Probability Stuff

- RV → All Random Variables have domains
- RV is some aspect of the world we are uncertain of
- Distributions must be $\forall x\ P(x) \geq 0$ and $\sum_x P(x) = 1$
- Joint Distributions have >1 RV, like temp and rain
- Probabilistic Inference → Computing a probability from other probabilities (conditional from joint) → New observations change and update belief
- RVs are independent if: $P(x,y) = P(x)P(y)$ $\quad P(x,y) = P(x|y)P(y) = P(y|x)P(x) \rightarrow P(x|y) = \frac{P(y|x)}{P(y)}P(x)$

- A probabilistic model is a joint distribution over a set of random variables
- Probabilistic models:
  - (Random) variables with domains
  - Assignments are called *outcomes*
  - Joint distributions: say whether assignments (outcomes) are likely
  - *Normalized*: sum to 1.0
  - Ideally: only certain variables directly interact
- Constraint satisfaction problems:
  - Variables with domains
  - Constraints: state whether assignments are possible
  - Ideally: only certain variables directly interact

Distribution over T,W

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

Constraint over T,W

T	W	P
hot	sun	T
hot	rain	F
cold	sun	F
cold	rain	T

An event is a set of outcomes

Marginal Distributions eliminate ≥1 RV

$P(a|b) = \frac{P(a,b)}{P(b)}$ Inference is $O(d^n)$ Time and Space

Normalization Trick → Get table for condition

Naive Bayes

$P(\text{class}|\text{features}) = \frac{P(\text{features}|\text{class})}{P(\text{features})} \cdot P(\text{class})$ → Assumes all features are independent of each other

- Bayes Net Models often used for classification → Input and output are both random variables
- Empirical Risk Minimization → Optimization Problem to find best results on test data from training data
- Also concerned about over and underfitting → Aim to reduce variance and limit hypothesis complexity → Relative Frequency Params = overfit
- Parameter Estimation of RVs → Using training data, maximize likelihood of the data (learning!)
- Laplace Smoothing → $P(\text{feature}|\text{class}) = \frac{\text{count}(\text{feature}|\text{class}) + 1}{\text{count}(\text{class}) + n \rightarrow \text{features}}$ → Can also extend to k $\quad P_{LAP}(x|y) = \frac{C(x,y)+k}{C(y)+k(\text{count}(\text{class}))}$

Smooth / Regularize Estimates → Ensure Generalization

Bad when |X| or |Y| are really big

- Tune Parameters on Training Set, tune hyperparameters on different data (held-out) then re-train and test → Choose best value after test
- Baselines help determine how hard a task is and define good accuracy | Confidence of Probabilistic Classifier: $\max_y P(y|x)$ ← not guaranteed
- Calibration → Weak Calibration: High Confidence = High Accuracy ; Strong Calibration: confidence predicts accuracy rate

Linear Classifiers

→ Inputs are feature values, features have weights, Sum is activation = $\sum_i w_i \cdot f_i(x) = w \cdot f(x)$ Dot Product!

- Start with weights = 0
- For each training instance: Binary
  - Classify with current weights

$$y = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$

  - If correct (i.e., y=y*), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

$$w = w + y^* \cdot f$$

Perceptron

- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

$$y = \arg\max_y w_y \cdot f(x)$$

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$

Multi Class

Perceptrons have [separatability], convergence, and a mistake bound

true: if some params get training set perfect

Even the best linear percepts will make mistakes, so go probabilistic

Using Sigmoid $\phi(z) = \frac{1}{1+e^{-z}}$

Best w? Batch GA

- Maximum likelihood estimation:

$$\max_w ll(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)};w)$$

Stochastic vs random → j

we want d·∇CJ

with: $P(y^{(i)} = +1|x^{(i)};w) = \frac{1}{1+e^{-w\cdot f(x^{(i)})}}$

$P(y^{(i)} = -1|x^{(i)};w) = 1 - \frac{1}{1+e^{-w\cdot f(x^{(i)})}}$

= Logistic Regression

Best w? Batch GA

- Maximum likelihood estimation:

$$\max_w ll(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)};w)$$

with: $P(y^{(i)}|x^{(i)};w) = \frac{e^{w_{y^{(i)}}\cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$

