can help us
- ***EarlyStopping:*** stops training when it observes that the loss between training and validation deviates too much ⇒ may have started to overfit the model
- ***ModelCheckpointer:*** a callback that saves the model example every time we make a better score in validation
- ***Tensorboard*** obseve the training data and model characteristics

11 Convolutional Neural Networks (CNN)

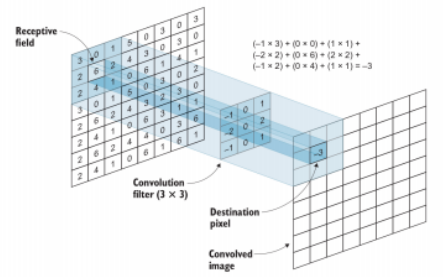
Covolution: is the mathematical operation of ***sliding two vectors*** relative to each other, in the case of images the idea is to slide a ***smaller matrix called a kernel*** across our entire image, the convolution can be: 1D, 2D or 3D, convolution kernels are often also called even **filters** because the applications in traditional computer vision convolutions, depending on the values and dimensions that are assigned to the kernel we can obtain different results



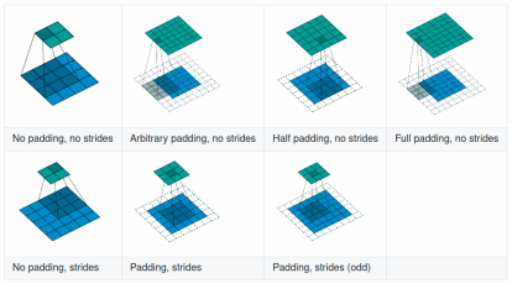
- ***Padding:*** since the convolution start inside our image the output matrix will have each dimension that is $2 \times (Kernel\ side//2)$ less, this determine strange layer size $\Rightarrow$ for this reason we add ***padding*** around the ***input matrix***, edges of zeros which allow the convolution to obtain more manageable dimensions, we will get an output that has the same dimensions as our input matrix



- ***Stride:*** is the number of cells we move with each convolution, basically meaning ***step size*** here, with $stride = 1$ consider all the pixels, with $stride = 2$ I skip 1 pixel, Blue maps are inputs, and cyan maps are outputs, the output pixel is determinated by the centerd pixel of the input matrix

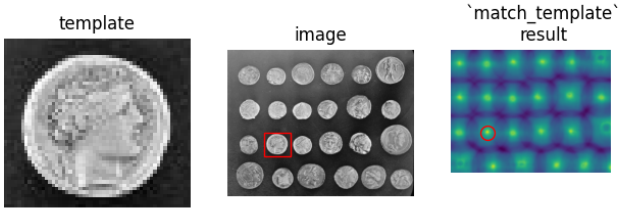


(Credit: [Elgendy, 2020])



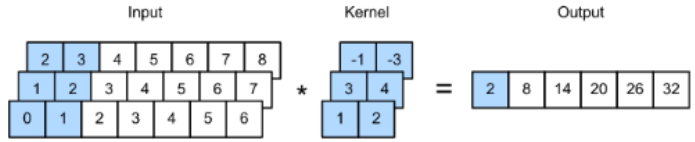
https://github.com/vdumoulin/conv_arithmetic

Template matching: is a traditional computer vision problem in which you need to ***find a template*** within a larger image, the traditional approach involves ***convolving our template over the entire image***, in each position a ***correlation index*** will be calculated and at the end the maximum obtained will be evaluated, traditional template matching ***suffers terribly from variations*** of: scales, shapes, calligraphy, prospect, rotation $\Rightarrow$ for this reason deep learning techniques

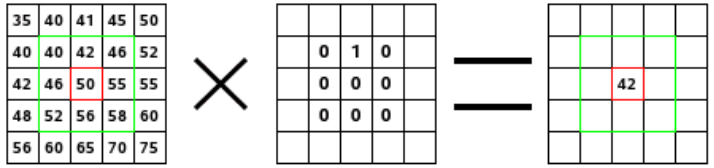


Layers: a layer is a group of neurons on the same level, types:

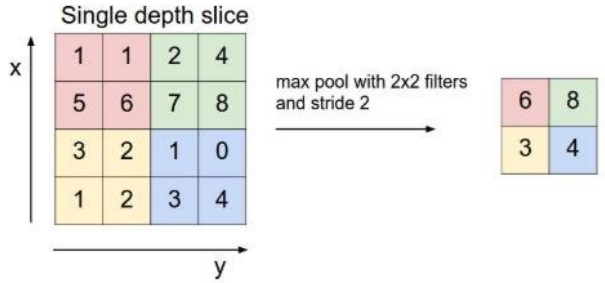
- ***Conv1D:*** 1D convolution layer is commonly used for ***time series***, intercept typical local features of signals over time



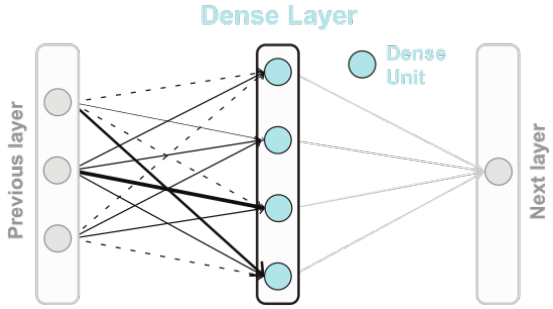
- ***Conv2D:*** 2D convolution layer that we need to define: size of our kernels, number of kernels we want to apply, activation function, set of other hyperparameters. ***Our kernel will be deep as the input layer***, they will all be representations with the same level of detail as our input matrix, in order to reduce the dimensionality we can use ***pooling layer*** to better interpret the features



- ***Polling:*** we need a way to ***reduce the dimensionality*** of our layer to be able to interpret our features at a higher level, we can get the: maximum, minimum, average, ect. Every x cells to construct a matrix that has size divided by x

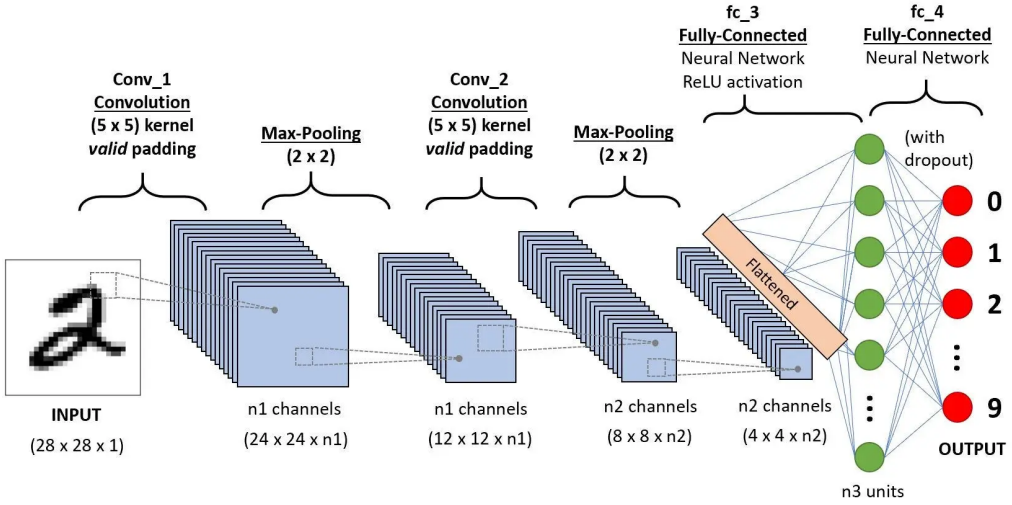


- ***Dense Layers:*** also known as ***fully connected layers***, layers where each neuron is connected to every neuron in the previous layer, these layers perform the final ***classification*** based on the features extracted by the convolutional and pooling layers



CNN: structure that uses ***convolution layers*** and possible ***pooling layers*** to perform various tasks, normally we tend to ***reduce the dimensionality*** of x and y through pooling and ***increase*** it on z via the number of kernels, at the end of the network we will then ***flatten*** the result and apply ***dense layers*** to classify the result of the convolutional features

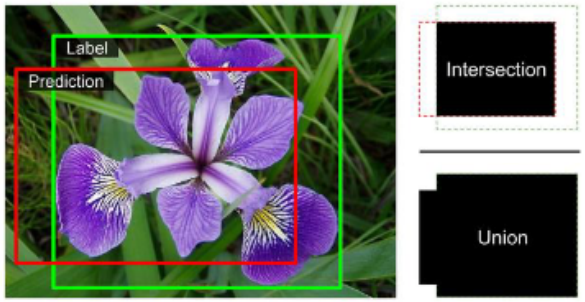
- ***z (depth dimension):*** corresponds to the number of filters (kernels) used in the convolutional layers, as the data passes through the convolutional layers the number of filters usually increases enhancing the depth dimension z, after applying 32 filters in the first convolutional layer the depth z becomes 32



Object classification: given an image we need to tell which class it belongs to

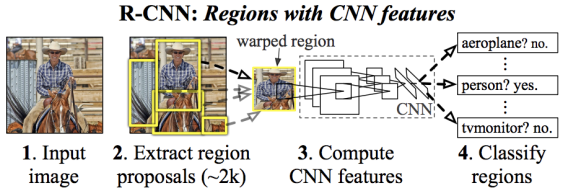
- ***LeNet5:*** simple CNN that performed well on the character recognition task but failed to scale to more complex tasks
- ***AlexNet:*** won the ImageNet competition in 2012, a lot of parameters
- ***Google LeNet:*** won the ImageNet competition in 2014, the network 10 times lighter than AlexNet and also the performance is given by the fact that the network is much deeper
- ***ResNet:*** won in 2015, diluting the information across multiple layers better perfomances acquired, implement the skip-connections

