

University of Stirling
Dept. Computing Science
CSC9YE, Artificial Intelligence
Course Handbook

Welcome to the course handbook for CSC9YE, Artificial Intelligence. This handy booklet will help you through the course and, if you use it properly, ensure that you are well prepared for the exam at the end.

Lecture slides for this course are available on-line, but you really need to hear the lectures that go with them. This booklet will help you get the most out of the lectures.

How to Use this Booklet

For each topic in the course, this booklet contains the following:

- A summary of the lecture and what you should learn from it.
- A set of questions which you should be able to answer after the lecture (and certainly before the exam!)

If you can answer the questions at the end of each topic, then you will do well in the exam. If there are questions that you cannot answer, use the suggested resources to learn more and if that fails, contact the course coordinator.

The topics covered in this course are:

1. Introduction to AI
2. Sensing, characterising, and acting in the environment, goals and agent design
3. Sensor processing and computer vision
4. Modelling the environment with propositional logic
5. Modelling the environment with first order logic
6. Machine learning concepts
7. Machine learning techniques
8. Tree search techniques
9. Informed search techniques
10. Philosophical foundations

There is also an assignment which tests your knowledge of all of the topics above. Details of the assignment, tutorials, and the time table are all available on the course web site.

Lecture 1. Introduction to AI

Computers are pretty stupid. They have great memory and fast processing speed, but compared to humans they have a very limited range of abilities. Computers are not good at understanding what they see or hear, they are not very good at learning from experience and they are not very good at using judgement or dealing with uncertainty. These are 'soft skills' that humans are very good at and the study of Artificial Intelligence (AI) is about finding ways to give computers those same skills.

Throughout this course we will refer to any computer, robot or other intelligent machine as an intelligent agent, or an AI agent, or just an agent.

Here is the first big difference between traditional computing and an AI approach: A traditional computer program tells the computer **how** to do something. An AI agent is told **what** must be achieved, but not necessarily given the instructions on how.

We say that AI is **goal oriented**. The agent is given a goal and a set of tools to use to reach the goal, but it is not programmed with rules on how to actually reach the goal. It must work that out for itself.

Many abilities are needed to allow an agent to work out how to reach a goal. We will look at the main three in this course:

- **Reasoning** is the ability to use facts that you already know to prove the truthfulness of other, new, facts
- **Learning** is the ability to change behaviour based on experience
- **Planning** is useful for choosing actions before committing to behaviour

To make use of these abilities, most AI agents employ a model of some aspect of the environment in which they operate. Such models help the agent to predict how their behaviour will affect the environment, or to understand what they are sensing in that environment. Part of this course will be about methods used to model an environment.

An important concept to understand relating to the environment is that of **state**. The state an agent is in defines

everything about the part of the environment they currently occupy. If the environment changes, then the state the agent is in changes too. State is partly physical location (if you move, you change state) and partly aspects of the environment that can be sensed (if it starts to rain, the environment state has changed).

Some goals are easier to define than others. An AI agent that uses a knowledge base to diagnose a problem with your car has a simple goal – apply rules to the symptoms it is given until an answer is found. An agent that needs to beat you at chess or steer a tank across a battle field may have many smaller goals, including avoiding negative outcomes. For this reason, agents often have a **performance measure** that they are constantly trying to maximise. Think of the performance measure as being like the game ‘Warmer, Colder’ as you guide a person to find a hidden object. Performance measures allow us to introduce the concept of **utility** – that is how useful an action is to the agent. You can think of utility a little like happiness. We try to act to maximise it within given constraints, even though we might not know what the final goal is.

Lecture 2. Sensing, characterising, and acting in the environment, plus goals and agent design

Before we start on computational techniques for producing intelligent behaviour, let's take a quick look at four non-computational aspects of an AI agent. They are:

- The goal, or performance measure for the agent
- The environment in which the agent operates
- Physical actuators the agent can control
- Sensors the agent uses to receive inputs from the environment

When designing an AI agent, these four aspects need to be considered before any decisions can be made about computational solutions. Here are a few questions an AI designer would consider based on these four aspects:

1. How will the agent use its sensors to measure its own performance? How will the agent know when it has reached the goal or how close it is to that goal? Will there be a specific teaching signal to allow the agent to learn? How will that be sensed?
2. What information will the agent need, and can it get that information from the environment via its sensors?
3. Is the problem to be solved physical or logical? In other words, does the agent need motors, wheels, etc. or will it just output to a screen or file? A motor is an example of an actuator – it makes a change to the environment.
4. Is there a feedback loop from actuator to environment to sensor? Imagine the simple task of balancing a pole on its end. Its position is sensed with a camera, then actuators move the base of the pole to correct any sensed imbalance, which changes the environment (the position of the pole) and so provides a new input to the sensors (a camera).
5. What qualities does the environment have? These qualities are generally considered of interest to the designer of an AI agent:
 - a. Fully observable (accessible) vs. partially observable (inaccessible)

- b. Deterministic vs. stochastic (nondeterministic)
- c. Episodic vs. sequential
- d. Discrete vs. continuous
- e. Static vs. dynamic
- f. Single agent vs. multi agent

The lecture slides cover these qualities in detail and the first tutorial also looks at them. Make sure you know what they all mean.

Different aspects of the environment will have an impact on other design decisions. Some types of sensor will be able to observe more than others. For example, a microphone might hear something coming before a camera can see it.

Questions from Lecture 2

1. What are the four aspects of agent design to consider before thinking about computational solutions?
2. What six characteristics of an agent's environment need to be considered when designing an AI agent?
3. Consider your own life. Characterise the environment in which you need to act, the sensors your body has, and the actuators you use to change the environment. Then consider a suitable goal and a suitable performance measure and give an example of how you would use your sensors to measure your performance measure.
4. Consider an emotion as an internal performance measure. How do these measures change the behaviour of a human? How might you use an artificial emotion to alter the behaviour of an AI agent?

