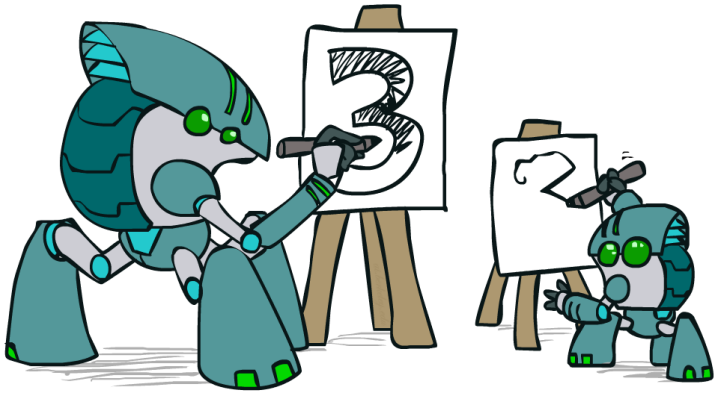
Introduction to Artificial Intelligence

Lecture 7: Machine learning and neural networks

Prof. Gilles Louppe
g.louppe@uliege.be

Today



Make our agents able to learn from experience.

- Machine learning
- Deep learning
  - Multi-layer perceptron
  - Convolutional neural networks
  - Recurrent neural networks

—

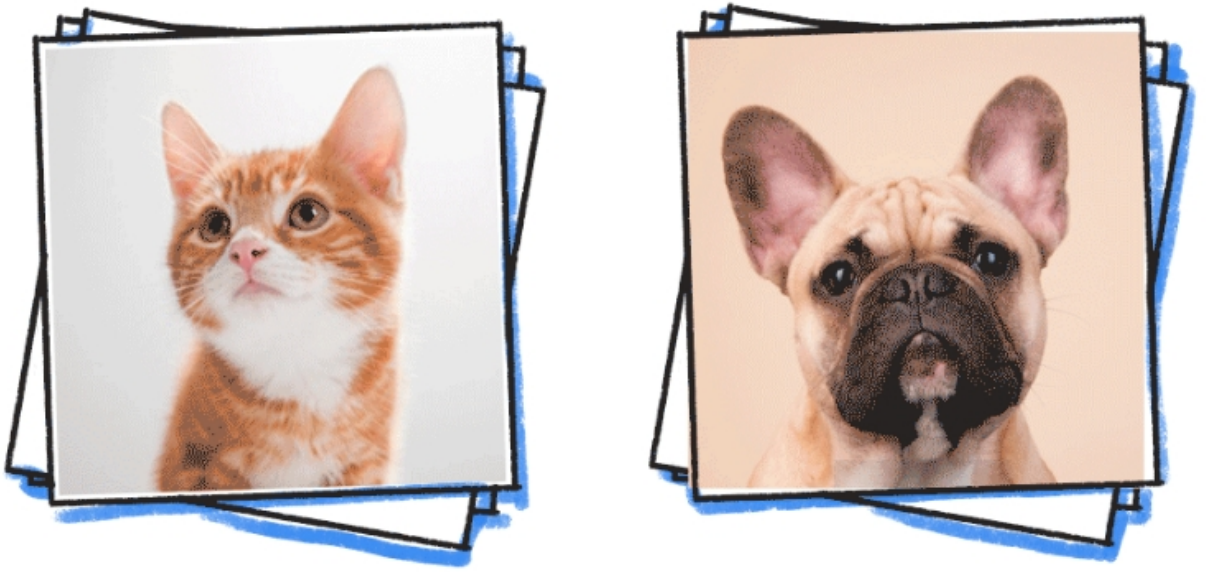
Image credits: CS188, UC Berkeley.

Learning agents

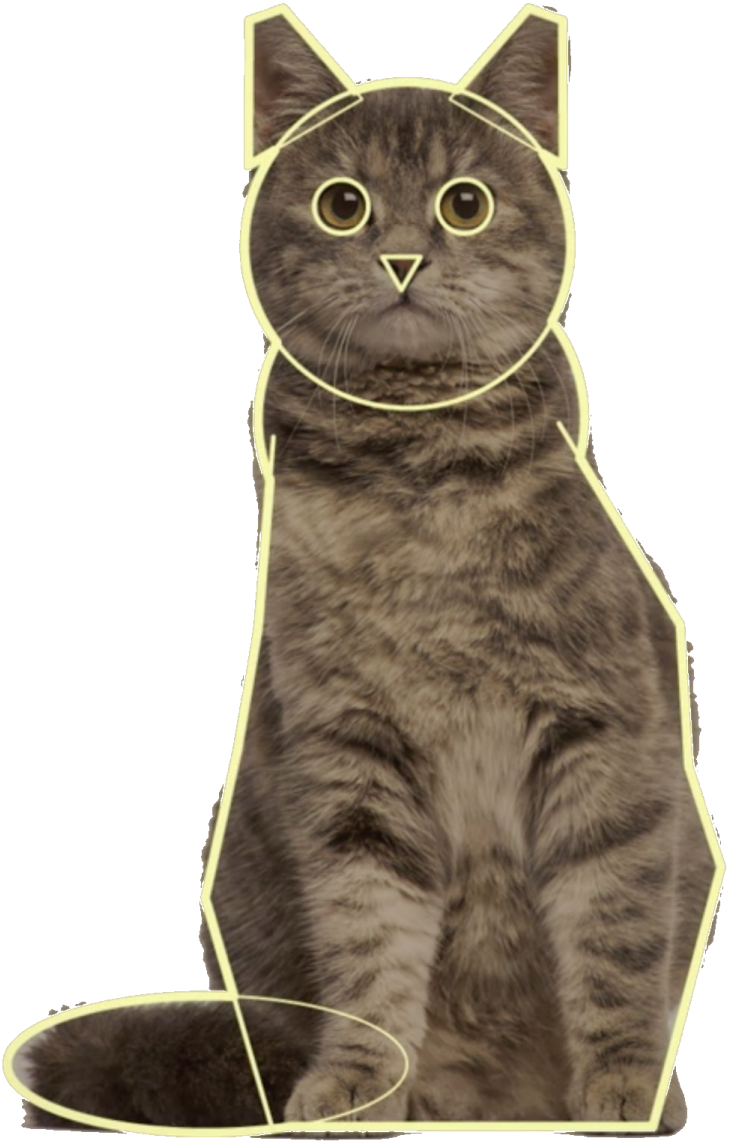
What if the environment is unknown?

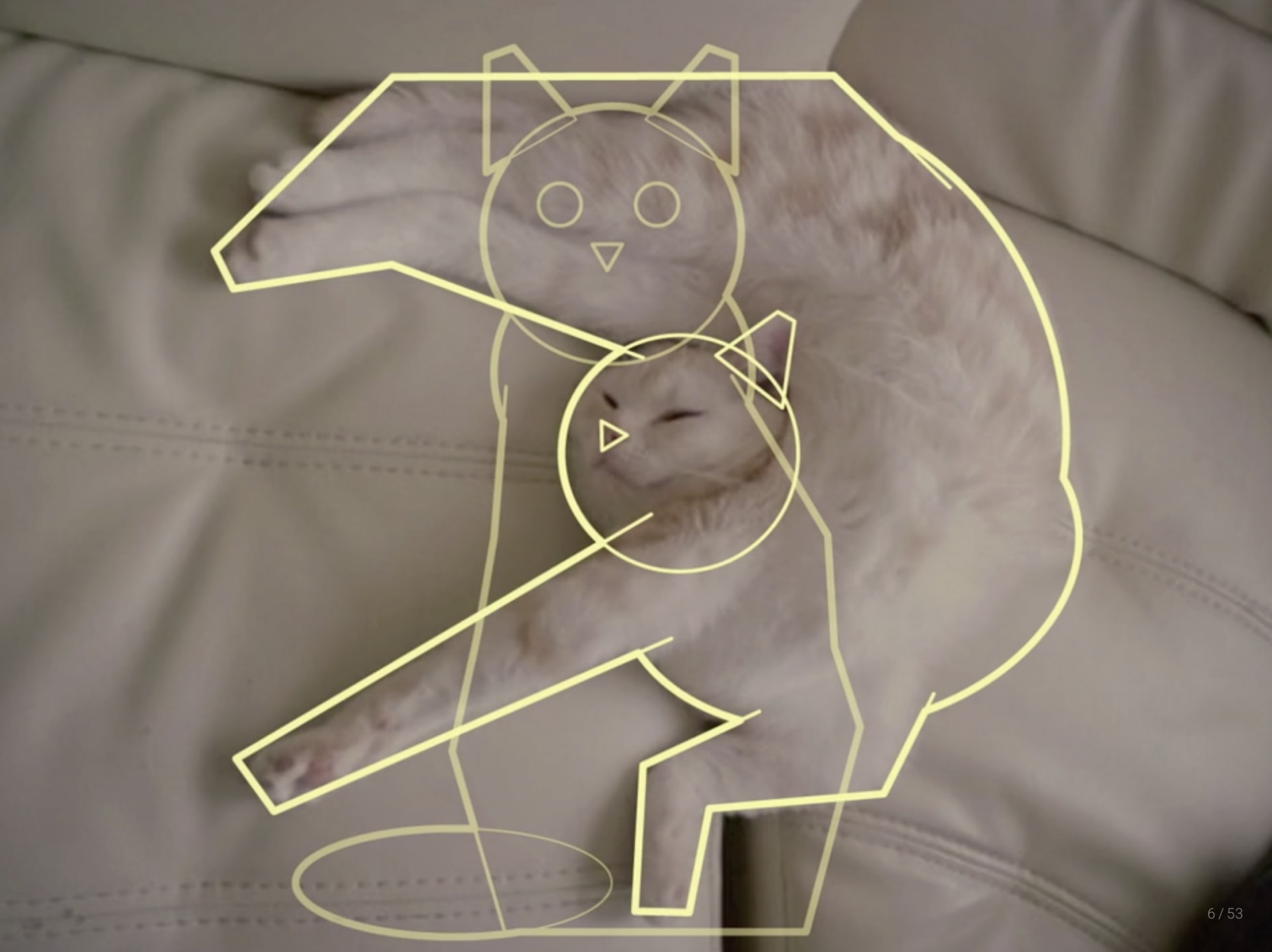
- Learning provides an automated way to modify the agent's internal decision mechanisms to improve its own performance.
- It exposes the agent to reality rather than trying to hardcode reality into the agent's program.

