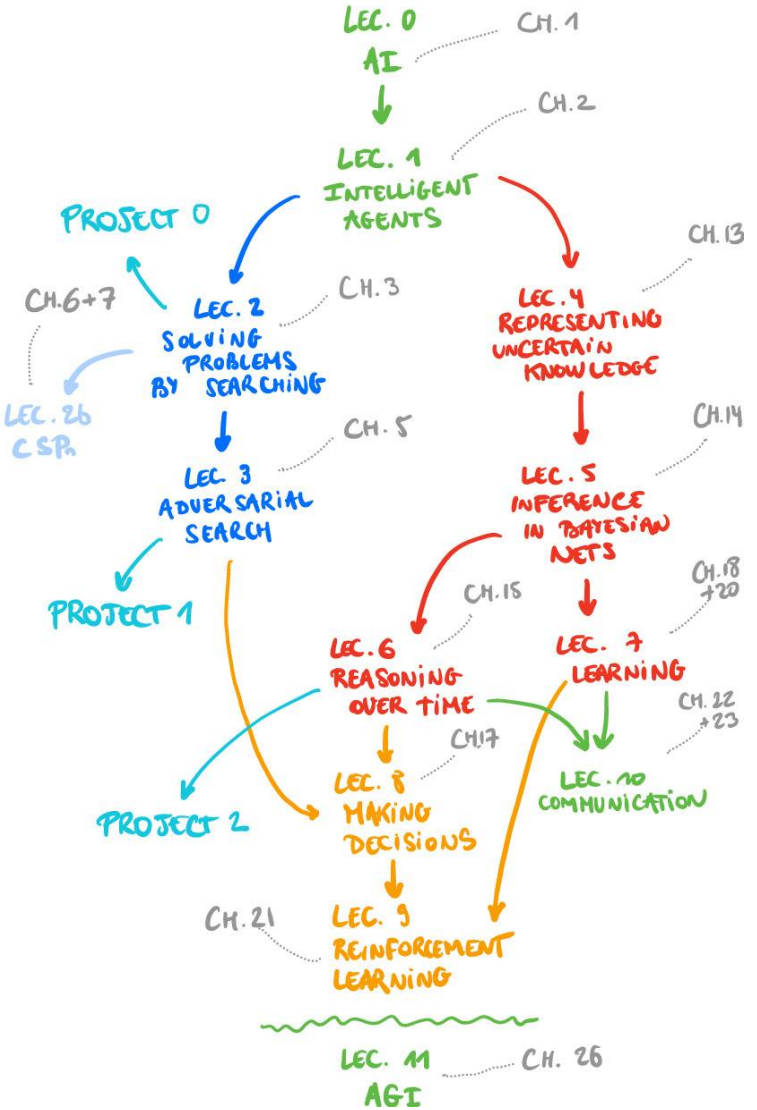
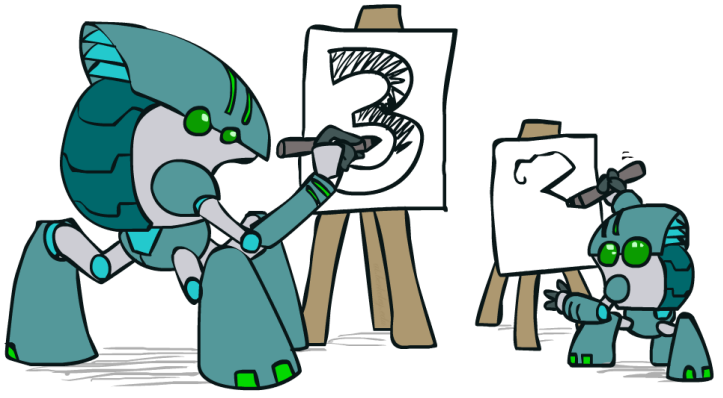
Introduction to Artificial Intelligence

Lecture 7: Machine learning and neural networks

Prof. Gilles Louppe
g.louppe@uliege.be


LEC. 0
AI
CH. 1
LEC. 1
INTELLIGENT AGENTS
CH. 2
PROJECT 0
CH. 13
CH. 6+7
LEC. 2
SOLVING PROBLEMS BY SEARCHING
CH. 3
LEC. 4
REPRESENTING UNCERTAIN KNOWLEDGE
LEC. 26
CSP
CH. 5
CH. 14
LEC. 3
ADVERSARIAL SEARCH
LEC. 5
INFERENCE IN BAYESIAN NETS
PROJECT 1
CH. 15
CH. 18 +20
LEC. 6
REASONING OVER TIME
LEC. 7
LEARNING
CH. 22 +23
CH. 17
PROJECT 2
LEC. 8
MAKING DECISIONS
LEC. 10
COMMUNICATION
CH. 21
LEC. 9
REINFORCEMENT LEARNING
LEC. 11
AGI
CH. 26

Today



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

- Bayesian learning
- Supervised learning

—

Image credits: CS188, UC Berkeley.

Intelligence?

What we cover in this course:

- Search algorithms, using a state space specified by domain knowledge.
- 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.

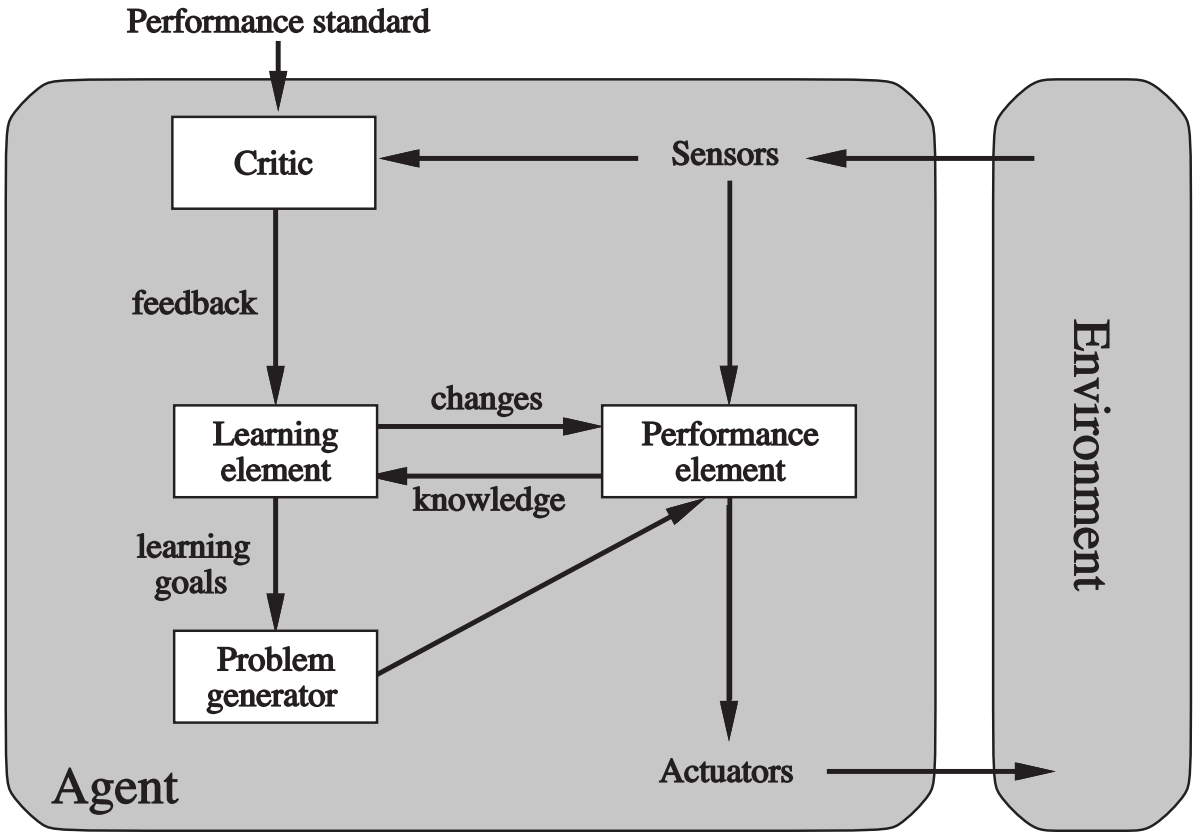
Sufficient to implement complex and rational behaviors, in some situations.

Aren't we missing something?

Learning agents

What if the environment is unknown?

- Learning can be used as a system construction method.
- Expose the agent to reality rather than 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

Bayesian learning

Bayesian learning

Frame learning as a Bayesian update of a probability distribution $\mathbf{P}(H)$ over a hypothesis space, where

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

Predictions use a likelihood-weighted average over the hypotheses:

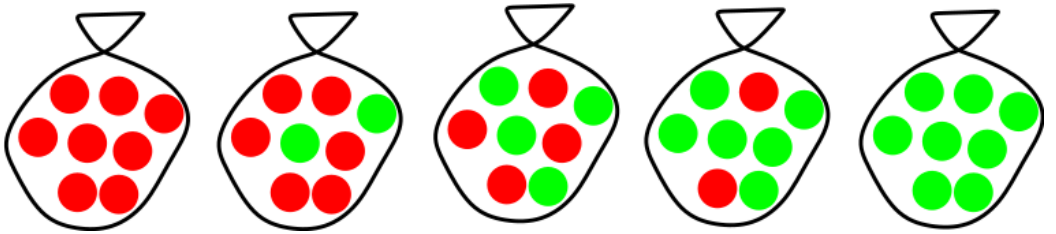
$$P(X|\mathbf{d}) = \sum_i P(X|\mathbf{d}, h_i)P(h_i|\mathbf{d}) = \sum_i P(X|h_i)P(h_i|\mathbf{d})$$

No need to pick one best-guess hypothesis!

