Machine Learning and AI

- Methods and Algorithms -

Personnal Notes

François Bouvier d'Yvoire

CentraleSupélec & Imperial College

Current Branch : `master`
Commit : `71140066dbf1fc8dbda5814d3729d43b905fdaf3`

Contents

Todo list

Intro

This document is currently under developement and none of the chapter are finish, nor in a good shape.

This Document is a simple survey of the Machine Learning and AI method that I encounter. Most of it are based on lecture notes from my dual MSc between CentraleSupélec and Imperial College. The first goal is to be used as a personnal reminder of what I have already encounter and understood.

The exemple below is a scheme wich classify the machine learning algorithms in a certain way. This document does not follow this classification. All the definition and concept will not be explains, as a lot of ressources are available on those subjects (Computer Sciences benefits from the most open-sourced field in research). Sometimes ressources might provided, but the reader is invited to look himself for the subjects he does not understand well.

Link figure

Figure 1 – Simple graph for algorithms classification in ML

Chapter 1

Common Machine Learning algorithms

This chapter is dedicated to the most common ML algorithms, a major part of the notes come from the mml-books.com

Add bibtex reference

It is intended to introduce different concept used in most Machine Learning Algorithms before moving to more specific algorithms.

1.1 Graphical Model for Probabilistic Inference

Based on the Probabilistic Inference Course of Marc Deisenroth (Imperial College)

1.1.1 Probabilistic Pipeline

Here is a simple pipeline of the inference process with a model

1.1.2 Probabilistic graphical Model

In order to deal with complex and big probalistic model, we can use different kind of graphs which represent relationships between random variables. We define the random variables as nodes and the probabiistic or functional relationship between variables as edges in the graphs. With them you can :

- Visualize the structure
- Have insights into properties (such as conditional independance)
- Design or Motivate new models
- Compute some inference and learning as graphical manipulations

There exists 3 kinds of model : Bayesian networks (directed graphical models), Markov random fields (undirected graphical models) and factor graphs (with nodes which are not random variables)/

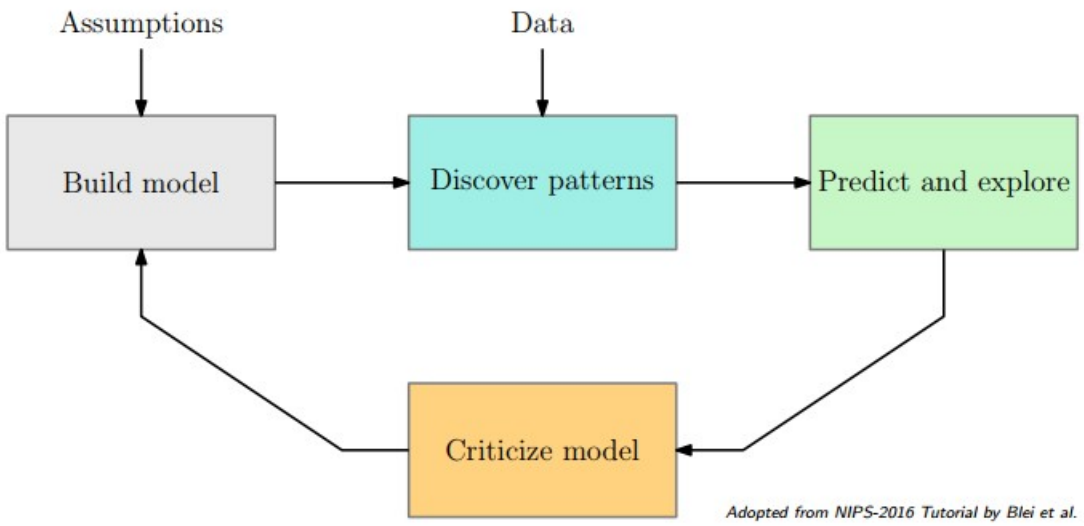

Figure 1.1 – Simple Pipeline of how to build model for inference

Bayesian Networks

add basic graphs exemple

They are defined by

- Nodes : Random variables
- Shaded nodes: Observed random variables
- Other : Latent Variables
- Edges : (a, b) $\Longleftrightarrow$ conditionnal distribution $p(b|a)$

add graphicals model of linear regression

D-Separation: A set A of nodes is d-Separated (conditionnaly independant) from B by C iff all path between A and B are blocked. C is tht set of observed variables in the graphical model.

Definition 1
A Path is **Blocked** *id it includes a node such that either :*

- *The arrow on the path meet either head-to-tail or tail-to-tail at the node, and the node is in C*

- *The arrow meet head-to-head at the node, and neither the node nor any of its descendants is in the set C*

Markov Random Fields Defined by : if X and Y are not connected directly, then X and Y are conditionnaly independant given everything else.

Then Joint distributions are the product of cliques in the graphs, and the full joint (of all variables) is the product of the joint of the maximum cliques

$$p(x) = fra1Z\Pi_C \psi_C(x_C)$$

Where C describe a maximal clique, x_C all the variable in C, ψ_C the clique potential and Z a normalisation constant.

Check for the condiational independance : Check if when remmoving the observable variable there is no path between two set of random variable.

The potentials can be seen as Energy Functions, which mean we can write $\psi_C(x_C) = exp(-E(x_C))$ with E some energy functions, and then $-log(p(x)) = \sum_C E(x_C)$ is the sum of energies of the clique potential.

