Queen Mary School Hainan
Queen Mary University of London

QHM5703 Principles of Machine Learning
(Introduction to) Neural Networks and Deep Learning

Dr Nikesh Bajaj
Dr Jesús Requena Carrión

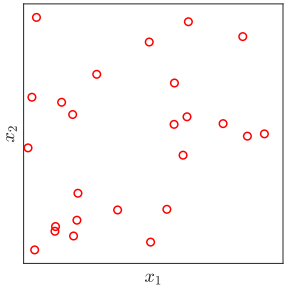
Week 15: 17/18 Dec 2025

Agenda

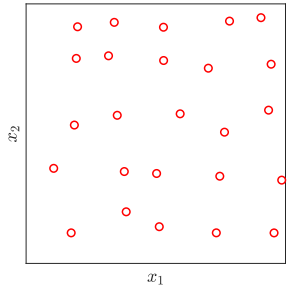
Patterns and structure

Patterns are regularities in our data and **structure** is a regularity in our target population. Machine learning project relies on discovering the underlying structure by identifying patterns in data.

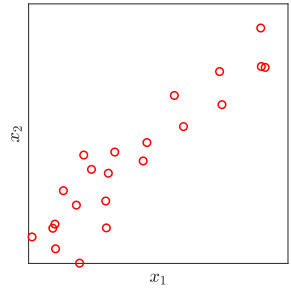
The main challenge is to distinguish **relevant** from **spurious patterns**.
Which of the patterns below suggests the **least underlying structure**?



(a)



(b)



(c)

Go to menti.com and use code: **DMT: 7318 3011** — **ICS: 5841 2762**

Short and wide datasets

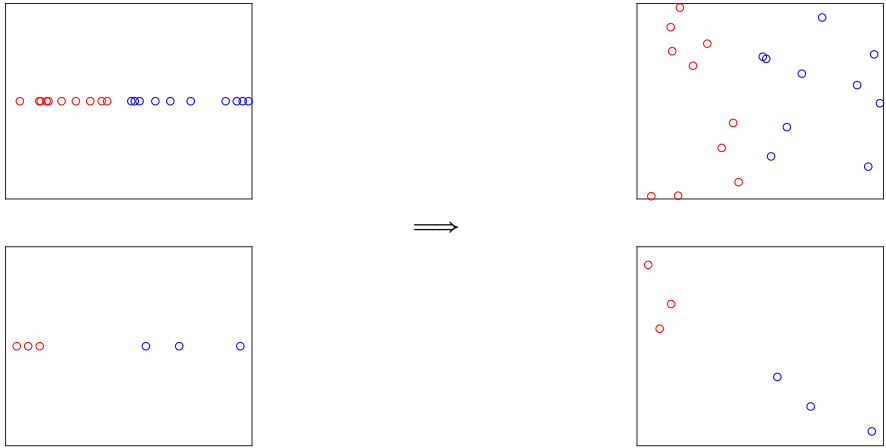
Using the dataset below, train a model that predicts the salary of an individual based on their age, height, weight, gender and postcode.

ID	Age	Height [m]	Weight [kg]	Gender	Postcode	**Salary**
0	30	1.68	75	M	EC2Y 8DS	100
1	40	1.78	70	F	E14 0QR	200
2	35	1.7	85	M	WC2H 8LH	150

Can you think of patterns in above dataset, to predict the salary from other attributes?

The Curse of Dimensionality

As the dimensionality increases, **data becomes sparser**. What happens if we add irrelevant attributes? What if we have fewer samples?



The Curse of Dimensionality

We find high dimensional datasets in many scenarios:

- **Complex** data types, such as grid data (pictures, audio files).
- Situations where we have **little prior knowledge** and we record everything, just in case.

Collecting as many attributes as possible seems like a good idea. Is it? Not necessarily: it **gives Randomness more opportunities to fool you**.

The **curse of dimensionality** is a warning. Irrelevant attributes **do not cancel each other out**, they show up as spurious patterns. Adding more attributes (*just in case*) can actually result in worse-performing models.

We can use **feature selection** techniques to reduce the dimensionality of the problem but first, let's use our **domain knowledge**.

Don’t let randomness fool you!

Agenda

Grid Data

We have seen many datasets, we specify a particular kind of data as **Grid Data**:

Grid Data: A data that lives on a grid. The values (predictors) of data are arranged in a way where the notion of neighborhood is used.

For instants, temporal or spatial arrangements of values is a grid data. For example

- Audio/Speech signal - samples are arranged in a temporal fashion
- Image - pixels are arranged in a spatial fashion.
- Graph data - nodes connected to neighbors.

Agenda

Neural networks: Not quite new

Neural networks were first proposed in the 1940s, became very popular in the 70s and 80s and fell out of favour in the 90s, as:

- Other algorithms performed better.
- They have a large number of parameters and are hard to train.
- Analysing them is difficult.
- **Convexity**.

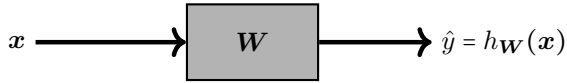
From 2010 onwards there has been a resurgence of interest in neural networks because of:

- Improved **optimisation** algorithms, increased **computational power** and **larger datasets**.
- Careful definition of **architectures**.
- Application of **transfer learning**.

What is a neural network?

A neural network is a **computing system** loosely inspired by the human nervous system. From a **functional** point of view, a neural network:

- Produces an **output** (label) given an **input** (predictors), .
- Has **weights** (parameters) that can be **tuned** (trained).



