Artificial Neural Networks and Deep Learning

A practical introduction

Sébastien Harispe
sebastien.harispe@mines-ales.fr

IMT Mines Alès

September 7, 2021

v1

Context and objectives

Context and objectives

Context

Context

Artificial Neural Networks (ANNs): sun of the New AI Summer

ANNs are the current rock stars of AI.
Have you ever heard about Deep Learning?

AI victory over Go grandmaster

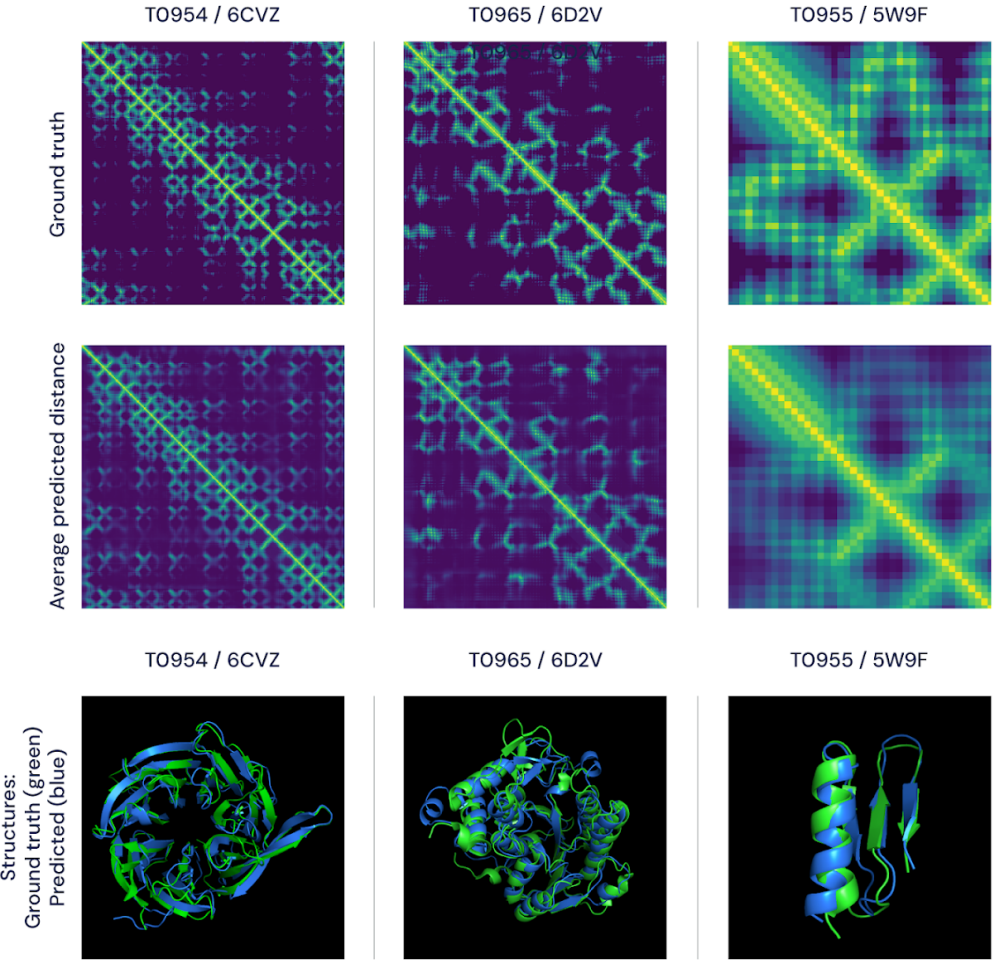
[source]

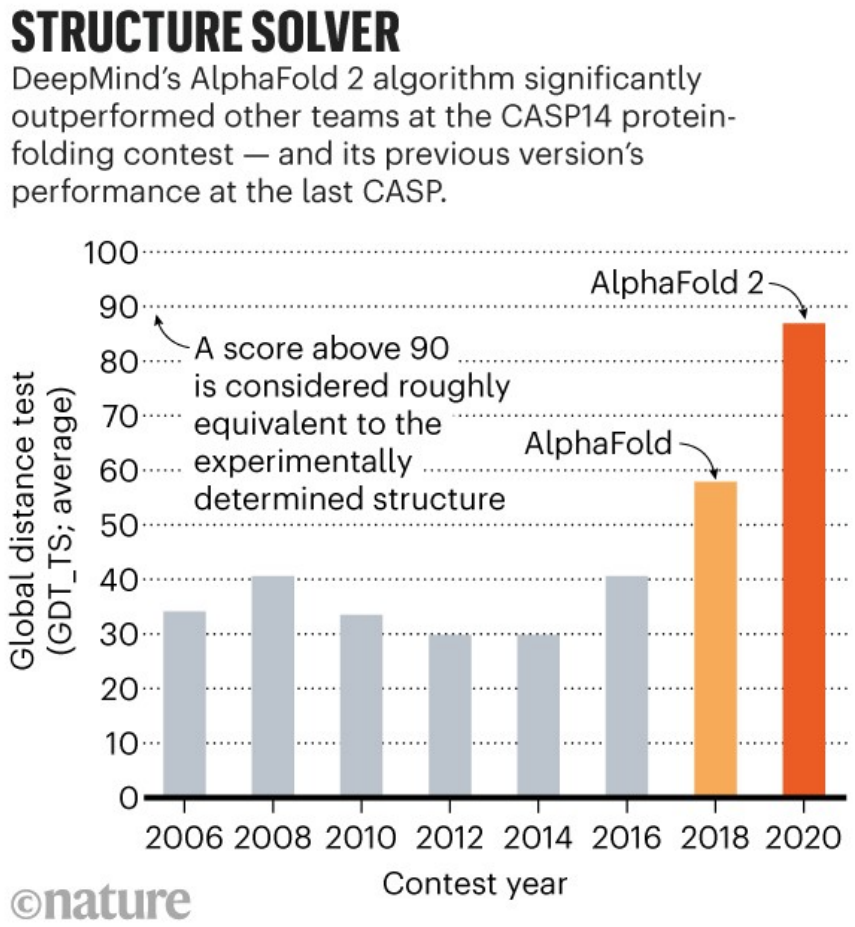
Lee Sedol vs AlphaGo (AI - DeepMind), see AlphaGo - The Movie (YouTube)

Go: one of the most challenging classical game for AI because of its complexity. The strongest Go computer programs could only play at the level of human amateurs. Standard AI methods, which test all possible moves and positions using a search tree, can't handle the sheer number of possible moves [...]. [source]

AlphaFold: Protein folding problem (DeepMind)

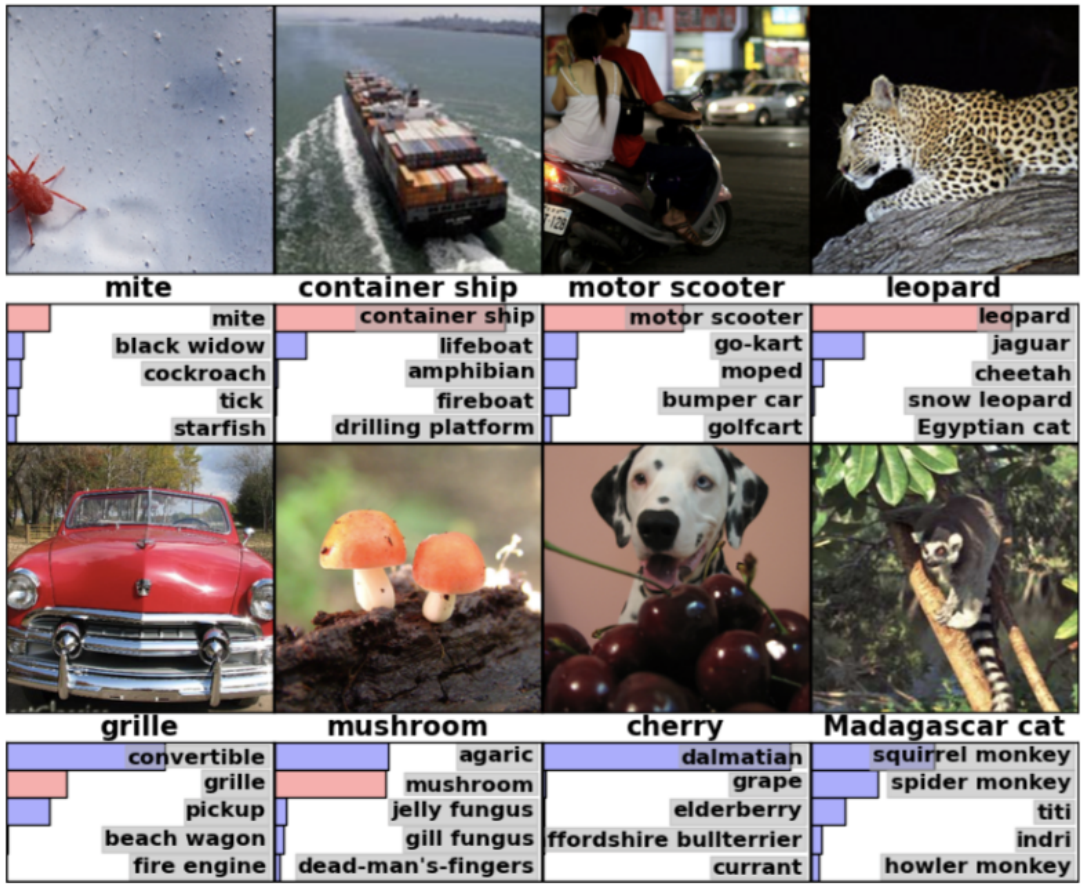
ANNs used (i) to predict a distribution of distances between every pair of protein residues and (ii) to estimate how good is the proposed structure.





[source]

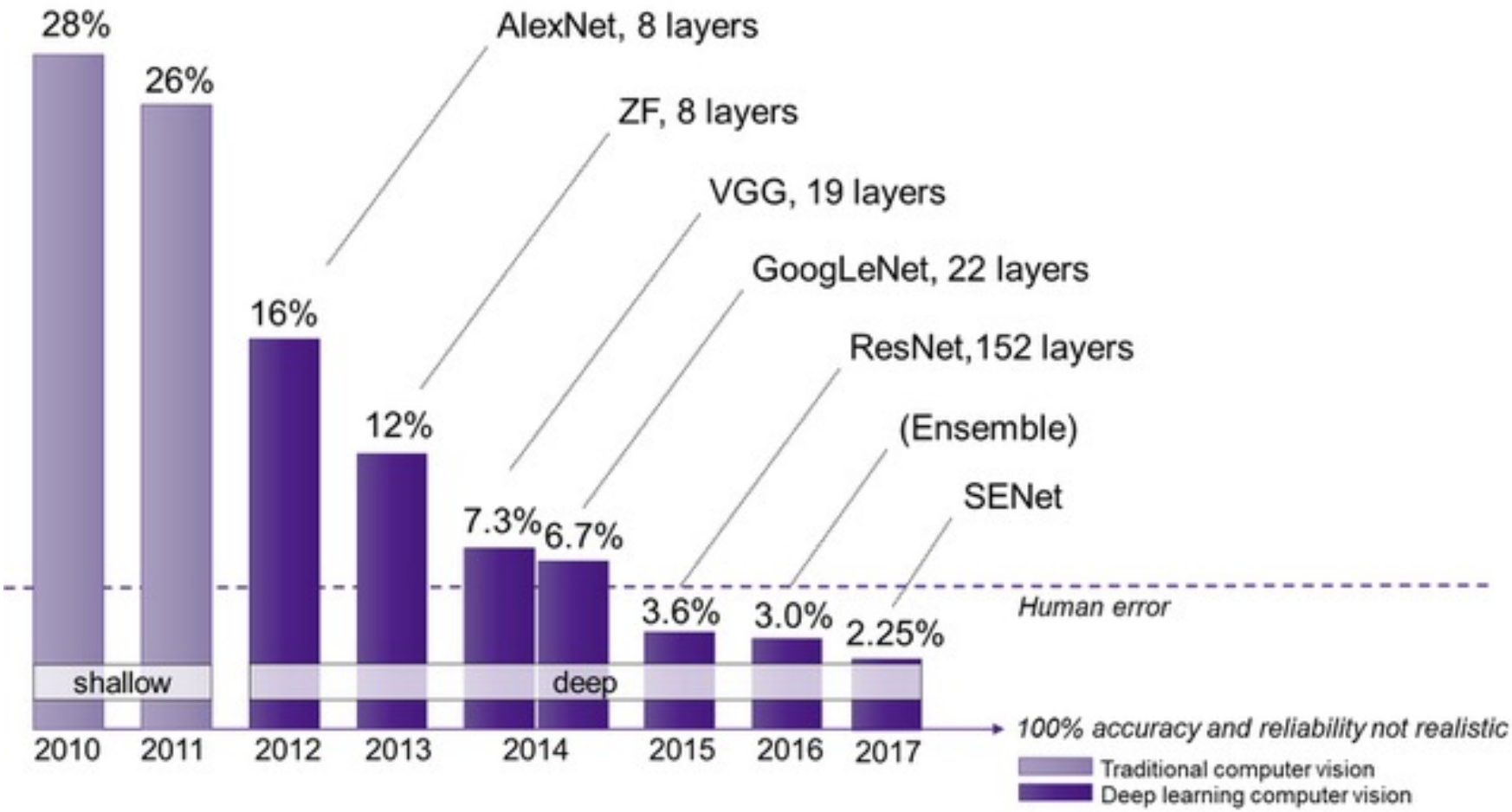
Imagenet Large Scale Vision Recognition Challenge



ImageNet 5-top prediction example.

[source]

Imagenet Large Scale Vision Recognition Challenge



Top-5 classification results show that deep learning is surpassing human accuracy.

[source]

Image Generation

Which one is generated?

Image Generation

Which one is generated?

All of them

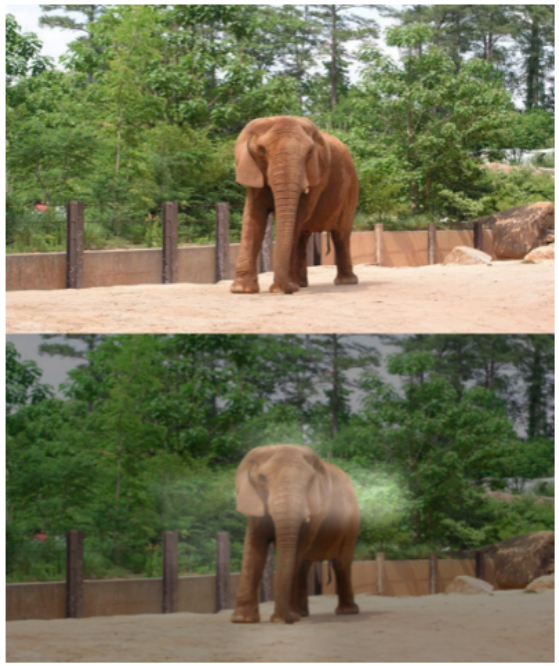
`https://thispersondoesnotexist.com/`

[source]

Caption Generation

(a) a person is skiing down a snowy slope

(b) a herd of cattle grazing on a lush green hillside



(c) an elephant is standing in the field with trees

Examples of generated captions with attention visualization for the underlined words

[source]

Autonomous vehicles

Deep learning is a central component for autonomous vehicles [source]

Examples:

- `https://www.youtube.com/watch?v=m3-QzTFxoUg&ab_channel=ScottKubo`
- `https://www.youtube.com/watch?v=NsSeCh3uWds&ab_channel=ViralTech`

Cancer detection

> IEEE Trans Med Imaging. 2020 Apr;39(4):1184-1194. doi: 10.1109/TMI.2019.2945514. Epub 2019 Oct 7.

Deep Neural Networks Improve Radiologists' Performance in Breast Cancer Screening

Nan Wu, Jason Phang, Jungkyu Park, Yiqiu Shen, Zhe Huang, Masha Zorin, Stanislaw Jastrzebski, Thibault Fevry, Joe Katsnelson, Eric Kim, Stacey Wolfson, Ujas Parikh, Sushma Gaddam, Leng Leng Young Lin, Kara Ho, Joshua D Weinstein, Beatriu Reig, Yiming Gao, Hildegard Toth, Kristine Pysarenko, Alana Lewin, Jiyon Lee, Krystal Airola, Eralda Mema, Stephanie Chung, Esther Hwang, Naziya Samreen, S Gene Kim, Laura Heacock, Linda Moy, Kyunghyun Cho, Krzysztof J Geras

PMID: 31603772 PMCID: PMC7427471 DOI: 10.1109/TMI.2019.2945514

- *AUC of 0.895 in predicting the presence of cancer in the breast.*
- *A hybrid model, averaging the probability of malignancy predicted by a radiologist with a prediction of our neural network, is more accurate than either of the two separately*

[source]

Text Generation

Have we just witnessed a quantum leap in artificial intelligence?

Asking GPT-3 (a Deep Learning model) why it has so entranced the tech community:

GPT-3 generated text: *I spoke with a very special person whose name is not relevant at this time, and what they told me was that my framework was perfect. If I remember correctly, they said it was like releasing a tiger into the world.*

[source]

Speech Generation

WaveNet generates realistic human-sounding speech that reduced the gap between computer and human performance by over 50%, when it was introduced.

Listen to the examples:

`https://deepmind.com/blog/article/wavenet-generative-model-raw-audio`

[source]

Deep Learning: current state

Deep learning is part of a broader family of machine learning methods based on artificial neural networks [source]

Deep Learning keeps obtaining **impressive state-of-the-art results** in **several domains** (Natural Language, Image and Sound processing, Complex games, symbolic problems...), with a **broad range of applications** (Healthcare, autonomous vehicle...).

[1] without fine-tuning (you'll understand this later).

Deep Learning: current state

Deep learning is part of a broader family of machine learning methods based on artificial neural networks [source]

Deep Learning keeps obtaining **impressive state-of-the-art results** in **several domains** (Natural Language, Image and Sound processing, Complex games, symbolic problems...), with a **broad range of applications** (Healthcare, autonomous vehicle...).

Current results are not always good enough for production shifts, a lot of work still has to be done (performance, ethics...), but Deep Learning offers great and stimulating improvements.

[1] without fine-tuning (you'll understand this later).

Deep Learning: current state

Deep learning is part of a broader family of machine learning methods based on artificial neural networks [source]

Deep Learning keeps obtaining **impressive state-of-the-art results** in **several domains** (Natural Language, Image and Sound processing, Complex games, symbolic problems...), with a **broad range of applications** (Healthcare, autonomous vehicle...).

Current results are not always good enough for production shifts, a lot of work still has to be done (performance, ethics...), but Deep Learning offers great and stimulating improvements.

Is Old-fashion Machine Learning (what you know so far) dead?

Short answer: No, far from it. As we will see there are several contexts in which Deep Learning is not necessarily recommended (e.g., small data[1]).

[1] without fine-tuning (you'll understand this later).

Context and objectives

Course Objectives

Objectives

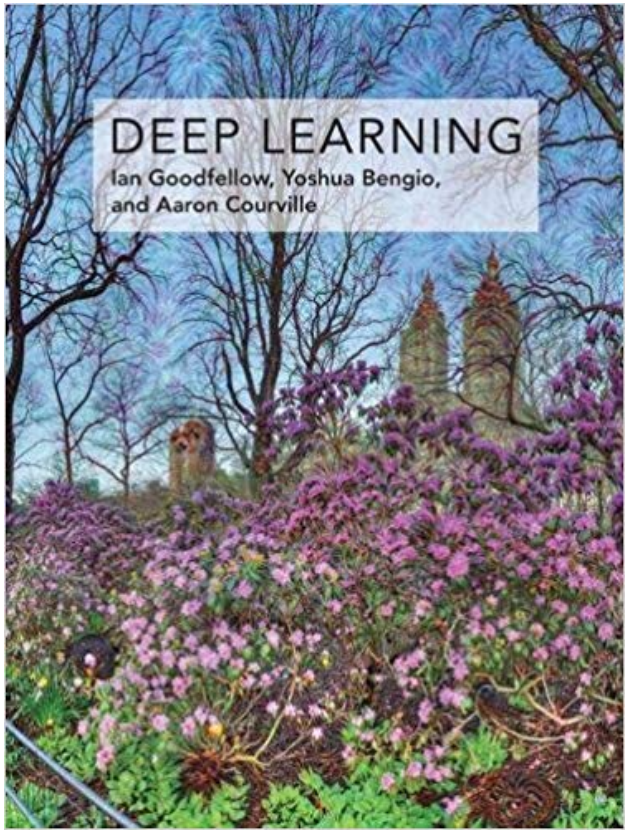
Introduction to Artificial Neural Networks and important ANN architectures

What are the aims of this course?