Add Image Denoising exemple

Factor Graphs (Un)directed graphical models express a global function of several variables as a product of factors over subsets of those variables

Factor graphs make this decomposition explicit by introducing additional nodes for the factors themselves

1.2 Linear Regression

The Linear regression problem correspond to find a linear mapping $f(x)$ based on noisy observation $y = f(x) + \epsilon$, where ϵ is a noise. Finding the regression function require :

- Choice of parameters (function classes, dimension)
- Choice of probabilistic model (Loss function, ...)
- Avoiding under and overfitting
- Modeling uncertainty on data

Definition 2

- $p(x,y)$ *is te joint distribution*
- $p(x)$ *and* $p(y)$ *are the marginal distributions*
- $p(y|x)$ *is the conditional distribution of* y *given* x

- *in the context of regression,* $p(y|x)$ *is called likelihood,* $p(x|y)$ *the posterior,* $p(x)$ *the prior and* $p(y)$ *the marginal likelihood or evidence.*
- *they are related by the Bayes' Theorem :* $p(x|y) = \frac{p(y|x)p(x)}{p(y)}$

1.2.1 Conjugacy

In order to compute the posterior, we require some calculations which imply the prior, and can be intractable as a closed form. But given a likelihood, it can exist prior which give closed-form solution for the posterior. This is the principle of conjugacy (between the likelihood and the prior)

1.2.2 Maximum Likelihood Estimation (MLE)

In order to obtain the parameters, we need to choose a quantity to optimize. The first and most common idea is to choose the Likelihood as a cost function to optimize. The seems natural because the likelihood represent the probability given the X variables to obtain the observed variables Y and we want this to be maximum on the known data (which are the training set).

$$\theta^* = \arg\max_{\theta} p(y|X,\theta)$$

To find the desired parameters that maximize the likelihood, we typically do gradient ascent (or gradient descent on the negative likelihood). For numerical reasons, we apply the log-transformation to the probleme and minimize the negative log-likelihood.

Closed-Form Solution

In some cases, a closed-form solution exist, which make computation easy (but not necesseraly cheap)

Maximum A Posteriori Estimation (MAP)

1.3 Gradient Descent

1.3.1 Simple Gradient Descent

1.3.2 Gradient Descent with Momentum

1.3.3 Stochastic Gradient Descent

1.4 Model Selection and Validation

1.4.1 Cross-Validation

1.4.2 Marginal Likelihood

1.5 Bayesian Linear Regression

1.5.1 Mean and Variance

1.5.2 Sample function

Chapter 2

Dimensionality Reduction and Feature Extraction

The objective of this chapter is to explain dimensionality Reduction, Feature extraction and methods to achieve this. Those method can be then applied to different kind of data and model, in any domain.

2.1 Principal Component Analysis

The objective of PCA is to find a set of features, via linear projections, that maximise the variance of the sample data. We need to decide how many dimension d we want to keep.

2.1.1 Simple algorithm

Data: Vectors x_i of **centered** data, number F features and n sample.
Dimension d of reduction.
Result: Y of size $d \times n$
Compute the product matrix of centered data : $\boldsymbol{X}\boldsymbol{X}^\top$;
Compute the Eigen Analysis $XX^\top = V\Lambda V^\top$;
Order the Eigen Value by descending value, and permute Column of V correspondly;
Compute the eigenvectors : $U = XV\Lambda^{-1/2}$;
Keep specific number of first components : U_d the d first d column of U ;
Compute the new features vectors : $Y = U_d^\top X$

Algorithm 1: Simple PCA Algorithm

2.1.2 Whitening PCA

The feature given by the PCA algorithm are un-correlated, but the variance in each dimension are not the same (in fact this are the eigen value of $XX^\top$). We can whitening the features (or "sphering" them) by making the covariance matrix equal to Identity.

Data: Vectors x_i of **centered** data, number F features and n sample.
Dimension d of reduction.
Result: Y of size $d \times n$ with Identity Covariance Matrix
Compute the PCA of X and keep U and Λ;
Compute the Eigen Analysis $XX^\top = V\Lambda V^\top$;
Compute the whitened features vectors : $Y = U_d \Lambda^{-1/2} X = (XV\Lambda^{-1})_d X$

Algorithm 2: Whitened PCA Algorithm

2.1.3 Kernel PCA

Sometimes we want to compute non-linear features extractions. We use the kernel method to compute this. For a dataset $X = (x_i)_{1\dots n}$ we know only the kernel matrix $K = [\phi(x_i)\phi(x_j)^\top]_{(1\dots n)^2} = X^\phi X^{\phi^\top}$ with ϕ the non-linear mapping.

Data: Vectors x_i of **centered** data, number F features and n sample.
Dimension d of reduction. K the kernel matrix
Result: Y of size $d \times n$
Compute the Eigen Analysis $K = \boldsymbol{X}^\phi \boldsymbol{X}^{\phi^\top} = V\Lambda V^\top$;
Order the Eigen Value by descending value, and permute Column of V correspondly;
Keep specific number of first components : V_d the d first d column of V ;
Compute the vector $g(x_t) = [k(x_i, x_t)]_{1..n}]$;
Compute the $E = \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ matrix ;
Compute the new features vectors : $y_t = \Lambda^{-1/2} V^\top (I - E)\left(g(x_t) - \frac{1}{n} K\mathbf{1}\right)$

Algorithm 3: Kernel PCA Algorithm

Chapter 3

Bayesian Algorithms

Bayesian Algorithms aims to produce not just function (for classification or regression) but distribution over function, which achieve to get the uncertainty of our model according to the uncertainty of the data.

3.1 Gaussian Process

This section is made with a great inspiration from the Probabilistic Inference from Marc Deisenroth (Imperial College London)

3.1.1 Problem setting

For a set of observation $y_i = f(x_i) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2$, we want to fine a distribution over **functions** $p(f)$ that explains the data. It's not exactly the same as a linear regression problem because we do not look for only one function.

This is a really power full process used in a lot of different problematic thanks to its robustness and known properties (in comparison to deep learning methods).

