Artificial Intelligence II

Dr Sean Holden

Computer Laboratory, Room FC06

Telephone extension 63725

Email: sbh11@cl.cam.ac.uk

www.cl.cam.ac.uk/∼sbh11/

Syllabus part I: advanced planning

New things to be looked at include some more advanced material on *planning algorithms:*

- *Heuristics and GraphPlan:* incorporating heuristics into partial-order planning, planning graphs, the GraphPlan algorithm. [1 lecture]
- *Planning using propositional logic:* representing planning problems using propositional logic, and generating plans using satisfiability solvers. [1 lecture]
- *Planning using constraint satisfaction:* representing planning problems so that they can be solved using constraint satisfaction solvers. [1 lecture]

There is no warranty attached to the stated lecture timings.

Syllabus part II: uncertainty in AI

We then delve into some more modern material which takes account of *uncertainty:*

- *Uncertainty and Bayesian networks:* review of probability as applied to AI, Bayesian networks, inference in Bayesian networks using both exact and approximate techniques, other ways of dealing with uncertainty. [4 lectures]
- *Utility and decision-making:* maximising expected utility, decision networks, the value of information. [1 lecture]

Please read the *supplementary notes on probability* handout.

Syllabus part III: uncertainty and time

We then look at how uncertain reasoning and learning can take place when *time* is to be taken into account:

- *Markov processes:* transition and sensor models.
- *Inference* in temporal models: filtering, prediction, smoothing and finding the most likely explanation.
- *Hidden Markov models*. [2 lectures]

Syllabus part IV: learning

Finally, we apply probability to *supervised learning* to obtain [1 lecture] more sophisticated models of learning.

- *Bayes theorem* as applied to supervised learning. [1 lecture]
- The *maximum likelihood* and *maximum a posteriori* hypotheses. [1 lecture]
- Applying the Bayesian approach to *neural networks*. [3 lectures]

We finish the course by taking a brief look at *reinforcement learning*.

- How can we learn from *rewards and punishments*?
- The $\mathcal{Q}$*-learning* algorithm. [1 lecture]

Reinforcement learning can be thought of as combining many of the elements covered in this course and in AI I, and thus provides a natural place to stop.

Books

Once again, the main single text book for the course is:

- *Artificial Intelligence: A Modern Approach*. Stuart Russell and Peter Norvig, Prentice Hall.

There is an accompanying web site at

`aima.cs.berkeley.edu`

Either the second or third edition should be fine, but avoid the first edition as it does not fit this course so well.

Chapter numbers given in these notes refer to the third edition.

Books

For some of the new material on neural networks you might also like to take a look at:

- *Pattern Recognition and Machine Learning*. Christopher M. Bishop. Springer, 2006.

For some of the new material on reinforcement learning you might like to consult:

- *Machine Learning*. Tom Mitchell. McGraw Hill, 1997.

For further material on planning try:

- *Automated Planning: Theory and Practice*. Malik Ghallab, Dana Nau and Paolo Traverso. Morgan Kaufmann, 2004.

Dire Warning

DIRE WARNING

This course contains quite a lot of:

1. Probability
2. Matrix algebra
3. Calculus

As I am an evil and vindictive person who likes to be unkind to kittens I will assume that you know everything on these subjects that was covered in earlier courses.

If you don't it is *essential* that you re-visit your old notes and make sure that you're at home with that material.

YOU HAVE BEEN WARNED

How's your maths?

To see if you're up to speed on the maths, have a go at the following:

Evaluate the integral

$$\int_{-\infty}^{\infty} \exp(-x^2)\, dx$$

Hint: this is a pretty standard result. Square the integral and change to polar coordinates.

How's your maths?

Following on from that, here's something a bit more challenging.

Evaluate the integral

$$\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \exp\left(-\frac{1}{2}\left(\mathbf{x}^T\boldsymbol{\Sigma}\mathbf{x} + \mathbf{x}^T\boldsymbol{\alpha} + \beta\right)\right)\, dx_1 \cdots dx_n$$

where $\boldsymbol{\Sigma}$ is a symmetric $n \times n$ matrix with real elements, $\boldsymbol{\alpha} \in \mathbb{R}^n$, $\beta \in \mathbb{R}$ and

$$\mathbf{x}^T = \left[\, x_1 \;\; x_2 \;\; \cdots \;\; x_n \,\right] \in \mathbb{R}^n$$

(This second one is a bit tricky. I'll show you the answer later...)

Planning II

We now examine:

- The way in which *basic heuristics* might be defined for use in planning problems.
- The construction of *planning graphs* and their use in obtaining more sensible heuristics.
- Planning graphs as the basis of the *GraphPlan* algorithm.
- Planning using *propositional logic*.
- Planning using *constraint satisfaction*.

Reading: Russell and Norvig, relevant sections of chapter 11.

A quick review

We used the following simple example problem.

The intrepid little scamps in the *Cambridge University Roof-Climbing Society* wish to attach an inflatable gorilla to the spire of a famous College. To do this they need to leave home and obtain:

- An *inflatable gorilla*: these can be purchased from all good joke shops.
- Some *rope*: available from a hardware store.
- A *first-aid kit*: also available from a hardware store.

They need to return home after they've finished their shopping.

How do they go about planning their jolly escapade?

The STRIPS language

STRIPS: "Stanford Research Institute Problem Solver" (1970).

States: are *conjunctions* of *ground literals* with *no functions*.

$$\texttt{At(Home)} \wedge \neg\texttt{Have(Gorilla)} \wedge \neg\texttt{Have(Rope)} \wedge \neg\texttt{Have(Kit)}$$

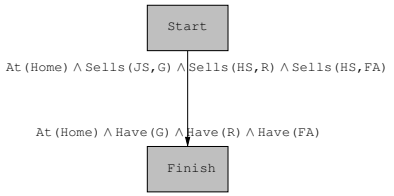
Goals: are *conjunctions* of *literals* where variables are assumed existentially quantified.

$$\texttt{At}(x) \wedge \texttt{Sells}(x, \texttt{Gorilla})$$

A planner finds a sequence of actions that makes the goal true when performed.

An example of partial-order planning

Here is the initial plan:

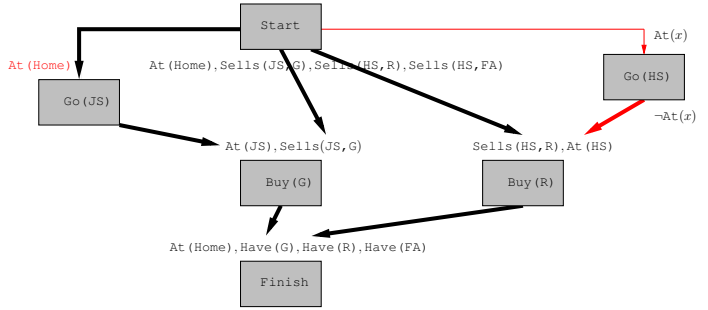


Thin arrows denote ordering.

An example of partial-order planning

There are two actions available:

An example of partial-order planning

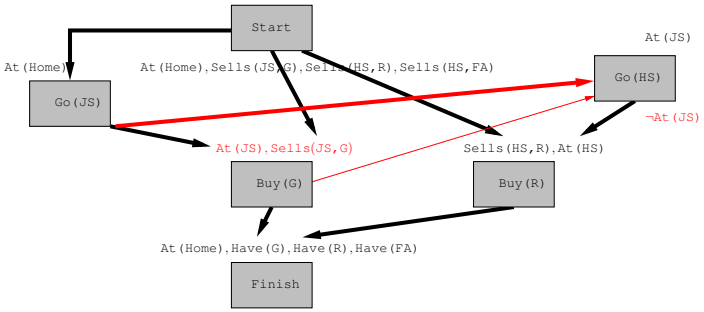


The `At(HS)` precondition is easy to achieve.

But if we introduce a causal link from `Start` *to* `Go(HS)` *then we risk invalidating the precondition for* `Go(JS)`.

An example of partial-order planning

The planner could backtrack and try to achieve the $\texttt{At}(x)$ precondition using the existing `Go(JS)` step.



This involves a threat, but one that can be fixed using promotion.

Using heuristics in planning

We found in looking at search problems that *heuristics* were a helpful thing to have.

Note that now:

- There is no simple representation of a *state*.
- Consequently it is harder to measure the distance to a *goal*.

Defining heuristics for planning is therefore more difficult than it was for search problems.

Using heuristics in planning

We can quickly suggest some possibilities.

For example

$$h = \text{number of unsatisfied preconditions}$$

or

$$h = \text{number of unsatisfied preconditions} - \text{number satisfied by the start state}$$

These can lead to underestimates or overestimates:

- Underestimates if actions can affect one another in undesirable ways.
- Overestimates if actions achieve many preconditions.

Using heuristics in planning

We can go a little further by learning from *Constraint Satisfaction Problems* and adopting the *most constrained variable* heuristic:

- Prefer the precondition satisfiable in the smallest number of ways.

This can be computationally demanding but two special cases are helpful:

- Choose preconditions for which no action will satisfy them.
- Choose preconditions that can only be satisfied in one way.

Planning graphs

Planning graphs can be used:

- To compute more sensible heuristics.
- To generate entire plans.

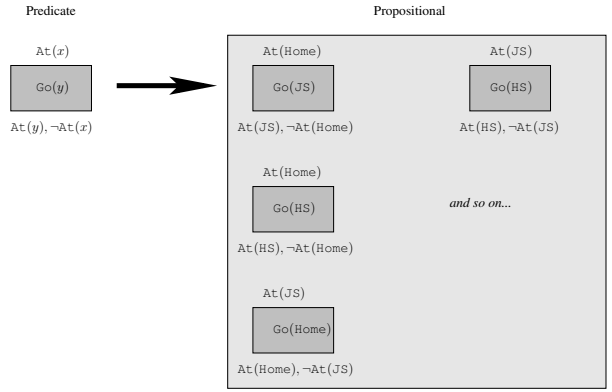
Also, planning graphs are *easy to construct*.

They apply only when it is possible to work entirely using *propositional* representations of plans.

Luckily, STRIPS can always be propositionalized...

Planning graphs

For example: the triumphant return of the gorilla-purchasing roof-climbers...



Planning graphs

A planning graph is constructed in levels:

- Level 0 corresponds to the *start state*.
- At each level we keep *approximate* track of all things that *could* be true at the corresponding time.
- At each level we keep *approximate* track of what actions *could* be applicable at the corresponding time.

The approximation is due to the fact that not all conflicts between actions are tracked. *So*:

- The graph can *underestimate* how long it might take for a particular proposition to appear, and therefore ...
- ... a heuristic can be extracted.

Planning graphs: a simple example

Our intrepid student adventurers will of course need to inflate their *gorilla* before attaching it to a *distinguished roof*. It has to be purchased before it can be inflated.

Start state: Empty.

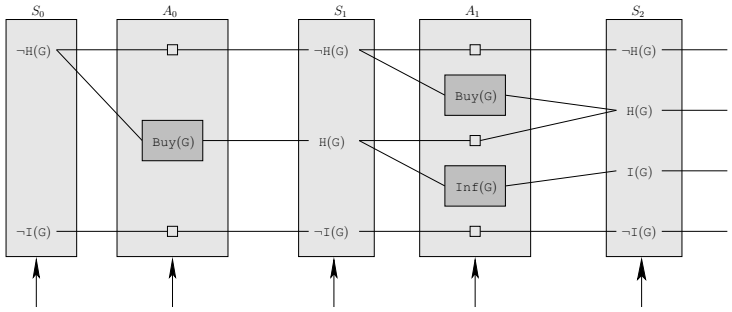
We assume that anything not mentioned in a state is false. So the state is actually

$\neg\texttt{Have(Gorilla)}$ and $\neg\texttt{Inflated(Gorilla)}$

Actions:

Goal: $\texttt{Have(Gorilla)}$ and $\texttt{Inflated(Gorilla)}$.

Planning graphs



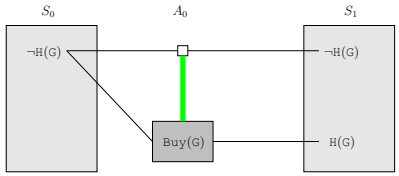
□ = a *persistence action*—what happens if no action is taken.

An action level A_i contains *all* actions that *could* happen given the propositions in S_i.

Mutex links

We also record, using *mutual exclusion (mutex) links* which pairs of actions could not occur together.

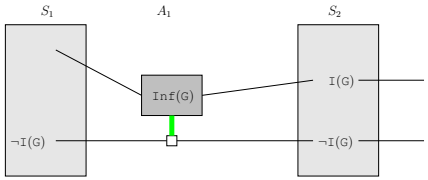
Mutex links 1: Effects are inconsistent.



The effect of one action negates the effect of another.

Mutex links

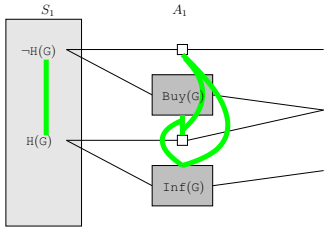
Mutex links 2: The actions interfere.



The effect of an action negates the precondition of another.

Mutex links

Mutex links 3: Competing for preconditions.



The precondition for an action is mutually exclusive with the precondition for another. (See next slide!)

Mutex links

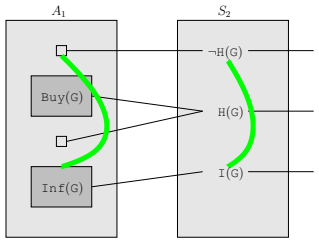
A state level S_i contains *all* propositions that *could* be true, given the possible preceding actions.

We also use mutex links to record pairs that can not be true simultaneously:

Possibility 1: pair consists of a proposition and its negation.

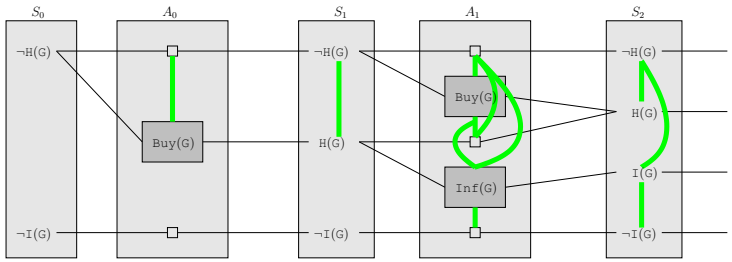
Mutex links

Possibility 2: all pairs of actions that could achieve the pair of propositions are mutex.



The construction of a planning graph is continued until two identical levels are obtained.

Planning graphs



Obtaining heuristics from a planning graph

To estimate the cost of reaching a single proposition:

- Any proposition not appearing in the final level has *infinite cost* and *can never be reached*.
- The *level cost* of a proposition is the level at which it first appears *but* this may be inaccurate as several actions can apply at each level and this cost does not count the *number of actions*. (It is however *admissible*.)
- A *serial planning graph* includes mutex links between all pairs of actions except persistence actions.

Level cost in serial planning graphs can be quite a good measurement.

Obtaining heuristics from a planning graph

How about estimating the cost to achieve a *collection* of propositions?

- *Max-level*: use the maximum level in the graph of any proposition in the set. Admissible but can be inaccurate.
- *Level-sum*: use the sum of the levels of the propositions. Inadmissible but sometimes quite accurate if goals tend to be decomposable.
- *Set-level*: use the level at which *all* propositions appear with none being mutex. Can be accurate if goals tend *not* to be decomposable.

Other points about planning graphs

A planning graph guarantees that:

1. *If* a proposition appears at some level, there *may* be a way of achieving it.
2. *If* a proposition does *not* appear, it can *not* be achieved.

The first point here is a loose guarantee because only *pairs* of items are linked by mutex links.

Looking at larger collections can strengthen the guarantee, but in practice the gains are outweighed by the increased computation.

Graphplan

The *GraphPlan* algorithm goes beyond using the planning graph as a source of heuristics.

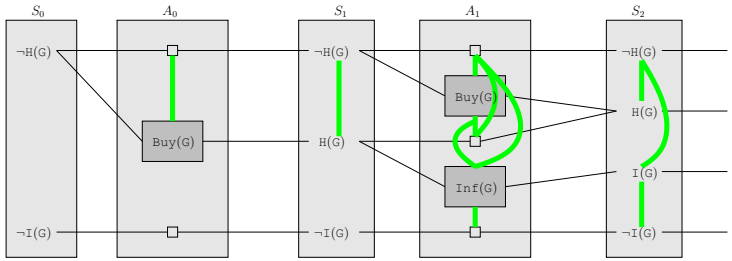
```
Start at level 0;
while(true) {
  if (all goal propositions appear in the current level
      AND no pair has a mutex link) {

      attempt to extract a plan;
      if (a solution is obtained)
        return the solution;
      else if (graph indicates there is no solution)
        return fail;
      expand the graph to the next level;
   }
   else
      expand the graph to the next level;
}
```

We *extract a plan* directly from the planning graph. Termination can be proved but will not be covered here.

Graphplan in action

Here, at levels S_0 and S_1 we do not have both `H(G)` and `I(G)` available with no mutex links, and so we expand first to S_1 and then to S_2.



At S_2 we try to extract a solution (plan).

Extracting a plan from the graph

Extraction of a plan can be formalised as a *search problem*.

States contain a *level*, and a collection of *unsatisfied goal propositions*.

Start state: the current final level of the graph, along with the relevant goal propositions.

Goal: a state at level S_0 containing the initial propositions.

Extracting a plan from the graph

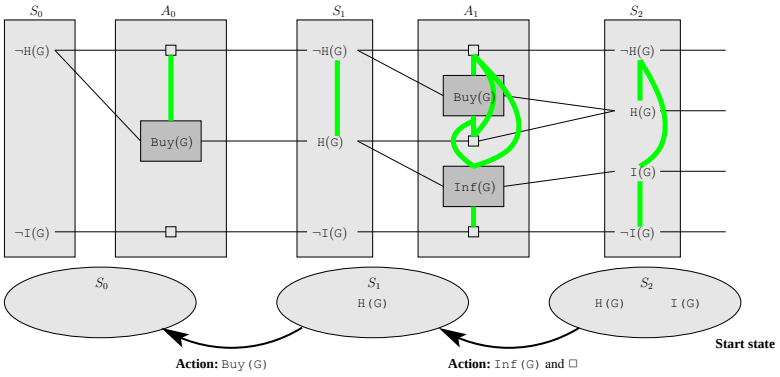
Actions: For a state S with level S_i, a valid action is to select any set X of actions in A_{i-1} such that:

1. no pair has a mutex link;
2. no pair of their preconditions has a mutex link;
3. the effects of the actions in X achieve the propositions in S.

The effect of such an action is a state having level S_{i-1}, and containing the preconditions for the actions in X.

Each action has a cost of 1.

Graphplan in action



Heuristics for plan extraction

We can of course also apply *heuristics* to this part of the process.

For example, when dealing with a *set of propositions*:

- Choose the proposition having *maximum level cost* first.
- For that proposition, attempt to achieve it using the action for which the *maximum/sum level cost of its preconditions is minimum*.

Planning III: planning using propositional logic

Last year we saw that plans might be extracted from a knowledge base via *theorem proving*, using *first order logic (FOL)* and *situation calculus*.

BUT: this might be computationally infeasible for realistic problems.

Sophisticated techniques are available for testing *satisfiability* in *propositional logic*, and these have also been applied to planning.

The basic idea is to attempt to find a model of a sentence having the form

description of start state
$\wedge$ descriptions of the possible actions
$\wedge$ description of goal

Propositional logic for planning

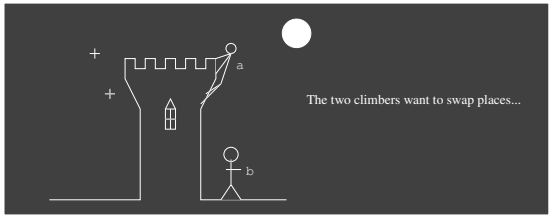
We attempt to construct this sentence such that:

- If M is a model of the sentence then M assigns $\top$ to a proposition if and only if it is in the plan.
- Any assignment denoting an incorrect plan will not be a model as the goal description will not be $\top$.
- The sentence is unsatisfiable if no plan exists.

Propositional logic for planning

Start state:

$$S = \mathtt{At}^0(\mathtt{a}, \mathtt{spire}) \wedge \mathtt{At}^0(\mathtt{b}, \mathtt{ground}) \wedge \neg\mathtt{At}^0(\mathtt{a}, \mathtt{ground}) \wedge \neg\mathtt{At}^0(\mathtt{b}, \mathtt{spire})$$



Remember that an expression such as $\mathtt{At}^0(\mathtt{a}, \mathtt{spire})$ is a *proposition*. The superscripted number now denotes time.

Propositional logic for planning

Goal:

$$G = \mathtt{At}^i(\mathtt{a}, \mathtt{ground}) \wedge \mathtt{At}^i(\mathtt{b}, \mathtt{spire}) \wedge \neg\mathtt{At}^i(\mathtt{a}, \mathtt{spire}) \wedge \neg\mathtt{At}^i(\mathtt{b}, \mathtt{ground})$$

Actions: can be introduced using the equivalent of successor-state axioms

$$\mathtt{At}^1(\mathtt{a},\mathtt{ground}) \leftrightarrow (\mathtt{At}^0(\mathtt{a}, \mathtt{ground}) \wedge \neg\mathtt{Move}^0(\mathtt{a}, \mathtt{ground}, \mathtt{spire})) \vee (\mathtt{At}^0(\mathtt{a}, \mathtt{spire}) \wedge \mathtt{Move}^0(\mathtt{a}, \mathtt{spire}, \mathtt{ground})) \quad (1)$$

Denote by A the collection of all such axioms.

Propositional logic for planning

We will now find that $S \wedge A \wedge G$ has a model in which $\texttt{Move}^0(\texttt{a}, \texttt{spire}, \texttt{ground})$ and $\texttt{Move}^0(\texttt{b}, \texttt{ground}, \texttt{spire})$ are $\top$ while all remaining actions are $\bot$.

In more realistic planning problems we will clearly not know in advance at what time the goal might expect to be achieved.

We therefore:

- Loop through possible final times T.
- Generate a goal for time T and actions up to time T.
- Try to find a model and extract a plan.
- Until a plan is obtained or we hit some maximum time.

Propositional logic for planning

Unfortunately there is a problem—we may, if considerable care is not applied, also be able to obtain less sensible plans.

In the current example

$$\texttt{Move}^0(\texttt{b}, \texttt{ground}, \texttt{spire}) = \top$$
$$\texttt{Move}^0(\texttt{a}, \texttt{spire}, \texttt{ground}) = \top$$
$$\boxed{\texttt{Move}^0(\texttt{a}, \texttt{ground}, \texttt{spire})} = \top$$

is a model, because the successor-state axiom (1) does not in fact preclude the application of $\texttt{Move}^0(\texttt{a}, \texttt{ground}, \texttt{spire})$.

We need a *precondition axiom*

$$\texttt{Move}^i(\texttt{a}, \texttt{ground}, \texttt{spire}) \rightarrow \texttt{At}^i(\texttt{a}, \texttt{ground})$$

and so on.

Propositional logic for planning

Life becomes more complicated still if a third location is added: `hospital`.

$$\texttt{Move}^0(\texttt{a}, \texttt{spire}, \texttt{ground}) \wedge \texttt{Move}^0(\texttt{a}, \texttt{spire}, \texttt{hospital})$$

is perfectly valid and so we need to specify that he can't move to two places simultaneously

$$\neg(\texttt{Move}^i(\texttt{a}, \texttt{spire}, \texttt{ground}) \wedge \texttt{Move}^i(\texttt{a}, \texttt{spire}, \texttt{hospital}))$$
$$\neg(\texttt{Move}^i(\texttt{a}, \texttt{ground}, \texttt{spire}) \wedge \texttt{Move}^i(\texttt{a}, \texttt{ground}, \texttt{hospital}))$$
$$\vdots$$

and so on.

These are *action-exclusion* axioms.

Unfortunately they will tend to produce *totally-ordered* rather than *partially-ordered* plans.

Propositional logic for planning

Alternatively:

1. Prevent actions occurring together if one negates the effect or precondition of the other.
2. Or, specify that something can't be in two places simultaneously

$$\forall x, i, \texttt{l1}, \texttt{l2} \quad \texttt{l1} \neq \texttt{l2} \rightarrow \neg(\texttt{At}^i(x, \texttt{l1}) \wedge \texttt{At}^i(x, \texttt{l2}))$$

This is an example of a *state constraint*.

Clearly this process can become very complex, but there are techniques to help deal with this.

Planning IV: planning using constraint satisfaction

Review of constraint satisfaction problems (CSPs)

We have:

- A set of n *variables* $V_1, V_2, \ldots, V_n$.
- For each V_i a *domain* D_i specifying the values that V_i can take.
- A set of m *constraints* $C_1, C_2, \ldots, C_m$.

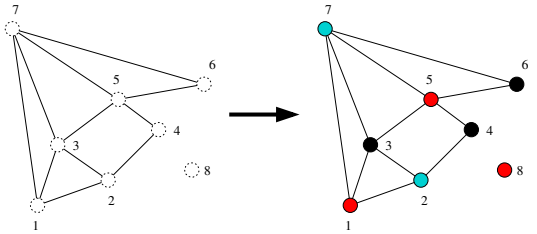
Each constraint C_i involves a set of variables and specifies an *allowable collection of values*.

- A *state* is an assignment of specific values to some or all of the variables.
- An assignment is *consistent* if it violates no constraints.
- An assignment is *complete* if it gives a value to every variable.

A *solution* is a consistent and complete assignment.

Example

We will use the problem of *colouring the nodes of a graph* as a running example.



Each node corresponds to a *variable*. We have three colours and directly connected nodes should have different colours.

Caution required: later on, edges will have a different meaning.

Example

This translates easily to a CSP formulation:

- The variables are the nodes
$$V_i = \text{node } i$$
- The domain for each variable contains the values black, red and cyan
$$D_i = \{B, R, C\}$$
- The constraints enforce the idea that directly connected nodes must have different colours. For example, for variables V_1 and V_2 the constraints specify
$$(B, R), (B, C), (R, B), (R, C), (C, B), (C, R)$$
- Variable V_8 is unconstrained.

Different kinds of CSP

This is an example of the simplest kind of CSP: it is *discrete* with *finite domains*. We will concentrate on these.

We will also concentrate on *binary constraints*; that is, constraints between *pairs of variables*.

- Constraints on single variables—*unary constraints*—can be handled by adjusting the variable's domain. For example, if we don't want V_i to be *red*, then we just remove that possibility from D_i.
- *Higher-order constraints* applying to three or more variables can certainly be considered, but...
- ...when dealing with finite domains they can always be converted to sets of binary constraints by introducing extra *auxiliary variables*.

How does that work?

The state-variable representation

Another planning language: the *state-variable representation*.

Things of interest such as people, places, objects *etc* are divided into *domains*:

$D_1 = \{\texttt{climber1}, \texttt{climber2}\}$
$D_2 = \{\texttt{home}, \texttt{jokeShop}, \texttt{hardwareStore}, \texttt{pavement}, \texttt{spire}, \texttt{hospital}\}$
$D_3 = \{\texttt{rope}, \texttt{inflatableGorilla}\}$

Part of the specification of a planning problem involves stating which domain a particular item is in. For example

$$D_1(\texttt{climber1})$$

and so on.

Relations and functions have arguments chosen from unions of these domains.

$$\texttt{above}(x, y) \subseteq \mathcal{D}_1^{\texttt{above}} \times \mathcal{D}_2^{\texttt{above}}$$

is a relation. The $\mathcal{D}_i^{\texttt{above}}$ are unions of one or more D_i.

The state-variable representation

The relation `above` is in fact a *rigid relation (RR)*, as it is unchanging: it does not depend upon *state*. (Remember *fluents* in situation calculus?)

Similarly, we have *functions*

$$\texttt{at}(x_1, s) : \mathcal{D}_1^{\texttt{at}} \times S \to \mathcal{D}^{\texttt{at}}.$$

Here, $\texttt{at}(x, s)$ is a *state-variable*. The domain $\mathcal{D}_1^{\texttt{at}}$ and range $\mathcal{D}^{\texttt{at}}$ are unions of one or more D_i. In general these can have multiple parameters

$$\texttt{sv}(x_1, \ldots, x_n, s) : \mathcal{D}_1^{\texttt{sv}} \times \cdots \times \mathcal{D}_n^{\texttt{sv}} \times S \to \mathcal{D}^{\texttt{sv}}.$$

A state-variable denotes assertions such as

$$\texttt{at}(\texttt{gorilla}, s) = \texttt{jokeShop}$$

where s denotes a *state* and the set S of all states will be defined later.

The state variable allows things such as locations to change—again, much like *fluents* in the situation calculus.

Variables appearing in relations and functions are considered to be *typed*.

The state-variable representation

Note:

- For properties such as a *location* a function might be considerably more suitable than a relation.
- For locations, everything has to be *somewhere* and it can only be in *one place at a time*.

So a function is perfect and immediately solves some of the problems seen earlier.

The state-variable representation

Actions as usual, have a *name*, a *set of preconditions* and a *set of effects*.

- *Names* are unique, and followed by a list of variables involved in the action.
- *Preconditions* are expressions involving state variables and relations.
- *Effects* are assignments to state variables.

For example:

$\texttt{buy}(x, y, l)$	
Preconditions	$\texttt{at}(x, s) = l$
	$\texttt{sells}(l, y)$
	$\texttt{has}(y, s) = l$
Effects	$\texttt{has}(y, s) = x$

The state-variable representation

Goals are sets of *expressions* involving *state variables*.

For example:

Goal:
$\texttt{at}(\texttt{climber}, s) = \texttt{home}$
$\texttt{has}(\texttt{rope}, s) = \texttt{climber}$
$\texttt{at}(\texttt{gorilla}, s) = \texttt{spire}$

From now on we will generally suppress the state s when writing state variables.

The state-variable representation

We can essentially regard a *state* as just a statement of what values the state variables take at a given time.

Formally:

- For each state variable `sv` we can consider all ground instances such as—$\texttt{sv}(\texttt{climber}, \texttt{rope})$—with arguments that are *consistent* with the *rigid relations*.

