# Deep learning

**Timon Deschamps**
timon.deschamps@univ-lyon1.fr

September 2025

# What you’ll learn

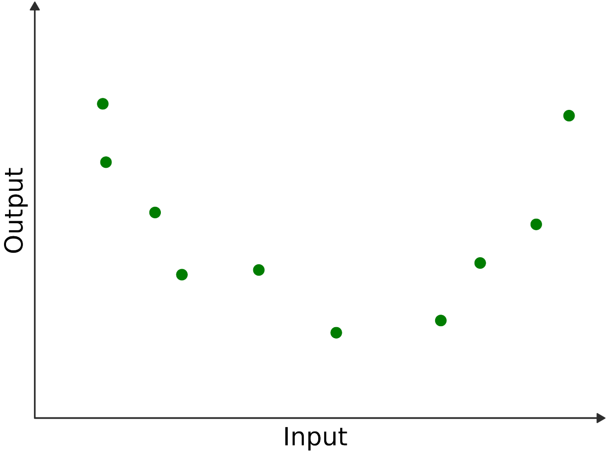
- Deep learning principles
- Perceptron, multilayer perceptron
- Convolutional neural networks
- Deep learning in practice
- Limitations of deep learning

# What is learning? What is our goal?

Formally, we want to learn a function $f(\cdot)$ that maps inputs to desired outputs.

**Goals**

- memorization
- generalization
- explainability, fairness, robustness, efficiency...

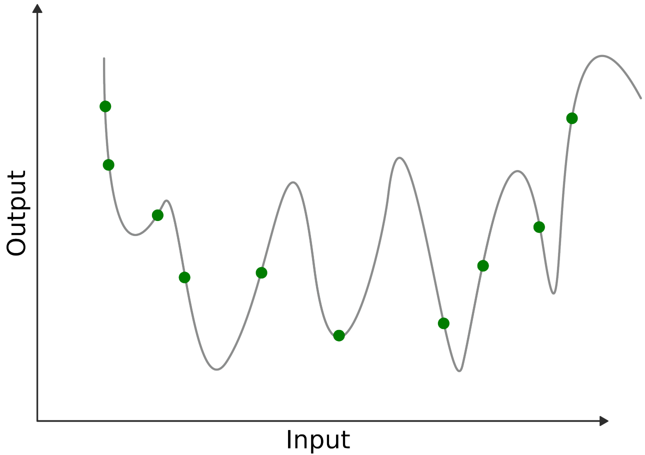


# What is learning? What is our goal?

Formally, we want to learn a function $f(\cdot)$ that maps inputs to desired outputs.

**Goals**

- memorization
- generalization
- explainability, fairness, robustness, efficiency...

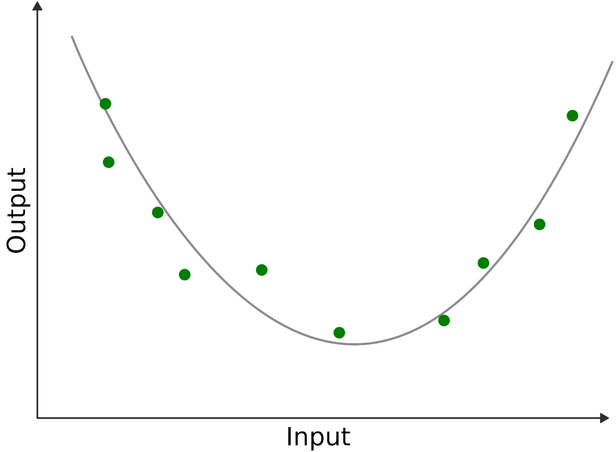


# What is learning? What is our goal?

Formally, we want to learn a function $f(\cdot)$ that maps inputs to desired outputs.

**Goals**

- memorization
- generalization
- explainability, fairness, robustness, efficiency...

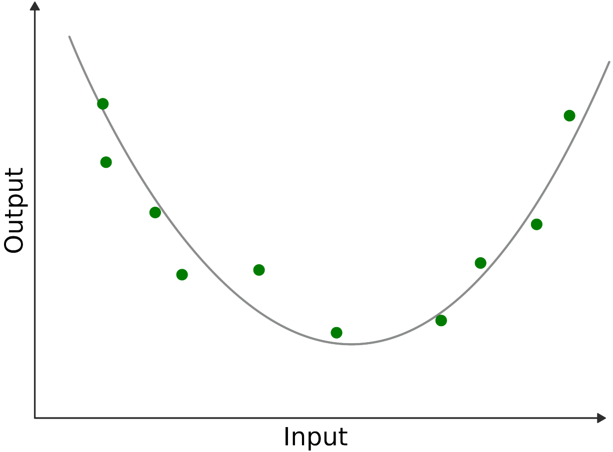


# What is learning? What is our goal?

Formally, we want to learn a function $f(\cdot)$ that maps inputs to desired outputs.

**Goals**

- memorization
- generalization
- explainability, fairness, robustness, efficiency...



# Types of learning

- supervised: $y = f(x)$, with $x \in \mathcal{X}$ and $y \in \mathcal{Y}$
  - regression: $\mathcal{Y}$ is continuous, e.g., $\mathbb{R}^n$
  - classification: $\mathcal{Y}$ is discrete, e.g., $\mathcal{Y} = \{\text{dog}, \text{cat}\}$
- unsupervised: $f(x)$, with $x \in \mathcal{X}$
  - clustering
  - dimensionality reduction
- reinforcement...

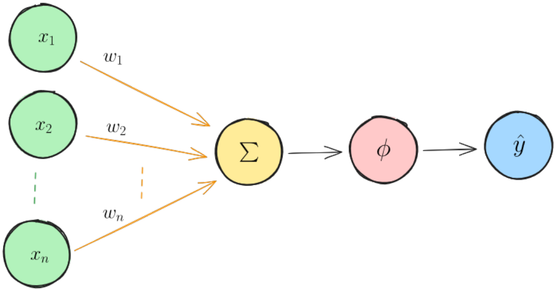
# Types of learning

- supervised: $y = f(x)$, with $x \in \mathcal{X}$ and $y \in \mathcal{Y}$
  - regression: $\mathcal{Y}$ is continuous, e.g., $\mathbb{R}^n$
  - classification: $\mathcal{Y}$ is discrete, e.g., $\mathcal{Y} = \{\text{dog}, \text{cat}\}$
- unsupervised: $f(x)$, with $x \in \mathcal{X}$
  - clustering
  - dimensionality reduction
- reinforcement...

# Types of learning

- supervised: $y = f(x)$, with $x \in \mathcal{X}$ and $y \in \mathcal{Y}$
  - regression: $\mathcal{Y}$ is continuous, e.g., $\mathbb{R}^n$
  - classification: $\mathcal{Y}$ is discrete, e.g., $\mathcal{Y} = \{\text{dog}, \text{cat}\}$
- unsupervised: $f(x)$, with $x \in \mathcal{X}$
  - clustering
  - dimensionality reduction
- reinforcement...

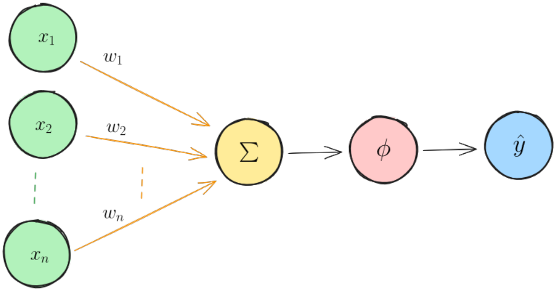
# The artificial neuron [McCulloch and Pitts, 1943]




- inputs/features $x_i$
- weights $w_i$
- sum of the products $\sum$
- activation function $\phi$
- output!

$$\hat{y} = \phi(\sum_{i=1}^{n} x_i w_i)$$

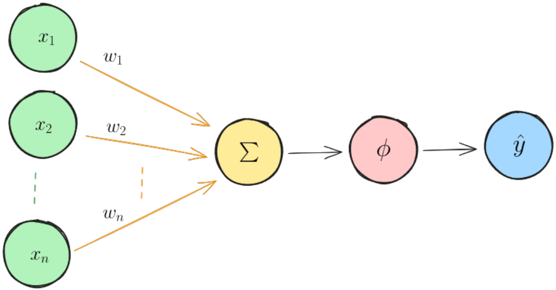
# The artificial neuron [McCulloch and Pitts, 1943]




- inputs/features $x_i$
- weights $w_i$
- sum of the products $\sum$
- activation function $\phi$
- output!

$$\hat{y} = \phi\left(\sum_{i=1}^{n} x_i w_i\right)$$

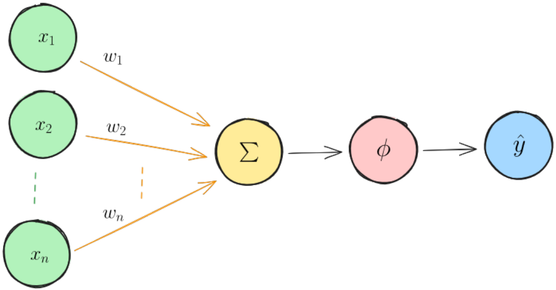
# The artificial neuron [McCulloch and Pitts, 1943]




- inputs/features $x_i$
- weights $w_i$
- sum of the products $\sum$
- activation function $\phi$
- output!

$$\hat{y} = \phi\left(\sum_{i=1}^{n} x_i w_i\right)$$

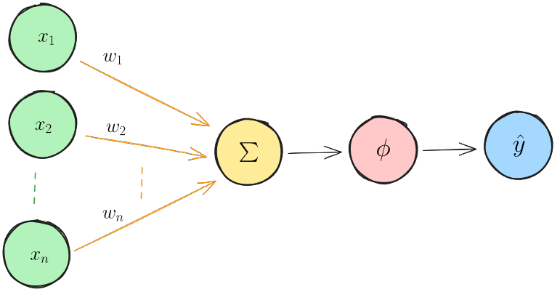
# The artificial neuron [McCulloch and Pitts, 1943]




- inputs/features $x_i$
- weights $w_i$
- sum of the products $\sum$
- activation function $\phi$
- output!

$$\hat{y} = \phi(\sum_{i=1}^{n} x_i w_i)$$

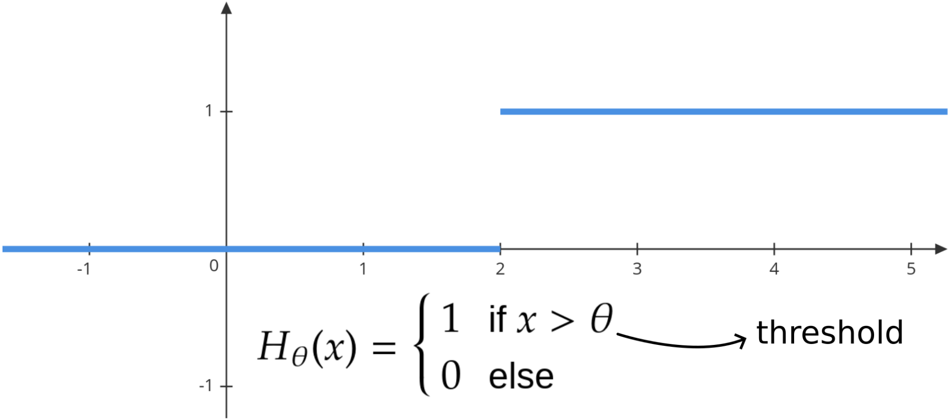
# The artificial neuron [McCulloch and Pitts, 1943]




- inputs/features $x_i$
- weights $w_i$
- sum of the products $\sum$
- activation function $\phi$
- output!

$$\hat{y} = \phi\left(\sum_{i=1}^{n} x_i w_i\right)$$

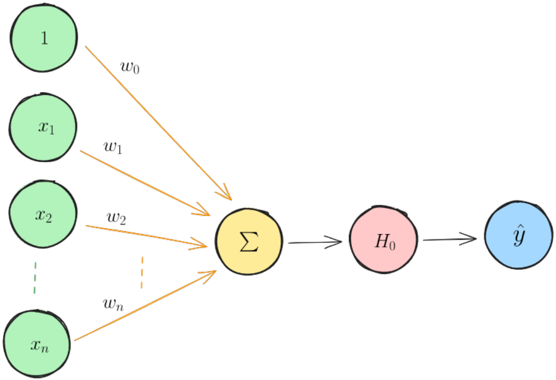
# The artificial neuron [McCulloch and Pitts, 1943]




$$H_\theta(x) = \begin{cases} 1 & \text{if } x > \theta \\ 0 & \text{else} \end{cases}$$

→ threshold

# The perceptron algorithm [Rosenblatt, 1957]




Using $x_0 = 1$ and $w_0 = -\theta$:

$$\hat{y} = H_\theta(\sum_{i=1}^{n} x_i w_i)$$
$$= H_0(\sum_{i=0}^{n} x_i w_i)$$
$$= H_0(\mathbf{x}^\top \mathbf{w})$$

with $\mathbf{x} = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix}$ and $\mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_n \end{bmatrix}$

# The perceptron: learning

Instead of using hand-set values for weights, Rosenblatt proposes to **learn** them.

**Learning rule:** $\Delta w_i = \eta x_i (y - \hat{y})$

$\rightarrow$ Intuitively, if the prediction is larger than the target, we need to reduce the weights, and vice versa.

