TDT4171 Notes

Jon Eirik Andresen

May 2018

13-Quantifying Uncertainty

An agent cannot know everything, and it cannot have a plan for every scenario, thus it needs to be able to handle uncertainty. When deciding what to do in an uncertain environment, an agent must act **rationally**, i.e in such a way as to be likely to achieve its goal, while spending the minimal amount of effort doing so.

Trying to create a simple logical rule for diagnostic work fails because of:

- **Laziness**: Too much work to list everything
- **Theoretical ignorance**: Don't know enough about the domain
- **Practical Ignorance**: Cannot run all the test necessary to know about this particular case.

To act under uncertain conditions, an agent uses a **utility function** where a high value indicates that the agent want it. It combines this utility function with probability theory, i.e which utility(state) $\cdot$ probability(way to get to state) is max. This is the **maximum expected utility (MEU)**.

Probability notation

Sample space Ω is the set of all possible configurations the world or system can be in. ω is a state in Ω such that

$$\sum_{\omega \in \Omega} P(\omega) = 1$$

. A collection of ω 's is called an event ϕ.

$P(x|y)$ means probability of x, given that y is true. y can be a who set of things, "anded" or "ored".

$$P(a|b) = \frac{P(a \wedge b)}{P(b)}$$

A random variable has a domain, which is all the things it can be. The variable is always upper case, while the elements of the domain is always lower case. For boolean variables $A = true$ is written a and $A = false$ is written $\neg a$.

For random variables one can have a **distribution P** which contains the domain, and its corresponding probabilities. For a continuous variable, or for any with an infinite domain, **P** is a function.

One can create **joint probability functions** $\mathbf{P}(A, B) = \mathbf{P}(\mathbf{A}|\mathbf{B})\mathbf{P}(\mathbf{B})$.

Full Joint Distributions

Is you write out a table containing all the probabilities of all the variables you want to look at. Summing up all the elements of one of them is called **marginalizing**. Using conditional probabilities we can write:

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y}|\mathbf{z})P(\mathbf{z})$$

This is called **conditioning**. When adding up terms that should be 1, there is often a common factor. This is a **normalizing** factor, and is often a probability. We don't need to know the value of this as the sum should always be 1.

A variable a is **independent** of b if $P(a|b) = P(a)$ and $P(a \wedge b) = P(a)P(b)$.

Bayes' Rule

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)}$$

If we have to variables a and b and want them to be independent when they are not, one can use a third variable c to separate them, given that a and b both depend on c, but not directly on each-other(or at least that we pretend they do). This is called **conditional independence** and gives the following distribution for X and Y:

$$\mathbf{P}(X, Y|Z) = \mathbf{P}(X|Z)\mathbf{P}(Y|Z)$$

A full distribution can be written as:

$$\mathbf{P}(Cause, Effect_1 ... Effect_n) = \mathbf{P}(Cause) \cdot \prod_i \mathbf{P}(Effect_i|Cause)$$

And is called a **(Naive) Bayesian Classifier** and is great unless you're a big maths nerd.

14-Probabilistic Reasoning

This is going to be about **Bayesian Networks**, which are directed, acyclic, graphs #AlgDat, where

- Each node is a random variable.
- Each node has a probability distribution that depends on its parent nodes. I.e a child node depends only on its parent nodes.

Each node will have with it a **conditional probability table** where the different combinations of values of the parents is written, and the corresponding value of the node. For nodes that "start" the tree, there is simply the probability of that node. Note that we often just write a and not $\neg a$ as both would be redundant.

If no system for what is dependent on what, or there is a silly amount of connections there are "efficient" ways to represent the CPT's called **canonical distributions** where the relationships is determined by functions rather than numerical values. Examples are:

- Deterministic nodes: Where the value of a node is determined without uncertainty. e.g. if the parents are Norwegian, Danish, Swedish and Finnish child Scandinavian just be a logical or of its parents.
- Noisy Or: Fucking magic

Construction of Bayesian Network

For a network, the **chain rule** for probabilities can be written as:

$$\mathbf{P}(X_i|X_{i-1},...,X_1) = \mathbf{P}(X_i|Parents(X_i))$$

. For this to hold, every node must be independent of nodes that are not its parents.

