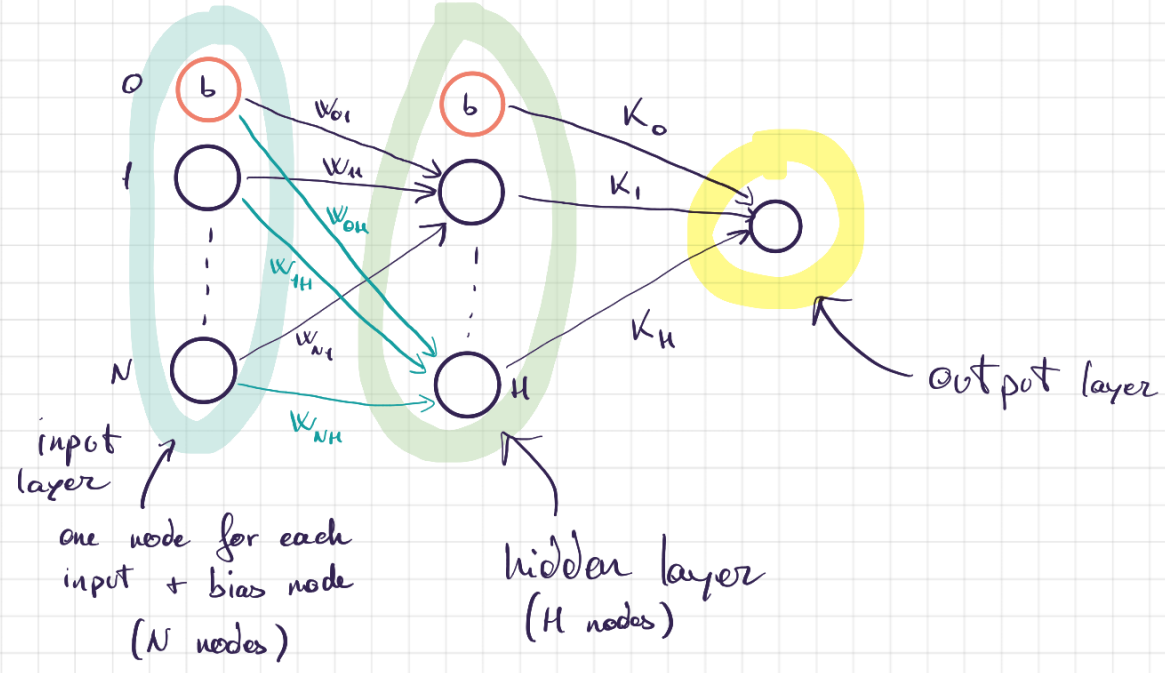
Static Stochastic Optimization

-- Neural Networks and Stochastic Gradient - - - - - - - -

NON-LINEAR NEURAL NETWORK

↳ used as a "model-free" method

$$\{W_{ih}\} \quad \begin{matrix} i = 0 \dots N \\ h = 1 \dots H \end{matrix}$$



↳ exactly as in LNN they have
- input nodes (+ bias node)
- output node

↳ additionally they have:
- hidden layer
- non-linear activation function

↳ between layers there are weights: W_{ih} between input and hidden; K_h between hidden and out

↳ the training process is about finding the best weights

> ! The non-linear part is really the activation function, because:
> ↳ a LC of LCs is still an LC

FORWARD PROPAGATION OF THE INPUT

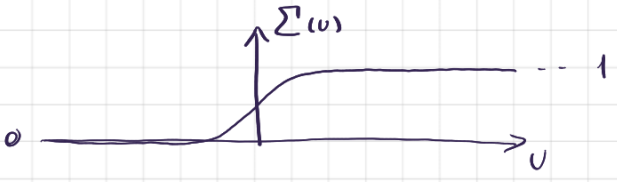
- **INPUT → HIDDEN** (x = input vector , $x_0 = 1$)

$$\bar{V}_h = \sum_{i=0}^{N} w_{hi} x_i \quad \rightarrow \text{intermediate hidden values}$$

$$V_h = \Sigma(U) \quad \rightarrow \text{final hidden values}$$

↳ $\Sigma(U) = \frac{1}{1+e^{-U}}$ → Sigmoid activation function

($\Sigma(\infty) = 1$, $\Sigma(-\infty) = 0$)



- **HIDDEN → OUTPUT**

$$o = \sum_{h=0}^{H} K_h V_h$$

TRAINING - BACKPROPAGATION ALGORITHM

↳ the aim is to adjust the weights in order to reduce the error between real and predicted.

$$\left[SSE = \sum_{p=1}^{n} (y_p - o_p)^2 \right] \rightarrow$$ since we need the minimum error we have to compute the derivatives

$$\frac{\partial SSE}{\partial W_{ih}} = 2 \sum (y_p - o_p) \frac{\partial o_p}{\partial W_{ih}} \quad \text{and} \quad \frac{\partial SSE}{\partial K_h} = 2 \sum (y_p - o_p) \frac{\partial o_p}{\partial K_h}$$

↳ remembering that:

$$\begin{cases} o_p = \sum_{h=0}^{H} K_h V_{ph} \\ V_{ph} = \sum (\bar{V}_{ph}) \\ \bar{V}_{ph} = \sum_{i=0}^{N} W_{ih} x_{pi} \end{cases}$$

↳ and computing $\Sigma'(\bar{V}_h) = \dfrac{e^{-\bar{V}_{ph}}}{(1+e^{-\bar{V}_{ph}})^2}$

$$= \frac{e^{-\bar{V}_{ph}}}{1+e^{-\bar{V}_{ph}}} \cdot \frac{1}{1+e^{-\bar{V}_{ph}}}$$

$$= (1 - V_{ph}) \cdot V_{ph}$$

↳ we have: $\dfrac{\partial o_p}{\partial W_{ih}} = \dfrac{\partial o_p}{\partial V_{ph}} \dfrac{\partial V_{ph}}{\partial \bar{V}_{ph}} \dfrac{\partial \bar{V}_{ph}}{\partial W_{ih}} = K_h V_{ph} (1 - V_{ph}) x_{pi}$

and: $\dfrac{\partial o_p}{\partial K_h} = V_{ph}$

↳ so we obtain:

$$\boxed{\frac{\partial SSE}{\partial W_{ih}} = 2 \sum_{p=1}^{n} (y_p - o_p) K_h V_{ph} (1 - V_{ph}) x_{pi}}$$

$$\boxed{\frac{\partial SSE}{\partial K_h} = 2 \sum_{p=1}^{n} (y_p - o_p) V_{ph}}$$

(!) The problem is that the SSE (cost function for our optimization) is NOT convex!
You cannot simply use Grad Desc
You need other approaches!