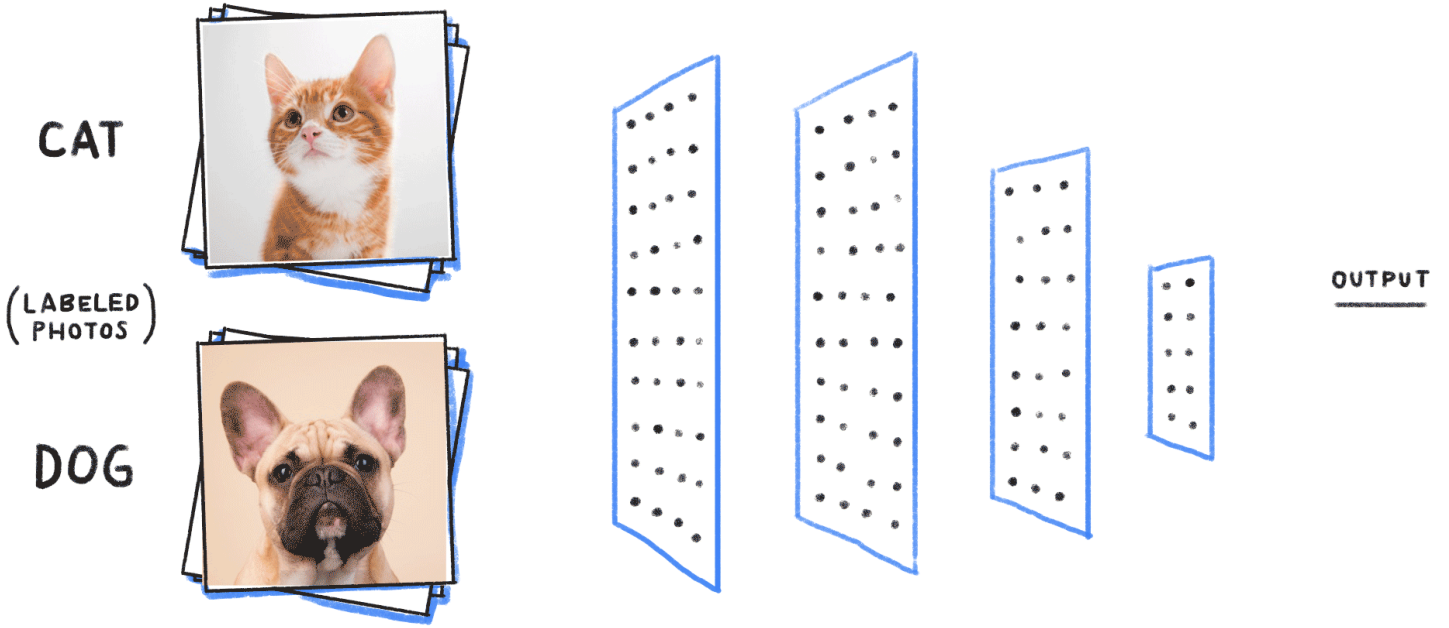
Machine learning



How would you write a computer program that recognizes cats from dogs?







The deep learning approach.

Problem statement

Let $\mathbf{d} \sim p(\mathbf{x}, y)$ be a dataset of N example input-output pairs

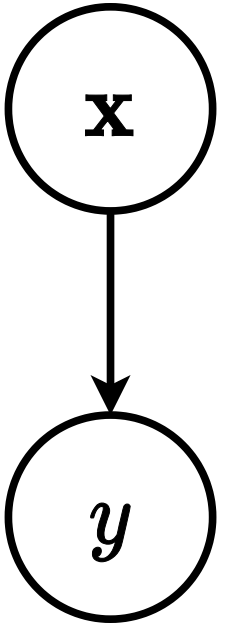
$$\mathbf{d} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N)\},$$

where $\mathbf{x}_i$ are the input values and y_i are the corresponding output values.

From this data, we want to identify a probabilistic model

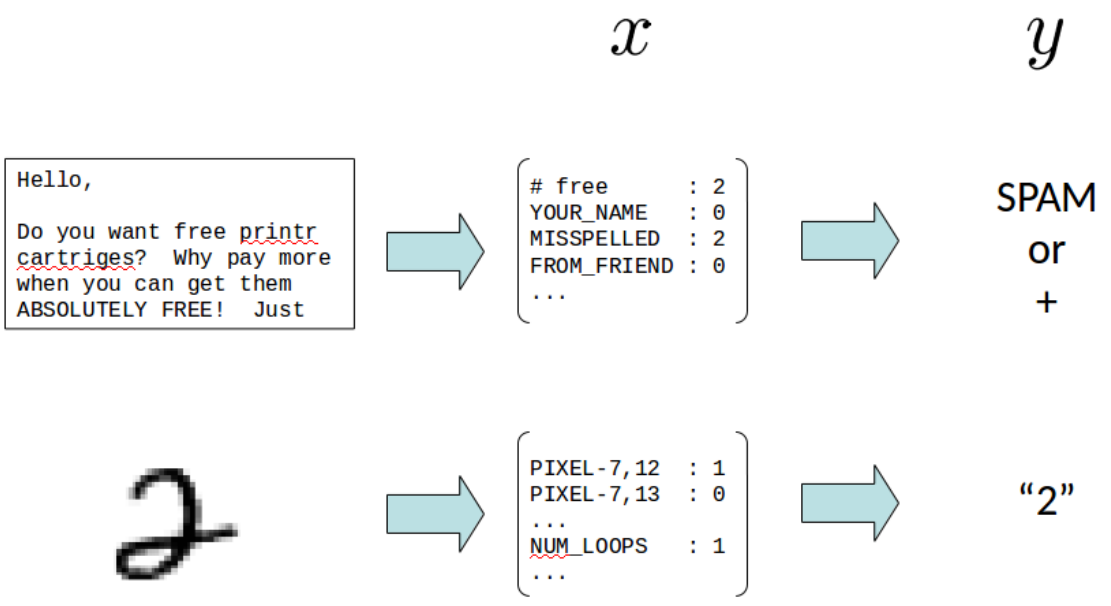
$$p(y|\mathbf{x})$$

that best explains the data.



Feature vectors

- Inputs $\mathbf{x} \in \mathbb{R}^d$ are described as real-valued vectors of d attributes or features values.
- If the data is not originally expressed as real-valued vectors, then it needs to be prepared and transformed to this format.

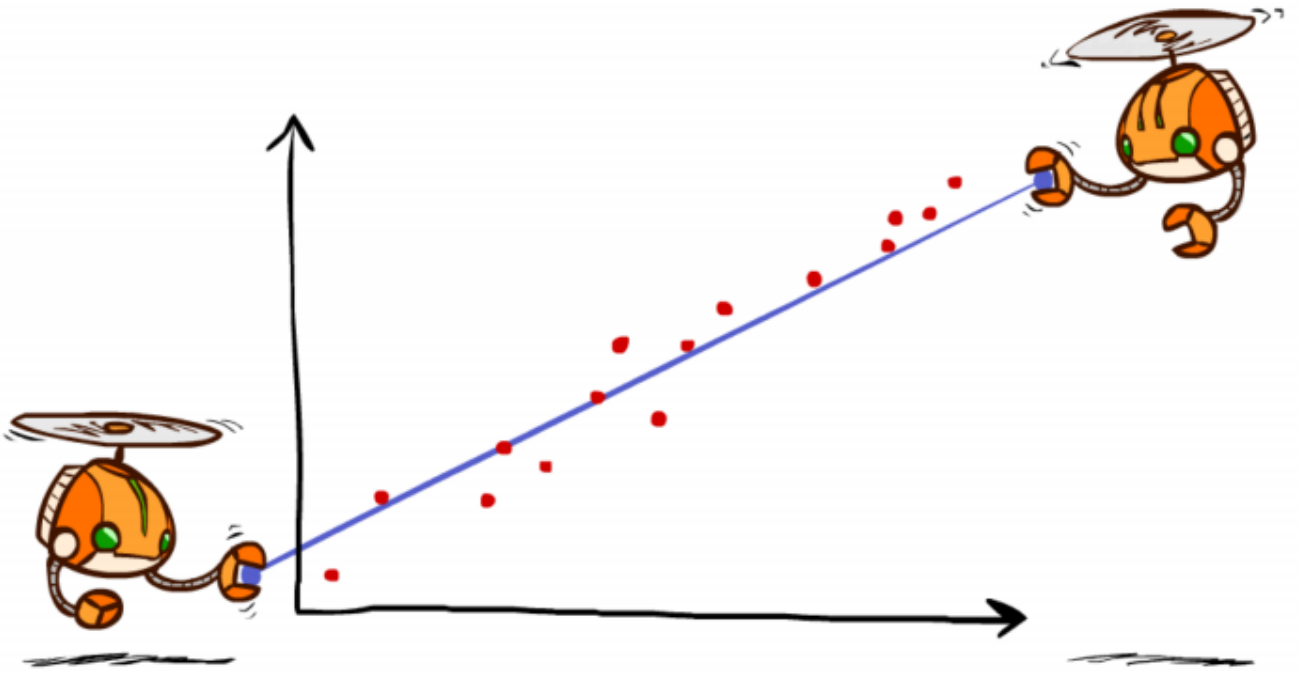


—

Image credits: CS188, UC Berkeley.

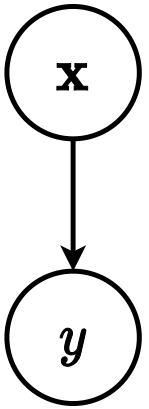
Linear regression

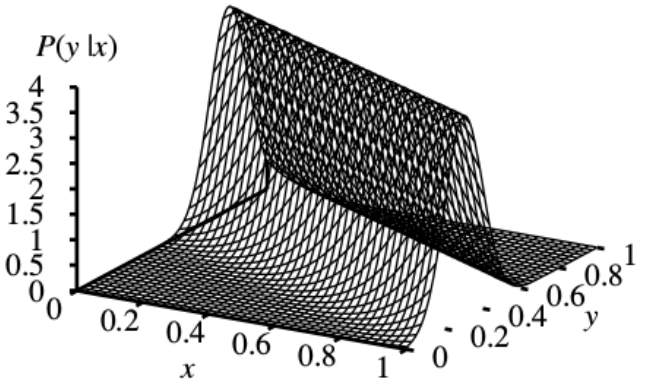
Let us first assume that $y \in \mathbb{R}$.



—

Image credits: CS188, UC Berkeley.





Linear regression considers a parameterized linear Gaussian model for its parametric model of $p(y|\mathbf{x})$, that is

$$p(y|\mathbf{x}) = \mathcal{N}(y|\mathbf{w}^T\mathbf{x} + b, \sigma^2),$$

where $\mathbf{w}$ and b are parameters to determine.

To learn the conditional distribution $p(y|\mathbf{x})$, we maximize

$$p(y|\mathbf{x}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\frac{(y-(\mathbf{w}^T\mathbf{x}+b))^2}{\sigma^2}\right)$$

w.r.t. $\mathbf{w}$ and b over the data $\mathbf{d} = \{(\mathbf{x}_j, y_j)\}$.

