Introduction to neural networks

Alexander Binder
Modification of slides made by Andreas Kleppe, Ole-Johan Skrede, me

25.01.2022



UiO : Department of Informatics
University of Oslo

- For the time being, it seems that teachers can only participate on Mattermost or Astroforum.
- We encourage collaboration between students.
  - Explaining or discussing with fellow students are often more educational for all than to provide working code.
  - Students are allowed to answer each others requests on Mattermost.
  - The mandatory exercises and the exam must be individual work.
- Highly recommend doing the weekly exercises.

Outline

1. Classification by logistic regression
2. Artificial neurons
3. Neural networks
4. Example: The XOR problem
5. More on representability

Goal of classification: For every input sample $x \in \mathcal{X}$, correctly predict which class y it belongs.

- For 2-class classification, $y \in \{-1, +1\}$ or $y \in \{0, 1\}$.

Initial idea (from last lecture): Apply a linear mapping and classify according to the sign of the output:

$$f(x) = w \cdot x + b, \; s(x) = sign(f(x))$$

While $f(x)$ has unbounded values, $s(x)$ is either -1 or 1, but:

- if $f(x) \approx 0$, we should be uncertain about the prediction.
- if $f(x) \gg 0$, we should be confident about the prediction 1.
- if $f(x) \ll 0$, we should be confident about the prediction -1.

How can we encode the uncertainty into a mapping from $(-\infty, +\infty)$ onto $[0, 1]$? We would like to map:

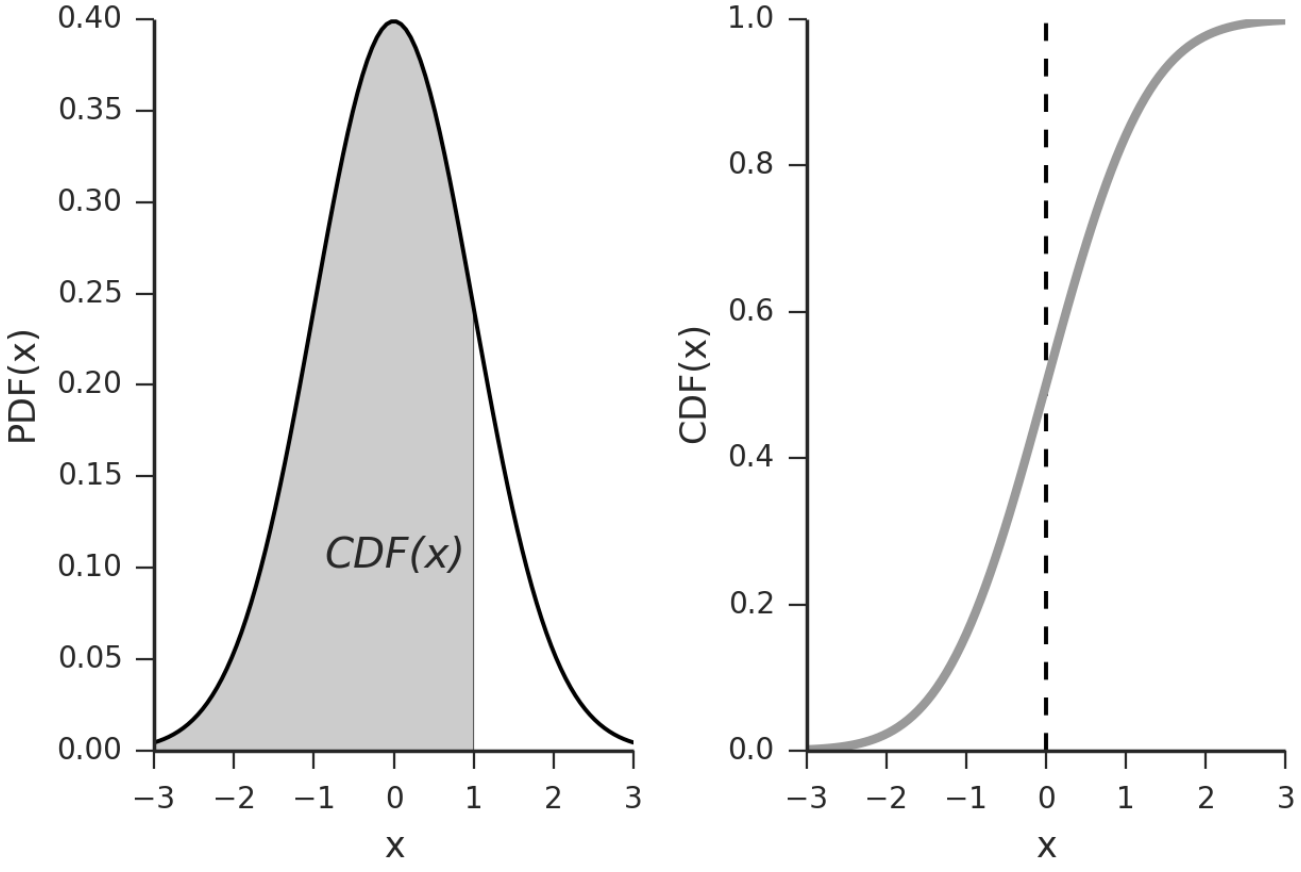
- $f(x) \approx 0$ to $s(f(x)) \approx 0.5$.
- $f(x) \gg 0$ to $s(f(x)) \approx 1$, in particular let $\lim_{u \to +\infty} s(u) = 1$
- $f(x) \ll 0$ to $s(f(x)) \approx 0$, in particular let $\lim_{u \to -\infty} s(u) = 0$

This would allow us to interpret $s(u)$ as a probability over inputs u.

There are many possible choices for the function $s(u)$.

We could e.g. use the cumulative distribution function (CDF) of any probability distribution function (PDF) with a median of 0.

- The normal (Gaussian) distribution is used in probit (from **prob**ability and un**it**) regression.



