Machine Learning for Human Vision and Language

Lecture 1:
Principles of deep learning in artificial networks

Ben Harvey

Welcome to Advanced Topics in Cognitive Science.
Many of you will remember me from Chris Janssen's classes on Cognitive Modelling

First, let's go through an outline of the course so you know what to expect and what we expect from you.

Course goals

- Explore the relationship between cognitive science and AI
- Focus on deep learning in artificial machine learning networks and comparison to biological systems
  - Which biological processes do deep networks imitate?
  - What is missing in artificial networks?
  - What might make AI/machine learning more like biological intelligence/learning
- Become familiar with the use of AI in cognitive science research
- Build some deep learning networks to do human-like tasks

Some of this lecture content is complex, and you have very different backgrounds.
But it will be examined, and you must pass this exam to pass the course.
So we will go slowly, start from first principles and avoid assuming any knowledge.
Please ask questions when you don't understand: the explanations and discussion that result are very valuable to the class format.

Assessment

- In pairs, students will complete two lab assignments to build simple deep learning systems to solve computer vision and linguistics tasks. Students will be graded on two written reports of their work. Grades depend on depth and completion (30% each)
- Students' understanding of lectures and reading assignments will be assessed in a final exam that determines 40% of the final grade. You are required to pass this exam to pass the course.

Date	Time	Format	Teacher	Room
06/09	13:15-15:00 15:00-17:00	Lecture 1 Lab 1 intro & start	Harvey Heeman	BBG 209
11/09	13:15-15:00	Lecture 2	Harvey	Bestuurs Van Lier
13/09	13:15-17:00	Lab 1	Heeman	BBG 209
18/09	13:15-15:00	Lecture 3	Harvey	Bestuurs Van Lier
20/09	13:15-17:00	Lab 1	Heeman	BBG 209
25/09	13:15-15:00	Lecture 4	Harvey	Bestuurs Van Lier
27/09	13:15-17:00	Lab 1	None (strike)	BBG 209
29/09	23:59	Deadline Lab 1		

Jessica Heeman

4

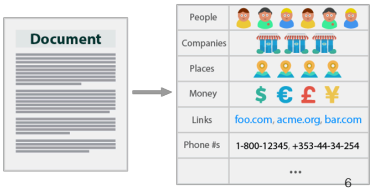
Here you can see the schedule for the first few weeks.
You have classes on Fridays where you can get help with your lab assignments, while Wednesday classes will be lectures from me.
There are four classes for help with the labs, but the first half of today's class is a lecture, and there is a strike for climate on the date of the last class. So on September 27th we cannot confirm that you will have help available, and we expect many of you will be on strike anyway.
This leave you with 10 classroom hours when you can get help with the assignment. We expect you to work on this between classes too.

Why deep learning?

- AI has made great advances in tasks that are:
  - Described by formal mathematical rules
  - Relatively simple for computers
  - Difficult for humans
- AI had been less effective 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

5

Deep learning tasks



6

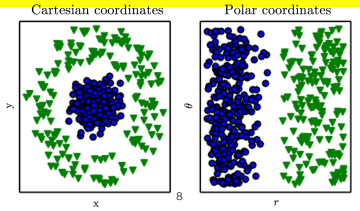
Deep learning approach

- Learn from experience (machine learning)
  - No formal rules of transformations
  - No ‘knowledge base’
  - No logical inference
- 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 exactly what a human is doing when learning about the world
SO the main inspiration used is the brain

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

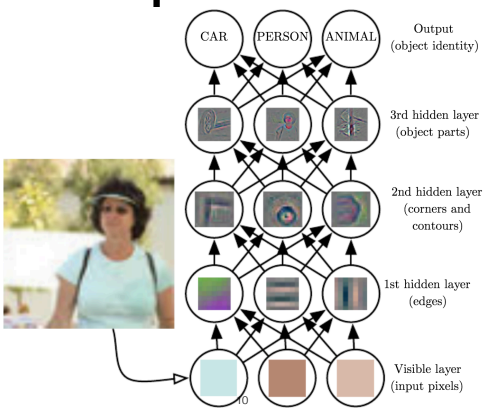


A simple task like this can be solved by choosing the right set of features, with minimal learning necessary.
However, for many tasks, it is hard to know which features or formats of the input are important in determining the output.
And these may be high-level features that need to be extracted first.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

Representations in deep networks

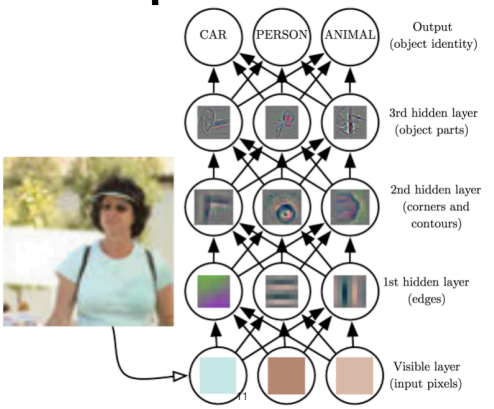
- Useful features may need to be transformed or extracted first
- So deep networks have multiple representations
  - Each is built from an earlier representation
- This can:
  - Transform features to a different format before learning their links to the output
  - Extract complex features from simpler 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

Representations in deep networks



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 based on their content without a human labelling all this content.

Representations in deep networks



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.
I deal with this in two ways. First, my online slides also contain notes with a fairly complete description of what I say.
Second, I record these lectures and also put these movies on Blackboard.
But please note that your other lecturers won't be doing this.

What is a deep network?

