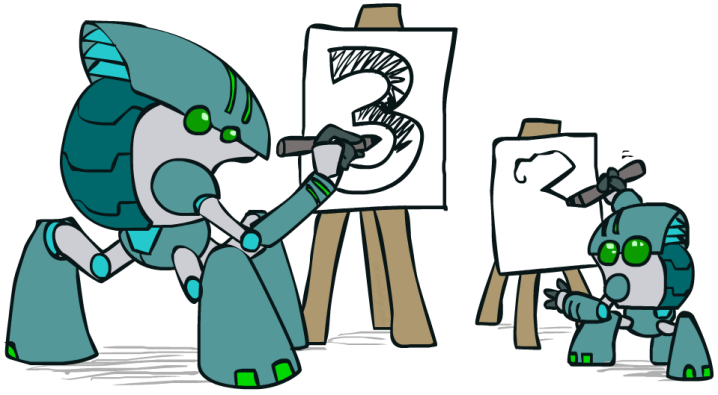
Introduction to Artificial Intelligence

Lecture 9: Learning

Prof. Gilles Louppe
g.louppe@uliege.be

LIÈGE université

Today



Make our agents capable of self-improvement through a learning mechanism.

- Statistical learning
- Supervised learning
  - Linear models
  - Perceptron
  - Neural networks
- Unsupervised learning

Image credits: CS188, UC Berkeley.

Intelligence?

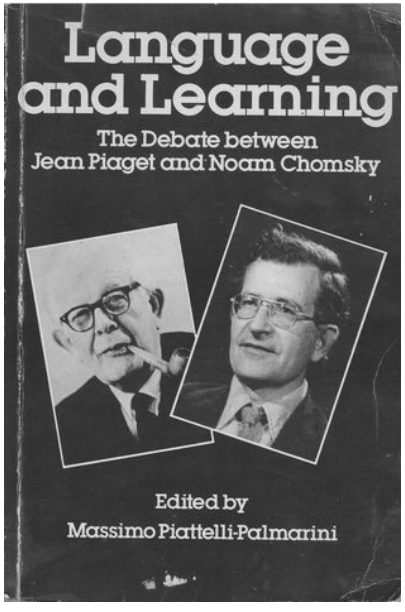
What we covered so far:

- Search algorithms, using a state space specified by domain knowledge.
- Constraint satisfaction problems, by exploiting a known structure of the states.
- Logical inference, using well-specified facts and inference rules.
- Adversarial search, for known and fully observable games.
- Reasoning about uncertain knowledge, as represented using domain-motivated probabilistic models.
- Taking optimal decisions, under uncertainty and possibly under partial observation.

Sufficient to implement complex and rational behaviors, in some situations. But is that intelligence? Aren't we missing a critical component?

Chomsky vs. Piaget

- Noam Chomsky's innatism:
  - State that humans possess a genetically determined faculty for thought and language.
  - The structures of language and thought are set in motion through interaction with the environment.
- Jean Piaget's constructivism:
  - Deny the existence of innate cognitive structure specific for thought and language.
  - Postulate instead all cognitive acquisitions, including language, to be the outcome of a gradual process of construction, i.e., a learning procedure.



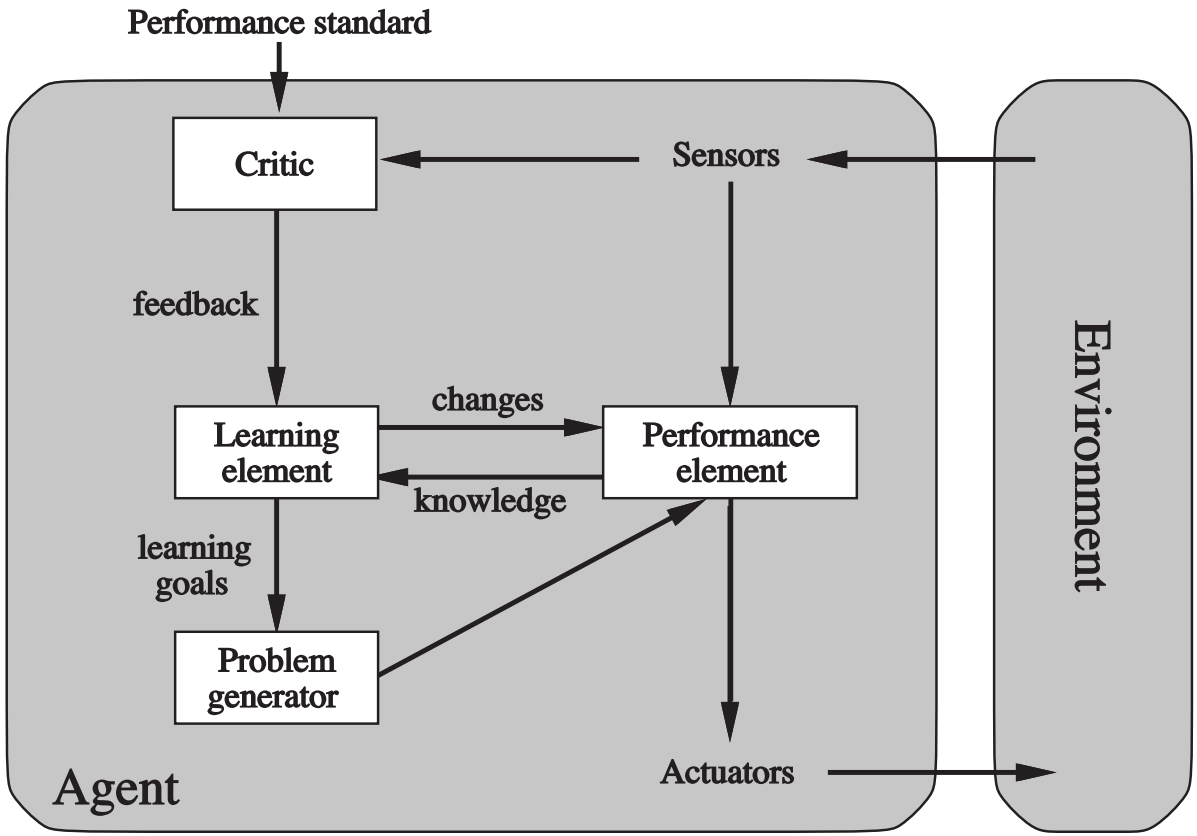
[Q] What about AI? Should it be a pre-wired efficient machine? Or a machine that can learn and improve? or maybe a bit of both?

The debate continues...

Learning agents

What if the environment is unknown?

- Learning can be used as a system construction method.
- Expose the agent to reality rather trying to hardcode reality into the agent's program.
- Learning provides an automated way to modify the agent's internal decision mechanisms to improve its own performance.


Performance standard
Critic
Sensors
feedback
changes
Learning element
Performance element
knowledge
learning goals
Problem generator
Actuators
Agent
Environment

The design of the learning element is dictated by:

- What type of performance element is used.
- Which functional component is to be learned.
- How that functional component is represented.
- What kind of feedback is available.

Performance element	Component	Representation	Feedback
Alpha–beta search	Eval. fn.	Weighted linear function	Win/loss
Logical agent	Transition model	Successor–state axioms	Outcome
Utility–based agent	Transition model	Dynamic Bayes net	Outcome
Simple reflex agent	Percept–action fn	Neural net	Correct action

Statistical learning

Bayesian learning

View learning as a Bayesian update of a probability distribution $P(H)$ over a hypothesis space, where

- H is the hypothesis variable
- values are $h_1, h_2, ...$
- the prior is $P(H)$
- $\mathbf{d}$ is the observed data

Given data, each hypothesis has a posterior probability

$$P(h_i|\mathbf{d}) = \frac{P(\mathbf{d}|h_i)P(h_i)}{P(\mathbf{d})},$$

where $P(\mathbf{d}|h_i)$ is called the likelihood.

Predictions use a likelihood-weighted average over the hypotheses:

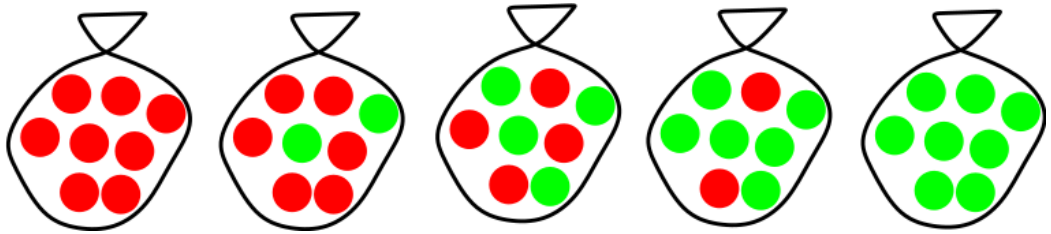
$$P(X|\mathbf{d}) = \sum_i P(X|\mathbf{d}, h_i)P(h_i|\mathbf{d}) = \sum_i P(X|h_i)P(h_i|\mathbf{d})$$

No need to pick one best-guess hypothesis!

Example

Suppose there are five kinds of bags of candies. Assume a prior $P(H)$:

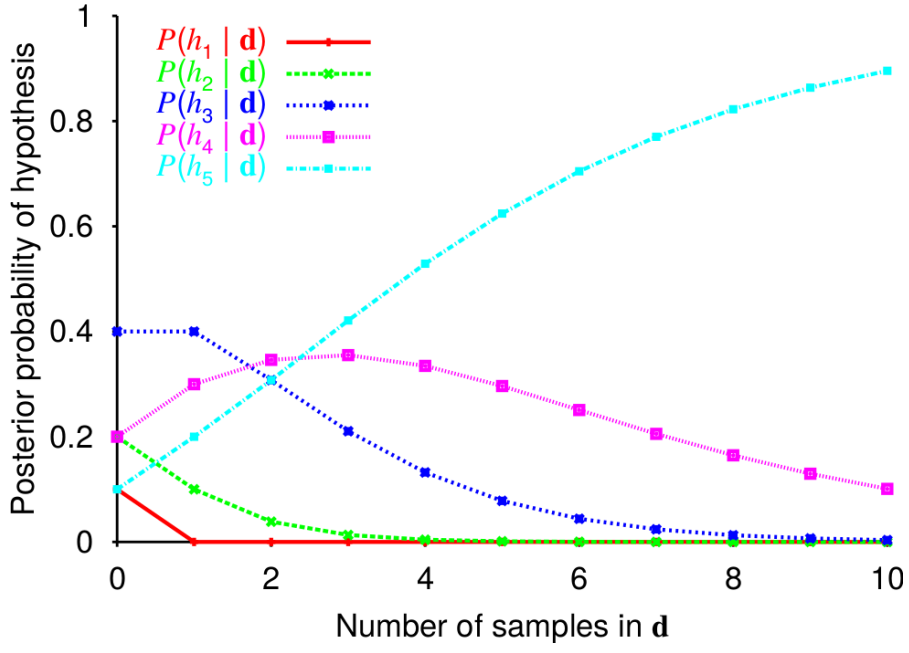
- $P(h_1) = 0.1$, with h_1: 100% cherry candies
- $P(h_2) = 0.2$, with h_2: 75% cherry candies + 25% lime candies
- $P(h_3) = 0.4$, with h_3: 50% cherry candies + 50% lime candies
- $P(h_4) = 0.2$, with h_4: 25% cherry candies + 75% lime candies
- $P(h_5) = 0.1$, with h_5: 100% lime candies



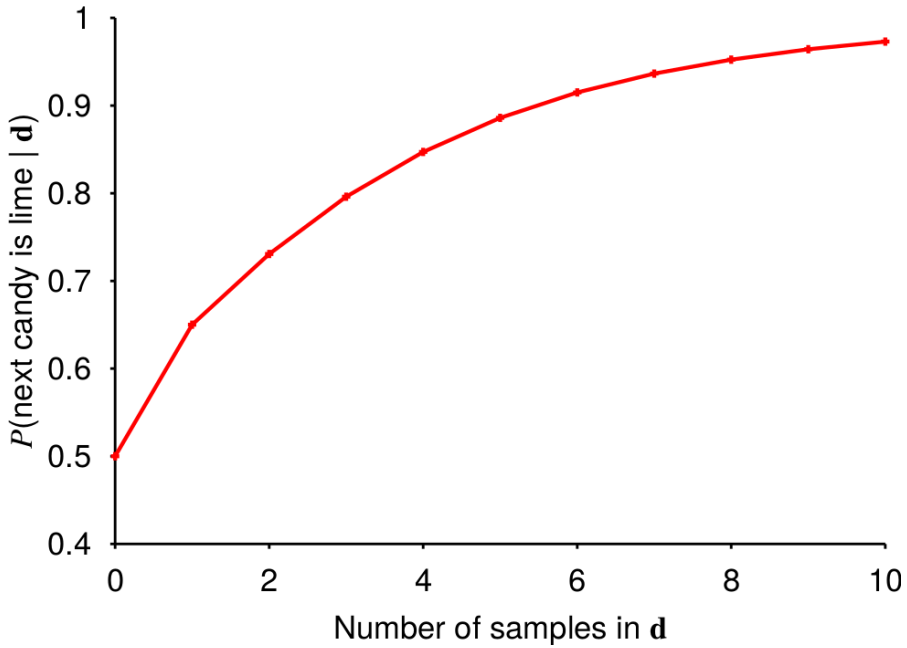
Then we observe candies drawn from some bag:

- What kind of bag is it?
- What flavor will the next candy be?

