Advanced Learning Models

Chapter I - Introduction, Deep Learning and Multilayer Perceptron

Julien Mairal & Xavier Alameda-Pineda

with the help of Jakob Verbeek and Laurent Besacier

MSIAM/MoSIG – 2019-2020

Table of Today's Contents

Course Organisation

Course Organisation

Goal

Introduce two major paradigms in machine learning called kernel methods (taught by Julien) and neural networks (taught by Xavi).

Ressources

You can visit (often) the web page of the course

`http://lear.inrialpes.fr/people/mairal/teaching/2018-2019/MSIAM/`

Grading

Homework (once, 30%), Data Challenge (30%) and final exam (40%).
Data challenge can be done in teams of two students.

Principles of Machine Learning

Common strategy: optimisation (for machine learning)

Optimisation is central in machine learning. As an example, supervised learning consists on estimating a **prediction function** $f : \mathcal{X} \to \mathcal{Y}$ given a set of labeled training data $\{(x_n, y_n)\}_{n=1}^N$, $x_n \in \mathcal{X}$ and $y_n \in \mathcal{Y}$:

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{N} \sum_{n=1}^{N} L(y_n, f(x_n))}_{\text{Empirical Risk}} + \underbrace{\lambda \Omega(f)}_{\text{Regularisation}}$$

Common strategy: optimisation (for machine learning)

Optimisation is central in machine learning. As an example, supervised learning consists on estimating a **prediction function** $f : \mathcal{X} \to \mathcal{Y}$ given a set of labeled training data $\{(x_n, y_n)\}_{n=1}^{N}$, $x_n \in \mathcal{X}$ and $y_n \in \mathcal{Y}$:

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{N} \sum_{n=1}^{N} L(y_n, f(x_n))}_{\text{Empirical Risk}} + \underbrace{\lambda \Omega(f)}_{\text{Regularisation}}$$

- $\mathcal{X}$ is the data space and x_n are the data points (visual features, bag-of-words, cepstral coefficients, raw images).

Common strategy: optimisation (for machine learning)

Optimisation is central in machine learning. As an example, supervised learning consists on estimating a **prediction function** $f : \mathcal{X} \to \mathcal{Y}$ given a set of labeled training data $\{(x_n, y_n)\}_{n=1}^N$, $x_n \in \mathcal{X}$ and $y_n \in \mathcal{Y}$:

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{N} \sum_{n=1}^{N} L(y_n, f(x_n))}_{\text{Empirical Risk}} + \underbrace{\lambda \Omega(f)}_{\text{Regularisation}}$$

- $\mathcal{X}$ is the data space and x_n are the data points (visual features, bag-of-words, cepstral coefficients, raw images).
- $\mathcal{Y}$ is the label space and y_n are the labels ($\mathcal{Y}$ could be $\{-1, 1\}$, $\{1, \ldots, K\}$ for binary/multiclass class., $\mathbb{R}$, $\mathbb{R}^K$) for regression).

Common strategy: optimisation (for machine learning)

Optimisation is central in machine learning. As an example, supervised learning consists on estimating a **prediction function** $f : \mathcal{X} \to \mathcal{Y}$ given a set of labeled training data $\{(x_n, y_n)\}_{n=1}^N$, $x_n \in \mathcal{X}$ and $y_n \in \mathcal{Y}$:

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{N} \sum_{n=1}^{N} L(y_n, f(x_n))}_{\text{Empirical Risk}} + \underbrace{\lambda \Omega(f)}_{\text{Regularisation}}$$

- $\mathcal{X}$ is the data space and x_n are the data points (visual features, bag-of-words, cepstral coefficients, raw images).
- $\mathcal{Y}$ is the label space and y_n are the labels ($\mathcal{Y}$ could be $\{-1, 1\}$, $\{1, \ldots, K\}$ for binary/multiclass class., $\mathbb{R}$, $\mathbb{R}^K$) for regression).
- L is the loss assessing the *dissimilarity* between y_n and $f(x_n)$.

Common strategy: optimisation (for machine learning)

Optimisation is central in machine learning. As an example, supervised learning consists on estimating a **prediction function** $f : \mathcal{X} \to \mathcal{Y}$ given a set of labeled training data $\{(x_n, y_n)\}_{n=1}^N$, $x_n \in \mathcal{X}$ and $y_n \in \mathcal{Y}$:

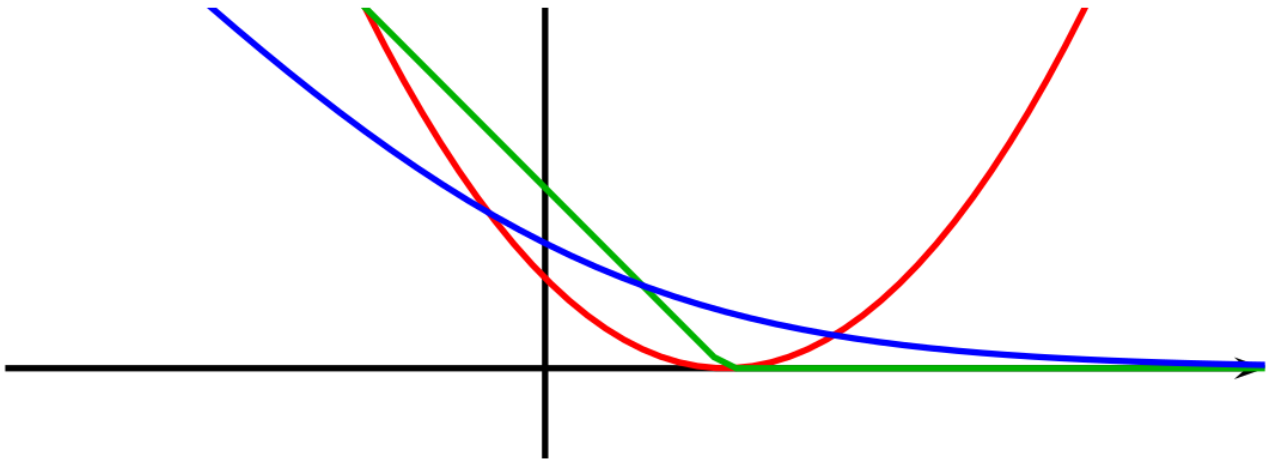
$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{N} \sum_{n=1}^{N} L(y_n, f(x_n))}_{\text{Empirical Risk}} + \underbrace{\lambda \Omega(f)}_{\text{Regularisation}}$$

- $\mathcal{X}$ is the data space and x_n are the data points (visual features, bag-of-words, cepstral coefficients, raw images).
- $\mathcal{Y}$ is the label space and y_n are the labels ($\mathcal{Y}$ could be $\{-1, 1\}$, $\{1, \ldots, K\}$ for binary/multiclass class., $\mathbb{R}$, $\mathbb{R}^K$) for regression).
- L is the loss assessing the *dissimilarity* between y_n and $f(x_n)$.
- Ω is the regulariser, ensuring that the optimal f is *reasonable*.

Examples

Examples of linear models on a P-dimensional feature space:

- Linear SVM: $\min_{w \in \mathbb{R}^P} \frac{1}{N} \sum_{n=1}^{N} \max(0, 1 - y_n w^\top x_n) + \lambda \|w\|_2^2.$
- Ridge: $\min_{w \in \mathbb{R}^P} \frac{1}{N} \sum_{n=1}^{N} \frac{1}{2}(y_n - w^\top x_n)^2 + \lambda \|w\|_2^2.$
- Logistic: $\min_{w \in \mathbb{R}^P} \frac{1}{N} \sum_{n=1}^{N} \log(1 + e^{(-y_n w^\top x_n)}) + \lambda \|w\|_2^2.$



Overall strategy

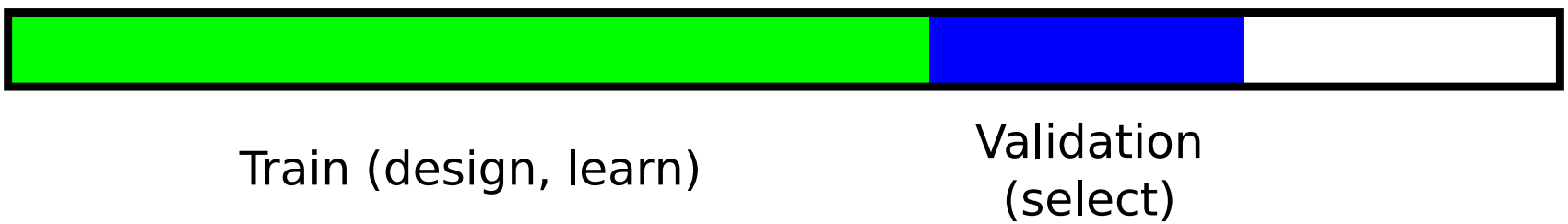
The previous formulation is called *empirical risk minimisation*:

- **Observe** the world, i.e. gather (and annotate) data.

Overall strategy

The previous formulation is called *empirical risk minimisation*:

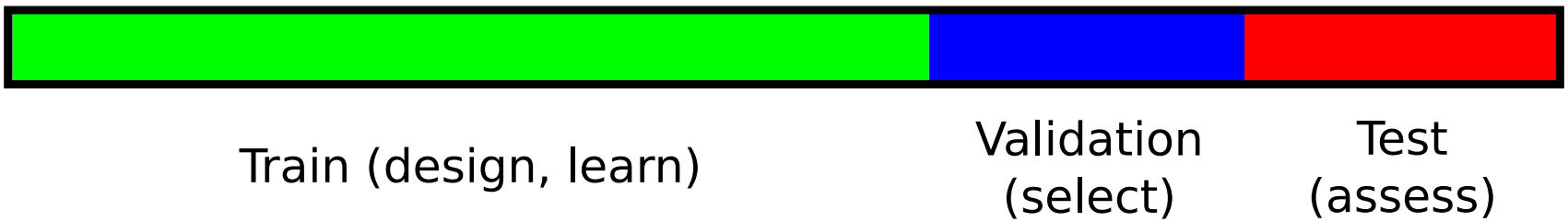
- **Observe** the world, i.e. gather (and annotate) data.
- **Model** the observed data. Design, learn and select the model.



Overall strategy

The previous formulation is called *empirical risk minimisation*:

- **Observe** the world, i.e. gather (and annotate) data.
- **Model** the observed data. Design, learn and select the model.
- **Assess** the quality of the model on **test** data.



Overall strategy

The previous formulation is called *empirical risk minimisation*:

- **Observe** the world, i.e. gather (and annotate) data.
- **Model** the observed data. Design, learn and select the model.
- **Assess** the quality of the model on **test** data.

Train (design, learn) | Validation (select) | Test (assess)

Test on NEW data is very important to quantify the generalisation error!!!

General principle

This principle is valid (and required) for a wide variety of tasks, and methods, namely: neural networks, kernel methods, etc.

Overall strategy: comments

Not so simple

Even linear models lead to challenging problems in optimization:

- algorithms that scale both in the problem size N and dimension P;
- are able to exploit the problem structure;
- come with statistical, convergence and numerical stability guarantees.

Overall strategy: comments

Not so simple

Even linear models lead to challenging problems in optimization:

- algorithms that scale both in the problem size N and dimension P;
- are able to exploit the problem structure;
- come with statistical, convergence and numerical stability guarantees.

Usable beyond supervised learning

$$\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{n=1}^{N} L(f(x_n)) + \lambda \Omega(f)$$

- L is not a classification loss;
- Example of paradigms: K-means, PCA, MoG, matrix factorisation.

ALM Paradigm 1: Deep neural network

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{N} \sum_{n=1}^{N} L(y_n, f(x_n))}_{\text{Empirical Risk}} + \underbrace{\lambda \Omega(f)}_{\text{Regularisation}}$$

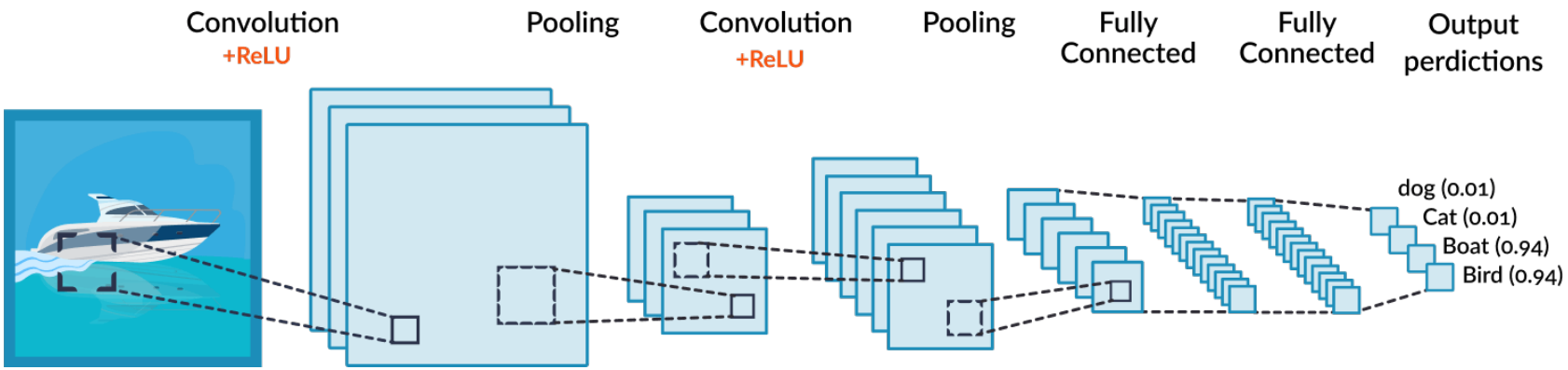
- In deep neural network, the prediction function space $\mathcal{F}$ is a combination of linear operators A_k and non-linear operators σ_k:

$$f(x) = \sigma_K(A_K \sigma_{K-1}(A_{K-1} \cdots \sigma_2(A_2 \sigma_1(A_1 x)) \cdots)).$$

- Optimising for $A_1, \ldots A_K$ yields an intractable, non-convex optimisation problem in huge dimension (millions!).
- Linear operation $A_1, \ldots A_K$ can be unconstrained (fully connected) or share some parameters (convolutions).

Paradigm 1: Deep Neural Networks

Example: convolutional neural network:



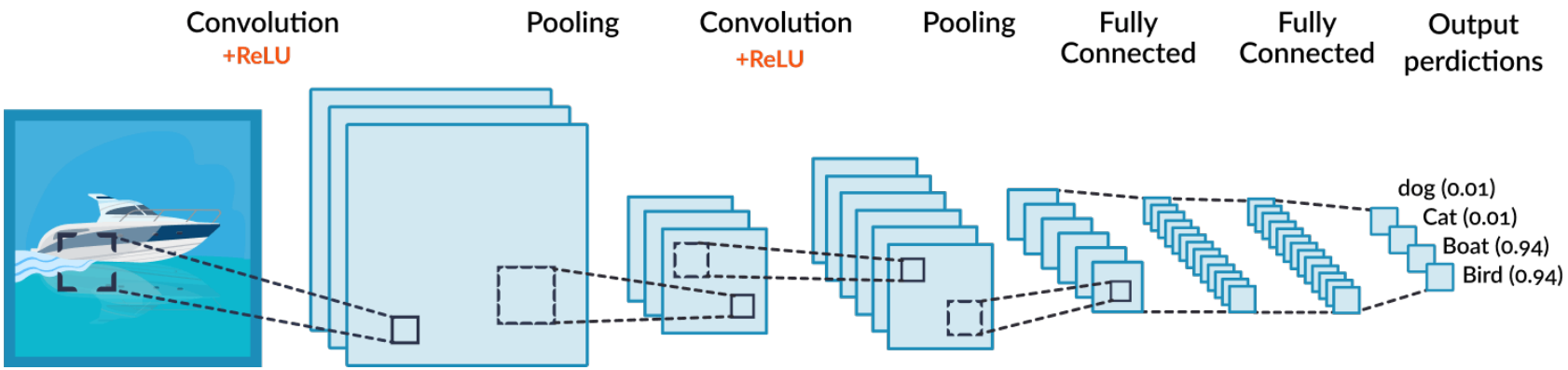
Main properties of CNNs:

- Multi-scale feature extraction;
- Invariant to certain basic transformations;
- Model the local stationarity (at several scales);
- State-of-the-art in many tasks/fields.

Image from `https://missinglink.ai/`.

Paradigm 1: Deep Neural Networks

Example: convolutional neural network:



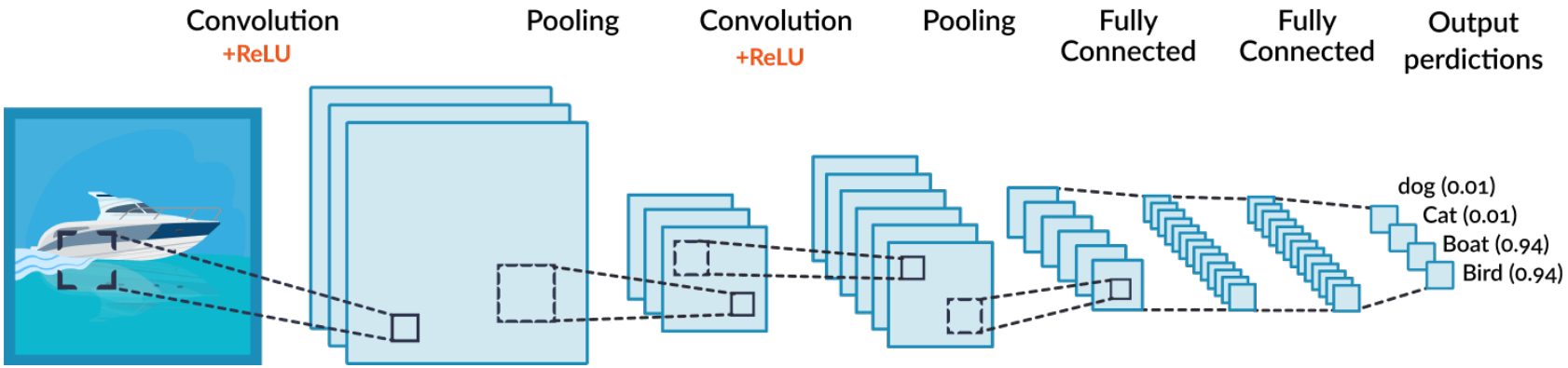
Main open problems:

- Little theoretical understanding;
- Huge amounts of labeled data (millions!);
- Manual design and parameter tuning/selection;
- Unclear how to regularise.

Image from `https://missinglink.ai/`.

Paradigm 1: Deep Neural Networks

Example: convolutional neural network:



Main advantages (for you):

- Huge academic and industrial effort;
- Many open source libraries (pytorch, keras, tensorflow, etc);
- Lots of researchers publish their models and their weights (reproducibility/reuse);
- Anyone with a GPU and some programing basics can run/re-train/modify/update a CNN.

Image from `https://missinglink.ai/`.

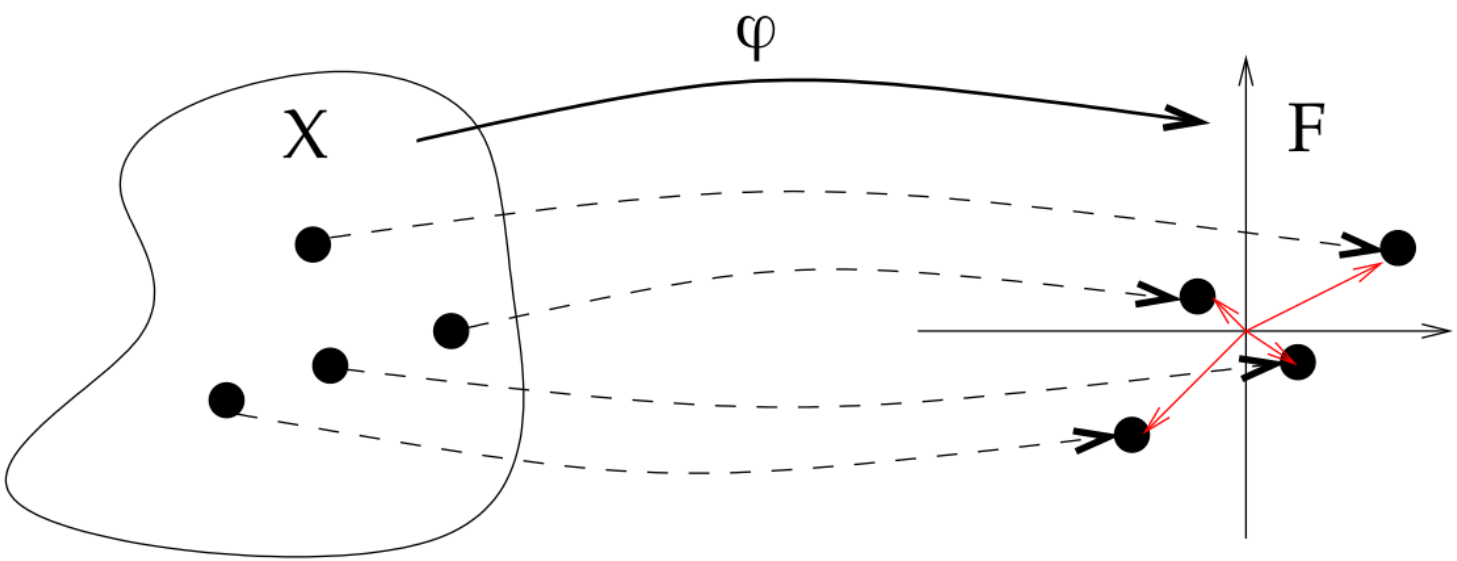
Paradigm 2: Kernel Methods

Map the data into a hilbert space $\mathcal{H}$

$$\min_{f \in \mathcal{H}} \frac{1}{N} \sum_{n=1}^{N} L(y_n, f(x_n)) + \lambda \|f\|_{\mathcal{H}},$$

and then work with linear forms:

$$\varphi : \mathcal{X} \to \mathcal{H} \qquad \text{and} \qquad f(x) = \langle \varphi(x), f \rangle_{\mathcal{H}}, \quad f \in \mathcal{H}.$$



Paradigm 2: Kernel Methods

Map the data into a hilbert space $\mathcal{H}$

$$\min_{f \in \mathcal{H}} \frac{1}{N} \sum_{n=1}^{N} L(y_n, f(x_n)) + \lambda \|f\|_{\mathcal{H}},$$

and then work with linear forms:

$$\varphi : \mathcal{X} \to \mathcal{H} \qquad \text{and} \qquad f(x) = \langle \varphi(x), f \rangle_{\mathcal{H}}, \quad f \in \mathcal{H}.$$

The main purpose is to **embed** the data in a vector space, where many **geometrical operations** are well-defined (angle, barycenters, projections). These includes infinite-dimensional vector spaces!

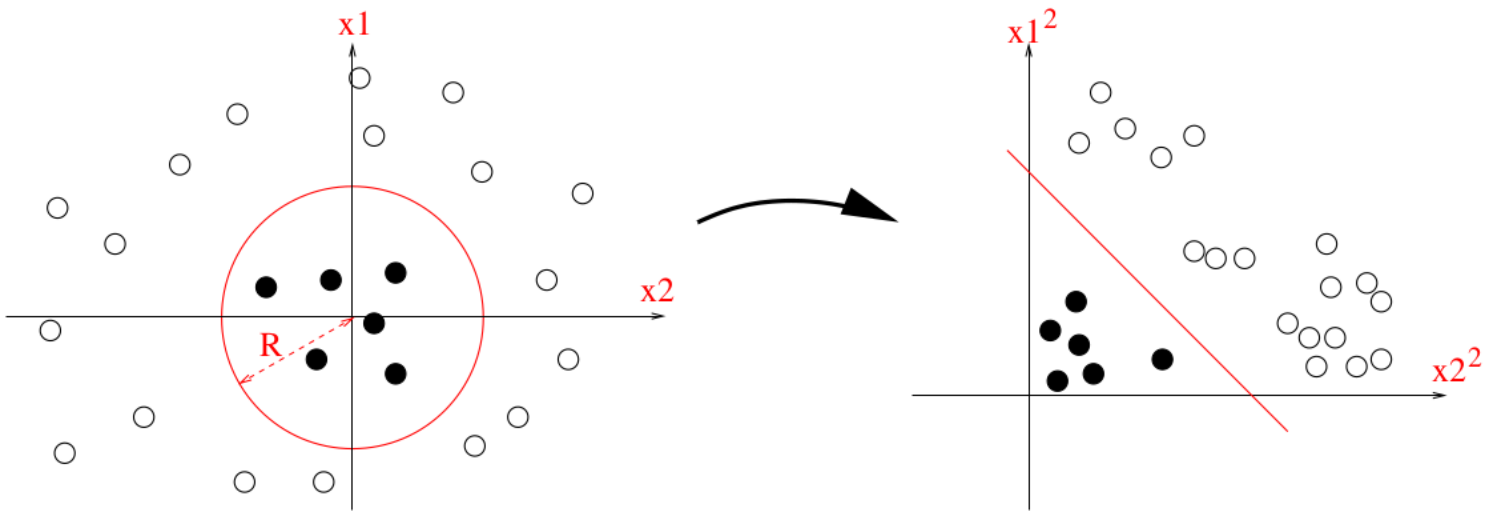
This priciple is generic and **assumes nothing** about the original feature space $\mathcal{X}$ (vectors, graphs, meshes, sequences, sets, etc).

Paradigm 2: Kernel Methods (II)

Kernel methods are very useful to

(i) compare non-vector data as if they were and (ii) compare vector data in a structure different than the Euclidean space:

- lift-up the data into a higher dimensional space where we hope data will be e.g. linearly separable, nice clusters;
- this is because the linear form $f(x) = \langle \varphi(x), f \rangle_{\mathcal{H}}$ in $\mathcal{H}$ may correspond to a non-linear model in $\mathcal{X}$.