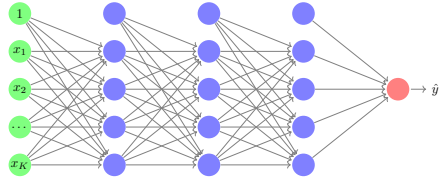
$\boldsymbol{W}$ denotes all the weights of the neural network and $h_{\boldsymbol{W}}(\boldsymbol{x})$ indicates explicytely that the prediction $\hat{y}$ depends on $\boldsymbol{W}$.

What is a neural network?

Neural networks consist of simple, interconnected computing units that (loosely) mimmic **neurons**. This architecture is appealing since:

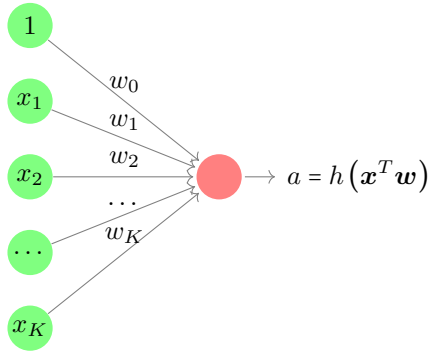
- **Neuroscience** suggests biological neural networks can solve any problem.
- **Mathematics** suggests artificial neural networks can reproduce any input/output relationship, provided they are complex enough.

Hence, the family of neural network models can be seen as a **universal machine**.



The perceptron

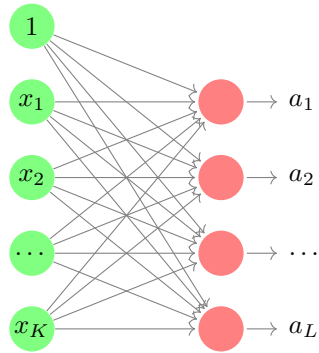
The perceptron is the **basic unit** of a neural network. It is defined by a **weight vector** $\boldsymbol{w}$ and an **activation function** $h(\cdot)$ that map an extended vector $\boldsymbol{x}$ to an output a. The coefficient w_0 is known as the **bias**.



The activation function is **non-linear**.

Layers

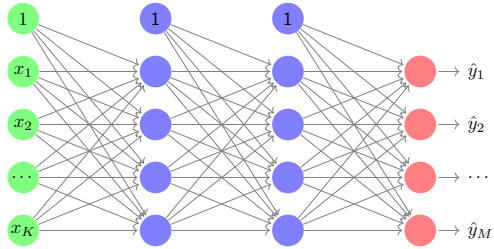
A **layer** is a collection of perceptrons that use the **same input**. Each perceptron within a layer produces a separate output.



A layer consisting of L perceptrons and K inputs has $L \times (K + 1)$ weights.

The architecture

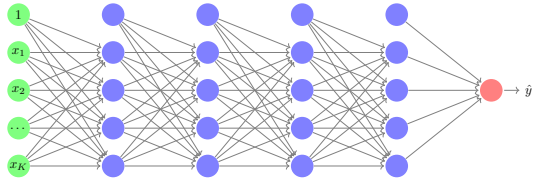
The **architecture** of a neural network describes how layers are connected. The **input layer** is the predictor vector, **hidden layers** produce internal features and the **output layer** produces the prediction.



Vector $\hat{\boldsymbol{y}} = [\hat{y}_1, \ldots, \hat{y}_M]^T = h_{\boldsymbol{W}}(\boldsymbol{x})$ is the output of the neural network. (*Note that here $\hat{y}_j$ is not the prediction for the j-th sample!*)

The neural network

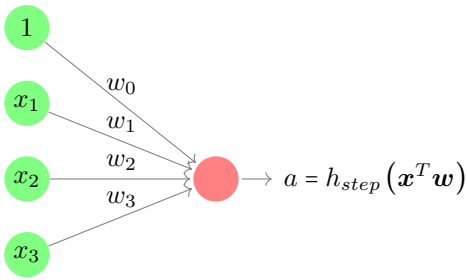
- A neural network consists of **perceptrons** arranged in **layers** that are connected according to an **architecture**.
- There is one **parameter** per connection (the connections's **weight**).
- Each layer produces intermediate **features**.
- The number of layers determines the depth of the neural network (hence the distinction between **shallow** and **deep** neural networks).



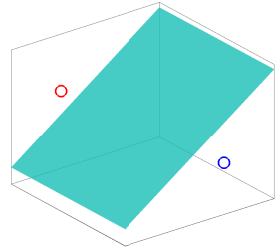
Agenda

The perceptron as a linear classifier

A perceptron that uses a **step** activation function is actually a **linear classifier**. What if it uses a **logistic function**?



$$h_{step}(d) = \begin{cases} 1, & \text{if } d > 0. \\ 0, & \text{otherwise.} \end{cases}$$

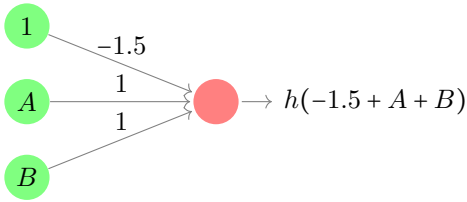


In a 3D predictor space, a perceptron with a step activation function separates two decision regions by a plane!

The perceptron as a basic logical function

We can use perceptrons to implement logical functions.

