# Introduction to Artificial Intelligence

Lecture 7: Machine learning and neural networks

Prof. Gilles Louppe

g.louppe@uliege.be

Today

Learning from data is a key component of artificial intelligence. In this lecture, we will introduce the principles of:

- Machine learning

- Neural networks
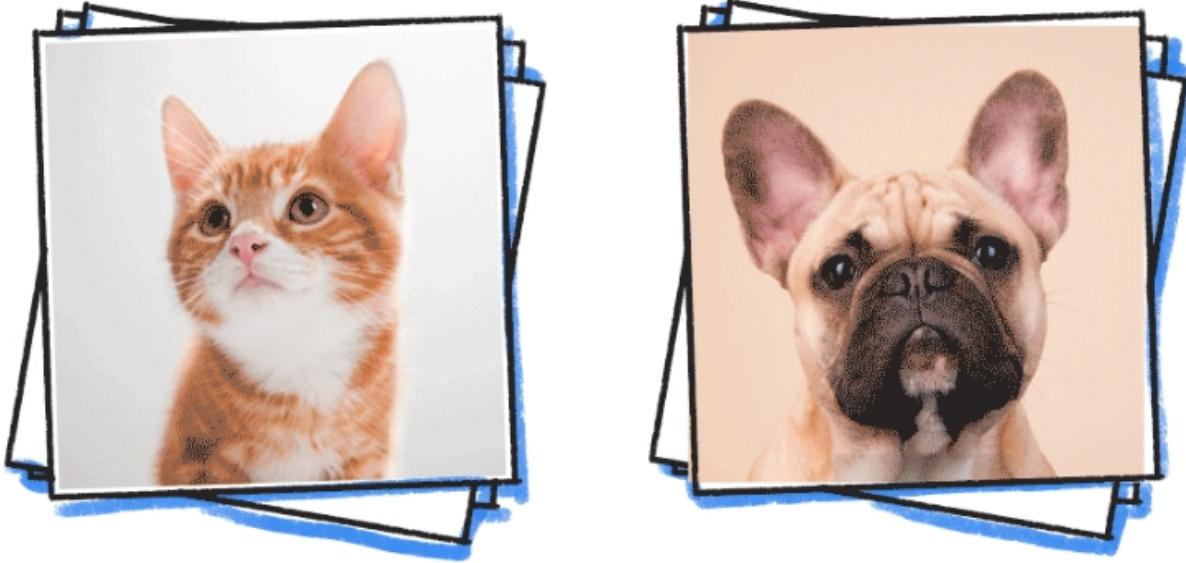
Learning agents

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.
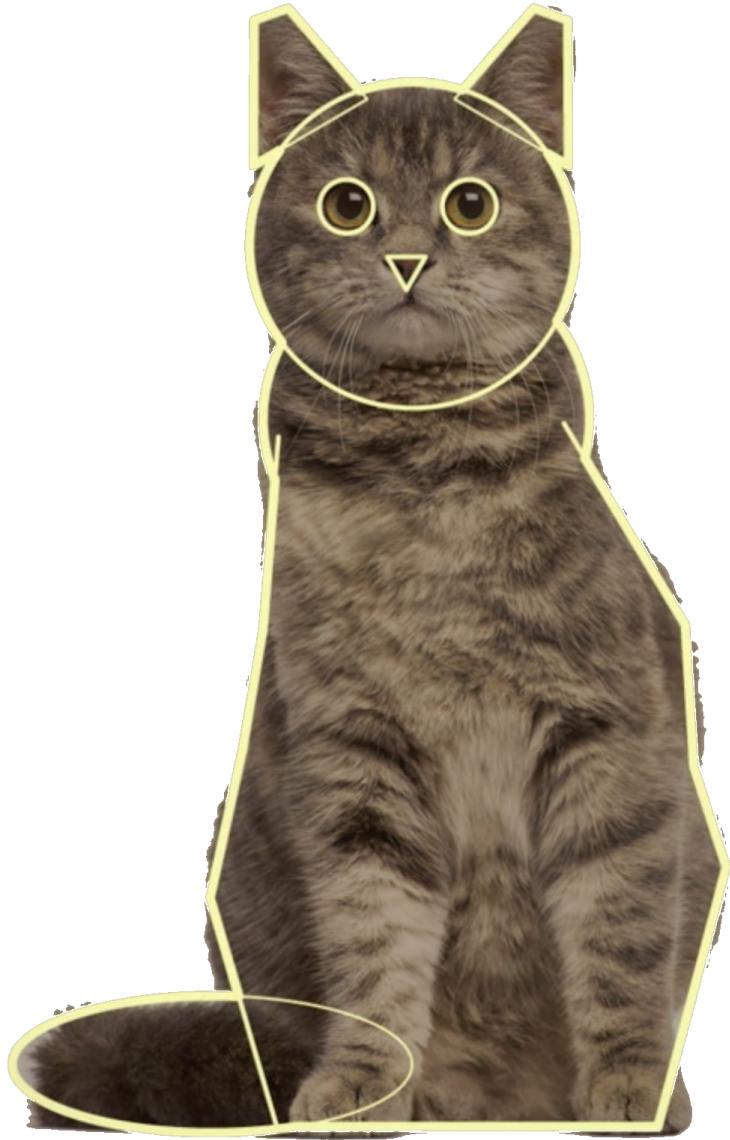
More generally, learning is useful for any task where it is difficult to write a program that performs the task but easy to obtain examples of desired behavior.

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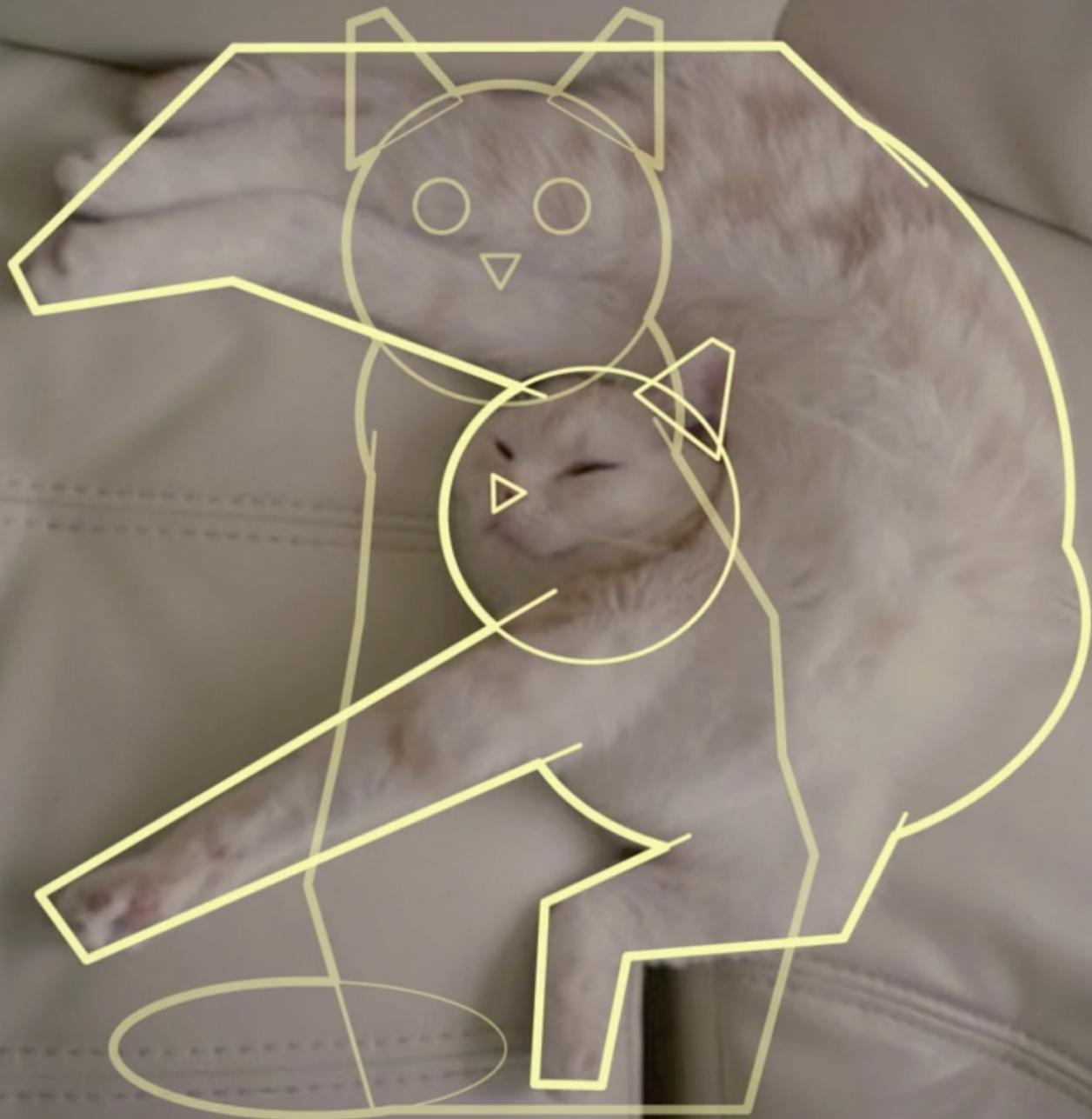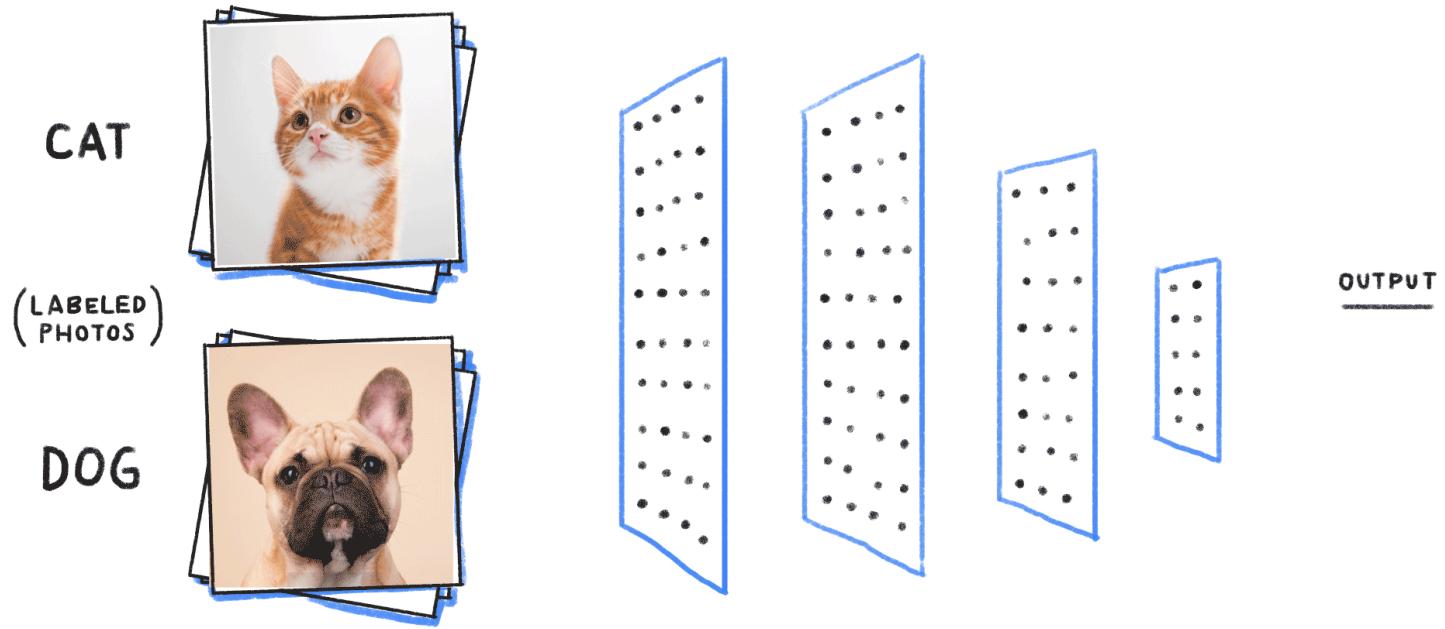
How would you write a computer program that recognizes cats from dogs?

CAT

(LABELED PHOTOS)

DOG

OUTPUT

The deep learning approach.

Problem statement

Let $\mathbf{d} \sim p(\mathbf{x}, y)$ be a dataset of N example input-output pairs
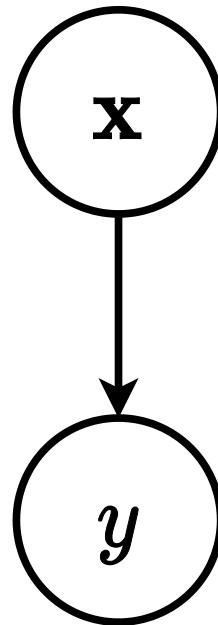
$$\mathbf{d} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N)\},$$
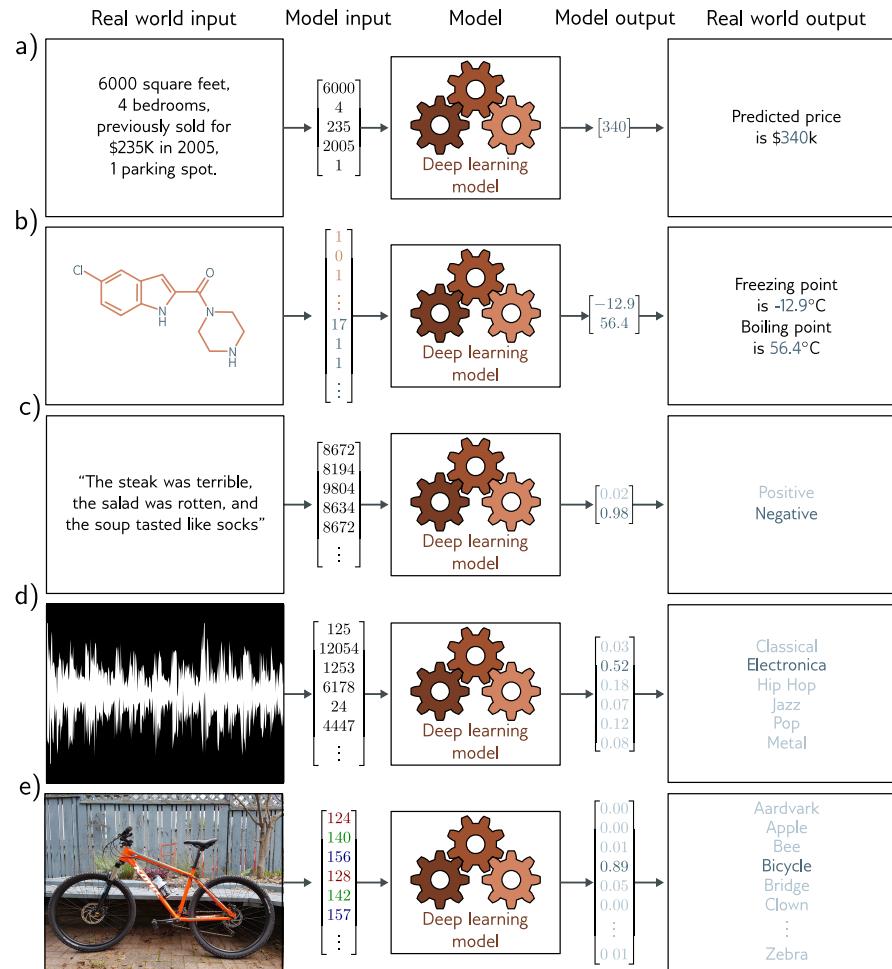
where $\mathbf{x}_i \in \mathbb{R}^d$ are d-dimensional vectors representing the input values and $y_i \in \mathcal{Y}$ are the corresponding output values.