Lecture 3 – Sensor Processing and Computer Vision

Processing the signals that enter our senses is one of the most amazing feats that our brains perform and we do it effortlessly. Computers aren't so lucky. You can plug a webcam into a computer and it will be able to record what it sees perfectly, but allowing the computer to understand what it sees is a great challenge.

A two year old can see a dog and say "Dog", even if it has never seen that particular dog before. This is a difficult task to perform. First of all, you have to work out which parts of the image are of interest. The scene containing the dog will also contain many other objects – let's call them the background – so the first challenge is to work out which parts of the image are foreground and which are background. Think about an image as being a matrix of pixels, just as a computer sees them. Looking at each pixel in turn, how do you know whether any given pixel is part of the foreground or the background? Answering this question is particularly difficult when you do not know what you are looking at. You need to know that you are looking at a dog before you can decide which parts of the image represent the dog, but you can't know what you are looking at until you have already separated out the foreground (the dog) from the background.

In other words, how do you tell where the dog starts and ends if you don't even know that you are looking at a dog?

Computer vision techniques are all designed to manipulate the pixels in an image in a variety of ways so that the computer may achieve the following:

- Identify different objects in an image and distinguish foreground from background
- Decide which parts of an image are important
- Identify objects in a scene as being examples of known objects
- For moving images, work out how objects in one frame relate to objects in the next frame

A key tool in the processing of images for computer vision is known as convolution. Convolution is a simple computational method that involves a small matrix of pixels known as a kernel. Think of the kernel as a very small image (5 x 5 pixels, for example). When we use a kernel to process an image, we say that we convolve the

image with the kernel. The process of convolution is very straight forward:

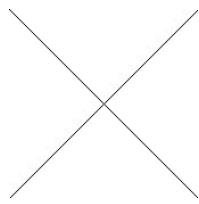
1. Place the kernel so that it is centred over each pixel in the image in turn – this will mean that the kernel covers an number of pixels in the image equal to the size of the kernel.
2. Multiply each value in the kernel by the value in the image below it.
3. Add these values up and divide by the kernel size (average it, in other words)
4. Place the result (the average) in a new image at the location of the centred pixel.
5. Repeat the above steps until the entire original image has been covered. The new image is what we call the ‘convolved image’.

Convolving an image with a kernel has the effect of highlighting anything in the image that has a similar pattern to the kernel and removing anything that doesn’t.

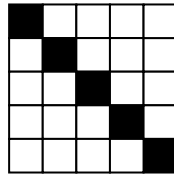
Here are some useful effects of using different kernels:

- Convolving an image with a kernel that consists of all the same value has the effect of blurring the original image. The larger the kernel, the more blurred the image looks.
- Convolving an image with a kernel that contains a horizontal edge highlights horizontal edges in the image.
- Convolving an image with a kernel that contains a vertical edge highlights vertical edges in the image.
- A two-way edge detection kernel known as a Sobel kernel is used to find edges in an image – more of this in the lecture.

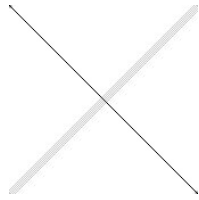
Here is a simple example:



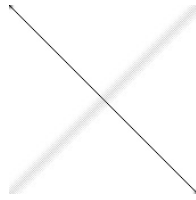
A simple image



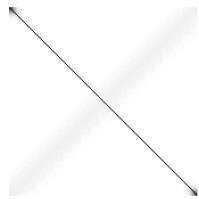
A kernel with a diagonal stripe



After 1 convolution



After 2 convolutions



After 10 convolutions

See how the part of the original image that matches the kernel is left intact, whereas the other part of the image has faded away.

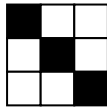
Other operations that are useful in computer vision include:

- Thresholding – Setting every pixel in the image below a certain intensity to zero and the rest to one in the hope that the foreground will be brighter than the background. Often done after blurring to remove texture
- Dilation – Spreading areas of foreground to remove apparent holes in solid objects
- Erosion – Eating away at the edges of solid objects to separate out different objects that are touching each other
- Edge detection – Finding the edges of objects to establish their shape.

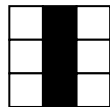
Finally, object recognition can be done in a number of ways, but a simple method is template matching. The shape of the object is compared to a library of known objects in the hope that it matches one. Objects need to be resized or rotated for the comparison. That way we can recognise a small dog or a dog walking up a hill!

Questions for Lecture 3

1. Light entering a digital camera is converted into pixels to be stored as an image. What is a pixel, and what does it tell us about the image captured by the camera?
2. Look at the kernel and the small part of an image below. If the black squares have a value of 1 and the white squares have a value of zero, what would be the resulting value in the middle pixel if the kernel were convoluted with the image?



Kernel



Image

3. What is a common use for the Sobel kernel? Draw the values of a 3 x 3 Sobel kernel.
4. Name two operations that you might need to perform on a shape extracted from an image to allow template matching to work?
5. We have looked at how to extract shape from an image. What other qualities might you want to extract from an image, or an object in an image to aid identification?

Lecture 4. Modelling the environment with propositional logic

It is much easier to behave intelligently in an environment if you have some kind of model of that environment. The human brain is very good at building mental models of the world in which we live. These models help us predict the results of our actions before we act, recognise new situations as examples of things we have seen in the past, and plan complex behaviours.

A computer model needs to offer these same abilities. Specifically, a model should:

- Represent the environment
- Provide a structure for the assimilation of new knowledge
- Be able to change in the light of new evidence, but not too readily
- Dry run without the need for sensor input or actual actuator output
- Be able to generate new facts from existing ones

In this section, we will look at a logic as a way of representing facts about an environment and using these facts to test or infer new facts. A logic is made up of a method for stating facts and a set of rules for manipulating those facts. The first type of logic we will consider is called **Propositional Logic**. It has the following qualities:

- Facts are stated as being either True or False
- Facts are represented using capital letter, e.g P means 'P is true', $\neg P$ means 'P is false'. These single facts are called **literals**
- Literals can be joined into sentences using logical operators: AND, OR, IMPLIES
- Sentences are stored in a knowledge base
- New facts can be checked against the knowledge base and added to it if they are consistent
- Inference can be used to derive new facts from those in the knowledge base

PL Operators

The following symbols are used as operators in PL:

| | | |
|-------------------|---|------------------------|
| \vee | = | Or |
| \wedge | = | And |
| \rightarrow | = | Implies |
| \leftrightarrow | = | Bi-directional implies |
| \vdash | = | Infer (therefore) |

Manipulating sentences in PL

There are a set of rules for manipulating sentences in PL, and some are shown below. You should learn to use these rules to re-write sentences to test their validity.