Object localization: classify which object is in an image but also to have an indication of where it is, the output network will provide probability that the object is there and additionally the coordinates x, y, w, h (width, height), labels that will also include the object's ***bounding box***, in this case we cant use MSE or cross-entropy as losses but we need something that indicates a degree of ***accuracy of our bounding box*** $\Rightarrow$ ***Intersection over Union (IoU)*** indicates the intersection area divided by the union area, we have perfection when the two BBs are ***perfectly overlapped***, we have 2 BBs because you're comparing the predicted ***bounding box from your neural network*** with the ***ground truth bounding box*** which is labeled manually or obtained from annotated datasets

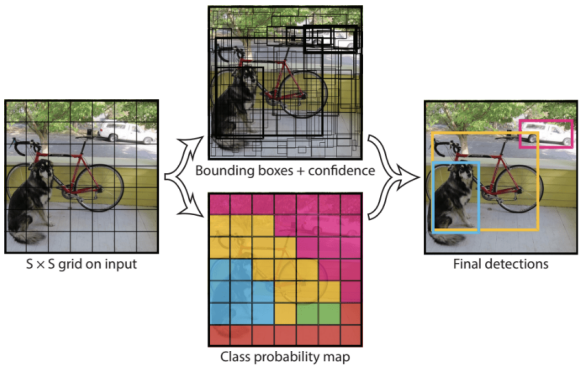


Object detection: when there are multiple objects in our image we want the ***bounding box and its confidence to be generated in output for each class***, dealing with just one object but possibly N in the image, solve by different network architectures:

- ***R-CNN:*** A first very simple algorithm proposes possible cropping of the image $\Rightarrow$ ***region proposal***, a CNN extracts deep features, a classifier that classifies the vector with deep features to say whether the object is present in the region or not.



- ***Faster R-CNN:*** evolution of the R-CNN, involves the use of ***a single model*** to carry out the three steps: of region-proposal, features extraction and classification
- ***Yolo:*** yolo networks are usually ***faster but less accurate than R-CNNs*** or other types of models, dividing the image into ***cells*** $\Rightarrow$ ***Grid-Based Detection***, each cell is responsible for classifying an object if the center of its bounding box falls inside it, YOLO uses predefined ***anchor boxes*** which are bounding boxes and each cell predicts offsets relative to these anchor boxes $\Rightarrow$ ***Anchor-Based Model***



12 Convolutional Neural Networks

Non maximal suppression (NMS): ***anchor based model*** such as Yolo, ***make N predictions*** of the same object because each anchor believes it owns the center of the object, we find ourselves having to choose between many predictions as to which one best represents the object, ***to filter bounding boxes:***

- We take the bb with the ***highest confidence*** and keep it as good
- We take all the other predictions of the same class, if the ***IoU exceeds a certain threshold*** we remove them from the set of proposals

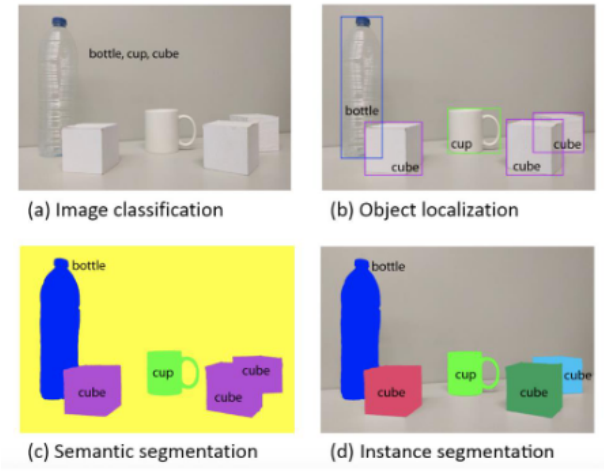
Object segmentation: we wanted to train an algorithm not only to frame our object in a bouding box but also to indicate the ***individual pixels***, ***semantic segmentation*** task instead deals with identifying only the ***class of each pixel*** without identifying whether they are instances of different objects

- ***U-Net:*** one of the main techniques for segmentation, it tries to create a U in which we have convolutions that gradually extract deeper and deeper features ⇒ different objects in the image, the deep features interpret the groups of pixels then the layers try to remap the information onto the larger image

  to **train U-Nets** we need as much expressive labels as we can: polygons that enclose the object of interest, real images/maps that we want to get out of the network, we also need to know how to move ***from a low resolution layer to a larger one***, scale up once we reach the lowest level of our CNN ⇒ ***Conv2DTranspose***, the layer will learn a kernel (as in convolution) which it will then use to upsample the previous layer
- ***Yolo and Mask R-CNN:*** readapted to support objects segmentation
- ***Segment Anything Model (SAM):*** It is a huge model compared to models like Yolo or U-Net and in fact it was largely trained in self-supervising learning on an infinite dataset (11 million images and more than 1 billion masks)

For implementing a segmentation model it is important to distinguish between Semantic and Instance Segmentation

- **Semantic segmentation:** we want just to know at ***what class the pixel belong***, the output is a matrix with the same dimension as the input image
- **Instance segmentation:** we want to ***distinguish each object*** of interest but not the background for example, the ***output*** of the model is the ***coordinates*** of the polygon detecting the object

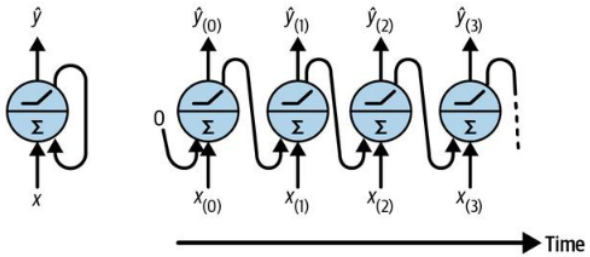


Anomalies: we don't know exactly what we're looking for but we just want to identify anomalies, better with ***Autoencoders*** but is also possible wiith ***convolutional features***

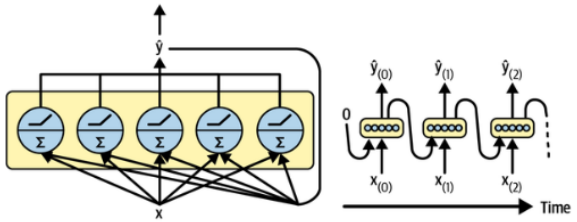
- ***Patchcore:*** one of the best algorithms for Anomaly Detection, during the training phase characterize the ***good samples*** selecting a set of features points, during inference will compare the new example with the memorized features of training samples

13 Recurrent Neural Networks (RNN)

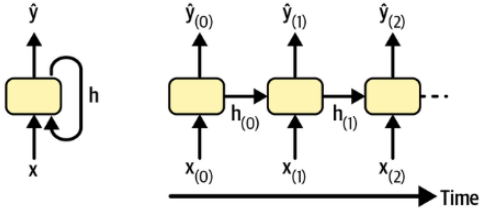
RNNs: Recurrent neural networks or RNNs are structures that allow us to process arbitrarily ***long sequences***, they are based on the ***recurrent neuron*** a structure capable of receiving an input and passing a value as a second input to itself



Each neuron will have two sets of ***weights*** relative to the: ***external input*** x_t and processed input which is the ***hidden state*** $y_{t-1}/h_{t-1} \Rightarrow$ neurons accumulate a ***memory*** of what has been observed



The passed value does not necessarily equal the output, we can also choose to pass a ***hidden state*** h as we carry out (unroll) the calculation



The RNN is very flexible and we can create very different structures:

- ***Sequence to Sequence:*** both input and output are calculated together
- ***Sequence to Vector:*** represent an arbitrary sequence in dense form
- ***Vector to Sequence:*** to unfold the contents of a vector
- ***Encoder Decoder:*** encodes a internally state which then passes to the decoder

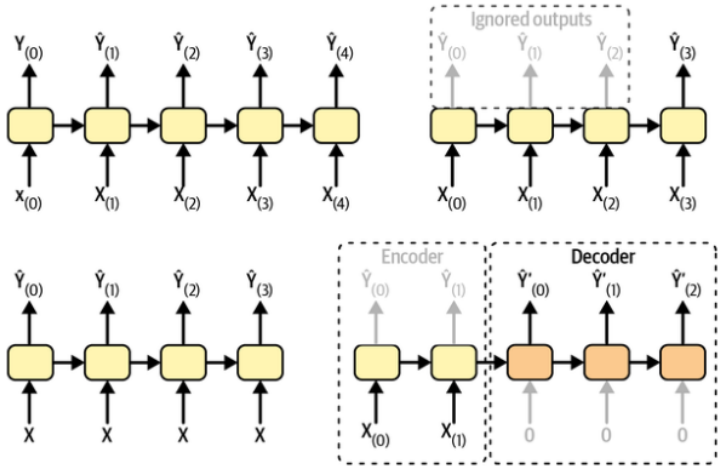


Figure 15-4. Sequence-to-sequence (top left), sequence-to-vector (top right), vector-to-sequence (bottom left), and encoder–decoder (bottom right) networks

NB: When modeling time series let's not forget the importance of traditional models: AR, MA, ARMA, ARIMA, ect.