From this data, we want to identify a probabilistic model
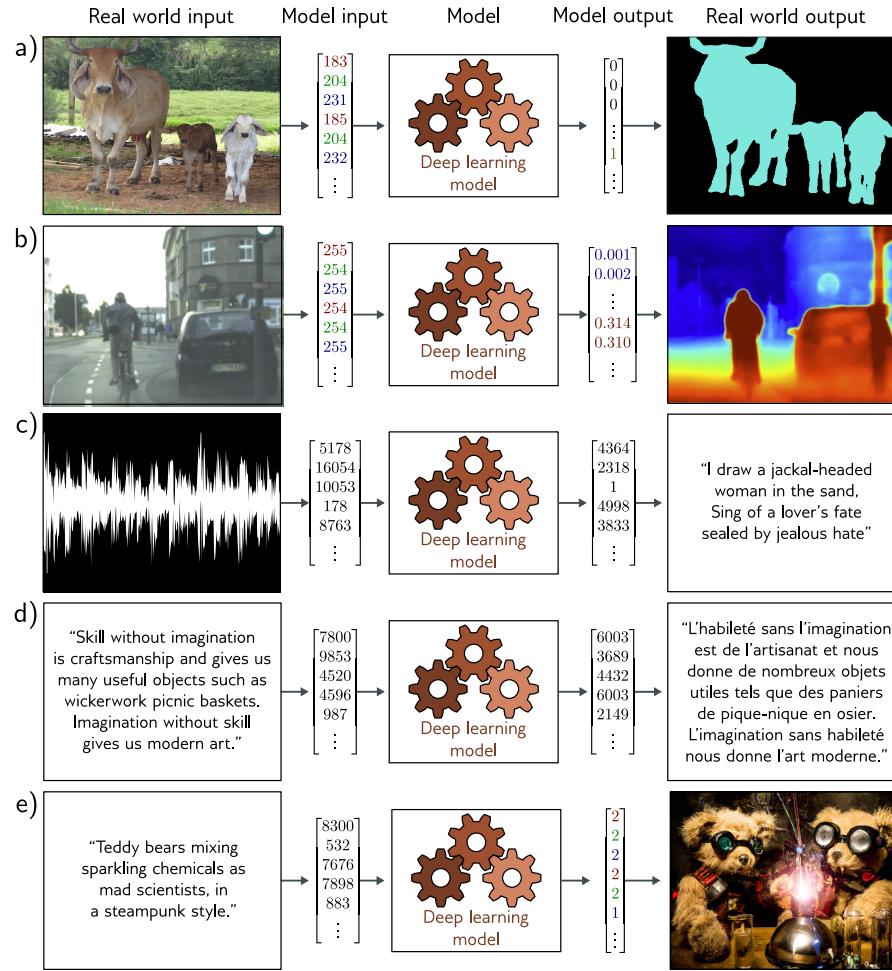
$$p_\theta(y|\mathbf{x})$$

that best explains the data.
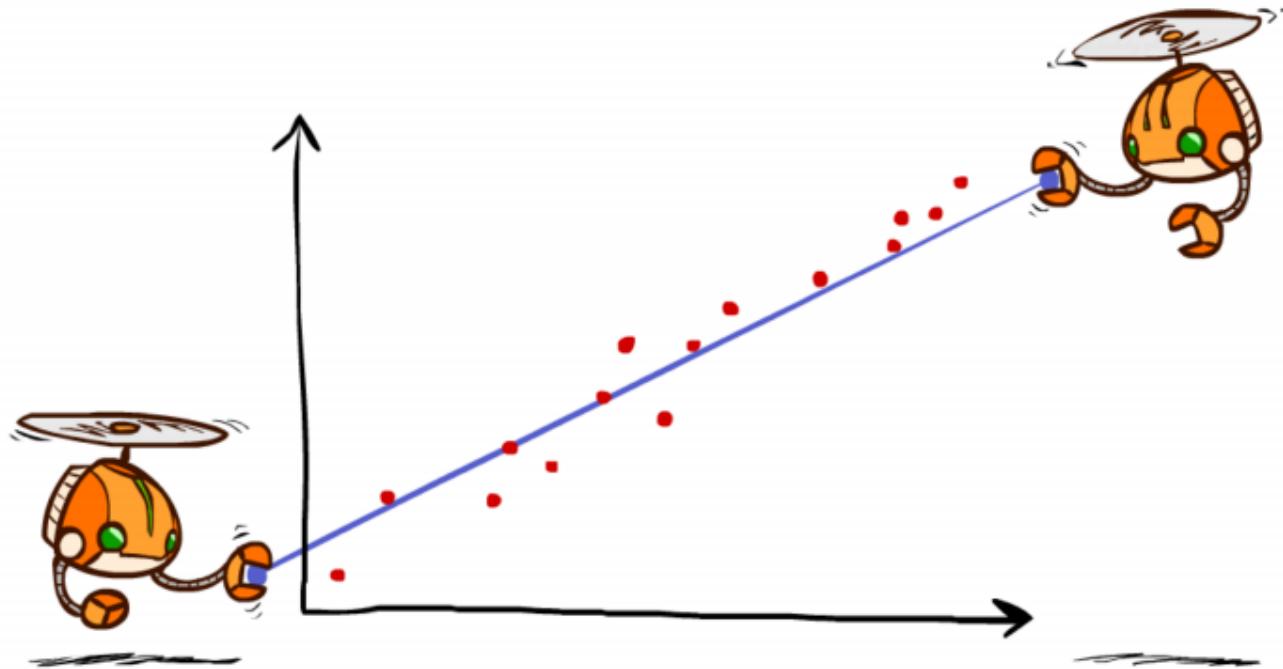
Regression ($y \in \mathbb{R}$) and classification ($y \in \{0, 1, ..., C-1\}$) problems.

Real world input   Model input   Model   Model output   Real world output

a)

b)

c)

"I draw a jackal-headed
woman in the sand,
Sing of a lover's fate
sealed by jealous hate"

d)

"Skill without imagination
is craftsmanship and gives us
many useful objects such as
wickerwork picnic baskets.
Imagination without skill
gives us modern art."

"L'habileté sans l'imagination
est de l'artisanat et nous
donne de nombreux objets
utiles tels que des paniers
de pique-nique en osier.
L'imagination sans habileté
nous donne l'art moderne."

e)

"Teddy bears mixing
sparkling chemicals as
mad scientists, in
a steampunk style."

Supervised learning with structured outputs ($y \in \mathcal{Y}$).

Linear regression

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

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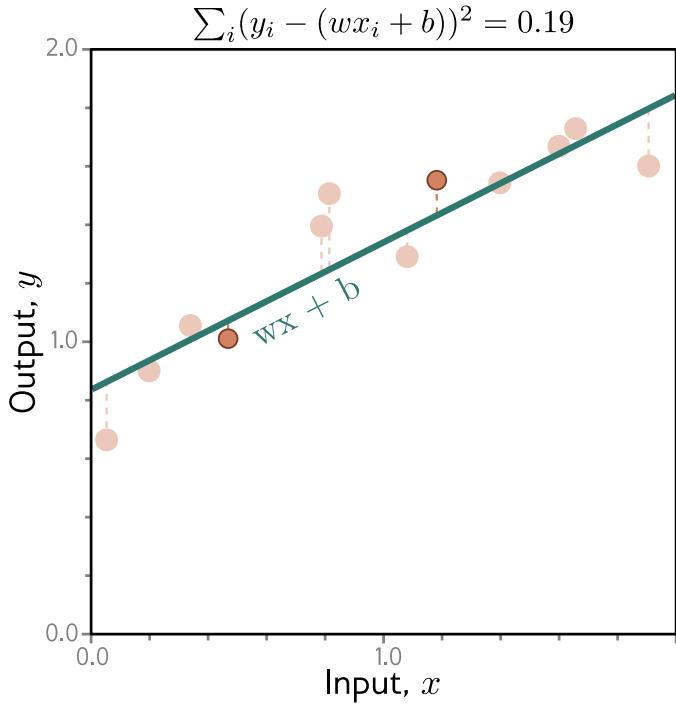
$$p(y|\mathbf{x}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2} \frac{(y - (\mathbf{w}^T\mathbf{x} + b))^2}{\sigma^2} \right)$$

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

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

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

$$\sum_i (y_i - (wx_i + b))^2 = 0.19$$

Minimizing the negative log-likelihood of a linear Gaussian model reduces to minimizing the sum of squared residuals.
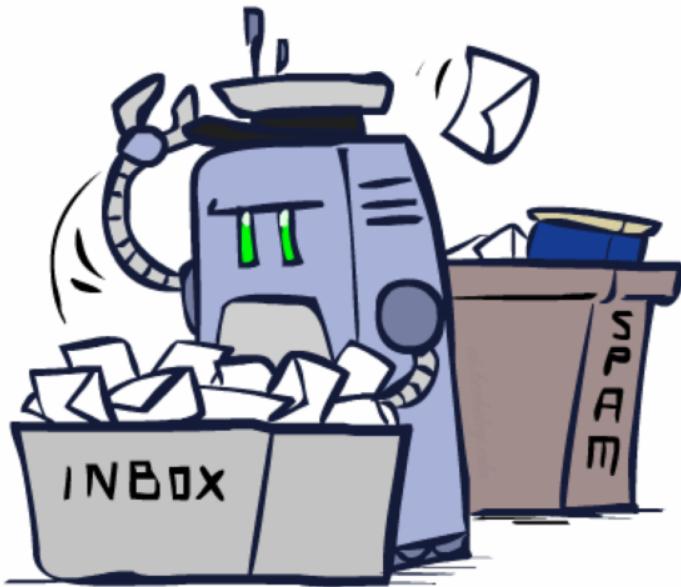
If we absorb the bias term b into the weight vector $\mathbf{w}$ by adding a constant feature $x_0 = 1$ to the input vector $\mathbf{x}$, the solution $\mathbf{w}^*$ is given analytically by

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y},$$

where $\mathbf{X}$ is the input matrix made of the stacked input vectors $\mathbf{x}_j$ (including the constant feature) and $\mathbf{y}$ is the output vector made of the output values y_j.

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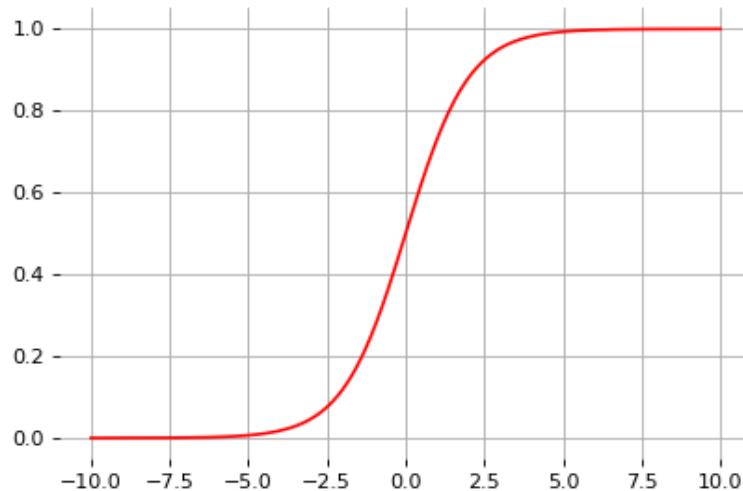
Let us now assume $y \in \{0, 1\}$.

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

$$\arg\max_{\mathbf{w},b} P(\mathbf{d}|\mathbf{w}, b)$$

$$= \arg\max_{\mathbf{w},b} \prod_{\mathbf{x}_i, y_i \in \mathbf{d}} P(Y = y_i|\mathbf{x}_i, \mathbf{w}, b)$$

$$= \arg\max_{\mathbf{w},b} \prod_{\mathbf{x}_i, y_i \in \mathbf{d}} \sigma(\mathbf{w}^T\mathbf{x}_i + b)^{y_i} (1 - \sigma(\mathbf{w}^T\mathbf{x}_i + b))^{1-y_i}$$

$$= \arg\min_{\mathbf{w},b} \underbrace{\sum_{\mathbf{x}_i, y_i \in \mathbf{d}} -y_i \log\sigma(\mathbf{w}^T\mathbf{x}_i + b) - (1 - y_i)\log(1 - \sigma(\mathbf{w}^T\mathbf{x}_i + b))}_{\mathcal{L}(\mathbf{w},b)=\sum_i \ell(y_i,\hat{y}(\mathbf{x}_i;\mathbf{w},b))}$$

This loss is an estimator of the cross-entropy

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

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

Unfortunately, there is no closed-form solution for the MLE of $\mathbf{w}$ and b.
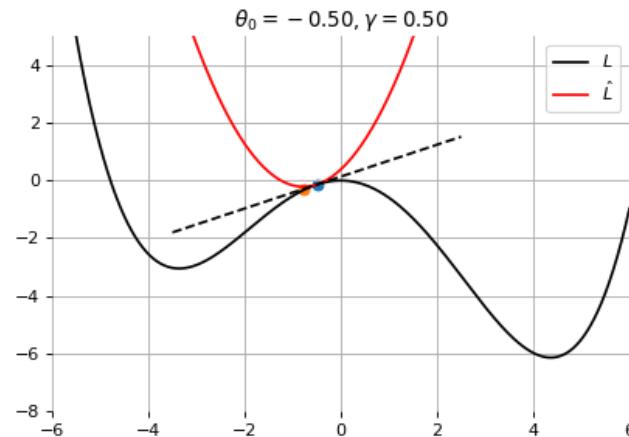
Gradient descent