To learn the conditional distribution $p(y|\mathbf{x})$, we maximize

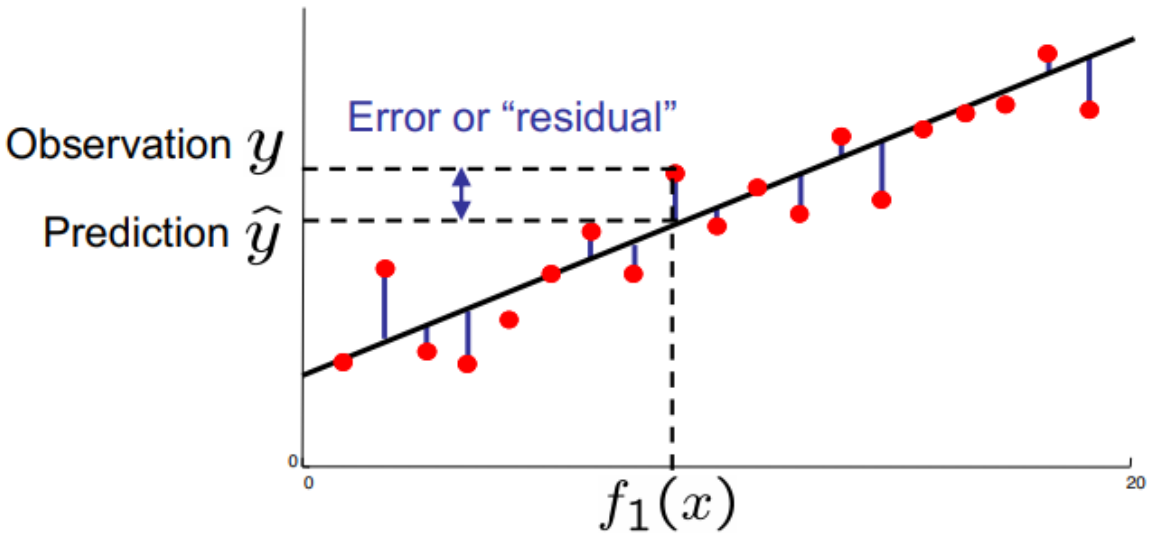
$$p(y|\mathbf{x}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\frac{(y-(\mathbf{w}^T\mathbf{x}+b))^2}{\sigma^2}\right)$$

w.r.t. $\mathbf{w}$ and b over the data $\mathbf{d} = \{(\mathbf{x}_j, y_j)\}$.

By constraining the derivatives of the log-likelihood to 0, we arrive to the problem of minimizing

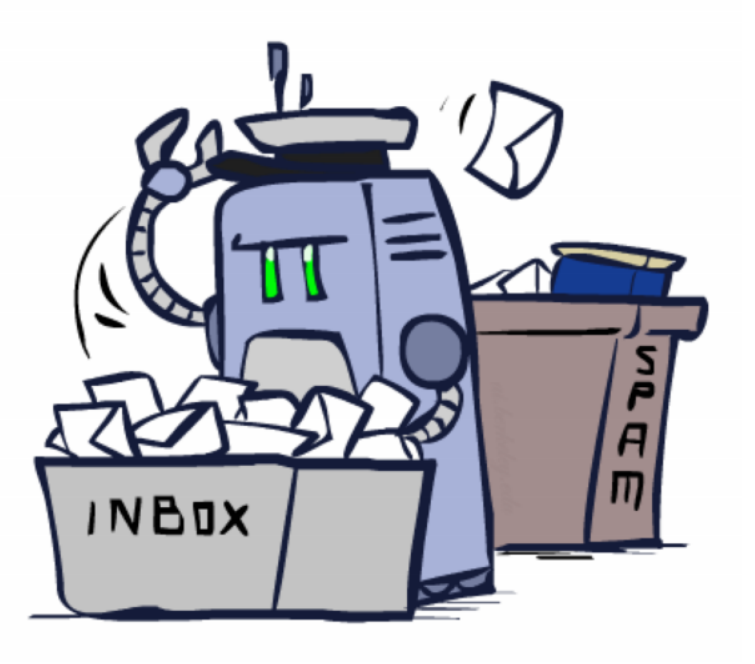
$$\sum_{j=1}^{N}(y_j - (\mathbf{w}^T\mathbf{x}_j + b))^2.$$

Therefore, minimizing the sum of squared errors corresponds to the MLE solution for a linear fit, assuming Gaussian noise of fixed variance.


Observation y
Prediction $\widehat{y}$
Error or “residual”
0
0
20
$f_1(x)$

Logistic regression

Let us now assume $y \in \{0, 1\}$.



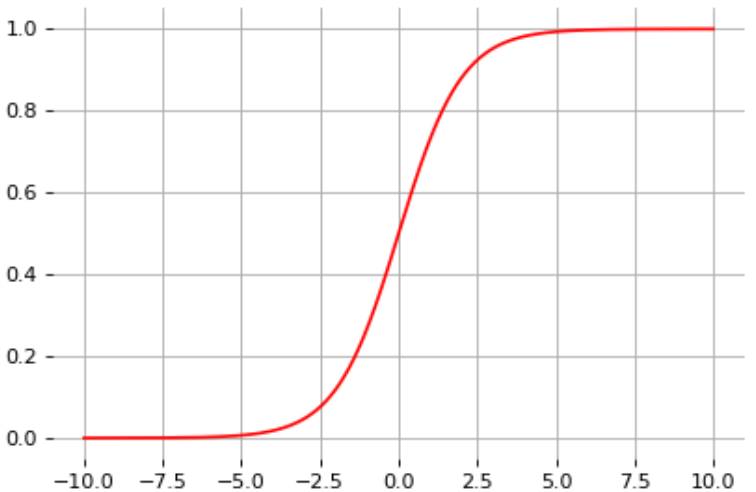
—

Image credits: CS188, UC Berkeley.

Logistic regression models the conditional as

$$P(Y = 1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b),$$

where the sigmoid activation function $\sigma(x) = \frac{1}{1+\exp(-x)}$ looks like a soft heavyside:



Following the principle of maximum likelihood estimation, we have

$$
\begin{aligned}
&\arg\max_{\mathbf{w},b} P(\mathbf{d}|\mathbf{w},b) \\
&= \arg\max_{\mathbf{w},b} \prod_{\mathbf{x}_i,y_i\in\mathbf{d}} P(Y=y_i|\mathbf{x}_i,\mathbf{w},b) \\
&= \arg\max_{\mathbf{w},b} \prod_{\mathbf{x}_i,y_i\in\mathbf{d}} \sigma(\mathbf{w}^T\mathbf{x}_i+b)^{y_i}(1-\sigma(\mathbf{w}^T\mathbf{x}_i+b))^{1-y_i} \\
&= \arg\min_{\mathbf{w},b} \underbrace{\sum_{\mathbf{x}_i,y_i\in\mathbf{d}} -y_i\log\sigma(\mathbf{w}^T\mathbf{x}_i+b)-(1-y_i)\log(1-\sigma(\mathbf{w}^T\mathbf{x}_i+b))}_{\mathcal{L}(\mathbf{w},b)=\sum_i \ell(y_i,\hat{y}(\mathbf{x}_i;\mathbf{w},b))}
\end{aligned}
$$

This loss is an instance of the cross-entropy

$$H(p,q)=\mathbb{E}_p[-\log q]$$

for $p=Y|\mathbf{x}_i$ and $q=\hat{Y}|\mathbf{x}_i$.

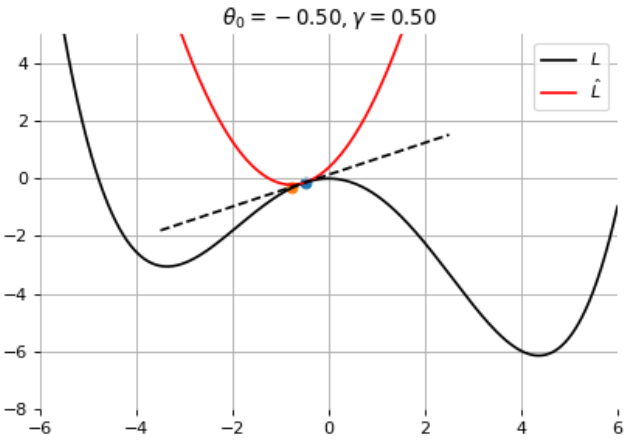
Gradient descent

Let $\mathcal{L}(\theta)$ denote a loss function defined over model parameters θ (e.g., $\mathbf{w}$ and b).

To minimize $\mathcal{L}(\theta)$, gradient descent uses local linear information to iteratively move towards a (local) minimum.

For $\theta_0 \in \mathbb{R}^d$, a first-order approximation around θ_0 can be defined as

$$\hat{\mathcal{L}}(\epsilon; \theta_0) = \mathcal{L}(\theta_0) + \epsilon^T \nabla_\theta \mathcal{L}(\theta_0) + \frac{1}{2\gamma} ||\epsilon||^2.$$



A minimizer of the approximation $\hat{\mathcal{L}}(\epsilon; \theta_0)$ is given for

$$\begin{aligned} \nabla_\epsilon \hat{\mathcal{L}}(\epsilon; \theta_0) &= 0 \\ &= \nabla_\theta \mathcal{L}(\theta_0) + \frac{1}{\gamma}\epsilon, \end{aligned}$$

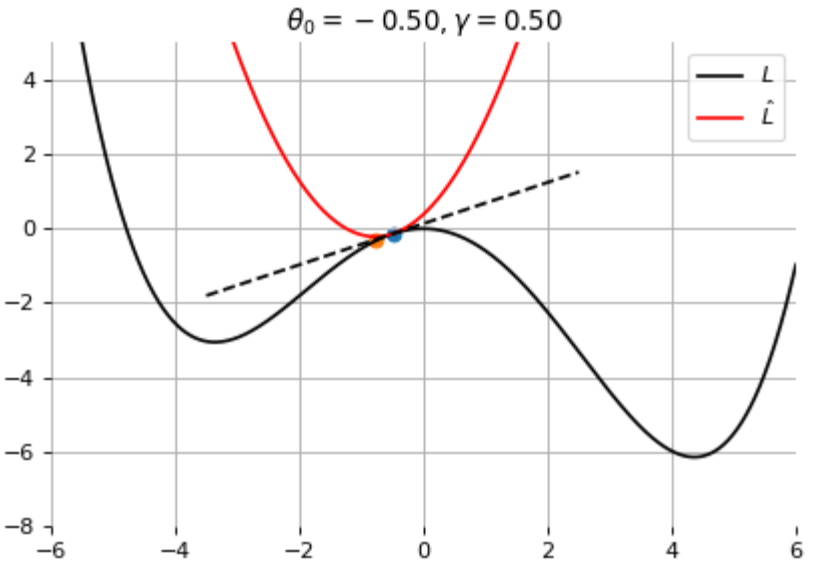
which results in the best improvement for the step $\epsilon = -\gamma \nabla_\theta \mathcal{L}(\theta_0)$.