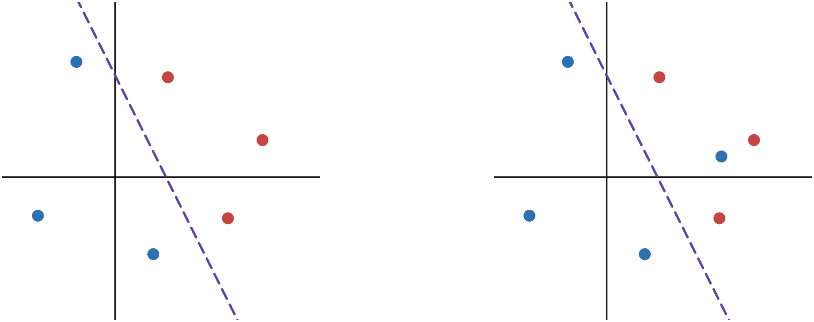
Let's learn the **OR** function by iterating on four learning examples:

$$\mathbf{x}^1 = \begin{bmatrix} \mathbf{1} \\ 0 \\ 0 \end{bmatrix} \rightarrow 0, \quad \mathbf{x}^2 = \begin{bmatrix} \mathbf{1} \\ 1 \\ 0 \end{bmatrix} \rightarrow 1, \quad \mathbf{x}^3 = \begin{bmatrix} \mathbf{1} \\ 0 \\ 1 \end{bmatrix} \rightarrow 1, \quad \mathbf{x}^4 = \begin{bmatrix} \mathbf{1} \\ 1 \\ 1 \end{bmatrix} \rightarrow 1$$

# The perceptron: properties

## Properties

1. linear classifier, i.e., separates space with an **hyperplan**
2. weight vector is orthogonal to the hyperplan, bias controls the y-intercept
3. converges for infinitesimally small $\eta$ if the training data is linearly separable



# From perceptron to SGD

**Problems with the perceptron:**

- can only perform binary classification
- does not converge when data is not linearly separable (or noisy)
- updates in an abrupt manner and does not use well classified samples

**Stochastic gradient descent (SGD):**

**Goal:** update weights to minimize the cost function $\mathcal{J}$

$$\Delta \mathbf{w} = -\eta \nabla \mathcal{J}(\mathbf{w})$$

- updates in a smoother way than perceptron (uses all samples)
- converges even for non linearly separable data (for appropriately chosen $\eta$)
- needs a differentiable cost function!

# From perceptron to SGD

## Problems with the perceptron:

- can only perform binary classification
- does not converge when data is not linearly separable (or noisy)
- updates in an abrupt manner and does not use well classified samples

## Stochastic gradient descent (SGD):

**Goal:** update weights to minimize the cost function $\mathcal{J}$

$$\Delta \mathbf{w} = -\eta \nabla \mathcal{J}(\mathbf{w})$$

- updates in a smoother way than perceptron (uses all samples)
- converges even for non linearly separable data (for appropriately chosen $\eta$)
- needs a differentiable cost function!

# Gradient descents

**Algorithm 1:** Gradient descent

**Data:** Training dataset of $N$ examples
**Result:** Optimized weights $\mathbf{w}$
**Initialize** weights randomly;
**while** *not converged* **do**
  Compute true gradient, $\nabla J(\mathbf{w}) = \frac{1}{N}\sum_1^N \mathcal{L}(x_i)$ // Expensive but convergence is theoretically guaranteed
  Update weights, $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla J(\mathbf{w})$;
**end**
**return** $\mathbf{w}$;

- Batch GD: $\mathcal{J}$ is the average of a loss $\mathcal{L}$ over the entire dataset
- Online GD: $\mathcal{J}$ is the loss on a single training example
- Mini-batch GD: $\mathcal{J}$ is the average loss over a subset of the training dataset

Gradient descent algorithms are **stochastic** when the training examples are selected randomly.

# Gradient descents

**Algorithm 2:** Online gradient descent

**Data:** Training dataset of $N$ examples
**Result:** Optimized weights $\mathbf{w}$
**Initialize** weights randomly;
**while** *not converged* **do**
  Compute estimate gradient, $\nabla J(\mathbf{w}) \simeq \mathcal{L}(x_i)$ // Faster, but noisier: one example is not representative of the training data
  Update weights, $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla J(\mathbf{w})$;
**end**
**return** $\mathbf{w}$;

- Batch GD: $\mathcal{J}$ is the average of a loss $\mathcal{L}$ over the entire dataset
- Online GD: $\mathcal{J}$ is the loss on a single training example
- Mini-batch GD: $\mathcal{J}$ is the average loss over a subset of the training dataset

Gradient descent algorithms are **stochastic** when the training examples are selected randomly.

# Gradient descents

**Algorithm 3:** Mini-batch gradient descent

**Data:** Training dataset of $N$ examples
**Result:** Optimized weights $\mathbf{w}$
**Initialize** weights randomly;
**while** *not converged* **do**
    Compute estimate gradient, $\nabla J(\mathbf{w}) \simeq \frac{1}{n} \sum_1^{n<N} \mathcal{L}(x_i)$ // Often best balance in practice
    Update weights, $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla J(\mathbf{w})$;
**end**
**return** $\mathbf{w}$;

- Batch GD: $\mathcal{J}$ is the average of a loss $\mathcal{L}$ over the entire dataset
- Online GD: $\mathcal{J}$ is the loss on a single training example
- Mini-batch GD: $\mathcal{J}$ is the average loss over a subset of the training dataset

Gradient descent algorithms are **stochastic** when the training examples are selected randomly.

# Gradient descents

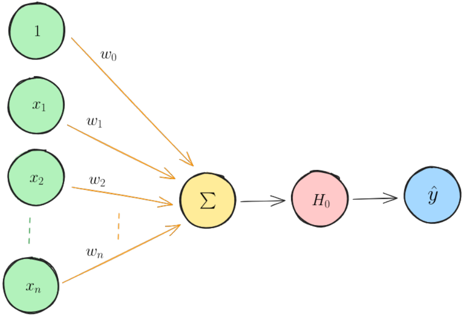
**Algorithm 4:** Mini-batch gradient descent

**Data:** Training dataset of $N$ examples
**Result:** Optimized weights $\mathbf{w}$
**Initialize** weights randomly;
**while** *not converged* **do**
  Compute estimate gradient, $\nabla J(\mathbf{w}) \simeq \frac{1}{n}\sum_{1}^{n<N} \mathcal{L}(x_i)$ // Often best balance in practice
  Update weights, $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla J(\mathbf{w})$;
**end**
**return** $\mathbf{w}$;

- Batch GD: $\mathcal{J}$ is the average of a loss $\mathcal{L}$ over the entire dataset
- Online GD: $\mathcal{J}$ is the loss on a single training example
- Mini-batch GD: $\mathcal{J}$ is the average loss over a subset of the training dataset

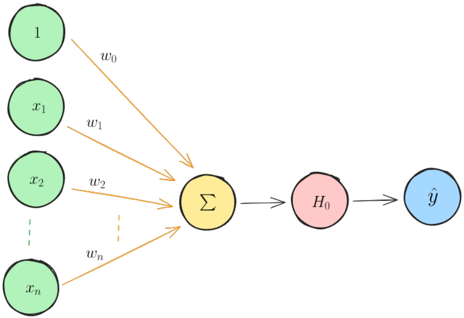
Gradient descent algorithms are **stochastic** when the training examples are selected randomly.

# SGD example: house price prediction



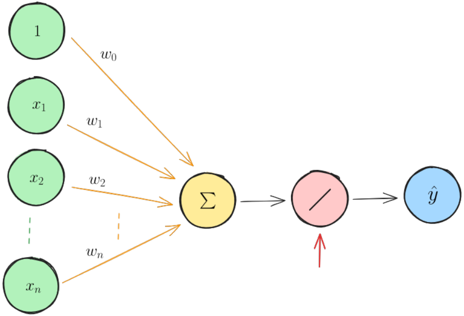

- features $x \in$ {surface area, number of rooms, exposure, parking...}
- labels $y \in \mathbb{R}$
  $\rightarrow$ need to change activation!
  $\phi(x) = x$ is simple, differentiable, and its codomain is $\mathbb{R}$
- What cost function should we use? let's try the average error
  $\frac{1}{n} \sum_x y - \hat{y}(x)$

# SGD example: house price prediction



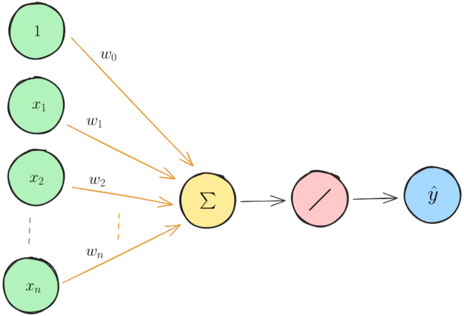

- features $x \in$ {surface area, number of rooms, exposure, parking...}
- labels $y \in \mathbb{R}$
  $\rightarrow$ need to change activation!
  $\phi(x) = x$ is simple, differentiable, and its codomain is $\mathbb{R}$
- What cost function should we use?
  let's try the average error
  $\frac{1}{n} \sum_x y - \hat{y}(x)$

# SGD example: house price prediction




- features $x \in$ {surface area, number of rooms, exposure, parking...}
- labels $y \in \mathbb{R}$
  $\rightarrow$ need to change activation!
  $\phi(x) = x$ is simple, differentiable, and its codomain is $\mathbb{R}$
- What cost function should we use? let's try the average error
  $\frac{1}{n} \sum_x y - \hat{y}(x)$

# SGD example: house price prediction




- features $x \in$ {surface area, number of rooms, exposure, parking...}
- labels $y \in \mathbb{R}$
  $\rightarrow$ need to change activation!
  $\phi(x) = x$ is simple, differentiable, and its codomain is $\mathbb{R}$
- What cost function should we use?
  let's try the average error
  $\frac{1}{n} \sum_x y - \hat{y}(x)$

# SGD: mean error

$\Delta \mathbf{w} = -\eta \nabla \mathcal{J}(\mathbf{w})$, and specifically:

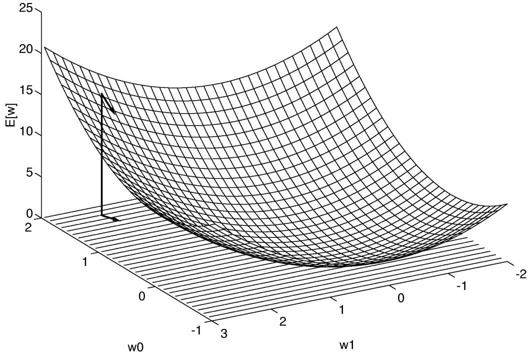
$$\begin{aligned}
\Delta w_i &= -\eta \frac{\partial}{\partial w_i} \mathcal{J}(\mathbf{w}) \\
&= -\eta \frac{\partial}{\partial w_i} \frac{1}{n} \sum_x y - \hat{y}(x) \\
&= -\eta \frac{1}{n} \sum_x \frac{\partial}{\partial w_i} y - \hat{y}(x) \\
&= -\eta \frac{1}{n} \sum_x \frac{\partial}{\partial w_i} y - \sum_i x_i w_i \\
&= \eta \frac{1}{n} \sum_x x_i
\end{aligned}$$

No dependence on the target! The weights will drift without ever converging.
$-10 + 10 = 0 \;\rightarrow$ loss should be non-negative!

# SGD: mean squared error (MSE)

$$\text{MSE} = \frac{1}{2n}\sum_x (y - \hat{y}(x))^2$$

$$\begin{aligned}
\Delta w_i &= -\eta \frac{\partial}{\partial w_i} \mathcal{J}(\mathbf{w}) \\
&= -\eta \frac{\partial}{\partial w_i} \frac{1}{2n} \sum_x (y - \hat{y}(x))^2 \\
&= \frac{-\eta}{2n} \sum_x \frac{\partial}{\partial w_i} (y - \hat{y}(x))^2 \\
&= \frac{-\eta}{2n} \sum_x -2x_i (y - \hat{y}(x)) \\
&= \frac{\eta}{n} \sum_x (y - \hat{y}(x)) x_i
\end{aligned}$$



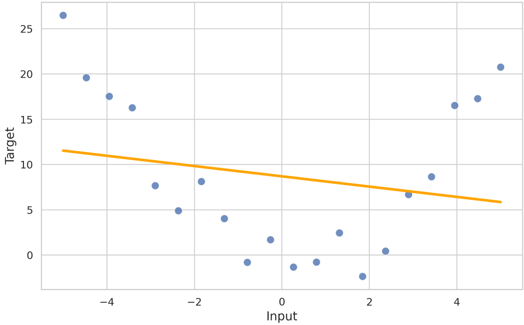

The choice of loss function is important!

# Beyond learning linear functions

We are learning weights for a perceptron: a linear combination of inputs.

**How can we learn non-linear functions?**

Use multiple layers of neurons!



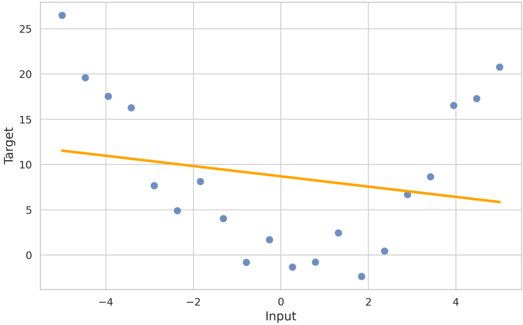

Our perceptron learns the linear best fit, but we can do better.

# Beyond learning linear functions

We are learning weights for a perceptron: a linear combination of inputs.

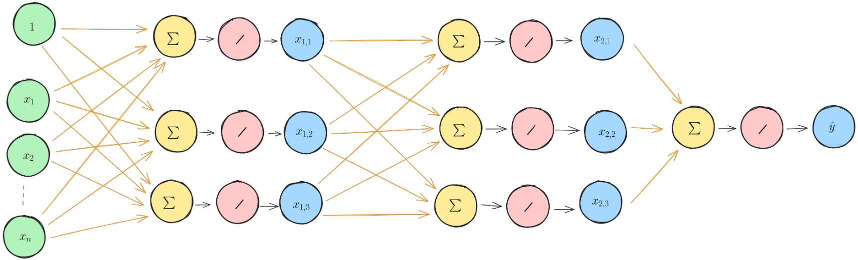
**How can we learn non-linear functions?**

Use multiple layers of neurons!




Our perceptron learns the linear best fit, but we can do better.

