LECTURE 7: BASIC NEURAL NETWORKS

EKATERINA MURAVLEVA

OVERVIEW OF THE COURSE

Lecture 1: General course information, CRISP-DM methodology

Lecture 2: Supervised learning/unsupervised learning. Classification/regression problems. Accuracy metrics (precision, recall, ROC-AUC scores). Concept of loss functions, overfitting / underfitting.

Lecture 3: Classical ML: Linear regression, logistic regression, support vector machine

Lecture 4: Classical ML: Decision trees, random forests, boosting.

Lecture 5: Classical ML: Dimensionality reduction: linear, non-linear methods.

Lecture 6: Classical ML: Clustering methods

Lecture 7: Basic neural networks

Lecture 8: Scalable algorithms

RECAP OF LECTURE 6

- Definition of clustering
- K-Means
- Cluster validity measures
- Hierarchical clustering
- Graph cuts
- DBScan

PLAN OF LECTURE 7

— Fully connected neural network
— Learning
— Backpropagation
— Types of NN architectures (brief summary)

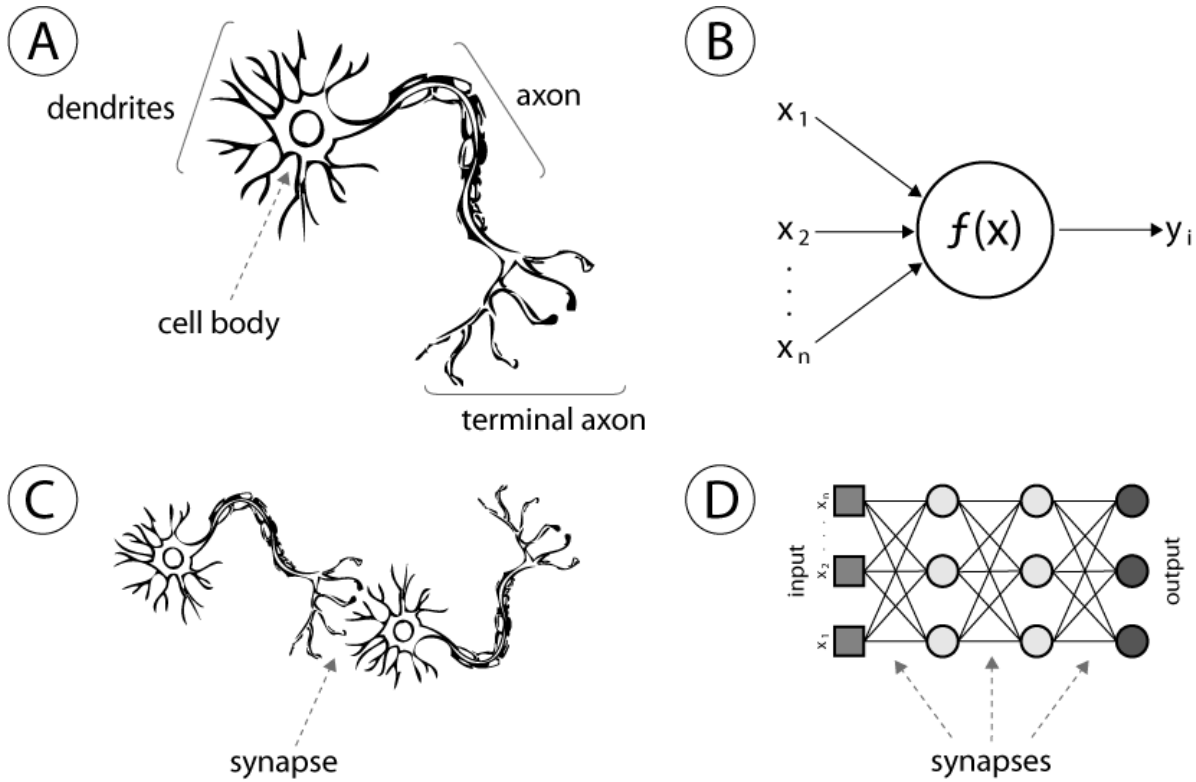
BRIEF HISTORY OF NEURAL NETWORKS

History of neural networks goes back to 1940.