We will use a simpler function called the logistic sigmoid function:

Definition: Logistic sigmoid function

$$s(u) = \frac{\exp(u)}{1+\exp(u)} = \frac{1}{\exp(-u)+1}\frac{\exp(u)}{\exp(u)} = \frac{1}{\exp(-u)+1}$$

Note that:

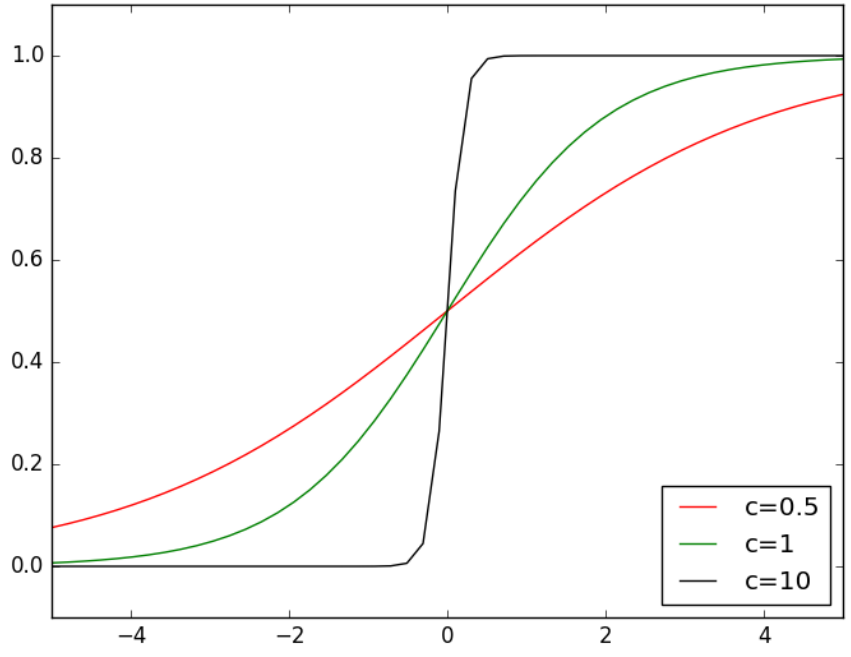
$$s(0) = \frac{1}{\exp(-0)+1} = \frac{1}{1+1} = 0.5$$

$$\lim_{u\to\infty} s(u) = \lim_{u\to\infty} \frac{1}{\exp(-u)+1} = \frac{1}{0+1} = 1$$

$$\lim_{u\to-\infty} s(u) = \lim_{u\to-\infty} \frac{\exp(u)}{1+\exp(u)} = \frac{0}{1+0} = 0$$

How does the logistic sigmoid function look?
Let's plot it for different scalings $c > 0$:

$$s(cu) = \frac{\exp(cu)}{1 + \exp(cu)} = \frac{1}{\exp(-cu) + 1}$$



Definition: Logistic regression model

Assume we have a linear (or affine) mapping $f_{w,b}(x) = w \cdot x + b$. Plugging it into the logistic sigmoid function $s(u)$ provides a logistic regression model:

$$s(f_{w,b}(x)) = \frac{\exp(w \cdot x + b)}{1 + \exp(w \cdot x + b)} = \frac{1}{\exp(-w \cdot x - b) + 1}$$

For 2-class classification problems, the output $s(f_{w,b}(x)) \in [0, 1]$ is the predicted probability that sample x has class label $y = 1$, that is, $P(Y = 1|X = x)$.

A logistic model may also be called a logit (from **log**istic and un**it**) model.

Denoting the probability as p, the logit function is the logarithm of the odds:

$$\text{logit}(p) = \log \frac{p}{1-p}$$

The logit function is the inverse of the logistic sigmoid function:

$$s(\text{logit}(p)) = \frac{\exp(\text{logit}(p))}{1+\exp(\text{logit}(p))} = \frac{\frac{p}{1-p}}{1+\frac{p}{1-p}} = \frac{p}{1-p+p} = \frac{p}{1} = p$$

In statistics, logistic regression is often used to estimate the odds ratio between two events.

- We interpreted $s(f_{w,b}(x))$ as the probability of $\{Y = 1|X = x\}$ predicted by our model with parameters w and b.
- Given training samples $x_1, \ldots, x_n$ and associated class labels $y_1, \ldots, y_n \in \{0, 1\}$, we would like to find suitable values for the model parameters w and b.
- If our logistic regression model was true, then the joint probability of observing the class labels would be:

$$P(Y_1 = y_1, \ldots, Y_n = y_n | X_1 = x_1, \ldots, X_n = x_n, w, b)$$

- If our logistic regression model was true, then the joint probability of observing the class labels would be:

$$P(Y_1 = y_1, \ldots, Y_n = y_n | X_1 = x_1, \ldots, X_n = x_n, w, b)$$

- Assuming that $(x_1, y_1), \ldots, (x_n, y_n)$ are realisations of independent and identically distributed random variables $(X_1, Y_1), \ldots, (X_n, Y_n)$, this is equal to:

$$\prod_{i=1}^{n} P(y_i | x_i, w, b)$$

- In maximum likelihood estimation, we consider this as a function of parameters w and b and maximise it.

The maximum likelihood estimates of the parameters w and b given the data points $(x_1, y_1), \ldots, (x_n, y_n)$ are therefore:

$$(w^*, b^*) = \text{argmax}_{(w,b)} \prod_{i=1}^{n} P(y_i|x_i, w, b)$$

- If $y_i = 1$, then $P(y_i|x_i, w, b)$ is our model output $s(f_{w,b}(x_i))$.
- If $y_i = 0$, then $P(y_i|x_i, w, b)$ is $1 - s(f_{w,b}(x_i))$.
- This can be written as (for $y_i \in \{0, 1\}$):

$$P(y_i|x_i, w, b) = s(f_{w,b}(x_i))^{y_i}(1 - s(f_{w,b}(x_i)))^{1-y_i}$$

Inserting this into the formula for the maximum likelihood estimates gives:

$$(w^*, b^*) = \operatorname{argmax}_{(w,b)} \prod_{i=1}^{n} s(f_{w,b}(x_i))^{y_i} (1 - s(f_{w,b}(x_i)))^{1-y_i}$$