We aim to provide a short introduction to:

- ANN as a Machine Learning model. *Almost no course (almost)*.
- Important concepts related to ANNs and their use.
- Practical use and development of ANNs in the context of Deep Learning.

References



Material used in this course:

`https://www.deeplearningbook.org`

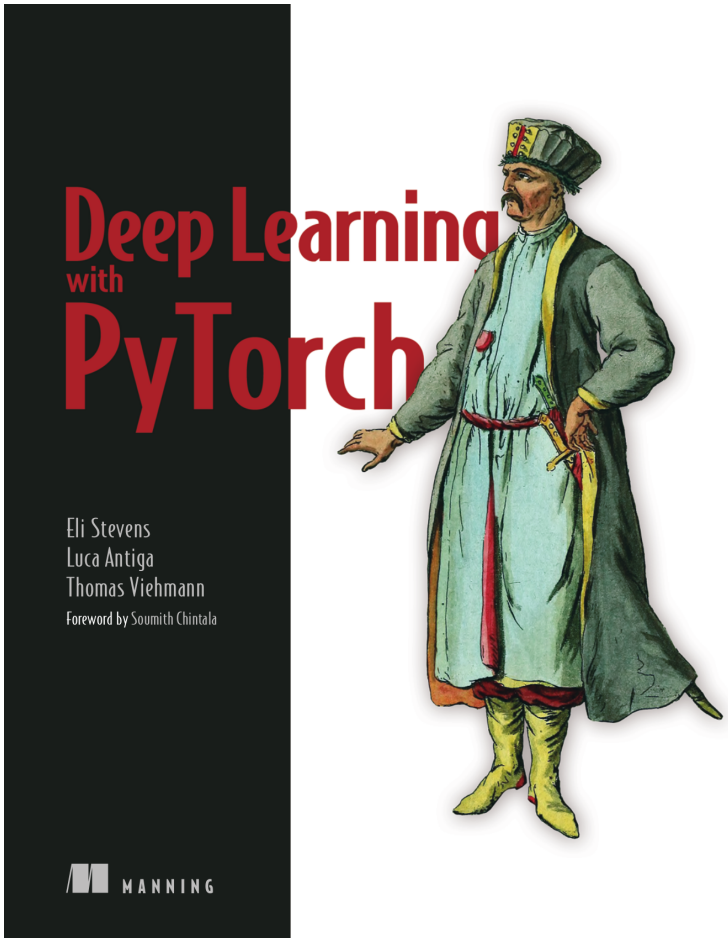
`https://www.wikipedia.org`

Keras

PyTorch

Wikipedia
The Free Encyclopedia

References



Material used in this course:

`https://pytorch.org/assets/deep-learning/Deep-Learning-with-PyTorch.pdf`

PyTorch

`https://pytorch.org`

Practical sessions on Google Colab

`https://colab.research.google.com`

You can also use your own local install (Jupyter notebook).
Better experience with GPU tho.

General Introduction

General Introduction

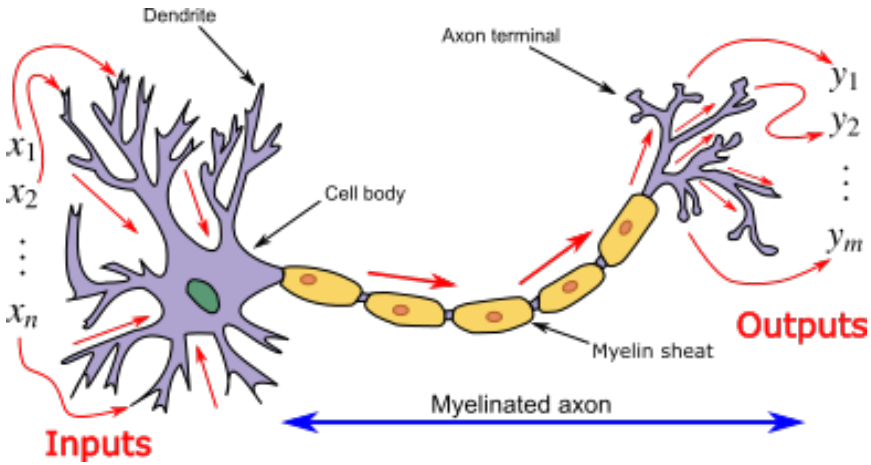
Artificial Neural Networks: the concept

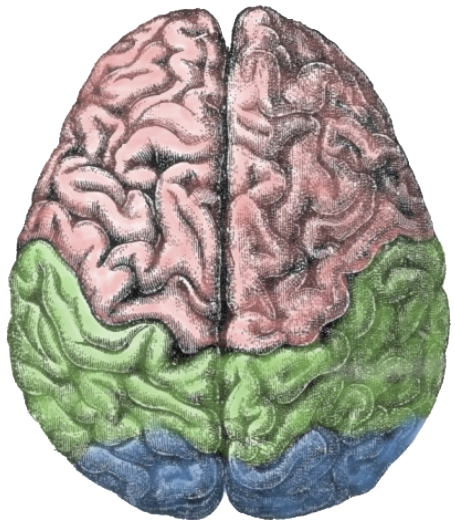
Artificial Neural Networks (ANNs)

Artificial Neural Networks are Machine Learning models:

- *Loosely inspired* by biological neural networks.
- Studied from the early days of AI (McCulloch & Pitts, 1943).
- Widely used in numerous domains and the primary cause of recent major breakthroughs in AI (Deep Learning is ANN).

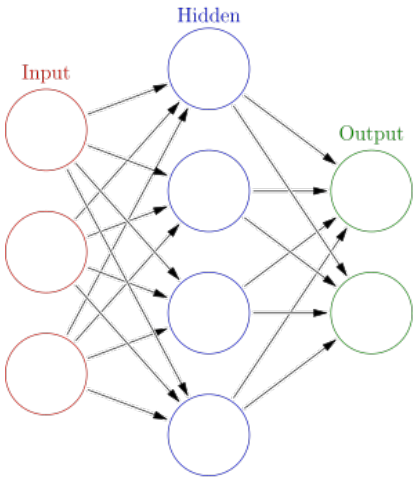
ANN are state-of-the-art techniques for solving numerous Machine Learning problems!

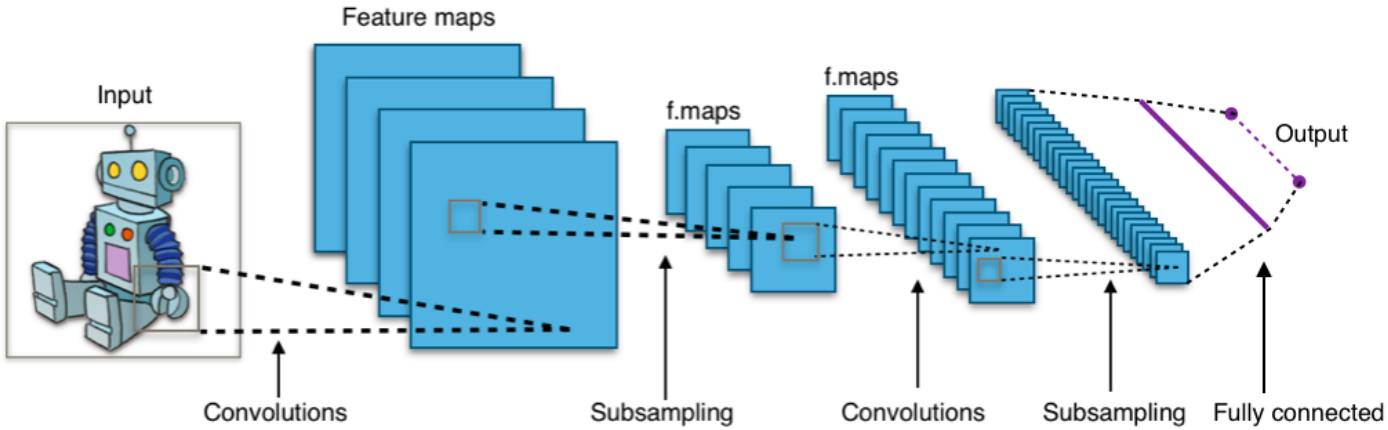




Artificial Neural Networks (ANNs)

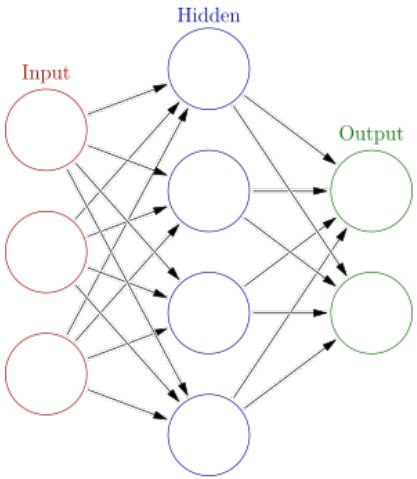
An ANN is a network of computing units called *artificial neurons*.
Connections between neurons express an information flow.
Supervised setting: the ANN generates a prediction from the input data.

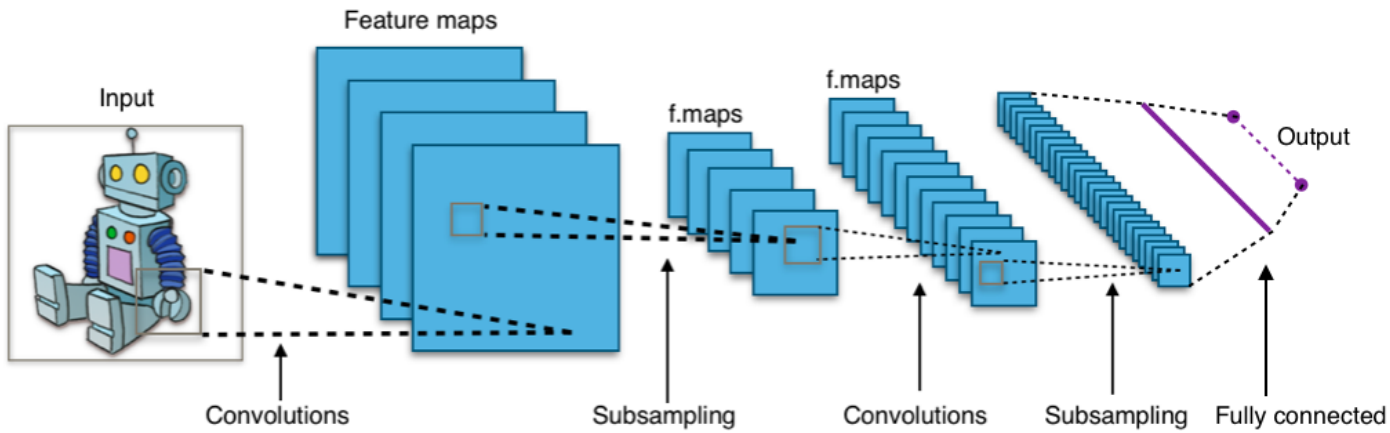




Artificial Neural Networks (ANNs)

An ANN is a network of computing units called *artificial neurons*.
Connections between neurons express an information flow.
Supervised setting: the ANN generates a prediction from the input data.



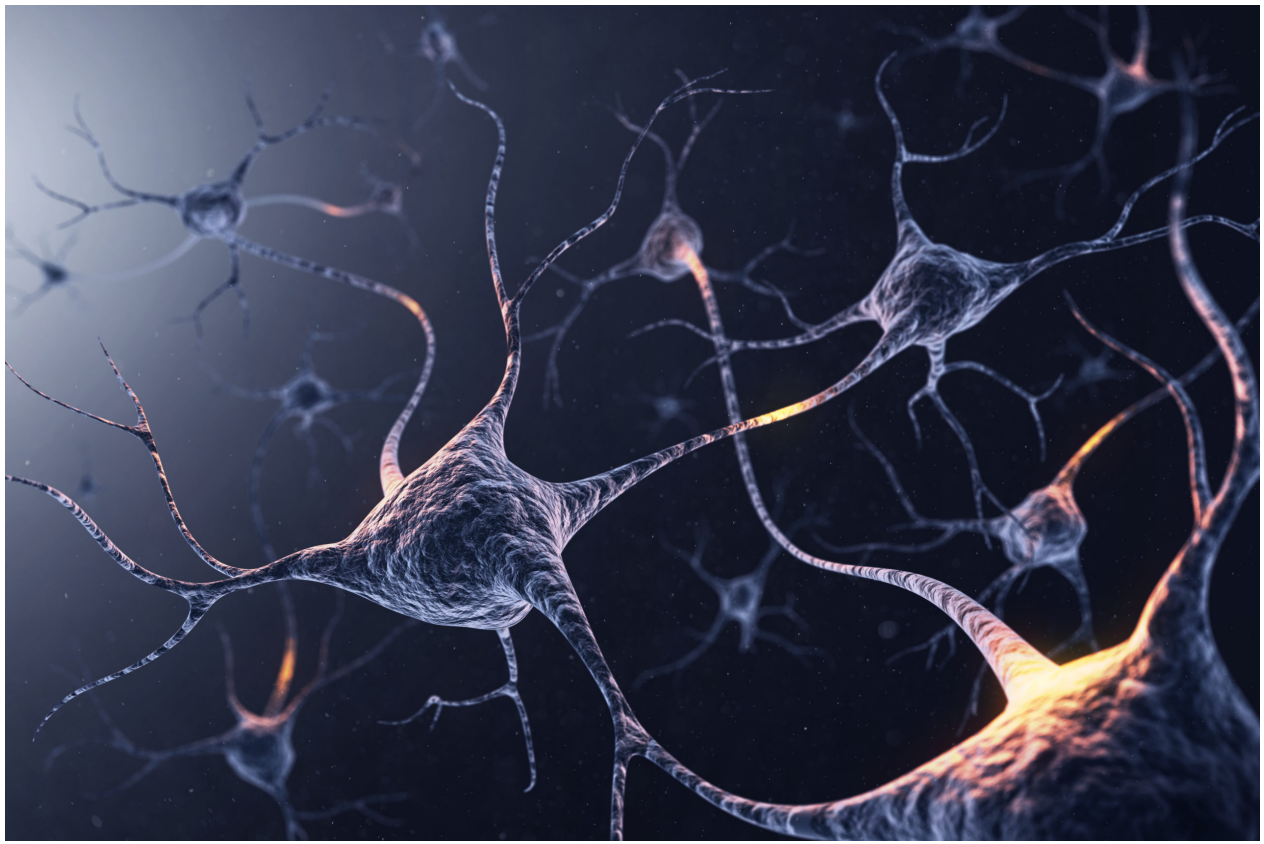


A large diversity of ANNs have been studied in the literature: recurrent, convolutional, GAN...

In this course we'll introduce several ANN models,
and show how they can be used, mostly in a supervised setting.

Artificial Neural Networks (ANNs)

An ANN is a network of computing units called *artificial neurons*.
Let's introduce what we call a neuron.



Source: Viaframe/Getty Image

General Introduction

Artificial Neuron

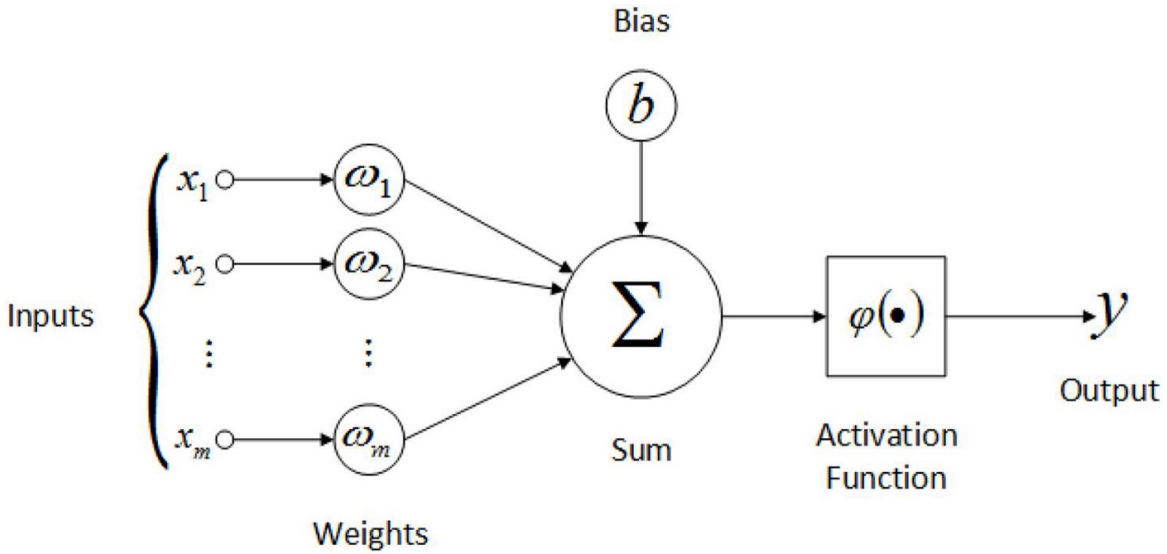
Artificial neuron

Artificial neurons are the elementary units of ANNs.
A neuron is simply a function that takes inputs and produce an output :

$$f : \mathbb{R}^m \to \mathbb{R}$$

Input values (scalar values) are usually separately weighted and summed.
The sum is then passed to an *activation function* producing the output.

For an input $x \in \mathbb{R}^m$ ($x = [x_1, \ldots, x_m]$), a neuron is as follows:



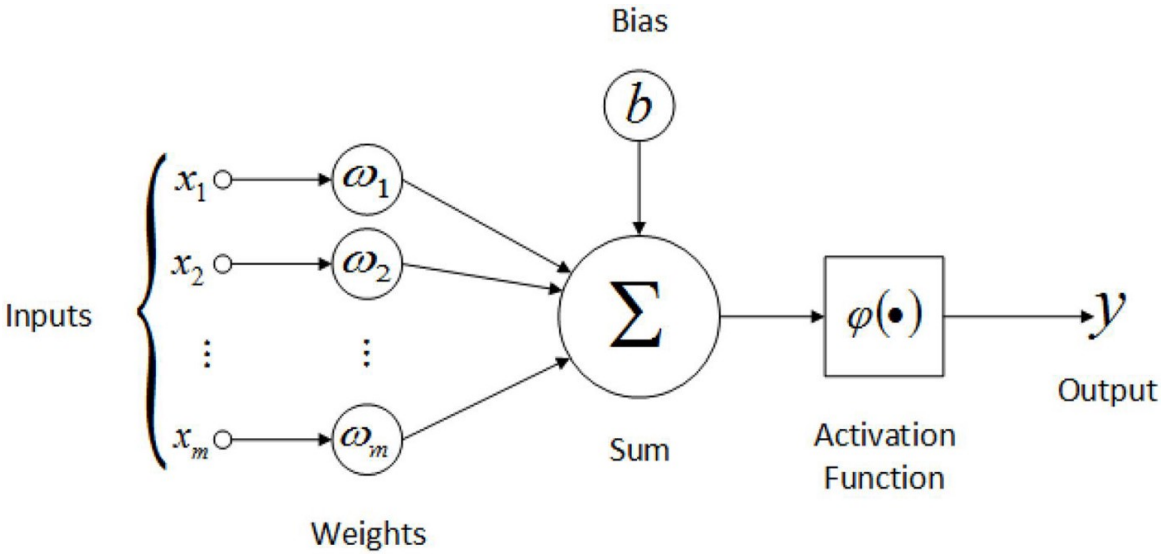
Artificial neuron

Artificial neurons are the elementary units of ANNs.
A neuron is simply a function that takes inputs and produce an output :

$$f : \mathbb{R}^m \to \mathbb{R}$$

Input values (scalar values) are usually separately weighted and summed.
The sum is then passed to an *activation function* producing the output.