Scientists tried to mimic **human brain**

and came up with the idea of

artificial neural networks



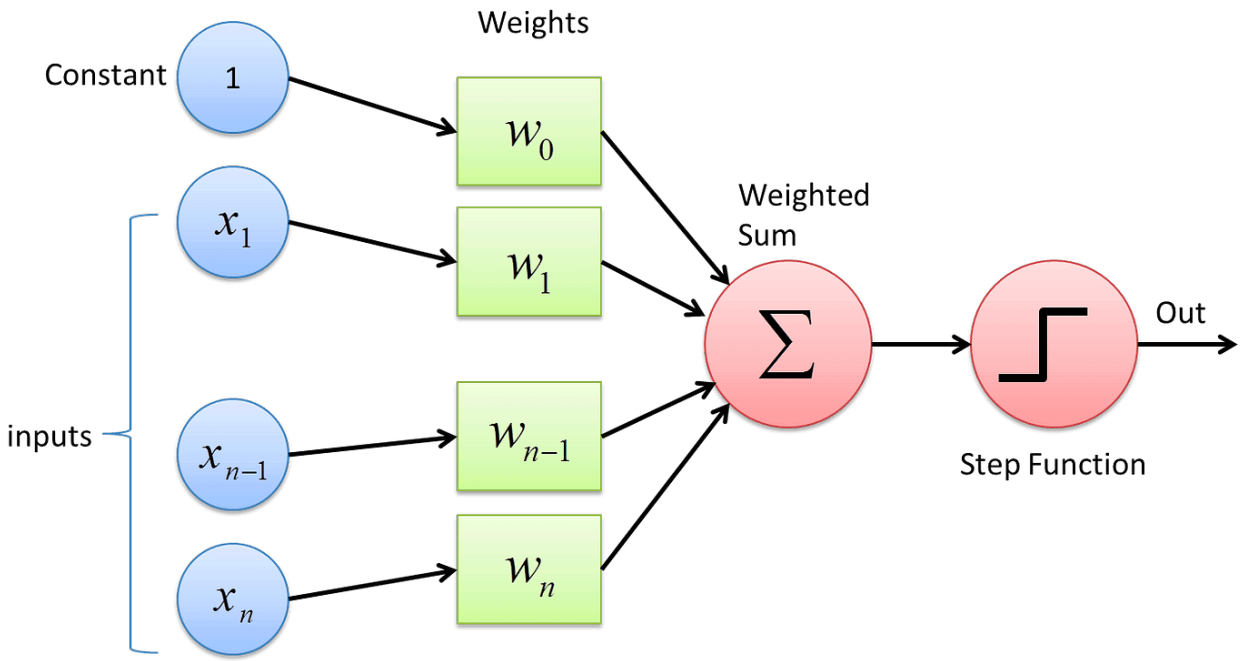
PERCEPTRON

— **Perceptrons** are the simplest form of neural networks, consisting of a single layer of interconnected nodes.

— They were first introduced in the 1950s by psychologist Frank Rosenblatt as a way to simulate the functioning of a single neuron in the brain.

— The basic structure of a perceptron consists of an input layer, a single layer of interconnected nodes, and an output layer. Each node in the input layer represents a feature or input variable, and each node in the output layer represents a class or category.

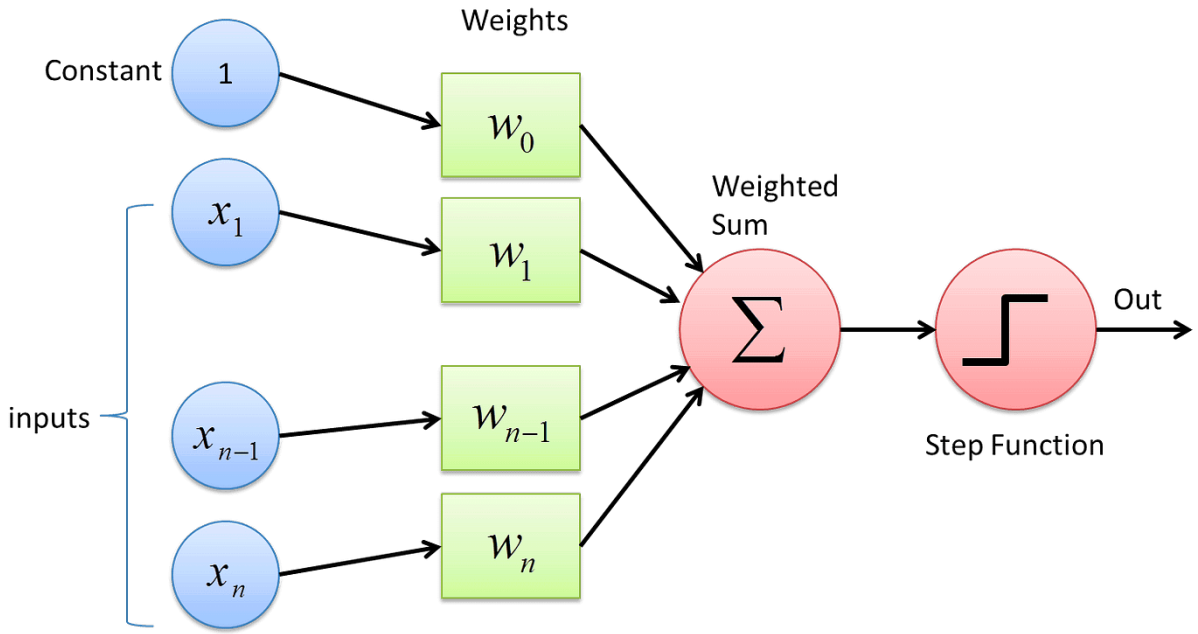
— The nodes in the hidden layer perform calculations based on the inputs and pass the result to the output layer, where the final prediction is made.