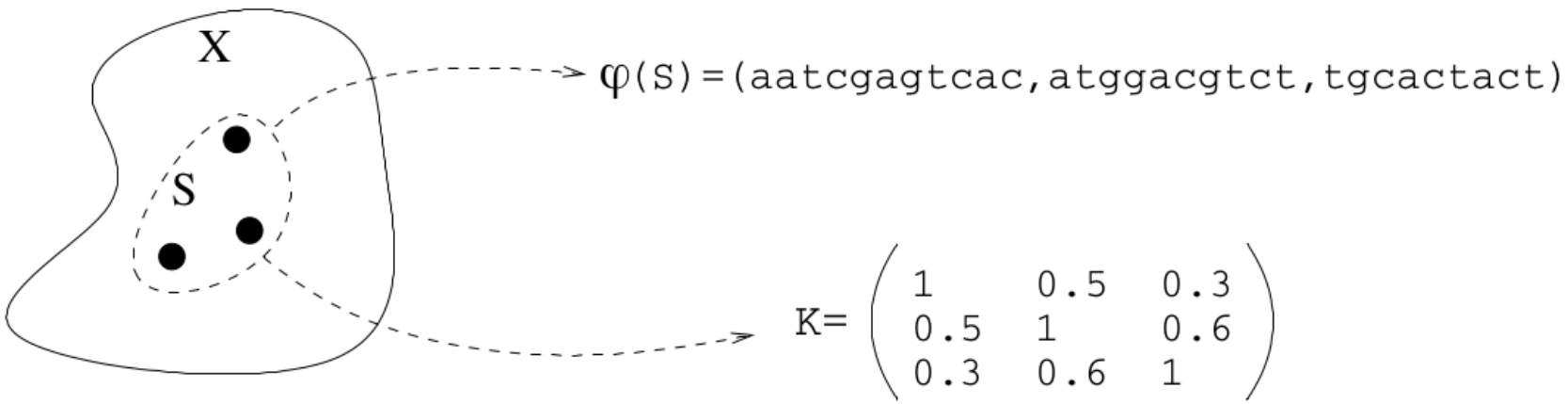


Paradigm 2: Kernel Methods (III)

How does it work? Pairwise comparisons

- Define a comparison function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.
- Represent a set of N data points $\mathcal{S} = \{x_1, \ldots, x_N\}$ by the $N \times N$ matrix of pair-wise comparisons $\mathbf{K}$:

$$\mathbf{K} \in \mathbb{R}^{N \times N} \qquad \mathbf{K}_{ij} = K(x_i, x_j).$$



Paradigm 2: Kernel Methods (IV)

Theorem (Aronszajn, 1950)

$K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive definite kernel if and only if there exists a Hilbert space $\mathcal{H}$ and a mapping $\varphi : \mathcal{X} \to \mathcal{H}$, such that:

$$K(x, x') = \left\langle \varphi(x), \varphi(x') \right\rangle_{\mathcal{H}}, \qquad \forall x, x' \in \mathcal{X}.$$

Paradigm 2: Kernel Methods (IV)

Theorem (Aronszajn, 1950)

$K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive definite kernel if and only if there exists a Hilbert space $\mathcal{H}$ and a mapping $\varphi : \mathcal{X} \to \mathcal{H}$, such that:

$$K(x, x') = \left\langle \varphi(x), \varphi(x') \right\rangle_{\mathcal{H}}, \qquad \forall x, x' \in \mathcal{X}.$$

RKHS

$\mathcal{H}$ is called the **reproducing kernel Hilbert space** (RKHS). Intuitively, for any $x \in \mathcal{X}$ we can build $f_x \in \mathcal{H}$ so that:

$$f_x : \mathcal{X} \to \mathbb{R} \quad \text{(by definition) and} \quad f_x(y) = K(x, y).$$

The mapping $x \to f_x$ is the mapping φ of the previous theorem.

Paradigm 2: Kernel Methods – Summary

Main features

- builds well-studied functional spaces to do machine learning;
- decoupling of data representation and learning algorithm;
- often convex optimization problems with natural regularisation;
- versatility: applies to vectors, sequences, graphs, sets, etc.

Paradigm 2: Kernel Methods – Summary

Main features

- builds well-studied functional spaces to do machine learning;
- decoupling of data representation and learning algorithm;
- often convex optimization problems with natural regularisation;
- versatility: applies to vectors, sequences, graphs, sets, etc.

Main limitations

- decoupling of data representation and learning may not be good;
- requires kernel design;
- has scalability problems.

More on this with Julien Mairal on Dec 12th, Jan 9th and 16th.

Deep Learning: Overview

What is Deep Learning?

Deep Learning is a field of machine learning, aiming to **learn representations** of data tailored to the problem at hand.

The conception of the model depends on the task to be solved, but the philosophy of deep learning is to process **raw data** (e.g. images, sounds).

It can be used in supervised, unsupervised and reinforcement learning. Supervised $\leftrightarrow$ annotated labels, unsupervised $\leftrightarrow$ no annotated labels, reinforcement $\leftrightarrow$ learning a sequence of actions to maximise a reward.

Why is it called Deep Learning and Deep Neural Networks?

Neural

As we will see later, the basic unit is called (artificial) neuron, and its design is inspired by the way brain cells function. Several inputs (scalars) are weighted and added, to trigger (or not) the single output (scalar).

Deep Learning and Deep Neural Networks

Neural

As we will see later, the basic unit is called (artificial) neuron, and its design is inspired by the way brain cells function. Several inputs (scalars) are weighted and added, to trigger (or not) the single output (scalar).

Deep Network

These neurons are grouped in layers, in different structures. To construct a network, we stack K layers on top of each other.

Weighted sum $\leftrightarrow$ linear operator (A), trigger $\leftrightarrow$ non-linear function (σ).

$$f(x) = \sigma_K(A_K\sigma_{K-1}(A_{K-1}\cdots\sigma_2(A_2\sigma_1(A_1x))\cdots)).$$

Deep Learning and Deep Neural Networks

Neural

As we will see later, the basic unit is called (artificial) neuron, and its design is inspired by the way brain cells function. Several inputs (scalars) are weighted and added, to trigger (or not) the single output (scalar).

Deep Network

These neurons are grouped in layers, in different structures. To construct a network, we stack K layers on top of each other.

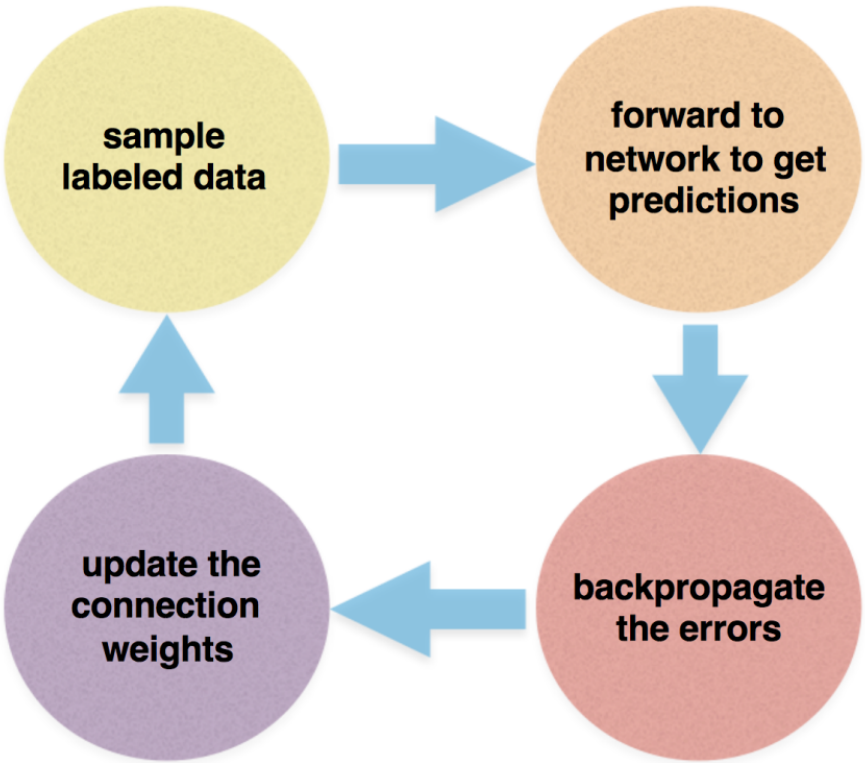
Weighted sum $\leftrightarrow$ linear operator (A), trigger $\leftrightarrow$ non-linear function (σ).

$$f(x) = \sigma_K(A_K\sigma_{K-1}(A_{K-1}\cdots\sigma_2(A_2\sigma_1(A_1x))\cdots)).$$

Learning

Once we have decided the structure, we need to devise a way to optimise for the parameters of the linear operators: $A_1, \ldots, A_K$.

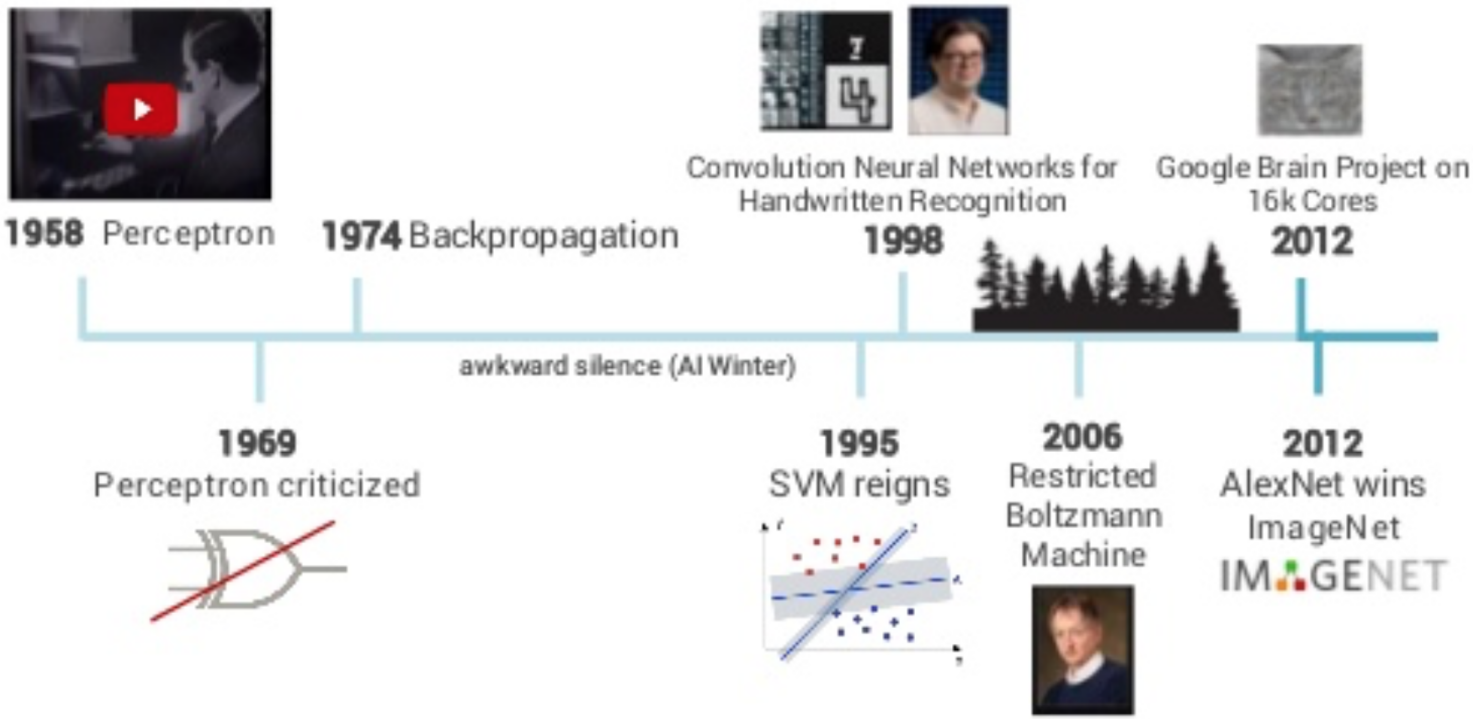
How to learn?



1. Sample a batch of annotated data.
2. Forward the network to predict, $f(x)$.
3. Compute the prediction error for back-propagation.
4. Update the weights by back-propagation.

We first generate an **error signal** measuring the difference between predictions and ground-truth (annotations). This signal is used to **update the weights** of the network.

A bit of history



In 2012 there was a three-fold break-through: data (ImageNet), computation (GPU) and architectures.

Image from `https://www.slideshare.net/LuMa921/deep-learning-a-visual-introduction`.

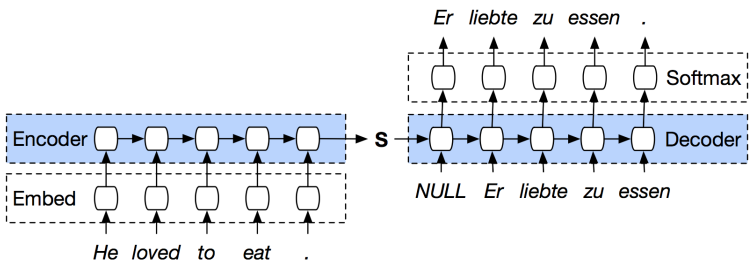
Success stories (I): convolutional neural networks

- For stationary signals such as audio, images, and video.
- Applications: object detection, image retrieval, pose estimation.

Image from He et al. "Mask R-CNN" in ICCV 2017.

Success stories (II): recurrent neural networks

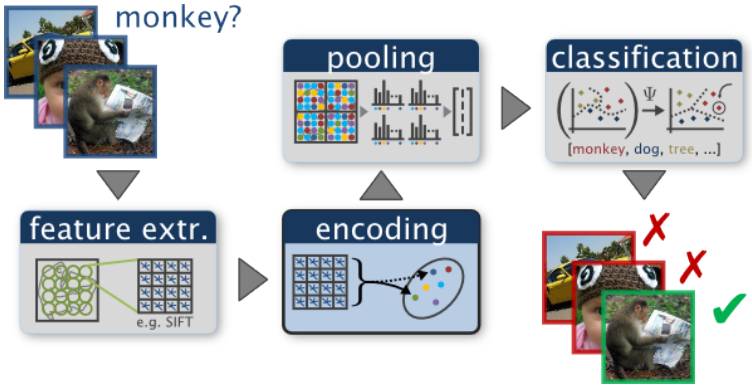
- For variable length sequence data, e.g. in natural language.
- Applications: sequence to sequence prediction (machine translation, speech recognition).



Input sentence:	Translation (PBMT):	Translation (GNMT):	Translation (human):
李克強此行將啟動中加總理年度對話機制，與加拿大總理杜魯多舉行兩國總理首次年度對話。	Li Keqiang premier added this line to start the annual dialogue mechanism with the Canadian Prime Minister Trudeau two prime ministers held its first annual session.	Li Keqiang will start the annual dialogue mechanism with Prime Minister Trudeau of Canada and hold the first annual dialogue between the two premiers.	Li Keqiang will initiate the annual dialogue mechanism between premiers of China and Canada during this visit, and hold the first annual dialogue with Premier Trudeau of Canada.

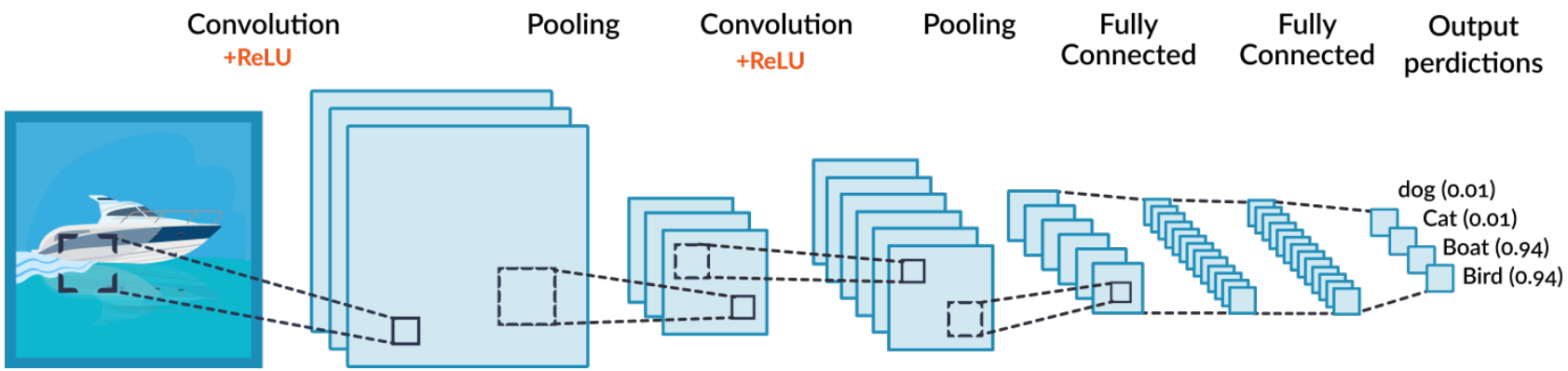
Images from: `https://smerity.com/media/images/articles/2016/` and `http://www.zdnet.com/article/google-announces-neural-machine-translation-to-improve-google-translate/`.

Where does the success come from?



Classic approach

- Feature extraction **engineered**
- Feature aggregation **unsupervised**
- Recognition **supervised**



Deep learning approach

- There is continuous between feature extraction and recognition.
- All information processing steps are jointly learned to optimise a task-dependent loss.

And at what cost?

- **Annotation** of millions of images by hand is quite tedious. Gathering such large-scale data-set for every new task is unfeasible. There are approaches to address that and mitigate that effect.
- **Storage** of these data: need of disk space, data servers, data transmission, etc.
- **Pre-processing** of these data points to satisfy the expected input (in terms of size, shape, format, statistical distribution, etc).
- **Computational** cost of learning the model parameters (hours, days or weeks on multi-GPU servers). Big tech companies have GPU *farms*.

So what is the **advantage**?

And at what cost?

- **Annotation** of millions of images by hand is quite tedious. Gathering such large-scale data-set for every new task is unfeasible. There are approaches to address that and mitigate that effect.
- **Storage** of these data: need of disk space, data servers, data transmission, etc.
- **Pre-processing** of these data points to satisfy the expected input (in terms of size, shape, format, statistical distribution, etc).
- **Computational** cost of learning the model parameters (hours, days or weeks on multi-GPU servers). Big tech companies have GPU *farms*.

So what is the **advantage**? Performance!

CNN for visual data

Intuitive idea: organise neurons spatially as "images" in a 2D grid.

Since 2012's AlexNet trained in ImageNet (10^6 images for 10^3 classes):

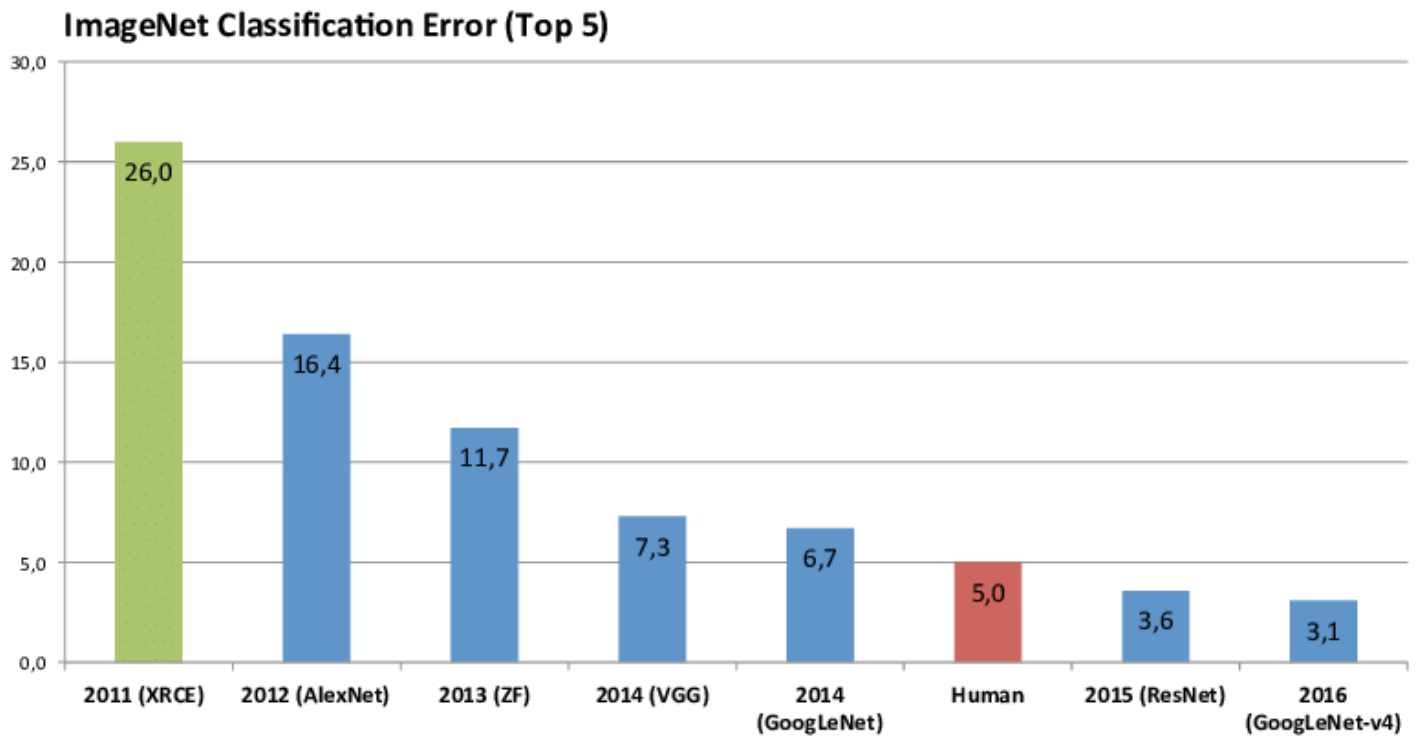
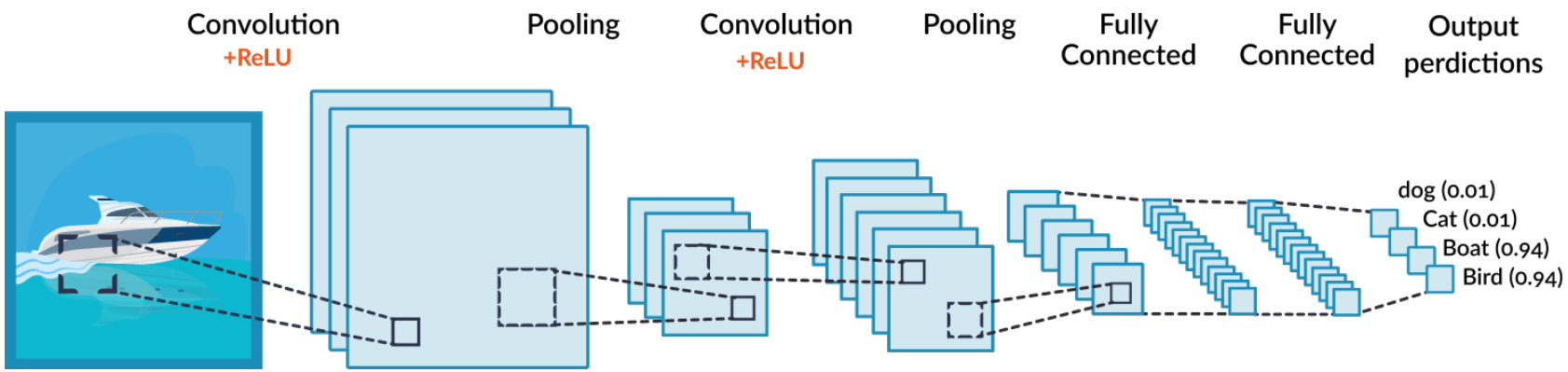


Image from `https://www.embedded-vision.com/`.

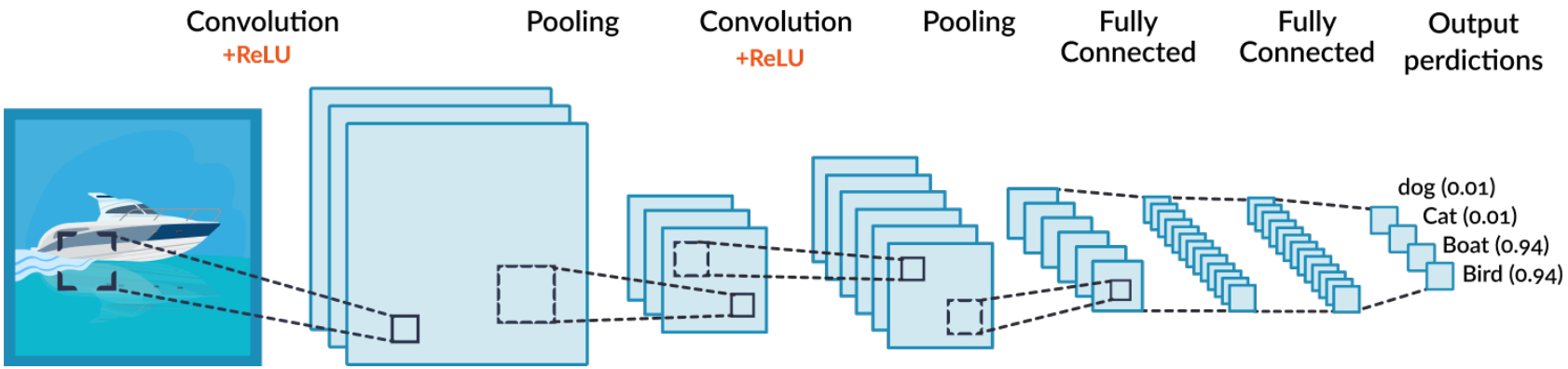
CNN for visual data

Intuitive idea: organise neurons spatially as "images" in a 2D grid.

- Convolution computes activations from one layer to next
  - Translation invariant (stationary signal)
  - Local connectivity (fast to compute)
  - # of parameters decoupled from input size (generalization)
- Pooling layers down-sample the signal every few layers
  - Multi-scale pattern learning
  - Degree of translation invariance



CNN: Hierarchical representations



Higher layers represent more abstract concepts!!!