1. Implication:

$$\begin{aligned} P \rightarrow Q &\equiv \neg P \vee Q \\ P \leftrightarrow Q &\equiv (\neg P \vee Q) \wedge (P \vee \neg Q) \end{aligned}$$

2. Negation:

$$\begin{aligned} \neg(P \vee Q) &\equiv \neg P \wedge \neg Q \\ \neg(P \wedge Q) &\equiv \neg P \vee \neg Q \\ \neg\neg P &\equiv P \end{aligned}$$

3. Distribution:

$$\begin{aligned} P \vee (Q \wedge R) &\equiv (P \vee Q) \wedge (P \vee R) \\ P \wedge (Q \vee R) &\equiv (P \wedge Q) \vee (P \wedge R) \end{aligned}$$

4. Simplification

$$\begin{aligned} P \wedge Q &\vdash P \\ P \wedge Q &\vdash Q \\ \neg(P \wedge \neg P) &\equiv \text{TRUE} \\ P \wedge \neg P &\equiv \text{FALSE} \end{aligned}$$

Example

The so-called Modus Ponens rule is: $(P \rightarrow Q), P \vdash Q$, which means “If P implies Q is true and P is true, then you can infer that Q is true.”

$$\begin{aligned} \text{Replace } P \rightarrow Q \text{ with } \neg P \vee Q: & (\neg P \vee Q) \wedge P \\ \text{Distribute:} & (\neg P \wedge P) \vee (Q \wedge P) \\ \text{Simplify} & \text{FALSE} \vee (Q \wedge P) \\ & Q \wedge P \vdash Q \end{aligned}$$

Reasoning in PL

Here is a simple reasoning rule:

$P \rightarrow Q, Q \rightarrow R$ therefore $P \rightarrow R$. This is the **transitive** rule.

Validity, Satisfiability and Entailment

A set of sentences said to be **satisfiable** if there is a combination of values (TRUE or FALSE) for each of its literals that make it true. It is said to be **valid** if all combinations of values of its literals make it true.

Here are some simple examples.

P is satisfiable (P=TRUE) but not valid (P=FALSE)

$P \wedge \neg P$ is not satisfiable (there is no value of P that makes it true)

$P \vee \neg P$ is satisfiable and valid – there is no value of P that makes it false

No something a little harder:

$(P \vee Q) \wedge (R \rightarrow \neg P)$ is satisfiable by setting $\neg P, Q, R$. It is not valid as P,R is an example where it is false.

$P \wedge (P \rightarrow Q) \wedge \neg Q$ is not satisfiable as there is no set of value for P and Q that make this true. Make sure you understand why.

A brute force approach to checking satisfiability is to enumerate every possible combination of TRUE and FALSE for every literal in the knowledge base until one that is true is found. For example, the check the satisfiability of $P \wedge Q$, you would try:

$\text{FALSE} \wedge \text{FALSE} = \text{FALSE}$ (not yet satisfiable)

$\text{FALSE} \wedge \text{TRUE} = \text{FALSE}$ (not yet satisfiable)

$\text{TRUE} \wedge \text{FALSE} = \text{FALSE}$ (not yet satisfiable)

$\text{TRUE} \wedge \text{TRUE} = \text{TRUE}$ (Satisfiable!)

We will see another way of checking satisfiability in a moment...

Finally, a knowledge base is said to **entail** a sentence if the sentence is true for every combination of values that makes the knowledge base true.

An example:

KB = $P \vee Q$ (The KB tells us that P or Q is true)

Now consider the sentence P (that is, we say that P is true). Is there a combination of P and Q that makes the knowledge base true, but leaves P false? Yes, there is $\neg P, Q$ means P is FALSE and Q is TRUE, making $P \vee Q$ (the KB) TRUE. This is an example where the KB is TRUE, but the

sentence is FALSE, so the knowledge base does not entail the sentence.

Consider the $KB = P \wedge Q$. Now, does this knowledge base entail P? Make sure you understand why it does.

Propositional Resolution

So far, we have talked about checking validity, satisfiability and entailment by enumerating all the possible combinations of TRUE and FALSE. A more elegant method is to re-write the sentences involved to provide a proof. This is done using a process called **resolution**. Any set of sentences may be re-written in the form:

$$(A \vee B) \wedge (C \vee D) \wedge \dots$$

That is a conjunction (string of ANDs) of disjunctions (terms separated by ORs).

This is called **conjunctive normal** form (CNF) and allows you to simplify a set of sentences by resolving any pair of clauses that contain both a literal and its negative (e.g. P and $\neg P$). These are called complementary literals. Here is an example:

$$(P \vee Q) \wedge (P \vee \neg Q) \quad \text{resolves to} \quad P$$

In other words, $(P \vee Q) \wedge (P \vee \neg Q)$ is true when P is true and false when P is false, so it can be replaced with P. Draw the truth table for P, Q and $(P \vee Q) \wedge (P \vee \neg Q)$ to verify this for yourself.

How do CNF and resolution help with checking proofs?

To check that a set of sentences is **satisfiable**, convert it to CNF and look for an empty clause (that is, an empty disjunction – where a literal and its negative cancel each other out leaving no other literals in the clause. E.g. P, ..., ..., $\neg P$). If you find an empty clause, the set of sentences is unsatisfiable. If no empty clause is produced and no further resolution is possible, then the sentences are satisfiable.

