Machine learning in Python summer school

Lecture 1:
Neural networks & convolutional networks

Ben Harvey

1

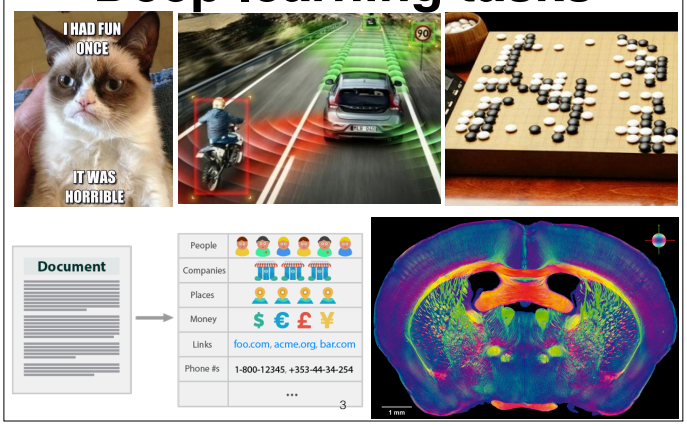
Welcome to our neural networks day.
This morning we will look at neural networks and deep convolutional neural networks, a type of machine learning network whose design is inspired by the function of neurons in the brain, and the structure of networks among these neurons.

Why deep learning?

- Neural networks are a very powerful way to link an input state with a desired output state (i.e. machine learning)
- Deep learning is particularly useful in tasks that are:
  - Hard/impossible to describe using formal mathematical rules
  - BUT easy for humans to perform
    - Intuitive or automatic
- Simulation of neural computation

2

Deep learning tasks



3

Image processing
Game opponents in complex games
Natural language processing
Simulation of biological neural systems

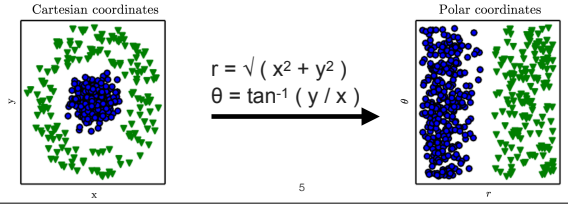
Deep learning approach

- Learn from experience (machine learning)
- Process inputs through a hierarchy of concepts
  - Each concept defined by its relationship to simpler concepts
  - So, build complicated concepts out of simpler concepts

This is what a human is doing when learning about the world
SO the main conceptual inspiration for deep learning is the brain. As we will see, the design of deep learning systems has also followed the function of the brain increasingly closely.
However, artificial neural networks simplify the processes involved considerably for computational efficiency.

Representations & features

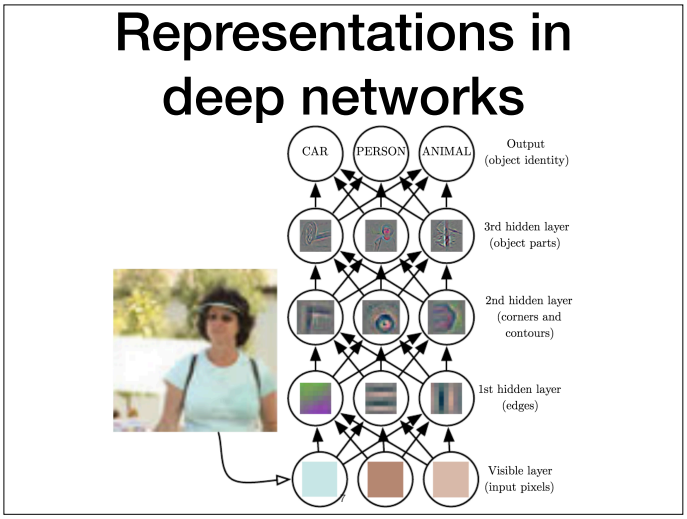
- Machine learning performance depends on the **representation** of the case to be classified
  - What information the computer is given about the situation
- Each piece of input information is known as a **feature**
  - The same feature can be represented in different formats
    - Often easy to convert between formats
  - The chosen format strongly affects the difficulty of the task



A simple task like this can be solved by choosing the right set of features, with minimal learning necessary.
However, for many tasks, it is hard to know which features or formats of the input are important in determining the output.
And these may be high-level features that need to be extracted first.
Deep learning aims to determine which formats of their representations are optimal for solving their task, through experience

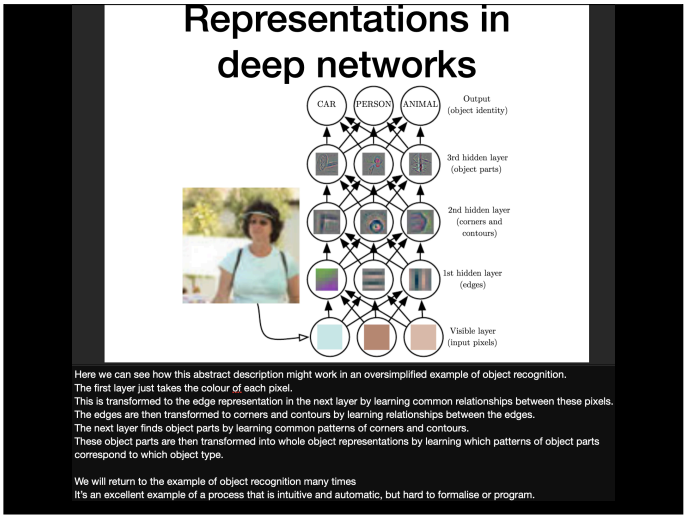
Representations in deep networks

- Useful features may need to be transformed or extracted first
- So deep networks have multiple representations
  - Each is built from an earlier representation
- This can:
  - Transform features to a different format before learning their links to the output
  - Extract complex features from simpler features
- Essentially multiple steps in a program
  - Each layer can be seen as the computer's memory state after executing a set of instructions
  - Deeper networks execute more instructions in sequence
- Just like a computer program, the individual steps are generally very simple
  - Complex outcomes emerge from interactions between many simple steps



Here we can see how this abstract description might work in an oversimplified example of object recognition.
The first layer just takes the colour of each pixel.
This is transformed to the edge representation in the next layer by learning common relationships between these pixels.
The edges are then transformed to corners and contours by learning relationships between the edges.
The next layer finds object parts by learning common patterns of corners and contours.
These object parts are then transformed into whole object representations by learning which patterns of object parts correspond to which object type.

We will return to the example of object recognition many times
It's an excellent example of a process that is intuitive and automatic, but hard to formalise or program.
It is also very useful for computers to do, so we can find images on the internet without a human labelling their content.
Finally, object recognition is a problem that has now been solved, so we can really see how the result works.