- Reinforcement Learning and Robotics
- Bayesian optimization
- Geo-statistics
- Sensor networks
- Time-series modelling and forecasting
- High-energy physics
- Medical application

Formally a Gaussian Process is a multivariate Gaussian distribution with infinite variables (countable or even uncountable).

3.1.2 Definition

A Gaussian process is defined by a mean function $m(\dot{)}$ and a covariance function (=kernel) $k(\dot{,}\dot{)}$.

The mean function represents the average of all the function.
the Covariance function allows us to compute covariance between any two functions. Notes that the function are unknown, and only this correlations are fully known.

3.1.3 Gaussian Process Inference

Considering X training inputs and y training target, the Bayes' theorem in the case of Gaussian Process become

$$p(f|X,y) = \frac{p(y|f,X)p(f)}{p(y|X)}$$

Which gives us a Likelihood $p(y|f,X) = \mathcal{N}(f(X), \sigma^2 \boldsymbol{I})$, a Marginal Likelihood $p(y|X) = \int p(y|f,X)p(f|X)df$ and a Posterior $p(f|y,X) = \mathbf{GP}(m_{post}, k_{post})$

How can we manage to work with the distribution over function, and then infinite dimension for calculus, etc. ? This is possible because each time you consider only finite sample, computing the joint distribution boils down to work on finite-dimensional multivariate Gaussian distributions

Then we can use the gaussian properties (including gaussian prior as conjugate, ...) to make prediction :
(We define X_*, f_*, etc as the test data and predicted function)

$$p(f(x_*)|X,y,x_*) = \mathcal{N}(m_{post}(x_*), k_{post}(x_*,x_*))$$

This can also be seen as

$$p(f(x_*)|X,y,x_*) = \mathcal{N}(\mathrm{E}[f_*|X,y,X_*], V[f_*|X,y,X_*])$$

with $\mathrm{E}[f_*|X,y,X_*]$ = prior mean + "Kalman gain" * error $= m(X_*) + k(X_*,X)(\boldsymbol{K} + \sigma_n^2\boldsymbol{I})^{-1}(y - m(X))$ and $V[f_*|X,y,X_*] = k(X_*,X_*) - k(X_*,X)(\boldsymbol{K} + \sigma_n^2\boldsymbol{I})^{-1})k(X,X_*)$

Chapter 4

Ensemble Algorithms

Idea : Aggregate the predictions of a group of predictors (either classifiers or regressors)

In many case the aggregated answer is better than best individual prediction.

add illustration from ML for imaging course

Even weak learner can achieve high accuracy, provided there are un sufficient number and sufficently diverse.

Voting toy exemple ?

Definition 3

- *Homogenous Ensemble Learning Use the same class of ML model (which are Weak Learner)*
- *Heterogenous Ensemble Learning Use different ML model*
- *Sequential : Base Learners are added one at a time, mislabelled examples are up-weighted each time(Ex : Boosting)*
- *Parallel : many independent base learners are trained simultaneously and then combined (Ex : Voting, Bagging, Random Forest)*

Definition 4

Weak Learner *: is defined to be a classifier that is only slightly correlated with the true classification (0.5 for bi-classifier)*

4.1 Bagging : Boostrap Aggregating

Reduce model variance through averaging

Add Bootstrapping 3 steps and Aggregating

4.1.1 Outof-bag error

4.2 Boosting

4.2.1 Principle

Rather than building independent weak learners in parallel and aggregating at end: – build weak learners in serial – but adaptively reweight training data prior to training each new weak learner – in order to give a higher weight to previously misclassified examples

4.2.2 Exemple ?

4.2.3 Several variants

• Adaboost (adaptive boosting) – Originally for classification – Can be used for regression too • Gradient Tree Boosting • XGBoost

Adaboost

Adaboost = Adaptive Boosting

Optimizing Adaboost with early termination

4.3 Decision Trees

Chapter 5

Deep Learning

5.1 Intro

5.1.1 Representation Learning

Representation Learning is the field where you try to learn the representation (features) of input. Then you can do classification or regression on the representative space.

5.1.2 Deep Learning vrsus classical Representation Learning

Add image with where Deep Learning is relative to ML

5.2 Optimisation : Gradient Descent

5.2.1 Stochastic GD versus Mini-Batch

Classicaly the learning is an optimization problem :

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{1}^{n} l(f_\theta(x_i), y_i)$$

Then we can calculate a gradient

$$\nabla_\theta \hat{L}(\theta) = \frac{1}{n} \sum_{1}^{n} \nabla_\theta l(f_\theta(x_i), y_i)$$

There is different possibility to update the parameters, the stochastic way (SGD) which learn only one by one exemple:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla_\theta l(f_\theta(x_t), y_t)$$

And the mini-batch which average the learning on a mini-batch (b < n the data set size) of examples.

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \frac{1}{b} \sum_{1}^{b} \nabla_\theta l(f_\theta(x_{tb+i}), y_{tb+i})$$

5.2.2 Convergence rate and computational complexity

$$\mathbb{E}(\hat{L}(\theta^{(k)} - \hat{L}(\theta^*))) \leq \frac{\alpha\sigma^2}{2m} + (1 - \alpha m)^k (\hat{L}(\theta^{(0)} - \hat{L}(\theta^*)))$$

With conclusion :

- Small step size $\Longrightarrow$ **Slower** initial convergence and Stalls at **more** accurate result
- Large step size $\Longrightarrow$ **Faster** initial convergence and Stalls at **less** accurate result
- Small batch size $\Longrightarrow$ Stalls at **less** accurate result and **Lower** iteration cost
- Small batch size $\Longrightarrow$ Stalls at **more** accurate result and **Higher** iteration cost

5.2.3 Deep Learning is not convex

which mean you don't know if local extrema is global extrema, and this is problematic for gradient descent method, which find local extrema.