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 θ_0, 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

$$\nabla_\epsilon \hat{\mathcal{L}}(\epsilon; \theta_0) = 0$$
$$= \nabla_\theta \mathcal{L}(\theta_0) + \frac{1}{\gamma}\epsilon,$$
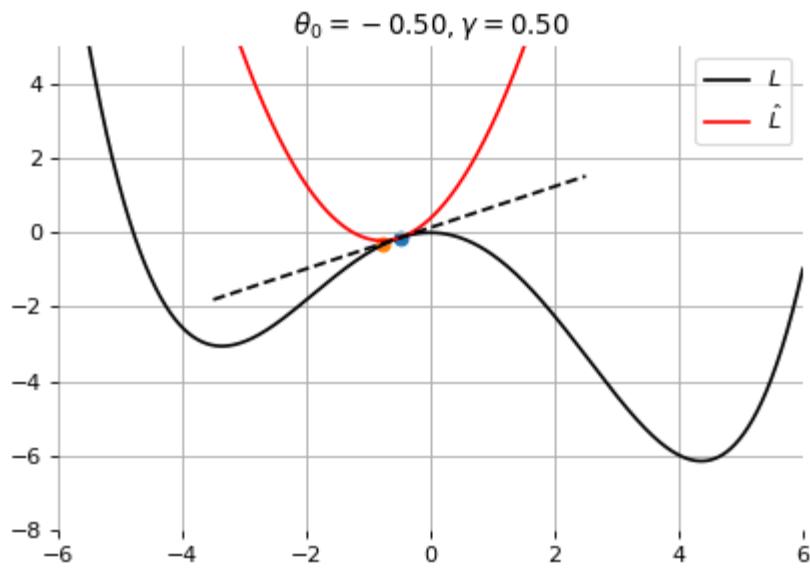
which results in the best improvement for the step $\epsilon = -\gamma \nabla_\theta \mathcal{L}(\theta_0)$.
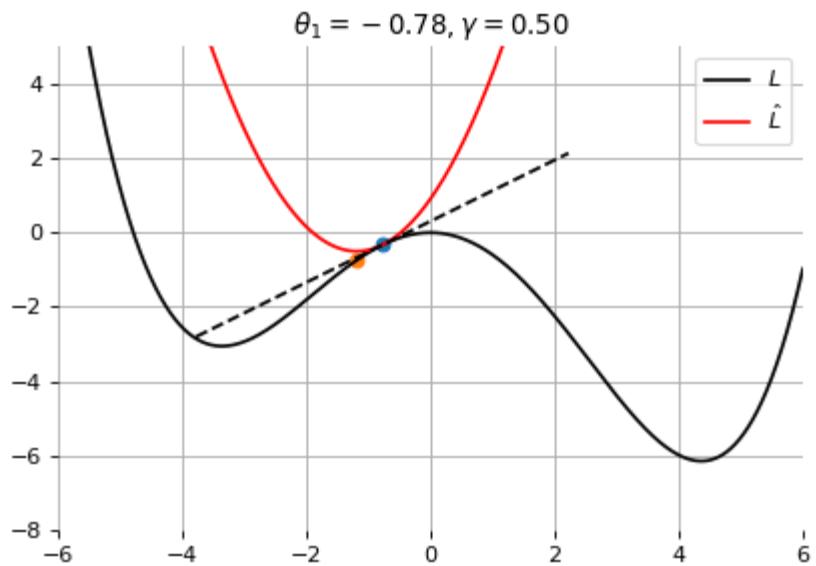
Therefore, model parameters can be updated iteratively using the update rule
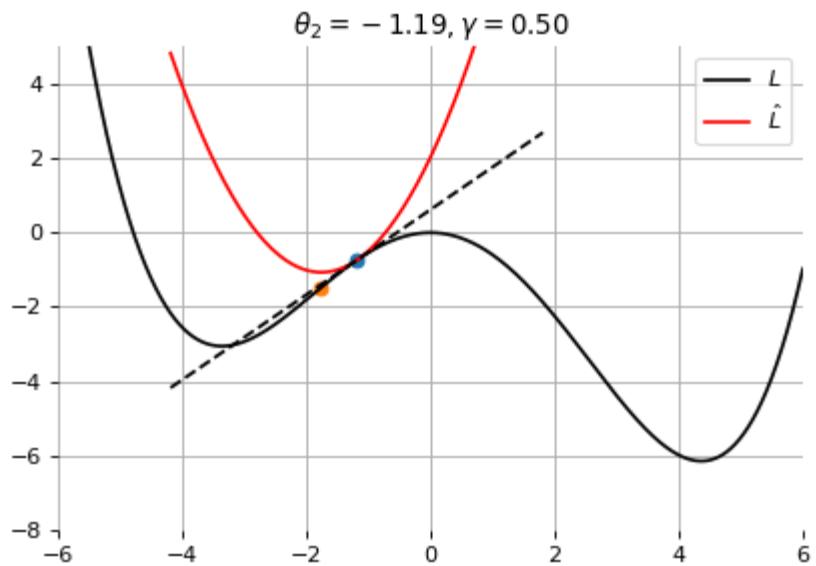
$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t),$$

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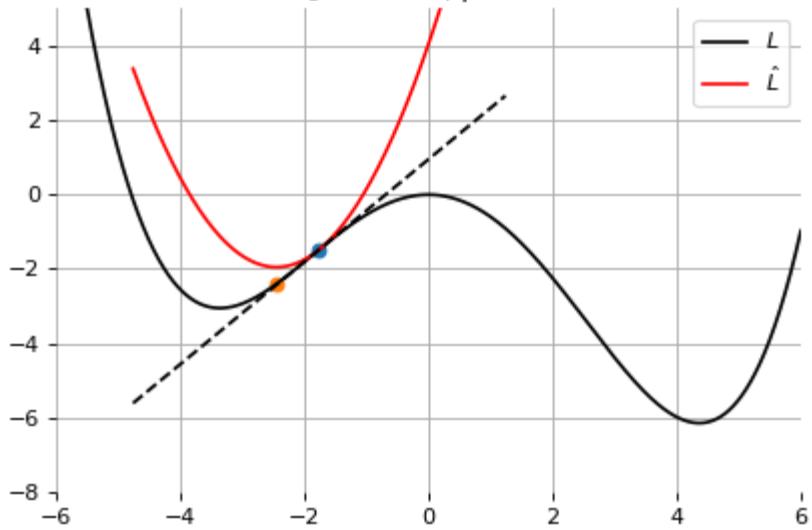
- θ_0 are the initial parameters of the model,
- γ is the learning rate.

$\theta_0 = -0.50, \gamma = 0.50$

$\theta_1 = -0.78, \gamma = 0.50$

$\theta_2 = -1.19, \gamma = 0.50$
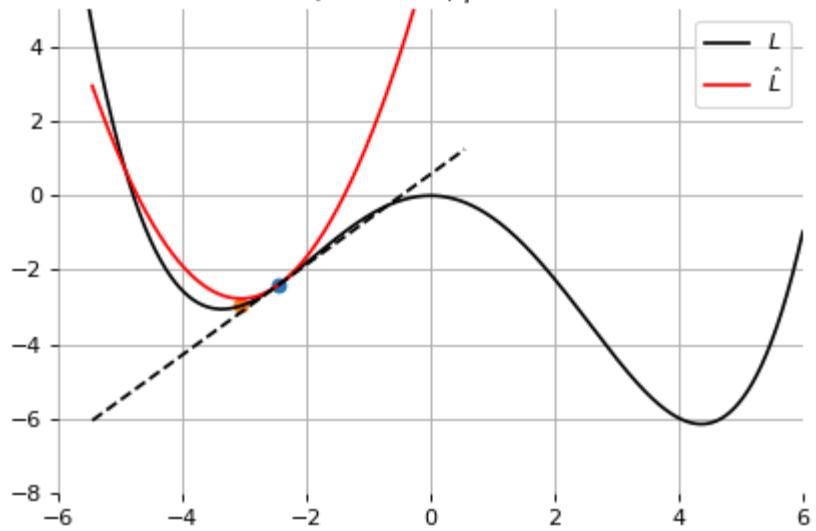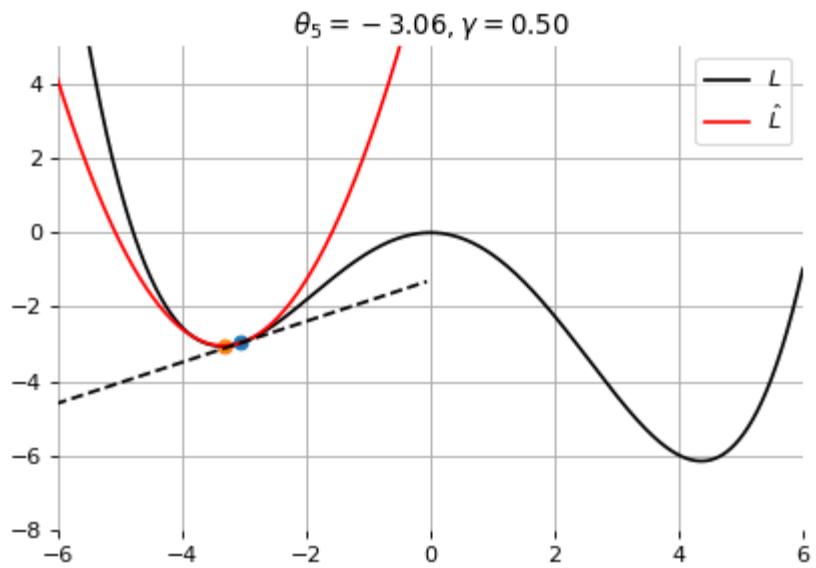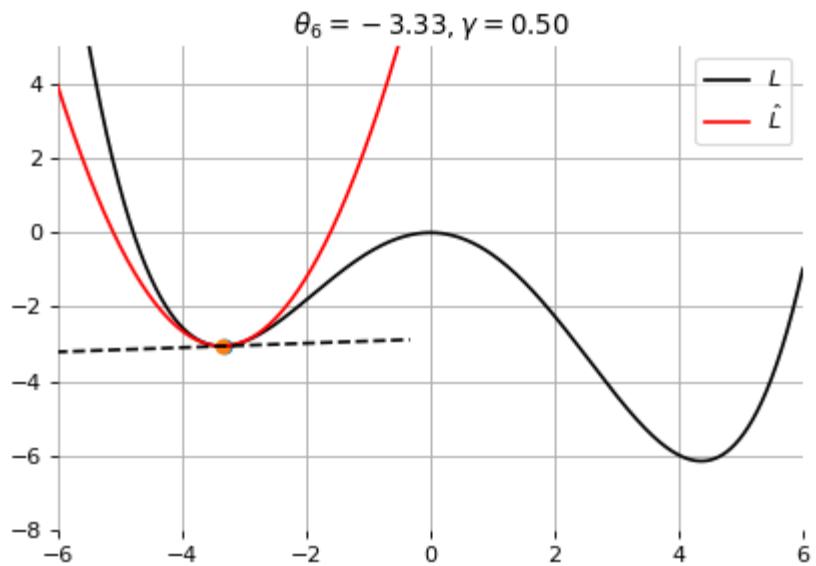
$\theta_3 = -1.76, \gamma = 0.50$
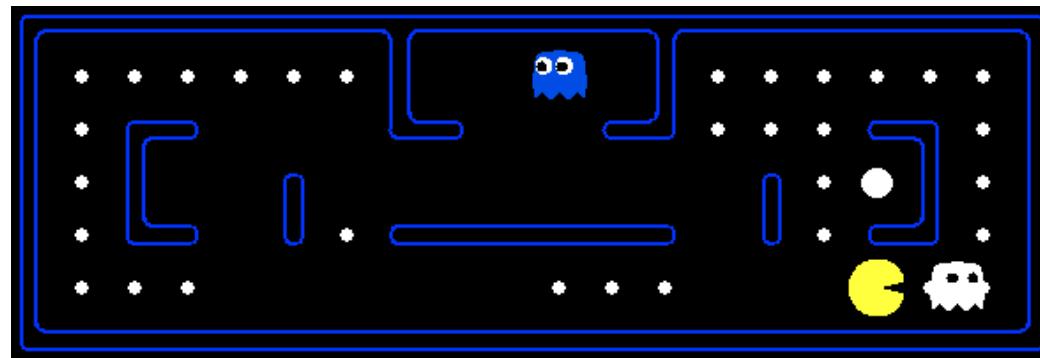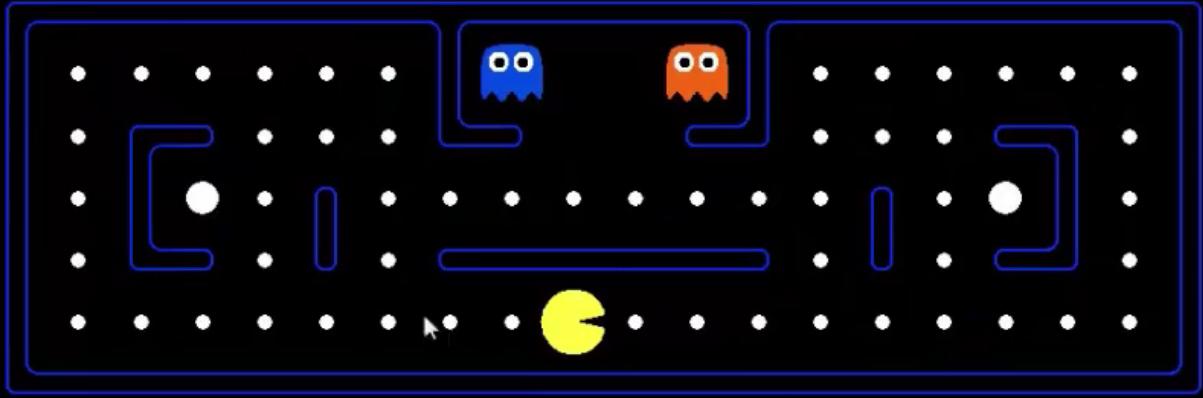
(Step-by-step code example)

Example: imitation learning in Pacman

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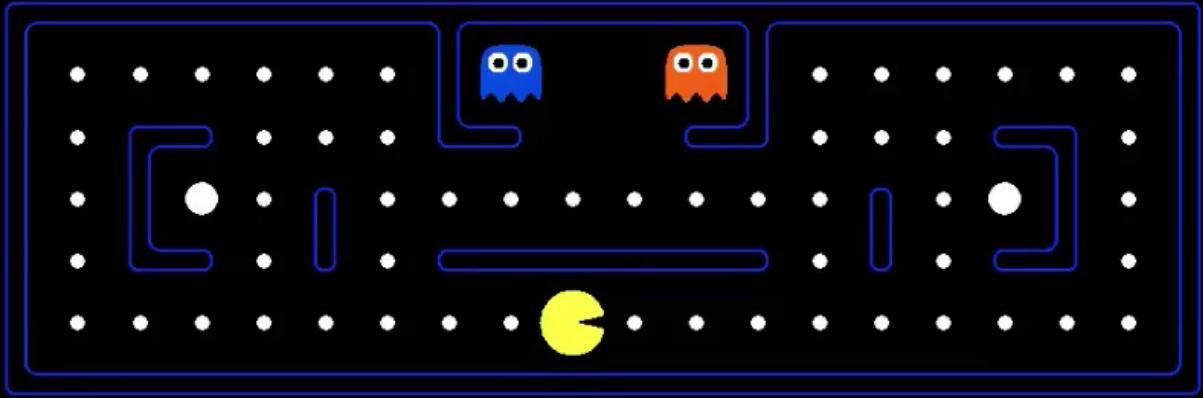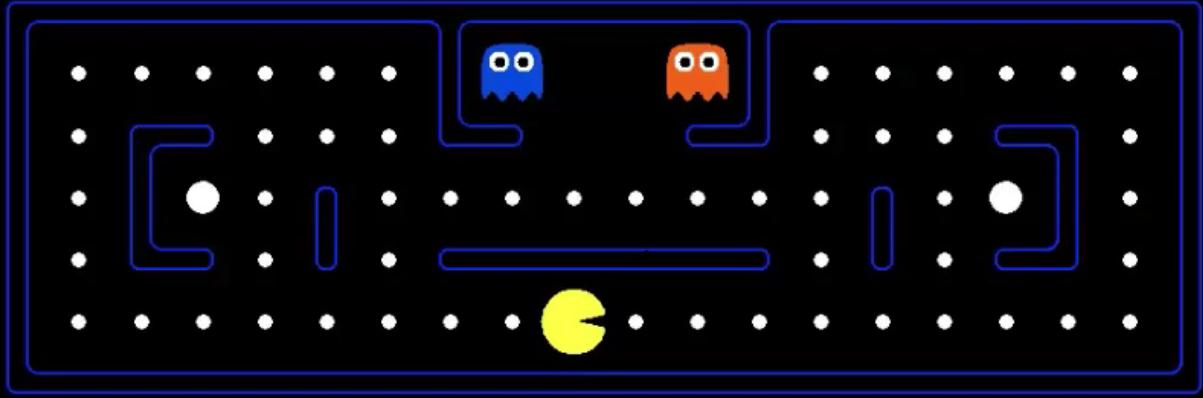
The agent observes a very good Minimax-based agent for two games and updates its weight vectors as data are collected.