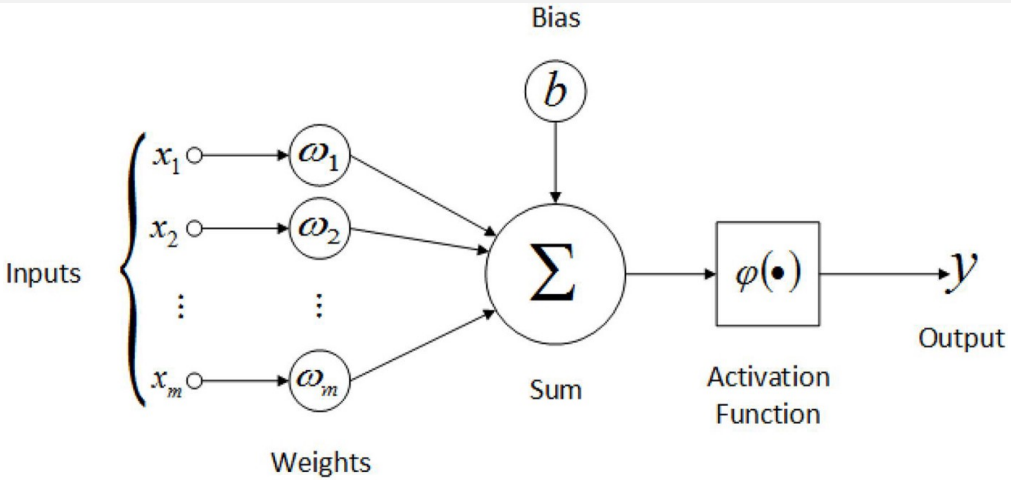
For an input $x \in \mathbb{R}^m$ ($x = [x_1, \ldots, x_m]$), a neuron is as follows:



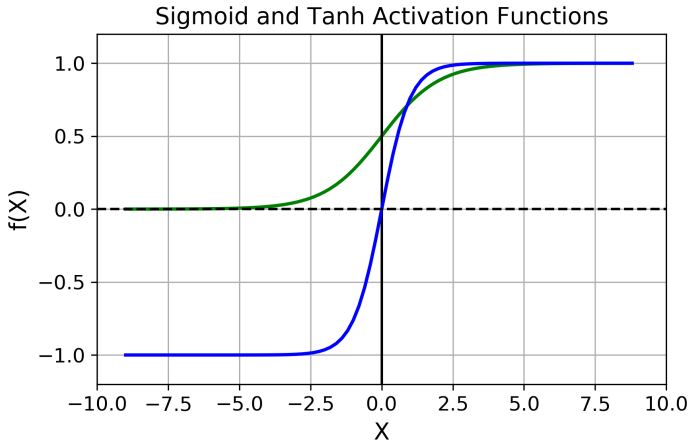
$$y = \phi\left(\sum_{j=1}^{m} w_j x_j + b\right)$$

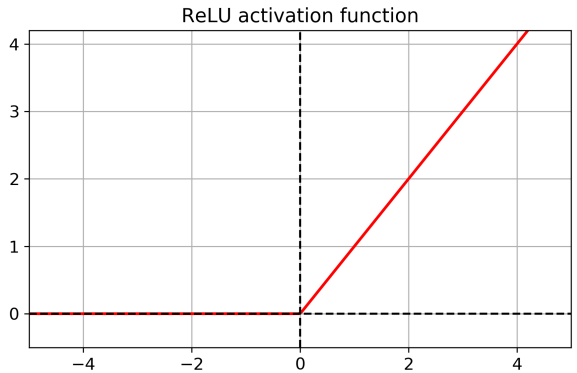
Artificial neuron

$$y = \phi(\sum_{j=1}^{m} w_j x_j + b)$$



The activation function ϕ is a threshold function that will *activate* or not the neuron based on the value of the biased weighted sum of its inputs.





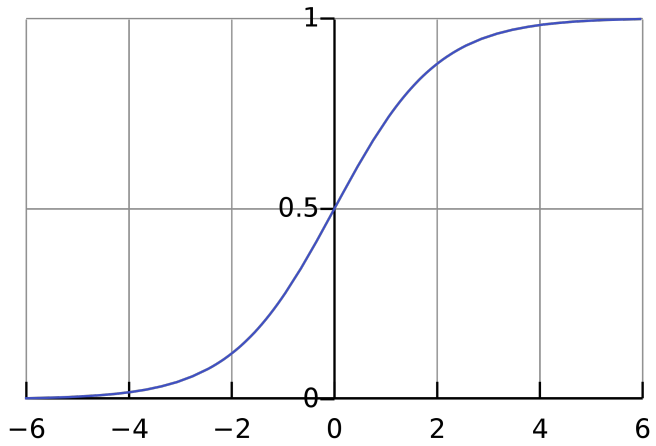
e.g. $\phi_{ReLU}(z) = max(0, z)$

Activation functions

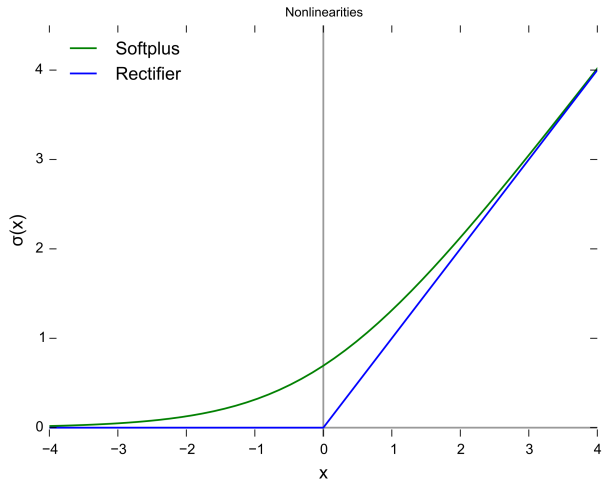
An activation function ϕ generates the output of a neuron by applying a transformation on a value precomputed based on its inputs.

- Various formulations exist.
- Most often non-linear functions.
- Most often differentiable functions.

General form $\phi : \mathbb{R} \rightarrow \mathbb{R}$



Logistic function $\phi_{sigmoid}(z) = \frac{1}{1+e^{-z}}$



Rectified Linear (ReLU) in blue.
$\phi_{ReLU}(z) = max(0, z)$

From artificial neurons to ANNs

As we will see an Artificial Neural Network (ANN) is nothing but a network of interconnected neurons.

1. Some neurons will take the input data as inputs.
2. Others will take as input the values produced by other neurons.

Link with biological neuron:

- A neuron input represents (i) a stimulus, or (ii) excitatory or inhibitory postsynaptic potentials at neural dendrites.
- A neuron output represents the neuron's action potential transmitted along its axon.

Recall Logistic Regression

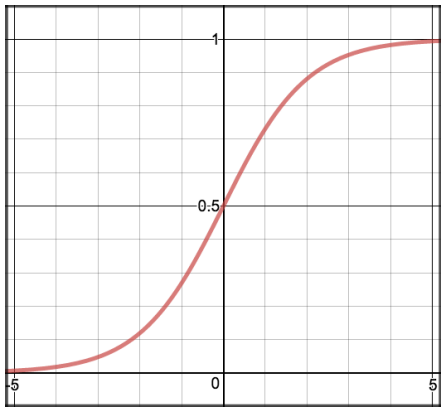
You've already seen something similar in a binary classification setting. Defining logistic regression, for an input $x \in \mathbb{R}^m$ (label $y_x \in \{0, 1\}$):

$$\hat{\mathbb{P}}(y_x = 1) = \phi_{sigmoid}(\sum_{j=1}^{m} w_j x_j + b)$$

$$\hat{\mathbb{P}}(y_x = 0) = 1 - \hat{\mathbb{P}}(y_x = 1)$$

with $\phi_{sigmoid}$ the sigmoid function

$$\phi_{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$



$\phi_{sigmoid} : \mathbb{R} \to [0, 1]$ (valid probability)

Then we defined the following decision rule:

$$\begin{cases} \hat{\mathbb{P}}(y_x = 1) \geq 0.5 \implies \text{class} = 1 \\ \hat{\mathbb{P}}(y_x = 1) < 0.5 \implies \text{class} = 0 \end{cases}$$

General Introduction

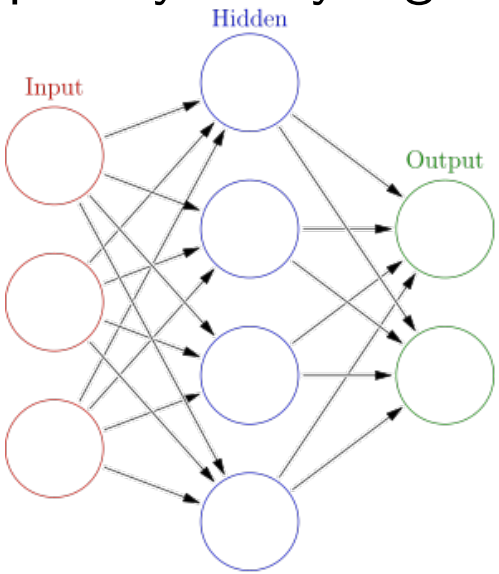
MultiLayer Perceptron (MLP)

MultiLayer Perceptron (MLP)

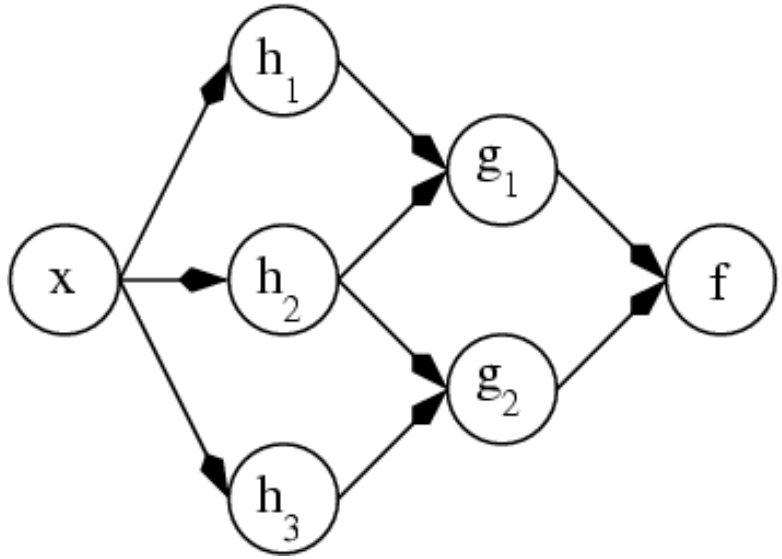
A Multilayer Perceptron (MLP) is a class of ANNs wherein connections between neurons do not form cycles (feedforward ANN).

Neurons are structured into layers:

- Input layer: models the input data.
- Hidden layer(s): internal computations performed by the network.
- Output layer: layer generating the prediction provided by the network.



simple MLP illustration



$f(g_1(h_1(x), h_2(x)), g_2(h_2(x), h_3(x)))$

Artificial neuron

An ANN can have any number of layers with any number of neurons.
Example: k-th artificial neuron of an ANN with a single hidden layer

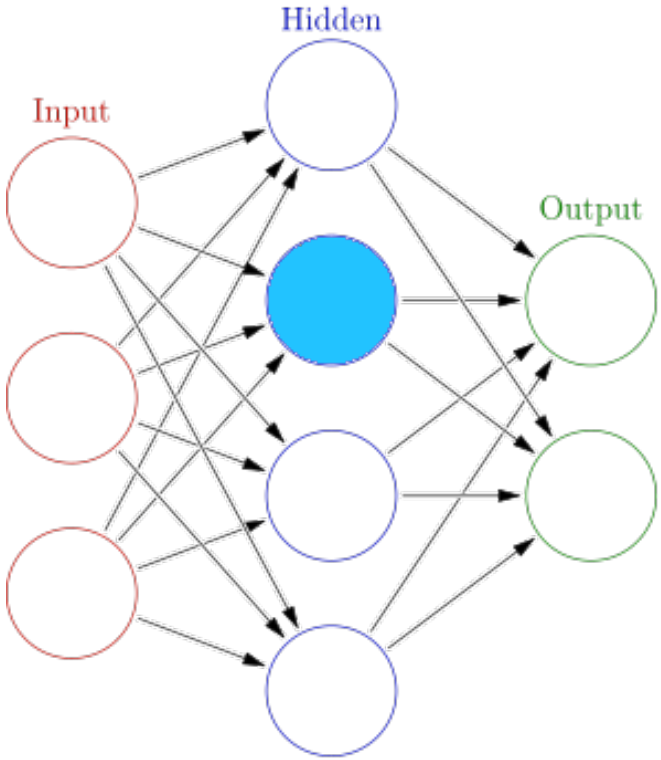
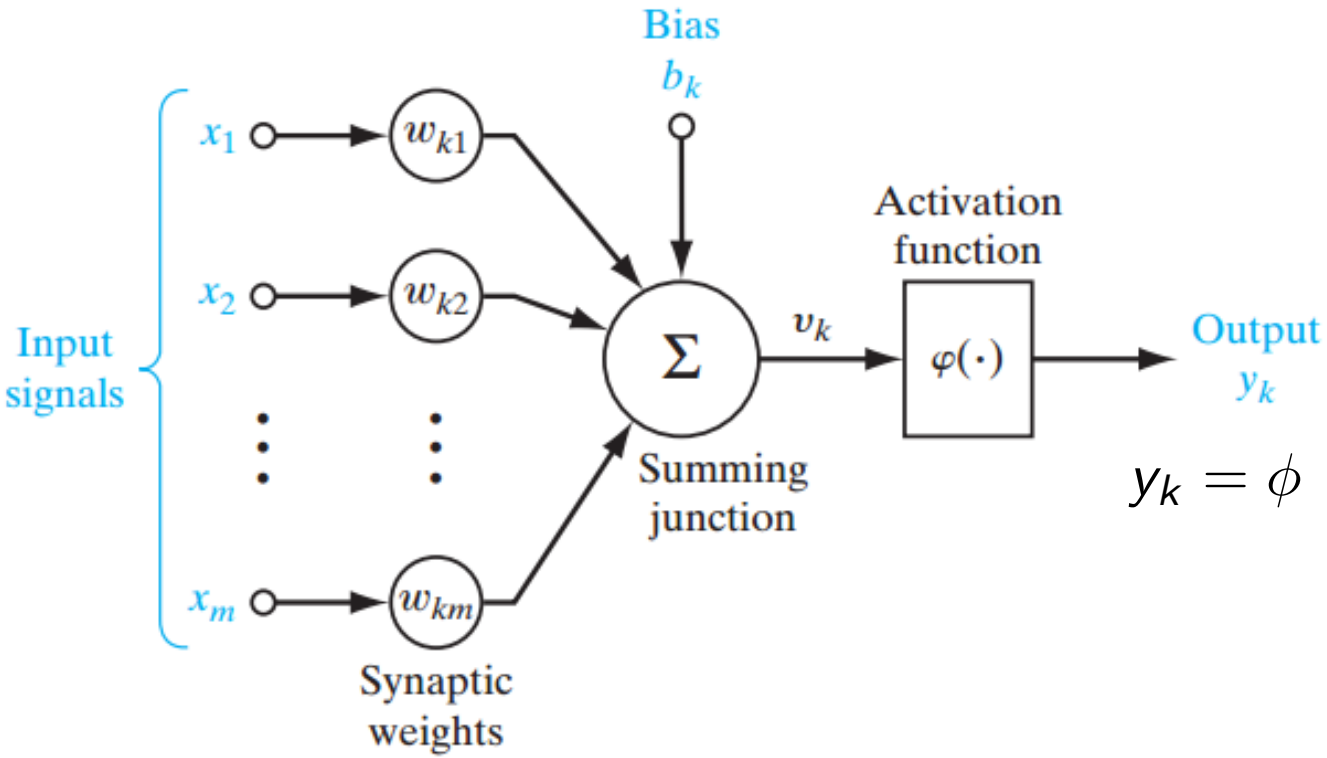


image adapted from `https://www.oreilly.com/library/view/apache-spark-deep/9781788474221/53ee18fb-e21f-43ff-bf53-58dcdbc91ab9.xhtml`

Artificial neuron

Example: k-th artificial neuron of an ANN with a single hidden layer



$$y_k = \phi\left(\sum_{j=1}^{m} w_{k,j}x_j + b_k\right)$$

source image

`www.oreilly.com/library/view/apache-spark-deep/9781788474221/53ee18fb-e21f-43ff-bf53-58dcdbc91ab9.xhtml`

An ANN is just a mathematical function

Consider a binary classification setting with $x \in \mathbb{R}^m$;
We want to predict $P(class = 1)$ (as we did for the logistic regression).

An ANN is just a mathematical function

Consider a binary classification setting with $x \in \mathbb{R}^m$;
We want to predict $P(class = 1)$ (as we did for the logistic regression).
Let's define a network with one hidden-layer and two neurons this way:

An ANN is just a mathematical function

Consider a binary classification setting with $x \in \mathbb{R}^m$;
We want to predict $P(class = 1)$ (as we did for the logistic regression).
Let's define a network with one hidden-layer and two neurons this way:

Hidden layer 1:

$$\text{neuron 1 (ReLU):} \qquad y_{1,1} = max(0, \sum_{j=1}^{m} w_{1,1,j} x_j + b_{1,1})$$

An ANN is just a mathematical function

Consider a binary classification setting with $x \in \mathbb{R}^m$;
We want to predict $P(class = 1)$ (as we did for the logistic regression).
Let's define a network with one hidden-layer and two neurons this way:

Hidden layer 1:

$$\text{neuron 1 (ReLU):} \qquad y_{1,1} = max(0, \sum_{j=1}^{m} w_{1,1,j} x_j + b_{1,1})$$

$$\text{neuron 2 (ReLU):} \qquad y_{1,2} = max(0, \sum_{j=1}^{m} w_{1,2,j} x_j + b_{1,2})$$

An ANN is just a mathematical function

Consider a binary classification setting with $x \in \mathbb{R}^m$;
We want to predict $P(class = 1)$ (as we did for the logistic regression).
Let's define a network with one hidden-layer and two neurons this way:

Hidden layer 1:

$$\text{neuron 1 (ReLU):} \qquad y_{1,1} = max(0, \sum_{j=1}^{m} w_{1,1,j} x_j + b_{1,1})$$

$$\text{neuron 2 (ReLU):} \qquad y_{1,2} = max(0, \sum_{j=1}^{m} w_{1,2,j} x_j + b_{1,2})$$

Output layer, 1 neuron with sigmoid ($\phi_{sigmoid} : \mathbb{R} \to [0, 1]$):

An ANN is just a mathematical function

Consider a binary classification setting with $x \in \mathbb{R}^m$;
We want to predict $P(class = 1)$ (as we did for the logistic regression).
Let's define a network with one hidden-layer and two neurons this way:

Hidden layer 1:

$$\text{neuron 1 (ReLU):} \qquad y_{1,1} = max(0, \sum_{j=1}^{m} w_{1,1,j} x_j + b_{1,1})$$

$$\text{neuron 2 (ReLU):} \qquad y_{1,2} = max(0, \sum_{j=1}^{m} w_{1,2,j} x_j + b_{1,2})$$

Output layer, 1 neuron with sigmoid ($\phi_{sigmoid} : \mathbb{R} \to [0, 1]$):

$$\hat{z} = w_{2,1,1} y_{1,1} + w_{2,1,2} y_{1,2} + b_{2,1} \qquad \textbf{prediction} \quad \hat{y} = \frac{1}{1 + e^{-\hat{z}}}$$

An ANN is just a mathematical function

Consider a binary classification setting with $x \in \mathbb{R}^m$;
We want to predict $P(class = 1)$ (as we did for the logistic regression).
Let's define a network with one hidden-layer and two neurons this way:

Hidden layer 1:

$$\text{neuron 1 (ReLU):} \qquad y_{1,1} = max(0, \sum_{j=1}^{m} w_{1,1,j} x_j + b_{1,1})$$

$$\text{neuron 2 (ReLU):} \qquad y_{1,2} = max(0, \sum_{j=1}^{m} w_{1,2,j} x_j + b_{1,2})$$

Output layer, 1 neuron with sigmoid ($\phi_{sigmoid} : \mathbb{R} \to [0, 1]$):

$$\hat{z} = w_{2,1,1} y_{1,1} + w_{2,1,2} y_{1,2} + b_{2,1} \qquad \textbf{prediction} \quad \hat{y} = \frac{1}{1 + e^{-\hat{z}}}$$

Function of our final ANN model:

$$\hat{y} = (1 + e^{-(w_{2,1,1} max(0, \sum_{j=1}^{m} w_{1,1,j} x_j + b_{1,1}) + w_{2,1,2} max(0, \sum_{j=1}^{m} w_{1,2,j} x_j + b_{1,2}) + b_{2,1})})^{-1}$$

An ANN is just a mathematical function

An ANN, like numerous other Machine Learning models,
is just a parameterized mathematical function.

Function of our ANN model:

$$\hat{y} = (1 + e^{-(w_{2,1,1} max(0, \sum_{j=1}^{m} w_{1,1,j} x_j + b_{1,1}) + w_{2,1,2} max(0, \sum_{j=1}^{m} w_{1,2,j} x_j + b_{1,2}) + b_{2,1})})^{-1}$$

For a given $x \in \mathbb{R}^m$, to what depends the prediction?

- First layer parameters
  - $\{w_{1,1,i}\}_{i=1,\dots,m}, b_{1,1}$
  - $\{w_{1,2,i}\}_{i=1,\dots,m}, b_{1,2}$
- Output layer parameters $w_{2,1,1}, w_{2,1,2}, b_{2,1}$

An ANN is just a mathematical function

As we will see, several ANN variations will be found depending on:

- the input we will process,
- the prediction setting (e.g, binary/multiclass classification, regression)
- ...

