Bayes Theorem

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- $P(h|D)$: Find best hypothesis given data
- $P(h)$: Prior probability of hypothesis
- $P(D)$: Evidence

D → training data
h → function mappings for target

From candidate hypotheses, iterate all of them with same data & find maximum a posteriori (MAP) hypothesis.

$$h_{MAP} = \text{argmax}\, P(h|D)$$

most probable hypothesis = highest a posteriori

$$= \text{argmax}\, \frac{P(D|h)P(h)}{P(D)} = \text{argmax}\, P(D|h)P(h)$$

- $P(D)$: This doesn't change from one hypothesis to another
- we are left with this!

★ If we don't have any prior or for some cases all $P(h)$ might be equal.

$$h_{ML} = \text{argmax}\, P(D|h) \qquad P(h) = \frac{1}{|H|}$$

↳ max likelihood hypothesis

ex//

priors:
- $P(\text{cancer}) = 0.008$
- $P(\neg\text{cancer}) = 0.992$ (dun)

data (test result):
- $P(+|\text{cancer}) = 0.98$
- $P(-|\text{cancer}) = 0.02$
- $P(+|\neg\text{cancer}) = 0.03$
- $P(-|\neg\text{cancer}) = 0.97$

→ likelihoods

Calculating MAPs:

You had test result (+), do you have cancer?

$$P(\text{cancer}|+) = P(+|\text{cancer})P(\text{cancer}) = 0.0078$$

Probability of not having cancer given your test is (+):

$$P(\neg\text{cancer}|+) = P(+|\neg\text{cancer})P(\neg\text{cancer}) = 0.0298$$

If A is n×n is have there a nonzero vector x such that $Ax = \lambda x$ Ax is a scalar multiple of x

Bayesian Inference

Likelihood: Probability of observing the data that was been observed assuming that the data came from a specific scenario.

ex// $P(\text{test} = \text{positive} \mid \text{disease})$

Prior: Previous knowledge on the subject.

ex// Disease is very rare.

Posterior Probability: What is the probability of you having the disease given that the test is positive.

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}$$

Posterior ← $P(\text{disease} \mid \text{test} = \text{positive})$ = Likelihood $P(\text{test} = \text{positive} \mid \text{disease})$ × Prior $P(\text{disease})$ / Evidence $P(\text{test} = \text{positive})$

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)} \rightarrow P(A \text{ & } B)$$

ex//

$P(\text{disease}) = 0.00148$

$P(\text{test} = (+) \mid \text{disease}) = 0.93$

$P(\text{test} = (-) \mid \text{disease}) = 0.99$

$$P(\text{disease} \mid \text{test} = (+)) = \frac{P(\text{Person has disease \& test } (+))}{P(\text{test } (+))}$$

$$P(\text{disease \& test } (+)) = \underbrace{P(\text{disease})}_{0.00148}\ \underbrace{P(\text{test} = (+) \mid P(\text{disease}))}_{0.93} = 0.0013$$

$$P(A \text{ & } B) = P(A \mid B)\, P(B) = \text{Likelihood} \times \text{Prior}$$

$$P(\text{disease} \mid \text{test } (+)) = \frac{0.0013}{0.011} \approx 0.12 \rightarrow \text{Posterior}$$

Evidence ← 0.011 → $P(\text{test} = \text{positive})$

$= P(\text{disease} \mid (+)) + P(\text{no disease} \mid (+))$

Bayes Updating

What if you take a 2nd test if first one is (+)?

Assume: Correctness of 2nd is not influenced by first.

P(disease) = 0.12 → having disease given that first test = (+)

$$P(\text{disease} \mid 2^{nd}\ \text{test} = (+)) = \frac{P(\text{disease})\,P(2^{nd} = (+) \mid P(\text{disease}))}{P(2^{nd} = (+))}$$

$$= 0.93$$

↳ 2 tests makes it much more probable than one (+) test

Bayes updating → updating a probability based on an event affecting it.

Update prior to get posterior

$$\text{prior} \xrightarrow[\text{Updating}]{\text{Bayesion}} \text{Posterior}$$

Bayesian Inference

Ex// 40 women, 20 asking protection, 20 doesn't.

4 becames pregnant — 16 becames pregnant

20 total pregnancy

4 treatment — 16 control group

p → pregnancy cares from treatment group 0.5

(assume it can take 0.2, 0.5, 0.8)

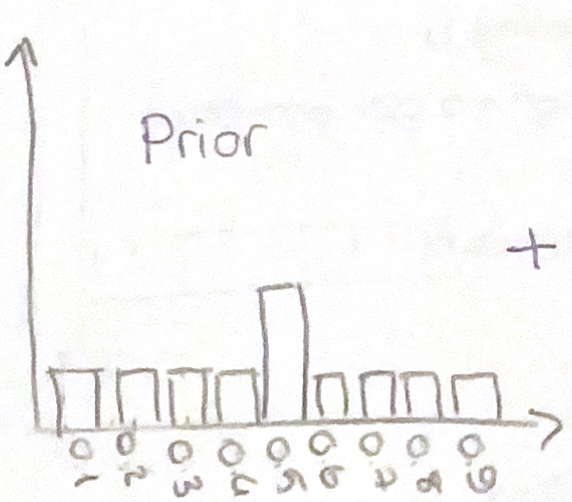
p	0.2	0.5	0.8
prior P(model)	0.06	0.52	0.06
likelihood P(data|model)	0.21	0.004	0.003
likelihood × prior	0.013	0.002	0
posterior P(model | data)	0.42	0.98	0

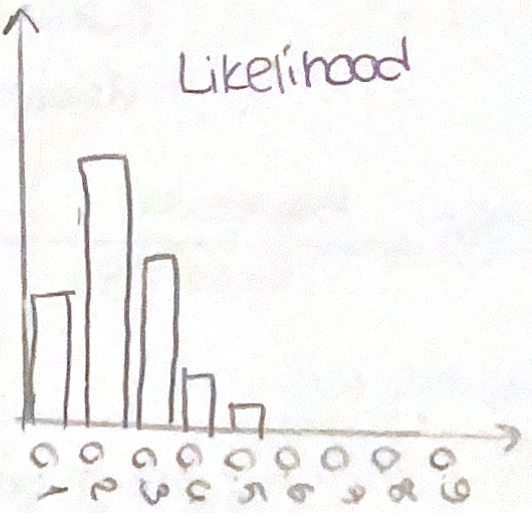
Binomial

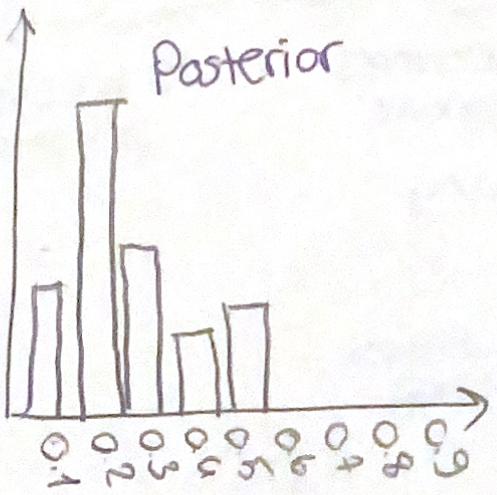
$P(\text{data}|\text{model}) = P(k=4|n=20, p)$

$P(n, p)$

Choose the model with highest posterior







Sample size büyüdükçe likelihood un posterior'a etkisi artar.