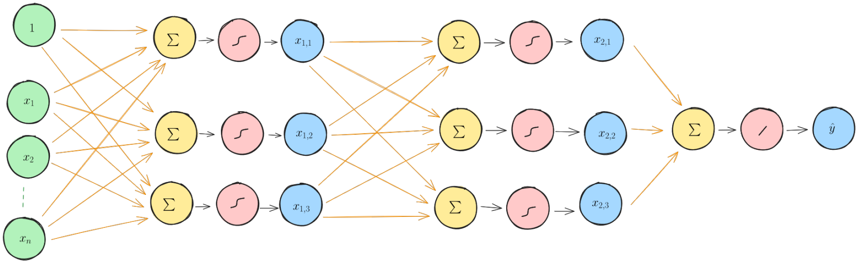
# Multi-layer perceptron (MLP)




**neural network:** a series of layers with weights and activations, transforming an input into an output.

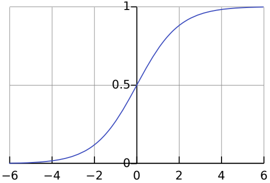
Can this learn non-linear function? Let's put it to the test!

# Multi-layer perceptron (MLP)




We need to introduce non-linearities, e.g., using the sigmoid as the activation functions in hidden layers.

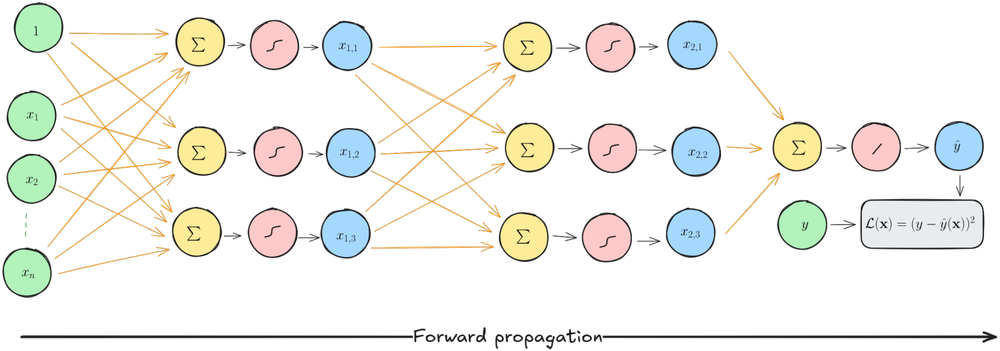
$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \sigma'(x) = (1-\sigma(x))\sigma(x)$$



# Learning with a MLP

Two phases:

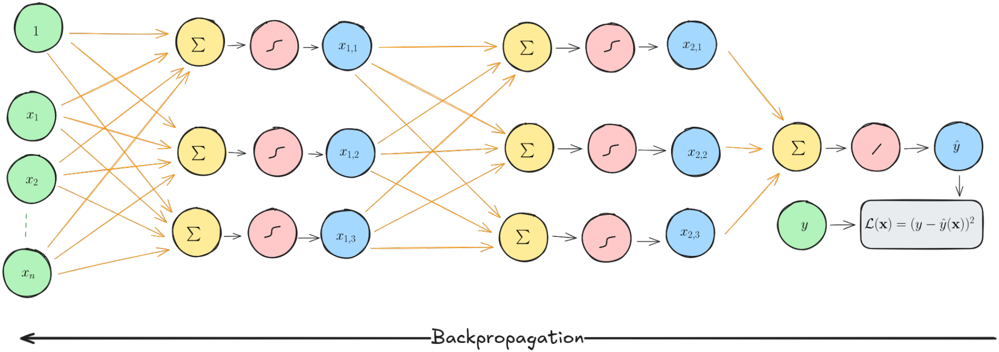
- **forward propagation** (inference)
  input passes through the network to produce the output, used to compute the loss
- backpropagation



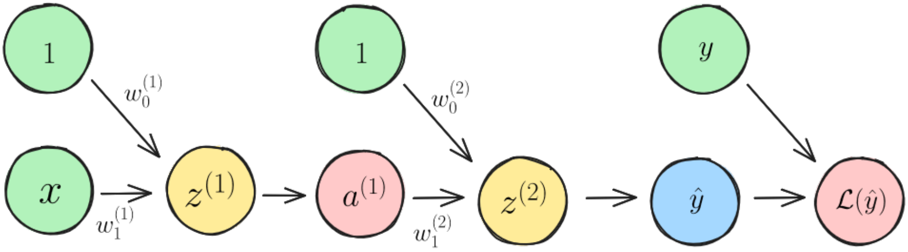
# Learning with a MLP

Two phases:

- forward propagation (inference)
- **backpropagation**
  gradients are propagated backward through the network, allowing us to perform an SGD update



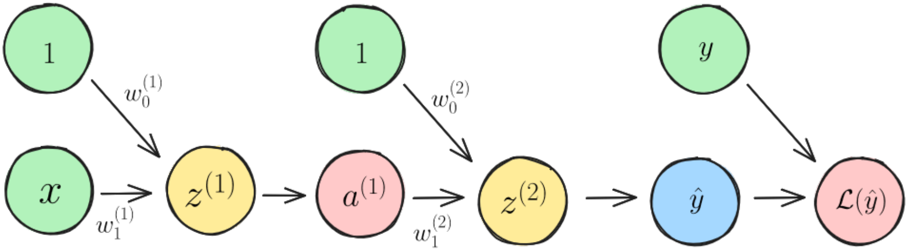
# Backpropagation: simple example




What is the influence of $w_1^{(1)}$ on $\mathcal{L}(\hat{y})$?
How should I modify its value to decrease the loss?

$$\frac{\partial \mathcal{L}}{\partial w_1^{(1)}} = ?$$

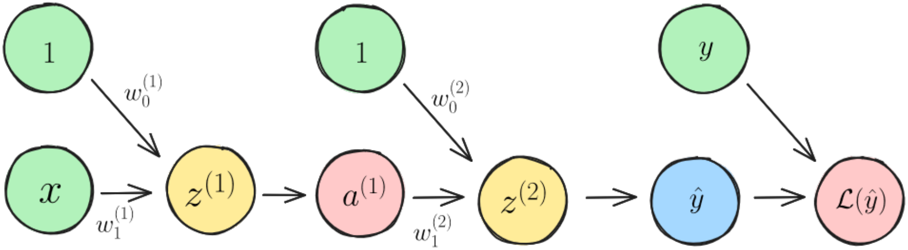
# Backpropagation: simple example




What is the influence of $w_1^{(1)}$ on $\mathcal{L}(\hat{y})$?
How should I modify its value to decrease the loss?

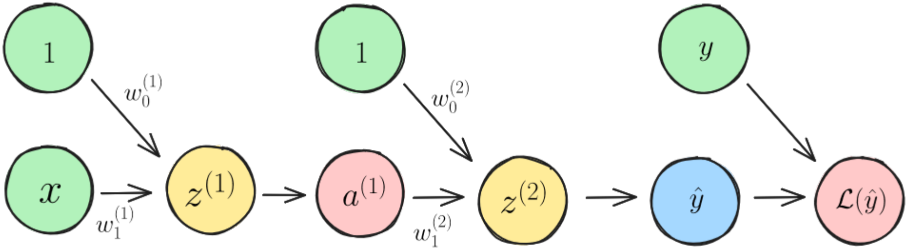
$$\frac{\partial \mathcal{L}}{\partial w_1^{(1)}} = ?$$

# Backpropagation: simple example



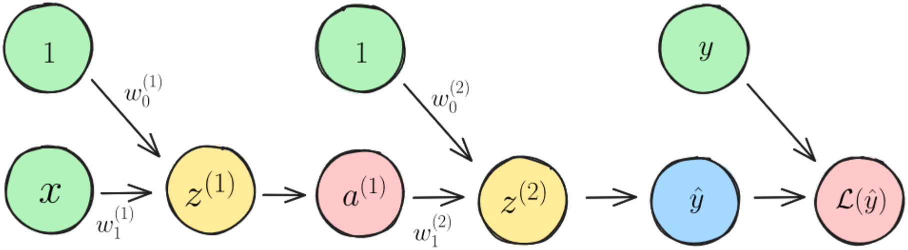

$$\frac{\partial \mathcal{L}}{\partial w_1^{(1)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial \hat{y}}}_{-2(y-\hat{y})} \cdot \underbrace{\frac{\partial \hat{y}}{\partial z^{(2)}}}_{1} \cdot \underbrace{\frac{\partial z^{(2)}}{\partial a^{(1)}}}_{w_1^{(2)}} \cdot \underbrace{\frac{\partial a^{(1)}}{\partial z^{(1)}}}_{\sigma'(z^{(1)})} \cdot \underbrace{\frac{\partial z^{(1)}}{\partial w_1^{(1)}}}_{x}$$

# Backpropagation: simple example



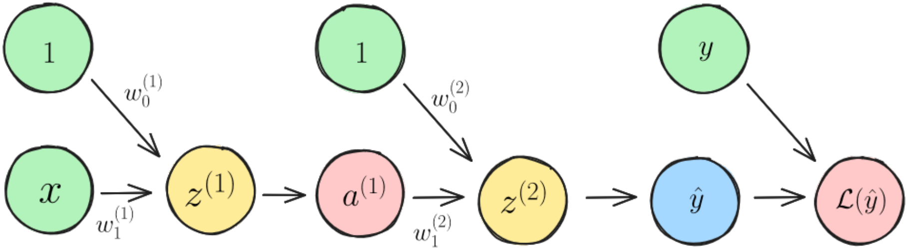

$$\frac{\partial \mathcal{L}}{\partial w_1^{(1)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial \hat{y}}}_{-2(y-\hat{y})} \cdot \underbrace{\frac{\partial \hat{y}}{\partial z^{(2)}}}_{1} \cdot \underbrace{\frac{\partial z^{(2)}}{\partial a^{(1)}}}_{w_1^{(2)}} \cdot \underbrace{\frac{\partial a^{(1)}}{\partial z^{(1)}}}_{\sigma'(z^{(1)})} \cdot \underbrace{\frac{\partial z^{(1)}}{\partial w_1^{(1)}}}_{x}$$

# Backpropagation: simple example



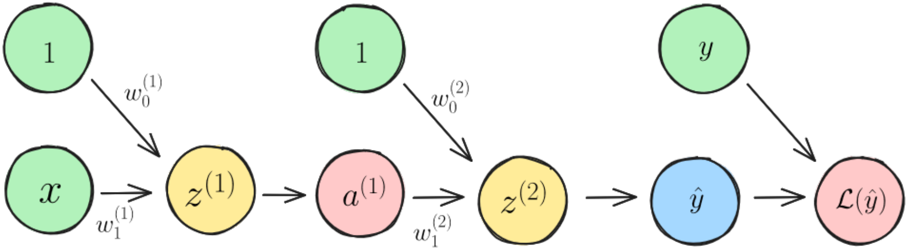

$$\frac{\partial \mathcal{L}}{\partial w_1^{(1)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial \hat{y}}}_{-2(y-\hat{y})} \cdot \underbrace{\frac{\partial \hat{y}}{\partial z^{(2)}}}_{1} \cdot \underbrace{\frac{\partial z^{(2)}}{\partial a^{(1)}}}_{w_1^{(2)}} \cdot \underbrace{\frac{\partial a^{(1)}}{\partial z^{(1)}}}_{\sigma'(z^{(1)})} \cdot \underbrace{\frac{\partial z^{(1)}}{\partial w_1^{(1)}}}_{x}$$

# Backpropagation: simple example



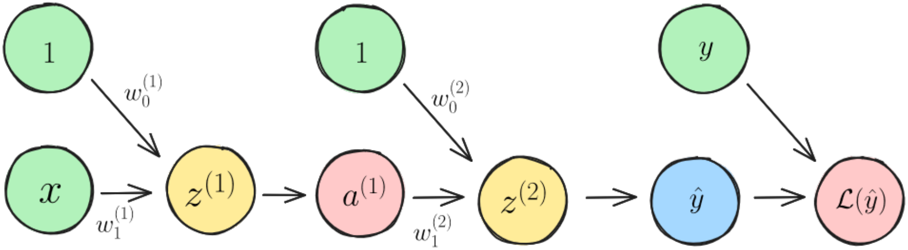

$$\frac{\partial \mathcal{L}}{\partial w_1^{(1)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial \hat{y}}}_{-2(y-\hat{y})} \cdot \underbrace{\frac{\partial \hat{y}}{\partial z^{(2)}}}_{1} \cdot \underbrace{\frac{\partial z^{(2)}}{\partial a^{(1)}}}_{w_1^{(2)}} \cdot \underbrace{\frac{\partial a^{(1)}}{\partial z^{(1)}}}_{\sigma'(z^{(1)})} \cdot \underbrace{\frac{\partial z^{(1)}}{\partial w_1^{(1)}}}_{x}$$

# Backpropagation: simple example



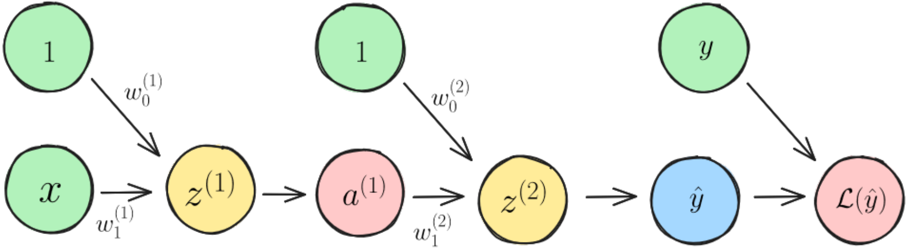

$$\frac{\partial \mathcal{L}}{\partial w_1^{(1)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial \hat{y}}}_{-2(y-\hat{y})} \cdot \underbrace{\frac{\partial \hat{y}}{\partial z^{(2)}}}_{1} \cdot \underbrace{\frac{\partial z^{(2)}}{\partial a^{(1)}}}_{w_1^{(2)}} \cdot \underbrace{\frac{\partial a^{(1)}}{\partial z^{(1)}}}_{\sigma'(z^{(1)})} \cdot \underbrace{\frac{\partial z^{(1)}}{\partial w_1^{(1)}}}_{x}$$