Add ref to convexity

5.3 Maximum Likelihood estimation

Kullback-Leibler divergence

cross entropy

Log-likelihood

L_2 as MLE

Logistic Regression

5.4 Regularization

In order to avoid certain problem (overfiting, saturation...) regularization can be added to the loss function, (can be named weight decay). This works in restricting the hypothesis class, and find explanation in different way, such as Ockahm Razor.

5.4.1 Generality

General form :

$$\min_{\theta} \hat{L}(\theta) + \beta R(\theta)$$

- Soft constraint
- Equivalent to hard constraint (Lagrangian)
- equivalent to the Maximum a posteriori interpretation (see Common ML)

The regularization reduce the capacity of the model, so a trade-off exist between large capacity but under-regularisation (overfitting or not convergence) and small capacity with over-regularization

Bayesian view todofill MAP equation

5.4.2 Geometric interpretation

5.4.3 L_1 Regularization

5.4.4 Noisy input

We can show that add an i.i.d noise $\epsilon \sim N(0, \beta \boldsymbol{I})$ is equivalent to weight decay.

5.4.5 Data Augmentation

5.4.6 Early stopping

Dropout

5.4.7 Covariate shift

5.4.8 Batch normalization

5.4.9 Input pre-processing

- Zero-centered data
- normalized data
- Decorrelated data
- Whitened data

5.5 Deep Feed-forward network

5.5.1 Perceptron

Activation function

- tanh
- sigmoid
- ReLU -> Logistic regression

Multi-layer perceptron Single layer is not enough

XOR Problem

Theorem 1
A perceptron with one hidden layer of finite width can arbitrarily accurately approximate any continuous function on a compact subset of $\mathbb{R}^n$ *(under mild assumptions on the activation function)*

What is better: Depth or Width ?

Popular output layer types

- 1D Regression
- Binary classification
- nD Regression
- Softmax (multi-class classification)

Backpropagation Use chain rule to calculate all the layer gradient relative to the output gradient.

Differentiable programming

5.6 Deep convolutionnal Neural network

See Imaging chapter first

Chapter 6

Imaging

6.1 Image Classification

6.1.1 Features and Classifier

6.1.2 Feature Extraction - without ML

- Why features ?
- What features should be :
  – Distinctive and discriminative – Local (to enable establishing correspondences) – Invariant to viewpoint changes or transformations (translations/rotations) – Invariant to illumination changes – Efficient to compute – Robust to noise and blurring – Should be hierarchical
- Many options possible : – Intensities – Gradient – Histogram – SIFT – SURF – BRIEF
- Pixel level bad : Not discriminative, Localiser and does not represent local pattern, not invariant to transformation.
- Patch Level : More discriminative, Not necessarily rotation invariant, semi-localised. Use Convolution for patch.
- Image Level : discriminative, Not localised, not necessarily rotation invariant. Get a feature map where each pixel correspond to local pattern

How to make rotation invariant ? → Use Histogram.

Filtering

- Use filter to identify features such as gradient, edge detection.
- Gradient are invariant to absolute illumination.

SIFT

- Scale-space Extrema Detection : For all scale, compute convolution with gaussian (blur at different level) and compute the difference of each scaled guassian, then Detect local extrema of Differences
- Keypoint Localisation :
- Orientation Assignment : Take a small window around the keypoint location.An orientation histogram with 36 bins covering 360 degrees is created. Each pixel votes for an orientation bin, weighted by the gradient magnitude after applying a Gaussian filter with he keypoint scale !). The keypoint will be assigned an orientation, which is the mode of the distribution
- Keypont descriptors.

OHoG (Histogram of Gradient)

SURF (Speeded-Up Robust Features)

BRIEF (Binary Robust Independant Elementary Features)

Local binary Patterns

Haar features

Convolution ineficient → Faster computation via integral images S

6.1.3 with ML

Classification or regression model, many options : Logistic regression, Naives Bayes, K-Nearest neighbors, Support vector Machines (does not deal good with computer vision , Boosting, Decision/Random forsest, Neural network.

Boosting and Decision trees are Ensemble forests.

add bias-variance error explanation in common ML

Add Classical ML pipeline for computer vision (images, features, classification)

why ?

6.2 Segmentation

- Quantify analysis
- Location and extent of object
- Generating 3D models for simulation

6.2.1 Semantic Segmentation

Chapter 7

NLP

7.1 How to represent word

7.1.1 Bag of Word

7.1.2 Mapping with concept

7.1.3 Features extraction

7.1.4 Vector space model

7.2 Learning word representation

7.2.1 Word embeddings : word2vec

Continous Bag-of-Words model

Skip-gram model

Word embedding layer

Negative Sampling

Exemple of vector representation

Other tricks

Other kind of embeddings word

7.3 Some Algorithm for NLP

7.3.1 Naives Bayes Classifier

Used a lot in anti-spam algorithm.

Chapter 8

Reinforcement Learning

8.1 Markov Reward and Decision Process

8.1.1 State Value Function Closed-form

For a Markov Reward Process $(\mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma)$, with

- $\mathcal{S}$ the States
- $\mathcal{P}$ The transition matrix
- $\mathcal{R}$ the reward matrix
- $\gamma$the discount

We define then the Return R_t and the State Value Function $\mathsf{v}(s) = \mathbb{E}[\mathsf{R}_t S_t = s]$
Then we have, in a vector form :

$$\mathbf{v} = (\mathbb{1} - \gamma\mathcal{P})^{-1}\mathcal{R}$$

Unfortunately, Matrix inversion in costly, so this is only feasible in small Markov Reward Process

8.1.2 Iterative Policy Evaluation Algorithm

In RL, we need not only the value of different state, but also a policy to define which action to take in any state. Then, we had an action space $\mathcal{A}$ and a policy π to a MRP to obtaine what is called a Markov Decision Process (MDP).