PROPERTIES OF THE PERCEPTRON

— The activation function is a crucial component of a perceptron. It determines the output of each node based on the weighted sum of its inputs. The most commonly used activation function in perceptrons is **the step function**, which outputs a 1 if the weighted sum is above a certain threshold and -1 otherwise.

— One of the **main limitations** of single-layer perceptrons is that they can only learn linearly separable patterns. This means that they can only classify data that can be separated by a straight line or plane. This restricts their ability to handle more complex data and makes them unsuitable for tasks such as image recognition or language processing.



PERCEPTRON LEARNING ALGORITHM

— The learning algorithm for perceptrons is known as the Perceptron Rule.

— It is a supervised learning algorithm that adjusts the weights of the connections between nodes based on the error between the predicted output and the actual output.

— This process is repeated for multiple iterations until the perceptron can accurately classify the data.

— The update makes the classification of the point correct.

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

MULTILAYER PERCEPTRON

— To overcome this limitation, **multi-layer perceptrons (MLPs)** were developed, which have one or more hidden layers between the input and output layers. These hidden layers allow MLPs to learn non-linear relationships in data.

—Multi-layer perceptrons (MLPs) are a type of artificial neural network that consists of multiple layers of interconnected nodes. They were developed in the 1980s as an extension of the single-layer perceptron, in order to overcome its limitations and improve its performance on more complex tasks.

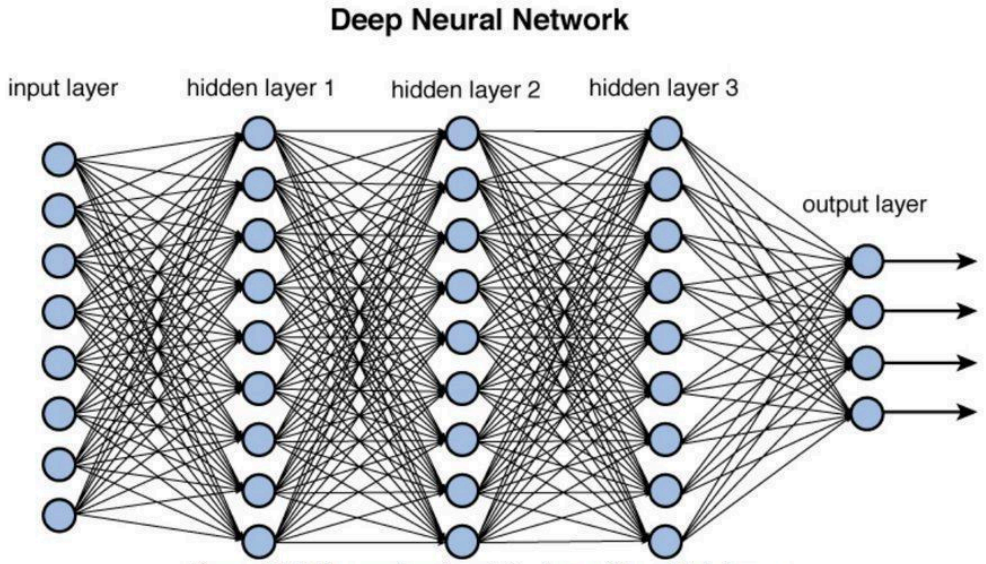


Figure 12.2 Deep network architecture with multiple layers.

MULTILAYER PERCEPTRON

— The architecture of a multi-layer perceptron typically consists of an **input layer**, one or **more hidden layers**, and **an output layer**. The input layer contains nodes that represent the input variables, and the output layer contains nodes that represent the predicted output.

— The hidden layers perform calculations based on the inputs and pass the results to the next layer until the final prediction is made in the output layer.