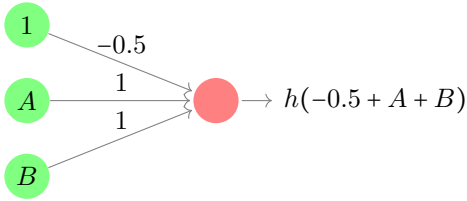
In the preceptron below, assuming that the activation function is a step and A and B can be either 0 or 1, the output $f(A, B)$ corresponds to a logical AND of A and B.



A	B	$f(A, B)$
0	0	0
0	1	0
1	0	0
1	1	1

The perceptron as a basic logical function

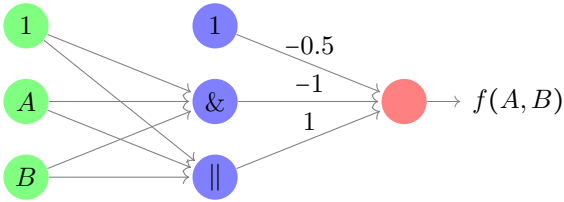
What logical function would you obtain if the coefficients were $w_0 = -0.5$, $w_A = 1$ and $w_B = 1$ instead?



A	B	$f(A, B)$
0	0	
0	1	
1	0	
1	1	

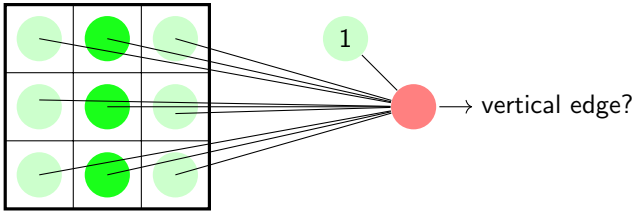
Adding depth to derive new logical functions

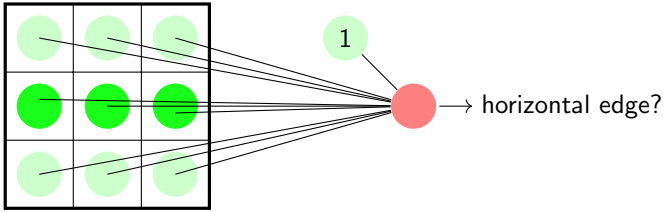
What about the following neural network? In this diagram, & denotes an AND perceptron, and || an OR perceptron.



A	B	&	||	f
0	0			
0	1			
1	0			
1	1			

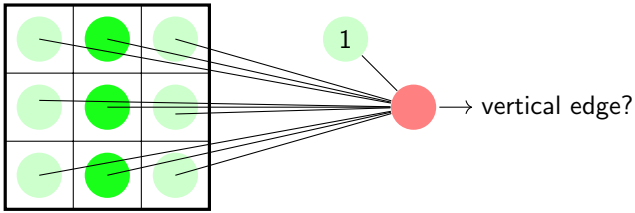
The perceptron as a grid pattern detector

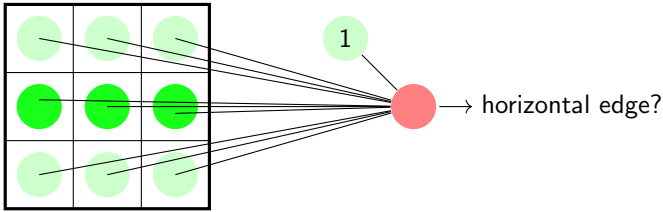




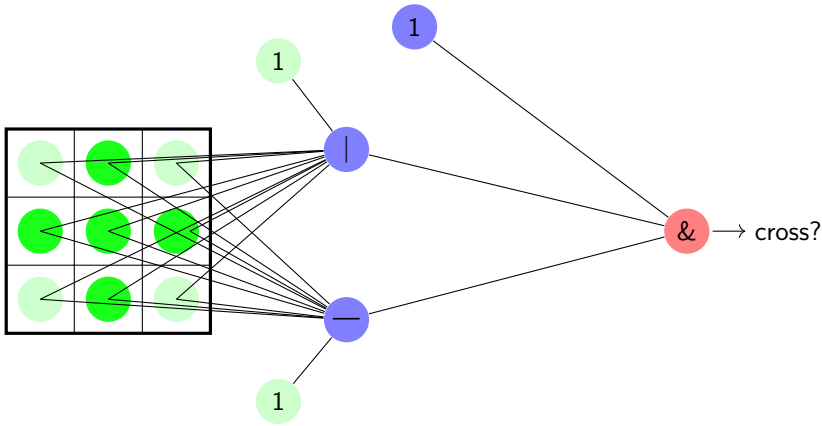
The perceptron as a grid pattern detector: Question

If dark green circle have high value (> 0) and light green circle have low values (< 0), what should be the weights of following perceptrons?