The first algorithm is a method to evaluate a policy. We don't need to store all the previous value of V while updating, and furthermore, it converges faster this

way.

Data: a MDP $(\mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma)$ and π the policy to be evaluate
Result: The value function V^{π}
Initialise V(s) =0 $\forall s$;
while **Convergence condition (number of epoch, $\Delta \leq$ threshold, ...)** **do**
  forall $s \in S$ **do**
    $v \leftarrow V(s)$;
    $V(s) \leftarrow \sum_{\alpha} \pi(s, \alpha) \sum_{s'} \mathcal{P}^{a}_{ss'} [\mathcal{R}^{a}_{ss'} + \gamma V(s')]$;
    $\Delta \leftarrow \max(\Delta, \mid v - V(s) \mid)$;
  end
end
Return V

Algorithm 4: Iterative Policy Evaluation Algorithm

8.2 Dynamic Programming in RL

In order to compute an optimal policy, we can do better than trying every policy and evaluating their V^{π}, the first class of algorithm enter in the Dynamic programming family. Convergence are assured by theorem relative to the Belmann equation (Policy Improvement theorem, Belmann principle of optimality, ...).

8.2.1 Policy Iteration Algorithm

First algorithm simply apply a greedy **deterministic** policy relative to the Value function, and recalculate the new Value function, until the policy is stable (and according to theorem, optimal).

Data: a MDP $(\mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma)$
Result: The optimal policy $\pi*$ and his corresponding value function $V^{\pi*}$
Initialise V(s)=0 and $\pi(s) \in \mathcal{A}, \forall s$;
while **Convergence condition (policy is stable, or number of epoch)** **do**
  Do a Policy Evaluation for π (with the Iterative Policy Evaluation for exemple) **forall** $s \in S$ **do**
    $b \leftarrow \pi(s)$;
    $\pi(s) \leftarrow \arg\max_{a} \sum_{s'} \mathcal{P}^{a}_{ss'} [\mathcal{R}^{a}_{ss'} + \gamma V(s')]$;
    policy-not-stable $\leftarrow (b \neq \pi(s)) \vee$ policy-not-stable
  end
end
Return V

Algorithm 5: Iterative Policy Evaluation Algorithm

This can be seen as the following scheme :

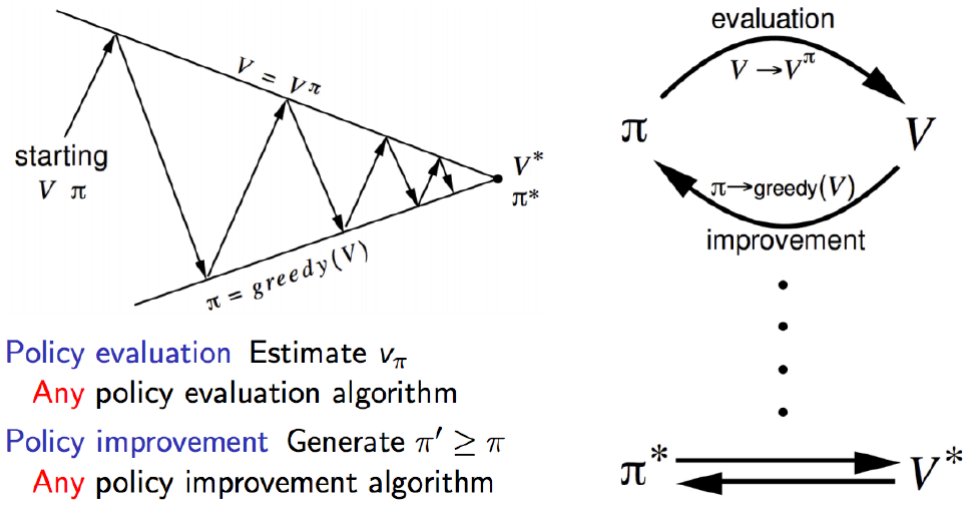

Figure 8.1 – Generialised Policy Iteration Algorithm

8.2.2 Value Iteration Algorithm

It can be shown that we don't need to wait for the convergence of the Value Function before iterating the policy in the previous algorithm. We can shorten some steps, and even get rid of explicity updating a policy (we are always using the greedy policy wrt. $V(s)$). This lead to the following algorithm :

Data: a MDP $(\mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma)$
Result: The optimal policy $\pi*$ and his corresponding value function $V^{\pi*}$
Initialise V(s)=0 $\forall s$;
while **Convergence condition (Value function is stable, or number of epoch)** **do**
  $\Delta = 0$;
  forall $s \in S$ **do**
    $v \leftarrow V(s)$;
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;
    $\Delta \leftarrow \max(\Delta, | v - V(s) |)$;
  end
end
Return $\forall s, \pi*(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;

Algorithm 6: Iterative Policy Evaluation Algorithm

8.2.3 Assynchronous Backup in RL

Definition 5

- *The **Backup** in RL is the fact to update the value of a state using values of futures state states. This can be understood as you check what's going on in the future, and backup this information to present time to update your knowledge.*
- *A **synchronous backup** is a backup made simultaneously for all states (can be done in parallel).*
- *An **assynchronous backup** apply a backup to only one selected state. Those methods can be much more efficient in reducing computation (only deal we required backup) and are still guaranteed to converge.*

For exemple the policy iteration algorithm can be compute both way, but the assynchronous way converge faster.