- A 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

Note that this definition does not specify 'machine' learning.
In this course, we will also look at biological neural networks like the brain, which are also deep networks

Lectures 1-4

- Lecture 1: Principles of deep learning in artificial networks
- Lecture 2: Deep learning in biological neurons and networks
- Lecture 3: Early visual processing
- Lecture 4: Higher visual processing, and what's missing

13

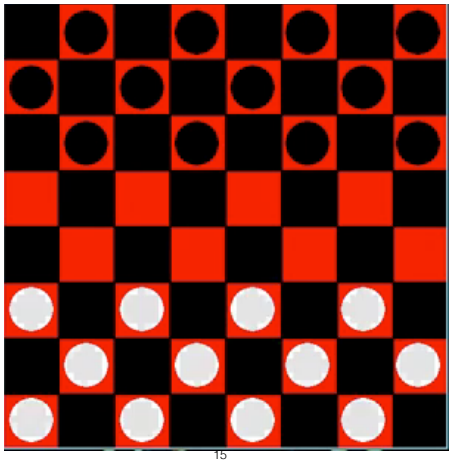
So, we're going to start by looking at artificial networks, a machine learning system.
We will compare these to the human brain's biological neural networks these artificial networks aim to imitate.
In my last two lectures, we will at the stages of visual processing involved in object recognition in both systems.
This will show what is missing in current artificial networks and how researchers are starting to build deep networks that fill these gaps.
To begin, let's look at what machine learning is, starting artificial networks from the start.

What is machine learning?

- The field of science that 'gives computers the ability to learn without being explicitly programmed' (Arthur Samuel, 1959)
-

14

So this field has been around for a long time Samuel showed a machine learning algorithm many games of checkers (draughts) to learn which board positions are likely to lead to a win. The only programmed rule was the way the pieces are allowed to move.



15

Here we see the black player (a machine learning program) consistently beating the white player (which moves randomly)

What is machine learning?

- The field of science that 'gives computers the ability to learn without being explicitly programmed' (Arthur Samuel, 1959)
- 'A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.' (Tom Mitchell, 1998)
  - Expressed in organisational terms, not cognitive terms

But Samuel's definition of machine learning uses the word 'learn' without any definitions
It also uses 'computer', which is a synonym of 'machine'
Mitchell provided a more formal definition
By describing the fundamental operation in terms of inputs and outputs, this definition avoids suggesting the machine can think.
Alan Turing: "Can machines think?"->"Can machines do what we (as thinking entities) can do?"

Learning to filter spam

- 'A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.' (Tom Mitchell, 1998)
- **Task:** 'Classify which emails are wanted (not spam) vs unwanted (spam)'
- **Experience:** Watching humans labels emails (training set)
- **Performance:** The proportion of new emails (test set) classified correctly

Note that this is a relatively simple machine learning example, and can be done without deep learning.
So, now that we have an idea of what machine learning is, how does machine deep learning work?

What is a deep network?

- A 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

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 Perception

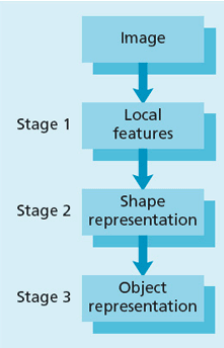
Why is it so difficult?

19

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.
The result is achieved using an 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.
For example, you might think that pointed ears tell you this is a cat, but some examples don't have them while clearly being recognised as cats.
So we can't recognise an object directly from its impression on the eye or camera sensor

---

The 20th century view of object recognition

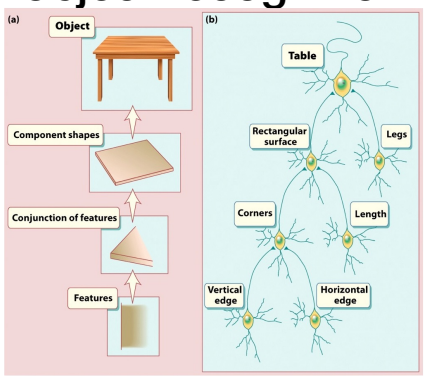


- Stage 1 builds a piecemeal representation of local image properties.
- Stage 2 builds a representation of larger-scale shapes and surfaces.
- Stage 3 matches shapes and surfaces with stored object representations-recognition.

20

Note this is already a multi-layer, hierarchical approach

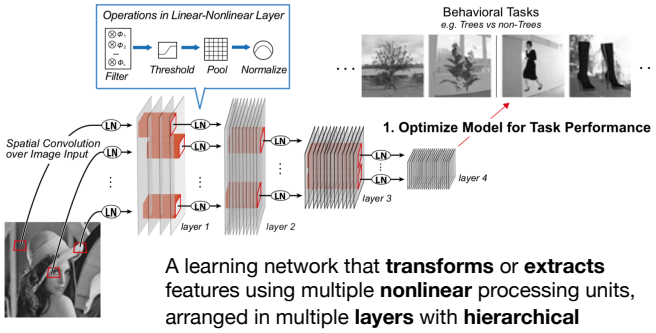
The 20th century view of object recognition



21

So we might think that our object representations are built from combinations of feature representations
Indeed, many objects are built from parts, so simplified parts that we recognise from all angles might let us build a 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.

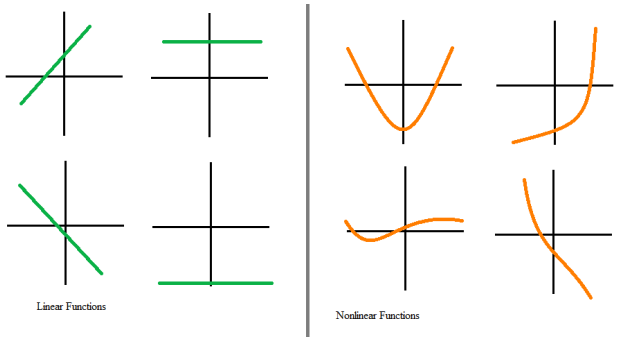
A deep network for object recognition



A 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.
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.
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



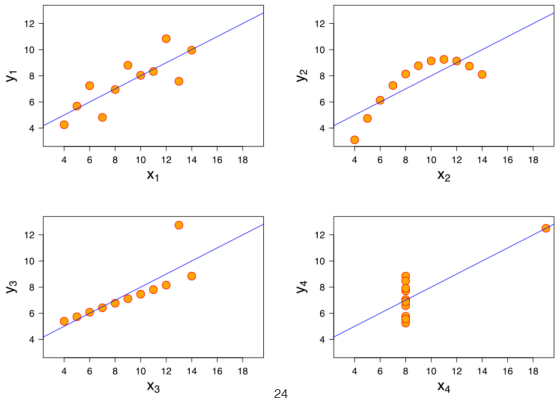
Y=A*X+B

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.

