# MACHINE LEARNING - FORMULAS

by Yannick Giovanakis

September 10, 2018

# Contents

# 1 Linear Regression

- Simple linear regression with D-dimensional input vector **x**

$$y(\boldsymbol{x}, \boldsymbol{b}) = w_0 + \sum_{j=1}^{D-1} w_j x_j = \boldsymbol{w}^T \boldsymbol{x}$$

- Linear regression with M fixed non-linear functions ( **basis functions**)

$$y(\boldsymbol{x}, \boldsymbol{b}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x})$$

  - Polynomial :$\phi_j(x) = x^j$
  - Gaussian : $\phi(x) = e^{-\frac{(x-\mu_j)^2}{2\sigma^2}}$
  - Sigmoid : $\phi(x) = \frac{1}{1+e^{\frac{(\mu_j - x)}{\sigma}}}$

## 1.1 Direct Approach - OLS

- Half of **residual sum of squares** RSS/SSE

$$L(w) = \frac{1}{2} \sum_{n=1}^{N} (y(x_n, \boldsymbol{w}) - t_n)^2$$

$$RSS(\boldsymbol{w}) = ||\epsilon||_2^2 = \sum^{N} \epsilon_i^2$$

- RSS in matrix form and $\epsilon = \boldsymbol{t} - \boldsymbol{\phi w}$

$$L(w) = \frac{1}{2} RSS(\boldsymbol{w}) = \frac{1}{2} (\boldsymbol{t} - \boldsymbol{\phi w})^T (\boldsymbol{t} - \boldsymbol{\phi w})$$

- Minimize LS

$$\frac{\partial L(w)}{\partial w} = -\boldsymbol{\phi}^T (\boldsymbol{t} - \boldsymbol{\phi w}) \quad \frac{\partial^2 L(w)}{\partial w \partial w^T} = \boldsymbol{\phi}^T \boldsymbol{\phi}$$

  - If $\boldsymbol{\phi}^T \boldsymbol{\phi}$ **singular** $\rightarrow$ infinite solutions

– If $\boldsymbol{\phi}^T\boldsymbol{\phi}$ **non - singular** , all **eigenvalues** $\geq 0$ :

$$-\boldsymbol{\phi}^T(\boldsymbol{t}-\boldsymbol{\phi w})=0 \rightarrow \boldsymbol{\phi}^{\boldsymbol{T}}\boldsymbol{\phi w}=\boldsymbol{\phi}^{\boldsymbol{T}}\boldsymbol{t}$$

$$\hat{\boldsymbol{w}}_{OLS}=(\boldsymbol{\phi}^{\boldsymbol{T}}\boldsymbol{\phi})^{-1}\boldsymbol{\phi}^{\boldsymbol{T}}\boldsymbol{t}$$

$$\hat{\boldsymbol{t}}=\boldsymbol{\phi}\hat{\boldsymbol{w}}=\boldsymbol{\phi}(\boldsymbol{\phi}^{\boldsymbol{T}}\boldsymbol{\phi})^{-1}\boldsymbol{\phi}^{\boldsymbol{T}}\boldsymbol{t}$$

$$\underbrace{\boldsymbol{\phi}(\boldsymbol{\phi}^{\boldsymbol{T}}\boldsymbol{\phi})^{-1}\boldsymbol{\phi}^{\boldsymbol{T}}}_{\text{Hat matrix}}$$

## 1.2 Direct Approach - Gradient Optimization

- Weight update with **stochastic gradient descent** and learning rate $\alpha$

$$\boldsymbol{w}^{(k+1)}=\boldsymbol{w}^{(k)}-\alpha^{(k)}\nabla L(X_n)$$

$$\boldsymbol{w}^{(k+1)}=\boldsymbol{w}^{(k)}-\alpha^{(k)}(\boldsymbol{w}^{(k)^T\phi}(x_n)-\boldsymbol{t_n})\boldsymbol{\phi}(x_n)$$

- Condition 1 for convergence $\sum_{k=0}^{\infty}\frac{1}{\alpha^{(k)}}=+\infty$
- Condition 2 for convergence $\sum_{k=0}^{\infty}\frac{1}{\alpha^{(k)^2}}<+\infty$

## 1.3 Discriminative approach - MLE

- Target t given by **deterministic** function y with a **Gaussian noise** $\epsilon \sim \mathcal{N}(0,\sigma^2)$

  $$t=y(\boldsymbol{x},\boldsymbol{w})+\epsilon$$

  So that $t \sim \mathcal{N}(y(x,w),\sigma^2)$

- Given N samples i.i.d and outputs , the **likelihood** is :

$$p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{w},\sigma^2)=\prod_{n=1}^{N}\mathcal{N}(t_n|\boldsymbol{w}^T\boldsymbol{\phi}(x_n),\sigma^2)=\prod_{n=1}^{N}\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(t-y(x,w))^2}{2\sigma^2}}$$

- Use **w** to approximate Gaussian mean using **MLE** (log-likelihood)

$$l(\boldsymbol{w})=lnp(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{w},\sigma^2)=\sum_{n=1}^{N}lnp(t_n|x_n,\boldsymbol{w},\sigma^2)=$$

$$-\frac{N}{2}ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}RSS(w) = -\frac{N}{2}ln(2\pi\sigma^2) - \frac{1}{4\sigma^2}(\boldsymbol{t} - \boldsymbol{w\phi})^T(t - w\phi)$$

$$\nabla l(\boldsymbol{w}) = -\boldsymbol{\phi}^T(\boldsymbol{t} - \boldsymbol{\phi w}) = -\boldsymbol{\phi^T\phi} + \boldsymbol{\phi^T\phi w} = 0$$

$$\boldsymbol{w}_{ML} = (\boldsymbol{\phi^T\phi})^{-1}\boldsymbol{\phi}^T t$$

Assuming Gaussian distribution $\hat{w}_{ML} = \hat{w}_{OLS}$)

- Case of **multiple outputs**

$$\hat{\boldsymbol{W}}_{ML} = (\boldsymbol{\phi^T\phi}^{-1}\boldsymbol{\phi^T T}$$

$$\hat{\boldsymbol{w}}_{ML} = (\boldsymbol{\phi^T\phi}^{-1}\boldsymbol{\phi^T t}_k)$$

## 1.4 Regularization

- Empirical loss on data ($L_D$) + size of parameters ($L_W$):

$$L(w) = L_D(w) + \lambda L_W(w)$$

- **Ridge Regression** with l2-norm

$$L_W(w) = \frac{1}{2}\boldsymbol{w^T w} = \frac{1}{2}||\boldsymbol{w}||_2^2$$

$$L(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}(t_i - \boldsymbol{w}^T\phi(x_i))^2 + \frac{\lambda}{2}||w||_2^2$$

  - Closed form solution: $\hat{\boldsymbol{w}}_{ridge} = (\lambda\boldsymbol{I} + \boldsymbol{\phi^T\phi})^{-1}\phi^T t$

- **Lasso Regression** with l1-norm

$$L_W(w) = \frac{1}{2}\boldsymbol{w} = \frac{1}{2}||\boldsymbol{w}||_1$$

$$L(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}(t_i - \boldsymbol{w}^T\phi(x_i))^2 + \frac{\lambda}{2}||w||_1$$

  - **No** closed-form solution (it's non-linear)

## 1.5 Bayesian Regression

- Posterior distribution $\propto$ likelihood $\cdot$ prior

$$p(\boldsymbol{w}|D) = \frac{p(D|\boldsymbol{w})p(\boldsymbol{w})}{p(D)}$$

$$p(D) = \int p(D|\boldsymbol{w})p(\boldsymbol{w})dw \rightarrow \textbf{ normalizing constant}$$

- Obtain most probable value for $w$ given the data using **Maximum a Posteriori** which is the **mode** of $p(\boldsymbol{w}|D)$

- Prediction of new point $x^*$ given data D (predictive distribution)

$$p(x^*|D) = \int p(x^*|\boldsymbol{w}, D)p(\boldsymbol{w}|D)dw = E[p(x^*|\boldsymbol{w}, D)]$$

- Assuming **Gaussian** likelihood, conjugate prior is also **Gaussian**
  - Prior : $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{w_0}m, S_0)$
  - Posterior : $p(\boldsymbol{w}|\boldsymbol{t}, \boldsymbol{\Phi}, \sigma^2) \propto \mathcal{N}(\boldsymbol{w}|\boldsymbol{w_0}, \boldsymbol{S_0})\mathcal{N}(\boldsymbol{t}|\boldsymbol{\Phi w}, \sigma^2\boldsymbol{I_N})) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{w_N}, \boldsymbol{S_N})$
  - $\boldsymbol{w}_N = \boldsymbol{S}_N \left(\boldsymbol{S}_0^{-1}\boldsymbol{w}_0 + \frac{\boldsymbol{\phi^T t}}{\sigma^2}\right) \rightarrow$ **MAP Estimator**
  - $\boldsymbol{S_N^{-1}} = \boldsymbol{S_0^{-1}} + \frac{\boldsymbol{\Phi^T \Phi}}{\sigma^2}$

- Degeneration to ML if $\boldsymbol{S}_0 \rightarrow \infty$

$$\boldsymbol{S_N^{-1}} = 0 + \frac{\boldsymbol{\Phi^T \Phi}}{\sigma^2}$$

$$\boldsymbol{S_N} = \sigma^2(\boldsymbol{\phi^T \phi})^{-1}$$

$$\boldsymbol{w_N} = \sigma^2(\boldsymbol{\phi^T \phi})^{-1}\frac{\boldsymbol{\Phi^T t}}{\sigma^2} = (\boldsymbol{\phi^T \phi})^{-1}\boldsymbol{\phi^T t}$$

- Degeneration to Ridge if $\boldsymbol{w_0} = 0, \boldsymbol{S_0} = \tau^2\boldsymbol{I}$

$$\lambda = \frac{\sigma^2}{\tau^2}$$

- **Posterior predictive distribution** takes into account all the models

$$p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{D}, \sigma^2) = \int \mathcal{N}(\boldsymbol{t}|\boldsymbol{w}^T\phi(x), \sigma^2)\mathcal{N}(\boldsymbol{w}|\boldsymbol{w_N}, \boldsymbol{S_N})d\boldsymbol{w}$$

$$= \mathcal{N}(t|\boldsymbol{w}_{\boldsymbol{N}}^T\phi(\boldsymbol{x}, \sigma_N^2(\boldsymbol{x})))$$

$$\sigma_N^2(\boldsymbol{x}) = \underbrace{\sigma^2}_{noiseintargetvalues} + \underbrace{\phi(\boldsymbol{x})^T\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x})}_{uncertaintywithparametervalues}$$

$$\text{if } \mathbf{N} \to \infty : \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}) \to 0$$

Variance depends only on $\sigma^2$

# 2 Classification

- Function f can be **non-linear** so non-linear in parameters but linear in **decision surfaces**

$$y(\boldsymbol{x}, \boldsymbol{w}) = f(\boldsymbol{x}^T\boldsymbol{w} + w_0)$$

- Approaches:
  - **Discriminant** : (direct approach) build function that directly maps input to class
  - **Probabilistic discriminative** : model conditional probability $p(C_k|x)$ directly using parametric models
  - **Probabilistic generative** : mode $p(x|C_k)$ , class conditional density , and $p(C_k)$ then use Bayes' Rule

## 2.1 Discriminant approach

### 2.1.1 Considerations

- Two class problem with model $y(x) = \boldsymbol{x}^T\boldsymbol{w} + w_0$
- Assign $C_1$ if $y(x) \geq 0$ , $C_2$ otherwise
- Leads to decision boundary $y(x) = 0$

**Direction of decision boundary**

- Two points on surface $x_A$ , $x_B$

$$y(x_A) = y(x_B) = 0$$

$$\begin{cases} x_A^T \boldsymbol{w} + w_0 = 0 \\ x_B^T \boldsymbol{w} + w_0 = 0 \end{cases}$$

The difference vector identifies the decision boundary and **w** is **orthogonal** to it (because scalar product is 0)

$$(x_A - x_B)\boldsymbol{w} = 0$$

This means that **w** changes the **direction** of the decision boundary.