Posterior probability of hypotheses



Prediction probability



- This example illustrates the fact that the Bayesian prediction eventually agrees with the true hypothesis.
- The posterior probability of any false hypothesis eventually vanishes (under weak assumptions).

Maximum a posteriori

Summing over the hypothesis space is often intractable.

Instead, maximum a posteriori estimation (MAP) consists in using the hypothesis

$$h_{MAP} = \arg\max_{h_i} P(h_i|\mathbf{d}).$$

That is, maximize $P(\mathbf{d}|h_i)P(h_i)$ or $\log P(\mathbf{d}|h_i) + \log P(h_i)$.

- Log terms can be be viewed as (negative number of) bits to encode data given hypothesis + bits to encode hypothesis.
  - This is the basic idea of minimum description length learning, i.e., Occam's razor.
- Finding the MAP hypothesis is often much easier than Bayesian learning.
  - It requires solving an optimization problem instead of a large summation problem.

Maximum likelihood

For large data sets, the prior $P(H)$ becomes irrelevant.

In this case, maximum likelihood estimation (MLE) consists in using the hypothesis

$$h_{MLE} = \arg\max_{h_i} P(\mathbf{d}|h_i)$$

That is, simply get the best fit to the data.

- Identical to MAP for uniform prior.
- Maximum likelihood estimation is the standard (non-Bayesian) statistical learning method.

Recipe

- Choose a parameterized family of models to describe the data (e.g., a Bayesian network).
  - requires substantial insight and sometimes new models.
- Write down the log-likelihood L of the data as a function of the parameters.
  - may require summing over hidden variables, i.e., inference.
- Write down the derivative of the log likelihood w.r.t. each parameter θ.
- Find the parameter values θ such that the derivatives are zero.
  - may be hard/impossible; modern optimization techniques help.

Parameter learning in Bayesian networks

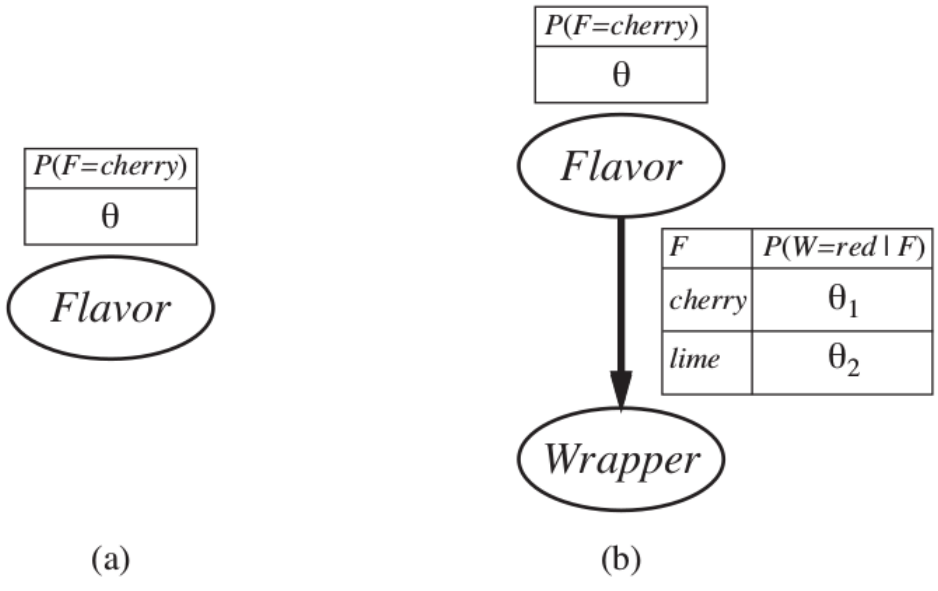


Figure 20.2 (a) Bayesian network model for the case of candies with an unknown proportion of cherries and limes. (b) Model for the case where the wrapper color depends (probabilistically) on the candy flavor.

MLE, case (a)

Bag from a new manufacturer; fraction θ of cherry candies?

- Any $\theta \in [0, 1]$ is possible: continuum of hypotheses h_θ.
- θ is a parameter for this simple binomial family of models.

Suppose we unwrap N candies, and get c cherries and $l = N - c$ limes. These are i.i.d. observations, so:

$$P(\mathbf{d}|h_\theta) = \prod_{j=1}^{N} P(d_j|h_\theta) = \theta^c(1-\theta)^l$$

Maximize this w.r.t. θ, which is easier for the log-likelihood:

$$L(\mathbf{d}|h_\theta) = \log P(\mathbf{d}|h_\theta) = c\log\theta + l\log(1-\theta)$$
$$\frac{dL(\mathbf{d}|h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{l}{1-\theta} = 0$$

Therefore $\theta = \frac{c}{N}$.

MLE, case (b)

Red/green wrapper depends probabilistically on flavor. E.g., the likelihood for a cherry candy in green wrapper:

$$\begin{aligned} & P(\text{cherry}, \text{green} | h_{\theta,\theta_1,\theta_2}) \\ &= P(\text{cherry} | h_{\theta,\theta_1,\theta_2}) P(\text{green} | \text{cherry}, h_{\theta,\theta_1,\theta_2}) \\ &= \theta(1-\theta_1) \end{aligned}$$

The likelihood for the data, given N candies, r_c red-wrapped cherries, g_c green-wrapped cherries, etc. is:

$$P(\mathbf{d} | h_{\theta,\theta_1,\theta_2}) = \theta^c (1-\theta)^l \theta_1^{r_c} (1-\theta_1)^{g_c} \theta_2^{r_l} (1-\theta_2)^{g_l}$$

$$\begin{aligned} L = & c \log \theta + l \log(1-\theta) + \\ & r_c \log \theta_1 + g_c \log(1-\theta_1) + \\ & r_l \log \theta_2 + g_l \log(1-\theta_2) \end{aligned}$$

Derivatives of L contain only the relevant parameter:

$$\frac{\partial L}{\partial \theta} = \frac{c}{\theta} - \frac{l}{1-\theta} = 0 \Rightarrow \theta = \frac{c}{c+l}$$
$$\frac{\partial L}{\partial \theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 \Rightarrow \theta_1 = \frac{r_c}{r_c+g_c}$$
$$\frac{\partial L}{\partial \theta_2} = \frac{r_l}{\theta_2} - \frac{g_l}{1-\theta_2} = 0 \Rightarrow \theta_2 = \frac{r_l}{r_l+g_l}$$

- This can be extended to any Bayesian network with parameterized CPTs.
- With complete data, maximum likelihood parameter estimation for a Bayesian network decomposes into separate optimization problems, one for each parameter.

Supervised learning

(mostly neural networks)

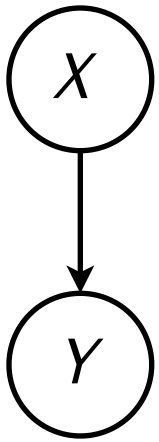
Problem statement

Assume data $\mathbf{d} \sim P(X, Y)$ 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 data and y_i was generated by an unknown function $y_i = f(\mathbf{x}_i)$.

- From this data, we wish to learn a function $h \in \mathcal{H}$ that approximates the true function f.
- $\mathcal{H}$ is huge! How do we find a good hypothesis?



In general, f will be stochastic. In this case, y is not strictly a function x, and we wish to learn the conditional $P(Y \mid X)$.

Most of supervised learning is actually (approximate) maximum likelihood estimation on (huge) parametric models.

Feature vectors

- Assume the input samples $\mathbf{x}_i \in \mathbb{R}^p$ are described as real-valued vectors of p 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.

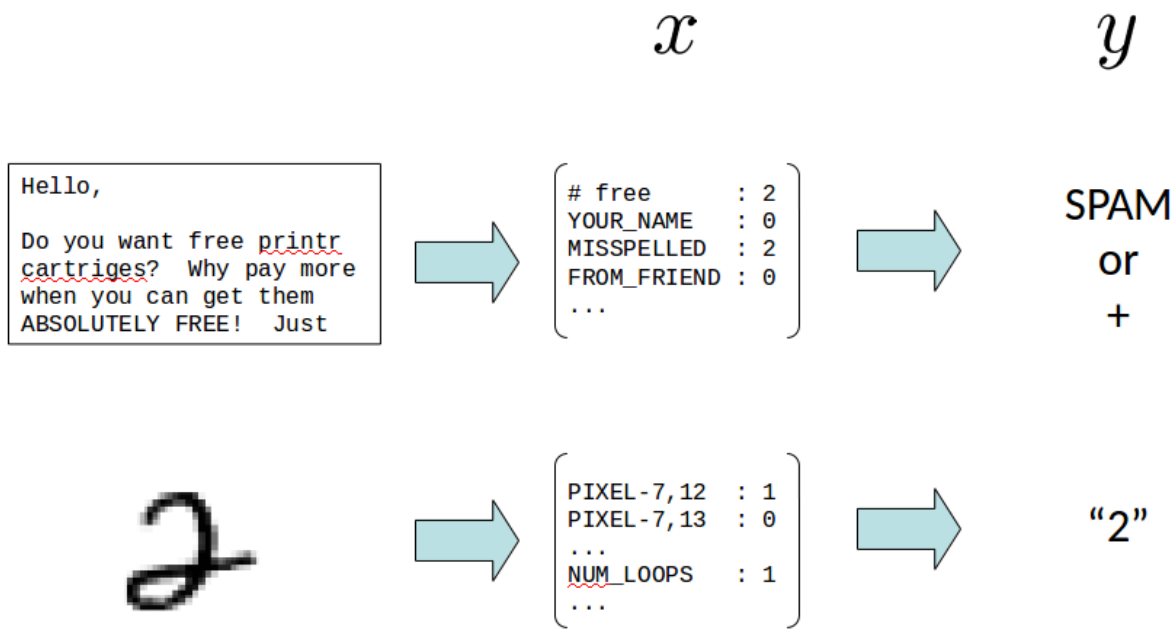
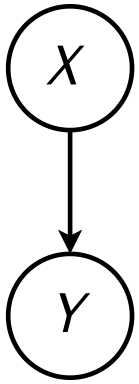
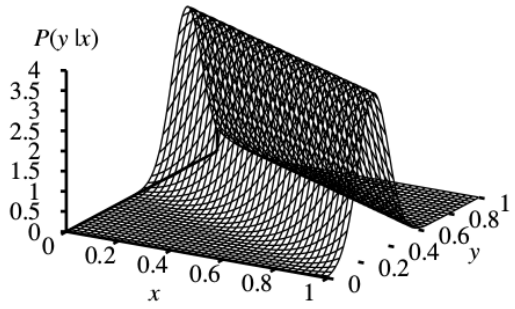
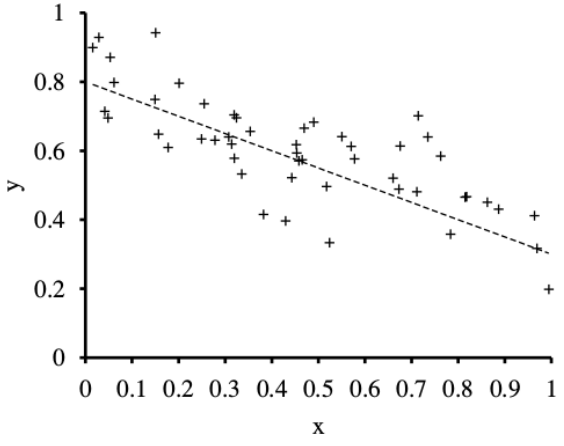


Image credits: CS188, UC Berkeley.

Linear regression







Let us assume a parameterized linear Gaussian model

$$y = \mathbf{w}^T \mathbf{x} + b + \epsilon$$

with one continuous parent X, one continuous child Y and $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

To learn the conditional distribution $p(y|x)$, we maximize

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

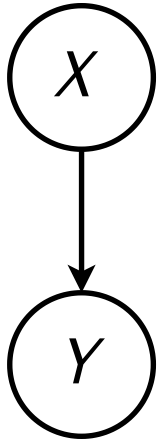
w.r.t. $\mathbf{w}$ and b over the data $\mathbf{d} = \{(x_j, y_j)\}$.