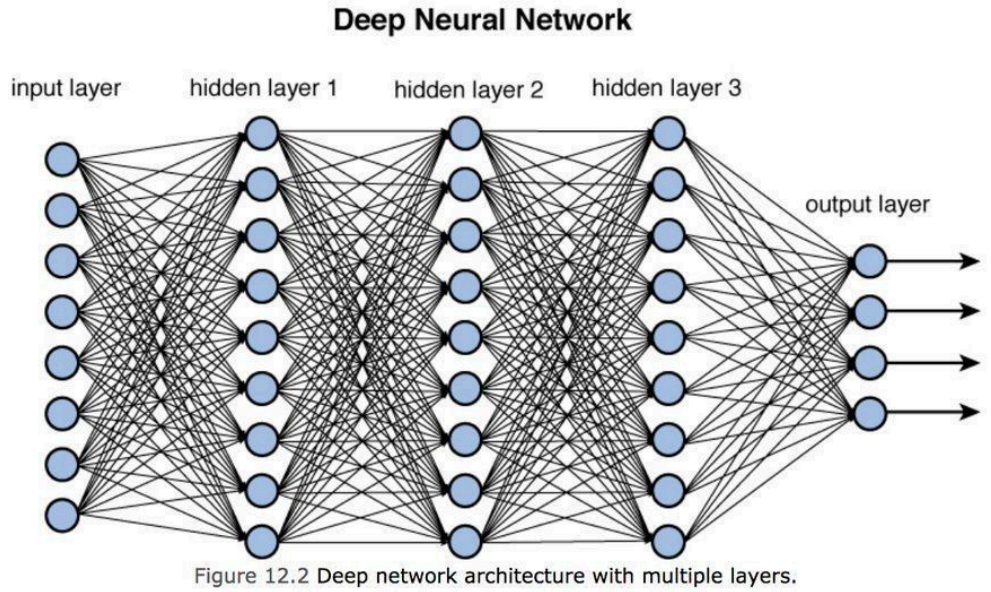


Figure 12.2 Deep network architecture with multiple layers.

MULTILAYER PERCEPTRON

— One of the key differences between single-layer perceptrons and MLPs is the use of non-linear activation functions.

— The most commonly used activation functions in MLPs are the sigmoid function and the Rectified Linear Unit (ReLU) function.

— The **sigmoid function** maps the input to a value between 0 and 1, while the **ReLU** function returns the input if it is positive, or 0 otherwise.

$y_k = f_k(y_{k-1}) = g(W_k y_{k-1} + b_k)$, where g is an activation function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad ReLu(x) = \max(x,0).$$

MULTILAYER PERCETRON AND DEEP NEURAL NETWORK

— Multilayer perceptron is also called **fully connected deep neural network.**

— It can be mathematically written as a superposition of simple functions: linear transformations and point wise activation functions.

$y_k = f_k(y_{k-1}) = g(W_k y_{k-1} + b_k)$, where g is an activation function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad ReLu(x) = \max(x,0).$$

MLP PARAMETERS

— In the supervised setting, we need to determine the parameters (weights) of the network.

— This can be done using gradient-based optimization using so-called **back propagation algorithm**

$y_k = f_k(y_{k-1}) = g(W_k y_{k-1} + b_k)$, where g is an activation function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad ReLu(x) = \max(x,0).$$

DEEP NEURAL NETWORK: TRAINING

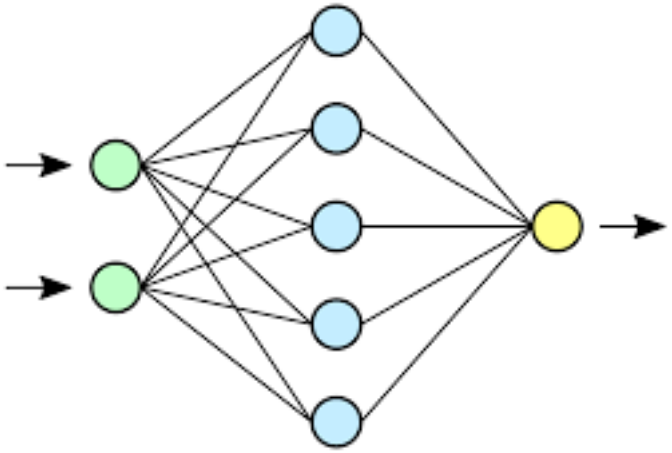
Deep neural network is parametrized by its parameters (weights and biases).

In supervised setting we need to solve the minimization of the form:

$$f(\theta) = \sum_{k=1}^{N} l(y_k, \hat{y}_k) \to \min, \quad \text{s.t.}\, \hat{y}_k = f(x_k, \theta)$$

INFORMAL ILLUSTRATION OF DEEP NEURAL NETWORK TRAINING

Suppose our neural network has 3 computational blocks



We take the the input and predict the output.