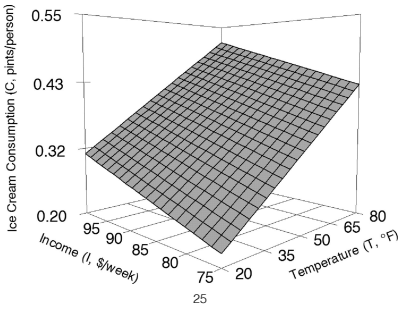
Why nonlinear functions?



Linear functions are therefore very limiting.
These fits to these four different data sets follow the same linear function.
The data also have very similar means, standard deviations and correlation coefficients, which are all linear operations.
However, the data sets clearly have very different relationships between X and Y.
So often a linear fit to an input is too simple to capture the relationship between input and output, except maybe in the first image.

Why nonlinear functions?

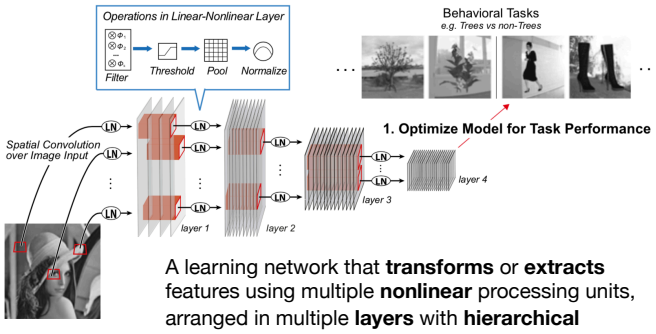
$$Y = B + A_1*X_1 + A_2*X_2 + \ldots + A_p*X_p$$



25

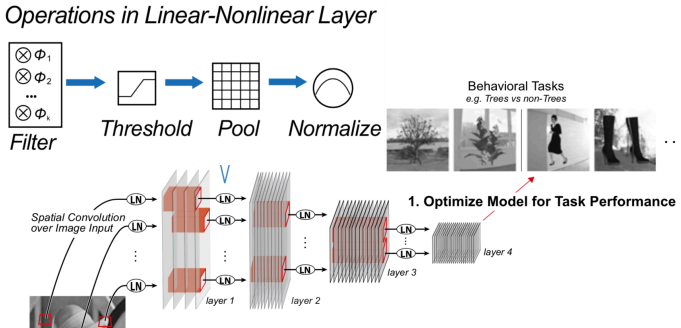
But in the context of deep networks, there is a more important problem with linear functions.
The many layers of a deep network repeatedly perform functions on the output of each stage, joining these functions together to add effects of multiple features of the input.
This becomes much harder to visualise, but this example, using only two input, gives an idea of the effect.
Joining multiple linear functions (by addition or multiplication) always results in a linear function. That function can have multiple inputs, but the output is always a linear function of those inputs.

A deep network for object recognition



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. 26

In this network, the inputs are the lightness 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.

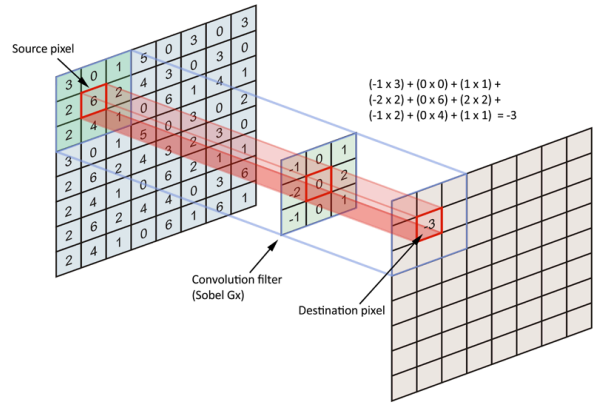


A learning network that **transforms** or **extracts** features using multiple **nonlinear** processing units, arranged in multiple **layers** with **hierarchical organisation** and different levels of **representation** and abstraction. 27

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. This is true whether the operations are in the same layer or different layers.
There is no step in this sequence of operations that has any special status, so the repeating sequence of these four operations effectively forms the layer. In some descriptions, the result of each operation is called a 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, pool and normalise functions are all linear.

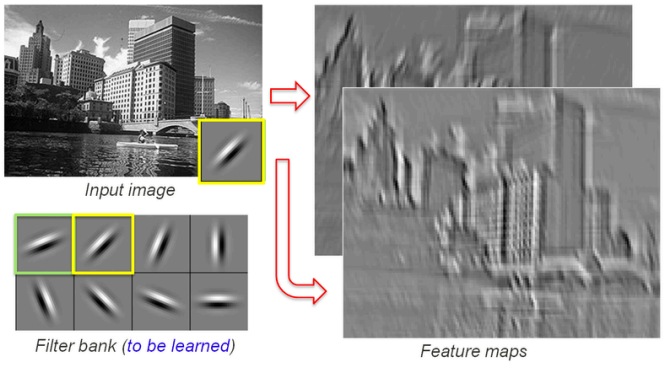
The filter/convolve operation



28

The filter or convolve operation is perhaps the most computationally important.
At the first processing layer, the input (or source) is a pixel map, a bitmap of the image giving the brightness of each pixel. In the simples case, a grayscale image is used.
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).
Using matrix multiplication, this filter is multiplied by a group of input pixels with a particular position, giving the match between the filter and a small part of the input image.
If the source pixels follow this filter pattern (light on the right, dark on the left), 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 (light on the right) 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. A matrix multiplication function called 'convolution' does this efficiently, so this is often called a convolve/convolution step, and the term deep convolutional network.