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.

Linear classification

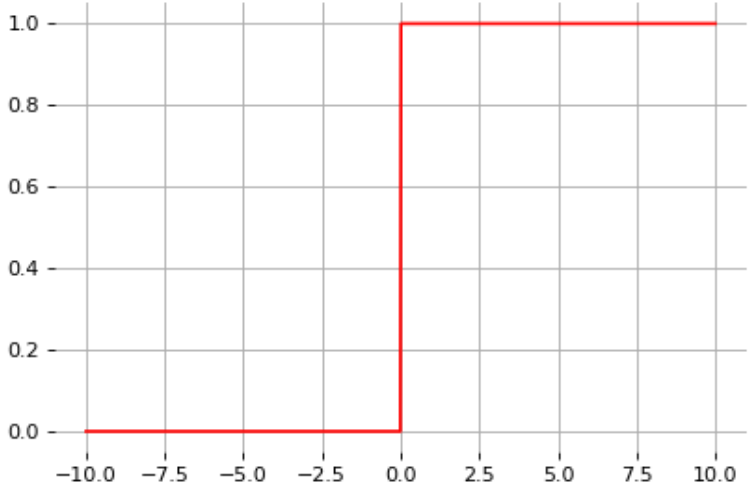


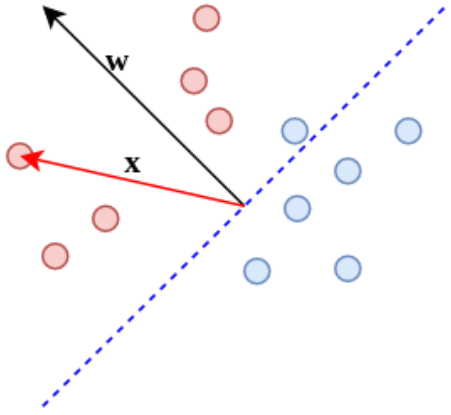
Let us now assume Y takes discrete values in $\{0, 1\}$.

Decision rules

The linear classifier model is a squashed linear function of its inputs:

$$h(\mathbf{x}; \mathbf{w}, b) = \text{sign}(\mathbf{w}^T \mathbf{x} + b)$$

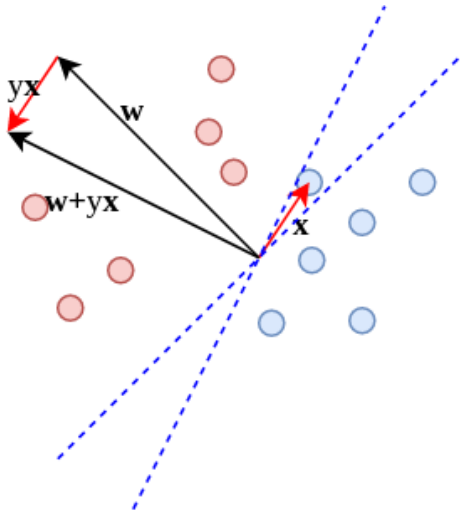




- Without loss of generality, the model can be rewritten without b as $h(\mathbf{x}; \mathbf{w}) = \text{sign}(\mathbf{w}^T \mathbf{x})$, where $\mathbf{w} \in \mathbb{R}^{p+1}$ and $\mathbf{x}$ is extended with a dummy element $x_0 = 1$.
- Predictions are computed by comparing the feature vector $\mathbf{x}$ to the weight vector $\mathbf{w}$. Geometrically, $\mathbf{w}^T \mathbf{x}$ corresponds to the $||\mathbf{w}||||\mathbf{x}|| \cos(\theta)$.
- The family $\mathcal{H}$ of hypothesis is induced from the set of possible parameters values $\mathbf{w}$, that is $\mathbb{R}^{p+1}$. Learning consists in finding a good vector $\mathbf{w}$ in this space.

Perceptron

- Start with $\mathbf{w} = 0$.
- For each training example $(\mathbf{x}, y)$:
  - Classify with current weights: $\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x})$
  - If $y = \hat{y}$, do nothing.
  - Otherwise, update parameters: $\mathbf{w} = \mathbf{w} + y\mathbf{x} - (1 - y)\mathbf{x}$



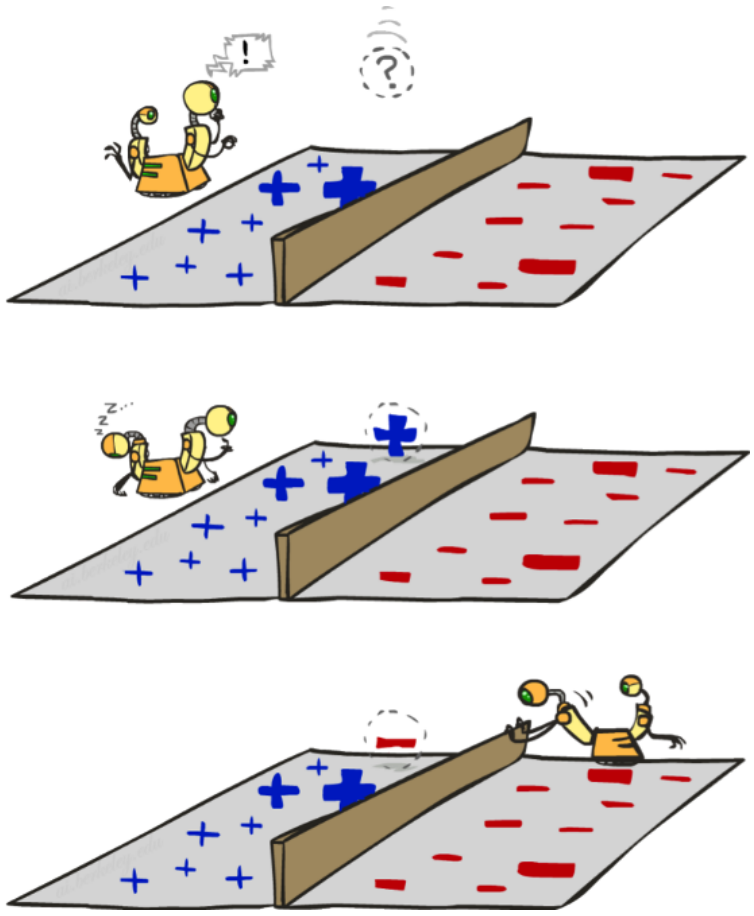
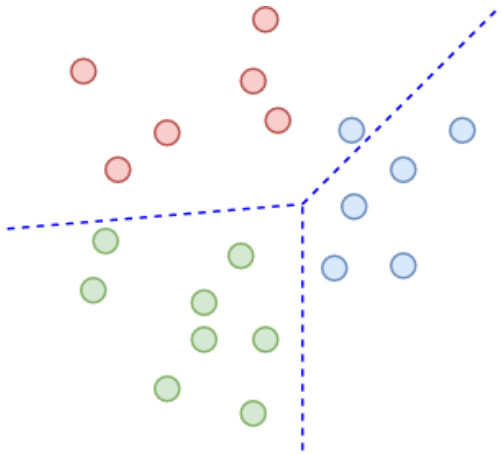


Image credits: CS188, UC Berkeley.

Multiclass perceptron

- If we have more than 2 classes, then
  - Define a weight vector $\mathbf{w}_c$ for each class c.
  - The activation for class c is $\mathbf{w}_c^T \mathbf{x}$.
- Learning:
  - Start with $\mathbf{w}_c = 0$ for all c.
  - For each training example $(\mathbf{x}, y)$:
    - Classify with current weights: $\hat{y} = \arg\max_c \mathbf{w}_c^T \mathbf{x}$
    - If $y = \hat{y}$, do nothing.
    - Otherwise, update parameters:
      - $\mathbf{w}_y = \mathbf{w}_y + \mathbf{x}$ (raise score of right answer)
      - $\mathbf{w}_{\hat{y}} = \mathbf{w}_{\hat{y}} - \mathbf{x}$ (lower score of wrong answer).



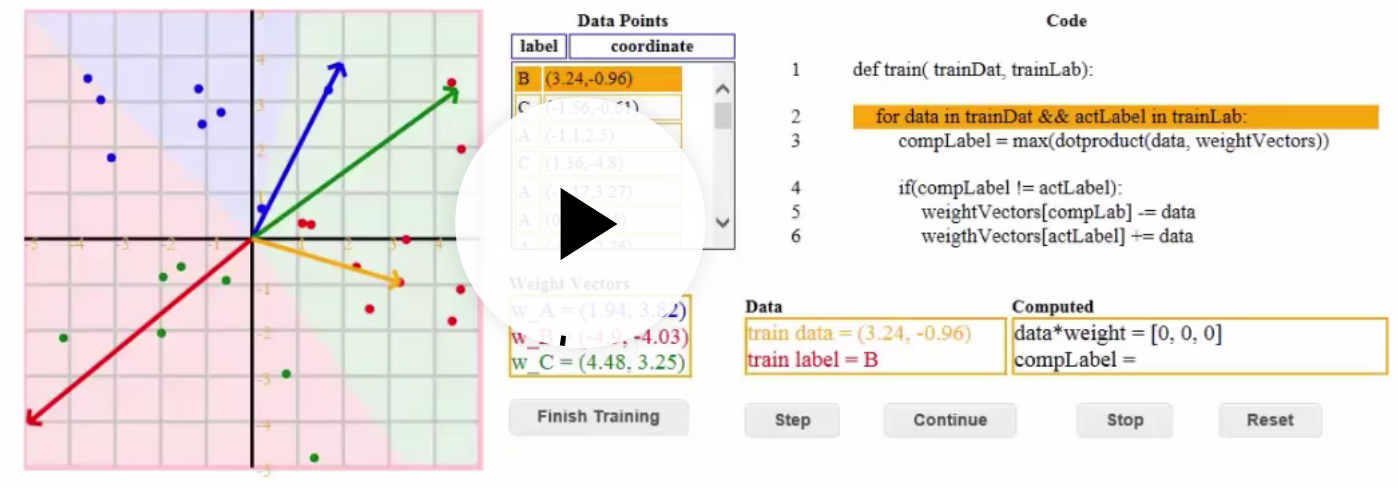


Image credits: CS188, UC Berkeley.

Apprenticeship

Can we learn to play Pacman 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 $f : \mathbb{R}^p \to \mathcal{A}$.
- This is a multiclass classification problem.

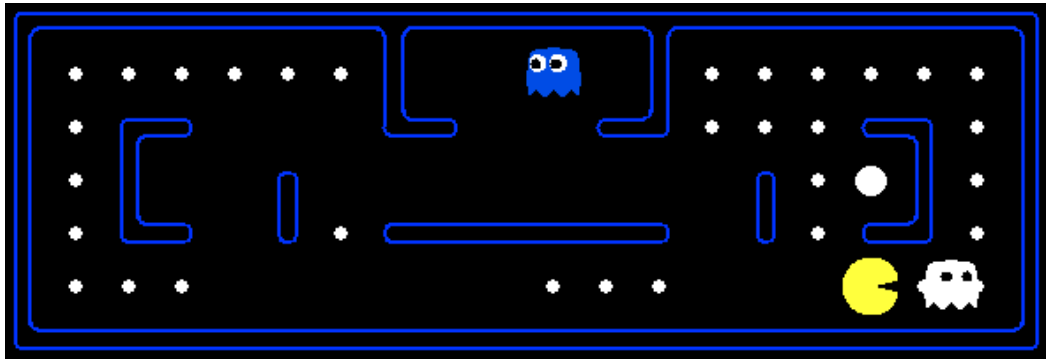
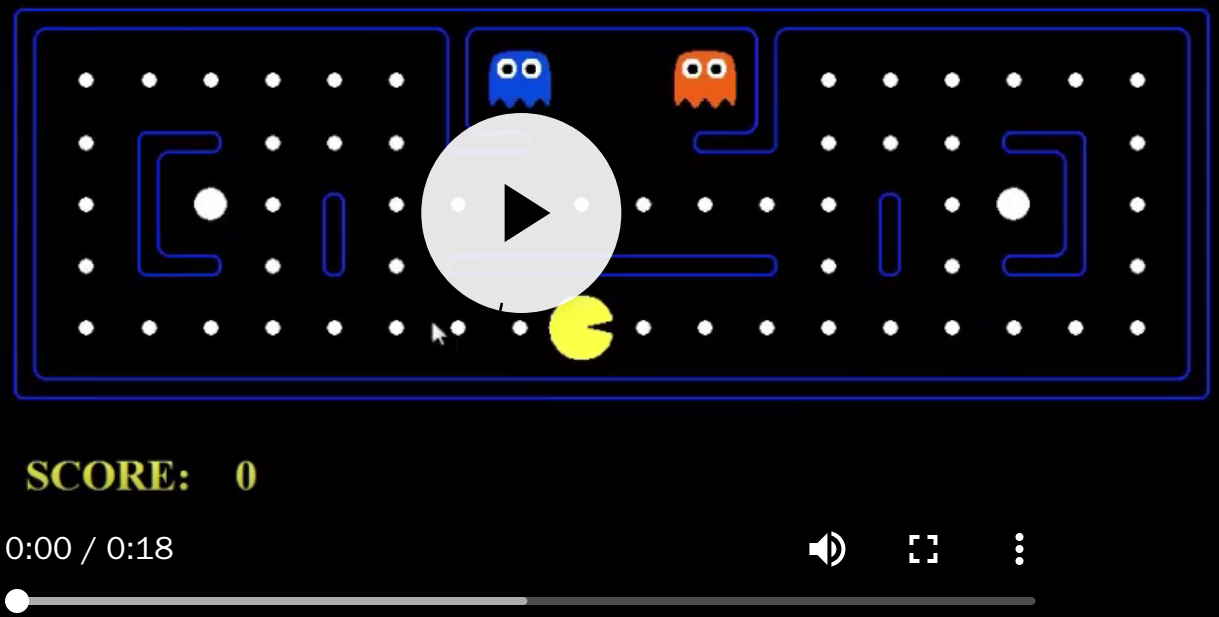


Image credits: CS188, UC Berkeley.