  Define X to be the set of all such ground instances.
- A state s is then just a set

$$s = \{(v = c) | v \in X\}$$

  where c is in the range of v.

This allows us to define the *effect of an action*.

A planning problem also needs a *start state* s_0, which can be defined in this way.

The state-variable representation

Considering all the *ground actions consistent with the rigid relations*:

- An action is *applicable in* s if all expressions $v = c$ appearing in the set of preconditions also appear in s.

Finally, there is a function γ that maps a state and an action to a new state

$$\gamma(s, a) = s'$$

Specifically, we have

$$\gamma(s, a) = \{(v = c) | v \in X\}$$

where either c is specified in an effect of a, or otherwise $v = c$ is a member of s.

Note: the definition of γ implicitly solves the *frame problem.*

The state-variable representation

A *solution* to a planning problem is a sequence $(a_0, a_1, \ldots, a_n)$ of actions such that...

- a_0 is applicable in s_0 and for each i, a_i is applicable in $s_i = \gamma(s_{i-1}, a_{i-1})$.
- For each goal g we have
$$g \in \gamma(s_n, a_n).$$

What we need now is a method for *transforming* a problem described in this language into a CSP.

We'll once again do this for a fixed upper limit T on the number of steps in the plan.

Converting to a CSP

Step 1: encode *actions* as *CSP variables*.

For each time step t where $0 \leq t \leq T-1$, the CSP has a variable

$$\texttt{action}^t$$

with domain

$$D^{\texttt{action}^t} = \{\texttt{a} | \texttt{a} \text{ is the ground instance of an action}\} \cup \{\texttt{none}\}$$

Example: at some point in searching for a plan we might attempt to find the solution to the corresponding CSP involving

$$\texttt{action}^5 = \texttt{attach}(\texttt{inflatableGorilla}, \texttt{spire})$$

WARNING: be careful in what follows to distinguish between *state variables, actions etc* in the planning problem and *variables* in the CSP.

Converting to a CSP

Step 2: encode *ground state variables* as *CSP variables*, with a complete copy of all the state variables *for each time step*.

So, for each t where $0 \leq t \leq T$ we have a CSP variable

$$\texttt{sv}_i^t(c_1, \ldots, c_n)$$

with domain $\mathcal{D}^{\texttt{sv}_i}$. (That is, the *domain* of the CSP variable is the *range* of the state variable.)

Example: at some point in searching for a plan we might attempt to find the solution to the corresponding CSP involving

$$\texttt{location}^9(\texttt{climber1}) = \texttt{hospital}.$$

Converting to a CSP

Step 3: encode the *preconditions for actions in the planning problem* as *constraints in the CSP problem*.

For each time step t and for each ground action $\texttt{a}(c_1, \ldots, c_n)$ with arguments *consistent with the rigid relations in its preconditions*:

For a precondition of the form $\texttt{sv}_i = v$ include constraint pairs

$$(\texttt{action}^t = \texttt{a}(c_1, \ldots, c_n), \\ \texttt{sv}_i^t = v)$$

Example: consider the action $\texttt{buy}(x, y, l)$ introduced above, and having the preconditions $\texttt{at}(x) = l$, $\texttt{sells}(l, y)$ and $\texttt{has}(y) = l$.

Assume $\texttt{sells}(y, l)$ is only true for

$$l = \texttt{jokeShop}$$

and

$$y = \texttt{inflatableGorilla}$$

(it's a very strange town) so we only consider these values for l and y. Then for each time step t we have the constraints...

Converting to a CSP

$\texttt{action}^t = \texttt{buy}(\texttt{climber1}, \texttt{inflatableGorilla}, \texttt{jokeShop})$ paired with $\texttt{at}^t(\texttt{climber1}) = \texttt{jokeShop}$
$\texttt{action}^t = \texttt{buy}(\texttt{climber1}, \texttt{inflatableGorilla}, \texttt{jokeShop})$ paired with $\texttt{has}^t(\texttt{inflatableGorilla}) = \texttt{jokeShop}$
$\texttt{action}^t = \texttt{buy}(\texttt{climber2}, \texttt{inflatableGorilla}, \texttt{jokeShop})$ paired with $\texttt{at}^t(\texttt{climber2}) = \texttt{jokeShop}$
$\texttt{action}^t = \texttt{buy}(\texttt{climber2}, \texttt{inflatableGorilla}, \texttt{jokeShop})$ paired with $\texttt{has}^t(\texttt{inflatableGorilla}) = \texttt{jokeShop}$
and so on...

Converting to a CSP

Step 4: encode the *effects of actions in the planning problem* as *constraints in the CSP problem*.

For each time step t and for each ground action $\texttt{a}(c_1, \ldots, c_n)$ with arguments *consistent with the rigid relations in its preconditions*:

For an effect of the form $\texttt{sv}_i = v$ include constraint pairs

$$(\texttt{action}^t = \texttt{a}(c_1, \ldots, c_n), \\ \texttt{sv}_i^{t+1} = v)$$

Example: continuing with the previous example, we will include constraints

$\texttt{action}^t = \texttt{buy}(\texttt{climber1}, \texttt{inflatableGorilla}, \texttt{jokeShop})$ paired with $\texttt{has}^{t+1}(\texttt{inflatableGorilla}) = \texttt{climber1}$
$\texttt{action}^t = \texttt{buy}(\texttt{climber2}, \texttt{inflatableGorilla}, \texttt{jokeShop})$ paired with $\texttt{has}^{t+1}(\texttt{inflatableGorilla}) = \texttt{climber2}$
and so on...

Converting to a CSP

Step 5: encode the *frame axioms* as *constraints in the CSP problem*.

An action must not change things not appearing in its effects. So:

For:

1. Each time step t.
2. Each ground action $\texttt{a}(c_1, \ldots, c_n)$ with arguments *consistent with the rigid relations in its preconditions*.
3. Each $\texttt{sv}_\texttt{i}$ that *does not appear in the effects of* $\texttt{a}$, and each $v \in \mathcal{D}^{\texttt{sv}_i}$

include in the CSP the ternary constraint

$$(\texttt{action}^t = \texttt{a}(c_1, \ldots, c_n), \\ \texttt{sv}_i^t = v, \\ \texttt{sv}_i^{t+1} = v)$$

Finding a plan

Finally, having encoded a planning problem into a CSP, we solve the CSP.

The scheme has the following property:

A solution to the planning problem with at most T steps exists if and only if there is a a solution to the corresponding CSP.

Assume the CSP has a solution.

Then we can extract a plan simply by looking at the values assigned to the $\texttt{action}^t$ variables in the solution of the CSP.

It is also the case that:

There is a solution to the planning problem with at most T steps if and only if there is a solution to the corresponding CSP from which the solution can be extracted in this way.

For a proof see:

Automated Planning: Theory and Practice

Malik Ghallab, Dana Nau and Paolo Traverso. Morgan Kaufmann 2004.

Uncertainty I: Probability as Degree of Belief

We now examine:

- How *probability theory* might be used to represent and reason with knowledge when we are *uncertain* about the world.
- How *inference* in the presence of uncertainty can in principle be performed using only basic results along with the *full joint probability distribution*.
- How this approach *fails* in practice.
- How the notions of *independence* and *conditional independence* may be used to solve this problem.

Reading: Russell and Norvig, chapter 13.

Uncertainty in AI

The (predominantly logic-based) methods covered so far have assorted shortcomings:

- Limited *epistemological commitment*—true/false/unknown.
- Actions are possible when *sufficient knowledge* is available...
- ...but this is not generally the case.
- In practice there is a need to cope with *uncertainty*.

For example in the Wumpus World:

- We can not make observations further afield than the current locality.
- Consequently inferences regarding pit/wumpus location *etc* will not usually be possible.

Uncertainty in AI

A couple of more subtle problems have also presented themselves:

- The *Qualification Problem:* it is not generally possible to guarantee that an action will succeed—only that it will succeed if *many other preconditions* do/don't hold.
- *Rational action* depends on the *likelihood* of achieving different goals, and their *relative desirability*.

Logic (as seen so far) has major shortcomings

An example:

$$\forall x\ \texttt{symptom}(x, \texttt{toothache}) \rightarrow \texttt{problem}(x, \texttt{cavity})$$

This is plainly incorrect. Toothaches can be caused by things other than cavities.

$$\begin{aligned}\forall x\ \texttt{symptom}(x, \texttt{toothache}) \rightarrow & \texttt{problem}(x, \texttt{cavity}) \vee \\ & \texttt{problem}(x, \texttt{abscess}) \vee \\ & \texttt{problem}(x, \texttt{gum-disease}) \vee \\ & \ldots\end{aligned}$$

BUT:

- It is *impossible to complete* the list.
- There's no clear way to take account of the *relative likelihoods* of different causes.

Logic (as seen so far) has major shortcomings

If we try to make a *causal rule*

$$\forall x\, \texttt{problem}(x, \texttt{abscess}) \rightarrow \texttt{symptom}(x, \texttt{toothache})$$

it's still wrong—abscesses do not always cause pain.

We need further information in addition to

$$\texttt{problem}(x, \texttt{abscess})$$

and it's still not possible to do this correctly.

Logic (as seen so far) has major shortcomings

FOL can fail for essentially three reasons:

1. *Laziness:* it is not feasible to assemble a set of rules that is sufficiently exhaustive.

   If we could, it would not be feasible to apply them.
2. *Theoretical ignorance:* insufficient knowledge *exists* to allow us to write the rules.
3. *Practical ignorance:* even if the rules have been obtained there may be insufficient information to apply them.

Instead of thinking in terms of the *truth* or *falsity* of a statement we want to deal with an agent's *degree of belief* in the statement.

- *Probability theory* is the perfect tool for application here.
- *Probability theory* allows us to *summarise* the uncertainty due to laziness and ignorance.

An important distinction

There is a fundamental difference between *probability theory* and *fuzzy logic*:

- When dealing with probability theory, statements remain *in fact* either *true* or *false*.
- A probability denotes an agent's *degree of belief* one way or another.
- Fuzzy logic deals with *degree of truth*.

In practice the use of probability theory has proved spectacularly successful.

Belief and evidence

An agent's beliefs will depend on what it has *perceived*: probabilities are based on *evidence* and may be altered by the acquisition of new evidence:

- *Prior (unconditional) probability* denotes a degree of belief in the absence of evidence.
- *Posterior (conditional) probability* denotes a degree of belief after evidence is perceived.

As we shall see *Bayes' theorem* is the fundamental concept that allows us to update one to obtain the other.

Making rational decisions under uncertainty

When using *logic*, we concentrated on finding an action sequence guaranteed to achieve a goal, and then executing it.

When dealing with *uncertainty* we need to define *preferences* among states of the world and take into account the *probability* of reaching those states.

Utility theory is used to assign preferences.

Decision theory combines probability theory and utility theory.

A *rational* agent should act in order to *maximise expected utility*.

Probability

We want to assign degrees of belief to propositions about the world.

We will need:

- *Random variables* with associated *domains*—typically Boolean, discrete, or continuous.
- All the usual concepts—events, atomic events, sets *etc*.
- Probability distributions and densities.
- Probability axioms (Kolmogorov).
- Conditional probability and Bayes' theorem.

So if you've forgotten this stuff now is a good time to re-read it.

Probability

The standard axioms are:

- Range
$$0 \leq \Pr(x) \leq 1$$
- Always true propositions
$$\Pr(\texttt{always true proposition}) = 1$$
- Always false propositions
$$\Pr(\texttt{always false proposition}) = 0$$
- Union
$$\Pr(x \vee y) = \Pr(x) + \Pr(y) - \Pr(x \wedge y)$$

Origins of probabilities I

Historically speaking, probabilities have been regarded in a number of different ways:

- *Frequentist:* probabilities come from measurements.
- *Objectivist:* probabilities are actual "properties of the universe" which frequentist measurements seek to uncover.

  An excellent example: quantum phenomena.

  A bad example: coin flipping—the uncertainty is due to our uncertainty about the initial conditions of the coin.
- *Subjectivist:* probabilities are an agent's degrees of belief.

  This means the agent is allowed to make up the numbers!

Origins of probabilities II

The *reference class problem*: even frequentist probabilities are subjective.

Example: Say a doctor takes a frequentist approach to diagnosis. She examines a large number of people to establish the prior probability of whether or not they have heart disease.

To be accurate she tries to measure "similar people". (She knows for example that gender might be important.)

Taken to an extreme, *all* people are *different* and there is therefore no *reference class*.

Origins of probabilities III

The *principle of indifference* (Laplace).

- Give equal probability to all propositions that are syntactically symmetric with respect to the available evidence.
- Refinements of this idea led to the attempted development by Carnap and others of *inductive logic*.
- The aim was to obtain the correct probability of any proposition from an arbitrary set of observations.

It is currently thought that no unique inductive logic exists.

Any inductive logic depends on prior beliefs and the effect of these beliefs is overcome by evidence.

Prior probability

A *prior probability* denotes the probability (degree of belief) assigned to a proposition *in the absence of any other evidence*.

For example

$$\Pr(\texttt{Cavity} = \texttt{true}) = 0.05$$

denotes the degree of belief that a random person has a cavity *before we make any actual observation of that person*.

To keep things compact, we will use

$$\Pr(\texttt{Cavity})$$

to denote the entire probability distribution of the random variable `Cavity`.

Instead of

$$\Pr(\texttt{Cavity} = \texttt{true}) = 0.05$$

$$\Pr(\texttt{Cavity} = \texttt{false}) = 0.95$$

write

$$\Pr(\texttt{Cavity}) = (0.05, 0.95)$$

Notation

A similar convention will apply for joint distributions. For example, if `Decay` can take the values `severe`, `moderate` or `low` then

$$\Pr(\texttt{Cavity}, \texttt{Decay})$$

is a 2 by 3 table of numbers.

	`severe`	`moderate`	`low`
`true`	0.26	0.1	0.01
`false`	0.01	0.02	0.6

Similarly

$$\Pr(\texttt{true}, \texttt{Decay})$$

denotes 3 numbers *etc*.

The full joint probability distribution

The *full joint probability distribution* is the joint distribution of *all* random variables that describe the state of the world.

This can be used to answer *any query*.

(But of course life's not really that simple!)

Conditional probability

We use the *conditional probability*

$$\Pr(x|y)$$

to denote the probability that a proposition x holds given that *all the evidence we have so far* is contained in proposition y.

From basic probability theory

$$\Pr(x|y) = \frac{\Pr(x \wedge y)}{\Pr(y)}$$

Conditional probability is *not* analogous to *logical implication*.

- $\Pr(x|y) = 0.1$ does *not* mean that if y is true then $\Pr(x) = 0.1$.
- $\Pr(x)$ is a *prior probability*.
- The notation $\Pr(x|y)$ is for use when y is the *entire evidence*.
- $\Pr(x|y \wedge z)$ might be very different.

Bayes theorem

From first principles

$$\Pr(x, y) = \Pr(x|y)\Pr(y)$$

$$\Pr(x, y) = \Pr(y|x)\Pr(x)$$

so

$$\Pr(x|y) = \frac{\Pr(y|x)\Pr(x)}{\Pr(y)}$$

The most important equation in modern AI?

When *evidence* e is involved this can be written

$$\Pr(Q|R, e) = \frac{\Pr(R|Q, e)\Pr(Q|e)}{\Pr(R|e)}$$

Bayes theorem

Taking another simple medical diagnosis example: *does a patient with a fever have malaria?* A doctor might know that

$$\Pr(\texttt{fever}|\texttt{malaria}) = 0.99$$

$$\Pr(\texttt{malaria}) = \frac{1}{10000}$$

$$\Pr(\texttt{fever}) = \frac{1}{20}$$

Consequently we can try to obtain $\Pr(\texttt{malaria}|\texttt{fever})$ by direct application of Bayes theorem

$$\Pr(\texttt{malaria}|\texttt{fever}) = \frac{0.99 \times 0.0001}{0.05} = 0.00198$$

or using the alternative technique

$$\Pr(\texttt{malaria}|\texttt{fever}) = \alpha \Pr(\texttt{fever}|\texttt{malaria})\Pr(\texttt{malaria})$$

if the relevant further quantity $\Pr(\texttt{fever}|\neg\texttt{malaria})$ is known.

Bayes theorem

- Sometimes the first possibility is easier, sometimes not.
- *Causal knowledge* such as

$$\Pr(\texttt{fever}|\texttt{malaria})$$

might well be available when *diagnostic knowledge* such as

$$\Pr(\texttt{malaria}|\texttt{fever})$$

is not.

- Say the incidence of malaria, modelled by $\Pr(\texttt{Malaria})$, suddenly changes. Bayes theorem tells us what to do.
- The quantity

$$\Pr(\texttt{fever}|\texttt{malaria})$$

would not be affected by such a change.

Causal knowledge can be more robust.

Using the full joint distribution to perform inference

We can regard the full joint distribution as a *knowledge base*.

We want to use it to obtain answers to questions.

	CP		¬CP	
	HBP	¬HBP	HBP	¬HBP
HD	0.09	0.05	0.07	0.01
¬HD	0.02	0.08	0.03	0.65

We'll use this medical diagnosis problem as a running example.

- HD = Heart disease
- CP = Chest pain
- HBP = High blood pressure

Using the full joint distribution to perform inference

The process is nothing more than the application of basic results:

- Sum atomic events:

$$\begin{aligned}\Pr(\texttt{HD} \vee \texttt{CP}) = &\Pr(\texttt{HD} \wedge \texttt{CP} \wedge \texttt{HBP}) \\ &+ \Pr(\texttt{HD} \wedge \texttt{CP} \wedge \neg\texttt{HBP}) \\ &+ \Pr(\texttt{HD} \wedge \neg\texttt{CP} \wedge \texttt{HBP}) \\ &+ \Pr(\texttt{HD} \wedge \neg\texttt{CP} \wedge \neg\texttt{HBP}) \\ &+ \Pr(\neg\texttt{HD} \wedge \texttt{CP} \wedge \texttt{HBP}) \\ &+ \Pr(\neg\texttt{HD} \wedge \texttt{CP} \wedge \neg\texttt{HBP}) \\ &= 0.09 + 0.05 + 0.07 + 0.01 + 0.02 + 0.08 \\ &= 0.32\end{aligned}$$

- Marginalisation: if A and B are sets of variables then

$$\Pr(A) = \sum_b \Pr(A \wedge b) = \sum_b \Pr(A|b)\Pr(b)$$

Using the full joint distribution to perform inference

Usually we will want to compute the *conditional probability* of *some variable(s)* given *some evidence*.

For example

$$\Pr(\texttt{HD}|\texttt{HBP}) = \frac{\Pr(\texttt{HD} \wedge \texttt{HBP})}{\Pr(\texttt{HBP})} = \frac{0.09 + 0.07}{0.09 + 0.07 + 0.02 + 0.03} = 0.76$$

and

$$\Pr(\neg\texttt{HD}|\texttt{HBP}) = \frac{\Pr(\neg\texttt{HD} \wedge \texttt{HBP})}{\Pr(\texttt{HBP})} = \frac{0.02 + 0.03}{0.09 + 0.07 + 0.02 + 0.03} = 0.24$$

Using the full joint distribution to perform inference

The process can be simplified slightly by noting that

$$\alpha = \frac{1}{\Pr(\texttt{HBP})}$$

is a constant and can be regarded as a *normaliser* making relevant probabilities sum to 1.

So a short cut is to avoid computing it as above. Instead:

$$\Pr(\texttt{HD}|\texttt{HBP}) = \alpha \Pr(\texttt{HD} \wedge \texttt{HBP}) = (0.09 + 0.07)\alpha$$

$$\Pr(\neg\texttt{HD}|\texttt{HBP}) = \alpha \Pr(\neg\texttt{HD} \wedge \texttt{HBP}) = (0.02 + 0.03)\alpha$$

and we need

$$\Pr(\texttt{HD}|\texttt{HBP}) + \Pr(\neg\texttt{HD}|\texttt{HBP}) = 1$$

so

$$\alpha = \frac{1}{0.09 + 0.07 + 0.02 + 0.03}$$

Using the full joint distribution to perform inference

The *general inference procedure* is as follows:

$$\Pr(Q|e) = \frac{1}{Z}\Pr(Q \wedge e) = \frac{1}{Z}\sum_u \Pr(Q, e, u)$$

where

- Q is the query variable.
- e is the evidence.
- u are the unobserved variables.
- $1/Z$ normalises the distribution.

Using the full joint distribution to perform inference

Simple eh?

Well, no...

- For n Boolean variables the table has 2^n entries.
- Storage and processing time are both $O(2^n)$.
- You need to establish 2^n numbers to work with.

In reality we might well have $n > 1000$, and of course it's *even worse* if *variables are non-Boolean*.

How can we get around this?

Exploiting independence

If I toss a coin and roll a dice, the full joint distribution of outcomes requires $2 \times 6 = 12$ numbers to be specified.

	1	2	3	4	5	6
head	0.014	0.028	0.042	0.057	0.071	0.086
tail	0.033	0.067	0.1	0.133	0.167	0.2

Here $\Pr(\texttt{Coin} = \texttt{head}) = 0.3$ and the dice has probability $i/21$ for the ith outcome.

BUT: if we assume the outcomes are independent then

$$\Pr(\texttt{Coin}, \texttt{Dice}) = \Pr(\texttt{Coin})\Pr(\texttt{Dice})$$

Where $\Pr(\texttt{Coin})$ has two numbers and $\Pr(\texttt{Dice})$ has six.

So instead of 12 numbers we only need 8.

Exploiting independence

Similarly, say instead of just considering HD, HBP and CP we also consider the outcome of the *Oxford versus Cambridge tiddlywinks competition* TC:

$$\Pr(\texttt{TC = Oxford}) = 0.2$$

$$\Pr(\texttt{TC = Cambridge}) = 0.7$$

$$\Pr(\texttt{TC = Draw}) = 0.1$$

Now

$$\Pr(\texttt{HD}, \texttt{HBP}, \texttt{CP}, \texttt{TC}) = \Pr(\texttt{TC}|\texttt{HD}, \texttt{HBP}, \texttt{HD}) \Pr(\texttt{HD}, \texttt{HBP}, \texttt{HD})$$

Assuming that the patient is not an *extraordinarily keen fan of tiddlywinks*, their cardiac health has nothing to do with the outcome, so

$$\Pr(\texttt{TC}|\texttt{HD}, \texttt{HBP}, \texttt{HD}) = \Pr(\texttt{TC})$$

and $2 \times 2 \times 2 \times 3 = 24$ numbers has been reduced to $3 + 8 = 11$.

Exploiting independence

In general you need to identify such independence through *knowledge of the problem*.

BUT:

- It generally does not work as clearly as this.
- The independent subsets themselves can be big.

Conditional independence

What happens if we have *multiple pieces of evidence*?

We have seen that to compute

$$\Pr(\texttt{HD}|\texttt{CP}, \texttt{HBP})$$

directly might well run into problems.

We could try using Bayes theorem to obtain

$$\Pr(\texttt{HD}|\texttt{CP}, \texttt{HBP}) = \alpha \Pr(\texttt{CP}, \texttt{HBP}|\texttt{HD}) \Pr(\texttt{HD})$$

However while HD is probably manageable, a quantity such as $\Pr(\texttt{CP}, \texttt{HBP}|\texttt{HD})$ might well still be problematic especially in more realistic cases.

Conditional independence

However although in this case we might not be able to exploit independence directly we *can* say that

$$\Pr(\texttt{CP}, \texttt{HBP}|\texttt{HD}) = \Pr(\texttt{CP}|\texttt{HD}) \Pr(\texttt{HBP}|\texttt{HD})$$

which simplifies matters.

Conditional independence:

- $\Pr(A, B|C) = \Pr(A|C) \Pr(B|C)$.
- If we know that C is the case then A and B are independent.

Although CP and HBP are *not* independent, they do not directly influence one another *in a patient known to have heart disease*.

This is much nicer!

$$\Pr(\texttt{HD}|\texttt{CP}, \texttt{HBP}) = \alpha \Pr(\texttt{CP}|\texttt{HD}) \Pr(\texttt{HBP}|\texttt{HD}) \Pr(\texttt{HD})$$

Naive Bayes

Conditional independence is often assumed even when it does not hold.

Naive Bayes:

$$\Pr(A, B_1, B_2, \ldots, B_n) = \Pr(A) \prod_{i=1}^{n} \Pr(B_i|A)$$

Also known as *Idiot's Bayes*.

Despite this, it is often surprisingly effective.

Uncertainty II - Bayesian Networks

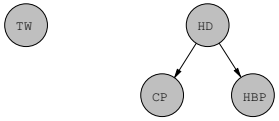
Having seen that in principle, if not in practice, the full joint distribution alone can be used to perform any inference of interest, we now examine a practical technique.

- We introduce the *Bayesian Network (BN)* as a compact representation of the full joint distribution.
- We examine the way in which a BN can be *constructed*.
- We examine the *semantics* of BNs.
- We look briefly at how *inference* can be performed.

Reading: Russell and Norvig, chapter 14.

Bayesian networks

Also called *probabilistic/belief/causal networks* or *knowledge maps*.



- Each node is a *random variable (RV)*.
- Each node N_i has a distribution

$$\Pr(N_i|\texttt{parents}(N_i))$$

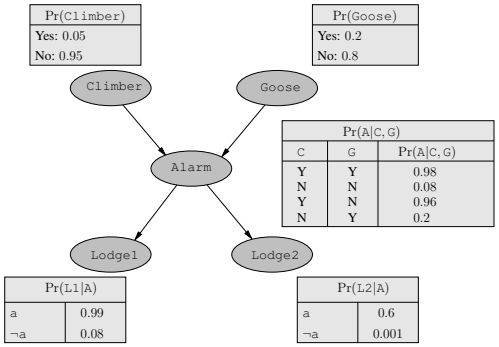
- A Bayesian network is a *directed acyclic graph*.
- Roughly speaking, an arrow from N to M means N directly affects M.

Bayesian networks

After a *regrettable incident* involving an *inflatable gorilla*, a famous College has decided to install an alarm for the detection of roof climbers.

- The alarm is *very* good at detecting climbers.
- Unfortunately, it is also sometimes triggered when one of the *extremely fat geese* that lives in the College lands on the roof.
- One porter's lodge is near the alarm, and inhabited by a chap with *excellent hearing* and a *pathological hatred* of roof climbers: he *always* reports an alarm. His hearing is so good that he sometimes thinks he hears an alarm, *even when there isn't one*.
- Another porter's lodge is a good distance away and inhabited by an *old chap* with *dodgy hearing* who likes to listen to his collection of *DEATH METAL* with the sound turned up.

Bayesian networks



Bayesian networks

Note that:

- In the present example all RVs are *discrete* (in fact Boolean) and so in all cases $\Pr(N_i|\text{parents}(N_i))$ can be represented as a *table of numbers*.
- `Climber` and `Goose` have only *prior* probabilities.
- All RVs here are Boolean, so a node with p parents requires 2^p numbers.

A BN with n nodes represents the full joint probability distribution for those nodes as

$$\Pr(N_1 = n_1, N_2 = n_2, \ldots, N_n = n_n) = \prod_{i=1}^{n} \Pr(N_i = n_i|\text{parents}(N_i)) \qquad (2)$$

For example

$$\begin{aligned}\Pr(\neg\mathtt{C}, \neg\mathtt{G}, \mathtt{A}, \mathtt{L1}, \mathtt{L2}) &= \Pr(\mathtt{L1}|\mathtt{A}) \Pr(\mathtt{L2}|\mathtt{A}) \Pr(\mathtt{A}|\neg\mathtt{C}, \neg\mathtt{G}) \Pr(\neg\mathtt{C}) \Pr(\neg\mathtt{G}) \\ &= 0.99 \times 0.6 \times 0.08 \times 0.95 \times 0.8\end{aligned}$$

Semantics

In general $\Pr(A, B) = \Pr(A|B) \Pr(B)$ so abbreviating $\Pr(N_1 = n_1, N_2 = n_2, \ldots, N_n = n_n)$ to $\Pr(n_1, n_2, \ldots, n_n)$ we have

$$\Pr(n_1, \ldots, n_n) = \Pr(n_n|n_{n-1}, \ldots, n_1) \Pr(n_{n-1}, \ldots, n_1)$$

Repeating this gives

$$\begin{aligned}\Pr(n_1, \ldots, n_n) &= \Pr(n_n|n_{n-1}, \ldots, n_1) \Pr(n_{n-1}|n_{n-2}, \ldots, n_1) \cdots \Pr(n_1) \\ &= \prod_{i=1}^{n} \Pr(n_i|n_{i-1}, \ldots, n_1) \qquad (3)\end{aligned}$$

Now compare equations (2) and (3). We see that BNs make the assumption

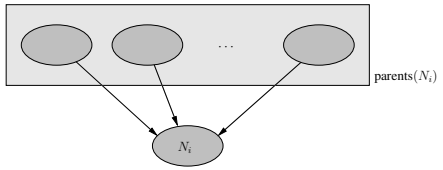
$$\Pr(N_i|N_{i-1}, \ldots, N_1) = \Pr(N_i|\text{parents}(N_i))$$

for each node, assuming that $\text{parents}(N_i) \subseteq \{N_{i-1}, \ldots, N_1\}$.

Each N_i is conditionally independent of its predecessors given its parents

Semantics

- When constructing a BN we want to make sure the preceding property holds.
- This means we need to take care over *ordering*.
- In general *causes should directly precede effects*.



Here, $\text{parents}(N_i)$ contains all preceding nodes having a *direct influence* on N_i.

Semantics

Deviation from this rule can have major effects on the complexity of the network.

That's bad! We want to keep the network simple:

- If each node has at most p parents and there are n Boolean nodes, we need to specify at most $n2^p$ numbers...
- ...whereas the full joint distribution requires us to specify 2^n numbers.

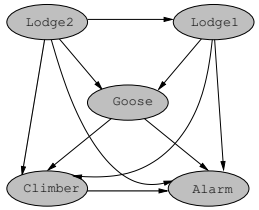
So: there is a trade-off attached to the inclusion of *tenuous* although *strictly-speaking correct* edges.