Probability formulas

$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$ → Product Rule

$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$ → Posterior probability

$P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$ → Law of Total Probability

Brute Force Max a Posteriori + Concept Learning

Assumptions:

1. Training data is noise free.
2. Target concept c is in the H. (↗ hyp space)
3. Every hypothesis has equal priors

Then: $P(h) = \frac{1}{|H|}$ $P(D|h)$: probability of data D given hypothesis h is 1 if data is consistent with h, 0 otherwise.

$$P(h|D) = \frac{P(D|h)\,P(h)}{P(D)} \quad (P(D|h) = 1 \text{ or } 0;\ P(h) = \frac{1}{|H|}) \rightarrow P(h|D) = \frac{1 \cdot \frac{1}{|H|}}{P(D)} = \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}}$$

$$P(D) = \frac{|VS_{H,D}|}{|H|} \qquad = \frac{1}{|VS_{H,D}|}$$

$|VS_{H,D}|$ → subset of hypotheses from H consistent with data.

$$P(h|D) = \begin{cases} \frac{1}{|VS_{H,D}|} & \text{if } h \text{ is consistent with } D \\ 0 & \text{otherwise} \end{cases}$$

Initially all $h = \frac{1}{|H|}$ as we add data posterior of inconsistent hypotheses become zero while consistent ones have a probability of $\frac{1}{\#\text{consistent ones}}$

Bayes Optimal Classifier

What if we have following hypotheses classifying a new instance as follows:

$h_1 \rightarrow$ classifies x as (+) with 0.4 $\rightarrow$ MAP!

$h_2 \rightarrow$ (−) with 0.3 $\rightarrow$ summing up (−) seems more probable

$h_3 \rightarrow$ (−) with 0.3 $\nearrow$

* If new example can only take on any value v_j from V, then $P(v_j|D)$ – correct classification – is

$$P(v_j|D) = \sum_i P(v_j|h_i)\,P(h_i|D)$$

look for the maximum ↙ ↓ how hypothesis classifies instance in each label & their probabilities ↓ prob of hypothesis given data

$$\text{argmax} \sum_i P(v_j|D)$$

$$= \text{argmax} \sum_i P(v_j|h_i)P(h_i|D)$$

↳ Bayes optimal classification

ex// $P(h_1|D)=0.4 \quad P(-|h_1)=0 \quad P(+|h_1)=1$

$P(h_2|D)=0.3 \quad P(-|h_2)=1 \quad P(+|h_2)=0$

$P(h_3|D)=0.3 \quad P(-|h_3)=1 \quad P(+|h_3)=0$

$\sum_h P(+|h_i)P(h_i|D) = 0.4$

$\sum_h P(-|h_i)P(h_i|D) = 0.6 \rightarrow$ classified as (−)!

* If we're learning boolean concepts we can take weighted vote among all hypotheses in version space with weights = posterior probabilities.

Naive Bayes Classifier

NB classifier applies to learning tasks where each instance x is described by a conjunction of attribute values & where target function $f(x)$ can only take any value from finite set V.

$$v_{MAP} = \text{argmax}\ P(v_j \mid a_1, a_2, \ldots a_n)$$

(v_j: class label; $a_1, a_2, \ldots a_n$: attributes (data))

↓ rewrite as

$$v_{MAP} = \underset{v_j \in V}{\text{argmax}} \frac{P(a_1, a_2, a_3 \ldots a_n \mid v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)}$$

↳ this is same across classes

$$= \underset{v}{\text{argmax}}\ P(a_1, a_2, \ldots, a_n \mid v_j) P(v_j)$$

↳ prob of attribute values given class × prob of class

- $P(v_j)$: class proportions in training data
- Assuming all attributes are independent of each other:

$$P(a_1, a_2, \ldots a_n \mid v_j) = \prod_i P(a_i \mid v_i)$$

↓ plugging these in

$$v_{NB} = \underset{v_j}{\text{argmax}}\ P(v_j) \prod_i P(a_i \mid v_j)$$

} Naive Bayes Classifier

↳ target value output by NB classifier

ex// Instance to predict:

Outlook	Temp	Humid	Wind
Sunny	cool	High	Strong

Target = PlayTennis (yes / no)

1. Estimate $P(v_j)$ from class proportions

$P(\text{PlayTennis} = \text{yes}) = 0.64$

$P(\text{PlayTennis} = \text{No}) = 0.36$

2. Estimate conditional probabilities

$P(\text{Wind} = \text{Strong} \mid \text{PlayTennis} = \text{yes}) = 0.33$

$P(W = \text{strong} \mid PT = \text{no}) = 0.6$

$P(\text{yes} \mid \text{inst}) = P(\text{yes}) \times P(\text{sunny} \mid \text{yes}) \times P(\text{cool} \mid \text{yes}) P(\text{high} \mid \text{yes}) P(\text{str} \mid \text{yes})$

$= 0.0206$

Calculate same for other labels:

$$P(\text{no} \mid \text{inst}) = \underbrace{P(\text{no})}_{P(v_j)} \underbrace{P(\text{sunny} \mid \text{no})\, P(\text{cool} \mid \text{no})\, P(\text{high} \mid \text{no})\, P(\text{Strong} \mid \text{no})}_{\prod_i P(a_i \mid v_j)}$$

$= 0.0206 \rightarrow$ Our prediction (larger than "yes")

Normalizing these: $\frac{0.0206}{0.0206 + 0.0053} = 0.795$

Bayesian Belief Networks

" Conditional independence is overrated "

Naive Bayes → All of the variables are conditionally independent

Belief Nets → Some of the variables are conditionally independent

ex// $P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z = z_k)$ ← if X is independent of Y then ignore Y!