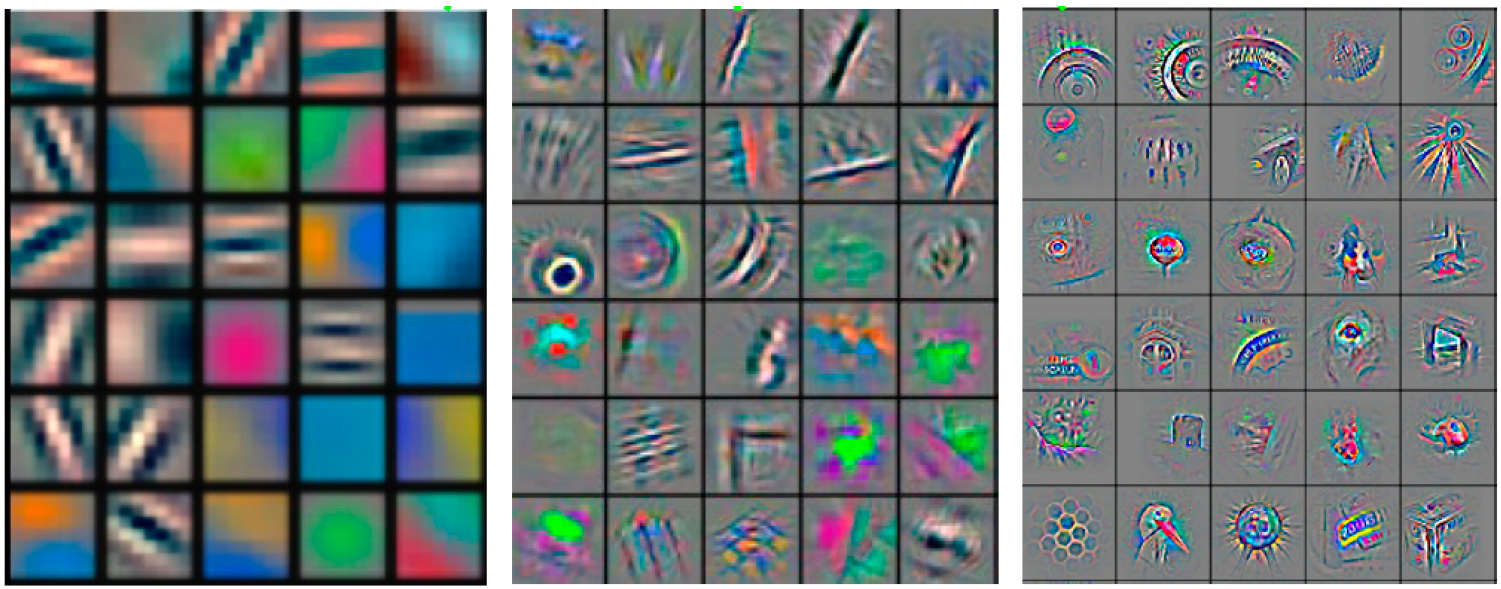


Image from M. Zeiler and Rob Fergus "Visualizing and understanding convolutional networks" ECCV 2014.

CNN: Applications

1. Image classification (all started with that).
   - Object-class or face-identity recognition.
   - Deeper and deeper networks (2012: 8 layers, now 100+ layers).
   - Other architectures: residual networks, skip connections.
   - The models trained on image classification, can be used for other tasks.

CNN: Applications

1. Image classification (all started with that).
   - Object-class or face-identity recognition.
   - Deeper and deeper networks (2012: 8 layers, now 100+ layers).
   - Other architectures: residual networks, skip connections.
   - The models trained on image classification, can be used for other tasks.

2. Object detection.
   - Localise the objects (bounding box, segmentation).
   - Classify the localised objects.

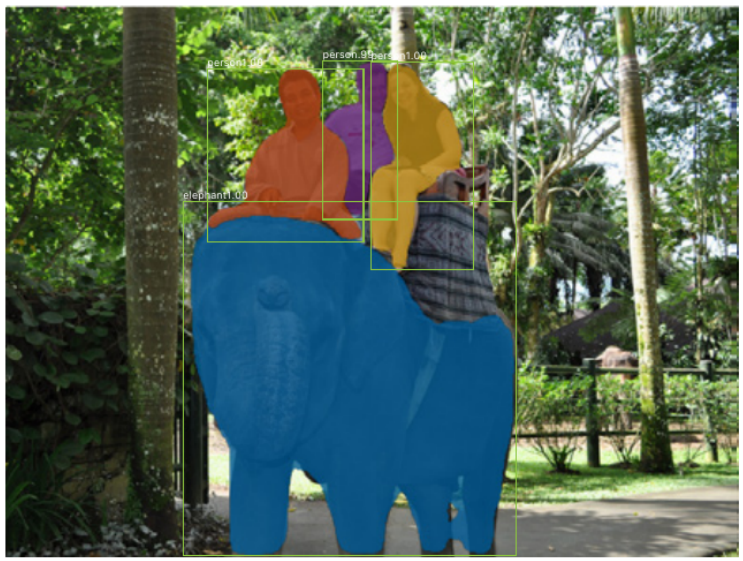


Image from He et al. "Mask R-CNN" in ICCV 2017.

CNN: Applications

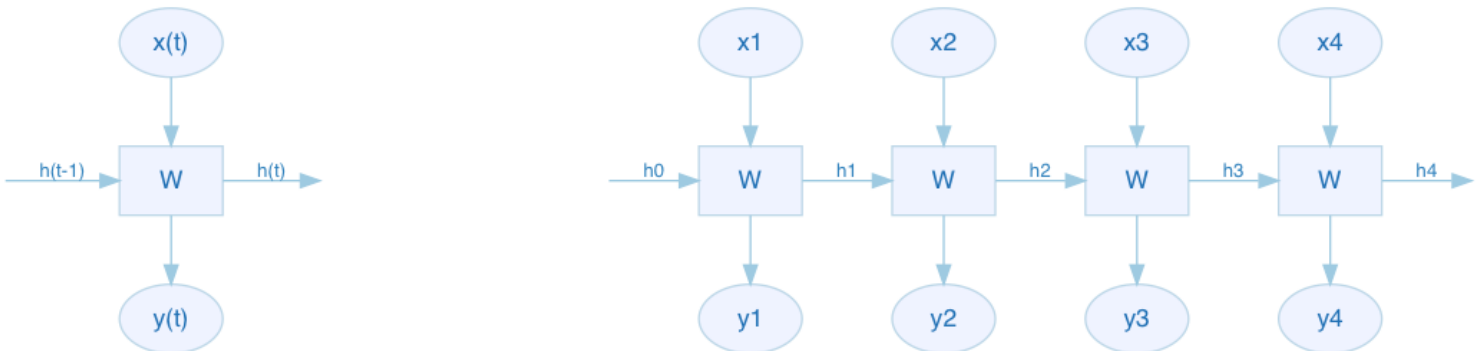
1. Image classification (all started with that).
   - Object-class or face-identity recognition.
   - Deeper and deeper networks (2012: 8 layers, now 100+ layers).
   - Other architectures: residual networks, skip connections.
   - The models trained on image classification, can be used for other tasks.
2. Object detection.
   - Localise the objects (bounding box, segmentation).
   - Classify the localised objects.
3. Scene text detection and reading.
   - Localise and read text.
   - Extreme variability $\rightarrow$ train with combined real/synthetic dataset.

Image from Gupta et al "Synthetic data for text localisation in natural images" In CVPR 2016.

Recurrent neural networks (RNN)

Motivation

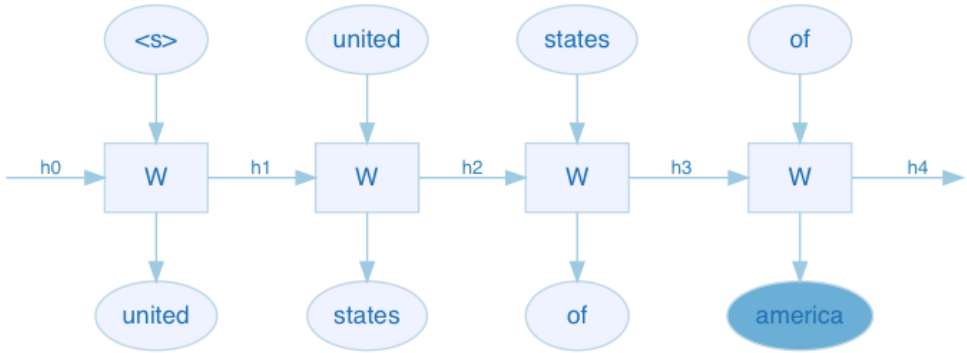
- Not all inputs have fixed length.
- Some problems are inherently "online": information has to be treated upon arrival while keeping track of the past observations.
- Exemplar applications: machine translation, speech recognition.



A RNN treats input $x(t)$ combined with the previous hidden representation $h(t-1)$ to update the hidden representation $h(t)$ and output $y(t)$.

RNNs as generative models

We can train RNN to generate language.



Example of generated text after training on Shakespeare:

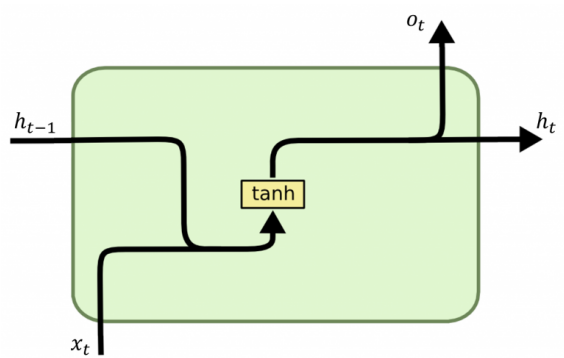
```
KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

Figure from http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

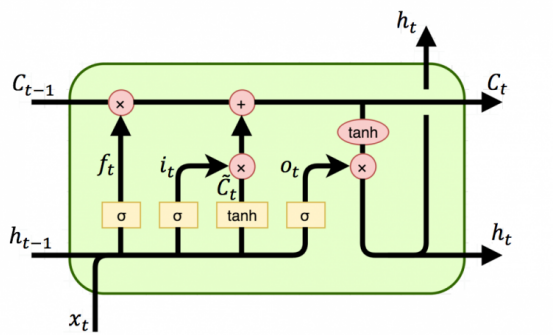
RNNs vanishing gradient

The **vanishing/exploding** gradient problem happens when sequences are too long. The error back-propagated is either too tiny/large to be usable.

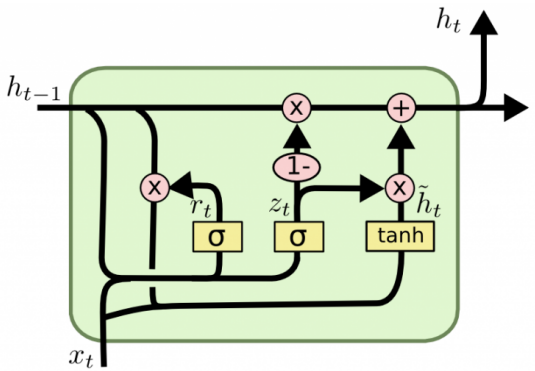
Alternative architectures have been proposed: they learn to remember or forget information through a **gating mechanism**. Examples: long-short term memory (LSTM) or gated recurrent units (GRU):



RNN



LSTM

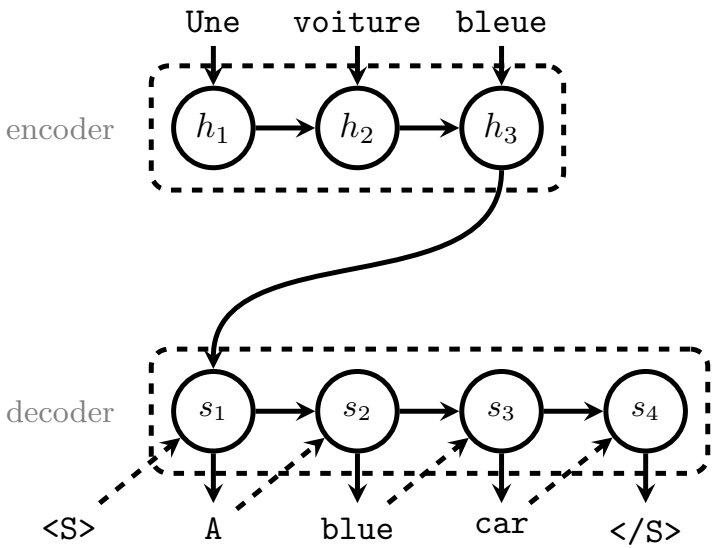


GRU

Figures from `http://dprogrammer.org/`.

RNN Applications

1. Machine Translation
   - End-to-end translation: map input sequence into a fixed vector, then decode target sequence.
   - Most on-line machine translation systems are based on that.



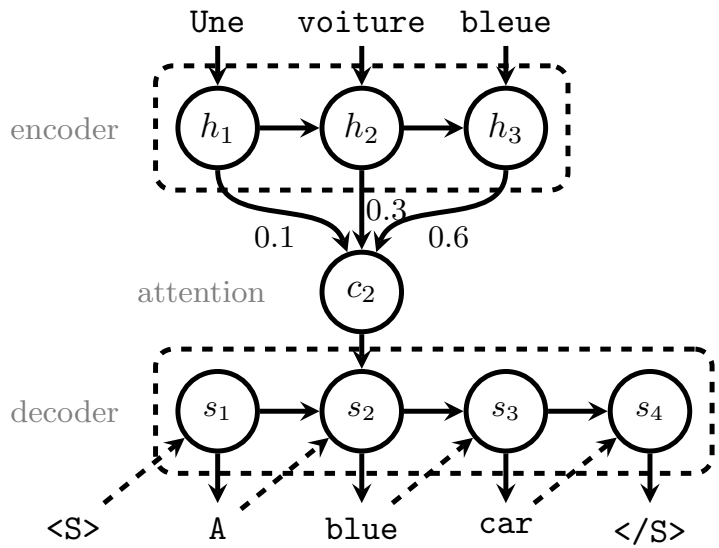


Figure from Alexandre Berard's thesis.

RNN Applications

1. Machine Translation
   - End-to-end translation: map input sequence into a fixed vector, then decode target sequence.
   - Most on-line machine translation systems are based on that.

2. Speech Transcription
   - Architecture similar to neural machine translation.
   - Speech encoder based on CNNs or pyramidal LSTMs.

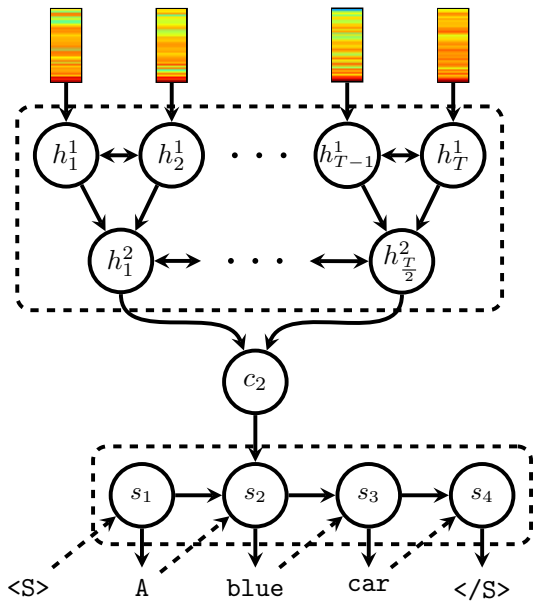


Figure from Alexandre Berard's thesis.

RNN Applications

1. Machine Translation
   - End-to-end translation: map input sequence into a fixed vector, then decode target sequence.
   - Most on-line machine translation systems are based on that.

2. Speech Transcription
   - Architecture similar to neural machine translation.
   - Speech encoder based on CNNs or pyramidal LSTMs.

3. Natural language image description
   - Beyond detection of a fixed set of object categories.
   - Generate word sequence from image data.

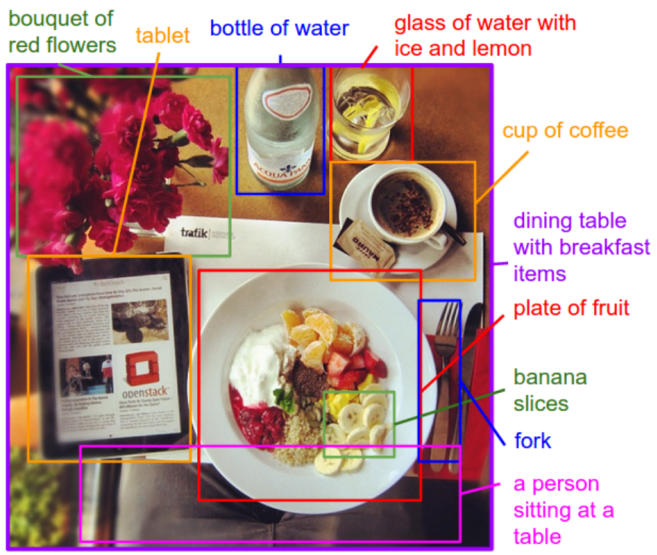


Figure from Karpathy et al "Deep visual-semantic alignments for generating image descriptions" in CVPR 2015.

Wrap-up

- Core idea of deep learning
  - Many processing layers from raw input to output
  - Joint learning of all layers for single objective
- A strategy that is effective across different disciplines
  - Computer vision, speech recognition, natural language processing, game playing, etc.
- Widely adopted in large-scale applications in industry
  - Face tagging on Facebook over 10 9 images per day
  - Speech recognition on iPhone
  - Machine translation at Google, Systran, DeepL, etc.
- Open source development frameworks available (pytorch, tensorflow and the like)
- Limitations: compute and data hungry
  - Parallel computation using GPUs
  - Re-purposing networks trained on large labeled data sets

Research directions (I)

- Optimal architectures and hyper-parameters
  - Possibly under constraints on computing power and memory
  - Hyper-parameters of optimization: learning to learn (meta learning)
- Extend to irregular data such as: (molecular) graphs, 3D meshes, (social) networks, circuits, trees, etc.
- Reduce reliance on supervised data.
  - Un-, semi-, self-, weakly- supervised, etc.
  - Data augmentation and synthesis (e.g. rendered images).
  - Pre-training, multi-task learning.
- Uncertainty and structure in the output space, many possible outputs, structured outputs, etc.

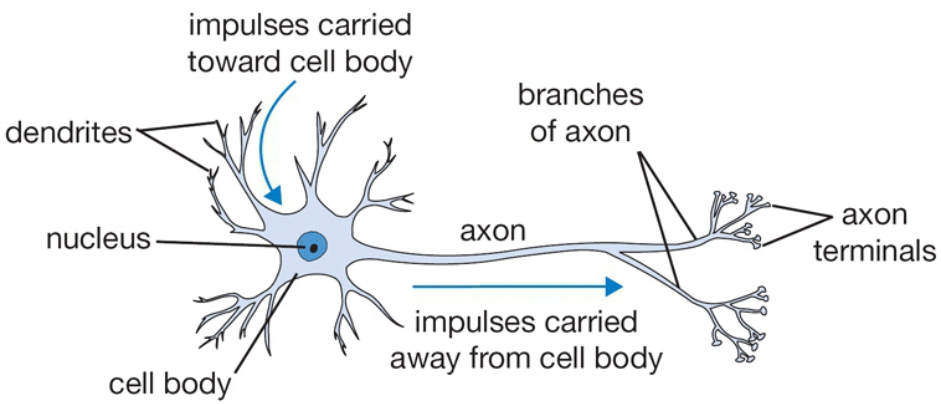
Research directions (II)

- Analyze learned representations
  - Better understanding of black boxes
  - Explainable AI
  - Neural networks to approximate/verify long standing models and theories (link with cognitive sciences)
- Robustness to adversarial examples that fool systems
- Introduce prior knowledge in the model
- Biases issues (gender, skin tone, appearance, etc)
- Common sense reasoning

Deep Learning: Basics

Biological motivation

- The neuron is the basic computational unit of the brain
  About 10^{11} neurons in the human brain.
- Simplified neuron model as a linear threshold unit.
  - Firing rate of electrical spikes modeled as continuous output quantity
  - Connection strength modeled by multiplicative weights
  - Cell activation given by sum of inputs
  - Output is non-linear function of activation
- Basic component in neural circuits for complex tasks



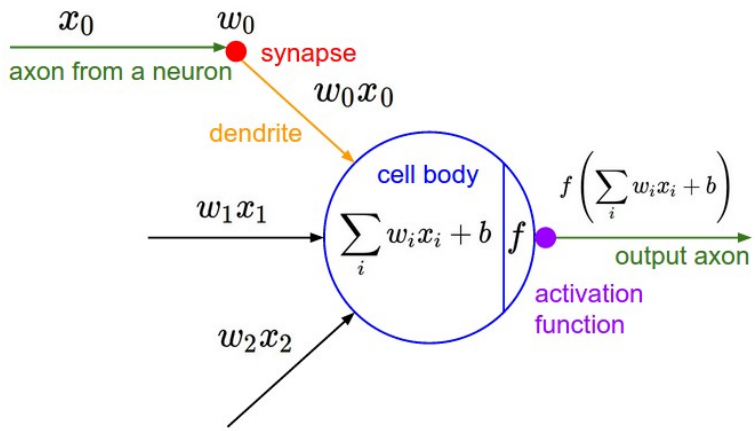


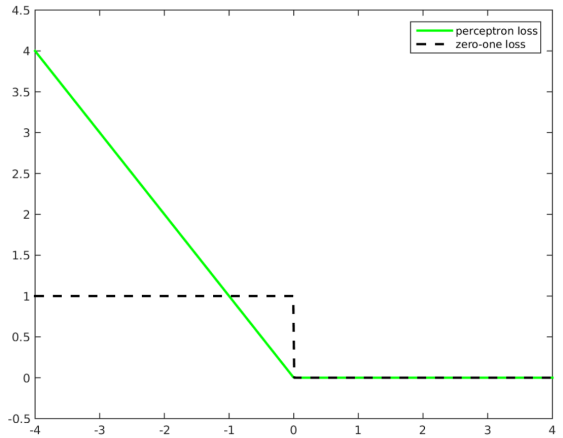
Figure from `http://cs231n.github.io/neural-networks-1/`.

Rosenblatt's Perceptron

- Binary classification based on the sign of a generalised linear function:

$$\text{sign}(f_w(x)) \quad \text{with} \quad f_w(x) = w^\top \phi(x).$$

Gradient is always zero!!! Not good for weight estimation.



- Objective function linear in the score over misclassified patterns:

$$\mathcal{L}(w) = -\sum_{t_i \neq f_w(x_i)} t_i f_w(x_i) = \sum_i \max(0, -t_i f_w(x_i)) \quad t_i \in \{-1, 1\}.$$

- Perceptron learning via stochastic gradient descent at iteration $r+1$:

$$w_{r+1} = w_r + \eta\, t_i\, \phi(x_i)\, [t_i f(x_i) < 0].$$

Let us analyse this equation.

Rosenblatt's Perceptron Weight Update

$$w_{r+1} = w_r + \eta\, t_i\, \phi(x_i)\, [t_i f(x_i) < 0].$$

- One sample "i" at a time $\rightarrow$ stochastic gradient descent (SGD).
  Nowadays we sample a *batch* of data (several points) $\rightarrow$ batch SGD.

Rosenblatt's Perceptron Weight Update

$$w_{r+1} = w_r + \eta\, t_i\, \phi(x_i)\, [t_i f(x_i) < 0].$$

- One sample "i" at a time $\rightarrow$ stochastic gradient descent (SGD). Nowadays we sample a *batch* of data (several points) $\rightarrow$ batch SGD.
- η is the so-called *learning rate*, associated to first-order gradient descent techniques.

Rosenblatt's Perceptron Weight Update

$$w_{r+1} = w_r + \eta\, t_i\, \phi(x_i)\, [t_i f(x_i) < 0].$$

- One sample "i" at a time $\rightarrow$ stochastic gradient descent (SGD). Nowadays we sample a *batch* of data (several points) $\rightarrow$ batch SGD.
- η is the so-called *learning rate*, associated to first-order gradient descent techniques.
- $t_i\, \phi(x)$ is the direction/module of the update.

Rosenblatt's Perceptron Convergence