A quick note on notes.
All of these slides will be available online, but you will find I use little text on my slides. This works better in class, but is hard to study from or refer back to.
I deal with this by giving you online slides also contain notes with a fairly complete description of what I say. This is very easy to make notes on and study from. Please note that your other lecturers won't give you such extensive notes.

So please focus of listening and understanding rather than taking notes.

You also have quite different backgrounds, and I may assume you have some background that you don't. And as a native English speaker I sometimes go a little fast.

So please stop me if I go over something too quickly or you find you are missing something.

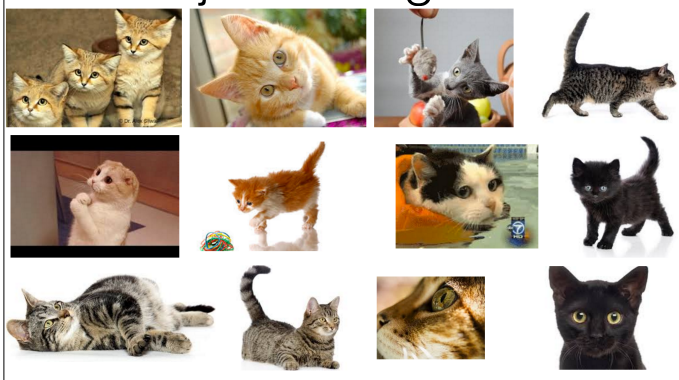
What is a deep network?

- A machine learning network that **transforms** or **extracts** features using:
  - Multiple **nonlinear** processing units
  - Arranged in multiple **layers** with
    - **Hierarchical organisation**
    - Different levels of **representation** and abstraction

9

This is a very broad definition, so we will use an example to see what it looks like
The example we will use in the first half of the course is object recognition.
This has been a major goal for deep learning in recent years, and is now largely solved, so we can investigate in depth how this works.
Object recognition may sound like an easy problem for computer vision, but…

Object recognition

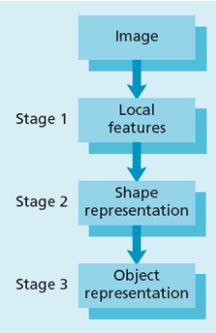


Why is it so difficult?

10

The identity of any object has little relationship to its impression on the retina.
Here we see the result of a google image search for pictures of cats.
This used Google's artificial deep network trained for object recognition.
For this network and also for human vision, objects can be recognised from different viewpoint and sizes, in different positions, and with different lighting conditions.
Also, examples of the same class of object often look very different.
So we can't recognise an object directly from its impression on the eye or camera sensor

The 20th century view of object recognition

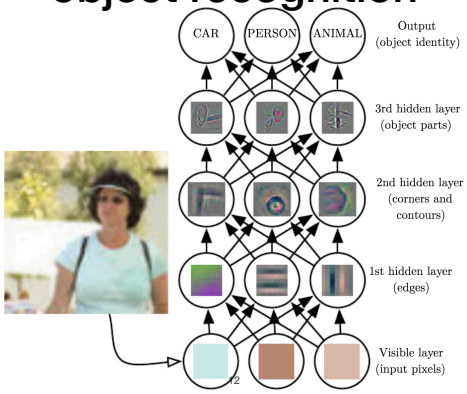


- Stage 1 builds a representation of local image features.
- Stage 2 builds a representation of larger-scale shapes and surfaces.
- Stage 3 matches shapes and surfaces with stored object representations-recognition.

11

Note this is already a multi-layer, hierarchical approach

The 20th century view of object recognition



12

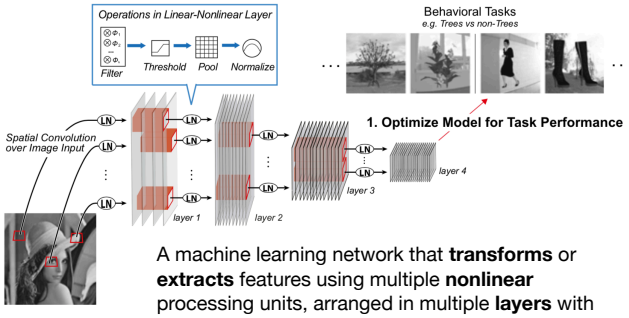
This was essentially the example we looked at earlier, here with the input at the bottom.

This has many similarities with a deep convolutional neural network, and is a good way to start thinking about how they work.
However, there are important differences between deep convolutional networks and what we see here.
Most importantly, the 2nd hidden layer is shown here to straightforwardly respond to corners and contours: straightforward combinations of edges.
Likewise, the 3rd hidden layer is shown to respond to object parts that could in turn be combinations of corners and contours.

Indeed, many objects are built from parts, so simplified parts that we recognise from all angles might let us build an object viewpoint-independent object representation.
However, no one has ever made a program that can do this for a large set of different objects.

It seems that the features considered in a model like this are too human.
When is a feature a corner and when is it a curve? Is there something in between? Can we define a corner, edge or surface so rigidly?
Essentially, all of these steps tend to limit the network to recognise specific examples, rather than generalise to all possible tables, which is the goal here.

A deep network for object recognition



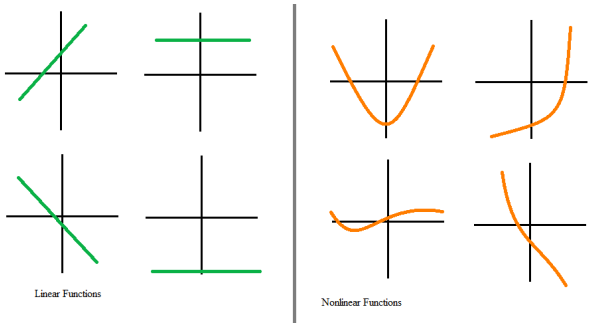
A machine learning network that **transforms** or **extracts** features using multiple **nonlinear** processing units, arranged in multiple **layers** with **hierarchical organisation** and different levels of **representation** and abstraction.

So we can see this network fulfils all the criteria of a deep network.
It takes an input image and transforms its features to extract the class of object the image contains.
It is arranged in multiple layers, with one feeding into the next, forming a hierarchy. This is all much like the 20th-century idea.
The first layer represents the image pixels, with minimal abstraction, while the last layer captures object identity, which is highly abstract for a computer system.

But what happens to get from one to the other is very different from the 20th-century view.
The middle network layers do not respond to concrete concepts that are easy for us to think about, like corners and object parts. Instead they respond to whatever transformation of features is most beneficial for subsequently determining object identity.
This transformation of features is not easy for a human to conceptualise, as we will see.