Semantics

As a rule, we should include the *most basic causes* first, then *the things they influence directly etc*.

What happens if you get this wrong?

Example: add nodes in the order `L2`,`L1`,`G`,`C`,`A`.



Semantics

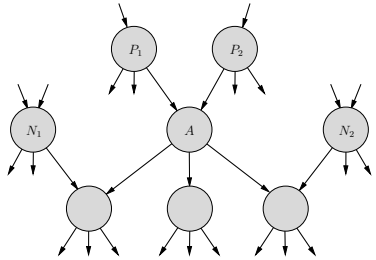
In this example:

- Increased connectivity.
- Many of the probabilities here will be quite unnatural and hard to specify.

Once again: *causal knowledge* is preferred to *diagnostic knowledge*.

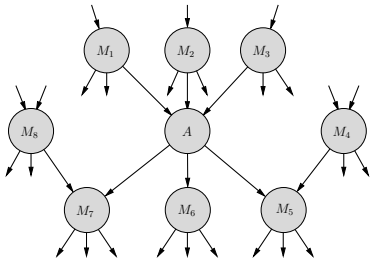
Semantics

As an alternative we can say directly what conditional independence assumptions a graph should be interpreted as expressing. There are two common ways of doing this.



Any node A is conditionally independent of the N_i—its *non-descendants*—given the P_i—its parents.

Semantics



Any node A is conditionally independent of all other nodes given the *Markov blanket* M_i—that is, its *parents*, its *children* and its *children's parents*.

More complex nodes

How do we represent

$$\Pr(N_i|\text{parents}(N_i))$$

when nodes can denote *general discrete and/or continuous RVs*?

- BNs containing both kinds of RV are called *hybrid BNs*.
- Naive *discretisation* of continuous RVs tends to result in both a reduction in accuracy and large tables.
- $O(2^p)$ might still be large enough to be unwieldy.
- We can instead attempt to use *standard and well-understood* distributions, such as the *Gaussian*.
- This will typically require only a small number of parameters to be specified.

More complex nodes

Example: functional relationships are easy to deal with.

$$N_i = f(\text{parents}(N_i))$$

$$\Pr(N_i = n_i|\text{parents}(N_i)) = \begin{cases} 1 & \text{if } n_i = f(\text{parents}(N_i)) \\ 0 & \text{otherwise} \end{cases}$$

More complex nodes

Example: a continuous RV with one continuous and one discrete parent.

$$\Pr(\texttt{Speed of car}|\texttt{Throttle position}, \texttt{Tuned engine})$$

where SC and TP are continuous and TE is Boolean.

- For a specific setting of ET = true it might be the case that SC increases with TP, but that some uncertainty is involved

$$\Pr(\texttt{SC}|\texttt{TP}, \texttt{et}) = N(g_{\texttt{et}}\texttt{TP} + c_{\texttt{et}}, \sigma^2_{\texttt{et}})$$

- For an un-tuned engine we might have a similar relationship with a different behaviour

$$\Pr(\texttt{SC}|\texttt{TP}, \neg\texttt{et}) = N(g_{\neg\texttt{et}}\texttt{TP} + c_{\neg\texttt{et}}, \sigma^2_{\neg\texttt{et}})$$

There is a set of parameters $\{g, c, \sigma\}$ for each possible value of the discrete RV.

More complex nodes

Example: a discrete RV with a continuous parent

$$\Pr(\texttt{Go roofclimbing}|\texttt{Size of fine})$$

We could for example use the *probit distribution*

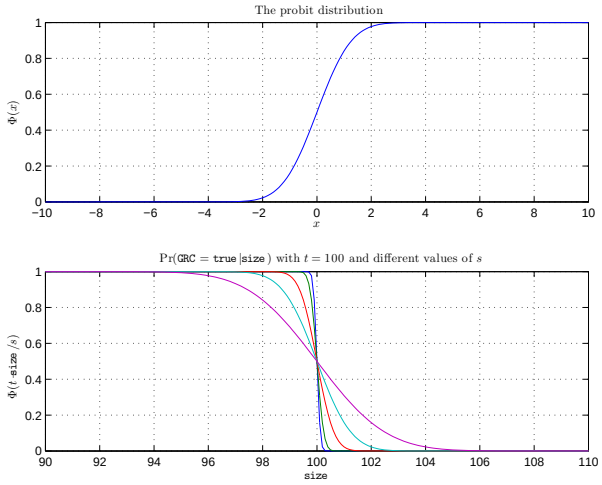
$$\Pr(\texttt{Go roofclimbing} = \texttt{true}|\texttt{size}) = \Phi\left(\frac{t - \texttt{size}}{s}\right)$$

where

$$\Phi(x) = \int_{-\infty}^{x} N(y)dy$$

and $N(x)$ is the Gaussian distribution with *zero mean and variance* 1.

More complex nodes



More complex nodes

Alternatively, for this example we could use the *logit distribution*

$$\Pr(\texttt{Go roofclimbing} = \texttt{true}|\texttt{size}) = \frac{1}{1 + e^{(-2(t-\texttt{size})/s)}}$$

which has a similar shape.

- Tails are longer for the logit distribution.
- The logit distribution tends to be easier to use...
- ...but the probit distribution is often more accurate.

Basic inference

We saw earlier that the full joint distribution can be used to perform *all inference tasks*:

$$\Pr(Q|e) = \frac{1}{Z}\Pr(Q \wedge e) = \frac{1}{Z}\sum_{u}\Pr(Q, e, u)$$

where

- Q is the query variable
- e is the evidence
- u are the unobserved variables
- $1/Z$ normalises the distribution.

Basic inference

As the BN fully describes the full joint distribution

$$\Pr(Q, u, e) = \prod_{i=1}^{n} \Pr(N_i|\text{parents}(N_i))$$

It can be used to perform inference in the *obvious* way

$$\Pr(Q|e) = \frac{1}{Z} \sum_{u} \prod_{i=1}^{n} \Pr(N_i|\text{parents}(N_i))$$

but as we'll see this is *in practice problematic*.

- More sophisticated algorithms aim to achieve this *more efficiently*.
- For complex BNs we resort to *approximation techniques*.

Other approaches to uncertainty: Default reasoning

One criticism made of probability is that it is *numerical* whereas human argument seems fundamentally different in nature:

- On the one hand this seems quite defensible. I certainly am not aware of doing *logical thought* through direct *manipulation of probabilities*, but...
- ...on the other hand, neither am I aware of *solving differential equations* in order to *walk*!

Default reasoning:

- Does not maintain *degrees of belief*.
- Allows something to be believed *until a reason is found not to*.

Other approaches to uncertainty: rule-based systems

Rule-based systems have some desirable properties:

- *Locality*: if we establish the evidence X and we have a rule $X \rightarrow Y$ then Y can be concluded regardless of any other rules.
- *Detachment*: once any Y has been established it can then be assumed. (It's justification is irrelevant.)
- *Truth-functionality*: truth of a complex formula is a function of the truth of its components.

These are not in general shared by probabilistic systems. What happens if:

- We try to attach measures of belief to rules and propositions.
- We try to make a truth-functional system by, for example, making belief in $X \wedge Y$ a function of beliefs in X and Y?

Other approaches to uncertainty: rule-based systems

Problems that can arise:

1. Say I have the causal rule

   $$\text{Heart disease} \xrightarrow{0.95} \text{Chest pain}$$

   and the diagnostic rule

   $$\text{Chest pain} \xrightarrow{0.7} \text{Heart disease}$$

   Without taking very great care to keep track of the reasoning process, these can form a *loop*.

2. If in addition I have

   $$\text{Chest pain} \xrightarrow{0.6} \text{Recent physical exertion}$$

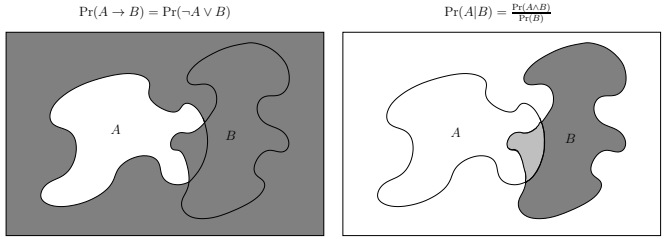
   then it is quite possible to form the conclusion that with some degree of certainty *heart disease is explained by exertion*, which may well be incorrect.

Other approaches to uncertainty: rule-based systems

In addition, we might argue that because heart disease is an explanation for chest pain the belief in physical exertion should *decrease*.

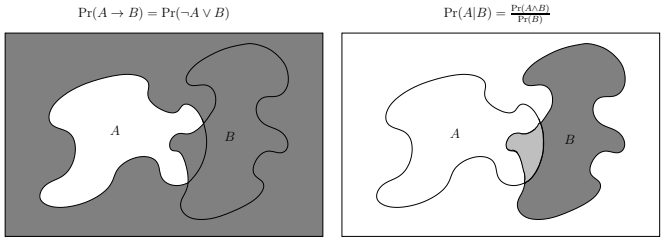
In general when such systems have been successful it has been through very careful control in setting up the rules.

In general, it is difficult to relate *implication* to *conditional probability*.



Implication and conditional probability

In general, it is difficult to relate *implication* to *conditional probability*.



Imagine that `fish` are very rare, and most fish can `swim`.

With implication,

$$\Pr(\texttt{fish} \to \neg\texttt{swim}) = \Pr(\neg\texttt{fish} \vee \neg\texttt{swim}) = \text{LARGE!}$$

With conditional probability,

$$\Pr(\neg\texttt{swim}|\texttt{fish}) = \frac{\Pr(\neg\texttt{swim} \wedge \texttt{fish})}{\Pr(\texttt{fish})} = \text{SMALL!}$$

Other approaches to uncertainty: Dempster-Shafer theory

Dempster-Shafer theory attempts to distinguish between *uncertainty* and *ignorance*.

Whereas the probabilistic approach looks at the *probability* of X, we instead look at the *probability* that the *available evidence supports* X.

This is denoted by the *belief function* $\text{Bel}(X)$.

Example: given a coin but no information as to whether it is fair I have no reason to think one outcome should be preferred to another

$$\text{Bel}(\text{outcome} = \text{head}) = \text{Bel}(\text{outcome} = \text{tail}) = 0$$

These beliefs can be updated when new evidence is available. If an expert tells us there is n percent certainty that it's a fair coin then

$$\text{Bel}(\text{outcome} = \text{head}) = \text{Bel}(\text{outcome} = \text{tail}) = \frac{n}{100} \times \frac{1}{2}.$$

We may still have a *gap* in that

$$\text{Bel}(\text{outcome} = \text{head}) + \text{Bel}(\text{outcome} = \text{tail}) \neq 1.$$

Dempster-Shafer theory provides a coherent system for dealing with belief functions.

Other approaches to uncertainty: Dempster-Shafer theory

Problems:

- The Bayesian approach deals more effectively with the quantification of how *belief changes* when *new evidence is available*.
- The Bayesian approach has a better connection to the concept of *utility*, whereas the latter is not well-understood for use in conjunction with Dempster-Shafer theory.

Uncertainty III: exact inference in Bayesian networks

We now examine:

- The basic equation for inference in Bayesian networks, the latter being hard to achieve if approached in the obvious way.
- The way in which matters can be improved a little by a small modification to the way in which the calculation is done.
- The way in which much better improvements might be possible using a still more informed approach, although not in all cases.

Reading: Russell and Norvig, chapter 14, section 14.4.

Performing exact inference

We know that in principle any query Q can be answered by the calculation

$$\mathbf{Pr}(Q|e) = \frac{1}{Z}\sum_{u}\mathbf{Pr}(Q, e, u)$$

where Q denotes the query, e denotes the evidence, u denotes unobserved variables and $1/Z$ normalises the distribution.

The naive implementation of this approach yields the *Enumerate-Joint-Ask* algorithm, which unfortunately requires $O(2^n)$ time and space for n Boolean random variables (RVs).

Performing exact inference

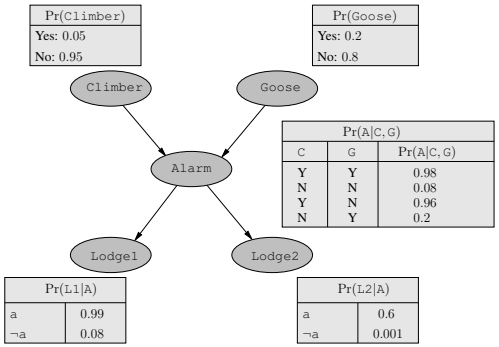
In what follows we will make use of some abbreviations.

- C denotes `Climber`
- G denotes `Goose`
- A denotes `Alarm`
- $L1$ denotes `Lodge1`
- $L2$ denotes `Lodge2`

Instead of writing out $\mathbf{Pr}(C = \top)$, $\mathbf{Pr}(C = \bot)$ *etc* we will write $\mathbf{Pr}(c)$, $\mathbf{Pr}(\neg c)$ and so on.

Performing exact inference

Also $\mathbf{Pr}(Q, e, u)$ has a particular form expressing conditional independences:



$$\mathbf{Pr}(C, G, A, L1, L2) = \mathbf{Pr}(C)\mathbf{Pr}(G)\mathbf{Pr}(A|C, G)\mathbf{Pr}(L1|A)\mathbf{Pr}(L2|A)$$

Performing exact inference

Consider the computation of the query $\mathbf{Pr}(C|l1, l2)$

We have

$$\mathbf{Pr}(C|l1, l2) = \frac{1}{Z}\sum_A \sum_G \mathbf{Pr}(C)\mathbf{Pr}(G)\mathbf{Pr}(A|C, G)\mathbf{Pr}(l1|A)\mathbf{Pr}(l2|A)$$

Here there are 5 multiplications for each set of values that appears for summation, and there are 4 such values.

In general this gives time complexity $O(n2^n)$ for n Boolean RVs.

Looking more closely we see that

$$\begin{aligned}\mathbf{Pr}(C|l1, l2) &= \frac{1}{Z}\sum_A \sum_G \mathbf{Pr}(C)\mathbf{Pr}(G)\mathbf{Pr}(A|C, G)\mathbf{Pr}(l1|A)\mathbf{Pr}(l2|A) \\ &= \frac{1}{Z}\mathbf{Pr}(C)\sum_A \mathbf{Pr}(l1|A)\mathbf{Pr}(l2|A)\sum_G \mathbf{Pr}(G)\mathbf{Pr}(A|C, G) \qquad (4) \\ &= \frac{1}{Z}\mathbf{Pr}(C)\sum_G \mathbf{Pr}(G)\sum_A \mathbf{Pr}(A|C, G)\mathbf{Pr}(l1|A)\mathbf{Pr}(l2|A)\end{aligned}$$

So for example...

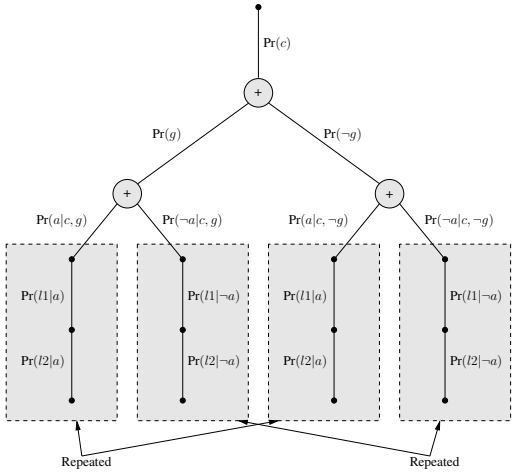
Performing exact inference

$$\begin{aligned}\mathbf{Pr}(c|l1, l2) = &\frac{1}{Z}\mathbf{Pr}(c)\left(\mathbf{Pr}(g)\left\{\begin{array}{c}\mathbf{Pr}(a|c, g)\mathbf{Pr}(l1|a)\mathbf{Pr}(l2|a) \\ +\mathbf{Pr}(\neg a|c, g)\mathbf{Pr}(l1|\neg a)\mathbf{Pr}(l2|\neg a)\end{array}\right\}\right. \\ &\left.+\mathbf{Pr}(\neg g)\left\{\begin{array}{c}\mathbf{Pr}(a|c, \neg g)\mathbf{Pr}(l1|a)\mathbf{Pr}(l2|a) \\ +\mathbf{Pr}(\neg a|c, \neg g)\mathbf{Pr}(l1|\neg a)\mathbf{Pr}(l2|\neg a)\end{array}\right\}\right)\end{aligned}$$

with a similar calculation for $\mathbf{Pr}(\neg c|l1, l2)$.

Basically straightforward, ***BUT*** optimisations can be made.

Performing exact inference



Optimisation 1: Enumeration-Ask

The *enumeration-ask* algorithm improves matters to $O(2^n)$ time and $O(n)$ space by performing the computation *depth-first*.

However matters can be improved further by avoiding the *duplication of computations* that clearly appears in the example tree.

Optimisation 2: variable elimination

Looking again at the fundamental equation (4)

$$\frac{1}{Z}\underbrace{\Pr(C)}_{C}\sum_{G}\underbrace{\Pr(G)}_{G}\sum_{A}\underbrace{\Pr(A|C,G)\Pr(l1|A)\Pr(l2|A)}_{A\quad L1\quad L2}$$

where C, G, A, $L1$, $L2$ denote the relevant *factors*.

The basic idea is to evaluate (4) from right to left (or in terms of the tree, bottom up) *storing results* as we progress and *re-using them* when necessary.

$\Pr(l1|A)$ depends on the value of A. We store it as a table $\mathbf{F}_{L1}(A)$. Similarly for $\Pr(l2|A)$.

$$\mathbf{F}_{L1}(A) = \begin{pmatrix} 0.99 \\ 0.08 \end{pmatrix} \quad \mathbf{F}_{L2}(A) = \begin{pmatrix} 0.6 \\ 0.001 \end{pmatrix}$$

as $\Pr(l1|a) = 0.99$, $\Pr(l1|\neg a) = 0.08$ and so on.

Optimisation 2: variable elimination

Similarly for $\Pr(A|C,G)$, which is dependent on A, C and G

$\mathbf{F}_A(A,C,G) =$

A	C	G	$\mathbf{F}_A(A,C,G)$
$\top$	$\top$	$\top$	0.98
$\top$	$\top$	$\bot$	0.96
$\top$	$\bot$	$\top$	0.2
$\top$	$\bot$	$\bot$	0.08
$\bot$	$\top$	$\top$	0.02
$\bot$	$\top$	$\bot$	0.04
$\bot$	$\bot$	$\top$	0.8
$\bot$	$\bot$	$\bot$	0.92

Can we write

$$\Pr(A|C,G)\Pr(l1|A)\Pr(l2|A) \tag{5}$$

as

$$\mathbf{F}_A(A,C,G)\mathbf{F}_{L1}(A)\mathbf{F}_{L2}(A) \tag{6}$$

in a reasonable way?

Optimisation 2: variable elimination

The answer is "yes" provided *multiplication of factors* is defined correctly. Looking at (4)

$$\frac{1}{Z}\Pr(C)\sum_{G}\Pr(G)\sum_{A}\Pr(A|C,G)\Pr(l1|A)\Pr(l2|A)$$

note that the values of the product (5) in the summation depend on the values of C and G external to it, and the values of A themselves. So (6) should be a table collecting values for (5) where correspondences between RVs are maintained.

This leads to a definition for multiplication of factors best given by example.

Optimisation 2: variable elimination

$$\mathbf{F}(A,B)\mathbf{F}(B,C) = \mathbf{F}(A,B,C)$$

where

A	B	$\mathbf{F}(A,B)$
$\top$	$\top$	0.3
$\top$	$\bot$	0.9
$\bot$	$\top$	0.4
$\bot$	$\bot$	0.1

B	C	$\mathbf{F}(B,C)$
$\top$	$\top$	0.1
$\top$	$\bot$	0.8
$\bot$	$\top$	0.8
$\bot$	$\bot$	0.3

A	B	C	$\mathbf{F}(A,B,C)$
$\top$	$\top$	$\top$	0.3×0.1
$\top$	$\top$	$\bot$	0.3×0.8
$\top$	$\bot$	$\top$	0.9×0.8
$\top$	$\bot$	$\bot$	0.9×0.3
$\bot$	$\top$	$\top$	0.4×0.1
$\bot$	$\top$	$\bot$	0.4×0.8
$\bot$	$\bot$	$\top$	0.1×0.8
$\bot$	$\bot$	$\bot$	0.1×0.3

Optimisation 2: variable elimination

This process gives us

$$\mathbf{F}_A(A,C,G)\mathbf{F}_{L1}(A)\mathbf{F}_{L2}(A) =$$

A	C	G	
$\top$	$\top$	$\top$	$0.98 \times 0.99 \times 0.6$
$\top$	$\top$	$\bot$	$0.96 \times 0.99 \times 0.6$
$\top$	$\bot$	$\top$	$0.2 \times 0.99 \times 0.6$
$\top$	$\bot$	$\bot$	$0.08 \times 0.99 \times 0.6$
$\bot$	$\top$	$\top$	$0.02 \times 0.08 \times 0.001$
$\bot$	$\top$	$\bot$	$0.04 \times 0.08 \times 0.001$
$\bot$	$\bot$	$\top$	$0.8 \times 0.08 \times 0.001$
$\bot$	$\bot$	$\bot$	$0.92 \times 0.08 \times 0.001$

Optimisation 2: variable elimination

How about

$$\mathbf{F}_{\overline{A},L1,L2}(C,G) = \sum_A \mathbf{F}_A(A,C,G)\mathbf{F}_{L1}(A)\mathbf{F}_{L2}(A)$$

To denote the fact that A has been summed out we place a bar over it in the notation.

$$\sum_A \mathbf{F}_A(A,C,G)\mathbf{F}_{L1}(A)\mathbf{F}_{L2}(A) = \mathbf{F}_A(a,C,G)\mathbf{F}_{L1}(a)\mathbf{F}_{L2}(a) + \mathbf{F}_A(\neg a,C,G)\mathbf{F}_{L1}(\neg a)\mathbf{F}_{L2}(\neg a)$$

where

$$\mathbf{F}_A(a,C,G) =$$

C	G	
$\top$	$\top$	0.98
$\top$	$\bot$	0.96
$\bot$	$\top$	0.2
$\bot$	$\bot$	0.08

$\mathbf{F}_{L1}(a) = 0.99$ $\mathbf{F}_{L2}(a) = 0.6$

and similarly for $\mathbf{F}_A(\neg a,C,G)$, $\mathbf{F}_{L1}(\neg a)$ and $\mathbf{F}_{L2}(\neg a)$.

Optimisation 2: variable elimination

$$\mathbf{F}_A(a,C,G)\mathbf{F}_{L1}(a)\mathbf{F}_{L2}(a) =$$

C	G	
$\top$	$\top$	$0.98 \times 0.99 \times 0.6$
$\top$	$\bot$	$0.96 \times 0.99 \times 0.6$
$\bot$	$\top$	$0.2 \times 0.99 \times 0.6$
$\bot$	$\bot$	$0.08 \times 0.99 \times 0.6$

$$\mathbf{F}_A(\neg a,C,G)\mathbf{F}_{L1}(\neg a)\mathbf{F}_{L2}(\neg a) =$$

C	G	
$\top$	$\top$	$0.02 \times 0.08 \times 0.001$
$\top$	$\bot$	$0.04 \times 0.08 \times 0.001$
$\bot$	$\top$	$0.8 \times 0.08 \times 0.001$
$\bot$	$\bot$	$0.92 \times 0.08 \times 0.001$

$$\mathbf{F}_{\overline{A},L1,L2}(C,G) =$$

C	G	
$\top$	$\top$	$(0.98 \times 0.99 \times 0.6) + (0.02 \times 0.08 \times 0.001)$
$\top$	$\bot$	$(0.96 \times 0.99 \times 0.6) + (0.04 \times 0.08 \times 0.001)$
$\bot$	$\top$	$(0.2 \times 0.99 \times 0.6) + (0.8 \times 0.08 \times 0.001)$
$\bot$	$\bot$	$(0.08 \times 0.99 \times 0.6) + (0.92 \times 0.08 \times 0.001)$

Optimisation 2: variable elimination

Now, say for example we have $\neg c$, g. Then doing the calculation explicitly would give

$$\sum_A \mathbf{Pr}(A|\neg c,g)\mathbf{Pr}(l1|A))\mathbf{Pr}(l2|A)$$
$$= \mathbf{Pr}(a|\neg c,g)\mathbf{Pr}(l1|a)\mathbf{Pr}(l2|a) + \mathbf{Pr}(\neg a|\neg c,g)\mathbf{Pr}(l1|\neg a)\mathbf{Pr}(l2|\neg a)$$
$$= (0.2 \times 0.99 \times 0.6) + (0.8 \times 0.08 \times 0.001)$$

which matches!

Continuing in this manner form

$$\mathbf{F}_{G,\overline{A},L1,L2}(C,G) = \mathbf{F}_G(G)\mathbf{F}_{\overline{A},L1,L2}(C,G)$$

sum out G to obtain $\mathbf{F}_{\overline{G,A},L1,L2}(C) = \sum_G \mathbf{F}_G(G)\mathbf{F}_{\overline{A},L1,L2}(C,G)$, form

$$\mathbf{F}_{C,\overline{G,A},L1,L2} = \mathbf{F}_C(C)\mathbf{F}_{\overline{G,A},L1,L2}(C)$$

and normalise.

Optimisation 2: variable elimination

What's the computational complexity now?

- For Bayesian networks with suitable structure we can perform inference in *linear* time and space.
- However in the worst case it is *#P-hard*, which is *worse than NP-hard*.

Consequently, we may need to resort to *approximate inference*.

Uncertainty IV: Simple Decision-Making

We now examine:

- The concept of a *utility function*.
- The way in which such functions can be related to reasonable axioms about *preferences*.
- A generalization of the Bayesian network, known as a *decision network*.
- How to measure the *value of information*, and how to use such measurements to design agents that can *ask questions*.

Reading: Russell and Norvig, chapter 16.

Simple decision-making

We now look at choosing an action by maximising *expected utility*.

A *utility function* $U(s)$ measures the *desirability* of a *state*.

If we can express a probability distribution for the states resulting from alternative actions, then we can act in order to maximise expected utility.

For an action a, let $\texttt{Result}(a) = \{s_1, \ldots, s_n\}$ be a set of states that might be the result of performing action a. Then the expected utility of a is

$$\texttt{EU}(a|E) = \sum_{s \in \texttt{Result}(a)} \Pr(s|a, E)U(s)$$

Note that this applies to *individual actions*. Sequences of actions will not be covered in this course.

Simple decision-making: all of AI?

Much as this looks like a complete and highly attractive method for an agent to decide how to act, it hides a great deal of complexity:

1. It may be hard to compute $U(s)$. You generally *don't know how good a state is until you know where it might lead on to*: planning *etc*...
2. Knowing what state you're currently in involves *most of AI*!
3. Dealing with $\Pr(s|a, E)$ involves *Bayesian networks*.

Utility in more detail

Overall, we now want to express *preferences* between different things.

Let's use the following notation:

$X > Y$: X is preferred to Y
$X = Y$: we are indifferent regarding X and Y
$X \geq Y$: X is preferred, or we're indifferent

X, Y and so on are *lotteries*. A lottery has the form

$$X = [p_1, O_1 | p_2, O_2 | \cdots | p_n, O_n]$$

where O_i are the outcomes of the lottery and p_i their respective probabilities. Outcomes can be *other lotteries* or actual states.

Axioms for utility theory

Given we are dealing with preferences it seems that there are some clear properties that such things should exhibit:

Transitivity: if $X > Y$ and $Y > Z$ then $X > Z$.

Orderability: either $X > Y$ or $Y > X$ or $X = Y$.

Continuity: if $X > Y > Z$ then there is a probability p such that

$$[p, X | (1-p), Z] = Y$$

Substitutability: if $X = Y$ then

$$[p, X | (1-p), L] = [p, Y | (1-p), L]$$

Axioms for utility theory

Monotonicity: if $X > Y$ then for probabilities p_1 and p_2, $p_1 \geq p_2$ if and only if

$$[p_1, X | (1-p_1), Y] \geq [p_2, X | (1-p_2), Y]$$

Decomposability:

$$[p_1, X | (1-p_1), [p_2, Y | (1-p_2), Z]] = [p_1, X | (1-p_1)p_2, Y | (1-p_1)(1-p_2), Z]$$

If an agent's preferences conform to the utility theory axioms—and note that we are *only* considering preferences, not numbers—then it is possible to define a utility function $U(s)$ for states such that:

1. $U(s_1) > U(s_2) \longleftrightarrow s_1 > s_2$
2. $U(s_1) = U(s_2) \longleftrightarrow s_1 = s_2$
3. $U([p_1, s_1 | p_2, s_2 | \cdots | p_n, s_n]) = \sum_{i=1}^{n} p_i U(s_i)$.

We therefore have a justification for the suggested approach.

Designing utility functions

There is complete freedom in how a utility function is defined, but clearly it will pay to define them carefully.

Example: the utility of money (for most people) exhibits a *monotonic preference*. That is, we *prefer* to have *more of it*.

But we need to talk about preferences between *lotteries*.

Say you've won $100,000$ pounds in a quiz and you're offered a coin flip:

- For heads: you win a total of $1,000,000$ pounds.
- For tails: you walk away with nothing!

Would you take the offer?

Designing utility functions

The *expected monetary value* (EMV) of this lottery is

$$(0.5 \times 1,000,000) + (0.5 \times 0) = 500,000$$

whereas the EMV of the initial amount is $100,000$.

BUT: most of us would probably refuse to take the coin flip.

The story is not quite as simple as this though: our attitude probably depends on *how much money we have to start with*. If I have M pounds to start with then I am in fact choosing between expected utility of

$$U(M + 100,000)$$

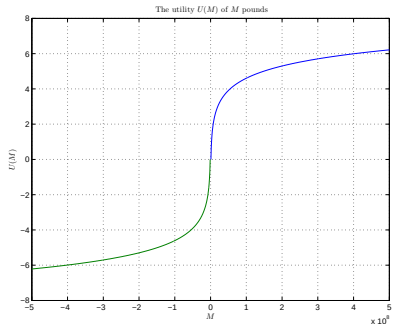
and expected utility of

$$(0.5 \times U(M)) + (0.5 \times U(M + 1,000,000))$$

If M is $50,000,000$ my attitude is much different to if it is $10,000$.