An RNN is nothing more than a neural network that makes use of recurrent neurons, we can build more or less deep structures also integrating dense parts, we are learning sequences which can be: sentences recognition, time data, etc. A recurring Layer ***SimpleRNN*** has a lot of hyperparameters to set

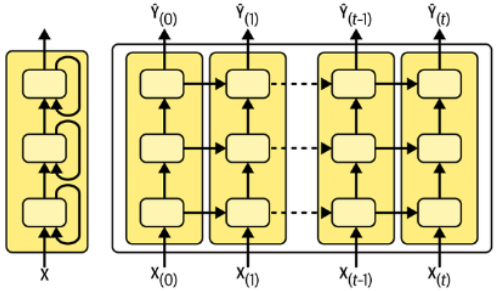


Figure 15-10. A deep RNN (left) unrolled through time (right)

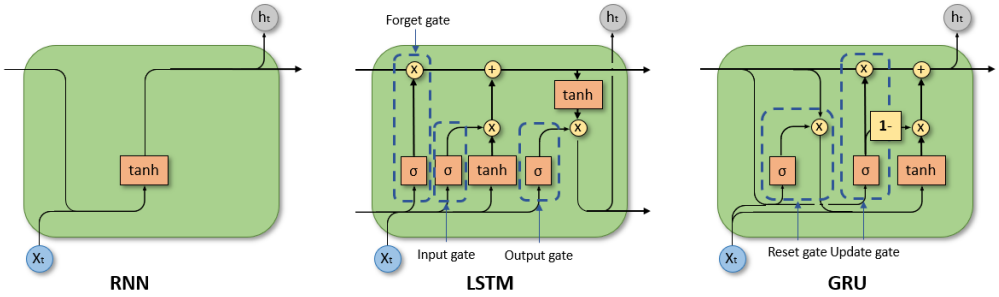
Forecasting: during inference, the RNN processes each time step sequentially, updating the hidden state with information from the current input and the previous hidden state. This step-by-step processing is necessary to maintain the temporal context and ensure that the prediction at each time step ***considers all relevant prior information*** $\Rightarrow$ I will use all the information till that point and the previous prediction ***to make the new*** one

RNN problems: so based on these problems new types of recurrent ***neurons***

- ***Slower Learning and Inference:*** RNNs process data in a sequence, meaning each step depends on the previous one. This sequential dependency makes parallelization difficult. To make a *prediction* at any point, RNNs need to consider the entire sequence up to that point $\Rightarrow$ all the information we have till now
- ***Vanishing and Exploding Gradients:*** when training RNNs, gradients can either *vanish* (become very small) or *explode* (become very large) as they are propagated back through many time steps. This problem makes it ***difficult for the network to learn long-term dependencies*** $\Rightarrow$ this bacause of the *hidden State Multiplication* that can shrink or grow uncontrollably
- ***Batch Normalization Issues:*** a technique to stabilize and accelerate training, is less effective in RNNs. This is because the hidden states are updated sequentially, and normalizing them batch-wise can disrupt the temporal dependencies

Improvement networks than RNNs:

- **Long short term memory (LSTM):** networks are more effective than traditional RNNs at retaining memory over long sequences, this capability is due to the sophisticated structure of ***LSTM neurons***, which include mechanisms for deciding what information to keep, update, and discard $\Rightarrow$ ***The three gates*** in LSTM neurons (forget gate, input gate, and output gate) work together to manage the cell state, enabling the network to learn long-term dependencies and carry forward memory efficiently
- **Gated recurrent Unit (GRU):** is a ***simplified variant of the LSTM***, designed to improve computational efficiency while maintaining the ability to capture long-term dependencies in sequential data. With its ***simpler architecture and fewer gates***, the GRU often performs as well as the LSTM.



Other models:

- ***1D Convolution - WaveNet:*** an alternative to RNNs for processing time series of unknown length, the most famous network is WaveNet, a series of 1D convolutions are concatenated, WaveNet intercepts local short-term patterns in the lower layers while in the higher layers it interprets long-term dependencies despite having no memory
- ***Bi-directional RNN:*** if we do not have to analyze the data as they arrive but we have the entire sequence available, the sequence is scanned in both directions and the model ***learns*** not only of the ***previous history*** but also of the ***future one***

14 Natural Language Processing (NLP)

Concerns text processing applications, from the text we may want to extract various information such as: identify the key parts of a sentence, search within a document, compare different sentences, generate questions and answers

Its difficult to deal with text because:

- We have many letters in the alphabet
- Similar letter combinations do not necessarily have the same meaning
- The same word takes on different meanings depending on the context
- The same concept can be expressed in a multitude of different written forms
- There are different languages and sayings especially on the internet

Operations: operations made on text and used in models

- ***Pre-processing:*** clean our input from a whole series of characters that can bother us: lowercase the words, remove articles, pluras and commas (spaces)
- ***Stemming:*** a way to get closer to a denser representation of our vocabulary keeping the roots of the words, we can use fixed rules or eploit them, in some language stemming is meaningless or complicated, it is not precise

Representations: we need to define a ***numerical representation for the text***

- ***One-hot encoding:*** for each word we build a vector of as many zeros as our vocabulary is large and with 1 in the position associated with the word, ***some problems:*** very large dictionary of words, find a way to represent words that are similar or have the same meaning, sparse representation
- ***Term Frequency (TF):*** after stemming we can represent our sentence vectorically by adding the one hot encodings of all the words, terms that appear more frequently in a document are considered more important ⇒ favors that simply repeat a term many times without adding any meaning

$$TF(t,d) = \frac{Number\ of\ times\ term\ t\ appears\ in\ document\ d}{Total\ number\ of\ terms\ in\ document\ d}$$

- ***Inverse Document Frequency (IDF):*** measures how unique or rare a term is across all documents in a corpus, helps to downscale the importance of terms that appear very frequently in many documents and are therefore less informative ⇒ **more informative rare words**

$$IDF(t) = 1 + log\left(\frac{Total\ number\ of\ docs}{Number\ of\ docs\ containing\ t}\right)$$

- ***TF-IDF:*** balances the importance of a term within a document against how common the term is across all documents in a corpus ⇒ limited representativeness, treats each word as independent while words in a sentence can be different

$$TFIDF(t,d) = TF(t,d) \cdot IDF(t)$$

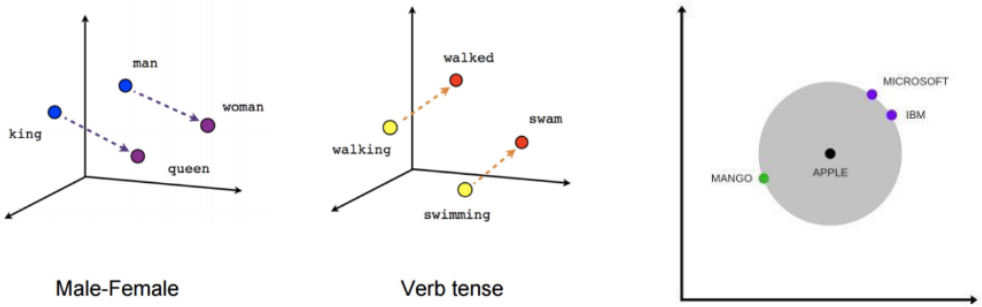
Similarities: input is a term/phrase, the term is pre-processed and stemmed, the term becames a vector, each word on the vector has a TF-IDF, get best term closest to the input phrase which means same TFIDF ⇒ compute a distance between vectors, typical approach is to use the *cosine similarity*

- ***N-Gram:*** is a contiguous sequence of N items (typically words) from a given sample of text, the N refers to the number of items in the sequence (1-gram, 2-gram), triplet is much more significant than considering the three words, ↑ N increases exponentially the cardinality of the dataset

«The quick brown fox jumps »

BOW: { 'quick' – 'brown' – 'fox' – 'jumps' }
2-gram: { 'quick' – 'brown' – 'fox' – 'jumps' – 'quick_brown' – 'brown_fox' – 'fox_jumps' }
3-gram: { 'quick' – 'brown' – 'fox' – 'jumps' – 'quick_brown' – 'brown_fox' – 'fox_jumps' – 'quick_brown_fox' – 'brown_fox_jumps' }

- ***Word2Vec:*** represent words with denser representations, vector that characterizes the word is a vector of limited cardinality ⇒ this vector is called ***embedding***, this vector contains the meaning of the word placed in an ***N-dimensional space***, we can relate words comparing their vectors, this dense representation allows us to find the most similar words in meaning, find opposite, apply operations on words such as subtractiong and summing meanings to a word (woman + king = queen)

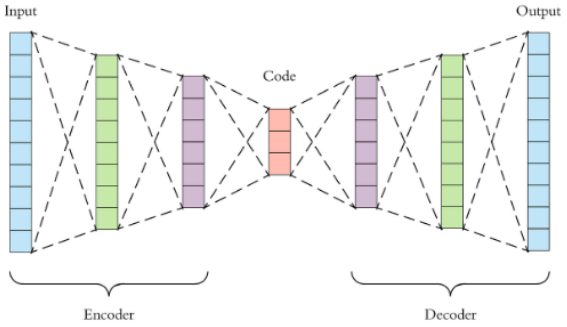


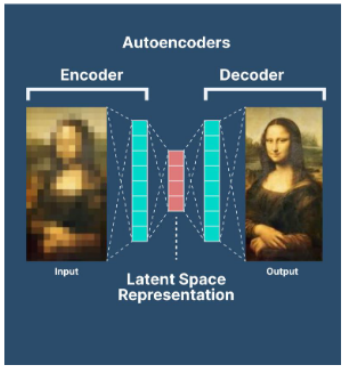
- ***Doc2Vec:*** expanding the representation no longer just to the single word but by encoding the entire document, encode the meaning of entire parts of text always in dense vectors