Example

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

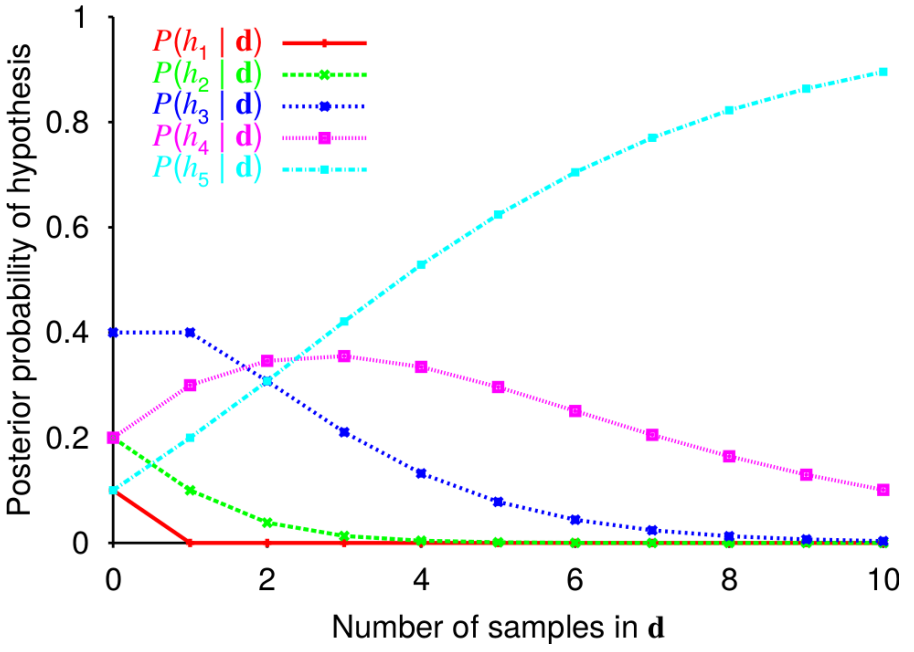
- $P(h_1) = 0.1$, with h_1: 100% cherry candies
- $P(h_2) = 0.2$, with h_2: 75% cherry candies + 25% lime candies
- $P(h_3) = 0.4$, with h_3: 50% cherry candies + 50% lime candies
- $P(h_4) = 0.2$, with h_4: 25% cherry candies + 75% lime candies
- $P(h_5) = 0.1$, with h_5: 100% lime candies



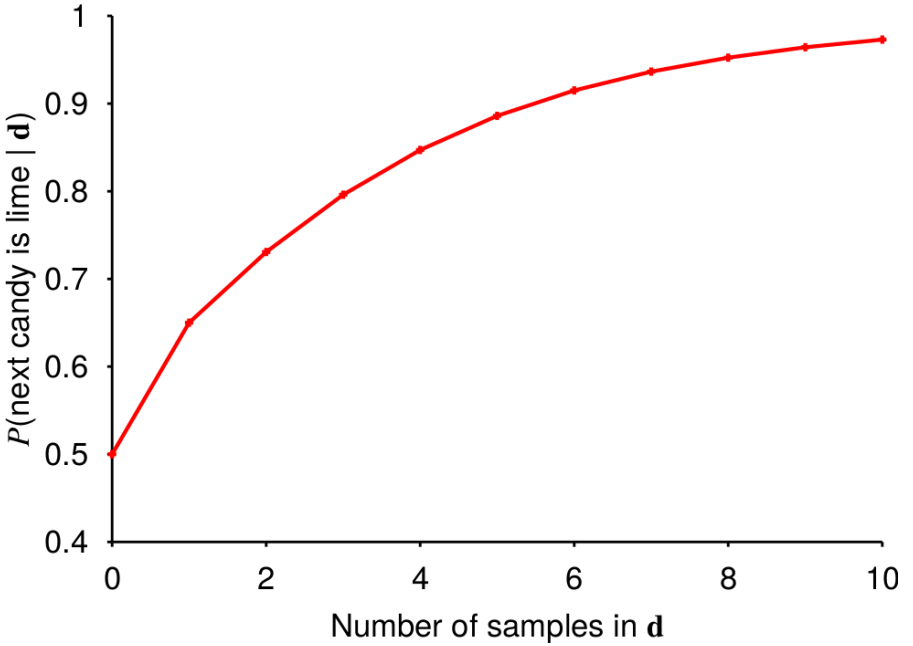
Then we observe candies drawn from some bag:

- 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 (MAP) estimation consists in using the hypothesis

$$\begin{aligned} h_{\text{MAP}} &= \arg\max_{h_i} P(h_i|\mathbf{d}) \\ &= \arg\max_{h_i} P(\mathbf{d}|h_i)P(h_i) \\ &= \arg\max_{h_i} \log P(\mathbf{d}|h_i) + \log P(h_i) \end{aligned}$$

- Log terms can be be viewed as (the 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 $\mathbf{P}(H)$ becomes irrelevant.

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

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

- 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).
- Write down the log-likelihood L of the parameters θ.
- Write down the derivative of the log likelihood of the parameters θ.
- Find the parameter values θ^* such that the derivatives are zero and check whether the Hessian is negative definite.

Parameter estimation in Bayesian networks

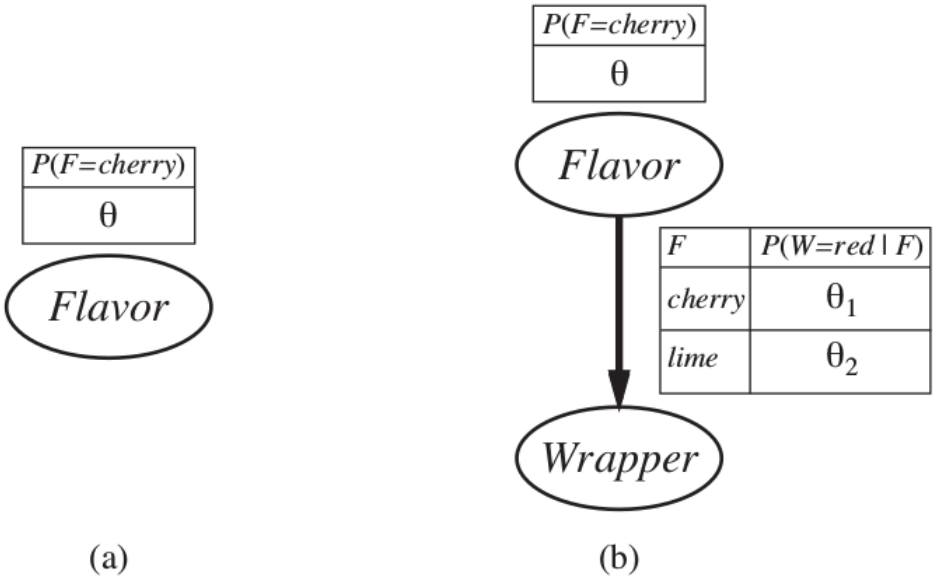


Figure 20.2 (a) Bayesian network model for the case of candies with an unknown proportion of cherries and limes. (b) Model for the case where the wrapper color depends (probabilistically) on the candy flavor.

MLE, case (a)

What is the fraction θ of cherry candies?

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

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

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

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

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

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

MLE, case (b)

Red and green wrappers depend probabilistically on flavor. E.g., the likelihood for a cherry candy in green wrapper:

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

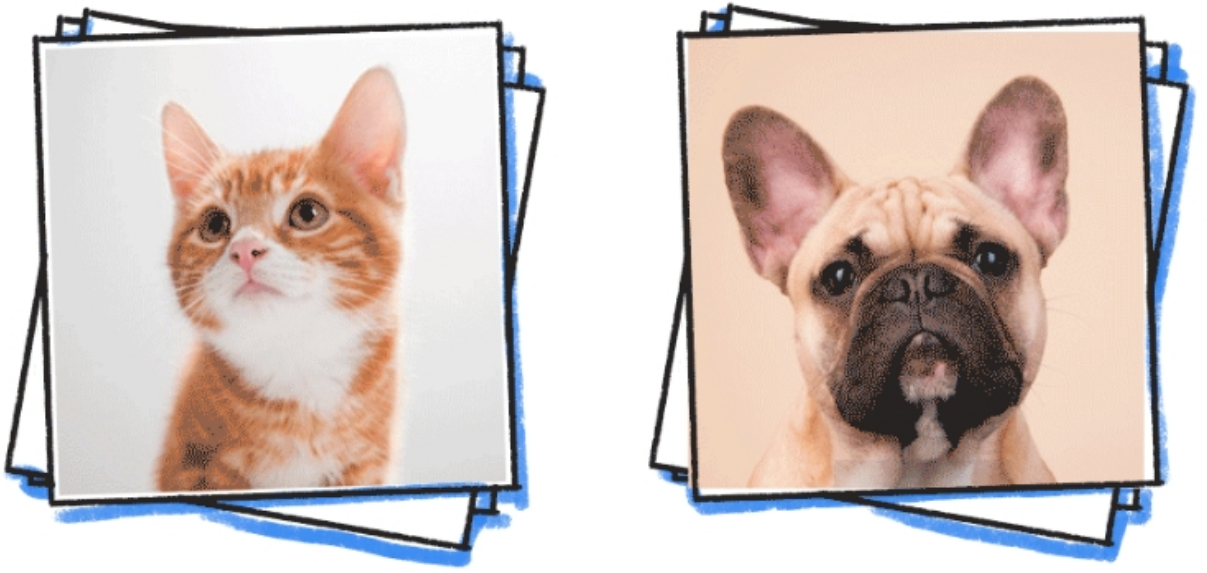
The likelihood for the data, given N candies, r_c red-wrapped cherries, g_c green-wrapped cherries, etc., is:

$$\begin{aligned}
P(\mathbf{d}|h_{\theta,\theta_1,\theta_2}) &= \theta^c(1-\theta)^l\theta_1^{r_c}(1-\theta_1)^{g_c}\theta_2^{r_l}(1-\theta_2)^{g_l} \\
L &= c\log\theta + l\log(1-\theta) + \\
&\quad r_c\log\theta_1 + g_c\log(1-\theta_1) + \\
&\quad r_l\log\theta_2 + g_l\log(1-\theta_2)
\end{aligned}$$

Derivatives of L contain only the relevant parameter:

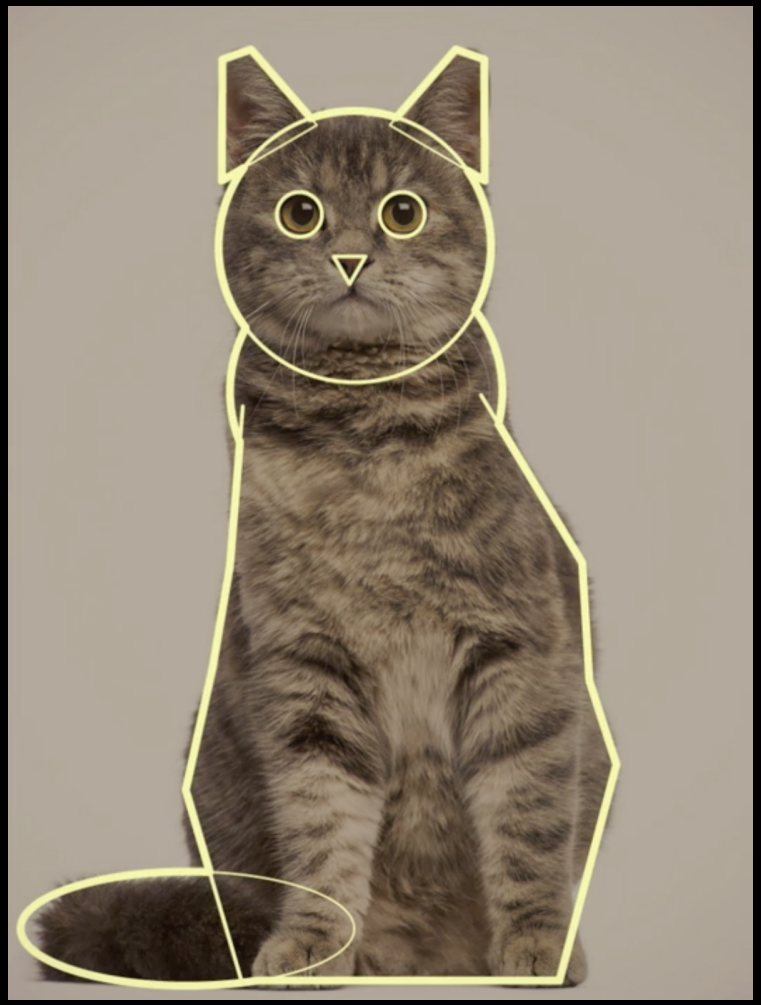
$$\frac{\partial L}{\partial \theta} = \frac{c}{\theta} - \frac{l}{1-\theta} = 0 \Rightarrow \theta = \frac{c}{c+l}$$
$$\frac{\partial L}{\partial \theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 \Rightarrow \theta_1 = \frac{r_c}{r_c+g_c}$$
$$\frac{\partial L}{\partial \theta_2} = \frac{r_l}{\theta_2}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