To check that a set of sentences (or a knowledge base) **entail** a given sentence, add the negation of the sentence to the knowledge base and perform resolution in search of an empty clause, just as above. If an empty clause is found, then the knowledge base does entail the sentence. This is because you have proved that the knowledge base with the negative of the sentence is not satisfiable (see how the process is the same as the satisfiability checking above).

Questions for Lecture 4.

1. “Heads I win, tails you lose”. State the sentences involved in specifying this in PL and show that you can never win. Remember that if I win, that implies that you lose.

Let: I = I win, Y = You win, H=Heads, T=Tails.

$I \rightarrow \neg Y$ (If I win, you lose)
 $Y \rightarrow \neg I$ (If you win, I lose)
 $H \rightarrow I, I \rightarrow \neg Y$ therefore $H \rightarrow \neg Y$ (transitive) –
Heads, I win, so you lose
 $T \rightarrow \neg Y$ So you lose.

2. Show that $\neg P \rightarrow Q, \neg Q$ implies P by rewriting $\neg P \rightarrow Q, \neg Q$

$\neg\neg P \vee Q, \neg Q$
 $P \vee Q, \neg Q$
P

3. Draw a truth table that shows that $(P \vee Q) \wedge (P \vee \neg Q)$ resolves to P

4. Convert the sentence $P \wedge (Q \rightarrow R)$ to CNF

Lecture 5 – First Order Logic (FoL)

Propositional logic is useful when you want to state a single fact about a single object. It is not useful, however, when you want to state more general facts. If you want to state that all birds can fly, you need to identify every bird in existence and state that it is true that each can fly. That could take a while.

First order logic has two main extensions to propositional logic in what can be represented, but the rules of combination and inference remain. First order logic's extensions are:

- The ability to describe features, qualities or relationships between things
- The ability to state that one, some or all examples of something have these qualities

For example, you could state that some birds can fly, all fish can swim or all people have parents at some point in their lives.

Here are the ingredients of first order logic:

- **Constants**, refer to things in the environment, usually by name. Examples are “Car”, “Person”, etc.
- **Predicates** tell you something about the constants or the relationship between them. Examples are “Swims”, “Is the father of” etc.
- **Functions** take a constant as an input and produce another constant as an output. For example, “Colour(my car) = Red”
- **Variables** are used in place of constants when making more general statements. You will see them in action below.
- **Quantifiers** tell you something about the scope of a statement and they are described below:

There are two main quantifiers used in FoL:

The **universal quantifier** tells you that what follows is true for ALL things. We use the upside down A symbol: \forall

$\forall x \text{ Blue}(x)$ means everything is Blue and is read “For all things, x, x is Blue”.

The **existential quantifier** tells you that what follows is true for at least one thing. We use a backwards E symbol: \exists

$\exists x \text{ Blue}(x)$ means at least one thing is Blue and is read “For all things, x , there exists at least one x that is Blue”.

$\forall x \text{ Blue}(x)$ means that you should take every object in the environment, replace the ‘ x ’ in the statement with the objects’s name (its constant) and join the resulting list with ANDs.

So in a world with only blueberries and blue cars:

$\forall x \text{ Blue}(x)$ expands to $\text{Blue}(\text{blueberries}) \text{ AND } \text{Blue}(\text{cars})$

$\exists x \text{ Blue}(x)$ expands in a similar way using OR. Let’s make our cars red now

$\exists x \text{ Blue}(x)$ expands to $\text{Blue}(\text{blueberries}) \text{ OR } \text{Blue}(\text{cars})$ is still true because one of the two things is blue.

Here is a final important point about using quantifiers in FoL

$\forall x \text{ Swan}(x) \rightarrow \text{Swims}(x)$ tells you that all swans can swim. Read it as “For each thing in the environment, which we will call ‘ x ’, x being a swan implies that x can swim”.

Two things to note here:

1. $\forall x$ refers to all **objects** in the environment, not just all swans so although the meaning might be reduced to all swans can swim, it actually says it is true for all things that if the thing is a swan, then it can swim. So it is also telling you that $\text{Swan}(\text{car}) \rightarrow \text{Swims}(\text{car})$, i.e. if it is true that a car is a swan (which it isn’t) then it is true that the car can swim. Note that the statement is still true as a whole.
2. Note the use of the implication sign, \rightarrow and compare it to the next point below:

$\exists x \text{ Blue}(x) \wedge \text{Round}(x)$ tells you that there is at least one round blue thing in the environment. Read it as “For at least one thing in the environment, which we will call ‘ x ’, it is true that x is blue and x is round.

Two things to note here:

1. It still applies to all things, just as above, but each instantiation of x with an object from the environment is separated by OR (\vee) so it doesn’t actually tell you which objects it applies to. In an environment with just blueberries and bananas, $\exists x \text{ Blue}(x) \wedge \text{Round}(x)$ expands to:

$(\text{Blue}(\text{blueberry}) \wedge \text{Round}(\text{blueberry})) \vee$
 $(\text{Blue}(\text{banana}) \wedge \text{Round}(\text{banana}))$

which tells you that blueberries are round and blue or bananas are round and blue (or both are) but it doesn't tell you which.

2. We now use And (\wedge) rather than the implication sign \rightarrow . See the exercise below and work out what it would mean to say $\exists x \text{ Blue}(x) \rightarrow \text{Round}(x)$ and why it is of no use to say it. I'll give you a clue: Note that $\text{Blue}(\text{banana}) \rightarrow \text{Round}(\text{banana})$ is true because $\text{FALSE} \rightarrow x$ is always true, regardless of x .

Make sure you remember this rule:

Implication (\rightarrow) for $\forall x$
And (\wedge) for $\exists x$

Finally, note that $\exists!$ means that there exists exactly one.

Lecture 5 – Questions

1. Recall from propositional logic that

$$P \rightarrow Q \quad \equiv \quad \neg P \vee Q$$

Show that $\text{FALSE} \rightarrow x$ is always true regardless of the value of x

2. Work out what it would mean to say $\exists x \text{ Blue}(x) \rightarrow \text{Round}(x)$ and why it is of no use to say it. To help you, imagine an environment with only blueberries and bananas and write the expansion (using OR) of the statement. Make use of the fact in question 1.