Designing utility functions

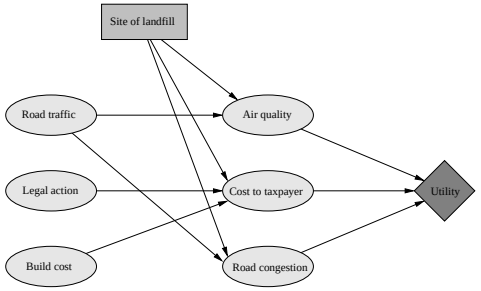
In fact, research shows that the utility of M pounds is for most people almost exactly proportional to $\log M$ for $M > 0$...



... and follows a similar shape for $M < 0$.

Decision networks

Decision networks—also known as *influence diagrams*...



... allow us to work *actions* and *utilities* into the formalism of *Bayesian networks*.

A decision network has three types of node...

Decision networks

A decision network has three types of node:

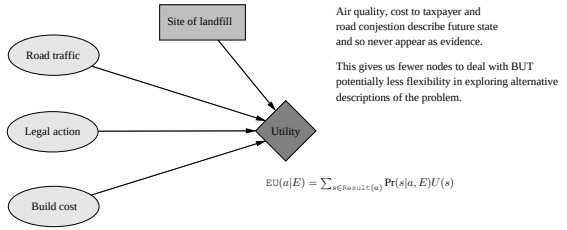
Chance nodes: are denoted by ovals. These are random variables (RVs) represented by a distribution conditional on their parents, as in Bayesian networks. Parents can be other chance nodes or a decision node.

Decision nodes: are denoted by squares. They describe possible outcomes of the decision of interest. Here we deal only with *single* decisions: multiple decisions require alternative techniques.

Utility nodes: are denoted by diamonds. They describe the utility function relevant to the problem, as a function of the values of the node's parents.

Decision networks

Sometimes such diagrams are simplified by leaving out the RVs describing the new state and converting current state and decision directly to utility:



Air quality, cost to taxpayer and road conjestion describe future state and so never appear as evidence.

This gives us fewer nodes to deal with BUT potentially less flexibility in exploring alternative descriptions of the problem.

This is an *action-utility table*. The utility no longer depends on a state but is the expected utility for a given action.

Evaluation of decision networks

Once a *specific* action is selected for a decision node it acts like a chance node for which a specific value is being used as *evidence*.

1. Set the current state chance nodes to their evidence values.
2. For each potential action
   - Fix the decision node.
   - Compute the probabilities for the utility node's parents.
   - Compute the expected utility.
3. Return the action that maximised $\texttt{EU}(a|E)$.

The value of information

We have been assuming that a decision is to be made with *all evidence available beforehand*. This is unlikely to be the case.

Knowing *what questions one should ask* is a central, and important part of making decisions. *Example*:

- Doctors do not diagnose by first obtaining results for all possible tests on their patients.
- They ask questions to decide what tests to do.
- They are informed in formulating which tests to perform by probabilities of test outcomes, and by the manner in which knowing an outcome might improve treatment.
- Tests can have associated costs.

The value of perfect information

Information value theory provides a formal way in which we can reason about what further information to gather using *sensing actions*.

Say we have evidence E, so

$$\texttt{EU}(\texttt{action}|E) = \max_a \sum_{s \in \texttt{Result}(a)} \Pr(s|a,E)U(s)$$

denotes how valuable the best action based on E must be.

How valuable would it be to learn about a *further piece of evidence*?

If we examined another RV E' and found that $E' = e'$ then the *best action might be altered* as we'd be computing

$$\texttt{EU}(\texttt{action}'|E,E') = \max_a \sum_{s \in \texttt{Result}(a)} \Pr(s|a,E,E')U(s)$$

BUT: because E' is a RV, and in advance of testing we don't know its value, we need to *average* over its *possible values* using our *current knowledge*.

The value of perfect information

This leads to the definition of the *value of perfect information* (VPI)

$$\texttt{VPI}_E(E') = \left\{ \sum_{e'} \textbf{Pr}(E' = e'|E)\texttt{EU}(\texttt{action}'|E, E' = e') \right\} - \texttt{EU}(\texttt{action}|E)$$

VPI has the following properties:

- $\texttt{VPI}_E(E') \geq 0$
- It is not necessarily additive, that is, it is possible that

$$\texttt{VPI}_E(E', E'') \neq \texttt{VPI}_E(E') + \texttt{VPI}_E(E'')$$

- It is independent of ordering

$$\begin{aligned}\texttt{VPI}_E(E', E'') &= \texttt{VPI}_E(E') + \texttt{VPI}_{E,E'}(E'') \\ &= \texttt{VPI}_E(E'') + \texttt{VPI}_{E,E''}(E')\end{aligned}$$

Agents that can gather information

In constructing an agent with the ability to ask questions, we would hope that it would:

- Use a good order in which to ask the questions.
- Avoid asking irrelevant questions.
- Trade off the *cost* of obtaining information against the *value* of that information.
- Choose a good time to *stop* asking questions.

We now have the means with which to approach such a design.

Agents that can gather information

Assuming we can associate a cost $C(E')$ with obtaining the knowledge that $E' = e'$ an agent can act as follows:

- Given a decision network and current percept.
- Find the piece of evidence E' maximising $\texttt{VPI}_E(E') - C(E')$.
- If $\texttt{VPI}_E(E') - C(E')$ is positive then find the value of E', else take the action indicated by the decision network.

This is known as a *myopic* agent as it requests a single piece of evidence at once.

Uncertainty V: probabilistic reasoning through time

We now examine:

- How an agent might operate by keeping track of the state of its environment in an uncertain world, and how alterations in world state and uncertainty in observing the world can be modelled using probability distributions.
- How inferences can be performed regarding the current state, past state and future states.
- The *Viterbi algorithm* for computing the most likely sequence.
- A slightly simplified system within this framework called a *hidden Markov model* (HMM), and the way in which some inference tasks can be simplified in the HMM case.

Reading: Russell and Norvig, chapter 15.

Probabilistic reasoning through time

A fundamental idea throughout the AI courses has been that an agent should keep track of the *state of the environment*:

- The environment's state *changes over time*.
- The knowledge of *how the state changes* may be *uncertain*.
- The agent's *perception* of the state of the environment *may be uncertain*.

For all the usual reasons related to *uncertainty*, we need to move beyond logic, situation calculus *etc*.

States and evidence

We model the (unobservable) state of the environment as follows:

- We use a *sequence*
$$(S_0, S_1, S_2, \ldots)$$
of *sets* of *random variables* (RVs).
- Each S_t is a *set* of RVs
$$S_t = \{S_t^{(1)}, \ldots, S_t^{(n)}\}$$
denoting the state of the environment at time t, where $t = 0, 1, 2, \ldots$.

Think of the state as changing over time.

$$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \cdots$$

States and evidence

At each time t there is also an *observable* set

$$E_t = \{E_t^{(1)}, \ldots, E_t^{(m)}\}$$

of random variables denoting the *evidence that an agent obtains about the state* at time t.

As usual capitals denote RVs and lower case denotes actual values. So actual values for the assorted RVs are denoted

$$S_t = \{s_t^{(1)}, \ldots, s_t^{(n)}\} = s_t$$
$$E_t = \{e_t^{(1)}, \ldots, e_t^{(m)}\} = e_t$$

Stationary and Markov processes

As t can in principle increase without bound we now need some simplifying assumptions.

Assumption 1: We deal with *stationary processes*: probability distributions do not change over time.

Assumption 2: We deal with *Markov processes*

$$\mathbf{Pr}(S_t|S_{0:t-1}) = \mathbf{Pr}(S_t|S_{t-1}) \tag{7}$$

where $S_{0:t-1} = (S_0, S_1, \ldots, S_{t-1})$.

(Strictly speaking this is a *first order Markov Process*, and we'll only consider these.)

$\mathbf{Pr}(S_t|S_{t-1})$ is called the *transition model*.

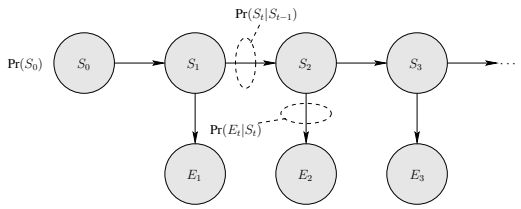
Stationary and Markov processes

Assumption 3: We assume that evidence only depends on the current state

$$\Pr(E_t|S_{0:t}, E_{1:t-1}) = \Pr(E_t|S_t) \tag{8}$$

Then

$\Pr(E_t|S_t)$ is called the *sensor model*.



$\Pr(S_0)$ is the *prior probability* of the starting state. We need this as there has to be some way of getting the process started.

The full joint distribution

Given:

1. The prior $\Pr(S_0)$.
2. The transition model $\Pr(S_t|S_{t-1})$.
3. The sensor model $\Pr(E_t|S_t)$.

along with the assumptions of stationarity and the assumptions of independence in equations 7 and 8 we have

$$\Pr(S_0, S_1, \ldots, S_t, E_1, E_2, \ldots, E_t) = \Pr(S_0) \prod_{i=1}^{t} \Pr(S_i|S_{i-1})\Pr(E_i|S_i) .$$

This follows from basic probability theory as for example

$$\begin{aligned}\Pr(S_0, S_1, S_2, E_1, E_2) &= \Pr(E_2|S_{0:2}, E_1)\Pr(S_2|S_{0:1}, E_1)\Pr(E_1|S_{0:1})\Pr(S_1|S_0)\Pr(S_0) \\ &= \Pr(E_2|S_2)\Pr(S_2|S_1)\Pr(E_1|S_1)\Pr(S_1|S_0)\Pr(S_0)\end{aligned}$$

Example: two biased coins

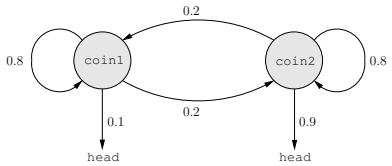
Here's a simple example with only *two states* and *two observations*.

I have *two biased coins*.

I *flip one* and *tell you the outcome*.

I then either *stay* with the same coin, or *swap* them.

This continues, producing a succession of outcomes:



Example: two biased coins

We'll use the following numbers:

- The prior $\Pr(S_0 = \texttt{coin1}) = 0.5$.
- The transition model

  $\Pr(S_t = \texttt{coin1}|S_{t-1} = \texttt{coin1}) = \Pr(S_t = \texttt{coin2}|S_{t-1} = \texttt{coin2}) = 0.8$
  $\Pr(S_t = \texttt{coin1}|S_{t-1} = \texttt{coin2}) = \Pr(S_t = \texttt{coin2}|S_{t-1} = \texttt{coin1}) = 0.2$

- The sensor model

  $\Pr(E_t = \texttt{head}|S_t = \texttt{coin1}) = 0.1$
  $\Pr(E_t = \texttt{head}|S_t = \texttt{coin2}) = 0.9$

Example: two biased coins

This is straightforward to simulate.

Here's an example of what happens:

```
[C2,C2,C1,C1,C1,C1,C1,C1,C1,C1,C1,C2,C1,C1,C1,C1,C1,C1,C1,C2,C2,C2,C2,C2,C2,C2,C2,C2,C2,C2,C2,C2,C2,C2,C2,C2,C2,C2,C2,C2,C2]

⇓

[Hd,T1,T1,T1,Hd,T1,Hd,T1,T1,T1,Hd,T1,Hd,T1,Hd,T1,T1,T1,T1,T1,Hd,T1,T1,Hd,Hd,Hd,Hd,Hd,Hd,Hd,Hd,T1,Hd,Hd,Hd,Hd,Hd,Hd,Hd,Hd,T1,Hd]
```

As expected, we tend to see runs of a single coin, and might expect to be able to guess which is being used as one favours heads and the other tails.

Example: 2008, paper 9, question 5

A friend of mine likes to climb on the roofs of Cambridge. To make a good start to the coming week, he climbs on a Sunday with probability 0.98. Being concerned for his own safety, he is less likely to climb today if he climbed yesterday, so

$$\Pr(\texttt{climb today}|\texttt{climb yesterday}) = 0.4$$

If he did not climb yesterday then he is very unlikely to climb today, so

$$\Pr(\texttt{climb today}|\neg\texttt{climb yesterday}) = 0.1$$

Unfortunately, he is not a very good climber, and is quite likely to injure himself if he goes climbing, so

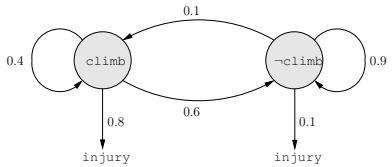
$$\Pr(\texttt{injury}|\texttt{climb today}) = 0.8$$

whereas

$$\Pr(\texttt{injury}|\neg\texttt{climb today}) = 0.1$$

Example: 2008, paper 9, question 5

This has a similar corresponding diagram:



We'll look at the rest of this exam question later.

Performing inference

There are four basic inference tasks that we might want to perform.

In each of the following cases, assume that we have observed the evidence

$$E_{1:t} = e_{1:t}$$

Task 1: *filtering*

Deduce what state we might now be in by computing

$$\Pr(S_t|e_{1:t}).$$

In the coin tossing question: *"If you've seen all the outcomes so far, infer which coin was used last"*.

In the exam question: *"If you observed all the injuries so far, infer whether my friend climbed today"*.

Performing inference

Task 2: *prediction*

Deduce what state we might be in some time in the future by computing

$$\Pr(S_{t+T}|e_{1:t}) \text{ for some } T > 0.$$

In the coin tossing question: *"If you've seen all the outcomes so far, infer which coin will be tossed T steps in the future"*.

In the exam question: *"If you've observed all the injuries so far, infer whether my friend will go climbing T nights from now"*.

Performing inference

Task 3: *Smoothing*

Deduce what state we might have been in at some point in the past by computing

$$\Pr(S_t|e_{1:T}) \text{ for } 0 \leq t < T.$$

In the coin tossing question: *"If you've seen all the outcomes so far, infer which coin was tossed at time t in the past"*.

In the exam question: *"If you've observed all the injuries so far, infer whether my friend climbed on night t in the past"*.

Performing inference

Task 4: *Find the most likely explanation*

Deduce the most likely sequence of states so far by computing

$$\operatorname*{argmax}_{s_{1:t}} \Pr(s_{1:t}|e_{1:t})$$

In the coin tossing question: *"If you've seen all the outcomes so far, infer the most probable sequence of coins used"*.

In the exam question: *"If you've observed all the injuries so far, infer the most probable collection of nights on which my friend climbed"*.

Filtering

We want to compute $\Pr(S_t|e_{1:t})$. This is often called the *forward message* and denoted

$$f_{1:t} = \Pr(S_t|e_{1:t})$$

for reasons that are about to become clear.

Remember that S_t is an RV and so $f_{1:t}$ is a *probability distribution* containing a probability for each possible value of S_t.

It turns out that this can be done in a simple manner with a *recursive estimation*. Obtain the result at time $t + 1$:

1. using the result from time t and...
2. ...incorporating new evidence e_{t+1}.

$$f_{1:t+1} = g(e_{t+1}, f_{1:t})$$

for a suitable function g that we'll now derive.

Filtering

Step 1:

Project the current state distribution forward

$$\begin{aligned}\Pr(S_{t+1}|e_{1:t+1}) &= \Pr(S_{t+1}|e_{1:t}, e_{t+1}) \\ &= c\Pr(e_{t+1}|S_{t+1}, e_{1:t})\Pr(S_{t+1}|e_{1:t}) \\ &= c\underbrace{\Pr(e_{t+1}|S_{t+1})}_{\text{Sensor model}}\underbrace{\Pr(S_{t+1}|e_{1:t})}_{\text{Needs more work}}\end{aligned}$$

where as usual c is a constant that normalises the distribution. Here,

- The first line does nothing but split $e_{1:t+1}$ into e_{t+1} and $e_{1:t}$.
- The second line is an application of Bayes' theorem.
- The third line uses *assumption 3* regarding sensor models.

Filtering

Step 2:

To obtain $\Pr(S_{t+1}|e_{1:t})$

$$\begin{aligned}\Pr(S_{t+1}|e_{1:t}) &= \sum_{s_t}\Pr(S_{t+1}, s_t|e_{1:t}) \\ &= \sum_{s_t}\Pr(S_{t+1}|s_t, e_{1:t})\Pr(s_t|e_{1:t}) \\ &= \sum_{s_t}\underbrace{\Pr(S_{t+1}|s_t)}_{\text{Transition model}}\quad\underbrace{\Pr(s_t|e_{1:t})}_{\text{Available from previous step}}\end{aligned}$$

Here,

- The first line uses marginalisation.
- The second line uses the basic equation $\Pr(A, B) = \Pr(A|B)\Pr(B)$.
- The third line uses *assumption 2* regarding transition models.

Filtering

Pulling it all together

$$\boxed{\Pr(S_{t+1}|e_{1:t+1}) = c\underbrace{\Pr(e_{t+1}|S_{t+1})}_{\text{Sensor model}}\sum_{s_t}\underbrace{\Pr(S_{t+1}|s_t)}_{\text{Transition model}}\quad\underbrace{\Pr(s_t|e_{1:t})}_{\text{From previous step}}} \tag{9}$$

This will be shortened to

$$f_{1:t+1} = c\text{FORWARD}(e_{t+1}, f_{1:t})$$

Here

- $f_{1:t}$ is a shorthand for $\Pr(S_t|e_{1:t})$.
- $f_{1:t}$ is often interpreted as a *message* being passed forward.
- The process is started using the *prior*.

Prediction

Prediction is somewhat simpler as

$$\begin{aligned}\boxed{\underbrace{\Pr(S_{t+T+1}|e_{1:t})}_{\text{Prediction at } t+T+1}} &= \sum_{s_{t+T}}\Pr(S_{t+T+1}, s_{t+T}|e_{1:t}) \\ &= \sum_{s_{t+T}}\Pr(S_{t+T+1}|s_{t+T}, e_{1:t})\Pr(s_{t+T}|e_{1:t}) \\ &= \boxed{\sum_{s_{t+T}}\underbrace{\Pr(S_{t+T+1}|s_{t+T})}_{\text{Transition model}}\underbrace{\Pr(s_{t+T}|e_{1:t})}_{\text{Prediction at } t+T}}\end{aligned}$$

However we do not get to make accurate predictions arbitrarily far into the future!

Smoothing

For smoothing, we want to calculate $\Pr(S_t|e_{1:T})$ for $0 \leq t < T$.

Again, we can do this in two steps.

Step 1:

$$\begin{aligned}\Pr(S_t|e_{1:T}) &= \Pr(S_t|e_{1:t}, e_{t+1:T}) \\ &= c\Pr(S_t|e_{1:t})\Pr(e_{t+1:T}|S_t, e_{1:t}) \\ &= c\Pr(S_t|e_{1:t})\Pr(e_{t+1:T}|S_t) \\ &= cf_{1:t}b_{t+1:T}\end{aligned}$$

Here

- $f_{1:t}$ is the forward message defined earlier.
- $b_{t+1:T}$ is a shorthand for $\Pr(e_{t+1:T}|S_t)$ to be regarded as *a message being passed backward*.

Smoothing

Step 2:

$$\begin{aligned}\boxed{b_{t+1:T}} = \Pr(e_{t+1:T}|S_t) &= \sum_{s_{t+1}} \Pr(e_{t+1:T}, s_{t+1}|S_t) \\ &= \sum_{s_{t+1}} \Pr(e_{t+1:T}|s_{t+1})\Pr(s_{t+1}|S_t) \\ &= \sum_{s_{t+1}} \Pr(e_{t+1}, e_{t+2:T}|s_{t+1})\Pr(s_{t+1}|S_t) \\ &= \sum_{s_{t+1}} \underbrace{\Pr(e_{t+1}|s_{t+1})}_{\text{Sensor model}}\underbrace{\Pr(e_{t+2:T}|s_{t+1})}_{b_{t+2:T}}\underbrace{\Pr(s_{t+1}|S_t)}_{\text{Transition model}} \\ &= \boxed{\text{BACKWARD}(e_{t+1:T}, b_{t+2:T})}\end{aligned} \tag{10}$$

This process is initialised with

$$b_{t+1:t} = \Pr(e_{T+1:T}|S_T) = (1, \ldots, 1)$$

The forward-backward algorithm

So: our original aim of computing $\Pr(S_t|e_{1:T})$ can be achieved using:

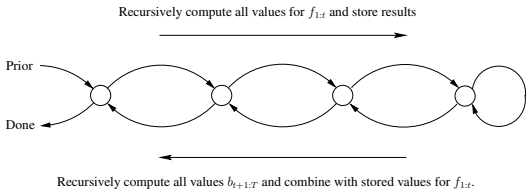
- A recursive process working from time 1 to time t (equation 9).
- A recursive process working from time T to time $t+1$ (equation 10).

This results in a process that is $O(T)$ given the evidence $e_{1:T}$ and smooths for a *single* point at time t.

To smooth at *all* points $1:T$ we can easily repeat the process obtaining $O(T^2)$.

Alternatively a very simple example of *dynamic programming* allows us to smooth at all points in $O(T)$ time.

The forward-backward algorithm



Computing the most likely sequence: the Viterbi algorithm

In computing the most likely sequence the aim is to obtain

$$\underset{s_{1:t}}{\operatorname{argmax}} \mathbf{Pr}(s_{1:t}|e_{1:t})$$

Earlier we derived the joint distribution for all relevant variables

$$\mathbf{Pr}(S_0, S_1, \ldots, S_t, E_1, E_2, \ldots, E_t) = \mathbf{Pr}(S_0) \prod_{i=1}^{t} \mathbf{Pr}(S_i|S_{i-1})\mathbf{Pr}(E_i|S_i)$$

Computing the most likely sequence: the Viterbi algorithm

We therefore have

$$\begin{aligned}
&\max_{s_{1:t}} \mathbf{Pr}(s_{1:t}, S_{t+1}|e_{1:t+1}) \\
&= c \max_{s_{1:t}} \mathbf{Pr}(e_{t+1}|S_{t+1})\mathbf{Pr}(S_{t+1}|s_t)\mathbf{Pr}(s_{1:t}|e_{1:t}) \\
&= c\mathbf{Pr}(e_{t+1}|S_{t+1}) \max_{s_t} \left\{ \mathbf{Pr}(S_{t+1}|s_t) \max_{s_{1:t-1}} \mathbf{Pr}(s_{1:t-1}, s_t|e_{1:t}) \right\}
\end{aligned}$$

This looks *a bit fierce*, despite the fact that:

- The second line is just Bayes' theorem applied to the joint distribution.
- The last line is just a re-arrangement of the second line.

Computing the most likely sequence: the Viterbi algorithm

There is however a way to visualise it that leads to a dynamic programming algorithm called the *Viterbi algorithm*.

Step 1: Simplify the notation.

- Assume there are n states $s_1, \ldots, s_n$ and m possible observations $e_1, \ldots, e_m$ at any given time.
- Denote $\mathbf{Pr}(S_t = s_j|S_{t-1} = s_i)$ by $p_{i,j}(t)$.
- Denote $\mathbf{Pr}(e_t|S_t = s_i)$ by $q_i(t)$.

It's important to remember in what follows that the *observations are known* but that we're *maximising over all possible state sequences*.

Computing the most likely sequence: the Viterbi algorithm

The equation we're interested in is now of the form

$$P = \prod_{t=1}^{T} p_{i,j}(t)q_i(t)$$

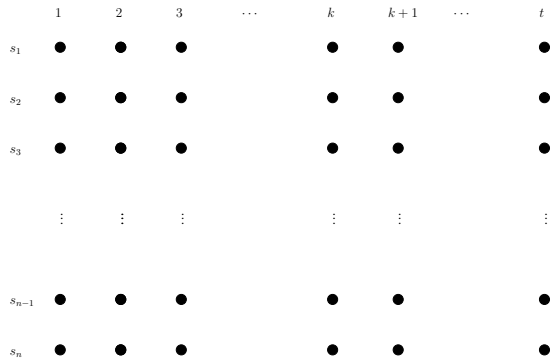
(The prior $\mathbf{Pr}(S_0)$ has been dropped out for the sake of clarity, but is easy to put back in in what follows.)

The equation P will be referred to in what follows.

It is in fact a *function* of *any given sequence of states*.

Computing the most likely sequence: the Viterbi algorithm

Step 2: Make a grid: columns denote time and rows denote state.



Computing the most likely sequence: the Viterbi algorithm

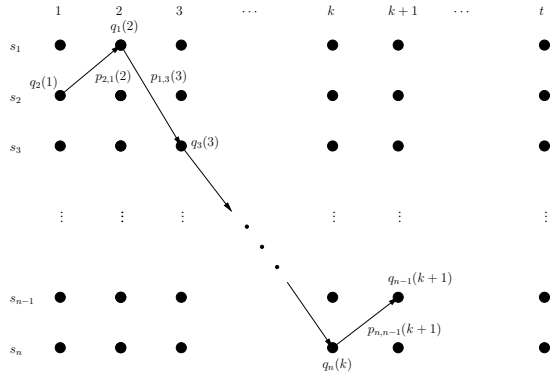
Step 3: Label the nodes:

- Say at time t the actual observation was e_t. Then label the node for s_i in column t with the value $q_i(t)$.
- Any sequence of states through time is now a path through the grid. So for any transition from s_i at time $t-1$ to s_j at time t label the transition with the value $p_{i,j}(t)$.

In the following diagrams we can often just write $p_{i,j}$ and q_i because the time is clear from the diagram.

So for instance...

Computing the most likely sequence: the Viterbi algorithm

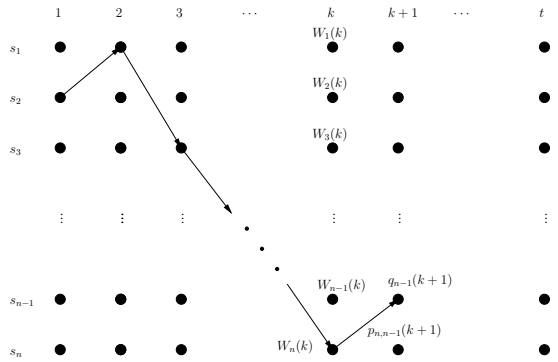


Computing the most likely sequence: the Viterbi algorithm

- The value of $P = \prod_{t=1}^{T} p_{i,j}(t)q_i(t)$ for any path through the grid is just the product of the corresponding labels that have been added.
- But we don't want to find the maximum by looking at all the possible paths because this would be time-consuming.
- The *Viterbi algorithm* computes the maximum by moving from one column to the next updating as it goes.
- Say you're at column k and *for each node* m in that column you know the *highest value* for the product to this point over *any possible path*. Call this:

$$W_m(k) = \max_{s_{1:k}} \prod_{t=1}^{k} p_{i,j}(t)q_i(t)$$

Computing the most likely sequence: the Viterbi algorithm



Computing the most likely sequence: the Viterbi algorithm

Here is the key point: you only need to know

- The values $W_i(k)$ for $i = 1, \ldots, n$ at time k.
- The numbers $p_{i,j}(k+1)$.
- The numbers $q_i(k+1)$.

to compute the values $W_i(k+1)$ for the next column $k+1$.

This is because

$$\boxed{W_i(k+1) = \max_j W_j(k) p_{j,i}(k+1) q_i(k+1)}$$

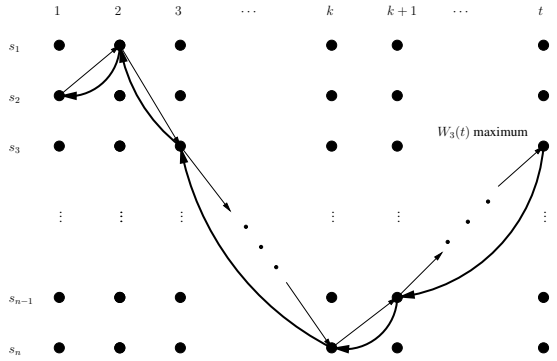
Computing the most likely sequence: the Viterbi algorithm

Once you get to the column for time t:

- The node with the largest value for $W_i(t)$ tells you the largest possible value of P.
- Provided you stored *the path taken to get there* you can *work backwards* to find *the corresponding sequence of states*.

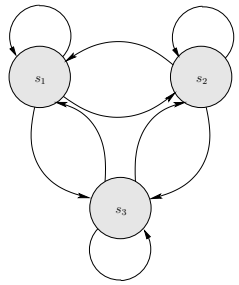
This is the *Viterbi algorithm*.

Computing the most likely sequence: the Viterbi algorithm



Hidden Markov models

Now for a specific case: hidden Markov models (HMMs). Here we have a *single*, *discrete* state variable S_i taking values $s_1, s_2, \ldots, s_n$. For example, with $n = 3$ we might have



	$\Pr(S_{t+1}\|S_t = s_1)$	$\Pr(S_{t+1}\|S_t = s_2)$	$\Pr(S_{t+1}\|S_t = s_3)$
s_1	0.3	0.2	0.2
s_2	0.1	0.6	0.3
s_3	0.6	0.2	0.5

Hidden Markov models

In this simplified case the conditional probabilities $\Pr(S_{t+1}|S_t)$ can be represented using the matrix

$$S_{ij} = \Pr(S_{t+1} = s_j | S_t = s_i)$$

or for the example on the previous slide

$$\mathbf{S} = \begin{pmatrix} 0.3 & 0.1 & 0.6 \\ 0.2 & 0.6 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{pmatrix} \begin{matrix} \leftarrow \Pr(S|s_1) \\ \leftarrow \Pr(S|s_2) \\ \leftarrow \Pr(S|s_3) \end{matrix}$$
$$= \begin{pmatrix} \Pr(s_1|s_1) & \Pr(s_2|s_1) & \cdots & \Pr(s_n|s_1) \\ \Pr(s_1|s_2) & \Pr(s_2|s_2) & \cdots & \Pr(s_n|s_2) \\ \vdots & \vdots & \ddots & \vdots \\ \Pr(s_1|s_n) & \Pr(s_2|s_n) & \cdots & \Pr(s_n|s_n) \end{pmatrix}$$

To save space, I am abbreviating $\Pr(S_{t+1} = s_i | S_t = s_j)$ to $\Pr(s_i|s_j)$.