To construct, do the following:

- Create a set of nodes $\{X_1,...X_n\}$. Order them such that cause comes before effect. Any order is ok, the suggested makes a smaller net.
- For $i \in \{1...n\}$ do:
  - From $X_k, < i$ choose X_k such that they form a minimal set of parents for X_i which must satisfy the above equation.
  - Create links from parents ot X_i
  - Write down the CPT, $\mathbf{P}(X_i|Parents(X_i))$
  - Note that parent nodes are chosen from nodes already placed in the graph.

One must use intuition, or knowledge, to determine what has a direct effect on things. Building a causal net, rather than a diagnostic net, will result in a more compact net and a net that has cpt that are easier to specify.

Inference by Enumeration

One often use a net to determine the distribution of **query variables**, X given **evidence variables**, or event, e, and **hidden variables** y. $\mathbf{y}$ are the variables between e and x. I.e One will ask for $\mathbf{P}(X|\mathbf{e})$.

Doing this by enumeration is:

$$\mathbf{P}(X|\mathbf{e}) = \alpha \sum_{y} \mathbf{P}(X, \mathbf{e}, \mathbf{y})$$

15-Probabilistic Reasoning of Time

When things change over time, the belief state of an agent will change. To describe this change, a **transition model** is used. Then the belief state at each time instant will be its own variable and jumping to the next time will be described by the transition model. Some syntax: $\mathbf{X}_t$ is the state at time t $\mathbf{E}_t$ is the observable variables at time t. And by observing at time t, $\mathbf{E}_t = \mathbf{e}_t$ i.e the evidence. Also needed is the relationship between the sensor and the state of the world; the **sensor model**.

Markov

To describe something over time, a **Markov Chain** is used. Where the probability of state X_t is given by the n previous states. n is the order of the model, and that $n < \infty$ is the **Markov assumption**. We also assume that the model is **stationary** i.e that the relationship between the present state and the past does not change as time goes in. The **sensor model** is that $\mathbf{E}_t$ only depends on $\mathbf{X}_t$ and not any past state. The transition between states and other states and states and evidence is given as CPT's. An example of a order 1 model is given below.

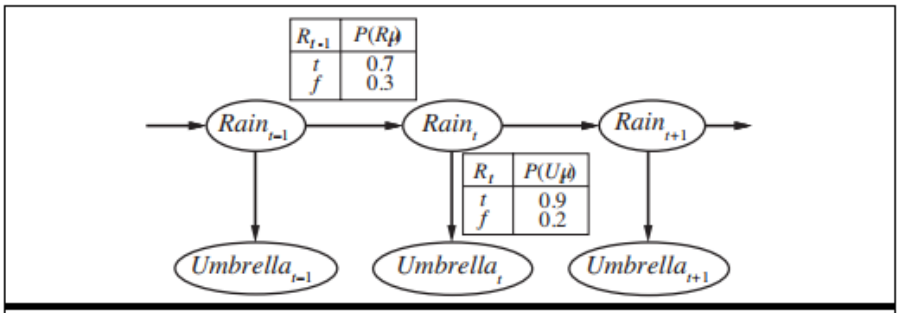

Figure 15.2 Bayesian network structure and conditional distributions describing the umbrella world. The transition model is $P(Rain_t \mid Rain_{t-1})$ and the sensor model is $P(Umbrella_t \mid Rain_t)$.

For the whole history the probability distribution, for an order 1 process, is given as:

$$\mathbf{P}(\mathbf{X}_{0:t}, \mathbf{E}_{1:t}) = \mathbf{P}(\mathbf{X}_0) \prod_{i=1}^{t} \mathbf{P}(\mathbf{X}_i|\mathbf{X}_{i-1})\mathbf{P}(\mathbf{E}_i|\mathbf{X}_i)$$

For a higher order model, add more X_k to the first term in the product. $\mathbf{P}(\mathbf{X}_0)$ is the initial distribution for X.

To increase the accuracy of a model you can do two things:

- Increase the **order** of the model.
- Increase the number of **state variables**.

Inference in Temporal Models

I.e get something useful out of this.