Machine learning

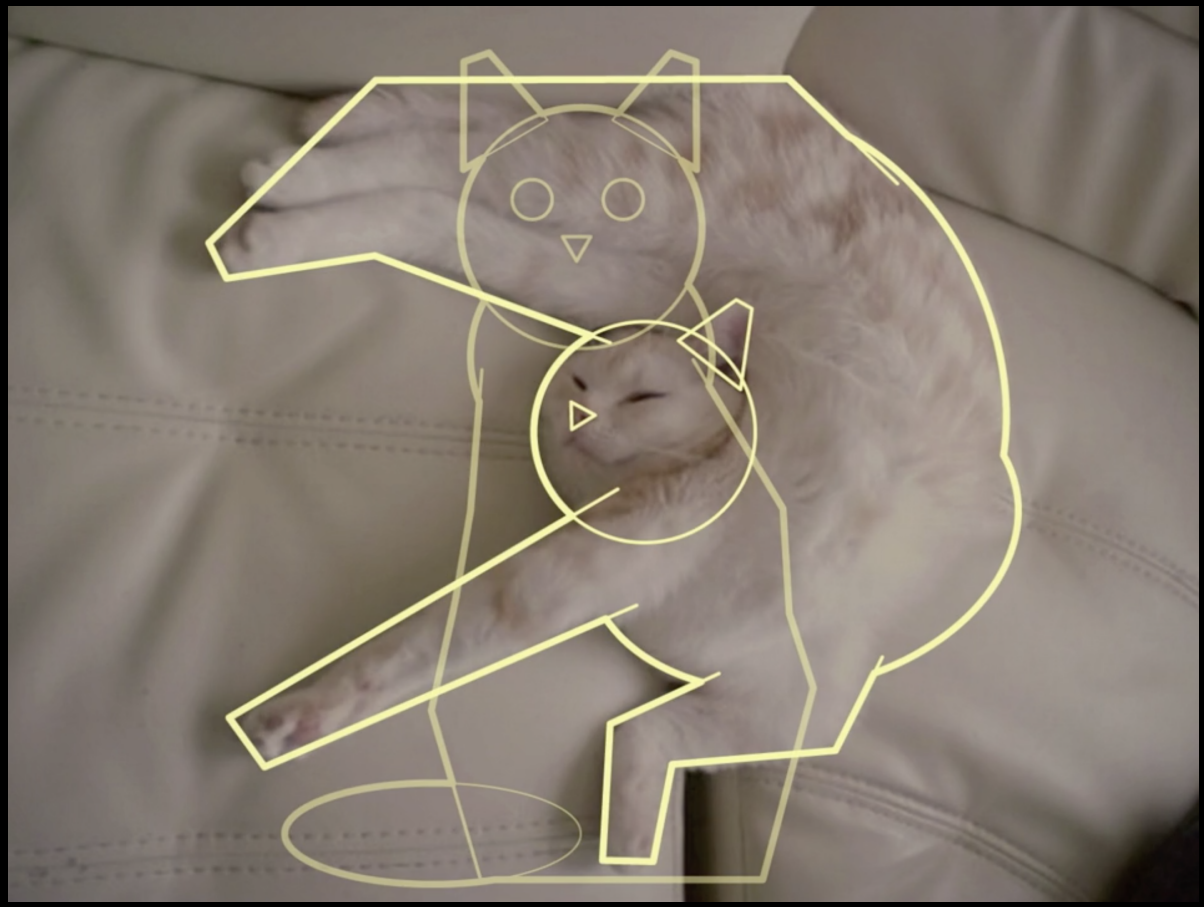


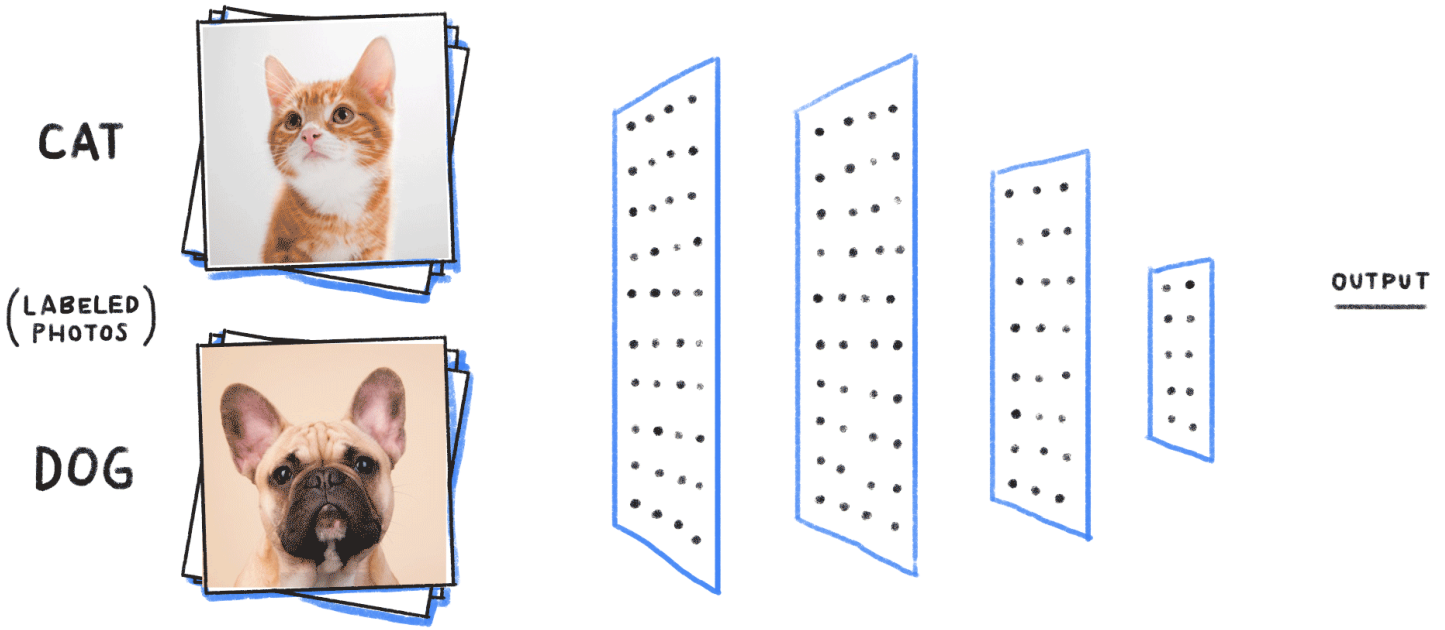
Exercise

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



The good old-fashioned approach.





The deep learning approach.

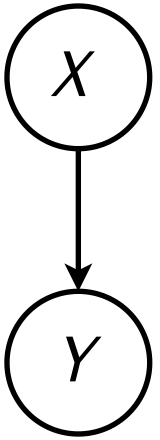
Problem statement

Let us assume data $\mathbf{d} \sim p(\mathbf{x}, y)$ 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 want to find a function $h \in \mathcal{H}$ that approximates the true function f.

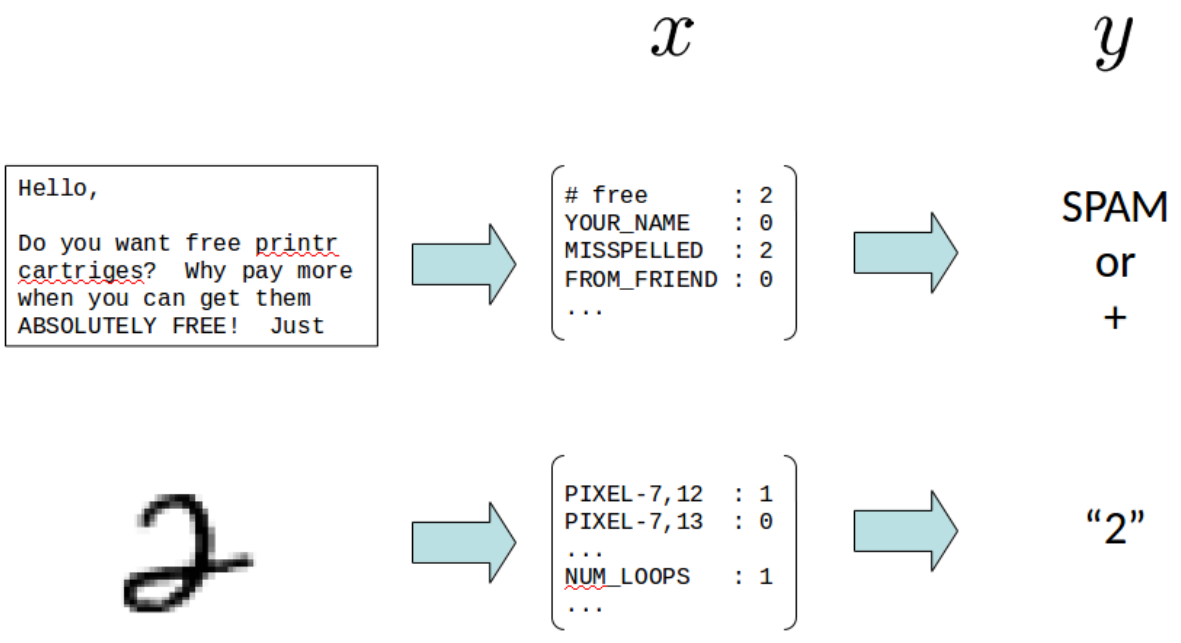


In general, f will be stochastic. In this case, y is not strictly a function x, and we wish to learn the conditional $p(y|\mathbf{x})$.