Hidden Markov models

The computations we're making are always conditional on some actual observations $e_{1:T}$.

For each t we can therefore use the sensor model to define a further matrix $\mathbf{E}_t$:

- $\mathbf{E}_t$ is square and diagonal (all off-diagonal elements are 0).
- The ith element of the diagonal is $\Pr(e_t|S_t = s_i)$.

So in our present example with 3 states, there will be a matrix

$$\mathbf{E}_t = \begin{pmatrix} \Pr(e_t|s_1) & 0 & 0 \\ 0 & \Pr(e_t|s_2) & 0 \\ 0 & 0 & \Pr(e_t|s_3) \end{pmatrix}$$

for each $t = 1, \ldots, T$.

Hidden Markov models

In the general case the equation for filtering was

$$\Pr(S_{t+1}|e_{1:t+1}) = c\Pr(e_{t+1}|S_{t+1}) \sum_{s_t} \Pr(S_{t+1}|s_t)\Pr(s_t|e_{1:t})$$

and the message $f_{1:t}$ was introduced as a representation of $\Pr(S_t|e_{1:t})$.

In the present case we can define $f_{1:t}$ to be the vector

$$f_{1:t} = \begin{pmatrix} \Pr(s_1|e_{1:t}) \\ \Pr(s_2|e_{1:t}) \\ \vdots \\ \Pr(s_n|e_{1:t}) \end{pmatrix}$$

Key point: *the filtering equation now reduces to nothing but matrix multiplication*.

What does matrix multiplication do?

What does matrix multiplication do? *It computes weighted summations*:

$$\mathbf{Ab} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^m a_{1,i} b_i \\ \sum_{i=1}^m a_{2,i} b_i \\ \vdots \\ \sum_{i=1}^m a_{n,i} b_i \end{pmatrix}$$

So the point at the end of the last slide shouldn't come as a big surprise!

Hidden Markov models

Now, note that if we have n states

$$\begin{aligned} \mathbf{S}^T f_{1:t} &= \begin{pmatrix} \mathbf{Pr}(s_1|s_1) & \cdots & \mathbf{Pr}(s_1|s_n) \\ \mathbf{Pr}(s_2|s_1) & \cdots & \mathbf{Pr}(s_2|s_n) \\ \vdots & \ddots & \vdots \\ \mathbf{Pr}(s_n|s_1) & \cdots & \mathbf{Pr}(s_n|s_n) \end{pmatrix} \begin{pmatrix} \mathbf{Pr}(s_1|e_{1:t}) \\ \mathbf{Pr}(s_2|e_{1:t}) \\ \vdots \\ \mathbf{Pr}(s_n|e_{1:t}) \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{Pr}(s_1|s_1)\mathbf{Pr}(s_1|e_{1:t}) + \cdots + \mathbf{Pr}(s_1|s_n)\mathbf{Pr}(s_n|e_{1:t}) \\ \mathbf{Pr}(s_2|s_1)\mathbf{Pr}(s_1|e_{1:t}) + \cdots + \mathbf{Pr}(s_2|s_n)\mathbf{Pr}(s_n|e_{1:t}) \\ \vdots \\ \mathbf{Pr}(s_n|s_1)\mathbf{Pr}(s_1|e_{1:t}) + \cdots + \mathbf{Pr}(s_n|s_n)\mathbf{Pr}(s_n|e_{1:t}) \end{pmatrix} \\ &= \begin{pmatrix} \sum_s \mathbf{Pr}(s_1|s)\mathbf{Pr}(s|e_{1:t}) \\ \sum_s \mathbf{Pr}(s_2|s)\mathbf{Pr}(s|e_{1:t}) \\ \vdots \\ \sum_s \mathbf{Pr}(s_n|s)\mathbf{Pr}(s|e_{1:t}) \end{pmatrix} \end{aligned}$$

Hidden Markov models

And taking things one step further

$$\begin{aligned} \mathbf{E}_{t+1}\mathbf{S}^T f_{1:t} &= \begin{pmatrix} \mathbf{Pr}(e_{t+1}|s_1) & & 0 \\ & \ddots & \\ 0 & & \mathbf{Pr}(e_{t+1}|s_n) \end{pmatrix} \begin{pmatrix} \sum_s \mathbf{Pr}(s_1|s)\mathbf{Pr}(s|e_{1:t}) \\ \sum_s \mathbf{Pr}(s_2|s)\mathbf{Pr}(s|e_{1:t}) \\ \vdots \\ \sum_s \mathbf{Pr}(s_n|s)\mathbf{Pr}(s|e_{1:t}) \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{Pr}(e_{t+1}|s_1) \sum_s \mathbf{Pr}(s_1|s)\mathbf{Pr}(s|e_{1:t}) \\ \mathbf{Pr}(e_{t+1}|s_2) \sum_s \mathbf{Pr}(s_2|s)\mathbf{Pr}(s|e_{1:t}) \\ \vdots \\ \mathbf{Pr}(e_{t+1}|s_n) \sum_s \mathbf{Pr}(s_n|s)\mathbf{Pr}(s|e_{1:t}) \end{pmatrix} \end{aligned}$$

Compare this with the equation for filtering

$$\mathbf{Pr}(S_{t+1}|e_{1:t+1}) = c\mathbf{Pr}(e_{t+1}|S_{t+1}) \sum_{s_t} \mathbf{Pr}(S_{t+1}|s_t)\mathbf{Pr}(s_t|e_{1:t})$$

Hidden Markov models

Comparing the expression for $\mathbf{E}_{t+1}\mathbf{S}^T f_{1:t}$ with the equation for filtering we see that

$$\boxed{f_{1:t+1} = c\mathbf{E}_{t+1}\mathbf{S}^T f_{1:t}}$$

and a similar equation can be found for b

$$\boxed{b_{t+1:T} = \mathbf{S}\mathbf{E}_{t+1} b_{t+2:T}}.$$

Exercise: derive this.

The fact that these can be expressed simply using only multiplication of vectors and matrices allows us to make an improvement to the forward-backward algorithm.

Hidden Markov models

The *forward-backward* algorithm works by:

- Moving up the sequence from 1 to T, computing and storing values for f.
- Moving down the sequence from T to 1 computing values for b and *combining* them with the stored values for f using the equation

$$\Pr(S_t|e_{1:T}) = cf_{1:t}b_{t+1:T}$$

Now in our simplified HMM case we have

$$f_{1:t+1} = c\mathbf{E}_{t+1}\mathbf{S}^T f_{1:t}$$

or multiplying through by $(\mathbf{E}_{t+1}\mathbf{S}^T)^{-1}$ and re-arranging

$$\boxed{f_{1:t} = \frac{1}{c}(\mathbf{S}^T)^{-1}(\mathbf{E}_{t+1})^{-1} f_{1:t+1}}$$

Hidden Markov models

So as long as:

- We know the *final* value for f.
- $\mathbf{S}^T$ has an inverse.
- Every observation has non-zero probability in every state.

We *don't* have to store T different values for f—we just work through, discarding intermediate values, to obtain the last value and then work backward.

Example: 2008, paper 9, question 5

A friend of mine likes to climb on the roofs of Cambridge. To make a good start to the coming week, he climbs on a Sunday with probability 0.98. Being concerned for his own safety, he is less likely to climb today if he climbed yesterday, so

$$\Pr(\texttt{climb today}|\texttt{climb yesterday}) = 0.4$$

If he did not climb yesterday then he is very unlikely to climb today, so

$$\Pr(\texttt{climb today}|\neg\texttt{climb yesterday}) = 0.1$$

Unfortunately, he is not a very good climber, and is quite likely to injure himself if he goes climbing, so

$$\Pr(\texttt{injury}|\texttt{climb today}) = 0.8$$

whereas

$$\Pr(\texttt{injury}|\neg\texttt{climb today}) = 0.1$$

Example: 2008, paper 9, question 5

You learn that on Monday and Tuesday evening he obtains an injury, but on Wednesday evening he does not. Use the filtering algorithm to compute the probability that he climbed on Wednesday.

Initially

$$f_{1:0} = \begin{pmatrix} 0.98 \\ 0.02 \end{pmatrix}$$

$$S = \begin{pmatrix} 0.4 & 0.6 \\ 0.1 & 0.9 \end{pmatrix}$$

$$E = \begin{pmatrix} 0.8 & 0 \\ 0 & 0.1 \end{pmatrix}$$

$$E' = \begin{pmatrix} 0.2 & 0 \\ 0 & 0.9 \end{pmatrix}$$

Example: 2008, paper 9, question 5

The update equation is

$$f_{1:t+1} = cE_{t+1}S^T f_{1:t}$$

so

$$f_{1:1} = \frac{c}{10,000}\begin{pmatrix} 8 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 4 & 1 \\ 6 & 9 \end{pmatrix}\begin{pmatrix} 98 \\ 2 \end{pmatrix} = \begin{pmatrix} 0.83874 \\ 0.16126 \end{pmatrix}$$

Repeating this twice more using E' rather than E the final time gives

$$f_{1:2} = \begin{pmatrix} 0.81268 \\ 0.18732 \end{pmatrix}$$

$$f_{1:3} = \begin{pmatrix} 0.10429 \\ 0.89571 \end{pmatrix}$$

so the answer is 0.1.

Example: 2008, paper 9, question 5

Over the course of the week, you also learn that he does not obtain an injury on Thursday or Friday. Use the smoothing algorithm to compute the probability that he climbed on Thursday.

The S, E and E' matrices are the same. The backward message starts as

$$b_{6:5} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

and the update equation is

$$b_{t:T} = SE_t b_{t+1:T}$$

Then working backwards

$$b_{5:5} = \frac{1}{100}\begin{pmatrix} 4 & 6 \\ 1 & 9 \end{pmatrix}\begin{pmatrix} 2 & 0 \\ 0 & 9 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.62 \\ 0.83 \end{pmatrix}$$

Example: 2008, paper 9, question 5

We also need one more forward step, which gives

$$f_{1:4} = \begin{pmatrix} 0.03249 \\ 0.96751 \end{pmatrix}$$

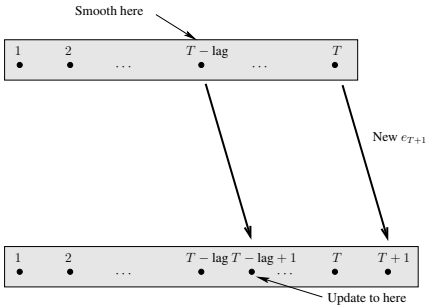
Finally

$$cf_{1:4}b_{5:5} = c\begin{pmatrix} 0.03249 \times 0.62 \\ 0.96751 \times 0.83 \end{pmatrix} = \begin{pmatrix} 0.02447 \\ 0.97553 \end{pmatrix}$$

giving the answer 0.02447.

Online smoothing

Say we want to smooth at a *fixed number of time steps*. We can also obtain a simple algorithm for updating the result each time a new e_{t+1} appears.



Online smoothing

As usual we need to calculate

$$cf_{1:T-\text{lag}}b_{T-\text{lag}+1:T}$$

to smooth at time $(T - \text{lag})$ if we've progressed to time T. So: assume $f_{1:T-\text{lag}}$ and $b_{T-\text{lag}+1:T}$ are known.

What can we now do when e_{T+1} arrives to obtain $f_{1:T-\text{lag}+1}$ and $b_{T-\text{lag}+2:T+1}$?

f is easy to update because as usual

$$f_{1:T-\text{lag}+1} = c\mathbf{E}_{T-\text{lag}+1}\mathbf{S}^T \underbrace{\boxed{f_{1:T-\text{lag}}}}_{\text{Known}}$$

Online smoothing

b is more tricky.

We know that

$$b_{T-\text{lag}+1:T} = \mathbf{SE}_{T-\text{lag}+1}b_{T-\text{lag}+2:T}$$

and continuing this recursion up to the end of the sequence at T gives

$$b_{T-\text{lag}+1:T} = \prod_{i=T-\text{lag}+1}^{T} \mathbf{SE}_i \times \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

Define

$$\boldsymbol{\beta}_{a:b} = \prod_{i=a}^{b} \mathbf{SE}_i$$

so

$$b_{T-\text{lag}+1:T} = \boldsymbol{\beta}_{T-\text{lag}+1:T} \times \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

Online smoothing

Now when e_{T+1} arrives we have

$$\begin{aligned} b_{T-\text{lag}+2:T+1} &= \prod_{i=T-\text{lag}+2}^{T+1} \mathbf{SE}_i \times \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \\ &= \boldsymbol{\beta}_{T-\text{lag}+2:T+1} \times \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \\ &= \mathbf{E}_{T-\text{lag}+1}^{-1}\mathbf{S}^{-1}\boldsymbol{\beta}_{T-\text{lag}+1:T}\mathbf{SE}_{T+1} \times \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \end{aligned}$$

Online smoothing

This leads to an easy way to update $\boldsymbol{\beta}$

$$\boxed{\boldsymbol{\beta}_{a+1:b+1} = \mathbf{E}_a^{-1}\mathbf{S}^{-1}\boldsymbol{\beta}_{a:b}\mathbf{SE}_{b+1}}$$

Using this gives the required update for b.

Supervised learning II: the Bayesian approach

We now place supervised learning into a probabilistic setting by examining:

- The application of Bayes' theorem to the *supervised learning problem*.
- Priors, the likelihood, and the posterior probability *of a hypothesis*.
- The *maximum likelihood* and *maximum a posteriori* hypotheses, and some examples.
- *Bayesian decision theory*: minimising the error rate.
- Application of the approach to *neural networks*, using approximation techniques.

Reading

There is some relevant material to be found in *Russell and Norvig* chapters 18 to 20 although the intersection between that material and what I will cover is small.

Almost all of what I cover can be found in:

- *Machine Learning*. Tom Mitchell, McGraw Hill 1997, chapter 6.
- *Pattern Recognition and Machine Learning*. Christopher M. Bishop, Springer, 2006.

Supervised learning: a quick reminder

We want to design a *classifier*, denoted $h(\mathbf{x})$

It should take an attribute vector

$$\mathbf{x}^T = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix}$$

and label it.

What we mean by *label* depends on whether we're doing *classification* or *regression*.

Supervised learning: a quick reminder

In *classification* we're assigning $\mathbf{x}$ to one of a set $\{\omega_1, \ldots, \omega_c\}$ of c *classes*.

For example, if $\mathbf{x}$ contains measurements taken from a patient then there might be three classes:

$\omega_1 =$ patient has disease
$\omega_2 =$ patient doesn't have disease
$\omega_3 =$ don't ask me buddy, I'm just a computer!

We'll often specialise to the case of two classes, denoted C_1 and C_2.

Supervised learning: a quick reminder

In *regression* we're assigning $\mathbf{x}$ to a *real number* $h(\mathbf{x}) \in \mathbb{R}$.

For example, if $\mathbf{x}$ contains measurements taken regarding today's weather then we might have

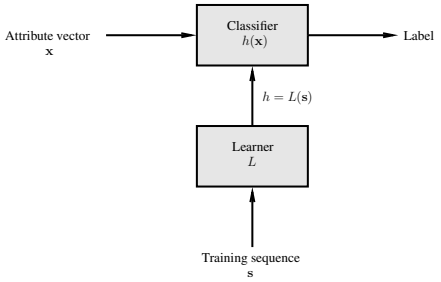
$$h(\mathbf{x}) = \text{estimate of amount of rainfall expected tomorrow}$$

For the two-class classification problem we will also refer to a situation somewhat between the two, where

$$h(\mathbf{x}) = \Pr(\mathbf{x} \text{ is in } C_1)$$

Supervised learning: a quick reminder

We don't want to design h explicitly.



So we use a *learner* L to infer it on the basis of a sequence $\mathbf{s}$ of *training examples*.

Supervised learning: a quick reminder

The *training sequence* $\mathbf{s}$ is a sequence of m *labelled examples*.

$$\mathbf{s} = \begin{pmatrix} (\mathbf{x}_1, y_1) \\ (\mathbf{x}_2, y_2) \\ \vdots \\ (\mathbf{x}_m, y_m) \end{pmatrix}$$

That is, examples of attribute vectors $\mathbf{x}$ with their correct label attached.

So a learner only gets to see the labels for a—most probably small—subset of the possible inputs $\mathbf{x}$.

Regardless, we aim that the hypothesis $h = L(\mathbf{s})$ will usually be successful at predicting the label of an input it hasn't seen before.

This ability is called *generalization*.

Supervised learning: a quick reminder

There is generally a set $\mathcal{H}$ of hypotheses from which L is allowed to select h

$$L(\mathbf{s}) = h \in \mathcal{H}$$

$\mathcal{H}$ is called the *hypothesis space*.

The learner can output a hypothesis explicitly or—as in the case of a multilayer perceptron—it can output a vector

$$\mathbf{w} = \begin{pmatrix} w_1 & w_2 & \cdots & w_W \end{pmatrix}$$

of *weights* which in turn specify h

$$h(\mathbf{x}) = f(\mathbf{w}; \mathbf{x})$$

where $\mathbf{w} = L(\mathbf{s})$.

Supervised learning: a quick reminder

In AI I you saw the *backpropagation algorithm* for training multilayer perceptrons, in the case of *regression*.

This worked by minimising a function of the weights representing the *error* currently being made:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m} \left(f(\mathbf{w}; \mathbf{x}_i) - y_i\right)^2$$

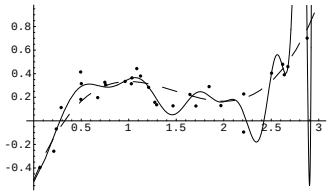
The summation here is over the training examples. The expression in the summation grows as f's prediction for $\mathbf{x}_i$ diverges from the known label y_i.

Backpropagation tries to find a $\mathbf{w}$ that minimises $E(\mathbf{w})$ by performing *gradient descent*

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \left.\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}_t}$$

Difficulties with classical neural networks

There are some well-known difficulties associated with neural network training of this kind.



BEWARE!!!

Sources of uncertainty

So we have to be careful. But let's press on with this approach for a little while longer...

The model used above suggests two sources of uncertainty that we might treat with probabilities.

- Let's *assume* we've selected an $\mathcal{H}$ to use, *and it's the same one nature is using*.
- We don't know how nature chooses h' from $\mathcal{H}$. We therefore model our uncertainty by introducing the *prior* distribution $\Pr(h)$ on $\mathcal{H}$.
- There is noise on the training examples.

It's worth emphasising at this point that in modelling noise on the training examples *we'll only consider noise on the labels*. The input vectors $\mathbf{x}$ are not modelled using a probability distribution.

The likelihood

We model our uncertainty in the training examples by specifying a *likelihood*:

$$\Pr(Y|h, \mathbf{x})$$

Translation: the probability of seeing a given label Y, when the input vector is $\mathbf{x}$ and the underlying hypothesis is h.

Example: two-class classification. A common likelihood is

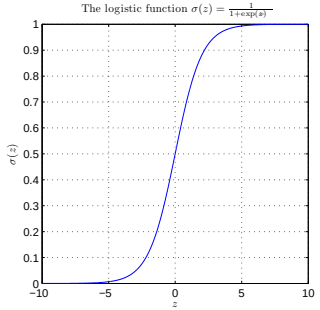
$$\Pr(Y = C_1|h, \mathbf{x}) = \sigma(h(\mathbf{x}))$$

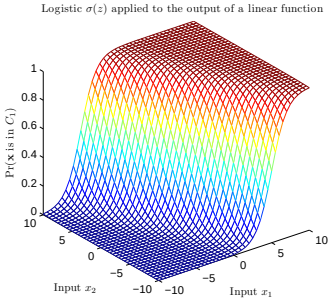
where

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

(*Note*: strictly speaking $\mathbf{x}$ should not appear in these probabilities because it's not a random variable. It is included for clarity.)

The likelihood





The likelihood

So: if we're given a training sequence, *what is the probability that it was generated using some* h?

For an example $(\mathbf{x}, y)$, y can be C_1 or C_2. It's helpful here to rename the classes as just 1 and 0 respectively because this leads to a nice simple expression. Now

$$\Pr(Y|h,\mathbf{x}) = \begin{cases} \sigma(h(\mathbf{x})) & \text{if } Y = 1 \\ 1 - \sigma(h(\mathbf{x})) & \text{if } Y = 0 \end{cases}$$

Consequently *when y has a known value* we can write

$$\Pr(y|h,\mathbf{x}) = [\sigma(h(\mathbf{x}))]^y \, [1 - \sigma(h(\mathbf{x}))]^{(1-y)}$$

If we assume that the examples are independent then the probability of seeing the labels in a training sequence $\mathbf{s}$ is straightforward.

The likelihood

Collecting the inputs and outputs in $\mathbf{s}$ together into separate matrices, so

$$\mathbf{y}^T = \begin{pmatrix} y_1 & y_2 & \cdots & y_m \end{pmatrix}$$

and

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_m \end{pmatrix}$$

we have the *likelihood of the training sequence*

$$\begin{aligned} \Pr(\mathbf{y}|h,\mathbf{X}) &= \prod_{i=1}^{m} \Pr(y_i|h,\mathbf{x}_i) \\ &= \prod_{i=1}^{m} [\sigma(h(\mathbf{x}_i))]^{y_i} \, [1 - \sigma(h(\mathbf{x}_i))]^{(1-y_i)} \end{aligned}$$

The likelihood

Another example: regression. A common likelihood in the regression case works by assuming that examples are corrupted by Gaussian noise with mean 0 and some specified variance σ^2

$$y = h(\mathbf{x}) + \epsilon, \text{ where } \epsilon \sim \mathcal{N}(0, \sigma^2)$$

As usual, the density for $\mathcal{N}(\mu, \sigma^2)$ is

$$p(Z) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right)$$

by adding $h(\mathbf{x})$ to ϵ we just shift its mean, so

$$p(y|h,\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-h(\mathbf{x}))^2}{2\sigma^2}\right)$$

The likelihood

Consequently if the examples are independent then the likelihood of a training sequence $\mathbf{s}$ is

$$\begin{aligned} p(\mathbf{y}|h,\mathbf{X}) &= \prod_{i=1}^{m} p(y_i|h,\mathbf{x}_i) \\ &= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - h(\mathbf{x}_i))^2}{2\sigma^2}\right) \\ &= \frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - h(\mathbf{x}_i))^2\right) \end{aligned}$$

where we've used the fact that

$$\exp(a)\exp(b) = \exp(a+b)$$

Bayes' theorem appears once more...

Right: we've take care of the uncertainty by introducing the *prior* $p(h)$ and the *likelihood of the training sequence* $p(\mathbf{y}|h,\mathbf{X})$.

By this point you hopefully want to apply Bayes' theorem and write

$$p(h|\mathbf{y}) = \frac{p(\mathbf{y}|h)p(h)}{p(\mathbf{y})}$$

where

$$p(\mathbf{y}) = \sum_{h\in\mathcal{H}} p(h,\mathbf{y}) = \sum_{h\in\mathcal{H}} p(\mathbf{y}|h)p(h)$$

and to simplify the expression we have now dropped the mention of $\mathbf{X}$ as the inputs are fixed. $p(h|\mathbf{y})$ is called the *posterior distribution*.

The denominator $Z = p(\mathbf{y})$ is called the *evidence* and leads on to fascinating issues of its own. Unfortunately we won't have time to explore them.

Bayes' theorem appears once more...

The boxed equation on the last slide has a very simple interpretation: *what's the probability that this specific h was used to generate the training sequence I've been given?*

Two natural learning algorithms now present themselves:

1. The *maximum likelihood hypothesis*

$$h_{\text{ML}} = \operatorname*{argmax}_{h\in\mathcal{H}} p(\mathbf{y}|h)$$

2. The *maximum a posteriori hypothesis*

$$\begin{aligned} h_{\text{MAP}} &= \operatorname*{argmax}_{h\in\mathcal{H}} p(h|\mathbf{y}) \\ &= \operatorname*{argmax}_{h\in\mathcal{H}} p(\mathbf{y}|h)p(h) \end{aligned}$$

Obviously h_{ML} corresponds to the case where the prior $p(h)$ is uniform.

Example: maximum likelihood learning

We derived an exact expression for the likelihood in the regression case above:

$$p(\mathbf{y}|h) = \frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - h(\mathbf{x}_i))^2\right)$$

Proposition: under the assumptions used, *any* learning algorithm that works by minimising the sum of squared errors on $\mathbf{s}$ finds h_{ML}.

This is clearly of interest: the notable example is the *backpropagation algorithm*.

We now prove the proposition...

Example: maximum likelihood learning

The proposition holds because:

$$
\begin{aligned}
h_{\text{ML}} &= \underset{h\in\mathcal{H}}{\operatorname{argmax}}\, p(\mathbf{y}|h) \\
&= \underset{h\in\mathcal{H}}{\operatorname{argmax}}\, \log p(\mathbf{y}|h) \\
&= \underset{h\in\mathcal{H}}{\operatorname{argmax}}\, \log \left[\frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - h(\mathbf{x}_i))^2\right)\right] \\
&= \underset{h\in\mathcal{H}}{\operatorname{argmax}}\, \log \left[\frac{1}{(2\pi\sigma^2)^{m/2}}\right] - \frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - h(\mathbf{x}_i))^2 \\
&= \underset{h\in\mathcal{H}}{\operatorname{argmax}}\, -\frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - h(\mathbf{x}_i))^2 \\
&= \underset{h\in\mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{m}(y_i - h(\mathbf{x}_i))^2
\end{aligned}
$$

Example: maximum likelihood learning

Note:

- If the distribution of the noise is *not Gaussian* a different result is obtained.
- The use of $\log$ above to simplify a maximisation problem is a standard trick.
- The Gaussian assumption is sometimes, but not always a good choice. *(Beware the Central Limit Theorem!)*.

The next step...

We have so far concentrated throughout our coverage of machine learning on choosing a *single hypothesis*.

Are we asking the right question though?

Ultimately, we want to generalise.

That means being presented with a new $\mathbf{x}$ and asking the question: *what is the most probable classification of* $\mathbf{x}$*?*

Is it reasonable to expect a single hypothesis to provide the optimal answer?

We need to look at what the optimal solution to this kind of problem might be...

Bayesian decision theory

What is the *optimal* approach to this problem?

Put another way: how should we make decisions in such a way that the outcome obtained is, on average, the best possible? Say we have:

- Attribute vectors $\mathbf{x} \in \mathbb{R}^d$.
- A set of *classes* $\{\omega_1, \ldots, \omega_c\}$.
- Several possible *actions* $\{\alpha_1, \ldots, \alpha_a\}$.

The actions can be thought of as saying *"assign the vector to class 1"* and so on.

There is also a *loss* $\lambda(\alpha_i, \omega_j)$ associated with taking action α_i when the class is ω_j.

The loss will sometimes be abbreviated to $\lambda(\alpha_i, \omega_j) = \lambda_{ij}$.

Bayesian decision theory

Say we can also *model* the world as follows:

- Classes have probabilities $\Pr(\omega)$ of occurring.
- The probability of seeing $\mathbf{x}$ when the class is ω has density $p(\mathbf{x}|\omega)$.

Think of nature choosing classes at random (although not revealing them) and showing us a vector selected at random using $p(\mathbf{x}|\omega)$.

As usual Bayes rule tells us that

$$\Pr(\omega|\mathbf{x}) = \frac{p(\mathbf{x}|\omega)\Pr(\omega)}{p(\mathbf{x})}$$

and now the denominator is

$$p(\mathbf{x}) = \sum_{i=1}^{c} p(\mathbf{x}|\omega_i)\Pr(\omega_i).$$

Bayesian decision theory

Say nature shows us $\mathbf{x}$ and we take action α_i.

If we *always* take action α_i when we see $\mathbf{x}$ then the *average* loss on seeing $\mathbf{x}$ is

$$R(\alpha_i|\mathbf{x}) = \mathbb{E}_{\omega\sim p(\omega|\mathbf{x})}\left[\lambda_{ij}|\mathbf{x}\right] = \sum_{j=1}^{c} \lambda(\alpha_i, \omega_j)\Pr(\omega_j|\mathbf{x}).$$

The quantity $R(\alpha_i|\mathbf{x})$ is called the *conditional risk*.

Note that this particular $\mathbf{x}$ is *fixed*.

Bayesian decision theory

Now say we have a *decision rule* $\alpha : \mathbb{R}^d \to \{\alpha_1, \ldots, \alpha_a\}$ telling us what action to take on seeing *any* $\mathbf{x} \in \mathbb{R}^d$.

The average loss, or *risk*, is

$$\begin{aligned} R &= \mathbb{E}_{(\mathbf{x},\omega)\sim p(\mathbf{x},\omega)}\left[\lambda(\alpha(\mathbf{x}), \omega)\right] \\ &= \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\mathbb{E}_{\omega\sim\Pr(\omega|\mathbf{x})}\left[\lambda(\alpha(\mathbf{x}), \omega)|\mathbf{x}\right]\right] \\ &= \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[R(\alpha(\mathbf{x})|\mathbf{x})\right] \qquad (11) \\ &= \int R(\alpha(\mathbf{x})|\mathbf{x})p(\mathbf{x})d\mathbf{x} \end{aligned}$$

where we have used the standard result from probability theory that

$$\mathbb{E}\left[\mathbb{E}\left[X|Y\right]\right] = \mathbb{E}\left[X\right].$$

(See the supplementary notes for a proof.)

Bayesian decision theory

Clearly the risk is minimised for the decision rule defined as follows:

$$\alpha \textit{ outputs the action } \alpha_i \textit{ that minimises } R(\alpha_i|\mathbf{x}), \textit{ for all } \mathbf{x} \in \mathbb{R}^d.$$

The provides us with the minimum possible risk, or *Bayes risk* $R^\star$.

The rule specified is called the *Bayes decision rule*.

Example: minimum error rate classification

In supervised learning our aim is often to work in such a way that we *minimise the probability of error*.