The filter/convolve operation

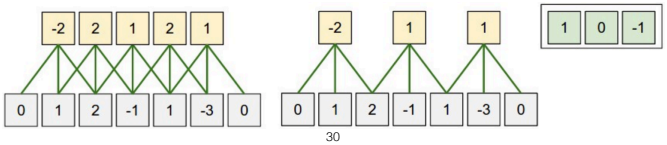


29

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 the filter pattern is seen, often called feature maps.
Each 'pixel' shown in each of these feature maps is no longer a pixel in the image, it is an abstraction of the pattern across a group of pixels.
The pixel in the feature map represents the activity of a processing unit or artificial 'neuron'
Here we see a set of eight filters with larger extents and a range of different orientations.
These have been set manually in the first image analysis layer, because we know edge detectors with different orientations are an important early stage of computer and human vision systems.
The relevant filters can also be determined by machine learning, particularly for later layers where the best filter choice is not so obvious. We'll look at that possibility later.
The size of these filters will also determine the spatial scale of the feature we can detect. This is normally set manually using a small value for computational efficiency. Higher layers can detect larger scale features.

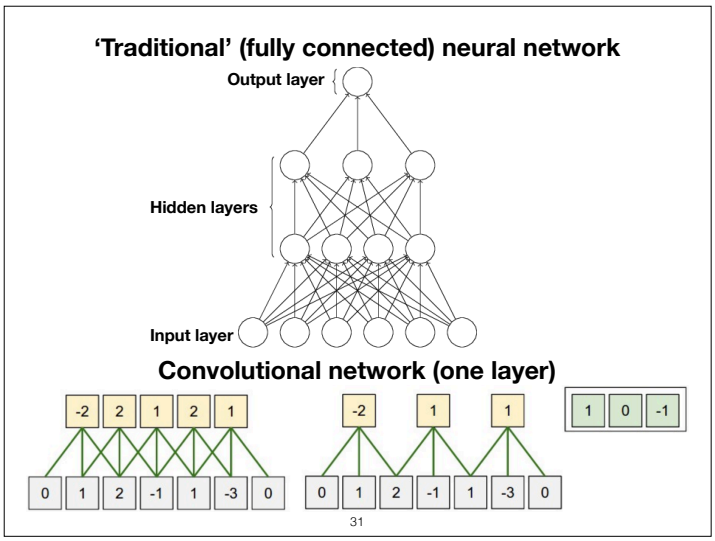
Parameters of the filter/convolve operation

- Depth: the number of filters
- Input image size (W)
- Filter size (F): the spatial scale of the features to detect
- Stride (S): how many pixels the filter moves (downsampling)
- Zero-padding (P): how many zeros are added around the input (often P=(F-1)/2)
- Per filter, number of processing units = output image size = (W-F+2P)/S+1



30

So there are some variable that determine how this step behaves.
The filter does not necessarily need to process every pixel position: we can downsample the image in this step by moving two or three input pixels for every output pixel
Zero-padding ensures that pixels at the edge get processed as a filter centre too. The zeros around the edge will not contribute to the filter output: anything multiplied by zero gives zero.
Notice that every part of this step is linear.
DESCRIBE FILTER PARAMETERS
GO THROUGH FILTER CALCULATIONS



So, how is a convolutional network different from other types of neural network?
We just saw a 1-dimensional convolutional network. Here, the same filter is used at all positions in the network. Activation of each upper layer node depends on the product of this same filter convolved with a spatially limited set of lower layer nodes.
In a traditional neural net, each node in the upper layer is connected to all nodes in the previous layer, with no constraint on the spatial spread of links between them, and also no constraint on the pattern of links.
In a convolutional network, different patterns of filters are represented in different feature maps, but in a traditional neural network any 'filter' can effective emerge for any node, so there is no need for multiple feature maps.
The constraint on the spatial spread of links between layers is particularly appropriate where the input layer nodes have a meaningful spatial relationship between them.
The most obvious examples of meaningful spatial relationships are in image processing, where the input layer units are image pixels and the feature maps are each an image of the spatial distribution of the features described by the filters.
We'll look at some other cases shortly.
In the image example, the layers and filters are 2-dimensional because the input image is 2-dimensional. Here we are looking at 1-dimensional examples.
But in a fully connected network, every unit in one layer is connected to every unit in the next. Because of this, spatial relationship do not effect the network's activity.
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.

Dimensionality of DCNNs

- 1-Dimensional DCNNs: Temporal sequences
  - e.g. Language
  - (recurrent networks)
- 2-Dimensional DCNNs: Image processing
  - e.g. Object recognition
  - e.g. Autonomous vehicles (2D sensors)
- 3-Dimensional DCNNs: 3D image processing
  - e.g. Medical image analysis
- 3-Dimensional DCNNs: Time + 2D image processing
  - e.g. Video analysis
  - (recurrent DCNNs)
- 4-Dimensional DCNNs: Time + 3D image processing
  - e.g. Time-resolved medical image analysis (functional neuroimaging)
  - (recurrent DCNNs)
- 4-Dimensional DCNNs: 4D image processing
  - We live in 3 spatial dimensions, but…

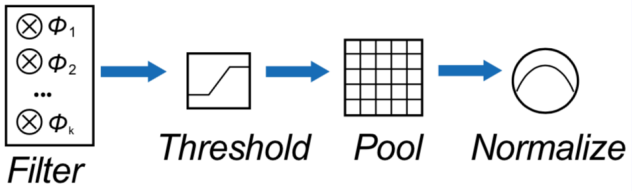
In DCNNs, the number of dimensions in the input determines the number of dimensions in the filter.
Different numbers of dimensions are suitable for different inputs and tasks.
The principles involved are the same regardless of dimensionality, so we will use examples for one and two dimension because they are easier to follow and easier to show you on a 2D screen.
One-dimensional convolutional networks are not appropriate for image inputs, but many types on input have meaningful relationships between neighbouring measurements, for example a series of measurements taken over time where some interesting pattern may emerge.