X is conditionally independent of Y given a value of Z

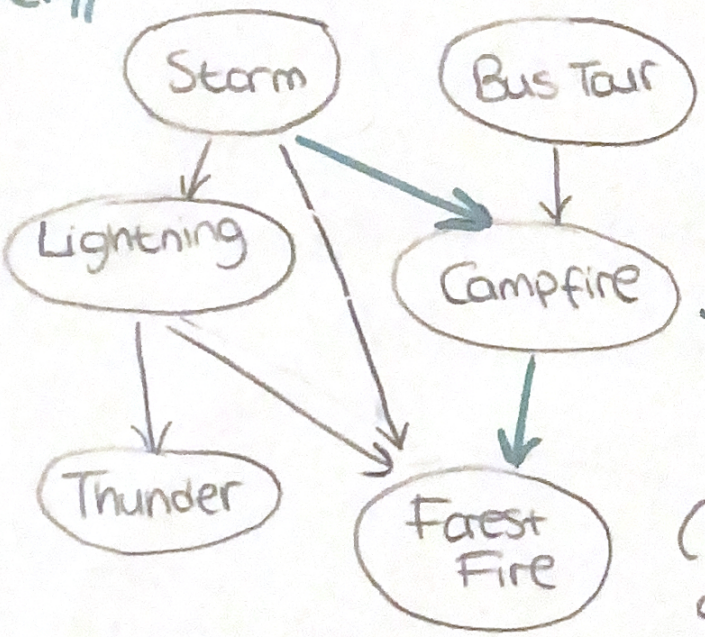
What NB does → $P(A_1, A_2 | V) = P(A_1 | A_2, V) P(A_2 | V)$

features → target

$= P(A_1 | V) P(A_2 | V)$

if all the features are independent, simply multiply all the likelihoods

ex//



What BBN does:

Features are conditionally independent of their non-descendants

(X is descendant of Y if there's directed path from X to Y)

$P(y_1, \dots y_n) = \prod P(y_i | Parents(y_i))$

→ Should be immediate parent (P → C)

	S,B	S,¬B	¬S,B	¬S,¬B
C	0.4	0.1	0.8	0.2
¬C	0.6	0.9	0.2	0.8

$P(C = True | S = True, B = True) = 0.4$

★ $P(T = True | L = True, C = True) = P(T | L)$

$P(L = True | S = True, B = True) = P(L | S)$

↳ Look at immediate parents only!

ex// Thunder is conditionally independent of other variables, given Lightning is True (we calculate Lightning by looking at its parents and go backwards in the graph for all parents)

Bayesian Belief Nets

Conditional independence assumptions may apply to a subset of variables.

ex: $P(X=x_i \mid Y=y_j, Z=z_k) = P(X=x_i \mid Z=z_k)$

↳ we can get rid of this

- Assumption: X is conditionally independent of Y given Z if distribution governing X is independent of value of Y given a value for Z, if above equation holds.

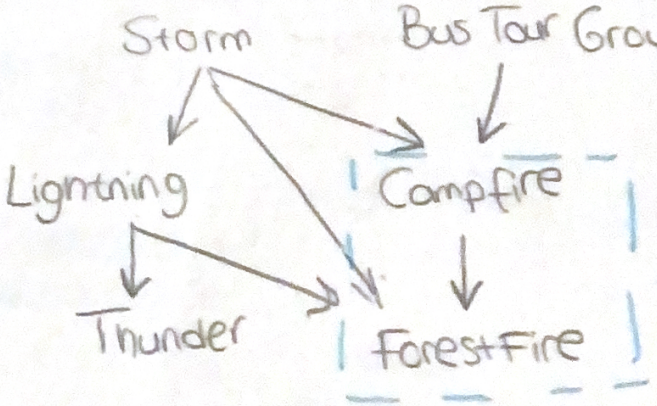
$P(X \mid Y, Z) = P(X \mid Z)$

↳ X ve Y independent'sa Y'den kurtulabilirsin

— Naive Bayesçi independence assumption'ı sayesinde:

$$P(A_1, A_2 \mid V) = P(A_1 \mid A_2, V) P(A_2 \mid V)$$
$$= P(A_1 \mid V) P(A_2 \mid V)$$

A Belief Network represents joint probability distribution for a set of variables



	S,B	S,¬B	¬S,B	¬S,¬B
C	0.4	0.1	0.8	0.2
¬C	0.6	0.9	0.2	0.8

"Forestfire is descendant of campfire"

$$P(y_1, \ldots, y_n) = \prod_{i=1}^{n} P(y_i \mid Parents(y_i))$$

Predecessors of y_i

graph'ta y'den önceki olaylar

Campfire is independent of it's nondescendants Lightning & Thunder.

$P(Campfire = True \mid Storm = True, BusTourG = True) = 0.4$

In a Bayesian Net, we can number each node s.t. parents will have smaller indexes than their children.

$S \rightarrow L \rightarrow T$

- $P(Campfire \mid S, \neg B) = 0.1$
- $P(Thunder, Lightning, Storm) = P(T \mid L)\, P(L \mid S)\, P(S)$
- $P(Campfire) = P(C \mid S, B)\, P(S)\, P(B) + P(C \mid S, \neg B)\, P(S)\, P(\neg B)$

$$+ P(C \mid \neg S, B)\, P(\neg S)\, P(B) + P(C \mid \neg S, \neg B)\, P(\neg S)\, P(\neg B)$$

- $P(B) = \sum_{i=1}^{n} P(B \mid A_i)$ where A_i add up to 1
- $P(C \mid S) = ?$ (we don't know about bustourgroup?)

$$P(C \mid S) = P(C \mid S, B)\, P(B) + P(C \mid S, \neg B)\, P(\neg B)$$ → storm = True, iterate over Bus tourgroup

- If most of the variables are unknown the complexity becomes exponential (2^n)
- We can use monte carlo simulation for this.