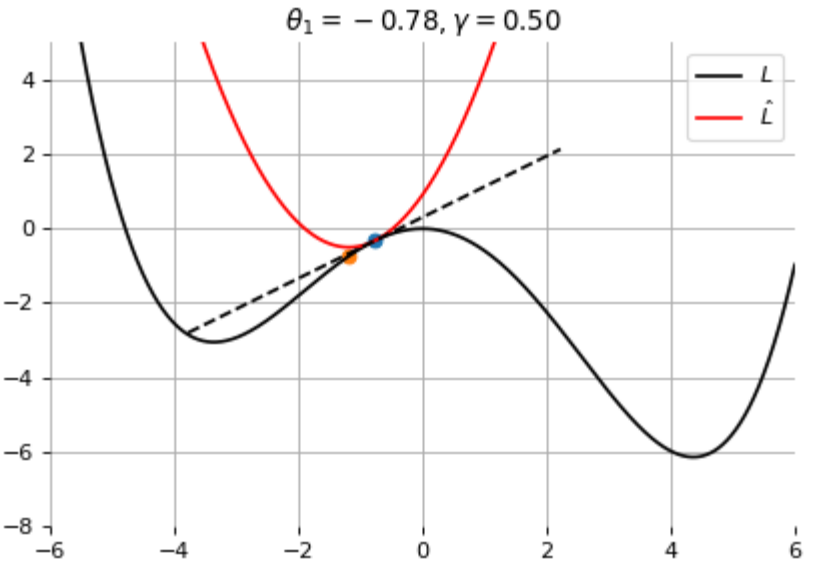
Therefore, model parameters can be updated iteratively using the update rule

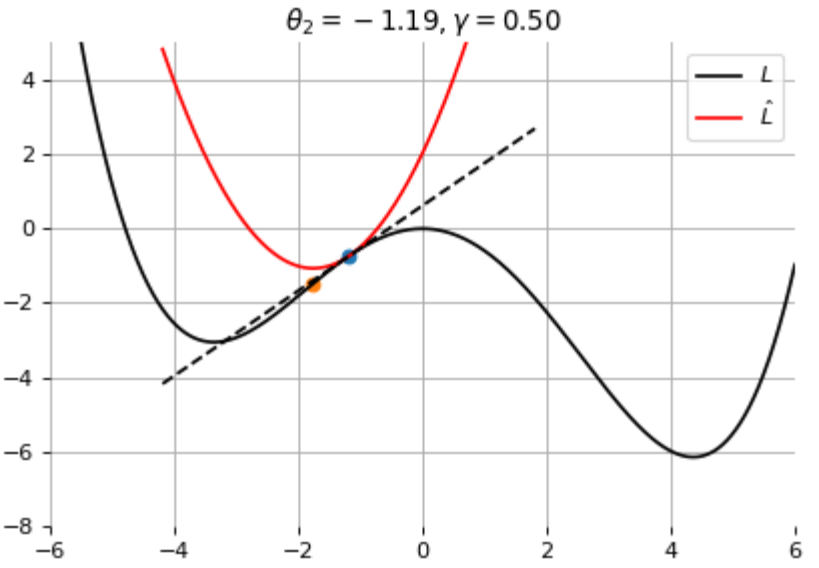
$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t),$$

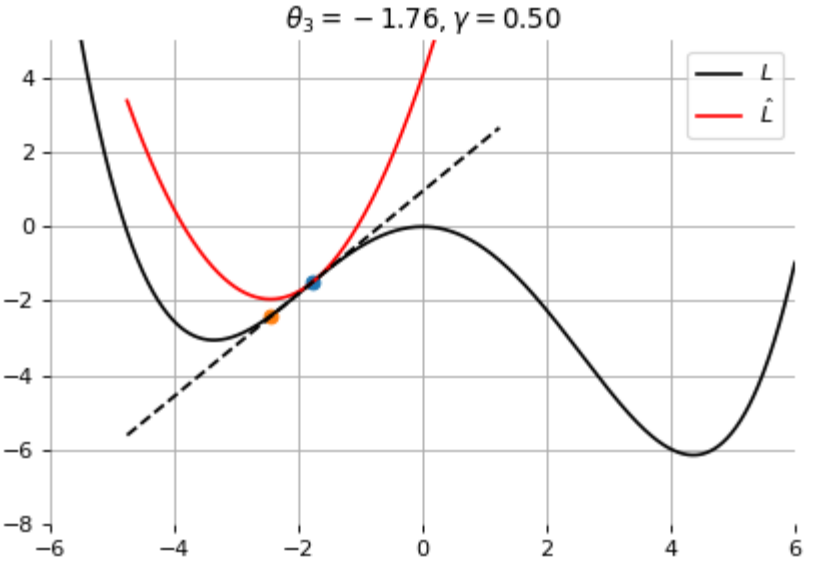
where

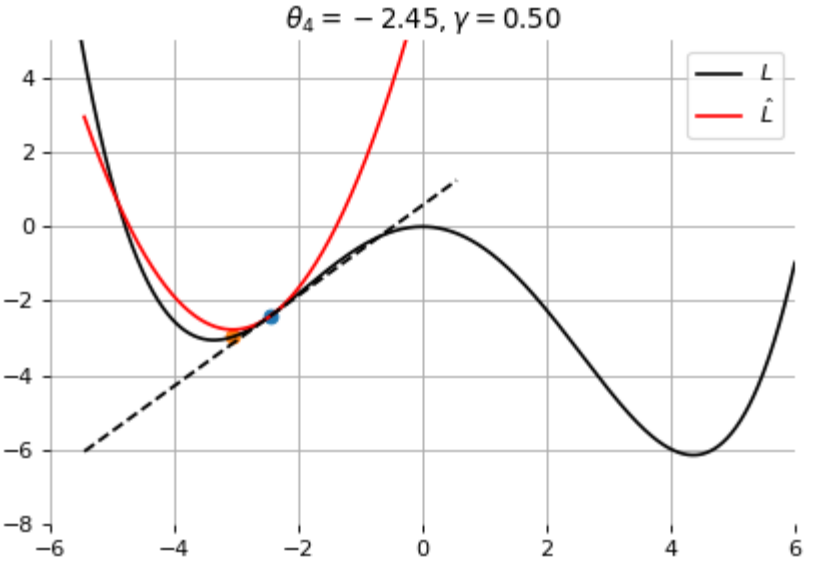
- θ_0 are the initial parameters of the model,
- γ is the learning rate.

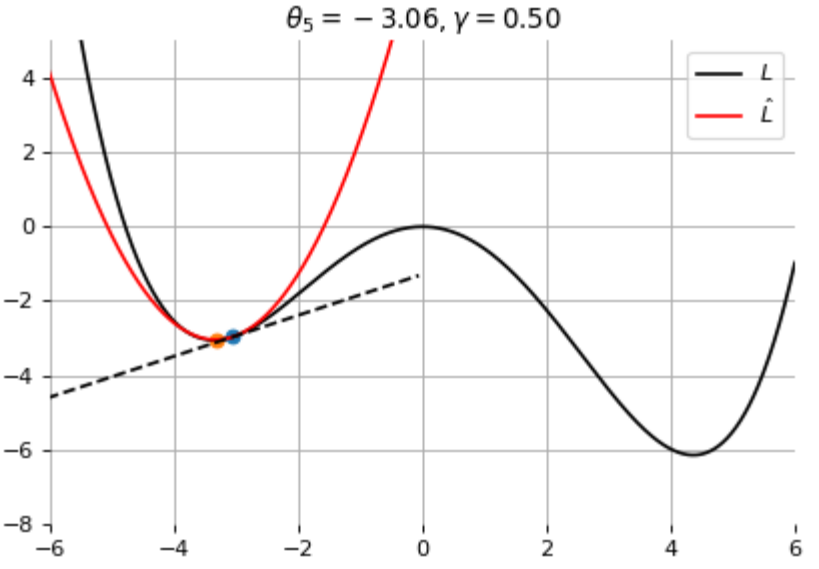

$\theta_0 = -0.50, \gamma = 0.50$
L
$\hat{L}$
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6

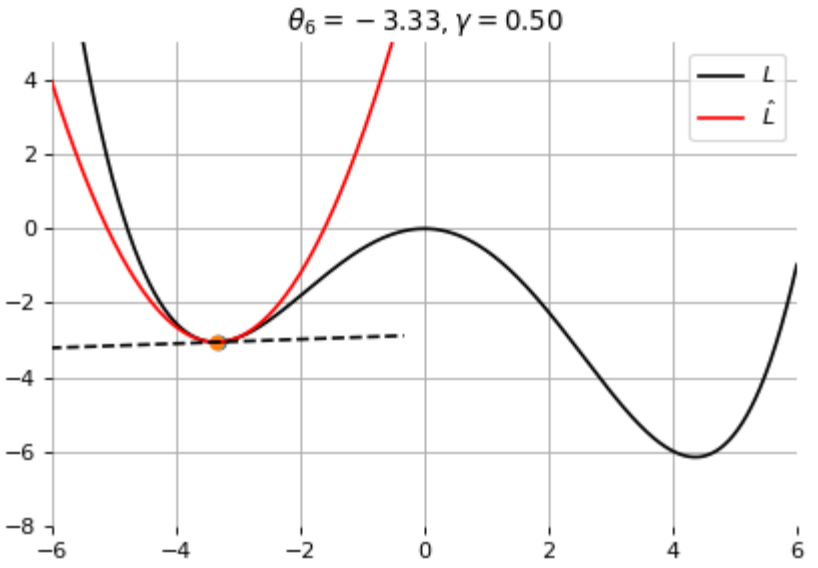

$\theta_1 = -0.78, \gamma = 0.50$
L
$\hat{L}$
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6

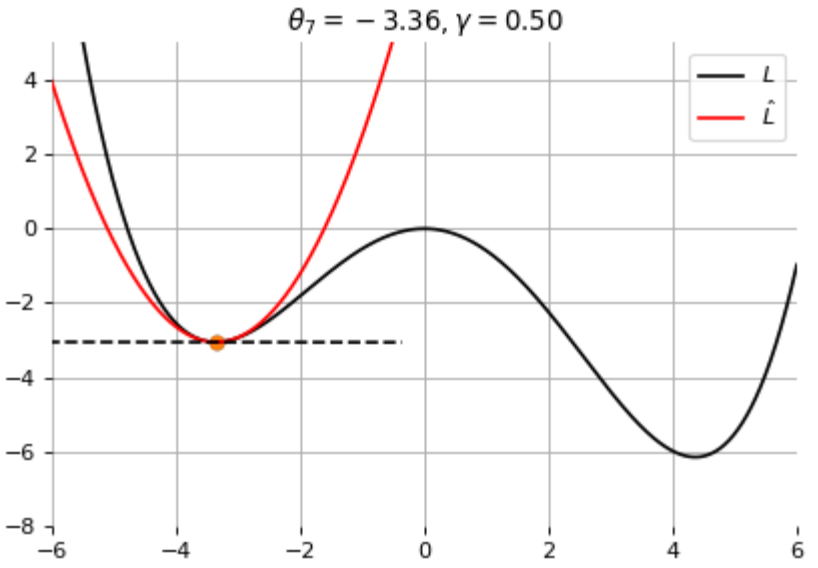

$\theta_2 = -1.19, \gamma = 0.50$
L
$\hat{L}$
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6


$\theta_3 = -1.76, \gamma = 0.50$
L
$\hat{L}$
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6


$\theta_4 = -2.45, \gamma = 0.50$
L
$\hat{L}$
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6


$\theta_5 = -3.06, \gamma = 0.50$
L
$\hat{L}$
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6


$\theta_6 = -3.33, \gamma = 0.50$
L
$\hat{L}$
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6