**Location of decision boundary**

- Point x on decision boundary

$$d(0, y(x) = \frac{|\mathbf{0} \cdot y(x)|}{||\boldsymbol{w}||} = \frac{|\sum 0 \cdot w_i + w_0|}{||\boldsymbol{w}||} = \frac{w_0}{||\boldsymbol{w}||}$$

Which means that the **location** is given by the bias term $w_0$

### 2.1.2 Multi-class problem

- **One-versus-the-rest** : $K - 1$ classifiers (each solves a 2 class problem)
- **One-versus-one** : $\frac{K(K-1)}{2}$ classifiers
- Using **k-linear discriminant** functions: $y_k(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{w}_k + w_{k0}$.
  In this case the decision boundary between k,j is $y_k(x) = y_j(x)$.

### 2.1.3 Least Squares for classification

- General model with 1-of-k encoding for target t

$$y_k = \boldsymbol{x}^T \boldsymbol{w}_k + w_{k0} = \tilde{\boldsymbol{W}}^T \tilde{\boldsymbol{x}}$$

$$\tilde{W} = \begin{bmatrix} w_{0,1} & ... & w_{0,k} \\ ... & ... & ... \\ w_{D,1} & ... & w_{D,k} \end{bmatrix} \quad \tilde{x} = \begin{bmatrix} 1 \\ x_0 \\ ... \\ x_D \end{bmatrix}$$

- Aim : find optimal weight matrix $\tilde{W}$ (same as OLS)

$$\tilde{\boldsymbol{W}} = (\tilde{\boldsymbol{X}}^{\boldsymbol{T}}\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^{T}\boldsymbol{T}$$

Assign input to class for which $t_k = \tilde{\boldsymbol{x}}^T\tilde{\boldsymbol{w}}_k$ is largest

- Method very bad because of **outliers**

### 2.1.4 Perceptron

- On-line algorithm with model

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

$$f = \begin{cases} +1 & a \geq 0 \\ -1 & a < 0 \end{cases}$$

- Finds hyperplane by minimizing distance of **missclassified points** to boundary

$$L_P(x) = -\sum_{n\in M} \boldsymbol{w}^T\phi(x_n)t_n$$

$$M = \text{ set of missclassified points}$$

- Minimizing with **stochastic gradient descent**

$$w^{(k+1)} = w_{(k)} - \alpha\nabla L_P(w) = w^{(k)} + \alpha\phi(x_n)t_n$$

## 2.2 Probabilistic Discriminative Approach

- In **logistic regression** the **posterior probability** of class $C_1$ can be written as **logistic sigmoid function**

$$p(C_1|\phi) = \frac{1}{e^{-\boldsymbol{w}^\phi}} = \sigma(\boldsymbol{w}^\sigma)$$

$$p(C_2|\phi) = 1 - p(C_1|\phi)$$

- Maximize probability of getting right label by using ML

$$y_n = \sigma(\boldsymbol{w}^T \phi_n)$$

$$p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}) = \prod_{n=1}^{N} y_n^{t_n}(1 - y_n)^{1-t_n}$$

$$L(w) = -lnp(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}) = -\sum_{n=1}^{N}(t_n ln(y_n) + (1 - t_n)ln(1 - y_n)) = \sum_{n=1}^{N} L_n$$

- Minimize now $L_n$

$$\frac{\partial L_n}{\partial y_n} = \frac{y_n - t_n}{y_n(1 - t_n)}$$

$$\frac{\partial y_n}{\partial \boldsymbol{w}} = y_n(1 - y_n)\phi_n$$

$$\frac{\partial L_n}{\partial \boldsymbol{w}} = \frac{\partial L_n}{\partial y_n} \cdot \frac{\partial y_n}{\partial \boldsymbol{w}} = (y_n - t_n)\phi_n$$

$$\nabla L(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - t_n)\phi_n$$

### 2.2.1 Multi-class using Softmax

- Softmax transformation of linear functions of feature variables

$$p(C_k|\phi) = y_k(\phi) = \frac{e^{\boldsymbol{w}_k^T \phi}}{\sum_j e^{\boldsymbol{w}_j^T \phi}}$$

- Determine parameters using ML (differently done in **generative approaches**)

$$p(\boldsymbol{T}|\boldsymbol{\Phi}, \boldsymbol{w_1}, ..., \boldsymbol{w_k}) = \prod_{n=1}^{N} \left( \underbrace{\prod_{k=1}^{K} p(C_k|\Phi_n)^{t_{nk}}}_{\text{only one term corresponding to right class}} \right) = \prod_{n=1}^{N} \left( \prod_{k=1}^{K} y_{nk}^{t_{nk}} \right)$$

where $y_{nk} = p(C_k|\phi_n) = \frac{e^{\boldsymbol{w}_k^T \phi}}{\sum_j e^{\boldsymbol{w}_j^T \phi}}$

- Minimize with negative logarithm to get the **cross-entropy** function

$$L(\boldsymbol{w_1}, ..., \boldsymbol{w_k}) = -ln(p(\boldsymbol{T}|\boldsymbol{\Phi}, \boldsymbol{w_1}, ..., \boldsymbol{w_k})) = -\sum_{n=1}^{N}\left(\sum_{k=1}^{k} t_{nk} ln(y_{nk})\right)$$

$$\nabla L_{\boldsymbol{w_j}}(\boldsymbol{w_1}, ..., \boldsymbol{w_k}) = \sum_{n=1}^{N}(y_{nj} - t_{nj})\phi_n$$

# 3 Bias-Variance trade-off and model selection

## 3.1 Free Lunch Theorem

- Generalization accuracy (= on test set) of learner L = $Acc_G(L)$
- All possible concepts $\mathcal{F}$
- True models to be learnt $y = f(x)$
- **No-Free Lunch Theorem**

$$\text{For any learner L: } \frac{1}{|\mathcal{F}|}\sum_{\mathcal{F}} Acc_G(L) = \frac{1}{2}$$

- For learner $L_1, L_2$

$$\text{if } \exists \text{ learning problem so that } Acc_G(L_1) > Acc_G(L_2)$$

$$\text{then } \exists \text{ learning problem } Acc_G(L_2) > Acc_G(L_1)$$

## 3.2 Bias-Variance

- Dataset D with N samples obtained by function $t_i = f(x_i) + \epsilon$

$$E[\epsilon] = 0$$

$$Var[\epsilon] = \sigma^2$$

- Aim is to find model $y(x)$ that approximates f as well as possible. Expected square error on unseen samples x

$$\begin{aligned} E[(t-y(x))^2] &= E[t^2 + y(x)^2 - 2ty(x)] = E[t^2] + E[y(x)^2] - 2E[ty(x)] \\ &= E[t^2] \pm E[t]^2 + E[y(x)^2] \pm E[y(x)]^2 - 2tE[y(x)] \\ &= Var[t] + E[t]^2 + Var[y(x)] + E[y(x)]^2 - 2tE[y(x)] \\ &= Var[t] + Var[y(x)] + (E[t]^2 + E[y(x)]^2 - 2tE[y(x)]) \\ &= \underbrace{Var[t]}_{\sigma^2} + \underbrace{Var[y(x)]}_{Variance} + \underbrace{E[(t-y(x)]^2}_{Bias^2} \end{aligned}$$

- **Bias** is difference between truth and what is expected to be learnt

$$bias^2 = \int (f(\boldsymbol{x}) - E[y(\boldsymbol{x})])^2 p(\boldsymbol{x}) d\boldsymbol{x}$$

  - Decreases with **more complex models**

- **Variance** is difference between what is learnt from a particular dataset and what is expected to be learnt

$$\int E[(y(\boldsymbol{x}) - \bar{y}(\boldsymbol{x}))^2] p(\boldsymbol{x}) d\boldsymbol{x}$$

$$\bar{y}(\boldsymbol{x}) = E[y(\bar{x})]$$

  - Decreases with **simpler models**
  - Decreases with **more samples**

- Bias-Variance can be calculated analytically in KNN

$$E[(t-y(x))^2] = \sigma^2 + \frac{\sigma^2}{K} + \left( f(\boldsymbol{x}) - \frac{1}{K} \sum_{i=1}^{K} f(\boldsymbol{x}_i) \right)^2$$

## 3.3 Train-Test Errors

- **Training error** $\rightarrow$ optimistically biased estimate of prediction

  - Regression $L_{train} = \frac{1}{N} \sum_{n=1}^{N} (t_n - y(\boldsymbol{x}_n))^2$

- Classification $L_{train} = \frac{1}{N} \sum_{n=1}^{N} (I(t_n \neq y(\boldsymbol{x}_n)))$

- **Prediction error**
  - Regression : $L_{true} = \int (f(\boldsymbol{x}) - y(\boldsymbol{x}))^2 p(\boldsymbol{x}) d\boldsymbol{x}$
  - Classification : $L_{true} = \int I(f(\boldsymbol{x}) \neq y(\boldsymbol{x})) p(\boldsymbol{x}) d\boldsymbol{x}$

- **Test error** $\rightarrow$ unbiased if not used during training

$$L_{train} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} (t_n - y(\boldsymbol{x}_n))^2$$

Different estimation methods for test error (never use **test data!**):
Direct approach

- **Leave-One-Out Cross Validation** : $D - \{n\}$ n-th data point moved to validation set (almost unbiased, slightly pessimistic)

$$L_{LOO} = \frac{1}{N} \sum_{n=1}^{N} (t_n - y_{D-\{n\}}(\boldsymbol{x}_n))^2$$

- **K-Fold Cross Validation** : divide training data into k equal parts $D_1, ..., D_k$ and train on $D - D_i$ (faster but more pessimistically biased)

$$L_{D_i} = \frac{k}{N} \sum_{(x_n, t_n) \in D_i} (t_n - y_{D-D_i}(\boldsymbol{x}))^2$$

$$L_{k-fold} = \frac{1}{k} \sum_{i=1}^{k} L_{D_i}$$

Adjustment techniques on training error to take into account model complexity:

- $\boldsymbol{C_p}$ : d total number of parameters, $(\tilde{\sigma})^2$ estimate of variance of $\epsilon$

$$C_p = \frac{1}{N}(RSS + 2d\tilde{\sigma}^2)$$

- **AIC** : L maximized value for likelihood of model

$$AIC = -2logL + 2d$$

- **BIC** : since $logN > 2$ for any $n > 7$ , BIC selects smaller models

$$BIC = \frac{1}{N}(RSS + log(N)d\tilde{\sigma}^2)$$

- **Adjusted R2** : large values indicate small test error

$$AdjR^2 = 1 - \frac{RSS(N-1)}{TSS(N-d-1)}$$

## 3.4 Model selection

- **Feature selection**

  1. $M_0$ null model ,no features,predicts mean
  2. $k = 1, ...M$ fit all $\binom{M}{k}$ model with k features
  3. Pick best among these $\binom{M}{k}$ called $M_k$
  4. Selected single best with CV,AIC,BIC...