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

Feature vectors

- Input samples $\mathbf{x} \in \mathbb{R}^d$ 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.

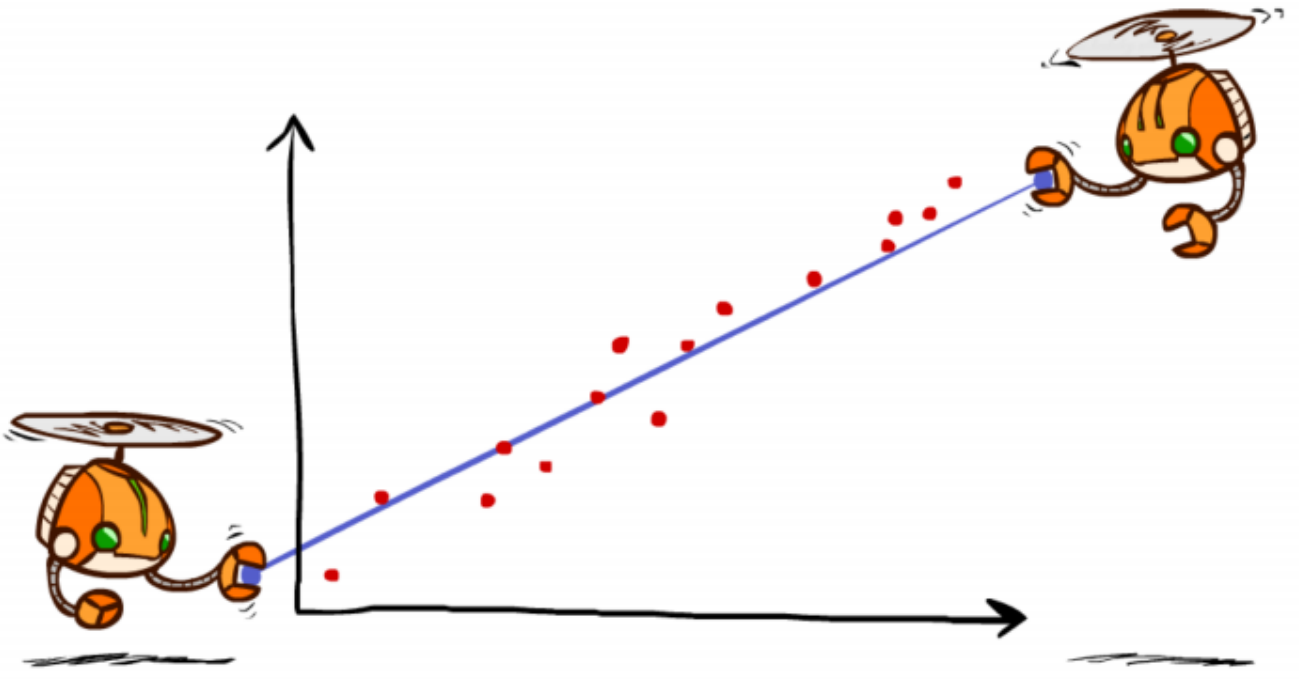


—

Image credits: CS188, UC Berkeley.

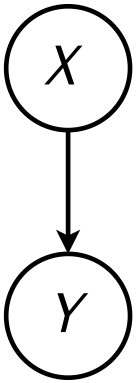
Linear regression

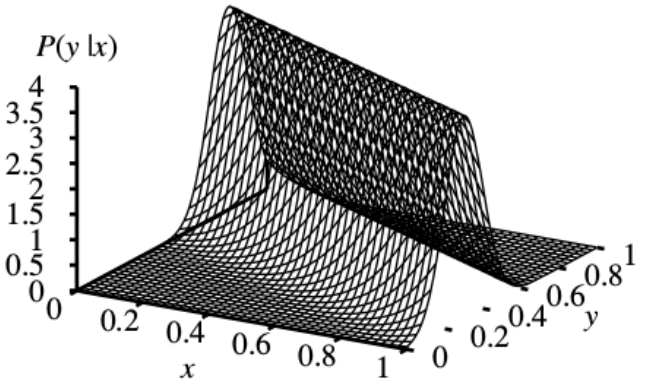
Let us first assume that $y \in \mathbb{R}$.



—

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

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

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

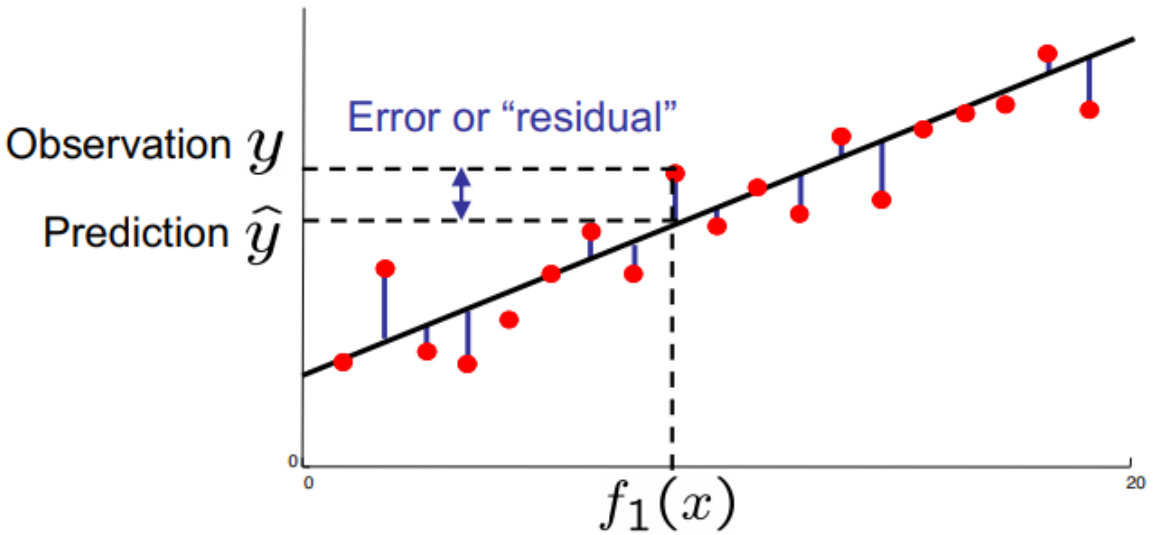
$$p(y|\mathbf{x}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y - (\mathbf{w}^T\mathbf{x} + b))^2}{2\sigma^2}\right)$$

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

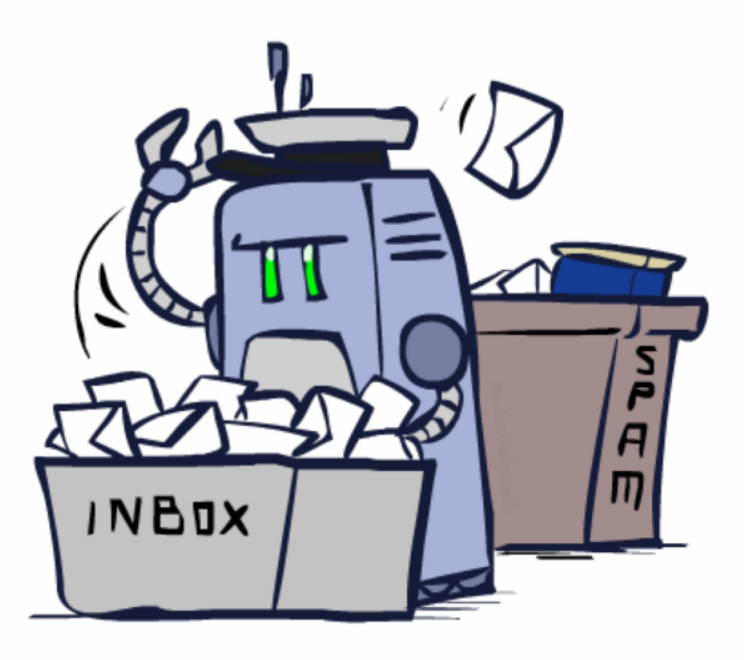
$$\sum_{j=1}^{N}(y_j - (\mathbf{w}^T\mathbf{x}_j + b))^2.$$

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 $y \in \{0, 1\}$.



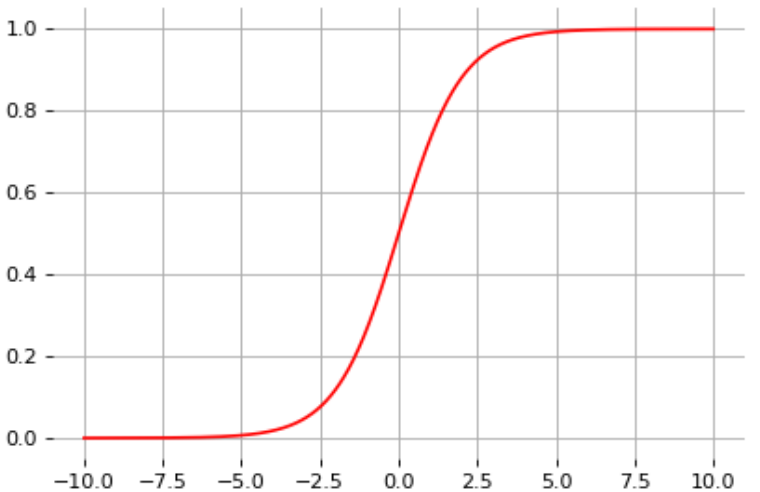
—

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

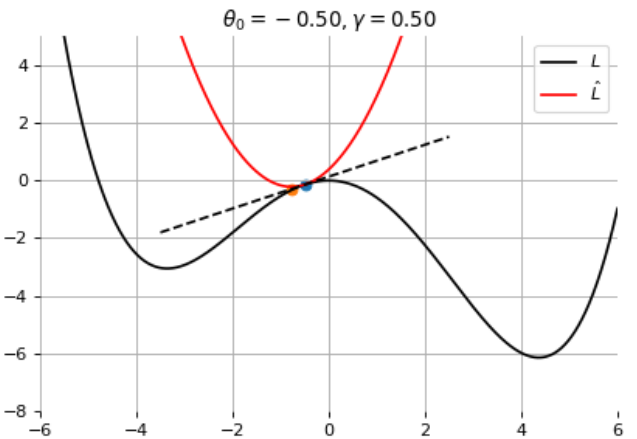
Gradient descent

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

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

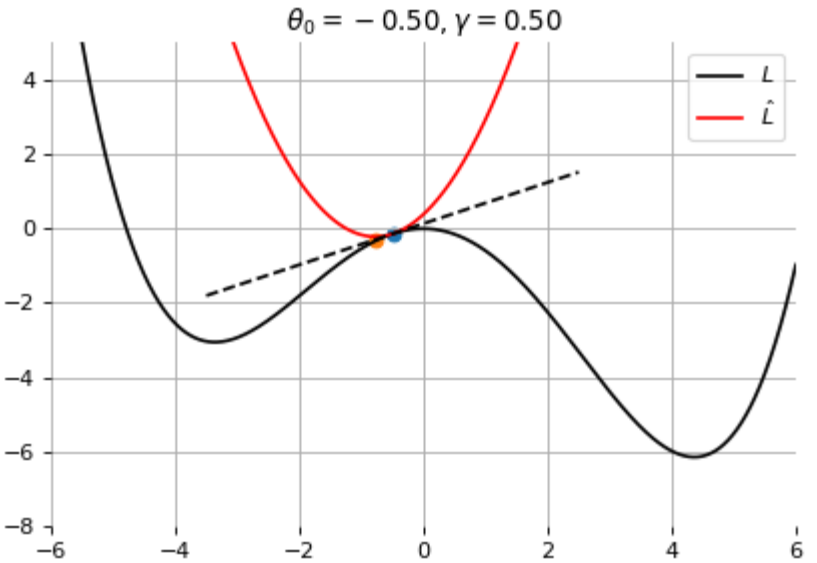
which results in the best improvement for the step $\epsilon = -\gamma \nabla_\theta \mathcal{L}(\theta_0)$.