3. What do the following FoL sentences mean in English?

- a. $\forall x \text{ Porche}(x) \rightarrow \text{Fast}(x)$

- b. $\forall x \text{ Fiat}(x) \wedge \text{Age}(x) > 5 \rightarrow \text{Rusty}(x)$

- c. $\exists x \text{ Porche}(x) \wedge \text{Red}(x)$

- d. $\neg \exists x \text{ Fiat}(x) \wedge \text{Age}(x) > 5 \wedge \neg \text{Rusty}(x)$

- e. $\exists! x \forall y \text{ Person}(x) \wedge \text{Car}(y) \rightarrow \text{Owner}(y) = x$

4. Now write the following in FoL

- a. All red cars are fast

- b. Some blue cars are fast

- c. There are old pilots and bold pilots, but no old, bold pilots

Lecture 6 – Machine Learning Concepts

We have suggested that knowledge based systems have the ability to learn because they can check new facts against their existing knowledge and store the new facts in the context of that knowledge. You could justifiably argue that this is simple rote learning and not a true attribute of intelligence. Humans can perform rote learning, but we are also very good at learning from experience, and generalising to new, similar experiences.

Tasks that require this kind of learning involve physical skills (you can't learn to ride a bike by rote), recognition of objects (you don't need to have seen and memorised every chair on the planet to know what to sit on when you enter a restaurant), and predicting future events (you learn from experience that you should take a rain coat when the sky looks dark in the morning).

In this section, we will be looking at the challenges and solutions involved in allowing computers to learn from experience. Agents that can learn are useful for a number of reasons. Often it is impossible to know the rules required to perform a task, and so impossible to program a computer to do so. In other cases, the environment changes, requiring the agent to change with it. This is best achieved with an agent that can learn, or adapt, rather than with constant re-programming.

Almost all learning follows the pattern of sensed input leading to some outcome that is to be learned. Learning to recognise a seat involves sensors (a camera, say) to see the chair and an outcome which is the classification of the object as a chair. Learning to ride a bike involves sensor input about balance, speed, obstacles ahead, etc. and requires a learned outcome that is the correct action to take to stay on the bike.

We will look at three types of learning in this section:

- Supervised Learning
- Un-supervised Learning
- Reinforcement Learning

There are many techniques for each type of learning – the differences between them are in the way the agent receives information, data, or feedback from the environment, and what it does with that data.

Here are the key differences. Make sure you understand them.

Supervised Learning

Remember we said that learning is about going from sensed inputs to a desired output? With supervised learning, this relationship is made explicit as the input, output pairs are presented to the agent together. These techniques are called **supervised** because the answer (the output) is given by the teacher. To teach an agent to recognise chairs, the supervised technique would show the agent lots of examples of the sensory input generated by many different types of chair and the agent would learn what they had in common.

Unsupervised Learning

There is not always a teacher to provide the correct answer during learning. In such cases, the agent must try to organise the inputs to its sensors to find patterns that are useful. It might observe that other agents tend to sit on things that have common features (a seat, legs, etc.) and so learn that the sensory input from such things may lead it to sit down. This can be summarised as noticing which things go together. There may be a time lag between one thing and another (dark clouds followed by rain) and there is a challenge in knowing which early sensor inputs led to which later occurring outcomes.

Reinforcement Learning

In this final type of learning, the agent never sees the correct output, it simply receives feedback about how well it is doing. This feedback – usually either punishment or reward – is the **reinforcement** mentioned in the technique's name. This is learning by trial and error. Learning to ride a bike is a good example of reinforcement learning – you try an action and you get either reward (you travel along) or punishment (you fall over). Reinforcement can come from the environment (as in the bike example) or a teacher.

Representation

The ability to generalise – that is, process something that has not been sensed before in a way that is most likely to be correct based on its similarity to things that have been sensed before – means that what has been learned needs to be represented as something more sophisticated than a list of facts. Often, the representation is some kind of functional model of the relationships or patterns in the sensory data to be learned.

The next lecture describes a few models for machine learning in more detail, but for now we will just consider the features, functions and requirements of a machine learning model. We have already said that most learning is concerned with the relationship between a pattern of sensed inputs and some outcome. The outcome is given as the target during supervised learning and during reinforcement learning, the outcome is the reward or punishment. With unsupervised learning, the outcome is not always clearly defined – it might just be to recognise the inputs as being familiar.

In cases where the inputs and outputs can be represented as numbers, one approach to machine learning that is common is known as **curve fitting**. If you wanted to represent the relationship between the heat you feel and the distance you stand from a fire, you could try standing at varying distances and note the temperature of your skin. A simple, fact based learning algorithm might just store each distance, temperature pair to search in the future. Imagine you collected data from 10cm and 15cm, but not 14cm. You couldn't use the list to generalise to the 14cm distance as it is not stored in your memory. Now let's say you work out the function that takes distance as input and provides temperature as output (say, $t = 100 - d^2$). Now you can generalise because there is no distance that will not provide a temperature. The process of starting with data (usually input / output pairs) and discovering a function that describes the relationship between the inputs and the outputs is known as curve fitting. In such cases, the model is represented by the function that has been discovered (or, as we shall see, learned).

Lecture 6 – Questions

1. If I show you 100 examples of different dogs, and tell you that they are dogs, and then do the same with 100 cats, and ask you to learn to tell them apart, which kind of learning (supervised, unsupervised or reinforcement) are you performing?
2. If I showed you all 200 animals from question 1, didn't tell you which were which, and asked you to work out how many different types of animal I had shown you, which type of learning would you have performed?
3. Imagine the dogs bit you and the cats didn't. What type of learning would you use to learn whether or not to stroke each animal?
4. The ability to generalise is essential for a learning agent. Using the cat and dog example from above, describe a scenario where generalisation would be used.
5. Describe the difference between representing knowledge as a database of facts and as a general model. How might we build a model to represent the relationship between people's weight and height?