In a recurrent network, the activation of a unit depends on its past activation. We will come back to this later in the course because this type of network is particularly useful for language processing and other temporal sequences.
One way that a recurrent network differs from a DCNN with time as of its dimension is that in a recurrent network the filter goes in one direction only: it extends into the past, but not the future.
Within a sentence, it is often best for a filter to extend in both directions: the meaning of a word depends on what comes after as well as before. But between sentences, it is not common for future sentences to affect the interpretation of the current sentence, while past sentences do affect the interpretation of the current sentence.

But it's also possible that the input image is three dimensional, for example a medical image like an MRI. Here, features may extend over three spatial dimensions. So feature maps are also three dimensional, describing the distribution of the feature in 3D space.
There is no theoretical limit to the number of spatial dimensions, except that the world has only three spatial dimensions.
Where time is a dimension, it is often treated quite differently to spatial dimensions. For example a filter will generally not have the same extent in time as in spatial dimensions. Time and space are in different and independent units.
The useful temporal extent of a filter will depend on frame rate, while the spatial extent will depend on spatial resolution.

The threshold/rectification operation

Operations in Linear-Nonlinear Layer



Y = f(X) = max(0, X)

33

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.

Therefore, the threshold depends always on the mean of the filter

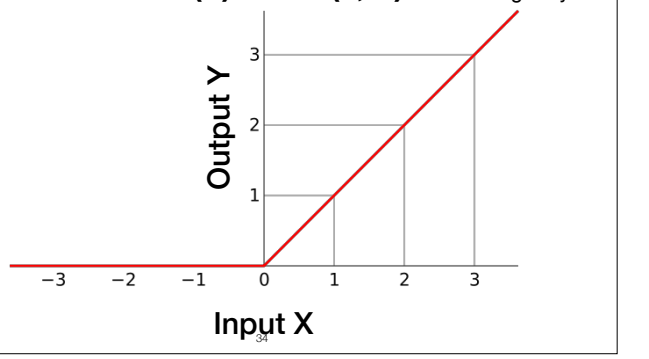
---

The threshold/rectification operation

an activation function using a rectified linear unit (ReLU)

Y = f(X) = max(0, X)

NO MAXIMUM!
Biologically there is



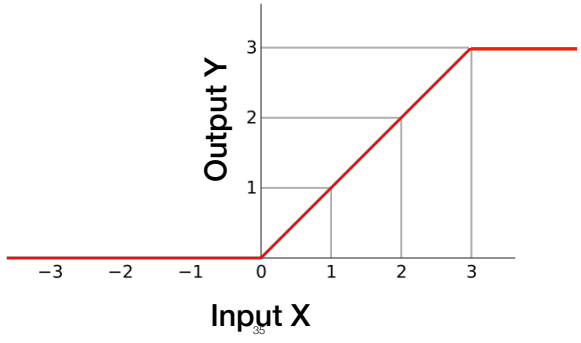
34

So, for input values (in the feature map) below zero, 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 unit implementing the activation function in deep networks is typically just called ReLU, or a rectified linear unit.
One issue with this function is that it has no maximum output, while a biological neuron does have a maximum firing rate.

The threshold/rectification operation

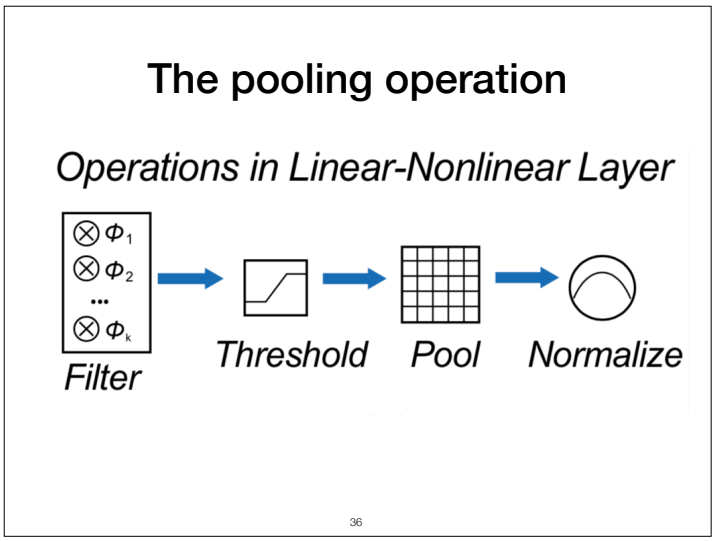
an activation function using a rectified linear unit (ReLU)

Y = f(X) = min(max(0, X), MaxActivation)