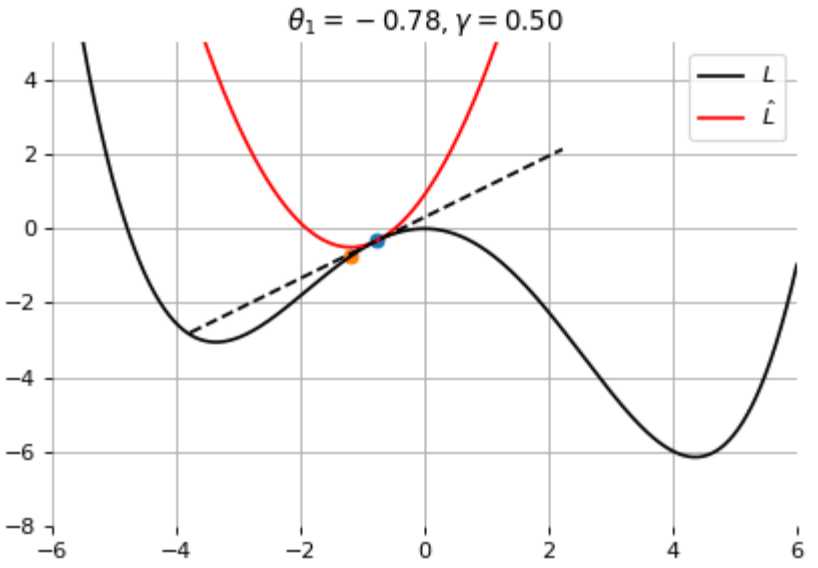
Therefore, model parameters can be updated iteratively using the update rule

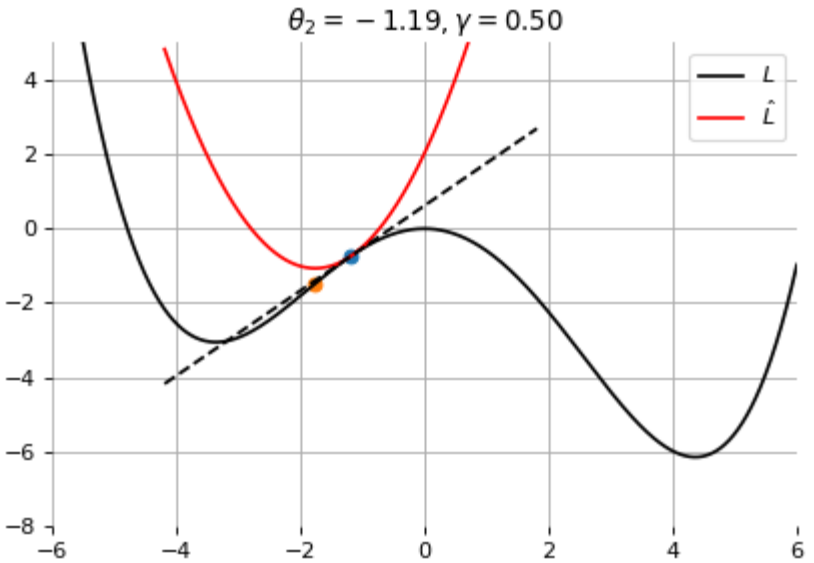
$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t),$$

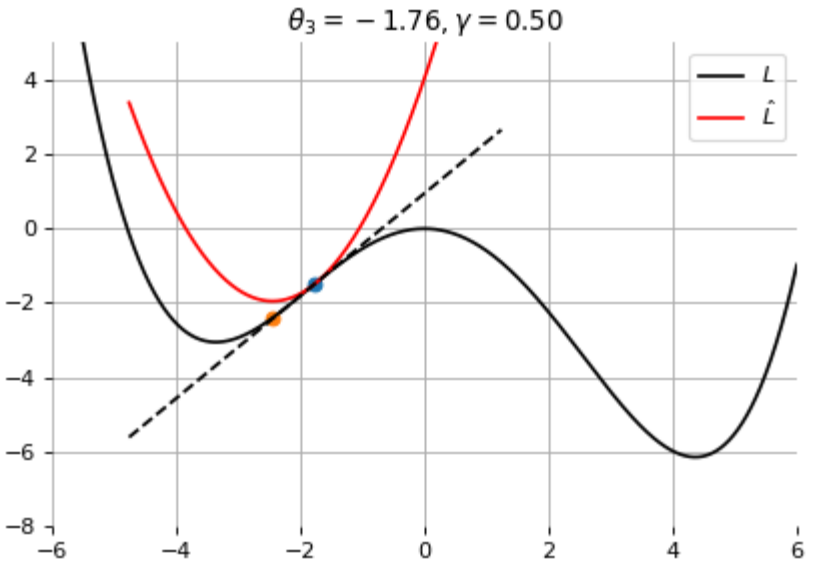
where

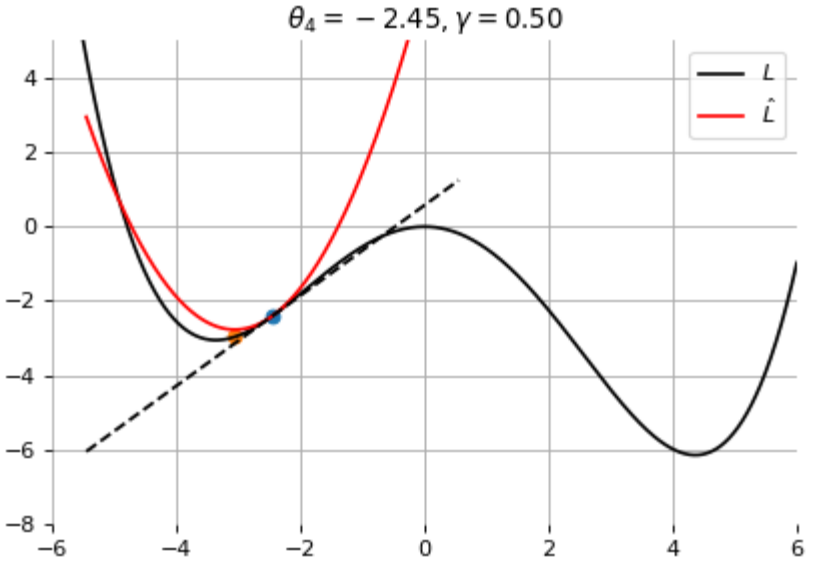
- θ_0 are the initial parameters of the model,
- γ is the learning rate.

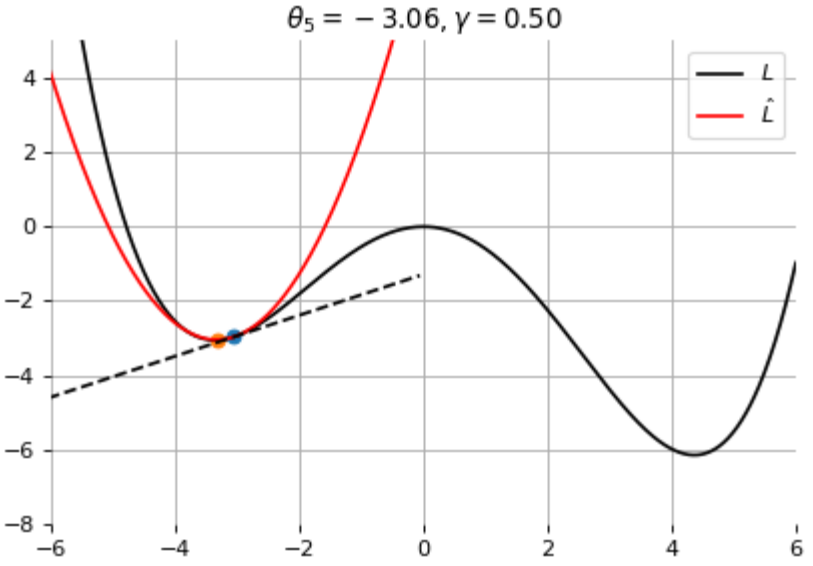

$\theta_0 = -0.50, \gamma = 0.50$
L
$\hat{L}$
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6

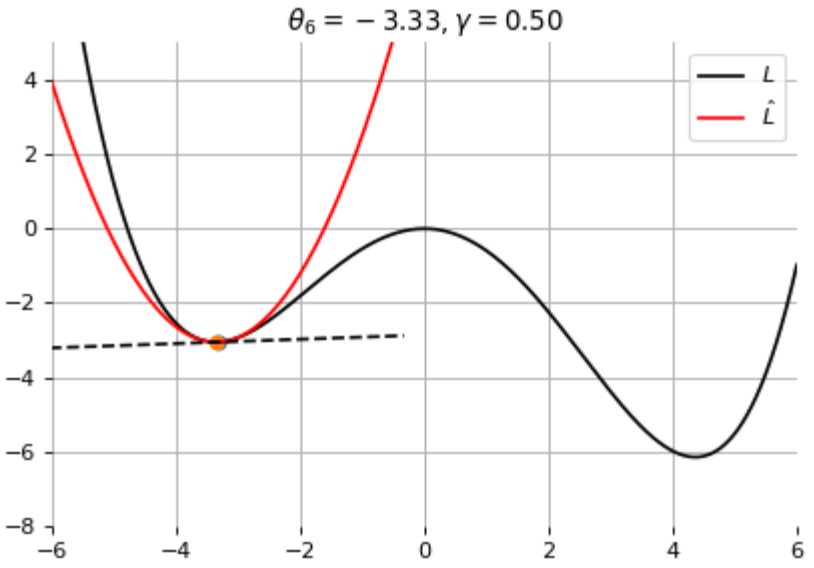

$\theta_1 = -0.78, \gamma = 0.50$
L
$\hat{L}$
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6