The last part of this definition is ‘nonlinear processing units’.
But what does nonlinear mean, why is that necessary, and how is it achieved here?

Nonlinear functions



Y=A*X+B

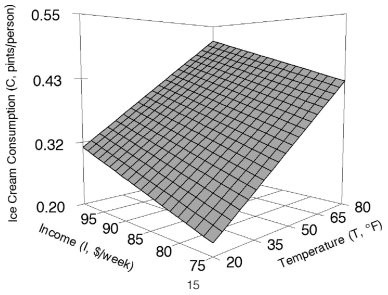
Y≠A*X+B

(Y is any other function of X)

In a linear function, the output (Y) of the function is simply the input (X) multiplied by a constant (A) and then added to another constant (B). The multiplier can be positive, negative or zero.
In a nonlinear function, there can be any other relationship between X and Y.
There must still be a relationship, Y is still a function of X, i.e. Y changes with X in some predictable way.
So we can see that non-linear functions can do a lot of things that linear functions can’t.

Why nonlinear functions?

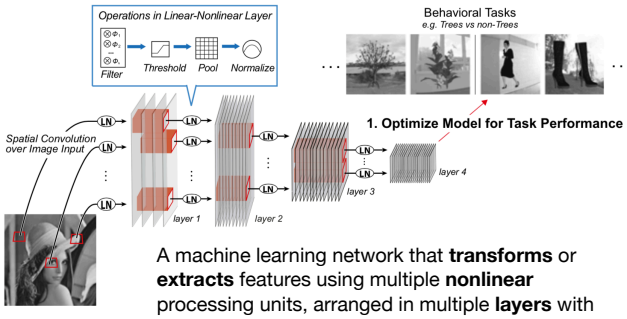
$$Y = B + A_1*X_1 + A_2*X_2 + \ldots + A_p*X_p$$



15

But in the context of deep networks, there is a more important problem with linear functions.
The many layers of a deep network repeatedly perform functions on the output of previous stages. By doing so, more and more features of the input affect the output.
This example, joining together only two inputs, gives an idea of the problem linear functions will face.
Joining multiple linear functions (by addition or multiplication) always results in a linear function of those multiple inputs.

A deep network for object recognition

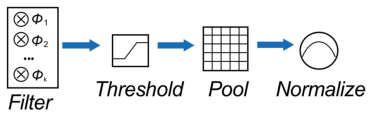


A machine learning network that **transforms** or **extracts** features using multiple **nonlinear** processing units, arranged in multiple **layers** with **hierarchical organisation** and different levels of **representation** and abstraction.

16

In this network, the inputs are the brightnesses of each image pixel.
There is no way these can be multiplied and summed together to give the likelihood this is an image of a tree.
Because, as we have seen, there is remarkably little relationship between an object's identity and the image it produces on the camera sensor.
Indeed, any operation that can be done with only linear functions of the input can be straightforwardly described by formal mathematical rules, so is not a good use for deep networks.

Operations in Linear-Nonlinear Layer



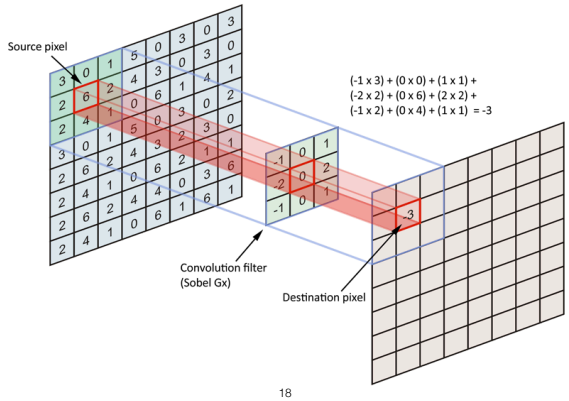
A machine learning network that **transforms** or **extracts** features using multiple **nonlinear** processing units, arranged in multiple **layers** with **hierarchical organisation** and different levels of **representation** and abstraction.

17

In our example, we have a complex nonlinear function with four operations or processing steps. These are filter, threshold, pool and normalise. These are very important to understand, so let's look at these steps in turn.

The output of one operation feeds into the next. The repeating sequence of these four operations effectively forms the layer.
The output of ALL these operations together forms the input to the next repetition of these operations, the next layer.
Note that this is described as a linear-nonlinear layer, which may be a little confusing.
The nonlinear function threshold is very important, but filter and normalise functions are linear. Pooling is also nonlinear, but optional.

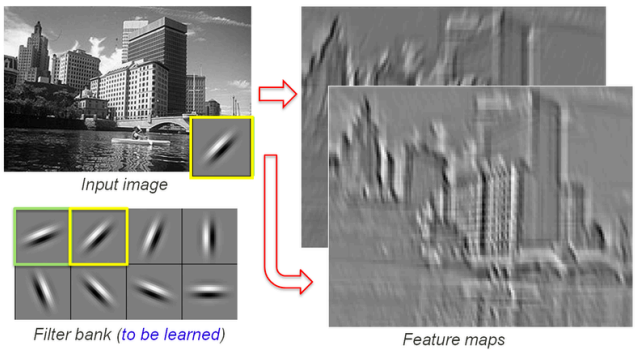
The filter/convolve operation



18

The filter or convolve operation is perhaps the most computationally important.
At the first processing layer, the input is simply the brightness of each pixel.
The convolution step looks for a pattern in a group of neighbouring pixels that corresponds to the convolution filter.
For this filter, this would be dark on the left (low numbers) and light on the right (high numbers).
In convolution, the weights in this filter are multiplied by a group of input pixels with a particular position and the products of these values are summed.
The result gives the match between the filter and a small part of the input image.
If the source pixels follow this filter pattern (dark on the left, light on the right), a high value will result. If the input area is all the same brightness, the result will be zero. If the source pixels are opposite to the filter (lighter on the left) the result will be negative.
Here, the match is poor: the pixel lightness on the left of the input source area is higher than on the right, so the output destination pixel gets assigned a low value.
Then, the filter is moved by one pixel in the input image to give a value for the next output pixel.
All source positions are multiplied by the filter to fill in all the destination pixels. The convolution function is usually implemented using a single matrix operation over the whole input

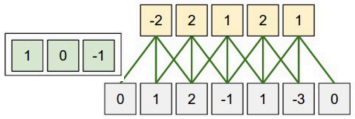
The filter/convolve operation



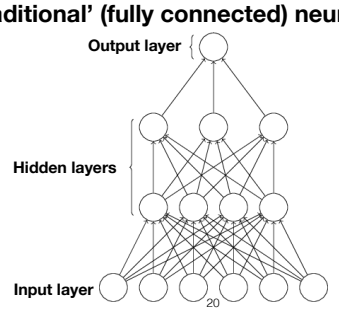
19

The filter we just used is only one example to illustrate the principle.
In fact, a large set of filters are used in parallel to produce multiple maps of where their filter patterns are seen, often called feature maps.
Each 'pixel' shown in each of these feature maps is no longer a pixel in the image, it quantifies the match between the filter and that position in the input image.
This is then the activity of a processing unit or artificial 'neuron'
Here we see a set of eight larger filters with a range of different orientations.
These have been set manually in this example, because we know edge detectors with different orientations are an important early stage of computer and human vision systems.
The pattern of weights within these filters are actually determined by machine learning, though in the first layer the result is an edge detector like this.
The size of these filters will also determine the spatial scale of the feature we can detect. This size is set manually as a parameter of the network layer.