  Bad if M too large (overfitting + computational cost)

  - Filter method = **PCA**, **SVD**...
  - Embedded = **Decision Tree,Lasso**...
  - Wrapper = GA + Algorithm to find weights with **forward** or **backward** step-wise selection

- **Regularization** : Ridge ,Lasso

- **Dimension Reduction** : unsupervised, transforms original features.
  **PCA** : project on the input subspace that account for most of the variance (repeat for m lines)

  1. Mean center data : $\bar{\boldsymbol{x}} = \frac{1}{N}\sum \boldsymbol{x}_n$
  2. Compute covariance matrix **S**
  3. Compute eigenvalues/eigenvectors of **S** :

$$S = \frac{1}{N-1}\sum_{n=1}^{N}(\boldsymbol{x_n} - \bar{\boldsymbol{x}})(\boldsymbol{x_n} - \bar{\boldsymbol{x}})^T$$

  4. Eigenvector $e_1$ with largest eigenvalue $\lambda_1$ is first principal component , $\frac{\lambda_k}{\sum\limits_i \lambda_i}$ is portion of **explained variance**

## 3.5 Model Ensembles

- **Bagging** : reduces variance,without increasing bias

$$Var(\bar{x}) = \frac{Var(x)}{N}$$

Multiple models → More training sets : **Bootstrap aggregation**, random sampling with replacement. Only good with **unstable learners** (bad when there is high bias).

- **Boosting** : sequentially learn weak classifiers ( start from equal weights then penalize samples with misspredictions)

# 4 PAC-Learning and VC Dimensions

## 4.1 Intro and Version Space

- Set of instances X
- Set of hypothesis H (finite)
- Set of possible target concepts C
- Training instances generated by a fixed,unknown probability distribution P over **X**.

> The learner observers a sequence D of training samples $\langle x, c(x)\rangle$ for some target concept $c \in C$ and it must output a hypothesis h estimating c, which is evaluated by its performance on subsequent instances drawn from P
>
> $$L_{true} = Pr_{x \in P}[c(x) \neq h(x)]$$

- $L_{true}$ cannot be measured as P is unknown so aim is to **bound** $L_{train}$ given $L_{true}$
- Given a **Version Space** ( subset of hypotheses in H consistent with training data : $L_{train} = 0$) , can $L_{true}$ also be bounded to be in the VS?

If the hypothesis space H is finite and D is a sequence of N $\geq$ 1 independent random samples of some target concept c, the for any $0 \leq \epsilon \leq 1$, the probability that $VS_{H,D}$ ,contains an hypothesis error greater than $\epsilon$ is less than $|H|e^{-\epsilon N}$

$$Pr(\exists h \in H : L_{train} = 0 \wedge L_{true} \geq \epsilon) \leq |H|e^{-\epsilon N}$$

Proof:

$$\begin{aligned}
&Pr((L_{train}(h_1) = 0 \wedge L_{true}(h_1) \geq \epsilon) \vee ... \vee (L_{train}(h_{|H|}) = 0 \wedge L_{true}(h_{|H|}) \geq \epsilon)) \\
&\quad \leq \sum_{h \in H} Pr(L_{train}(h) = 0 \wedge L_{true}(h) \geq \epsilon) \\
&\quad \leq \sum_{h \in H} Pr(L_{train}(h) = 0 | L_{true}(h) \geq \epsilon) \\
&\quad \leq \sum_{h \in H} (1-\epsilon)^N \\
&\quad \leq |H|(1-\epsilon)^N \\
&\quad \leq |H|e^{-\epsilon N}
\end{aligned}$$

- That probability should be at most $\delta$

$$|H|e^{-\epsilon N} \leq \delta$$

- N can be computed as

$$N \geq \frac{1}{\epsilon}(ln|H| + ln(\frac{1}{\delta})$$

- $\epsilon$ can be computed as

$$\epsilon \geq \frac{1}{N}(ln|H| + ln(\frac{1}{\delta})$$

N.B. $ln|H|$ is still exponential as $|H| = 2^{2^M}$

## 4.2 PAC-Learning

- **PAC Learning**

> Given a class C of possible target concepts defined over a set of instances X of length n , and a Learner L using hypothesis space H. C is **PAC- LEARNABLE** if there exists an algorithm L such that for every $f \in C$ for any distribution P , for any $\epsilon$ such that $0 \leq \epsilon \leq \frac{1}{2}$ and $\delta$ such that $0 \leq \delta \leq \frac{1}{2}$ , algorithm L with probability at least $1 - \delta$ outputs a concept h such that $L_{true} \leq \epsilon$ using a number of samples that is polynomial of $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$

- **Efficient PAC-Learnable**

> C is **efficiently PAC- LEARNABLE** if there exists an algorithm L such that for every $f \in C$ for any distribution P , for any $\epsilon$ such that $0 \leq \epsilon \leq \frac{1}{2}$ and $\delta$ such that $0 \leq \delta \leq \frac{1}{2}$ , algorithm L with probability at least $1 - \delta$ outputs a concept h such that $L_{true} \leq \epsilon$ using a number of samples that is polynomial of $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$ , M and size(c).

- **Agnostic Learning** : empty VS ( no $L_{train} = 0$ )

$$L_{true}(h) \leq L_{train} + \epsilon$$

Hoeffding bound: $Pr(E[\bar{X}] - \bar{X} > \epsilon) < e^{-2n\epsilon^2}$

> Hypothesis space H finite, dataset D with i.i.d samples ,$0 \leq \epsilon \leq 1$ , for any learned hypothesis h :
>
> $$Pr(L_{true}(h) - L_{train}(h) > \epsilon) \leq |H| e^{-2N\epsilon^2}$$

- **PAC-Bound vs B/V Trade-off**

$$L_{true}(h) \leq \underbrace{L_{train}(h)}_{Bias} + \underbrace{\sqrt{\frac{ln|H| + ln\frac{1}{\delta}}{2N}}}_{Variance}$$

Large $|H|$: **low bias** , **high variance**
Small $|H|$: **high bias** , **low variance** (tighter bound)

## 4.3 VC Dimension

- For $|H| \to \infty$ PAC-Learning bound cannot be used but number of points N required can be found using an alternative method ( "Axis -aligned rectangles" )

$$N \geq \left(\frac{4}{\epsilon}\right) ln \left(\frac{4}{\delta}\right)$$

The best way for $|H| \to \infty$ is to bound the error as a function of the number of points that can be completely labeled

- **Dichotomy**

| A **dichotomy** of a set S is a partition of S into two disjoint subsets. |
|---|

- **Shattering**

| A set of instances S is **shattered** by a hypothesis space H if and only if for every dichotomy of S there exists some hypothesis in H **consistent** (= classify correctly) with this dichotomy. |
|---|

- **VC-Dimension**

| The VC dimension, VC(H) , of an hypothesis space H defined over instance space X is the size of the **largest finite subset** shattered by H. If arbitrarily large finite sets of X can be shattered by H , then VC(H) = $\infty$ |
|---|

Rule of thumb : number of parameters = max number of points

- Number of samples to guarantee an error of at most $\epsilon$ with probability at least $(1 - \delta)$

$$N \geq \frac{1}{\epsilon}\left(4log_2\left(\frac{2}{\delta}\right) + 8VC(H)log_2\left(\frac{13}{\epsilon}\right)\right)$$

- PAC Bound and VC Dimension

$$L_{true} \leq L_{train} + \sqrt{\frac{VC(H)\left(ln\frac{2N}{VC(H)} + 1\right) + ln\frac{4}{\delta}}{N}}$$

Choose hypothesis space H so to minimize the bound on expected true error

- VC-Dimension properties

| The VC Dimension of a space $\|H\| \leq \infty$ is bounded from above<br>$$VC(H) \leq log_2(\|H\|)$$<br>If $\|H\| = d$ then there are at least $2^d$ functions in H. Since there are at least $2^d$ labelings : $\|H\| \geq 2^d$ |
|---|

| A concept class C with $VC(C) = \infty$ is **not** PAC -Learnable |
|---|

# 5 Kernel Methods

- Make linear models work in non-linear settings by mapping data into higher dimensions where it exhibits linear patterns. Mapping changes feature representation, which can be expensive but made "affordable" with the **kernel trick**.Can be used if algorithm used scalar product than can be replaced by kernel function.

$$\text{Mapping } \phi : \boldsymbol{x} \rightarrow \{x_1^2, x_m^2, x_1x_2, ..., x_1x_M, ..., x_{M-1}x_M\}$$

Inefficient and many features (can blow up)

Solution : dual problem with kernels

$$k(x, x') = \phi(\boldsymbol{x})^T\phi(\boldsymbol{x'})$$

$\phi$ fixed non-linear feature space mapping (**basis function**)

- $k(\boldsymbol{x}, \boldsymbol{x'})$ = similarity between **x ,x'**
- Simplest kernel = linear kernel

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \boldsymbol{x^T x'}$$

- Kernel is symmetric in its arguments

$$k(\boldsymbol{x}, \boldsymbol{x'}) = k(\boldsymbol{x'}, \boldsymbol{x})$$

- **Stationary kernels**

$$k(\boldsymbol{x}, \boldsymbol{x'}) = k(\boldsymbol{x} - \boldsymbol{x'})$$

- **Homogeneous kernels**

$$k(\boldsymbol{x}, \boldsymbol{x'}) = k(||\boldsymbol{x} - \boldsymbol{x'}||)$$

## 5.1 Dual representation

– Linear regression model with L2 regularized RSS

$$L(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{w}^T\phi(x_n) - t_n)^2 + \frac{\lambda}{2}\boldsymbol{w^T w}$$

$$\boldsymbol{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}(\boldsymbol{w^T}\phi(\boldsymbol{x_n} - t_n)\phi(\boldsymbol{x}_n) = \sum_{n=1}^{N} a_n\phi(\boldsymbol{x_n}) = \boldsymbol{\Phi^T a}$$

– Design matrix $\boldsymbol{\Phi}$ has $n^{th}$ row $= \phi(\boldsymbol{x_n})^T$

– Coefficients $\boldsymbol{a_n}$ are functions of $\mathbf{w} = -\frac{1}{\lambda}(\boldsymbol{w^T\phi(x_n)} - t_n)$