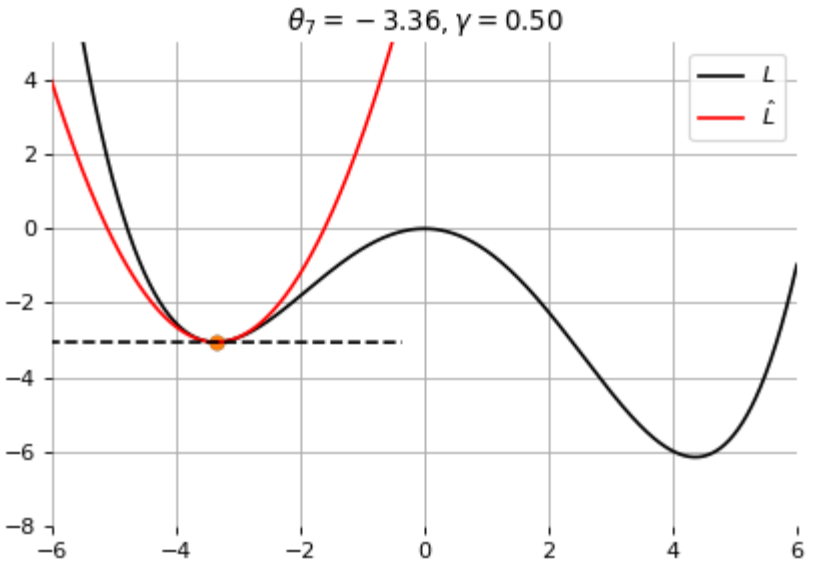

$\theta_2 = -1.19, \gamma = 0.50$
L
$\hat{L}$
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6


$\theta_3 = -1.76, \gamma = 0.50$
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

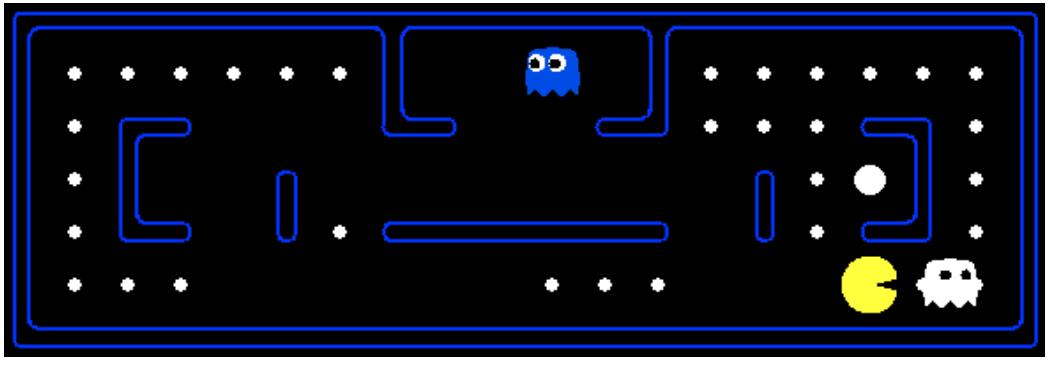

$\theta_6 = -3.33, \gamma = 0.50$
L
$\hat{L}$
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6


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

Apprenticeship

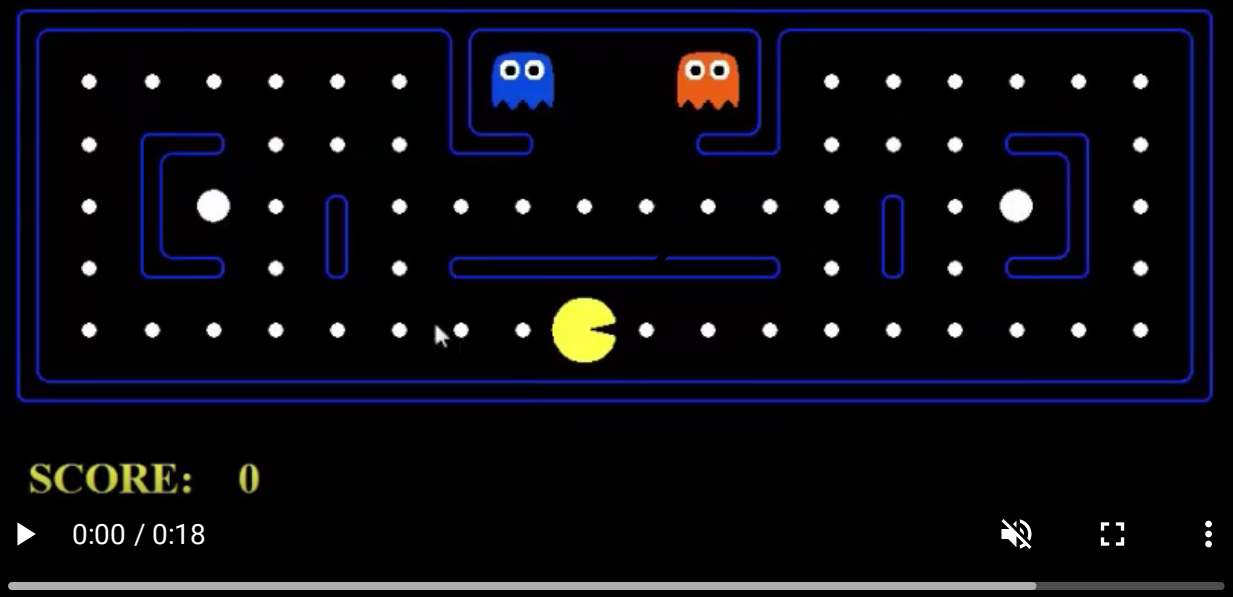
Can we learn to play Pacman only from observations?

- 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.



—

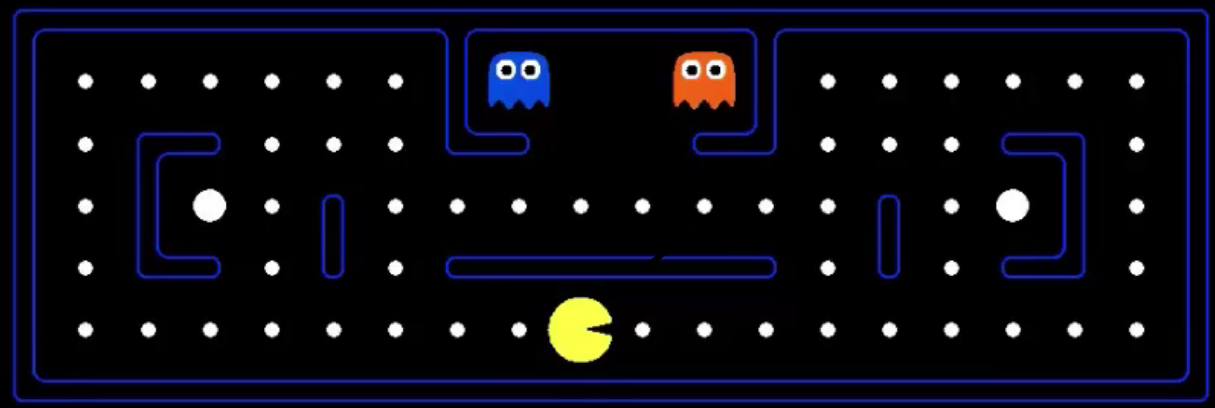
Image credits: CS188, UC Berkeley.



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

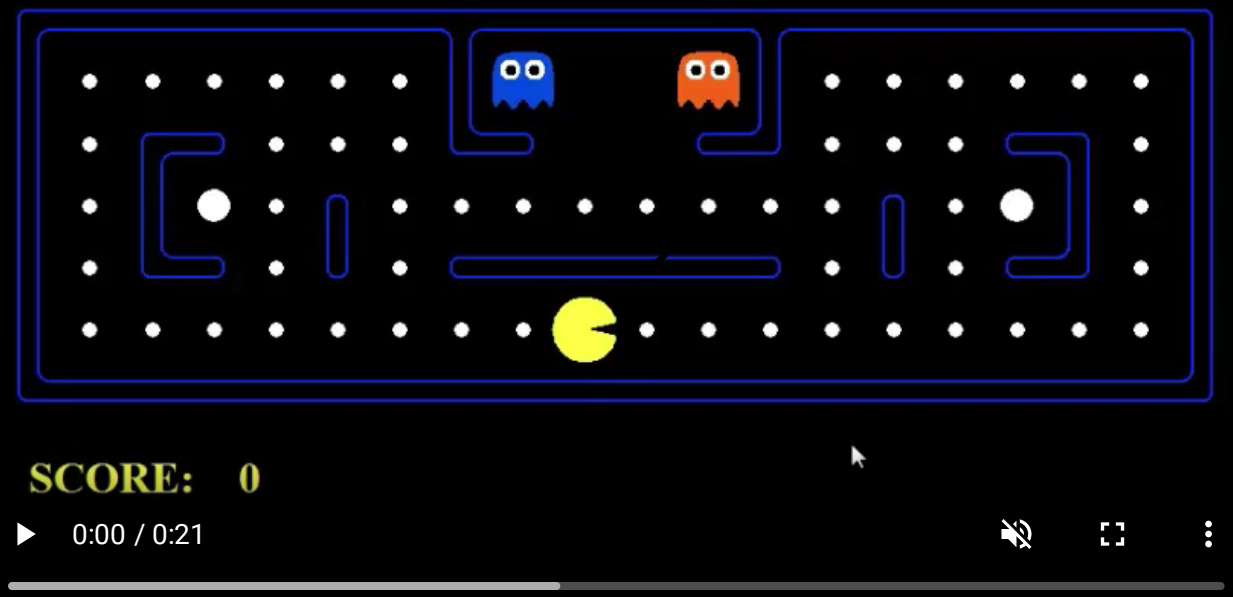
—

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)