# Backpropagation: simple example




$$\frac{\partial \mathcal{L}}{\partial w_1^{(1)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial \hat{y}}}_{-2(y-\hat{y})} \cdot \underbrace{\frac{\partial \hat{y}}{\partial z^{(2)}}}_{1} \cdot \underbrace{\frac{\partial z^{(2)}}{\partial a^{(1)}}}_{w_1^{(2)}} \cdot \underbrace{\frac{\partial a^{(1)}}{\partial z^{(1)}}}_{\sigma'(z^{(1)})} \cdot \underbrace{\frac{\partial z^{(1)}}{\partial w_1^{(1)}}}_{x}$$

# Backpropagation: simple example



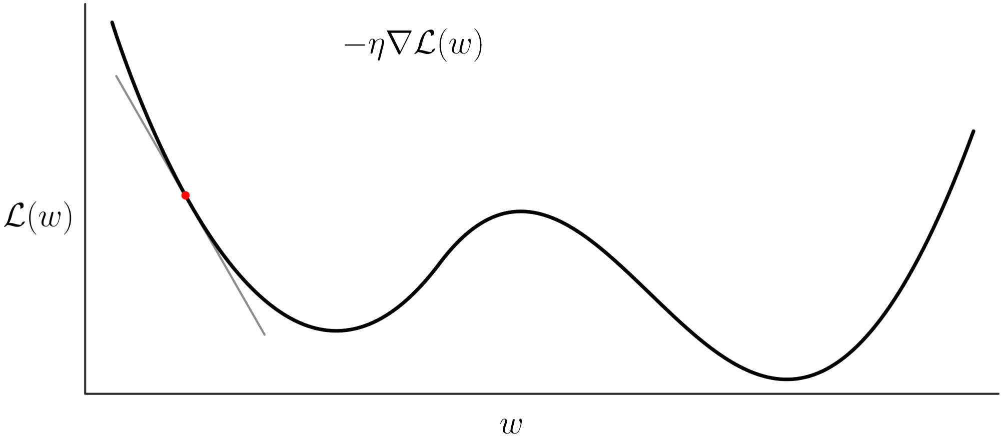

Remembering SGD, $\Delta w = -\eta \nabla \mathcal{J}(w)$

Hence, $\Delta w_1^{(1)} = \eta 2(y - \hat{y}) w_1^{(2)} \sigma(z^{(1)})(1 - \sigma(z^{(1)}))x$

We can reuse computations:

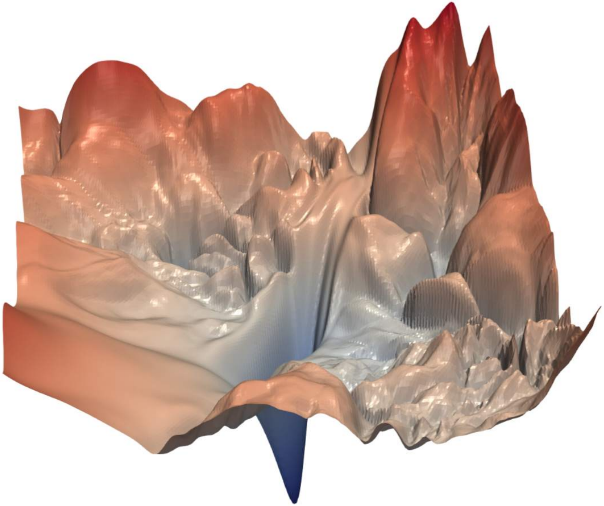
$$\Delta w_0^{(1)} = \eta 2(y - \hat{y}) w_1^{(2)} \sigma(z^{(1)})(1 - \sigma(z^{(1)}))x$$
$$\Delta w_0^{(2)} = \eta 2(y - \hat{y}) w_1^{(2)} \sigma(z^{(1)})(1 - \sigma(z^{(1)}))x$$

# Learning rate



# Finding the optimal weights

In practice, the loss landscape is very complex with billions of dimensions!



# MLP: summary

Multi-layer perceptrons are **universal approximators**: they can approximate any continuous function given that they are wide/deep enough.

**But** convergence can be ineffective (non-convex and high-dimensional space, vanishing gradients...) and may require some tricks in practice.

To play around with MLPs online: https://playground.tensorflow.org

# Convolutional Neural Networks (CNN) [LeCun et al., 1989]

How can we learn from images as inputs?

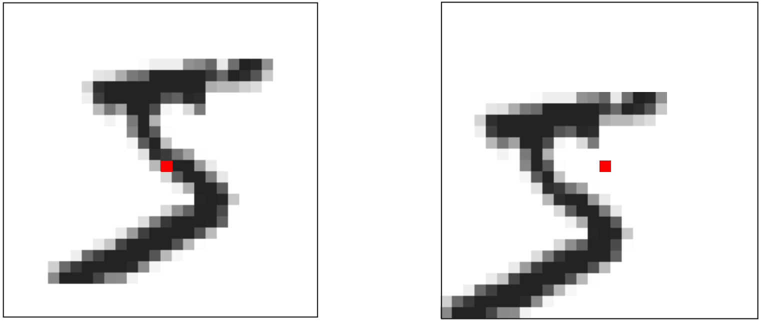
# Convolutional Neural Networks (CNN) [LeCun et al., 1989]

How can we learn from images as inputs?

## (Bad) solution

We can use an MLP! However:

- a huge number of weights to learn (an image has at least 1000 dimensions)
- the problem of translation

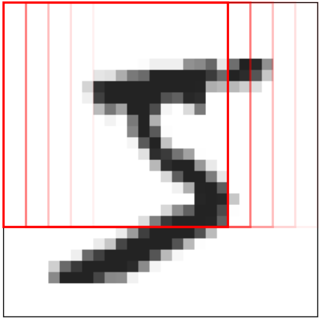


# Convolutional Neural Networks (CNN) [LeCun et al., 1989]

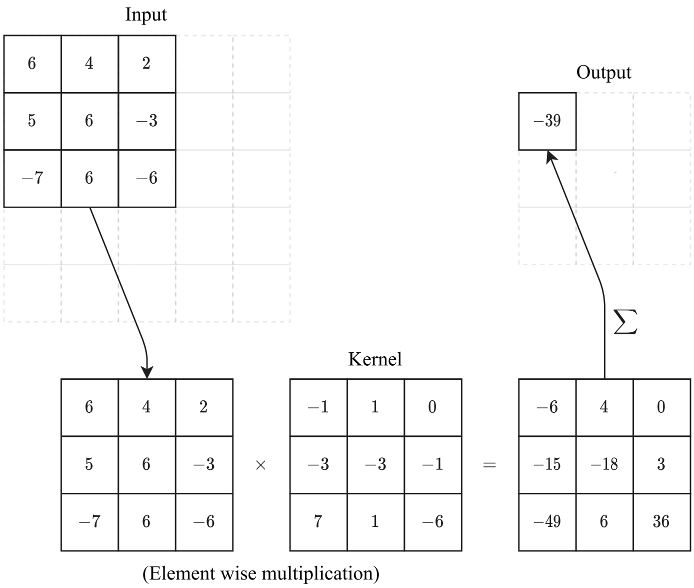
How can we learn from images as inputs?

## Good solution

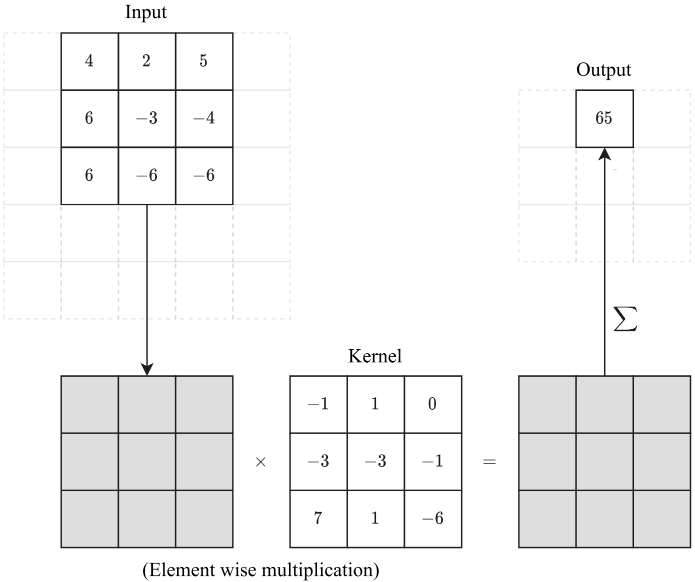
Adapt the neurons and network to perform convolution.