- ***Char-RNN:*** encode all knowledge within a model, we do not have a dense vector representation but it is the model itself that contains the information on how to interpret and type the characters ⇒ since characters are a sequence use an RNN that is able to predict the next single character, need to encode our dataset so that given a sequence it predicts the next character as it unfolds

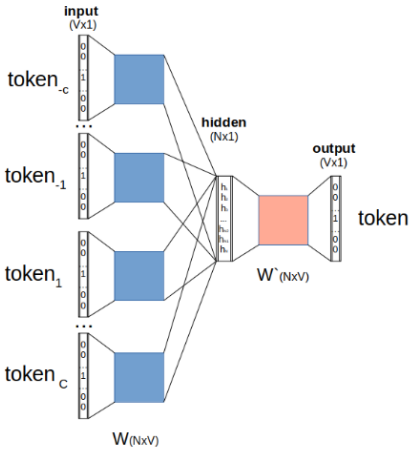
How to training this Dense representation: Word2Vect, Doc2Vect

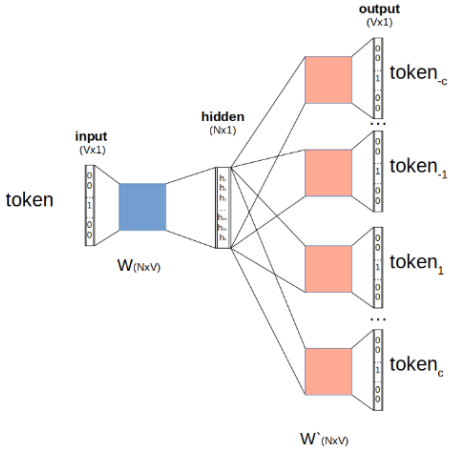
- ***Autoencoder:*** is a neural network structure that takes N inputs compresses them into a much smaller dimensionally intermediate representation which then decompresses to reconstruct the original vector.





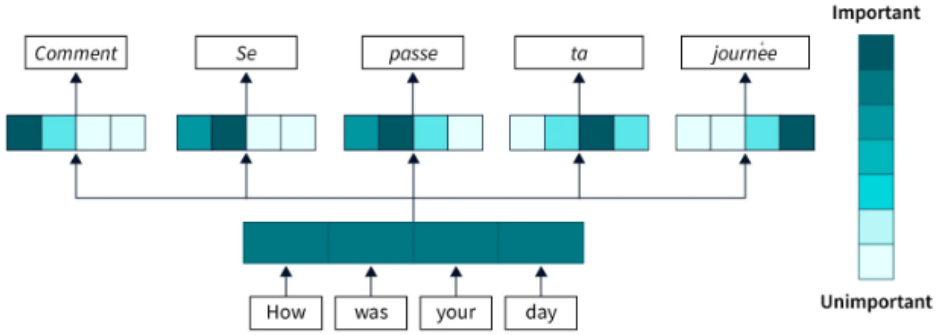
- ***CBOW and Skip-Gram:*** two similar approaches to training embedding vectors, *CBOW (left)* aims to reconstruct a missing token by passing the context as input, the input matrices are held constant, Skip-Gram (right) instead tries to reconstruct the most probable words given a term, output matrices may vary

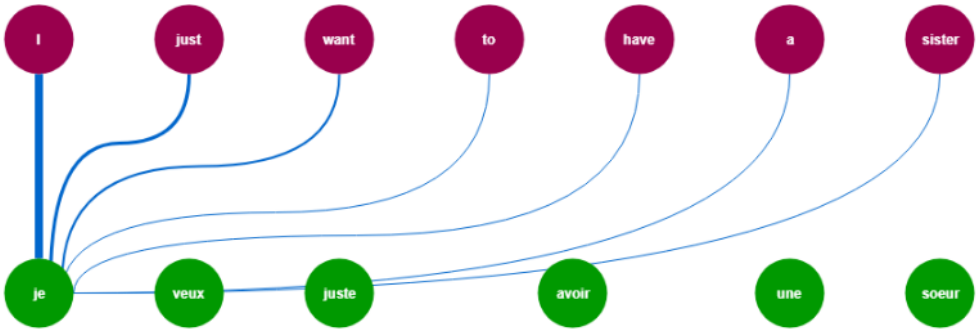




Mechanisms:

- In the case of text generation, there is a risk of ending up in some loops ⇒ generate the same word repeatedly, ***temperature*** is used: we divide the probability of each output with a temperature value and between the most probable are randomly one is chosen, ***low temperature*** (tending towards 1) will always and only prefer the most probable output, ***high temperature***s (for example 2-3) tend to randomly choose between different outputs and therefore generate more randomly
- ***Attention:*** mechanism that allows the neural network can selectively focus only on different parts of the sequence, we are able to intercept more complex meanings because we link words that are perhaps distant from each other, neural translation with attention (encore/decoder and RNN) dynamically calculate where our decoder needs to pay attention at each instant, neural traslation without RNN to give life to an architecture entirely based on attention called ***Transformer:*** there is no scrolling of the text but only an encoding of the input position (positional encoding) which is given to the head that calculates attention

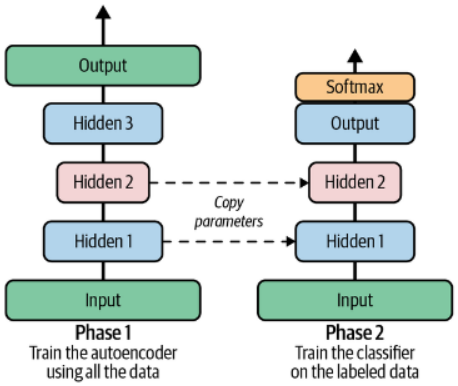




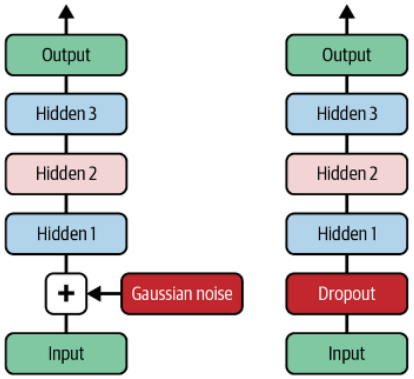
15 Autoencoders, GANs & Diffusion Models

Autoencoders: excellent structure for compressing very sparse information into smaller (denser vector spaces), compressing and decompressing information ⇒ ***learn a latent representation in an unsupervised way***, usefull method also for: identify anomalous data, rebuild missing patterns, generate new data. Autoencoder is defined by an ***encoder*** and a ***decoder***, encoder compresses preserving the maximum amount of information, decoder decompresses it trying to reconstruct it in its original form, the cost function is simply the difference between input and output. Main advantages of autoencoders is to ***extract highly significant features*** in a totally unsupervised way

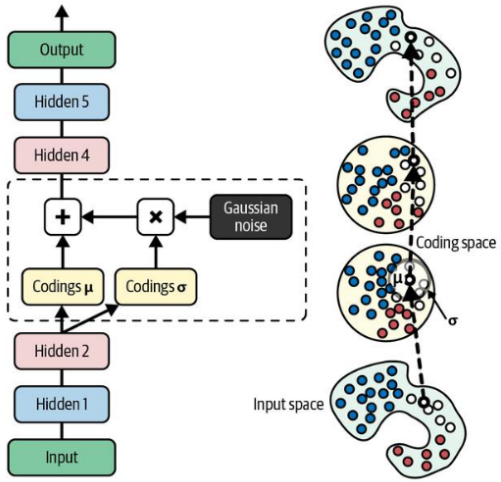
- ***Pre-train:*** we can build models even with a few labeled data and much unlabeled data available, in fact we can train our encoder and decoder then remove the second one and attach a classification head with much fewer parameters



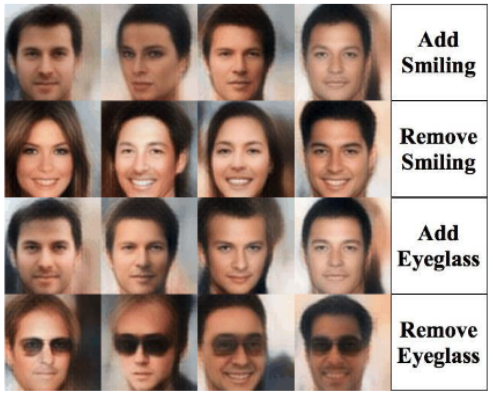
- ***Denoising autoencoder:*** another apporach to make the autoencoder learn useful features is to ***add noise*** or ***add dropout*** to the input, in this way our input is ruined and must be reconstructed from the other features by combining them, we can also use this technique to clean potentially dirty data within the dataset or as a real task



- ***Variational autoencoder (VAE):*** new category of autoencoders, they are probabilistic objects both in training and inference and are generative in the sense that they can produce new outputs similar to those on which they were trained, the encoder estimates a mean and variance for each position of the encoded vector these are used to sample a new example (the result of the sum of Gaussian noise) which will then be decoded by the decoder