Layers

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

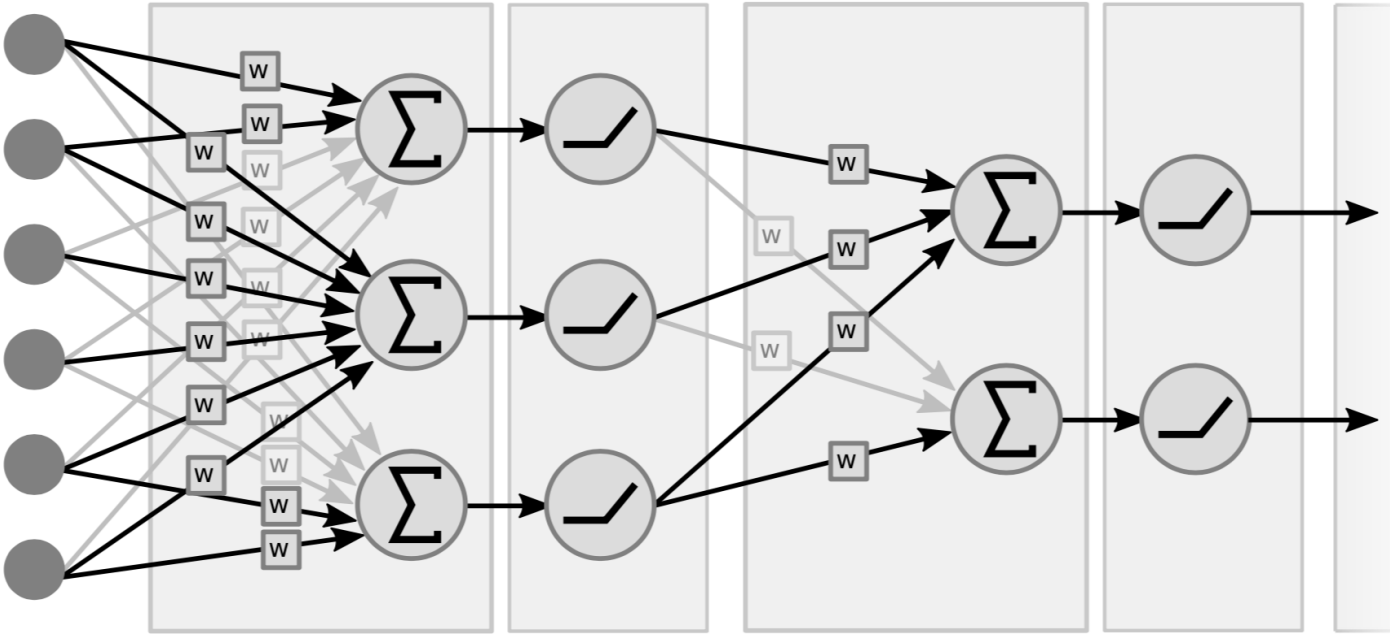
Multi-layer perceptron

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 $\{\mathbf{W}_k, \mathbf{b}_k, ... | k = 1, ..., L\}$ and can be determined through gradient descent.

- 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 $\hat{y} \in \mathbb{R}$.



—

Credits: PyTorch Deep Learning Minicourse, Alfredo Canziani, 2020.

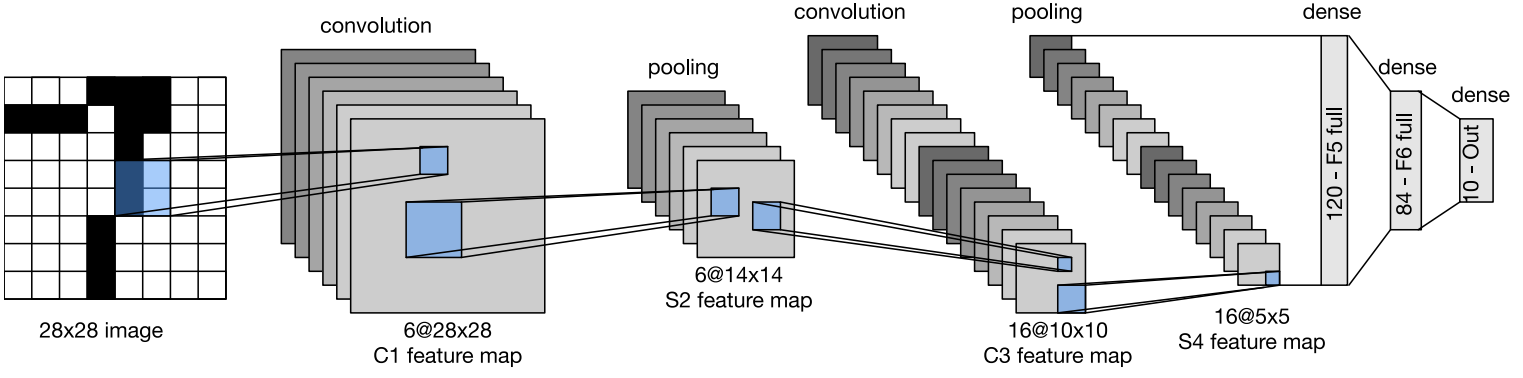
(demo)

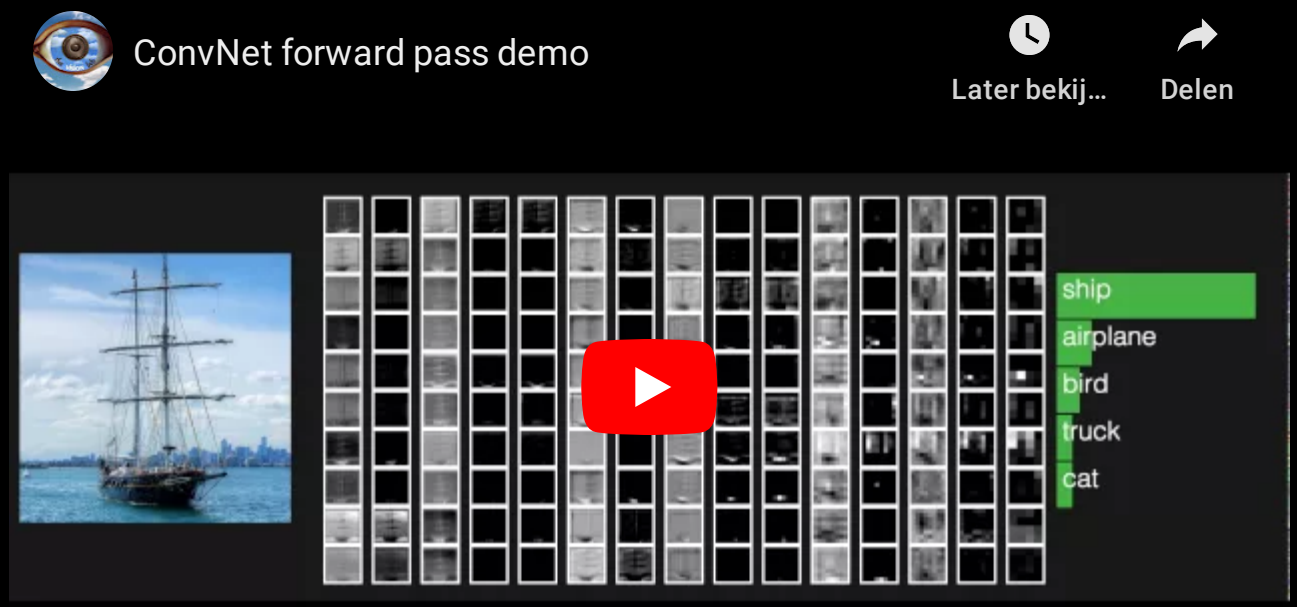
Convolutional networks

Convolutional neural networks extend fully connected architectures with

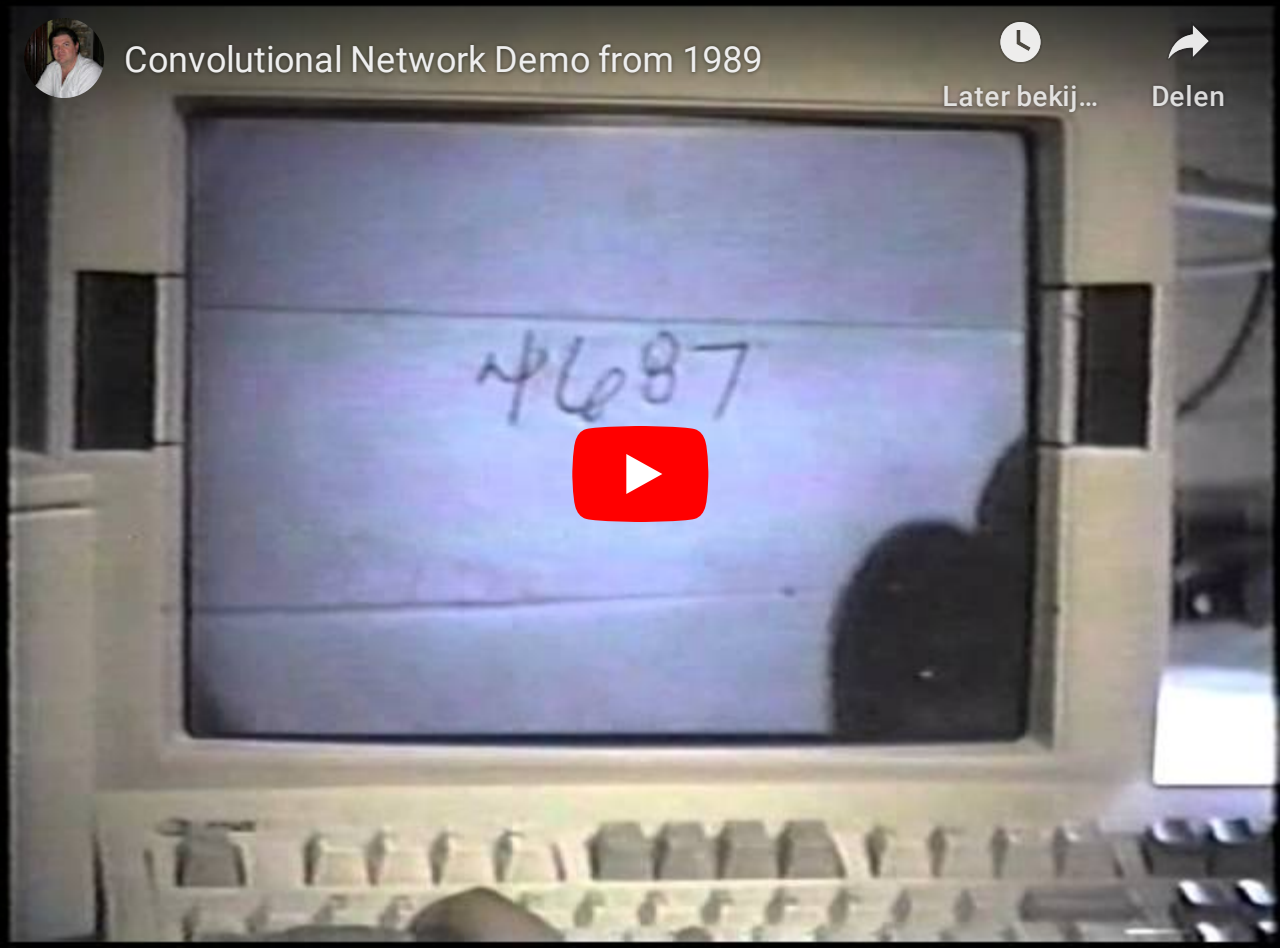
- convolutional layers: cross-correlation of the input through learnable kernels.
- pooling layers: reduce the input dimension by pooling (e.g., averaging) clusters of input values.