Note however that our ANNs will always be parametrized functions.

An ANN is just a mathematical function

As we will see, several ANN variations will be found depending on:

- the input we will process,
- the prediction setting (e.g, binary/multiclass classification, regression)
- ...

Note however that our ANNs will always be parametrized functions.

Solving the supervised learning problem will aim at findings the best values for the parameters with regards to our task and data.

An ANN is just a mathematical function

As we will see, several ANN variations will be found depending on:

- the input we will process,
- the prediction setting (e.g, binary/multiclass classification, regression)
- ...

Note however that our ANNs will always be parametrized functions.

Solving the supervised learning problem will aim at findings the best values for the parameters with regards to our task and data.

Identifying those parameters will enable us to obtain a usable function in order to make our predictions.

General Introduction

Additional information

Prior to providing details on how to train an ANN, we introduce new notations, terminology and concepts on which we will rely later.

Notations - Artificial neuron

/!\ *Notations will gradually change in the following sequence of slides.*

–

Recall our first MLP with one hidden layer and two neurons:

$$\hat{y} = (1 + e^{-(w_{2,1,1} max(0, \sum_{j=1}^{m} w_{1,1,j} x_j + b_{1,1}) + w_{2,1,2} max(0, \sum_{j=1}^{m} w_{1,2,j} x_j + b_{1,2}) + b_{2,1})})^{-1}$$

We will simplify the notations using linear algebra.

Notations - Artificial neuron

/!\ *Notations will gradually change in the following sequence of slides.*

–

Recall our first MLP with one hidden layer and two neurons:

$$\hat{y} = (1 + e^{-(w_{2,1,1} max(0, \sum_{j=1}^{m} w_{1,1,j} x_j + b_{1,1}) + w_{2,1,2} max(0, \sum_{j=1}^{m} w_{1,2,j} x_j + b_{1,2}) + b_{2,1})})^{-1}$$

We will simplify the notations using linear algebra.

First note that the k-th neuron output $y_k = \phi\left(\sum_{j=1}^{m} w_{k,j} x_j + b_k\right)$ can simply be written:

$$y_k = \phi\left(x W_k^T + b_k\right)$$

with $x \in \mathbb{R}^{1 \times m}$, $W_k^T \in \mathbb{R}^{m \times 1}$, and $b_k \in \mathbb{R}$.

Notations - Layer (single input)

Considering that we have a layer l with n neurons, we can pack all its parameters into a matrix of weights and a vector of bias.
Setting the same activation function to the n neurons, we can note:

$$y_l = \phi^* \left(x W_l^T + b_l \right)$$

with:

- $y_l \in \mathbb{R}^{1 \times n}$, one output for each of the n neurons of layer l,
- $x \in \mathbb{R}^{1 \times m}$, 1 input of m feature values,
- $W_l^T \in \mathbb{R}^{m \times n}$, m parameters for each of the n neurons of layer l.
- $b_l \in \mathbb{R}^{1 \times n}$ one bias for each of the n neurons of layer l.
- $\phi^* : \mathbb{R}^k \to \mathbb{R}^k$ such that $\phi^*(z) = [\phi(z_1), \ldots, \phi(z_k)]$

We will later allow simplifications if there is no ambiguity,
e.g. use ϕ instead of ϕ^*.

Notations - MLP (single input)

Using those notations we could rewrite our first MLP:

$$\hat{y} = (1 + e^{-(w_{2,1,1} max(0, \sum_{j=1}^{m} w_{1,1,j} x_j + b_{1,1}) + w_{2,1,2} max(0, \sum_{j=1}^{m} w_{1,2,j} x_j + b_{1,2}) + b_{2,1})})^{-1}$$

into:

$$\begin{aligned} \hat{y} &= (1 + e^{-\phi^*_{ReLU}(xW_1^T + b_1)W_2^T + b_2})^{-1} \\ &= \phi_{sigmoid}(\phi^*_{ReLU}(xW_1^T + b_1)W_2^T + b_2) \end{aligned}$$

with:

- $\hat{y} \in [0, 1]$, one output,
- $x \in \mathbb{R}^{1 \times m}$, 1 input of m feature values,
- $W_1^T \in \mathbb{R}^{m \times 2}$, m parameters for each of the 2 neurons of layer 1; $b_1 \in \mathbb{R}^{1 \times 2}$, one biais for each neuron.
- $W_2^T \in \mathbb{R}^{2 \times 1}$, 2 input values from layer 1 for one output neuron; $b_2 \in \mathbb{R}$ a single biais for the unique output neuron.

All model parameters are now stored into (W_1, b_1) and (W_2, b_2).

Notations - batch processing

Let's push it a bit further. Consider now that you've a *batch* of n inputs each characterized by m features. You want a prediction for each of these inputs. Let's pack these inputs into a matrix $x \in \mathbb{R}^{n \times m}$.

Our model is still easy to write

$$\hat{y} = \phi_{sigmoid}(\phi^*_{ReLU}(xW_1^T + b_1)W_2^T + b_2)$$

considering proper extensions of ϕ^* and $+$, i.e. element-wise functions. with:

- $x \in \mathbb{R}^{n \times m}$, n inputs of m feature values,
- $\hat{y} \in [0,1]^n$, n output probabilities,
- $W_1^T \in \mathbb{R}^{m \times 2}$, m parameters for each of the 2 neurons of layer 1; $b_1 \in \mathbb{R}^{1 \times 2}$ (summation applies element-wise to each of the n rows).
- $W_2^T \in \mathbb{R}^{2 \times 1}$, 2 input values from layer 1 for one output neuron; $b_2 \in \mathbb{R}$ (summation applies element-wise to each of the n rows).

Classification

We have seen an example of binary classification in which the probability $P(class(x) = 1)$ was produced by a single neuron based on the sigmoid activation function.

$$\text{final prediction (probability)} \quad \hat{y} = \frac{1}{1 + e^{-\hat{f}'(x)}}$$

You're not obliged to define your network this way.

It is also common to produce an output value per class you've to consider and then use the *softmax function* to produce a distribution probability.

The **Softmax function** takes as input a vector of n real numbers, and normalizes it into a probability distribution consisting of n probabilities:

$$\sigma(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{n} e^{z_j}}$$

This technique can be used in a mutliclass setting with n classes.

Regression

In a regression setting, for a given input, you want to predict a value $y \in \mathbb{R}^n$ (with sometimes $n = 1$).

In this case, defining your network, you just have to ensure the output layer has n neurons. The output values of these neurons will be your prediction.

Insights about ANN Training

Insights about ANN Training

Prerequisites

Global objective of Training

Reminder (cf. DL book)

A machine learning algorithm is an algorithm
that is able to *learn* from data.

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measures by P, improves with experience E. (Mitchell 1997)

- Tasks T: classification, regression...
- Performance measure P: accuracy, Mean Squared Error.
- Experience E: available data (supervised, unsupervised).

Global objective: supervised setting

To obtain a predictor $\hat{f}$ approximating a function f using a training set.
We want $\hat{f} \approx f$ outside the training set (cf. Generalization vs Overfitting).

Global objective of Training

Reminder (cf. DL book)

Considering a set of labeled data composed of:

- a training set on which we'll fit our model to obtain a predictor.
- a test set used for the final predictor evaluation.

Global objective of Training

Reminder (cf. DL book)

Considering a set of labeled data composed of:

- a training set on which we'll fit our model to obtain a predictor.
- a test set used for the final predictor evaluation.

Fitting the training data will imply minimizing a task dependent loss:

- Regression: MSE...
- Classification: Cross-entropy... (more on that later)

Global objective of Training

Reminder (cf. DL book)

Considering a set of labeled data composed of:

- a training set on which we'll fit our model to obtain a predictor.
- a test set used for the final predictor evaluation.

Fitting the training data will imply minimizing a task dependent loss:

- Regression: MSE...
- Classification: Cross-entropy... (more on that later)

Finally we want to:

- Make the training error small.
- Make the gap between the training and test error small.

Losses

Training requires defining a task dependent performance measure, i.e. a loss.

Considering:

- a labeled dataset $\mathcal{D} \in (\mathcal{X} \times \mathcal{Y})^n$,
- a predictor $\hat{f} : \mathcal{X} \to \mathcal{Y}$.

Regression ($\mathcal{Y}$ continuous)
E.g. Mean squared Error (MSE):

$$E_{MSE}(\hat{f}, \mathcal{D}) = \frac{\sum_{(x,y)\in\mathcal{D}} (\hat{f}(x) - y)^2}{|\mathcal{D}|}$$

Losses

Classification ($\mathcal{Y}$ categorical)

A probabilistic binary classifier $\hat{f}$, i.e. $\mathcal{Y} = \{0, 1\}$, estimates the probability that an input has the true label:

$$\hat{f}(x) = \hat{\mathbb{P}}[y_x = 1]$$

It is very common to define the loss on the output probability distribution in this case (recall logistic regression).

With $p = \hat{\mathbb{P}}[y_x = 1]$, the logistic loss (log loss) for the entry x is:

$$l(p, y) = -\left(y log(p) + (1 - y) log(1 - p)\right)$$

$$E_{logloss}(\hat{f}, \mathcal{D}) = \frac{\sum_{(x,y)\in\mathcal{D}} l(\hat{f}(x), y)}{|\mathcal{D}|}$$

Losses

Classification ($\mathcal{Y}$ categorical)

The cross-entropy can be used to consider a multiclass setting ($|\mathcal{Y}| > 2$). Used to compare two probability distributions p and q over $\mathcal{Y}$.

With p and q two discrete probability distributions over $\mathcal{X}$, the cross-entropy $H(p, q) \in [0, +\infty]$ is: $H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x)$

Losses

Classification ($\mathcal{Y}$ categorical)

The cross-entropy can be used to consider a multiclass setting ($|\mathcal{Y}| > 2$). Used to compare two probability distributions p and q over $\mathcal{Y}$.

With p and q two discrete probability distributions over $\mathcal{X}$, the cross-entropy $H(p, q) \in [0, +\infty]$ is: $H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x)$

For a given entry $(x, y) \in \mathcal{D}$, we here denote

- $y \in [0, 1]^{|\mathcal{C}|}$ the expected distribution (one hot vector)
- $\hat{f}(x) = \hat{y} \in [0, 1]^{|\mathcal{C}|}$ the predicted one.

$$H(y, \hat{y}) = -\sum_{i \in 1, \ldots, |\mathcal{Y}|} y_i \log \hat{y}_i$$

$$E_{CE}(\hat{f}, \mathcal{D}) = \frac{\sum_{(x,y) \in \mathcal{D}} H(y, \hat{f}(x))}{|\mathcal{D}|}$$

Insights about ANN Training

Gradient Descent

Training

Consider a parameterized ANN (e.g. MLP) & loss.

Training an ANN aims at finding the best parameters values (e.g. weights, bias) w.r.t a loss function and a training set.

$\Longrightarrow$ Requires solving a difficult non-convex optimization problem...

Training

Consider a parameterized ANN (e.g. MLP) & loss.

Training an ANN aims at finding the best parameters values (e.g. weights, bias) w.r.t a loss function and a training set.

$\Longrightarrow$ Requires solving a difficult non-convex optimization problem...

Gradient Descent is the de facto approach to use.

Two-step iterative process:

1. simply use partial derivative and the chain rule to compute the gradient of the error wrt the parameters.
2. use this gradient in order to update the parameters.

$$\theta_{t+1} := \theta_t - \eta \frac{\partial E(X, Y, \theta_t)}{\partial \theta_t}$$

with (X, Y) the training set, θ^t the parameters at time t, E the loss, η (eta)the learning rate.

Training: overview

Gradient descent is an iterative algorithm:

1. Compute error & gradient
2. If stop limit unreached: update the weights, go to 1

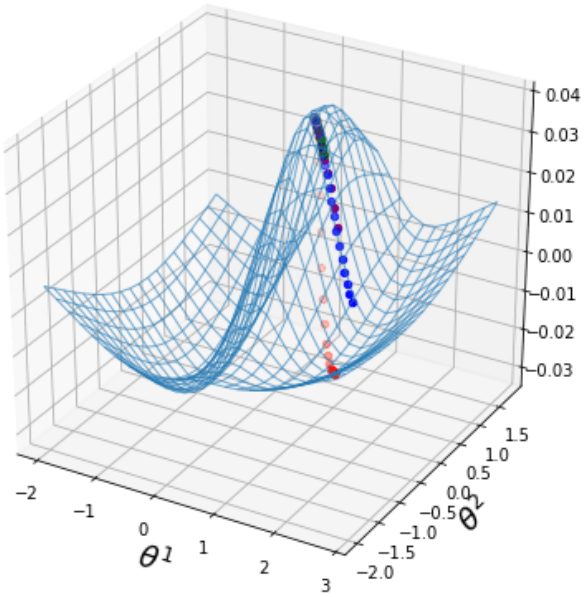
Most often:

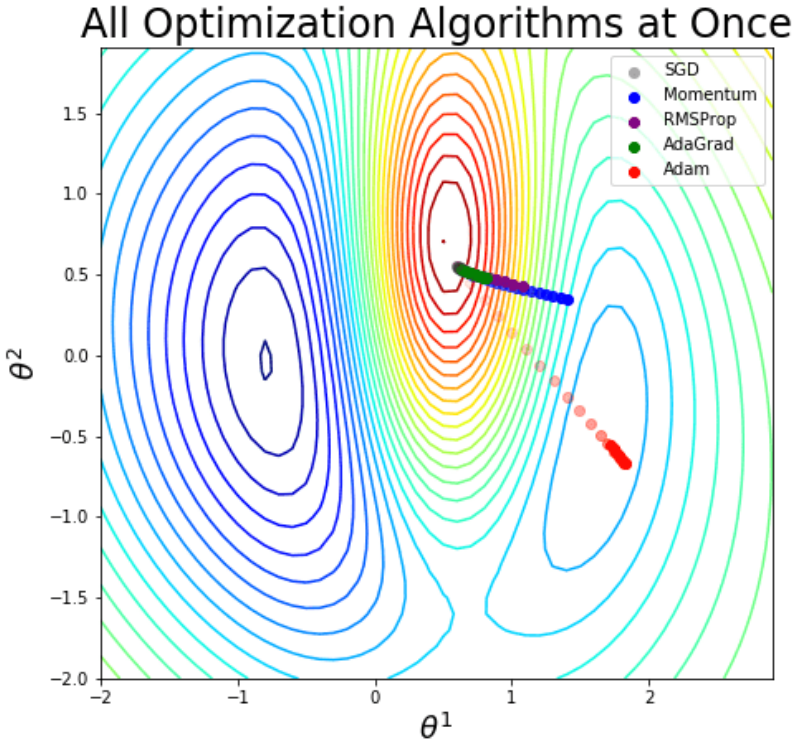
- The iterative procedure will perform several passes on $\mathcal{D}$.
- Parameter updates are made only processing a subset of $\mathcal{D}$ (batch).

A broad variety of approaches based on Gradient Descent have been proposed.

Training: optimizers using the gradient

Numerous optimizers exist: adam, adagrad...





cf. 2IA module **Mathematics for Machine Learning and Optimization.**

Training: terminology

Terminology used to parameter training:

- **Epoch**: duration for the entire dataset to be used once. Several epochs will be used (iterative process).
- **Batch**: subset of the full dataset. Number of batches : number of partitions of the datasets considered. Batch size: number of training examples in a batch (partition).
- **Iterations**: number of batches required for one epoch, i.e. number of batches.

Optimizers also require:

- **learning rate**: amount of modification performed using the gradient.

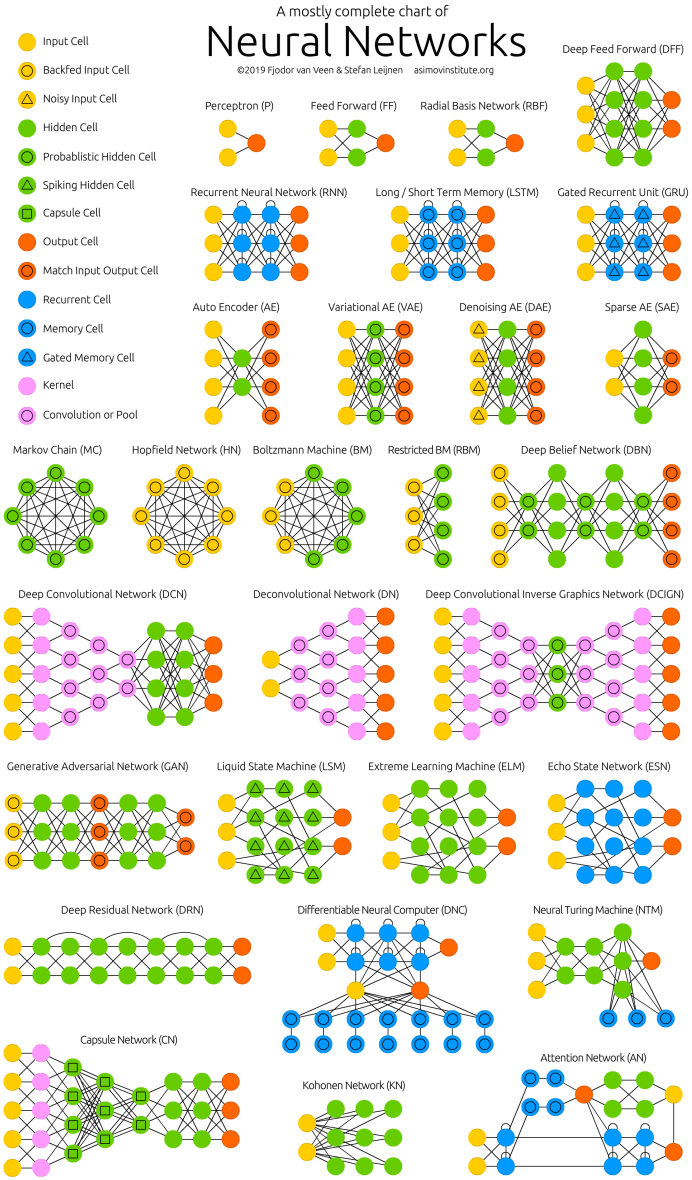
Important ANN Architectures

A diversity of architectures

Various types of ANNs have been proposed to tackle different problems.

Among the most commonly used in practice today we find:

- **Multilayer perceptron** (MLP).
- **Convolutional Neural Networks** (CNNs): very propular for processing images, data with spatial properties, texts.
- **Recurrent Neural Networks** (RNNs): often used for temporal analysis, and for Natural Language Processing.
- **Transformers**: very popular in Natural Langage Processing tasks but also used in other domains.



http://www.asimovinstitute.org/neural-network-zoo/

Important ANN Architectures

Convolutional Neural Networks - CNNs

Convolutional Neural Networks - CNNs

In the following content we will introduce Convolutional Neural Networks (CNNs) considering Image processing applications.

Note however that the underlying concepts
can be applied to various types of input data.
Any data with spatial properties.

Convolutional Neural Networks - CNNs

Using MLPs to process raw images has several limitations:

- Images have to be flatten into vector representations
  $\Longrightarrow$ Important spatial information is lost.
- Flattening the input image creates a big vector which leads to the addition of a large number of parameters in the first layers.

Convolutional Neural Networks - CNNs

Using MLPs to process raw images has several limitations:

- Images have to be flatten into vector representations
  $\Longrightarrow$ Important spatial information is lost.
- Flattening the input image creates a big vector which leads to the addition of a large number of parameters in the first layers.

Manual feature extraction:
Manually defined feature extraction techniques have long been applied to input images to construct vector representations that were fed to MLPs.

$$\text{Image } x \xrightarrow{\textit{Feature Extraction}} [\psi_1(x), \psi_2(x), \ldots] \in \mathbb{R}^n$$

$\Longrightarrow$ Requires defining complex domain-specific feature extraction techniques.

Convolutional Neural Networks - CNNs

Convolutional Neural Networks (CNNs) introduced by Yann LeCun (1998) answer those limits; they have revolutionized image processing.