The Perceptron 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.

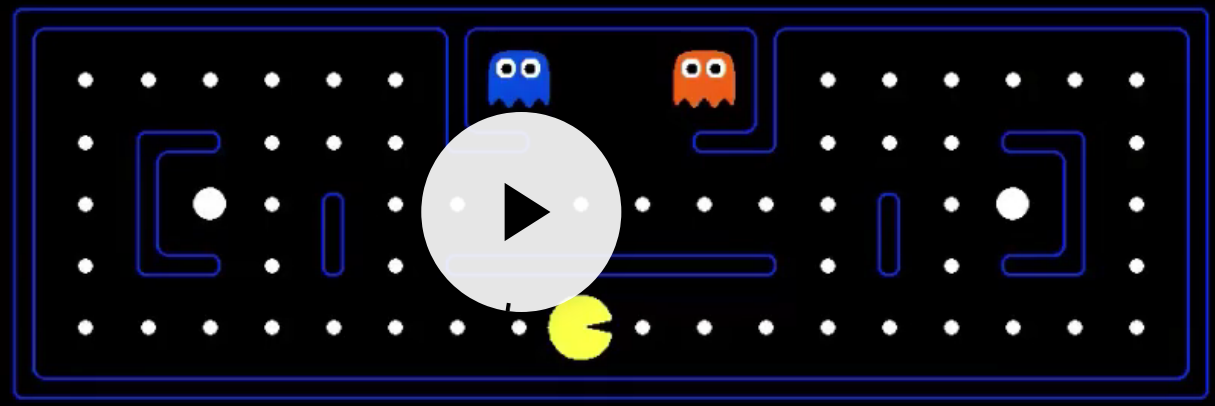
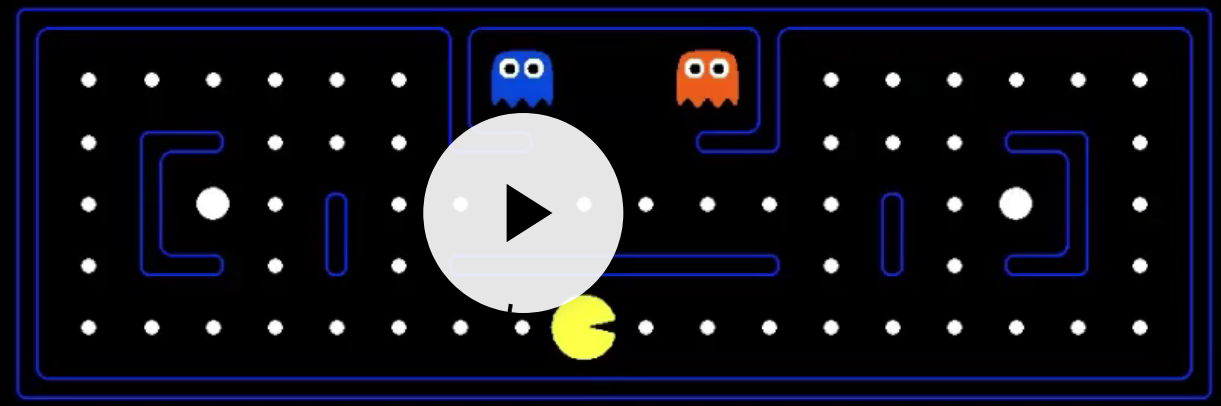


Image credits: CS188, UC Berkeley.



After two training episodes, the Perceptron agents plays.
No more Minimax!

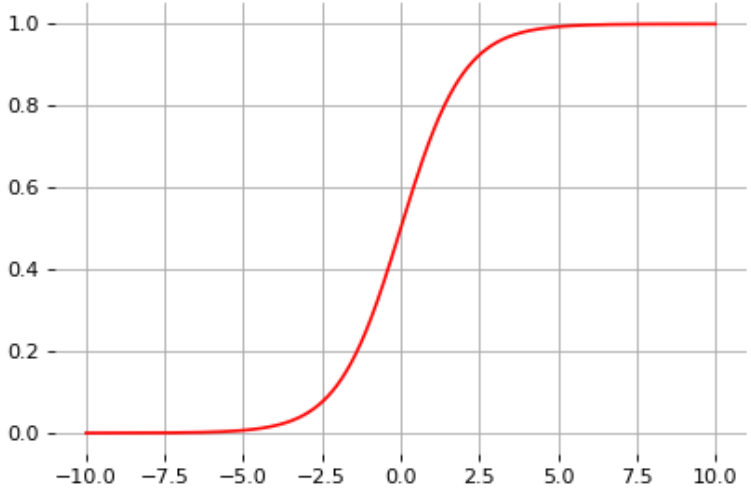
Image credits: CS188, UC Berkeley.

Logistic regression

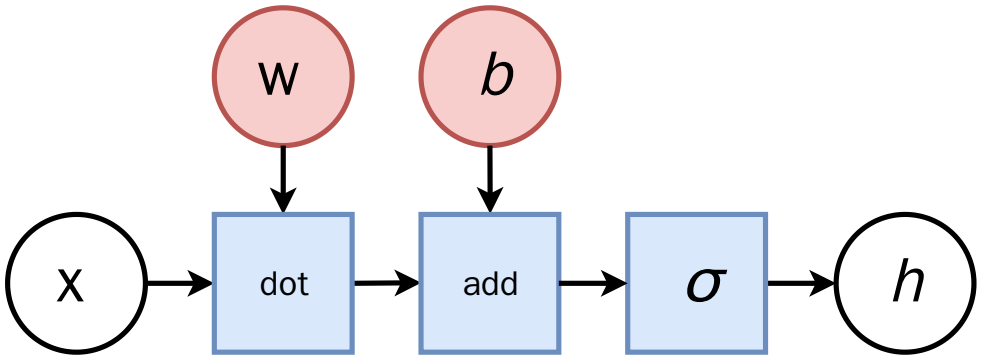
An alternative to the hard threshold model based on the sign function is to consider that $P(Y = 1|\mathbf{x})$ varies smoothly with $\mathbf{x}$. The logistic regression model postulates

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

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



In terms of matrix operations, the computational graph of h can be represented as:



where

- white nodes correspond to inputs and outputs;
- red nodes correspond to model parameters;
- blue nodes correspond to intermediate operations, which themselves produce intermediate output values (not represented).

This unit is the core component all neural networks!

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

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

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

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

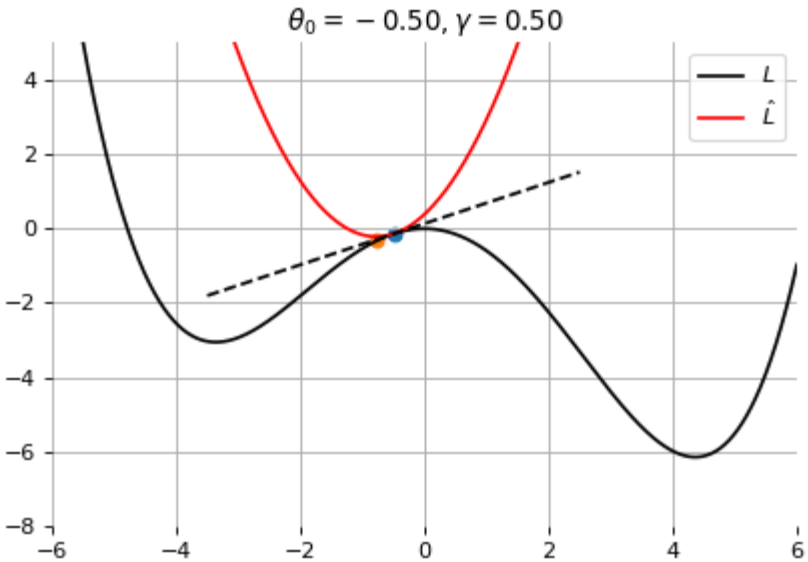
which results in the best improvement for the step $\epsilon = -\gamma \nabla_\theta \mathcal{L}(\theta_0)$.

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

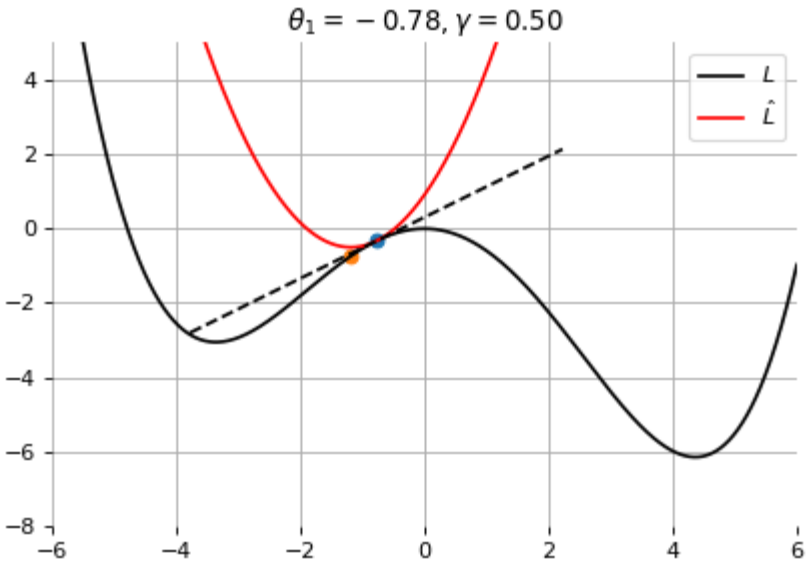
$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t)$$

Notes:

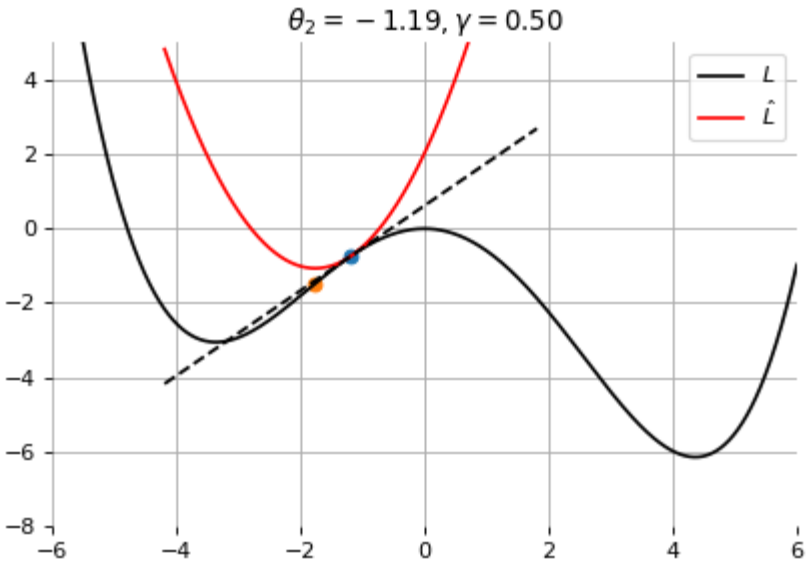
- θ_0 are the initial parameters of the model;
- γ is the learning rate;
- both are critical for the convergence of the update rule.