These are specifically designed for processing spatially structured data (e.g., images, sequences) with known shift invariance.





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.

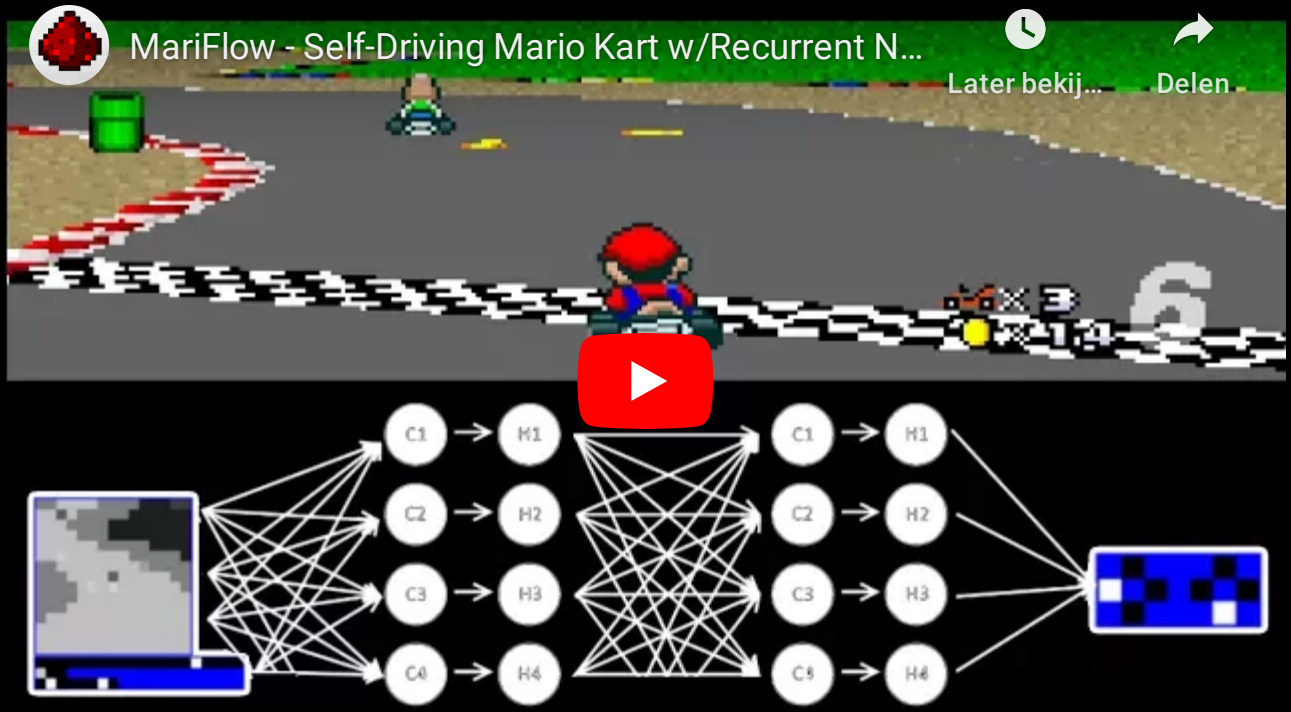
For example,

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

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.

Deep Learning as an architectural language

*People are now building a new kind of software by **assembling networks of parameterized functional blocks** and by **training them from examples using some form of gradient-based optimization**.*

Yann LeCun, 2018.

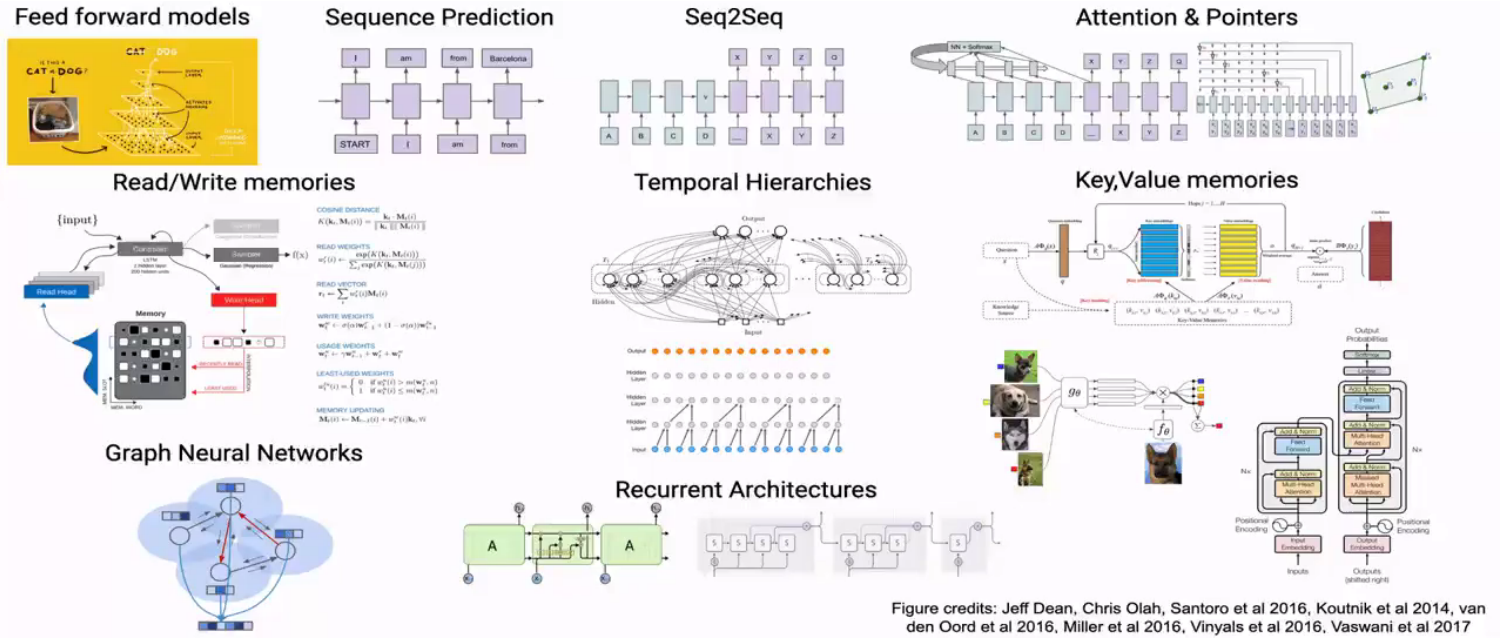
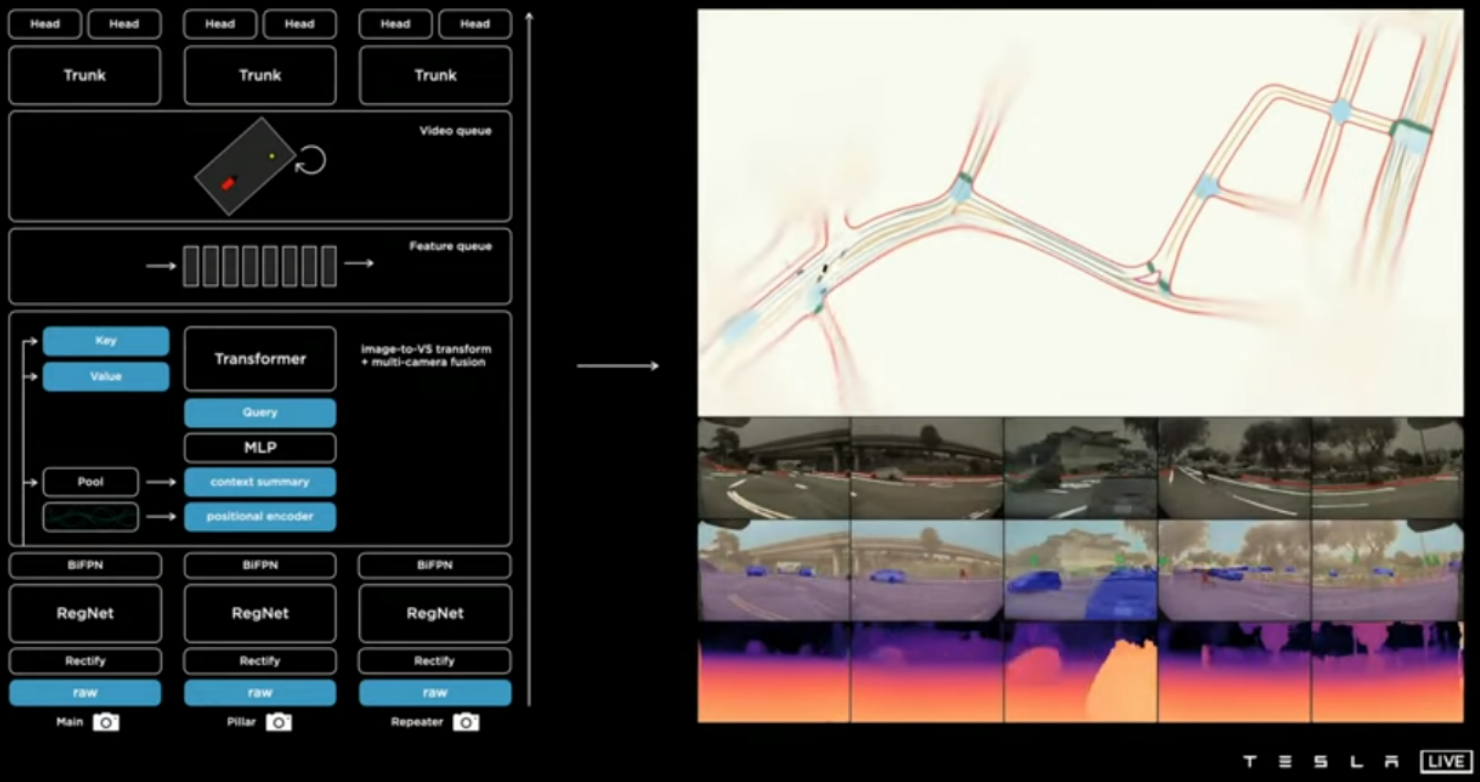


Figure credits: Jeff Dean, Chris Olah, Santoro et al 2016, Koutnik et al 2014, van den Oord et al 2016, Miller et al 2016, Vinyals et al 2016, Vaswani et al 2017

The toolbox

AI beyond Pacman

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



Hydranet (Tesla, 2021)

Solving impactful and challenging problems

Improving Tuberculosis Monitoring with Deep Learning

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.

The end.