– **Gram matrix $\boldsymbol{K = \Phi^T\Phi}$** , $NxN$ matrix:

$$K_{nm} = \phi(x_n)^T\phi(x_m) = k(\boldsymbol{x_n, x_m})$$

$$K = \begin{bmatrix} k(\boldsymbol{x_1, x_1}) & ... & k(\boldsymbol{x_1, x_N}) \\ ... & ... & ... \\ k(\boldsymbol{x_N, x_1}) & ... & k(\boldsymbol{x_N, x_N}) \end{bmatrix}$$

– Error function in terms of Gram Matrix of Kernel

$$L(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{a^T\Phi\Phi^T\Phi\Phi^T a} - \boldsymbol{a^T\Phi\Phi^T t} + \frac{1}{2}\boldsymbol{t^T t} + \frac{\lambda}{2}\boldsymbol{a^T\Phi\Phi^T a}$$

$$L_a = \frac{1}{2}\boldsymbol{a^T}KK\boldsymbol{a} - \boldsymbol{a}^TK\boldsymbol{t} + \frac{1}{2}\boldsymbol{t^T t} + \frac{\lambda}{2}\boldsymbol{a^T}K\boldsymbol{a}$$

Solving for a using $\boldsymbol{w = \Phi^T a}$ and $a = -\frac{1}{\lambda}(\boldsymbol{w^T}\phi(x_n) - t_n)$

$$\boldsymbol{a} = (K + \lambda\boldsymbol{I}_N)^{-1}\boldsymbol{t}$$

Solution for **a** can be expressed as linear combination of elements of $\phi(x)$ whose coefficients depends entirely in terms of $k(x, x')$ from which the original formulation of w can be recovered.

– **Prediction** for new input **x**

$$y(\boldsymbol{x}) = \boldsymbol{w}^T\phi(\boldsymbol{x}) = \boldsymbol{a}^T\Phi\phi(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^T(K + \lambda \boldsymbol{I}_N)^{-1}\boldsymbol{t}$$

$$\text{where } \boldsymbol{k(x)} = k(\boldsymbol{x_n}, \boldsymbol{x})$$

- Advantages
  - avoid working with feature vector $\phi(x)$
  - build a feature space with high dimensionality
  - leverage possibility to use not only vectors of real numbers but also over objects (sets,logic formulas...)
- Disadvantage
  - Invert big $NxN$ matrix instead of (possibly) smaller $MxM$ matrix

## 5.2 Constructing Kernels

- 1. Method :Choose feature space mapping $\phi(x)$ and use it to find kernel.Example with 1-D input space:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \phi(\boldsymbol{x})^T\phi(\boldsymbol{x'}) = \sum_{i=1}^{N} \phi_i(\boldsymbol{x})\phi_i(\boldsymbol{x'})$$

- 2.Method : Construct kernels directly , making sure it is valid ( = corresponds to some **scalar product** in some feature space). Example $k(x, z) = (x^T z)^2$ in 2-D space:

$$\begin{aligned} k(x, z) = (x^T z)^2 &= (x_1 z_1 + x_2 z_2)^2 = x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= (x_1^2, \sqrt{2}x_1 x_2, x_2^2)(z_1^2, \sqrt{2}z_1 z_2, z_2^2)^T = \phi(\boldsymbol{x})^T\phi(\boldsymbol{z}) \end{aligned}$$

- Necessary and sufficient condition for a function $k(\boldsymbol{x}, \boldsymbol{x'})$ to be a kernel is that the Gram matrix K is **semi-definite** for all possible choices of the set $\{x_n\}$:

$$\boldsymbol{x}^T K \boldsymbol{x} \geq 0 \text{ for non-zero vectors } \boldsymbol{x}$$

$$\sum_n \sum_m K_{n,m} \boldsymbol{x_n} \boldsymbol{x_m}$$

Mercer's theorem : Any continuous, symmetric,positive semi-definite (= no negative eigenvalues) kernel function K(x,y) can be expressed as a **dot product** in a high-dimensional space

## 5.3 List of valid kernels

Given kernels $k_1(\boldsymbol{x}, \boldsymbol{x'}), k_2\boldsymbol{x}, \boldsymbol{x'}$

1. $k(\boldsymbol{x}, \boldsymbol{x'}) = ck_1(\boldsymbol{x}, \boldsymbol{x'})$
2. $k(\boldsymbol{x}, \boldsymbol{x'}) = f(\boldsymbol{x})k_1(\boldsymbol{x}, \boldsymbol{x'})f(\boldsymbol{x})$
3. $k(\boldsymbol{x}, \boldsymbol{x'}) = q(k_1(\boldsymbol{x}, \boldsymbol{x'}))$, q() polynomial with non negative coefficients
4. $k(\boldsymbol{x}, \boldsymbol{x'}) = exp(k_1(\boldsymbol{x}, \boldsymbol{x'}))$
5. $k(\boldsymbol{x}, \boldsymbol{x'}) = k_1(\boldsymbol{x}, \boldsymbol{x'}) + k_2(\boldsymbol{x}, \boldsymbol{x'})$
6. $k(\boldsymbol{x}, \boldsymbol{x'}) = k_1(\boldsymbol{x}, \boldsymbol{x'}) \cdot k_2(\boldsymbol{x}, \boldsymbol{x'})$
7. $k(\boldsymbol{x}, \boldsymbol{x'}) = k_3(\boldsymbol{\phi(x)}, \boldsymbol{\phi(x')})$
8. $k(\boldsymbol{x}, \boldsymbol{x'}) = \boldsymbol{x}^T A\boldsymbol{x'}$ A symmetric positive semi-definite matrix
9. $k(\boldsymbol{x}, \boldsymbol{x'}) = k_a(\boldsymbol{x_a}, \boldsymbol{x'_a})+k_b(\boldsymbol{x_b}, \boldsymbol{x'_b})$ where xa,xb are variables with x=(xa,xb)
10. $k(\boldsymbol{x}, \boldsymbol{x'}) = k_a(\boldsymbol{x_a}, \boldsymbol{x'_a}) \cdot k_b(\boldsymbol{x_b}, \boldsymbol{x'_b})$

- Common kernel : **Gaussian**

$$k(\boldsymbol{x}, \boldsymbol{x'}) = e^{-\frac{||\boldsymbol{x}-\boldsymbol{x'}||^2}{2\sigma^2}}$$

$$\text{Valid : } ||x - x'||^2 = (x - x')^T(x - x') = x^T + x'^T x' - 2x^T x'$$

$$k(x, x') = e^{-\frac{x^T x}{2\sigma^2}} e^{-\frac{x^T x'}{\sigma^2}} e^{-\frac{x'^T x'}{2\sigma^2}}$$

Valid by combining rules 2 and 4 + linear kernel

# 6 SVM

SVM use :

- Subset of training samples (**support vectors**)
- Vector of weights **a**
- Similarity function $K(x, x')$ (**kernel**)
- Class prediction for new sample $x_q (t_i \in \{-1, 1\})$

$$f(x_q) = sign\left(\sum_{m \in S} \alpha_m t_m k(x_q, x_m) + b\right)$$

where

  - S is the set of indices of **support vectors**

- Can be seen as generalization of perceptron $f(x_q) = sign\left(\sum_{j=1}^{M} w_j \phi_j(x_q)\right)$

Where

$$w_j = \sum_{n=1}^{N} \alpha_n t_n \phi_j(x_n)$$

$$f(x_q) = sign\left(\sum_{j=1}^{M}\left(\sum_{n=1}^{N} \alpha_n t_n \phi_j(x_n)\right)\phi_j(x_q)\right)$$

$$f(x_q) = \sum_{n=1}^{N} \alpha_n t_n \left(\phi(x_n)\phi(x_q)\right)$$

- Get a much powerful learner by replacing dot product with **similarity function** ( kernel matrix)

## 6.1 Learning in SVM

- **Margin** : smallest distance between separating hyperplane and any of the samples. Aim is to **maximize margin** = maximize distance of the closest points.

- Assuming a separating hyperplane exists:

$$y(x_n) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x_n}) + b$$

$$t_n y(x_n) > 0 \quad \forall n$$

$$\text{Distance of point to surface} : \frac{t_n y(x_n)}{||\boldsymbol{w}||} = \frac{t_n(\boldsymbol{w}^T \phi(x_n) + b)}{||\boldsymbol{w}||}$$

- **Weigh optimization problem**

$$\text{margin} = min_n t_n(\boldsymbol{w}^T \phi(\boldsymbol{x_n}) + b)$$

$$\boldsymbol{w^*} = argmax_{\boldsymbol{w},b} \left( \frac{1}{||w||_2} min_n(t_n(\boldsymbol{w}^T \phi(x_n) + b) \right)$$

- Problem hard to solve. So fix margin (it's **scale invariant**) to be 1:

$$t_n(\boldsymbol{w}^T \phi(\boldsymbol{x_n}) + b) = 1$$

- Now **minimize weights**
  - **Minimize**: $\frac{1}{2}||\boldsymbol{w}||_2^2$ (Equivalent to maximize $\frac{1}{||\boldsymbol{w}||}$)
  - **Subject to**: $t_n(\boldsymbol{w}^T \phi(\boldsymbol{x_n}) + b) \geq 1, \quad \forall n$

### 6.1.1 Constraint optimization recap

- **Minimize** $f(\boldsymbol{w}) \rightarrow$ quadratic
- **Subject to** $h_i(\boldsymbol{w}) = 0, for = 1, 2, ... \rightarrow$ quadratic
- $\boldsymbol{w}^*, \nabla f(\boldsymbol{w}^*)$ solution must lie in subspace spanned by

$$\{\nabla h_i(\boldsymbol{w}^* : i = 1, 2...)\}$$

- **Lagrangian function**

$$L(\boldsymbol{w}, \lambda) = f(\boldsymbol{w}) + \sum_i \lambda_i h_i(\boldsymbol{w})$$

- **Lagrangian multipliers** $\lambda_i$ , to solve

$$\nabla L(\boldsymbol{w}^*, \boldsymbol{\lambda}^*) = 0$$

- With **inequality constraints**
  - **Minimze** $f(\boldsymbol{w})$
  - **Subject to** $g_i(\boldsymbol{w}) \leq 0 \quad i = 1, 2...$
  - **Subject to** $h_i(\boldsymbol{w}) = 0 \quad i = 1, 2..$
- Lagrange multipliers for **equality** are $\lambda_i$ and for **inequality** are $\alpha_i$
- **KKT** conditions

$$\lambda(\boldsymbol{w^*}, \boldsymbol{\alpha^*}, \boldsymbol{\lambda^*}) = 0$$

$$h_i(\boldsymbol{w^*}) = 0$$

$$g_i(\boldsymbol{w^*}) \leq 0$$

$$\alpha_i^* \geq 0$$

$$\alpha_i^* g_i(\boldsymbol{w}) = 0$$

- Constraints are either active $g_i(\boldsymbol{w^*}) = 0$ (**IT'S A SUPPORT VECTOR**) or its multiplier is zero $\alpha_i^* = 0$

### 6.1.2 Dual and Primal Problem

- Primal : weights over **features**
- Dual : weights over **instances** $\rightarrow$ easier and has more weights=0
- **Lagrangian function**

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\boldsymbol{w}||_2^2 - \sum_{n=1}^{N} \alpha_n (t_n(\boldsymbol{w}^T \phi(\boldsymbol{x_n}) + b) - 1)$$

- Gradient for w,b:

$$\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n t_n \phi(x_n)$$

$$0 = \sum_{n=1}^{N} \alpha_n t_n$$

- Rewrite Lagrangian

- **Maximize:** $\tilde{L}(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m k(\boldsymbol{x_n}, \boldsymbol{x_m})$
- **Subject to:** $\alpha_n \geq 0 \quad \forall n$
- **Subject to:** $\sum_{n=1}^{N} \alpha_n t_n = 0$

## 6.2 Prediction

$$y(x) = sign\left(\sum_{n=1}^{N} \alpha_n t_n k(x, x_n) + b\right)$$

$$b = \frac{1}{N_S} \sum_{n \in S} \left(t_n - \sum_{m \in S} \alpha_m t_m k(\boldsymbol{x_n}, \boldsymbol{x_m})\right)$$

- As number of dimension increases also the number of **support vectors** increases. Ideally $N_S << N$ , otherwise overfit

## 6.3 Solution Techniques

- **Sequential minimal optimization** :
  1. Find sample $x_i$ that violates KKT
  2. Select second sample heuristically
  3. Joint optimize $\alpha_i, \alpha_j$

## 6.4 Noisy data and Slack Variables

With noisy data missclassification should be allowed using a **relaxation** of the problem.

