School of Electronic Engineering and Computer Science
Queen Mary University of London

Machine Learning
Neural Networks and Deep Learning

Dr Chao Liu

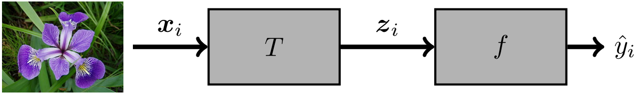
Credit to Dr Jesus Requena

Nov 2023

Agenda

Machine learning pipelines: Reminder

A machine learning pipeline is a sequence of data operations.



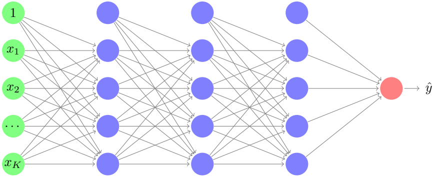
A simple pipeline consists of a first transformation stage followed by a machine learning model:

- 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

From a cognitive point of view, 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.

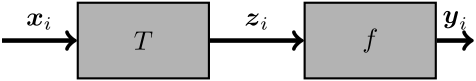
Training considerations

- **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 $f(\mathbf{z})$.



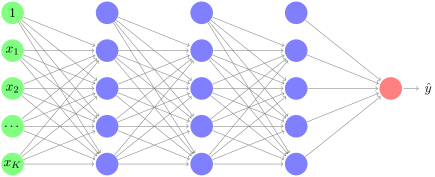
We are in essence **transferring an already learnt transformation** and reusing it for a different problem:

- 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!).

Agenda

Fully-connected layer

So far we have considered **fully-connected** (FC) layers are, 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

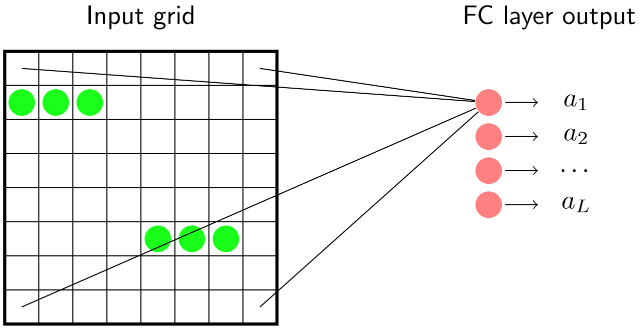
Images and time series are complex data types consisting of individual attributes associated to a **regular grid** defining a **spatial relationship**.

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: 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.

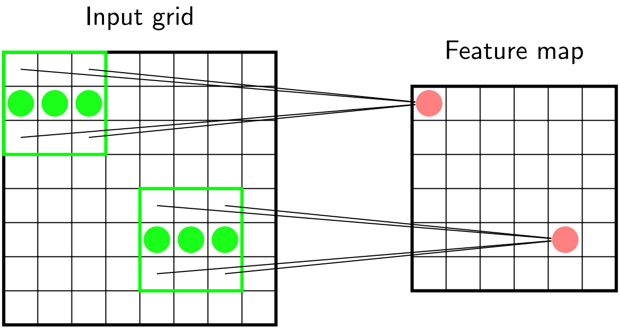


Equivariance in grid data: The convolutional layer

Convolutional layers impose additional restrictions:

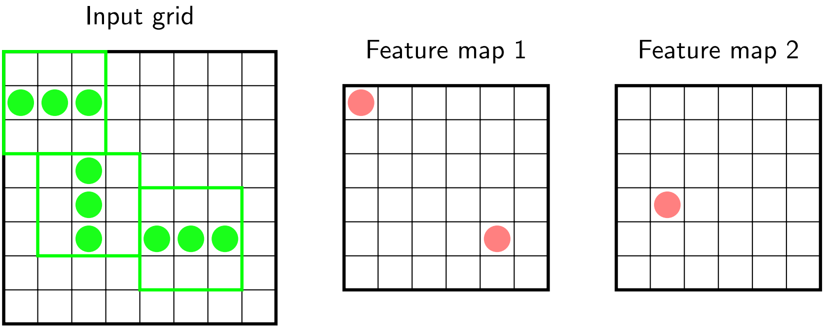
- Perceptrons are arranged as a grid known as **feature map**,
- focus on different **limited regions** in the input grid and
- **share** their parameters, represented as a grid called **kernel**.