Taking the logarithm does not change the location of the maximum (and it will make optimisation easier):

$$\begin{aligned}
(w^*, b^*) &= \operatorname{argmax}_{(w,b)} \sum_{i=1}^{n} \log(s(f_{w,b}(x_i))^{y_i} * (1 - s(f_{w,b}(x_i)))^{1-y_i}) \\
&= \operatorname{argmax}_{(w,b)} \sum_{i=1}^{n} \log(s(f_{w,b}(x_i))^{y_i}) + \log(1 - s(f_{w,b}(x_i))^{1-y_i}) \\
&= \operatorname{argmax}_{(w,b)} \sum_{i=1}^{n} y_i \log(s(f_{w,b}(x_i))) + (1 - y_i) \log(1 - s(f_{w,b}(x_i)))
\end{aligned}$$

For convention, we will divide by n and minimise the negative:

$$(w^*, b^*) = \operatorname{argmin}_{(w,b)} \frac{1}{n} \sum_{i=1}^{n} -y_i \log(s(f_{w,b}(x_i))) - (1 - y_i) \log(1 - s(f_{w,b}(x_i)))$$

Minimising this expression provides the maximum likelihood estimates of the weight parameter w and the bias parameter b
... in the logistic regression model $s(f_{w,b}(x))$
... based on our data points $(x_1, y_1), \ldots, (x_n, y_n)$.

This expression is called the cross-entropy loss of our data points and is the average cross-entropy loss of the individual data points.

Definition: Cross-entropy loss for 2-class classification

The cross-entropy loss of a data point (x_i, y_i) is defined as:

$$L(x_i, y_i | w, b) = -y_i \log(s(f_{w,b}(x_i))) - (1 - y_i) \log(1 - s(f_{w,b}(x_i)))$$

For one-hot labels $y_1, \ldots, y_n \in \{0, 1\}$, the cross-entropy loss of a data point (x_i, y_i) is the negative logarithm of the predicted probability of the ground-truth class:

$$L(x_i, y_i | w, b) = \begin{cases} -\log(s(f_{w,b}(x_i))) & \text{if } y_i = 1 \\ -\log(1 - s(f_{w,b}(x_i))) & \text{if } y_i = 0 \end{cases}$$

Does such a log-probability make sense as a loss?

We derived that we want to minimise the loss function:

$$(w^*, b^*) = \operatorname{argmin}_{(w,b)} \frac{1}{n} \sum_{i=1}^{n} -y_i \log(s(f_{w,b}(x_i))) - (1 - y_i) \log(1 - s(f_{w,b}(x_i)))$$

$$= \operatorname{argmin}_{(w,b)} \frac{1}{n} \sum_{i=1}^{n} L(x_i, y_i | w, b)$$

Let's use gradient descent.

The gradient of the logistic sigmoid function is:

$$\frac{\partial s(x)}{\partial x} = \frac{\partial}{\partial x}\frac{1}{\exp(-x)+1} = \frac{\exp(-x)}{(\exp(-x)+1)^2}$$
$$= \frac{1}{\exp(-x)+1}\left(\frac{\exp(-x)+1-1}{\exp(-x)+1}\right) = s(x)(1-s(x))$$

Thus:

$$\frac{\partial \log(s(x))}{\partial x} = \frac{1}{s(x)}\frac{\partial s(x)}{\partial x} = 1 - s(x)$$
$$\frac{\partial \log(1-s(x))}{\partial x} = -1 * \frac{1}{1-s(x)}\frac{\partial s(x)}{\partial x} = -s(x)$$

With $f_{w,b}(x) = w \cdot x + b$, we get:

$$\begin{aligned}
\frac{\partial L}{\partial w} &= \frac{\partial}{\partial w}\frac{1}{n}\sum_{i=1}^{n} -y_i \log(s(f_{w,b}(x_i))) - (1-y_i)\log(1 - s(f_{w,b}(x_i))) \\
&= \frac{\partial}{\partial w}\frac{1}{n}\sum_{i=1}^{n} -y_i \log(s(w \cdot x_i + b)) - (1-y_i)\log(1 - s(w \cdot x_i + b)) \\
&= \frac{1}{n}\sum_{i=1}^{n} -x_i y_i (1 - s(w \cdot x_i + b)) + x_i(1-y_i)s(w \cdot x_i + b) \\
&= \frac{1}{n}\sum_{i=1}^{n} -x_i y_i + x_i s(w \cdot x_i + b) = \frac{1}{n}\sum_{i=1}^{n} x_i(s(w \cdot x_i + b) - y_i) \\
\frac{\partial L}{\partial b} &= \frac{1}{n}\sum_{i=1}^{n} s(w \cdot x_i + b) - y_i
\end{aligned}$$

The gradient of a data point (x_i, y_i) is thus given by:

$$\frac{\partial L(x_i, y_i|w, b)}{\partial w} = x_i(s(w \cdot x_i + b) - y_i)$$
$$\frac{\partial L(x_i, y_i|w, b)}{\partial b} = s(w \cdot x_i + b) - y_i$$

This holds also for vector-shaped $w = (w_1, \ldots, w_d, \ldots, w_D)$

Observation 1:

Nice interpretation: The partial derivatives are the difference between the predicted value $s(w \cdot x_i + b)$ and the true value $y_i \in \{0, 1\}$, with or without weighting by the sample x_i.

Observation 2:
If both classes in the data can be perfectly separated by a linear mapping, then optimisation will try to make the weights w to go to infinity. This is a source of instability during optimisation time.

- If $y_i = 1$, then the loss for the data point (x_i, y_i) is $-\log s(f_{w,b}(x_i))$.
- For $-\log s(f_{w,b}(x_i)))$ to be 0, then $s(f_{w,b}(x_i))$ need to be 1.
- For $s(f_{w,b}(x_i)) = \frac{1}{\exp(-f_{w,b}(x_i))+1}$ to be 1, then $\exp(-f_{w,b}(x_i))$ need to be 0.
- For $\exp(-f_{w,b}(x_i))$ to be 0, then $f_{w,b}(x_i) = w \cdot x_i + b \to \infty$.
- For $f_{w,b}(x_i) = w \cdot x_i + b \to \infty$ to happen, one can use: $c(w \cdot x_i + b) = cw \cdot x_i + cb,\ c \to \infty$

Observation 2:
If both classes in the data can be perfectly separated by a linear mapping, then optimisation will try to make the weights w to go to infinity. This is a source of instability during optimisation time.