ex: $P(S) = 0.9$

$P(B) = 0.5$

$P(C \mid B, S) = 0.4$

$P(C) = ?$

P(S)	P(B)	P(C | S, B)
S → T	B → T	C → F
S → T	B → F	B → F
...	...	...
S → F	B → T	C → T

} Generate cases

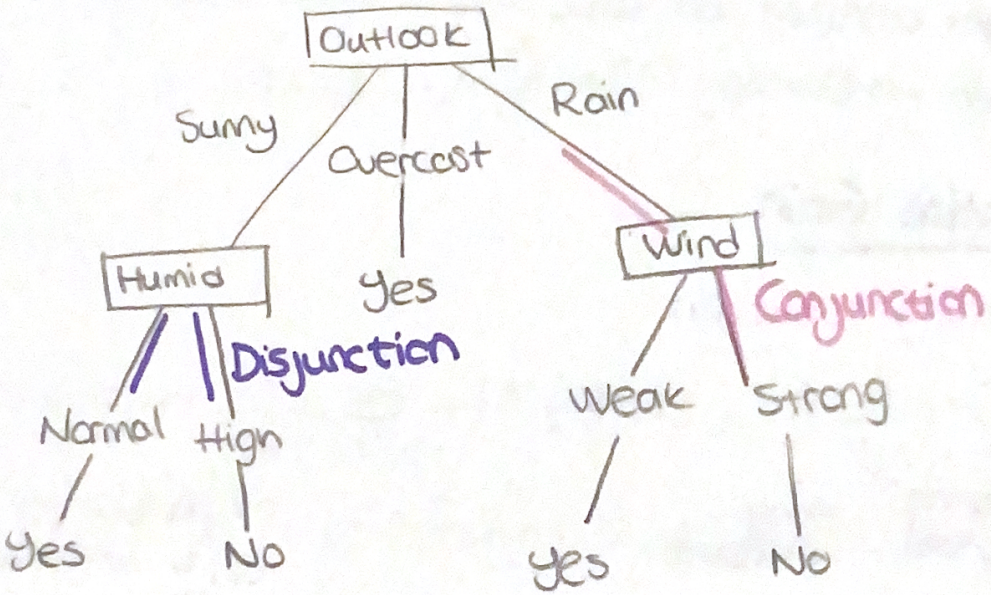
↳ consider these cases

ratio obtained will be approximation to the campfire

Decision Tree Learning

Represents disjunction of conjunction of constraints. on values attributes can get.
(∨) (∧)

(Outlook = Sunny ∧ Humidity = Normal) ∨ (Outlook = Overcast)
∨ (Outlook = Rain ∧ Wind = Weak)



ID3 Algorithm

1. Create a root node for the tree
2. If all examples are (+) return single node tree [+]
3. Do the same if all examples are (-)
4. If attributes are empty return tree root with most common value of target attribute

Otherwise begin

Dataset's entropy − Entropy of attribute

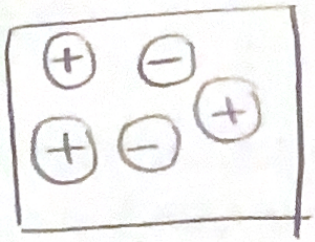
1. Find attribute with best information gain = A
2. Put that attribute to root node (A)
3. For each possible value of A (v_i) do:
    1. Add a new tree branch below root, $A = v_i$
    2. If examples of $A = v_i$ empty, put the most common target label
    3. If examples of $A = v_i$ exist, add the subtree with ID3(Examples v_i, Target attribute, Attributes − A)

↳ recursion

Attribute Selection

Entropy

S



$$\text{Entropy}(S) = -\underbrace{p_+}_{0.6}\log_2 p_+ - \underbrace{p_-}_{0.4}\log_2 p_-$$

$$\rightarrow -0.6\log_2 0.6 - 0.4\log_2 0.4$$

Target attribute can take values more than 2!

Then $\rightarrow \text{Entropy}(S) = \sum_{i=1}^{n} -p_i \log_2 p_i$

Information Gain

For entropy reduction

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v_i} \frac{|S_v|}{|S|}\text{Entropy}(S_v)$$

- S: önceki feature (parent)
- A: aday attribute
- v_i: parent'ta o değeri alan example sayısı / parent size

Choosing attributes

Plurality value → return the most frequent value.

Entropy of dataset → $H(Goal) = B\left(\frac{p}{p+n}\right)$

feature → $\sum_k P(v_k) \log_2 \frac{1}{P(v_k)}$

$= -\sum_k P(v_k) \log_2 P(v_k)$

Information gain → $G(y, x) = E(y) - E(y|x)$ (target, target given x → test for x)

$Remainder(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right)$ (proportion, entropi)