Lecture 7 – Machine Learning Techniques

In this lecture, you will be introduced to some machine learning techniques. Each technique would fill an entire course by itself, but from this lecture, you should know the following about each:

- The circumstances in which it would be used
- How knowledge is stored
- The basic idea behind the learning algorithm
- How generalisation is achieved

The three techniques we will consider are:

- Neural Networks
- Decision Trees
- Bayesian Networks

All three have some things in common. They all learn from data, and they require that data to be organised in a certain way. The data must be represented as a list of values assigned to a set of variables. Variables describe something that has been measured or observed. Variables that describe a person might include age, height, weight, name, gender, etc. The values that each variable can take make up the data. Possible values for gender are Male and Female. Possible values for weight are any real number within a sensible range.

Variables are arranged into one of two roles, which we will call inputs and outputs. The goal of learning is to discover the relationship between the inputs and the outputs and represent that relationship as some kind of model. The three techniques mentioned above are different models for representing the learned relationship, as you will see.

Here are some examples of machine learning applications that have used the techniques discussed here:

- Learning to predict when a driver will fall asleep at the wheel based on their use of the steering wheel
- Learning to predict which customers who visit a web site will go on to buy something
- Detecting when a cow is at its most fertile so that it may be introduced to a bull
- Learning the best combination of letters and phone calls to make to collect an unpaid debt

These examples all have one thing in common. During the learning phase, we knew the output that would later need to be predicted. In the example where the computer learned to spot sleepy drivers, we collected the data by asking people to drive around an airfield until they were nearly asleep. While they drove, we measured their EEG activity using electrodes on their scalp. This told us how awake they really were. The computer learned the relationship between steering behaviour (the inputs, which would be known in the future when the system was actually working in the car) and the state of alertness of the driver (which would not be known when the system was working in a production car, as the electrodes would not be fitted!).

So, the machine learning approach is often as follows:

1. Collect data describing what the machine will be able to sense (or measure, or get data describing) along with data describing what will need to be predicted, or guessed in the future. These are the inputs and outputs, respectively.
2. Learn the relationship between the inputs and the outputs
3. When using the system that has been created in this way, measure the inputs and predict the outputs.

You should also note that it is possible to relax the distinction between inputs and outputs, and have a machine learn the relationship between all the variables. After learning, when the machine is in operation, it is then possible to provide it with the values of any subset of variables and it will predict (or infer) the values of those that you could not measure at the time.

Make sure you understand the roles of the following and the relationship between them:

- Variables and values
- Inputs and outputs

Now we can look at the techniques themselves. These techniques are all supervised. We provide the inputs and the outputs and the machine learns the relationship between the two.

Neural Networks

Neural networks are inspired by the way in which the brain learns. In reality, they are not as flexible or robust as the brain, but they do provide a powerful method for training computers using data. A neural network is, as the name suggests, a network of connected 'neurons'. The neurons are actually very simple functions and in most networks every function is identical. The connections in the network are of varying strengths – we call them weights, and it is these weights that represent the knowledge in the network.

The functions in the neurons have inputs and outputs, just like any function. The inputs come to the neuron along the weighted connections and are passed through the function. The result is sent along the weighted connection that leaves the neuron, connecting it to the inputs of yet more neurons. One final point to note is that the output of each neuron is multiplied by the weight on its connection and the result forms the input to the target neuron.

If you have lots of simple functions (the neurons) and you join them together in the correct way, you can represent any new function of arbitrary complexity. Neural networks are sometimes known as 'universal approximators' due to their ability to represent any function.

Learning in a neural network is the process of changing the weighted values on the connections to cause the network as a whole to become the correct function for representing the relationship between the inputs and the outputs. A very common method for choosing the weights is known as error back propagation. This learning process works as follows:

1. Data containing input and output patterns is collected
2. The network starts with random weights – all the connections are in place, but their weights are just random values
3. The following is then repeated many times:
 - a. Input patterns are presented to special neurons in the network designed to receive inputs – you can think of these as simple sensor neurons
 - b. The network passes the values along its weights, through the functions in its neurons (which are identical, and don't change – remember) until an output is generated at another set of special neurons – the output neurons.

- c. The answer given by the network is compared to the true answer (which is known because it is in the training data) and an error value is calculated – usually the difference between the two values.
 - d. The error is used to change the weights slightly so that if the same input pattern were seen again, the error would be slightly smaller
4. With enough data and sufficient repetitions, this process can find the set of weights that produces the smallest possible error across the training data and hence, represents the relationship between the inputs and outputs.

Notice that the structure of the neural network (input neurons and output neurons) mirrors the way we specify a machine learning problem – inputs and outputs. Generally, there is a single input unit in the network for each variable in the input set. This makes it difficult to add new variables to an existing network, so you need to identify the right variables when you collect the data.

Decision Trees

Decision trees are another way of representing relationships learned from data. They associate input patterns with output patterns in a similar way to neural networks. Generally, however, the tasks that decision trees are used for involve classifying an input pattern as being an example of one of a set of categories that the agent has previously seen. Nodes of the tree represent single variables from the data and branches represent each possible value (or range of values for numeric data) the variable can take. Leaf nodes represent the class of an example pattern that takes the route through the tree that leads to that leaf.

Decision trees learn by splitting the data into increasingly small subsets so that each split separates out the most examples possible. See the lecture slides for an example tree. It is possible to design a decision tree by hand, rather than using data. They are also easy to change if you need to add an explicit fact to a tree that has been learned already.

Bayesian Networks

Bayesian networks represent knowledge by a series of related tables of probabilities. The tables of probabilities represent how often each variable takes each of its possible values. Variables are linked to other variables in the network and their tables show the conditional probabilities of one variable taking each value, given that the value of the related variables are known. Bayesian networks are

built by counting the probabilities in the data. They provide answers using Bayes' rule to calculate the probabilities that are not known from the values that are.