- For data points where $y_i = 0$, a loss $-\log(1 - s(f_{w,b}(x_i))) = 0$ implies that $w \cdot x_i + b$ needs to go to $-\infty$.
- For linearly separable problems, the total loss will thus approach 0 as $|w|$ and $|b|$ approach infinity (and w is appropriately oriented). The optimisation may therefore result in numerical instability.
- So making $P(Y = y_i|X = x_i) = 1$ for all data points requires increasing $|w|$ (and $|b|$ if $b \neq 0$) to a very large value.

Why this does not happen when the training dataset is not linearly separable?

- No matter the orientation of w, there will be always a data point (x_i, y_i) that is misclassified.
- Suppose $y_i = 1$ so that the loss of this sample is $-\log s(f_{w,b}(x_i))$.
- We can simulate $|w| \to \infty$, $|b| \to \infty$ by using as weight cw, cb and let go $c \to \infty$.
- Because (x_i, y_i) is misclassified, $f_{w,b}(x_i) = cwx_i + cb < 0$, and thus $\lim_{c\to\infty} cwx_i + cb = -\infty$.

Why this does not happen when the training dataset is not linearly separable?

- No matter the orientation of w, there will be always a data point (x_i, y_i) that is misclassified.
- Because $(x_i, y_i = 1)$ is misclassified, $f_{w,b}(x_i) = cwx_i + cb < 0$, and thus $\lim_{c\to\infty} cwx_i + cb = -\infty$.
- Therefore, the predicted probability of the correct class $s(f_{w,b}(x_i)) \underbrace{\to}_{c\to\infty} 0$, making the loss $-\log s(f_{w,b}(x_i))$ go to infinity.
- Because the loss is always non-negative, this single misclassification would make the total loss go to infinity if $|w|, |b|$ goes to infinity.
- Thus, the absolute value of the weight will not go to infinity if the training dataset is not linearly separable.

Why this does not happen when the training dataset is not linearly separable?

- No matter the orientation of w, there will be always a data point (x_i, y_i) that is misclassified.
- There will instead be a trade-off between pushing $P(Y = y_i|X = x_i)$ close to 1 for correct classified samples and not pushing it close to 0 for misclassified samples.

Key takeaways:

- Applying the logistic sigmoid function to a linear mapping provides a logistic regression model.
- Outputs a score in $[0, 1]$ which can be interpreted as the predicted probability $P(Y = 1|X = x)$ of the class labelled 1.
- One minus score $(1 - s(x))$ is our predicted probability of class label 0.
- Logistic regression is therefore a classification algorithm.
- The cross-entropy loss can be derived as the function to optimise by maximum likelihood estimation.
- The cross-entropy loss is the sum of the negative logarithms of the predicted probabilities weighted by the ground-truth distribution.
- For one-hot labels, this becomes the negative logarithm of the predicted probability of the ground-truth class.

1 Classification by logistic regression

2 Artificial neurons

3 Neural networks

4 Example: The XOR problem

5 More on representability

What is an artificial neuron?

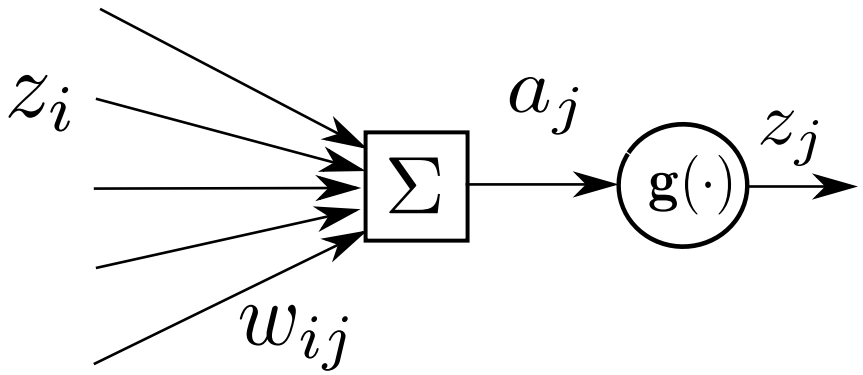
A mathematical function that for instance can:

- Take some inputs $\{z_i\}_{i=1}^{d}$.
- Depend on some weights w_{ij} and bias b_j.
- Apply some non-linear activation function $g(\cdot)$.
- Output z_j.

Forward equation:

$$a_j = \sum_{i=1}^{d} z_i w_{ij} + b_j \qquad \text{linear mapping}$$

$$z_j = g(a_j) \qquad \text{nonlinear activation}$$

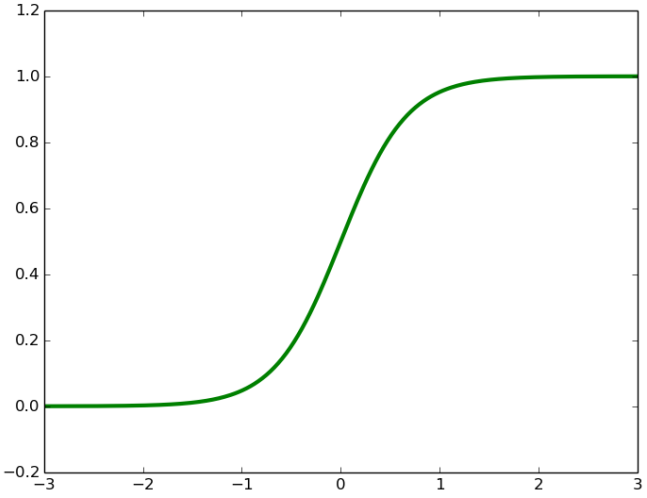


Activation functions

- A function $g : \mathbb{R}^1 \longrightarrow \mathbb{R}^1$
- Introduces non-linearity.
- Should be differentiable if you are planning to use gradient-based optimisation.
- Typically prevents or reduces weak signals from passing through.
- Intuition: Determines what the neuron fires for different outputs of the linear mapping.