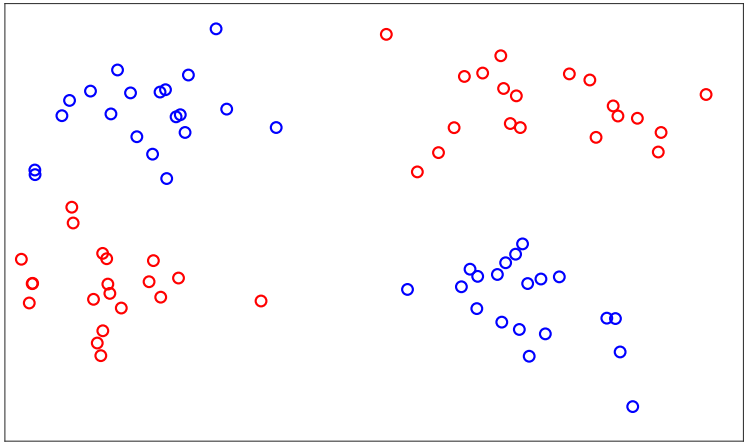


Adding depth to detect derived grid patterns



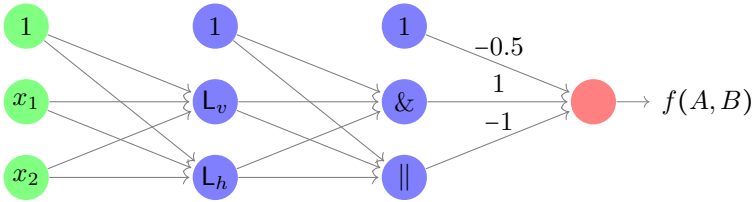
Combining linear classifiers and logical functions: Question

Build a neural network classifier for the following dataset.



Combining linear classifiers and logical functions

In this diagram, L_v and L_h represent perceptrons that implement a vertical and horizontal linear boundaries, respectively. Their outputs indicate whether samples are on one side of the boundary or another.



Versatility of neural networks as a computing system

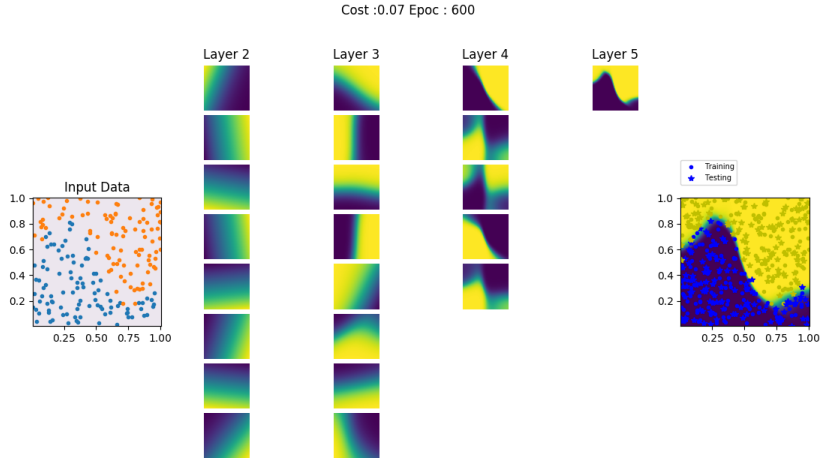
Using a single perceptron we can implement:

- Linear boundaries.
- Logical functions.
- Grid pattern detectors.

Connecting perceptrons allows us to implement more complex operations, e.g. complex boundaries, complex logical functions and complex grid patterns. This can be seen as building complexity from basic operations.

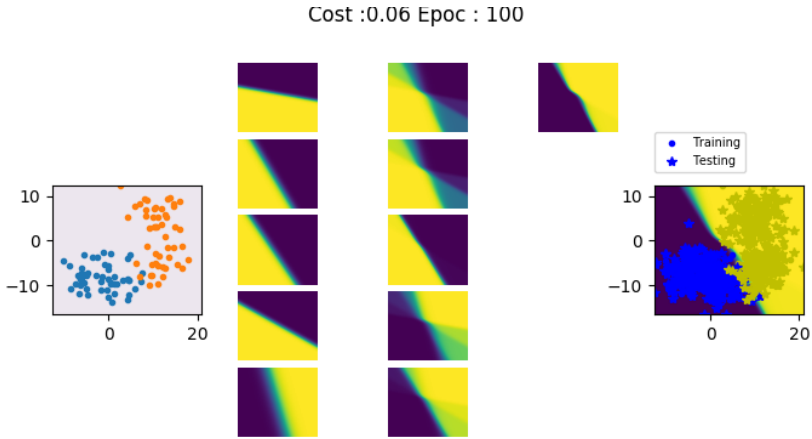
So far we have seen neural networks from a computation angle and have specified each network by ourselves. Note this is **not** machine learning!

Complex Features: Example 1



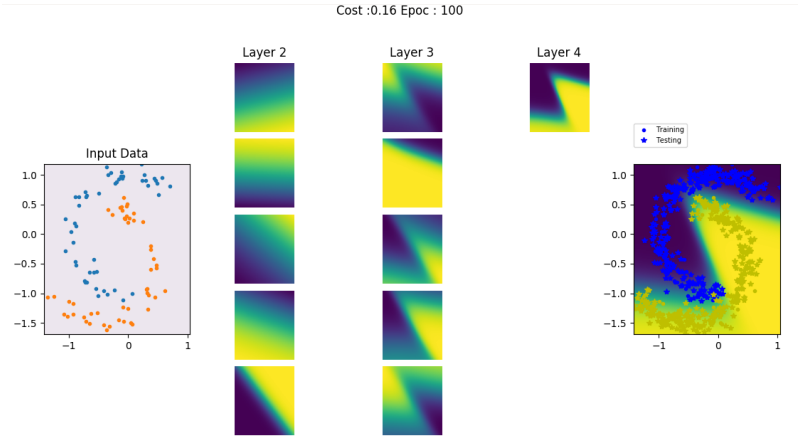
Source: https://github.com/Nikeshbajaj/DeepLearning_from_scratch