What loss should we consider in these circumstances? From basic probability theory

$$\Pr(A) = \mathbb{E}\left[I(A)\right]$$

where

$$I(A) = \begin{cases} 1 & \text{if } A \text{ happens} \\ 0 & \text{otherwise} \end{cases}$$

(See the supplementary notes for a proof.)

Example: minimum error rate classification

So if we are addressing a supervised learning problem with c classes $\{\omega_1, \ldots, \omega_c\}$ and we interpret action α_i as meaning 'the input is in class ω_i', then a loss

$$\lambda_{ij} = \begin{cases} 1 & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases}$$

means that the risk R is

$$R = \mathbb{E}\left[\lambda\right] = \Pr(\alpha(\mathbf{x}) \text{ is in error})$$

and the Bayes decision rule minimises the probability of error.

Example: minimum error rate classification

Now, what is the Bayes decision rule?

$$\begin{aligned} R(\alpha_i|\mathbf{x}) &= \sum_{j=1}^{c} \lambda(\alpha_i, \omega_j) \Pr(\omega_j|\mathbf{x}) \\ &= \sum_{i \neq j} \Pr(\omega_j|\mathbf{x}) \\ &= 1 - \Pr(\omega_i|\mathbf{x}) \end{aligned}$$

so $\alpha(\mathbf{x})$ should be *the class that maximises* $\Pr(\omega_i|\mathbf{x})$.

THE IMPORTANT SUMMARY: Given a new $\mathbf{x}$ to classify, choosing the class that maximises $\Pr(\omega_i|\mathbf{x})$ is the best strategy if your aim is to obtain the minimum error rate!

Bayesian learning II

Bayes decision theory tells us that in this context we should consider the quantity $\Pr(\omega_i|\mathbf{s}, \mathbf{x})$ where the involvement of the training sequence has been made explicit.

$$\begin{aligned} \Pr(\omega_i|\mathbf{s}, \mathbf{x}) &= \sum_{h \in \mathcal{H}} \Pr(\omega_i, h|\mathbf{s}, \mathbf{x}) \\ &= \sum_{h \in \mathcal{H}} \Pr(\omega_i|h, \mathbf{s}, \mathbf{x}) \Pr(h|\mathbf{s}, \mathbf{x}) \\ &= \sum_{h \in \mathcal{H}} \Pr(\omega_i|h, \mathbf{x}) \Pr(h|\mathbf{s}). \end{aligned}$$

Here we have re-introduced $\mathcal{H}$ using marginalisation. In moving from line 2 to line 3 we are assuming some independence properties.

Bayesian learning II

So our classification should be

$$\omega = \underset{\omega \in \{\omega_1,\ldots,\omega_c\}}{\operatorname{argmax}} \sum_{h \in \mathcal{H}} \Pr(\omega|h,\mathbf{x})\Pr(h|\mathbf{s})$$

If $\mathcal{H}$ is infinite the sum becomes an integral. So for example for a neural network

$$\omega = \underset{\omega \in \{\omega_1,\ldots,\omega_c\}}{\operatorname{argmax}} \int_{\mathbb{R}^W} \Pr(\omega|\mathbf{w},\mathbf{x})\Pr(\mathbf{w}|\mathbf{s})\, d\mathbf{w}$$

where W is the number of weights in $\mathbf{w}$.

Bayesian learning II

Why might this make any difference? (Aside from the fact that we now know it's optimal!)

Example 1: Say $|\mathcal{H}| = 3$ and $h(\mathbf{x}) = \Pr(\mathbf{x}$ is in class $C_1)$ for a 2 class problem.

$$\Pr(h_1|\mathbf{s}) = 0.4$$
$$\Pr(h_2|\mathbf{s}) = \Pr(h_3|\mathbf{s}) = 0.3$$

Now, say we have an $\mathbf{x}$ for which

$$h_1(\mathbf{x}) = 1$$
$$h_2(\mathbf{x}) = h_3(\mathbf{x}) = 0$$

so h_{MAP} says that $\mathbf{x}$ is in class C_1.

Bayesian learning II

However,

$$\begin{aligned}\Pr(\text{class } 1|\mathbf{s},\mathbf{x}) &= 1 \times 0.4 + 0 \times 0.3 + 0 \times 0.3 \\ &= 0.4 \\ \Pr(\text{class } 2|\mathbf{s},\mathbf{x}) &= 0 \times 0.4 + 1 \times 0.3 + 1 \times 0.3 \\ &= 0.6\end{aligned}$$

so class C_2 is the more probable!

In this case *the Bayes optimal approach in fact leads to a different answer*.

A more in-depth example

Let's take this a step further and work through something a little more complex in detail. For a two-class classification problem with $h(x)$ denoting $\Pr(C_1|h,x)$ and $x \in \mathbb{R}$:

Hypotheses: We have three hypotheses

$$h_1(x) = \exp(-(x-1)^2)$$
$$h_2(x) = \exp(-(2x-2)^2)$$
$$h_3(x) = \exp(-(1/10)(x-3)^2)$$

Prior: The prior is $\Pr(h_1) = 0.1$, $\Pr(h_2) = 0.05$ and $\Pr(h_3) = 0.85$.

A more in-depth example

We see the examples $(0.5, C_1)$, $(0.9, C_1)$, $(3.1, C_2)$ and $(3.4, C_1)$.

Likelihood: For the individual hypotheses the likelihoods are given by

$$\Pr(\mathbf{s}|h) = h(x_1)h(x_2)[1 - h(x_3)]h(x_4)$$

Which in this case tells us

$$\begin{aligned}\Pr(\mathbf{s}|h_1) &= 0.0024001365\\ \Pr(\mathbf{s}|h_2) &= 0.0031069836\\ \Pr(\mathbf{s}|h_3) &= 0.0003387476\end{aligned}$$

Posterior: Multiplying by the priors and normalising gives

$$\begin{aligned}\Pr(h_1|\mathbf{s}) &= 0.3512575000\\ \Pr(h_2|\mathbf{s}) &= 0.2273519164\\ \Pr(h_3|\mathbf{s}) &= 0.4213905836\end{aligned}$$

A more in-depth example

Now let's classify the point $x' = 2.5$.

We need

$$\begin{aligned}\Pr(C_1|\mathbf{s}, x') &= \Pr(C_1|h_1)\Pr(h_1|\mathbf{s}) + \Pr(C_1|h_2)\Pr(h_2|\mathbf{s}) + \Pr(C_1|h_3)\Pr(h_3|\mathbf{s})\\ &= 0.6250705317\end{aligned}$$

So: it's most likely to be in class C_1, but not with great certainty.

The Bayesian approach to neural networks

Let's now see how this can be applied to *neural networks*. We have:

- A neural network computing a function $f(\mathbf{w}; \mathbf{x})$.
- A training sequence $\mathbf{s} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$, split into

$$\mathbf{y} = (\ y_1\ \ y_2\ \ \cdots\ \ y_m\ )$$

and

$$\mathbf{X} = (\ \mathbf{x}_1\ \ \mathbf{x}_2\ \ \cdots\ \ \mathbf{x}_m\ )$$

The *prior distribution* $p(\mathbf{w})$ is now on the weight vectors and Bayes' theorem tells us that

$$p(\mathbf{w}|\mathbf{s}) = p(\mathbf{w}|\mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{w}, \mathbf{X})p(\mathbf{w}|\mathbf{X})}{p(\mathbf{y}|\mathbf{X})}$$

Nothing new so far...

The Bayesian approach to neural networks

As usual, we don't consider uncertainty in $\mathbf{x}$ and so $\mathbf{X}$ will be omitted. Consequently

$$p(\mathbf{w}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{y})}$$

where

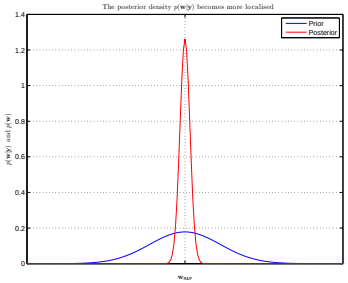
$$p(\mathbf{y}) = \int_{\mathbb{R}^W} p(\mathbf{y}|\mathbf{w})p(\mathbf{w})d\mathbf{w}$$

$p(\mathbf{y}|\mathbf{w})$ is a model of the noise corrupting the labels and as previously is the *likelihood function*.

The Bayesian approach to neural networks

$p(\mathbf{w})$ is typically a *broad distribution* to reflect the fact that in the absence of any data we have little idea of what $\mathbf{w}$ might be.

When we see some data the above equation tells us how to obtain $p(\mathbf{w}|\mathbf{y})$. This will typically be *more localised*.



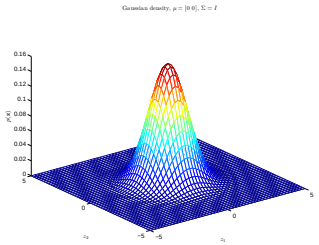
To put this into practice we need expressions for $p(\mathbf{w})$ and $p(\mathbf{y}|\mathbf{w})$.

Reminder: the general Gaussian density

Reminder: we're going to be making a lot of use of the general *Gaussian density* $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ in d dimensions

$$p(\mathbf{z}) = (2\pi)^{-d/2}|\boldsymbol{\Sigma}|^{-1/2}\exp\left[-\frac{1}{2}\left((\mathbf{z}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{z}-\boldsymbol{\mu})\right)\right]$$

where $\boldsymbol{\mu}$ is the *mean vector* and $\boldsymbol{\Sigma}$ is the *covariance matrix*.



The Gaussian prior

A common choice for $p(\mathbf{w})$ is the *Gaussian prior* with zero mean and

$$\Sigma = \sigma^2\mathbf{I}$$

so

$$p(\mathbf{w}) = (2\pi)^{-W/2}\sigma^{-W}\exp\left[-\frac{\mathbf{w}^T\mathbf{w}}{2\sigma^2}\right]$$

Note that σ controls the distribution of other parameters.

- Such parameters are called *hyperparameters*.
- Assume for now that they are both fixed and known.

Hyperparameters can be learnt using $\mathbf{s}$ through the application of more advanced techniques.

The Bayesian approach to neural networks

Physicists like to express quantities such as $p(\mathbf{w})$ in terms of a measure of *"energy"*. The expression is therefore usually re-written as

$$\boxed{p(\mathbf{w}) = \frac{1}{Z_W(\alpha)}\exp\left(-\frac{\alpha}{2}||\mathbf{w}||^2\right)}$$

where

$$E_W(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||^2$$

$$Z_W(\alpha) = \left(\frac{2\pi}{\alpha}\right)^{d/2}$$

$$\alpha = \frac{1}{\sigma^2}$$

This is simply a re-arranged version of the more usual equation.

The Gaussian noise model for regression

We've already seen that for a regression problem with zero mean Gaussian noise having variance σ_n^2

$$y_i = f(\mathbf{x}_i) + \epsilon_i$$
$$p(\epsilon_i) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\frac{\epsilon_i^2}{2\sigma_n^2}\right)$$

where f corresponds to some unknown network, the likelihood function is

$$p(\mathbf{y}|\mathbf{w}) = \frac{1}{(2\pi\sigma_n^2)^{m/2}} \exp\left(-\frac{1}{2\sigma_n^2}\sum_{i=1}^{m}(y_i - f(\mathbf{w};\mathbf{x}_i))^2\right)$$

Note that there are now two variances: σ^2 for the prior and σ_n^2 for the noise.

The Bayesian approach to neural networks

This expression can also be rewritten in physicist-friendly form

$$\boxed{p(\mathbf{y}|\mathbf{w}) = \frac{1}{Z_{\mathbf{y}}(\beta)} \exp\left(-\beta E_{\mathbf{y}}(\mathbf{w})\right)}$$

where

$$\beta = \frac{1}{\sigma_n^2}$$
$$Z_{\mathbf{y}}(\beta) = \left(\frac{2\pi}{\beta}\right)^{m/2}$$
$$E_{\mathbf{y}}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - f(\mathbf{w};\mathbf{x}_i))^2$$

Here, β is a second *hyperparameter*. Again, we assume it is fixed and known, although it can be learnt using s using more advanced techniques.

The Bayesian approach to neural networks

Combining the two boxed equations gives

$$\boxed{p(\mathbf{w}|\mathbf{y}) = \frac{1}{Z_S(\alpha,\beta)} \exp(-S(\mathbf{w}))}$$

where

$$\boxed{S(\mathbf{w}) = \alpha E_W(\mathbf{w}) + \beta E_{\mathbf{y}}(\mathbf{w})}$$

The quantity

$$Z_S(\alpha,\beta) = \int_{\mathbb{R}^W} \exp(-S(\mathbf{w}))d\mathbf{w}$$

normalises the density. Recall that this is called the *evidence*.

Example I: gradient descent revisited...

To find h_{MAP} (in this scenario by finding $\mathbf{w}_{\text{MAP}}$) we therefore maximise

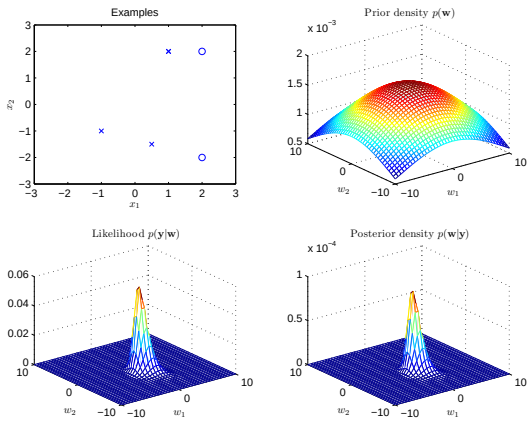
$$p(\mathbf{w}|\mathbf{y}) = \frac{1}{Z_S(\alpha,\beta)} \exp(-(\alpha E_W(\mathbf{w}) + \beta E_{\mathbf{y}}(\mathbf{w})))$$

or equivalently find

$$\mathbf{w}_{\text{MAP}} = \underset{\mathbf{w}}{\text{argmin}} \frac{\alpha}{2}||\mathbf{w}||^2 + \frac{\beta}{2}\sum_{i=1}^{m}(y_i - f(\mathbf{w};\mathbf{x}_i))^2$$

This algorithm has also been used a lot in the neural network literature and is called the *weight decay* technique.

Example II: two-class classification in two dimensions



The Bayesian approach to neural networks

What happens as the number m of examples increases?

- The first term *corresponding to the prior* remains fixed.
- The second term *corresponding to the likelihood* increases.

So for small training sequences the prior dominates, but for large ones h_{ML} is a good approximation to h_{MAP}.

The Bayesian approach to neural networks

Where have we got to...? We have obtained

$$p(\mathbf{w}|\mathbf{y}) = \frac{1}{Z_S(\alpha,\beta)}\exp(-(\alpha E_W(\mathbf{w}) + \beta E_{\mathbf{y}}(\mathbf{w})))$$

$$Z_S(\alpha,\beta) = \int_{\mathbb{R}^W} \exp(-(\alpha E_W(\mathbf{w}) + \beta E_{\mathbf{y}}(\mathbf{w})))d\mathbf{w}$$

Translating the expression for the *Bayes optimal* solution given earlier into the current scenario, we need to compute

$$p(Y|\mathbf{y},\mathbf{x}) = \int_{\mathbb{R}^W} p(y|\mathbf{w},\mathbf{x})p(\mathbf{w}|\mathbf{y})\,d\mathbf{w}$$

Easy huh? *Unfortunately not...*

The Bayesian approach to neural networks

In order to make further progress it's necessary to perform integrals of the general form

$$\int_{\mathbb{R}^W} F(\mathbf{w})p(\mathbf{w}|\mathbf{y})d\mathbf{w}$$

for various functions F and this is generally not possible.

There are two ways to get around this:

1. We can use an *approximate form* for $p(\mathbf{w}|\mathbf{y})$.
2. We can use *Monte Carlo* methods.

Method 1: approximation to $p(\mathbf{w}|\mathbf{y})$

The first approach introduces a *Gaussian approximation* to $p(\mathbf{w}|\mathbf{y})$ by using a *Taylor expansion* of

$$S(\mathbf{w}) = \alpha E_W(\mathbf{w}) + \beta E_{\mathbf{y}}(\mathbf{w})$$

at $\mathbf{w}_{\text{MAP}}$.

This allows us to use a *standard integral*.

The result will be *approximate* but we hope it's good!

Let's recall how Taylor series work...

Reminder: Taylor expansion

In one dimension the Taylor expansion about a point $x_0 \in \mathbb{R}$ for a function $f : \mathbb{R} \to \mathbb{R}$ is

$$f(x) \approx f(x_0) + \frac{1}{1!}(x - x_0)f'(x_0) + \frac{1}{2!}(x - x_0)^2 f''(x_0) + \cdots + \frac{1}{k!}(x - x_0)^k f^k(x_0)$$

What does this look like for the kinds of function we're interested in? We can try to approximate

$$\exp\left(-f(x)\right)$$

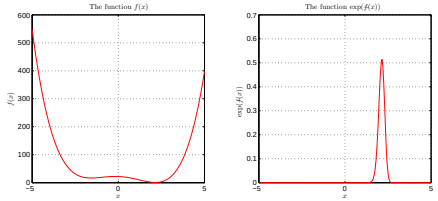
where

$$f(x) = x^4 - \frac{1}{2}x^3 - 7x^2 - \frac{5}{2}x + 22$$

This has a form similar to $S(\mathbf{w})$, but in one dimension.

Reminder: Taylor expansion

The functions of interest look like this:



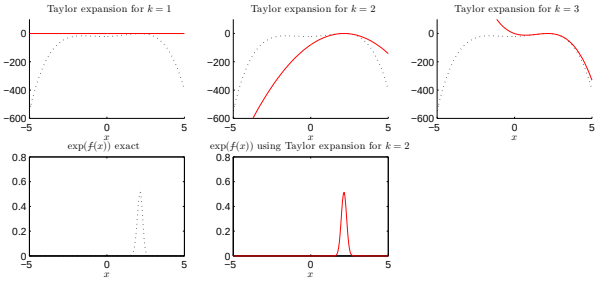
By replacing $-f(x)$ with its Taylor expansion about its maximum, which is at

$$x_{\text{max}} = 2.1437$$

we can see what the approximation to $\exp(-f(x))$ looks like. Note that the $\exp$ hugely emphasises peaks.

Reminder: Taylor expansion

Here are the approximations for $k = 1$, $k = 2$ and $k = 3$.



The use of $k = 2$ looks promising...

Reminder: Taylor expansion

In *multiple dimensions* the Taylor expansion for $k = 2$ is

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \frac{1}{1!}(\mathbf{x} - \mathbf{x}_0)^T \nabla f(\mathbf{x})\big|_{\mathbf{x}_0} + \frac{1}{2!}(\mathbf{x} - \mathbf{x}_0)^T \nabla^2 f(\mathbf{x})\big|_{\mathbf{x}_0} (\mathbf{x} - \mathbf{x}_0)$$

where ∇ denotes *gradient*

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1} \;\; \frac{\partial f(\mathbf{x})}{\partial x_2} \;\; \cdots \;\; \frac{\partial f(\mathbf{x})}{\partial x_n} \right)$$

and $\nabla^2 f(\mathbf{x})$ is the matrix with elements

$$M_{ij} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}$$

(Although this looks complicated, it's just the obvious extension of the 1-dimensional case.)

Method 1: approximation to $p(\mathbf{w}|\mathbf{y})$

Applying this to $S(\mathbf{w})$ and expanding around $\mathbf{w}_{\text{MAP}}$

$$S(\mathbf{w}) \approx S(\mathbf{w}_{\text{MAP}}) + (\mathbf{w} - \mathbf{w}_{\text{MAP}})^T \nabla S(\mathbf{w})\big|_{\mathbf{w}_{\text{MAP}}} + \frac{1}{2}(\mathbf{w} - \mathbf{w}_{\text{MAP}})^T \mathbf{A} (\mathbf{w} - \mathbf{w}_{\text{MAP}})$$

notice the following:

- As $\mathbf{w}_{\text{MAP}}$ *minimises* the function the first derivatives are zero and the corresponding term in the Taylor expansion *disappears*.
- The quantity $\mathbf{A} = \nabla\nabla S(\mathbf{w})\big|_{\mathbf{w}_{\text{MAP}}}$ can be simplified.

This is because

$$\begin{aligned} \mathbf{A} &= \nabla\nabla(\alpha E_W(\mathbf{w}) + \beta E_{\mathbf{y}}(\mathbf{w}))\big|_{\mathbf{w}_{\text{MAP}}} \\ &= \alpha \mathbf{I} + \beta \nabla\nabla E_{\mathbf{y}}(\mathbf{w}_{\text{MAP}}) \end{aligned}$$

Method 1: approximation to $p(\mathbf{w}|\mathbf{y})$

Defining

$$\Delta\mathbf{w} = \mathbf{w} - \mathbf{w}_{\text{MAP}}$$

we now have

$$S(\mathbf{w}) \approx S(\mathbf{w}_{\text{MAP}}) + \frac{1}{2}\Delta\mathbf{w}^T \mathbf{A} \Delta\mathbf{w}$$

The vector $\mathbf{w}_{\text{MAP}}$ can be obtained using any standard optimisation method (such as *backpropagation*).

The quantity $\nabla\nabla E_{\mathbf{y}}(\mathbf{w})$ can be evaluated using an *extended form of backpropagation*.

A useful integral

Dropping *for this slide only* the special meanings usually given to vectors $\mathbf{x}$ and $\mathbf{y}$, here is a useful standard integral:

If $\mathbf{A} \in \mathbb{R}^{n \times n}$ is symmetric then for $\mathbf{b} \in \mathbb{R}^n$ and $c \in \mathbb{R}$

$$\int_{\mathbb{R}^n} \exp\left(-\frac{1}{2}\left(\mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{b} + c\right)\right) d\mathbf{x} = (2\pi)^{n/2} |\mathbf{A}|^{-1/2} \exp\left(-\frac{1}{2}\left(c - \frac{\mathbf{b}^T \mathbf{A}^{-1} \mathbf{b}}{4}\right)\right)$$

At the beginning of the course, two exercises were set involving the evaluation of this integral.

To make this easy to refer to, let's call it the *BIG INTEGRAL*.

Method 1: approximation to $p(\mathbf{w}|\mathbf{y})$

We now have

$$p(\mathbf{w}|\mathbf{y}) \approx \frac{1}{Z(\alpha,\beta)} \exp\left(-S(\mathbf{w}_{\text{MAP}}) - \frac{1}{2}\Delta\mathbf{w}^T\mathbf{A}\Delta\mathbf{w}\right)$$

where $\Delta\mathbf{w} = \mathbf{w} - \mathbf{w}_{\text{MAP}}$ and using the *BIG INTEGRAL*

$$Z(\alpha,\beta) = (2\pi)^{W/2}|\mathbf{A}|^{-1/2}\exp(-S(\mathbf{w}_{\text{MAP}}))$$

Our earlier discussion tells us that given a new input $\mathbf{x}$ we should calculate

$$p(Y|\mathbf{y},\mathbf{x}) = \int_{\mathbb{R}^W} p(y|\mathbf{w},\mathbf{x})p(\mathbf{w}|\mathbf{y})d\mathbf{w}$$

$p(y|\mathbf{w},\mathbf{x})$ is just the *likelihood* so...

Method 1: approximation to $p(\mathbf{w}|\mathbf{y})$

The likelihood we're using is

$$p(y|\mathbf{w},\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - f(\mathbf{w};\mathbf{x}))^2}{2\sigma^2}\right)$$
$$\propto \exp\left(-\frac{\beta}{2}(y - f(\mathbf{w};\mathbf{x}))^2\right)$$

and plugging it into the integral gives

$$p(y|\mathbf{x},\mathbf{y}) \propto \int_{\mathbb{R}^W} \exp\left(-\frac{\beta}{2}(y - f(\mathbf{w};\mathbf{x}))^2\right) \exp\left(-\frac{1}{2}\Delta\mathbf{w}^T\mathbf{A}\Delta\mathbf{w}\right) d\mathbf{w}$$

which *has no solution!*

We need *another approximation...*

Method 1: approximation to $p(\mathbf{w}|\mathbf{y})$

If we *assume* that $p(\mathbf{w}|\mathbf{y})$ is narrow (this depends on $\mathbf{A}$) then we can introduce a *linear approximation* of $f(\mathbf{w};\mathbf{x})$ at $\mathbf{w}_{\text{MAP}}$:

$$f(\mathbf{w};\mathbf{x}) \approx f(\mathbf{w}_{\text{MAP}};\mathbf{x}) + \mathbf{g}^T\Delta\mathbf{w}$$

where $\mathbf{g} = \nabla f(\mathbf{w};\mathbf{x})|_{\mathbf{w}_{\text{MAP}}}$.

By linear approximation we just mean the Taylor expansion for $k = 1$.

This leads to

$$p(Y|\mathbf{y},\mathbf{x}) \propto \int_{\mathbb{R}^W} \exp\left(-\frac{\beta}{2}\left(y - f(\mathbf{w}_{\text{MAP}};\mathbf{x}) - \mathbf{g}^T\Delta\mathbf{w}\right)^2 - \frac{1}{2}\Delta\mathbf{w}^T\mathbf{A}\Delta\mathbf{w}\right) d\mathbf{w}$$

and this integral can be evaluated using the *BIG INTEGRAL* to give *THE ANSWER...*

Method 1: approximation to $p(\mathbf{w}|\mathbf{y})$

Finally

$$p(Y|\mathbf{y},\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(y - f(\mathbf{w}_{\text{MAP}};\mathbf{x}))^2}{2\sigma_y^2}\right)$$

where

$$\sigma_y^2 = \frac{1}{\beta} + \mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}.$$

Hooray! But what does it mean?

Method 1: approximation to $p(\mathbf{w}|\mathbf{y})$

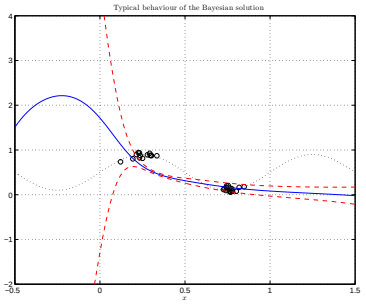
This is a *Gaussian density*, so we can now see that $p(Y|\mathbf{y}, \mathbf{x})$ *peaks* at $f(\mathbf{w}_{\text{MAP}}; \mathbf{x})$. That is, the *MAP solution*.

The *variance* σ_y^2 can be interpreted as a measure of *certainty*.

- The first term of σ_y^2 is $1/\beta$ and corresponds to the noise.
- The second term of σ_y^2 is $\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}$ and corresponds to the width of $p(\mathbf{w}|\mathbf{y})$.

Or *interpreted graphically...*

Method 1: approximation to $p(\mathbf{w}|\mathbf{y})$



Method II: Markov chain Monte Carlo (MCMC) methods

The second solution to the problem of performing integrals

$$I = \int F(\mathbf{w}) p(\mathbf{w}|\mathbf{y}) d\mathbf{w}$$

is to use *Monte Carlo* methods. The basic approach is to make the approximation

$$I \approx \frac{1}{N} \sum_{i=1}^{N} F(\mathbf{w}_i)$$

where the $\mathbf{w}_i$ have distribution $p(\mathbf{w}|\mathbf{y})$. Unfortunately, generating $\mathbf{w}_i$ with a *given distribution* can be non-trivial.

MCMC methods

A simple technique is to introduce a random walk, so

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \epsilon$$

where ϵ is zero mean spherical Gaussian and has small variance. Obviously the sequence $\mathbf{w}_i$ does not have the required distribution. However, we can use the *Metropolis algorithm*, which does *not* accept all the steps in the random walk:

1. If $p(\mathbf{w}_{i+1}|\mathbf{y}) > p(\mathbf{w}_i|\mathbf{y})$ then accept the step.
2. Else accept the step with probability $\frac{p(\mathbf{w}_{i+1}|\mathbf{y})}{p(\mathbf{w}_i|\mathbf{y})}$.

In practice, the Metropolis algorithm has several shortcomings, and a great deal of research exists on improved methods, see:

R. Neal, "Probabilistic inference using Markov chain Monte Carlo methods," University of Toronto, Department of Computer Science Technical Report CRG-TR-93-1, 1993.

Approximate inference for Bayesian networks

MCMC methods also provide a method for performing *approximate inference* in *Bayesian networks*.

Say a system can be in a state $\mathbf{s}$ and moves from state to state in discrete time steps according to a probabilistic transition

$$\Pr(\mathbf{s} \to \mathbf{s}')$$

Let $\pi_t(\mathbf{s})$ be the probability distribution for the state after t steps, so

$$\pi_{t+1}(\mathbf{s}') = \sum_{\mathbf{s}} \Pr(\mathbf{s} \to \mathbf{s}')\pi_t(\mathbf{s})$$

If at some point we obtain $\pi_{t+1}(\mathbf{s}) = \pi_t(\mathbf{s})$ for all $\mathbf{s}$ then we have reached a *stationary distribution* π. In this case

$$\forall \mathbf{s}' \pi(\mathbf{s}') = \sum_{\mathbf{s}} \Pr(\mathbf{s} \to \mathbf{s}')\pi(\mathbf{s})$$

There is exactly one stationary distribution for a given $\Pr(\mathbf{s} \to \mathbf{s}')$ provided the latter obeys some simple conditions.

Approximate inference for Bayesian networks

The condition of *detailed balance*

$$\forall \mathbf{s}, \mathbf{s}' \pi(\mathbf{s}) \Pr(\mathbf{s} \to \mathbf{s}') = \pi(\mathbf{s}') \Pr(\mathbf{s}' \to \mathbf{s})$$

is sufficient to provide a π that is a stationary distribution. To see this simply sum:

$$\begin{aligned}
\sum_{\mathbf{s}} \pi(\mathbf{s}) \Pr(\mathbf{s} \to \mathbf{s}') &= \sum_{\mathbf{s}} \pi(\mathbf{s}') \Pr(\mathbf{s}' \to \mathbf{s}) \\
&= \pi(\mathbf{s}') \underbrace{\sum_{\mathbf{s}} \Pr(\mathbf{s}' \to \mathbf{s})}_{=1} \\
&= \pi(\mathbf{s}')
\end{aligned}$$

If all this is looking a little familiar, it's because we now have an excellent application for the material in *Mathematical Methods for Computer Science*. That course used the alternative term *local balance*.