REMARKS AND TRICKS

- if TOL too small $\longrightarrow$ overfitting
- sigmoid function is $\begin{cases} \approx 0 & \text{for } v < -2 \\ \approx 1 & \text{for } v > 2 \end{cases}$
  - ↳ it may be needed to have wider range $\longrightarrow$ sigmoid normalization

LARGE DATASET PROBLEMS

↳ in general we want to solve the Empirical Risk Minimization:

$$\min \frac{1}{N} \sum \overbrace{(y_i - \xi_i^T x)^2}^{\text{error for every point, predict}}$$

↳ we can generalize: $\min F(x) \rightarrow \min \frac{1}{N} \sum f_i(x)$

f for i-th point, predict

↳ we can minimize through gradient descent (GD):

$$x_{t+1} = x_t - \mu \nabla F(x) = x_t - \mu \nabla \frac{1}{N} \sum f_i(x)$$

$$= x_t - \mu \frac{1}{N} \sum \nabla f_i(x)$$

(!) The sumation would be computationally unfeasible for a really large dataset!

↳ we need a less expensive way to minimize the function

STOCHASTIC GRADIENT

↳ instead of using $\sum_{i=1}^{n} \nabla f_i(x_t)$

we will use $\tilde{g}(x) = \nabla f_I(x)$, $I \sim U(1, N)$

↳ will it converge ?

$$E[\tilde{g}(x)] = \frac{1}{N}\nabla f_1(x) + \dots + \frac{1}{N}\nabla f_N(x) = \frac{1}{N}\sum \nabla f_i(x) = \nabla F(x)$$

↳ basically, instead of using the error of all the points, we use the error of only one point at the time

↳ it won't take the best step everytime, but in the long run it will be like having the real gradient as step all along

↳ moreover in ML we don't want the optimum because usually it means overfitting

BATCH GRADIENT DESCENT

↳ a further optimization can be considering B points instead of only one ($B << N$)

$$x_{t+1} = x_t - \mu \frac{1}{B} \sum_{j=1}^{B} \nabla f_{I_j}$$

$$E\left[\frac{1}{B}\sum \nabla f_{I_j}\right] = \frac{1}{B}\sum E[\nabla f_{I_j}] = \frac{1}{B}\sum \nabla F(x) = F(x)$$

RANDOM COORDINATE DESCENT

↳ if the problem is high dimensionality we could compute only partial derivatives rather than full gradient

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_d} \right)$$

we use:

random uniform

$$\tilde{g}(x) = d \left(0, \dots, 0, \frac{\partial f}{\partial x_j}, 0, \dots, 0 \right) \quad \text{where } j \sim U[1, d]$$

as before

$$E[\tilde{g}(x)] = \frac{1}{d} \sum d \left(0, \dots, 0, \frac{\partial f}{\partial x_j}, 0, \dots, 0 \right) = \nabla f(x)$$

ABSENCE OF MODEL

↳ despite non-linear NN are considered model-free there is actually a model

$$O_p = \sum_{h=0}^{H} K_h \sum \left(\sum_{i=0}^{N} w_i x_{pi} \right)$$

(sum) (sig) (sum)

↳ since it can became quite complicated we don't use any of its analytical expression, but we compute the output algorithmically

Discrete Stochastic Optimization

$$\max.\ c^T x + q^T y \quad \text{s.t. } Ax + Gy \le b,\ x \in \mathbb{R}^n,\ y \in \{0,1\}^M$$

constrained optimization problem

$y \in \{0,1\}^M$ → discrete

→ it is a more difficult problem with respect with the continous case

↳ if solution space is small → exhaustive search

↳ if solution space is large → heuristics

↳ an heuristic is an inexact algorithm that finds a good solution (not necessarily optimal) in a reasonable amount of time

METAHEURISTICS

↳ they are DESIGN PATTERNS and ALGORITHMIC IDEAS

↳ to transform it into heuristic algorithms you must exploit problem-dependent information

↳ need to have two ingredients:
- solution representation
- neighbour

- Solution representation examples:
  - ↳ assignment problem $[1\ 0\ 1\ 0\ 0\ 1]$
  - ↳ "travelling salesmen" $[N-3\ \ 5\ \ \dots\ \ 0\ \ N]$

- Neighbour examples: $[1\ 0\ 1\ 0\ 0\ 1]$ → $[1\ 0\ 0\ 0\ 0\ 1]$ or $[1\ 0\ 0\ 1\ 0\ 1]$

  $[3\ 1\ 4\ 5\ 2]$ → $[3\ 4\leftrightarrow 1\ 5\ 2]$ or $[3\ 4\ 5\leftrightarrow 1\ 2]$

LOCAL SEARCH

↳ easiest metaheuristic :

Algorithm 1 Local Search

1: create initial solution s_{min}
2: **while** while stopping criteria not met **do**
3: $s' = argmin_{s \in \mathcal{N}}(smin)$
4: **if** $c(s') < c(s_{min})$ **then**
5: $s_{min} = s'$
6: **end if**
7: **end while**
8: return s_{min}

→ at every step your new solution is the best solution in the neighbourhood of the solution at that time

GRASP

↳ Greedy Randomized Adaptive Search Procedure

↳ repeatedly applies local search from different starting solutions

↳ different variants are possible :

- First improvement : while creating neigh. solutions, explore the FIRST one that is better
- Best improvement : create full neigh. sol. space then take the best one