Complex Features: Example 2



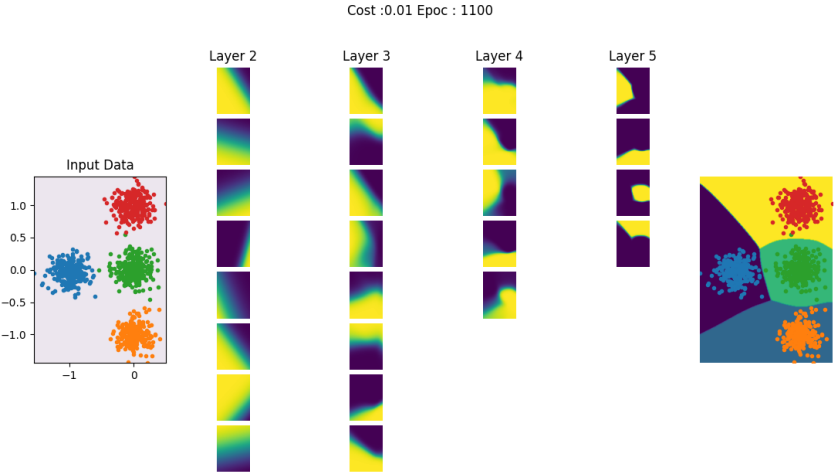
Source: https://github.com/Nikeshbajaj/DeepLearning_from_scratch

Complex Features: Example 3



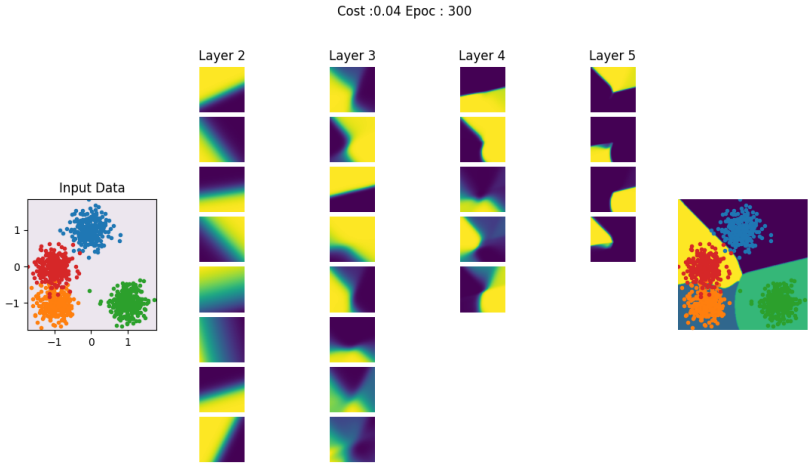
Source: https://github.com/Nikeshbajaj/DeepLearning_from_scratch

Complex Features: Example 4



Source: https://github.com/Nikeshbajaj/DeepLearning_from_scratch

Complex Features: Example 5

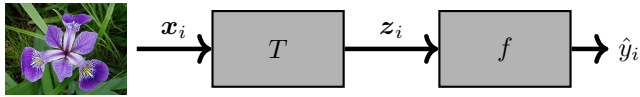


Source: https://github.com/Nikeshbajaj/DeepLearning_from_scratch
Visualisation: https://nikeshbajaj.github.io/teaching/mldl101/cover.gif

Agenda

Machine learning pipelines: Reminder

A machine learning pipeline is a sequence of data operations.



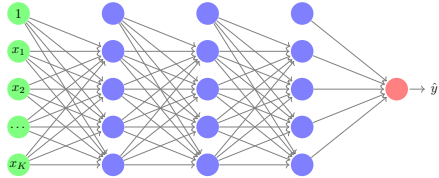
A simple pipeline consists of a first transformation stage followed by a machine learning model:

- The transformation stage produces **derived** features.
- The model uses the derived features as input.
- The transformation stage can also be trained.

Neural networks as tunable pipelines

A neural network can be seen as an entire tunable machine learning pipeline, where:

- Each perceptron defines one **derived feature** or **concept** using other features, raw or derived.
- **Increasing the number of perceptrons** in a layer allows to create more new concepts per layer.
- **Increasing the number of layers** (deeper networks) allows to create concepts of increasing complexity.



Deep neural networks: Computational angle

Large neural networks are appealing, as they give us the necessary **flexibility to create new and increasingly complex concepts** that might be relevant to make a prediction.

However:

- Higher flexibility increases the risk of **overfitting** (which we can see as a network creating and using irrelevant concepts).
- Large number of parameters need to be tuned, therefore the **computational requirements** might be too high.

For **complex inputs**, such as pictures consisting of millions of pixels, this is even more severe.

Training neural networks: Cost function

Every Machine Learning algorithm needs a **model**, a **cost function** and an **optimisation** method.

Given a dataset $\{(\boldsymbol{x}_i, y_i), 1 \leq i \leq N\}$, where labels can take on the values 0 or 1, a common cost function for classification is the **negative log-likelihood function**, defined as:

$$l(\boldsymbol{W}) = -\frac{1}{N} \sum_{n=1}^{N} y_i \log[\hat{y}_i] + (1 - y_i) \log[1 - \hat{y}_i]$$

where $\hat{y}_i = h_{\boldsymbol{W}}(\boldsymbol{x}_i)$. This cost function can be extended to multi-class classifiers.

Gradient descent and back-propagation

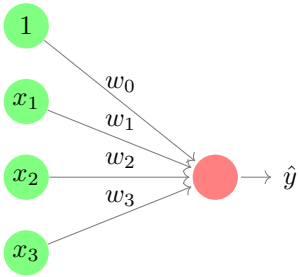
Gradient descent is the method of choice to find the optimal set of coefficients $\boldsymbol{W}$ for the cost function $l(\boldsymbol{W})$.