- Slack variable $\xi_i$ that allow to violate the constraints, but a **penalty cost** C
  - **Minimize:** $||w||_2^2 + C \sum_i \xi_i$
  - **Subject to:** $t_i(\boldsymbol{w}^T x_i + b) \geq 1 - \xi_i$
  - **Subject to:** $\xi_i > 0$
- **C** allows to trade-off **bias** and **variance** (CV to find right C)

- **High C** : complex solution
- **Low C** : simpler solution

- Dual with Slack
  - **Maximize:** $\tilde{L}(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m k(\boldsymbol{x_n}, \boldsymbol{x_m})$
  - **Subject to:** $0 \leq \alpha_n \leq C \quad \forall n$
  - **Subject to:** $\sum_{n=1}^{N} \alpha_n t_n = 0$
  - **Support vectors** : $\alpha_n > 0$
  - **Point on margin**: $\alpha_n < C$
  - **Lies inside margin** : $\alpha_n = C$ , is correctly classified $\xi_i \leq 1$ or **misclassified** if $\xi_i > 1$

## 6.5 Bounds

- Bound on VC dimension **decreases with margin** : **large margin** = **low VC Dimension** = **low variance**
- Margin bound is very pessimistic
- Use **Leave-one-out Bound**, can be easily computed

$$L_h \leq \frac{E[\text{Number of support vectors}]}{N}$$

Remove 1 samples at the time, if you remove a non-support vector , margin does not change. If you remove support vector margin changes!

# 7 Reinforcement Learning

- **Agent** : learns and makes decision, acting and observing the environment. At each step t ,the agent can:
  - Perform action $a_t$
  - Receive observation $o_t$

- Receive scalar reward $r_t$

- **The environment** : interacts with the agent (anything that the agent can control). At each step t,it can :
  - Receive action $a_t$
  - Emit observation $o_t$
  - Emit scalar reward $r_t$

- The **History** is a sequence of actions,observations and rewards:

$$h_t = a_1, o_1, r_1, ..., a_t, o_t, r_t$$

  - It influences the next action to be chosen by the agent and which observation /reward the environment will emit.

- The **state** is the information used to determine what happens next . It is a function of the history

$$s_t^a = f(a_1, o_1, r_1, ..., a_t, o_t, r_t)$$

- The state of the environment is the private internal representation of the environment. It is usually **not visible** to the agent ( it is in some card games for example)

- If environment is **fully observable**

$$o_t = s_t^a = s_t^e$$

- **Reinforcement Learning** is useful when:
  - Dynamics of environment are **unknown** or **difficult to model**
  - The model of the environment is **too complex** to be solved exactly, so approximate solutions are needed.

## 7.1 Markov Decision Process

"Future is independent on the past given the present"

- A **stochastic process** $X_t$ is **Markovian** if and only if

$$P(X_{t+1} = j|X_t = i, X_{t-1} = k_{t-1}, ..., X_1 = k_1, X_0 = k_0) = P(X_{t+1} = j|X_t = i)$$

- Means that the **current** state is a sufficient statistic to calculate the probability of the next value ( no past is needed).
- If transition probabilities are **time invariant**

$$p_{ij} = P(X_{t+1} = j|X_t = i) = P(X_t = j|X_0 = i)$$

**Markov Decision Process**

- An MDP is Markov reward process with **decisions**.It models an environment in which all states are Markov and time is divided into **stages**.

$$\langle S, A, P, R, \gamma, \mu \rangle$$

  - **S** : set of states (finite)
  - **A** : set of actions (finite). Can be function of state
  - **P** : state-transition probability matrix $P(s'|s, a)$ of size $(|S| + |A|)x|S|$ $P(s'|s, a)$
  - **R** : reward function $R(s, a) = E[r|s, a]$
  - $\boldsymbol{\gamma}$: discount factor $\in [0, 1]$
    How much the reward will lose in 1 time-step
    - $\gamma \to 0$ **myopic evaluation**
    - $\gamma \to 1$ **far-sighted evaluation**
  - $\boldsymbol{\mu}$: set of initial probabilities $\mu_i^0 = P(X_0 = i)$

### 7.1.1 Reward and Goals

- **Sutton Hypothesis**: goals and purposes can be thought of as the **maximization** of the cumulative sum of a received scalar reward
- Goal can be defined by infinite **different reward functions** :but it must always be outside the agent's control which can simply measure success step by step (**explicitly** and **frequently**).
- Time horizons can be:
  - **finite**: horizon reduces at each step so at every time there is a different optimization problem
  - **indefinite**: until some stopping criteria is met , like **absorbing states** (e.g.: Blackjack)
  - **infinite** : ideally infinite (e.g.: Pole balancing)
- **Cumulative rewards** can be:
  - **total reward** : $V = \sum_{i=1}^{\infty} r_i$
  - **average reward**: $V = \lim_{n\to\infty} \frac{r_1+...+r_n}{n}$
  - **discounted reward:** $V = \sum_{i=1}^{\infty} \gamma^{i-1} r_i$

**Infinite time-horizon discounted return**

- $v_t = r_{t+1} + \gamma r_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$

### 7.1.2 Policies

**Policies**

- **Policy** : mapping from states to probabilities of selecting each possible action. Decides which action an agent selects.
- Policies can be :
  - **Markovian** $\subset$ History-dependent
  - **Deterministic** $\subset$ Stochastic

- **Stationary** $\subset$ Non-stationary

- Policy $\pi$ is a distribution over actions given a state

$$\pi(a|s) = P[a|s]$$

- MDP policies depend on the current state (stationary property).
  Given an MDP $M$ and a policy $\pi$
  - State sequence $s_1, s_2...$ is a **Markov** process $\langle S, P^\pi, \mu \rangle$
  - The state and reward sequence $s_1, r_2, s_2, ...$ is a **Markov reward process** ( Markov Chain ) $\langle S^\pi, P^\pi, R^\pi, \gamma, \mu \rangle$

$$P^\pi = \sum_{a \in A} \pi(a|s) P(s'|s, a)$$

$$R^\pi = \sum_{a \in A} \pi(a|s) P(s'|s, a)$$

### 7.1.3 Value Function

- Given a policy $\pi$ it is possible to define the utility of **each state** using **Policy Evaluation**

- The **state-value function** $V^\pi(s)$ of an MDP is the expected return starting from state s and then following policy $\pi$

$$V^\pi = E_\pi[v_t | s_t = s]$$

- For control purposes it is better to define the value of each action in each state using**action-value function** $Q^\pi(s, a)$

$$Q^\pi(s, a) = E_\pi[v_t | s_t = s, a_t = a]$$

### 7.1.4 Bellman Equations

- **Bellman equation** for decomposed state-value function (immediate reward + discounted value of successor state )

$$\begin{aligned} V^\pi(s) &= E_\pi[r_{t+1} + \gamma V^\pi(s_{t+1} | s_t = s] \\ &= \sum_{a \in A} \pi(a|s) \left( R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^\pi(s') \right) \end{aligned}$$

- **Bellman equation** for decomposed state-action function

$$\begin{aligned} Q^\pi(s,a) &= E_\pi[r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1} | s_t = s, a_t = a] \\ &= R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^\pi(s') \\ &= R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \sum_{a' \in A} \pi(a'|s') Q^\pi(a', s') \end{aligned}$$

- In **matrix form** using induced MRP (N.B.: max eigenvalue $\gamma P^\pi = 1 \cdot \gamma = \gamma$)

$$V^\pi = R^\pi + \lambda P^\pi V^\pi$$

$$V^\pi = (I - \gamma P^\pi)^{-1} R^\pi$$

  - $\gamma \leq 1$ matrix is **not singular**
  - $\gamma = 1$ matrix is **singular**

### 7.1.5 Bellman Operators

- Bellman operator for $V^\pi$ is defined as $T^\pi : R^{|S|} x R^{|S|}$ (maps value functions to value functions)

$$(T^\pi V^\pi)(s) = \sum_{a \in A} \pi(a|s) \left( R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^\pi(s') \right)$$

- Bellman operator for $Q^\pi$ is defined as $T^\pi : R^{|S|x|A|} x R^{|S|x|A|}$ (maps action-value functions to action-value functions)

$$(T^\pi Q^\pi)(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \sum_{a \in A} \pi(a'|s') Q^\pi(s', a')$$

- Expectation equation become:

$$T^\pi V^\pi = V^\pi$$

$$T^\pi Q^\pi = Q^\pi$$

- **Properties**

- **Monotonicity** : if $f_1 \leq f_2$

$$T^\pi f_1 \leq T^\pi f_2 \quad T^* f_1 \leq T^* f_2$$

- $V^\pi$ is a **fixed point** of the operator, so any other input will result in a vector different from $V^\pi$. In the space of vectors ,each point is a different vector with its state components. Somewhere there is a vector $V^\pi$ which has a closed form solution
- If $0 < \gamma < 1$ then $T^\pi$ is a **contraction** wrt to the **maximum norm** : only 1 fixed points,all other points moved towards it.
- Max-norm contractions for two vectors $f_1, f_2$

$$||T^\pi f_1 - T^\pi f_2||_\infty \leq ||f_1 - f_2||_\infty$$

$$||T^* f_1 - T^* f_2||_\infty \leq ||f_1 - f_2||_\infty$$

- $V^\pi, V^*$ are fixed points of $T^\pi, T^*$
- For any vector $f \in R^{|S|}$ and policy $\pi$

$$\lim_{k\to\infty} (T\pi)^k f = V^\pi$$

$$\lim_{k\to\infty} (T*)^k f = V^*$$

### 7.1.6 Optimality functions and operators

- Optimal policy $\pi^*$ : a policy that is better than or equal to all other policies

$$\pi^* \geq \pi \quad \forall \pi$$

  - $\pi^*$ achieves **optimal value function** $V^{\pi^*} = V^*$
  - $\pi^*$ achieves **optimal action function** $Q^{\pi^*} = Q^*$
  - There is always one for any MDP

$$\pi^*(a|s) = \begin{cases} 1 & \text{if } a = argmax_{a\in A} Q^*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