(!) Exploration :
go around randomly in the entire solution space

Exploitation :
go around the neighbourhood of the current solution

LNS

↳ Large Neighbourhood Search

↳ neighbourhood is created by changing large portions of solution

[1 0 0 1 0 1 0 0 1 1]
↓ destroy
[1 0 0 1 0 x x x 1 x x]
↓ repair
[1 0 0 1 0 1 0 1 1 0 0]

Algorithm 2 Large Neighborhood Search

1: create initial solution $s_{min} = s \in \mathcal{S}(I)$
2: **while** while stopping criteria not met **do**
3: $s' = r(d(s))$ ← destroy, repair
4: **if** accept(s, s') **then** ← we can accept worse solution
5: $s = s'$
6: **if** $c(s) < c(s_{min})$ **then** ← is better?
7: $s_{min} = s$
8: **end if**
9: **end if**
10: **end while**
11: return s_{min}

ALNS

↳ Adaptive LNS

↳ we can have many destroy/rebuild operators

$D = \{ d_i \}$ all destroy operators

$R = \{ r_i \}$ all repair operators

↳ to each d_i and r_i we assign a weight that determines the probability to choose it

$$p(d_i) = \frac{w(d_i)}{\sum w(d_j)} \quad ; \quad p(r_i) = \frac{w(r_i)}{\sum w(r_j)}$$

↳ once an operator is used we have to update weights:

↳ $v(h)$ =: # times operator was used

↳ $s(h)$ =: operator score

↳ $s(h)$ is increased by

decreasing ↓

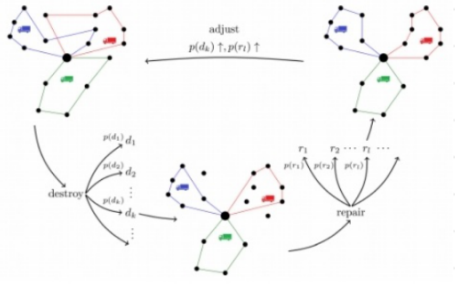
- δ_1 if new sol is best
- δ_2 if new sol improves
- δ_3 if new sol is accepted
- δ_4 if new sol is rejected

↳ $w(h)$ is calculated by:

$$w(h) = \begin{cases} (1-\rho)\, w(h) + \rho \frac{s(h)}{v(h)} & \text{if } v(h) > 0 \\ (1-\rho)\, w(h) & \text{if } v(h) = 0 \end{cases}$$

Algorithm 3 ALNS

```
1: create initial solution s.
2: while stopping criteria not met do
3:     for i = 1 to p_u do                                   ┐
4:         select r ∈ R, d ∈ D according                     │
           to probabilities p                                │
5:         s' = r(d(s))                                      │
6:         if accept(s,s') then                              │
7:             s = s'                                        │
8:             if c(s) < c(s_min) then                       │ LNS
9:                 s_min = s                                 │
10:            end if                                        │
11:        end if                                            │
12:    end for                                               ┘
13:    adjust the weights w and proba-                       ┐
       bilities p                                            │ ADAPT
14: end while                                                ┘
```

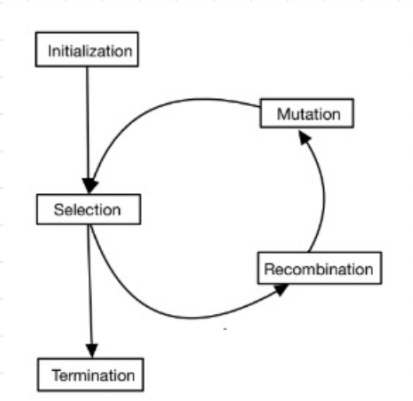


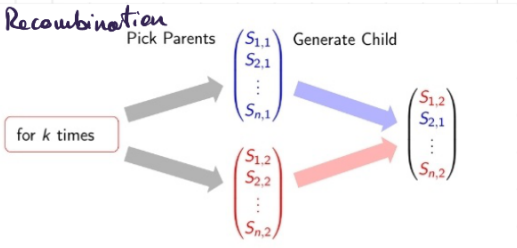
GENETIC ALGORITHMS

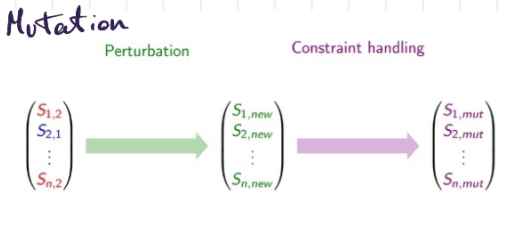
↳ mimics the natural selection

↳ the algorithm works like this:

1. Create an initial population of "m" solutions
2. Generate $K > m$ offsprings by
   - ↳ recombination
   - ↳ mutation
3. Apply natural selection to find the "m" most fit individuals to be used as new generation
4. Repeat from step 2 until stopping criterion







KERNEL SEARCH

↳ **Idea**: • Often in an optimal solution there are only a few non-zero values
• Solving the fully continous problem gives an idea of the solution to the discrete

↳ Example: [0,9 0,8 0,2 0,5 0,7 0,8]

↳ **KERNEL** = set of promising (likely not zero) variables

↳ Kernel Search Algorithm:

① Solve the continous relaxation of the discrete problem
↳ identify the KERNEL (Δ) of the problem
↳ partition the remaining into "q" groups called BUCKETS (B_j)