If a correct solution w^* exists, then the perceptron learning rule will converge to a correct solution in a finite number of iterations.

- Assume all input live in ball of radius M, that w^* has unit norm and it has some margin $t\langle w^*, x\rangle > \delta$.
- Given an initial estimate w_0, its update $w_1 = w_0 + tx$ safistifes: $\langle w^*, w_1\rangle = \langle w^*, w_0\rangle + t\langle w^*, x\rangle > \langle w^*, w_0\rangle + \delta$.
- Moreover, since $t\langle w, x\rangle < 0$ for misclassified samples, we have: $\langle w_1, w_1\rangle = \langle w_0, w_0\rangle + 2t\langle w_0, x\rangle + \langle x, x\rangle < \langle w_0, w_0\rangle + \langle x, x\rangle < \langle w_0, w_0\rangle + M$.
- After r updates: $\langle w^*, w_r\rangle > \langle w^*, w_0\rangle + r\delta$ and $\langle w_r, w_r\rangle < \langle w_0, w_0\rangle + rM$.
- Then (check it!):

$$a(r) := \frac{\langle w^*, w_r\rangle}{\sqrt{\langle w_r, w_r\rangle}} \quad \Rightarrow a(r) \leq 1 \text{ and } a(r) > \frac{\delta\sqrt{r}}{\sqrt{M}} \text{ large } r$$

- $\Rightarrow$ the number of iterations is limited!

Rosenblatt's Perceptron Limitations

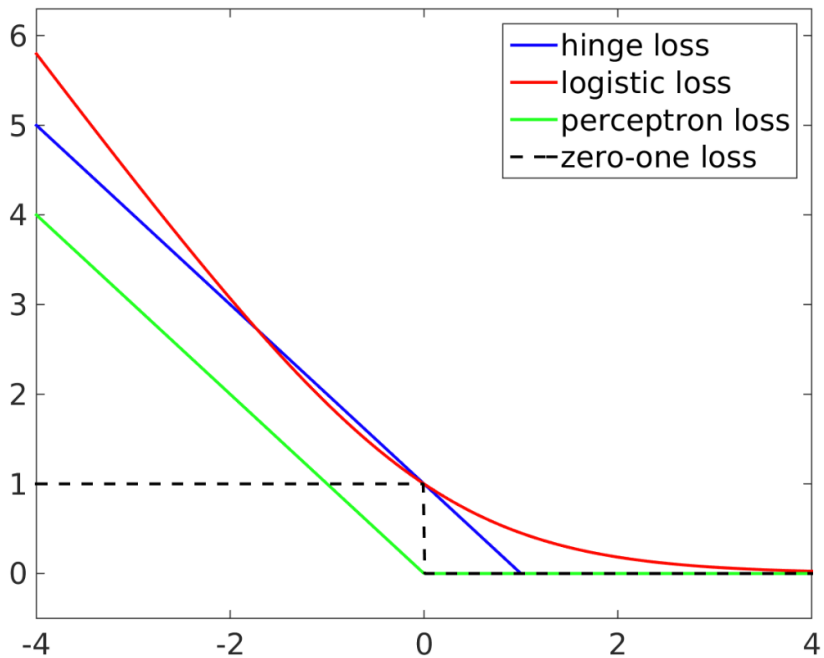
- Perceptron convergence theorem (Rosenblatt, 1962) states that:
  - If training data is linearly separable, then learning algorithm finds a solution in a finite number of iterations.
  - Faster convergence for larger margin.
- If training data is linearly separable then the found solution will depend on the initialization and ordering of data in the updates
- If training data is not linearly separable, then the perceptron learning algorithm will not converge
- No direct multi-class extension
- No probabilistic output or confidence on classification

Relation to logistic/hinge regression

- Perceptron loss similar to hinge loss without the notion of margin
- Not a bound on the zero-one loss
- Loss is zero for any separator, not only for large margin separators
- All are based on (generalised) linear score functions, relying on pre-defined non-linear data transformation or kernel

$$f(x) = w^\top \phi(x)$$

.

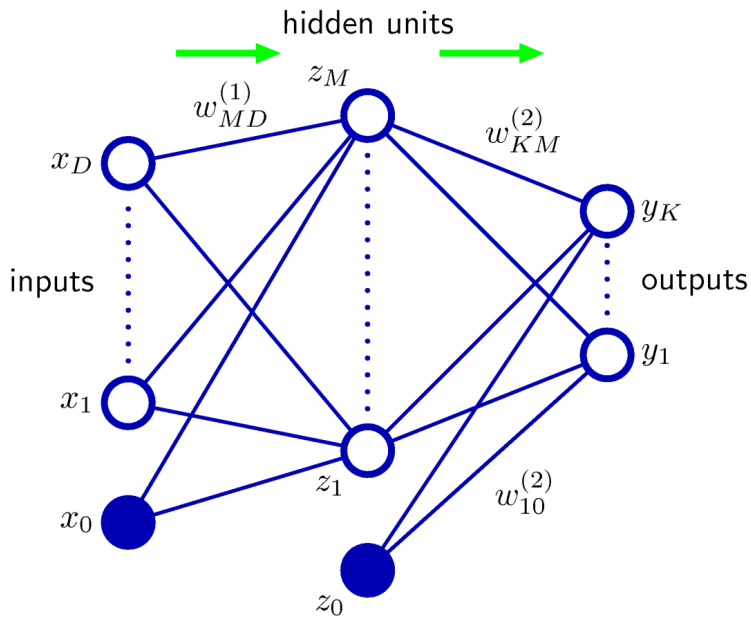


Multi-Layer Perceptron (MLP)

- Instead of using generlised linear functions, learn the features!
- Each unit (neuron) in MLP computes:
  1. Linear function of the features (activations) in the previous layer
  2. Followed by scalar non-linearity (not the perceptron's "step")

$$z_j = h^{(1)}\left(\sum_i w_{ij}^{(1)} x_i\right)$$

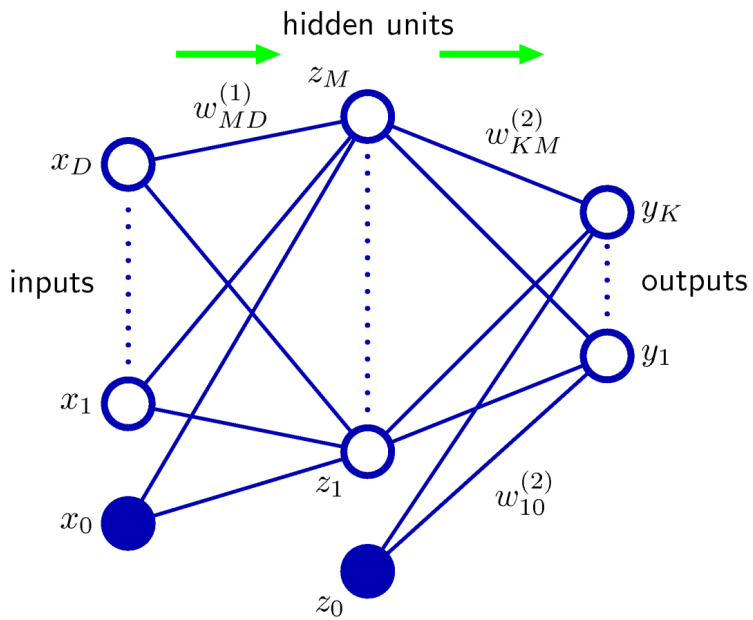
$$y_k = h^{(2)}\left(\sum_j w_{jk}^{(2)} z_j\right)$$



Multi-Layer Perceptron (MLP)

- Instead of using generlised linear functions, learn the features!
- Each unit (neuron) in MLP computes:
  1. Linear function of the features (activations) in the previous layer
  2. Followed by scalar non-linearity (not the perceptron's "step")

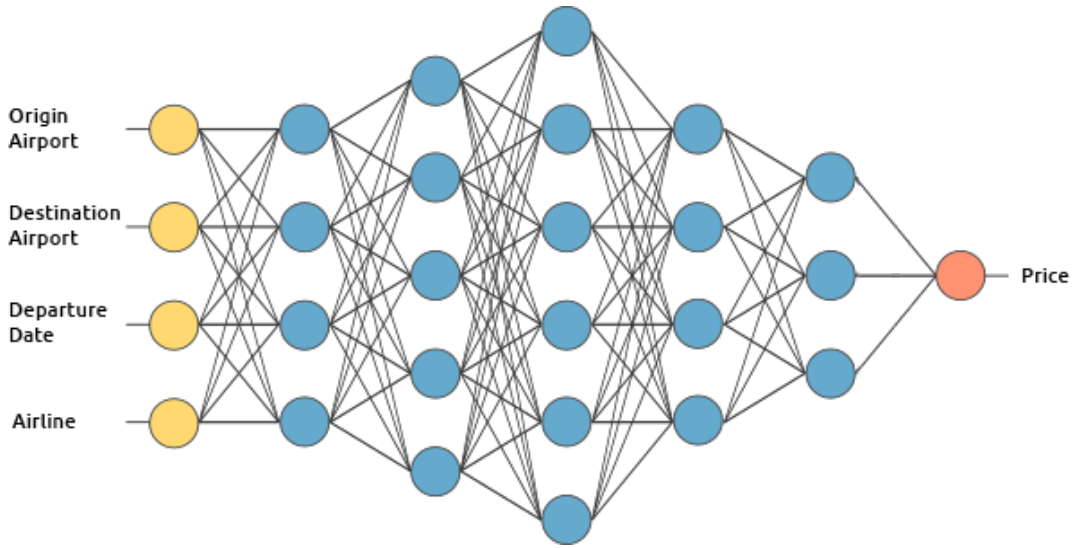
- If activations are linear, then it remans a linear model.
- Two-layer MLPs can uniformly approximate any continuous function on a compact input domain provided a sufficiently large number of hidden units.



Deep Learning: Feed Forward Networks

Feed Forward Networks

- MLP Architecture can be generalized
  - More than two layers of computation
  - Skip-connections from previous layers
- Feed-forward nets are restricted to directed acyclic graphs of connections. Ensures that output can be computed from the input in a single feed-forward pass from the input to the output.



- Important issues in practice
  - Designing network architecture (# nodes, layers, non-linearities, etc)
  - Learning the network parameters (Non-convex optimization)
  - Sufficient training data (Data augmentation, synthesis)

Simple output case: multiclass classification

- Each output neuron is mapped to the posterior prob. of the class:

$$y_k \rightarrow p(c_k|x) = \frac{\exp(y_k)}{\sum_m \exp(y_m)} \quad \text{So-called } \textit{softmax} \text{ operation.}$$

- Multi-class logistic regression loss (also called cross-entropy loss):

$$\mathcal{L} = -\sum_k l_k \log p(c_k|x) \rightarrow l_k = 1 \Leftrightarrow k \text{ is the correct class.}$$

  $\Rightarrow$ We are maximising the log-probability of the correct class.

- We are now learning the classifier and the data representation simultaneously $\rightarrow$ **representation learning**.

  But how do we do that?

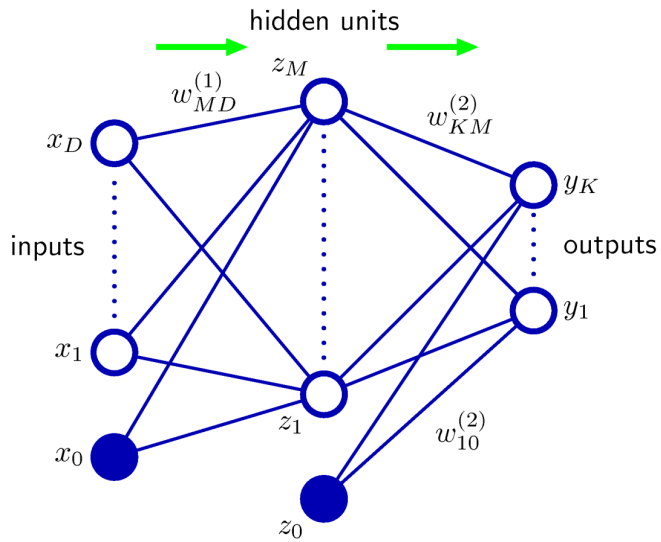
Computing the gradient

Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$

$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$



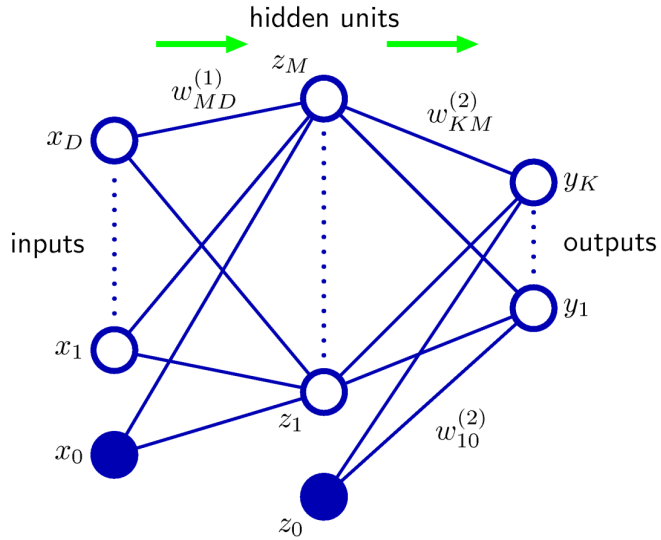
Computing the gradient

Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$

$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

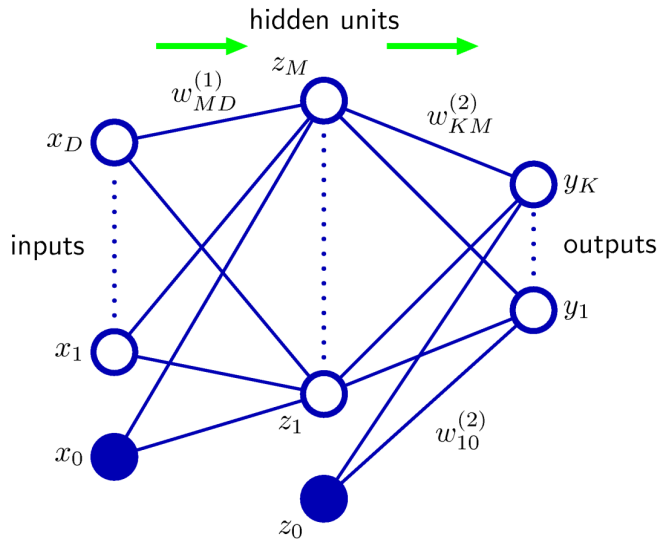


$$\frac{\partial \mathcal{L}}{\partial y_k} =$$

Computing the gradient

Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$



$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

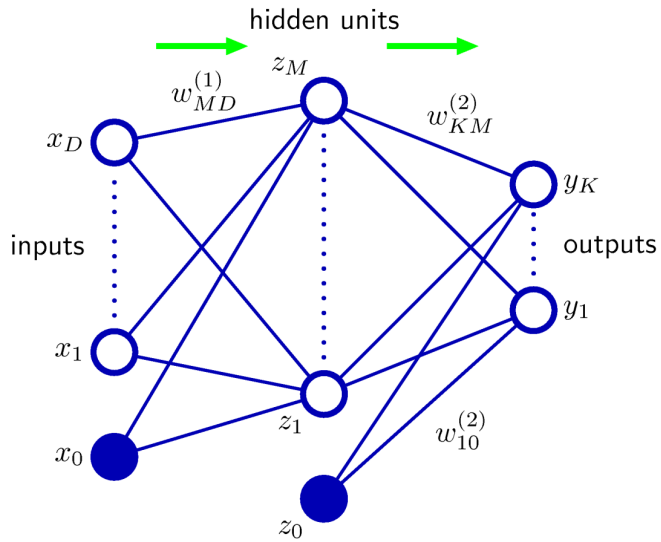
$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

$$\frac{\partial \mathcal{L}}{\partial y_k} = \begin{cases} 0 & l_k = 0 \\ 1 - \frac{e^{y_k}}{\sum_\ell e^{y_\ell}} & l_k = 1 \end{cases}$$

Computing the gradient

Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$



$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

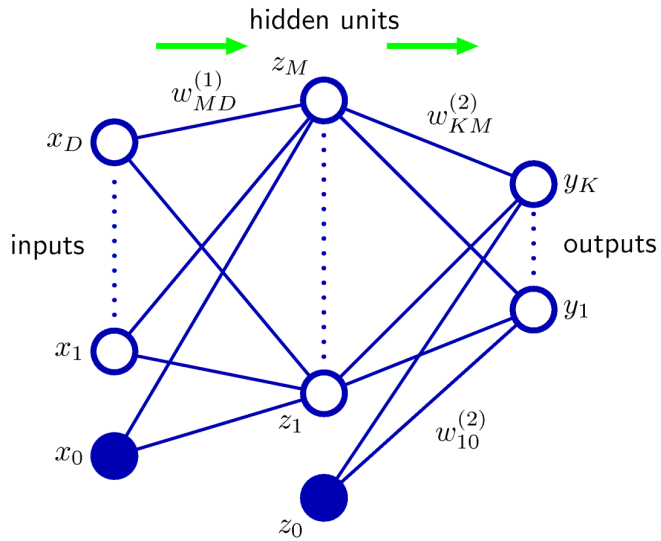
$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

$$\frac{\partial y_k}{\partial w_{km}^{(2)}} = \qquad h'(t) = \frac{-e^{-t}}{(1+e^{-t})^2}$$

Computing the gradient

Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$



$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

$$\frac{\partial y_k}{\partial w_{km}^{(2)}} = h'\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right) z_m \qquad h'(t) = \frac{-e^{-t}}{(1+e^{-t})^2}$$

Computing the gradient

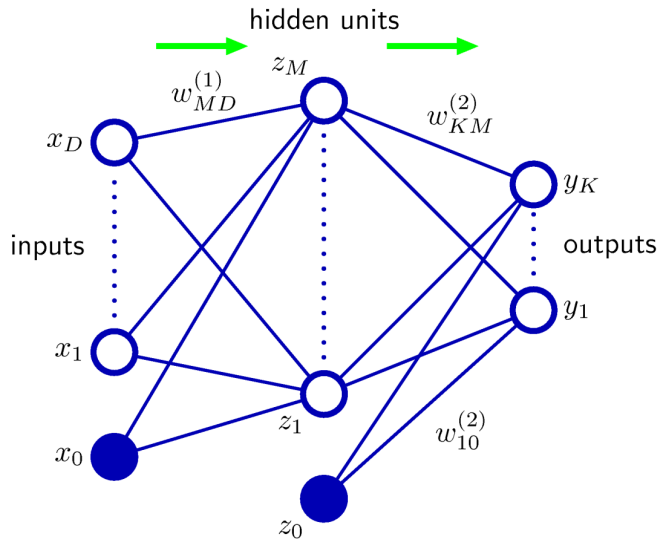
Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$

$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

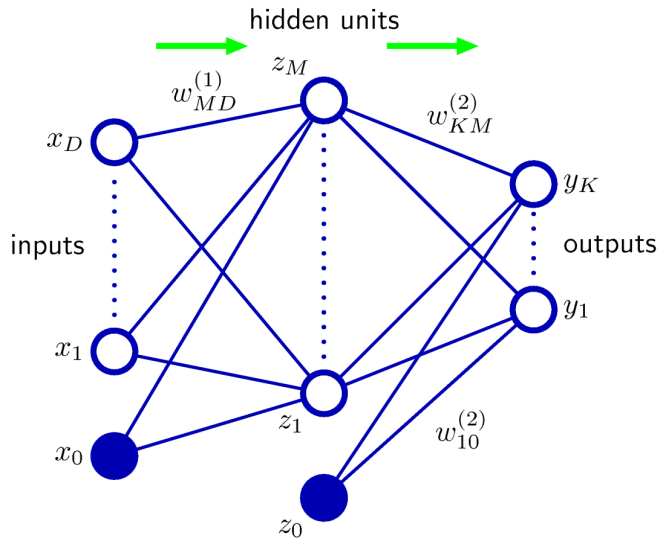
$$\frac{\partial y_k}{\partial z_m} =$$



Computing the gradient

Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$