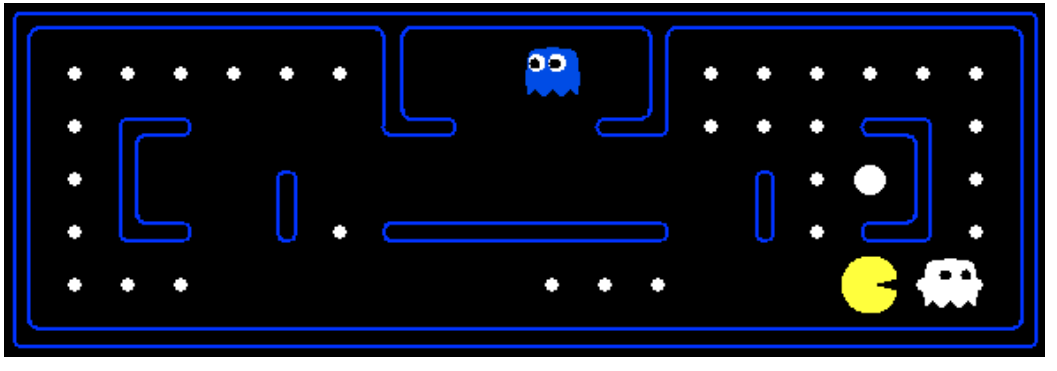

$\theta_7 = -3.36, \gamma = 0.50$
L
$\hat{L}$
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6

(Step-by-step code example)

Apprenticeship

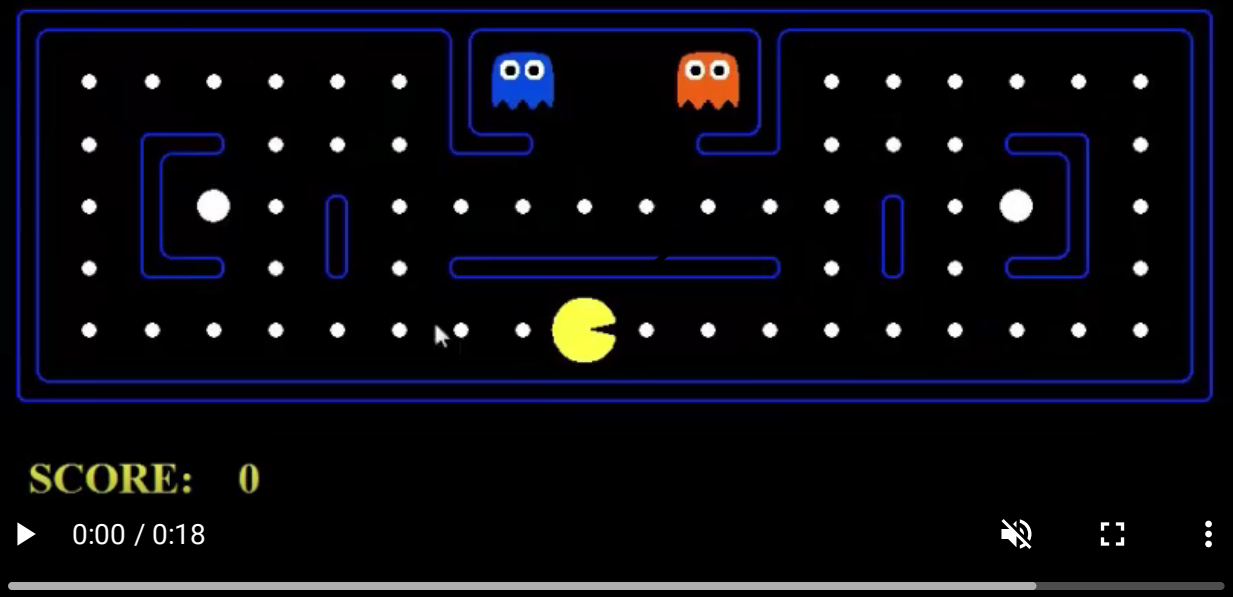
Can we learn to play Pacman only from observations?

- Feature vectors $\mathbf{x} = g(s)$ are extracted from the game states s. Output values y corresponds to actions a .
- State-action pairs $(\mathbf{x}, y)$ are collected by observing an expert playing.
- We want to learn the actions that the expert would take in a given situation. That is, learn the mapping $f : \mathbb{R}^d \to \mathcal{A}$.
- This is a multiclass classification problem that can be solved by combining binary classifers.



—

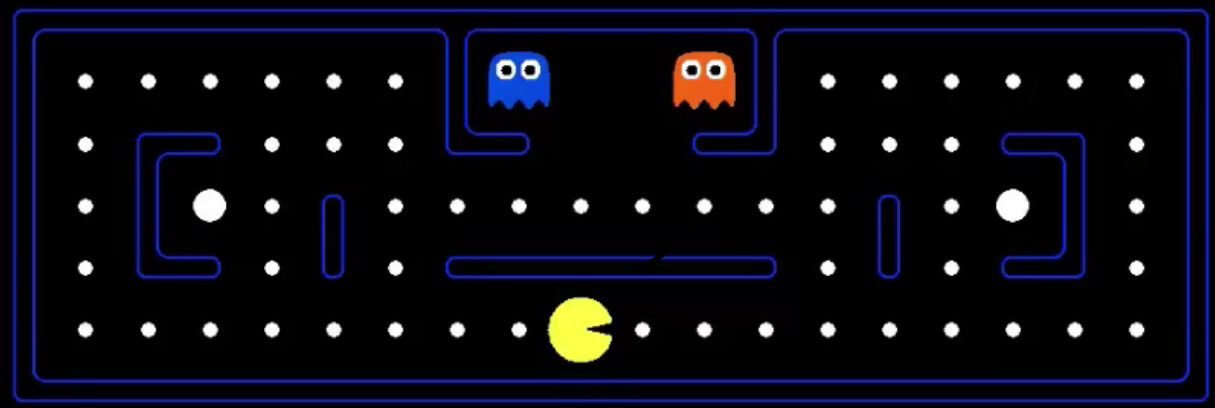
Image credits: CS188, UC Berkeley.



The agent observes a very good Minimax-based agent for two games and updates its weight vectors as data are collected.

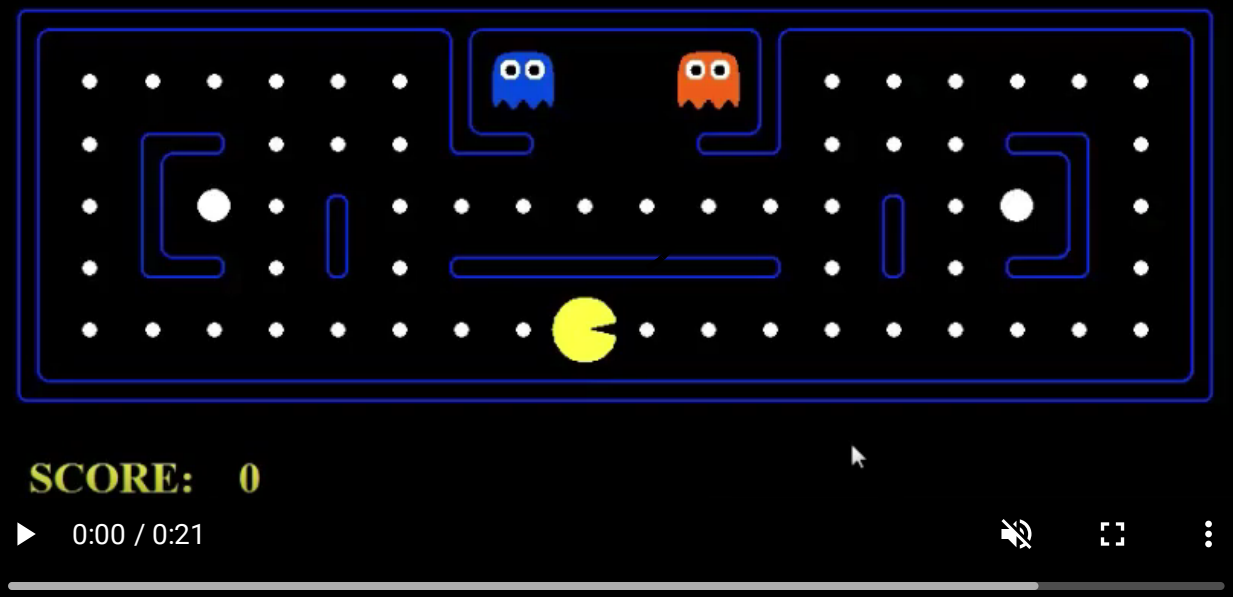
—

Image credits: CS188, UC Berkeley.



—

Image credits: CS188, UC Berkeley.



After two training episodes, the ML-based agents plays.
No more Minimax!

—
Image credits: CS188, UC Berkeley.

Deep Learning

(a short introduction)

Multi-layer perceptron

So far we considered the logistic unit $h = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right)$, where $h \in \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$.

These units can be composed in parallel to form a layer with q outputs:

$$\mathbf{h} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b})$$

where $\mathbf{h} \in \mathbb{R}^q$, $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{W} \in \mathbb{R}^{d\times q}$, $b \in \mathbb{R}^d$ and where $\sigma(\cdot)$ is upgraded to the element-wise sigmoid function.

Similarly, layers can be composed in series, such that:

$$\begin{aligned}
\mathbf{h}_0 &= \mathbf{x} \\
\mathbf{h}_1 &= \sigma(\mathbf{W}_1^T \mathbf{h}_0 + \mathbf{b}_1) \\
&\dots \\
\mathbf{h}_L &= \sigma(\mathbf{W}_L^T \mathbf{h}_{L-1} + \mathbf{b}_L) \\
f(\mathbf{x}; \theta) &= \mathbf{h}_L
\end{aligned}$$

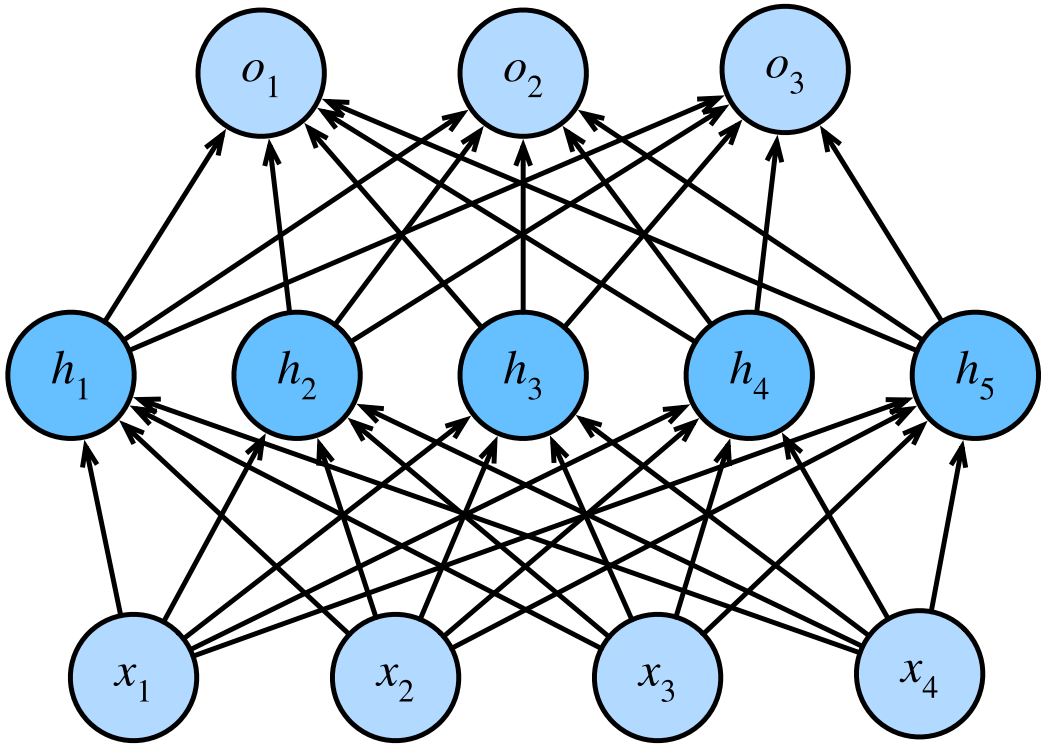
where θ denotes the model parameters $\{\mathbf{W}_k, \mathbf{b}_k, ... | k = 1, ..., L\}$ and can be determined through gradient descent.