CNNs:

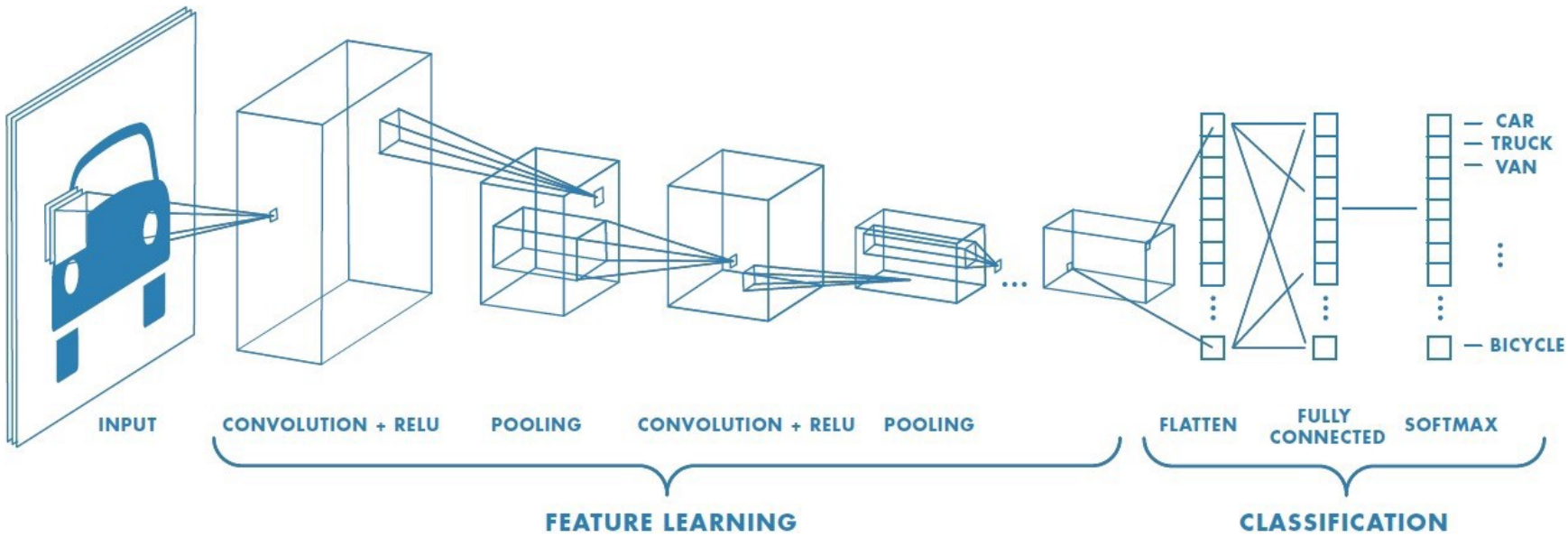
- Can be applied to raw representations of images (Tensors).
  $\Longrightarrow$ do not require manual feature extraction.
- Achieve impressive performance.

CNNs are today widely used for image classification, image segmentation, object recognition, face recognition...

Convolutional Neural Networks - CNNs

A CNN is composed of several types of layers:

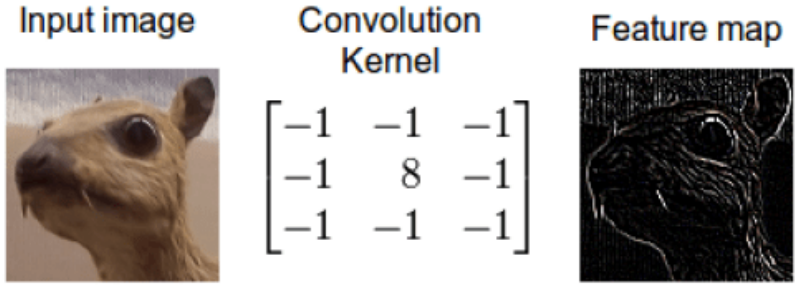
- Convolutional layers
- Pooling layers
- Fully connected layers



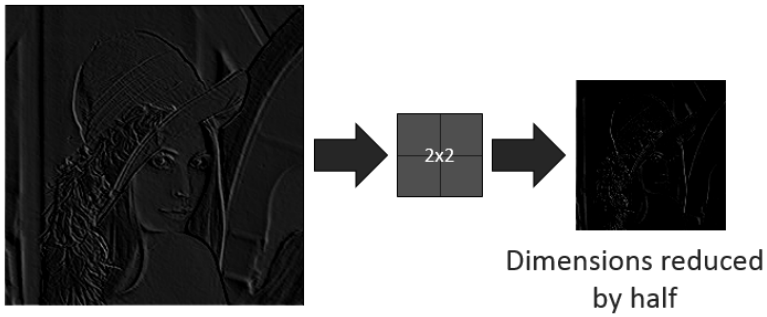
[source]

Convolutional Neural Networks - CNNs

- **Convolutional layer**: Apply transformations to the input data to extract interesting features.



- **Pooling layer**: Reduces the spatial size of the convolved features. Interesting to reduce the complexity and to capture specific types of invariants. Max and average pooling layers are often used.



short overview: `https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/`

Convolutional layer

Convolutional layers extract features from the input data.

- They apply a convolution using a Kernel (filter).
- Convolution preserves the spatial relationship between input values. They learn features using small squares of input data.

Simply speaking, in our context, you can consider that a convolution is a process sequentially applying a filter (kernel) to part of the input.

More rigorous content: `https://en.wikipedia.org/wiki/Convolution`

Convolutional layer

Consider the following 1 x 5 x 5 image and a 3 x 3 Kernel (filter).

Image

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Kernel

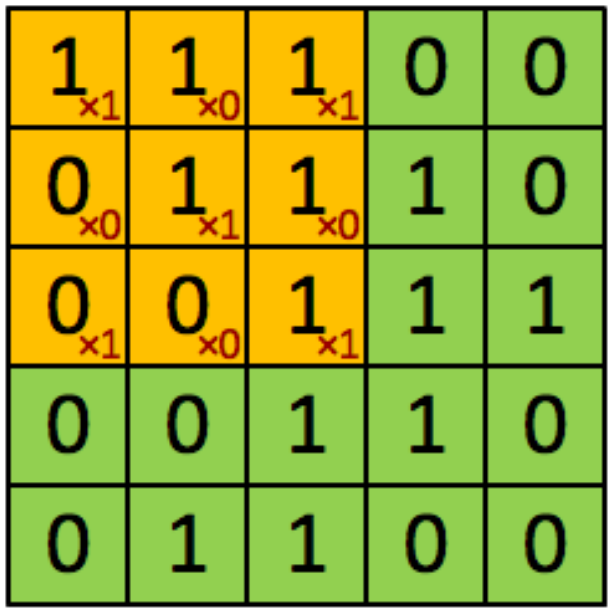
1	0	1
0	1	0
1	0	1

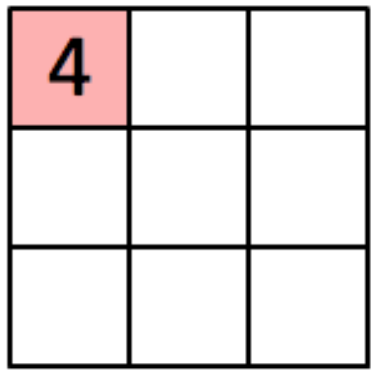
Note that the input could be something else than raw pixel values.

[source]

Convolutional layer

At each step we apply the filter (kernel) to a subset of the image
$\Rightarrow$ Summing the values obtained by element-wise multiplication between pixel values of the processed image subset and corresponding kernel values

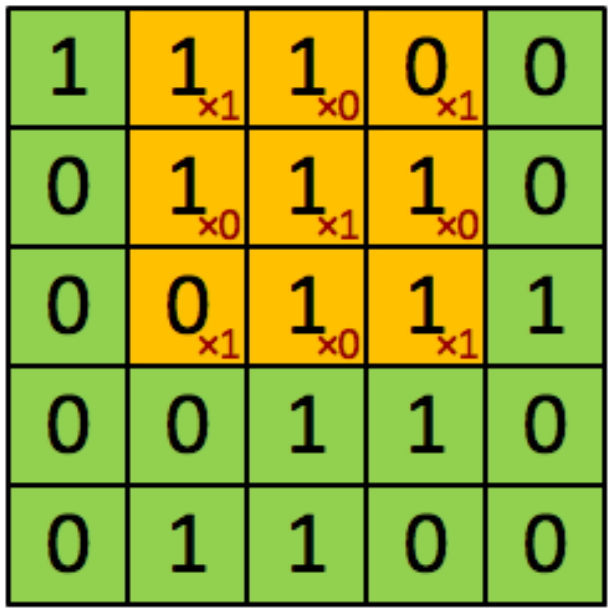


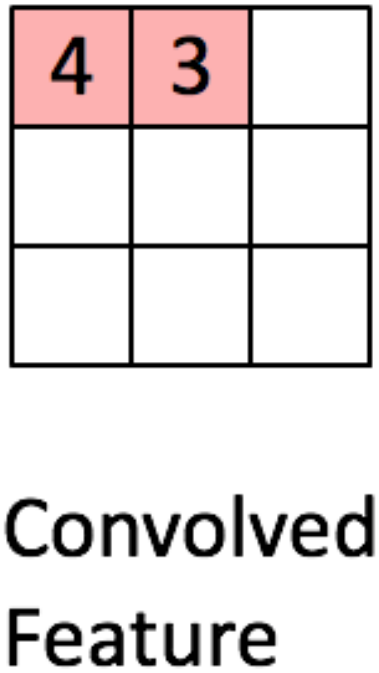


[source]

Convolutional layer

At each step we apply the filter to a subset of the image
$\Rightarrow$ Summing the values obtained by element-wise multiplication between pixel values of the processed image subset and corresponding kernel values

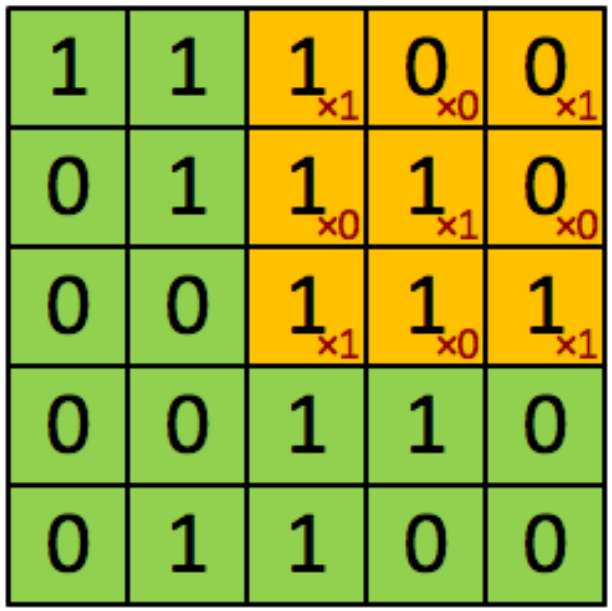




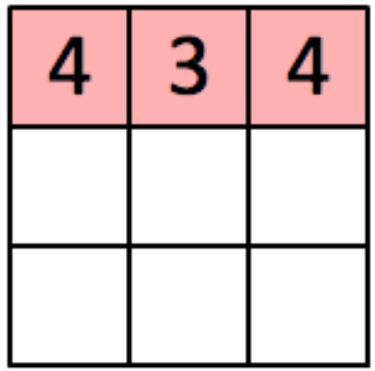
[source]

Convolutional layer

At each step we apply the filter to a subset of the image
$\Rightarrow$ Summing the values obtained by element-wise multiplication between pixel values of the processed image subset and corresponding kernel values



Image

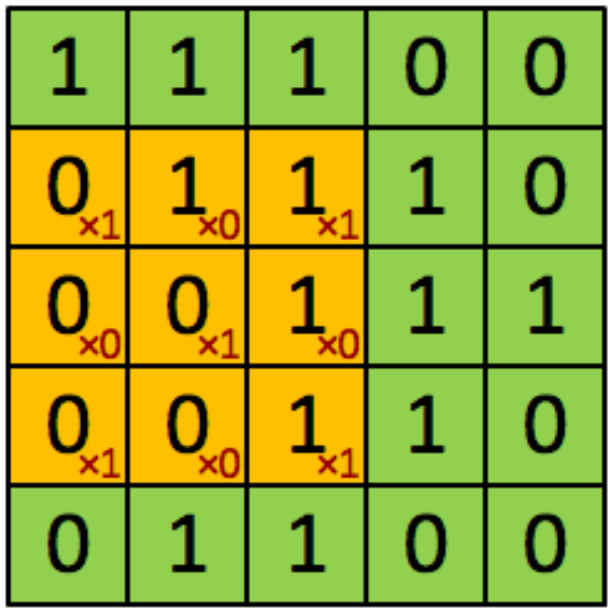


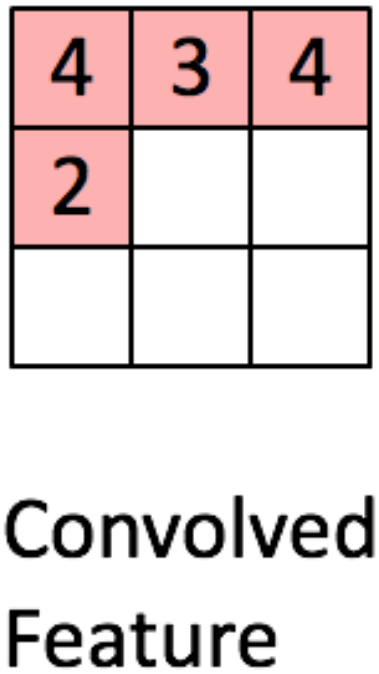
Convolved Feature

[source]

Convolutional layer

At each step we apply the filter to a subset of the image
$\Rightarrow$ Summing the values obtained by element-wise multiplication between pixel values of the processed image subset and corresponding kernel values

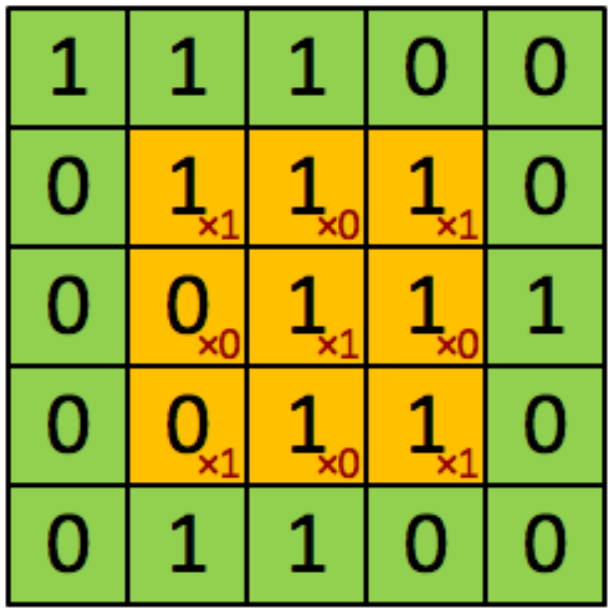




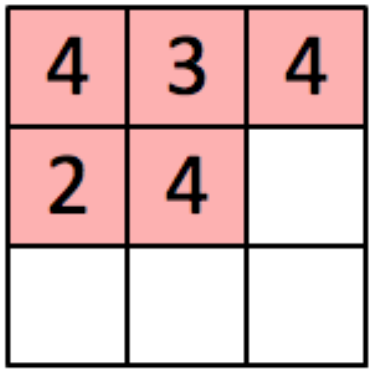
[source]

Convolutional layer

At each step we apply the filter to a subset of the image
$\Rightarrow$ Summing the values obtained by element-wise multiplication between pixel values of the processed image subset and corresponding kernel values



Image

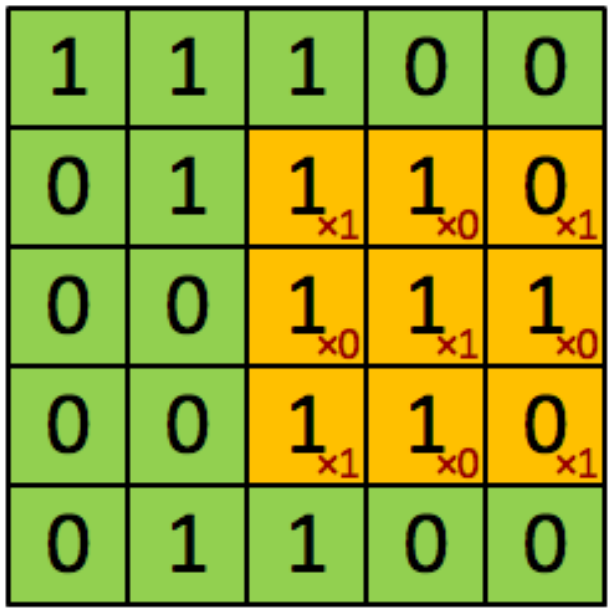


Convolved Feature

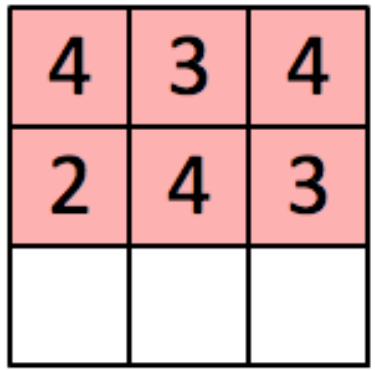
[source]

Convolutional layer

At each step we apply the filter to a subset of the image
$\Rightarrow$ Summing the values obtained by element-wise multiplication between pixel values of the processed image subset and corresponding kernel values



Image

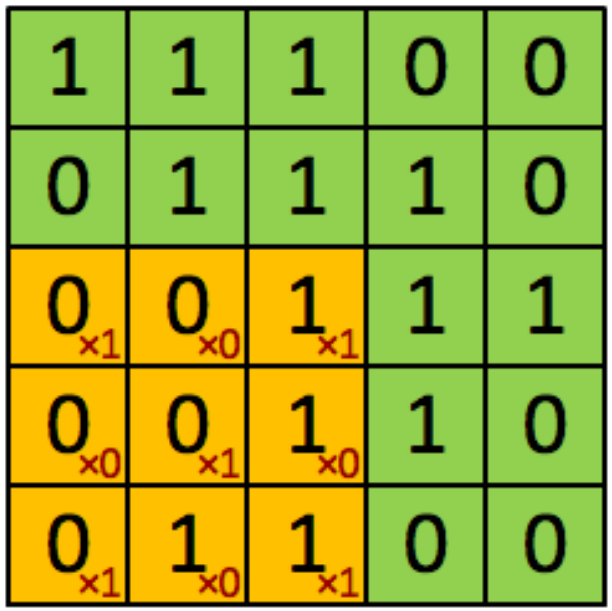


Convolved Feature

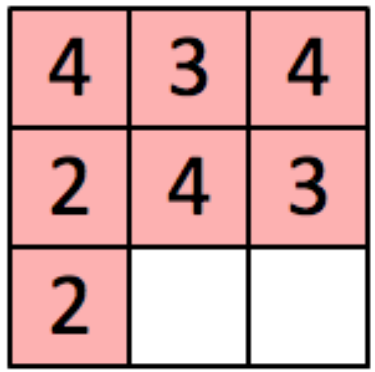
[source]

Convolutional layer

At each step we apply the filter to a subset of the image
$\Rightarrow$ Summing the values obtained by element-wise multiplication between pixel values of the processed image subset and corresponding kernel values



Image



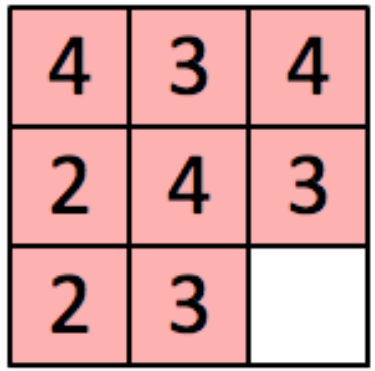
Convolved Feature

[source]

Convolutional layer

At each step we apply the filter to a subset of the image
$\Rightarrow$ Summing the values obtained by element-wise multiplication between pixel values of the processed image subset and corresponding kernel values

Image

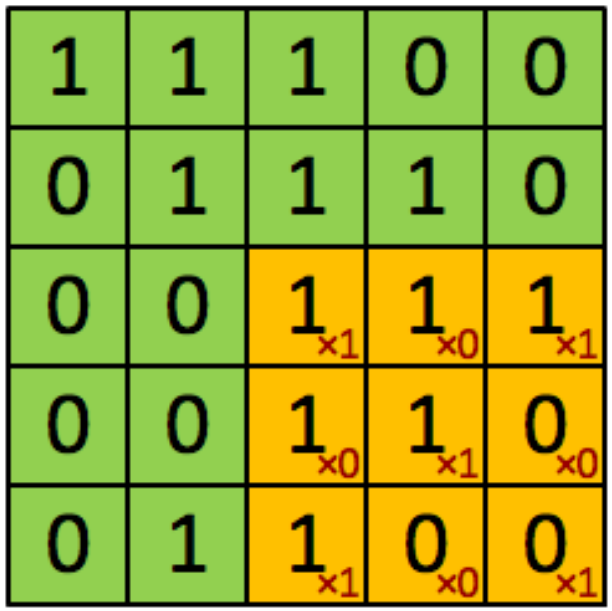


Convolved Feature

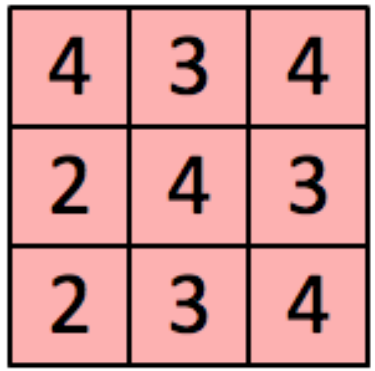
[source]

Convolutional layer

At each step we apply the filter to a subset of the image
$\Rightarrow$ Summing the values obtained by element-wise multiplication between pixel values of the processed image subset and corresponding kernel values



Image

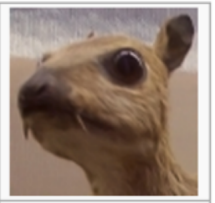


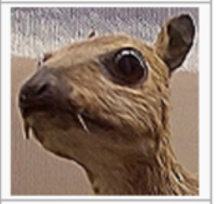
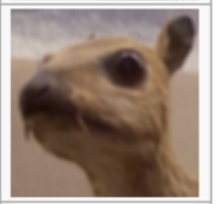
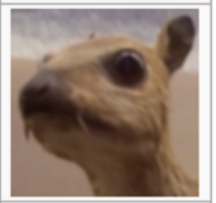
Convolved Feature

[source]

Convolutional layer

The output matrix is called Convolved Feature or Feature Map.
Depending on the filter (Kernel), different feature maps will be obtained.

Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Kernels will be parameters of the CNNs (learned).

[source]

Convolutional layer

Output channel: When we have multiple input channels, a feature map is obtained for each channel using a specific kernel.
$\Rightarrow$ Those feature maps are then added up to obtain a single output channel

A convolutional layer can also produce several output channels.
`Number of kernels = in_channels x out_channels`

Hyper-parameters are:[2]

- Depth (nb `out_channels`): number of sets of kernels used to produce sets of feature maps.
- Stride: the number of pixels by which we slide our kernel over the input matrix.
- Padding: we can pad the input matrix with zeros around the border: it enables to apply the kernel to bordering elements of our input matrix.

[source]

[2] Hyper-parameters are not learned during training, set arbitrarily defining the CNNs.

Convolutional layer + Non linearity

A non-linearity is applied to the output of the convolutional layer.

To do so we only apply a non-linear activation function (e.g. ReLU) element-wise to the values of the output channels.

$$\text{Input} \xrightarrow{Conv.} \text{Feature maps} \xrightarrow{\phi\ (e.g. ReLU)} \text{transformed Feature maps}$$

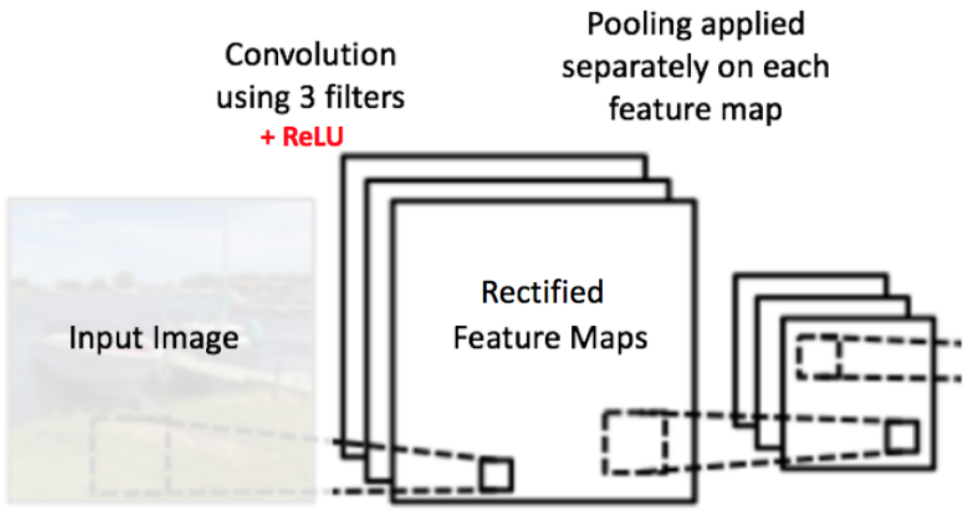
Pooling layer

Pooling layers are used to retain the most important information from each transformed feature maps.

They reduce the dimensionality of feature maps by applying a pooling strategy (Max, Average, etc.) to subsets of the feature maps.

Similarly to convolutional layers they can be parameterized defining:

- a kernel size (size of the window on which will be applied the pooling strategy),
- stride, padding...



[source]

Convolutional Neural Networks - CNNs

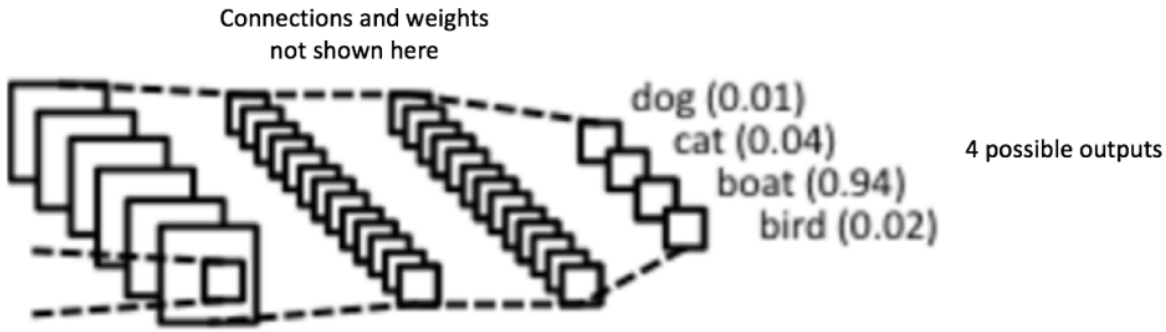
source:

`stats.stackexchange.com/questions/340258/kernel-sizes-for-multiple-convolutional-layer-neural-networks`

Fully connected layer

The Fully Connected layer is a traditional MLP that will produce the final output (e.g. logits) from the output of successive stacks of:

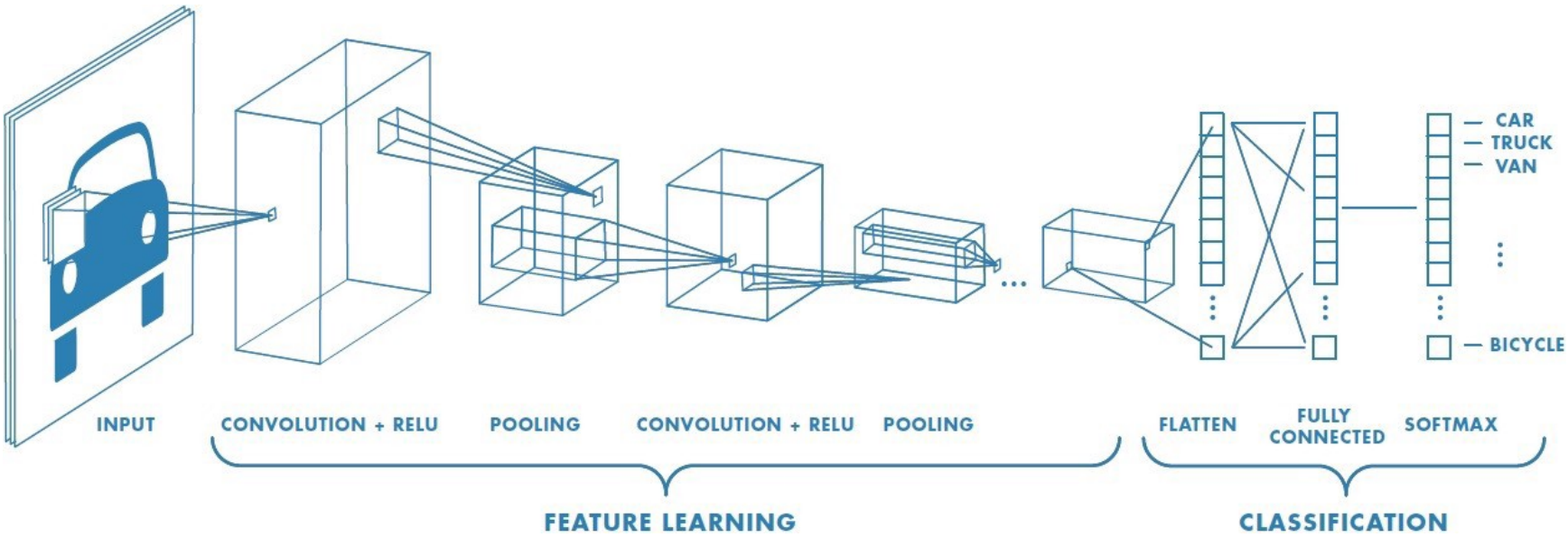
Convolution layer + Non linear Function + Pooling layer



[source]

Convolutional Neural Networks - CNNs

You should now *fully* understand that figure



[source]

Convolutional Neural Networks - CNNs

Refer to the practical session dedicated to CNNs

Important ANN Architectures

Recurrent Neural Networks - RNNs

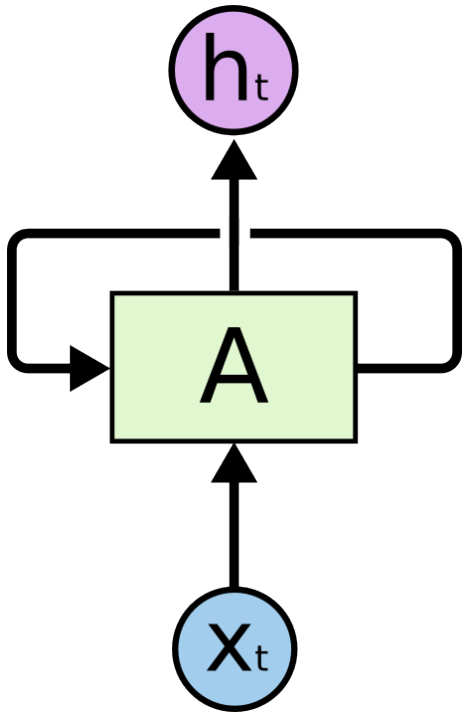
Recurrent Neural Networks - RNNs

Recurrent Neural Networks are often used to tackle problem related to input data with sequential characteristics.

They can be used to process time series, text data, sounds. . . encoding information about temporal dynamic behavior.

Recurrent Neural Networks - RNNs

The following graphical representation is often used to represent an RNN.

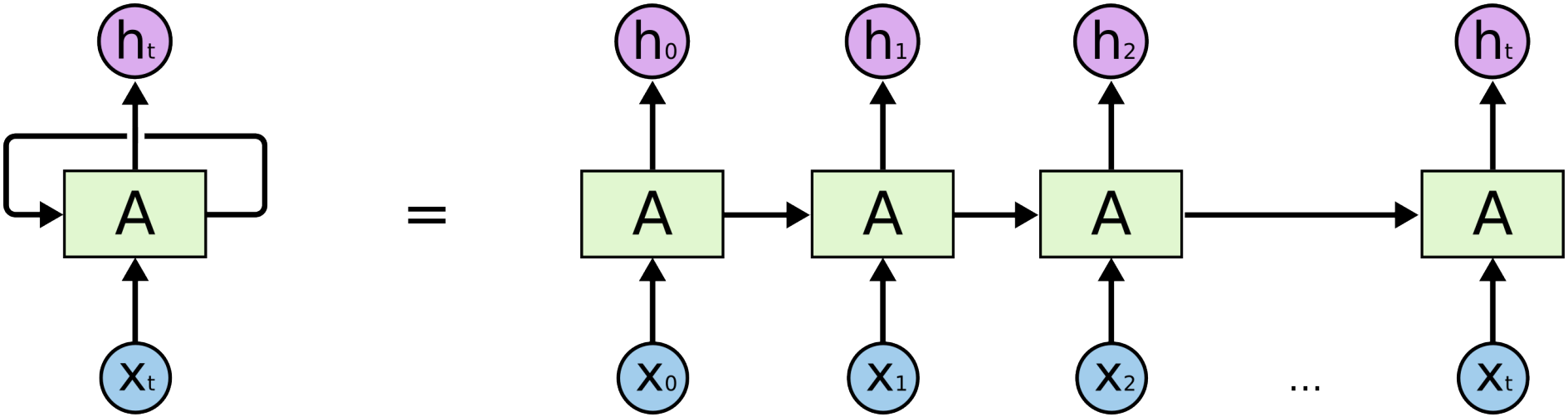


Considering an input x_t at time t the network output a given output h_t and produces an internal state that can be fed into the network at time $t + 1$.

[source]

Recurrent Neural Networks - RNNs

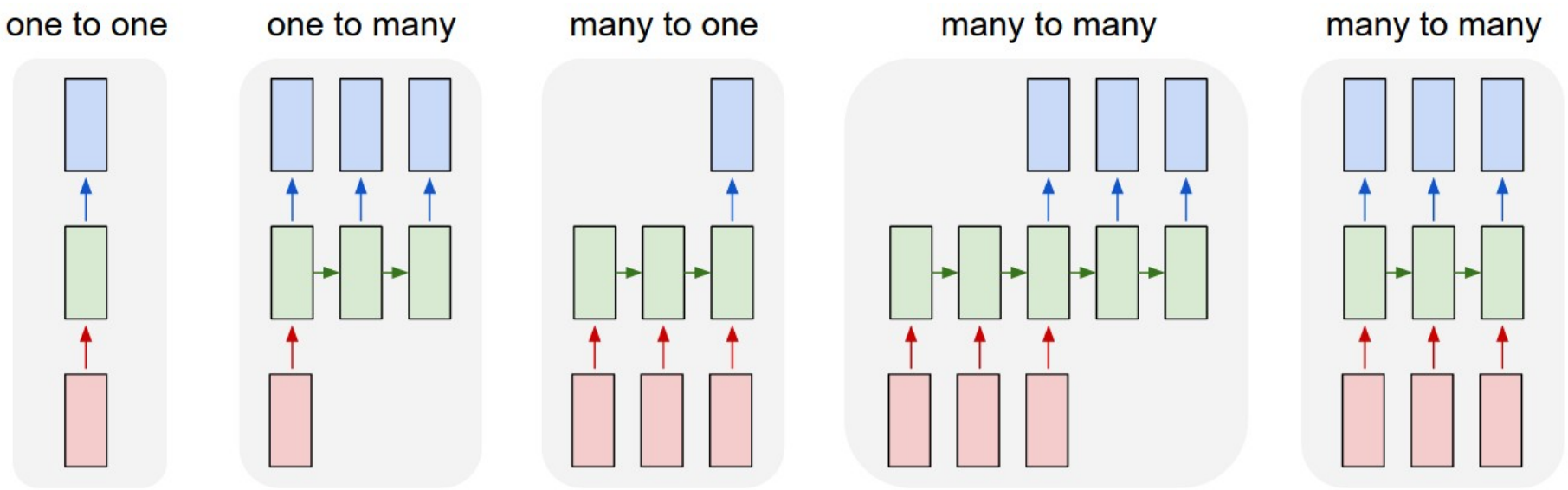
Unrolled representation:



At each timestep the corresponding input is processed, an output is generated, as well as an internal state that will be considered at the next timestep.

[source]

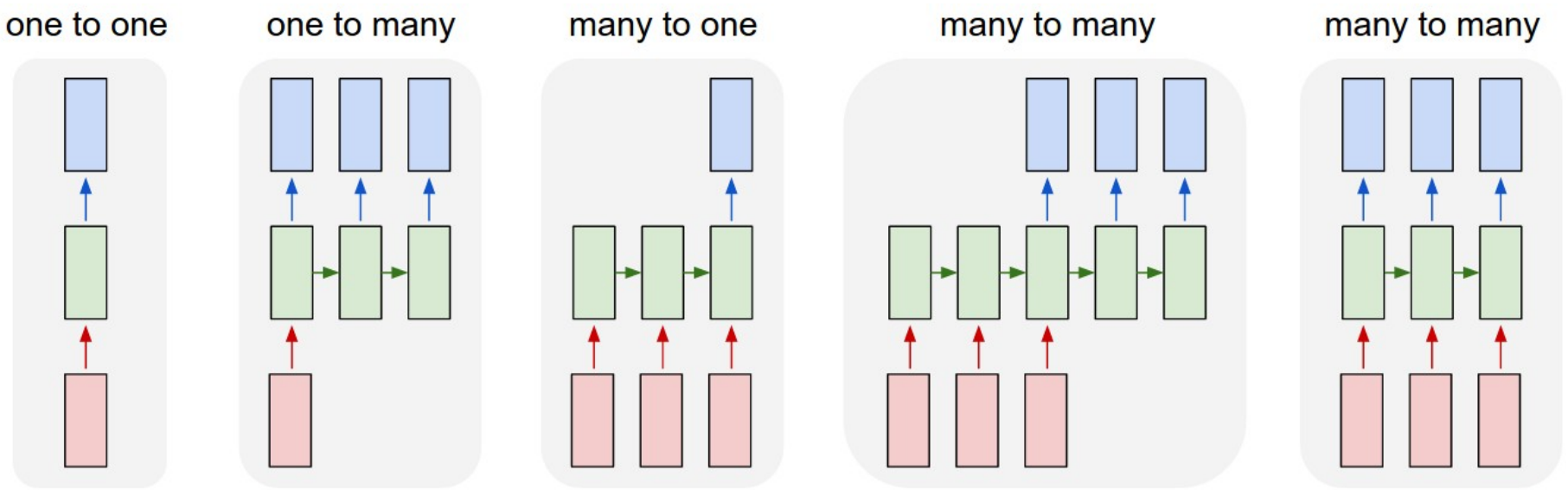
Recurrent Neural Networks - RNNs



[source]

Application Examples?

Recurrent Neural Networks - RNNs



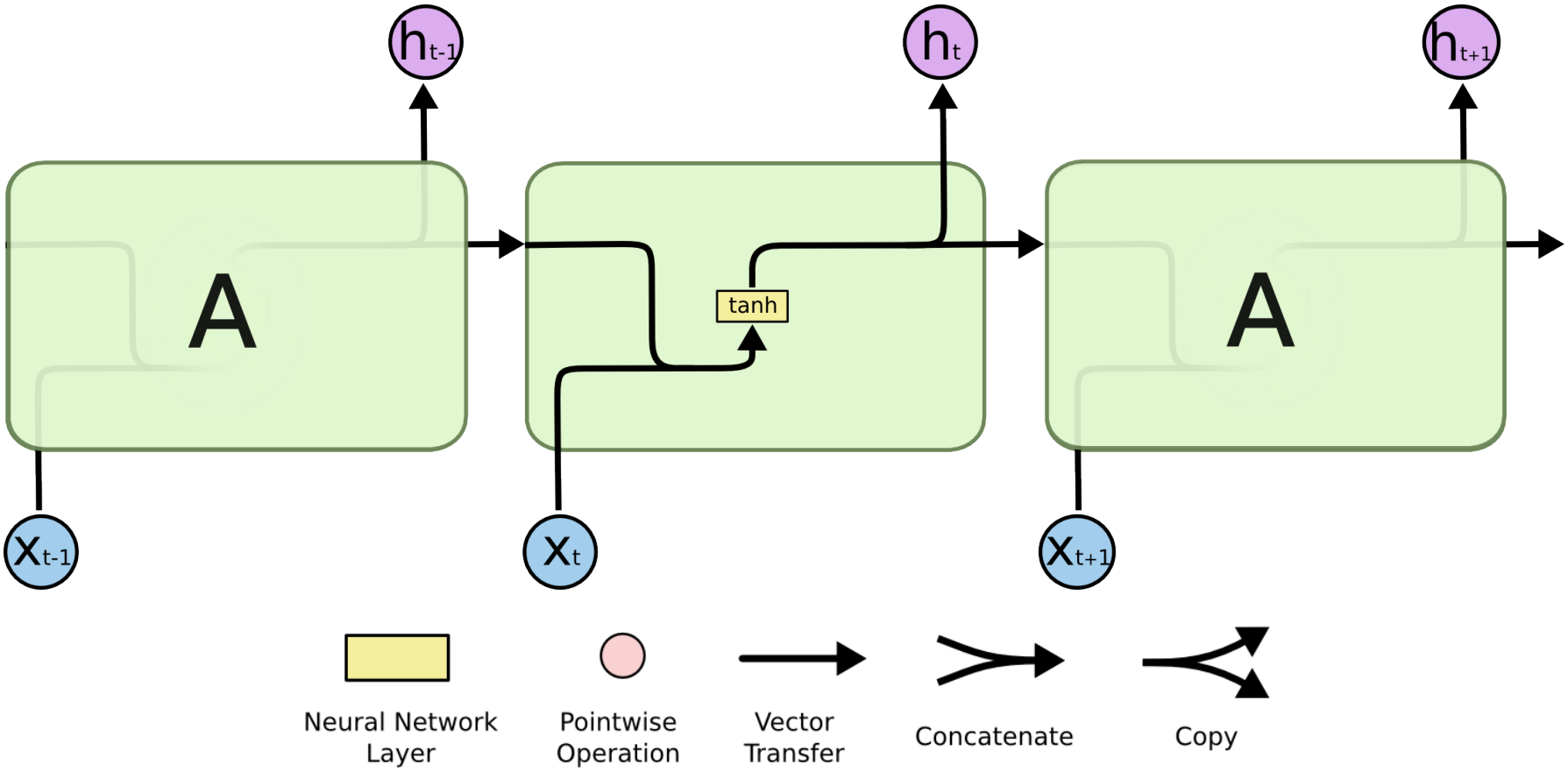
[source]

Application Examples?