We need to compute all intermediate inputs.

Our output is different from the ground truth.

We need to correct the previous block (hidden layer) to make a correct output.

Then we need to correct previous input to make a better output.

In such a way, we correct the last block, then the previous, etc.

FORMAL DESCRIPTION OF GRADIENT DESCENT

We just update the parameters using the gradient descent:

$$\theta_{k+1} = \theta_k - \lambda \nabla f(\theta_k),$$

and for sufficiently small λ we will have

$$f(\theta_{k+1}) < f(\theta_k)$$

Thus, we need an efficient way to compute the gradient!

STOCHASTIC OPTIMIZATION

Recall that we have optimization of the form

$$f(\theta) = \sum_{k=1}^{N} l(y_k, \hat{y}_k), \hat{y}_k = f(x_k, \theta) \to \min$$

The sum goes over all the points in the dataset. Suppose we have million points in the dataset.

The gradient is the some of the gradients of each term.

Computing all is too much for a single gradient.

Instead, we can select a random sample from the dataset (called **batch**) and some over it

$$\hat{f}(\theta) = \sum_{k=1}^{B} l(y_{i_k}, \hat{y}_{i_k}) \approx f(\theta)$$

The size of the batch is **typically small** (32-1024 samples), and is determined by the limitations of the memory (larger batch sizes do not fit).

MODIFICATIONS OF STOCHASTIC OPTIMIZATION

Besides vanilla Stochastic Gradient Descent (SGD)

there are many variants, which intend to speed up the computations.

Among them:

— SGD with momentum

— Adaptive Momentum optimization (Adam)-mostly widely used method.

The idea of **momentum** is as follows: we have a noisy estimate of the gradients, if we smooth it, we get a better direction for changing the parameters.

Disadvantage: need to store additional parameters

(for SGD two times more memory, for Adam 3 times more memory).

BACKPROPAGATION

The gradient of the cost functional for neural networks is computed using the process called **backpropagation**.

Fundamental fact (Baur-Strassen theorem):

if we can evaluate the function of P variables using M operations,

we can evaluate the gradient (i.e., P numbers) using cM operations, where c is a small constant.

This is highly non-trivial result (and often surprising even for professionals in numerical analysis).

FORMAL DESCRIPTION OF BACKPROPAGATION

The backpropagation is derived for the computation of gradients of superposition of functions.

Suppose we have 3 layers, given by 3 functions:

$$y_1 = f_1(x, \theta_1), \quad y_2 = f_2(y_1, \theta_2), \quad y_3 = f_3(y_2, \theta_3).$$

We want to compute $\frac{\partial y_3}{\partial \theta_1}, \frac{\partial y_3}{\partial \theta_2}, \frac{\partial y_3}{\partial \theta_3}$.

FORMAL DESCRIPTION OF BACKPROPAGATION (2)

Suppose we have 3 layers, given by 3 functions:

$$y_1 = f_1(x, \theta_1), \quad y_2 = f_2(y_1, \theta_2), \quad y_3 = f_3(y_2, \theta_3).$$

We want to compute $\frac{\partial y_3}{\partial \theta_1}, \frac{\partial y_3}{\partial \theta_2}, \frac{\partial y_3}{\partial \theta_3}$.

Computing the gradient with respect to θ_3 is easy, we need to compute the gradient of y_3.

Computing the gradient with respect to θ_2 can be done using **chain rule.**

$\frac{\partial y_3}{\partial \theta_2} = \frac{f_3(y_2, \theta_3)}{\partial y_2} \frac{\partial y_2}{\partial \theta_2}$, and the latter is also available.

Want you need to compute, is the product of the **Jacobian** of the layer function times the vector.

For basic layers it is implemented In a cheap way.

IMPORTANT PROBLEM IN TRAINING: GRADIENT EXPLOSION/VANISHING.

When you take the product in the backpropagation, the gradient can **explode** or go to **vanish** (go to zero), resulting in unstable learning.

$$\nabla_{\theta_{k+1}} f = J_k \ldots J_{d-2} J_{d-1} \frac{\partial f}{\partial y_d}$$

Question: What will be the consequence of vanishing gradient? Of exploding gradient?

There are solutions to this problem, that we do not cover — **batch normalization** and **residual connections**.

SUMMARY OF BACKPROPAGATION

— We consider superposition of simple functions

— We compute the forward pass and store all intermediate activations

— We go backwards through the computational graph, computing gradients and Jacobian matrix-by-vector product

— This is an efficient implementation of the classical chain rule.