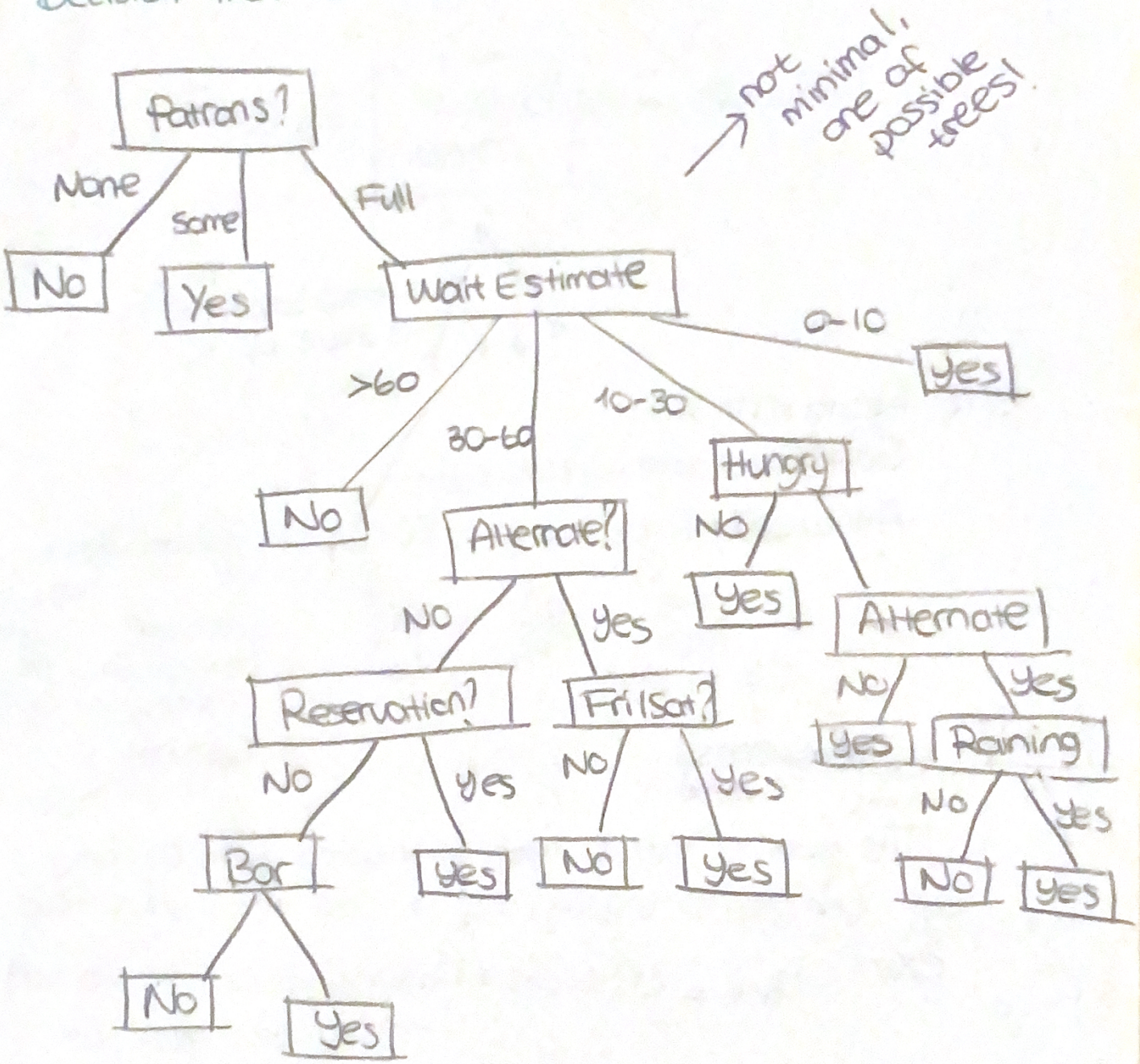
↳ A'nın aldığı değerler

↳ her value için hesaplayıp topla

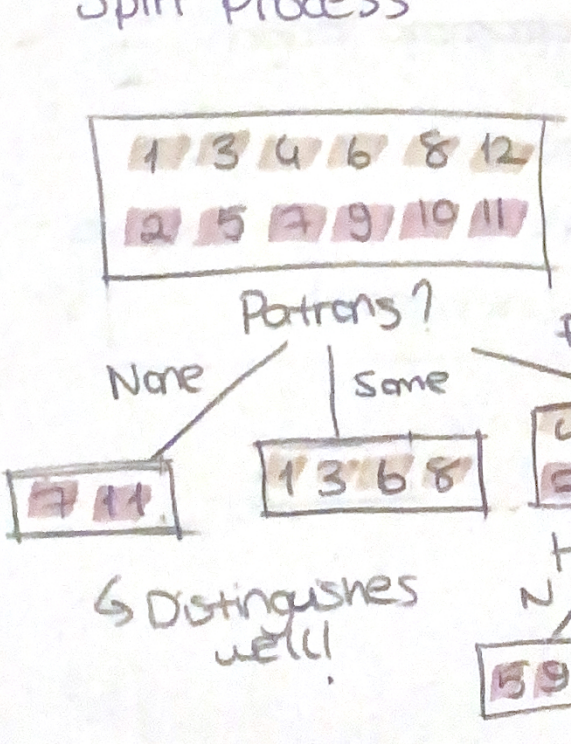
Root node'da → dataset'in entropisi − feature'ın entropisi

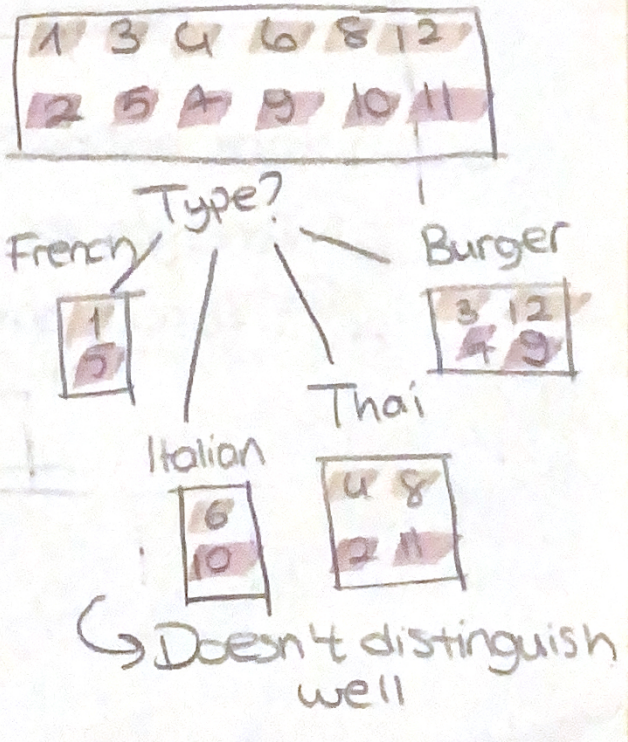
IG = Parent entropy − Feature'ın entropisi

Decision Tree From Examples

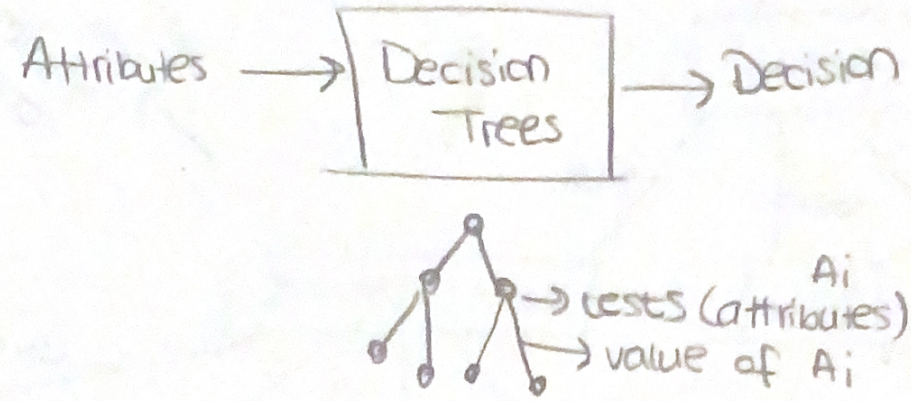


Split Process





Decision Trees



ex:(Restaurant Wait

Goal Predicate → "will wait"

Attributes → Price, Hungry, Fri/Sat, Alternate, Bar..

- Price → price range ($, $$, $$$)
- Hungry → yes/no
- Fri/Sat → true/false
- Alternate → alternate restaurant nearby (yes/no)
- Bar → bar area to wait (yes/no)

Expressiveness

The goal is true if input attributes satisfy one of the paths leading to a leaf with value true.

ex// Path = (Patrons = Full ∧ WaitEstimate = 0-10)

↳ bu true leaf'ine giden path'lerden biri

Goal ↔ (Path1 ∨ Path2 ..)

n tane attribute'ün boolean function sayısı

2^n fonksiyon var. n attribute'lu truth table'da

2^{2^n} farklı fonksiyon var. (intractable)

Concept Learning

Problem of searching through a predefined space of potential hypotheses for the hypothesis that fits the data best

- Each constraint can be
  - a value (water = warm)
  - don't care (water = ?)
  - no value allowed (water = "∅")

? : Feature has no predictive power. (almost redundant)

★ We're looking for a conjunction of constraints that describe (+) examples the best.

★ most general hypothesis → <?, ?, ?, ?, ?, ?>

Every day (instance) is a positive example, can't do classification

Most specific hypothesis → <∅, ∅, ∅, ∅, ∅, ∅>

No day (instance) is positive example

- Try to induce general functions from specific training examples.