Obtaining the gradient is easy, but can be computationally expensive. **Back-propagation** is an efficient algorithm to **compute the gradient**. This gradient is then used by the optimisation algorithm to update $\boldsymbol{W}$.

Back-propagation exploits the **chain rule of calculus**. It turns out that to compute the gradient in one layer, we just need information from the next layer. Back-propagation starts from the output: it obtains the cost and proceeds backwards calculating the gradients of the hidden units.

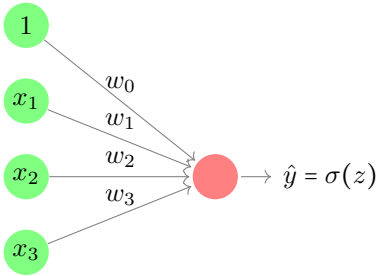
Gradient descent and back-propagation

Linear: $\hat{y} = w_0 + \sum_i w_i x_i$



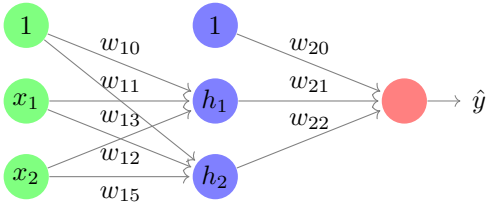
Gradient descent and back-propagation

Non-Linear: $\hat{y} = \sigma\left(w_0 + \sum_i w_i x_i\right)$
$\sigma(z) = \frac{1}{1+e^{-}z}$



Gradient descent and back-propagation

$\hat{y} = \sigma\left(w_{20} + \sum_i w_{2i} h_i\right)$
$h_1 = \sigma\left(w_{1,0} + \sum_i w_{1i} x_i\right)$
$h_2 = \sigma\left(w_{13} + \sum_i w_{1i+3} x_i\right)$
$\sigma(z) = \frac{1}{1+e^{-}z}$



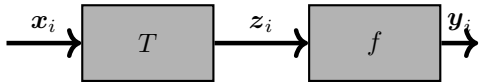
Training considerations

- **Initialisation**: If the initial weights are zero, back-propagation fails. Initial weight values should be random.
- **Overfitting**: Neural networks can have millions of parameters. Use regularisation and validation-based early stop to avoid overfitting.
- **Non-convexity**: The cost function has multiple local minima. Retrain from different random starting values.
- **Scaling of the inputs**: Range of input values can affect the values of the weights. Standardise to ensure inputs are treated equally.
- **Architecture**: Different architectures suit different problems.

Transfer learning

A neural network that has been successfully trained for a problem A can be reused for a related problem B, for instance:

- We can leave the early stages **unchanged**, becoming a fixed transformation stage $T(\mathbf{x})$.
- We can **retrain** the late stages using new data $f(\mathbf{z})$.



We are in essence **transferring an already learnt transformation** and reusing it for a different problem:

- No need to train $T(\mathbf{x})$ (same parameters for problems A and B).
- The optimal parameters of $f(\mathbf{z})$ for problem B will be close to the ones found for problem A (shorter training time!).

Transfer learning: A little more with math

Key mathematical concepts to define transfer learning

- Source: $\mathcal{X}_s$, $\mathcal{D}_s = \{\mathcal{X}_s, P_s(X)\}$, $\mathcal{T}_s = \{\mathcal{Y}_s, P_s(Y|X)\}$
- Target: $\mathcal{X}_T$, $\mathcal{D}_T = \{\mathcal{X}_T, P_T(X)\}$, $\mathcal{T}_T = \{\mathcal{Y}_T, P_T(Y|X)\}$
- Source dataset $\{\mathcal{X}_s, \mathcal{Y}_s\}$
- Target dataset (limited) $\{\mathcal{X}_T, \mathcal{Y}_T\}$

- $\mathcal{D}_s \neq \mathcal{D}_T$
- $\mathcal{T}_s \neq \mathcal{T}_T$

Transfer learning: Examples

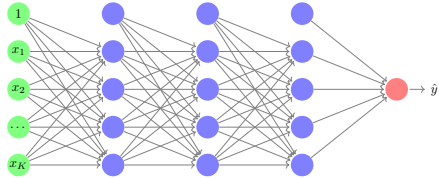
Examples: similarity of domain

- Image datasets
- Audio datasets
- Biomedical signals

Agenda

Fully-connected layer

So far we have considered **fully-connected** (FC) layers, where all the perceptrons receive all the outputs from the previous layer. FC layers have a **large number of parameters** and training them can be challenging.



Assuming the input of this neural network is an RGB picture consisting of 1000×1000 pixels, how many parameters per perceptron are there in the first FC layer?

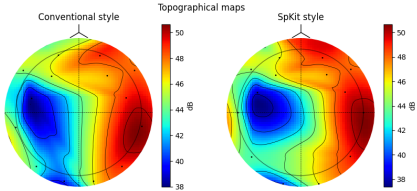
Equivariance in grid data

Images and time series are complex data types consisting of individual attributes associated to a **regular grid** defining a **spatial relationship**.

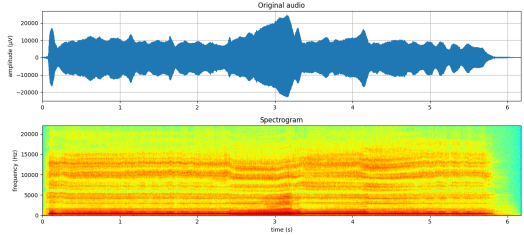
Some grid data exhibit the **equivariance** property, according to which the same pattern can be expected in different locations of the grid.