We could for instance use the logistic sigmoid function:

$$g(x) = \frac{1}{\exp(-x) + 1}$$



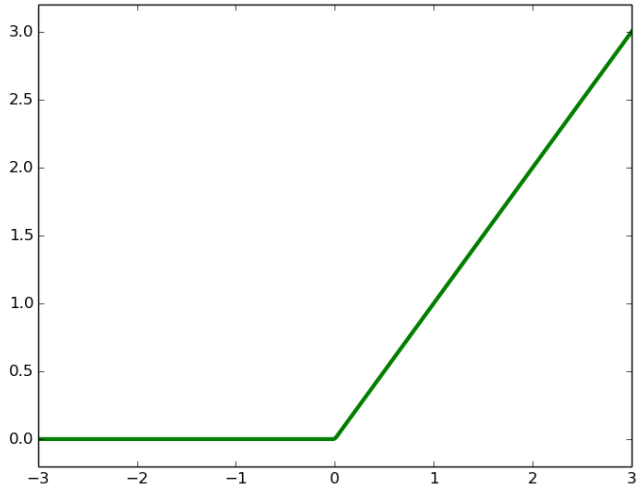
- Called sigmoid activation.
- Historically popular.
- Currently rarely used.

Suffer from gradient saturation: its derivative gets exponentially small for large $|x|$:

$$\frac{\partial s(x)}{\partial x} = \frac{e^{-x}}{(e^{-x}+1)^2} \leq \begin{cases} e^{-x} & x \geq 0 \\ \frac{1}{e^{-x}+1} & x < 0 \end{cases}$$

An alternative is to simply set negative values to 0, i.e. let:

$$g(x) = \max(0, x)$$

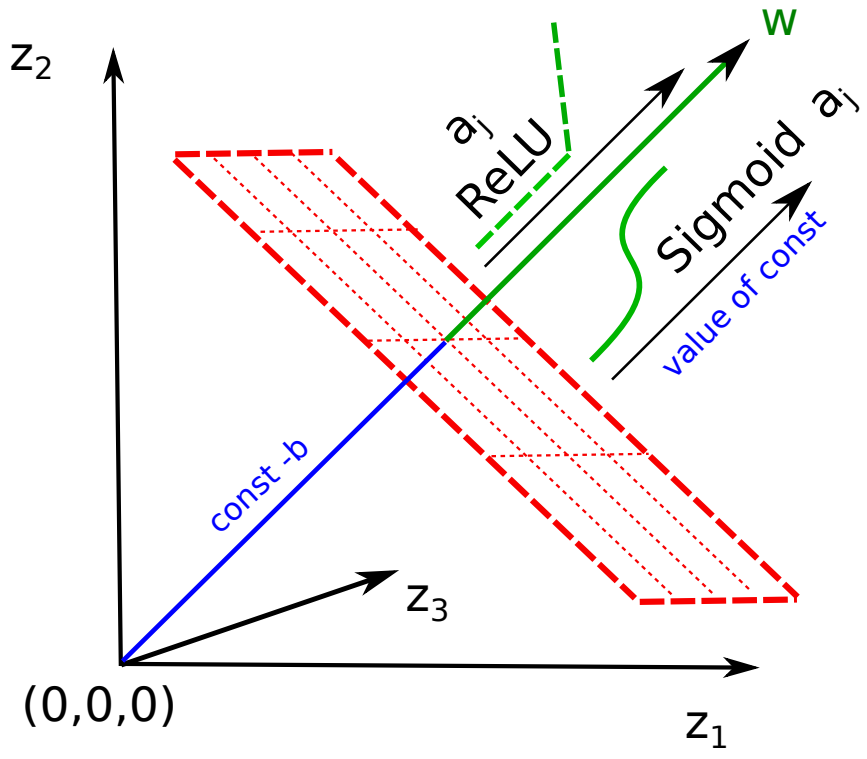


- Called ReLU activation.
- Gradient do not saturate.
- Not differentiable in 0.
  - Not of practical concern.
  - Could define it to be 1.

The gradient is 0 for negative inputs.

- Might cause inactive nodes to remain inactive.
- Might still work fine in practice.
- There exists alternatives with non-zero gradient for negative inputs, e.g. leaky ReLU: $g(x) = \max(0.01x, x)$

A popular choice, particularly with convolutional neural networks.



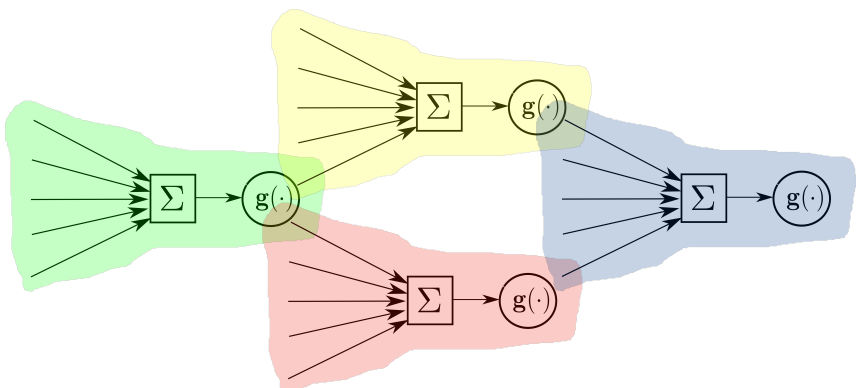
- The weights w defines the orientation of the plane.
- The bias b defines the position of the plane.
- Output of activation function is constant in the red plane.
- Output of activation function varies along the direction of w.

Outline

1. Classification by logistic regression
2. Artificial neurons
3. Neural networks
4. Example: The XOR problem
5. More on representability

A neural network is a directed graph structure made from connected neurons.

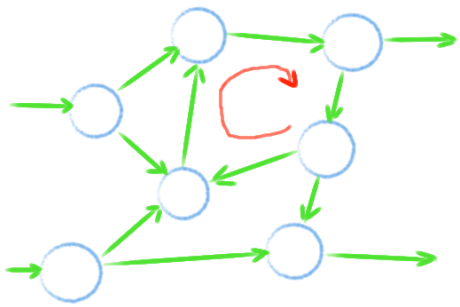
- A neuron can be input to many other neurons.
- A neuron can receive input to many other neurons.
- Each neuron has same structure $(g(\cdot), \sum)$ but different parameters (w_{ij}, b_j).
- Can stack neurons in layers.