= Multi-Class Logistic Regression

- Gradient Ascent → Type of continuous optimization through partial derivatives and go in direction of steepest gradient

Neural Networks

: Directed Graph of neurons; output of layer becomes next input | Composition of weighted sum and sigmoid function

At test time, given x_1 and x_0, compute output. Training → give many x_1 and x_0 with outputs and estimate weights

The neuron is a function that takes a vector of inputs and transforms them into an output

There is always a hidden bias

- We learn the weights through supervised learning | Loss func. $= \frac{1}{2}\sum_j |y_j - f(x_j)|^2$ We want to minimize this
- We can also use gradient descent to minimize loss function → $w_{new} = w_{old} - \alpha \nabla E(w_{old})$, where α = learning rate

The complete recipe for training a single sigmoid neuron

- Start with some initial weights $w = [w_0, w_1, \ldots]$
- *Repeat until convergence*
  - For each input x, use the current weights w to compute the output of the neuron $f_w(x)$ (call it o)
  - Compute the loss $E(w)$ between $f_w(x)$ and y
  - Compute the gradient of the loss wrt each w_i

$$\frac{\partial E}{\partial w_i} = -\sum_j (y^j - o^j) o^j (1 - o^j) x_i^j$$

  - Update each weight w_i based on the gradient

$$w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}$$

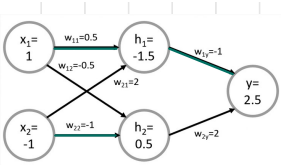
Same idea as single neuron

- Start with some initial weights $w = [w_0, w_1, \ldots]$
- *Repeat until convergence*
  - For each input x, use the current weights w to compute the output of the network $f_w(x)$
  - Compute the loss $E(w)$ between $f_w(x)$ and y
  - Compute the gradient of the loss wrt each w_i
    - slightly more complicated equation
  - Update each weight w_i based on the gradient

$$w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}$$

Gradient of a weight *w* depends only on

- output of the neuron it is connected to
- gradient from a neuron in the next layer
- weight of the edge connected these two neurons



GD will try to increase highlighted weights if desired output is 0

- Only unknowns are the weights. Individual needs to design # of layers, neurons / layer, neuron connections, activation function, loss function, and what GD method to use + learning hyperparameter(s)

Bayes Nets → A technique for describing complex joint distributions using simple, local distributions (Graphical Models)

·Unconditional independence is very rare, conditional independence is our most basic form of knowledge on uncertain environments

$X \perp\!\!\!\perp Y \mid Z$ iff $\forall x,y,z : P(x, y \mid z) = P(x \mid z)\, P(y \mid z)$ | iff $P(x \mid z, y) = P(x \mid z)$ | ·Bayes Net nodes can be assigned or unassigned

·If there are no interactions between variables → Absolute independence | Chain Rule: $P(X_1, X_2, \ldots X_n) = P(X_1)\, P(X_2 \mid X_1)\, P(X_3 \mid X_1, X_2)$

Bayes' Net Semantics

- A set of nodes, one per variable X
- A directed, acyclic graph
- A conditional distribution for each node
  - A collection of distributions over X, one for each combination of parents' values

$$P(X \mid a_1 \ldots a_n)$$

  - CPT: conditional probability table
  - Description of a noisy "causal" process

$P(X \mid A_1 \ldots A_n)$

A Bayes net = Topology (graph) + Local Conditional Probabilities

Probabilities in BNs

- Bayes' nets implicitly encode joint distributions
  - As a product of local conditional distributions
  - To see what probability a BN gives to a full assignment, multiply all the relevant conditionals together:

$$P(x_1, x_2, \ldots x_n) = \prod_{i=1}^{n} P(x_i \mid parents(X_i))$$

- Example:

$P(+cavity, +catch, -toothache)$

Probabilities in BNs

- Why are we guaranteed that setting

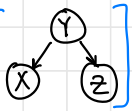
$$P(x_1, x_2, \ldots x_n) = \prod_{i=1}^{n} P(x_i \mid parents(X_i))$$

results in a proper joint distribution?