Convolutional network (one layer)



'Traditional' (fully connected) neural network



20

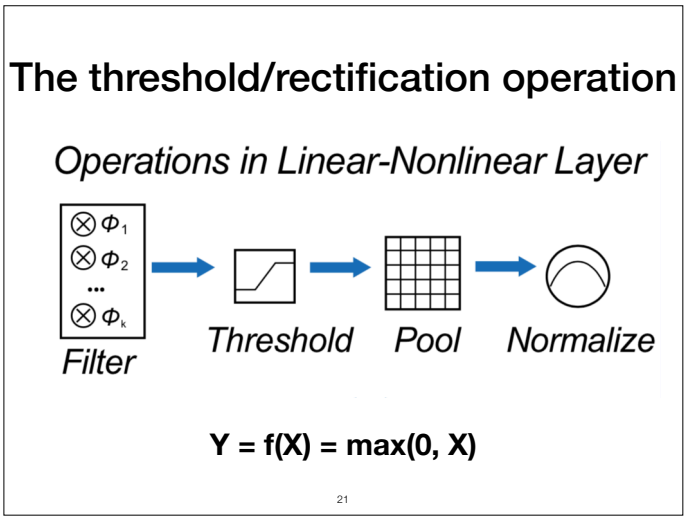
So, how is a convolutional network different from other types of neural network?
Here we see a 1-dimensional convolutional network.
Activation of each upper layer node depends on the product of the same filter (on the left) convolved with a spatially limited set of lower layer nodes.
In an earlier design, a fully-connected neural net, each node in a higher layer is connected to all nodes in the previous layer, with no constraint on the spatial spread of links between them. Again, the pattern of weights within these links is learned.
In a convolutional network, results of convolution with different filters are represented in different feature maps.
But in a fully-connected neural net the input to a higher layer node comes from all low nodes in the previous layer. There is then no need to test a filter at all positions.

So instead of mapping an input onto a feature map of nodes in the higher layer, each single node in the higher layer is a complete analysis of activity in the WHOLE

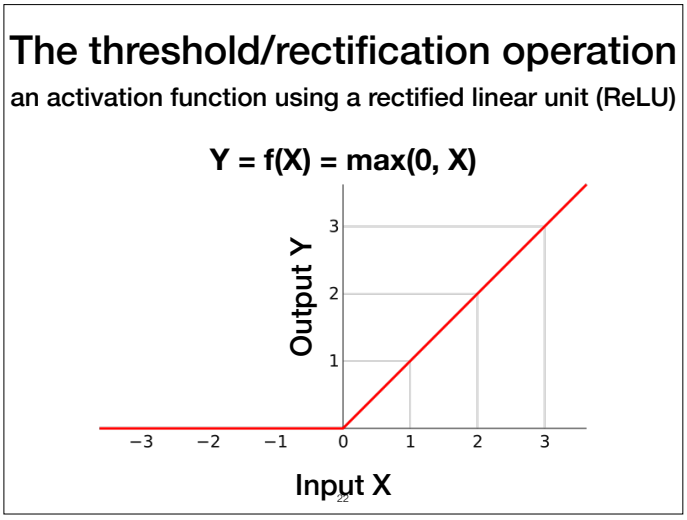
lower layer, with no spatial representation at all.
-In other words, a fully-connected network can find any pattern in its inputs in one layer.
-A side effect of this is that the number of dimension in the input is irrelevant in a fully connected network, so we can simply view each network layer as 1-dimensional.
-Adding another layer will all the network to detect higher-order patterns: patterns within the patterns that are not present in the inputs themselves.
-This gives a lot of flexibility, and I would argue too flexibility for a process like object recognition.

In a convolutional network, the constraint on the spatial spread of links between layers is particularly useful where the input has meaningful spatial relationship between neighbouring nodes.
-The most obvious examples of meaningful spatial relationships are in image processing, where the input layer units are image pixels and the feature maps each give the spatial distribution of the features described by

---



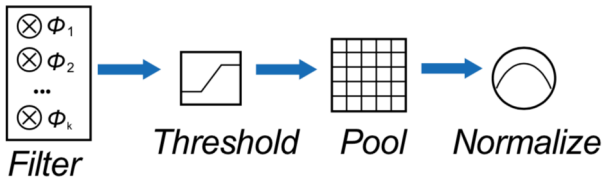
The nonlinearity is introduced by the threshold or rectification operation.
The goal of this operation is to only activate the output feature map if its value reaches a certain level, or threshold.
If we use filters that have a mean of zero, the threshold is typically zero.



So, for values below zero in the feature map, after convolution, the output of the operation is zero.
For input values above zero, the output equals the input.
So this introduces a simple nonlinearity around zero.
Other ‘activation functions’ can be used to map input to output, but this is the most common in deep networks.
It is so common that the operation implementing the activation function in deep networks is typically just called ReLU, or a rectified linear unit.

The pooling operation

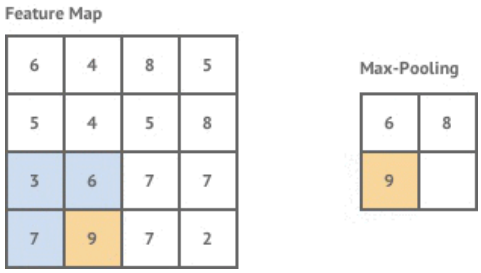
Operations in Linear-Nonlinear Layer



23

As a result of the filter operation, the response of each unit depends on several neighbouring inputs. So the units after filtering respond to a certain area of the input image, and the activation of neighbouring units will often be similar.
After several filter steps, each integrating inputs over an area, each unit will respond very similarly to an extensive area of the input.
So neighbouring units are representing very similar information.
The pooling operation therefore downsamples the units to improve computational efficiency.