DEEP NETWORKS IN PRACTICE: CAT VS DOGS

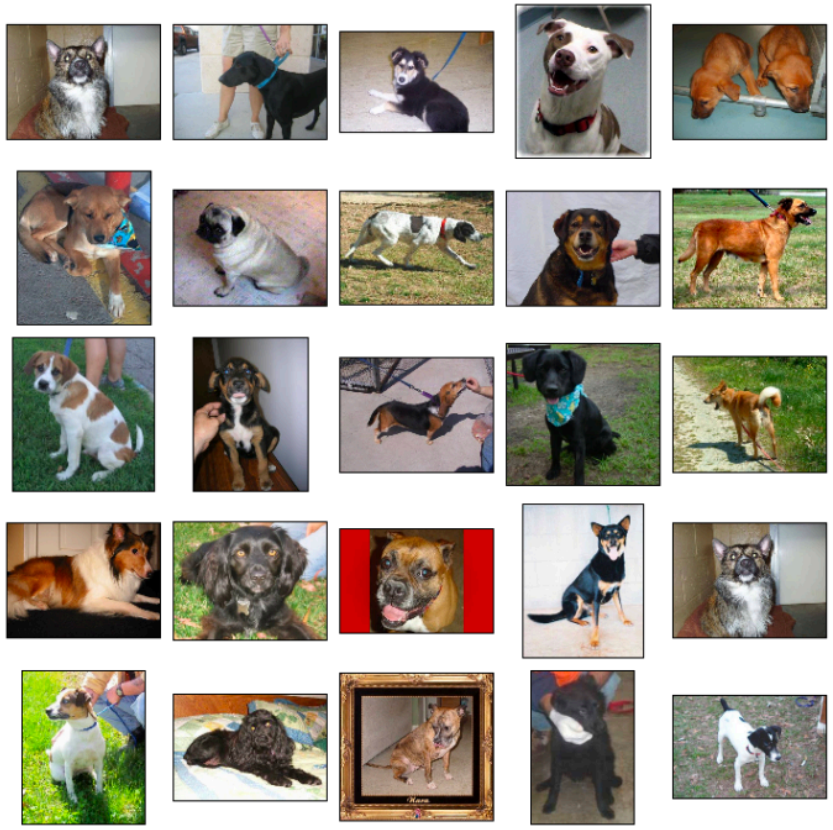
AI could not recognize cat vs dogs in 2006.

Best algorithms could do it with 60% accuracy.

In 2011, DNN algorithms can do it with 99.9%

What changed? New algorithms have been proposed?

Answer is no. We first learned to use old algorithms properly, and only then new ideas were proposed.



IMAGENET

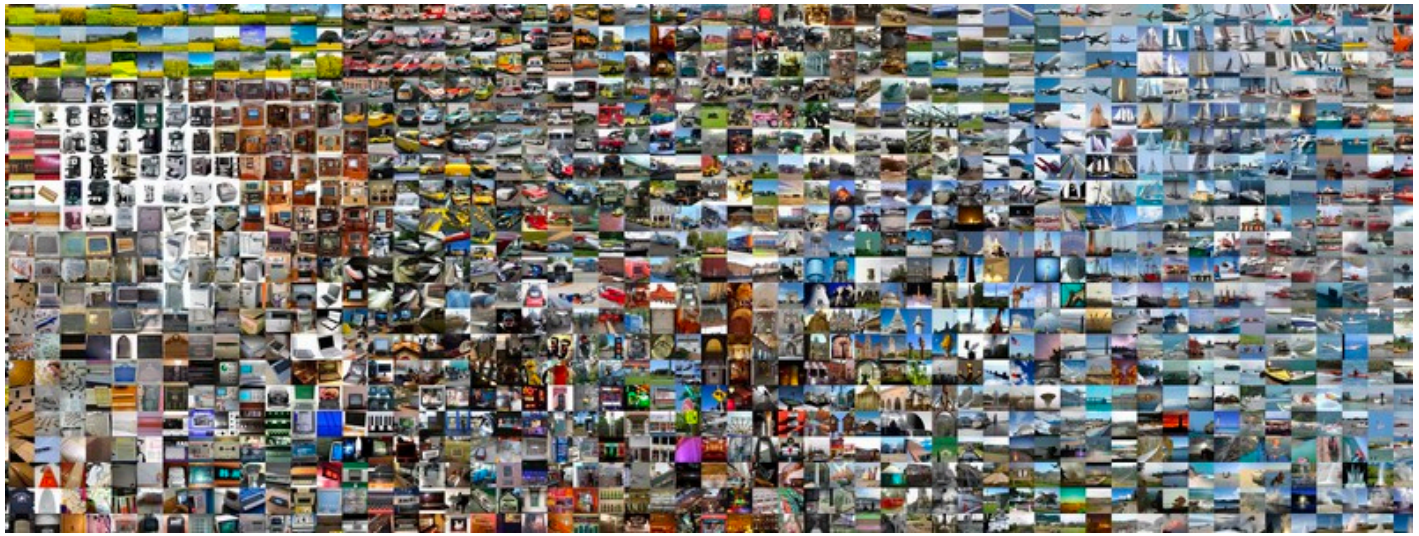
Imagenet:

10M manually labelled images

with 1000 classes.