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.



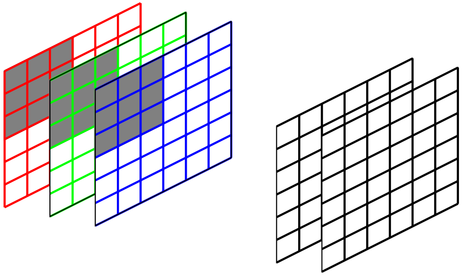
Convolutional layers

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.



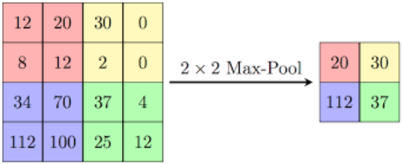
Pooling layers

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.

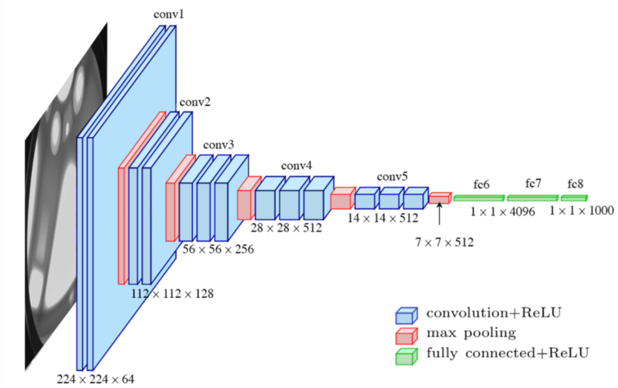
In classification, it is common to see architectures in which:

- 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.

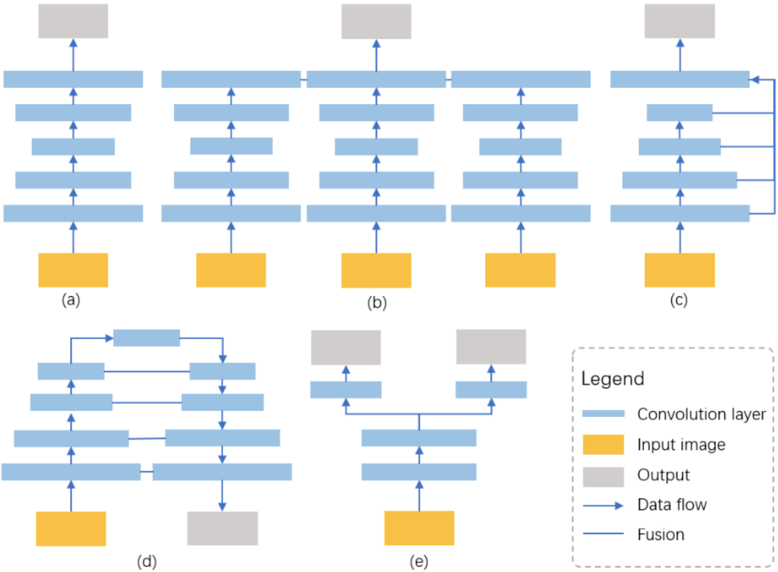
The same intermediate concepts can be useful for different goals. We can use **transfer learning** to reuse existing solutions.

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

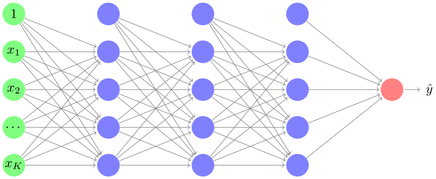


Agenda

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.