- **Optimal state value function** $V^*$ is the maximum value function over all polices

$$V^* = max_\pi V^\pi(s)$$

- **Optimal action-value function** $Q^*$ is the maximum action value function over all policies

$$Q^*(s,a) = max_\pi(s,a)$$

- **Bellman optimality equation for V**

$$V^*(s) = max_a \left\{ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \right\}$$

- **Bellman optimality equation for Q**

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) max_a Q^*(s',a')$$

### 7.1.7 Solution strategies

- Bellman optimality operator non-linear so no closed form solution but many iterative methods
  - Dynamic programming : **Policy Iteration** , **Value Iteration**
  - Linear programming
  - Reinforcement Learning : **Q-Learning** , **SARSA**

# 8 Solving MDPs

## 8.1 Policy Search

- Solving MDPs = finding **optimal policy**
- **Brute force** naive approach :
  - enumerate all the Markovian policies
  - evaluate each policy

- retrun best one

- Brute force complexity : $|A|^{|S|}$. Improved with
  - Restrict search to subset of possible best policies
  - Use **stochastic** optimization algorithm

## 8.2 Dynamic Programming

Recursively solves **subproblems** and then combines the solution.

- **Optimal substructure** : optimal solutions can be decomposed into subproblems ( **MDPs Bellman Equation**)
- **Overlapping subproblems**: subproblems may recur many times so solutions can be cached and reused ( MDPs **Value-Function**)
- **Fully known MDPs** used for
  - **Prediction** :
    - input $\langle S, A, R, P, \gamma, \mu \rangle$
    - policy $MRP\langle S, P^\pi, R^\pi, \gamma, \mu \rangle$
    - output: **value function** $\boldsymbol{V^\pi}$
  - **Control**:
    - input $MRP\langle S, P^\pi, R^\pi, \gamma, \mu \rangle$
    - output : value-function $V^*$ and optimal policy $\pi^*$
- Case of **finite-horizon** MDPs (non-stationary policy!) : use backward induction
  - **Backward recursion**

$$V_k^*(s) = max_{a \in A_k} \left\{ R_k(s,a) + \sum_{s' \in S_{k+1}} P_k(s'|s,a) V_{k+1}^*(s') \right\} \quad k = N-1, ...0$$

  - **Optimal policy**

$$\pi^*(s) = max_{a \in A_k} \left\{ R_k(s,a) + \sum_{s' \in S_{k+1}} P_k(s'|s,a) V_{k+1}^*(s') \right\} \quad k = 0, ...N-1$$

– Total cost $N|S||A|$ vs $|A|^{N|S|}$ of brute search

### 8.2.1 Policy Iteration

For a given policy $\pi$ compute $V^\pi$

- State value function for policy $\pi$

$$V^\pi(s) = E\left\{\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right\}$$

- Bellman Equation for $V^\pi$

$$V^\pi(s) = \sum_{a\in A} \pi(a|s)\left[R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a)V^\pi(s')\right]$$

- Closed form solution $V^\pi = (I - \gamma P^\pi)^{-1} R^\pi \rightarrow$ **huge complexity**
- **Policy iteration** : **policy evaluation** + **policy improvement**
- **Policy evaluation**: full policy evaluation backup

$$V_{k+1}(s) \leftarrow \sum_{a\in A} \pi(a|s)\left[R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a)V_k(s')\right]$$

  – With **sweep** : state backup to update only modified states

  – After a few iterations, even if optimal value function is not found,the **optimal policy** has usually converged (depends on shape of V).This is also what makes it faster than **Value Iteration**

- **Policy improvement** : consider a deterministic policy $\pi$ and a given state s, would it be better to perform an action $a \neq \pi(s)$? By acting greedily :

$$\pi'(s) = argmax_{ainA} Q^\pi(s,a)$$

$$Q^\pi(s,\pi'(s)) = max_{a\in A} Q^\pi(s,a) \geq Q^\pi(s,\pi(s)) = V^\pi(s)$$

**Policy improvement theorem**

Let $\pi$ and $\pi'$ be any pair of deterministic policies such that $Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s), \forall s \in S$ , then policy $\pi'$ must be **as good or better** than $\pi$ , $V^{pi'} \geq V^{\pi}, s \in S$

Proof

$$\begin{aligned} V^{\pi}(s) &\leq Q_{\pi}(s, \pi'(s)) = E_{\pi'}[r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s] \\ &\leq E_{\pi'}[r_{t+1} + \gamma Q^{\pi}(s_{t+1}, \pi'(s_{t+1})) | s_t = s] \\ &\leq E_{\pi'}[r_{t+1} \gamma r_{t+2} + \gamma^2 Q^{\pi}(s_{t+2}, \pi'(s_{t+2})) | s_t = s] \\ &\leq E_{\pi'}[r_{t+1} \gamma r_{t+2} + ... | s_t = s] = V^{\pi'}(s) \end{aligned}$$

- If improvement stop $V^{\pi} = V^{\pi'} \rightarrow$ **Bellman optimality equation**

$$V^{\pi} = V^{\pi'} = V^*$$

$$\pi = \pi^*$$

### 8.2.2 Value iteration

Find the **optimal policy** by iteratively applying **Bellman Optimality Equation** without an explicit policy.

- max norm $||V||_{\infty} = max_{s \in S} |V(s)|$

Value iteration converges to the optimal state-value function $\lim_{x \to \infty} V_k = V^*$

$$||V_{k+1} - V^*||_{\infty} = ||T^* V_k - T^* V^*||_{\infty} \leq \gamma ||V_k - V^*||_{\infty} \leq$$

$$... \leq \gamma^{k+1} ||V^0 - V*||_{\infty} \rightarrow \infty$$

$$||V_{i+1} - V_i||_{\infty} < \epsilon \implies ||V_{i+1} - V^*||_{\infty} < \frac{2\epsilon}{1 - \gamma}$$

## 8.3 Infinite Horizon Linear Programming

- Value iteration convergence:

$$V^*(s) = max_a \left\{ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \right\}$$

- LP formulation for $V^*$:
  - $\min_v \sum_{s \in S} \mu(s) V(s)$
  - subject to : $V(s) geq R(s,a) + \sum_{s' \in S} P(s'|s,a) V(s') \quad \forall s \in S, \forall a \in A$
- $|S|$ variables , $|S||A|$ constraints
- **Optimal Bellman Operator** $T^*$
  - $\min_v \mu^I V$
  - s.t. $V \geq T^* V$

  Thanks to the monotonicity property $U \geq V \rightarrow T^* U \geq T^* V$ and by repeated application $V \geq T^* V \geq T^{*^2}(V) \geq T^{*^3}(V) ... \geq T^{*^\infty}(V) = V^*$. Since any feasible solution must satisfy $V \geq T^*(V)$ it satisfies also $V \geq V^*$

### 8.3.1 Dual Problem

- $$max_\lambda \sum_{s \in S} \sum_{a \in A} \lambda(s,a) R(s,a)$$
- s.t.
$$\sum_{a' \in A} \lambda(s',a') = \mu(s) + \gamma \sum_{s \in S} \sum_{a \in A} \lambda(s,a) P(s'|s,a), \forall s' \in S$$
- s.t.
$$\lambda \geq 0 \forall s \in S, a \in A$$

  In this case $\lambda(s,a) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s, a_t = a)$.
  The optimal policy is
$$\pi^*(s) = argmax_a \lambda(s,a)$$

# 9 RL in finite domains

Model is **not known** (**model free)** but it is possible to interact with the environment

- **Model-free prediction**: **estimate** value function of an unknown MDP ( MDP +policy)
- **Model-free control: optimize** the value function of an unknown MDP

# 10 Monte Carlo RF

- Model-Free : no known MDP
- Learns directly from experience : **complete** episodes ( no bootstrapping)
- Simple idea **value=mean return**
- Works only with **episodic MDPs** where all episodes **terminate**

Can be used for

- Prediction
  - **Input** : episodes of experience $\{s_1, a_1, r_2, ...s_T\}$ generated by policy $\pi$
  - **Output** : value function $V^\pi$
- Control
  - **Input** : episodes of experience $\{s_1, a_1, r_2, ...s_T\}$ generated by policy $\pi$
  - **Output** : optimal value function $V^*$
  - **Output** : optimal policy $\pi^*$

## 10.1 Monte-Carlo Prediction

- Goal : estimate value function for a given policy by averaging the returns observed after visits to states. As more returns are observed the average should **converge** to the **expected value**

- X is R.V. with mean $\mu$ and variance $\sigma^2$
  - **Empirical mean**:
$$\hat{\mu}_n = \frac{1}{n}\sum_{i=0}^{n} x_i$$
  - $E[\hat{\mu}_n] = \mu$ , $Var[\hat{\mu}_n] = \frac{Var[X]}{n}$
  - Weak law ($\hat{\mu}_n \rightarrow^P \mu$) and string law ($\hat{\mu}_n \rightarrow_{a.s.} \mu$) of large numbers
- Policy evaluation of $V^\pi$ uses **empirical mean** instead of expected return and can be found using **two-approaches**
  - **First-Visit MC**: average returns only for the first time s is visited (**unbiased estimator** ) in an episode
  - **Every-Visit-MC** : average returns for every time s is visited (**biased but consistent**)
- Mean can be computed **incrementally** :
$$\hat{\mu}_k = \frac{1}{k}\sum_{j=1}^{k} x_j = \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) = \frac{1}{k}(x_k + (k-1)\hat{\mu}_{k-1}) = \hat{\mu}_{k-1} + \frac{1}{k}(x_k - \hat{\mu}_{k-1})$$
- This can be used for **incremental updates**, for each state s with return $v_t$ and being N the time-steps s has been visited:
$$N(s_t) \leftarrow N(s_t) + 1$$
$$V(s_t) \leftarrow V(s_t) + \frac{1}{N(s_t)}(v_t - V(s_t))$$
(Discounted return - Expected reward)
- A **running mean** should be used in **non-stationary problems** , which may no converge though :
$$V(s_t) \leftarrow V(s_t) + \alpha(v_t - V(s_t))$$

### 10.1.1 Stochastic Approximation of Mean Estimator

Let X be random variable in $[0, 1]$ with mean $\mu = E[x]$. Let $x_i \sim X, i = 1...n$ iid realizations of X.
The estimator (**exponential average**)

$$\mu_i = (1 - \alpha_i)\mu_{i-1} + a_i\mu_i$$

with $\mu_1 = x_1$ and $a_i$ learning rates

- if $\sum_{i\geq 0} \alpha_i = \infty$ and $\sum_{i\geq 0} \alpha_i^2 < \infty$ then

  $$\hat{\mu}_i \rightarrow_{a.s.} \mu$$

  which means that the estimator $\hat{\mu}_n$ is **consistent**

# 11 Temporal Difference Learning

- Model-Free : no known MDP
- Learns directly from experience : **incomplete** episodes ( bootstrapping)
- Updates guess with a guess