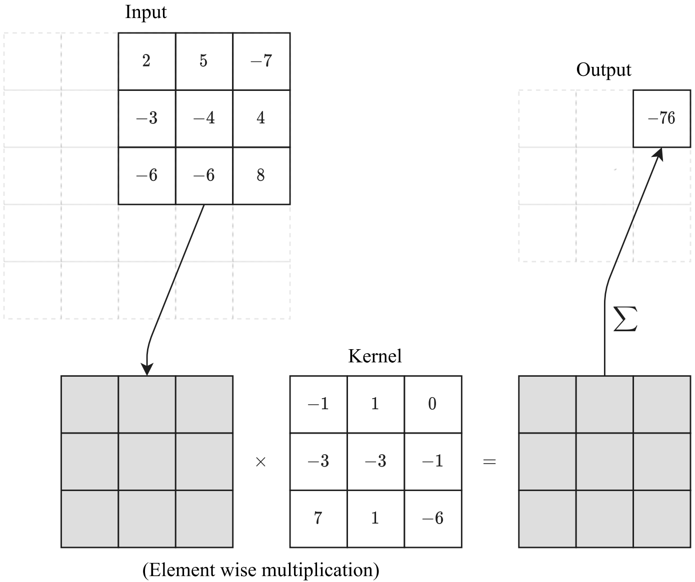
# CNN: the convolution operation



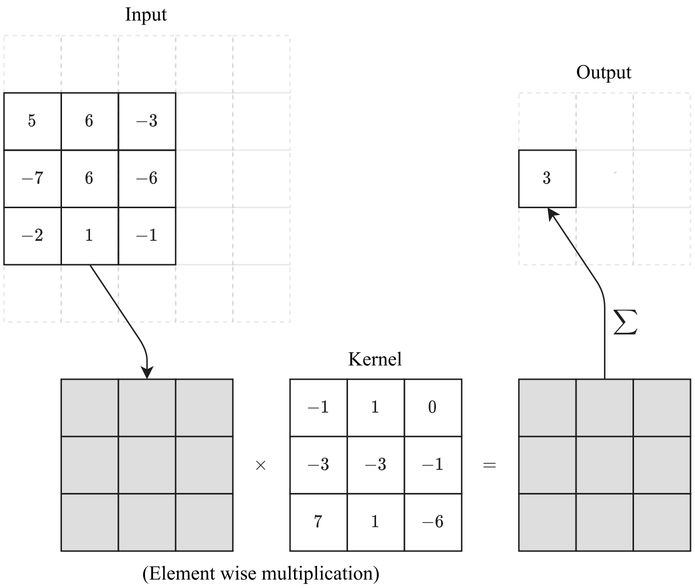
# CNN: the convolution operation



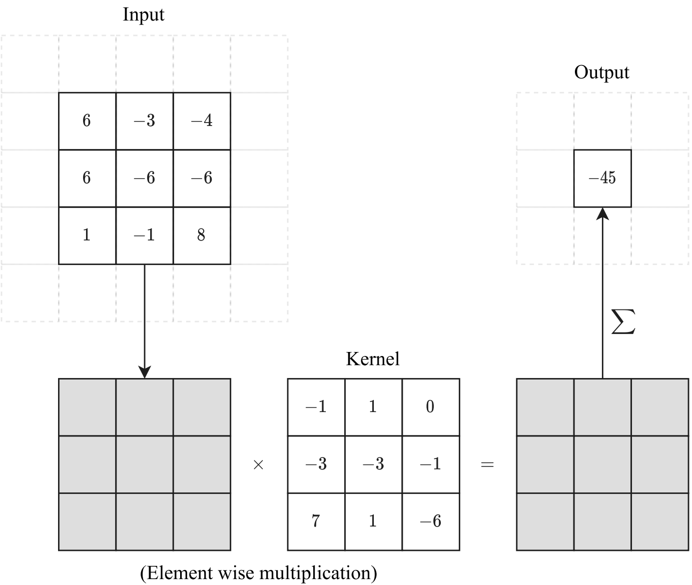
# CNN: the convolution operation



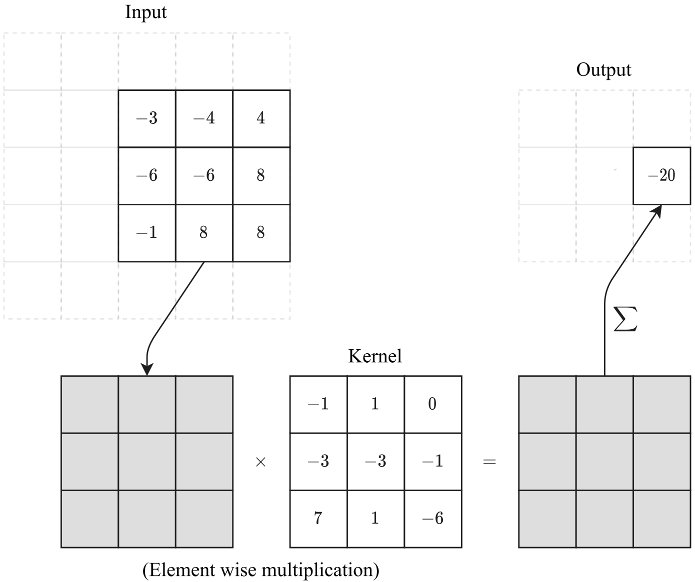
# CNN: the convolution operation



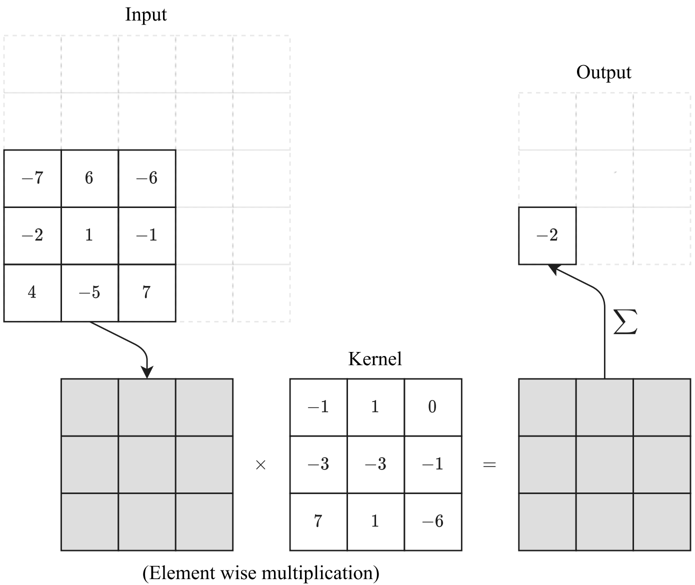
# CNN: the convolution operation



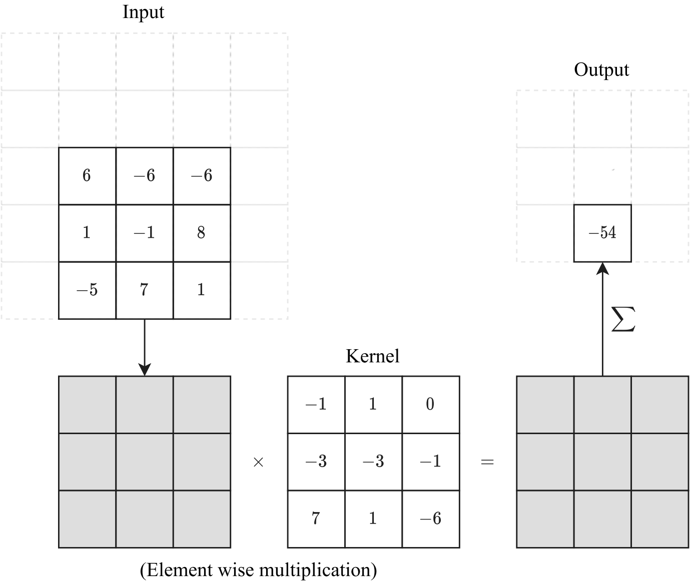
# CNN: the convolution operation



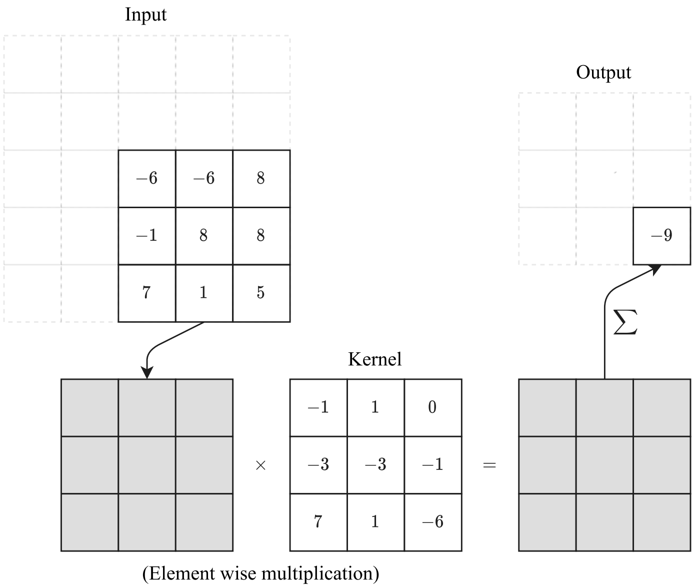
# CNN: the convolution operation



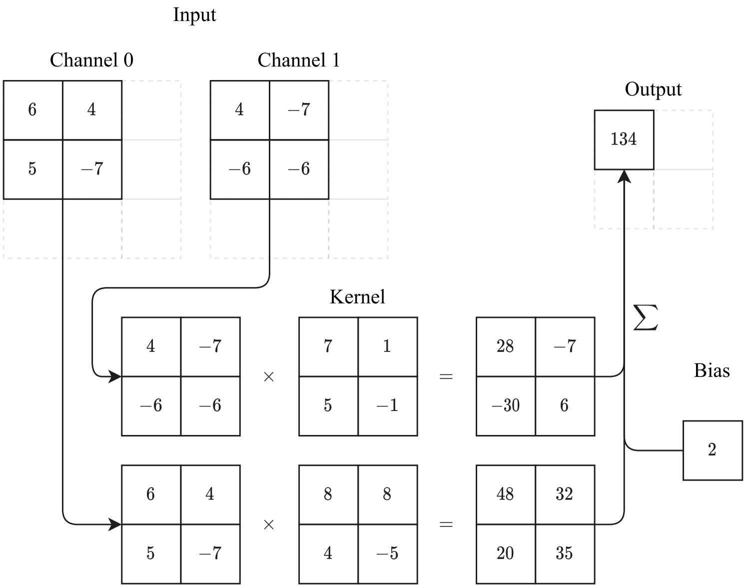
# CNN: the convolution operation



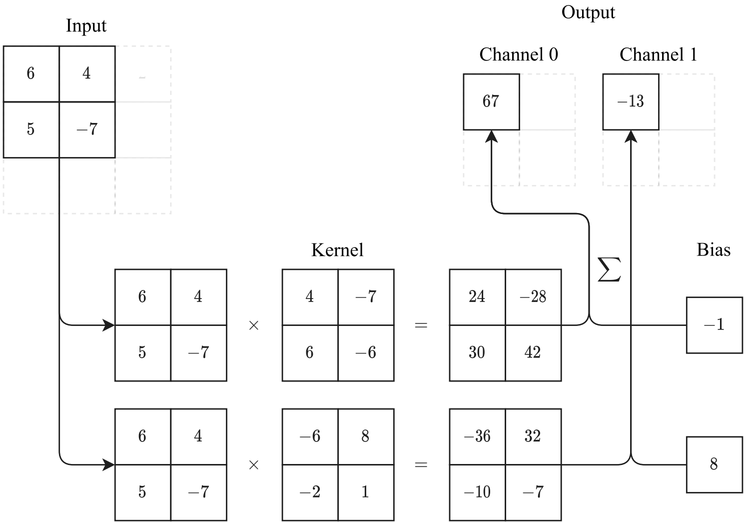
# CNN: the convolution operation



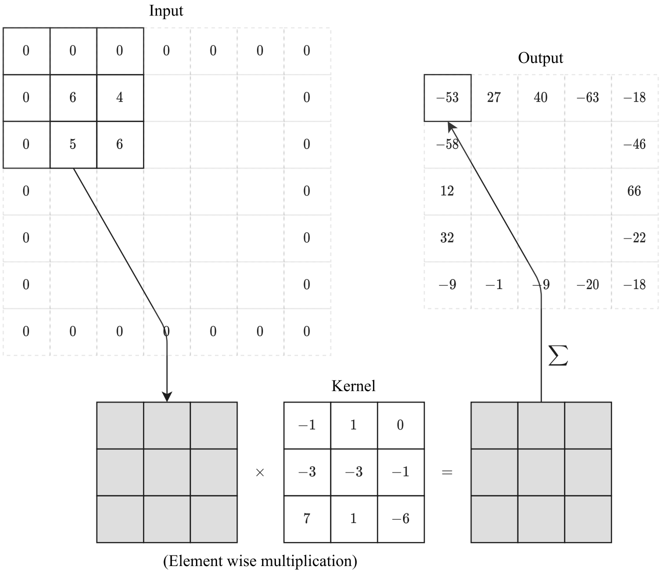
# CNN: convolutional neuron



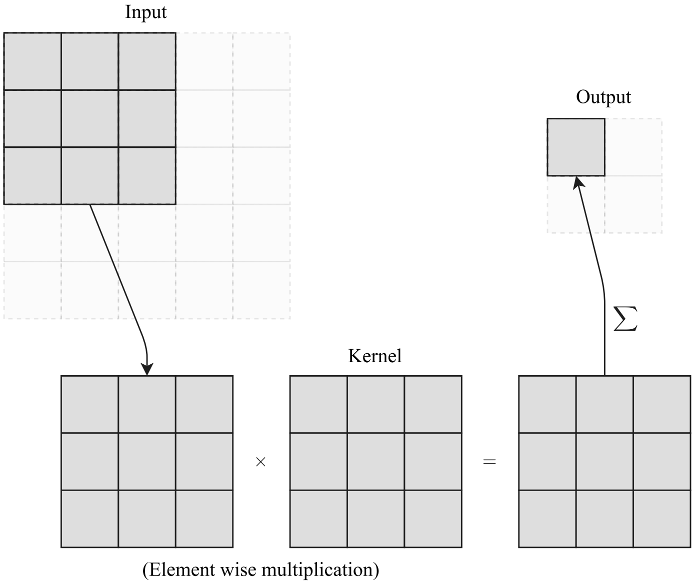
# CNN: convolutional layer



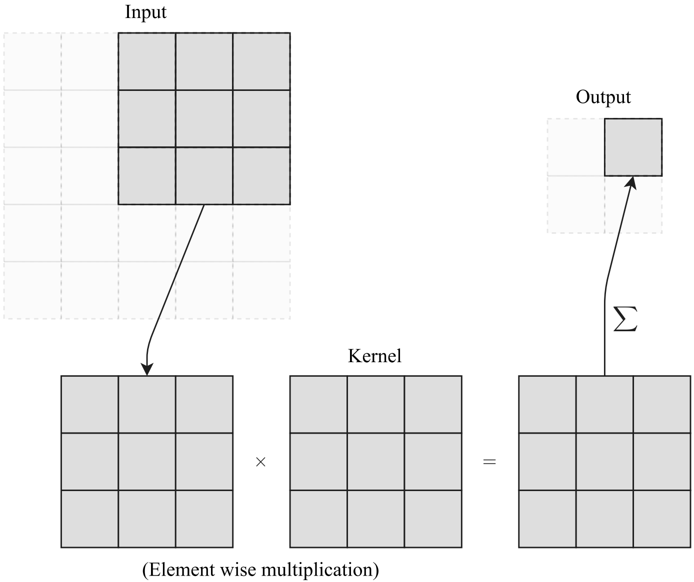
# Convolutional layer - padding



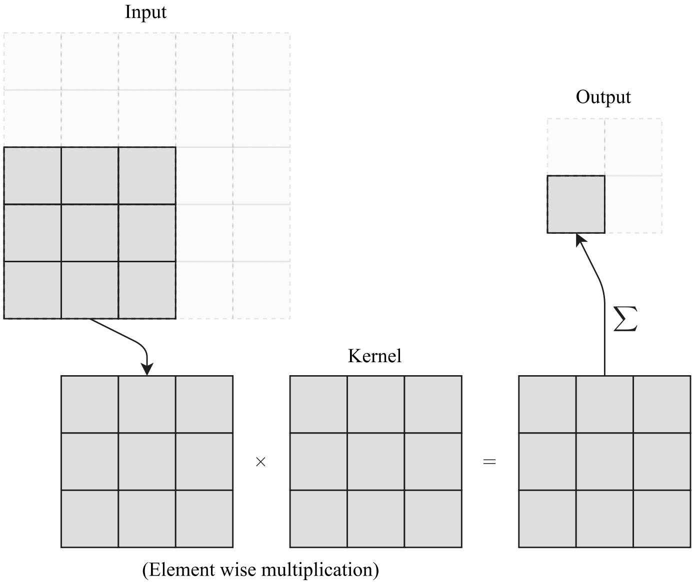
# Convolutional layer - stride



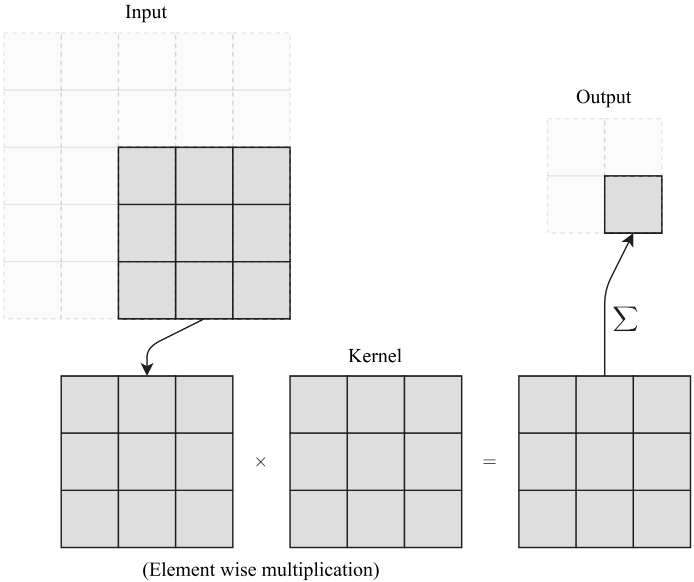
# Convolutional layer - stride



# Convolutional layer - stride



# Convolutional layer - stride



# Convolutional layer - summary

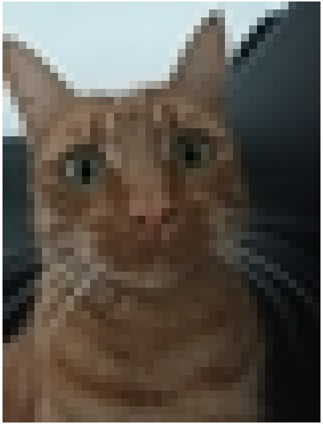
- Input size (of the layer and of every neuron): Channel $\times$ Width $\times$ Height
- Output size (of a neuron):

$$\underbrace{W_{out}}_{\text{width or height}} = \underbrace{\left\lfloor \frac{\overbrace{W_{in} + 2 \times padding}^{\text{Input total width}} - kernel\_size}{stride} \right\rfloor}_{\text{Number of possible kernel positions}} \underbrace{+1}_{\text{Starting position}}$$

- Output size (of a layer): Number of neurons $\times$ $W_{out}$ $\times$ $H_{out}$

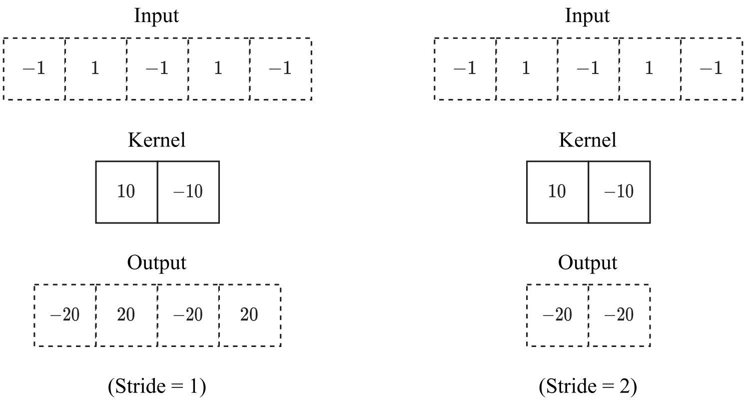
# Pooling layer

A high resolution/dimensionality may not be needed to recognize the content...