- **Filtering/State estimation**
  - Compute the current **belief state** given all previous evidence. $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$.
  - The algorithms work by computing $\mathbf{P}(\mathbf{X}_i|\mathbf{e}_{1:i-1}$ and then updating it to $\mathbf{P}(\mathbf{X}_i|\mathbf{e}_{1:i}$. That way, you only do two calculations at each time-step, and the computation and storage requirements remain constant.
  - The algorithm is called **FORWARD**
- **Prediction**
  - Given evidence, predict the a future state. $\mathbf{P}(\mathbf{X}_{t+k}|\mathbf{e}_{1:t})$ for some $k > 0$.
  - If you try to predict far into the future, the system will reach its **stationary distribution** and then never change. The time it takes to reach that is called **mixing time**.
- **Smoothing**
  - Given evidence, predict a past state. $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ for some $k \in (0 \rightarrow t-1)$. This is better than estimation, as there now is more data.
  - $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) = \alpha \mathbf{f}_{1:k} \times \mathbf{b}_{k+1:t}$ the $\times$ representing a element-wise multiplication.
  - This is done by running the **FORWARD** algorithm up to k and then recursively running what is called **BACKWARD** the other way.1
  - If you want to do this online, do the smoothing over a time slice that "keeps up" with the running of time.
- **Most Likely Explanation**

- Compute $\text{argmax}_{\mathbf{X}_{1:t}} P(\mathbf{X}_{1:t}|\mathbf{e}_{1:t})$
- Again there is a recursive way. Use the forward algorithm, but replace $\mathbf{f}$ with $\mathbf{m}$ with $\mathbf{m}_{1:t} = \max_{x1...x_{t-1}} \mathbf{P}(x1...x_{t-1}, X_t|e_{1:t})$
- Both time and space grows with t

$$b_{k+1:t} = \textbf{BACKWARD}(b_{k+2:t})$$
$$\textbf{BACKWARD}(\mathbf{e}_{k+1:t}|\mathbf{X}_t) = \sum_{\mathbf{x}_k} P(\mathbf{e}_{k+1}|\mathbf{x}_{k+1} P(\mathbf{e}_{k+2:t})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \quad (1)$$

Hidden Markov Models

Above, there were no assumptions on how to get from one state to another =¿ everything was cancer. Now, we assume that there is only one state variable. If there are more, just combine them into a single one, a touple if you like. Doing this will give you some dank as matrix notation.

With a single discrete variable X_t with S possible states, the **transition model** from X_t to X_{t+1} becomes and $S \times S$ matrix T which contains the probabilities for transitioning from state i to state j.

$$T_{ij} = P(X_t = j|X_{t-i} = i)$$

Since we are given the evidence e_t we only need $P(e_t|X_t) = i$, not the whole P for E. What you do is you take the vector of probabilities and put it into a $S \times S$ diagonal matrix O_t with the of-diagonal elements set to 0.

Now the **forward** becomes:

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^\intercal \mathbf{f}_{1:t}$$

And **backwards** becomes:

$$\mathbf{b}_{k+1:t} = \mathbf{T}\mathbf{O}_{k+1}\mathbf{b}_{k+2:t}$$

Neat.

0.0.1 Dynamic Bayesian Networks

Usually way more compact than a HMM.

To construct it, one needs the prior dist of the variables and the transition and sensor models. To specify the two last, one also needs the network topology.

Modes of failure IRL sensors will sometimes fail. And they fail in the following ways:

- **Transient Failure**: The sensor gives garbage data every once in a while. To fix, one adds a probability of sensor failure to the model. This has the effect of adding some "inertia" to the measurement, as to allow for some garbage data before failing completely. This is called a **transient failure model**

- **Persistent Failure**: Failing and staying failed (meIrl). There will be a small probability of of the battery having broken given that it was not broken in the previous time-step, however, if it was broken, then it will stay broken with probability 1.

Whenever there are many object we are modeleing, trying to determine which observation came from which object is called **data accociation** and is hard. When there is a lot of difference between the objects, one can use a **nearest nighbor filter** assigning the closest obervation to the closest prediction. If there is little difference, one can maximize the full join probability by using the **Hungarian algorithm**.

16-Making Simple Decisions

A little notation and intro

If an agent does action a, the probability of outcome state s' is:

$$P(RESULT(a) == s'|a, \mathbf{e})$$

To describe how much an agent wants to be in a state, we define a **utility function** $U(s)$, and the **expected utility** as

$$E\left[U(a|e)\right] = \sum_{s'} P(RESULT(a) = s'|a, \mathbf{e})U(s')$$

A **rational agent** will choose an action that will maximize its utility function i.e. $action = \text{argmax}_a E\left[U(a|\mathbf{e})\right]$

The agent prefers A over B: $A \succ B$ The agent is indifferent: $A \sim B$. Either prefers or is indifferent: $A \succeq B$.

A **lottery** is a set of outcomes $S_1, ..., S_n$ with associated probabilities $p_1, ..., p_n$.