## 11.1 TD Prediction

- Learn $V^\pi$ online from experience under policy $\pi$
- Monte-Carlo incremental update $V(s_t) \leftarrow V(s_t) + \alpha(v_t - V(s_t))$
- Simplest Temporal Difference learning algorithm **TD(0)** : update value $V(s_t)$ towards **estimated return** $r_{t+1} + \gamma V(s_{t+1})$

  $$V(s_t) \leftarrow V(s_t) + \alpha(\boldsymbol{r_{t+1} + \gamma V(s_{t+1})} - V(s_t))$$

  where

  - $r_{t+1} + \gamma V(s_{t+1})$ is the **TD - Target**
  - $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ is the **TD-Error**

## 11.2 MC vs TD

**TD**

- learns **before** knowing final outcome or **without** the final outcome
- learn **online** at every step
- can learn from **incomplete** episodes
- works in **continuing** non-terminating MDPs
- **low variance ,some bias**:
  - **TD-target** is a **biased** estimate of $V^\pi$ unless $V^\pi(s_{t+1}) = V(s_{t+1})$
  - **TD-target** has much lower variance as target depends on **one random** action,transition and reward
- **TD(0)** converges to $V^\pi(s)$
- Is sensitive to initial values

**MC**

- must wait **until end of episode** before return is known
- learns only from **complete** sequences
- works only for **episodic** MDPs
- **high variance ,no bias**:
  - the return $v_t$is an **unbiased** estimate of $V^\pi$
  - return $v_t$ has high variance as it depends on **many random** actions,transitions and rewards
- **Not** sensitive to initial values

**Batch updating**

- MC and TD converge for experience $\to \infty$

- If finite amount of experience : repeat experience, given a V the increments are computed a computed at every time step t at **non-terminal** states but **value function** is changed only once, by the sum of all increments.This is repeat until convergence.
  - TD(0) **converges** deterministically to a **single answer** independent of $\alpha$ (which must be sufficiently small ). It converges to the **max likelihood** Markov model

$$\hat{P}(s'|s,a) = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T} \mathbf{1}(s_t^k, a_t^k, s_{t+1}^k = s, a, s')$$

$$\hat{R}(s,a) = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T} \mathbf{1}(s_t^k, a_t^k = s, a) r_t^k$$

  - MC constant $\alpha$ also converges in the same conditions but to a **different answer**. It converges to the **minimum mean squared error**

$$\sum_{k=1}^{K} \sum_{t=1}^{T} (v_t^k - V(s_t^k))^2$$

- Markov **memoryless** property :
  - exploited by TD(0) $\rightarrow$ works better in Markovian environments
  - not exploited by MC $\rightarrow$ works better in Non-Markovian environments

## 11.3 *n*-Step TD Prediction

- *n-steps* enable **bootstrapping** to occur over multiple steps

$$\begin{array}{ll} n = 1 \quad (TD) & v_t^{(1)} = r_{t+1} + \gamma V(s_{t+1}) \\ n = 2 & v_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V(s_{t+2}) \\ \vdots & \vdots \\ n = \infty \quad (MC) & v_t^{(\infty)} = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{T-1} r_T) \end{array}$$

n-step Return: $v_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n})$

- $n$-step TD learning

$$V(s_t) \leftarrow V(s_t) + \alpha(v_t^{(n)} - V(s_t))$$

- $n$-steps can be **averaged** over **different n**.E.g. average 2-step and 4-step return

$$\frac{1}{2}v^{(2)} + \frac{1}{2}v^{(4)}$$

which combines information about 2 different time steps. More time-steps can be combined using **lambda return**

- **$\boldsymbol{\lambda}$ return $\boldsymbol{v_t^\lambda}$** combines all $n$-step returns ,each weighted proportional to $\lambda^{n-1}$ and is normalized by a factor of $1-\lambda$ to ensure they sum up to 1

$$v_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}v_t^{(n)}$$

  - **Forward-view** $TD(\lambda)$ look into the future to compute the return. For each state visited, look forward in time to all future rewards and decide how best to combine them:

$$V(s_t) \leftarrow V(s_t) + \alpha(v_t^\lambda - V(s_t))$$

    Need whole episodes like MC

  - **Backward-view** $TD(\lambda)$ updates weight vector on every step of an episode rather than only at the end, and thus it estimates may be better **sooner**.The computations are **equally distributed over time** rather than at the end of each episode . Finally it can be applied to **incomplete episodes**

$$e_0(s) = 0, \quad e_t(s) = \gamma\lambda e_{t-1}(s) + \mathbf{1}(s = s_t)$$

$$V(s) \leftarrow V(s) + \alpha\delta_t e_t(s)$$

    Where $e_t$ is an **eligibility trace** a heuristic that combines **frequency** and **recency** heuristic.
    $delta_t$ is the TD error.
    $\lambda$ is the **trace decay** parameter that tells at which rate the trace falls.

    * $\lambda = 0$ only current state is updated , which is equivalent to $TD(0)$
    * $\lambda = 1$ the sum **telescopes** into MC Error. In this case it is equivalent to MC Every Visit if **updated offline**. If update **online** the answer can be different.

* Convergence problems can be solved by setting the trace to 1 instead of adding 1

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & s \neq s_t \\ 1 & s = s_t \end{cases}$$

# 12 Model-Free Control

- Model free control solves problems where model is unknown but experience can be sampled or the model is known but too complex to use except for the experience.
- How can learning the **optimal policy** be achieved while acting in a sub-optimal way in order to **explore all actions**?
  - **On-Policy** : learn about policy $\pi$ from experience samples from $\pi$. It learns action-values for **near optimal policies**
  - **Off-Policy** : learn about policy $\pi$ sampled from $\bar{\pi}$. Target policy is **different** than behavioural policy

## 12.1 On-Policy

### 12.1.1 On-policy : Monte-Carlo Control

- **Policy evaluation** performed on $Q(s,a)$ instead of $V(s)$ because $V(S)$ **requires a model** for **greedy policy improvement**

$$\pi' = argmax_{(a\in A)}\{R(s,a) + P(s'|s,a)V(s')\}$$

$$vs.$$

$$\pi' = argmax_{(a\in A)}Q(s,a)$$

- **Policy improvement** performed using $\boldsymbol{\epsilon} - greedy$ exploration that ensures **continual** exploration where all **m-actions** are tried with **non-zero probability**

$$\pi(s,a) = \begin{cases} \frac{\epsilon}{m} + 1 - \epsilon & \text{if } a^* = argmax_{(a\in A)}Q(s,a) \\ \frac{\epsilon}{m} & \text{otherwise} \end{cases}$$

For any $\epsilon$-greedy policy $\pi$, the $\epsilon$-greedy policy $\pi'$ wrt to $Q^\pi$ is an **improvement**

$$\begin{aligned} Q^\pi(s, \pi'(s)) &= \sum_{a \in A} \pi'(a|s) Q^\pi(s,a) \\ &= \frac{\epsilon}{m} \sum_{a \in A} Q^\pi(s,a) + (1-\epsilon) max_{a \in A} Q^\pi(s,a) \\ &\geq \frac{\epsilon}{m} \sum_{a \in A} Q^\pi(s,a) + (1-\epsilon) max_{a \in A} \frac{\pi(a|s) - \frac{\epsilon}{m}}{1-\epsilon} Q^\pi(s,a) \\ &= \sum_{a \in A} \pi(a|s) Q^\pi(s,a) = V^\pi(s) \end{aligned}$$

Therefore $V^{\pi'} \geq V^\pi$

- In this scenario the policy evaluation **approximates** the unknown real $Q^\pi \rightarrow Q \approx Q^\pi$
- Policies are labeled as **GLIE** ( Greedy in the limit of infinite exploration ) if they satisfy the conditions
  - All state-action pairs are explored **infinitely** many times

$$\lim k \rightarrow \infty N_k(s,a) = \infty$$

  - The policy converges to a **greedy policy**

$$\lim k \rightarrow \infty \pi_k(a|s) = \mathbf{1}(a = argmax_{(a \in A)} Q_k(s', a')$$

- Sample k-th episode using $\pi : \{s_1, a_1, r_2, ..., s_T\} \sim \pi$. For each state-action in the episode

$$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{1}{N(s_t, a_t)} (v_t - Q(s_t, a_t))$$

Improve policy based on new action-value function

$$\epsilon \leftarrow \frac{1}{k}$$

$$\pi \leftarrow \epsilon - greedy(Q)$$

GLIE Monte-Carlo control **converges** to the **optimal** action-value function $Q(s,a) \leftarrow Q^*(s,a)$

- Relevant time scales
  - Behavioural time-scale $\frac{1}{1-\gamma}$ discount factor
  - Sampling in the estimation of Q : $\alpha$ learning rate
  - Exploration $\epsilon$

$$1-\gamma >> \alpha >> \epsilon$$

$$\text{Initially} : 1-\gamma \approx \epsilon \approx \alpha$$

  Then decrease $\epsilon$ faster than $\alpha$

### 12.1.2 On-Policy : TD with SARSA

- Many advantages over MC
  - Lower variance
  - Online
  - Incomplete sequences
- Uses update rule

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

  which is done after every transition from a non-terminal state s. If s' is terminal $Q(s',a') = 0$

SARSA converges to the **optimal** action-value function $Q(s,a) \rightarrow Q^*(s,a)$ if

- GLIE sequence of policies $\pi_t(s,a)$
- **Robbin-Monroe** sequence of steps-size $\alpha_t$

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

- **Policy Evaluation** : **SARSA**
- **Policy improvement** : $\boldsymbol{\epsilon - greedy}$
- $\lambda$ return is supported by SARSA as well
  - **Forward view** : update Q(s,a) to $\lambda -$ return $v_t^\lambda$
  - **Backward view** : incorporate eligibility traces for state-action pairs $e_t(s,a) = \gamma\lambda e_{t-1}(s,a) + \mathbf{1}(s_t, a_t = s, a)$

## 12.2 Off-Policy

- Learn about **target policy** $\pi(a|s)$ while following behavioural policy $\bar{\pi}(a|s)$
- Use **Importance sampling** : estimate expectation of a different distribution wrt the distribution used to draw samples

$$E_{x\sim P}[f(x)] = \sum P(x)f(x) = \sum Q(x)\frac{P(x)}{Q(x)}f(x) = E_{x\sim Q}[\frac{P(x)}{Q(x)}f(x)]$$