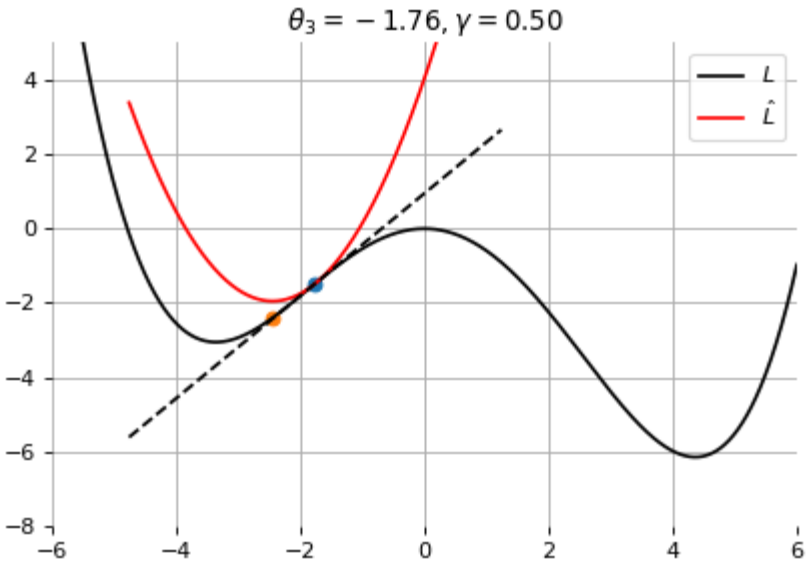
Example 1: Convergence to a local minima



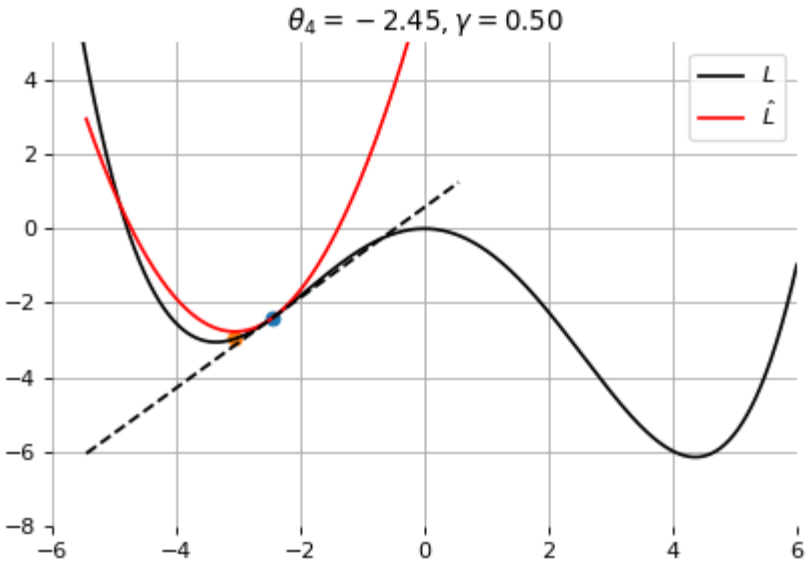
Example 1: Convergence to a local minima



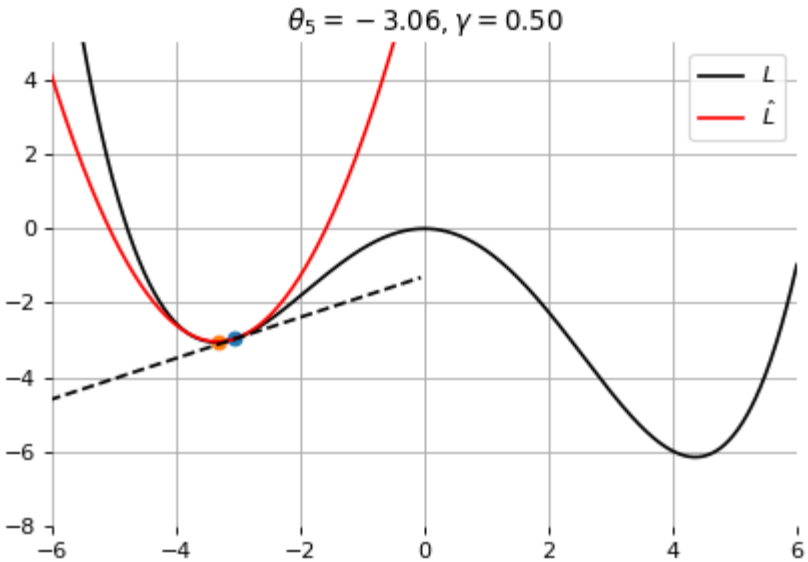
Example 1: Convergence to a local minima



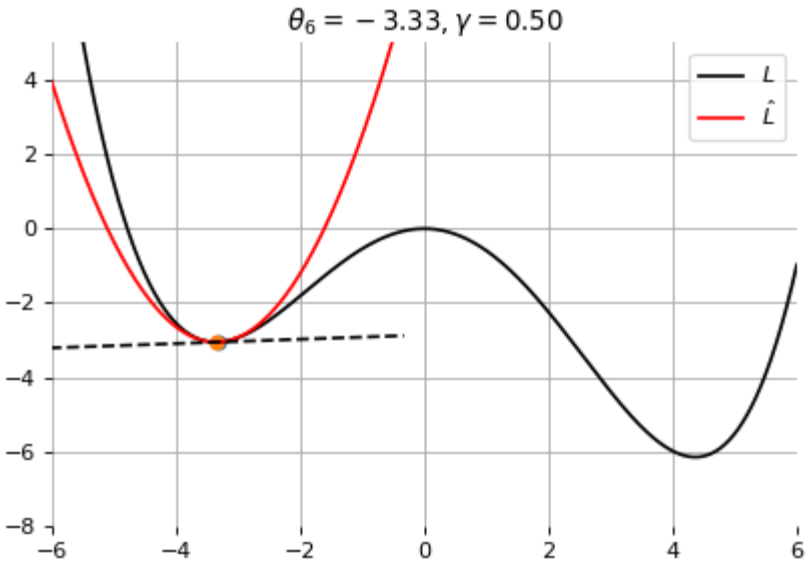
Example 1: Convergence to a local minima



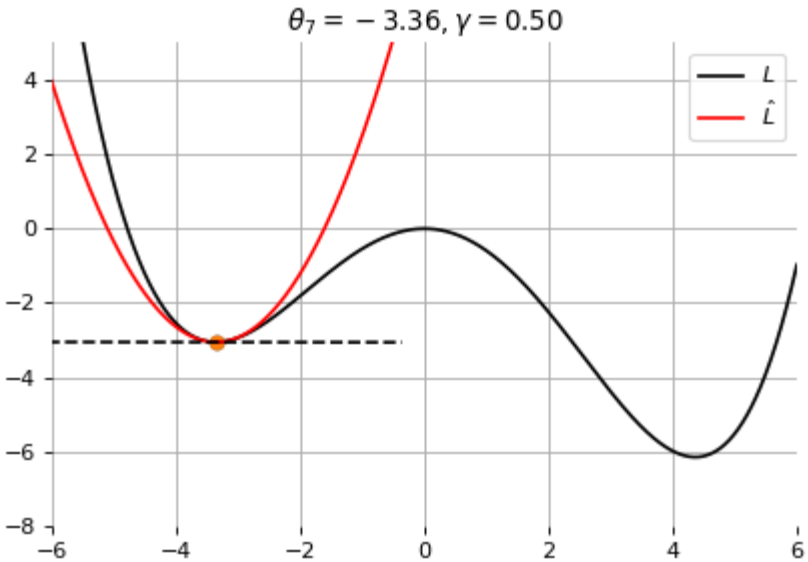
Example 1: Convergence to a local minima



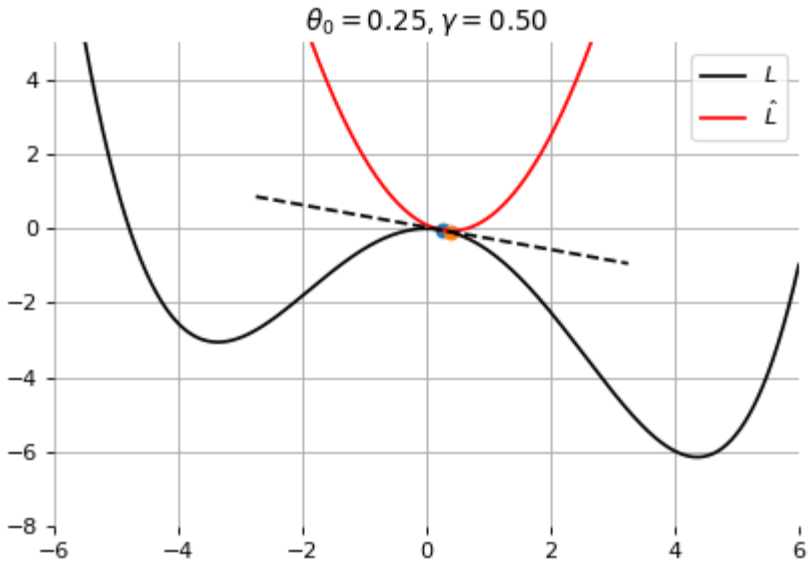
Example 1: Convergence to a local minima



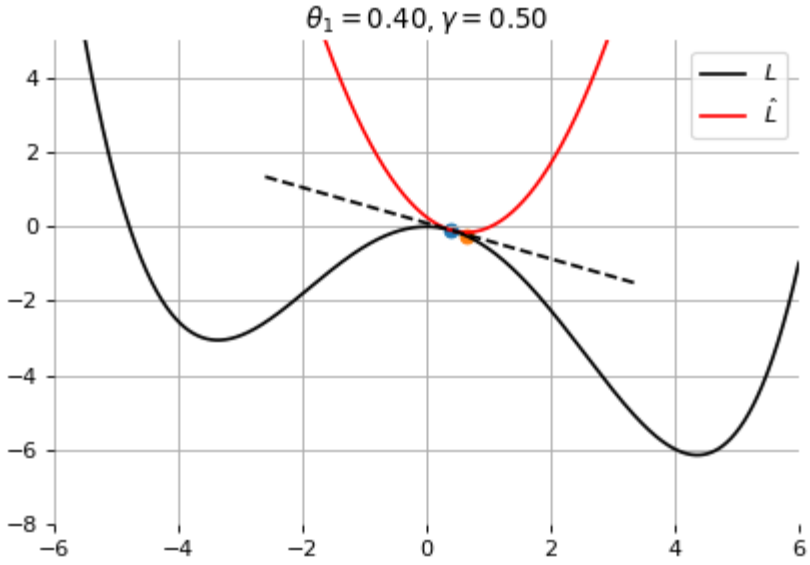
Example 1: Convergence to a local minima



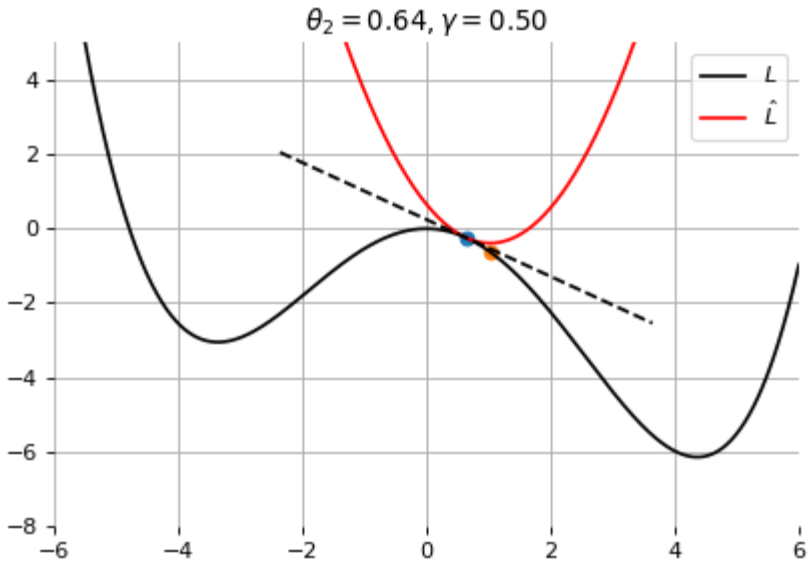
Example 1: Convergence to a local minima