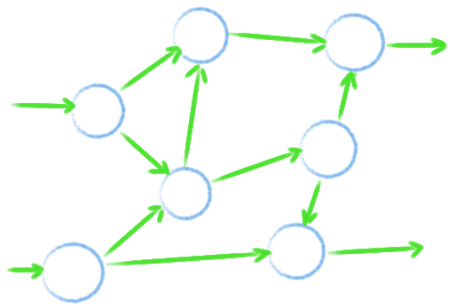
Definition: Neural Network

Any directed graph built from neurons is a neural network.

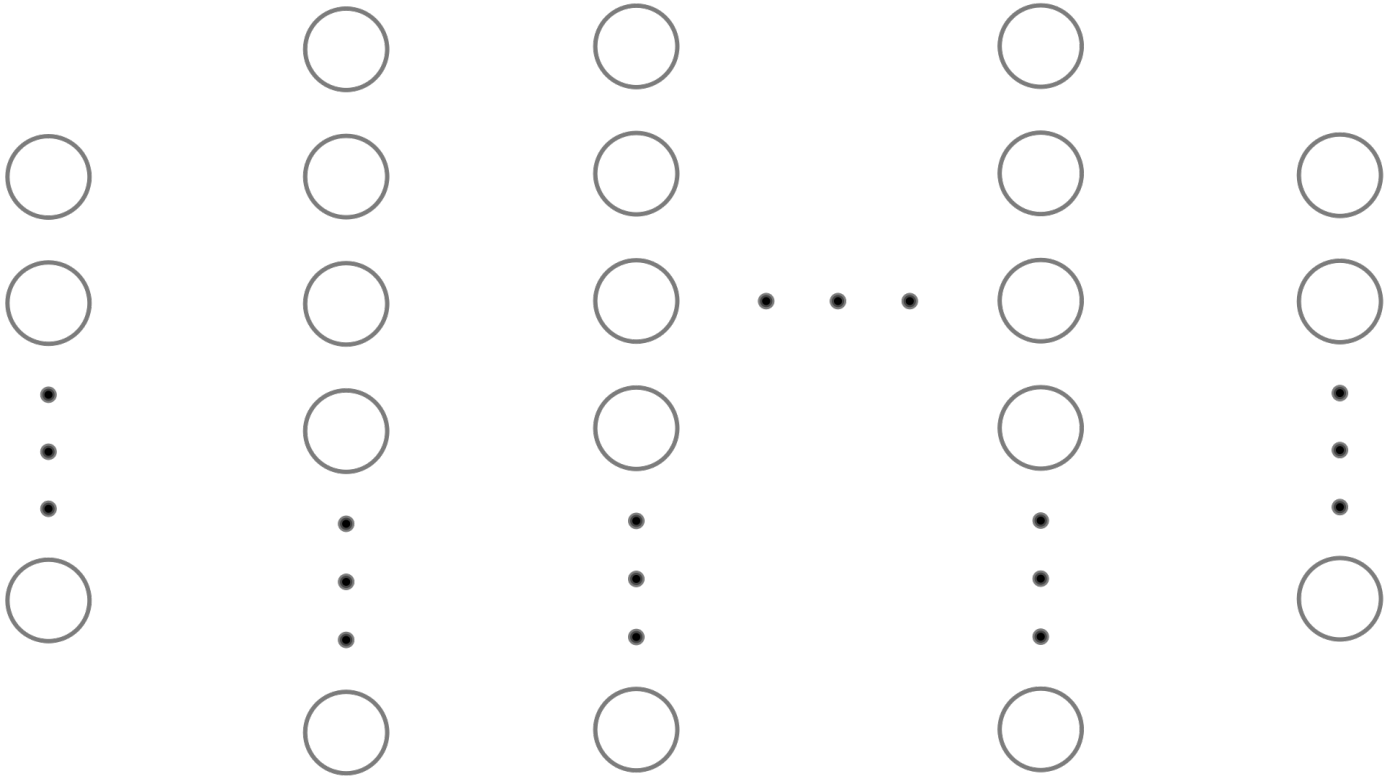
Two important types: recurrent and feedforward neural networks