Prioritised Sweeping

Definition 6

*A **sweep** consists of applying a backup operation to each state. A synchronous backup compute a sweep at every iteration whereas a assynchronous backup may never compute a complete sweep (depending on the strategy).*

*A **prioritised swweping** use a criteria and a priority queue in order to decide which value will be backup next.*

An implementation of a prioritised sweeping can be made using the **Bellman Error**

$$\| v(s) - \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s'} P[s'|s,a] v(s') \right) \|$$

as priority criteria. It requires the knowledge of the **reverse dynamics** $P[s'|s,a]$.

Data: a MDP $(\mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma)$
Result: The optimal policy $\pi*$ and his corresponding value function $V^{\pi*}$
Initialise V(s)=0 and the Bellman Error $B(s), \forall s$;
Initialise the priority queue in ordering the s with $B(s)$;
while **Convergence condition (Value function is stable, or number of epoch)** **do**
  select s the head of the queue $v \leftarrow V(s)$;
  $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$;
  Update $B(s)$ and the priority queue ;
end
Return $\forall s, \pi*(s) = \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$;

Algorithm 7: Prioritised sweeping with Bellman error

Real-time Dynamic Programming The idea here is to use the **agent** experience to guide the selection of states. Each iteration, you backup the state was just visited.

Data: a MDP $(\mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma)$
Result: The optimal policy $\pi*$ and his corresponding value function $V^{\pi*}$
Initialise V(s)=0 $\forall s$;
choose a starting state $s = s_0$;
while **Convergence condition (Value function is stable, or number of epoch)** **do**
  $v \leftarrow V(s)$;
  $V(s) \leftarrow \max_a(\mathcal{R}^a_s + \gamma \sum_{s'} P[s'|s,a]V(s'))$;
  choose the next step from s (greedy policy);
end
Return $\forall s, \pi*(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;

Algorithm 8: Prioritised sweeping with Bellman error

8.2.4 Properties and drawbacks of Dynamic Programming

- Dynamic Programming uses **full-width** backup $\Longrightarrow$ faster convergence but requires lot of ressource
- Require **complete** knowledge of the MDP $\Longrightarrow$ this is not really Machine Learning, but more optisation (all parameters are actually known).
- DP is effective for **medium size problems** (millions of state, backup not to expensive)

8.3 Model-Free Learning

The dynamic programing requires lot of information to be compute, especially what is called the **model**, the states, the actions, the rewards and the transitions. We might want to learn form model which are difficult to observe, and with partial information, etc. That's what we need **model-free learning** algorithms, which can deal with model without knowing in advance the transition and the rewards.

8.3.1 Monte-Carlo Algorithms

Monte-Algorithms are the class of algorithms which learn from episodes of experiences. Which mean basicaly, you play the story fully with what you know, then analyse what append, learn some knowledge from this, and try again. This is different from DP because here you are learning from "real life" experience.

(First Visit) Monte-Carlo Policy Evaluation The first visit MC only take into account the returns from the first visit (i.e. if a state is visited more than once in a trace, it does not count)

Data: a uncomplete MDP ($\mathcal{S}$, $\mathcal{R}$, traces T wich contain series of state and reward, π a policy, and still γ)
Result: The State Value of the MDP for π
Initialise $\hat{V}(s), \forall s$;
Initialise $Returns(s), \forall s$ with empty list **while** **Convergence condition (Value function is stable, or number of epoch)** **do**
  Get a trace τ from T ;
  forall **s in τ** **do**
    R $\leftarrow$ return from the **first appearance** of s in τ ;
    Append R to Returns(s) ;
    $\hat{V}(s) \leftarrow average(Returns(s))$;
  end
end
Return $\hat{V}$;

Algorithm 9: First Visit Monte-Carlo Policy Evaluation

Every Visit Monte-Carlo Policy Evaluation You can have the same algorithm, with taking into account all the visit from a trace.

Data: a uncomplete MDP ($\mathcal{S}$, $\mathcal{R}$, traces T wich contain series of state and reward, π a policy, and still γ)
Result: The State Value of the MDP for π
Initialise $\hat{V}(s), \forall s$;
Initialise $Returns(s), \forall s$ with empty list **while** **Convergence condition (Value function is stable, or number of epoch)** **do**
  Get a trace τ from T ;
  forall **s in τ** **do**
    R $\leftarrow$ returns from **all appearances** of s in τ ;
    Append R to Returns(s) ;
    $\hat{V}(s) \leftarrow average(Returns(s))$;
  end
end
Return $\hat{V}$;

Algorithm 10: Every Visit Monte-Carlo Policy Evaluation

You cannot backup death : if the trace lead to a death state (with no reward during the trace, or infinite negative reward), then the MC will not be able to backup

any sensitive data.

Batch vs Online Monte-Carlo

Incremental Monte-Carlo Update

Runing Mean for Non-Stationnary World

8.3.2 Monte-Carlo Control Algorithms

Monte-Carlo Policy Improvement

Greedy Policy Improvement over State Value Function

Greed Policy Improvement over State-Action Value Function

Exploring Starts Problem

Don't forget Starting to explore

On Policy Soft Control

On-Policy ϵ-greedy first-visit Monte-Carlo control Algorithm

Monte-Carlo Batch Learning to Control

Monte-Carlo Iterative Learning to Control

8.3.3 Temporal Difference Learning

Temporal Difference Value Function Estimation Algorithm

Add Comparison between MC and TD learning

8.3.4 Temporal Difference Learning Control Algorithm

SARSA - On Policy learning Temporal Difference Control: Here is the Sarsa Algorithm :