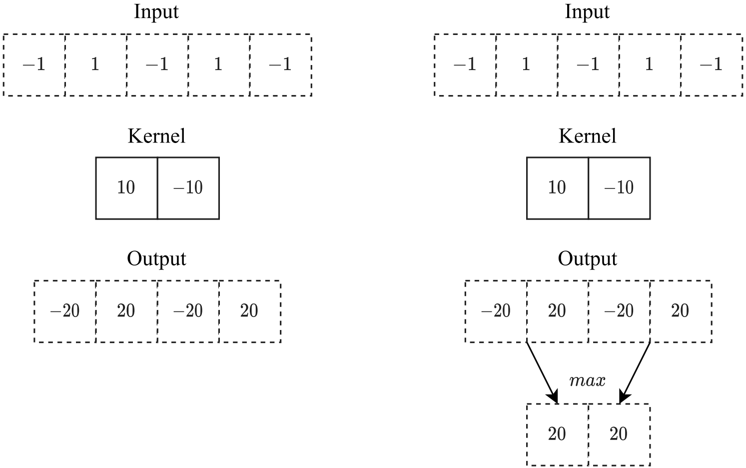
# Pooling layer

A high resolution/dimensionality may not be needed to recognize the content...
...but increasing the stride can be risky...

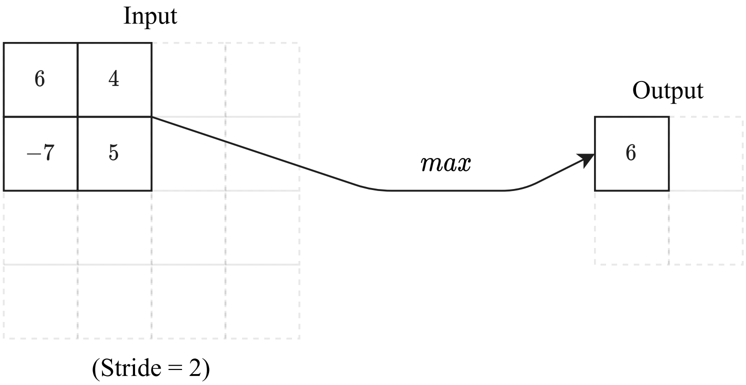


# Pooling layer

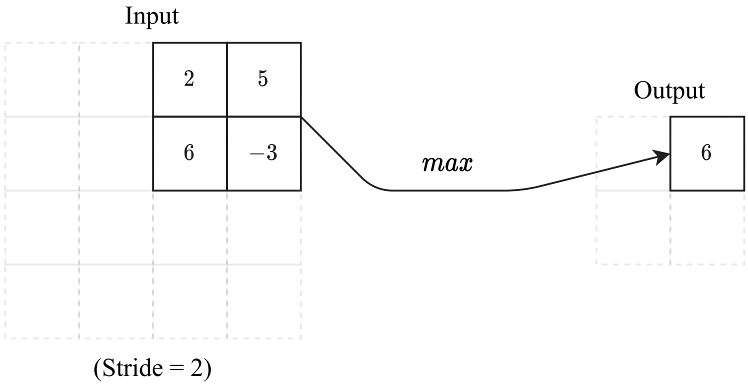
A high resolution/dimensionality may not be needed to recognize the content...
...but increasing the stride can be risky...
...so we tend to prefer max or average pooling



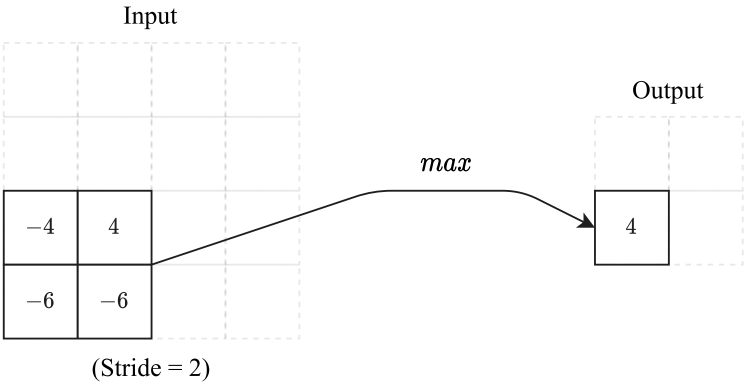
# Pooling layer



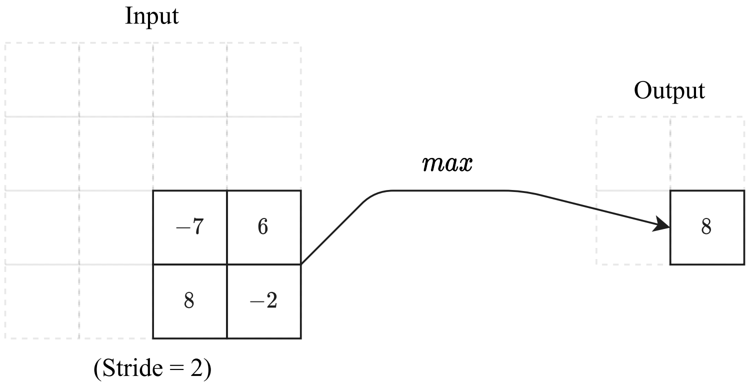
# Pooling layer



# Pooling layer



# Pooling layer

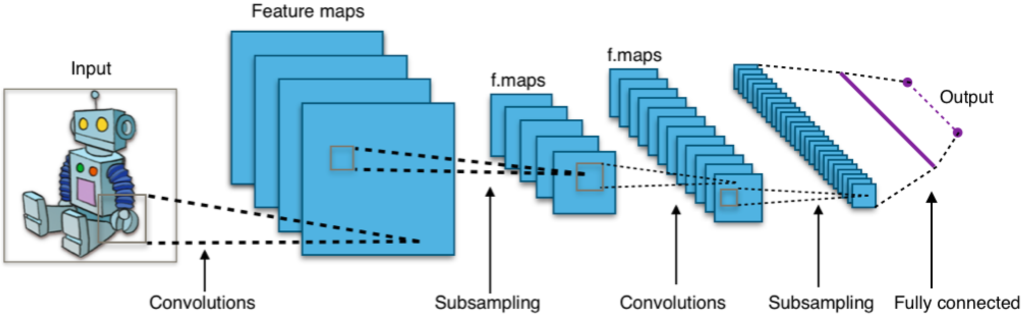


# Pooling layer

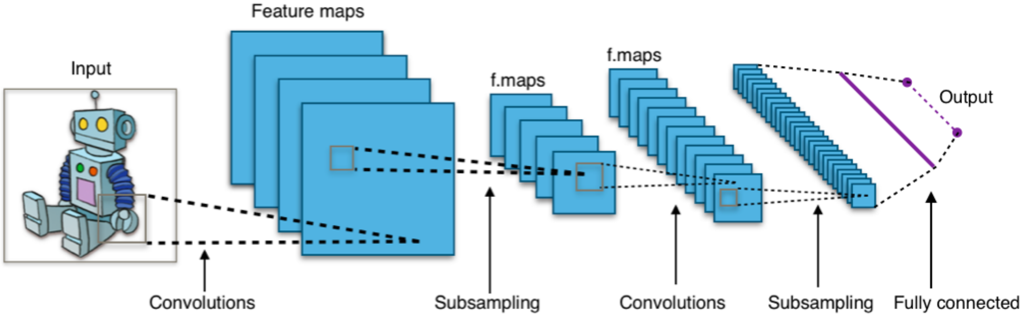
Adding pooling layers helps with:

- removing redundant information
- reducing the amount of computations and memory needed
- making the model more robust to small variations in the input

# CNN - architecture and learning



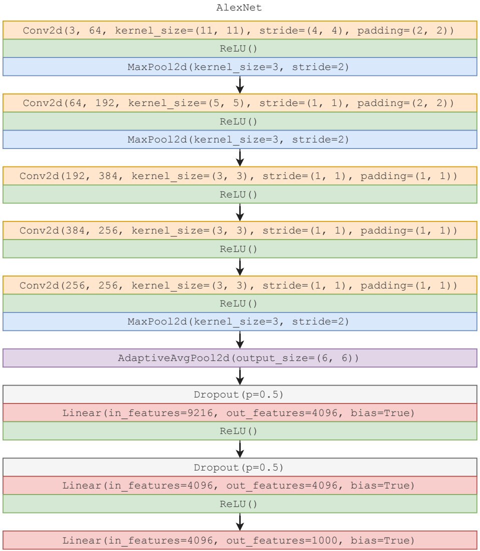
# CNN - architecture and learning



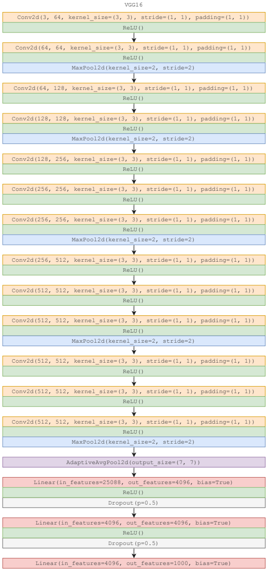

Each neuron does a weighted sum: we can apply SGD on the loss function!

- the weights (kernel) are shared between the neurons of a convolutional layer, so the gradient is aggredated (sum or average)
- for pooling layers, we either backpropagate where the data come from (for max pooling), or do as for any other weighted sum (for average pooling)

# Convolutional Neural Network - AlexNet (2012)



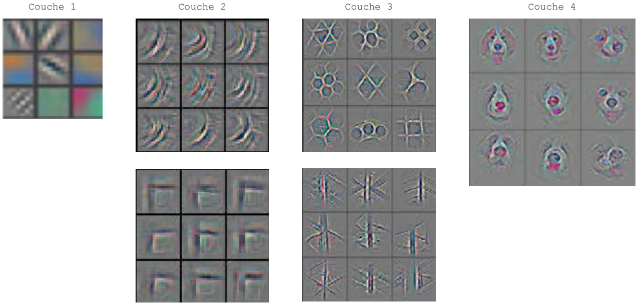
# Convolutional Neural Network - VGG16 (2014)



# CNN summary

## Principles

- Use of convolution for translation invariance and weights sharing
- Pooling to reduce dimensionality
- A MLP at the end of the architecture (no more spatial structure)
- Architecture adaptable to 1D (audio), 3D (video), or graphs...



# Deep learning summary

## Why does it work?

- Universal approximator + a huge amount of data/GPUs… only part of the story
- Hierarchical and automatic learning of features
- Local minima seems quite good [Choromanska et al., 2015]
- Deals better with data/tasks in practice (compared to shallow networks)
- Easy to incorporate inductive biases (e.g., convolution for images)

# Deep learning summary

**Why does it work?**

- Universal approximator + a huge amount of data/GPUs... only part of the story
- Hierarchical and automatic learning of features
- Local minima seems quite good [Choromanska et al., 2015]
- Deals better with data/tasks in practice (compared to shallow networks)
- Easy to incorporate inductive biases (e.g., convolution for images)

# Deep learning summary

## Why does it work?

- Universal approximator + a huge amount of data/GPUs... only part of the story
- Hierarchical and automatic learning of features
- Local minima seems quite good [Choromanska et al., 2015]
- Deals better with data/tasks in practice (compared to shallow networks)
- Easy to incorporate inductive biases (e.g., convolution for images)

# Deep learning summary

**Why does it work?**

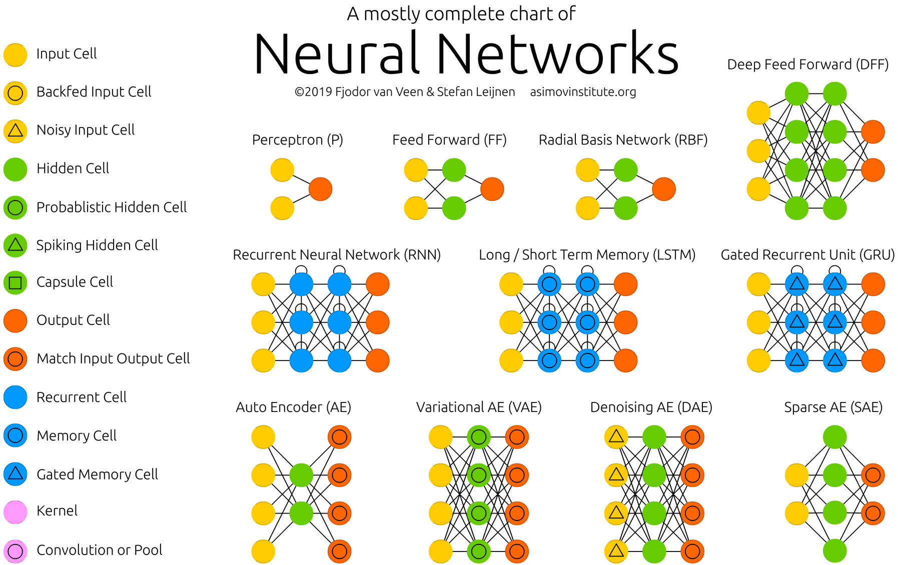
- Universal approximator + a huge amount of data/GPUs... only part of the story
- Hierarchical and automatic learning of features
- Local minima seems quite good [Choromanska et al., 2015]
- Deals better with data/tasks in practice (compared to shallow networks)
- Easy to incorporate inductive biases (e.g., convolution for images)

# Deep learning summary

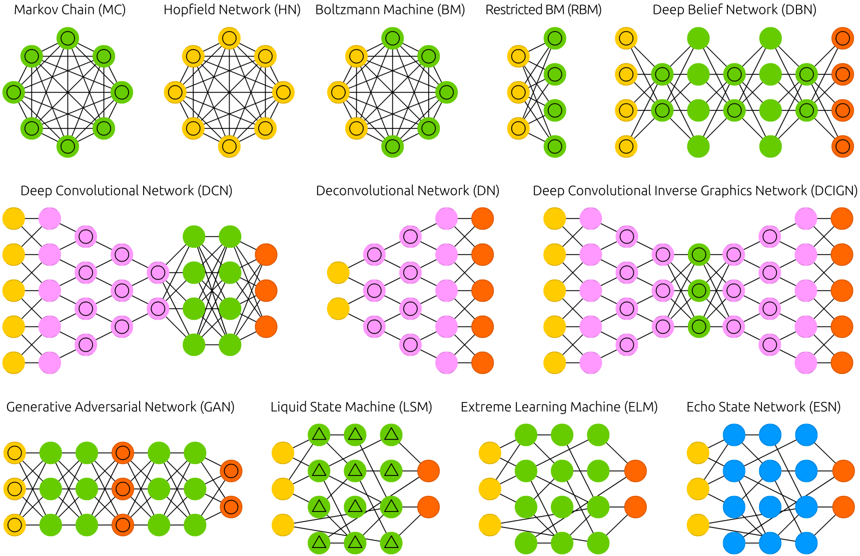
## Why does it work?