② Solve the restricted discrete problem over the KERNEL variables

③ For each BUCKET solve the problem $P[\Delta \cup B_i]$ adding two constraints:

$$\sum_{j \in B_i} x_j \geq 1 \rightarrow \text{force to add at least an}$$

$$score_{new} \geq score_{old} \rightarrow \text{improve the score}$$

↳ all variables with values ≥ 1 are added to Δ

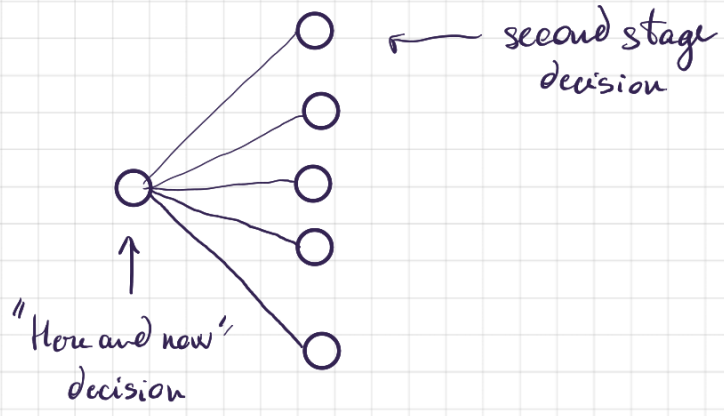
ML HEURISTICS

- TWO STEP OPTIMIZATION - - - - - -

↳ it happens in 3 phases:

① producing components
② observe end item demand
③ assembling end items

↳ usually scenarios are considered forming a tree:



↳ one idea can be to estimate the expected values by means of Monte Carlo techniques

↳ another idea may be solving the expected value problem
↳ by looking at the mean demand of the end items given the various scenarios.

↳ the solution of the Two Stage is given by:
- computing the solution of every scenario (or subsample if MC)
- computing the mean profit

$$\max_{x\in\mathbb{R}^I, y\in\mathbb{R}^J} \; - \sum_{i\in\mathcal{I}} C_i x_i + \mathbb{E}\left[\sum_{j\in\mathcal{J}} P_j y_j^s\right]$$

↳ the Two Stage solution generally gives better results since:

$$VSS = TSP - EVP$$

↑
it describes the gain of computing TSP (which is more computationally demanding)

where
- VSS =: value of stochastic solution
- TSP =: two stage problem
- EVP =: expected value problem

Dynamics Stochastic Optimization

↳ something happens dynamically in time

↳ we want to optimize a function that depends on the evolution of the process

- MARKOV CHAINS - - - - - - - - - - -

↳ dynamical models (system evolves in time)

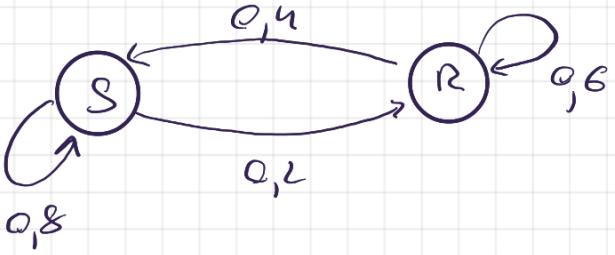
↳ they are special because simple

↳ example: weather forecast

↳ only two states RAINY (R) and SUNNY (S)

↳ (strong assumption) Tomorrow's weather only depends on today's weather

↳ $S \to S = 0,8$
$S \to R = 0,2$
$R \to R = 0,6$
$R \to S = 0,4$



↳ it can be resumed in the following

transition matrix: $P = \begin{matrix} \\ S \\ R \end{matrix} \begin{matrix} \begin{matrix} S & \quad R \end{matrix} \\ \begin{bmatrix} 0,8 & 0,2 \\ 0,4 & 0,6 \end{bmatrix} \end{matrix}$

Def MARKOV PROPRIETY IN DISCRETE TIME

↳ let $(X_n)_{n=0}^{\infty}$ be a discrete time stochastic process with a discrete state space S.

↳ $(X_n)_{n=0}^{\infty}$ is a DTMC (discrete time Markov chain) if for any $j, i, i_{n-1}, \dots, i_0 \in S$, the Markov propriety:

$$P(X_{n+1} = j \mid X_n = i, X_{n-1} = i_{n-1}, \dots, X_0 = i_0) = P(X_{n+1} = j \mid X_n = i) = p(i,j).$$

(X_{n+1}: tomorrow; X_n: today)

> ! In Markov Chains X_{i+1} is only dependent on X_i, and not on the further past

↳ $p(i,j)$ are called transition probabilities

↳ if the state space S is finite (say cardinality k) then the transition probabilities can be organized into the transition matrix $(k \times k)$ with non-negative entries such that row sums are 1:

$$p = \begin{pmatrix} p(1,1) & \cdots & p(1,k) \\ \vdots & & \vdots \\ p(k,1) & \cdots & p(k,k) \end{pmatrix}$$

$$\sum_{j \in S} p(i,j) = 1.$$

JOINT PROBABILITY OF A TRAJECTORY

↳ we can apply the Markov propriety to compute the probability of observing a trajectory $x_0\, x_1 \dots x_n$:

$$P[X_n = i_n\,,\, X_{n-1} = i_{n-1}\,,\, \dots\,,\, X_0 = i_0] =$$

$$= P[X_n = i_n \mid X_{n-1} = i_{n-1}\,,\, \dots\,,\, X_0 = i_0] \cdot P[X_{n-1} = i_{n-1}\,,\, \dots\,,\, X_0 = i_0]$$

by Markov Propriety

$$= p(i_{n-1}, i_n)\, P[X_{n-1} = i_{n-1} \mid X_{n-2} = i_{n-2}\,,\, \dots\,,\, X_0 = i_0]\, P[X_{n-2} = i_{n-2} \dots X_0 = i_0]$$

$$= p(i_{n-1}, i_n)\, p(i_{n-2}, i_{n-1}) \dots p(i_0, i_1)\, P[X_0 = i_0]$$

↳ to compute the joint probability of the trajectory is sufficient to know the transition matrix and the initial distribution $\alpha(i) = P[X_0 = i]$

N-STEP TRANSITION CHAPMAN-KOLOMOV EQS.