Version Spaces & Candidate Elimination Alg.

Candidate Elimination Algorithm: Output a description of the set of all hypotheses consistent with the training examples

Version space: Subset of hypotheses in H that is consistent with training data.

- General boundary G is the set of maximally general members of H consistent with data.
- Specific boundary S is minimally general members of H consistent with data.

Version Space = General boundary + Specific Boundary + Everything in Between

Candidate Elimination Learning Algorithm

1. G ve S'yi tanımla $G_0 = \langle ?, ?, ?, ?, ?, ? \rangle$

$S_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

2. Datayı tara. (for d in D)
   - Eğer d pozitif örnekse:
     - G'nin içinde d ile çelişen hipotezleri sil.
     - S'deki hipoteği minimal şekilde genelleştir.
     - S'de daha genel bir hipotez varsa kaldır.

S'de d ile çelişen hipotezler varsa

     - S ile d çelişmiyorsa güncelleme yapma.
   - Eğer d negatif örnekse
     - S'de çelişen örnekleri sil.
     - G'de d ile çelişen hipotez varsa (g)
       - g'yi sil.
       - G'yi d'ye göre genelle.
       - G'de daha spesifik olan hipotezleri sil

Candidate Elimination Example

Training Examples:

<Sunny, Warm, Normal, Strong, Warm, Same> → yes
<Sunny, Warm, High, Strong, Warm, Same> → yes
<Rainy, Cold, High, Strong, Warm, Change> → No

1. $S_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$
   $G_0 = \langle ?, ?, ?, ?, ?, ? \rangle$
2. Training Ex #1: (+) example, inconsistent with S
   $G_1 = \langle ?, ?, ?, ?, ?, ? \rangle$
   $S_1 = \langle S, W, N, S, W, S \rangle$
3. Ex #2: (+) example, inconsistent with S
   $G_2 = \langle ?, ?, ?, ?, ?, ? \rangle$
   $S_2 = \langle S, W, ?, S, W, S \rangle$
   ↳ generalized
4. Ex #3: (−) example, inconsistent with G
   $S_3 = \langle S, W, ?, S, W, S \rangle$
   $G_3 = \langle Sunny, ?, ?, ?, ?, ? \rangle, \langle ?, W, ?, ?, ?, ? \rangle$
   $\langle ?, ?, ?, ?, ?, S \rangle$ → Alternative maximally general hypothesis
5. Ex #4: $\langle S, W, H, S, C_m, C_m \rangle$ → (+)
   Inconsistent with S
   $S_4 = \langle S, W, ?, S, ?, ? \rangle$ → Son 2 si S_3 ile geliştiyor
   $G_4 = \langle S, ?, ?, ?, ?, ? \rangle, \langle ?, W, ?, ?, ?, ? \rangle$
   ↳ $\langle ?, ?, ?, ?, ?, S \rangle$ yi alcardık sonunda C olduğu için (inconsistent)

Concept Learning

Problem of searching through a predefined space of potential hypotheses for the hypothesis that fits the data best

- Each constraint can be
  - a value (water = warm)
  - don't care (water = ?)
  - no value allowed (water = "∅")

? : Feature has no predictive power. (almost redundant)

★ We're looking for a conjunction of constraints that describe (+) examples the best.

★ most general hypothesis → <?, ?, ?, ?, ?, ?>

Every day (instance) is a positive example, can't do classification

Most specific hypothesis → <∅, ∅, ∅, ∅, ∅, ∅>

No day (instance) is positive example

- Try to induce general functions from specific training examples.

Inductive Bias

Some of the hypotheses are ~~impossible~~ to exist.

ex//

1	S	W	N	S	C	C	(+)
2	C	W	N	S	C	C	(+)
3	R	W	N	S	C	C	(-)

↳ sadece bu farklı

elimizdeki en spec set → <?, W, N, S, C, C>

↳ çok genel ve (-)'i de kapsıyor.

↓

biased learner

An Unbiased Learner

Inductive Bias: Set of all assumptions made by an ML algorithm to perform induction.

↳ generalization

Inductive bias of candidate elimination: target concept c is contained in given hypothesis space H.

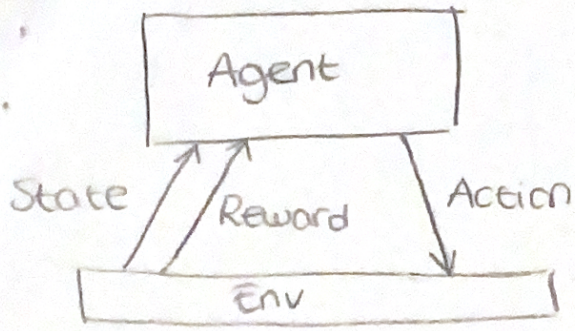
Rote Learner: Store examples, classify x iff it matches previously observed example (training data)

Candidate elimination: Assumes target concept can be explained in hypothesis space.

Find-S: Assumes all instances are negative unless the opposite is given. (Data'daki (+) örnekler dışında hepsi (-))

↓ weakest to strongest bias

Reinforcement Learning



$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} \dots$

Agent chooses action a_i in state s_i and gets reward i. But goal is to maximize:

$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots \quad 0 \leq \gamma < 1$

Reward on the long term

γ: discount factor (a parameter on how much we care about immediate & future rewards)

- Learn a control policy
  $\pi : S \rightarrow A$
  (S: set of states, A: set of actions)
- Take a from A given current state s from S
- Problems
  - Delayed Reward & Temporal Credit Assignment
    Determine which of the actions in its sequence are to be credited for eventual rewards.
  - Exploration vs Exploitation
    Trade-off in choosing exploration of unknown states & actions (more info) or exploitation of states & actions that are known to yield reward.

The Learning Task

Markov Decision Process

$S \rightarrow$ set of states
$A \rightarrow$ set of actions

} At each t_i in s_t, perform a_t, get $r(s_t, a_t)$, produce $s_{t+1} = \delta(s_t, a_t)$

These only depend on current state.

- $\pi : S \rightarrow A$
  learn this
- $V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$
  $= \sum_{i=0}^{\infty} \gamma^i r_{t+i}$
  (V^{π}: cumulative reward)
  $0 \leq \gamma < 1$
  - When $\gamma = 0 \rightarrow$ only immediate reward is considered
  - When $\gamma \hat{\leq} 1 \rightarrow$ Future rewards are more important

Optimal policy is:

$\pi \equiv \operatorname{argmax} V^{\pi}(s)$

pick the policy with the most value

Definitions

Action (A): All the possible moves agent can make.

State (S): Current situation returned by environment

Reward (R): An immediate return sent back from the environment to evaluate last action

Policy (π): Strategy that agent employs to determine next action based on current state.