- Universal approximator + a huge amount of data/GPUs... only part of the story
- Hierarchical and automatic learning of features
- Local minima seems quite good [Choromanska et al., 2015]
- Deals better with data/tasks in practice (compared to shallow networks)
- Easy to incorporate inductive biases (e.g., convolution for images)

# Deep learning zoo

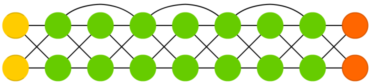


# Deep learning zoo

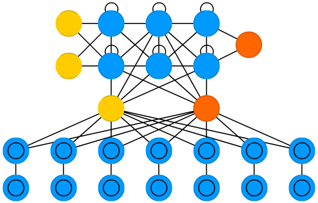


# Deep learning zoo

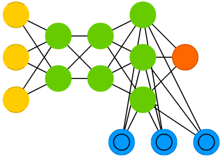
Deep Residual Network (DRN)



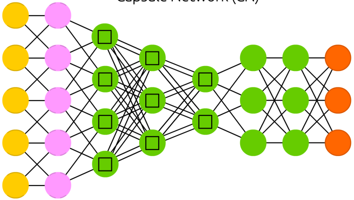
Differentiable Neural Computer (DNC)



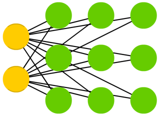
Neural Turing Machine (NTM)



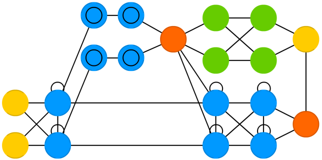
Capsule Network (CN)



Kohonen Network (KN)

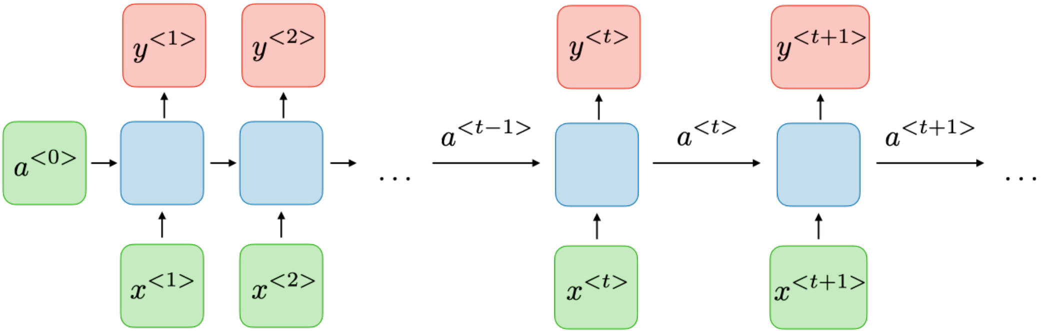


Attention Network (AN)



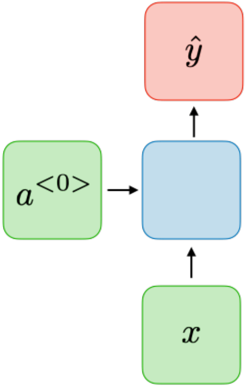
https://www.asimovinstitute.org/neural-network-zoo/

# Recurrent neural networks (RNN)




Diagrams by the Amidi brothers

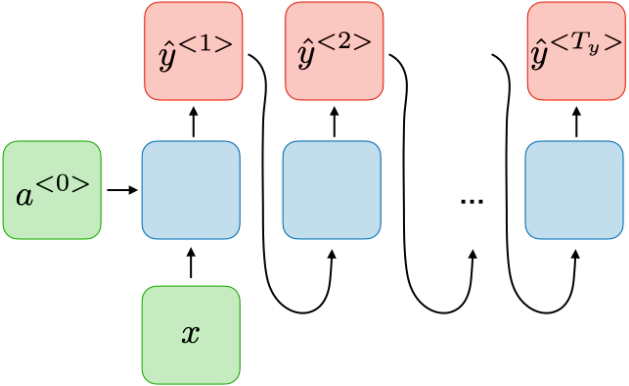
# Recurrent neural networks (RNN) - applications




one to one

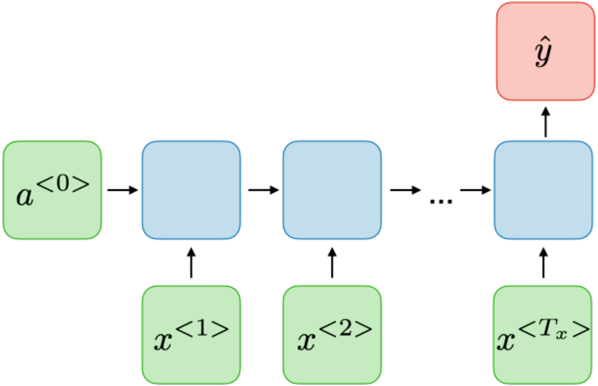
Diagrams by the Amidi brothers

# Recurrent neural networks (RNN) - applications



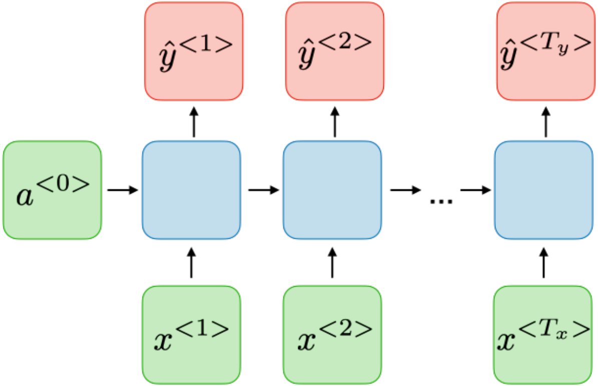

one to many

# Recurrent neural networks (RNN) - applications




many to one

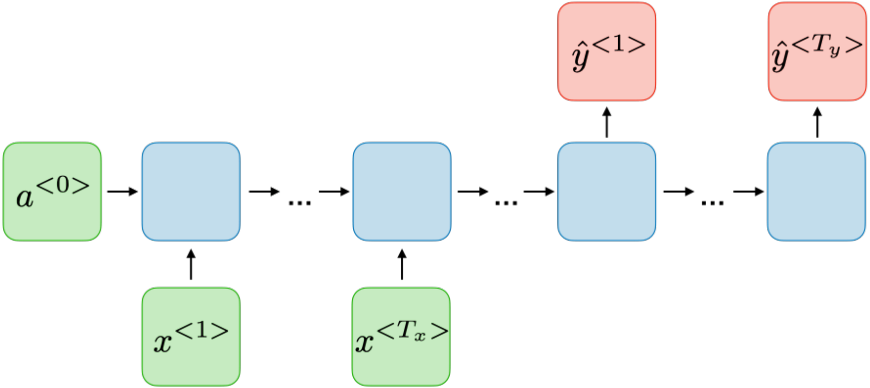
# Recurrent neural networks (RNN) - applications




many to many (aligned)

Diagrams by the Amidi brothers

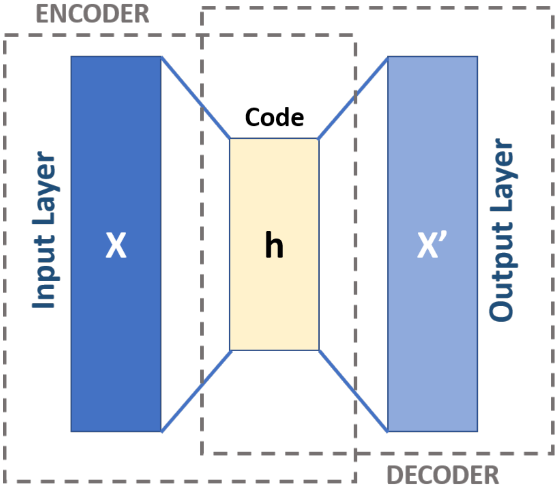
# Recurrent neural networks (RNN) - applications




many to many (split)

Diagrams by the Amidi brothers

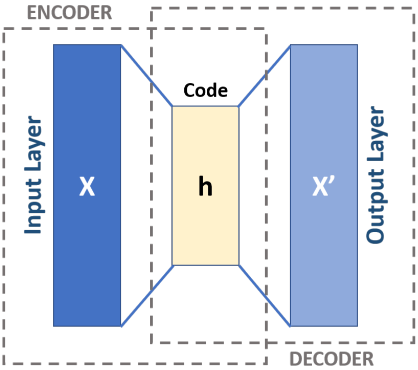
# Autoencoders (AE)




Unsupervised: $\mathcal{L}(x) = d(x, D(E(x)))$

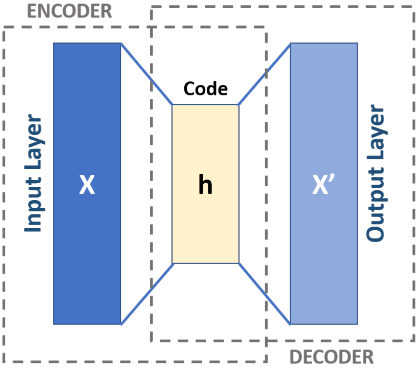
# Autoencoders (AE) - applications

- Denoising:
  $\mathcal{L}(x) = d(x, D(E(x + \epsilon)))$
- Dimensionality reduction using the latent variable/code
- Fraud detection: reconstruction error increases on anomalous data points
- Image compression (convolutional autoencoders)



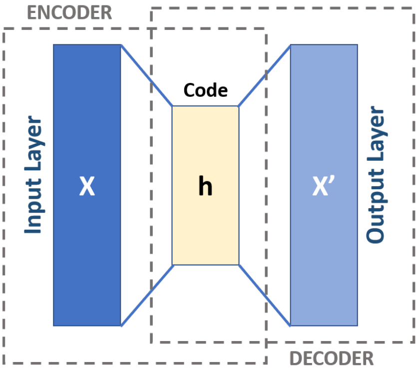
# Autoencoders (AE) - applications

- Denoising:
  $\mathcal{L}(x) = d(x, D(E(x + \epsilon)))$
- Dimensionality reduction using the latent variable/code
- Fraud detection: reconstruction error increases on anomalous data points
- Image compression (convolutional autoencoders)



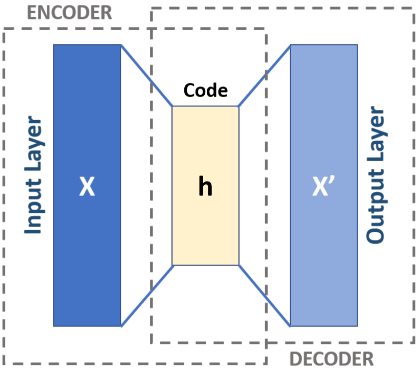
# Autoencoders (AE) - applications

- Denoising:
  $\mathcal{L}(x) = d(x, D(E(x + \epsilon)))$
- Dimensionality reduction using the latent variable/code
- Fraud detection: reconstruction error increases on anomalous data points
- Image compression (convolutional autoencoders)

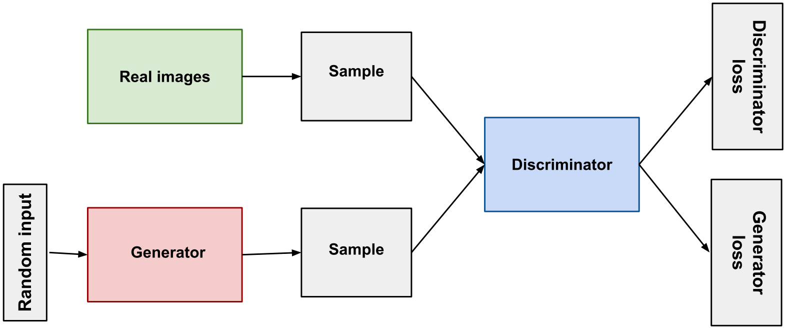


# Autoencoders (AE) - applications

- Denoising:
  $\mathcal{L}(x) = d(x, D(E(x+\epsilon)))$
- Dimensionality reduction using the latent variable/code
- Fraud detection: reconstruction error increases on anomalous data points
- Image compression (convolutional autoencoders)



# Generative adversarial networks (GAN)




https://thispersondoesnotexist.com/

# Deep learning in practice

- Data (pre) processing
- Choice of the model
- Training
- Getting the better performances