Once the VAE is trained our latent space turns out to be entirely sampleable so we can go and code an example to take a starting point and then start modifying the vector to see how it changes, generate new samples fixing certain settings in training, autoencoder still uses compression and therefore a significant loss of detail



Generative Adversarial Networks (GAN): generative networks for creating new plausible data, its made of 2 networks:

- ***generator:*** whose task is to generate new plausible examples, starts its generation from a random noise vector, the same vector is equivalent once trained to the VAE embedding that we can use to generate new data
- ***discriminator:*** which must discriminate if the given example is true or generated

the generator tries to cheat the discriminator while the second always tries to discover whether the data provided to it is true or not, GANs are ***more difficult to train*** than other models: because we will have two networks to train together and balance, also the generator could discover flaws in the discriminator and start generating only objects of a class in which the discriminator has more difficult, two networks pushing together training will collapse and weights will be equal to 0. GANs don't have a linear training cycle like other models so we have to write the optimization cycle ourselves, start training the discriminator by creating a new example of the noise generator and adding it to the training batch

- ***Deep Convolutional GANs (DCGAN):*** built from deep convolutional networks, mainly used for image generation, they are fully convolutional do not use pooling but stride and use a lot of bath normalization, generating increasingly complex images and share with Word2Vec the projection of data into a space in which it is possible to perform mathematical operations between images
- (Nvidia) by ***adding convolutional layers*** to both discriminator and generator it makes easier to train GANs with a greater level of detail, Nvidia presented the ***StyleGANs***, the generator is divide between ***style*** coded as noise vectors to be added to the image and ***structure*** which is continuously disturbed by noise, in this way the style is encoded at different levels and added to the image while the structure remains unchanged
- ***GANomaly:*** VAE and GAN learn the structure of the dataset and to generate new plausible examples ⇒ ***reverse the process*** understanding if an example is plausible or anomalous, we try to combine the features of VAE and GAN to identify anomalous parts of an image, we can exploit this approach when the dataset has only good observations and a few or almost no anomalous ones

Anomaly Detection: There are two big categories:
- ***Outliers detection:*** finding the dirty data within our dataset i.e. the outliers
- ***Novelty detection:*** understand if it is an outlier or not on new data

Models: ***OneClass SVM***, use the encoder of our **VAE** to estimate whether or not our example is similar to the others available, in image processing most promising techniques is called ***Patchcore:*** is based on extracting a set of features from a backbone placed in a *Memory Bank* which are selected and compared using Nearest Neighbor with the new example to evaluate

Diffusion Models: are generative models that iteratively try to reconstruct an output by removing the noise present, better quality of the results than GANs, training is done by proving each time noisier images while the model continuously tries to subtract this noise to preserve the information within it

Image Generation models:
- GAN adversarial training
- VAE maximize variational lower bound
- Diffusion models gradually add Gaussian noise and then reverse

(?) Other problems

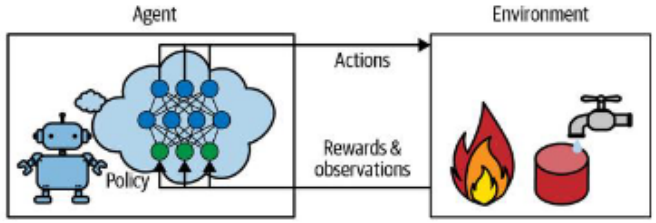
- **Bounding boxes Tracking:** tracing the bounding boxes means making a detection every frame, finding the bbs of the same class and re-assigning them to the closest object, one algorithm is ***ByteTrack*** keeps all the detections made by a detection network and weight them to perform Multi-Object Tracking (MOT).
- **Optical Tracking:** trackers that are based on ***optical features***, do not look at bounding boxes but only at a measure of similarity between subsequent frames of the same area, the objects to be tracked must be selected and then they can continue to be tracked in the frames
- **Track Anything:** algorithm built on ***segment anything***, able to segment objects and modify them

3D Reconstruction: 3D reconstruction given 1 or more images, technique called NERF, very high levels of reliability with just a few shots

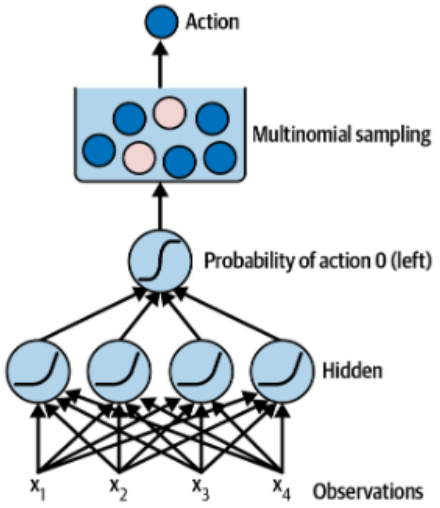
16 Reinforcement Learning (RL)

It is based on the concept of an ***agent*** who through ***observations*** decides to perform ***actions*** in an environment to maximize his ***reward***

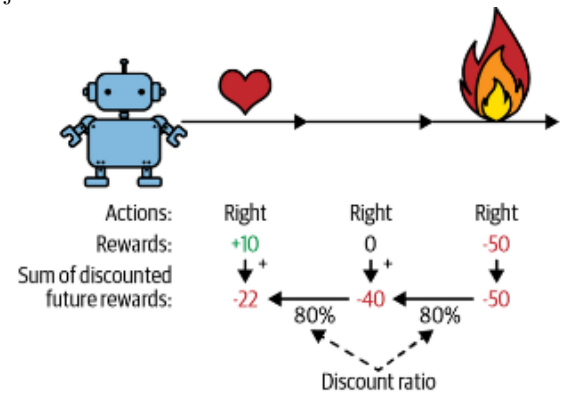
- the algorithm that the agent uses to determine its actions is its ***policy***, the policy could be an NN or another algorithm, each policy can have *one or more parameters* that are gradually adjusted to find the *optimal policy* in a process called ***policy search***, we could choose to randomize the next action or use genetic algorithms to randomize only part of the policy parameters



- facing the environment if we tend to optimized our model a search for ***better solutions*** ⇒ ***exploration***, must always be balanced with optimization of those that we consider to be more performing ⇒ ***exploitation***
- once the network has been defined we must choose at every moment what is the ***best move to maximize our rewards***, we don't know at any moment what the best move is but we necessarily have to get to the ***end of the game to evaluate our score*** ⇒ need to choose policies that allow us to make decisions without having visibility of the immediate result, must therefore ***assign to each move a probability*** that will lead us to maximize the reward in the future
- ***assigning a score to each move*** once reached the reward, we are not necessarily able to assign to each of them whether it had a positive or negative impact ⇒ know problem as ***credit assignment problem***

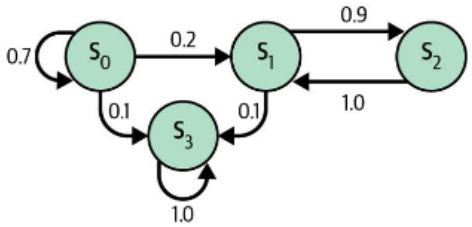


- one way of assigning the reward to each action is to weight the future reward backwards using a ***discount factor*** range that discounts the impact of the different actions over time, ***discount factor close to 0*** only immediate decisions will have an impact, ***discount factor close to 1*** more distant decisions will contribute to achieving the objective



- ***Policy gradients:*** evaluate the gradient of the policy with respect to the reward, one way ***REINFORCE algorithm: Learning from Experience***
  - ***Practice Makes Progress:*** the agent plays the game many times, following its current policy
  - ***Evaluating Choices:*** we calculate how much each action contributed to the final score, rewinding and seeing which choices were more valuable in the long run
  - ***Refining the Policy:*** If an action led to a good outcome, we want to make that choice more likely in the future ⇒ ***positive gradient***, if an action led to a bad outcome, we want to make it less likely ⇒ ***negative gradient***
  - ***Learning from All Games:*** we average the goodness of each action across all the games played and we use this average information to update the agent's policy, making it more likely to choose actions that lead to higher rewards in the future ⇒ helps the agent learn from this feedback by adjusting its policy

- ***Markov Decision Process (MDP):*** a markov chain is a stable memoryless stochastic process in which it transitions from one state to another with a certain probability, each state I an choose to carry out actions, the outcome of which is defined by probabilities



State evaluation:

- ***Bellman optimality:*** optimal value of a state depends only on the optimal values of the states that follow it, each state a value based on ***Bellman optimality equation***, the value of a state is equal to the immediate reward you get for being there + the discounted value of the best next state you can reach from that state ,discounted because future rewards are worth less than immediate ones

$$value = state\ value + disconted\ value\ of\ best\ future\ state\ u\ can\ reach$$

Value iteration algorithm uses Bellman's equation to estimate the optimal value of each state, starts with an initial guess for the value of all states, and then iteratively updates these values ⇒ simple, guaranteed to converge, works well small number of states ⇒ computationally expensive, slow to learn, especially for complex environments, no optimal policy indicator for the agent

- ***Q-value:*** represent the expected future reward (discounted) that an agent can obtain by taking a specific action in a particular state, they are denoted as $Q(s, a)$ where s is current state and a is the action considering taking, focusing on specific actions within a state, Bellman equation guides the update of these Q-values enabling the agent to learn the optimal policy through dynamic programming