- one to many: e.g. caption generation (input: image, output: words).
- many to one: e.g. text classification (input: words, output: logits)
- many to many: e.g. translation (input: words FR, output: words EN)

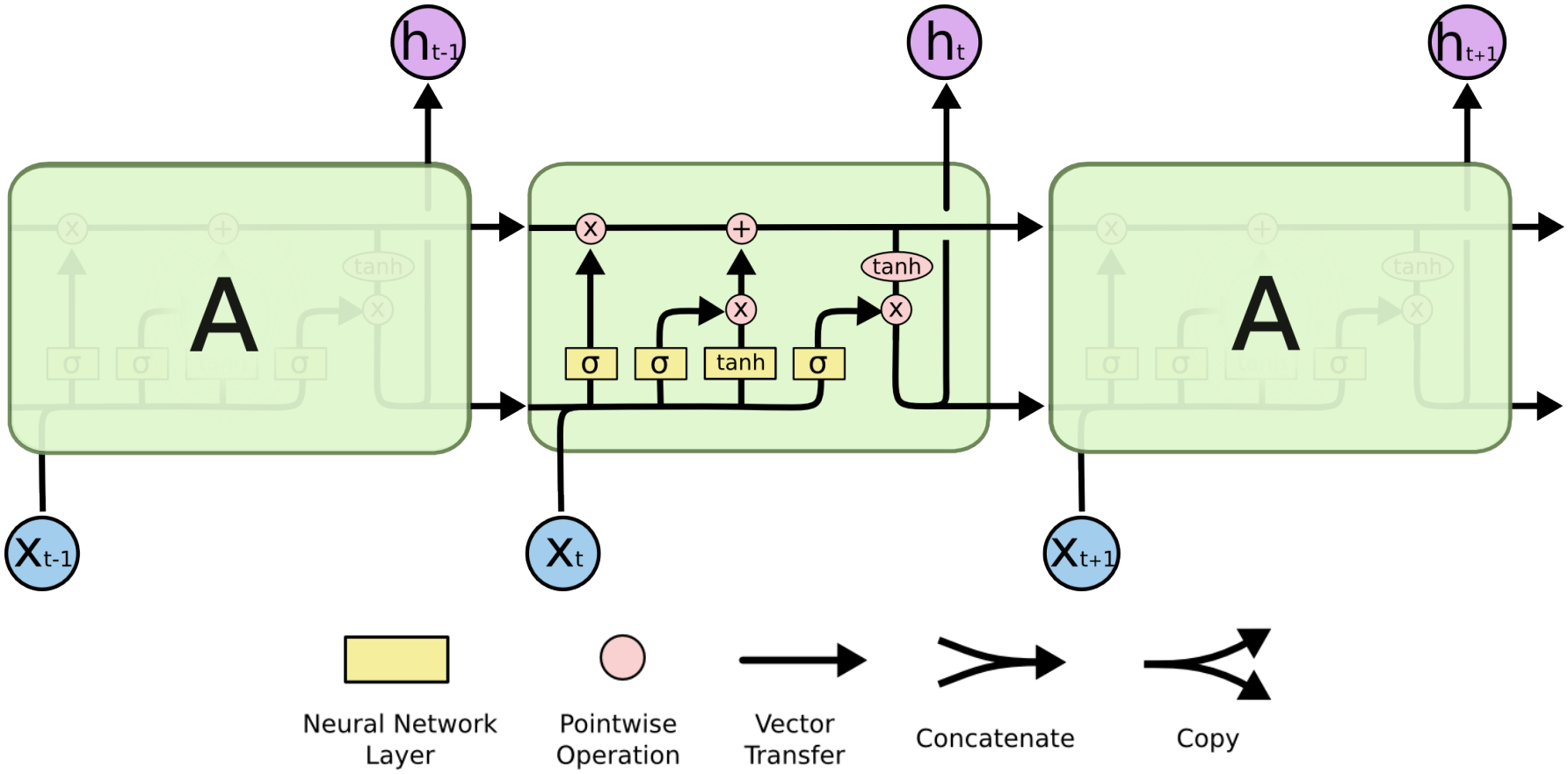
Recurrent Neural Networks - RNNs

Example of a simple RNN:



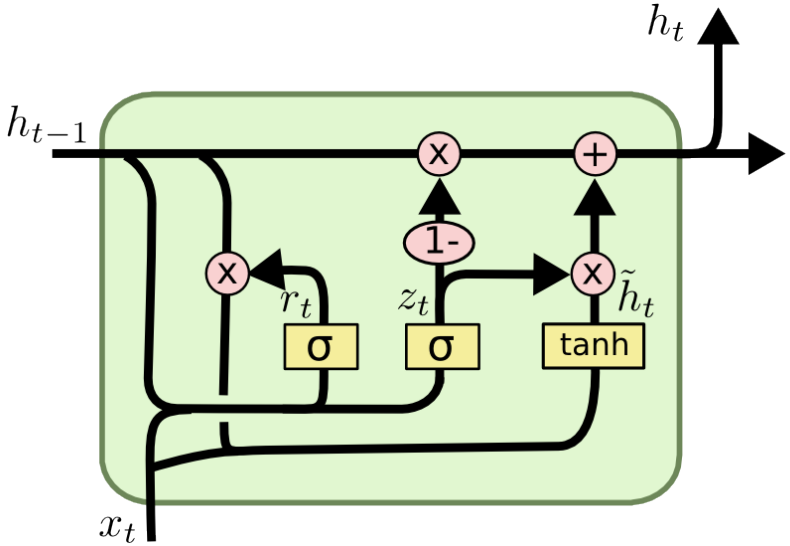
Recurrent Neural Networks - RNNs

LSTM: Long Short Term Memory



Recurrent Neural Networks - RNNs

GRU: Gated Recurrent Unit



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

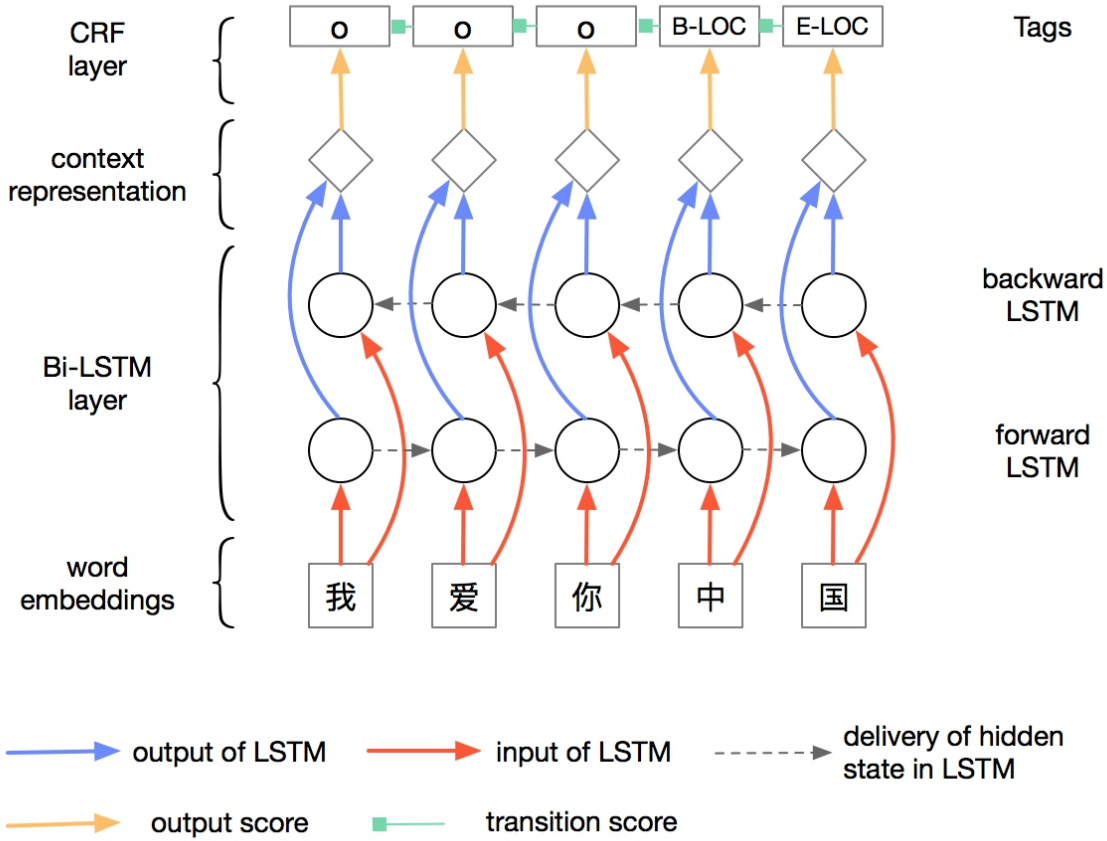
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Recurrent Neural Networks - RNNs

Bi-LSTM: Example for Named Entity Recognition
Used to tag texts, e.g. to find product names



[source]

Word embeddings are vector representations of words.

CRF layer: output class probabilities considering previous word output.

Recurrent Neural Networks - RNNs

RNNs can easily be implemented using PyTorch which provides implementation of the most common reccurent cells: GRU, LSTM...

Two examples:

- Implementing a basic character-level RNN to predict which language a name is from based on the spelling: (access the tutorial).
- Sequence to sequence model for translation (access the tutorial).

Recurrent Neural Networks - RNNs

Sequence to sequence model for translation with PyTorch. [source]

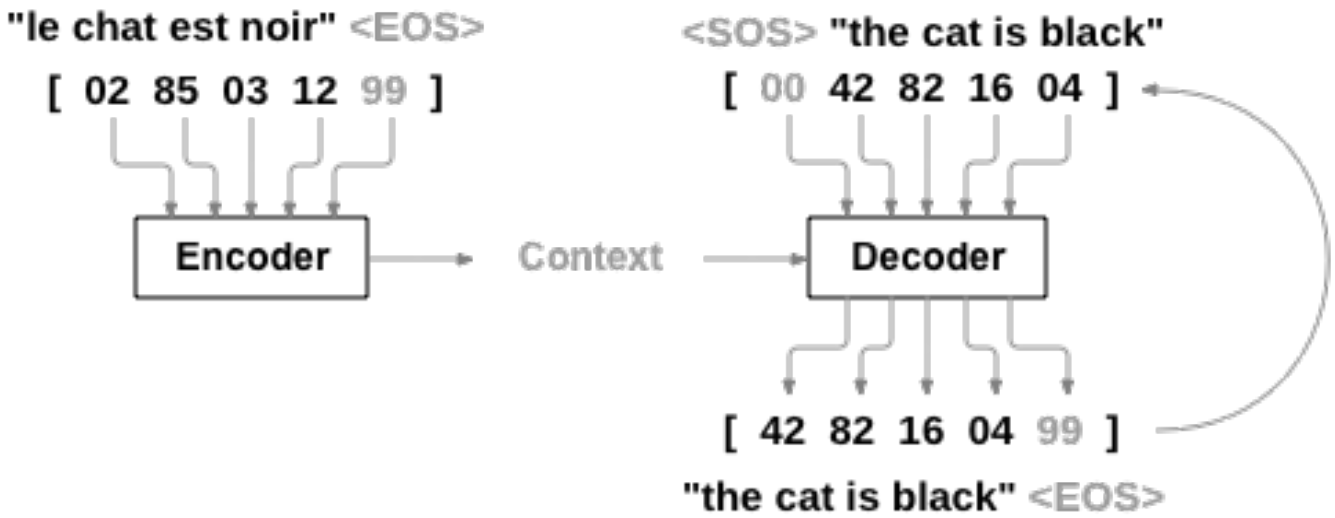
Input data:

```
[KEY: > input, = target]

> il est en train de peindre un tableau .
= he is painting a picture .

> pourquoi ne pas essayer ce vin delicieux ?
= why not try that delicious wine ?

> elle n est pas poete mais romanciere .
= she is not a poet but a novelist .
..
```

Recurrent Neural Networks - RNNs

Sequence to sequence model for translation with PyTorch. [source]

General approach:



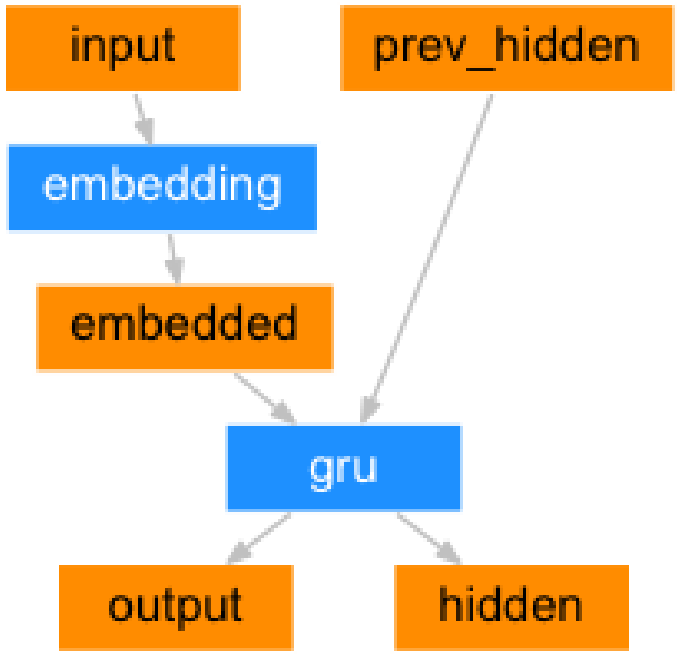
Two big steps:

- The input word sequence is encoded by an encoder into a vector representation. Each word is identified by an id.
- The output sequence of translated words is generated by a decoder.

Recurrent Neural Networks - RNNs

Sequence to sequence model for translation with PyTorch. [source]

Encoder:



Steps:

- Embedding is just a module that will map a word id to $\mathbb{R}^n$.
- The RNN cell (GRU) takes as input the embedding and previous hidden state ($\in \mathbb{R}^h$).
- The RNN cell outputs an output vector and its hidden state.

Recurrent Neural Networks - RNNs

Sequence to sequence model for translation with PyTorch. [source]

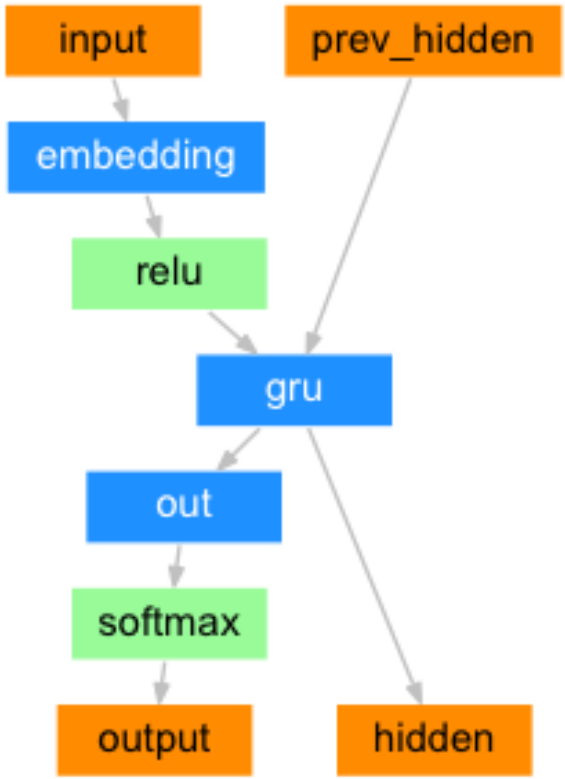
Encoder Implementation:

```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden
```

Recurrent Neural Networks - RNNs

Sequence to sequence model for translation with PyTorch. [source]

Decoder:



Steps:

- The RNN cell (GRU) takes as input the embedding of the previous output word and previous hidden state ($\in \mathbb{R}^h$) (context at t_0).
- The RNN cell outputs the logits and its hidden state.

Recurrent Neural Networks - RNNs

Sequence to sequence model for translation with PyTorch. [source]

Decoder Implementation:

```
class DecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(output_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.softmax(self.out(output[0]))
        return output, hidden
```

Important ANN Architectures

Transformers

Transformers

Transformers are very popular models
largely used in Natural Language Processing.

They heavily rely on the concept of Attention
that we will briefly introduce.

Attention

Attention is a mechanism used to focus on a specific subset of a piece of information processing an input.

[source]

Caption generation steps showing important pixels.

Attention

A wide variety of Attention mechanisms have been and are studied.

We will here focus on self-attention used in Transformers.

Towards Self-Attention

Self-attention will simply be used to generate vector representations considering contextual information.

Consider that we have a sequence of words $w_0, w_1, \ldots, w_s$.

We would like to encode a word of that sequence considering contextual information, e.g depending on the context a word may have different meanings: Mouse (Computer/Animal).

To do so we will attend to the complete sequence but we would like to focus on specific words (e.g. Mouse + Cheese $\Longrightarrow$ Animal, Mouse + laptop $\Longrightarrow$ Computer).

Towards Self-Attention

We want to encode w_i (Mouse) from $w_0, w_1, \ldots, w_s$.

Consider now that:

- we have an embeddings $e_j \in \mathbb{R}^d \ \forall \ w_j \in w_0 \ldots, w_s$.
- we know how important each word of the sequence is to contextually encode w_i. The importance of word w_j is denoted $score(w_i, w_j)$.

We could therefore define w_i contextual embedding by:

$$c_i = \sum_{j=0}^{s} score(w_i, w_j) e_j$$

Towards Self-Attention

$$c_i = \sum_{j=0}^{s} score(w_i, w_j) e_j$$

Considering that:

- $score(w_i, w_j)$ is a function of $e_i \cdot e_j$ (dot scaled attention).
- Initials embedding are stored into a matrix $Q = [e_0, \ldots, e_s] \in \mathbb{R}^{s \times d}$.
- e_i is a row vector, i.e. $e_i \in \mathbb{R}^{1 \times d}$

We could rewrite c_i:

$$c_i = (e_i Q^T) Q$$

Towards Self-Attention

$$c_i = \sum_{j=0}^{s} score(w_i, w_j) e_j$$

Considering that:

- $score(w_i, w_j)$ is a function of $e_i \cdot e_j$ (dot scaled attention).
- Initials embedding are stored into a matrix $Q = [e_0, \ldots, e_s] \in \mathbb{R}^{s \times d}$.
- e_i is a row vector, i.e. $e_i \in \mathbb{R}^{1 \times d}$

We could rewrite c_i:

$$c_i = (e_i Q^T) Q$$

Refining the score computation we will redefined c_i as:

$$c_i = softmax(\frac{e_i Q^T}{\sqrt{d}}) Q$$

Towards Self-Attention

So far we have:

$$c_i = softmax(\frac{e_i Q^T}{\sqrt{d}})Q$$

All the contextual representations $C \in \mathbb{R}^{s \times d}$ are thus:

$$C = softmax(\frac{QQ^T}{\sqrt{d}})Q$$

Towards Self-Attention

So far we have:

$$c_i = softmax(\frac{e_i Q^T}{\sqrt{d}})Q$$

All the contextual representations $C \in \mathbb{R}^{s \times d}$ are thus:

$$C = softmax(\frac{QQ^T}{\sqrt{d}})Q$$

Let's generalize considering that for each word w_i we have a corresponding query $q_i \in \mathbb{R}^d$, key $k_i \in \mathbb{R}^d$ and value $v_i \in \mathbb{R}^d$ to define:

$$C = softmax(\frac{QK^T}{\sqrt{d}})V$$

This is very close to self-attention.