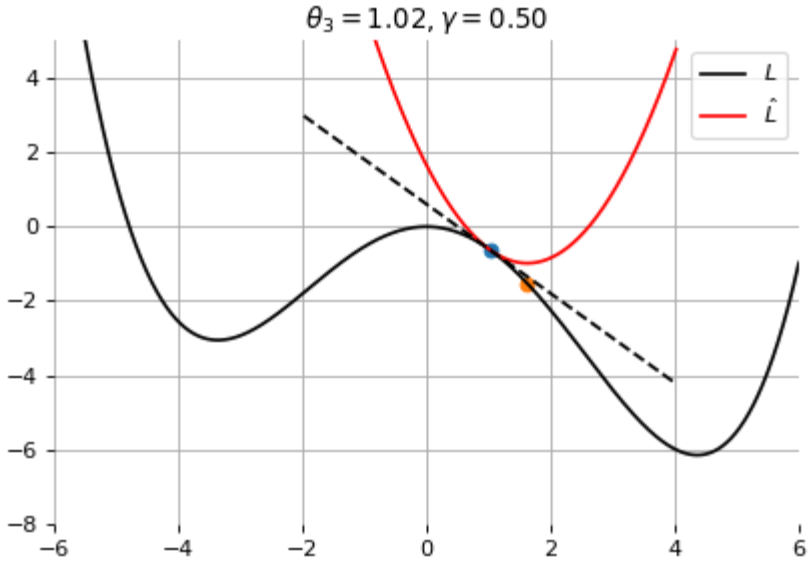
Example 2: Convergence to the global minima



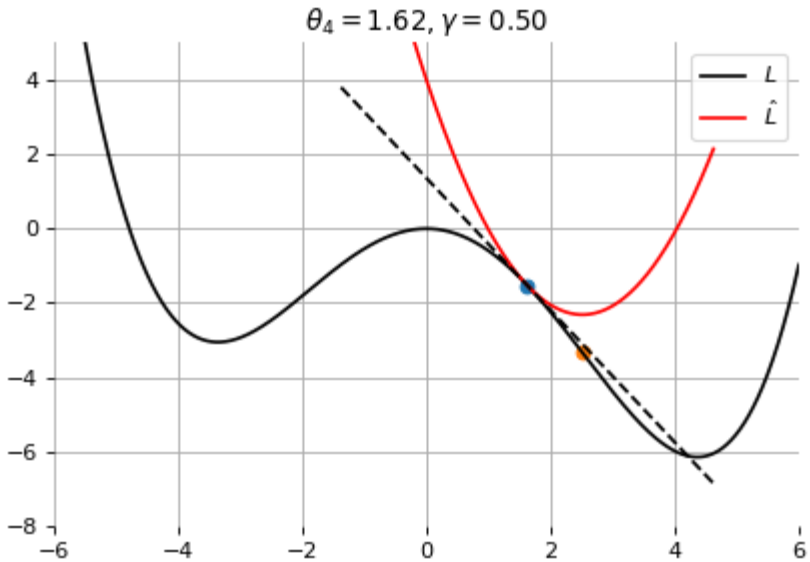
Example 2: Convergence to the global minima



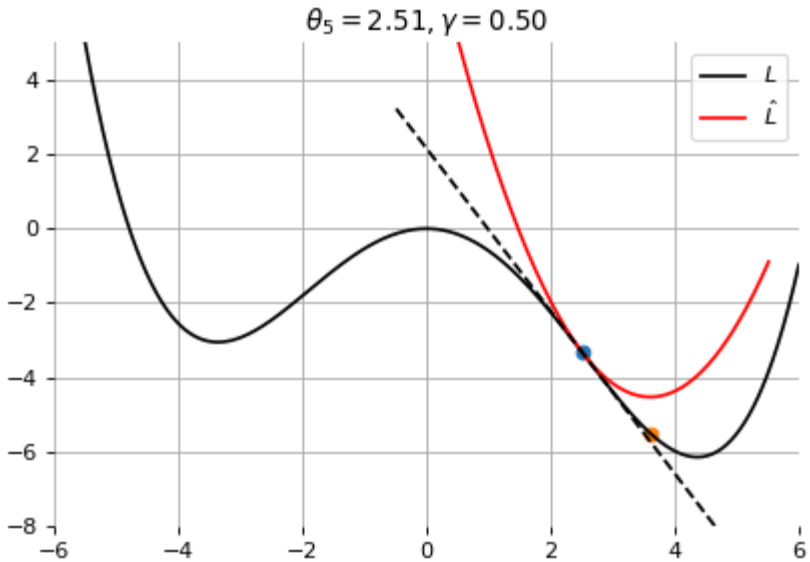
Example 2: Convergence to the global minima



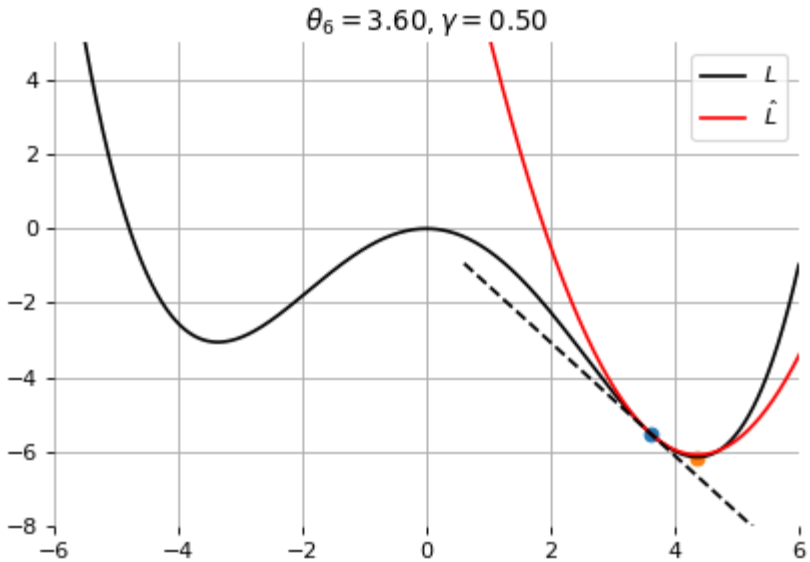
Example 2: Convergence to the global minima



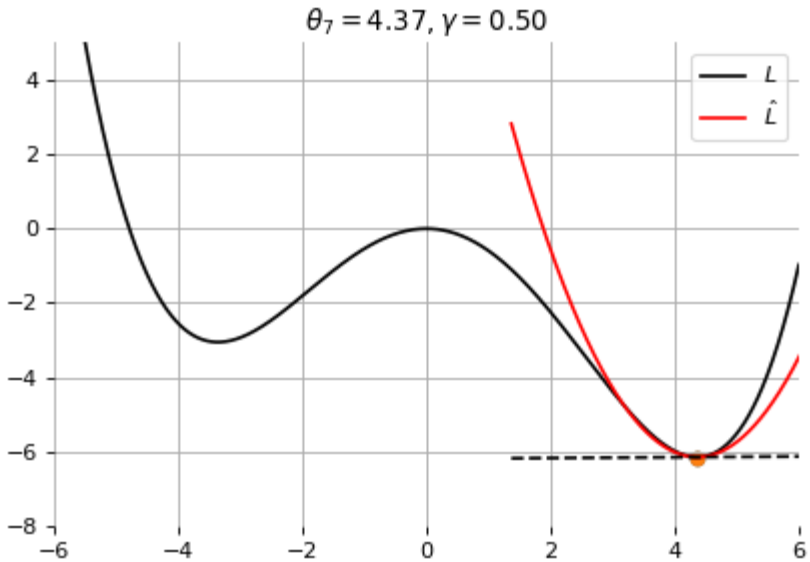
Example 2: Convergence to the global minima



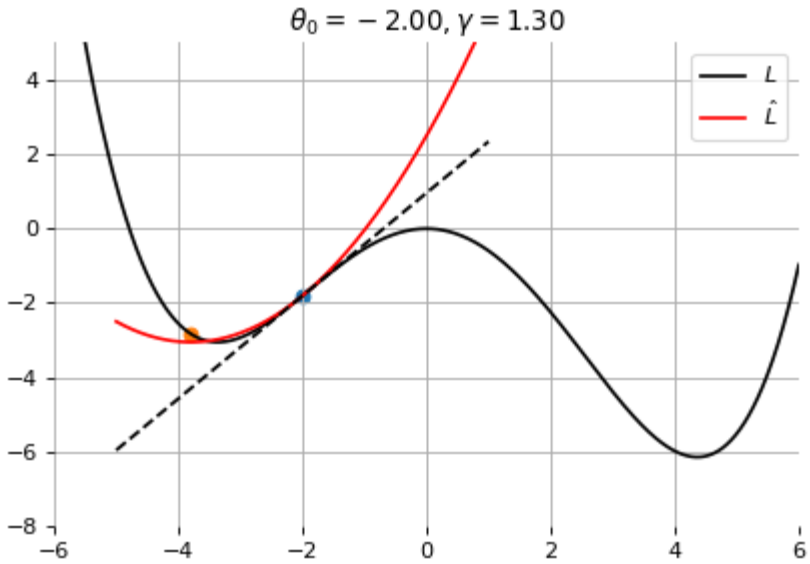
Example 2: Convergence to the global minima



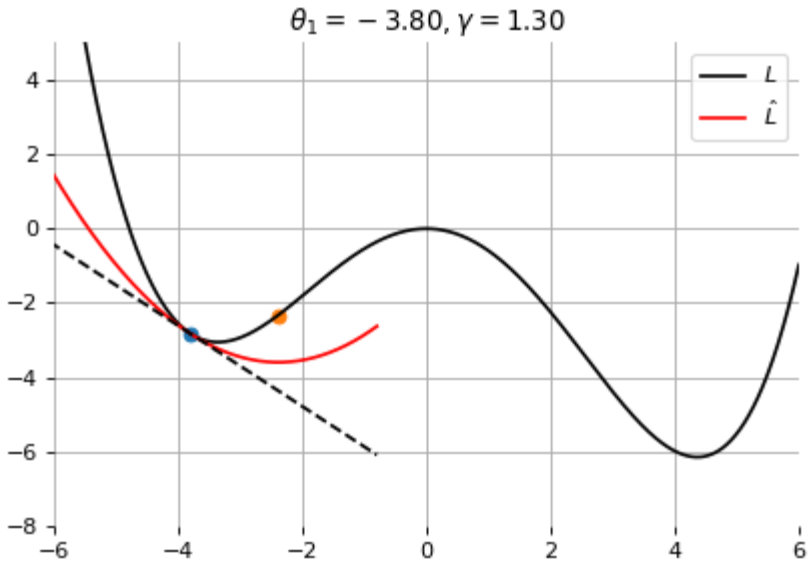
Example 2: Convergence to the global minima



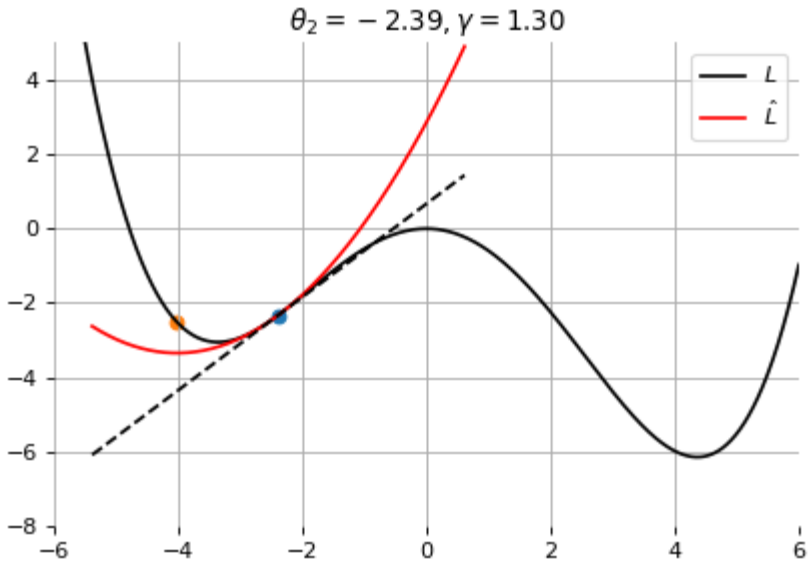
Example 2: Convergence to the global minima



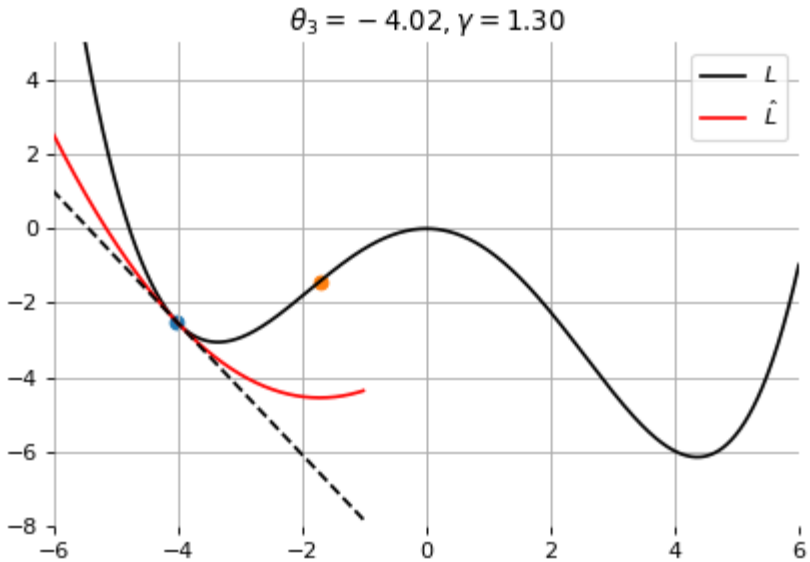
Example 3: Divergence due to a too large learning rate