$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

$$\frac{\partial y_k}{\partial z_m} = h'\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right) w_{km}^{(2)}$$

Computing the gradient

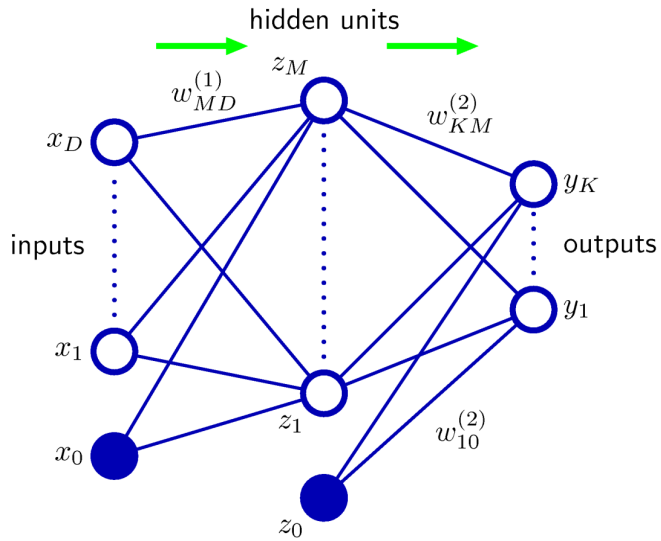
Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$

$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

$$\mathcal{L} = -\sum_k I_k \log p(c_k|x), \quad I_k = 1 \text{ or } 0.$$

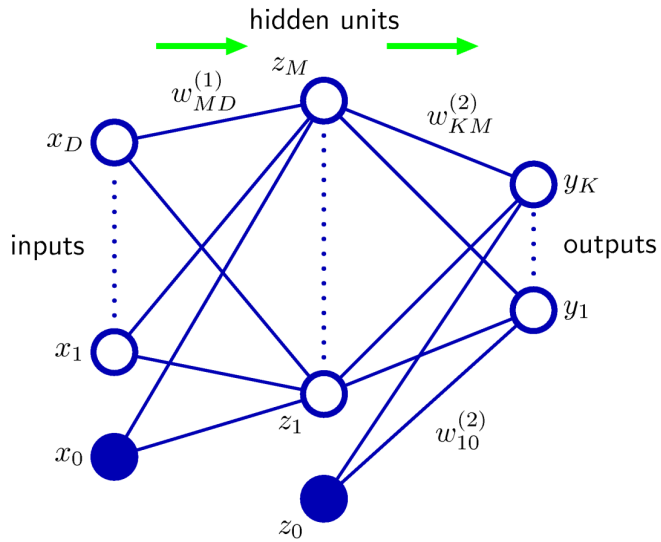
$$\frac{\partial \mathcal{L}}{\partial w_{km}^{(2)}} =$$



Computing the gradient

Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$



$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

$$\frac{\partial \mathcal{L}}{\partial w_{km}^{(2)}} = \sum_k \left(1 - \frac{e^{y_k}}{\sum_\ell e^{y_\ell}}\right) h'\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right) z_m$$

Computing the gradient

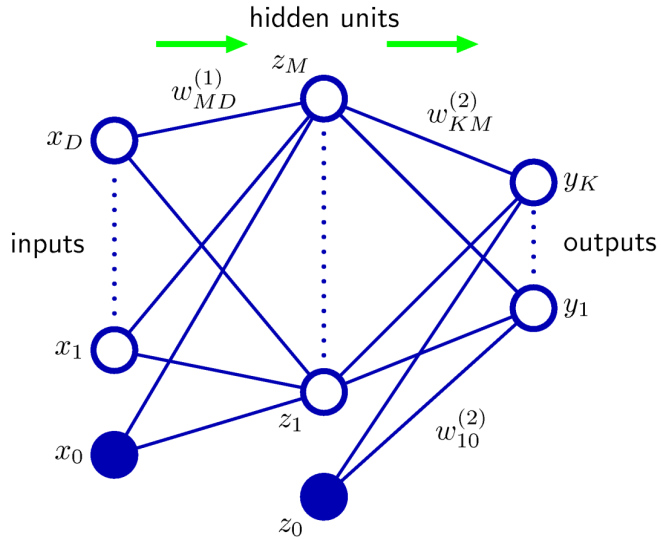
Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$

$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

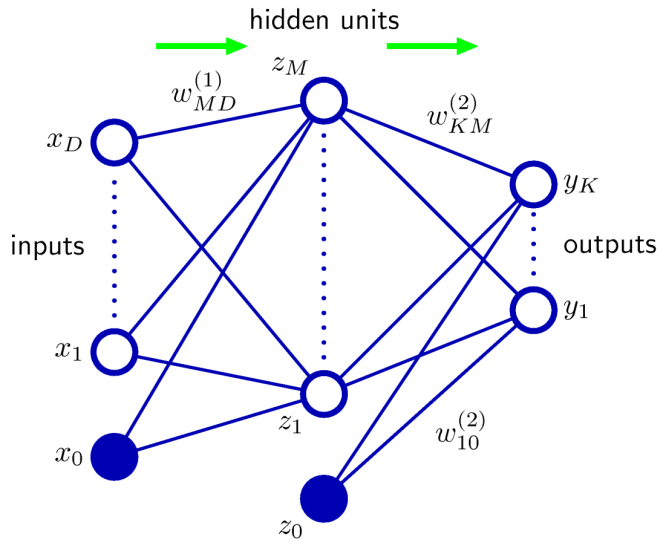
$$\frac{\partial \mathcal{L}}{\partial z_m} =$$



Computing the gradient

Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$



$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

$$\frac{\partial \mathcal{L}}{\partial z_m} = \sum_k \left(1 - \frac{e^{y_k}}{\sum_\ell e^{y_\ell}}\right) h'\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right) w_{km}^{(2)}$$

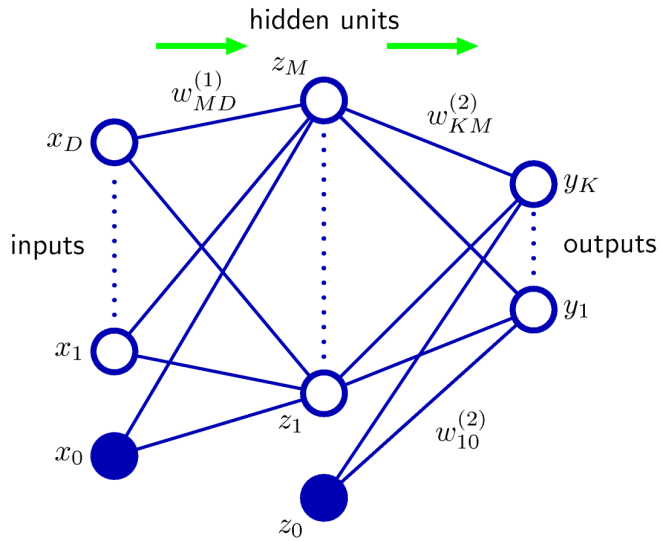
Computing the gradient

Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$

$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

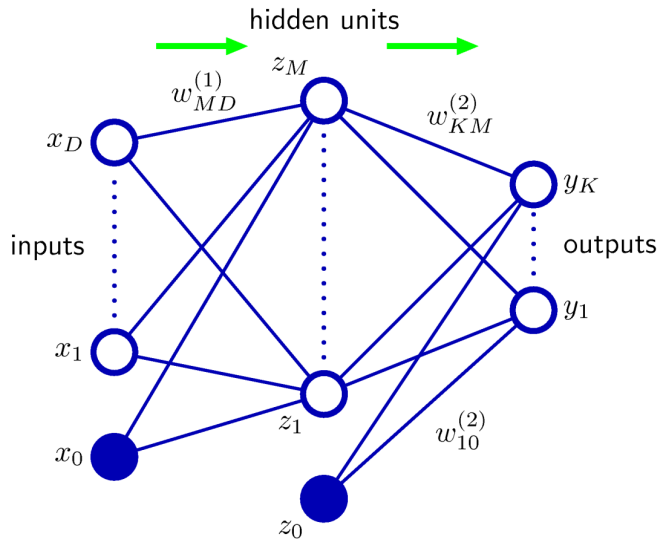


$$\frac{\partial z_m}{\partial w_{md}^{(1)}} =$$

Computing the gradient

Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$



$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

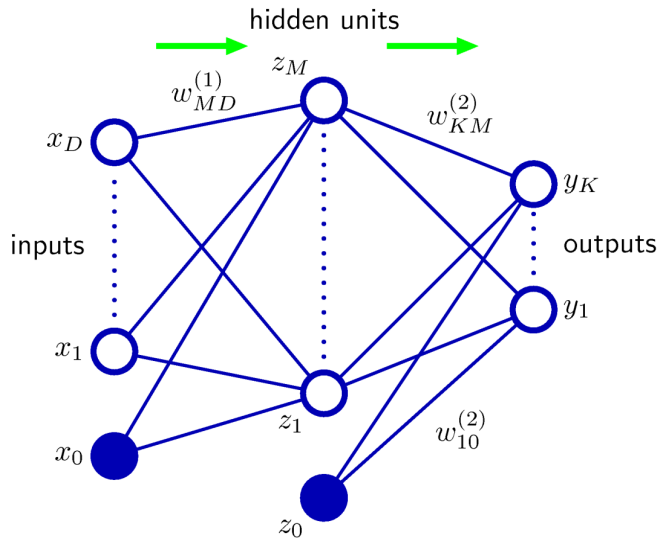
$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

$$\frac{\partial z_m}{\partial w_{md}^{(1)}} = h'\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right) x_d$$

Computing the gradient

Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$



$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

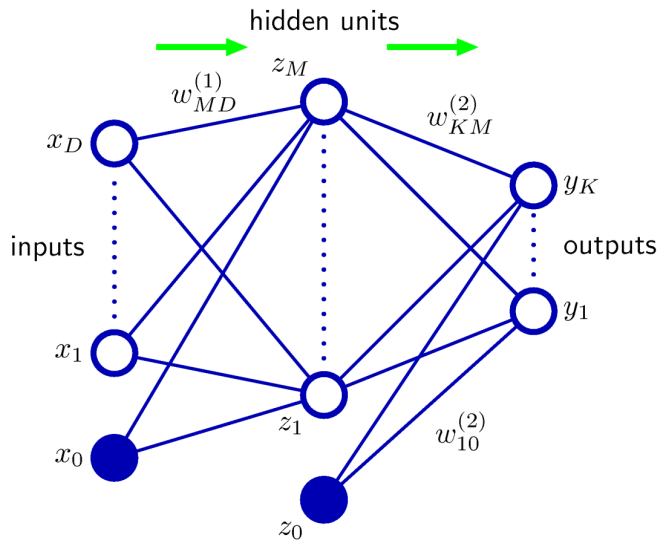
$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

$$\frac{\partial \mathcal{L}}{\partial w_{md}^{(1)}} =$$

Computing the gradient

Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$



$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

$$\frac{\partial \mathcal{L}}{\partial w_{md}^{(1)}} = \sum_k \left(1 - \frac{e^{y_k}}{\sum_\ell e^{y_\ell}}\right) h'\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right) h'\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right) x_d$$

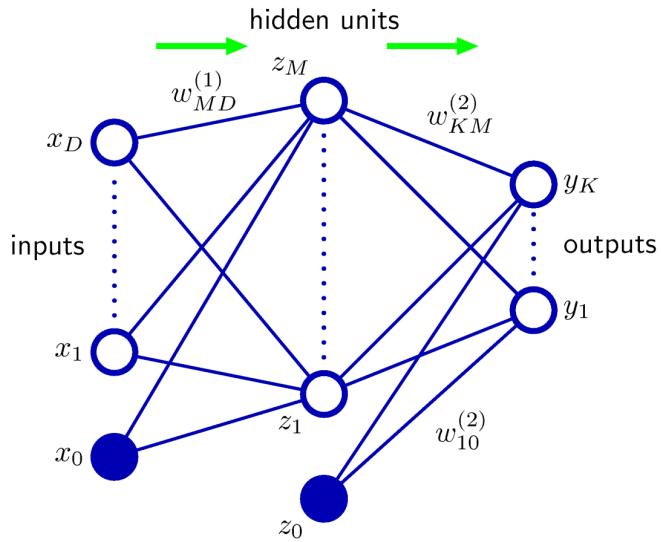
Computing the gradient

Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$

$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

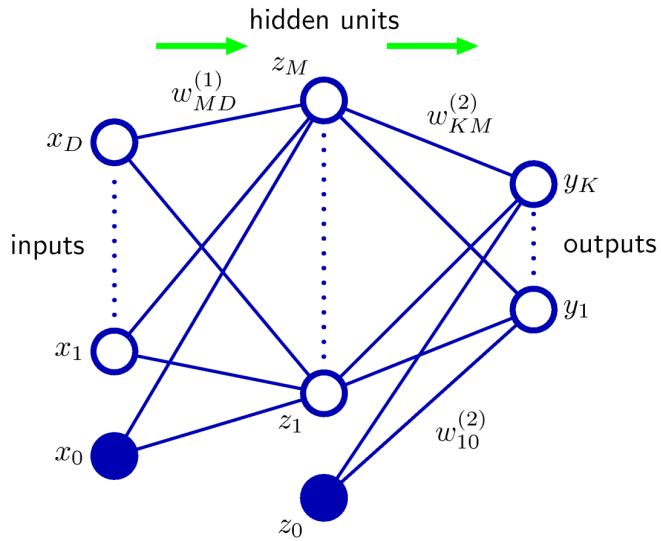


$$\frac{\partial \mathcal{L}}{\partial x_d} =$$

Computing the gradient

Recall:

$$z_m = h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right), \quad y_k = h\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right),$$



$$p(c_k|x) = \frac{e^{y_k}}{\sum_m e^{y_m}}, \quad h(t) = 1/(1+e^{-t}),$$

$$\mathcal{L} = -\sum_k l_k \log p(c_k|x), \quad l_k = 1 \text{ or } 0.$$

$$\frac{\partial \mathcal{L}}{\partial x_d} = \sum_{k,m} \left(1 - \frac{e^{y_k}}{\sum_\ell e^{y_\ell}}\right) h'\left(\sum_{m=0}^{M} w_{km}^{(2)} z_m\right) h'\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right) w_{md}^{(1)}$$

Activation functions

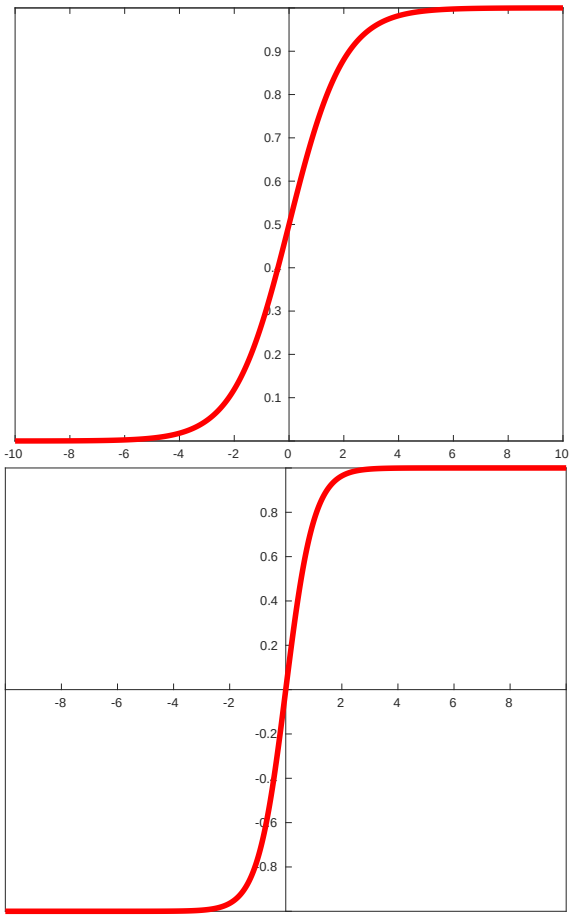
Many possible. Simple non-linear scala functions.

Sigmoid: $h(t) = 1/(1 + e^{-t})$.

- Squeezes reals to $[0, 1]$.
- Smooth step function.
- Historically popular because of biologically inspired.

Tan-h: $h(t) = (e^{t} - e^{-t})/(e^{t} + e^{-t})$.

- Squeezes reals to $[-1, 1]$.
- Similar to sigmoid.
- Limitation: saturated neurons "kill" the gradients.



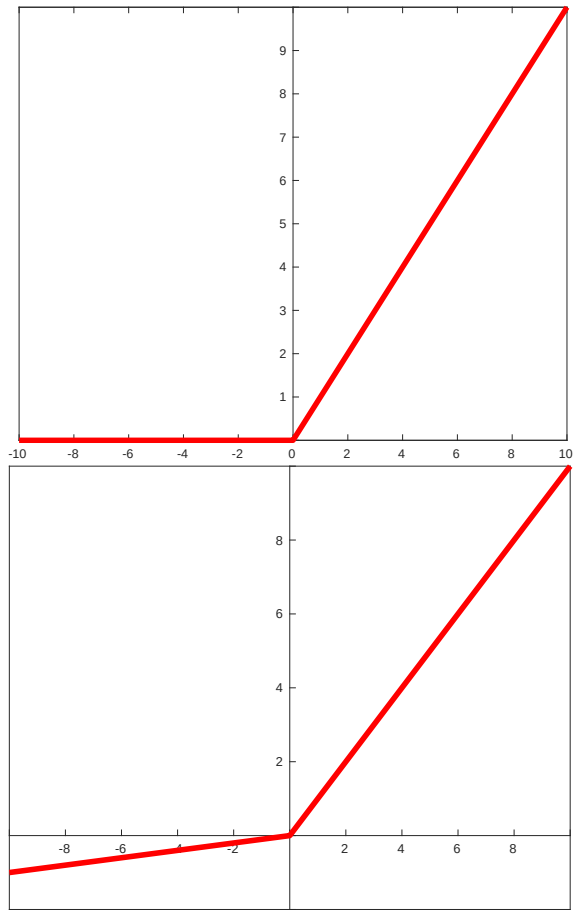
Activation functions

ReLU: $h(t) = \max(0, t)$.

- Rectified Linear Unit.
- Does not saturate.
- Converges much faster than previous. Widely used.

Leaky ReLU: $h(t) = \max(\alpha t, t)$.

- Does not saturate in neither region.
- Computationally efficient.
- Faster convergence.



Training feedforward neural networks

- Non-convex optimisation problem in very high dimension (millions!).

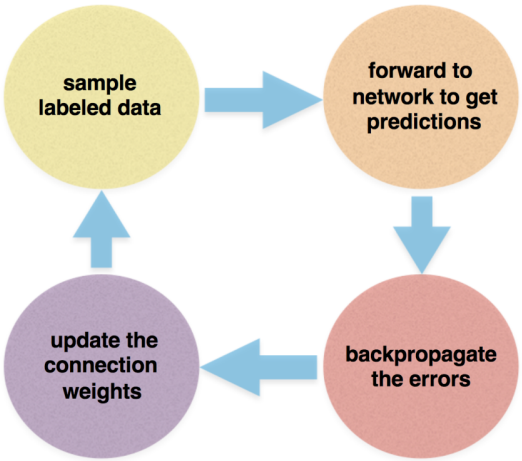
$$\frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\phi(x;W),l)+\lambda\Omega(W),\quad \phi(\cdot;W)\text{ is the DNN.}$$

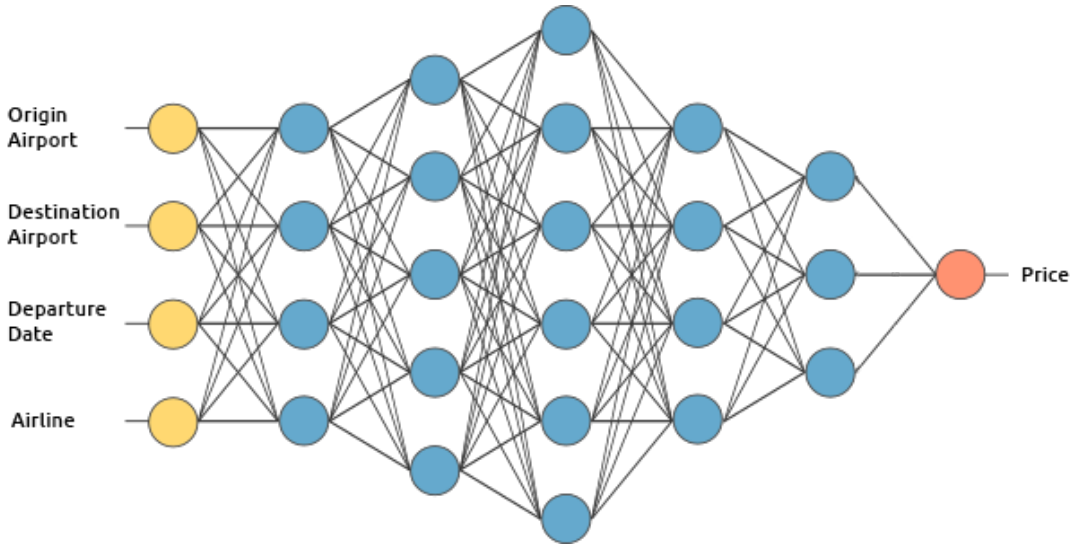
- Many equivalent local minima exist.
- Regularisation, Ω: using "weight decay" or "drop-out."
- Label smoothing to avoid overfitting:

$$\mathcal{L}=(1-\epsilon)\log(p(l|x))+\epsilon\log(1-p(l|x)).$$

- Weight update using gradient descent. For large datasets, we will use batch-stochastic techniques.

Training feedforward neural networks (II)

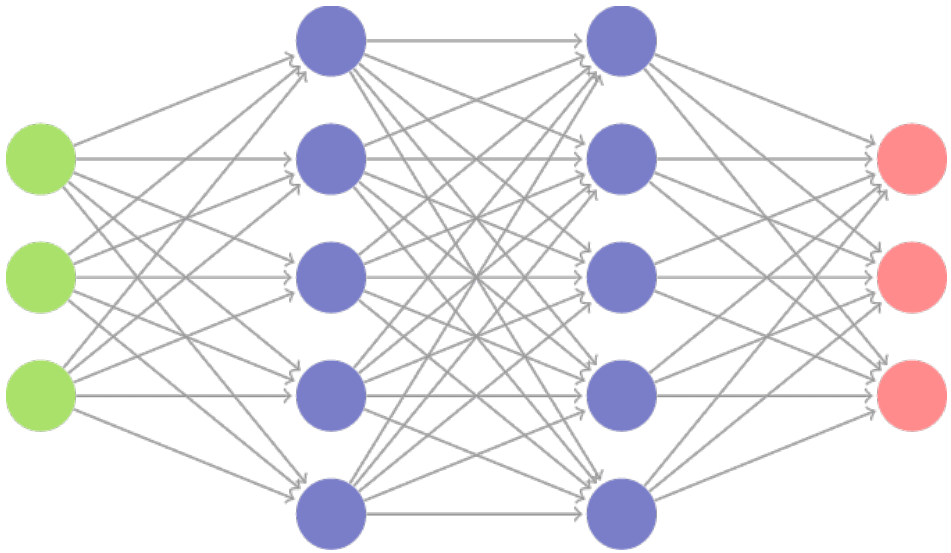




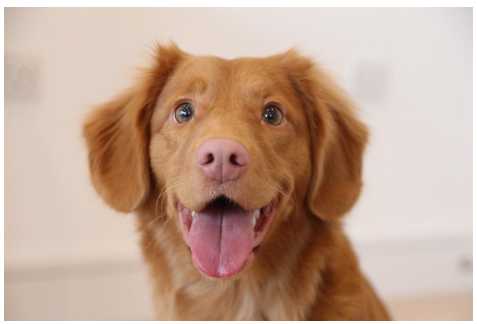
1. Sample a batch of data (input-output) from the dataset.
2. Feed-forward the data through the network.
3. Back-propagate the error through the network (gradient chain rule).
4. Update the weights with gradient.

Deep Learning: Convolutional Neural Networks

Motivation

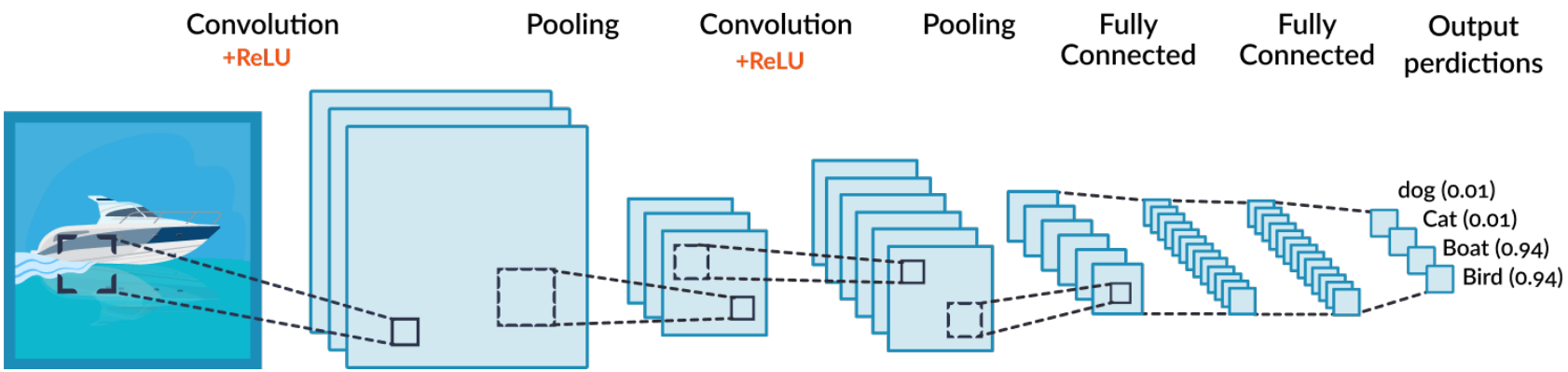


Is this input representation suitable for images?

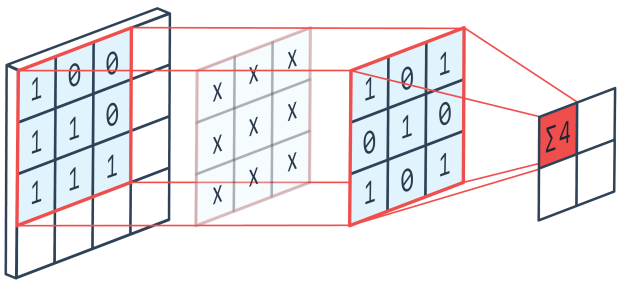
CNN: Overview

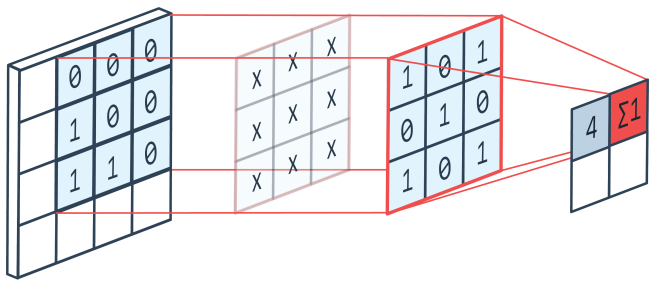
- A convolutional neural network is a special feedforward network
- Hidden units are organized into grid, as is the input
- Linear mapping from layer to layer takes form of convolution
  - Translation invariant processing
  - Local processing
  - Decouples # of parameters from input size
  - Same net can process inputs of varying size



2D convolutions

2D convolutions are linear operations:

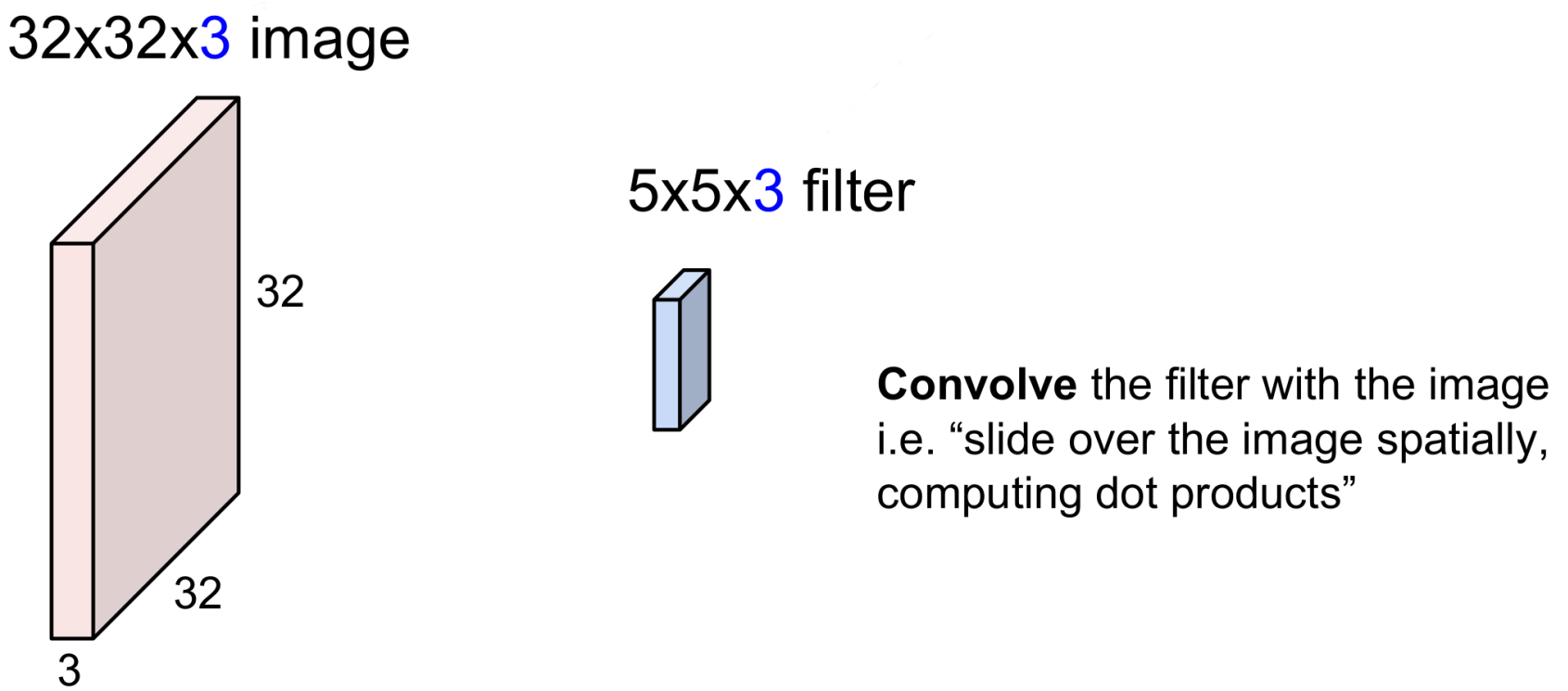




- Start from the top-left position.
- Multiply-and-add.
- Store in the output image.
- Go to next pixel position.

Convolutions in ConvNets