This model is the multi-layer perceptron, also known as the fully connected feedforward network.

Output layer

Hidden layer

Input layer



—

Credits: Dive into Deep Learning, 2023.

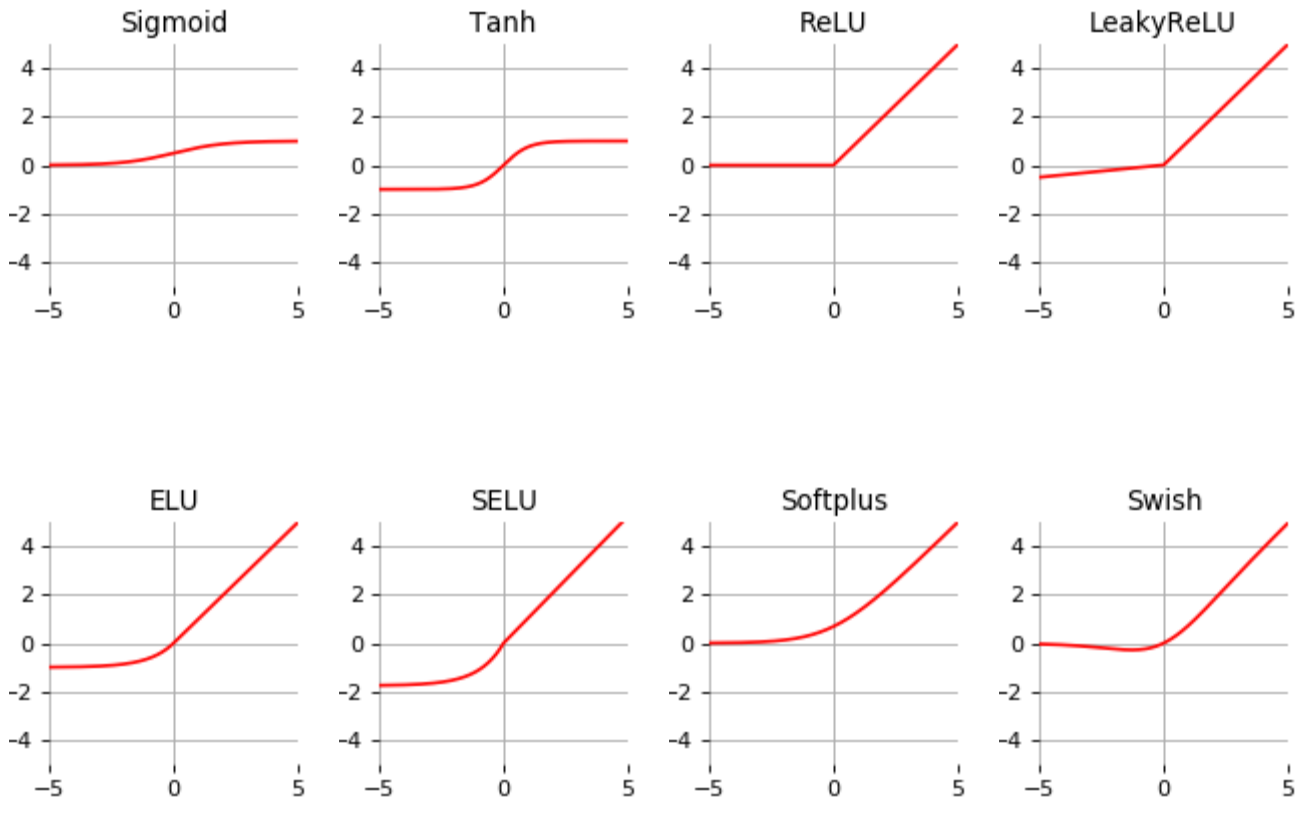
- For binary classification, the width q of the last layer L is set to 1, which results in a single output $h_L \in [0, 1]$ that models the probability $p(y = 1|\mathbf{x})$.
- For multi-class classification, the sigmoid activation σ in the last layer can be generalized to produce a vector $\mathbf{h}_L \in \triangle^C$ of probability estimates $p(y = i|\mathbf{x})$.

  This activation is the $\mathbf{Softmax}$ function, where its i-th output is defined as

$$\mathbf{Softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)},$$

  for $i = 1, ..., C$.

Activation functions are key to the expressiveness of neural networks. They are usually chosen to be non-linear, such as the sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$ or the ReLU function $\mathrm{ReLU}(x) = \max(0, x)$.



(Step-by-step code example)

Neural networks for images

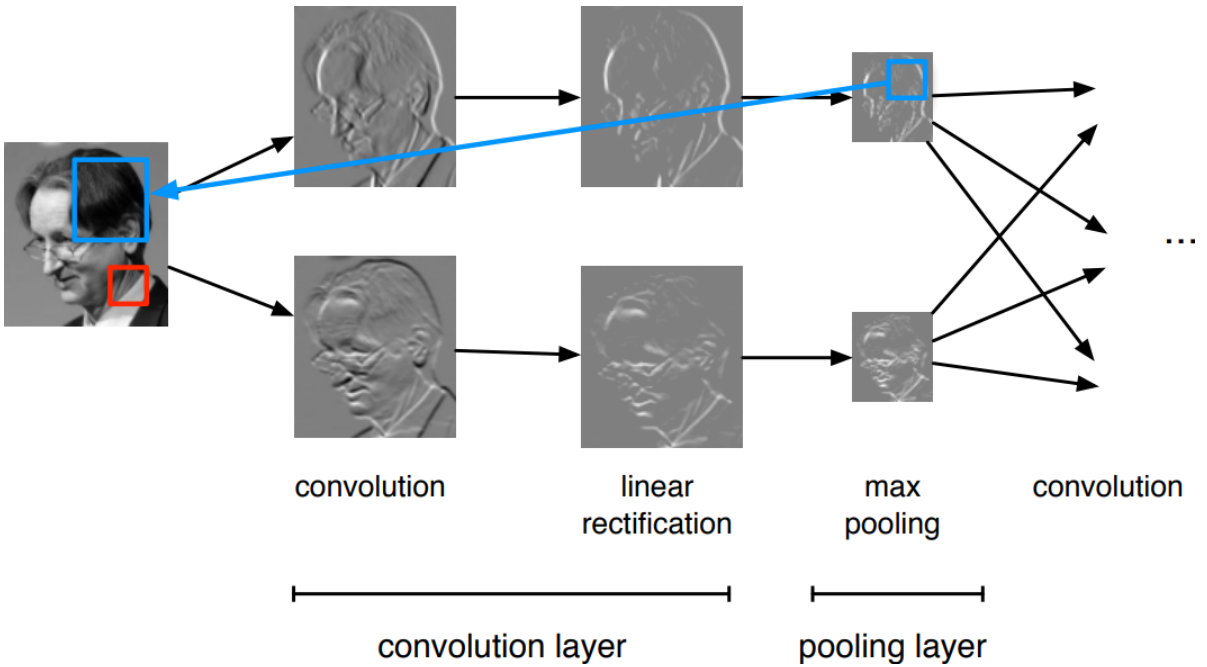
The MLP architecture is appropriate for tabular data, but not for images. We want to design a neural architecture such that:

- in the earliest layers, the network respond similarly to similar patches of the image, regardless of their location;
- the earliest layers focus on local regions of the image, without regard for the contents of the image in distant regions;
- in the later layers, the network combines the information from the earlier layers to focus on larger and larger regions of the image, eventually combining all the information from the image to classify the image into a category.

Convolutional networks

Convolutional neural networks extend fully connected architectures with

- convolutional layers acting as local feature detectors;
- pooling layers acting as spatial down-samplers.

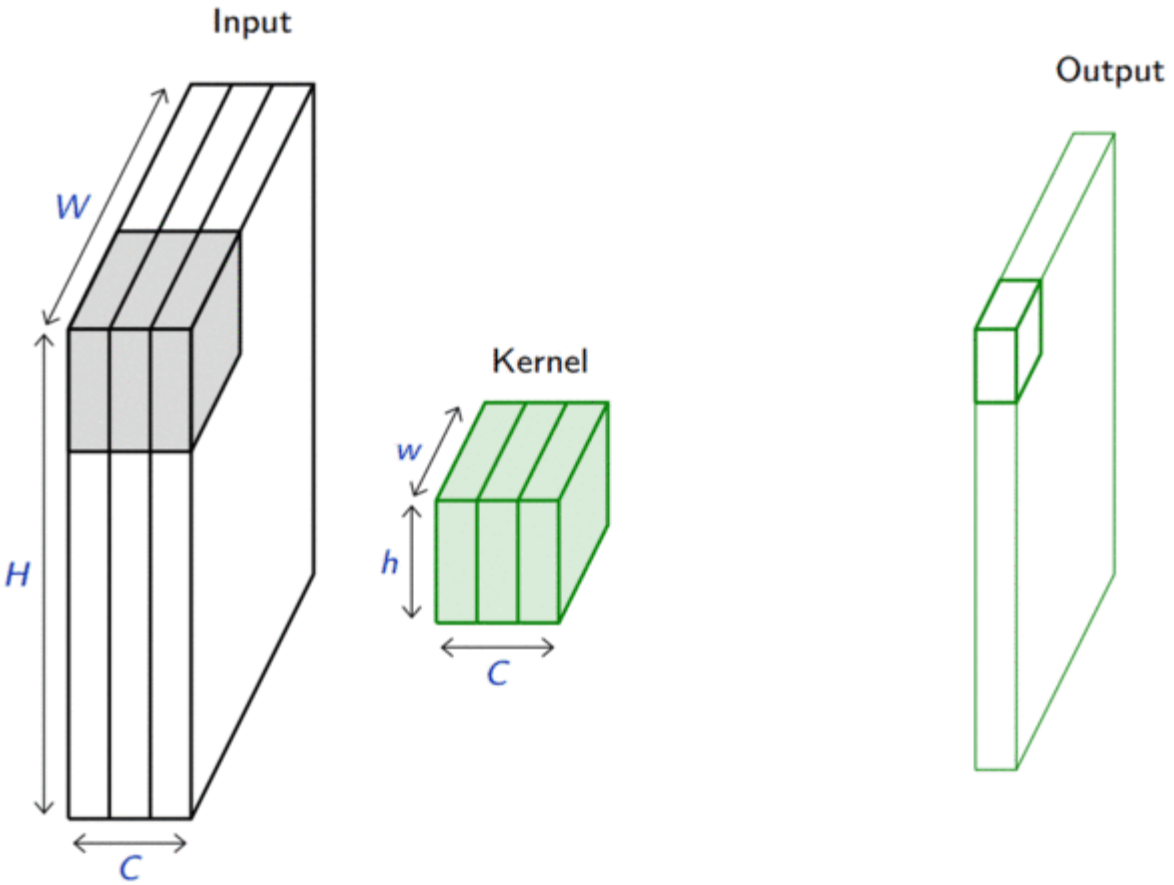


Convolutional layers

A convolutional layer is defined by a set of K kernels $\mathbf{u}$ of size $c \times h \times w$, where h and w are the height and width of the kernel, and c is the number of channels of the input.