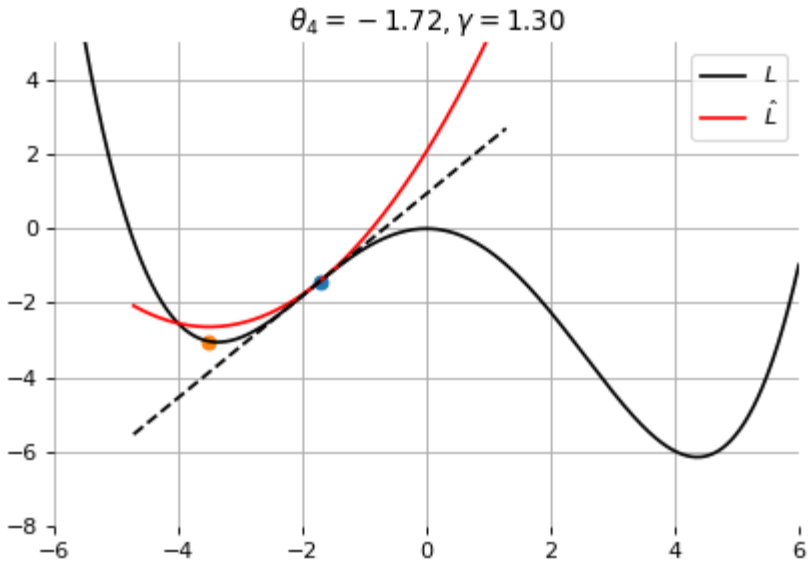
Example 3: Divergence due to a too large learning rate



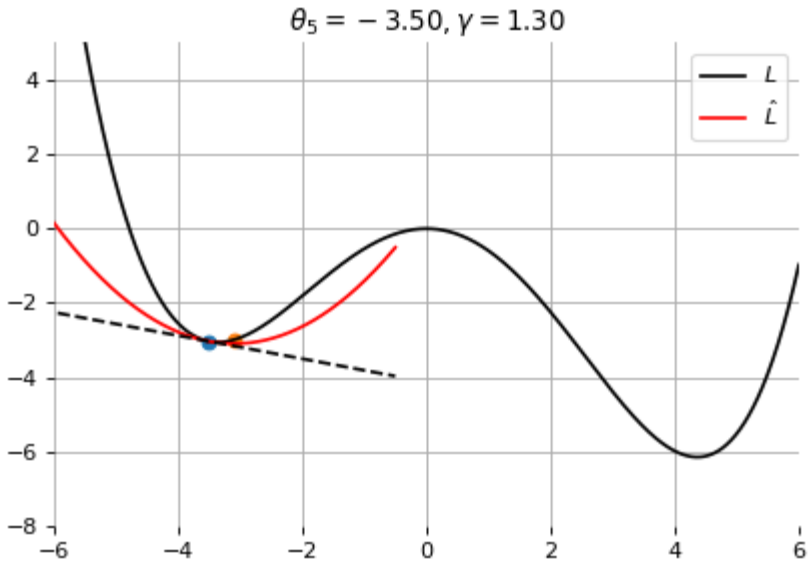
Example 3: Divergence due to a too large learning rate



Example 3: Divergence due to a too large learning rate



Example 3: Divergence due to a too large learning rate



Example 3: Divergence due to a too large learning rate

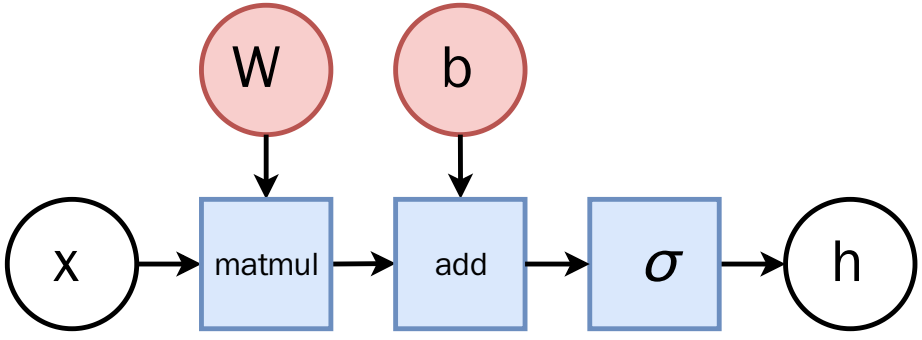
Layers

So far we considered the logistic unit $h = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right)$, where $h \in \mathbb{R}, \mathbf{x} \in \mathbb{R}^p$, $\mathbf{w} \in \mathbb{R}^p$ 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 $\mathbf{h} \in \mathbb{R}^q, \mathbf{x} \in \mathbb{R}^p, \mathbf{W} \in \mathbb{R}^{p\times q}, b \in \mathbb{R}^q$ and where $\sigma(\cdot)$ is upgraded to the element-wise sigmoid function.



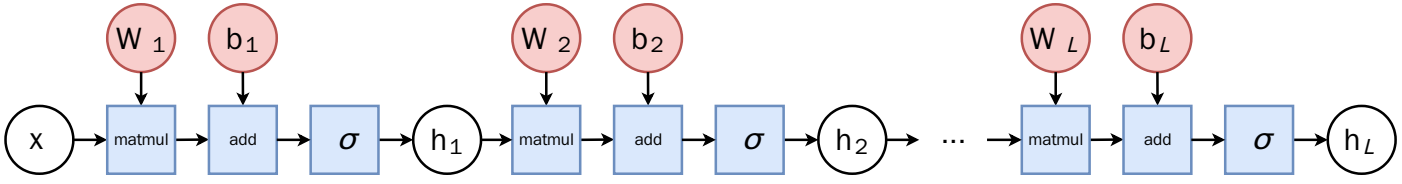
Multi-layer perceptron

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

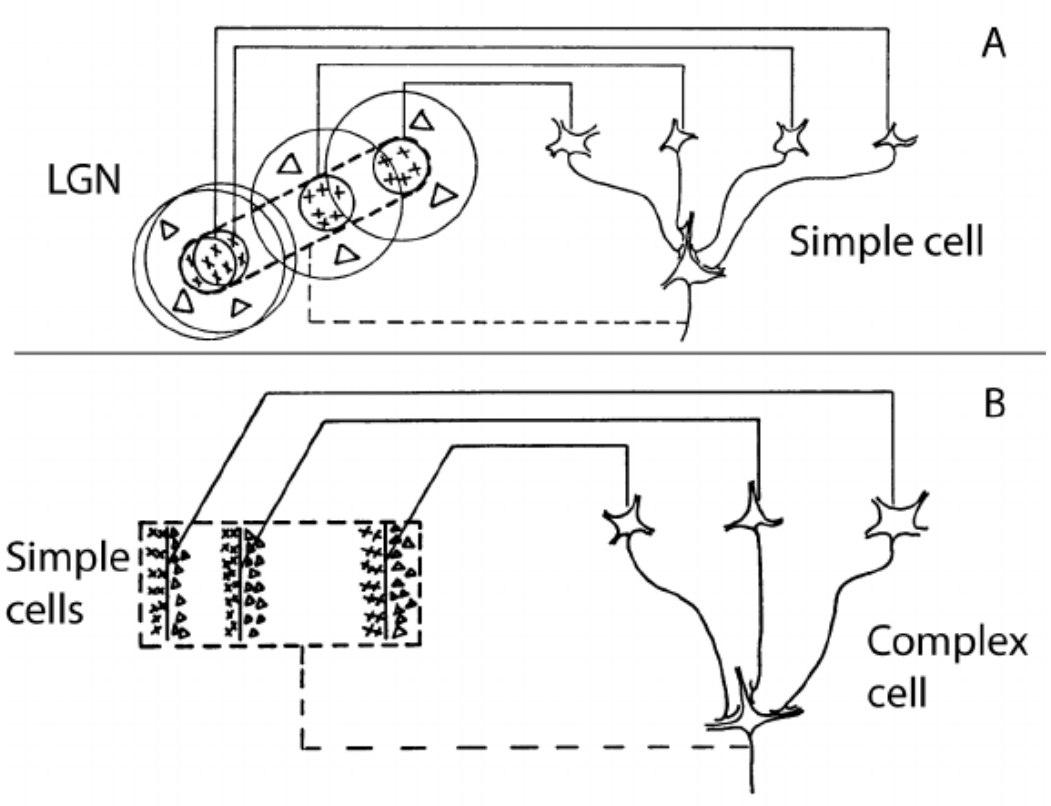
$$
\begin{aligned}
\mathbf{h}_0 &= \mathbf{x} \\
\mathbf{h}_1 &= \sigma(\mathbf{W}_1^T \mathbf{h}_0 + \mathbf{b}_1) \\
&\dots \\
\mathbf{h}_L &= \sigma(\mathbf{W}_L^T \mathbf{h}_{L-1} + \mathbf{b}_L) \\
f(\mathbf{x}; \theta) &= \mathbf{h}_L
\end{aligned}
$$

where θ denotes the model parameters $\{\mathbf{W}_k, \mathbf{b}_k, ... | k = 1, ..., L\}$.

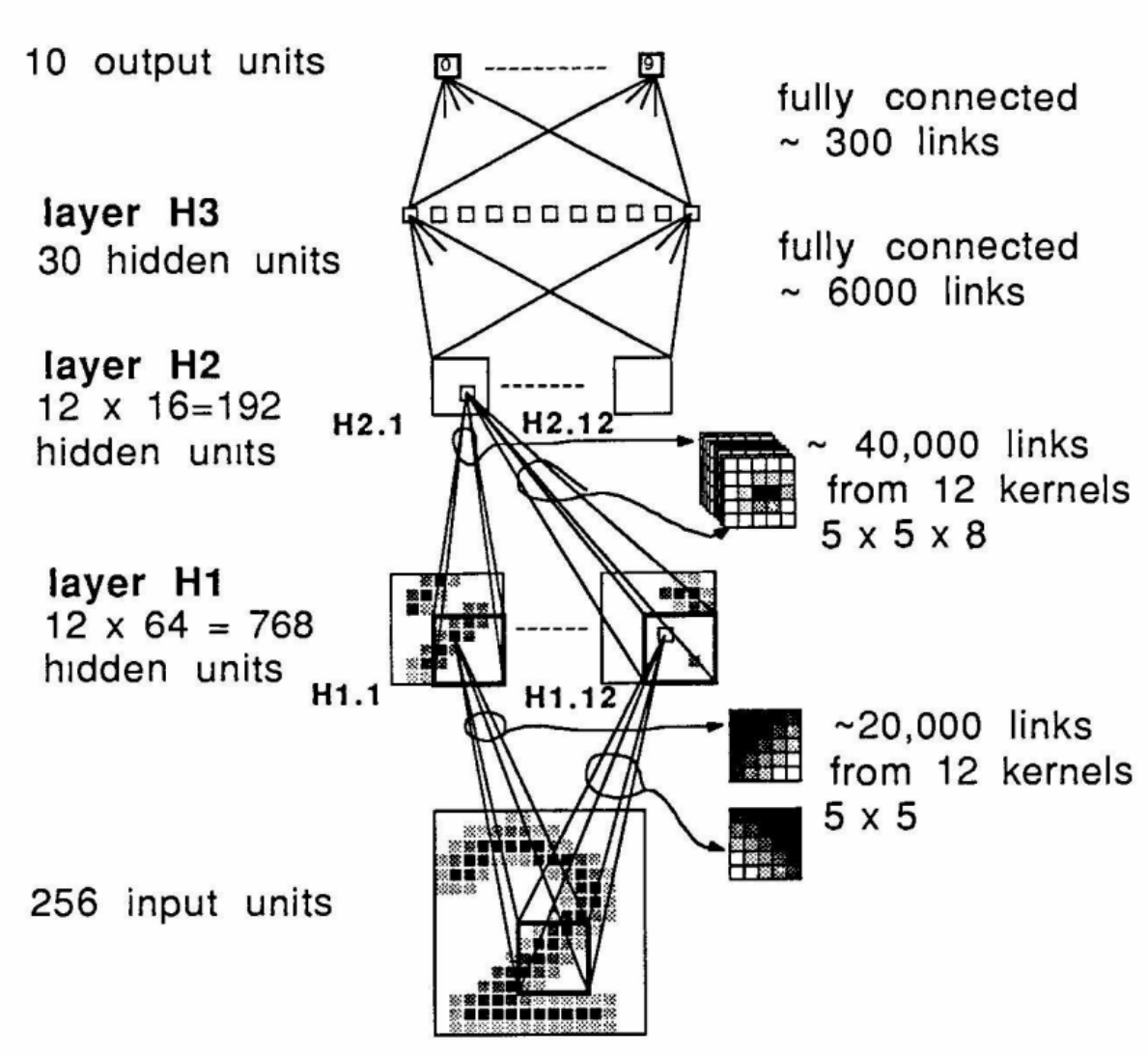
- This model is the multi-layer perceptron, also known as the fully connected feedforward network.
- Optionally, the last activation σ can be skipped to produce unbounded output values $\hat{y} \in \mathbb{R}$.