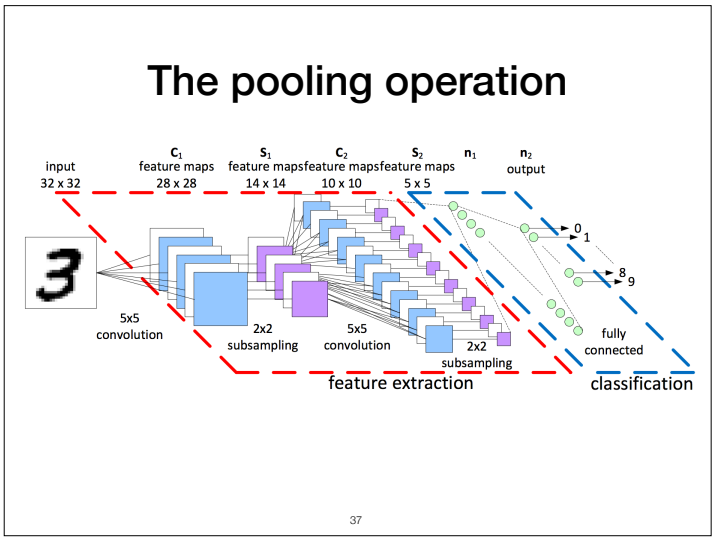


35

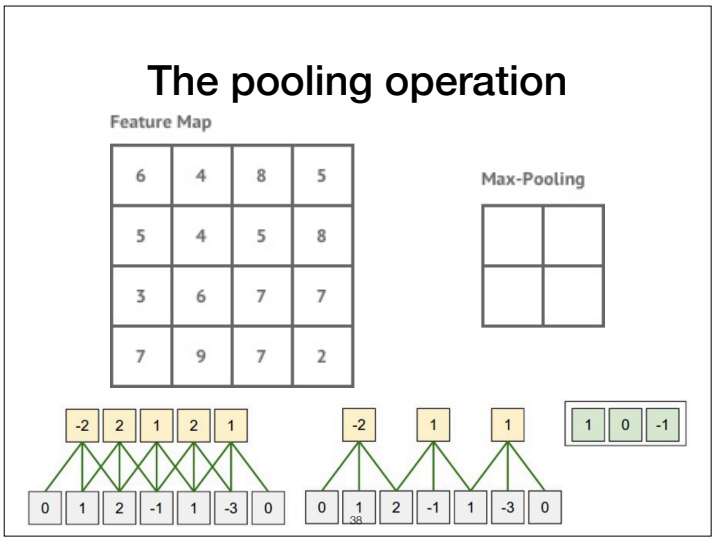
In practice, the maximum firing rate of a biological neuron is rarely reached, but a high maximum output is sometimes included, particularly is modelling a biological brain.



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.



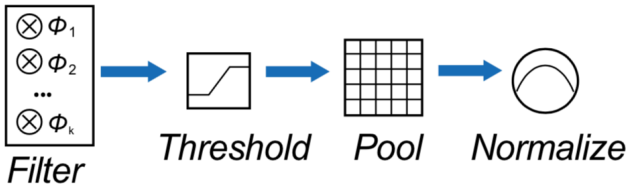
Furthermore, from one input image, we have gone to several feature maps (C1 here) at the filter/convolve operation.
The next filter/convolve operation will turn each of these into yet more feature maps.
So to avoid an explosion of computational load, it's very important to reduce the size of these maps.



This pooling operation is typically a simple max operation, taking the maximum of a square of 2x2 neighbouring units of the feature map.
This is very similar to the 'stride' parameter in the filter/convolve operation, so it is more efficient to increase the stride and skip the pooling.
BUT increasing stride may miss the maximum value, indeed both of the '2's are missed in this example.
The maximum value will be important for subsequent filtering.
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

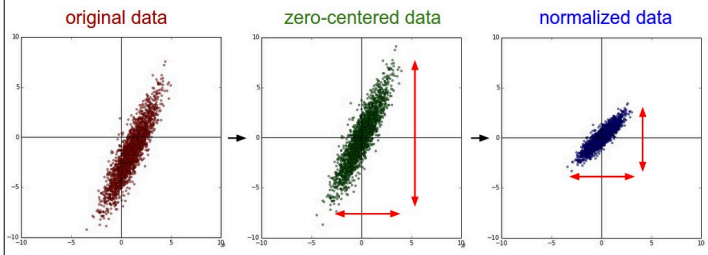
Operations in Linear-Nonlinear Layer



39

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 and an arbitrary range.
Furthermore, this range 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 may contribute very differently.

The normalisation operation

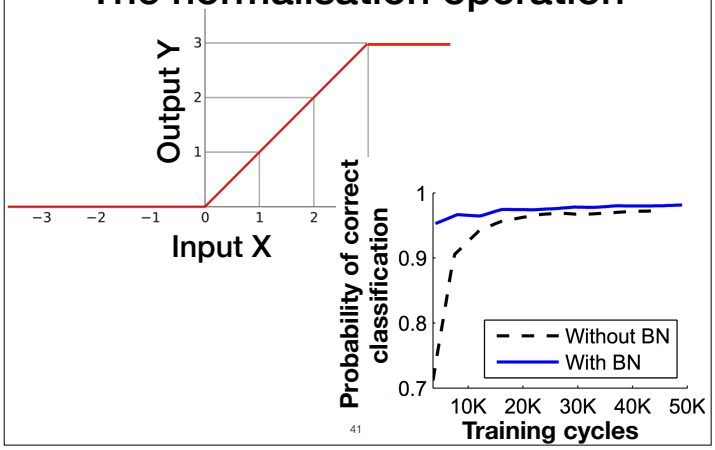


tmp = X-mean(X) then Y=tmp/std(tmp)

40

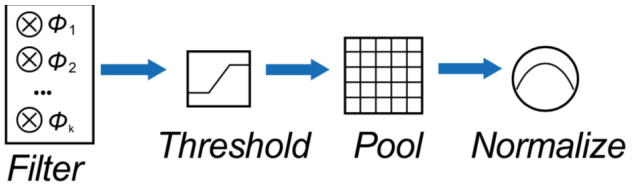
So the normalisation operation linearly scales the data to have a mean of zero activation for each feature map's responses to all images.
The first step of normalisation is the subtract the mean response of each feature map from all responses, i.e. to zero-center the data.
The next step is to divide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