The pooling operation

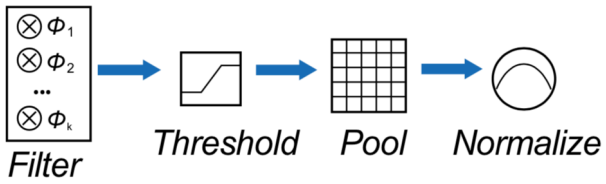


24

This pooling operation is typically a simple ‘max’ operation, taking the maximum of a square of 2x2 neighbouring units of the feature map in this example.
So the pooling operation discards some data in favour of computational efficiency.
As computers and deep network implementations become faster and more efficient, it should be less necessary to have pooling layers. This would generally improve network performance but reduce speed.

The normalisation operation

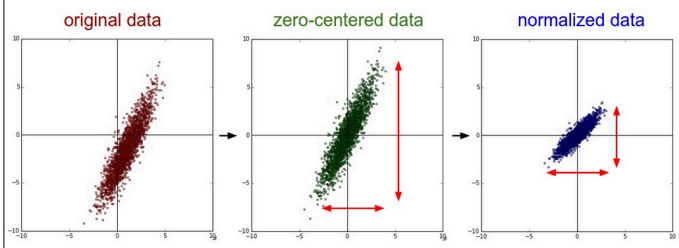
Operations in Linear-Nonlinear Layer



25

The threshold and pool operations use max functions.
As a result, even if the convolution filter has a mean of zero, by the pool stage we have a mean activation above zero.
Furthermore, the range range of activations levels can be very different between feature maps, effectively weighting some feature maps to contribute more to the result than others.
Subsequent layers will have a problem here because subsequent filtering steps operate across multiple feature maps that will then contribute different amount to the output.

The normalisation operation



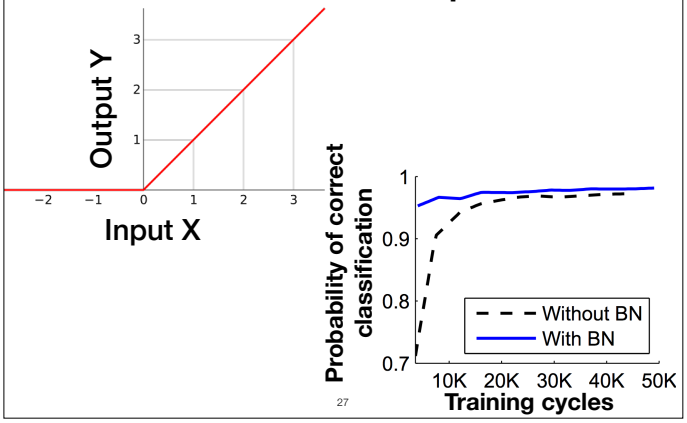
tmp = input-mean(input) then output=tmp/std(input)

26

So the normalisation operation linearly scales the data to have a mean of zero activation for each feature map's responses to all images.
The first step of normalisation is the subtract the mean response of each feature map from all responses, i.e. to zero-center the data.
The next step is to divide the activation of each layer by its standard deviation.
This makes the normalised data have a mean of zero and a standard deviation of one.

Normalisation is necessary for both theoretical and practical reasons
First, machine learning generally assumes that data reflects measurements of independent and identically-distributed (IID) variables.
Normalisation forces identical distributions.

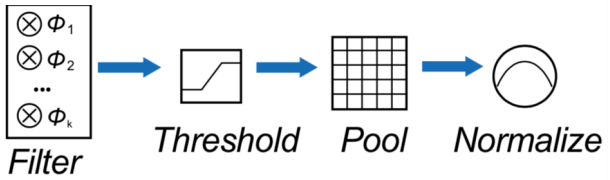
The normalisation operation



27

Second, if the threshold activation function depends whether the unit's response is above or below zero, having zero-mean inputs (after normalisation) and zero-mean filters, about half of the units will be active and half inactive. This even split of activation is a very efficient way to store information in a network of limited size.

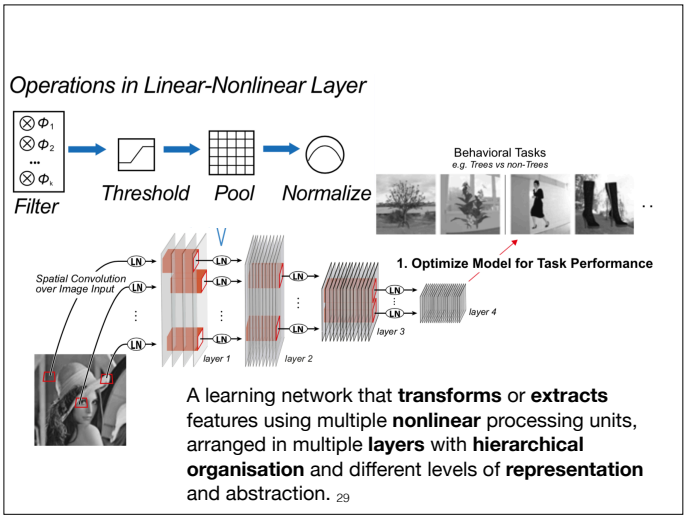
Third, having the same range for all feature maps and layers means the same threshold (zero) in the threshold function can be used throughout the network.
As a result of these consideration and other technical considerations, network performance increases far more quickly with normalisation, and final classification accuracy by the network is also higher.



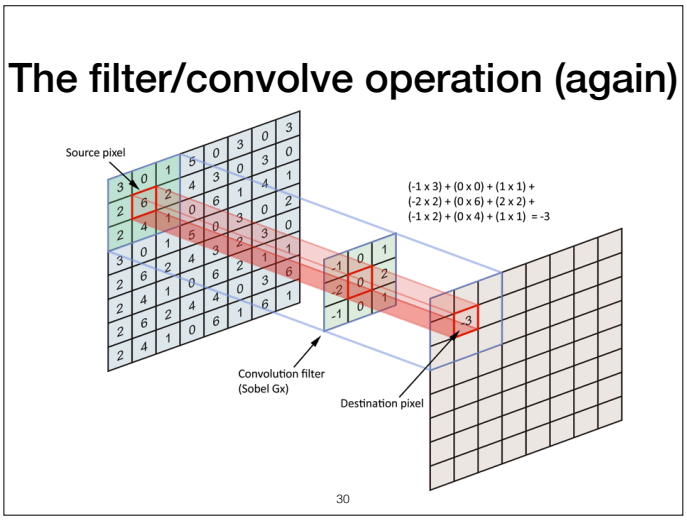
- Filter/convolve: determine how well each group of nearby pixels matches each of a group of filters
- Threshold/rectify: introduce a nonlinearity by setting negative activations of units to zero
- Pool: Downsample the units to improve computational efficiency
- Normalise: Rescale responses of each feature map to have mean zero and standard deviation one, so each feature map contributes similarly to classification