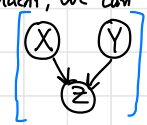
- Chain rule (valid for all distributions): $P(x_1, x_2, \ldots x_n) = \prod_{i=1}^{n} P(x_i \mid x_1 \ldots x_{i-1})$
- Assume conditional independences: $P(x_i \mid x_1, \ldots x_{i-1}) = P(x_i \mid parents(X_i))$
  - → Consequence: $P(x_1, x_2, \ldots x_n) = \prod_{i=1}^{n} P(x_i \mid parents(X_i))$
- Not every BN can represent every joint distribution
  - The topology enforces certain conditional independencies

·When BNs reflect true causal patterns, they are simpler, easier to think about | Topology of BN encodes conditional indep.

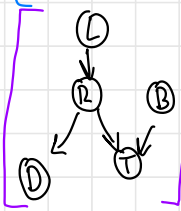
[X → Y → Z] X and Z are not necessarily independent, we can use D-separation | Given Y, X is independent of Z



Once again, X is not guaranteed to be independent of Z unless we are given Y (Common Cause)



X and Y are independent, but if given Z, they are not (Common Effect)



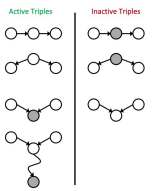
Shaded nodes are evidence nodes

If two nodes are connected by an undirected path are not blocked by a shaded node, they are conditionally independent

Given a BN, run the D-separation algorithm to build a list of conditional independences → Set of distributions represented

Active / Inactive Paths

- Question: Are X and Y conditionally independent given evidence variables {Z}?
  - Yes, if X and Y "d-separated" by Z
  - Consider all (undirected) paths from X to Y
  - No active paths = independence!
- A path is active if each triple is active:
  - Causal chain A → B → C where B is unobserved (either direction)
  - Common cause A ← B → C where B is unobserved
  - Common effect (aka v-structure) A → B ← C where B *or one of its descendents* is observed
- All it takes to block a path is a single inactive segment



Multiple Elimination

$P(R, T, L)$

R	T	L	
+r	+t	+l	0.024
+r	+t	-l	0.056
+r	-t	+l	0.002
+r	-t	-l	0.018
-r	+t	+l	0.027
-r	+t	-l	0.063
-r	-t	+l	0.081
-r	-t	-l	0.729

Sum out R →

$P(T, L)$

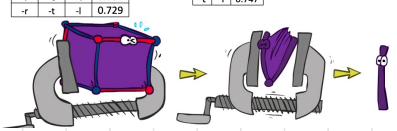
T	L	
+t	+l	0.051
+t	-l	0.119
-t	+l	0.083
-t	-l	0.747

Sum out T →

$P(L)$

L	
+l	0.134
-l	0.866

Bad



Marginalizing Early! (aka VE)

$P(R)$

R	
+r	0.1
-r	0.9

Join R →

$P(R, T)$

R	T	
+r	+t	0.08
+r	-t	0.02
-r	+t	0.09
-r	-t	0.81

Sum out R →

$P(T)$

T	
+t	0.17
-t	0.83

Join T →

Sum out T →

Good

$P(T \mid R)$

R	T	
+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(L \mid T)$

T	L	
+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

$P(L \mid T)$

T	L	
+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

$P(L \mid T)$

T	L	
+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

$P(T, L)$

T	L	
+t	+l	0.051
+t	-l	0.119
-t	+l	0.083
-t	-l	0.747

$P(L)$

L	
+l	0.134
-l	0.866

Evidence

- If evidence, start with factors that select that evidence
  - No evidence uses these initial factors:

$P(R)$

R	
+r	0.1
-r	0.9

$P(T \mid R)$

R	T	
+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(L \mid T)$

T	L	
+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

  - Computing $P(L \mid +r)$ the initial factors become:

$P(+r)$

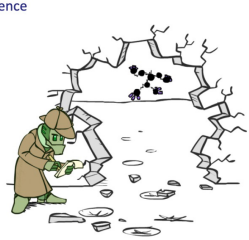
R	
+r	0.1

$P(T \mid +r)$

R	T	
+r	+t	0.8
+r	-t	0.2

$P(L \mid T)$

T	L	
+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9



- We eliminate all vars other than query + evidence

C	P(C)
red	0.6
green	0.1
blue	0.3

From uniform distr

Prior Sampling

Call rand() $^{n \text{ times}}$ on:

0 ≤ rand ≤ 0.6 → R

0.6 < rand ≤ 0.7 → G

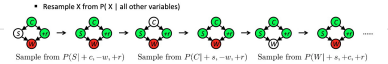
0.7 < rand ≤ 1.0 → B

Rejection Sampling: ·Reject / Throw out samples / stop compute when a condition is not met. If P(C|ts), stop early if we get -s.