W 1
b1
W 2
b2
W L
b L
x
matmul
add
σ
h1
matmul
add
σ
h2
...
matmul
add
σ
h L

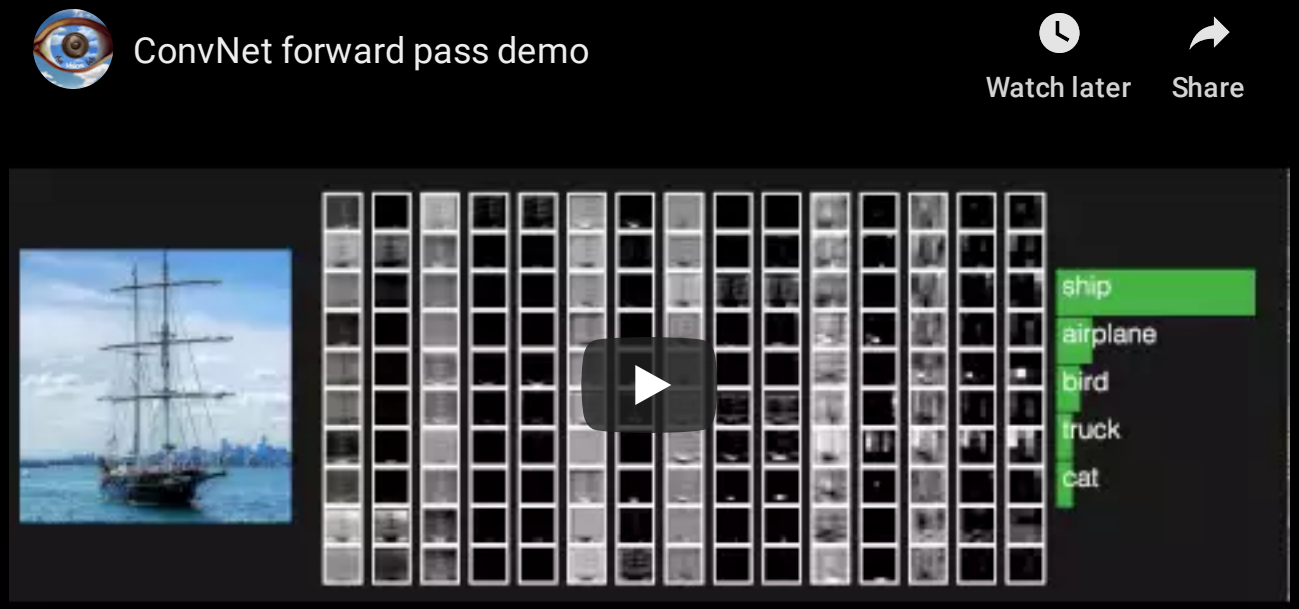
Convolutional networks



Hubel and Wiesel, 1962



Convolutional network (LeCun et al, 1989)



Deep neural networks learn a hierarchical composition of features.

LeNet-1, LeCun et al, 1993.

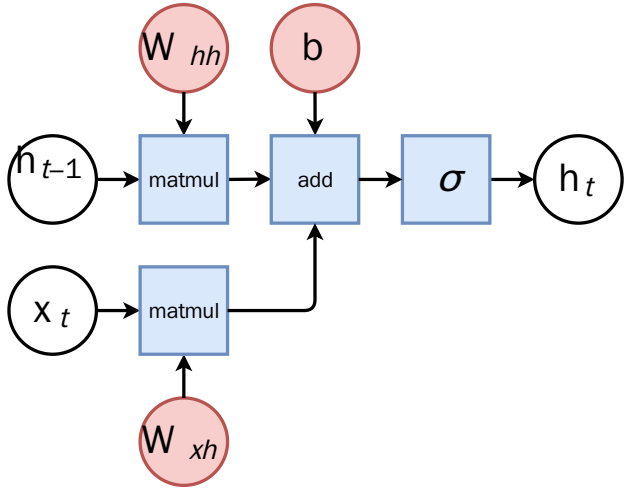
Recurrent networks

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,

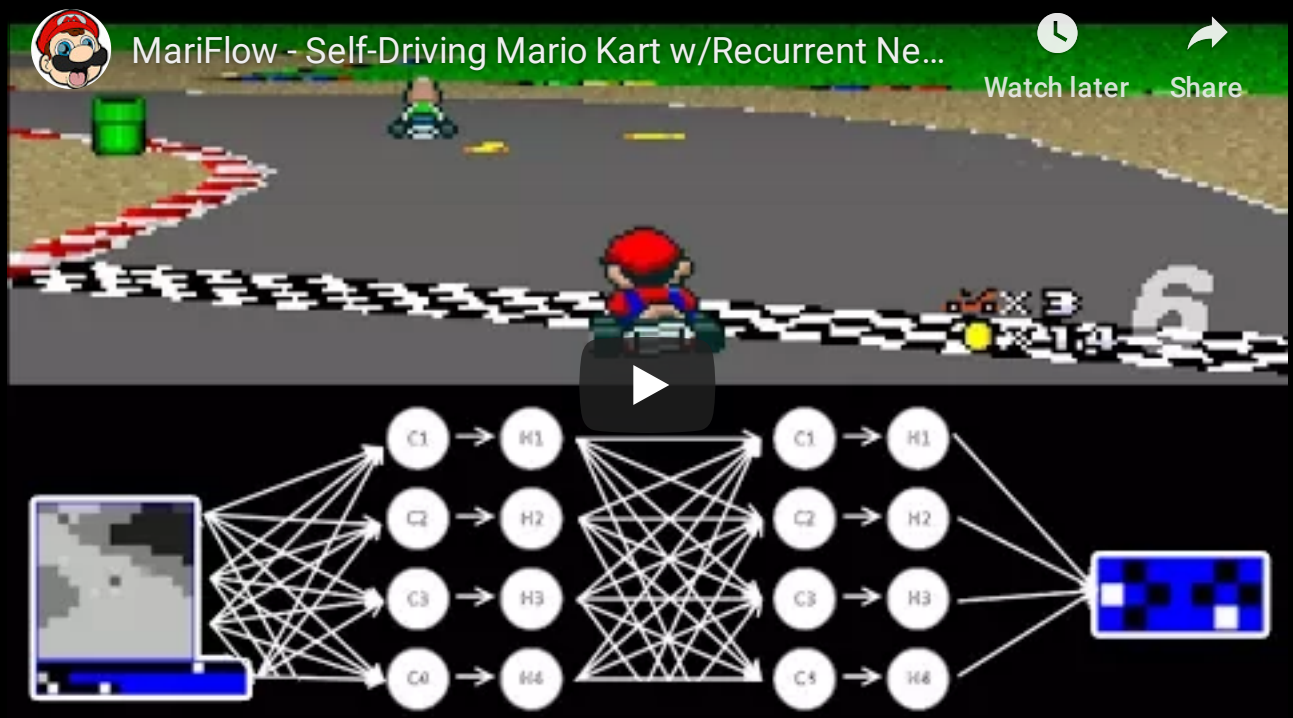
$$\mathbf{h}_t = \sigma(\mathbf{W}_{xh}^T \mathbf{x} + \mathbf{W}_{hh}^T \mathbf{h}_{t-1} + \mathbf{b}),$$

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.

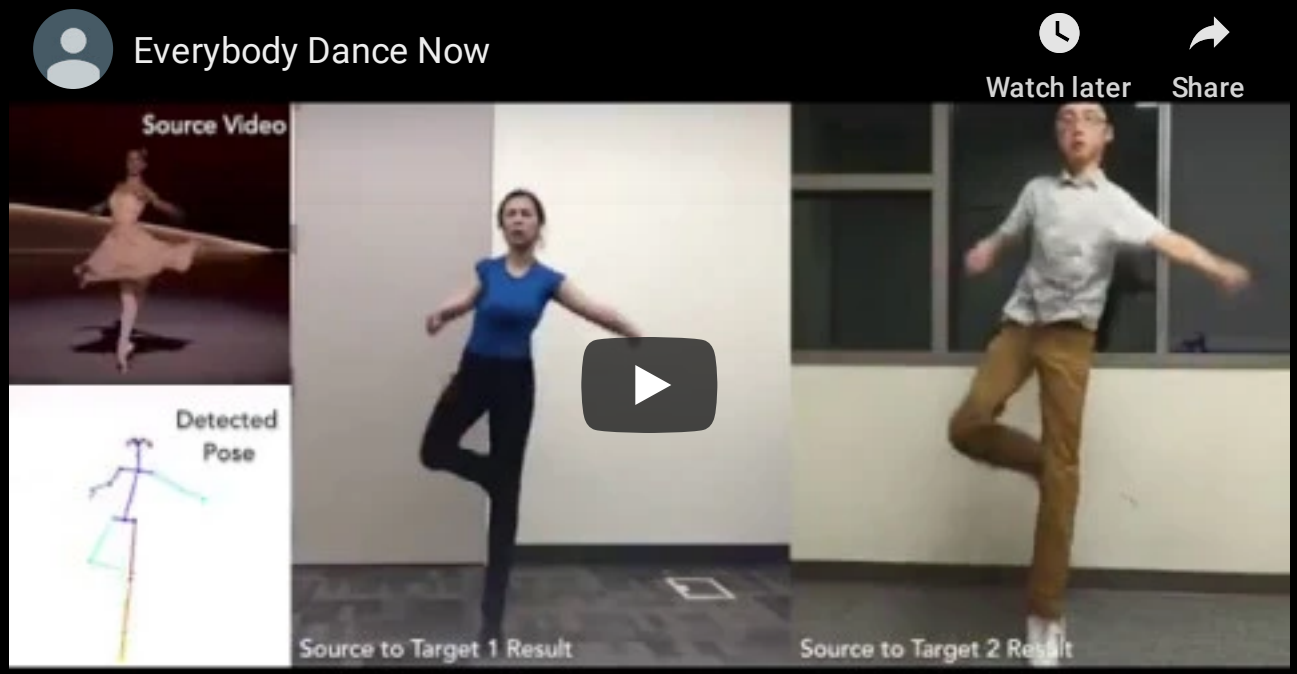
Applications

Neural networks are now at the core of many state-of-the-art systems, including:

- Image recognition
- Speech recognition and synthesis
- Natural language processing
- Scientific studies
- Reinforcement learning
- Autonomous agents

... and many many many others.

Autonomous drone navigation with deep learning



Pose detection and video synthesis

Deep learning and AI at NVIDIA

Unsupervised learning

Unsupervised learning

- Most of the learning performed by animals and humans is unsupervised.
  - Without labeled examples nor rewards.
- We learn how the world works by observing it:
  - We learn that the world is 3-dimensional.
  - We learn that objects can move independently of each other.
  - We learn object permanence.
  - We learn to predict what the world will look one second or one hour from now.

Credits: Yann Lecun (NYU), Deep Learning, 2017

Common sense

We build a model of the world through predictive unsupervised learning.

- This predictive model gives us common sense.
- Unsupervised learning discovers regularities in the world.

If I say: "Bernard picks up his bag and leaves the room".

You can infer:

- Bernard stood up, extended his arm to pick the bag, walked towards the door, opened the door, walked out.
- He and his bag are not in the room anymore.
- He probably did not dematerialized or flied out.

How do we do that?

We have no clue! (mostly)

Summary

- Learning is (supposedly) a key element of intelligence.
- Statistical learning aims at learning probabilistic models (their parameters or structures) automatically from data.
- 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.
- Reinforcement learning = learning to behave in an unknown environment from sparse rewards.
- Unsupervised learning = learning a model of the world by observing it.

The end.

Going further?

- ELEN0062: Introduction to Machine Learning
- INFO8004: Advanced Machine Learning
- INFO8010: Deep Learning