There are some requirements of a preference relation that a utility function must obey for lotteries A, B and C:

- **Orderability** Either $A \succ B$ or $B \succ A$ or $A \sim B$ must hold.
- **Transitivity** $(A \succ B) \wedge (B \succ C) \implies (A \succ C)$.
- **Continuity** $A \succ B \succ C \implies \exists p\left[p, A; 1-p, C\right] \sim B$.
- **Substitutability** $A \sim B \implies [p, A; 1-p, C] \sim [p, B; 1-p, C]$ also holds for $\succ$.
- **Monotonixity** $A \succ B \implies (p > q \iff [p, A; p-1, B] \succ [q, A; q-1, B]$
- **Decomposability** You can always break down a complex lottery to a simpler one.

A utility function maps a set of lotteries to real numbers. They are often one the set [01], with $u_\perp 0 =$ being the worst possible outcome and $u_\top = 1$ the best. To determine the utility of every lottery, one can use a **standard lottery** and compare that to the lottery one is curious about, i.e adjust p until $U(S) = [p, u_\top; (1-p), u_\perp]$. Now the utility of S will be given by p.

Multiattribute Utility Functions

For many problems or decisions, there are multiple variables affecting the decision and create a vector $\mathbf{X} = X_1 ... X_n$.

If some multiattribute lottery is better in every way than another it has **strict dominance** over it. For a case with random variables, there can be **stochastic dominance**. THIS IS NOT ABOUT EXPECTED VALUE! But rather about **cumulative distributions**, i.e. the probability that a random variable $X \leq x$. If the cumulative distribution of a variable is $\leq$ than another for $\forall x$ then it stochasticly dominates it, given that the utility function in monotonically non decreasing.

To represent the utility of multiattribute cases we need some structure to the utility function. If attributes display **mutual preferential independence**, one attribute does not effect how other attributes interact with one another. If this is the case, then, for a deterministic case, the **value function** is simply and addition of the values of the attributes. When there is uncertainty one uses a **multiplicative utility function** which, for three attributes with **mutual utility independence**, where preferences on lotteries on $\mathbf{X}$ independent of preferences on lotteries on $\mathbf{Y} \implies$ **X MUI** of $\mathbf{Y}$, is:

$$U = k_1 U_1 + k_2 U_2 + k_3 U_3 + k_1 k_2 U_1 U_2 + k_1 k_3 U_1 U_3 + k_2 k_3 U_2 U_3 + k_1 k_2 k_3 U_1 U_2 U_3$$

Where k_i are constants and $U_i = U_i(x_i)$ can be developed independently of other Us.

Decision Networks

A network with three parts and influence arrows between them:

- Chance nodes: Oval that represent random variables.
- Decision nodes: Rectangles that represent choices made by the "user".
- Utility node: Diamonds that represent the final utility score.

The Value of Information

The value of getting to know the state of a random variable exactly is given by the **value of perfect information** (VPI). The value of getting information about the state of e_j is the average value of getting all the different possible

states k e_j can take. Then the VPI, with α being the best current action, becomes:

$$VPI_{\mathbf{e}}(E_j) = \left(\sum_k P(E_j = e_{jk}|\mathbf{e}) \mathrm{E}\left[U(\alpha_{e_{jk}}|\mathbf{e}, E_j = e_{jk}\right] \right) - \mathrm{E}\left[U(\alpha|\mathbf{e}\right]$$

An information gathering agent will gather information untill the current best action is better than the expected best from getting the best information. Such an agent is **myopic** if it only cares about finding the next piece of information and not a set of decisions or questions.

17-Making Complex Decisions

In the previous section, we only cared about doing one thing and then thinking all over again. We're going to do a similar thing here, but we are going to compute the optimal next set of actions given a current state. This will be done such that we have an optimal action for every state in the **environment**. This is called an **optimal policy** and is denoted π^*. The transitions between states is often stochastic, and modelled for example by a Bayesian network. To determine where to go we will use a **reward function** $R(s)$ to give the agent a reward, which can be both negative and positive, in each state. The whole shebang is called a **Markovian decision process (MDP)**.

The **time horizon** of the decision problem is how many step you are allowed to take. It can be finite or infinite. If it is finite, the process is **non-stationary** because the optimal policy will change over time, even if you start in the same state.