Likelihood weighting → Only generate samples with a certain condition by weighting consistencies with evidence higher than things that are inconsistent, calculate weighted average

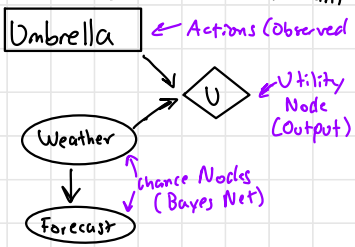
Gibbs Sampling Example: P(S | +r)

- Step 1: Fix evidence
  - R = +r
- Step 2: Initialize other variables
  - Randomly
- Steps 3: Repeat
  - Choose a non-evidence variable X
  - Resample X from P(X | all other variables)



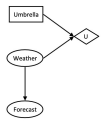
Sample from $P(S|+c,-w,+r)$ Sample from $P(C|+s,-w,+r)$ Sample from $P(W|+s,+c,+r)$

Decision Networks → We want to get maximum expected utility (MEU)



Decision Networks

- Action selection
  - Instantiate all evidence
  - Set action node(s) each possible way
  - Calculate posterior for all parents of utility node, given the evidence
  - Calculate expected utility for each action
  - Choose maximizing action



7

Decision Networks

Umbrella = leave

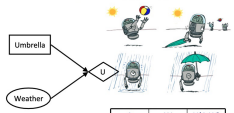
$$EU(\text{leave}) = \sum_w P(w)U(\text{leave}, w)$$
$$= 0.7 \cdot 100 + 0.3 \cdot 0 = 70$$

Umbrella = take

$$EU(\text{take}) = \sum_w P(w)U(\text{take}, w)$$
$$= 0.7 \cdot 20 + 0.3 \cdot 70 = 35$$

Optimal decision = leave

$$MEU(\varnothing) = \max_a EU(a) = 70$$



W	P(W)
sun	0.7
rain	0.3

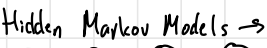
A	W	U(A,W)
leave	sun	100
leave	rain	0
take	sun	20
take	rain	70

8

Important to understand the value of acquiring evidence. After gaining information, we can calculate the gain in utility. Find the MEU given info, then subtract original MEU to see the gain. (Cost analysis comes from this.

- No such thing as a noisy variable

Hidden Markov Models → A markov chain over states X, while observing outputs at each step



Parameters (transition probabilities) specify how the state evolves over time

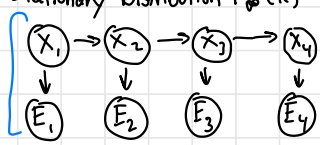
$P(X_1)$ and $P(X_t | X_{t-1})$ | First order Markov Property → Each time step only depends on the previous one

Forward Sim.

$$P(x_t) = \sum_{x_{t-1}} P(x_t | x_{t-1}) P(x_{t-1})$$

The chain is just a Bayes Net and we can use BN properties if we truncate at a certain length

Stationary Distribution $P_\infty(X) = P_{\infty+1}(X) = \sum_i P(X|i)\, P_\infty(x)$



Hidden Markov Model Defined by Initial Distribution $P(X_1)$, transitions $P(X_t | X_{t-1})$ and emissions $P(E_t | X_t)$. HMM future states depend on the past and Current observation is independent of everything given state

Observation

- Assume we have current belief P(X | previous evidence):

$$B'(X_{t+1}) = P(X_{t+1}|e_{1:t})$$

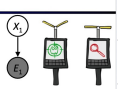
- Then, after evidence comes in:

$$P(X_{t+1}|e_{1:t+1}) = P(X_{t+1}, e_{t+1}|e_{1:t})/P(e_{t+1}|e_{1:t})$$
$$\propto_{X_{t+1}} P(X_{t+1}, e_{t+1}|e_{1:t})$$
$$= P(e_{t+1}|e_{1:t}, X_{t+1})P(X_{t+1}|e_{1:t})$$
$$= P(e_{t+1}|X_{t+1})P(X_{t+1}|e_{1:t})$$