Value (V): Expected long-term return with discount, as opposed to short term reward.

$V^{\pi}(s)$ → Long term return of current state under policy π

Q-value: $Q^{\pi}(s,a)$ → Long term return of current state s, taking action a under policy π.

Q-Learning

Reward = $Q(s,a)$ → probability of a reward in action-state pair } Q-value

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$

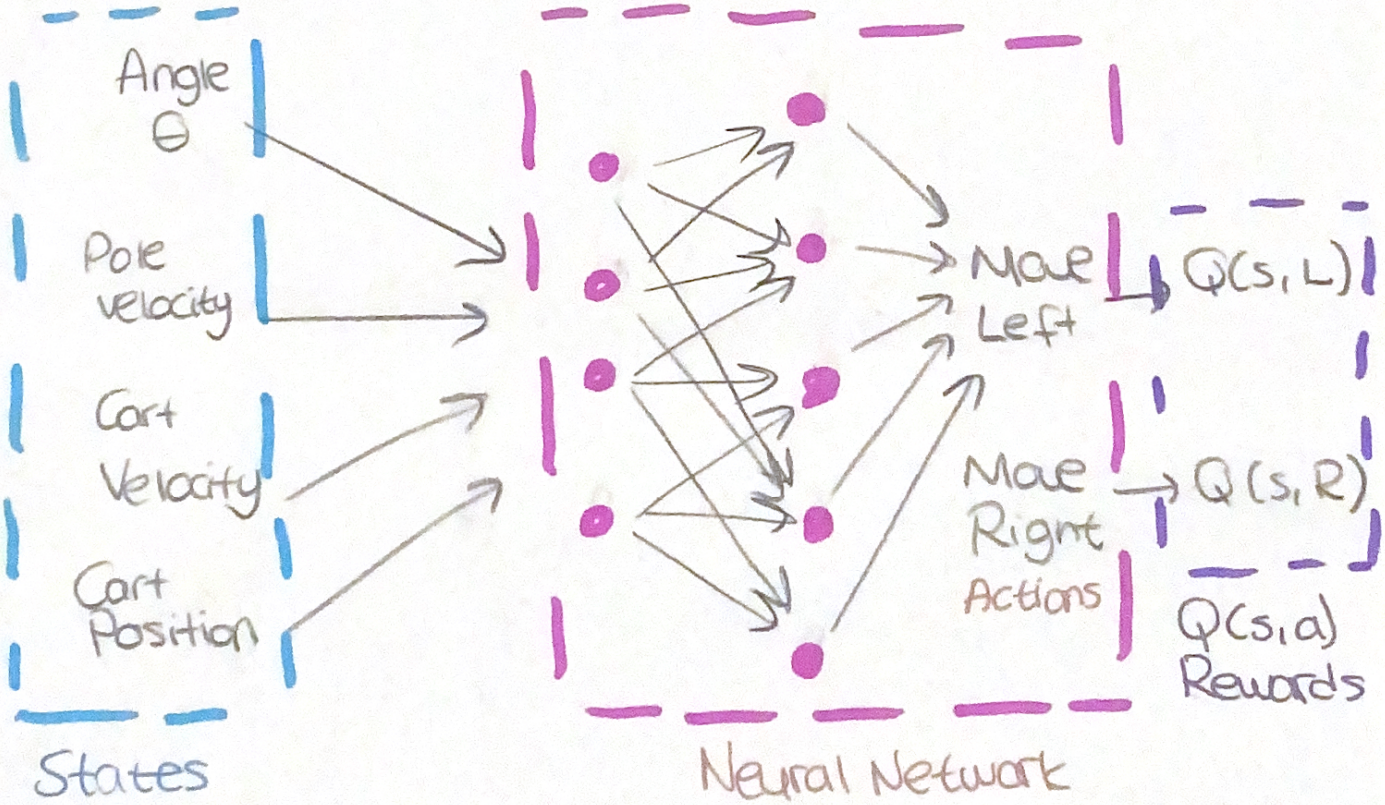
- r: reward by making action a
- $\max_{a'}$: next reward r'
- s': next state
- a': action in next state

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$ ↓ update

Cartpole Example

- Actions → move L & move R (discrete)
- State space → $(\theta, \dot{\theta}, x, \dot{x})$ (continuous)
  - ↳ we must shift state in every slight change in angle, not feasible to hold in Q-table

Deep Q-Network ← Approximate $Q(s,a)$



★ It is essential to have a loss function that minimizes error between approximation and true $Q(s,a)$ calculated from equation

$$(S, A, R, P, \rho)$$

- S → state
- A → action
- R → reward
- P → transition probs
- ρ → initial state distribution

Reward Function → $r_{t+1} = R(s_t, a_t, s_{t+1})$

↳ This formulation is called a Markov Decision Process

MDP: A method to select an action a given state s. Then observe o' & s' based on transition probs P

★ Most DQNs are flat CNNs + batch normalization

★ DQN is the agent's learning in the env.

Markov Decision Processes

Special stochastic time control process for decision making which assumes random probability & a decision maker having complete control.

S: Set of states. At each time step, the state of the environment is an element $s \in S$.

A: Set of actions. At each time step, agent chooses an action $a \in A$ to perform.

$p(s_{t+1} | s_t, a_t)$: State transition model that describes how the env state changes when user performs an action a depending on current state s

$p(r_{t+1} | s_t, a_t)$: Reward model that describes the real valued reward value that the agent receives from the env after performing an action. (Depends on state & action)

Y: Discount factor that controls importance of future rewards.

— In MDP, we search for a policy function that the agent or decision maker will choose in next state s.

(Contd)

- Once we specify policy & fix actions for each state,
- state depends on $\pi(s)$ & $P_r(s_{t+1}=s' \mid s_t=s, a_t=a)$

$\pi(s)$ ↓ policy on state s

$P_r(s_{t+1}=s' \mid s_t=s, a_t=a)$ ↓ transition probability

Reward = total discounted reward

$$R_t = r_t + \gamma r_{t+1} + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

↳ discounted sum of all rewards obtained from time t.