↳ calculate the transition probabilities in two steps (for example):

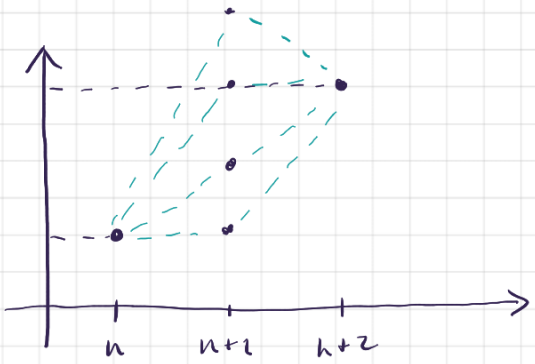
$$P^{(2)}(i,j) = P[X_{n+2} = j \mid X_n = i]$$

↳ applying the law of total probabilities:

$$P^{(2)}(i,j) = P[X_{n+2} = j \mid X_n = i]$$

$$= \sum_K P[X_{n+2} = j \mid X_{n+1} = K] \cdot P[X_{n+1} = K \mid X_n = i]$$

$$= \sum_K p(i,K)\, p(K,j)$$



$$\boxed{P^{(2)}(i,j) = \sum_K p(i,K)\, p(K,j)}$$

↳ note that:

$$P^{(2)} = P \cdot P = P^2$$

and more in general

$$P^{(m+n)} = P^{(m)} \cdot P^{(n)} = P^{m+n}$$

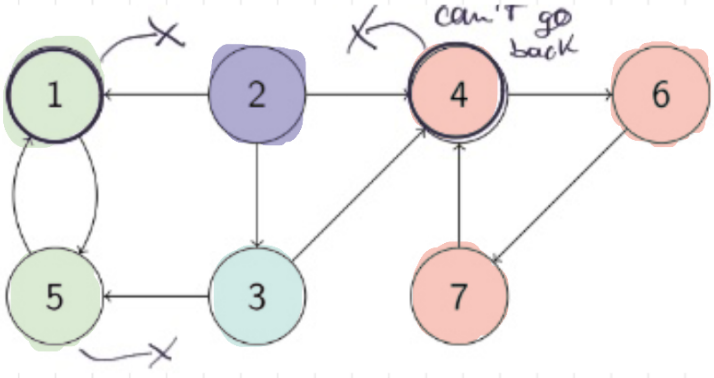
↳ computationally → $\left[P^2 \right]_{\text{IN\_STATE, FIN\_STATE}}$

CLASSIFICATION OF STATES and COMMUNICATION CLASSES

↳ if $\exists\ m \geq 0$ such that $p^{(m)}(i,i) > 0$

j is reachable from i (and write $i \to j$)

↳ if both $i \to j$ and $j \to i$ we say that the state i communicates with state j (and write $i \leftrightarrow j$)



Def class ?

↳ we can define:

- RECURRENT CLASSES : are visited again and again
  ↳ once in a recurrent class you can't exit
- TRANSIENT CLASSES : are visited only once

LIMITING DISTRIBUTION

↳ is a distribution π such that

$$\lim_{n\to\infty} p^{(n)}(x,y) = \pi(y)$$ regardless of x

↳ which means in the long run what is the probability of turning to "y" ?

ASYMPTOTIC FREQUENCY

↳ the long run proportion of visits to "y"

$$\lim_{n\to\infty} \frac{N_n(y)}{n}$$ → number of visits to "y" in the first "n" steps

(!) They are defined as independent concepts but under certain conditions they are the same

STATIONARY DISTRIBUTION

↳ given $v = \begin{pmatrix} P(X_1 = 1) \\ \vdots \\ P(X_1 = n) \end{pmatrix}$ so $v(i) = P(X_1 = i)$

↳ how do you compute v knowing:

- transition matrix P
- initial distribution vector α

↳ $$v(i) = P(X_1 = i) = \sum_j P(X_1 = i \mid X_0 = j)\, P(X_0 = j)$$
$$= \sum_j p(j,i)\, \alpha(j)$$

↳ so basically $$\left[\begin{pmatrix} v \end{pmatrix} = \begin{pmatrix} \alpha \end{pmatrix} \begin{pmatrix} P \end{pmatrix} \right]$$

↳ "v" describes the probability to be in state i at time 1

↳ being π the eigenvector of P corresponding to the eigenvalue $= 1$

↳ if $\alpha = \pi$ then $v = \pi$ for all times and so we call it a **stationary distribution**

↳ $\pi \to \pi(i) = P(X_0 = i) = P(X_n = i)$

↳ the probability to be in state "i" at any time is the same

(!) To compute π you have to solve a linear system:

$$\pi = \pi \cdot P$$

- Markov Decision Process - - - - - - - - - - -

↳ Markov chains are dynamical models whose time evolution follow a random law with a special dependence between observations

↳ they are defined by:
- initial distribution α
- transition matrix P

↳ let us consider:
- a specific action can change transition matrix P
- each action is associated with a reward (gain/cost)

↳ a function that associates to any state an action to be performed is called POLICY

Def MARKOV DECISION PROCESS
- a set of actions called A (includes both transition matrix and rewards)
- a set of states S
- a policy $\mu : s \in S \rightarrow a \in A$ (the policy is what I'm optimizing over)

ACTIONS

↳ each action defines:

- a transition matrix P_a : describes the probabilities of going from "s_t" to "s_t'" given that at s_t we took action "a"

  $[P_a = p(s_t, a_t, s_t')]$

- a reward matrix R_a: describes the gain/cost of taking action "a"

  $[R_a = r(s_t, a_t)]$

POLICIES

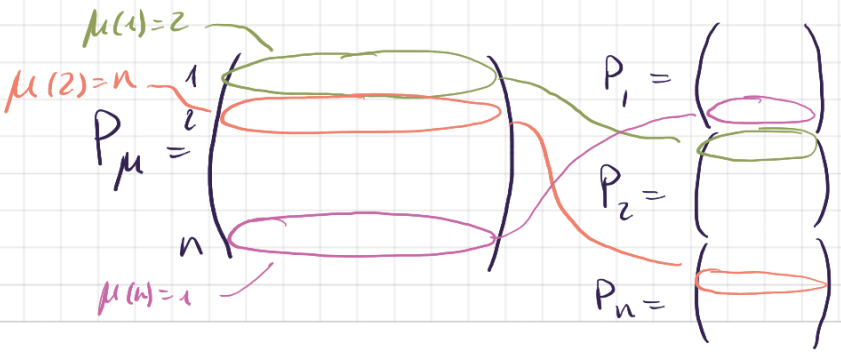
↳ a policy is a function $\mu: S \to A$ that that prescribes which action to take when you are in a given state