Self-Attention

Considering:

- Queries $Q \in \mathbb{R}^{n \times d_k}$
- Keys $K \in \mathbb{R}^{n \times d_k}$
- Values $V \in \mathbb{R}^{n \times d_v}$

Scaled-dot product Attention is defined by:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Self-Attention

Considering:

- Queries $Q \in \mathbb{R}^{n \times d_k}$
- Keys $K \in \mathbb{R}^{n \times d_k}$
- Values $V \in \mathbb{R}^{n \times d_v}$

Scaled-dot product Attention is defined by:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

And generalized to multi-head:

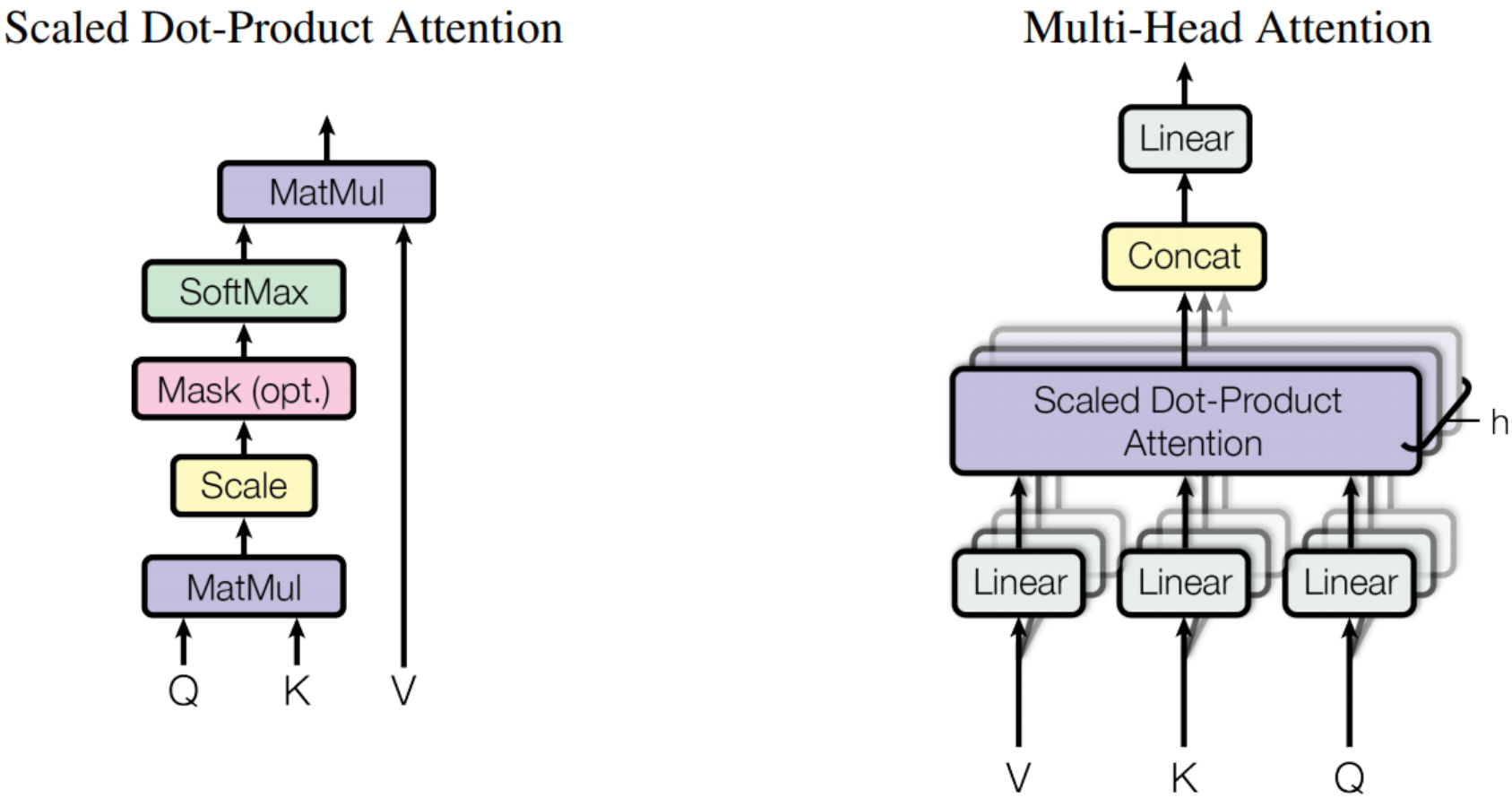
$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O$$

with

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

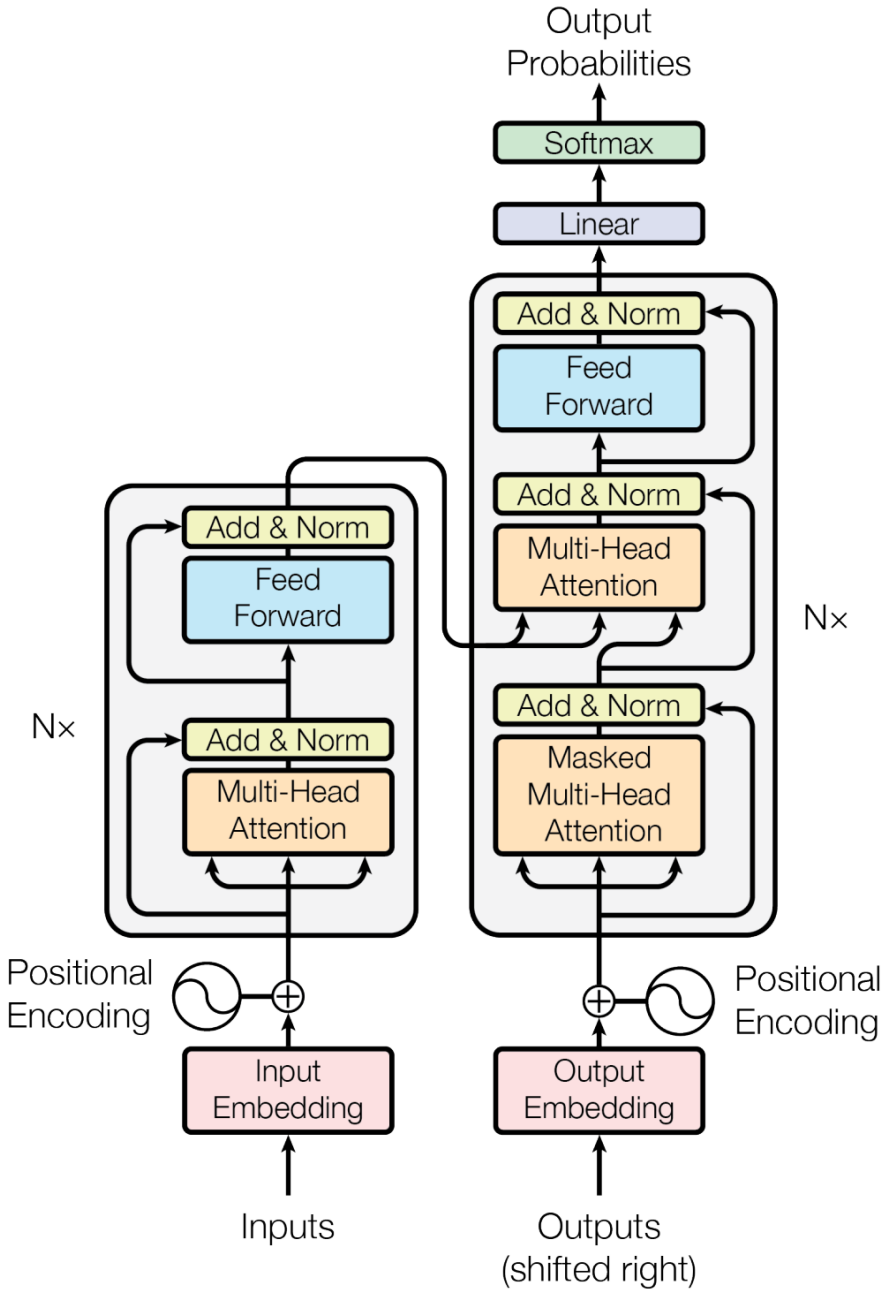
Where $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ ($h \approx 8$ usually, $d_k = d_v = d_{model}/h = 64$).

Attention



[source]

Transformers



[source]

Deep Learning Techniques

Data augmentation

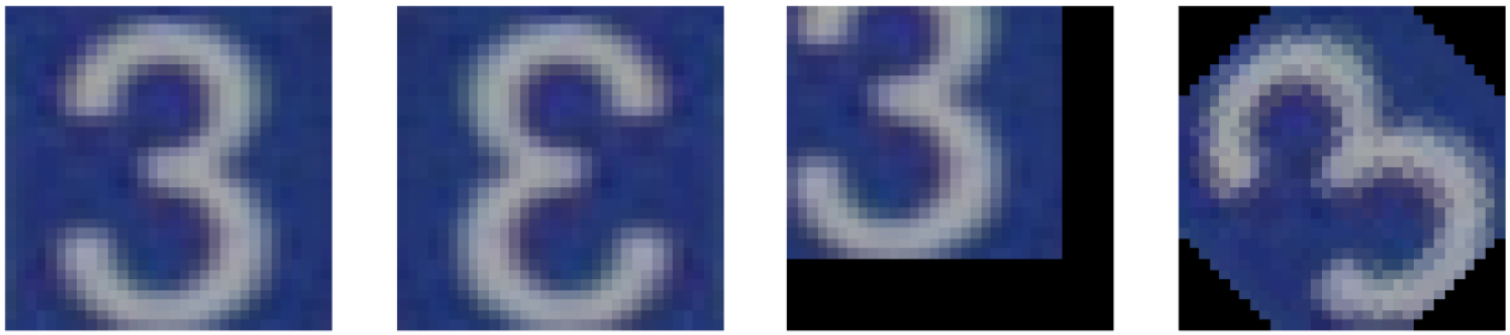
Most often

More data $\Longrightarrow$ Better model.

Data augmentation aims at increasing the data available for training modifying an existing dataset.

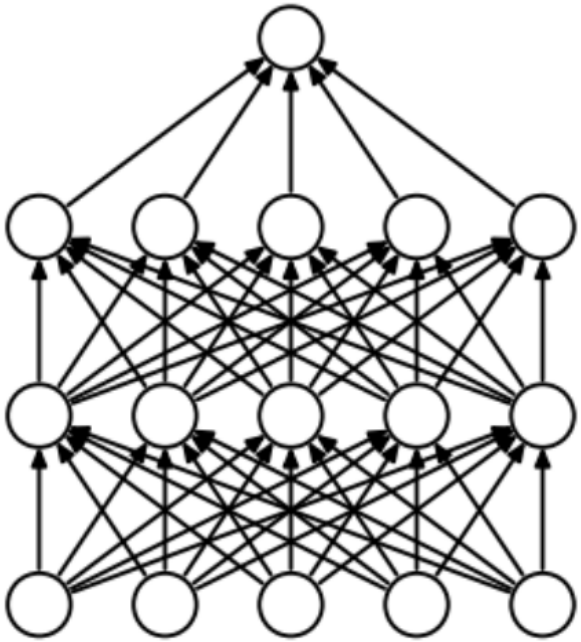
How to augment data? Any idea?

Example: images can be modified using techniques such as cropping, padding, and horizontal/vertical flipping.

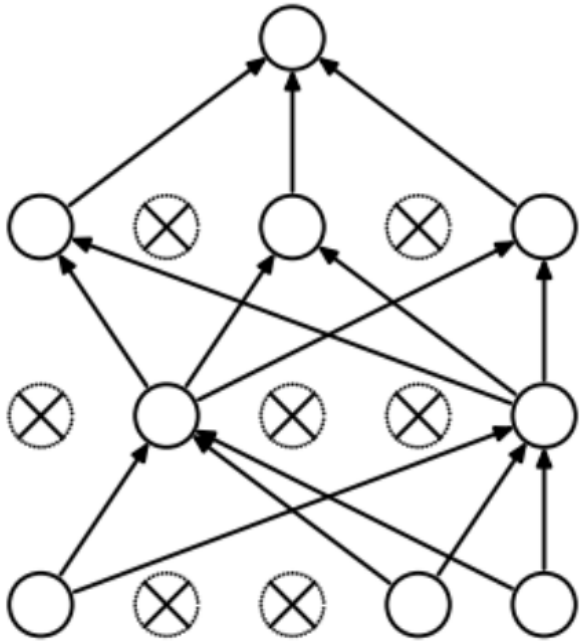


Dropout

- Efficient **regularization technique** (to fight overfitting).
- Randomly remove non-output units from the ANN during training.
- Provides an inexpensive approximation to ensemble learning.



(a) Standard Neural Net

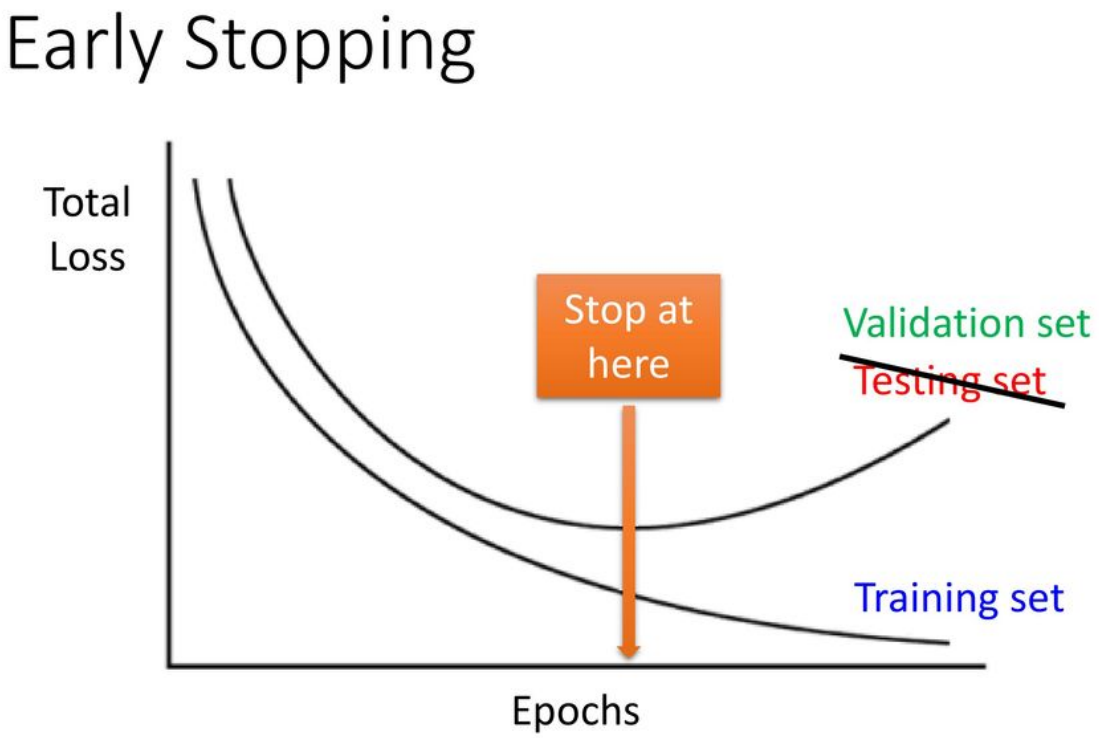


(b) After applying dropout.

https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab

Early-stopping

- Efficient **regularization technique** (to fight overfitting).
- Stop training when error stops decreasing on a validation dataset.
  Note: *using ANNs, training is performed during epochs*.



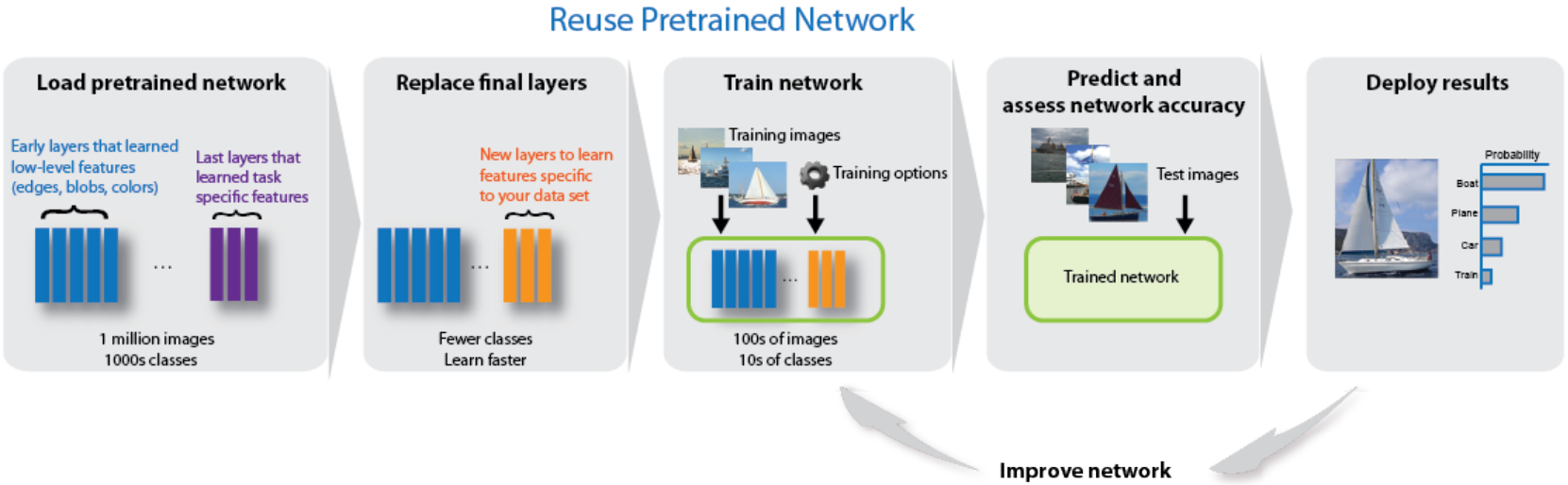
Keras: http://keras.io/getting-started/faq/#how-can-i-interrupt-training-when-the-validation-loss-isnt-decreasing-anymore

Fine-tuning

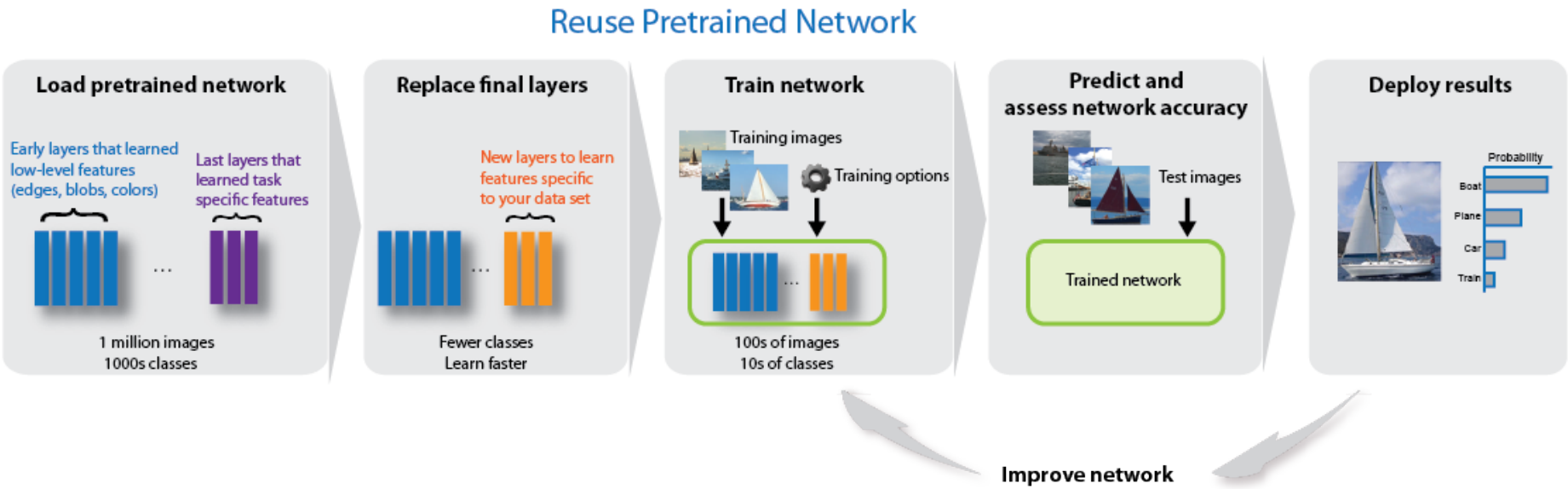
An existing model can be reused for building a new model (to shorten training time and reduce computations, to deal with small datasets).

Example: using a model for object detection in images that have been trained for a different number of classes or a different image quality...
Classical techniques:

- truncate the last layer (e.g. to reduce the number of classes).
- use a reduced learning rate altering the existing model.
- freeze the weight of the first layers (some features may be generic).



Fine-tuning



https://www.mathworks.com/help/deeplearning/deep-learning-with-images.html