SCORE: 0

▶ 0:00 / 0:18   🔇   ⛶   ⋮

SCORE: 0

▶ 0:00 / 0:21 🔇 ⬚ ⋮

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

Deep Learning

(a short introduction)

Shallow networks

A shallow network is a function

$$f : \mathbb{R}^{d_{\text{in}}} \rightarrow \mathbb{R}^{d_{\text{out}}}$$

that maps multi-dimensional inputs $\mathbf{x}$ to multi-dimensional outputs $\mathbf{y}$ through a hidden layer $\mathbf{h} = [h_0, h_1, ..., h_{q-1}] \in \mathbb{R}^q$, such that

$$h_j = \sigma \left(\sum_{i=0}^{d_{\text{in}}-1} w_{ji} x_i + b_j \right)$$

$$y_k = \sum_{j=0}^{q-1} v_{kj} h_j + c_k,$$
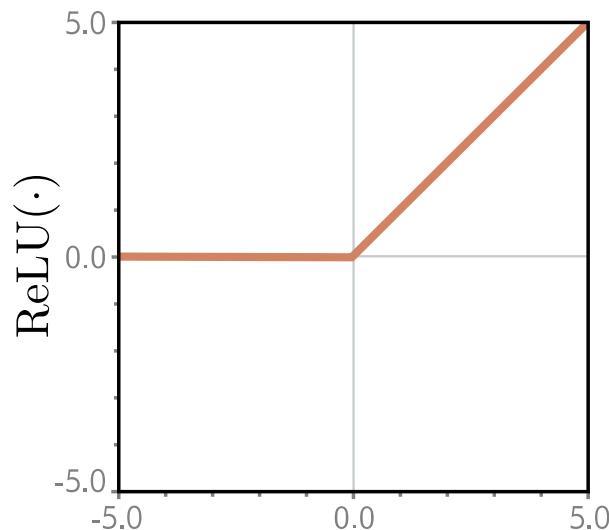
where w_{ji}, b_j, v_{kj} and c_k ($i = 0, ..., d_{\text{in}} - 1, j = 0, ..., q - 1$, $k = 0, ..., d_{\text{out}} - 1$) are the model parameters and σ is an activation function.
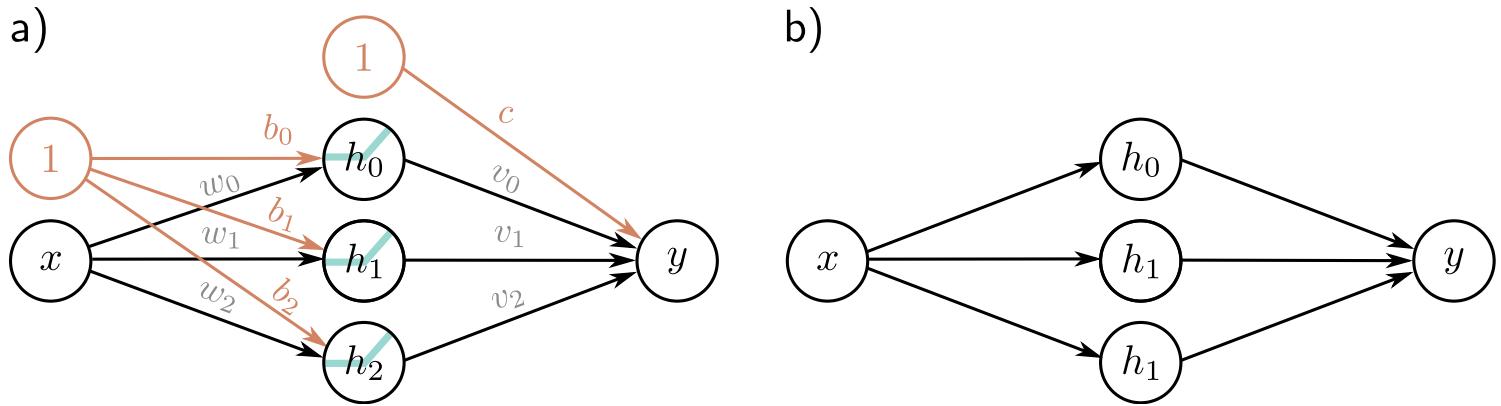
Single-input single-output networks

We first consider the case where $d_{\text{in}} = 1$ and $d_{\text{out}} = 1$ for the single-input single-output network
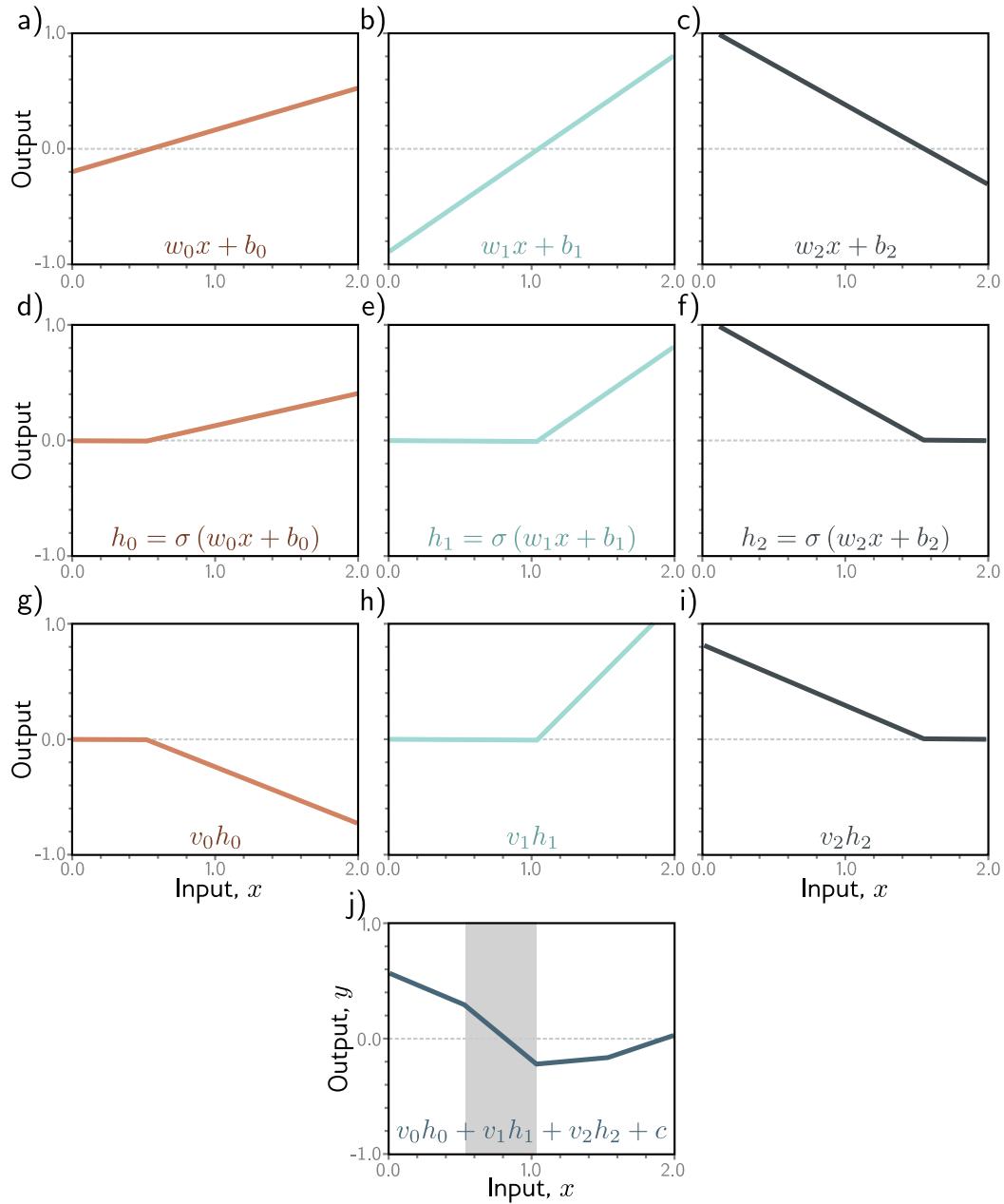
$$y = v_0 \sigma(w_0 x + b_0) + v_1 \sigma(w_1 x + b_1) + v_2 \sigma(w_2 x + b_2) + c$$
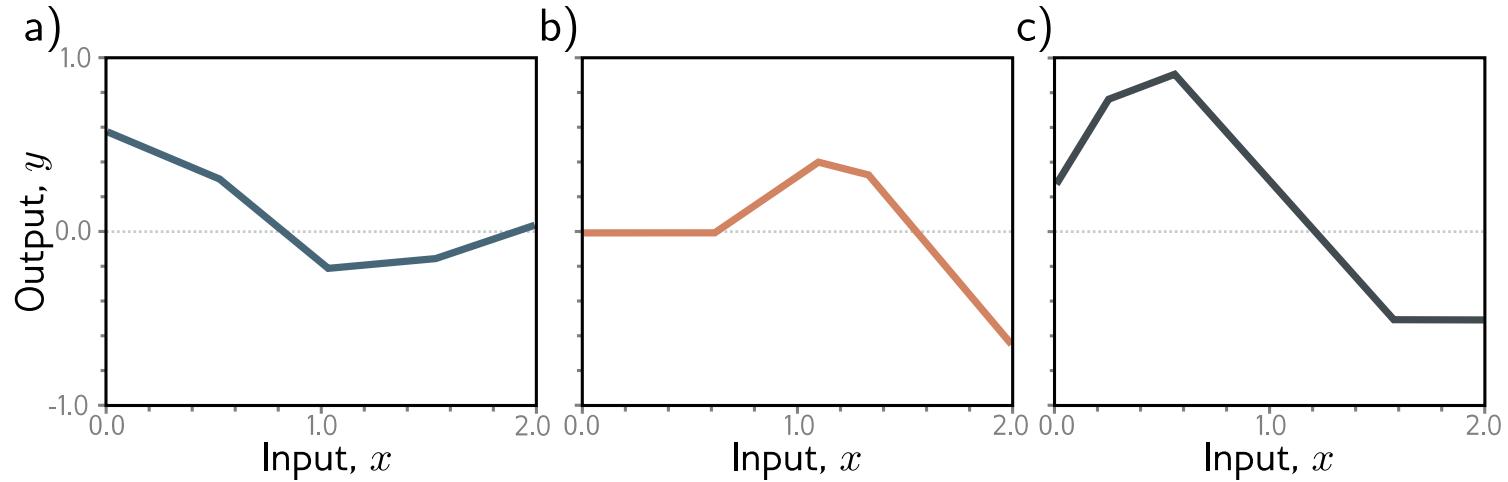
where $w_0, w_1, w_2, b_0, b_1, b_2, v_0, v_1, v_2$ and c are the model parameters and where the activation function σ is $\text{ReLU}(\cdot) = \max(0, \cdot)$.

a) The input x is on the left, the hidden units h_0, h_1 and h_2 are in the middle, and the output y is on the right. Computation flows from left to right.

b) More compact representation of the same network where we omit the bias terms, the weight labels and the activation functions.

a) $w_0 x + b_0$

b) $w_1 x + b_1$

c) $w_2 x + b_2$

d) $h_0 = \sigma \left(w_0 x + b_0 \right)$

e) $h_1 = \sigma \left(w_1 x + b_1 \right)$

f) $h_2 = \sigma \left(w_2 x + b_2 \right)$

g) $v_0 h_0$

h) $v_1 h_1$

i) $v_2 h_2$

j) $v_0 h_0 + v_1 h_1 + v_2 h_2 + c$
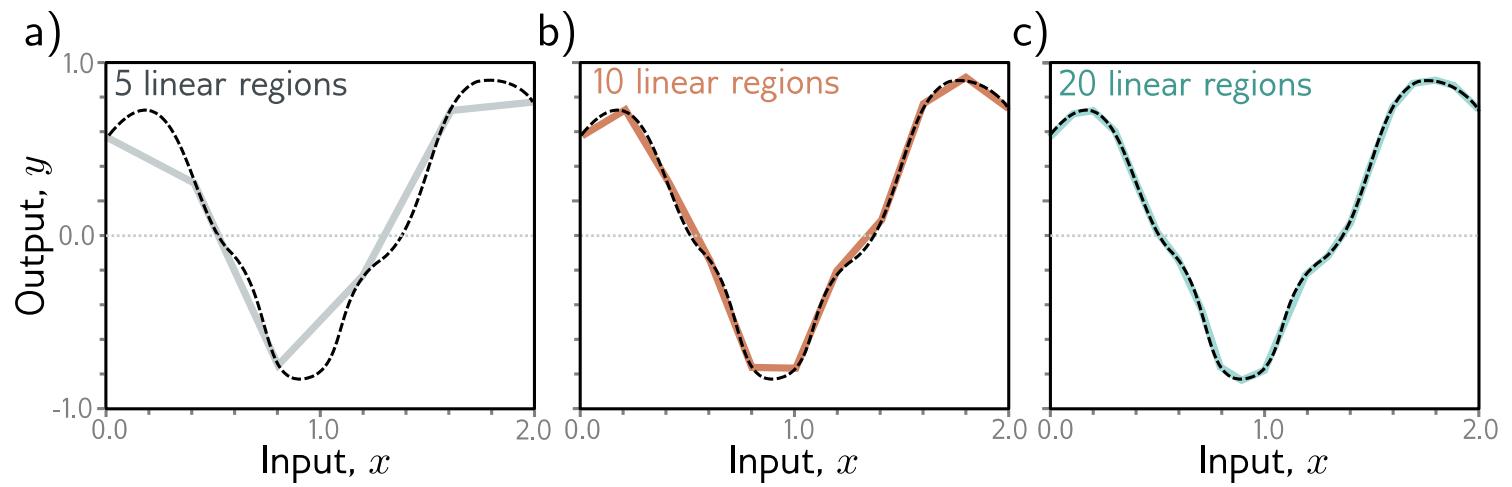
This network defines a family of piecewise linear functions where the positions of the joints, the slopes and the heights of the functions are determined by the 10 parameters w_0, w_1, w_2, b_0, b_1, b_2, v_0, v_1, v_2 and c.

Universal approximation theorem

The number q of hidden units h_j in a measure of the *capacity* of the shallow network. With ReLU activation functions, the hidden units define (up to) q joints in the input space, hence defining $q+1$ linear regions in the output space.

The universal approximation theorem states that a single-hidden-layer network with a finite number of hidden units can approximate any continuous function on a compact subset of $\mathbb{R}^d$ to arbitrary accuracy.

Multivariate outputs

To extend the network to multivariate outputs $\mathbf{y} = [y_0, y_1, .., y_{d_{\text{out}}-1}]$, we simply add more output units as linear combinations of the hidden units.