The normalisation operation



41

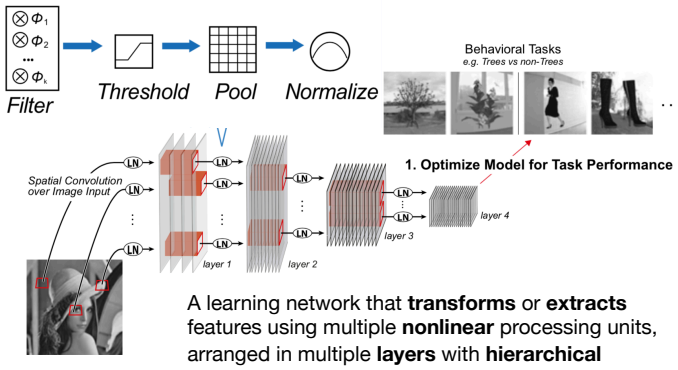
This is necessary for both theoretical and practical reasons
First, machine learning generally assumes that data reflects measurements of independent and identically-distributed (IID) variables. Normalisation forces identical distibutions.
Second, if the activation function depends whether the unit's response is above or below zero, having zero-mean inputs 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 maximum threshold in the activation function can be used throughout the network.
Fourth, as a result of these consideration and other technical considerations, training rates are far better after 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 (and maybe set a maximum activation)
- 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

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

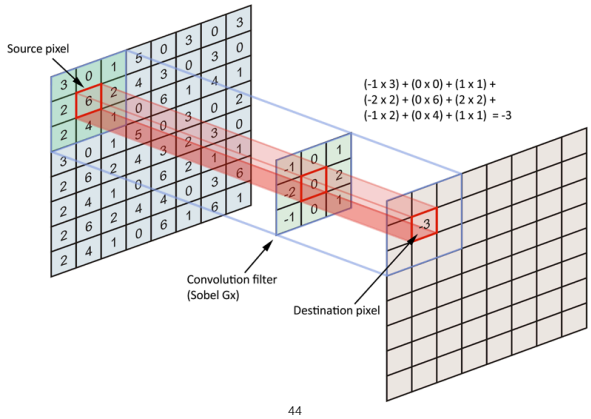
Operations in Linear-Nonlinear Layer



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.

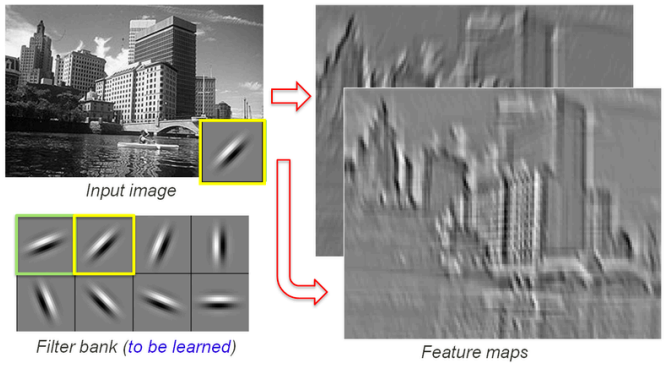
In a deep network, there are several layers performing similar operations and transformations of features, which all follow the same principles. Subsequent layers generally use the same operations in the same way, but the filter/convolve operation differs between the first layer and subsequent layers.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

The filter/convolve operation (again)



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

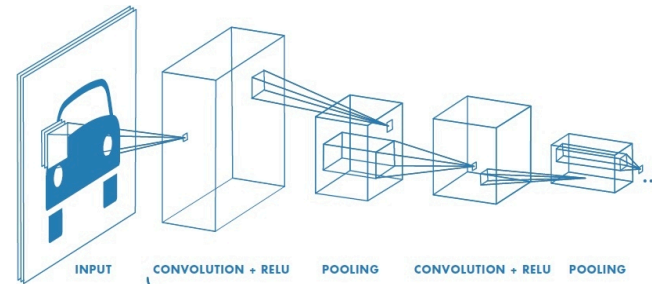
The filter/convolve operation (again)



45

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

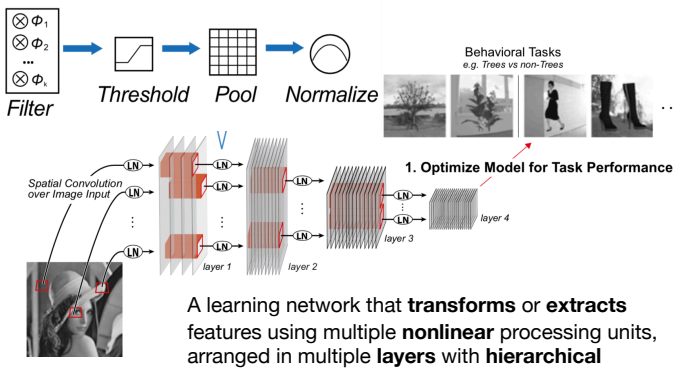
The filter/convolve operation (again)



46

So in subsequent filter operations, the filter is also three-dimensional. It will normally span ALL the feature maps in its input to give the response at a single point in a single feature map of the next layer. Remember that the result of a matrix multiplication can be a single number regardless of the size of the inputs to the multiplication.
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 potentially different spatial responses required from each colour.
So we can see the input image as a special case of the inputs to all layers. Any convolution step generally considers filter patterns that cross all of its input feature maps.