If you want to know more about machine learning and conditional probabilities, an easy introduction can be found here: <http://www.cs.stir.ac.uk/~kms/schools/rps/index.php> look for the information sheet linked to at the bottom of the page.

Like decision trees, Bayesian networks can be designed by hand, but in reality using machine learning to build them from data is preferable for all but the simplest of networks.

Lecture 8 – Searching and Planning

In the next two topics, we look at two essential related abilities for an intelligent agent – searching and planning. We have already seen that machine learning can be used to model the world and make predictions about it. Imagine a simple model that predicts the result of any action we take. The inputs to the model describe the action and the outputs describe the result. Some results are good, some are bad. Without a model, we might have to keep trying different actions until we got the result we wanted. With a model, we could ‘dry run’ each possible action until the model predicted the result we wanted. We could then perform that action in reality – hopefully achieving the desired result. We have performed a very simple search. We put each possible action through the model, predicted the result, and kept going until we found the result we wanted.

Intelligence is rarely simply about picking one optimal action and performing it. Intelligence requires the agent to perform a series of actions, one after the other, all of which move the agent towards a goal. The performance measure helps you choose at each step, but it might not direct you to the goal. Deciding on a series of actions to move towards a goal is known as **planning**.

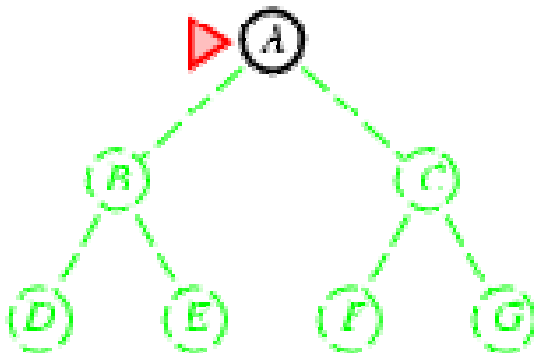
Imagine your life is made up of binary decisions, and you make one decision a day. On day 1, you have two choices. On day 2, you have two choices for each of those made on day 1, so four possibilities in all. By day 3 you could have taken 8 different combinations, then 16 on day 4 and so on. Imagine that a certain combination of decisions is correct – it allows you to reach the goal – but you don’t know how many steps there are from your current state to the goal. If your model can predict the outcome of each decision, you can predict the outcome for any series of actions (a plan). How do you find the right plan?

You can think of the space that you need to search as a tree of possibilities. On day one, it branches in two – one branch for each decision. At level 2, which represents decision 2, each node branches again, producing 4 nodes at the next level. The tree might keep branching forever as more decisions are made. Each node of the tree is an outcome. Some outcomes (perhaps only one, but possibly more than one) are desirable. These are the goal outcomes that we have talked about already. Finding a goal outcome involves traversing the tree of possibilities until you arrive at a node that represents a goal.

There are a few different approaches to searching the tree. They are: Depth First, Breadth First, Limited Depth First, and Iterative Deepening Search. The lectures describe these approaches in full and also tell you about their relative advantages and problems.

Lecture 8 – Questions

1. When faced with a decision with several options, how would an intelligent agent use a model to pick the best option?
2. What is the relationship between a decision and a plan?
3. Which tree searching method runs the risk of continuing down a branch forever even though the goal is only a few steps from the start point? Why?
4. Which tree searching method solves the problem above most efficiently?
5. Which tree searching method uses the most memory? How much does it use for a binary branching tree of depth d ?
6. How do predictive models fit into the use of a tree search for planning?
7. Look at the tree below. Draw on it the route taken by each of the tree searching methods mentioned here



Lecture 9 – Informed Searching

In the last lecture we talked about tree searching. This provided a structured way of searching for a goal state. It ignored performance measures, however. In this lecture we will see how performance measures can be used to speed up searching.

Once again we use a model, but this time the model predicts the performance score of an action. Now, the task of searching can be thought of in one of two ways. We might just search for the series of actions that gives the highest total performance score, or we might use the performance score as a guide to help us navigate more quickly towards the goal. This second point assumes that actions that takes us closer to the goal score a higher performance measure than those that do not. You can think of this like the child's game hotter, colder, where an object is hidden in a room and the child finds it guided by being told how close (hot) they are getting.

An obvious performance measure is distance to the goal. This assumes that the distance is measureable (which might be tricky, given that we don't know where the goal is).

Imagine a model that maps actions from different states in the environment to performance measure. You could build such a model using one of the machine learning methods discussed already. This model can be thought of as a function which, if it were plotted against environmental states, would show a curve with a high point (global maximum) at the goal. Searching is the process of finding the state that produces that high score. This is easier if the curve smoothly leads up to this goal state.

In any given state in the environment, the agent could then predict the performance measure for each possible action available to it and take the one with the best performance measure. The agent will move closer and closer to the goal using this method, which is known as **local search**.

One common problem with local search is that the best way to the goal might require a local, temporary decrease in performance measure. Sometimes you need to move away from the goal to avoid a local obstacle, for example. Sometimes an agent will find that they are not at the goal, but that every possible action makes the performance measure worse. This is known as the problem of **local maxima** (or **local minima**, depending on whether you are

trying to maximise a score or minimise a cost). See the hill walking example in the lectures for more on this.

In some environments, searching for the state with the best performance measure score involves moving from state to state in small steps, picking the best one each time. This is what was described above. In other environments, it is possible to place the agent in any desired state. In such cases, we can choose one of a number of optimisation techniques to decide which state to choose.

We will look at two types of optimisation method. Firstly, **steepest descent searching**, which is used to find the point where a function reaches zero. This involves estimating the slope of the performance measure function and taking big steps to where that curve is most likely to reach zero. The two methods you need to understand are called the **secant method** and the **Newton-Raphson method**. These methods are demonstrated in the lectures and looked at in the tutorial.

The second type of search technique we will consider does not use the slope of the fitness function. Rather, there are a number of ways of deciding where to look next. Here we consider two methods based on the idea of a **population** of collaborating searches, rather than a single search. The idea here is that a number of searches all proceed in parallel, starting at different parts of the search space. Each solution receives a performance score and these scores are used to decide where to look next. However, instead of each search following its own independent path, the searches communicate and collaborate to speed up the overall search.