### 12.2.1 Off-Policy : Monte Carlo

1. Uses return generated from $\bar{\pi}$ to evaluate $\pi$
2. Weight return $v_t$ calculated according to **similarity** between policies
3. Multiple **importance sample corrections** along with episode :

$$v_t^\mu = \frac{\pi(a_t|s_t)}{\bar{\pi}(a_t|s_t)}\frac{\pi(a_{t+1}|s_{t+1})}{\bar{\pi}(a_{t+1}|s_{t+1})}...\frac{\pi(a_T|s_T)}{\bar{\pi}(a_T|s_T)}v_t$$

4. Update towards **corrected return**

$$Q(s_t|a_t) \leftarrow Q(s_t|a_t) + \alpha(\boldsymbol{v_t} - Q(s_t, a_t))$$

   Cannot be used where $\bar{\pi} = 0$ and $\pi$ is not zero

- Importance sampling can **dramatically** increase **variance**

### 12.2.2 Off-Policy : TD with SARSA

1. Use **TD Targets** generated from $\pi$ to evaluate $\bar{\pi}$
2. Weight TD target $r + \gamma Q(s', a')$ according to **similarity** between policies
3. Use **single** importance sampling correction

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left( r_{t+1} + \gamma \frac{\pi(a|s)}{\bar{\pi}(a|s)} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right)$$

- Has much lower variance than MC

### 12.2.3 Off-Policy : TD with Q-Learning

- Learn about optimal policy $\pi = \pi*$
- From experience sampled from behaviour policy $\bar{\pi}$ , estimate

$$Q(s, a) \approx Q^*(s, a)$$

- Uses update rule

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma max_{a \in A} Q(s', a') - Q(s, a))$$

- It is possible to use **eligibility traces** with Q-Learning using **Watkins**
  - Zero-out eligibility traces after a non-greedy action
  - Do max when backing up at first non-greedy choice

$$e_t(s, a) = \begin{cases} 1 + \gamma \lambda e_{t-1}(s, a) & if s = s_t, a = a_t, Q_{t-1}(s_t, a_t) = max_a Q_{t-1}(s_t, a) \\ 0 & if Q_{t-1}(s_t, a_t) \neq max_a Q_{t-1}(s_t, a) \\ \gamma \lambda e_{t-1}(s, a) & otherwise \end{cases}$$

  This gives an **disadvantage** in early learning as the eligibility trace will be **zeroed-out** frequently. A solution is to use **Peng's** Q-Learning (hard to implement)

  - Back-up max action **at the end**
  - Never cut traces

# 13 Multi-Armed Bandit

- $Q(s,a)$ often not known and online **decision making** faces two challenges : **exploration** vs **exploitation**. Depending on the situation sometimes gaining in the future by making sacrifices in the present is useful.
- Exploration is done for example with
  - $\epsilon - greedy$ : does not achieve optimal policy
  - **Softmax-Bolzman Distribution**

$$\pi(a_i|s) = \frac{e^{\hat{Q}(\frac{a_i|s}{\tau})}}{\sum_{a \in A} e^{\frac{\hat{Q}(a|s)}{\tau}}}$$

    which weights the action according to $\hat{Q}(s,a)$ and $\tau$ is a **temperature** parameter that increases over time. This approach selects with high probability an action with high reward.

- MAB is a special case of MDP
  - **Single** state S (expectation of reward will change over time stochastically )
  - Set of **arms**
  - P Probability matrix (only 1 state , probability of staying in that state is **1**)
  - R reward function (can be **deterministic**, **stochastic** , **adversary**)
  - $\gamma = 1$, $\mu = 1$

## 13.1 Stochastic MAB

- A MAB setting is a tuple $\langle A, R \rangle$
  - A set of N possible arms
  - R is a set of **unknown** distributions $r_{i,t} \sim R(a_i)$ and $E[R(a_i)] = R(a_i)$ (for simplicity can be assumed to be $\in [0,1]$)
- Procedure is as follows:

1. Each time step t agents selects single arm $a_{i_t}$
2. Environment generates reward $r_{i_t}$
3. Agents updates info by means of history $h_t$

- Aim is to **maximize cumulative reward**

$$\sum_{t=1}^{T} r_{i_t,t}$$

### 13.1.1 Regret

- Assume best arm is $a^*$ with expected reward $R^* = R(a) = max_{a \in A} E[R(a)]$. Then the **loss** at each time step is

$$R(a^*) - R(a_{i_t})$$

- Over a time horizon T on **average** the loss (called **Expected Pseudo Regret**) is

$$L_T = TR^* - E[\sum_{t=1}^{T} R(a_{i_t})]$$

- Maximization of reward = minimization of cumulative regret

- Second formulation using **difference** between $a_i$ and optimal one $a^*$

$$\Delta_i = R^* - R(a_i)$$

Being $N_t(a_i)$ the **number of times** arm $a_i$ has been pulled , the **regret** becomes

$$L_T = TR^* - E[\sum_{t=1} R(a_{i_t})] = E[\sum_{t=1} R^* - R(a_{i_t})]$$

$$= \sum_{a \in A} E[N_t(a_i)](R^* - R(a_i)) = \sum_{a \in A} E[N_T(a_i)]\Delta_i$$

### 13.1.2 Lower Bound

- $\Delta_i$ implies that any algorithm performance is implied by the **similarity among arms**. The more they are similar = the more difficult the problem

Given a MAB stochastic problem ,any algorithm satisfies

$$\lim_{T\to\infty} L_T \geq logT \sum_{a_i|\Delta_i>0} \frac{\Delta_i}{KL(R(a_i), R(a^*))}$$

### 13.1.3 Pure Exploitation Algorithm

- Always select action $a_{i_t} = argmax_a \hat{R}_t(a)$ where the expected reward is

$$\hat{R}_t(a_i) = \frac{1}{N_t(a_i)} \sum_{j=1}^{t} r_{i,j} \mathbf{1}\{a_i = a_{i_j}\}$$

- Might **not converge to optimal solution**
- Not considering **uncertainty** in $\hat{R}_t(a) \rightarrow$ need to provide explicit bonus for **exploration**
- Two formulations
  - **Frequentist :** $R(a_1), ..., R(a_N)$ unknown parameters and a policy selects each time step an arm based on history
  - **Bayesian**: $R(a_1), ..., R(a_N))$ are random variables with priors and a policy selects at each time step an arm based on observation history. The more **uncertain** the more we want the algorithm to explore the option

### 13.1.4 Upper Confidence Bound

- Instead of using empiric estimate use **upper bound** $U(a_i)$ over expected value $R(a_i)$

$$U(a_i) := \hat{R}_t(a_i) + B_t(a_i) \geq R(a_i)$$

  with **high probability**

- Bound $B_t(a_i)$ will depend on how much information we have on the arm (for example number of times the arm has been pulled)

- Upper bound can be set using **Hoeffeding Inequality Bound** : $X_1, ..., X_t$ iid random variables with support in $[0, 1]$ and identical mean $E[X_i] =: X$ ,$\bar{X}_t = \frac{\sum_{i=1}^{t} X_i}{t}$ the sample mean , then

$$P(X > \bar{X}_t + u) \leq e^{-2tu^2}$$

Applied to each arm

$$P(R(a_i) > \hat{R}(a_i) + B_t(a_i)) \leq e^{-2tB_t(a_i)^2}$$

Then it can be computed as:

1. Pick probability p that the value exceed the bound

$$e^{-2N_t(a_i)B_t(a_i)^2} = p_t$$

2. Solve to find $B_t$

$$B_t(a_i) = \sqrt{\frac{-logp_t}{2N_t(a_i)}}$$

3. Reduce value over time $p_t = t^{-4}$

$$B_t(a_i) = \sqrt{\frac{2logt}{N_t(a_i)}}$$

4. Ensure to select optimal action as samples increase

$$\lim_{t\to\infty} B_t(a_i) = 0 \implies \lim_{t\to\infty} U_t(a_i) = R(a_i)$$

- **UCB1** exploits upper bound. At each time step t:
  - Compute $\hat{R}(a_i)$
  - Compute $B_t(a_i)$
  - Play arm $a_{i_t} = argmax_{a_i \in A} \left(\hat{R}_t(a_i) + B_t(a_i)\right)$

At finite time T the **expected total regret of the UCB1** applied to stochastic MAB is

$$L_T \leq 8logT \sum_{i|\Delta_i>0} \frac{1}{\Delta_i} + (1 + \frac{\pi^2}{3}) \sum_{i|\Delta_i>0} \Delta_i$$

- **Thompson Sampling** is a general **Bayesian** technique for online learning

  1. Consider Bayesian prior for each arm $f_1, .., f_N$
  2. At each round t, sample from each on the distributions $\hat{r}_1, ..., \hat{r}_N$
  3. Pull the arm $a_i$ with the **highest sampled value** $i_t = argmax_i \hat{r}_i$
  4. Update prior with new info

  Example with prior $f_i(0) = Beta(1, 1)$ for each arm $a_i$. Keep distribution $f_i(t) = Beta(\alpha_t, \beta_t)$ incorporating information form each arm $a_i$

  - **Success** : $f_i(t+1) = Beta(\alpha_t + 1, \beta_t)$
  - **Failure** : $f_i(t+1) = Beta(\alpha_t, \beta_t + 1)$

  At time T, the **expected total regret** of the algorithm is

$$L_T \leq O\left(\sum_{i|\Delta_i>0} \frac{\Delta_i}{KL(R_{a_i}, R(a^*))}(logT + loglogT)\right)$$

## 13.2 Adversarial MAB

- Setting tuple $\langle A, R \rangle$
- A set of N arms
- R reward vector of $r_{i_t}$ rewards decided by an **adversarial player**
- Each step agent selects 1 arm and adversary chooses reward which the agents receives.
- Goal : **maximize cumulative reward**

$$\sum_{t=1}^{T} r_{i_t,t}$$

### 13.2.1 Regret

- Cannot compare received gain with **optimal one** and we cannot use **deterministic** algorithms ( adversary decides reward!). So only **weak regret** can be defined , compared with **best constant action**

$$L_T = max_i \sum_{t=1}^{T} r_{i,t} - \sum_{t=1}^{T} r_{i_t,t}$$

### 13.2.2 Lower Bound

Let **sup** be the supremum over all distribution of rewards such that for all $i \in \{1, ..., N\}$ the rewards $r_{i,1}, ..., r_{i,j} \in 0, 1$ are iid , and let **inf** be the infimum over all forecasters. Then

$$\inf \sup E[L_T] \geq \frac{1}{20}\sqrt{TN}$$

where the expectation is taken wrt both **random generation rewards** and **internal randomization** of forecaster

### 13.2.3 EXP3

- Variation of **Softmax**
- Probability of choosing an arm

$$\pi(a_i) = (1 - \beta)\frac{w_t(a_i)}{\sum_j w_t(a_j)} + \frac{\beta}{N}$$

where

$$w_{t+1}(a_i) = \begin{cases} w_t(a)e^{-\eta\frac{r_{i,t}}{\pi_t(a_i)}} & \text{if } a_i \text{ has been pulled} \\ w_t(a_i) & \text{otherwise} \end{cases}$$

At time T the **expected total regret** of EXP3 algorithm is with $\beta = \eta = \sqrt{\frac{NlogN}{(e-1)T}}$

$$E[L_T] \leq O(\sqrt{TNlogN})$$