Equivariance in grid data: Examples

Images with no Equivariance.



Brain Activity



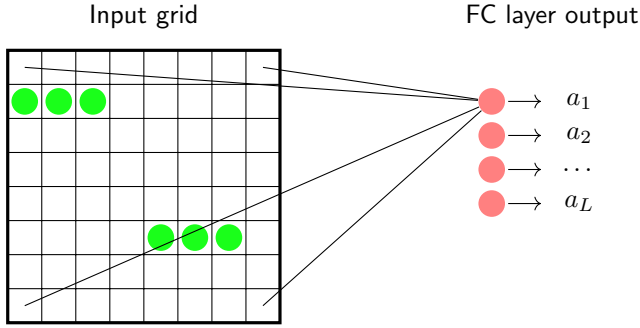
Spectogram of an audio

MRI Images
CT Scan Images

Equivariance in grid data: FC layer

How would a FC layer identify a short horizontal segment in the following 8×8 grid?

- Each perceptrion connected to all inputs: 8×8+1 weights.
- As many perceptrons as potential locations, L.
- Total of $L \times (8 \times 8 + 1)$ weights.

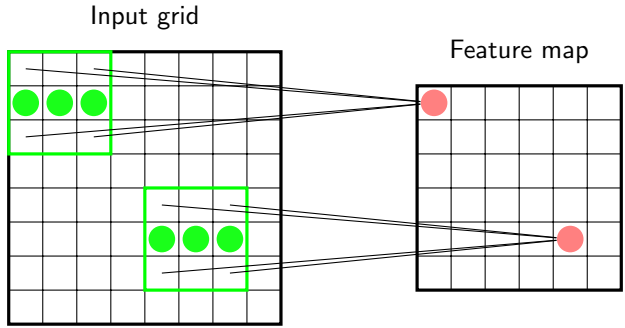


Equivariance in grid data: The convolutional layer

Convolutional layers impose additional restrictions:

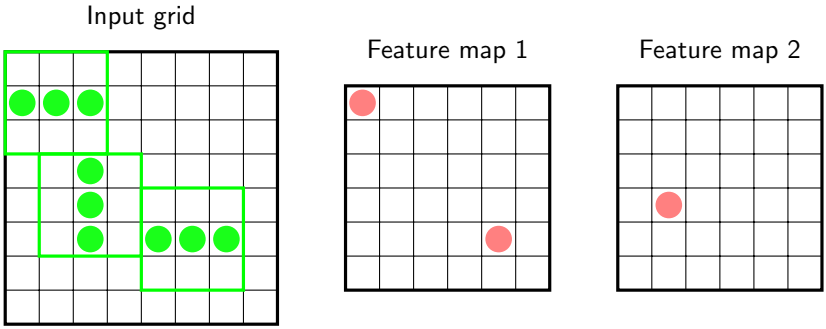
- Perceptrons are arranged as a grid known as **feature map**,
- focus on different **limited regions** in the input grid and
- **share** their parameters, represented as a grid called **kernel**.

The feature map is efficiently calculated as a **convolution** of the kernel and the input or in other words, **filtering** the input with the kernel.



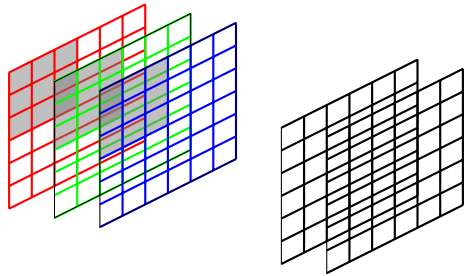
Convolutional layers

Convolutional layers can have **several feature maps**, each of which is associated to a **different concept**. They form a **stack** of maps.



Convolutional layers

- The **dimensions of a kernel** are $H \times W \times D$, there H is the height, W is the width and D is the depth (number of input feature maps).
- The total number of **weights per kernel** is $H \times W \times D + 1$ (bias).
- **Training** a convolutional layer means using data to tune the weights of each kernel.



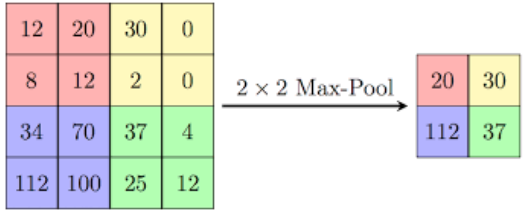
Pooling layers

Pooling **reduces the size** of feature maps. Pooling layers are defined by size of the area they are reducing to a single number and are inserted between successive convolutional layers.

They come in two flavours:

- **Max pooling**: The output is the largest value within the filter area.
- **Average pooling**: The output is the average of the values within the filter area.

Note that pooling layers do not need to be trained!



Deep learning architectures

Deep neural networks are **not arbitrary** sequences of **arbitrary** layers. On the contrary, they have a **predefined architecture** that is suitable for a specific goal.