The Bellman equation helps you understand that the best decision in a room depends not just on the immediate reward from staying there, but also on the potential rewards you could get by taking different paths. ***Q-values*** represent your estimate of how good each path is, considering the rewards you might get further down the maze. By constantly updating your Q-values based on your exploration (trying different paths and seeing their outcomes), you gradually learn the best route to take ⇒ ***optimal policy***

Algorithms:

- ***Monte Carlo approach:*** he agent waits until the end of an episode (game or simulation) to evaluate the rewards received for all the actions taken. This information is then used to update the agent's policy (the strategy it uses to choose actions) ⇒ inefficiency - waiting for the end especially for long-term planning, the agent only learns from the final outcome not from intermediate steps

- ***Temporal Differencies (TD) algorithm:*** allows agents to learn from experience without needing a complete model of the environment, bridges the gap between waiting for the end (Monte Carlo) and using only immediate rewards, TD methods estimate the value of a state by combining the immediate reward received with the estimated value of the next state (based on a map or past exploration) ⇒ ***Bootstrapping***, advantages: ***fast learning*** learn online and updating their policies as they explore the environment, they can ***learn from incomplete episodes***
- ***Q-learning (QL):*** core concepts: Q-values, Bellman Equation and Temporal Difference Learning, is an algorithm that will watch an agent ***play almost randomly against the environment*** to try to estimate Q-values, ***over time*** the ***Q-table*** gets populated with more accurate estimates of future rewards for each state-action pair, the agent gradually learns which actions lead to higher rewards in the long run allowing it to develop an optimal policy ⇒ allows agents to learn optimal policies through exploration and experience, ***balance exploration vs explotation***:
  - throughout the training phase a ***greedy policy*** is often used ε, therefore with a small probability ε of not choosing the best action but of randomizing it to continue exploring
- ***Deep Q-learning (DQN):*** environments with large number of states ⇒ use a deep neural network to estimate Q-values ***DQN***, we will optimize the network by iteratively interacting with the environment observing the rewards, discounting them for the actions and giving the value as a learning objective to the DQN

Theory	*agent, actions, rewads, policy, policy search*
	exploration, explotation, end of the game
Maximixe reward:	*what actions leads to highest value*
	each action a probability: Policy
	each action a score: credit assignment problem
	discount factor
	policy gradients: REINFORCE alg
Algorithms	*REINFORCE, Value iteration, MDP*
	TD, QL, DQL

17 Riassunto modelli

Regression	*Multiple regression, Polynomial regression, Regression tree,*
Classification	*one-vs-rest (hyperplane), Perceptron, Logistic regression, Naiive Bayes, Decision tree, Large margin classification*
	Linear Support Vector Classifier (L-SVC),
Both	*K-NN, AdaBoost, Gradient Boosting, XGBoost, SVM, Gaussian Processes (GP), Neural Networks*
Anomaly detection	*One Class SVM, CNN (Pathcore), Autoencoders (pre-train VAE + classifier), GANomaly*
Reduce dimensionality	*PCA, ICA, t-distributed stochastic neighbor embedding (t-SNE), Multi Dimensional Scaling (MDS), Isomap*
Clustering methods	*K-Means, DBSCAN, OPTICS, Hierarchical clustering*
Recommender systems	*Content based, Item based (distance metric), User based (distance metric)*
Neural Networks	*Multi Layer Perceptron (MLP), Residual neural network (ResNet), Convolutional Neural Networks (CNN), Non maximal suppression (NMS)*
	Recurrent Neural Networks (RNN), Transformer, Autoencoders, Generative Adversarial Networks (GAN), Diffusion Models
Generative models	*VAE, GAN, Diffusion Models, text generator: char RNN, Transformers*
Reinf. Learning	*REINFORCE algorithm, Monte Carlo approach, Temporal Differencies (TD) algorithm, Q-learning (QL), Deep Q-learning (DQN)*

CNN	*convolutional layers, obj classification, obj localization, obj detection, obj segmentation, semantic segmentation, anomaly detection*
	LeNet5, AlexNet, Google LeNet, ResNet, R-CNN, Faster R-CNN, Yolo, NMS, U-Net, Yolo and Mask R-CNN
	Segment Anything Model (SAM), PatchCore
RNN	*recurrent neuron, SimpleRNN, indetify pattern in a sequence, LTSM, GRU, Bi-directional RNN, 1D Convolution - WaveNet, Char-RNN*
NLP	*One-hot encoding, Term Frequency (TF), Inverse Document Frequency (IDF), TF-IDF, N-Gram, Word2Vec, Doc2Vec, Char-RNN*
	process text, info from text, Autoencoder, CBOW, kip-Gram, temperature, Transformer
Autoencoders	*compressing very sparse information in dense rappresentation, compressing and decompressing*
	learn latent representation in an unsupervised way + features, rebuild missing patterns, generate new data, identify anomalous data
	pre-train, Denoising autoencoder, Variational autoencoder (VAE)
GAN	*generator, discriminator, Deep Convolutional GAN (DCGAN) = style + structure, StyleGANs, GANomaly*

Pre-train	*fine tuning, converge, dirty dataset, high learning rate, supervised pre-train, unsupervised pre-train, fixed weight*
Prevent overf.	*pre-train layers, low learning rate, regularization techniques, layer*
Net. structure	*sequetial model, wide and deep, multi input/output*
Layers	*Softmax, Dropouts, Batch normalization, flatten, concatenate, min, max, averagem reshape, Residual blocks*
	Conv1D, Conv2D, Polling, Dense Layers, Conv2DTranspose, SimpleRNN

Non parametric:	*K-NN, SVM, Gaussian Processes (GP),*
Unsupervised	
Both	*clusteriing methods, Recommender systems*

Approfondimenti

Premises

x, u, φ, ϕ	*they represent inputs, inputs, features*
β, θ, ω	*represent the parameters*
$\hat{y}, y(\omega), \hat{t}$	*represent our estimates*
$x, \boldsymbol{x}, \boldsymbol{X}$	*single data, vector of the variables, complete matrix*
e	*irreducible error of the data*

OLS Matrices

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_{11} & \boldsymbol{x}_{12} & \dots & \boldsymbol{x}_{1p} \\ \boldsymbol{x}_{21} & \boldsymbol{x}_{22} & \dots & \boldsymbol{x}_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{x}_{n1} & \boldsymbol{x}_{n2} & \dots & \boldsymbol{x}_{np} \end{bmatrix}, \qquad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}, \qquad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Metrics

Mean Squared Error (MSE)

$$E(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} \left[y(x_n, \boldsymbol{w}) - t_n \right]^2 = Var(\boldsymbol{w}) + bias^2(\boldsymbol{w})$$

Regression

Classic regression model

$$y = \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_0 + e$$

OLS (Ordinary least squares):

$$(\boldsymbol{X}^T \boldsymbol{X})\hat{\boldsymbol{\beta}} = \boldsymbol{X}^T \boldsymbol{y}$$

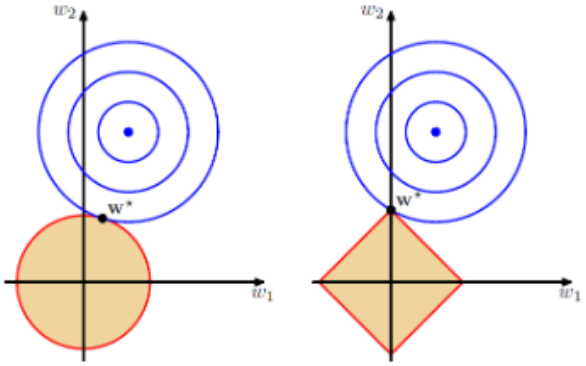
Polynomial regression

$$y(x, \boldsymbol{w}) = w_0 + w_1 x \ldots + w_M x^M = \sum_{j=0}^{M} \omega_j x^j$$

Regularization techniques

$$\widetilde{E}_{aug}(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} [y(x_n, \boldsymbol{w}) - t_n]^2 + \lambda \cdot \Omega(\boldsymbol{w})$$

Lasso regression L1	$\Omega(\boldsymbol{w}) = \sum_{j=1}^{M} \lvert w_j \rvert^{q=1}$
Ridge regression L2	$\Omega(\boldsymbol{w}) = \sum_{j=1}^{M} (w_j)^2$
Elastic net L1 + L2	$\Omega(\boldsymbol{w}) = \beta \sum_{j=1}^{M} (w_j)^2 + (1-\beta) \sum_{j=1}^{M} \lvert w_j \rvert^{q=1}$

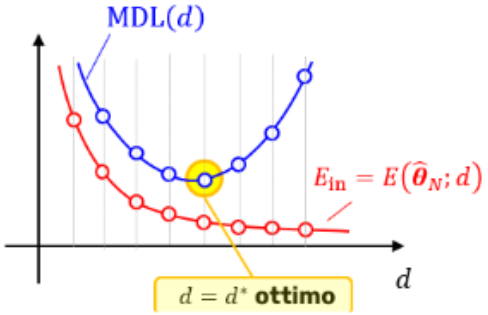


Ridge Regression Lasso Regression

Penalizations formulas

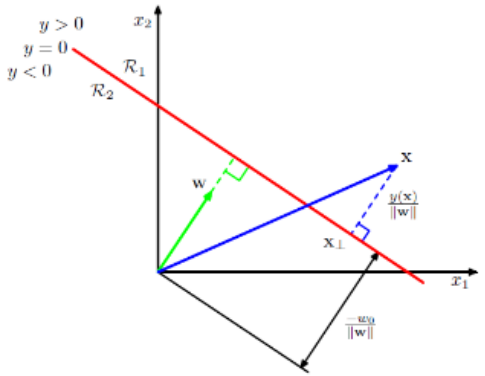
Assuming we have d parameters, N data, where $E(\hat{\boldsymbol{\theta}}_n; d)$ is our performance measure