28

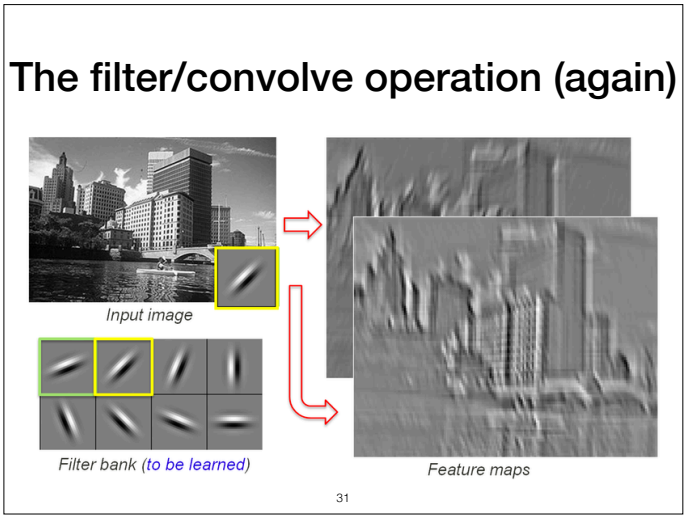
So, we have now done one layer of deep network operations on an image. Let's summarise.



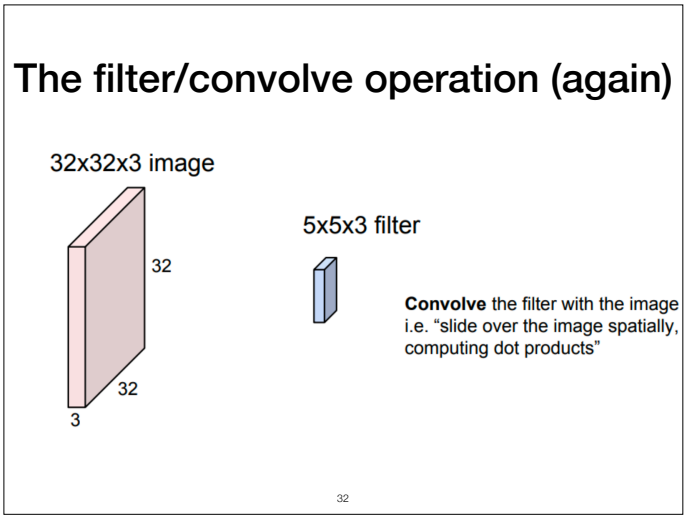
In a deep network, there are several layers performing similar operations and transformations of features, which all follow the same principles. Subsequent layers generally use the same operations in the same way, but the filter/ convolve operation differs between the first layer and subsequent layers.



In the first layer, we are convolving a 2-dimensional input image with a two-dimensional filter to give a 2-dimensional feature map.

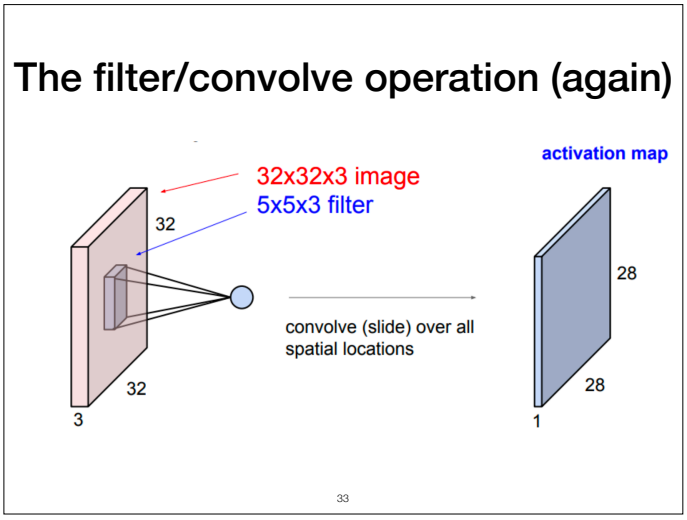


But we do this for several filters, so the next layer is three-dimensional, with the third dimension corresponding to the number of filters used.

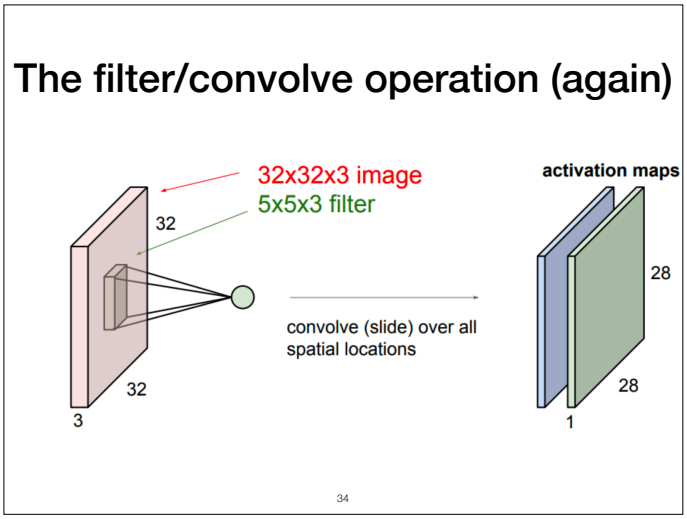


So in subsequent filter operations, the previous layer's activation and the filter are three-dimensional.
This is even true for some input images: colour images are also treated as three dimensional, with the third dimension being the three colour channels.
So for colour images, the filters used are also three dimensional, with activation then requiring specific colours or different spatial responses in each colour.
The filter will span ALL the feature maps or colour channels in its input.