For example, a network with two output units y_0 and y_1 might have the following structure:
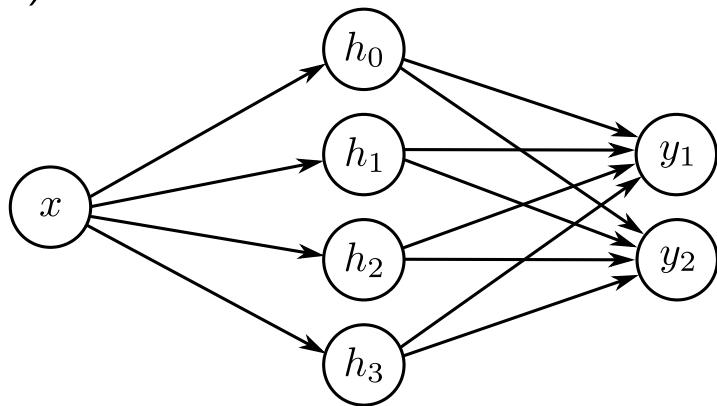
$$h_0 = \sigma\left(w_0 x + b_0\right)$$
$$h_1 = \sigma\left(w_1 x + b_1\right)$$
$$h_2 = \sigma\left(w_2 x + b_2\right)$$
$$h_3 = \sigma\left(w_3 x + b_3\right)$$
$$y_0 = v_{00} h_0 + v_{01} h_1 + v_{02} h_2 + v_{03} h_3 + c_0$$
$$y_1 = v_{10} h_0 + v_{11} h_1 + v_{12} h_2 + v_{13} h_3 + c_1$$
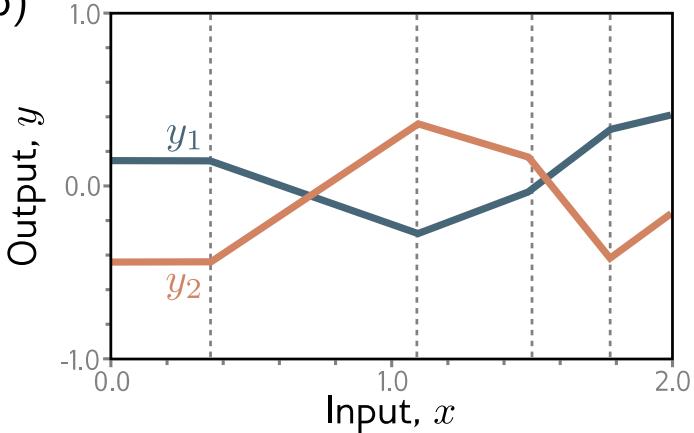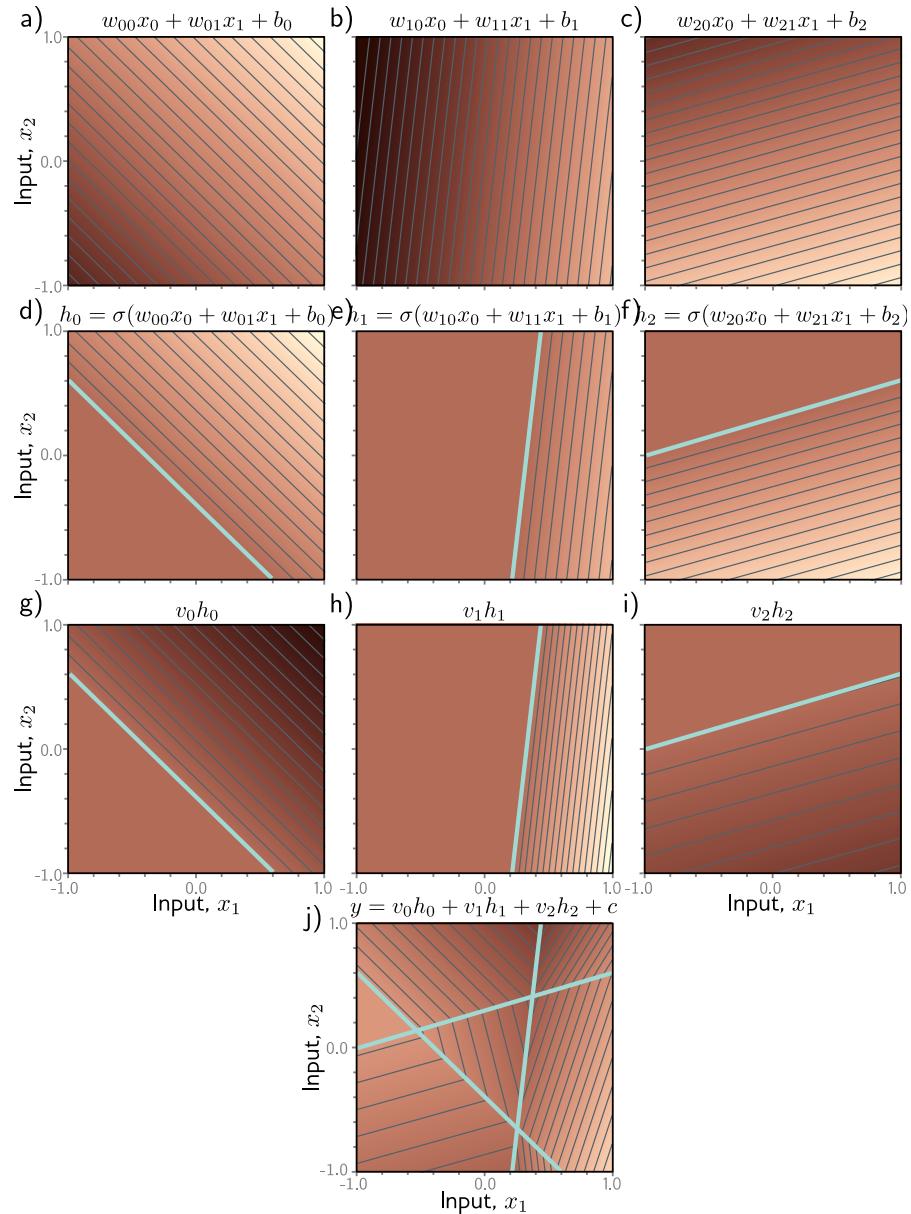
a) With two output units, the network can model two functions of the input x.

b) The four joints of these functions are constrained to be at the same positions, but the slopes and heights of the functions can vary independently.

Multivariate inputs

To extend the network to multivariate inputs $\mathbf{x} = \left[x_0, x_1, ..., x_{d_{\text{in}}-1} \right]$, we extend the linear relations between the input and the hidden units.

For example, a network with two inputs $\mathbf{x} = \left[x_0, x_1 \right]$ might have three hidden units h_0, h_1 and h_2 defined as

$$
h_0 = \sigma \left(w_{00} x_0 + w_{01} x_1 + b_0 \right)
$$
$$
h_1 = \sigma \left(w_{10} x_0 + w_{11} x_1 + b_1 \right)
$$
$$
h_2 = \sigma \left(w_{20} x_0 + w_{21} x_1 + b_2 \right).
$$

a) $w_{00}x_0 + w_{01}x_1 + b_0$
b) $w_{10}x_0 + w_{11}x_1 + b_1$
c) $w_{20}x_0 + w_{21}x_1 + b_2$
d) $h_0 = \sigma(w_{00}x_0 + w_{01}x_1 + b_0)$
e) $h_1 = \sigma(w_{10}x_0 + w_{11}x_1 + b_1)$
f) $h_2 = \sigma(w_{20}x_0 + w_{21}x_1 + b_2)$
g) $v_0 h_0$
h) $v_1 h_1$
i) $v_2 h_2$
j) $y = v_0 h_0 + v_1 h_1 + v_2 h_2 + c$

Deep networks

We first consider the composition of two shallow networks, where the output of the first network is fed as input to the second network as
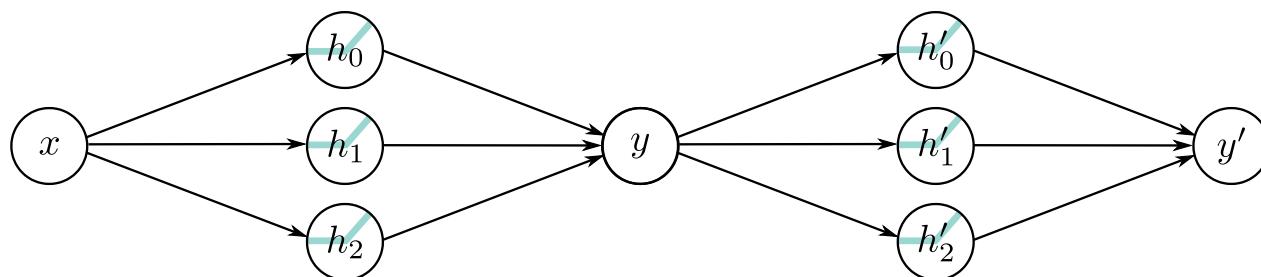
$$h_0 = \sigma \left(w_0 x + b_0 \right)$$
$$h_1 = \sigma \left(w_1 x + b_1 \right)$$
$$h_2 = \sigma \left(w_2 x + b_2 \right)$$
$$y = v_0 h_0 + v_1 h_1 + v_2 h_2 + c$$
$$h_0' = \sigma \left(w_0' y + b_0' \right)$$
$$h_1' = \sigma \left(w_1' y + b_1' \right)$$
$$h_2' = \sigma \left(w_2' y + b_2' \right)$$
$$y' = v_0' h_0' + v_1' h_1' + v_2' h_2' + c'.$$
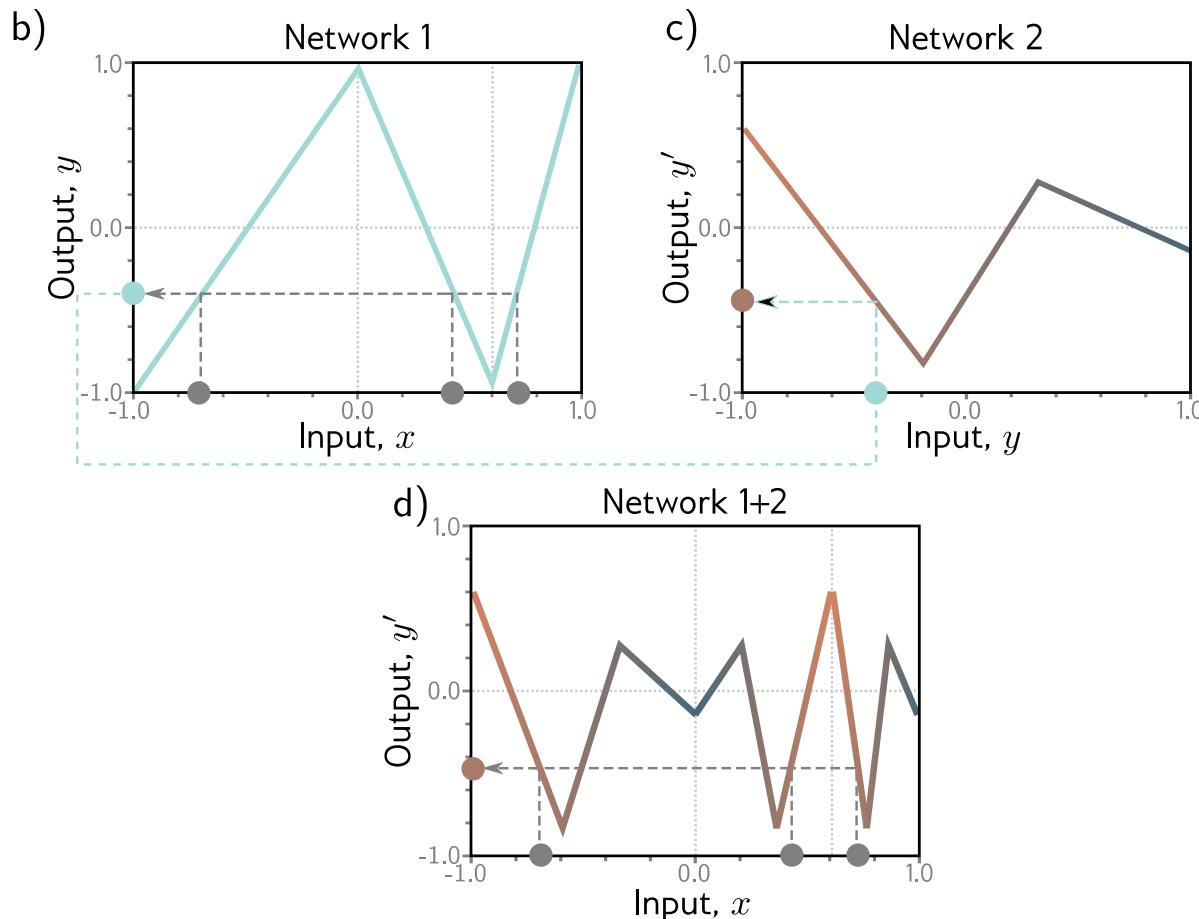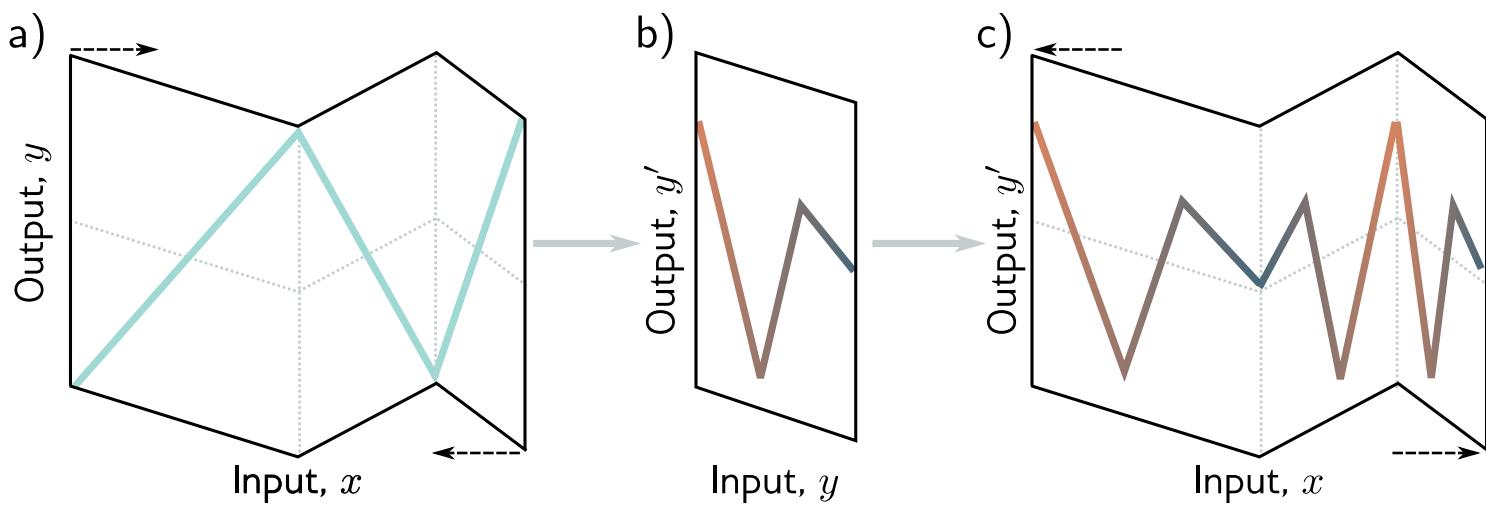
With **ReLU** activation functions, this network also describes a family of piecewise linear functions. However, each linear region defined by the hidden units of the first network is further divided by the hidden units of the second network.

a)

Output, y

Input, x

b)

Output, y'

Input, y

c)
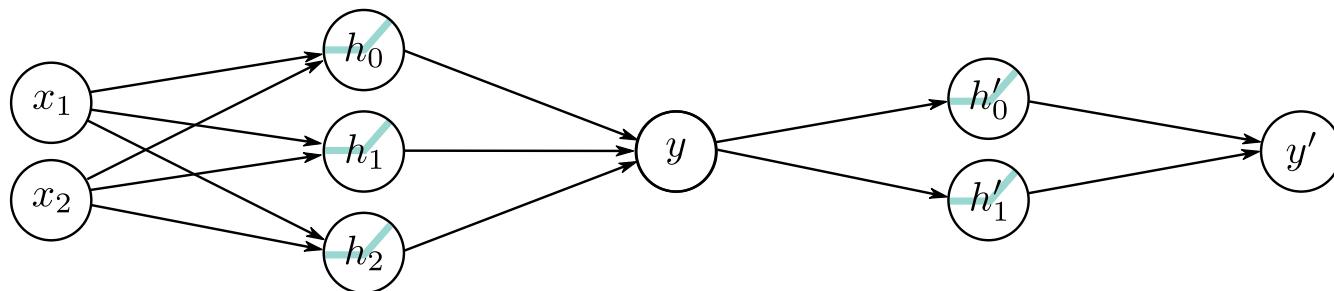
Output, y'

Input, x

Folding interpretation of a deep network:
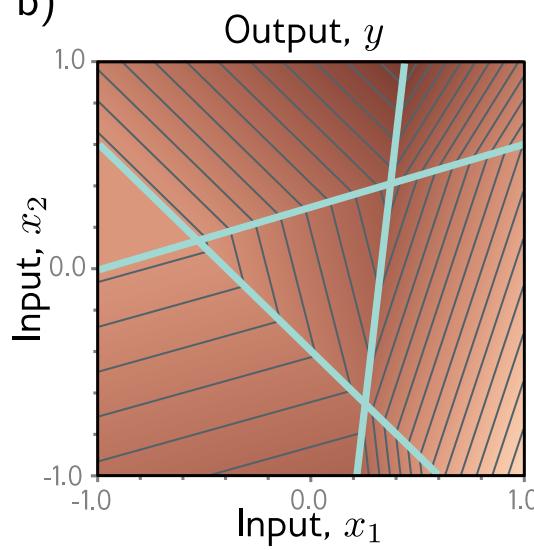
a) The first network folds the input space back on itself.
b) The second network applies its function to the folded space.
c) The final output is revealed by unfolding the folded space.