# Deep learning in practice

- Data (pre) processing
  - Balanced/representative data
  - Data augmentations: e.g., changing the color, zoom, or orientation for images
  - Normalizing data: $\frac{x-\bar{x}}{\sigma(x)}$
  - Use of mini batchs
  - Use of train/test/validation datasets
- Choice of the model
- Training
- Getting the better performances

# Deep learning in practice

- Data (pre) processing
  - Balanced/representative data
  - Data augmentations: e.g., changing the color, zoom, or orientation for images
  - Normalizing data: $\frac{x-\bar{x}}{\sigma(x)}$
  - Use of mini batchs
  - Use of train/test/validation datasets
- Choice of the model
- Training
- Getting the better performances

# Deep learning in practice

- Data (pre) processing
  - Balanced/representative data
  - Data augmentations: e.g., changing the color, zoom, or orientation for images
  - Normalizing data: $\frac{x-\bar{x}}{\sigma(x)}$
  - Use of mini batchs
  - Use of train/test/validation datasets
- Choice of the model
- Training
- Getting the better performances

# Deep learning in practice

- Data (pre) processing
  - Balanced/representative data
  - Data augmentations: e.g., changing the color, zoom, or orientation for images
  - Normalizing data: $\frac{x-\bar{x}}{\sigma(x)}$
  - Use of mini batchs
  - Use of train/test/validation datasets
- Choice of the model
- Training
- Getting the better performances

# Deep learning in practice

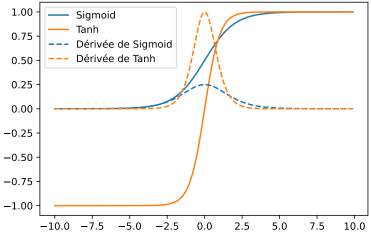
- Data (pre) processing
  - Balanced/representative data
  - Data augmentations: e.g., changing the color, zoom, or orientation for images
  - Normalizing data: $\frac{x-\bar{x}}{\sigma(x)}$
  - Use of mini batchs
  - Use of train/test/validation datasets
- Choice of the model
- Training
- Getting the better performances

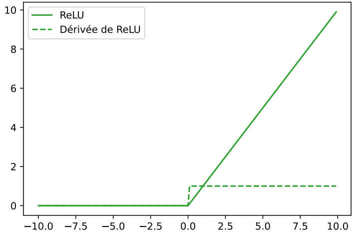
# Deep learning in practice

- Data (pre) processing
- Choice of the model
  - Type of architecture w.r.t. the data / problem
  - Choice of the activation function
- Training
- Getting the better performances

# Deep learning in practice

- Data (pre) processing
- Choice of the model
  - Type of architecture w.r.t. the data / problem
  - Choice of the activation function




- Training
- Getting the better performances

# Deep learning in practice

- Data (pre) processing
- Choice of the model
- Training
  - Loss function:
    - regression: (Mean) squared error: $\frac{1}{2}\sum_{x,i}(t_i - y_i)^2$
    - classification: softmax + cross entropy: $\sum_x -\log \frac{e^{y_t}}{\sum_i e^{y_i}}$
  - Regularisation: $+||\mathbf{w}||$ in the loss function or *drop out* (some weights are randomly set to 0)
  - Choice of the optimizer: SGD, SGD + momentum, Adagrad, Adam
  - Overfitting, underfitting, early stopping
- Getting the better performances

# Deep learning in practice

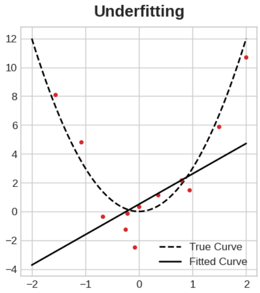
- Data (pre) processing
- Choice of the model
- Training
  - Loss function:
    - regression: (Mean) squared error: $\frac{1}{2}\sum_{x,i}(t_i - y_i)^2$
    - classification: softmax + cross entropy: $\sum_x -\log \frac{e^{y_t}}{\sum_i e^{y_i}}$
  - Regularisation: $+||\mathbf{w}||$ in the loss function or *drop out* (some weights are randomly set to 0)
  - Choice of the optimizer: SGD, SGD + momentum, Adagrad, Adam
  - Overfitting, underfitting, early stopping
- Getting the better performances

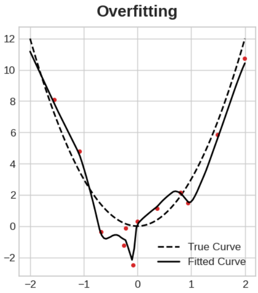
# Deep learning in practice

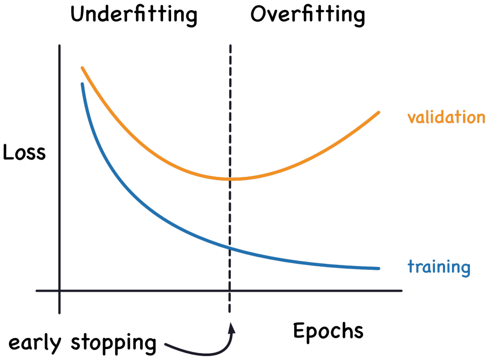
- Data (pre) processing
- Choice of the model
- Training
  - Loss function:
    - regression: (Mean) squared error: $\frac{1}{2}\sum_{x,i}(t_i - y_i)^2$
    - classification: softmax + cross entropy: $\sum_x -\log \frac{e^{y_t}}{\sum_i e^{y_i}}$
  - Regularisation: $+||\mathbf{w}||$ in the loss function or *drop out* (some weights are randomly set to 0)
  - Choice of the optimizer: SGD, SGD + momentum, Adagrad, Adam
  - Overfitting, underfitting, early stopping
- Getting the better performances

# Deep learning in practice

## Overfitting, underfitting, and early stopping



# Deep learning in practice

- Data (pre) processing
- Choice of the model
- Training
  - Loss function:
    - regression: (Mean) squared error: $\frac{1}{2}\sum_{x,i}(t_i - y_i)^2$
    - classification: softmax + cross entropy: $\sum_x -\log \frac{e^{y_t}}{\sum_i e^{y_i}}$
  - Regularisation: $+||\mathbf{w}||$ in the loss function or *drop out* (some weights are randomly set to 0)
  - Choice of the optimizer: SGD, SGD + momentum, Adagrad, Adam
  - Overfitting, underfitting, early stopping
- Getting the better performances

# Deep learning in practice

- Data (pre) processing
- Choice of the model
- Training
- Getting the better performances
  - Hyperparameters: start with default values...
    - for the MLP, usually a big $1^{\text{st}}$ layer, then decreasing size
    - for the CNN, usually the number of channels increases at each layer (to "compensate" the decreasing size of the feature maps)
  - ...then empirically find what works on the validation set (typically with a *grid search*)
  - *Fine tuning*: use of a ("general") pre trained model that is locally adapted to the data

# Deep learning in practice

- Data (pre) processing
- Choice of the model
- Training
- Getting the better performances
  - Hyperparameters: start with default values...
    - for the MLP, usually a big $1^{\text{st}}$ layer, then decreasing size
    - for the CNN, usually the number of channels increases at each layer (to "compensate" the decreasing size of the feature maps)
  - ...then empirically find what works on the validation set (typically with a *grid search*)
  - *Fine tuning*: use of a ("general") pre trained model that is locally adapted to the data

# Deep learning in practice

- Data (pre) processing
- Choice of the model
- Training
- Getting the better performances
  - Hyperparameters: start with default values…
    - for the MLP, usually a big $1^{\text{st}}$ layer, then decreasing size
    - for the CNN, usually the number of channels increases at each layer (to ”compensate” the decreasing size of the feature maps)
  - …then empirically find what works on the validation set (typically with a *grid search*)
  - *Fine tuning*: use of a (”general”) pre trained model that is locally adapted to the data

# *Limitations - Over training?*

| data aug | dropout | weight decay | top-1 train | top-5 train | top-1 test | top-5 test |
|---|---|---|---|---|---|---|
| ImageNet 1000 classes with the original labels | | | | | | |
| yes | yes | yes | 92.18 | 99.21 | 77.84 | 93.92 |
| yes | no | no | 92.33 | 99.17 | 72.95 | 90.43 |
| no | no | yes | 90.60 | 100.0 | 67.18 (72.57) | 86.44 (91.31) |
| no | no | no | 99.53 | 100.0 | 59.80 (63.16) | 80.38 (84.49) |
| Alexnet (Krizhevsky et al., 2012) | | | - | - | - | 83.6 |
| ImageNet 1000 classes with random labels | | | | | | |
| no | yes | yes | 91.18 | 97.95 | 0.09 | 0.49 |
| no | no | yes | 87.81 | 96.15 | 0.12 | 0.50 |
| no | no | no | 95.20 | 99.14 | 0.11 | 0.56 |

Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:1611.03530.

# Limitations - Bias

| Classifier | Metric | All | F | M | Darker | Lighter | DF | DM | LF | LM |
|---|---|---|---|---|---|---|---|---|---|---|
| **MSFT** | PPV(%) | 93.7 | 89.3 | 97.4 | 87.1 | 99.3 | 79.2 | 94.0 | 98.3 | **100** |
| | Error Rate(%) | 6.3 | 10.7 | 2.6 | 12.9 | 0.7 | **20.8** | 6.0 | 1.7 | 0.0 |
| | TPR (%) | 93.7 | 96.5 | 91.7 | 87.1 | 99.3 | 92.1 | 83.7 | **100** | 98.7 |
| | FPR (%) | 6.3 | 8.3 | 3.5 | 12.9 | 0.7 | **16.3** | 7.9 | 1.3 | 0.0 |
| **Face++** | PPV(%) | 90.0 | 78.7 | 99.3 | 83.5 | 95.3 | 65.5 | **99.3** | 94.0 | 99.2 |
| | Error Rate(%) | 10.0 | 21.3 | 0.7 | 16.5 | 4.7 | **34.5** | 0.7 | 6.0 | 0.8 |
| | TPR (%) | 90.0 | 98.9 | 85.1 | 83.5 | 95.3 | 98.8 | 76.6 | **98.9** | 92.9 |
| | FPR (%) | 10.0 | 14.9 | 1.1 | 16.5 | 4.7 | **23.4** | 1.2 | 7.1 | 1.1 |
| **IBM** | PPV(%) | 87.9 | 79.7 | 94.4 | 77.6 | 96.8 | 65.3 | 88.0 | 92.9 | **99.7** |
| | Error Rate(%) | 12.1 | 20.3 | 5.6 | 22.4 | 3.2 | **34.7** | 12.0 | 7.1 | 0.3 |
| | TPR (%) | 87.9 | 92.1 | 85.2 | 77.6 | 96.8 | 82.3 | 74.8 | **99.6** | 94.8 |
| | FPR (%) | 12.1 | 14.8 | 7.9 | 22.4 | 3.2 | **25.2** | 17.7 | 5.20 | 0.4 |

Buolamwini, Joy, and Timnit Gebru. "Gender shades: Intersectional accuracy disparities in commercial gender classification." Conference on Fairness, Accountability and Transparency. 2018

# Limitations - Environmental impact

- Impact of digital technologies is estimated between 1.5 and 4% of global greenhouse gases emissions ($\sim$37 billions of tons eqCO2 in 2023)
- AI contribution (very) difficult to estimate, but clearly growing.
- For instance (estimations), GPT3 training required 1,287 MWh ($\sim$500 tons eqCO2)...
- ...but ChatGPT inference needs 564 MWh ($\sim$220 tons eqCO2) every day (i.e. $\sim$80000 tons eqCO2 per year).

Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G., & Friday, A. (2021). The climate impact of ICT: A review of estimates, trends and regulations. arXiv preprint arXiv:2102.02622.

# Limitations - Environmental impact

- Impact of digital technologies is estimated between 1.5 and 4% of global greenhouse gases emissions (∼37 billions of tons eqCO2 in 2023)
- AI contribution (very) difficult to estimate, but clearly growing.
- For instance (estimations), GPT3 training required 1,287 MWh (∼500 tons eqCO2)...
- ...but ChatGPT inference needs 564 MWh (∼220 tons eqCO2) every day (i.e. ∼80000 tons eqCO2 per year).

Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G., & Friday, A. (2021). The climate impact of ICT: A review of estimates, trends and regulations. arXiv preprint arXiv:2102.02622.

# Limitations - Environmental impact

- Impact of digital technologies is estimated between 1.5 and 4% of global greenhouse gases emissions (∼37 billions of tons eqCO2 in 2023)
- AI contribution (very) difficult to estimate, but clearly growing.
- For instance (estimations), GPT3 training required 1,287 MWh (∼500 tons eqCO2)...
- ...but ChatGPT inference needs 564 MWh (∼220 tons eqCO2) every day (i.e. ∼80000 tons eqCO2 per year).

Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G., & Friday, A. (2021). The climate impact of ICT: A review of estimates, trends and regulations. arXiv preprint arXiv:2102.02622.
Patel, D., & Ahmad, A. (2023). The Inference Cost Of Search Disruption–Large Language Model Cost Analysis. Verfügbar unter https://www. semianalysis. com/p/theinference-cost-of-search-disruption.

# Limitations - Environmental impact

- Impact of digital technologies is estimated between 1.5 and 4% of global greenhouse gases emissions ($\sim$37 billions of tons eqC02 in 2023)
- AI contribution (very) difficult to estimate, but clearly growing.
- For instance (estimations), GPT3 training required 1,287 MWh ($\sim$500 tons eqCO2)...
- ...but ChatGPT inference needs 564 MWh ($\sim$220 tons eqCO2) every day (i.e. $\sim$80000 tons eqCO2 per year).

Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G., & Friday, A. (2021). The climate impact of ICT: A review of estimates, trends and regulations. arXiv preprint arXiv:2102.02622.
Patel, D., & Ahmad, A. (2023). The Inference Cost Of Search Disruption–Large Language Model Cost Analysis. Verfügbar unter https://www. semianalysis. com/p/theinference-cost-of-search-disruption.

# References


Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204. PMLR, 2015.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4): 115–133, 1943.

Murray Rosenblatt. Some purely deterministic processes. *Journal of Mathematics and Mechanics*, pages 801–810, 1957.