To calculate the utility if a state, we use the different states we're going to be in as attributes in a multi attribute model. We are going to assume that the agent has a **stationary** preference between state sequences, meaning that the starting time of the sequence does not matter. We use the following utilities:

- **Additive rewards** $U_h([s_0, s_1, ...]) = \sum_i R(s_i)$
- **Discounted rewards** $U_h([s_0, s_1, ...]) = \sum_i \gamma^{i-1} \cdot R(s_i), \gamma \in (0, 1)$ for $gamma = 1$ this becomes additive reward. We are going to assume additive for the rest of the chapter.

When we have an infinite horizon, the utility might also be infinite. Comparing infinite utilities is impossible, unless you use trick:

- If you use discounted rewards, the the utility is bounded by $\frac{R_{max}}{1-\gamma}$ which is fine.
- A **proper policy** is one where the agent reaches a terminal state in finite time. Then, there is no point in comparing infinite utilities. If there are no proper policies to be found, use discounted rewards.

- Use an **average reward** as a metric, which is always finite. Don't worry about this.

Using discounted rewards, the **expected utility** of a policy π is:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

S_t is the probability distribution of the state sequence, depending on the policy, starting state and the transition model. For discounted rewards with infinite horizon, the optimal policy π^* is independent of the starting state. Note that the state sequence will obviously depend on the starting state. Now, how to find these optimal policies:

Value Iteration

Calculate the utility of every state and use them to calculate the optimal action for each state.

To calculate the utility of a state s, we use the **Bellman equation**, where the utility is the reward in the current state plus the discounted utility of the best neighbour:

$$U(s) = R(s) + \gamma \cdot \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

Since this is non-linear, it is solved by iteration and starting with random utilities for each state.

Policy Iteration

Instead of updating the utility, we update the policy directly. We start with a random policy and utility function. The algorithm consists of calculation the current utility function given the current policy. Then calculate a new maximum expected utility policy based on $\pi^*(s) = \text{argmax}_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$. Evaluating the utility is now no longer non-linear as the action in each state is set by the current policy. This gives us a simpler Bellman equation:

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s'|s,\pi(s'))U_i(s')$$

Solving this is possible in $O(n^3)$ for n states.

Partially Observable MDP's

Now, let's see what happens when the agent does not know everything. The problem still contains a transistion model $P(s'|s,a)$, actions $A(s)$ and a reward function $R(s)$, but now we also have a **sensor model** $P(s|e)$. This result in the agent having a belief state as well, and the task for finding out where we're at is **filtering**. The current belief state b' is then given as $b' = \text{FORWARD}(b,a,e)$

Learning from Examples

Learning means that an agent improves its performance by making observations. All aspects of an agent can be learned or improved, even the agents utility function. These are the ways of learning we are going to talk about:

- **Unsupervised learning** No feedback during learning.
- **Reinforcement learning** Learning from rewards or punishments
- **Supervised learning** Get the raw world input and a right response, mapped by a function f. Learn to connect these through an approximate function called a **hypothesis** h

A hypothesis **generalizes** well if it can predict the output of new data well. A hypothesis is **consistent** if it agrees with all supplied data. A learning problem is **realizable** if $f \in \mathcal{H}$, where $\mathcal{H}$ is the **hypothesis space** of the problem.

Decision Trees

A function that takes a vector of attribute values and returns a single valued output. The tree is represented by boxes representing attributes, arrows with corresponding values for the attributes, either to new attributes, or to a decision. The tree i created from examples, where a greedy algorithm, choosing the most important attribute to add to the tree at each iteration. By most important, it means the attribute which most effectively separates the training examples. Then run the algorithm on the remaining examples. The algorithm is shown below:

function DECISION-TREE-LEARNING(*examples*, *attributes*, *parent_examples*) **returns** a tree

  if *examples* is empty **then return** PLURALITY-VALUE(*parent_examples*)
  else if all *examples* have the same classification **then return** the classification
  else if *attributes* is empty **then return** PLURALITY-VALUE(*examples*)
  else
    $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, *examples*)
    tree $\leftarrow$ a new decision tree with root test A
    for each value v_k of A **do**
      $exs \leftarrow \{e : e \in examples \text{ and } e.A = v_k\}$
      subtree $\leftarrow$ DECISION-TREE-LEARNING(*exs*, *attributes* $- A$, *examples*)
      add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
    return *tree*

To decide what attribute is a good one, we use **entropy** to determine the **information gain**. The entropy H of a random variable V, with values v_k and probabilities $P(v_k)$ is:

$$H(V) = -\sum_k P(v_k)\log_2(P_k)$$

For a Boolean variable, with probability q of being true, H becomes B:

$$B(q) = -(q\log_2(q) + (1-q)\log_2(1-q))$$

If a set contains p positive examples and n negative, then the entropy of the goal attribute is:

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

An attribute A with d possible attributes can divide the remaining examples E into sets $E_1, ..., E_k, ..., E_d$ each p_k and n_k negative and positive examples and will then require an additional $B\left(\frac{p_k}{p_k+n_k}\right)$ bits to classify. The remaining attribute, after testing on an attribute A is:

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

Then, the gain of testing on A is:

$$Gain(A) = B\left(\frac{p}{n+p}\right) - Remainder(A)$$

So at each step, chose the attribute that maximizes the gain.