- Or, compactly:

$$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1})B'(X_{t+1})$$

- Basic idea: beliefs "reweighted" by likelihood of evidence
- Unlike passage of time, we have to renormalize



Online Belief Updates

- Every time step, we start with current P(X | evidence)
- We update for time:

$$P(x_t|e_{1:t-1}) = \sum_{x_{t-1}} P(x_{t-1}|e_{1:t-1}) \cdot P(x_t|x_{t-1})$$

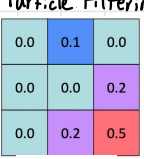
- We update for evidence:

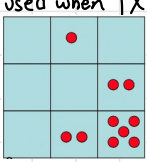
$$P(x_t|e_{1:t}) \propto_X P(x_t|e_{1:t-1}) \cdot P(e_t|x_t)$$

- The forward algorithm does both at once (and doesn't normalize)

43

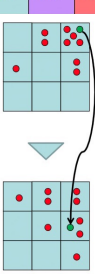
Particle Filtering used when |X| is too big for exact inference, so we take an approximate inference (Particle = Sample)





Instead of a massive state space, budget some # of particles and store their locations

To get probability distribution iterate over non-empty states. | Choose P << |X|

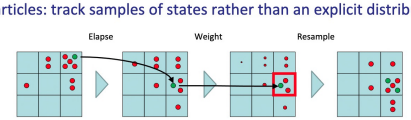


We can track the movement of the particle by sampling P(X'|x) → x'. If we have enough particles, the distribution should match almost exact

Weights picked by product of likelihood of the evidence given the state represented by the particle and ancestor weight

Recap: Particle Filtering

- Particles: track samples of states rather than an explicit distribution



8

[Demos: ghostbusters particle filtering (L15D3,4,5)]

Search Problems have
- State Space
- Successor Function
- Start State and goal test
- Solution is a set of steps to go from Start → Goal

State Space Graph
- Nodes are world states
- Arcs are successors / transitions

- Search Trees are path representations from State Space Graphs

Depth First Search
- LIFO Stack implementation fringe
- Time: If depth is finite, $O(b^m)$
- Space: $O(bm)$
- Complete only if we prevent cycles
- Not optimal, only finds leftmost solution

Breadth First Search
- FIFO Queue Fringe
- Process all nodes above shallowest solution s
- Time and Space: $O(b^s)$
- Complete because solution exists
- Optimal only if costs are all equal

Iterative Deepening combines DFS space with BFS time

Uniform Cost Search
- Expand Cheapest Node First
- Fringe is a priority queue based on cumulative cost
- Processes all nodes with cost less than cheapest solution
- T + S: $O(b^{C/\varepsilon})$ → C = solution cost, ε = minimum arc cost
- Complete and Optimal
- Explores in all directions, no information about goal location

Heuristic Estimates how close a state is to the goal
- Specific to problem

An arc is consistent if for all vars in the tail domain, there is a value in the head domain that doesn't violate any constraints

Greedy Search
- Expand the node closest to goal per heuristic

A* Search (Optimal)
- Combination of UCS and Greedy
- UCS does backward costs, greedy does forward
  - A* is the sum of these
- Stop when goal dequeue
- Mainly expands towards the goal, but hedges bets to ensure optimality

Heuristic Admissibility
- Admissible if heuristic value is ≤ true cost

Graph Search
- Idea is to never expand a state more than once

Estimated heuristic costs should always ≤ true cost

Tree Search
- A* Optimal if heuristics are admissible, UCS is the case h=0

Graph Search **Consistency = Admissibility**
- A* Optimal if heuristic is consistent, UCS optimal (h=0)

Constraint Satisfaction Problems
- Search Problem Subset
- States with variables, domain, goal test checks for variable combinations
- Constraint Graph Nodes = Vars; Arcs = Limits

Backtracking Search
- One var at a time, check cons. as you go
- Filtering gets rid of bad domains via forward checking

Arc Consistency is a good way too

Least Con. Val. & Min. Rem. Values