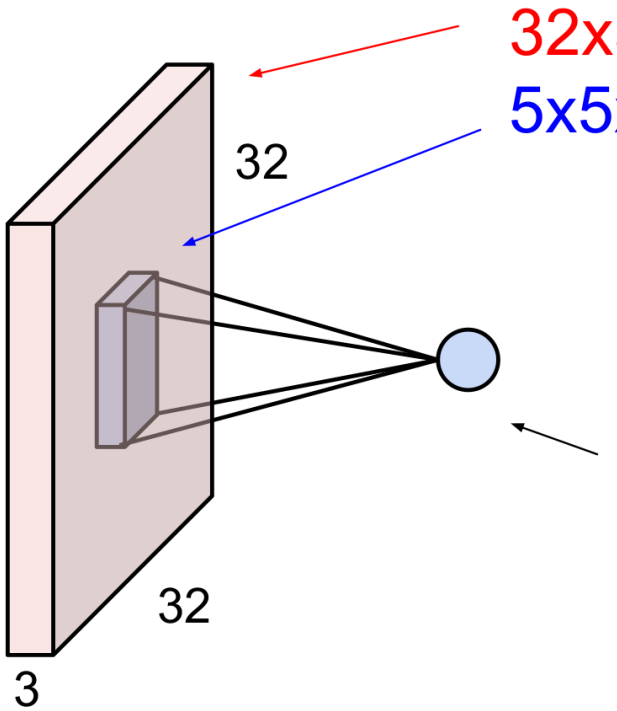
In ConvNets (or CNNs), the convolution filters or kernels have depth:



Slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolutions in ConvNets (II)

What is the width and height of the output image?

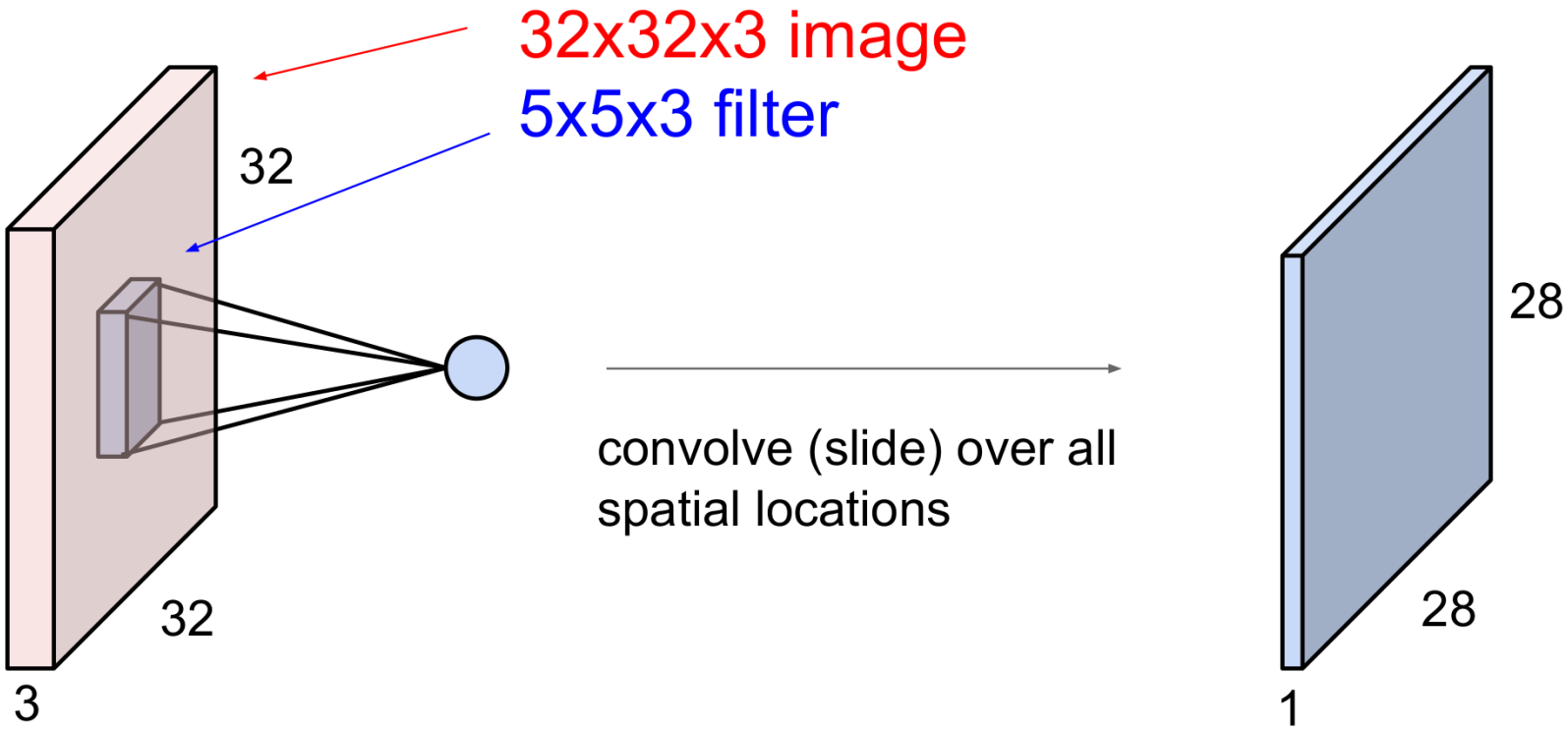


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

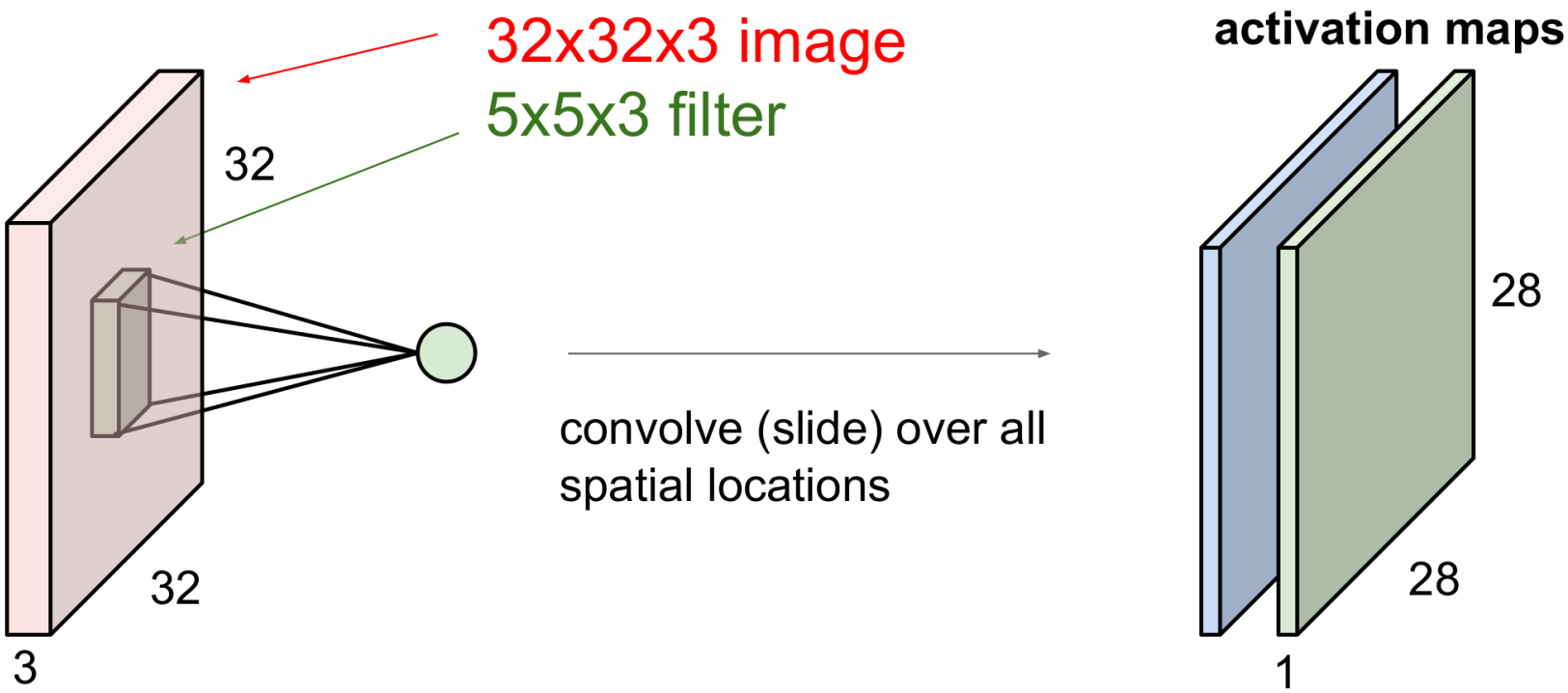
Slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolutions in ConvNets (III)



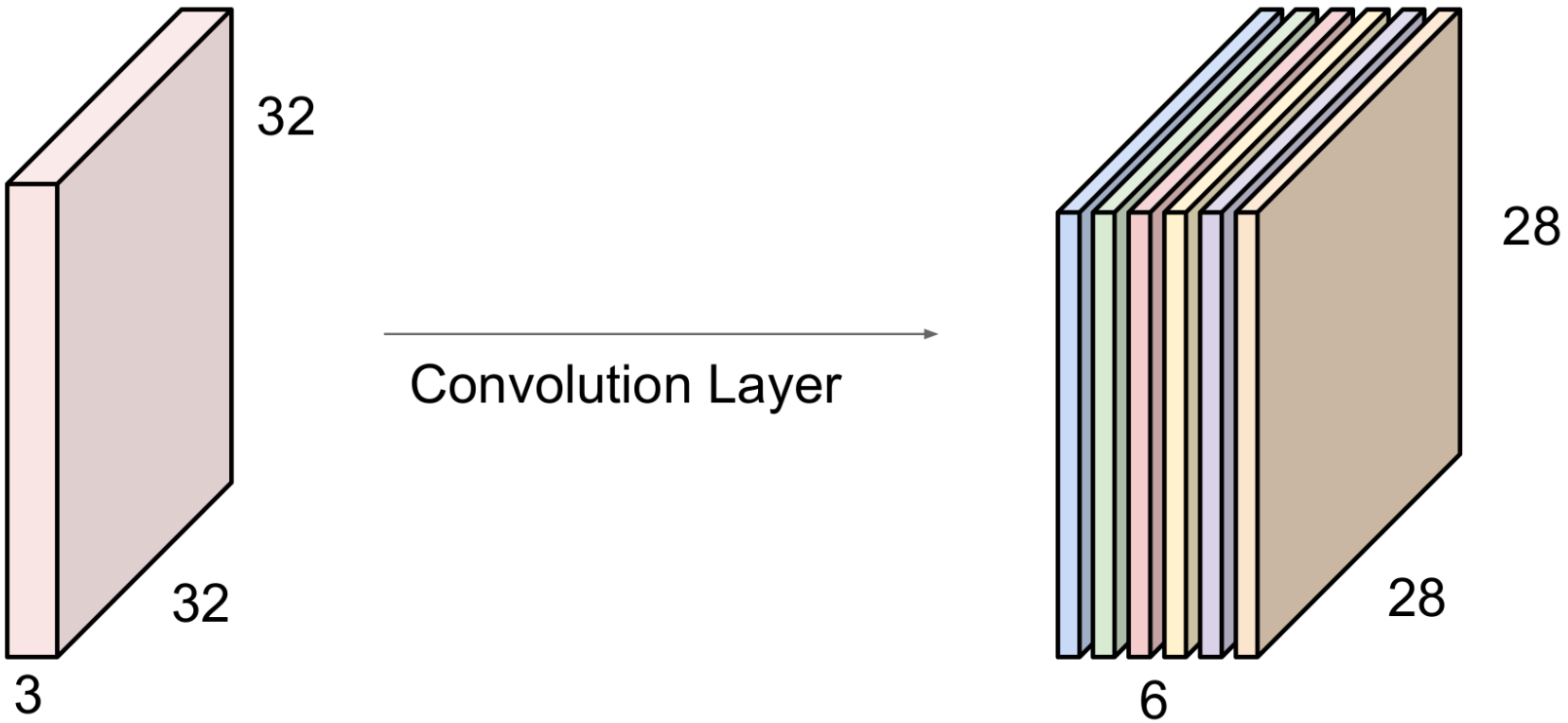
Slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolutions in ConvNets (IV)



Slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

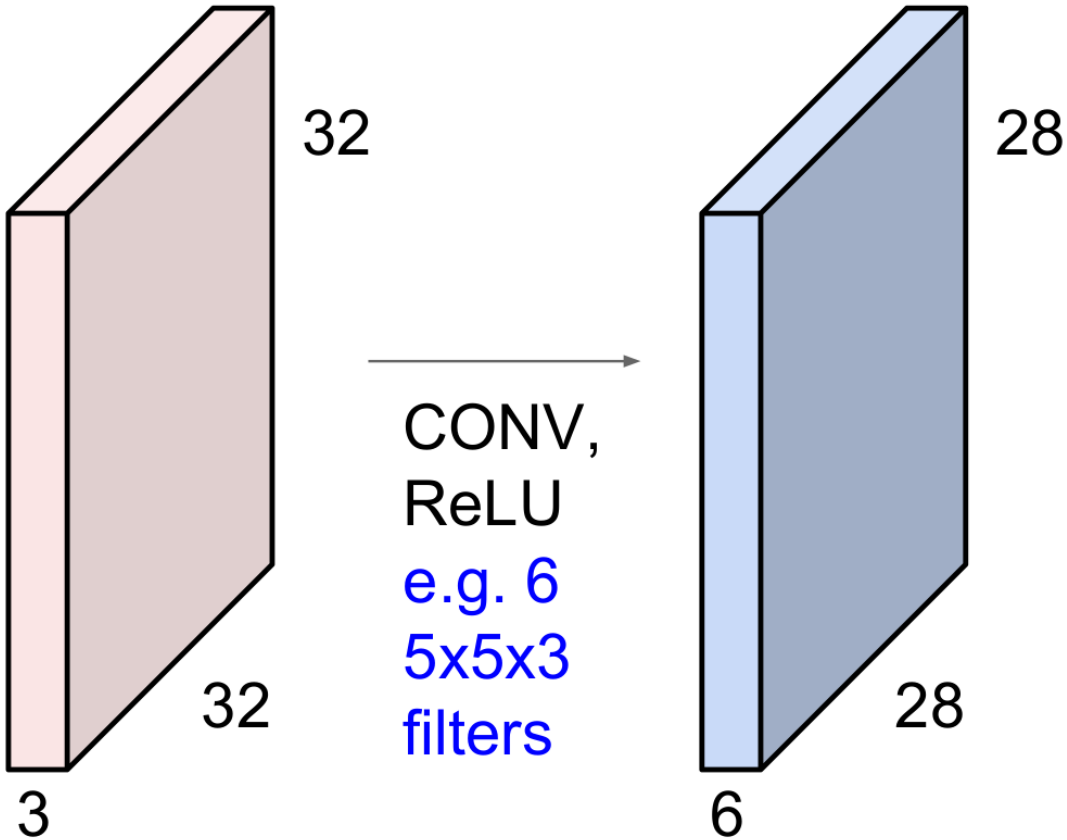
Convolutions in ConvNets (V)



Slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolutions in ConvNets (VI)

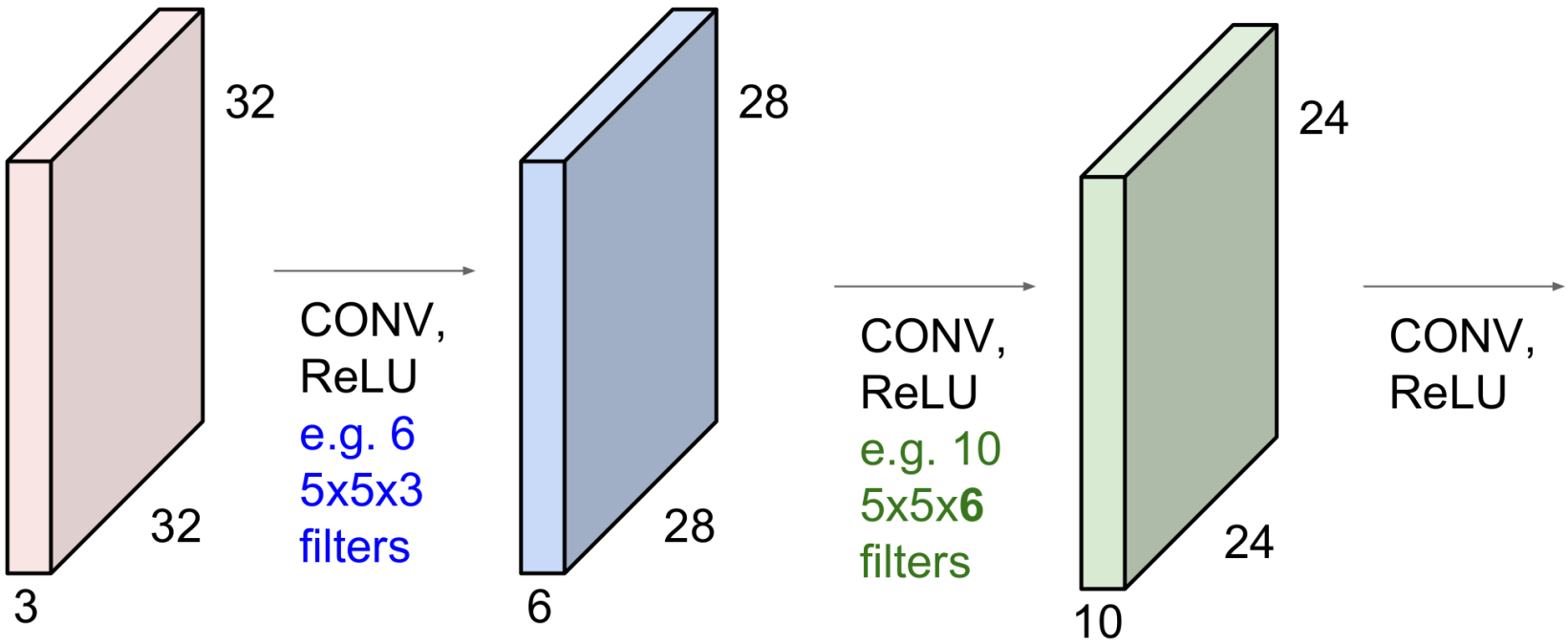
The filters **must** have the same depth as the input. The output depth corresponds to the number of filters.



Slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolutions in ConvNets (VII)

We can repeat the operation with a second convolutional layer.



Slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

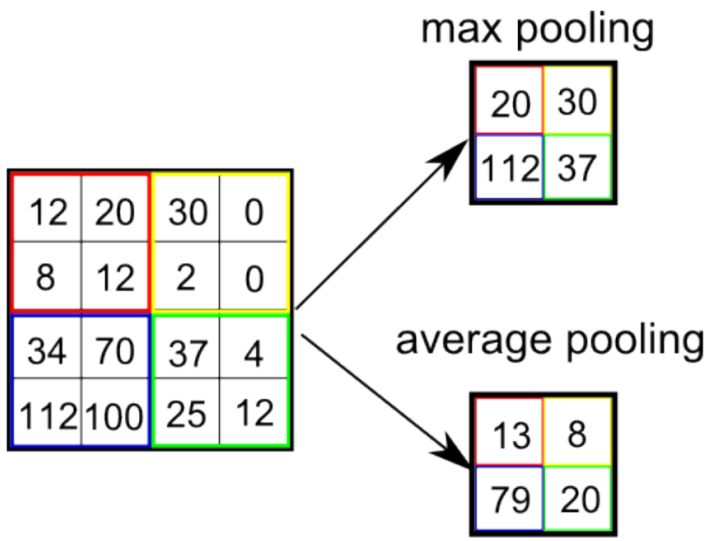
Different types of convolutions

- Standard
- Padded (full, half, etc)
- Strided
- Dilated
- etc.

  `http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html`

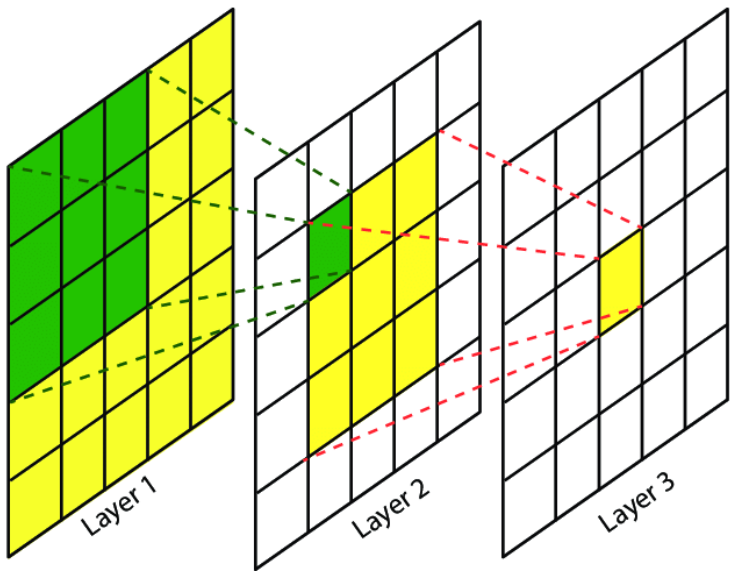
Pooling

- Applied separately per feature channel
- Effect: invariance to small translations of the input
- Max and average pooling most common, other things possible
- Parameter free layer
- Similar to strided convolution with special non-trainable filter



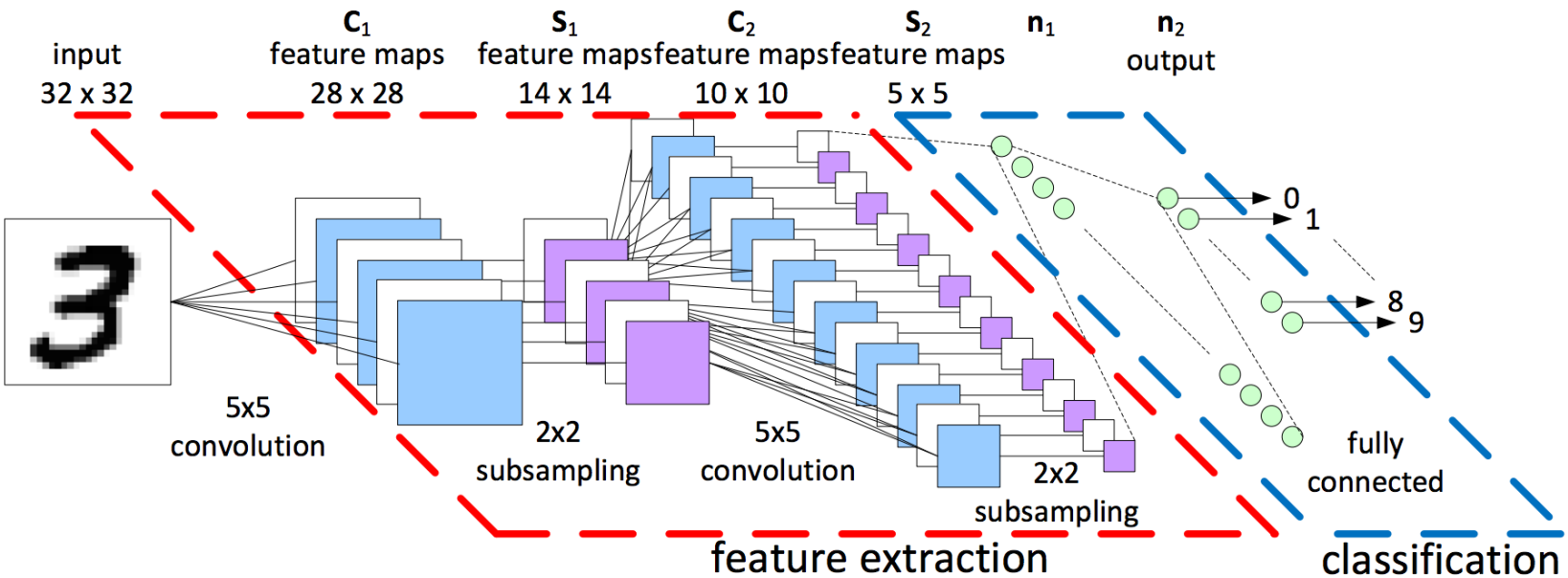
Receptive field

- Receptive field is area in original image impacting a certain unit. Later layers can capture more complex patterns over larger areas.
- Receptive field size grows linearly over convolutional layers. If we use a convolutional filter of size $w \times w$, then each layer the receptive field increases by $w - 1$.
- Receptive field size increases exponentially over layers with striding. Regardless whether they do pooling or convolution.



Fully connected layers

- Convolutional and pooling layers typically followed by several "fully connected" (FC) layers, i.e. a standard MLP
- FC layer connects all units in previous layer to all units in next layer
- Assembles all local information into global vectorial representation

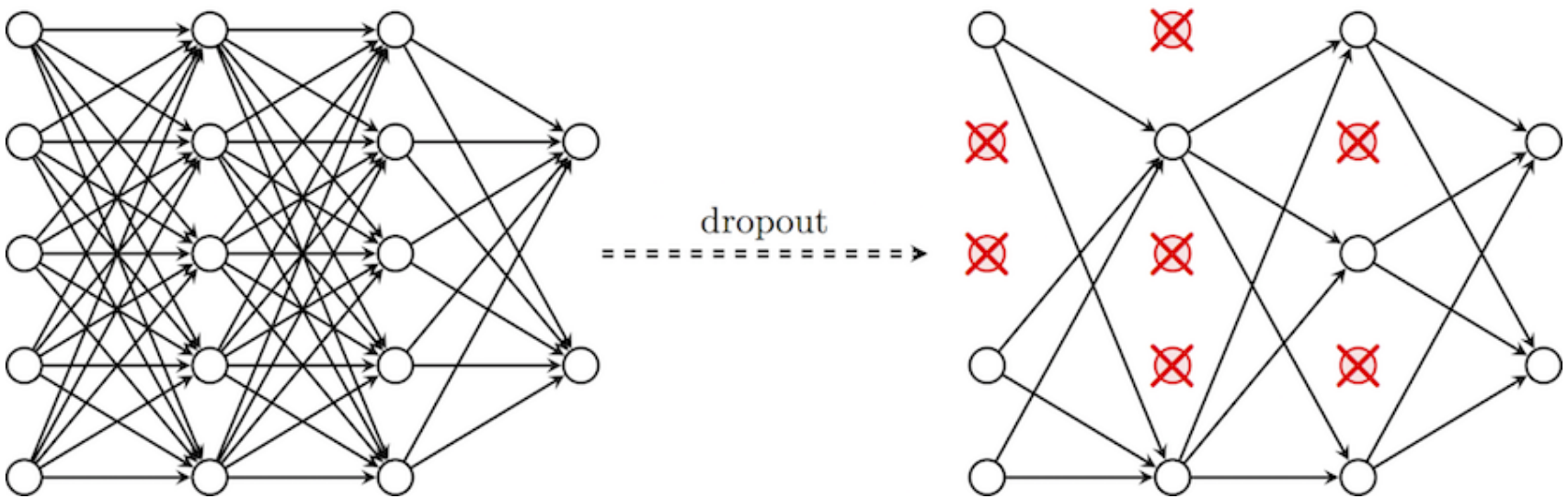


Weight initialisation

- Intuition 1: if the weights are too small, then the signal shrinks and becomes tiny.
- Intuition 2: if the weights are too high, the signal may become too huge to be used.
- We cannot initialise the network weights to very small (or zero) values.
- We usually draw the initial weights from a Gaussian distribution with standard deviation of $\sqrt{2/n}$, where n is the number of outputs to the neuron. $\rightarrow$ Ensures that the signal/gradient stay on the same scale.

Drop-out regularisation

- Main idea: deactivate a subset of neurons, randomly selected at every batch during training.
- If forces the network to be redundant, hence robust.
- Different paths to recognise the same pattern.



Batch normalisation

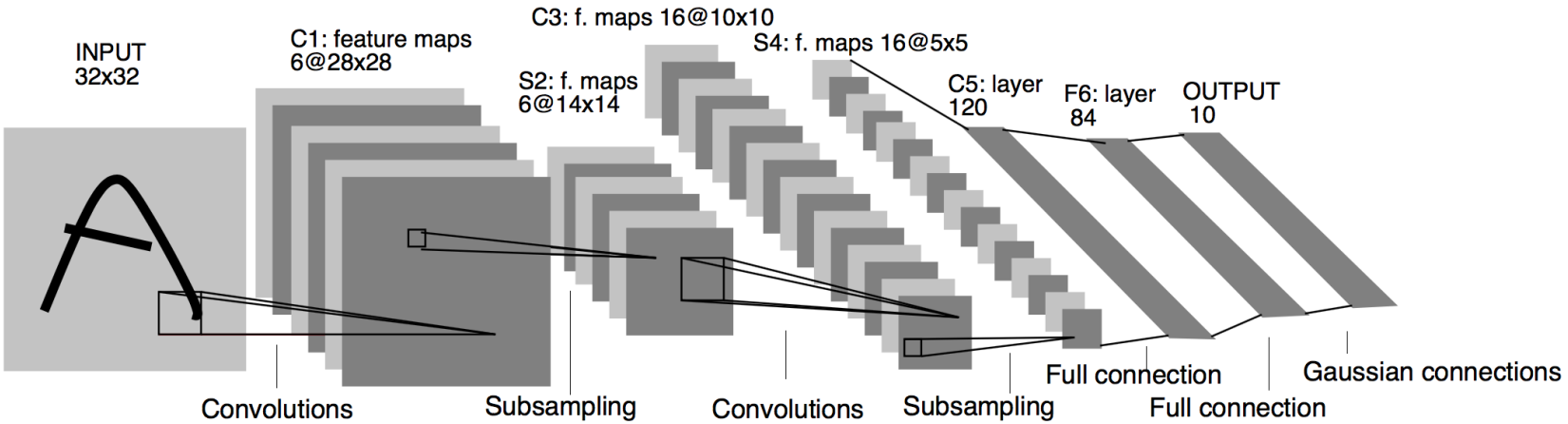
- Motivation: for very deep networks with ReLU, we quickly overflow (very large values).
- Main idea: the output of the linear combinations must follow a standard Gaussian distribution (zero mean, unit variance).
- Rationale: then all layers work at a similar regime.
- Usually after the fully connected or convolutional layers and before the non-linear activation.
- Compute the mean and variance for each neuron and:

$$x^{\text{new}} = \frac{x^{\text{old}} - \mu}{\sqrt{\nu}}$$

.

- Improves gradient flow and allows for larger learning rates.
- It is an unsupervised process that can take place at test time as well.

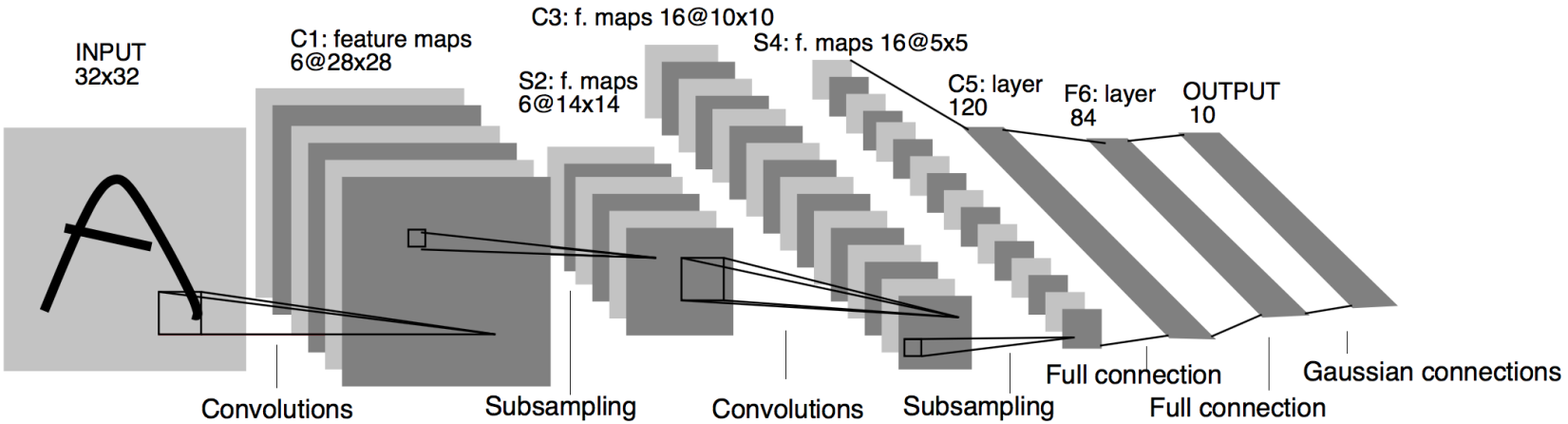
CNN Architectures: Lenet (1998)



Let's compute the # parameters and activations (input image is 32×32):

- C1 (6 filters of 5×5):

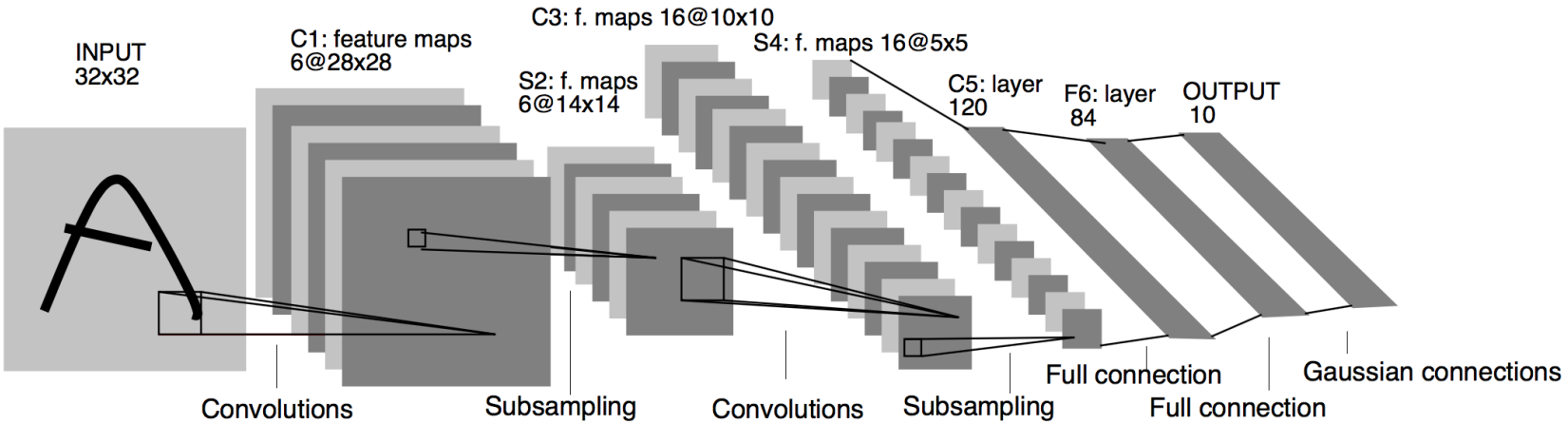
CNN Architectures: Lenet (1998)



Let's compute the # parameters and activations (input image is 32×32):

- C1 (6 filters of 5×5): #par: $(5 \times 5 + 1) \times 6 = 156$ #act: $28 \times 28 \times 6 = 4704$.
- S2 (subsampling):

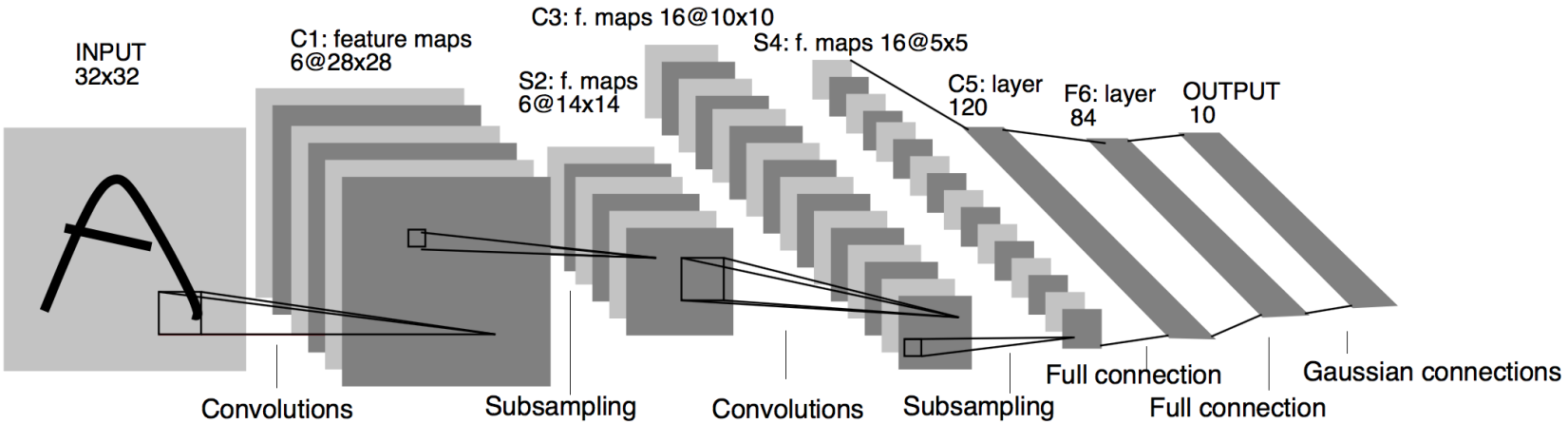
CNN Architectures: Lenet (1998)



Let's compute the # parameters and activations (input image is 32×32):

- C1 (6 filters of 5×5): #par: $(5 \times 5 + 1) \times 6 = 156$ #act: $28 \times 28 \times 6 = 4704$.
- S2 (subsampling): #par : 0 #act: $14 \times 14 \times 6 = 1176$.
- C3 (16 filters of 5×5):

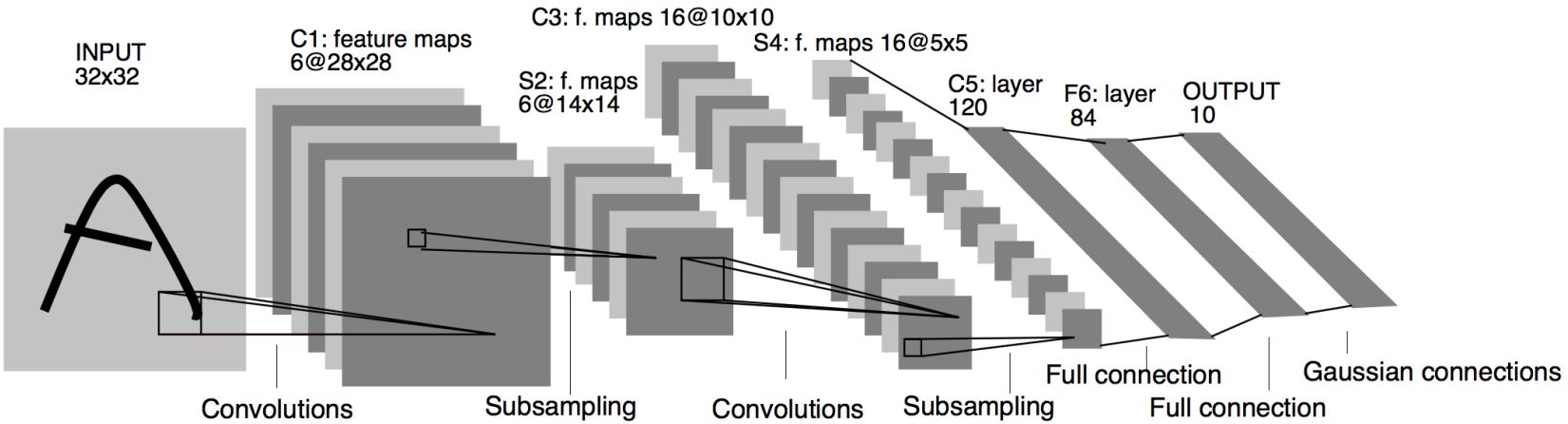
CNN Architectures: Lenet (1998)



Let's compute the # parameters and activations (input image is 32×32):

- C1 (6 filters of 5×5): #par: $(5 \times 5 + 1) \times 6 = 156$ #act: $28 \times 28 \times 6 = 4704$.
- S2 (subsampling): #par : 0 #act: $14 \times 14 \times 6 = 1176$.
- C3 (16 filters of 5×5): #par: $(5 \times 5 + 1) \times 16 \times 6 = 2496$ #act: $10 \times 10 \times 16 = 1600$.
- S4 (subsampling):

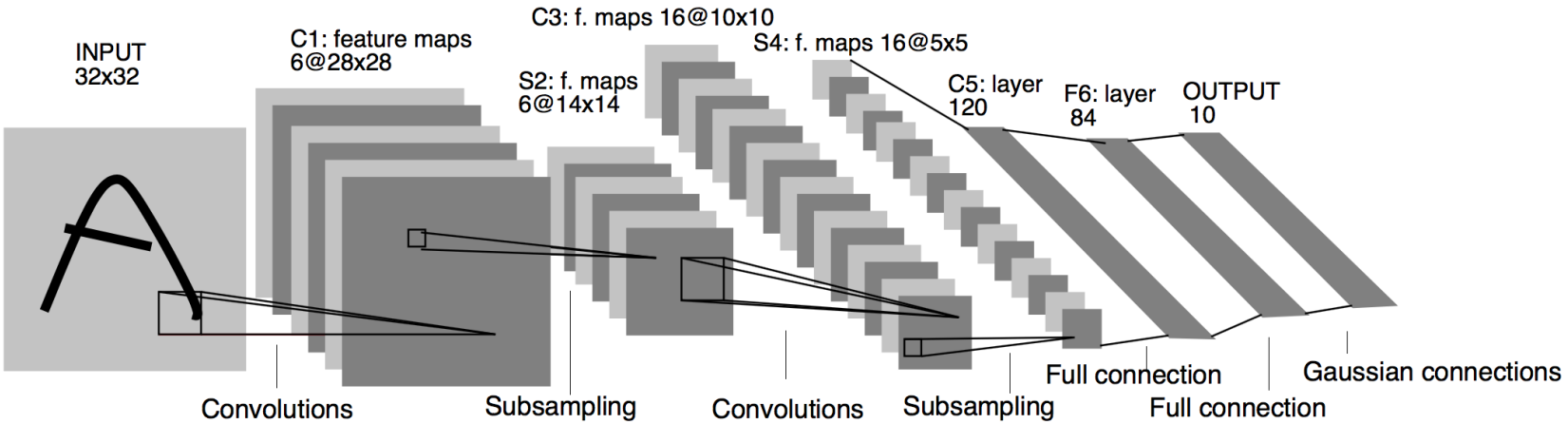
CNN Architectures: Lenet (1998)



Let's compute the # parameters and activations (input image is 32×32):

- C1 (6 filters of 5×5): #par: $(5 \times 5 + 1) \times 6 = 156$ #act: $28 \times 28 \times 6 = 4704$.
- S2 (subsampling): #par : 0 #act: $14 \times 14 \times 6 = 1176$.
- C3 (16 filters of 5×5): #par: $(5 \times 5 + 1) \times 16 \times 6 = 2496$ #act: $10 \times 10 \times 16 = 1600$.
- S4 (subsampling): #par: 0 #act: $5 \times 5 \times 16 = 400$.
- C5 (FC output 120):

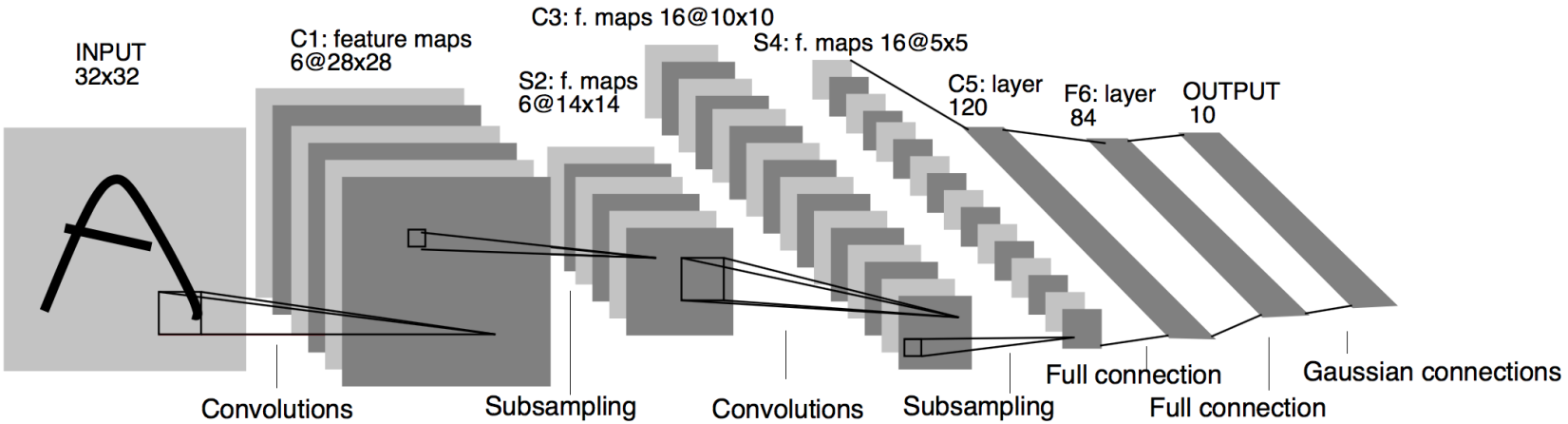
CNN Architectures: Lenet (1998)



Let's compute the # parameters and activations (input image is 32×32):

- C1 (6 filters of 5×5): #par: $(5 \times 5 + 1) \times 6 = 156$ #act: $28 \times 28 \times 6 = 4704$.
- S2 (subsampling): #par : 0 #act: $14 \times 14 \times 6 = 1176$.
- C3 (16 filters of 5×5): #par: $(5 \times 5 + 1) \times 16 \times 6 = 2496$ #act: $10 \times 10 \times 16 = 1600$.