To avoid **overfitting** when creating a tree we use **decision tree pruning** where we remove nodes that are bad. To determine bad nodes, we use a **significance test**.

Case Based Reasoning

CBR is a structured way of using previously seen examples of problems and corresponding solutions to find a solution to a new problem. A key feature is that once a new problem has been solved, the system learns from that problem.

CBR-cycle

In all CBR methods one can find this cycle:

- **Retrieve** the most similar case from memory.
- **Reuse** information form that case to propose a solution.
- **Revise** what you proposed.
- **Retain** what you think is going to be useful in the future.

26 and 27 Philosophical Foundations and Ethics

Weak AI is when a machine can *act* as if intelligent. **Strong AI** is when a machine is really intelligent and not simply simulation intelligence. Most people thing weak is possible, and don't care about strong.

Weak AI and Objections to its possibility

The Argument from disability

There are claims that a computer can never to "x". Many of those who in the past were believed to be impossible for machines to achieve, even things that require human insight, have been achieved at a similar or even higher level than what humans have been able to. However, there are still things machines cannot do.

The Mathematical Objection

Read Gödel's incompleteness theorem if you want. Some say that since AI are running on Turing machines, they must be limited by Gödel. However, they are not really Turing machines as they are finite. Also, just because an agent cannot do something another agent can, that doesn't mean its bad. Also, it is not possible to prove that humans are not also subject to what Gödel says.

The Argument from Informality

Human thought and behaviour is far to complex to model using logic and rules, the argument goes. A human learns from some experience, but when having mastered something, doesn't use an explicit set of rules for making decisions. To acheive this computers would have to achieve the following:

- Generalizations from backgrond knowlegde. Which computers now can.
- Learn autonomously. Which computers now can thants to unsupervised learning and reinforcement learning.
- Learn a lot of features. HAVE YOU HEARD OF OUR LORD AND SAVOUR CNN?
- Direct the search of information. Go to the last part of chapter 16...

Strong AI

Is a machine that acts as though it is thinking **conscious**? There are two parts to this question related to **phenomenology**, i.e does the machine feel alive? And **intentionality**, i.e does the machine have desires and beliefs regarding the real world.

Turing said that since we cannot know that other humans a conscious, it is polite to assume that AI can be as conscious as we think other humans are.

Most philosopher now think that a state of mind is directly linked to a physical state, which allows, at least in theory, for strong AI.

When trying to describe what a mental state is philosophers have one looks at either the **wide content**, the brain, the history, and everything, while **narrow content** only looks at the brain. The brain here can also be the AI.

Functionalismcan be described by a sort of ship of Theseus argument where slowly a brain is replaced by a equally functioning computer, neuron by neuron. By doing this, and if the resulting "brain" says it is conscious there are three conclusions to make:

- Whatever created consciousness before is still there $\implies$ the "new" creation is still conscious.
- or, the conscious mental events have no causal connection to behaviour, and are not in an electronic brain $\implies$ it is not!
- The experiment is impossible, so who cares...

If the second case is true, then consciousness is an **epiphenomenal** part of the body.

Ethics and Risks

Here, we look at some issues surrounding the development of AI.

Automation might take over for people

It is already happening, and society will have to change to overcome should it be more of an issue.

People might have too much or too little leisure time

More technology has always ment that we are more productive, not that we work less. However, if we get AI to work, maybe we can stop working.

People might loose their sense of being unique

It might shake up our world view as much as say Darwin or a heliocentric view of the solar system.

AI can be used for bad

More military technology can cause greater wars or remove humans from making the decisions.

Legal accountability

Who is accountable when an AI makes or recommends a decision? Up until now it is still the one who receives the recommendation.

AI might mean the end of humans

A poorly written utilifunction might cause a machine to kill us all. Also, learning systems might develop into systems we no longer understand or have influence over.