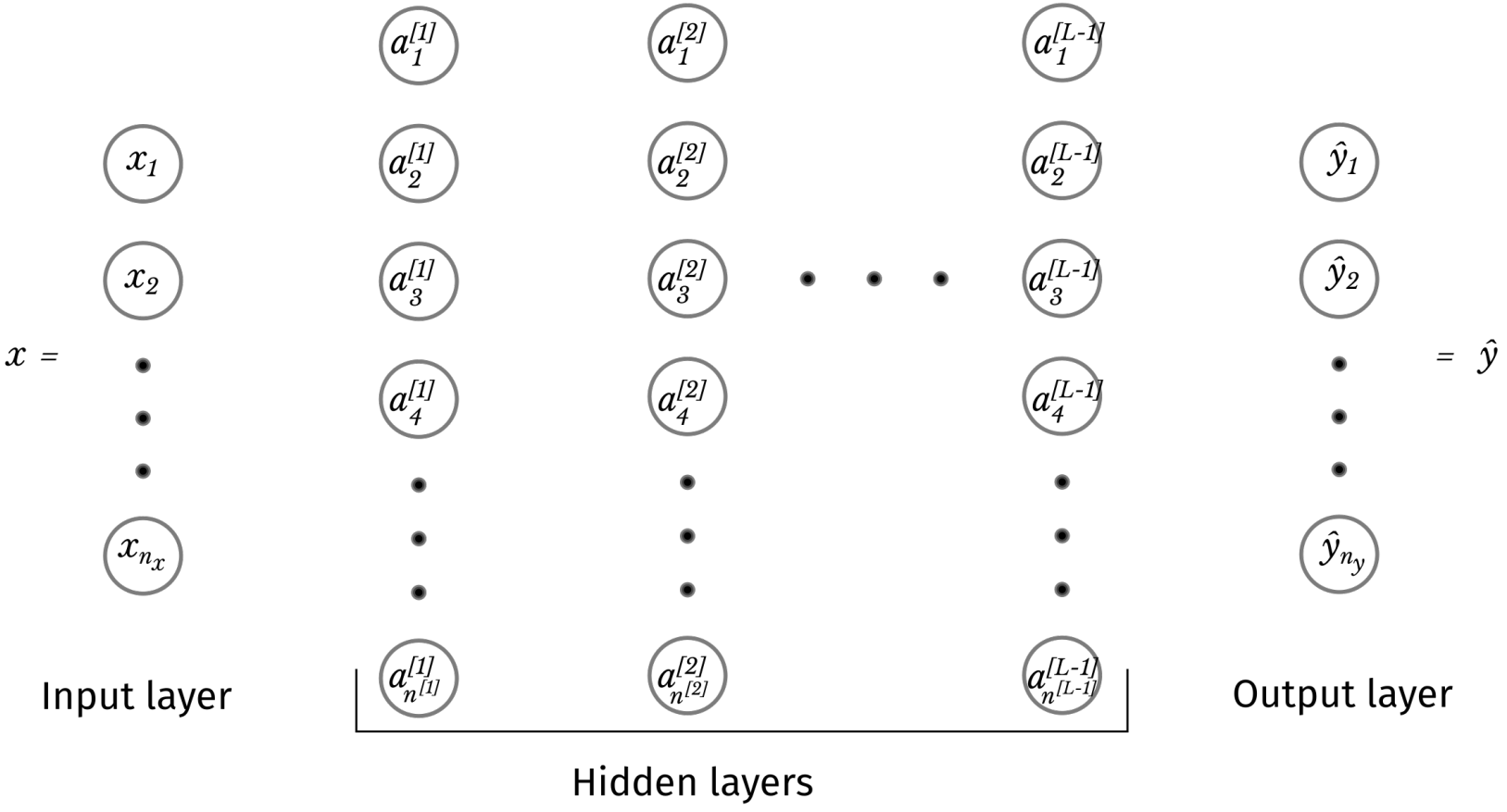
recurrent (not covered in this lecture)
e.g. for speech processing



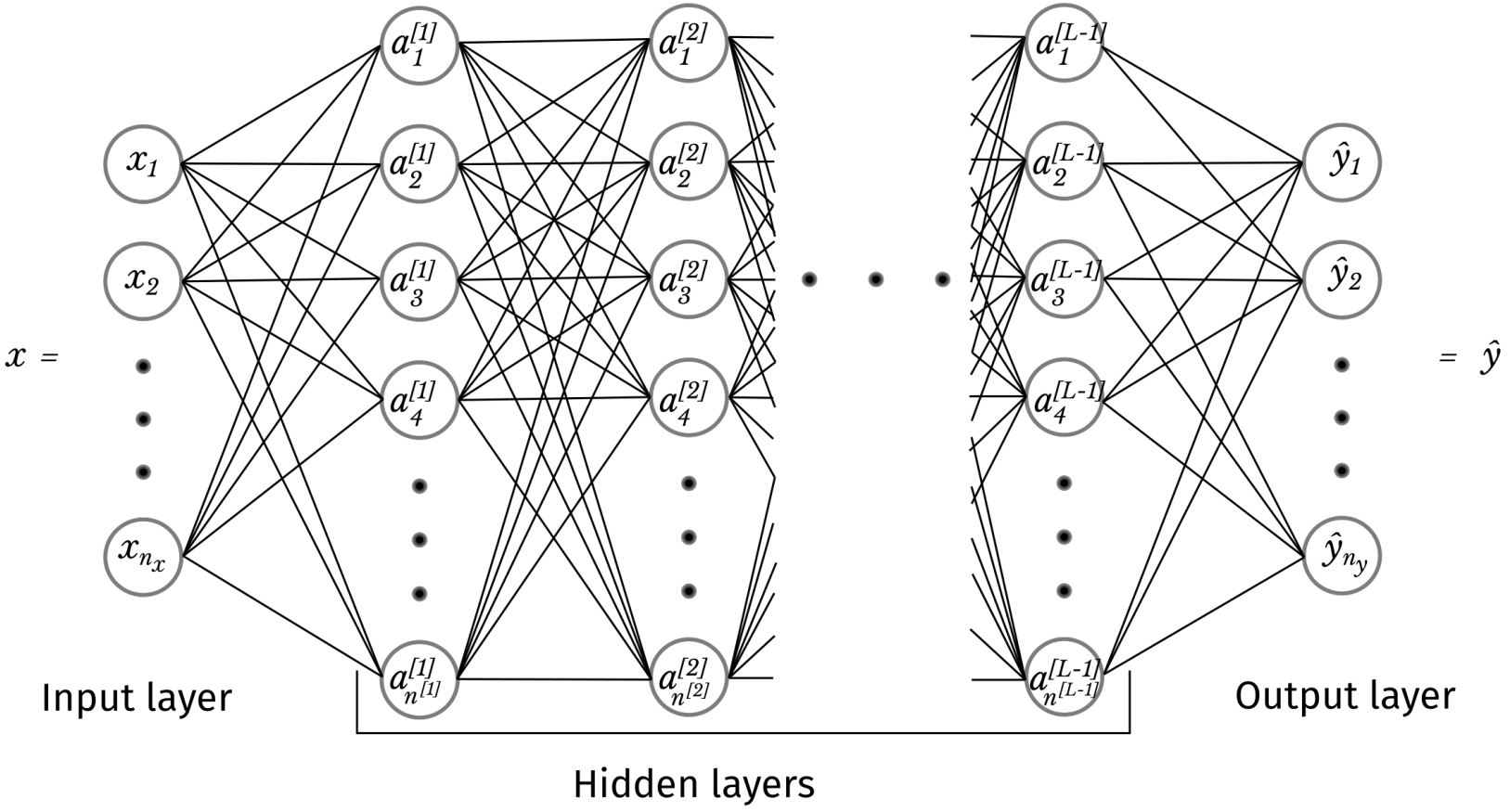
feedforward
e.g. for image classification