Considered to be a tough case,

now it is quite easy case.



CONVOLUTIOONAL NEURAL NETWORKS

The biggest breakthrough for deep neural networks have been image classification using CNN (convolutional neural networks)

Image in ImageNet has size 224x224x3, around 150k parameters.
A linear layer will require a matrix of size 150k x 150k, which is too much.

Instead, it has been proposed to use a **subclass** of linear transformations, which is very well suited for images — convolutions.

CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE

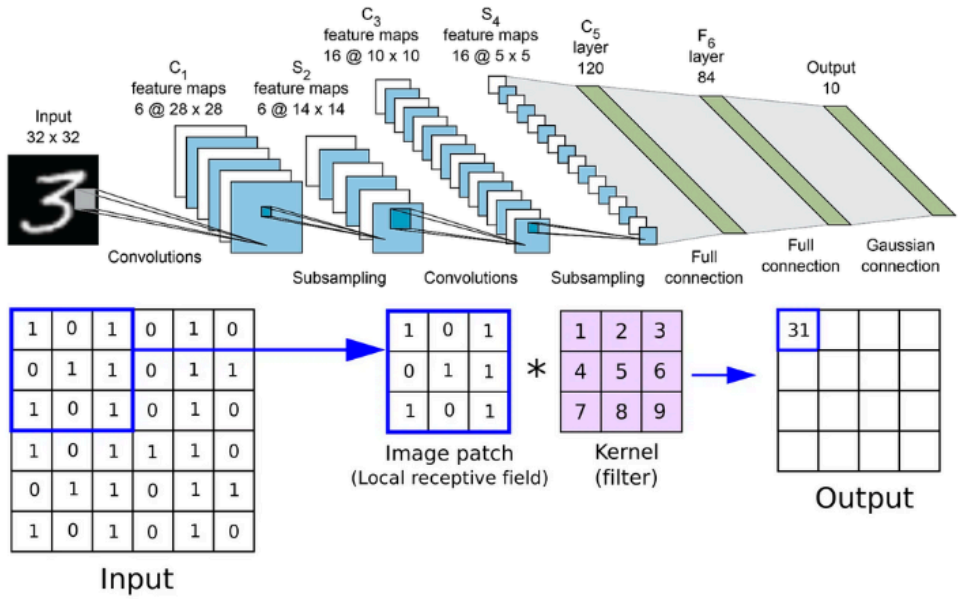
An example of the convolutional neural network architecture is shown on the right.

Each layer operates on a three-dimensional array:

Width x Height x Channels

We can **mix channels,** but in spatial dimensions we do a convolution with a filter (see image on the right).