Akaike Information Criterion	$AIC(d) = 2 \cdot \frac{d}{N} + ln[E(\hat{\boldsymbol{\theta}}_N; d)]$
Minimum Description Length	$MDL(d) = ln[N] \cdot \frac{d}{N} + ln[E(\hat{\boldsymbol{\theta}}_N; d)]$



Binary classification - hyperplane

$$y(x) = w^T x + w_0$$



Perceptron

Transformation fucntion $\quad y(\boldsymbol{x}) = f(\boldsymbol{w}^T \phi(\boldsymbol{x}))$

The Maximal Margin Classifier

- Classification using a separating hyperplane
- There will exist an infinite number of such hyperplanes
- In order to construct a classifier based upon a separating hyperplane ⇒ **maximize the margin hyperplane**
- We can compute the **perpendicular distance** from each training observation to a given separating hyperplane; the smallest such distance is the **minimal distance** from the observations to the hyperplane, and is known as the **margin**
- The maximal margin classifier is a very natural way to perform classification, **if a separating hyperplane exists ⇒ hard margin**
- We can extend the concept of a separating hyperplane in order to develop a hyperplane that ***almost*** separates the classes, using a so-called **soft margin**
- The generalization of the maximal margin classifier to the **non separable case** is known as the s**upport vector classifier**

Support Vector Classifiers

- We might be willing to consider a classifier based on a hyperplane that does not perfectly separate the two classes
- support vector classifier allows some observations to be on the incorrect side of the margin, or even the incorrect side of the hyperplane
- The hyperplane is chosen to correctly separate most of the training observations into the two classes, but **may misclassify a few observations**
- The slack variable ϵ_i tells where the i-th observation is located relative to the hyperplane and relative to the margin
- C defines the weight of how much samples inside the margin contribute to the overall error. Consequently, with C you can adjust how hard or soft your large margin classification should be. C of 0, samples inside the margins are not penalized anymore - which is the one possible extreme of disabling the large margin classification. With an infinite C you have the other possible extreme of hard margins.
- The support vector classifier is a natural approach for classification in the two-class setting, if the **boundary between the two classes is linear**

$\epsilon_i = 0$	*i-th observation is on the correct side*
$\epsilon_i > 0$	*i-th observation is on the wrong side of the margin*
$\epsilon_i > 1$	*i-th observation is wrong side of the hyperplane*
$\sum_i \epsilon_i \leq C$	*number and severity of the violations to the margin*

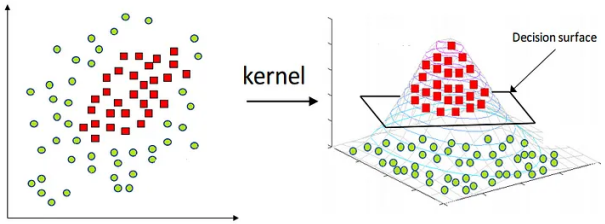
Support Vector Machines

- we could address the problem of possibly non-linear boundaries between classes by enlarging the feature space
- The support vector machine allows to enlarge the feature space used by the support vector classifier in a way that leads to efficient computations
- SVM proposes an efficient computational approach for enlarging the feature space ⇒ **kernel**, **kernel trick**

Kernel

Un kernel è una funzione matematica che permette di trasformare i dati in uno spazio di dimensioni più elevate senza calcolare esplicitamente le coordinate in quello spazio. Questo è utile perché molti problemi che non sono linearmente separabili nello spazio originale possono diventare linearmente separabili in uno spazio di dimensioni più elevate.

Nel contesto delle SVM, il kernel è utilizzato per trasformare i dati originali in uno spazio di caratteristiche più alto (higher-dimensional feature space) dove è più facile trovare un iperpiano che separi le classi. Ciò consente di utilizzare le SVM anche per problemi non linearmente separabili.



Linear Decision Boundary

A linear decision boundary is chosen when the data is linearly separable or can be adequately separated using a straight line (in two dimensions) or a hyperplane (in higher dimensions).

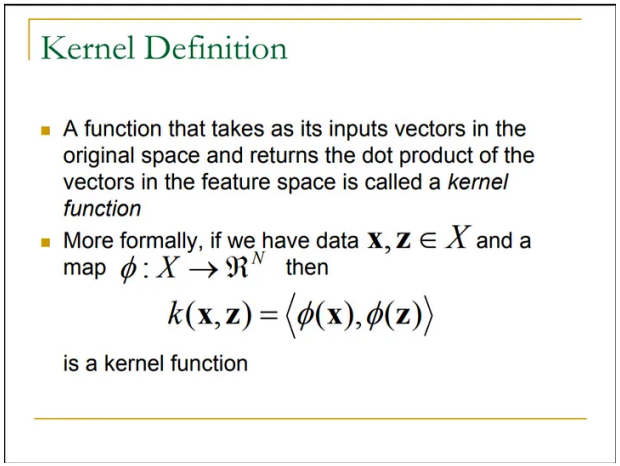
Non Linear Decision Boundary

A non-linear decision boundary is chosen when the data cannot be separated by a straight line or hyperplane in its original feature space. Non-linear SVMs achieve this by using kernel functions to transform the data into a higher-dimensional space where a linear decision boundary can be applied.

Kernel

- There can be many transformations that allow the data to be linearly separated in higher dimensions, but not all of these functions are actually kernels
- The kernel function has a special property that makes it particularly useful in training support vector models, and the use of this property in optimizing non-linear support vector classifiers is often called the kernel trick
- The **trick** is that kernel methods represent the data only through a set of pairwise similarity comparisons between the original data observations $\boldsymbol{x}$ (with the original coordinates in the lower dimensional space), ***instead of*** explicitly applying the **transformations** $\phi(x)$ and representing the data by these transformed coordinates in the higher dimensional feature space.
- Our kernel function accepts inputs in the original lower dimensional space and returns the dot product of the transformed vectors in the higher dimensional space

- The ultimate benefit of the kernel trick is that the objective function we are optimizing to fit the higher dimensional decision boundary **only includes the dot product** of the transformed feature vectors. Therefore, we can just substitute these dot product terms with the kernel function, and we don't even use $\phi(x)$.
- our data is only linearly separable as the vectors $\phi(x)$ in the higher dimensional space, and we are finding the optimal separating hyperplane in this higher dimensional space without having to calculate or in reality even know anything about $\phi(x)$



Equation 5-9. Kernel trick for a 2nd-degree polynomial mapping

$$\phi(\mathbf{a})^T \cdot \phi(\mathbf{b}) = \begin{pmatrix} a_1^2 \\ \sqrt{2}\, a_1 a_2 \\ a_2^2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1^2 \\ \sqrt{2}\, b_1 b_2 \\ b_2^2 \end{pmatrix} = a_1^2 b_1^2 + 2 a_1 b_1 a_2 b_2 + a_2^2 b_2^2$$

$$= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right)^2 = \left(\mathbf{a}^T \cdot \mathbf{b} \right)^2$$

Algoritmo t-SNE

Il t-distributed stochastic neighbor embedding (t-SNE) è una tecnica di riduzione dimensionale utilizzata per proiettare dati con un numero molto grande di caratteristiche in uno spazio bidimensionale (2D) o tridimensionale (3D) per facilitarne la visualizzazione. Per comprendere meglio come funziona l'algoritmo t-SNE, è utile esaminare le due distribuzioni di probabilità coinvolte e il processo di minimizzazione della distanza KL (Kullback-Leibler).

- **Distribuzione di probabilità nello spazio N-dimensionale:** t-SNE calcola una distribuzione di probabilità che misura la similarità tra coppie di punti nello spazio originale ad alta dimensionalità. La probabilità che un punto j sia il vicino di un punto i è calcolata utilizzando una distribuzione Gaussiana centrata su i. Questo implica che i punti vicini avranno una probabilità alta, mentre i punti lontani avranno una probabilità bassa
- **Distribuzione di probabilità nello spazio ridotto (2D o 3D):** t-SNE utilizza una distribuzione t di Student con un numero basso di gradi di libertà (di solito 1) per confrontare le distanze tra i punti. Questo permette di mantenere i vicini stretti vicini e di separare meglio i punti che sono lontani tra loro
- **Minimizzazione della distanza Kullback-Leibler (KL):** L'algoritmo t-SNE cerca di fare in modo che le due distribuzioni di probabilità siano il più simili possibile. La distanza KL è una misura di divergenza tra due distribuzioni di probabilità. L'obiettivo è minimizzare questa distanza affinché la distribuzione delle distanze nello spazio ridotto rispecchi il più possibile la distribuzione delle distanze nello spazio originale.