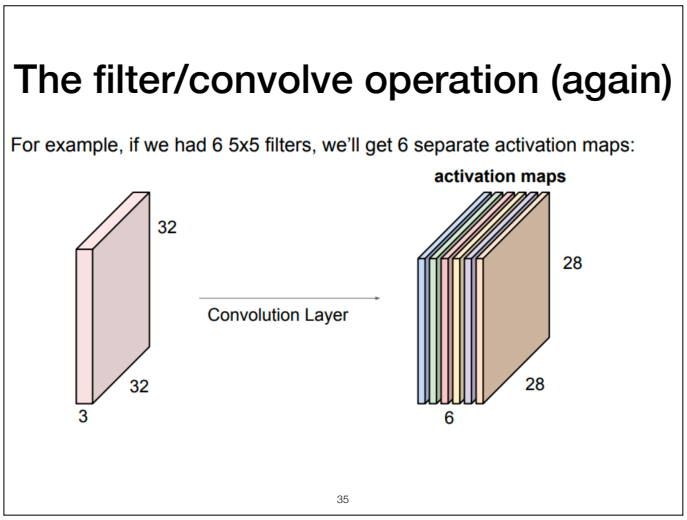
The filter's weights in each of these channels are independent. So this would allow a filter that only responds to red edges (spatial structure in the red channel only), only to green edges (structure in the red channel only), or to edges of any colour (the same spatial structure in all three channels).



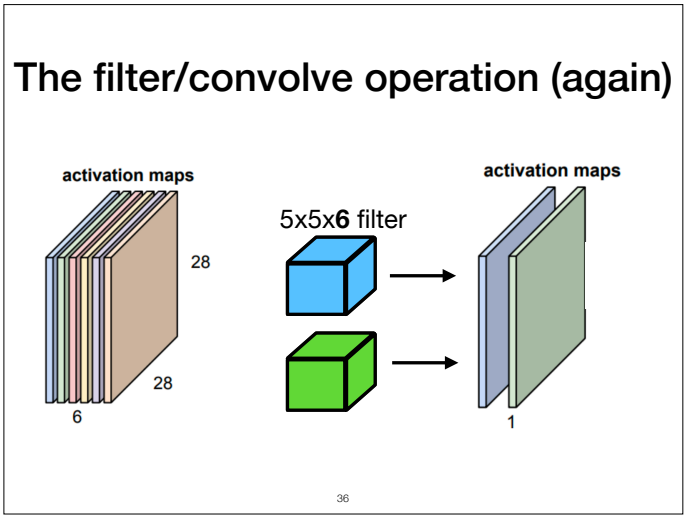
However, the result of a sum of the products of the filter and the corresponding part of the input is a single number regardless of the size or dimensionality of the inputs to the multiplication. So, even with 3-dimensional inputs and filters, this sum gives the response at a single point in a single feature map of the next layer. Convolved over every position in the input, we then produce a 2-dimensional feature map showing the match between that 3-d filter and the 3-d input at every position.



We repeat this with several 3-dimensional filters acting on the same 3-dimensional input to give several 2-dimensional features maps.

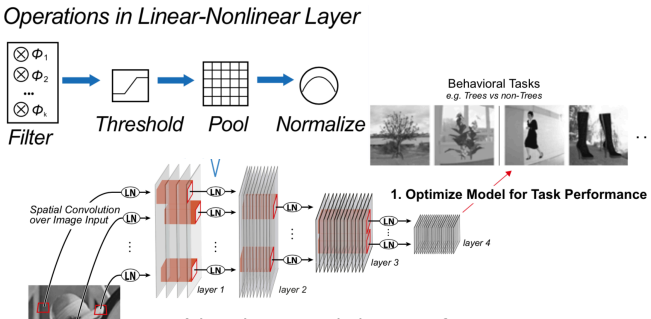


We repeat this with several 3-dimensional filters acting on the same 3-dimensional input to give several 2-dimensional features maps.



The resulting stack of feature maps is then the input of the next layer. Any filters used in the next layer will operate over ALL of these feature maps, 6 maps in this example. Again, this one filter will produce a single 2-dimensional feature map. Multiple filters will again produce a stack of 2-dimensional feature maps for the next layer.

So we can see the black and white input image that we looked at earlier as a special case of the inputs to all layers: a black and white input image is like a single feature map, so the filters only need to be a single pixel deep to operate over all feature maps (i.e. one).



A learning network that **transforms** or **extracts** features using multiple **nonlinear** processing units, arranged in multiple **layers** with **hierarchical organisation** and different levels of **representation** and abstraction. 37

As we get higher up the network, these filters get harder to understand in two important ways. First, the filter shape crosses multiple independent feature maps. An edge detector applied to an image is easy enough to conceptualise: We can make images of the 2-dimensional image and the 2-dimensional filter.
But such a 3-dimensional filter crossing multiple feature maps is harder to conceptualise.

Second, the input feature maps become more abstract. It gets very hard to conceptualise what feature is represented.
So both the filter and the feature map become a pattern within a pattern within a pattern. It's very hard to conceptualise these abstract, higher order patterns. Conveniently, we don't need to: the computer does this for us.

Inverting an object recognition DCNN

38

But it is important to understand that deep networks have increasingly complex representations of objects in the images, over several network layers
To get a feel for what features are represented in different at different levels, we can modify the input image to find a version that produces the strongest possible response at each layer.
After training to classify the objects in natural images, successive layers respond best to increasingly complex relationships between pixels in the input image. These relationships follow the local correlations and patterns found in natural images