Similarly, composing a multivariate shallow network with a shallow network further divides the input space into more linear regions.


a)

b) Output, y

c)

d) Output, y'
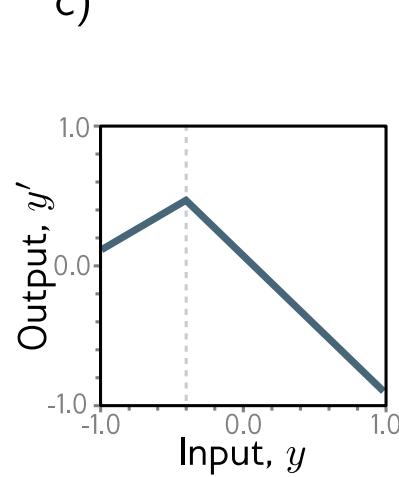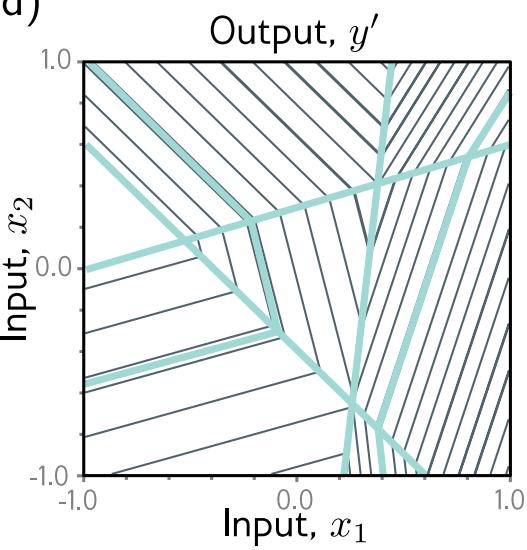
From composing shallow networks to deep networks

Since the operation from $[h_0, h_1, h_2]$ to y is linear and the operation from y to $[h'_0, h'_1, h'_2]$ is also linear, their composition in series is linear.

It follows that the composition of the two shallow networks is a special case of a deep network with two hidden layers where the first layer is defined as

$$
\begin{aligned}
h_0 &= \sigma\left(w_0 x + b_0\right) \\
h_1 &= \sigma\left(w_1 x + b_1\right) \\
h_2 &= \sigma\left(w_2 x + b_2\right),
\end{aligned}
$$

the second layer is defined from the outputs of the first layer as
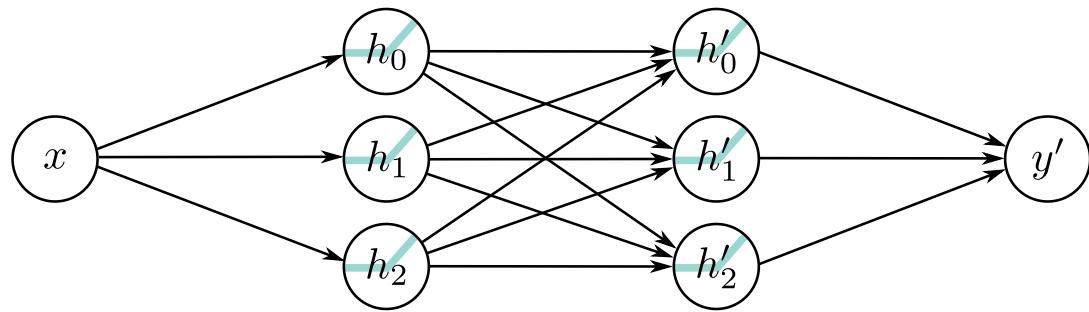
$$
\begin{aligned}
h'_0 &= \sigma\left(w'_{00} h_0 + w'_{01} h_1 + w'_{02} h_2 + b'_0\right) \\
h'_1 &= \sigma\left(w'_{10} h_0 + w'_{11} h_1 + w'_{12} h_2 + b'_1\right) \\
h'_2 &= \sigma\left(w'_{20} h_0 + w'_{21} h_1 + w'_{22} j_2 + b'_2\right),
\end{aligned}
$$
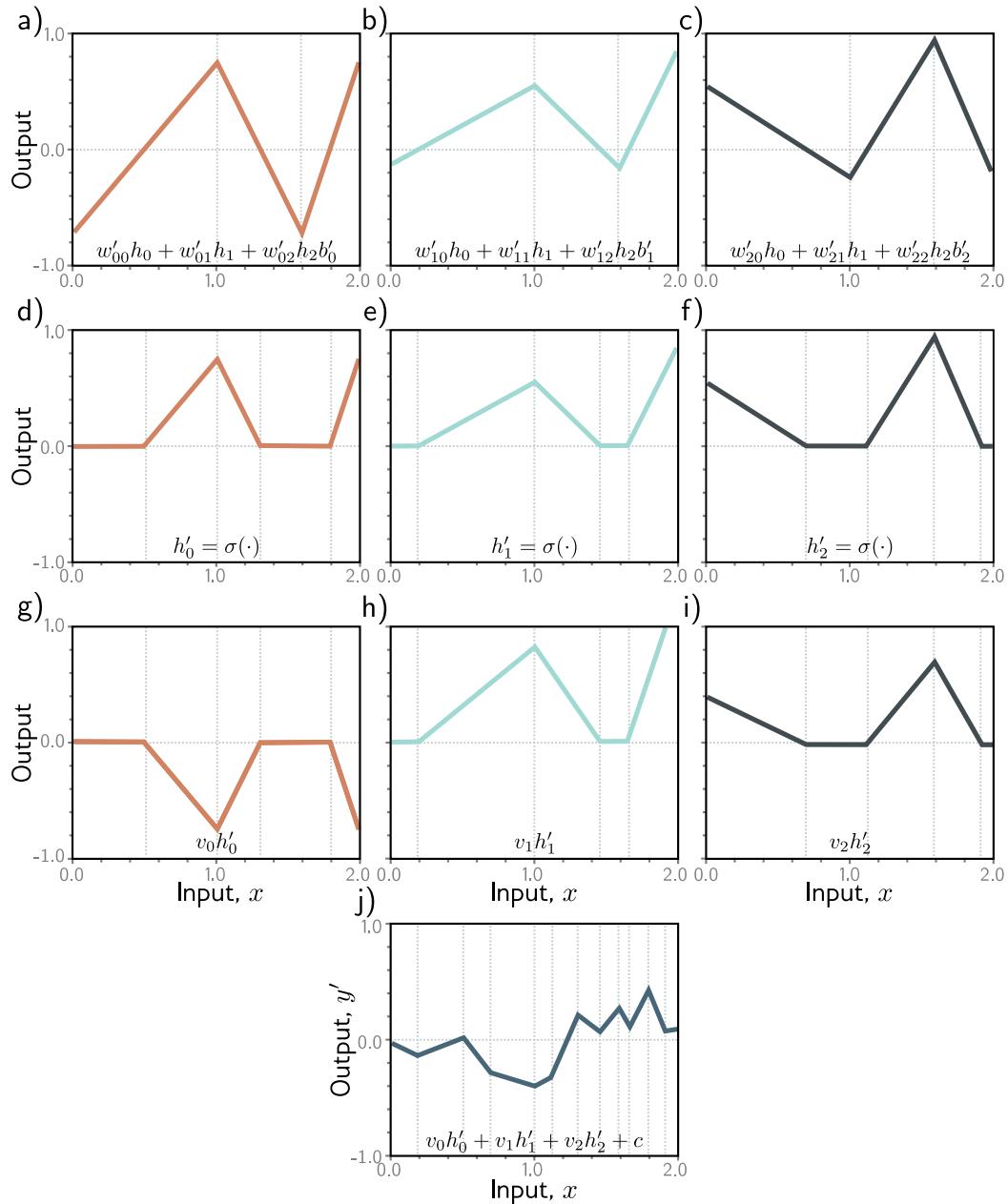
and the output is defined as

$$
y = v_0 h'_0 + v_1 h'_1 + v_2 h'_2 + c.
$$

a) $w'_{00}h_0 + w'_{01}h_1 + w'_{02}h_2 b'_0$

b) $w'_{10}h_0 + w'_{11}h_1 + w'_{12}h_2 b'_1$

c) $w'_{20}h_0 + w'_{21}h_1 + w'_{22}h_2 b'_2$

d) $h'_0 = \sigma(\cdot)$

e) $h'_1 = \sigma(\cdot)$

f) $h'_2 = \sigma(\cdot)$

g) $v_0 h'_0$

h) $v_1 h'_1$

i) $v_2 h'_2$

j) $v_0 h'_0 + v_1 h'_1 + v_2 h'_2 + c$

General formulation

The computation of a hidden layer can be written in matrix form as

$$
\mathbf{h} = \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_{q-1} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{00} & w_{01} & \cdots & w_{0(d_{\text{in}}-1)} \\ w_{10} & w_{11} & \cdots & w_{1(d_{\text{in}}-1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{(q-1)0} & w_{(q-1)1} & \cdots & w_{(q-1)(d_{\text{in}}-1)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{d_{\text{in}}-1} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{q-1} \end{bmatrix} \right)
$$

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

where $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$ is the input vector, $\mathbf{W} \in \mathbb{R}^{d_{\text{in}} \times q}$ is the weight matrix of the hidden layer and $\mathbf{b} \in \mathbb{R}^q$ is the bias vector.

Hidden layers can be composed in series to form a deep network with L layers such that
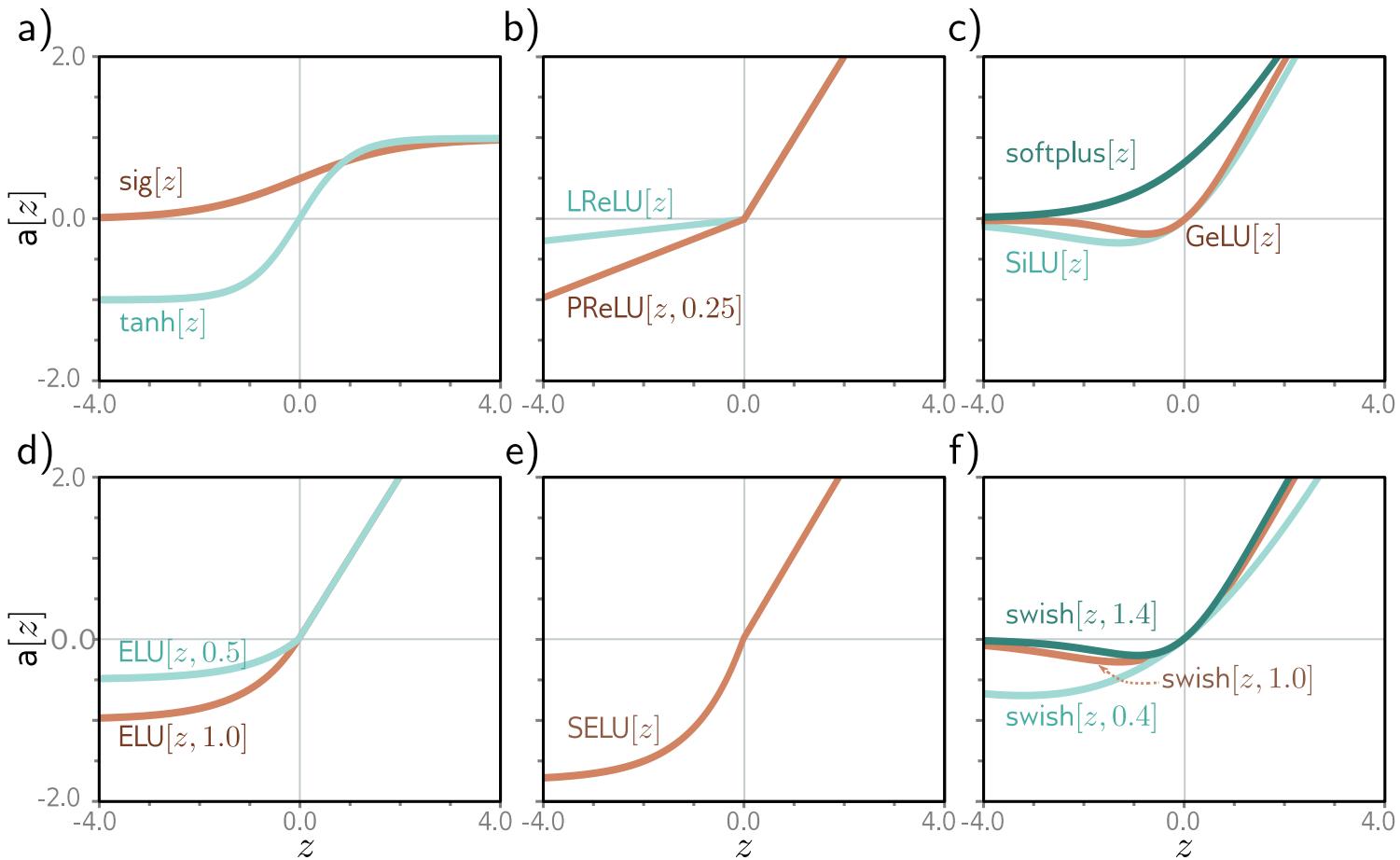
$$\mathbf{h}_0 = \mathbf{x}$$
$$\mathbf{h}_1 = \sigma(\mathbf{W}_1^T \mathbf{h}_0 + \mathbf{b}_1)$$
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2^T \mathbf{h}_1 + \mathbf{b}_2)$$
$$\vdots$$
$$\mathbf{h}_L = \sigma(\mathbf{W}_L^T \mathbf{h}_{L-1} + \mathbf{b}_L)$$
$$\mathbf{y} = \mathbf{h}_L,$$

where $\mathbf{W}_\ell \in \mathbb{R}^{q_{l-1} \times q_\ell}$ is the weight matrix of the ℓ-th layer, $\mathbf{b}_\ell \in \mathbb{R}^{q_\ell}$ is the bias vector of the ℓ-th layer, and $\mathbf{h}_\ell \in \mathbb{R}^{q_\ell}$ is the hidden vector of the ℓ-th layer.

This model is known as the feedforward neural network, the fully connected network, or the **multilayer perceptron** (MLP).

Activation functions

The choice of the activation function σ is crucial for the expressiveness of the network and the optimization of the model parameters.

Output layers

- For regression, the width q of the last layer L is set to the dimensionality of the output d_{out} and the activation function is the identity $\sigma(\cdot) = \cdot$, which results in a vector $\mathbf{h}_L \in \mathbb{R}^{d_{\text{out}}}$.