↳ Under a given policy the system is again a Markov chain that evolves according to matrix P_μ

↳ IDEA: when you are in state "i" you are going to select the next state based on the probabilities on the i-th row of $P_{\mu(i)}$

i.e.

given $\mu = \begin{matrix} 1 \\ 2 \\ \\ n \end{matrix} \begin{pmatrix} 2 \\ n \\ \vdots \\ 1 \end{pmatrix}$



PERFORMANCE METRICS

↳ two time horizons:

- Infinite time horizon: $\text{opt}\ E\left[\sum_{0}^{+\infty} \gamma^t r(s_t, a_t) \mid s_0\right]$
- Finite time horizon: $\text{opt}\ E\left[\sum_{0}^{T-1} \gamma^t r(s_t, a_t) + \gamma^T R(s_T) \mid s_0\right]$

↳ where γ is the discount factor

↳ it is needed because if a reward B takes more time than reward A, then B should be bigger than A in order to be worth the wait

↳ the discount factor $0 < \gamma < 1$ makes so that the problem is easier to deal with: $B > A$, $\gamma B = A$

- Dynamic Programming -

↳ a technique of solving MDP

↳ it is based on Bellman optimality principle (1953):

"An optimal policy is made of a set of optimal sub policies"

↳ it was applied to the shortest path problem and implied that if $P_{s \to t}$ is optimal, then both $P_{s \to i}$ and $P_{i \to t}$ are optimal

↳ this system was indeed Markovian, since how we get to "i" has no influence on the length of $P_{i \to t}$

DP - DECOMPOSITION

↳ since we talk about DP, we have different notation:

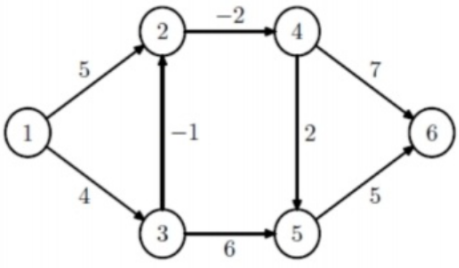
- $X_t = a_t$: action
- $q_t(S_t, x_t, \underbrace{\mathcal{E}_{t+1}}_{\text{parameters}}) = S_t$: state
- $f_t(S_t, x_t) = r_t$: reward

↳ a decomposition is feasible if:

- we assume Markov structure
- objective function takes an additive form

↳ the value function $V(i)$ is the function to be optimized

SHORTEST PATH EXAMPLE



Start = N_1

Destination = N_6 → $V(N_6) = 0$

↳ How do we get to 6?

↳ through N_4 or N_5

$$V(N_4) = \min\left[7 + V(N_6), 2 + V(N_5)\right] = 7$$

$$V(N_5) = \min\left[5 + V(N_6)\right] = 5$$

STOCHASTIC DP FOR FINITE TIME HORIZONS

↳ a solution is given by going backward in time

$$V_T(S_T) = F_T(S_T) \quad \forall S_T \qquad \left(\text{we can assign } V_T(S_T) = 0\right)$$

↳ we can find the value function at each time instant by recursive unfolding:

$$V_{T-1}(S_{T-1}) = \text{opt} \; f_{T-1}(S_{T-1}, X_{T-1}) + \gamma E\left[V_T\left(g_T(S_{T-1}, X_{T-1}, \mathcal{E}_T) \,\middle|\, S_{T-1}, X_{T-1}\right)\right]$$

$$= \dots =$$

$$= \text{opt} \; f_0(S_0, x_0) + \gamma E\left[V_1\left(g_1(S_0, x_0, \mathcal{E}_1) \,\middle|\, S_0, x_0\right)\right]$$

STOCHASTIC DP FOR INFINITE TIME HORIZONS

↳ we focus on THIS type of problems

$$\text{opt}\ E\left[\sum_{t}^{\infty} \gamma^{t} f(s_t, x_t)\right]$$

↳ by assuming stationary model, we drop t and get

$$V(s) = \underset{x \in \chi(s)}{\text{opt}}\ f(s,t) + \gamma E[V(g(s,x,t))]$$

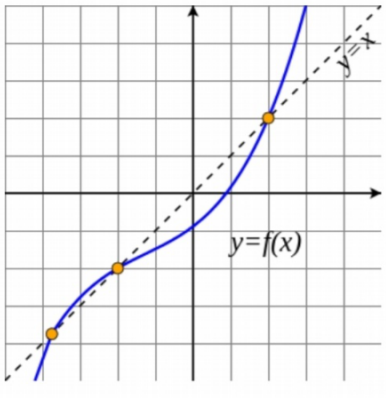
$$= \underset{x \in \chi(s)}{\text{opt}}\ f(s,t) + \gamma \sum_{i \in S} \pi_{i,x,j}\ V(j)$$

← policy matrix as probability matrix

(!) Since $V: S \to \mathbb{R}^n$ and $S = \{1 \dots n\}$ ⟹ V^* is a fixed point

FIXED POINT

↳ fixed point is x such that $F(x) = x$

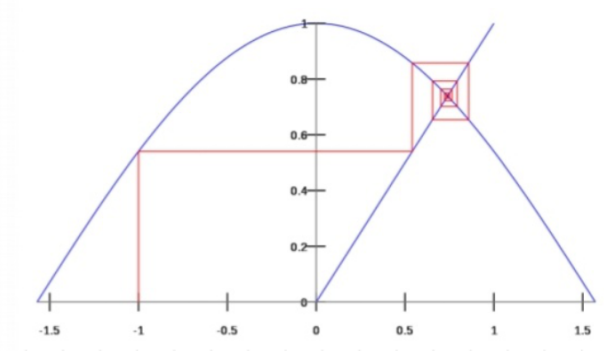


we can use the "fixed point iteration" to find it

$x_0 \to F(x_0) = x_1$

$x_1 \to F(x_1) = x_2$

...