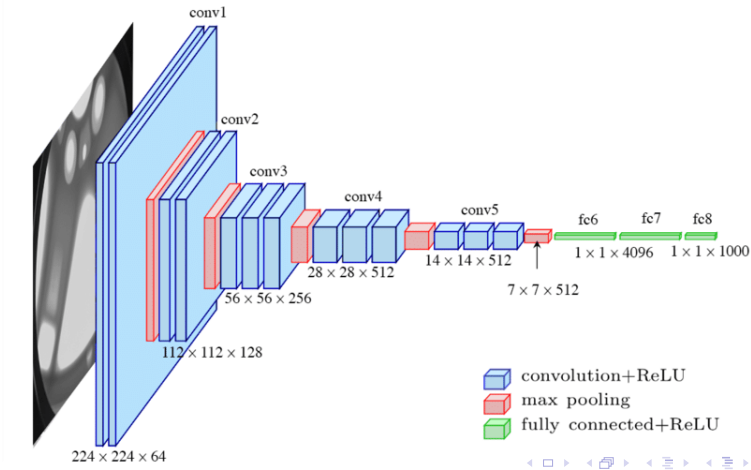
In classification, it is common to see architectures in which:

- The first layers define a **few, simple** concepts, the last layers define **many, complex** concepts.
- Feature maps **shrink** as we move deeper into the network.

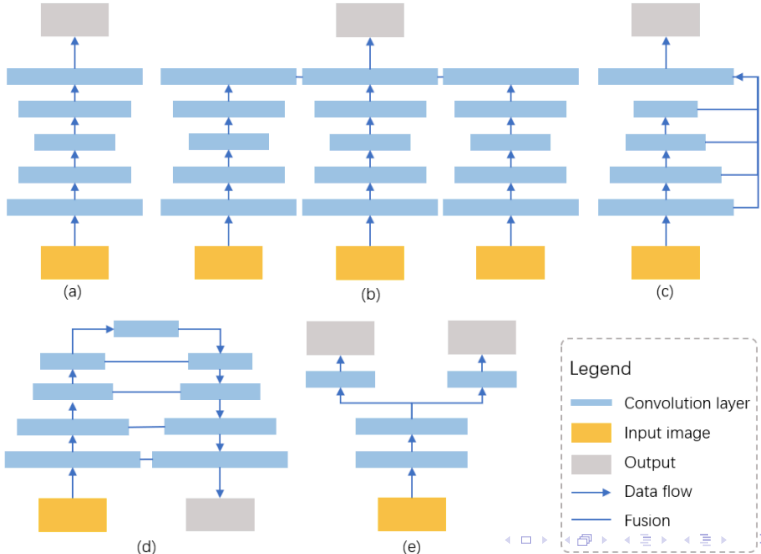
The same intermediate concepts can be useful for different goals. We can use **transfer learning** to reuse existing solutions.

Example: The VGG16 network

- VGG16 was designed for the ImageNet Large-Scale Visual Recognition Challenge (dataset of images belonging to 1000 classes).
- VGG16 has 16 layers, 3×3 kernels and 138 million weights.



More architectures

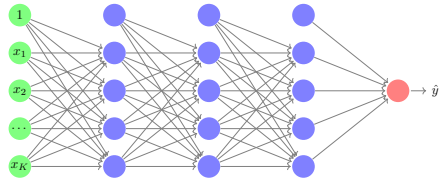


Agenda

Neural networks: the three views

We can approach neural networks from three angles:

- **Functional**: system that **maps** an input $\boldsymbol{x}$ to an output $\hat{y}$. By tuning its set of parameters $\boldsymbol{W}$, we change the mapping.
- **Cognitive**: system that **creates new concepts** from raw data and other concepts. By tuning $\boldsymbol{W}$, we create different concepts.
- **Computational**: pipeline of interconnected computing units called **perceptrons**. By tuning $\boldsymbol{W}$, we change the computation.



Neural networks: A family of machine learning models

Neural networks should not be seen as a machine learning model in the same sense as, for instance, a logistic regression classifier.

- Neural networks are a **family** of machine learning models.
- Each neural network has an **architecture**, the simplest one of which is one single perceptron.
- Some architectures are **not substantially different** from other machine learning models (e.g. the perceptron is a linear classifier).
- We can see neural networks as a **framework** to create new models.
- We **train** one specific architecture and can use **validation** approaches to choose the right architecture.

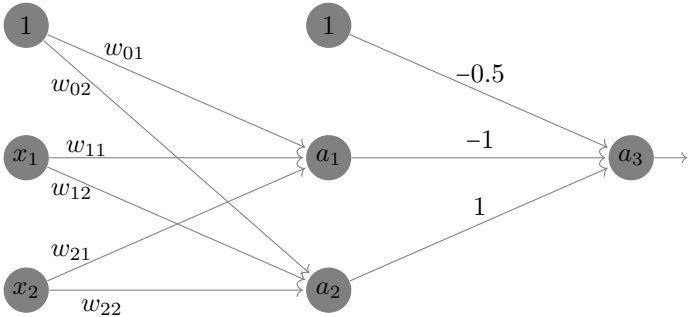
We should **avoid** saying *neural networks perform well* for a given problem. Instead, we should say *this neural network architecture performs well*.

Neural networks: Universal machines

For any problem, we can find a neural network architecture that solves it. In this sense, neural networks are said to be universal machines.

- Note that this does **not mean** that a specific neural network architecture can solve any problem.
- The existence of a suitable neural network architecture does **not imply** that we will be able to find it.
- Flexibility is achieved adding **complexity**, which increases the risk of **overfitting**.
- **Neural network experts** design the right architecture for a problem and reuse existing solutions using the principles of transfer learning.
- **Neural network brutes** are unaware of the Monkey Theorem and use all the computational power and data available to train as many complex architectures as possible. Their **carbon footprint** is huge.

Ex 1



Queen Mary
University of London