Data: State $\mathcal{S}$, Action $\mathcal{A}$, Reward $\mathcal{R}$ and Discount γ
Result: The optimal Q(S, A) State-Action Value Function and a greedy policy w.r.t Q
Initialise Q(s, a) $\forall a, s$ with Q(terminal state, a) = 0 ;
while **Convergence condition (number of epoch, $\Delta \leq$ threshold, ...)** **do**
  Initialise a state S;
  Choose action A from S with ϵ-greedy policy derived from Q;
  while **S is not a terminal State** **do**
    Take action A, observe reward R and next state S';
    Choose action A' from S' with ϵ-greedy policy derived from Q;
    Update $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$;
    $S \leftarrow S', A \leftarrow A'$
  end
end
Return Q and π the derived policy

Algorithm 11: SARSA algorithm with ϵ-greedy policy

Theorem 2
Convergence of Sarsa
$Q(s,a) \rightarrow Q^{\infty}(s,a)$ *under :*

- *GLIE (Greedy in the Limite with infinite exploration), which mean every state is visited infinitely many times and that the policy converge toward a greedy-policy (ex: ϵ-greedy with $\epsilon \rightarrow 0$).*
- *Robbins-Monroe sequence of step-sizes α_t : which imply $\sum \alpha_t$ diverge and $\sum \alpha_t^2$ converge.*

Remark 1
the ϵ-greedy policy can be replaced by any policy derived from Q. (Because Q is the one updated by the algorithm)

SARSA-Lambda

Hindsight Experience Replay

Q-Learning: Off-Policy Temporal Difference Learning

8.4 Reinforcement Learning with Function Approximation

8.4.1 Exemple of features

Coarse Coding

Tile Coding

Radial-Basis Function

Deep Learning

8.4.2 Monte-Carlo with Value Function Approximation

8.4.3 Temporal Difference Learning with Value Function Approximation

8.4.4 Q-Learning with FA

8.4.5 subsection name

8.5 Deep Learning Reinforcement Learning

8.5.1 Experience Replay

8.5.2 Target Network

8.5.3 Clipping of Rewards

8.5.4 Skipping of Frames

Chapter 9

Logic-Based Learning

add ref to lecturer

9.1 Inductive Logic Programming (ILP)

9.1.1 Introduction to Concept Learning

In Concept Learning we aim to compute a definition of a concept, expressed in a given langage (called **hypothesis space***) that satisfies positive exemple and none of negatives ones. As an exemple, it can be a regexp which can match all words of a set, and none of an other set (see Regexp Golf)*

Machine Learning Task *The Inductive Logique Programming is a subset of Machine Learning wgere prior knowledge are expressed in declarative language. The task is then a search problem for an hypothesis that would minimise a loss function.*

Given :

- *A language of examples* L_e
- *a language of hypotheses* L_h
- *an unknown target function* $f : L_e \rightarrow Y$
- *a set E of training exemples* $E = \{(e_i, f(e_i))\}$
- *e loss function*

We want to find the hypothesis $h \in L_h$ *that minimises the loss function* ($h = \arg\min_{h_j \in L_h} loss(h_j, E)$). *We want the hypothese h to approximates as much as possible the function h*

Different loss can be choosed, such as $l(h,E) = \frac{1}{|E|}\sum(f(e_i) - h(e_i)$ *or the squared differenced*

Data Mining Task *In a Data Mining Task, the objective is also to discover hypothesis, but the loss is replaced by a quality criterion $Q(h,E)$ such that the search is now to find h such as $Q(h,E) = \frac{|c(h,E)|}{|E|} \geq \epsilon$. This expresses a notion of coverage. This is essential for the concept learning, as we refer generaly to the set of exemple covered by an hypothesis h*

Predictive ILP *Given*

- *A set of observation in L_e with positive examples E_+ and negative examples E_-*
- *Background knowledge B*
- *hypothesis language L_h*
- *a cover relation (notion of covers and rejects)*

we want a the best cover Relation $c(H) = E_+$

When E is noisy, we can add a quality criterion such as before which act as the precision test.

ILP as search of program clauses *The ILP machine learning tasks can be seen as search program. Assuming for now that the representing exemple, the hypothesis use the same logical formalism and that we are interested to learn definite clauses, we can look over all L_h the best arg for our criterions.*
The naives method are obviously none computable in most cases.

Chapter 10

Argumentation Framework

This chapter are notes from the Imperial Course Machine Arguing from Francesca Toni. add ref

introduction *Argument Framework are a field in AI which provide way of evaluate any debate problem. It is useful to resolve conflict, to explain decision or to deal with incomplete information.*

10.1 Abstract Argumentation

10.1.1 Simple AA

Definition 7
an ***AA framework*** *is a set* Args *of arguments and a binary relation* attacks. $(\alpha, \beta) \in$ attacks *means* α *attacks* β.

Semantics in AA *In order to define a "winning" set of argument, we need to provide semantics over the the framework. This is like recipes which determine good set of arguments.*

Definition 8

- ***conflict-free***
- ***admissible:*** *c-f and attacks each attacking argument.*
- ***preferred:*** *maximally admissible.*
- ***complete:*** *admissible + contains each argument it defends.*
- ***stable:*** *c-f + attacks each argument not in it.*
- ***grounded:*** *minimally complete.*

- ***sceptically preferred:*** *Intersection of all prefered.*
- ***ideal:*** *maximal admissible and contain in all prefered (i.e. in the sceptically preferred).*

Definition 9
Semi-stable extension: *complete such as $A \cup A^+$ is maximal. A^+ is the set of attacked argument by A.*

add ref to ASPARTIX and CONARG

10.1.2 Algorithms for AA

Computing Grounded extensions *Use the same algorithms as grounded labelling, but only output the IN arguments as the grounded extensions.*
the grounded extensions in unique.