Approximate inference for Bayesian networks

Recalling once again the basic equation for performing probabilistic inference

$$\Pr(Q|e) = \frac{1}{Z} \Pr(Q \wedge e) = \frac{1}{Z} \sum_{u} \Pr(Q, u, e)$$

where

- Q is the query variable.
- e is the evidence.
- u are the unobserved variables.
- $1/Z$ normalises the distribution.

We are going to consider obtaining samples from the distribution $\Pr(Q, U|e)$.

Approximate inference for Bayesian networks

The evidence is fixed. Let the *state* of our system be a specific set of values for the *query variable and the unobserved variables*

$$\mathbf{s} = (q, u_1, u_2, \ldots, u_n) = (s_1, s_2, \ldots, s_{n+1})$$

and define $\overline{\mathbf{s}}_i$ to be the state vector *with* s_i *removed*

$$\overline{\mathbf{s}}_i = (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_{n+1})$$

To move from $\mathbf{s}$ to $\mathbf{s}'$ we replace one of its elements, say s_i, with a new value s_i' sampled according to

$$s_i' \sim \Pr(S_i | \overline{\mathbf{s}}_i, e)$$

This has detailed balance, and has $\Pr(Q, U|e)$ as its stationary distribution.

Approximate inference for Bayesian networks

To see that $\Pr(Q, U|e)$ is the stationary distribution

$$\begin{aligned}\pi(\mathbf{s})\Pr(\mathbf{s} \to \mathbf{s}') &= \Pr(\mathbf{s}|e)\Pr(s_i'|\overline{\mathbf{s}}_i, e) \\ &= \Pr(s_i, \overline{\mathbf{s}}_i|e)\Pr(s_i'|\overline{\mathbf{s}}_i, e) \\ &= \Pr(s_i|\overline{\mathbf{s}}_i, e)\Pr(\overline{\mathbf{s}}_i|e)\Pr(s_i'|\overline{\mathbf{s}}_i, e) \\ &= \Pr(s_i|\overline{\mathbf{s}}_i, e)\Pr(s_i', \overline{\mathbf{s}}_i|e) \\ &= \Pr(\mathbf{s}' \to \mathbf{s})\pi(\mathbf{s}')\end{aligned}$$

As a further simplification, sampling from $\Pr(S_i|\overline{\mathbf{s}}_i, e)$ is equivalent to sampling S_i conditional on its parents, children and children's parents.

Approximate inference for Bayesian networks

So:

- We successively sample the query variable and the unobserved variables, conditional on their parents, children and children's parents.
- This gives us a sequence $\mathbf{s}_1, \mathbf{s}_2, \ldots$ which has been sampled according to $\Pr(Q, U|e)$.

Finally, note that as

$$\Pr(Q|e) = \sum_u \Pr(Q, u|e)$$

we can just ignore the values obtained for the unobserved variables. This gives us $q_1, q_2, \ldots$ with

$$q_i \sim \Pr(Q|e)$$

Approximate inference for Bayesian networks

To see that the final step works, consider what happens when we estimate the expected value of some function of Q.

$$\begin{aligned}\mathbb{E}[f(Q)] &= \sum_q f(q)\Pr(q|e) \\ &= \sum_q f(q) \sum_u \Pr(q, u|e) \\ &= \sum_q \sum_u f(q)\Pr(q, u|e)\end{aligned}$$

so sampling using $\Pr(q, u|e)$ and ignoring the values for u obtained works exactly as required.

Artificial Intelligence II: further notes on machine learning

We now look at several issues that need to be considered when *applying machine learning algorithms in practice*:

- We often have more examples from some classes than from others.
- The *obvious* measure of performance is not always the *best*.
- Much as we'd love to have an optimal method for *finding hyperparameters*, we don't have one, and it's *unlikely that we ever will*.
- We need to exercise care if we want to claim that one approach is superior to another.

Supervised learning

As usual, we want to design a *classifier*.

It should take an attribute vector

$$\mathbf{x}^T = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix}$$

and label it.

We now denote a classifier by $h_{\boldsymbol{\theta}}(\mathbf{x})$ where

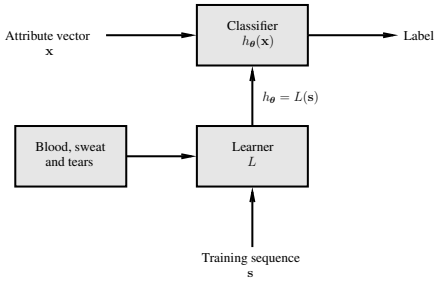
$$\boldsymbol{\theta}^T = \begin{pmatrix} \mathbf{w} & \mathbf{p} \end{pmatrix}$$

denotes any weights $\mathbf{w}$ and (hyper)parameters $\mathbf{p}$.

To keep the discussion and notation simple we assume a *classification problem* with *two classes* labelled $+1$ *(positive examples)* and -1 *(negative examples)*.

Supervised learning

Previously, the learning algorithm was a box labelled L.

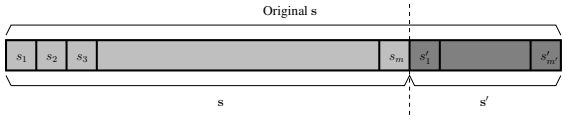


Unfortunately that turns out not to be enough, so a new box has been added.

Measuring performance

How do you assess the performance of your classifier?

1. That is, *after training*, how do you know how well you've done?
2. In general, the only way to do this is to divide your examples into a smaller *training set* $\mathbf{s}$ of m examples and a *test set* $\mathbf{s}'$ of m' examples.



The *GOLDEN RULE*: *data used to assess performance must NEVER have been seen during training*.

This might seem obvious, but it was a major flaw in a lot of early work.

Measuring performance

How do we choose m and m'? Trial and error!

Assume the training is complete, and we have a classifier $h_{\boldsymbol{\theta}}$ obtained using only $\mathbf{s}$. How do we use $\mathbf{s}'$ to assess our method's performance?

The obvious way is to see how many examples in $\mathbf{s}'$ the classifier classifies correctly:

$$\hat{\text{er}}_{\mathbf{s}'}(h_{\boldsymbol{\theta}}) = \frac{1}{m'} \sum_{i=1}^{m'} \mathbb{I}(h_{\boldsymbol{\theta}}(\mathbf{x}'_i) \neq y'_i)$$

where

$$\mathbf{s}' = \begin{pmatrix} (\mathbf{x}'_1, y'_1) & (\mathbf{x}'_2, y'_2) & \cdots & (\mathbf{x}'_{m'}, y'_{m'}) \end{pmatrix}^T$$

and

$$\mathbb{I}(z) = \begin{cases} 1 & \text{if } z = \text{true} \\ 0 & \text{if } z = \text{false} \end{cases}.$$

This is just an estimate of the *probability of error* and is often called the *accuracy*.

Unbalanced data

Unfortunately it is often the case that we have *unbalanced data* and this can make such a measure misleading. For example:

If the data is naturally such that *almost all examples are negative* (medical diagnosis for instance) then simply *classifying everything as negative* gives a high performance using this measure.

We need more subtle measures.

For a classifier h and any set $\mathbf{s}$ of size m containing m^+ positive examples and m^- negative examples...

Unbalanced data

Define

1. The *true positives*
$$P^+ = \{(\mathbf{x}, +1) \in \mathbf{s} | h(\mathbf{x}) = +1\}, \text{ and } p^+ = |P^+|$$
2. The *false positives*
$$P^- = \{(\mathbf{x}, -1) \in \mathbf{s} | h(\mathbf{x}) = +1\}, \text{ and } p^- = |P^-|$$
3. The *true negatives*
$$N^+ = \{(\mathbf{x}, -1) \in \mathbf{s} | h(\mathbf{x}) = -1\}, \text{ and } n^+ = |N^+|$$
4. The *false negatives*
$$N^- = \{(\mathbf{x}, +1) \in \mathbf{s} | h(\mathbf{x}) = -1\}, \text{ and } n^- = |N^-|$$

Thus $\hat{\text{er}}_{\mathbf{s}}(h) = (p^+ + n^+)/m$.

This allows us to define more discriminating measures of performance.

Performance measures

Some standard performance measures:

1. Precision $\frac{p^+}{p^+ + p^-}$.
2. Recall $\frac{p^+}{p^+ + n^-}$.
3. Sensitivity $\frac{p^+}{p^+ + n^-}$.
4. Specificity $\frac{n^+}{n^+ + p^-}$.
5. False positive rate $\frac{p^-}{p^- + n^+}$.
6. Positive predictive value $\frac{p^+}{p^+ + p^-}$.
7. Negative predictive value $\frac{n^+}{n^+ + n^-}$.
8. False discovery rate $\frac{p^-}{p^- + p^+}$.

In addition, plotting sensitivity (true positive rate) against the false positive rate while a parameter is varied gives the *receiver operating characteristic (ROC)* curve.

Performance measures

The following specifically take account of unbalanced data:

1. Matthews Correlation Coefficient (MCC)
$$\text{MCC} = \frac{p^+ n^+ - p^- n^-}{\sqrt{(p^+ + p^-)(n^+ + n^-)(p^+ + n^-)(n^+ + p^-)}}$$
2. F1 score
$$\text{F1} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

When data is unbalanced these are preferred over the accuracy.

Validation and crossvalidation

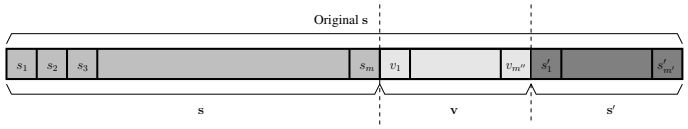
The next question: how do we choose hyperparameters?

Answer: *try different values and see which values give the best (estimated) performance*.

There is however a problem:

If I use my test set $\mathbf{s}'$ to find good hyperparameters, *then I can't use it to get a final measure of performance*. (See the Golden Rule above.)

Solution 1: make a further division of the complete set of examples to obtain a third, *validation* set:



Validation and crossvalidation

Now, to choose the value of a hyperparameter p:

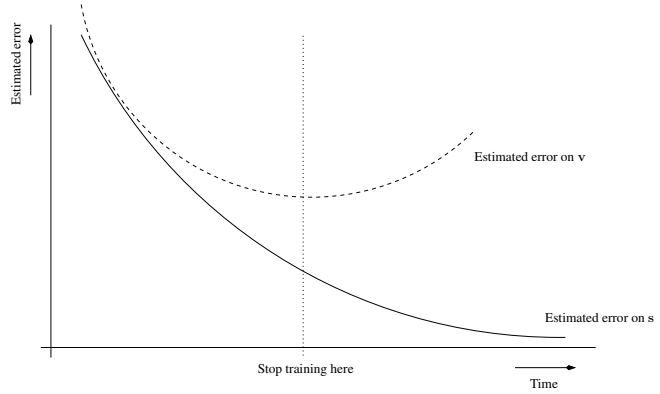
For some range of values $p_1, p_2, \ldots, p_n$

1. Run the training algorithm using training data $\mathbf{s}$ and with the hyperparameter set to p_i.
2. Assess the resulting $h_{\boldsymbol{\theta}}$ by computing a suitable measure (for example accuracy, MCC or F1) using $\mathbf{v}$.

Finally, select the $h_{\boldsymbol{\theta}}$ with maximum estimated performance and assess its *actual* performance using $\mathbf{s}'$.

Validation and crossvalidation

This was originally used in a similar way when deciding the best point at which to *stop training* a neural network.



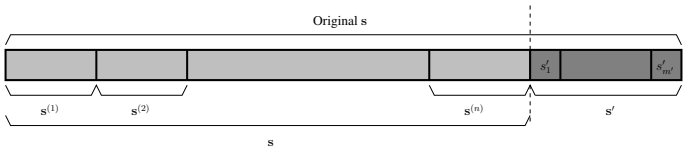
The figure shows the typical scenario.

Crossvalidation

The method of *crossvalidation* takes this a step further.

We our complete set into training set $\mathbf{s}$ and testing set $\mathbf{s}'$ as before.

But now instead of further subdividing $\mathbf{s}$ just once we divide it into n *folds* $\mathbf{s}^{(i)}$ each having m/n examples.



Typically $n = 10$ although other values are also used, for example if $n = m$ we have *leave-one-out* cross-validation.

Crossvalidation

Let $\mathbf{s}_{-i}$ denote the set obtained from $\mathbf{s}$ by *removing* $\mathbf{s}^{(i)}$.

Let $\hat{\mathrm{er}}_{\mathbf{s}^{(i)}}(h)$ denote any suitable error measure, such as accuracy, MCC or F1, computed for h using fold i.

Let $L_{\mathbf{s}_{-i},\mathbf{p}}$ be the classifier obtained by running learning algorithm L on examples $\mathbf{s}_{-i}$ using hyperparameters p.

Then,

$$\frac{1}{n}\sum_{i=1}^{n}\hat{\mathrm{er}}_{\mathbf{s}^{(i)}}(L_{\mathbf{s}_{-i},\mathbf{p}})$$

is the *n-fold crossvalidation error estimate*.

So for example, let $\mathbf{s}_j^{(i)}$ denote the jth example in the ith fold. Then using accuracy as the error estimate we have

$$\frac{1}{m}\sum_{i=1}^{n}\sum_{j=1}^{m/n}\mathbb{I}(L_{\mathbf{s}_{-i},\mathbf{p}}(\mathbf{x}_j^{(i)}) \neq y_j^{(i)})$$

Crossvalidation

Two further points:

1. What if the data are unbalanced? *Stratified crossvalidation* chooses folds such that the proportion of positive examples in each fold matches that in $\mathbf{s}$.
2. Hyperparameter choice can be done just as above, using a basic search.

What happens however if we have multiple hyperparameters?

1. We can search over all combinations of values for specified ranges of each parameter.
2. This is the *standard method in choosing parameters for support vector machines (SVMs)*.
3. With SVMs it is generally limited to the case of only two hyperparameters.
4. Larger numbers quickly become infeasible.

Comparing classifiers

Imagine I have compared the *Bloggs Classificator 2000* and the *CleverCorp Discriminotron* and found that:

1. Bloggs Classificator 2000 has estimated accuracy 0.981 on the test set.
2. CleverCorp Discriminotron has estimated accuracy 0.982 on the test set.

Can I claim that the CleverCorp Discriminotron is the better classifier?

Answer:

NO! NO! NO! NO! NO! NO! NO! NO! NO!!!!!!!!!!!!!

Comparing classifiers

NO!!!!!!!

Note for next year: include photo of grumpy-looking cat.

Assessing a single classifier

From *Mathematical Methods for Computer Science*:

The *Central Limit Theorem*: If we have independent identically distributed (iid) random variables $X_1, X_2, \ldots, X_n$ with mean

$$\mathbb{E}[X] = \mu$$

and standard deviation

$$\mathbb{E}\left[(X-\mu)^2\right] = \sigma^2$$

then as $n \to \infty$

$$\frac{\hat{X}_n - \mu}{\sigma/\sqrt{n}} \to N(0,1)$$

where

$$\hat{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i.$$

Assessing a single classifier

We have tables of values z_p such that if $x \sim N(0,1)$ then

$$\Pr(-z_p \leq x \leq z_p) > p.$$

Rearranging this using the equation from the previous slide we have that with probability p

$$\mu \in \left[\hat{X}_n \pm z_p\sqrt{\frac{\sigma^2}{n}}\right]. \tag{12}$$

We don't know σ^2 but it can be estimated using

$$\sigma^2 \simeq \frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \hat{X}_n\right)^2.$$

Alternatively, when X takes only values 0 or 1

$$\sigma^2 = \mathbb{E}\left[(X-\mu)^2\right] = \mathbb{E}\left[X^2\right] - \mu^2 = \mu(1-\mu) \simeq \hat{X}_n(1-\hat{X}_n).$$

Assessing a single classifier

The *actual probability of error* for a classifier h is

$$\text{er}(h) = \mathbb{E}\left[\mathbb{I}(h(\mathbf{x}) \neq y)\right]$$

and we are *estimating* $\text{er}(h)$ using the *accuracy*

$$\hat{\text{er}}_{\mathbf{s}}(h) = \frac{1}{m}\sum_{i=1}^{m}\mathbb{I}(h(\mathbf{x}_i) \neq y_i)$$

for a test set $\mathbf{s}$.

We can find a confidence interval for this estimate using precisely the derivation above, simply by noting that the X_i are the random variables

$$X_i = \mathbb{I}(h(\mathbf{x}_i) \neq y_i).$$

Assessing a single classifier

Typically we are interested in a 95% confidence interval, for which $z_p = 1.96$.

Thus, when $m > 30$ (so that the central limit theorem applies) we know that, with probability 0.95

$$\boxed{\text{er}(h) = \hat{\text{er}}_{\mathbf{s}}(h) \pm 1.96\sqrt{\frac{\hat{\text{er}}_{\mathbf{s}}(h)(1-\hat{\text{er}}_{\mathbf{s}}(h))}{m}}}.$$

Example: I have 100 test examples and my classifier makes 18 errors. With probability 0.95 I know that

$$\begin{aligned}\text{er}(h) &= 0.18 \pm 1.96\sqrt{\frac{0.18(1-0.18)}{100}} \\ &= 0.18 \pm 0.075.\end{aligned}$$

This should perhaps *raise an alarm* regarding our suggested comparison of classifiers above.

Assessing a single classifier

There is an important distinction to be made here:

1. The *mean of* X is μ and the *variance of* X is σ^2.
2. We can also ask about the mean and variance of $\hat{X}_n$.
3. The *mean of* $\hat{X}_n$ is
$$\begin{aligned}\mathbb{E}\left[\hat{X}_n\right] &= \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] \\ &= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[X_i\right] \\ &= \mu.\end{aligned}$$
4. It is left as an *exercise* to show that the *variance of* $\hat{X}_n$ is
$$\sigma^2_{\hat{X}_n} = \frac{\sigma^2}{n}.$$

Comparing classifiers

We are using the values z_p such that if $x \sim N(0,1)$ then
$$\Pr(-z_p \leq x \leq z_p) > p.$$
There is an *alternative* way to think about this.

1. Say we have a random variable Y with variance σ_Y^2 and mean μ_Y.
2. The random variable $Y - \mu_Y$ has variance σ_Y^2 and mean 0.
3. It is a straightforward exercise to show that dividing a random variable having variance σ^2 by σ gives us a new random variable with variance 1.
4. Thus the random variable $\frac{Y-\mu_Y}{\sigma_Y}$ has mean 0 and variance 1.

So: with probability p
$$\begin{aligned}Y &= \mu_Y \pm z_p\sigma_Y \\ \mu_Y &= Y \pm z_p\sigma_Y.\end{aligned} \tag{13}$$
Compare this with equation (12). *You need to be careful to keep track of whether you are considering the mean and variance of a single RV or a sum of RVs.*

Comparing classifiers

Now say I have classifiers h_1 *(Bloggs Classificator 2000)* and h_2 *(CleverCorp Discriminotron)* and I want to know something about the quantity
$$d = \text{er}(h_1) - \text{er}(h_2).$$
I estimate d using
$$\hat{d} = \hat{\text{er}}_{\mathbf{s}_1}(h_1) - \hat{\text{er}}_{\mathbf{s}_2}(h_2)$$
where $\mathbf{s}_1$ and $\mathbf{s}_2$ are *two* independent test sets.

Notice:

1. The estimate of d is a sum of random variables, and *we can apply the central limit theorem*.
2. The estimate is *unbiased*
$$\mathbb{E}\left[\hat{\text{er}}_{\mathbf{s}_1}(h_1) - \hat{\text{er}}_{\mathbf{s}_2}(h_2)\right] = d.$$

Comparing classifiers

Also notice:

1. The two parts of the estimate $\hat{\text{er}}_{\mathbf{s}_1}(h_1)$ and $\hat{\text{er}}_{\mathbf{s}_2}(h_2)$ are each sums of random variables and *we can apply the central limit theorem to each*.
2. The variance of the estimate is the sum of the variances of $\hat{\text{er}}_{\mathbf{s}_1}(h_1)$ and $\hat{\text{er}}_{\mathbf{s}_2}(h_2)$.
3. Adding Gaussians gives another Gaussian.
4. *We can calculate a confidence interval for our estimate*.

With probability 0.95
$$\boxed{d = \hat{d} \pm 1.96\sqrt{\frac{\hat{\text{er}}_{\mathbf{s}_1}(h_1)(1-\hat{\text{er}}_{\mathbf{s}_1}(h_1))}{m_1} + \frac{\hat{\text{er}}_{\mathbf{s}_2}(h_2)(1-\hat{\text{er}}_{\mathbf{s}_2}(h_2))}{m_2}}}. \tag{14}$$
In fact, if we are using a split into training set $\mathbf{s}$ and test set $\mathbf{s}'$ we can generally obtain h_1 and h_2 using $\mathbf{s}$ and use the estimate
$$\hat{d} = \hat{\text{er}}_{\mathbf{s}'}(h_1) - \hat{\text{er}}_{\mathbf{s}'}(h_2).$$

Comparing classifiers—hypothesis testing

This still doesn't tell us directly about *whether one classifier is better than another*—whether h_1 is better than h_2.

What we actually want to know is whether

$$d = \text{er}(h_1) - \text{er}(h_2) > 0.$$

Say we've measured $\hat{D} = \hat{d}$. Then:

- Imagine the *actual value* of d is 0.
- Recall that the *mean* of $\hat{D}$ is d.
- So *larger* measured values $\hat{d}$ are *less likely*, even though some random variation is inevitable.
- If it is highly *unlikely* that when $d = 0$ a measured value of $\hat{d}$ would be observed, then we can be confident that $d > 0$.
- Thus we are interested in
$$\Pr(\hat{D} > d + \hat{d})$$

This is known as a *one-sided bound*.

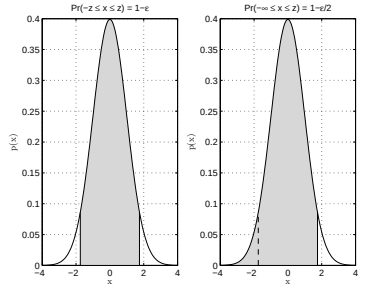
One-sided bounds

Given the *two-sided bound*

$$\Pr(-z_\epsilon \leq x \leq z_\epsilon) = 1 - \epsilon$$

we actually need to know the *one-sided bound*

$$\Pr(x \leq z_\epsilon).$$



Clearly, if our random variable is *Gaussian* then $\Pr(x \leq z_\epsilon) = 1 - \epsilon/2$.

Comparing algorithms: paired t-tests

We now know how to compare *hypotheses* h_1 and h_2.

But we still haven't properly addressed the comparison of *algorithms*.

- Remember, a learning algorithm L maps training data $\mathbf{s}$ to hypothesis h.
- So we *really* want to know about the quantity
$$d = \mathbb{E}_{\mathbf{s} \in S^m} [\text{er}(L_1(\mathbf{s})) - \text{er}(L_2(\mathbf{s}))].$$
- This is the *expected difference* between the *actual errors* of the *two different* algorithms L_1 and L_2 .

Unfortunately, we have *only one set of data* $\mathbf{s}$ available and we *can only estimate* errors $\text{er}(h)$—we don't have access to the *actual quantities*.

We can however use the idea of *crossvalidation*.

Comparing algorithms: paired t-tests

Recall, we subdivide $\mathbf{s}$ into n *folds* $\mathbf{s}^{(i)}$ each having m/n examples

and denote by $\mathbf{s}_{-i}$ the set obtained from $\mathbf{s}$ by *removing* $\mathbf{s}^{(i)}$. Then

$$\frac{1}{n} \sum_{i=1}^{n} \hat{\text{er}}_{\mathbf{s}^{(i)}}(L(\mathbf{s}_{-i}))$$

is the n*-fold crossvalidation error estimate*. Now we estimate d using

$$\hat{d} = \frac{1}{n} \sum_{i=1}^{n} \left[\hat{\text{er}}_{\mathbf{s}^{(i)}}(L_1(\mathbf{s}_{-i})) - \hat{\text{er}}_{\mathbf{s}^{(i)}}(L_2(\mathbf{s}_{-i})) \right].$$

Comparing algorithms: paired t-tests

As usual, there is a *statistical test* allowing us to assess *how likely this estimate is to mislead us*.

We will not consider the derivation in detail. With probability p

$$d \in \left[\hat{d} \pm t_{p,n-1}\sigma_{\hat{d}}\right].$$

This is analogous to the equations seen above, however:

- The parameter $t_{p,n-1}$ is analogous to z_p.
- The parameter $t_{p,n-1}$ is related to the area under the *Student's t-distribution* whereas z_p is related to the area under the normal distribution.
- The relevant estimate of *standard deviation* is

$$\sigma_{\hat{d}} = \sqrt{\frac{1}{n(n-1)}\sum_{i=1}^{n}\left(d_i - \hat{d}\right)^2}$$

where

$$d_i = \hat{\text{er}}_{\mathbf{s}^{(i)}}(L_1(\mathbf{s}_{-i})) - \hat{\text{er}}_{\mathbf{s}^{(i)}}(L_2(\mathbf{s}_{-i})).$$

Reinforcement Learning

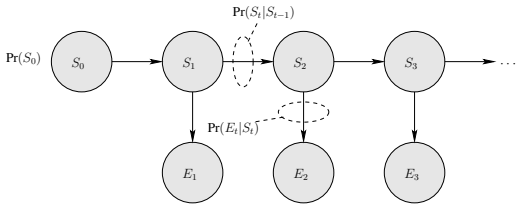
We now examine:

- Some potential shortcomings of hidden Markov models, and of supervised learning.
- An extension know as the *Markov Decision Process (MDP)*.
- The way in which we might *learn from rewards* gained as a result of *acting within an environment*.
- Specific, simple algorithms for performing such learning, and their convergence properties.

Reading: Russell and Norvig, chapter 21. Mitchell chapter 13.

Reinforcement learning and HMMs

Hidden Markov Models (HMMs) are appropriate when our agent models the world as follows

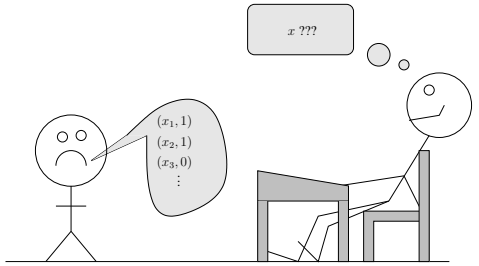


and only wants to infer information about the *state* of the world on the basis of observing the available *evidence*.

This might be criticised as un-necessarily restricted, although it is very effective for the right kind of problem.

Reinforcement learning and supervised learning

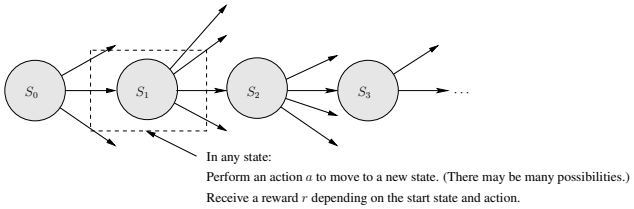
Supervised learners learn from *specifically labelled chunks of information*:



This might also be criticised as un-necessarily restricted: there are other ways to learn.

Reinforcement learning: the basic case

We now begin to model the world in a more realistic way as follows:



The agent can *perform actions* in order to *change the world's state*.

If the agent performs an action in a particular state, then it *gains a corresponding reward*.

Deterministic Markov Decision Processes

Formally, we have a set of states

$$S = \{s_1, s_2, \ldots, s_n\}$$

and in each state we can perform one of a set of actions

$$A = \{a_1, a_2, \ldots, a_m\}.$$

We also have a function

$$\mathcal{S} : S \times A \to S$$

such that $\mathcal{S}(s, a)$ is the new state resulting from performing action a in state s, and a function

$$\mathcal{R} : S \times A \to \mathbb{R}$$

such that $\mathcal{R}(s, a)$ is the *reward* obtained by executing action a in state s.

Deterministic Markov Decision Processes

From the point of view of the agent, there is a matter of considerable importance:

The agent does not have access to the functions $\mathcal{S}$ and $\mathcal{R}$.

It therefore has to *learn* a *policy*, which is a function

$$p : S \to A$$

such that $p(s)$ provides the action a that should be executed in state s.

What might the agent use as its criterion for learning a policy?

Measuring the quality of a policy

Say we start in a state at time t, denoted s_t, and we follow a policy p. At each future step in time we get a reward. Denote the rewards r_t, r_{t+1}, ... and so on.

A common measure of the quality of a policy p is the *discounted cumulative reward*

$$\begin{aligned} V^p(s_t) &= \sum_{i=0}^{\infty} \epsilon^i r_{t+i} \\ &= r_t + \epsilon r_{t+1} + \epsilon^2 r_{t+2} + \cdots \end{aligned}$$

where $0 \leq \epsilon \leq 1$ is a constant, which defines a trade-off for how much we value immediate rewards against future rewards.

The intuition for this measure is that, on the whole, we should like our agent to prefer rewards gained quickly.

Measuring the quality of a policy

Other common measures are the *average reward*

$$\lim_{T\to\infty}\frac{1}{T}\sum_{i=0}^{T} r_{t+i}$$

and the *finite horizon reward*

$$\sum_{i=0}^{T} r_{t+i}$$

In these notes we will only address the discounted cumulative reward.

Two important issues

Note that in this kind of problem we need to address two particularly relevant issues:

- The *temporal credit assignment* problem: that is, how do we decide which specific actions are important in obtaining a reward?
- The *exploration/exploitation* problem. How do we decide between *exploiting* the knowledge we already have, and *exploring* the environment in order to possibly obtain new (and more useful) knowledge?

We will see later how to deal with these.

The optimal policy

Ultimately, our learner's aim is to learn the *optimal policy*

$$p_{\text{opt}}(s) = \operatorname*{argmax}_p V^p(s)$$

for some initial state s. We will denote the optimal discounted cumulative reward as

$$V_{\text{opt}}(s) = V^{p_{\text{opt}}}(s).$$

How might we go about learning the optimal policy?

Learning the optimal policy

The only information we have during learning is the individual rewards obtained from the environment.