Doesn't always work

↓ BUT

IT WORKS FOR OUR DP ASSUMPTIONS

VALUE ITERATION

Algorithm 1 Value Iteration

1: Start with an initial value function $V^{(0)}$.
2: k=0, stop = False
3: **while** stop **do**
4: → Fixed point iteration

$$V^{(k+1)}(s) = \underset{x \in \mathcal{X}(s)}{\text{opt}} \; f(s,x) + \gamma \sum_{j \in \mathcal{S}} \pi_{s,x,j} V^{(k)}(j) \; \forall \, i \in \mathcal{S}$$

↳ you can take max if

5: **if** $||V^{(k+1)} - V^{(k)}||_\infty < \epsilon$ **then** → end condition
6: stop = True
7: **end if**
8: **end while**

The optimal policy is:
$\mu^*(s) = \arg\text{opt}_{x \in \mathcal{X}(s)} f(s,x) + \gamma \sum_{j \in \mathcal{S}} \pi_{s,x,j} V^{(K)}(j) \; \forall \, i \in \mathcal{S}$ → K is optimal iter.

→ basically fixed point iteration over Value func.
→ each iteration has to solve a stochastic optimization problem

GRID WORLD EXAMPLE

↳ go from (0,0) to (2,2)
avoid (2,1)

2	0	-1	+1
1	0	0	0
0	0	0	0
	0	1	2

→ maximize rewards

①

0	-1	+1
0	0	γ
0	0	0

②

0	-1	+1
0	γ^2	γ
0	0	γ^2

③

0	-1	+1
γ^3	γ^2	γ
0	γ^3	γ^2

④

γ^4	-1	+1
γ^3	γ^2	γ
γ^4	γ^3	γ^2

SOL

γ^4	-1	+1
γ^3	γ^2	γ
γ^4	γ^3	γ^2

① $V_{00} = \max\left[0 + \gamma V_{01},\; 0 + \gamma V_{10}\right] = 0$

$\vdots$

$V_{21} = \max\left[0 + \gamma V_{22},\; 0 + \gamma V_{11},\; 0 + \gamma V_{20}\right] = \max(\gamma, 0, 0) = \gamma$

POLICY ITERATION

Algorithm 2 Policy Iteration

1: Start with an initial policy $\mu^{(0)}$.
2: k=0, stop = False
3: **while** stop **do**
4: evaluate policy $\mu^{(k)}$ by solving: $(I - \gamma\Pi_\mu)V_\mu = f_\mu$ ← find V_μ
5: rollout

$$\mu^{(k+1)}(s) = \arg\operatorname{opt}_{x\in\mathcal{X}(s)} f(s,x) + \gamma \sum_{j\in\mathcal{S}} \pi_{s,\mu(s),j} V_\mu(j), i \in \mathcal{S}$$

↑ rollout

6: **if** $\mu^{(k)} = \mu^{(k+1)}$ **then**
7: stop = True
8: **end if**
9: **end while**

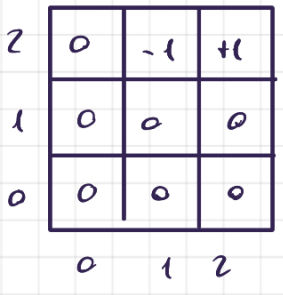
→ to build a new (and better) policy you find the V_{μ_k} (the current policy) then evaluate μ_{k+1} using argopt (procedure called rollout)

↳ we can demonstrate that μ_{k+1} is better than μ_k if μ_k is not optimal (we do NOT demonstrate)

↳ iterations are more expensive (Linear system + Opt > Opt)

GRID WORLD EXAMPLE

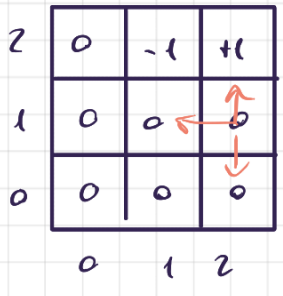
↳ go from (0,0) to (2,2) avoid (2,1)



→ maximize rewards

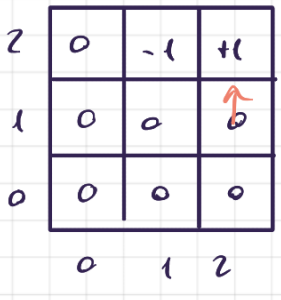
① In the beginning all moves at random

$V(1,2) = 0 + \gamma\left(\frac{1}{3}V_{(2,2)} + \frac{1}{3}V_{(1,1)} + \frac{1}{3}V_{(0,2)}\right)$



② After rollout we get best move

$V(1,2) = 0 + \gamma(1V_{(2,2)} + 0 + 0)$



↳ this for every square each iteration

GENERALIZED POLICY ITERATION

↳ if V_{μ_k} cannot be solved with $(I - \gamma \Pi_\mu) V_\mu = g_\mu$ we can evaluate it with value iteration

↳ we stop value iteration after K iterations then we perform rollout

- ↳ $K = 0$: pure policy iteration
- ↳ $K = K^*$: pure value iteration (to convergence)
- ↳ $0 < K < K^*$: generalized policy iteration

(!) In general this is the approach taken

Reinforcement Learning

Q-FACTORS

↳ in GPI we have to solve $\arg\operatorname{opt}_{x \in \mathcal{X}(s)} f(s,x) + \gamma \sum_{j \in \mathcal{S}} \pi_{s,x,j} V(j)$

but we may not know $\pi_{s,x,j}$ (the probability matrix)

↳ we introduce Q-factors:

$Q(s,x) : \mathcal{S} \times \mathcal{X} \to \mathbb{R}$ ↦ aims at measuring the value of taking action x while in state s

↳ recall that: $V_\mu(s) = f(s, \mu(s)) + \gamma \sum_{j \in \mathcal{S}} \pi_{s,\mu(s),j} V_\mu(j), s \in \mathcal{S}$

while: $Q_\mu(s,x) = f(s,x) + \gamma \sum_{j \in \mathcal{S}} \pi_{s,x,j} V_\mu(j), s \in \mathcal{S}, x \in \mathcal{X}$