Computing the grounded labelling: *Here is an algorithm to compute a grounded labelling*

Data: *An AA Framework*
Result: *The grounded Labelling*
Label all unatacked argument with IN ;
while **The IN and the OUT are not stable** **do**
    Label OUT the arguments attacks by IN ;
    Label IN the arguments only attacked by OUT;
end
Label the still unlabelled UNDEC;

Algorithm 12: Computing the grounded labelling

Computing membership in preferred/grounded/ideal extensions; *In order to compute membership, we use Dispute Tree.*

We compute a dispute tree for an argument, and apply the different semantics which are easier to compute on a tree than on a graph.

add algos of computing dispute tree + def of semantics

Data: *An AA Framework, and a arg* α
Result: *Finite or infinite dispute tree corresponding to* α
add a root node Label α *as proponent ;*
while The tree is not stable **do**
  for every $\beta \in$ *proponent, and for every* (γ, β) *add a child-node* γ *as opponent;*
  for every $\beta \in$ *opponent, and for every* (γ, β) *add a node* γ *as proponent in the limit of one child-node per opponent ;*
end

Algorithm 13: Computing dispute Tree

Computing stable extension: *We use answer set programming with logical program.*

10.1.3 AA with Suppport

Bipolar Abstract Argumentation

We add a ***Support*** *relation to a classic AA Framework (*$BAA = \langle Args, Attacks, Supports \rangle$*). There are different semantics :*

Semantics in BAA with deductive support *We can deduce an AA Framework from a BAA with* $Attacks' = Attacks \cup Attacks_{sup} \cup Attacks_{s-med}$ *with*

Definition 10

- $Attacks_{sup}$ *is* ***the supported attacks*** $\implies$ α *attacks every argument that its supports attacks (supports of supports are supports)*
- $Attacks_{s-med}$ *is the super mediated attacks* $\implies$ α *attacks every argument whose supports an argument attacked or* $attacked_{sup}$ *by* α

Then, we apply the AA semantics to $\langle Args, Attacks' \rangle$*. Those semantics are the d-X semantics, where X replace every semantic from AA (grounded, complete, etc.)*

QuAD Framework *We focus here one the QuAD (****Quantitative Argument Debate****) which add a numerical strenght to any argument, and give rule for updating strenght regarding the supporters or attackers.*

Definition 11

- *We define the QuAD Framework as* $\langle \mathcal{A}, \mathcal{C}, \mathcal{P}, \mathcal{R}, \mathcal{BS} \rangle$ *Where* $\mathcal{A}$ *is the set of answer arguments,* $\mathcal{C}$ *is the set of con args,* $\mathcal{P}$ *are the pro args, which are disjoint,* $\mathcal{R}$ *the relation (with* $\mathcal{R}^+$ *the supporters and* $\mathcal{R}^-$ *the attackers) and* $\mathcal{BS}$ *the Base Score, which give score to each args.*
- *The* ***Base funcion*** *is* $f(v_1, v_2) = v_1 + (1 - v_1)v_2$

- *the **aggregation function** as a recursive definition :*

$$\begin{aligned} n = 0 &\implies \mathcal{F}(S) = 0 \\ n = 1 &\implies \mathcal{F}(S) = v_1 \\ n = 2 &\implies \mathcal{F}(S) = f(v_1, v_2) \\ n > 2 &\implies \mathcal{F}(S) = f(\mathcal{F}(S \backslash \{v_n\}, v_n)) \end{aligned}$$

- *the **Combination function** is defined by:*
  $c(v_0, v_a, v_s) = v_0 - v_0 \cdot |v_s - v_a|$ *if* $v_a \geq v_s$
  $c(v_0, v_a, v_s) = v_0 + (1 - v_0) \cdot |v_s - v_a|$ *if* $v_a < v_s$
- *the **Strength Function** is defined as :* $\mathcal{S}(\alpha) = c(\mathcal{BS}(\alpha), \mathcal{F}(SEQ_S(\mathcal{R}^-(\alpha))), \mathcal{F}(SEQ_S(\mathcal{R}^+(\alpha))))$

*The DF-QuAD (for **Discontinuity free**-QuAD) algorithm is simply to apply these semantics to a BAA framework, in setting initial weights to any arguments.*

10.1.4 Argument Mining

10.1.5 AA with Preference

When taking into account preference in a AA, (in the form $X < Y = X$ is less prefered to Y) we can still use the classic AA semantics but we can improved it by using the preferences informations :

- *By **deleting attacks** (an (x, y) attacks succeed iff $x \not\prec y$)*
- *By **reversing attacks** (if (x, y) and $x < y$ then we reverse by replacing this attack by (y, x)).*
- *By **Selecting Amongst extension**, using the preference to select extension with the same extension (ex : differentiate 2 stable extensions). Look at **Rich PAF** for some formal definition.*

10.1.6 AA with Probabilities

10.2 Assumption-Based Argumentation

***Assumption Based Argumentation** introduce arguments as Assumptions $a \in \mathcal{A}$, or deduction σ with premises and conclusions achevied with Rules $R \subseteq \mathcal{R}$(deductive rules, as $q \wedge a \implies p$) defined in a language $\mathcal{L}$*

10.2.1 Simple ABA

*We can define **attacks** in ABA : an argument $A_1 \vdash \sigma_1$ attacks $A_2 \vdash \sigma_2$ iff σ_1 is the contrary of one of the assumptions in A_2.*

10.2.2 ABA more DDs

10.2.3 p-acyclic ABA

Definition 12
*a **positive-acyclic ABA** is a AB framework where the dependancy graph of AB is acyclic.*
*The **dependancy graph** is a graph of all the rules, where assumption ($\mathcal{A}$) are deleted.*

10.3 ArgGame

Chapter 11

Useful Computation

11.1 Data Centering using Matrix Multiplication

$$X - M = X\left(I_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top\right)$$