- S4 (subsampling): #par: 0 #act: $5 \times 5 \times 16 = 400$.
- C5 (FC output 120): #par: $(400 + 1) \times 120 = 48120$ #act: 120.
- F6 (FC output 84):

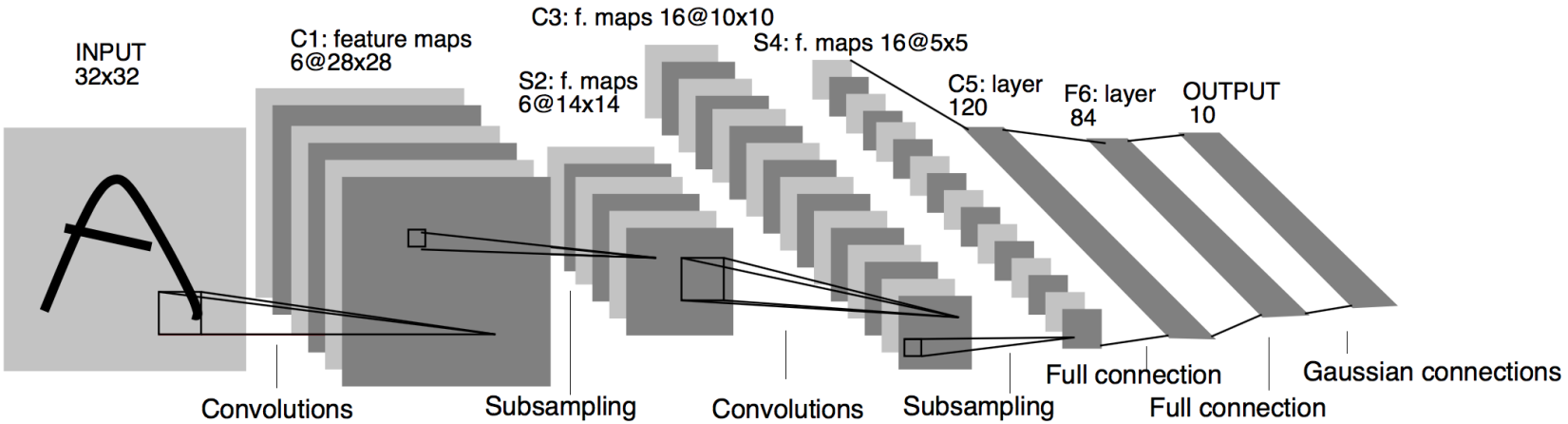
CNN Architectures: Lenet (1998)



Let's compute the # parameters and activations (input image is 32×32):

- C1 (6 filters of 5×5): #par: $(5 \times 5 + 1) \times 6 = 156$ #act: $28 \times 28 \times 6 = 4704$.
- S2 (subsampling): #par : 0 #act: $14 \times 14 \times 6 = 1176$.
- C3 (16 filters of 5×5): #par: $(5 \times 5 + 1) \times 16 \times 6 = 2496$ #act: $10 \times 10 \times 16 = 1600$.
- S4 (subsampling): #par: 0 #act: $5 \times 5 \times 16 = 400$.
- C5 (FC output 120): #par: $(400 + 1) \times 120 = 48120$ #act: 120.
- F6 (FC output 84): #par: $120 * 84 = 10080$ #act: 84.
- Class (FC output 10):

CNN Architectures: Lenet (1998)



Let's compute the # parameters and activations (input image is 32×32):

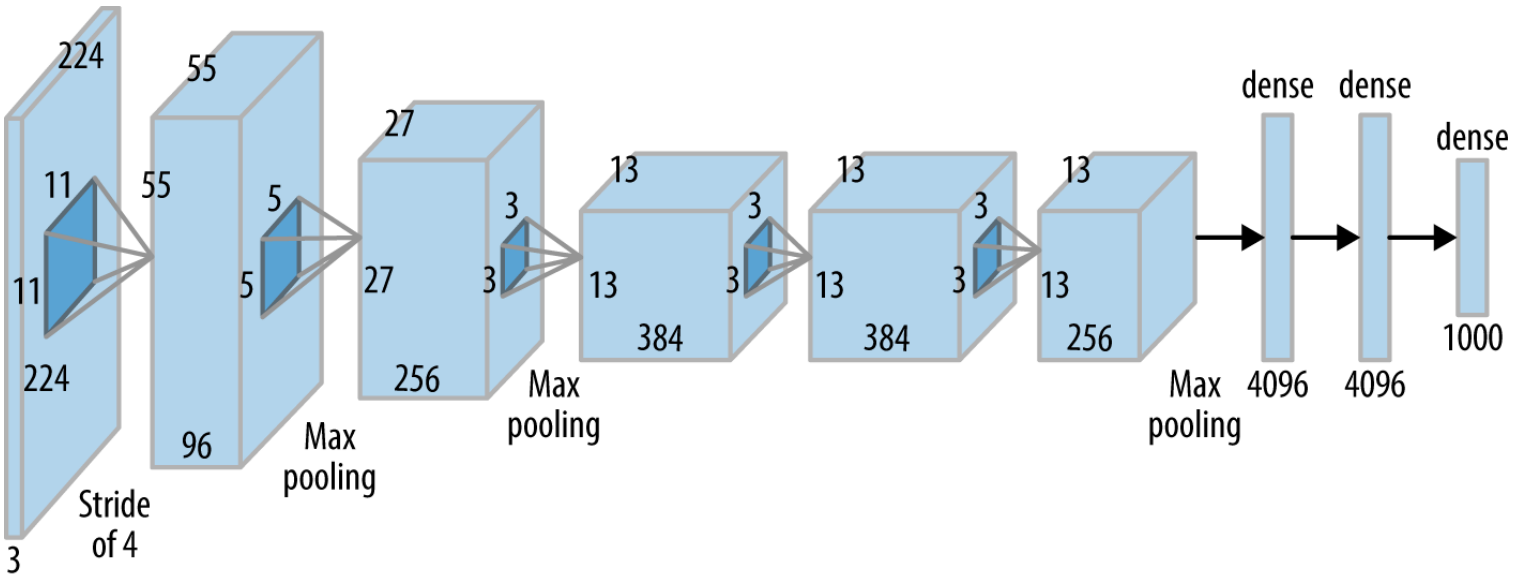
- C1 (6 filters of 5×5): #par: $(5 \times 5 + 1) \times 6 = 156$ #act: $28 \times 28 \times 6 = 4704$.
- S2 (subsampling): #par : 0 #act: $14 \times 14 \times 6 = 1176$.
- C3 (16 filters of 5×5): #par: $(5 \times 5 + 1) \times 16 \times 6 = 2496$ #act: $10 \times 10 \times 16 = 1600$.
- S4 (subsampling): #par: 0 #act: $5 \times 5 \times 16 = 400$.
- C5 (FC output 120): #par: $(400 + 1) \times 120 = 48120$ #act: 120.
- F6 (FC output 84): #par: $120 * 84 = 10080$ #act: 84.
- Class (FC output 10): #par: $84 * 10 = 840$ #act: 10.

What changed?

- Large training **data**sets for computer vision
  - 1.2 million images of 1000 classes in ImageNet challenge (2012)
  - 200 million faces to train face recognition nets (2015)
- **GPU**-based implementation: much faster than CPU
  - Parallel computation for matrix products
  - Krizhevsky & Hinton, 2012: six days on two GPUs (see next slide)
  - Rapid progress in GPU compute performance
- Network **architectures**
- Industrially backed **open-source** software (Pytorch, TensorFlow, etc)

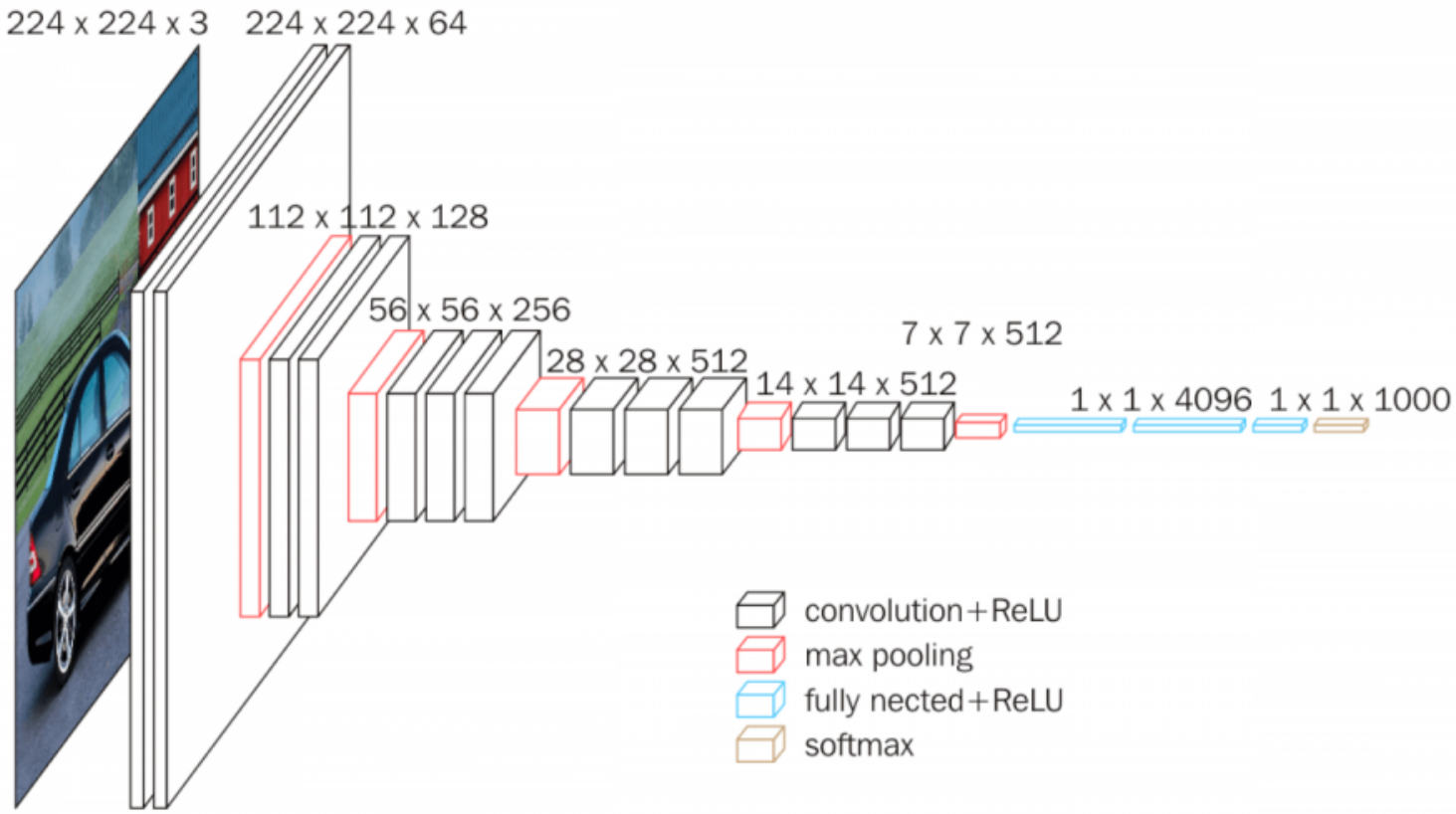
AlexNet CNN (2012)

- Winner ImageNet 2012 image classification challenge, huge impact (+50k citations). CNNs improving "traditional" computer vision techniques on uncontrolled images.
- Compared to LeNet
  - Inputs at 224x224 rather than 32x32
  - 5 rather than 3 conv layers
  - More feature channels in each layer
  - $\sim$ 60 million parameters
  - ReLU non-linearity



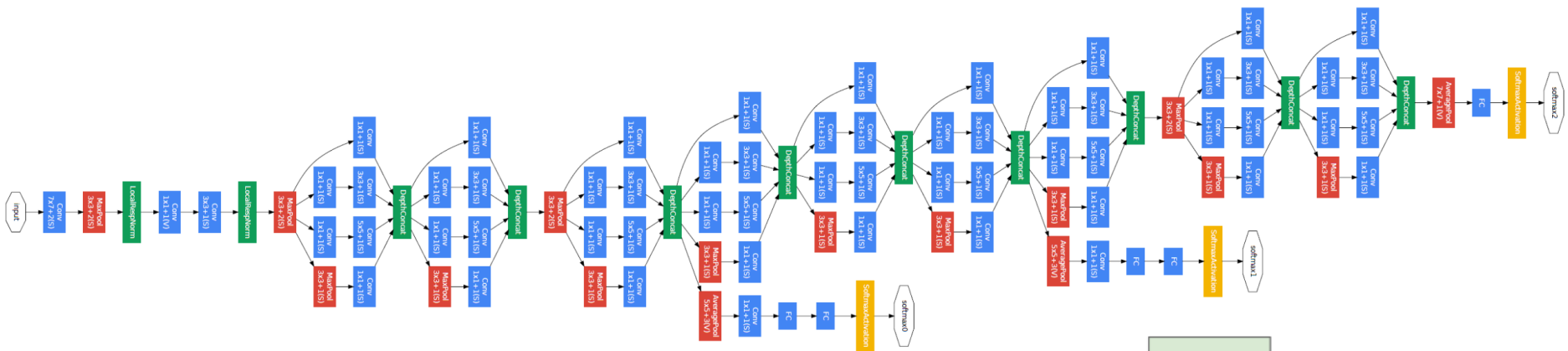
VGG CNN (2015)

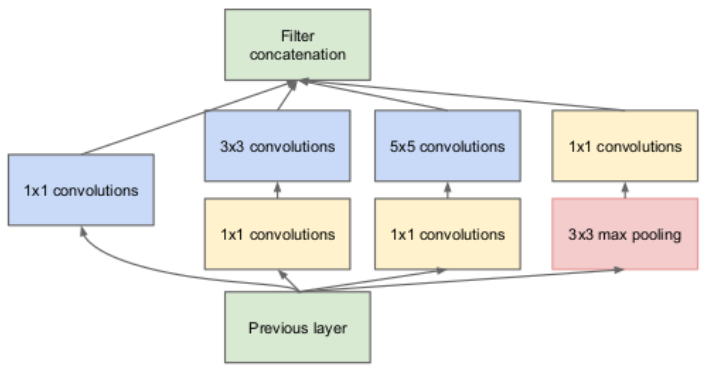
- Double the number of layers (up to 16/19).
- Only small 3×3 filters (rather than 11 in AlexNet). Same receptive field, less parameters, better learned.
- About 140 million parameters.



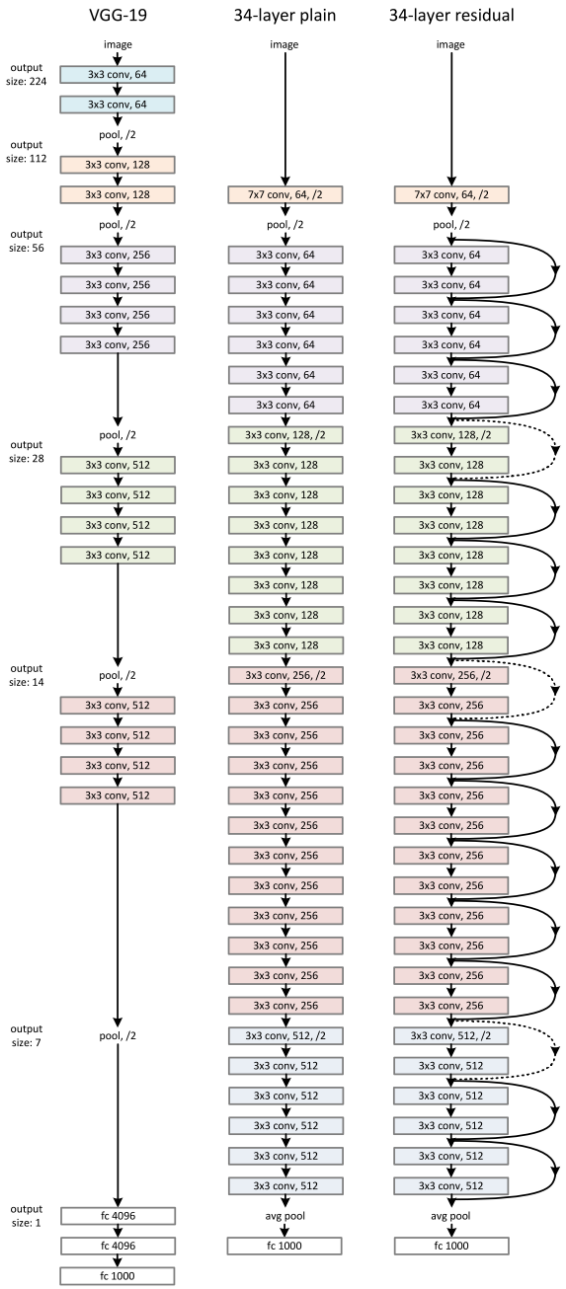
GoogleNet Inception CNN (2015)

- Reduced number of parameters (5 million) but more layers (27 or 48).
- Inception module to compress features before convolution.
- Replaces fully-connected with average pooling.
- Intermediate loss functions to improve training.

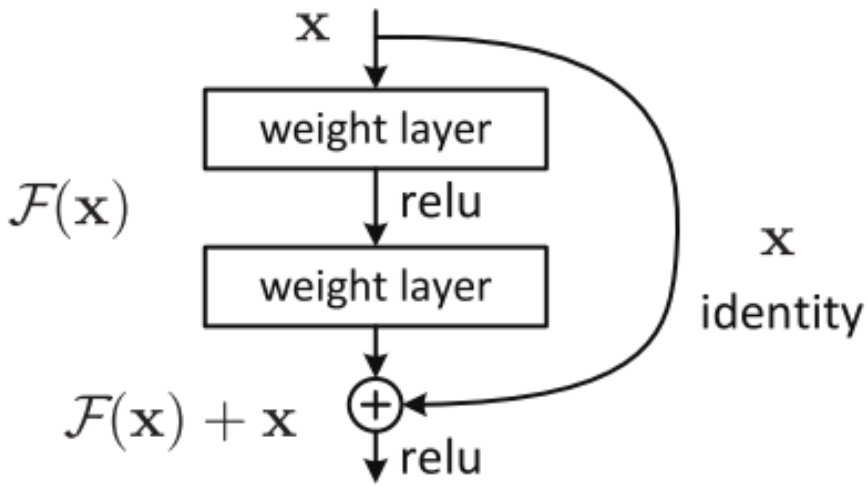




Res(idual)Net CNN (2015)

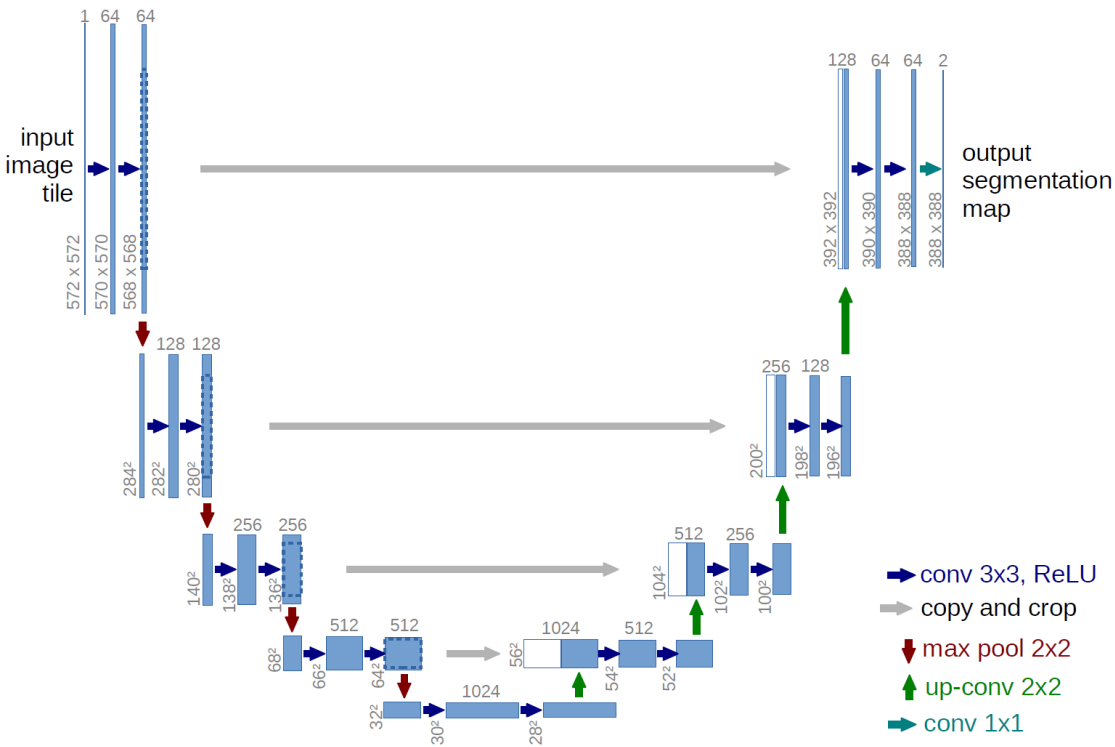


- Many more layers (34, 50, 110, 1200), multi-GPU training is required.
- Residual module to ensure gradient flow.
- Residual block does not require intermediate losses.



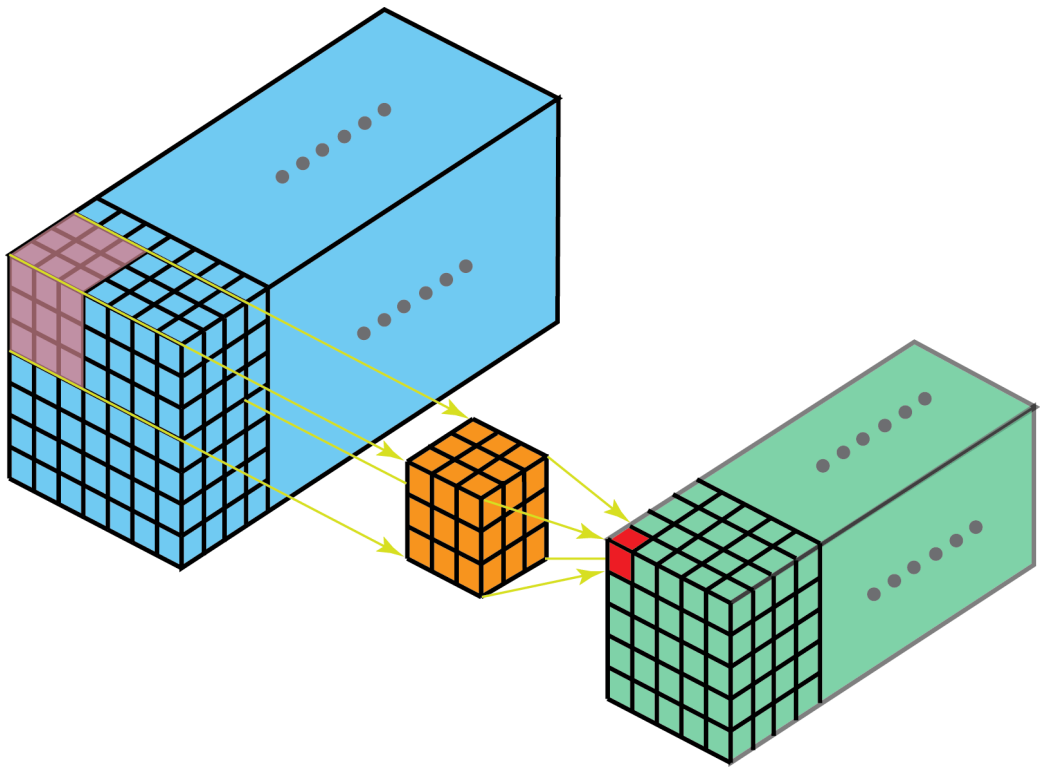
U-Net (2015)

- Convolution/deconvolution architecture for tissue segmentation.
- Convolutions downsample the feature maps (and increase the # channels).
- Deconvolutions restore high-resolution image.
- Skip connections allow to transfer information from intermediate representations to the deconvolutions.



C3D or 3D ConvNets (2015)

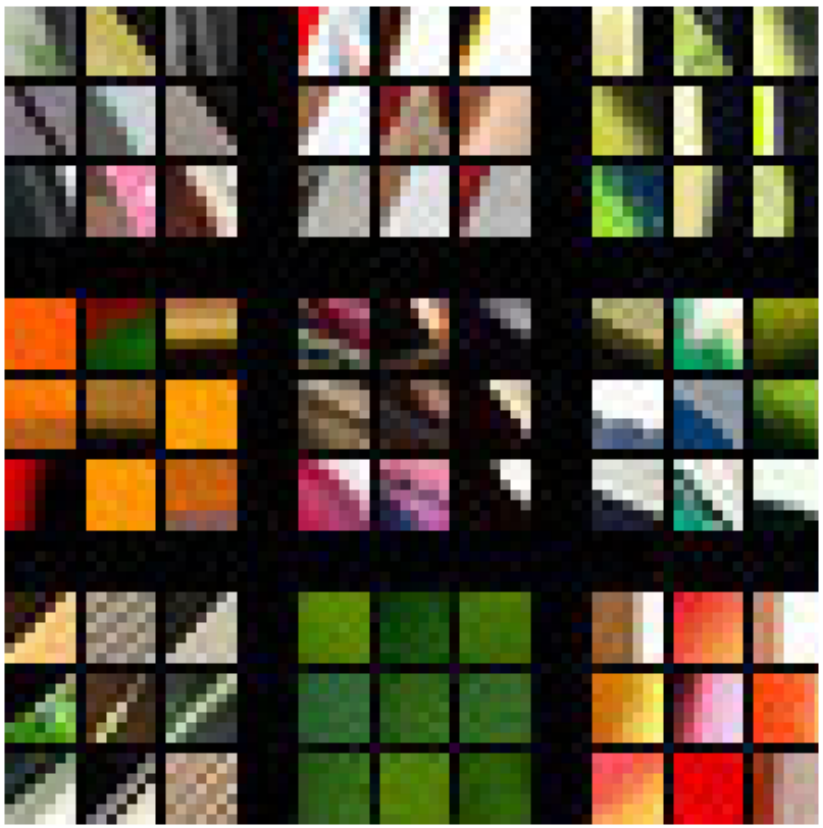
- Consider convolutional filters with less channels than the input.
- The kernel can move also along channels, and convolve the input.
- Can be used to process video frames (contactenated in a cube).



Understanding activations in CNN

Higher layers, more complicated concepts (Zeiler & Fergus 2014).

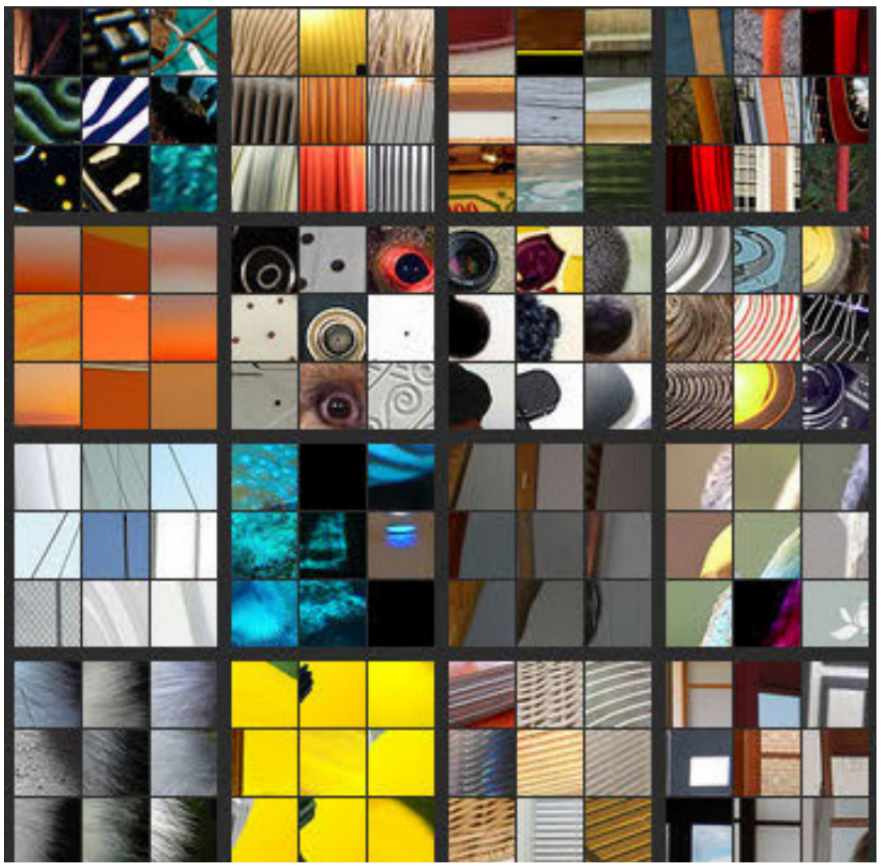
Layer 1: simple edges and color detectors.



Understanding activations in CNN

Higher layers, more complicated concepts (Zeiler & Fergus 2014).

Layer 2: corners, centers, ...



Understanding activations in CNN

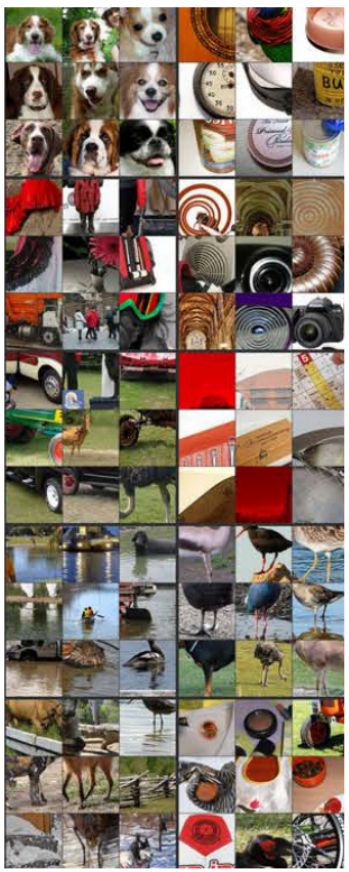
Higher layers, more complicated concepts (Zeiler & Fergus 2014).

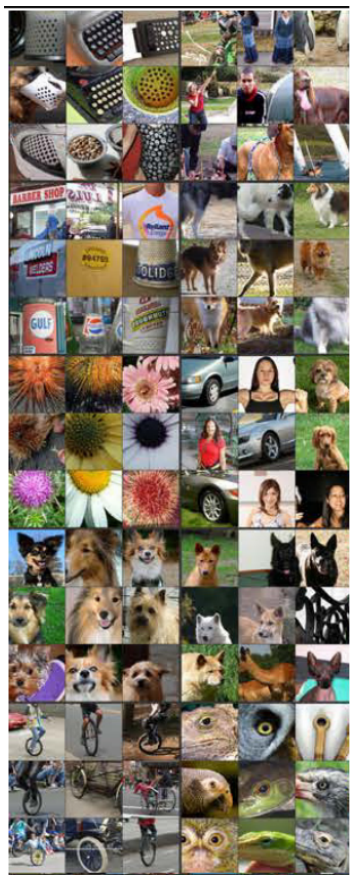
Layer 3: various object parts.

Understanding activations in CNN

Higher layers, more complicated concepts (Zeiler & Fergus 2014).

Layer 4 & 5: discriminative object parts/full objects.





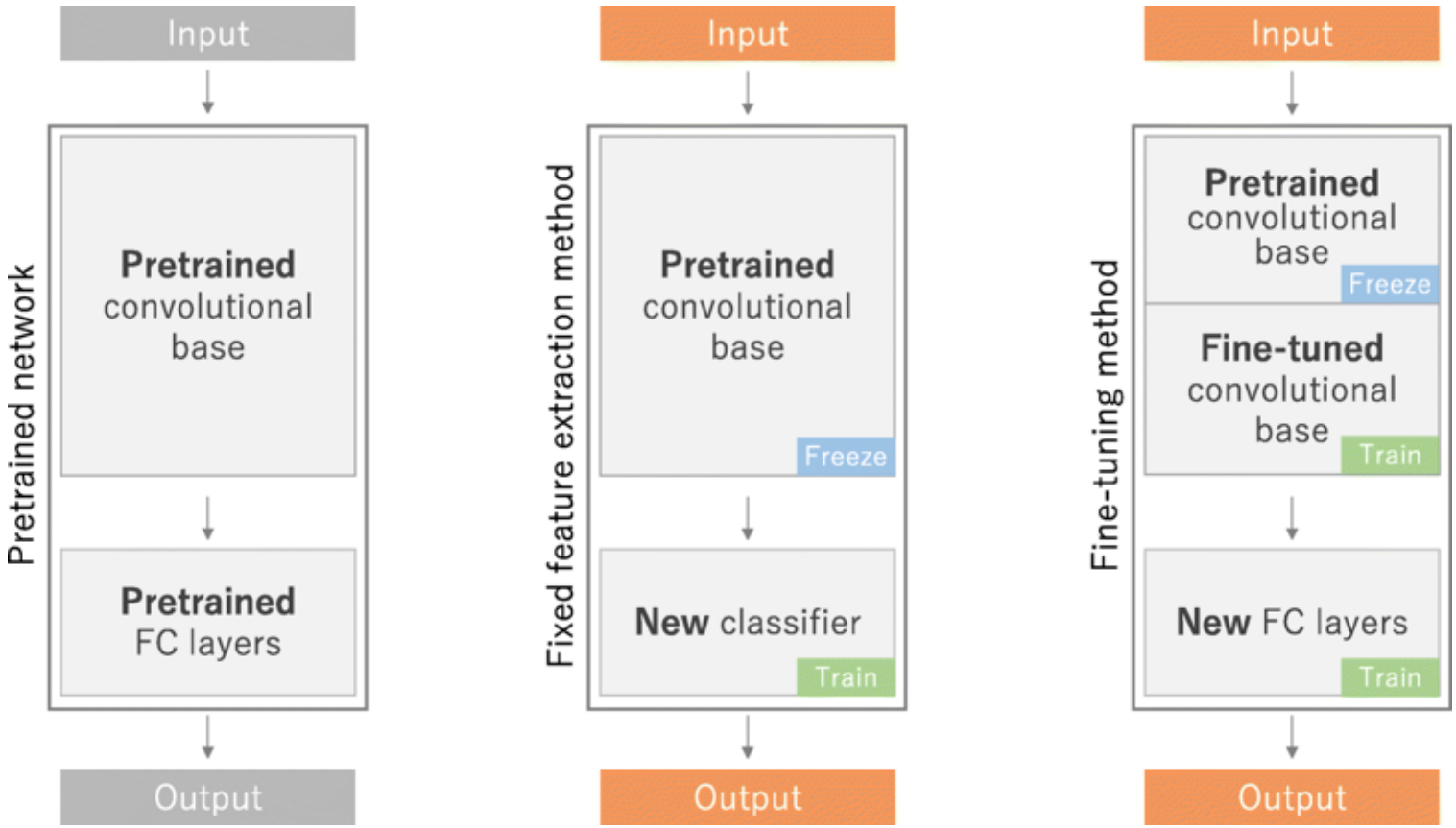
Finetunning pre-trained CNNs

- Early CNN layers extract generic features that seem useful for different tasks.
  Object localization, semantic segmentation, action recognition, etc.

- On some datasets too little training data to learn CNN from scratch.
  For example, only few hundred objects bounding box to learn from.

- Pre-train AlexNet/VGGnet/ResNet/DenseNet on large scale dataset.
  In practice mostly ImageNet classification: millions of labeled images.
- Fine-tune CNN weights for task at hand, perhaps modifying the architecture.
  - Replace classification layer, add bounding box regression, ...
  - Reduced learning rate and possibly freezing early network layers

Finetunning pre-trained CNNs (II)



From Yamashita et al "Convolutional neural networks: an overview and application in radiology" Insights into Imaging, 2018.

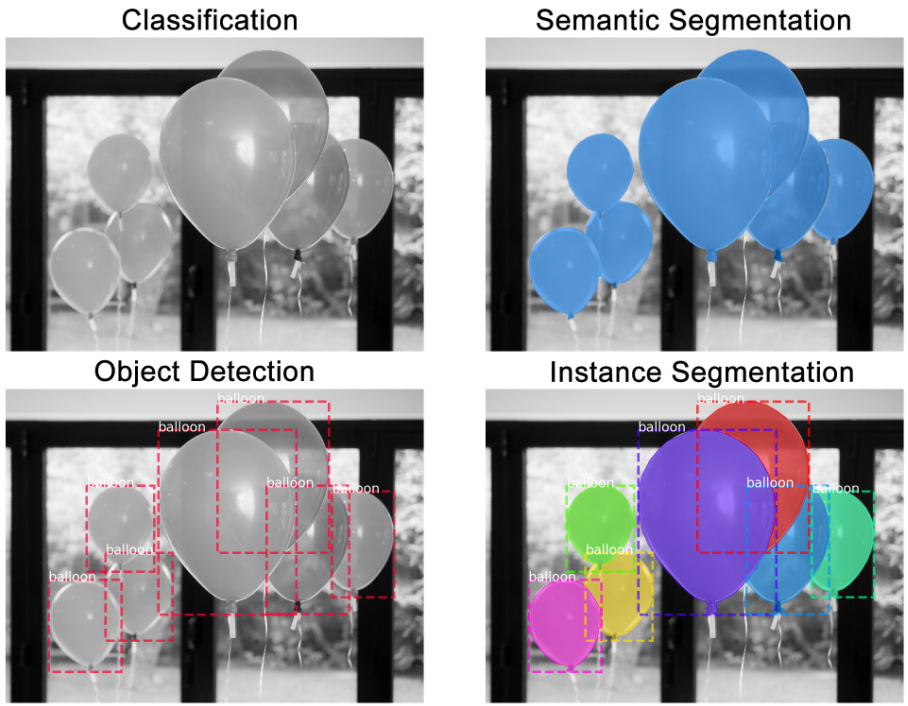
Fine-tunning example

Human body pose regression

- Remove the last layers of the network.
- Add new layers regression the limbs position.
- Change the loss e.g. Euclidean distance.
- Fine-tune previously trained layers and train the new ones from scratch.

Common uses and extensions

- Object category localisation
- Semantic segmentation
- Instance segmentation



Common uses and extensions

- Object category localisation
- Semantic segmentation
- Instance segmentation
- Multi-person tracking
- Combinations with other methods (e.g. CRF for segmentation)
- Multi-architecture training and search