OPTICS

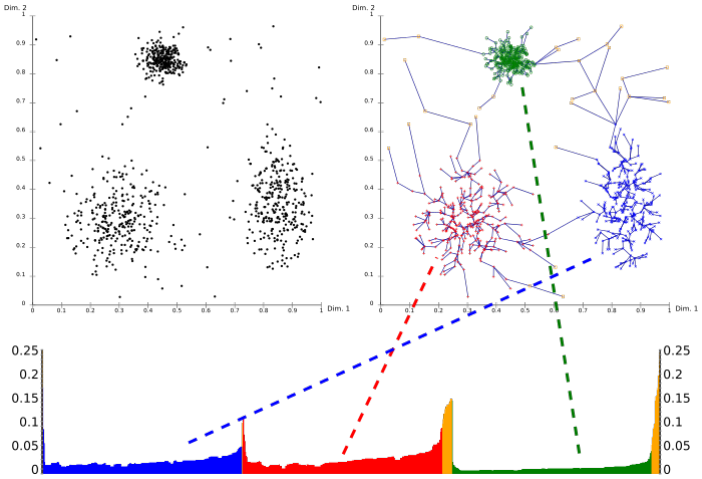
The OPTICS (Ordering Points To Identify the Clustering Structure) algorithm is a density-based clustering technique similar to DBSCAN, but more flexible and capable of finding clusters with varying densities. The goal is to order points in such a way that the clustering structure can be identified via a reachability plot. This ordering reflects the structure of clusters in the dataset. Additionally, a special distance is stored for each point that represents the density that must be accepted for a cluster so that both points belong to the same cluster.

Key Concepts:

- $\boldsymbol{\varepsilon}$: the maximum distance (radius) to consider
- ***MinPts:*** the min number of points required to form a dense region/cluster
- A point p is a **core point** if at least $MinPts$ points are found within its $\varepsilon-$neighborhood $N_\varepsilon(p)$
- ***Core Distance:*** describes the distance to the $MinPts - th$ closest point
- ***Reachability Distance:*** the minimum distance required for a point p to reach another point o, considering o as a core point. It is calculated as:

$$reachability\_distance(p, o) = max(core\_distance(o), distance(o, p))$$

- **Initialization:**
  - For each point, calculate the core distance if the point is a core point
  - Initialize all reachability distances as infinite
- **Building the Ordered List**
  - Start with an arbitrary unvisited point and insert it into the ordered list
  - Update the reachability distance for each neighboring point (distance $\leq \varepsilon$) if the starting point is a core point.
  - Select the next unvisited point with the smallest reachability distance and repeat the process and continue until all points have been visited
- **Extracting Clusters:** once the ordered list is completed, clusters can be identified by examining the **reachability-plot (a special kind of dendrogram)**, ***valleys*** in the reachability plot represent dense clusters, while **peaks** indicate separation between clusters. The yellow points in this image are considered noise



The upper right part visualizes the **spanning tree** produced by OPTICS, and the lower part shows the reachability plot as computed by OPTICS.

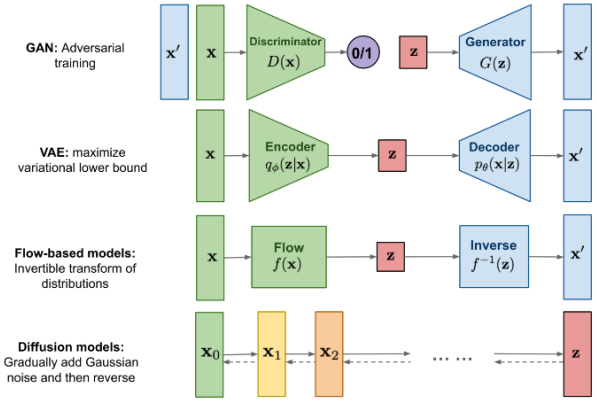
Neural network

- Multi Layer Perceptron (MLP)

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

Copyright © Sebastian Raschka 2016 (http://sebastianraschka.com)

Image processing models



Unsupervised learning

- standard approach is to try different actions and see which ones lead to the highest score
- Instead of just picking the best actions overall, we analyze the impact of individual choices
- We use a concept called "policy" - a set of rules that guide the agent's decisions.
- We want to improve the policy by adjusting it in ways that lead to higher rewards.

Core Concepts:

- **Q-values:** Q-learning uses Q-values, which represent the expected future reward (discounted) an agent can obtain by taking a specific action (a) in a particular state (s). These are stored in a Q-table.
- **Bellman Equation:** This equation guides how Q-values are updated. It states that the value of a state is equal to the immediate reward received for being in that state, plus the discounted value of the best action that can be taken from that state.
- **Temporal Difference Learning:** Q-learning leverages TD learning's concept of bootstrapping. Instead of waiting for the end of an episode to learn, it estimates the value of a state using the immediate reward and the estimated value of the next state.

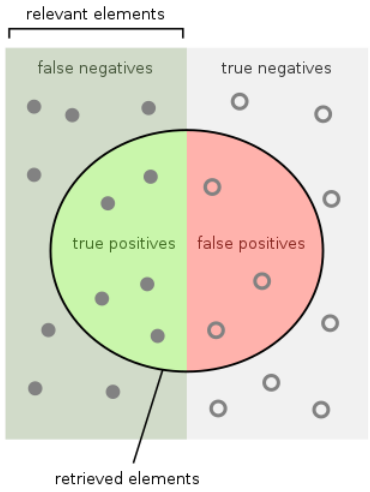
Terms

- ***Neural network (NN):*** consists of interconnected nodes (artificial neurons) arranged in layers: input, hidden, output. NNs are powerful for various tasks like classification, regression, and pattern recognition. They are computationally less expensive to train compared to DNNs
- ***Deep neural network (DNN):*** is essentially a neural network with multiple hidden layers (usually more than 3). These additional layers allow DNNs to learn more complex relationships and patterns in data, making them suitable for tasks like: image recognition, Natural language processing (NLP), Speech recognition.

Training DNNs requires more data and computational power compared to basic NNs. There's a risk of overfitting (memorizing the training data instead of learning general patterns).

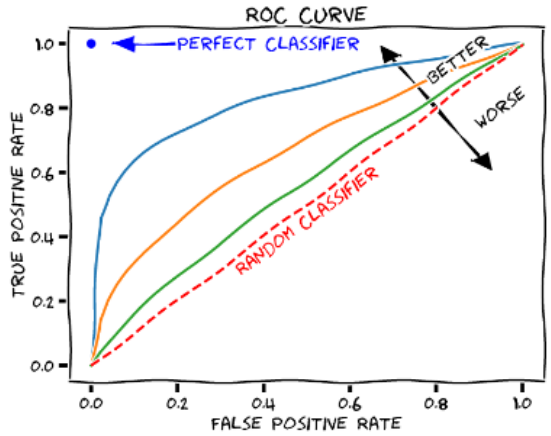
Classificazione

- ***Precision:*** This focuses on the accuracy of your positive predictions. In your example, it asks: "Out of all the images where you said there's a dog, how many actually have a dog?" A high precision means your model rarely makes mistakes when identifying dogs.
- ***Recall:*** This focuses on the completeness of your positive predictions. It asks: "Out of all the images that actually have a dog, how many did your model identify correctly?" A high recall means your model finds most of the dogs in the data.
- ***True Positive (TP):*** This is a correct positive prediction. The model identifies something as positive, and it actually is positive in reality.
- ***True Negative (TN):*** This is a correct negative prediction. The model identifies something as negative, and it actually is negative in reality.
- ***False Positive (FP):*** This is an incorrect positive prediction. The model identifies something as positive, but it's actually negative.This is also sometimes called a Type I error
- ***False Negative (FN):*** This is an incorrect negative prediction. The model identifies something as negative, but it's actually positive. This is also sometimes called a Type II error.



ROC - Receiver Operating Curves

- The X-axis of the ROC curve represents the False Positive Rate (FPR), which is the proportion of negative instances incorrectly classified as positive. The Y-axis represents the True Positive Rate (TPR), which is the proportion of positive instances correctly classified as positive.
- Ideally, a good classifier would strive to maximize the TPR (correctly classifying positive instances) while minimizing the FPR (incorrectly classifying negative instances).
- The diagonal red line represents a random classifier, a classifier that performs no better than random chance. The farther a curve moves towards the top left corner (1,1), the better the performance of the classifier.
- A perfect classifier, by definition, flawlessly separates positive and negative instances. Regardless of the chosen ***threshold***, it will always correctly classify all positive cases (TPR = 1) and make no mistakes with negative cases (FPR = 0).
- Each point on a ROC curve in a binary classification task corresponds to a specific classification threshold and the resulting performance of the model at that threshold.
- ***Lowering the Threshold:*** This tends to increase the TPR (more positives are captured) but also increases the FPR (more mistakes are made by including irrelevant negatives). ***Raising the Threshold:*** This tends to decrease the TPR (fewer positives are captured) but also decreases the FPR (fewer mistakes are made)
- ***Lower Threshold (Right X-axis):*** A lower threshold allows the model to classify more instances as positive, potentially including some false positives. ***Higher Threshold (Left X-axis):*** A higher threshold makes the model more cautious, potentially missing some true positives (decreases FPR)



18 Decision tree

- ***Nodes:*** These are the decision points in the tree structure. The root node is at the top, and branches lead to child nodes based on the values of features.
- ***Separated Categories:*** In classification tasks, the goal is to divide the data into distinct categories (like "spam" or "not spam" for emails)
- ***Impurity Measure:*** This is a mathematical value that tells you how "mixed up" the categories are within a node. A high impurity means the data points at that node belong to various categories, and the split isn't very effective. Conversely, a low impurity indicates a good separation, with most data points belonging to the same category