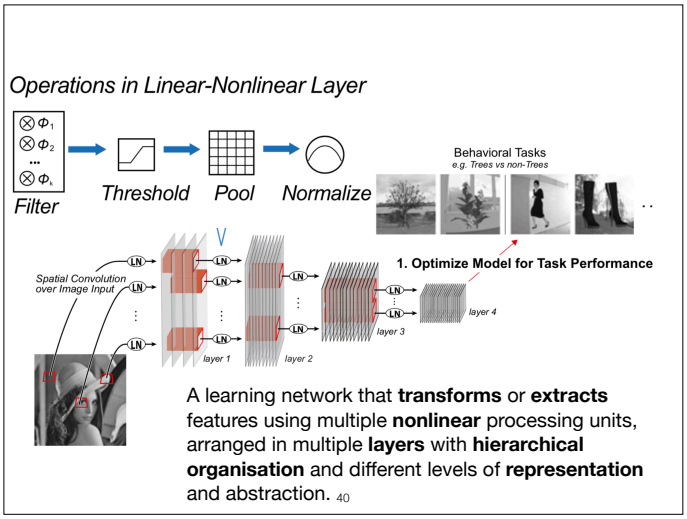
---

Shared weights

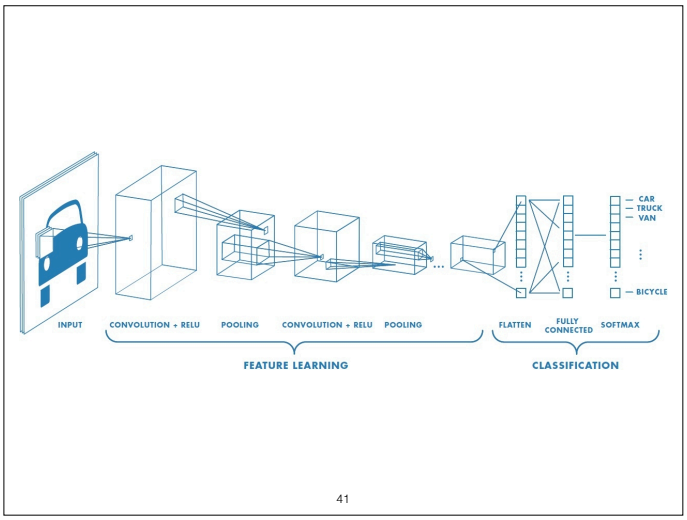
- Filters generally have a single set of weights for all positions in the feature map because:
  - If a feature is useful to compute at one position, it is probably also useful at another position.
  - The filter values are weights that need to be learned. Using one filter across all positions greatly reduces the number of weights, improving computational efficiency.
  - The convolution operation is a very fast matrix function. If filters are not fixed, the convolution operation cannot be used.

39

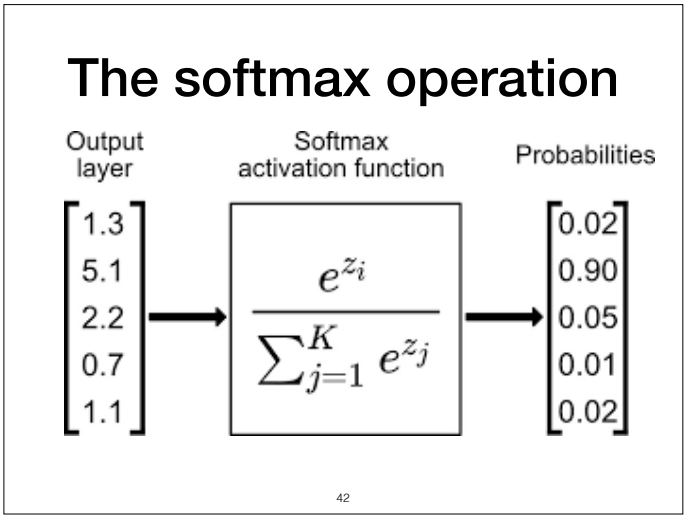
One filter is convolved with the previous feature map stack to give one new feature map.
It would also be possible for the filter to change at different positions in the feature map.
However, in artificial deep networks, generally a single filter is used across all positions, for three very good reasons.
AT END: We will see in our next class only the first of these constraints applies in biological deep networks.



As we get higher in the network, more spatial integration occurs as a result of repeated convolution and pooling, so the spatial dimensions of the feature maps shrink.
At the same time, the number of interesting feature combinations increases, so the feature maps become increasingly narrow, but stacked increasingly high.
In the end, some classification or decision must be made, which is the fundamental goal of the network.

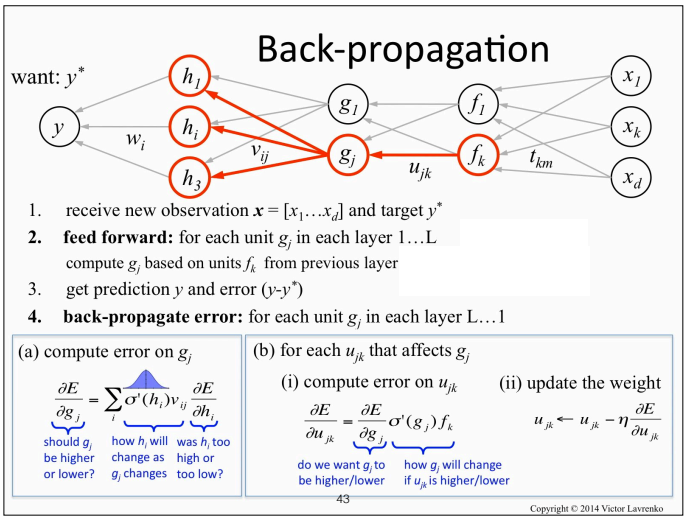


The last stage of the network, after several convolution layers, uses the activity pattern after the final convolutional layer for classification.
To compare this activity pattern with previously-seen patterns, the spatial relationships are first discarded, 'flattening' the last feature map into a line of independent units.
Each of these is connected to units that represent the possible candidate classifications, the labels that describe the input image.
Here we use a fully-connected layer to link every unit to every possible classification with different learned weights.
Essentially the labels see which top-layer pattern they were trained on resembles the current top-layer response pattern most closely.



So, the weights through our network will transform each input image into some 'score', reflecting the match between the top layer's pattern of activation and the pattern of activation by previous examples of each category.
This SCORE must then be converted to a PROBABILITY that this input image falls into each category.
This should take into account not just the score for one category, but also the scores for all other categories: the relative scores determine the probabilities.
This is almost always done with the normalised exponential function, or 'softmax'.
That is, a constant e raised to the power of the score, divided by the sum of this exponent over all classification scores. As a result, the probabilities to sum up to one.

The math is not particularly important to know, but note that, following an exponential function, an output layer score that is only slightly higher than another leads to a probability that is MUCH higher.



So far, we have carefully ignored what filters are used in higher levels.
It is straightforward for the human researcher to design a simple filter, like an edge detector, to operate on an early layer, like the input image.
But when we get to more abstract filters operating over multiple feature maps whose responses are already hard to conceptualise, the optimal convolutional filter structures must be learned by experience.
Here, the nodes are pixels in a feature map, and the connections between these are filters.
So, to learn the optimal filter structure, the network is optimising the WEIGHTS of all these connections to improve task performance based on previous outcomes.

As we have seen, the filter is a limited group of connections, so the learning is much simpler than in a fully-connected network.
But just like learning in a fully-connected neural net, this uses back-propagation of error, a mathematically complex process that is notoriously difficult to understand and explain.
I will simplify as much as possible. Again, for this class, the

---

Deep learning in artificial neural networks

- Useful for achieving tasks that are difficult to describe formally
  - Difficult for computers, intuitive for humans
- A form of machine learning performing multiple sequential nonlinear feature transformations in hierarchical layers
- Between each feature map layer, a few simple operations
  - Convolution checks each position's match with a specific filter kernel
  - Thresholding introduces a nonlinearity
  - Pooling downsamples the image, taking the maximum
  - Normalisation rescales responses so each feature map contributes similarly
- Final fully-connected layer links pattern of most abstracted, top-level features to most likely classification
- Softmax determines probability of each classification
- Match or conflict of expected and actual classifications used as the basis for backpropagation of error (complex mathematical process)
  - Adjusts filter structure: link between layers, machine learning target