Assuming as input a 3D tensor $\mathbf{x} \in \mathbb{R}^{C \times H \times W}$, the output of the convolutional layer is a set of K feature maps of size $H' \times W'$, where $H' = H - h + 1$ and $W' = W - w + 1$. Each feature map $\mathbf{o}$ is the result of convolving the input with a kernel, that is

$$\mathbf{o}_{j,i} = \sum_{c=0}^{C-1} (\mathbf{x}_c \circledast \mathbf{u}_c)[j,i] = \sum_{c=0}^{C-1} \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} \mathbf{x}_{c,n+j,m+i} \mathbf{u}_{c,n,m}$$


Input
W
H
C
Kernel
w
h
C
Output

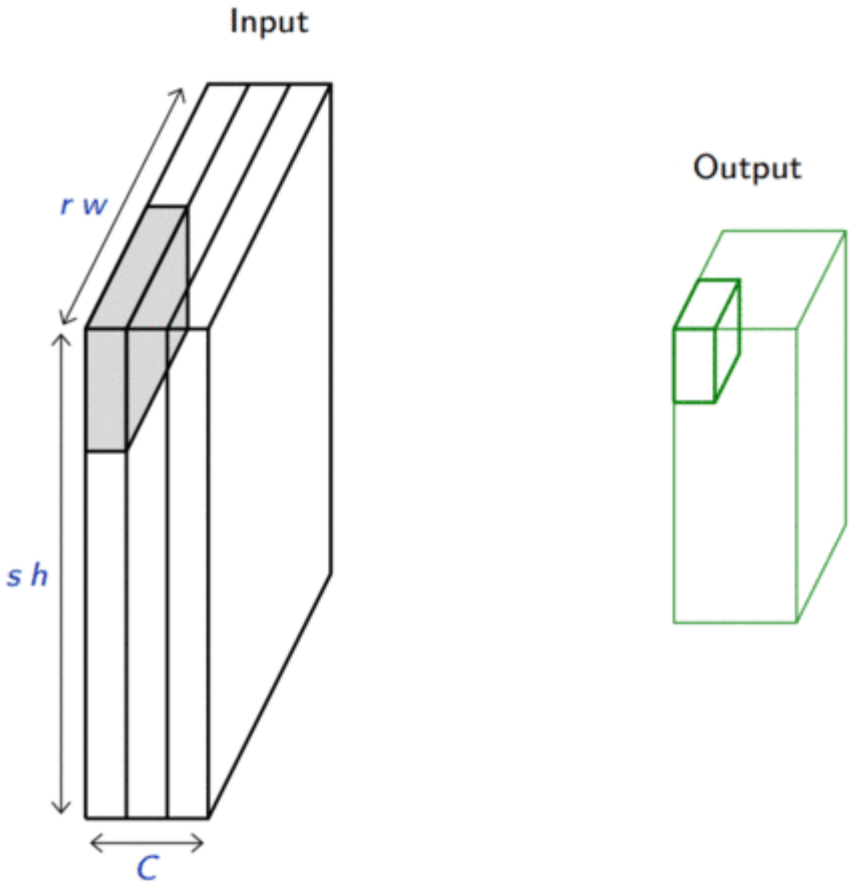
—

Pooling layers

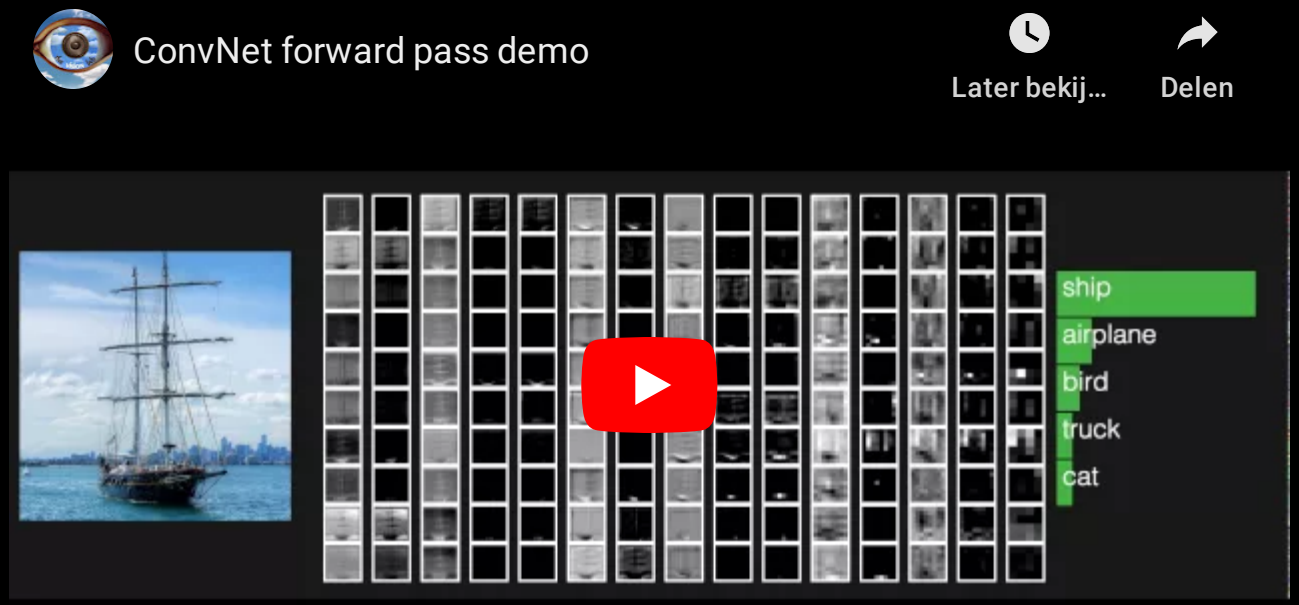
Pooling layers are used to progressively reduce the spatial size of the representation, hence capturing longer-range dependencies between features.

Considering a pooling area of size $h \times w$ and a 3D input tensor $\mathbf{x} \in \mathbb{R}^{C \times (rh) \times (sw)}$, max-pooling produces a tensor $\mathbf{o} \in \mathbb{R}^{C \times r \times s}$ such that

$$\mathbf{o}_{c,j,i} = \max_{n<h, m<w} \mathbf{x}_{c,rj+n,si+m}.$$


Input
r w
s h
C
Output

—



A convolutional network combines convolutional, pooling and fully connected layers.

(Step-by-step code example)

Recurrent networks

When the input is a sequence $\mathbf{x}_{1:T}$, the feedforward network can be made recurrent by computing a sequence $\mathbf{h}_{1:T}$ of hidden states, where $\mathbf{h}_t$ is a function of both $\mathbf{x}_t$ and the previous hidden states in the sequence.

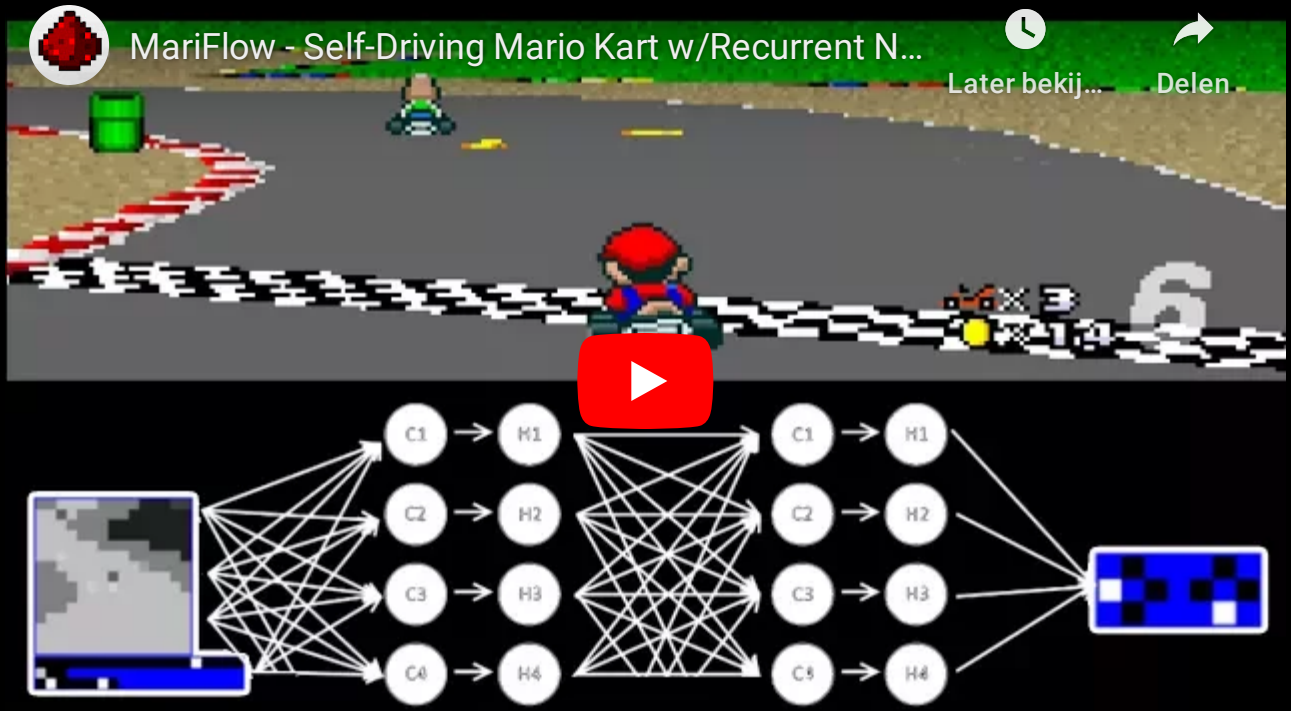
For example,

$$\mathbf{h}_t = \sigma(\mathbf{W}_{xh}^T \mathbf{x} + \mathbf{W}_{hh}^T \mathbf{h}_{t-1} + \mathbf{b}),$$

where $\mathbf{h}_{t-1}$ is the previous hidden state in the sequence.