- For binary classification, the width q of the last layer L is set to 1 and the activation function is the sigmoid $\sigma(\cdot) = \frac{1}{1+\exp(-\cdot)}$, 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 $\mathrm{Softmax}$ function, where its i-th output is defined as

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

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

Loss functions

The parameters (e.g., $\mathbf{W}_\ell$ and $\mathbf{b}_\ell$ for each layer ℓ) of a deep network $f(\mathbf{x}; \theta)$ are learned by minimizing a loss function $\mathcal{L}(\theta)$ over a dataset $\mathbf{d} = \{(\mathbf{x}_j, \mathbf{y}_j)\}$ of input-output pairs.

The loss function is derived from the likelihood:

- For regression, assuming a Gaussian likelihood, the loss is the mean squared error $\mathcal{L}(\theta) = \frac{1}{N} \sum_{(\mathbf{x}_j, \mathbf{y}_j) \in \mathbf{d}} (\mathbf{y}_j - f(\mathbf{x}_j; \theta))^2$.

- For classification, assuming a categorical likelihood, the loss is the cross-entropy $\mathcal{L}(\theta) = -\frac{1}{N} \sum_{(\mathbf{x}_j, \mathbf{y}_j) \in \mathbf{d}} \sum_{i=1}^{C} y_{ij} \log f_i(\mathbf{x}_j; \theta)$.

(Step-by-step code example)

MLPs on images?

The MLP architecture is appropriate for tabular data, but not for images.

- Each pixel of an image is an input feature, leading to a high-dimensional input vector.

- Each hidden unit is connected to all input units, leading to a high-dimensional weight matrix.
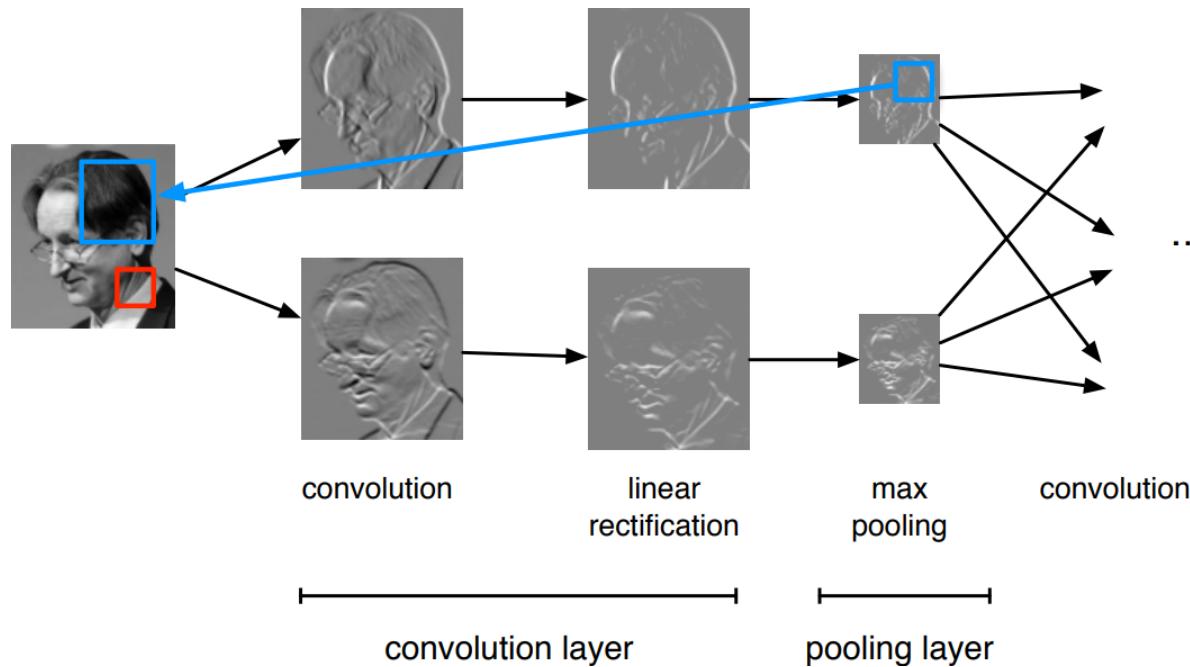
We want to design a neural architecture such that:

- in the earliest layers, the network responds 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.
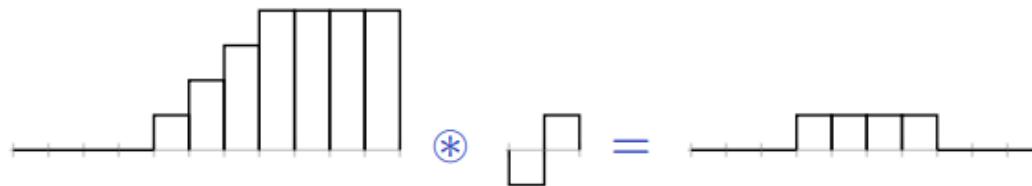
1d convolution

For the one-dimensional input $\mathbf{x} \in \mathbb{R}^W$ and the convolutional kernel $\mathbf{u} \in \mathbb{R}^w$, the discrete convolution $\mathbf{x} \circledast \mathbf{u}$ is a vector of size $W - w + 1$ such that

$$(\mathbf{x} \circledast \mathbf{u})[i] = \sum_{m=0}^{w-1} \mathbf{x}_{m+i}\, \mathbf{u}_m.$$
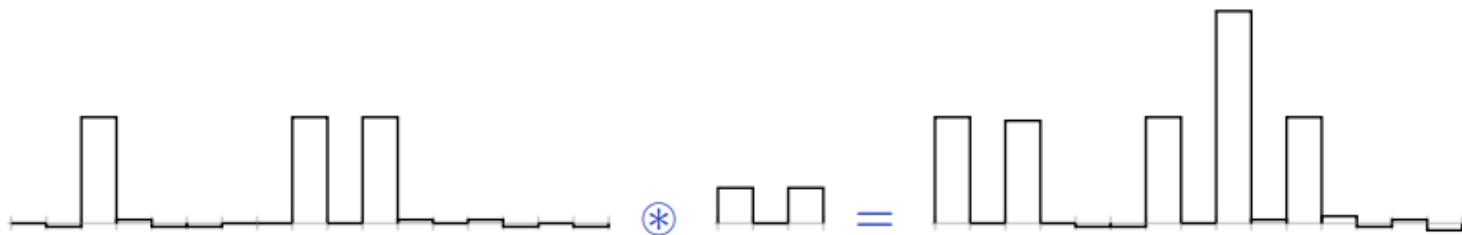
Convolutions can implement differential operators:

$$(0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4) \circledast (-1, 1) = (0, 0, 0, 1, 1, 1, 1, 0, 0, 0)$$

or crude template matchers:

$$(0, 0, 3, 0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0) \circledast (1, 0, 1) = (3, 0, 3, 0, 0, 0, 3, 0, 6, 0, 3, 0)$$
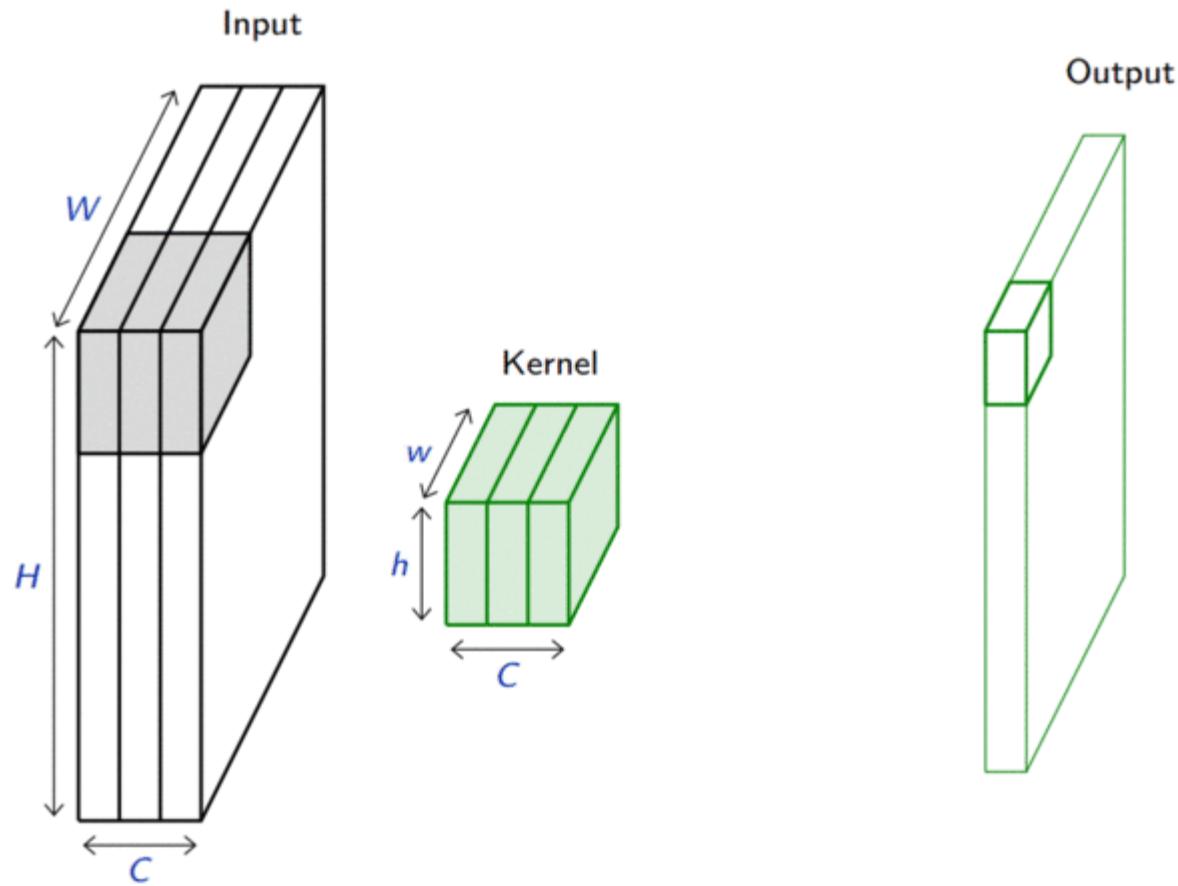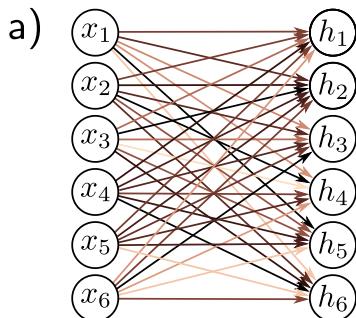
Convolutional layers

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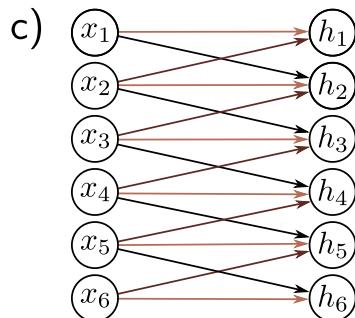
$$\mathbf{o}_{j,i} = (\mathbf{x} \circledast \mathbf{u})[j,i] = \sum_{c=0}^{C-1} \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} \mathbf{x}_{c,n+j,m+i} \, \mathbf{u}_{c,n,m}$$
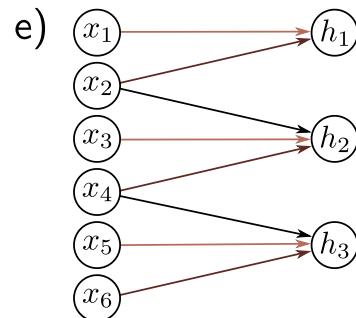
Input

W

H

C

Kernel

w

h

C

Output

Convolutional layers (c-f) are a special case of fully connected layers (a-b) where hidden units are connected to local regions of the input through shared weights (the kernels).
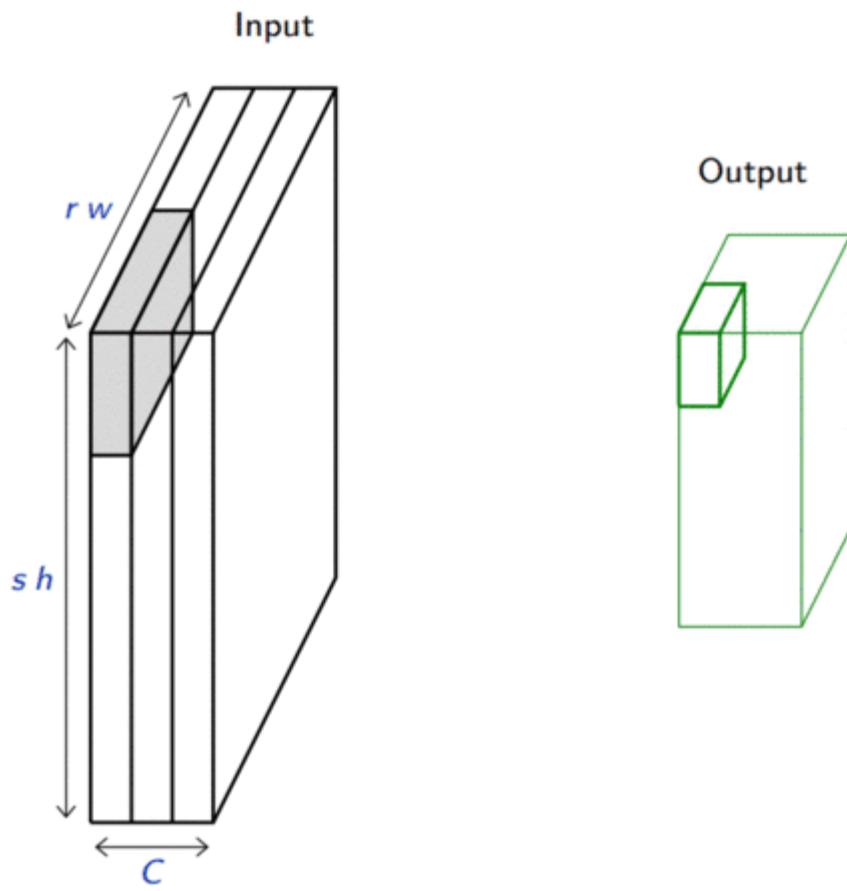
- The connectivity allows the network to learn local patterns in the input.

- Weight sharing allows the network to learn the same patterns at different locations in the input.