Also, one can use **pooling operation**



CONVOLUTION AND POOLING

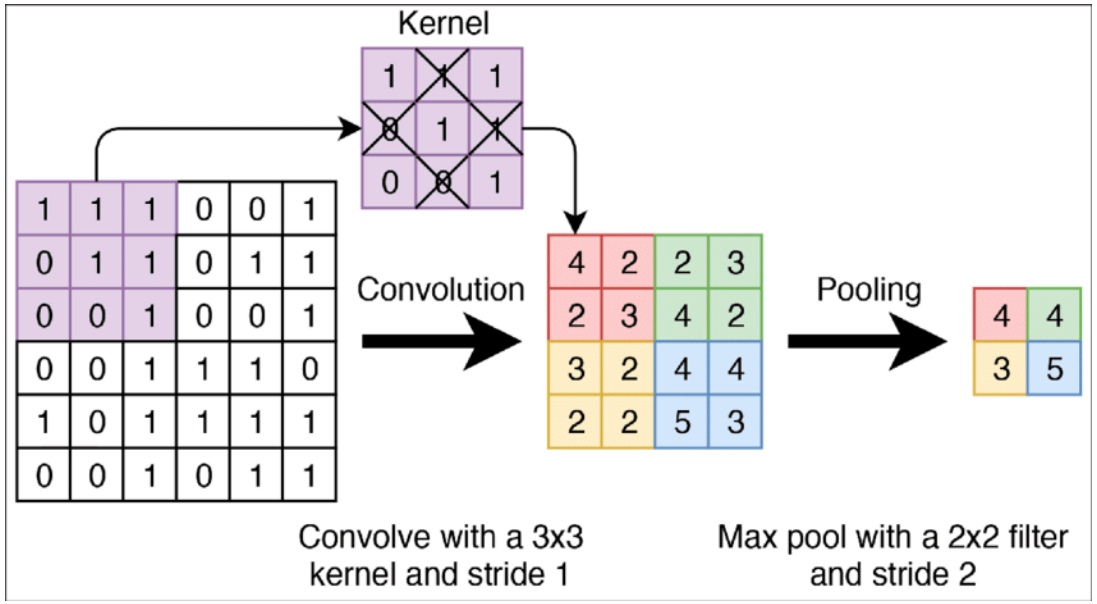
Convolution is defined by a **kernel**

There are different types of pooling, namely **max-pooling** and **average pooling**.

In Max-pooling we take the maximum over the block as an output.

In average pooling, we average the pixels.

This reduces the spatial resolution (number of neurons).



SUMMARY OF CONVOLUTIONAL NEURAL NETWORKS PART

Vanilla convolution for images is parametrized by a 4-dimensional tensor:

$W_1 \times W_2 \times C_{in} \times C_{out}$, i.e. the size of the convolution, the number of input channels, and the number of output channels.

These parameters are learned in a supervised way.

Convolutions use the locality of the images.

Non-local dependencies are obtained from **pooling**

MODIFICATION OF CNN

There were several modifications of CNN, including:

— Residual connections: instead of $y_{k+1} = F(y_k)$ we write a layer as $y_{k+1} = y_k + F(y_k)$. This allows training of deeper networks

— Special convolutions with fewer number of parameters

— Specially designed basic computational blocks.

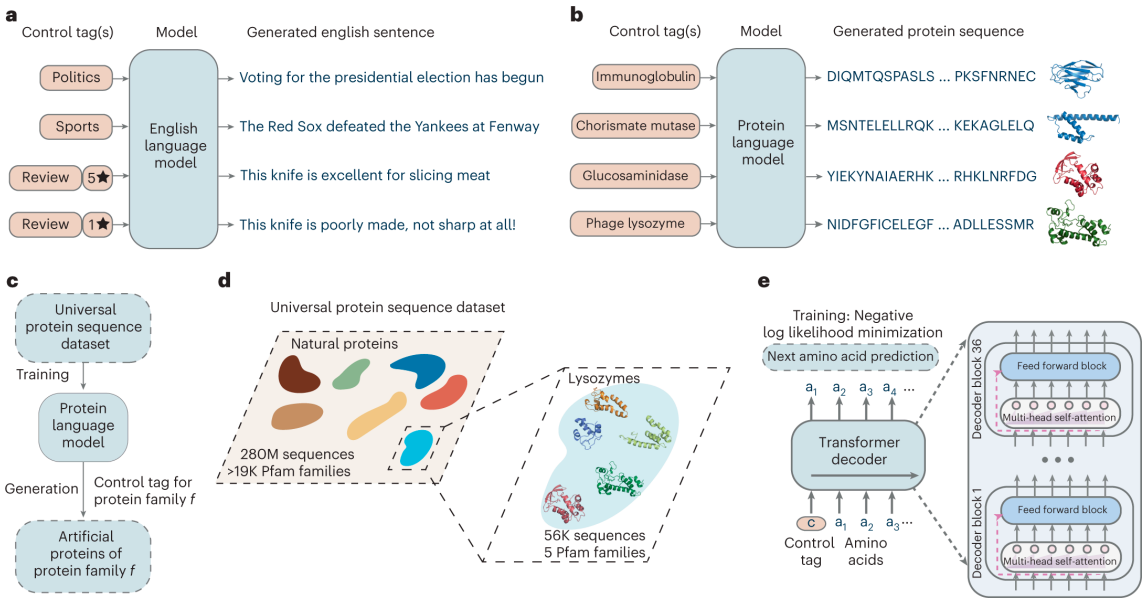
— Current good choices for image classification is a ResNet network, or EfficientNet network.

WORKING WITH SEQUENCES

— Neural networks can be applied for working with sequences, such as text or DNA sequences

— Classical approaches include recurrent neural network and their variants.

— SOTA (state-of-the art) approaches use transformer-based models.



RECAP OF LECTURE 7

— Fully connected neural network

— Learning

— Backpropagation

— Types of NN architectures (brief summary)

RECENT ADVANCES IN NEURAL NETWORKS

- Deep learning and its impact on neural networks
- Generative adversarial networks (GANs)
- Autoencoders and their applications
- Transfer learning and pre-trained models

NEXT LECTURE

Scalable algorithms