↳ at optimality we also have that:

$$Q^*(s,x) = f(s,x) + \gamma \sum_{j \in \mathcal{S}} \pi_{s,x,j} V^*(j), s \in \mathcal{S}, x \in \mathcal{X}$$

$$V^*(s) = \operatorname*{opt}_{x \in \mathcal{X}(s)} Q^*(s,x), s \in \mathcal{S}$$

↳ so we get $Q^*(s,x) = f(s,x) + \gamma \sum_{j \in \mathcal{S}} \pi_{s,x,j} \operatorname*{opt}_{x \in \mathcal{X}(s)} Q^*(s,x), s \in \mathcal{S}, x \in \mathcal{X}$

↳ now we don't need to optimise an expected value but to compute the expected value of an optimized value

↳ thus in our optimization we can concentrate only on the inner part

↳ BUT NO-FREE-LUNCH: now we have values for $|\mathcal{S}| \times |\mathcal{X}|$ and not only $|\mathcal{S}|$

POST-DECISION STATE VARIABLES

↳ by introducing an intermediate state we can simplify our evaluation

↳ from
- $S_t \longrightarrow x_t \xrightarrow{\xi_{t+1}} S_{t+1}$

to
- $S_t \longrightarrow x_t \longrightarrow S_t^x \xrightarrow{\xi_{t+1}} S_{t+1}$

↳ so we "mapped": actions → states
and by doing so we can turn $Q(s,a)$ into $V(s)$
thus reducing our $|S| \times |X|$ problem to $|S|$ problem

↳ it turns our problem into a Markov Reward Process (MRP):

MDP: $S_1\ A_1\ R_1\ S_2\ A_2\ R_2\ S_3\ A_3\ R_3 \ldots$

MRP: $S_1\ R_1\ S_2\ R_2\ S_3\ R_3 \ldots$

(!) It is NOT always applicable

REINFORCEMENT LEARNING

↳ R_t: reward at time t

↳ G_t: cumulative reward

$(G_t = R_{t+1} + \gamma R_{t+2} + \dots)$

- **Monte Carlo for RL**

  ↳ episodic learning

  ↳ run random choices and evaluate their rewards

  ↳ by concluding N episodes we can update Q as:

$$Q_\mu^{(N)} = \frac{1}{N} \sum_i^N G^{(i)}(S,a) \rightarrow \text{mean of cumulative rewards for } (S,a)$$

  ↳ but this way we have to compute first all the N episodes, then learn

  ↳ can we learn at each episode?

$$Q^{(N)}(S,a) = \frac{1}{N} \sum_i^N G^{(i)}(S,a)$$

$$= \frac{1}{N} \left(\sum_{i=1}^{N-1} G^i(S,a) + G^N(S,a) \right)$$

$$= \frac{1}{N} \left[(N-1) Q_\mu^{(N-1)}(S,a) + G^N(S,a) \right]$$

$$= Q^{(N-1)}(S,a) + \frac{1}{N} \left[G^{(N)}(S,a) - Q_\mu^{(N-1)}(S,a) \right]$$

  ↳ Now at each episode we update as

$$NEW_{estimate} = OLD_{estimate} + Step\,Size \left(Target - OLD_{estimate} \right)$$

- **Temporal Difference Learning** [TD(0)]

↳ in MC learning we have to wait untill the end of the episode, which may be very long

MC-Learning: $Q^{(N)}(s,a) = Q^{(N-1)}(s,a) + \frac{1}{N}\left[G^{(N)}(s,a) - Q_{\mu}^{(N-1)}(s,a)\right]$

↑ cumulative reward untill the end of the episode

↳ we need $G^{(N)}(s,a)$ for MC-Learning and it is compitable only at the end of the episode.

↳ CAN WE APPROXIMATE IT?

$$G^{N} \simeq r_{t+1} + \gamma Q_{\mu}^{(N-1)}(s_{t+1}, a_{t+1})$$

they are equal in expectation not in value

↳ thus we get an update equation like:

TD[0] - LEARNING: $Q^{(N)}(s,a) = Q^{(N-1)}(s,a) + \frac{1}{N}\left[r_{t+1} + \gamma Q_{\mu}^{(N-1)}(s_{t+1}, a_{t+1}) - Q_{\mu}^{(N-1)}(s,a)\right]$

new estimate — last estimate

↳ now we can update at EACH VISIT

SARSA

Algorithm 1 SARSA

Choose step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
initialize $Q(s, a) \forall s, a$, arbitrarily except for terminal states
for each episode **do**
  initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  for each step of episode, until s is terminal **do**
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s, a) = Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s = s'$, $a = a'$
  end for
end for

Q-LEARNING

Algorithm 2 Q-Learning

Choose step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
initialize $Q(s, a) \forall s, a$, arbitrarily except for terminal states
for each episode **do**
  initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  for each step of episode, until s is terminal **do**
    Take action a, observe r, s'
    $Q(s, a) = Q(s, a) + \alpha[r + \gamma \text{opt}_{a' \in A(j)} Q(s', a') - Q(s, a)]$
    $s = s'$
  end for
end for

DIFFERENCES BETWEEN SARSA & Q-LEARNING

↳ **Q-Learning**: always evaluates current state as if it will take the BEST actions from now on (according to Q)
  ↳ focuses more on **exploitation**

↳ **SARSA** : it chooses the next action before state evaluation (with ε-greedy, see below) then evaluates current state and forces to take that action
  ↳ focuses more on **exploration**

(!) SARSA will often be considered as "safer" as it eventually considers all next actions, thus:
  ↳ if current state is on the optimal route, but it's next state actions may be risky it will evaluate more negatively

ε-GREEDY

↳ a technique that FORCES EXPLORATION

- given ε
  - $(1-\varepsilon)$ chance to take random action
  - ε chance to take optimal action (according to Q

↳ as $\varepsilon \to 0$, also SARSA → Q-Learning