The first type of population based search we will consider is the genetic algorithm (GA). Based on the ideas of biological evolution, a GA produces a population of potential solutions and scores each with a performance measure – known as the fitness function. It then takes the better performing solutions and combines them (usually in pairs) to produce offspring that inherit some features from each parent. This produces a new generation of solutions and the process is repeated. Offspring also mutate a little to ensure that new solutions are tried.

Another biologically inspired method of searching is called particle swarm optimisation (PSO). Again, a population of searches is used but this time, searches advance by moving the poor solutions towards the best solution in small steps. If a better solution is found on the way, the other searches start moving towards this new best solution instead. Think

of a swarm of bees looking for flowers. When one finds some, it signals the others and they start to move towards it. If another bee finds better flowers on its way, it signals this fact and the bees start towards this new solution instead.

Here are the things you need to understand from this lecture:

- How a search space and its performance measure can be represented as a surface (or a curve) and how the gradient of that surface can be used to speed up searching;
- The names of and general idea behind the Newton-Raphson and Secant search methods;
- What the problem of local minima is and how it might be solved
- How GAs and PSO work – you need to know more detail than is given in this document, specifically:
 - How crossover (breeding) and mutation are implemented
 - Different ways of selecting which solutions will breed and which will die

Lecture 10 – Can a machine be intelligent?

The question of whether or not a machine could be intelligent has interested philosophers, engineers and sci-fi writers for a long time. New technologies bring new expectations, and none more so than the digital computer. A computer can remember millions of facts and perform complex mathematical calculations at incredible speeds – you'd need to be pretty intelligent to do that, right? Actually, no. Searching a database or multiplying numbers together requires nothing more than simple rule following. You should have learned from this course that intelligence involves a lot more than that. You have seen the state of the art for machine vision, learning, reasoning and planning and you have probably realised that machines have a long way to go before one could look at a beautiful sunset and feel inspired to write a moving poem about the moment.

Never mind the current technology, though. Is there a reason why a machine could never be intelligent? Is there some fundamental aspect to the human (or animal) brain that a computer could never simulate? These are the questions we will consider in this lecture, and we will be guided by the work of one of the better examples of an intelligent human, Alan Turing.

In 1950, Turing wrote an influential paper on machine intelligence. He introduced the first concepts you'll need to learn:

- **Weak AI** refers to the idea that a computer might be able to **act** intelligently, but still lack an internal understanding
- **Strong AI** refers to the idea that a computer doesn't just act intelligently, but somehow **is** intelligent. Now, a lot of argument exists about what is required to actually be intelligent, rather than just behaving so. Words like consciousness and understanding are used, but no clear consensus exists, as we shall see.

Turing devised a game, now known as the **Turing test**, which would test whether or not a computer had achieved weak AI. The game involved two players - a computer in conversation with a human - being viewed by a human judge. In the game, both players must try to convince the judge that they are the human, while asking the other questions that might catch it out. If the computer wins as often as the human, then the computer displays weak AI.

Anti-AI thinker, John Searle conceived his own ‘game’, the Chinese room, as an illustration of why the computer that passes the Turing test is not really intelligent. He argued that an English speaking person in a room full of books of rules could, given time, accept written Chinese questions and use the rules to provide intelligent responses, in Chinese, without understanding any Chinese. The person would be like the computer, showing weak AI but understanding nothing (so not having strong AI).

Several arguments are made against the Chinese room argument – the most important ones involving the fact that understanding requires meaning, which is the connection between symbols and things in the real world. Given that connection, the person in the room would eventually learn Chinese and so gain understanding, just as a computer would.

Turing foresaw Searle’s argument, and many others in that 1950 paper, where he listed 9 likely objections to the idea of an intelligent machine, along with reasons why the objections are false.

The objections are covered in the lecture, and in Turing’s paper (of course) available here:

<http://loebner.net/Prizef/TuringArticle.html>

In summary, they are:

- The Theological Objection
- The "Heads in the Sand" Objection
- The Mathematical Objection
- The Argument from Consciousness
- Arguments from Various Disabilities
- Lady Lovelace's Objection
- Argument from Continuity in the Nervous System
- The Argument from Informality of Behaviour
- The Argument from Extrasensory Perception

Most of the arguments are covered by this shorter list of ideas:

Dualism – the idea that there is some separate ‘quality’ to the human mind that is not part of its physical substance. People use words like ‘mind’, ‘consciousness’, ‘free will’, even ‘soul’. These things are hard to define and harder still to study. If it is true that some aspect of intelligence is the result of this unknowable ‘mind’, then computers, which don’t have minds, can never be intelligent.

Determinism – the idea that computers can only follow rules, but people are somehow excused from that limitation. Simple arguments against AI state that it is impossible to think of every rule an intelligent machine would need, so impossible to program one. We have seen how computers can learn and evolve behaviour they have not been programmed to display. Other arguments are about creativity – writing a poem and understanding its emotional charge is an often cited example. If there is something about creativity that is not governed by rules, what is it?

Things to remember:

- The difference between weak and strong AI
- Exactly how the Turing test works and which type of AI it tests for
- Searle's Chinese room argument and a response to it
- Turing's nine objections and an answer to each. How do ideas of dualism and determinism fit with these objections
- Think about how some of the techniques we have studied in this course relate to the objections discussed by Turing

Lecture 10 – Questions

1. What is the difference between strong and weak AI?
2. Name two of the objections to AI that Turing addressed. Pick one that addresses the dualist argument and one that is about determinism. Explain why they address each subject.
3. AI is goal oriented rather than rule following. Give examples of how a computer can solve a problem that is stated as a goal rather than as a set of rules and discuss how this ability addresses the objection to AI that it would be impossible to write a rule for every scenario that a computer might meet.
4. Creativity might be described as the cycle of generation, judgement, and re-generation. How does this correspond to the ideas of a genetic algorithm?