↓ prevents reward from going to infinity

Value Iteration VS Policy Iteration

- Constantly refines value function v (or Q)

Find $Q(s,a)$

$a = \text{argmax}\, Q(s,a)$

- Defines policy function that converges to most optimal policy (through policy gradient)

Find $\pi(s)$

Sample $a \sim \pi(s)$

} both uses MDP

Other RL algorithms

SARSA: (State-Action-Reward-State-Action) uses MDP to adjust value of Q-function based on next state (modified Q-learning that uses extra action & state

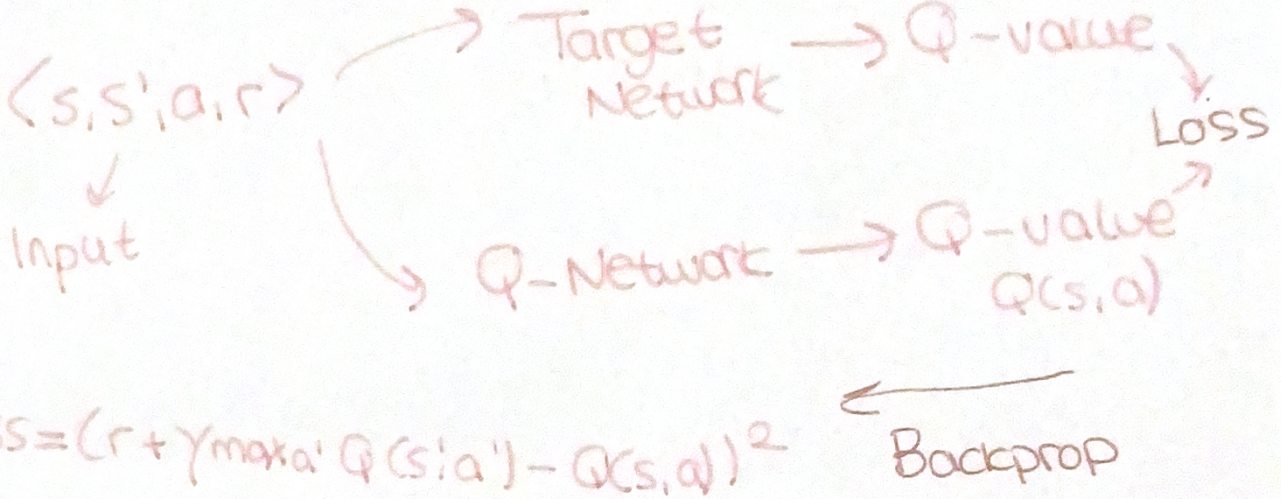
Monte Carlo Methods: Directly learns from experience & past a-s pairs without any prior knowledge of MDP probs. MC uses policy iteration.

Experience Replay

Neural network training process becomes more stable when training is done on random batch of previous experiences

Experience Replay: Memory that stores experiences as a tuple $<s, s', a, r>$

- s → State of agent
- s' → s_{t+1}
- a → Action taken on state s
- r → Immediate reward in s for a



$$Loss = (r + \gamma \max_{a'} Q(s', a') - Q(s, a))^2$$

← Backprop

Deep Q-Learning Steps

1. Provide state of environment to agent. Get Q-values from Target Net & Q-Net.
2. Pick action a, based on epsilon value
3. Perform action a.
4. Observe reward & next state s'. Put them in $<s, s', a, r>$
5. Sample random batches from experience replay memory & perform training on Q-Net.
6. Each n^{th} iteration, copy weights from Q to Target.
7. Repeat 2-7 for each episode.

Episode → all states between initial & terminal states.

Epoch → Forward & backward pass

Genetic Algorithms

- Genetic algorithms are used to exploit a search space for the best solution.

1. Pool of hypotheses is iterated.
2. Every hypothesis is evaluated on a fitness function
3. A new population is generated with most fit individuals from current population. Some of them are passed intact & some of them are used for crossover & mutation.

Basic Algorithm

Chromosomes

Chromosomes are solutions encoded in strings.

ex: 1 0 0 1 1

Crossover

11100101 > 11111101
01011101 > 01000101

Mutation

11100101

↓

turn this into a 0

Selection Strategies

- Roulette wheel

A	B	C	D
0.4	0.3	0.2	0.1

Total Fitness

} Every chromosome has chance of being picked according to it's fitness value

- Tournament Selection

Choose k individuals at random
Then choose the best individual from the pool.

Schema Theorem

Characterizes evolution of population (collectively)

Schemas → 1*0* → 1101
                → 1001
                → 1100
                → 1000

* → don't care

Population Evaluation

Schema Theorem

Given: Schema → 1 * 1 * * (1st, 3rd)

Chromosomes		fitness	
C_1	00101	10	
C_2	11101	25 *	→ fit chromosomes
C_3	00000	15	
C_4	10010	20	↗
C_5	11111	30 *	

Calculate:

- $m(s,t)$ → # of instances of schema s in the population at time t
- $f(h)$ → fitness of hypothesis h
- $\bar{f}(t)$ → average fitness of all individuals in pop at t
- $\hat{u}(s,t)$ → average fitness of instances of s.

- $Pr(h) = \frac{f(h)}{\sum f(h_i)}$ } fitness of h
- $Pr(h \in s) = \sum \frac{f(h)}{n\bar{f}(t)} = \frac{\hat{u}(s,t)}{n\bar{f}(t)} m(s,t)$ (hyp. in schema)
- $\hat{u}(s,t) = \frac{\sum_{h \in s} f(h)}{m(s,t)}$ → schemaya uyanların fitness toplamı; ↳ schemaya uyanların sayısı

prob that we'll pick a hypothesis in the schema

Solution:

1) $\hat{u}(s,t) = \frac{25+30}{2} = 27.5$
2) $\bar{f}(t) = \frac{100}{5} = 20$ — 27.5 > 20 elements of schema expected to increase
   $m(s,t) = 2$
3) $P_c = 1$
   $d(s) = 3-1$
   $L = 5$

4) $E[m(s,t+1)] \geq \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t) \times \left(1 - P_c \frac{d(s)}{L-1}\right) \times (1-P_m)^{\#\text{bits}}$

$E[m(s,t+1)] \geq 1.375$

↳ Expected # is decreasing due to crossover

Genetic Algorithms

Expressing hypotheses as logic functions & genomes

IF $a_1 = T \wedge a_2 = F$ THEN $c = T$

a_1	a_2	c
10	01	1

Modelling Problems & Defining Fitness Functions

A company owns 3 buses & employs 5 drivers. Only one driver can drive any bus on a single day. Each driver cannot work for more than two consecutive days in a week. Company uses buses every day. Make a weekly schedule (7 days) denoting drivers assigned to each of three buses. Provide encoding & fitness function.

Genes (Drivers) → A, B, C, D, E

Occupied drivers for a given day → ABC

Assuming bus order is ignored

ABC
ABD
ABE
ACD
ACE
ADE
BCD
BCE
BDE
CDE

10

ABC CDE ABD ABE CDE ABC CDE

ABC
ABD
⋮
CDE
↳ can be modeled with idleness

ABC → 11100
CDE

Fitness: How many days each crew has left before a day off.

$$F = d_1 + d_2 + \dots + d_m = \sum_{i=1}^{m} d_i$$

d = # Days an employee was before a day off

m = # employees