Notice how this is similar to filtering and dynamic decision networks:

- $\mathbf{h}_t$ can be viewed as some current belief state;
- $\mathbf{x}_{1:T}$ is a sequence of observations;
- $\mathbf{h}_{t+1}$ is computed from the current belief state $\mathbf{h}_t$ and the latest evidence $\mathbf{x}_t$ through some fixed computation (in this case a neural network, instead of being inferred from the assumed dynamics).
- $\mathbf{h}_t$ can also be used to decide on some action, through another network f such that $a_t = f(\mathbf{h}_t; \theta)$.



A recurrent network playing Mario Kart.

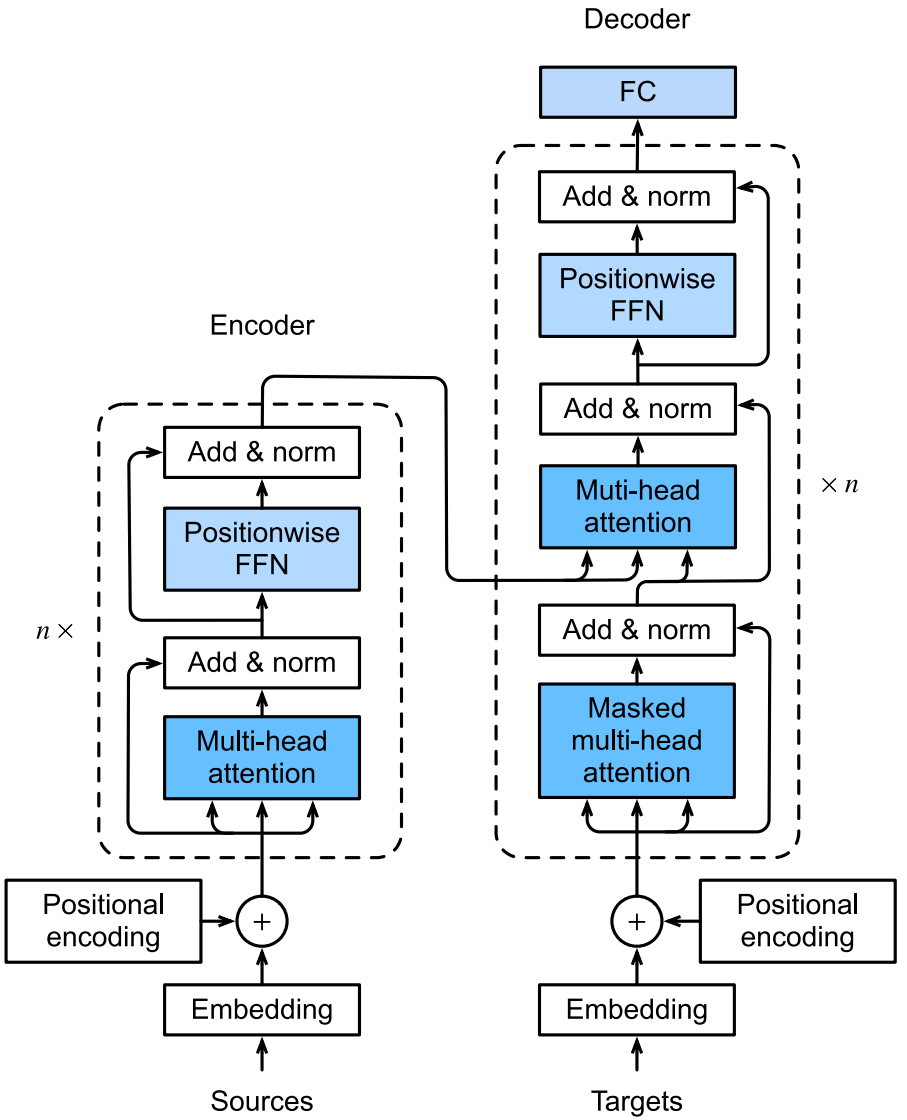
Transformers

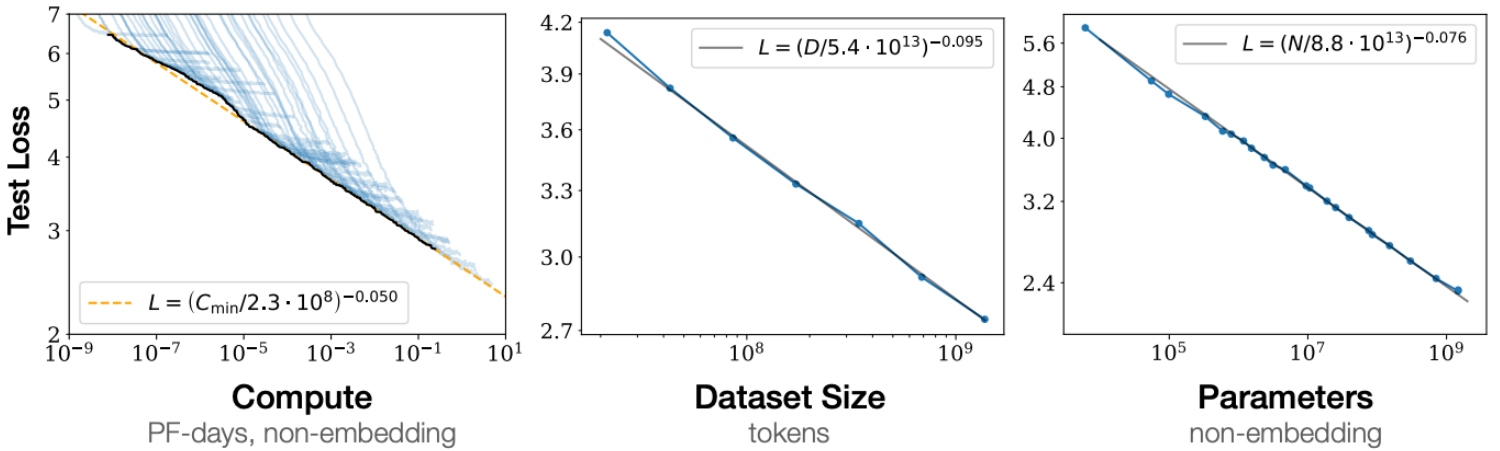
Transformers are a type of neural network architecture that are based on self-attention mechanisms instead of fully connected or convolutional layers.

For language, transformers are trained as classifiers

$$p(w_t | w_{1:t-1})$$

where w_t is the next word in the sequence and $w_{1:t-1}$ are the previous words.


Decoder
FC
Add & norm
Positionwise FFN
Add & norm
Muti-head attention
Add & norm
Masked multi-head attention
× n
Encoder
n ×
Add & norm
Positionwise FFN
Add & norm
Multi-head attention
Positional encoding
Positional encoding
Embedding
Embedding
Sources
Targets

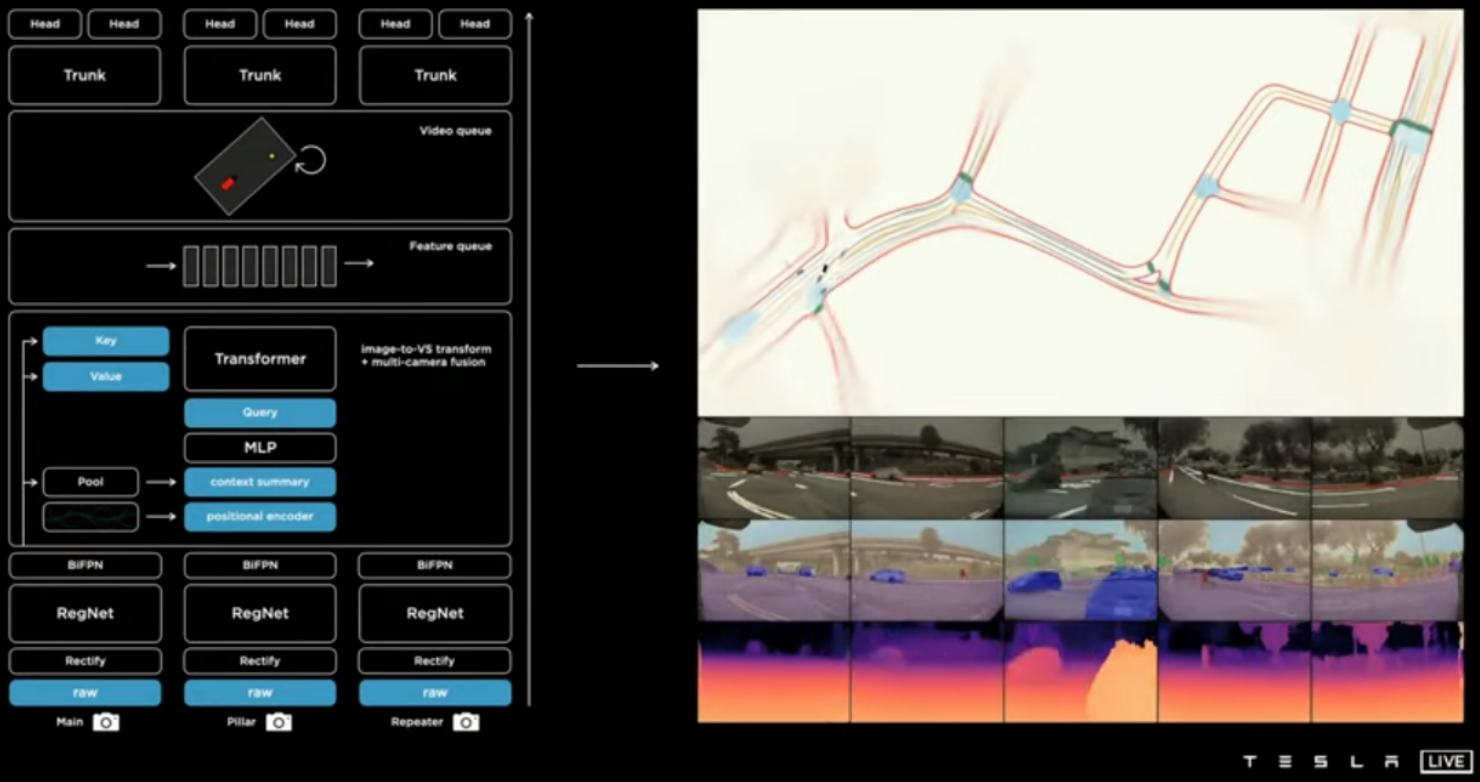


A brutal simplicity:

- The more data, the better the model.
- The more parameters, the better the model.
- The more compute, the better the model.

AI beyond Pacman

How AI Helps Autonomous Vehicles See Outside the Box
(See also other episodes from NVIDIA DRIVE Labs)



Hydranet (Tesla, 2021)

Improving Tuberculosis Monitoring with Deep Learning

Summary

- Learning is a key element of intelligence.
- Supervised learning is used to learn functions from a set of training examples.
  - Linear models are simple predictive models, effective on some tasks but usually insufficiently expressive.
  - Neural networks are defined as a composition of squashed linear models.

For the last forty years we have programmed computers; for the next forty years we will train them.

Chris Bishop, 2020.

The end.