We could try to learn $V_{\text{opt}}(s)$ directly, so that states can be compared:

Consider s as better than s' if $V_{\text{opt}}(s) > V_{\text{opt}}(s')$.

However we actually want to compare *actions*, not *states*. Learning $V_{\text{opt}}(s)$ might help as

$$p_{\text{opt}}(s) = \operatorname*{argmax}_a \left[\mathcal{R}(s,a) + \epsilon V_{\text{opt}}(\mathcal{S}(s,a))\right]$$

but *only if we know* $\mathcal{S}$ and $\mathcal{R}$.

As we are interested in the case where these functions are *not* known, we need something slightly different.

The $\mathcal{Q}$ function

The trick is to define the following function:

$$\mathcal{Q}(s,a) = \mathcal{R}(s,a) + \epsilon V_{\text{opt}}(\mathcal{S}(s,a))$$

This function specifies the discounted cumulative reward obtained if you do action a in state s *and then follow the optimal policy*.

As

$$p_{\text{opt}}(s) = \operatorname*{argmax}_a \mathcal{Q}(s,a)$$

then provided one can learn $\mathcal{Q}$ *it is not necessary to have knowledge of* $\mathcal{S}$ *and* $\mathcal{R}$ *to obtain the optimal policy*.

The $\mathcal{Q}$ function

Note also that

$$V_{\text{opt}}(s) = \max_\alpha \mathcal{Q}(s,\alpha)$$

and so

$$\boxed{\mathcal{Q}(s,a) = \mathcal{R}(s,a) + \epsilon \max_\alpha \mathcal{Q}(\mathcal{S}(s,a),\alpha)}$$

which suggests a simple learning algorithm.

Let Q' be our learner's estimate of what the exact $\mathcal{Q}$ function is.

That is, in the current scenario Q' is a table containing the estimated values of $\mathcal{Q}(s,a)$ for all pairs (s,a).

$\mathcal{Q}$-learning

Start with all entries in Q' set to 0. (In fact we will see in a moment that random entries will do.)

Repeat the following:

1. Look at the current state s and choose an action a. (We will see how to do this in a moment.)
2. Do the action a and obtain some reward $\mathcal{R}(s,a)$.
3. Observe the new state $\mathcal{S}(s,a)$.
4. Perform the update

$$Q'(s,a) = \mathcal{R}(s,a) + \epsilon \max_\alpha Q'(\mathcal{S}(s,a),\alpha)$$

Note that this can be done in *episodes*. For example, in learning to play games, we can play multiple games, each being a single episode.

Convergence of $\mathcal{Q}$-learning

This looks as though it might converge!

Note that, if the rewards are at least 0 and we initialise Q' to 0 then,

$$\forall n, s, a \; Q'_{n+1}(s,a) \geq Q'_n(s,a)$$

and

$$\forall n, s, a \; Q(s,a) \geq Q'_n(s,a) \geq 0$$

However, we need to be a bit more rigorous than this...

Convergence of $\mathcal{Q}$-learning

If:

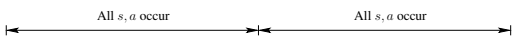
1. The agent is operating in an environment that is a deterministic MDP.
2. Rewards are bounded in the sense that there is a constant $\delta > 0$ such that
$$\forall s, a \; |\mathcal{R}(s, a)| < \delta$$
3. All possible pairs s and a are visited infinitely often.

Then the $\mathcal{Q}$-learning algorithm converges, in the sense that
$$\forall a, s \; Q'_n(s, a) \to \mathcal{Q}(s, a)$$
as $n \to \infty$.

Convergence of $\mathcal{Q}$-learning

This is straightforward to demonstrate.

Using condition 3, take two stretches of time in which all s and a pairs occur:



Define
$$\xi(n) = \max_{s,a} |Q'_n(s, a) - \mathcal{Q}(s, a)|$$
the maximum error in Q' at n.

What happens when $Q'_n(s, a)$ is updated to $Q'_{n+1}(s, a)$?

Convergence of $\mathcal{Q}$-learning

We have,
$$\begin{aligned}
&|Q'_{n+1}(s, a) - \mathcal{Q}(s, a)| \\
&\quad = |(\mathcal{R}(s, a) + \epsilon \max_\alpha Q'_n(\mathcal{S}(s, a), \alpha)) - (\mathcal{R}(s, a) + \epsilon \max_\alpha \mathcal{Q}(\mathcal{S}(s, a), \alpha))| \\
&\quad = \epsilon |\max_\alpha Q'_n(\mathcal{S}(s, a), \alpha) - \max_\alpha \mathcal{Q}(\mathcal{S}(s, a), \alpha)| \\
&\quad \leq \epsilon \max_\alpha |Q'_n(\mathcal{S}(s, a), \alpha) - \mathcal{Q}(\mathcal{S}(s, a), \alpha)| \\
&\quad \leq \epsilon \max_{s,a} |Q'_n(s, a) - \mathcal{Q}(s, a)| \\
&\quad = \epsilon \xi(n).
\end{aligned}$$
Convergence as described follows.

Choosing actions to perform

We have not yet answered the question of how to choose actions to perform during learning.

One approach is to choose actions based on our current estimate Q'. For instance
$$\text{action chosen in current state } s = \operatorname*{argmax}_a Q'(s, a).$$
However we have already noted the trade-off between exploration and exploitation. It makes more sense to:

- *Explore* during the early stages of training.
- *Exploit* during the later stages of training.

This seems particularly important in the light of condition 3 of the convergence proof.

Choosing actions to perform

One way in which to choose actions that incorporates these requirements is to introduce a constant λ and choose actions *probabilistically* according to

$$\Pr(\text{action } a|\text{state } s) = \frac{\lambda^{Q'(s,a)}}{\sum_a \lambda^{Q'(s,a)}}$$

Note that:

- If λ is *small* this promotes *exploration*.
- If λ is *large* this promotes *exploitation*.

We can vary λ as training progresses.

Improving the training process

There are two simple ways in which the process can be improved:

1. If training is episodic, we can store the rewards obtained during an episode and update *backwards* at the end.

   This allows better updating at the expense of requiring more memory.

2. We can remember information about rewards and occasionally *re-use* it by re-training.

Nondeterministic MDPs

The $\mathcal{Q}$-learning algorithm generalises easily to a more realistic situation, where the outcomes of actions are *probabilistic*.

Instead of the functions $\mathcal{S}$ and $\mathcal{R}$ we have *probability distributions*

$$\Pr(\text{new state}|\text{current state}, \text{action})$$

and

$$\Pr(\text{reward}|\text{current state}, \text{action}).$$

and we now use $\mathcal{S}(s,a)$ and $\mathcal{R}(s,a)$ to denote the corresponding random variables.

We now have

$$V^p = \mathbb{E}\left(\sum_{i=0}^{\infty} \epsilon^i r_{t+i}\right)$$

and the best policy p_{opt} maximises V^p.

$\mathcal{Q}$-learning for nondeterministic MDPs

We now have

$$\begin{aligned}\mathcal{Q}(s,a) &= \mathbb{E}(\mathcal{R}(s,a)) + \epsilon \sum_{\sigma} \Pr(\sigma|s,a) V^{\text{opt}}(\sigma) \\ &= \mathbb{E}(\mathcal{R}(s,a)) + \epsilon \sum_{\sigma} \Pr(\sigma|s,a) \max_{\alpha} \mathcal{Q}(\sigma,\alpha)\end{aligned}$$

and the rule for learning becomes

$$\boxed{Q'_{n+1} = (1-\theta_{n+1}) Q'_n(s,a) + \theta_{n+1}\left[\mathcal{R}(s,a) + \max_{\alpha} Q'_n(\mathcal{S}(s,a),\alpha)\right]}$$

with

$$\boxed{\theta_{n+1} = \frac{1}{1+v_{n+1}(s,a)}}$$

where $v_{n+1}(s,a)$ is the number of times the pair s and a has been visited so far.

Convergence of $\mathcal{Q}$-learning for nondeterministic MDPs

If:

1. The agent is operating in an environment that is a nondeterministic MDP.
2. Rewards are bounded in the sense that there is a constant $\delta > 0$ such that
$$\forall s, a \; |\mathcal{R}(s, a)| < \delta$$
3. All possible pairs s and a are visited infinitely often.
4. $n_i(s, a)$ is the ith time that we do action a in state s.

and also...

Convergence of $\mathcal{Q}$-learning for nondeterministic MDPs

...we have

$$0 \leq \theta_n < 1$$
$$\sum_{i=1}^{\infty} \theta_{n_i(s,a)} = \infty$$
$$\sum_{i=1}^{\infty} \theta^2_{n_i(s,a)} < \infty$$

then with probability 1 the $\mathcal{Q}$-learning algorithm converges, in the sense that

$$\forall a, s \; Q'_n(s, a) \to \mathcal{Q}(s, a)$$

as $n \to \infty$.

Alternative representation for the Q' table

But there's always a catch...

We have to store the table for Q':

- Even for quite straightforward problems it is HUGE!!! - certainly big enough that it can't be stored.
- A standard approach to this problem is, for example, to represent it as a *neural network*.
- One way might be to make s and a the inputs to the network and train it to produce $Q'(s, a)$ as its output.

This, of course, introduces its own problems, although it has been used very successfully in practice.

It might be covered in *Artificial Intelligence III*, which unfortunately does not yet exist.

A (very) brief introduction into how to learn hyperparameters

So far in our coverage of the Bayesian approach to neural networks, the *hyperparameters* α and β were assumed to be known and fixed.

- But this is not a good assumption because...
- ...α corresponds to the width of the prior and β to the noise variance.
- So we really want to learn these from the data as well.
- How can this be done?

We now take a look at one of several ways of addressing this problem.

The Bayesian approach to neural networks

Earlier we looked at the Bayesian approach to *neural networks* using the following notation. We have:

- A neural network computing a function $f(\mathbf{w}; \mathbf{x})$.
- A training sequence $\mathbf{s} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$, split into

$$\mathbf{y} = (\ y_1\ \ y_2\ \ \cdots\ \ y_m\ )$$

and

$$\mathbf{X} = (\ \mathbf{x}_1\ \ \mathbf{x}_2\ \ \cdots\ \ \mathbf{x}_m\ )$$

The *prior distribution* $p(\mathbf{w})$ is now on the weight vectors and Bayes' theorem tells us that

$$p(\mathbf{w}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{y})}$$

In addition we have a *Gaussian prior* and a likelihood assuming *Gaussian noise*.

The Bayesian approach to neural networks

The prior and likelihood depend on α and β respectively so we now make this clear and write

$$p(\mathbf{w}|\mathbf{y}, \alpha, \beta) = \frac{p(\mathbf{y}|\mathbf{w}, \beta)p(\mathbf{w}|\alpha)}{p(\mathbf{y}|\alpha, \beta)}$$

(Don't worry about recalling the *actual expressions* for the prior and likelihood just yet, they appear in a few slides time.)

In the earlier slides we found that the Bayes classifier should in fact compute

$$p(Y|\mathbf{y}, \mathbf{x}, \alpha, \beta) = \int_{\mathbb{R}^W} p(y|\mathbf{w}, \mathbf{x}, \beta)p(\mathbf{w}|\mathbf{y}, \alpha, \beta)\, d\mathbf{w}$$

and we found an approximation to this integral. (Again, the necessary parts of the result are repeated later.)

Hierarchical Bayes and the evidence

Let's write down directly something that might be useful to know:

$$p(\alpha, \beta|\mathbf{y}) = \frac{p(\mathbf{y}|\alpha, \beta)p(\alpha, \beta)}{p(\mathbf{y})}$$

If we know $p(\alpha, \beta|\mathbf{y})$ then a straightforward approach is to *use the values for α and β that maximise it*.

Here is a standard trick: *assume that the prior $p(\alpha, \beta)$ is flat*, so that we can just maximise

$$p(\mathbf{y}|\alpha, \beta)$$

This is called *type II maximum likelihood* and is one common way of doing the job.

As usual there are other ways of handling α and β, some of which are regarded as more "correct".

Hierarchical Bayes and the evidence

The quantity

$$p(\mathbf{y}|\alpha, \beta)$$

is called the *evidence*.

When we re-wrote our earlier equation for the posterior density of the weights, making α and β explicit, we found

$$p(\mathbf{w}|\mathbf{y}, \alpha, \beta) = \frac{p(\mathbf{y}|\mathbf{w}, \alpha, \beta)p(\mathbf{w}|\alpha, \beta)}{p(\mathbf{y}|\alpha, \beta)}$$

So *the evidence is the denominator in this equation*.

This is the *common pattern* and leads to the idea of *hierarchical Bayes*: the *evidence for the hyperparameters* at one level is the *denominator in the relevant application of Bayes theorem*.

An expression for the evidence

We have already *derived everything necessary* to write an *explicit equation for the evidence* for the case of regression that we've been following.

First, as we know about a lot of expressions involving $\mathbf{w}$ we can introduce it by the standard trick of *marginalising*:

$$\begin{aligned} p(\mathbf{y}|\alpha,\beta) &= \int p(\mathbf{y},\mathbf{w}|\alpha,\beta)d\mathbf{w} \\ &= \int p(\mathbf{y}|\mathbf{w},\alpha,\beta)p(\mathbf{w}|\alpha,\beta)d\mathbf{w} \\ &= \int p(\mathbf{y}|\mathbf{w},\beta)p(\mathbf{w}|\alpha)d\mathbf{w} \end{aligned}$$

where we've made the obvious independence simplifications.

The two densities in this integral *are just the likelihood and prior we've already studied*.

We've just conditioned on α and β, which previously were constants but are now being treated as random variables.

An expression for the evidence

Here are the actual expression for the prior and likelihood.

The prior is

$$p(\mathbf{w}|\alpha) = \frac{1}{Z_W(\alpha)}\exp\left(-\alpha E_W(\mathbf{w})\right)$$

where

$$Z_W(\alpha) = \left(\frac{2\pi}{\alpha}\right)^{W/2} \text{ and } E_W(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||^2$$

and the likelihood is

$$p(\mathbf{y}|\mathbf{w},\beta) = \frac{1}{Z_{\mathbf{y}}(\beta)}\exp\left(-\beta E_{\mathbf{y}}(\mathbf{w})\right)$$

where

$$Z_{\mathbf{y}}(\beta) = \left(\frac{2\pi}{\beta}\right)^{m/2} \text{ and } E_{\mathbf{y}}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - h(\mathbf{w};\mathbf{x}_i))^2$$

Both of these equations have been copied directly from earlier slides: *there is nothing to add*.

An expression for the evidence

That gives us

$$p(\mathbf{y}|\alpha,\beta) = \left(\frac{2\pi}{\alpha}\right)^{-W/2}\left(\frac{2\pi}{\beta}\right)^{-m/2}\int \exp\left(-S(\mathbf{w})\right)d\mathbf{w}$$

where

$$S(\mathbf{w}) = \alpha E_W(\mathbf{w}) + \beta E_{\mathbf{y}}(\mathbf{w})$$

This is *exactly the integral we first derived an approximation for*.

Specifically

$$\int \exp\left(-S(\mathbf{w})\right)d\mathbf{w} \simeq (2\pi)^{W/2}|\mathbf{A}|^{-1/2}\exp(-S(\mathbf{w}_{\text{MAP}}))$$

where

$$\mathbf{A} = \alpha\mathbf{I} + \beta\nabla\nabla E_{\mathbf{y}}(\mathbf{w}_{\text{MAP}})$$

and $\mathbf{w}_{\text{MAP}}$ is the *maximum a posteriori solution*.

An expression for the evidence

Putting all that together we get an *expression for the logarithm of the evidence*:

$$\begin{aligned} \log p(\mathbf{y}|\alpha,\beta) \simeq & \frac{W}{2}\log\alpha - \frac{m}{2}\log 2\pi + \frac{m}{2}\log\beta \\ & - \frac{1}{2}\log|\mathbf{A}| \\ & - \alpha E_W(\mathbf{w}_{\text{MAP}}) - \beta E_{\mathbf{y}}(\mathbf{w}_{\text{MAP}}) \end{aligned}$$

Again, we're using the fact that we want to *maximise the evidence* and this is equivalent to *maximising its logarithm* which turns a product into a more friendly sum.

Maximising the evidence

We want to maximise this, so let's differentiate it with respect to α and β.

For α

$$\frac{\partial \log p(\mathbf{y}|\alpha,\beta)}{\partial \alpha} = \frac{W}{2\alpha} - E_W(\mathbf{w}_{\text{MAP}}) - \frac{1}{2}\frac{\partial \log |\mathbf{A}|}{\partial \alpha}$$

How do we handle the final term? This is straightforward if we can compute the *eigenvalues* of $\mathbf{A}$.

Recall that the n eigenvalues λ_i and n eigenvectors $\mathbf{v}_i$ of an $n \times n$ matrix $\mathbf{M}$ are defined such that

$$\mathbf{M}\mathbf{v}_i = \lambda_i \mathbf{v}_i \text{ for } i = 1, \ldots, n$$

and the eigenvectors are orthonormal

$$\mathbf{v}_i^T \mathbf{v}_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

One standard result is that *the determinant of a matrix is the product of its eigenvalues*.

$$|\mathbf{M}| = \prod_{i=1}^{n} \lambda_i$$

Maximising the evidence

We have

$$\mathbf{A} = \alpha \mathbf{I} + \beta \nabla\nabla E_{\mathbf{y}}(\mathbf{w}_{\text{MAP}})$$

Say the eigenvalues of $\beta \nabla\nabla E_{\mathbf{y}}(\mathbf{w}_{\text{MAP}})$ are λ_i. *(These can be computed using standard numerical algorithms.)*

Then the eigenvalues of $\mathbf{A}$ are $\alpha + \lambda_i$ and

$$\begin{aligned} \frac{\partial \log |\mathbf{A}|}{\partial \alpha} &= \frac{\partial}{\partial \alpha}\left(\log \prod_{i=1}^{W} (\alpha + \lambda_i)\right) \\ &= \frac{\partial}{\partial \alpha}\left(\sum_{i=1}^{W} \log(\alpha + \lambda_i)\right) \\ &= \sum_{i=1}^{W} \frac{1}{\alpha + \lambda_i}\frac{\partial(\alpha + \lambda_i)}{\partial \alpha} \end{aligned}$$

This remains tricky because *the eigenvalues might be functions of* α.

Maximising the evidence

To make further progress, assume *(sometimes correct, sometimes not!)* that the λ_i *do not* depend on α.

In that case

$$\begin{aligned} \frac{\partial \log |\mathbf{A}|}{\partial \alpha} &= \sum_{i=1}^{W} \frac{1}{\alpha + \lambda_i} \\ &= \text{Trace}(\mathbf{A}^{-1}) \end{aligned}$$

because $\mathbf{M}^{-1}$ has eigenvalues $1/\lambda_i$ and the trace of a matrix is equal to the sum of its eigenvalues.

Finally, equating the derivative to zero gives:

$$\frac{W}{2\alpha} - E_W(\mathbf{w}_{\text{MAP}}) - \frac{1}{2}\text{Trace}(\mathbf{A}^{-1}) = 0$$

or

$$\alpha = \frac{1}{2E_W(\mathbf{w}_{\text{MAP}})}\left(W - \sum_{i=1}^{W} \frac{\alpha}{\alpha + \lambda_i}\right)$$

which can be used to update the value for α.

Maximising the evidence

We can now repeat the process to obtain an update for β:

$$\frac{\partial \log p(\mathbf{y}|\alpha,\beta)}{\partial \beta} = \frac{m}{2\beta} - E_{\mathbf{y}}(\mathbf{w}_{\text{MAP}}) - \frac{1}{2}\frac{\partial \log |\mathbf{A}|}{\partial \beta}$$

In this case

$$\begin{aligned} \frac{\partial \log |\mathbf{A}|}{\partial \beta} &= \frac{\partial}{\partial \beta}\left(\sum_{i=1}^{W} \log(\alpha + \lambda_i)\right) \\ &= \sum_{i=1}^{W} \frac{1}{\alpha + \lambda_i}\frac{\partial}{\partial \beta}(\alpha + \lambda_i) \\ &= \sum_{i=1}^{W} \frac{1}{\alpha + \lambda_i}\frac{\partial \lambda_i}{\partial \beta} \end{aligned}$$

and again we have a *potentially tricky derivative*.

Maximising the evidence

As the λ_i are the eigenvalues of $\beta\nabla\nabla E_{\mathbf{y}}(\mathbf{w}_{\text{MAP}})$ we have

$$\frac{\partial\lambda_i}{\partial\beta} = \frac{\lambda_i}{\beta}$$

(can you see why?) so

$$\frac{\partial \log|\mathbf{A}|}{\partial\beta} = \frac{1}{\beta}\sum_{i=1}^{W}\frac{\lambda_i}{\alpha+\lambda_i}$$

Equating the derivative to zero gives

$$\beta = \frac{1}{2E_{\mathbf{y}}(\mathbf{w}_{\text{MAP}})}\left(m - \sum_{i=1}^{W}\frac{\lambda_i}{\alpha+\lambda_i}\right)$$

which can be used to update the value for β.

Maximising the evidence

Here's why the derivative works.

Say

$$\mathbf{M} = \nabla\nabla E_{\mathbf{y}}(\mathbf{w}_{\text{MAP}})$$

so we're interested in $\partial\lambda_i/\partial\beta$ when the λ_i are the eigenvalues of $\beta\mathbf{M}$. Thus

$$(\beta\mathbf{M})\mathbf{v}_i = \lambda_i\mathbf{v}_i$$

and using the fact that the eigenvectors are orthonormal

$$\beta\mathbf{v}_i^T\mathbf{M}\mathbf{v}_i = \lambda_i\mathbf{v}_i^T\mathbf{v}_i = \lambda_i.$$

So

$$\mathbf{v}_i^T\mathbf{M}\mathbf{v}_i = \frac{\lambda_i}{\beta}$$

and

$$\frac{\partial\lambda_i}{\partial\beta} = \mathbf{v}_i^T\mathbf{M}\mathbf{v}_i = \frac{\lambda_i}{\beta}.$$

Maximising the evidence

Summary:

Define

$$\theta_t = \sum_{i=1}^{W}\frac{\lambda_i}{\alpha_t+\lambda_i}$$

where the subscript denotes the fact that we're using the following equations to periodically update our estimates of α and β.

Collecting the two update equations together we have

$$\boxed{\alpha_{t+1} = \frac{\theta_t}{2E_W(\mathbf{w}_{\text{MAP}})}}$$

and

$$\boxed{\beta_{t+1} = \frac{m-\theta_t}{2E_{\mathbf{y}}(\mathbf{w}_{\text{MAP}})}}$$

Maximising the evidence

This suggests a *method for the overall learning process*:

1. Choose the initial values α_0 and β_0 at random.
2. Choose an initial weight vector $\mathbf{w}$ according to the prior.
3. Use a standard optimisation algorithm to iteratively estimate $\mathbf{w}_{\text{MAP}}$.
4. While the optimisation progresses, periodically use the equations above to re-estimate α and β.

Step 4 requires that we compute an eigendecomposition, which might well be time-consuming. If necessary we can make a simplification.

When $m >> W$ it is reasonable to expect that $\theta_t \simeq W$ an so we can use

$$\boxed{\alpha_{t+1} = \frac{W}{2E_W(\mathbf{w}_{\text{MAP}})}}$$

and

$$\boxed{\beta_{t+1} = \frac{m}{2E_{\mathbf{y}}(\mathbf{w}_{\text{MAP}})}}$$

An alternative: integrate the hyperparameters out

While choosing α and β by maximising the evidence leads to an effective algorithm, it might be argued that a more correct way to deal with these parameters would be to *integrate them out*.

$$p(\mathbf{w}|\mathbf{y}) = \int\int p(\mathbf{w}, \alpha, \beta|\mathbf{y}) d\alpha d\beta.$$

(Recall the *general equation for probabilistic inference* where we integrate out unobserved random variables.)

Re-arranging this we have

$$\begin{aligned}\int\int p(\mathbf{w}, \alpha, \beta|\mathbf{y}) d\alpha d\beta &= \frac{1}{p(\mathbf{y})} \int\int p(\mathbf{y}|\mathbf{w}, \alpha, \beta) p(\mathbf{w}, \alpha, \beta) d\alpha d\beta \\ &= \frac{1}{p(\mathbf{y})} \int\int p(\mathbf{y}|\mathbf{w}, \alpha, \beta) p(\mathbf{w}|\alpha, \beta) p(\alpha, \beta) d\alpha d\beta \\ &= \frac{1}{p(\mathbf{y})} \int\int p(\mathbf{y}|\mathbf{w}, \beta) p(\mathbf{w}|\alpha) p(\alpha) p(\beta) d\alpha d\beta\end{aligned}$$

where we're assuming α and β are independent.

An alternative: integrate the hyperparameters out

In order to continue we need to specify priors on α and β.

On this occasion we have a good reason to choose particular priors, as α and β are *scale parameters*.

In general, a scale parameter σ is one that appears in a density of the form

$$p(x|\sigma) = \frac{1}{\sigma} f\left(\frac{x}{\sigma}\right)$$

The standard deviation of a Gaussian density is an example.

What happens to this density if we *scale* x such that $x' = cx$?

Standard result number 1

We need to recall how to deal with *transformations of continuous random variables*.

Say we have a random variable x with *probability density* $p_x(x)$.

We then transform x to $y = f(x)$ where f is strictly increasing.

What is the probability density function of y? There is a standard method for computing this. (See NST maths, or the 1A Probability course.)

$$p_y(y) = \frac{p_x(f^{-1}(y))}{f'(f^{-1}(y))}$$

An alternative: integrate the hyperparameters out

Applying this when $x' = cx$ we have

$$f(x) = cx$$

$$f^{-1}(x') = \frac{x'}{c}$$

$$f'(x) = c$$

and so

$$p_{x'}(x') = \frac{1}{c\sigma} f\left(\frac{x'}{c\sigma}\right) = \frac{1}{\sigma'} f\left(\frac{x'}{\sigma'}\right)$$

Thus the transformation leaves the density essentially unchanged, and in particular we want the densities $p(\sigma)$ and $p(\sigma')$ to be identical.

It turns out that this forces the choice

$$p(\sigma) = \frac{c'}{\sigma}.$$

This is an *improper prior* and it is conventional to take $c' = 1$.

Standard result number 2

Returning to the integral of interest

$$\frac{1}{p(\mathbf{y})}\int\int p(\mathbf{y}|\mathbf{w},\beta)p(\mathbf{w}|\alpha)p(\alpha)p(\beta)d\alpha d\beta$$

Taking the integral for α first we have

$$\begin{aligned}\int p(\mathbf{w}|\alpha)p(\alpha)d\alpha &= \int \frac{1}{\alpha Z_W(\alpha)}\exp(-\alpha E_W(\mathbf{w}))d\alpha \\ &= \int \frac{1}{\alpha}\left(\frac{\alpha}{2\pi}\right)^{W/2}\exp\left(-\frac{\alpha}{2}||\mathbf{w}||^2\right)d\alpha\end{aligned}$$

and to evaluate this we use the following *standard result*:

$$\int_0^\infty x^n \exp(-ax)dx = \frac{\Gamma(n+1)}{a^{n+1}}$$

where $n > -1$ and $a > 0$. So the integral becomes

$$(2\pi)^{-W/2}\frac{\Gamma(W/2)}{E_W(\mathbf{w})^{W/2}}$$

An alternative: integrate the hyperparameters out

The *actual value for the evidence* is

$$\begin{aligned}-\log p(\mathbf{w}|\mathbf{y}) &= -\log\left(\frac{1}{p(\mathbf{y})}\frac{1}{Z_{\mathbf{y}}(\alpha,\beta)}\exp(-(\alpha E_W(\mathbf{w}) + \beta E_{\mathbf{y}}(\mathbf{w})))\right) \\ &= \alpha E_W(\mathbf{w}) + \beta E_{\mathbf{y}}(\mathbf{w}) + \text{constant}\end{aligned}$$

and *we want to minimise this* so we need

$$\boxed{\alpha\frac{\partial E_W(\mathbf{w})}{\partial \mathbf{w}} + \beta\frac{\partial E_{\mathbf{y}}(\mathbf{w})}{\partial \mathbf{w}} = 0}$$

This should make us *VERY VERY HAPPY* because if we equate the two boxed equations we get

$$\alpha = \frac{W}{2E_W(\mathbf{w})}$$

and

$$\beta = \frac{m}{2E_{\mathbf{y}}(\mathbf{w})}$$

and so the result for *integrating out the hyperparameters* agrees with the result for *optimising the evidence*.

An alternative: integrate the hyperparameters out

Repeating the process for β and using the same standard result we have

$$\begin{aligned}\int p(\mathbf{y}|\mathbf{w},\beta)p(\beta)d\beta &= \int \frac{1}{\beta}\left(\frac{\beta}{2\pi}\right)^{m/2}\exp(-\beta E_{\mathbf{y}}(\mathbf{w}))d\beta \\ &= (2\pi)^{-m/2}\frac{\Gamma(m/2)}{E_{\mathbf{y}}(\mathbf{w})^{m/2}}\end{aligned}$$

Combining the two expression we obtain

$$\begin{aligned}-\log p(\mathbf{w}|\mathbf{y}) &= -\log\left(\frac{1}{p(\mathbf{y})}(2\pi)^{-W/2}\frac{\Gamma(W/2)}{E_W(\mathbf{w})^{W/2}}(2\pi)^{-m/2}\frac{\Gamma(m/2)}{E_{\mathbf{y}}(\mathbf{w})^{m/2}}\right) \\ &= \frac{W}{2}\log E_W(\mathbf{w}) + \frac{m}{2}\log E_{\mathbf{y}}(\mathbf{w}) + \text{constant}\end{aligned}$$

and *we want to minimise this* so we need

$$\boxed{\frac{W}{2}\frac{1}{E_W(\mathbf{w})}\frac{\partial E_W(\mathbf{w})}{\partial \mathbf{w}} + \frac{m}{2}\frac{1}{E_{\mathbf{y}}(\mathbf{w})}\frac{\partial E_{\mathbf{y}}(\mathbf{w})}{\partial \mathbf{w}} = 0}$$