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

(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.
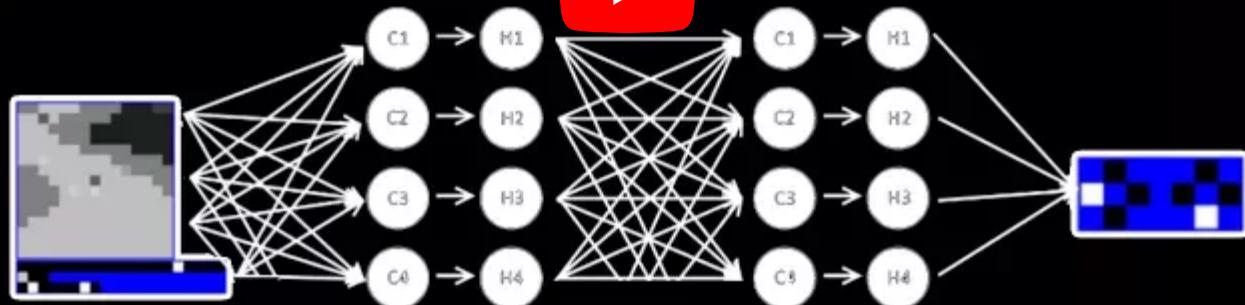
For example,

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

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

Notice how this is similar to filtering and dynamic decision networks:
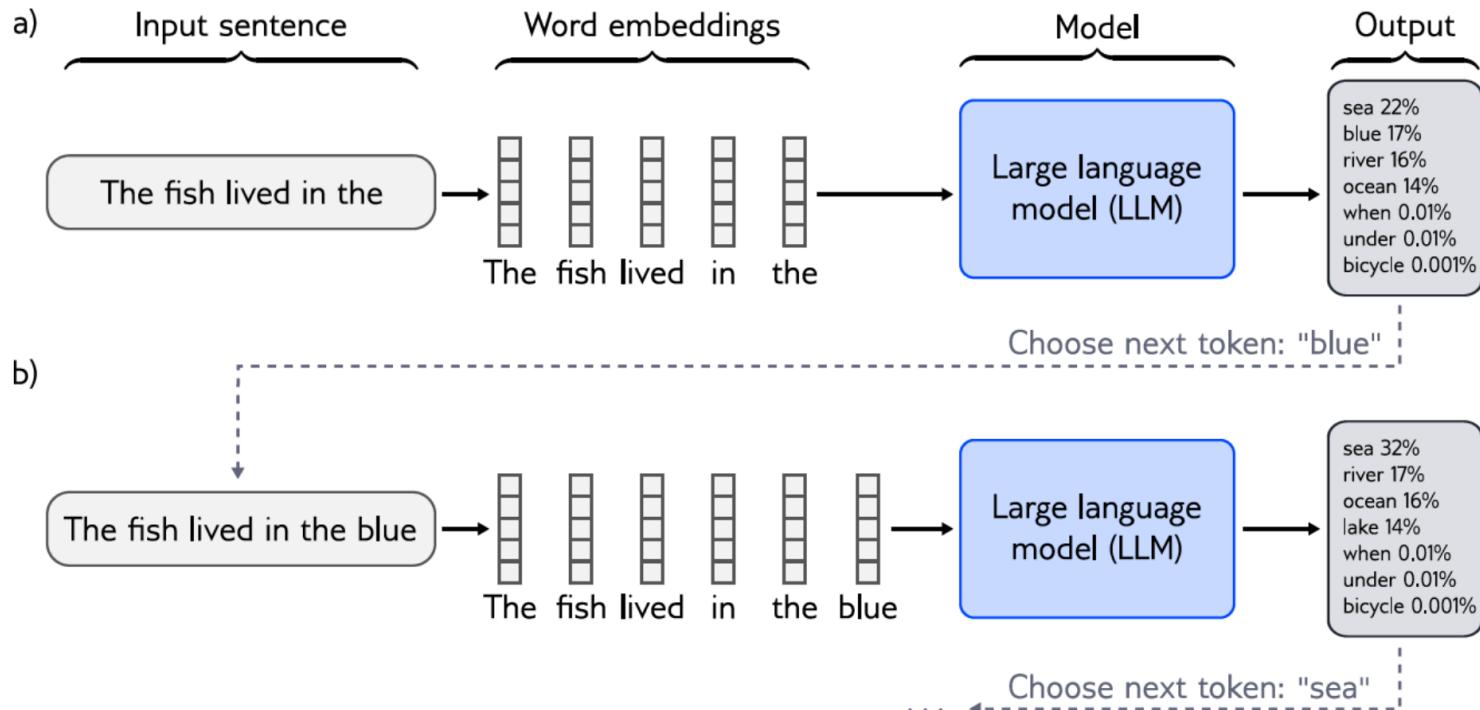
- $\mathbf{h}_t$ can be viewed as some current belief state;

- $\mathbf{x}_{1:T}$ 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 deep neural networks at the core of large-scale language models.
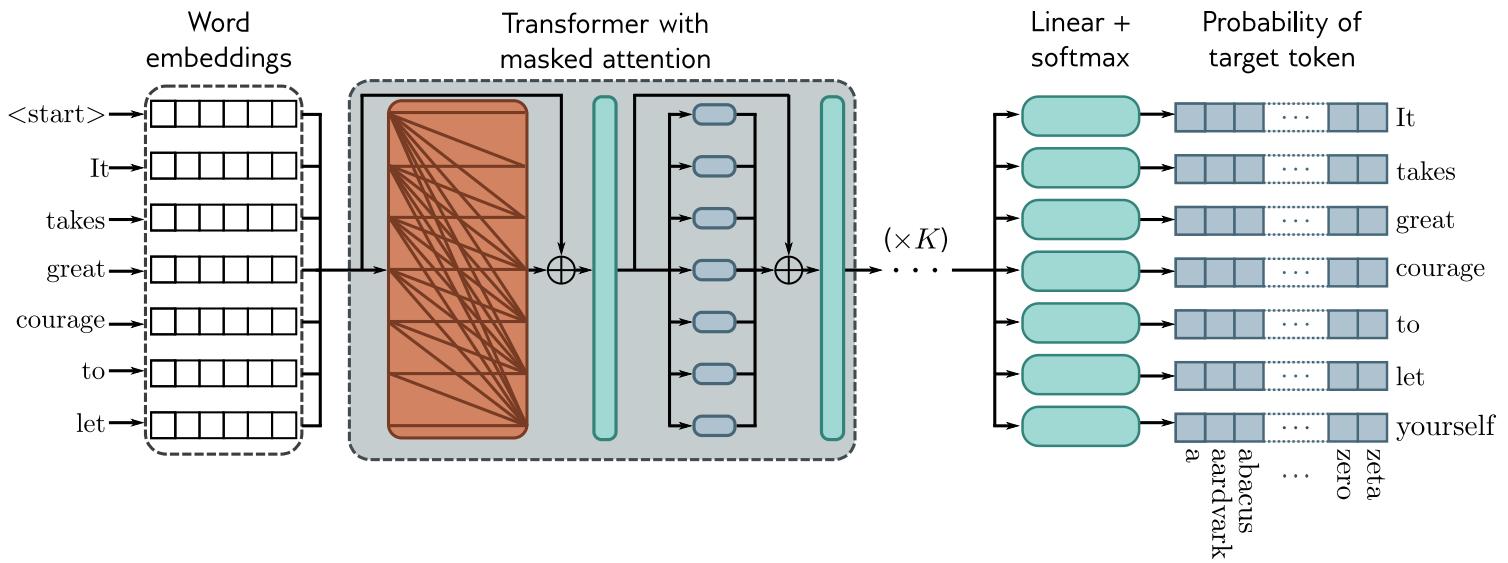
For language modeling, transformers define an **autoregressive model** that predicts the next word in a sequence given the previous words.

Formally,
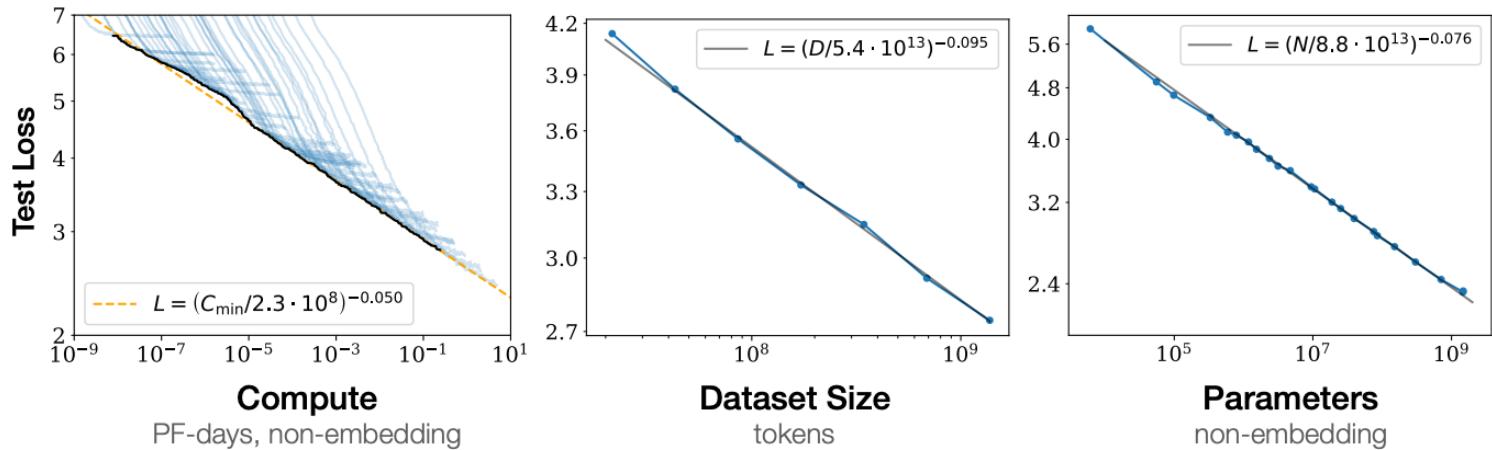
$$p(w_{1:t}) = p(w_1) \prod_{t=2}^{T} p(w_t | w_{1:t-1}),$$

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

The decoder-only transformer is a stack of K transformer blocks that process the input sequence in parallel using (masked) self-attention.

The output of the last block is used to predict the next word in the sequence, as in a regular classifier.

$$L = (C_{min}/2.3 \cdot 10^8)^{-0.050}$$

$$L = (D/5.4 \cdot 10^{13})^{-0.095}$$

$$L = (N/8.8 \cdot 10^{13})^{-0.076}$$

Compute
PF-days, non-embedding

Dataset Size
tokens

Parameters
non-embedding

Scaling laws

- The more data, the better the model.

- The more parameters, the better the model.

- The more compute, the better the model.
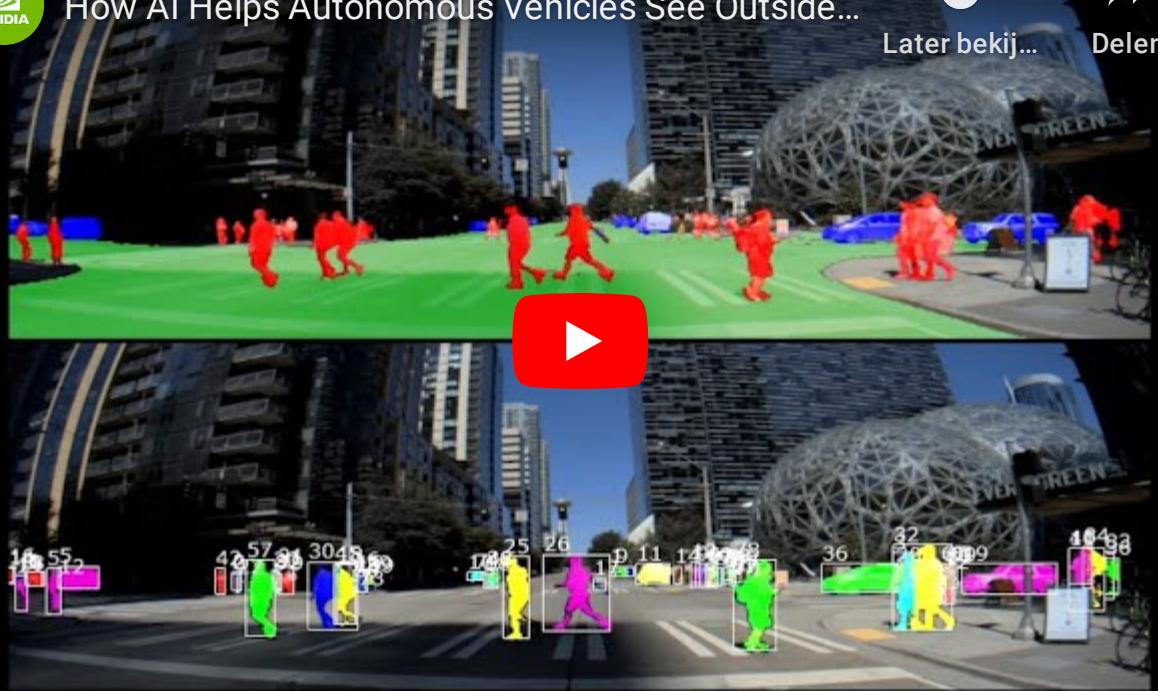
# AI beyond Pacman
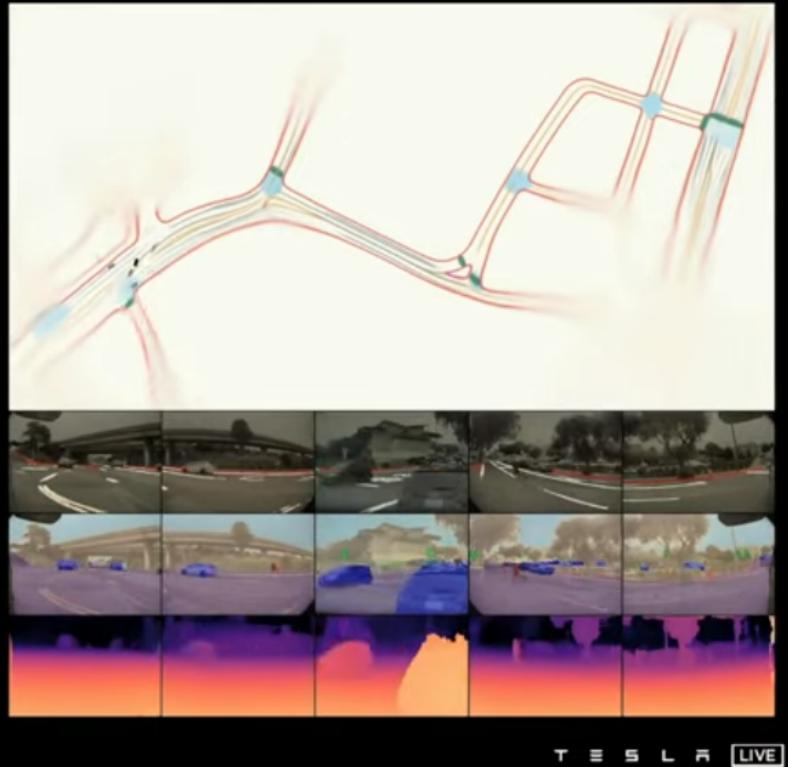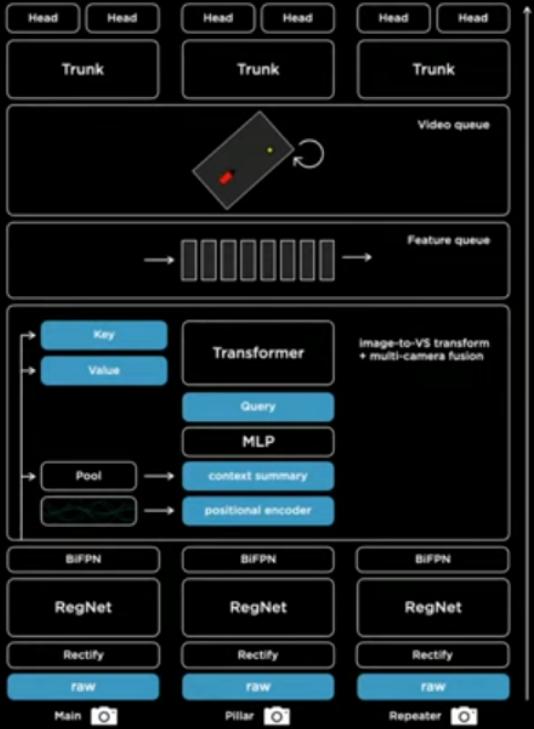
How AI Helps Autonomous Vehicles See Outside the Box
(See also other episodes from NVIDIA DRIVE Labs)

Hydranet (Tesla, 2021)

How machine learning is advancing medicine (Google, 2018)

Summary

- Deep learning is a powerful tool for learning from data.

- Neural networks are composed of layers of neurons that are connected to each other.

- The weights of the connections are learned by minimizing a loss function.

- Convolutional networks are used for image processing.

- Transformers are used for language processing.

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

Chris Bishop, 2020.

The end.