Operations in Linear-Nonlinear Layer



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. 47

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, but such a high-dimensional filter 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

http://thepsychreport.com/technology-2/googles-psychedelic-art-this-is-your-computer-brain-on-drugs/

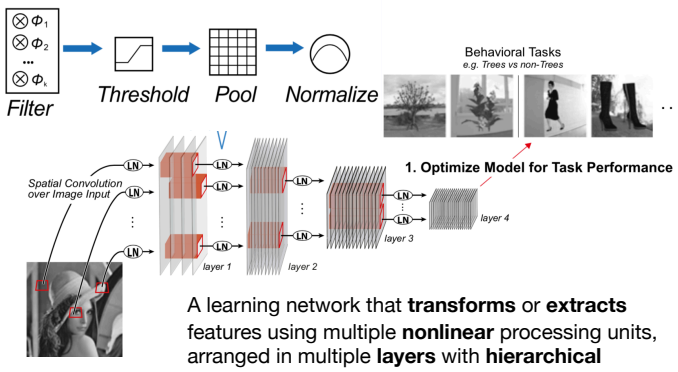
But it is important to understand that deep networks have increasingly complex representations of objects in the images, over several network layers
To get a feel for what features are represented in different at different levels, we can 'invert' the network, activating different feature maps and their connections back to the original points in the image.
This imposed the features that each layer detected onto the original image. Here they use the example image of George Seurat's Sunday on La Grande Jatte
Successive layers find increasingly complex relationships between points in the original image, following 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. It is very computationally demanding to do this if the set is too large.
  - The convolution operation is a very fast matrix function. If filters are not fixed, the convolution operation cannot be used

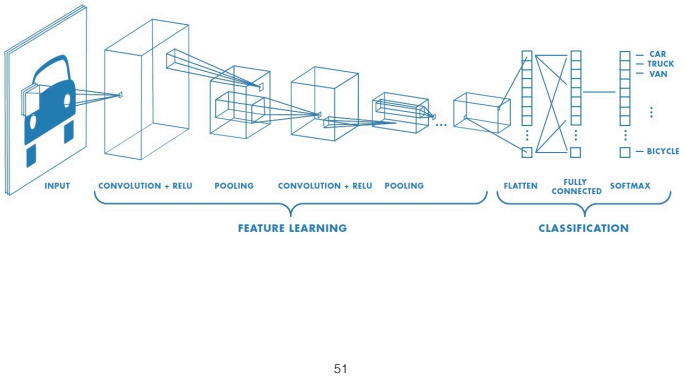
One filter is convolved with the previous feature map stack to give one new feature map.
It would also be possible for the filter to change at different positions in the feature map.
However, in artificial deep networks, generally a single filter is used across all positions, for three very good reasons.
AT END: We will see that these constraints do not apply in biological deep networks.

Operations in Linear-Nonlinear Layer



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.

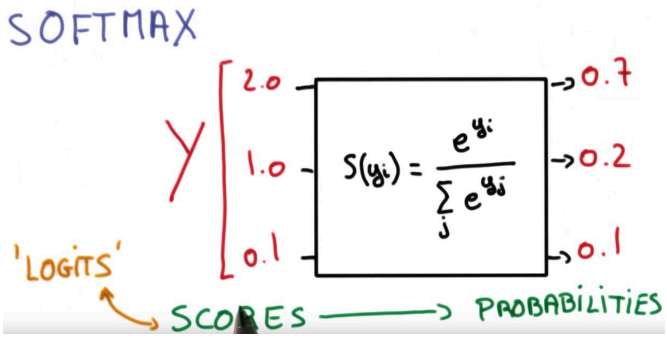
As we get higher in the network, more spatial integration occurs as a result of repeated convolution and pooling, so the spatial dimensions of the feature maps shrink.
At the same time, the number of interesting feature combinations increases, so the feature maps become increasingly narrow, but stacked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



51

The last stage of the network, after several convolution layers, uses the activity in the final convolutional layer for classification.
Here, the spatial relationships are first discarded, 'flattening' the last feature map into a line of independent units.
Each of these is connected to all the others, giving a fully-connected layer that looks at all links between every unit and all the others.
The activation pattern in these top-layer units is then associated with a particular output, a classification or label that describes the input image. Essentially the labels see which pattern they were trained on resembles the top-layer response pattern most closely.

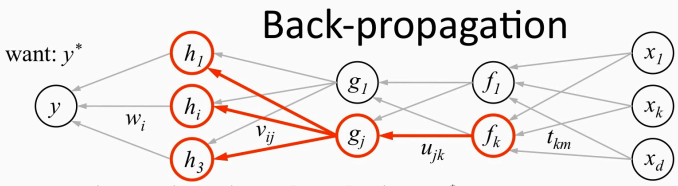
The softmax operation



52

So, the weights through our network will transform each input image into a 'score', reflecting the match between the top layer's 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 is almost always done with the 'softmax' function, or normalised exponential function.
That is, constant e (the natural log base) raised to the power of the score, divided by the sum of these exponents for all scores to normalise the probabilities to sum up to one. (The math is not particularly important to know)

Back-propagation



1. receive new observation $\boldsymbol{x} = [x_1 \ldots x_d]$ and target y^*
2. **feed forward:** for each unit g_j in each layer 1…L
   compute g_j based on units f_k from previous layer

53

Copyright © 2014 Victor Lavrenko

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, we can no longer design a filter.
Instead, the filter structures are targets for machine learning.
Indeed, the convolution filters are the main links between different layers of our network, so they effectively form the weights of connections between the nodes in a neural network.
Here, the nodes are pixels in a feature map, and the connections between these are filters.
So, to learn the weight of connections, the network learns the structure of the filters.
We constrain the filter's size, maybe 3x3 or 5x5 pixels spanning all feature maps in a layer.
But the filter's structure, the weights from the different pixels across the feature maps, is learned.
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 math is not as important as what it is achieving.

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 required response
- Softmax determines probability of desired response
- Match or conflict of expected and actual outputs used as the basis for backpropagation of error (complex mathematical process)
  - Adjusts filter structure: link between layers, machine learning target