Nodes and layers



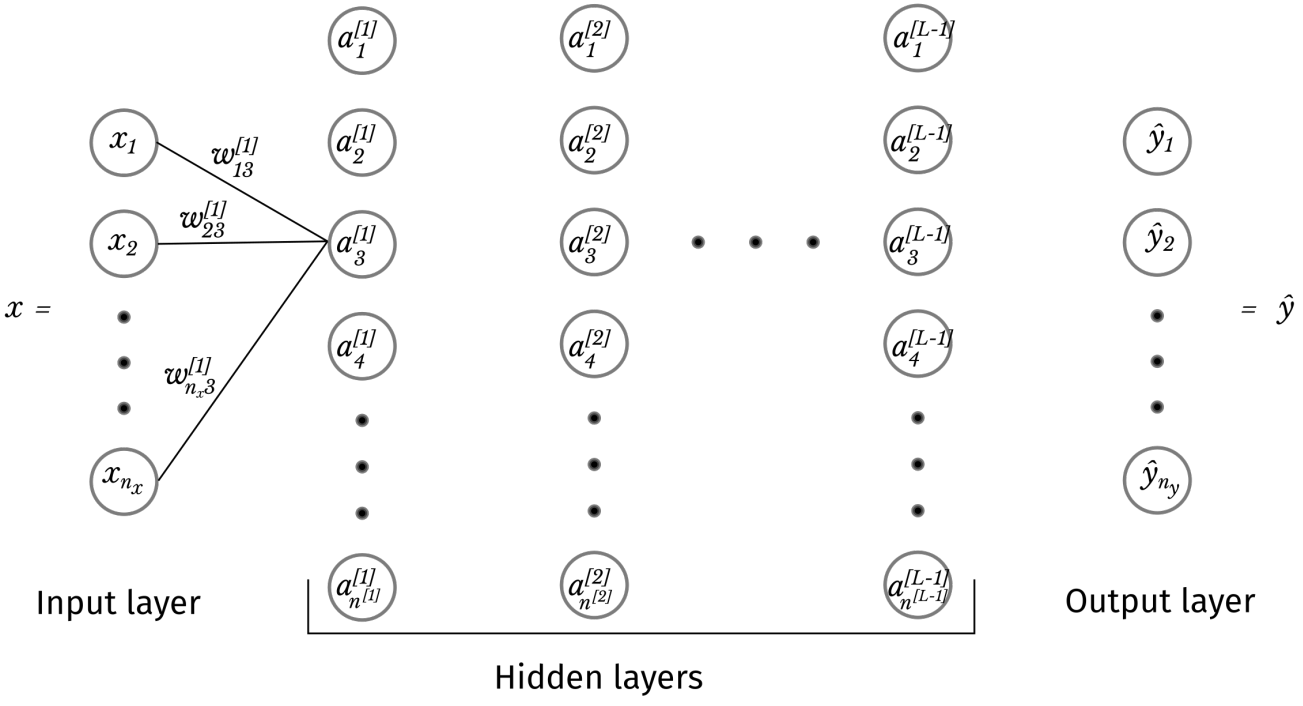
Dense (fully-connected) feedforward neural network



Notation — for reference

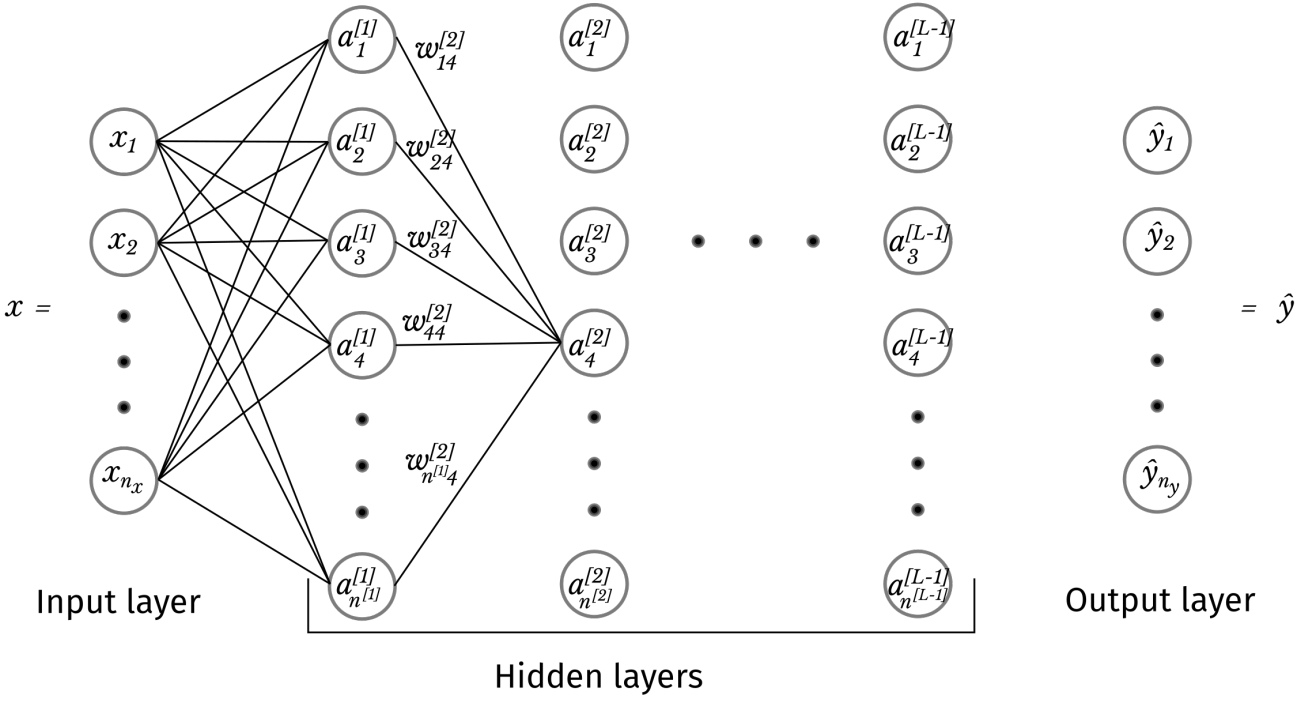
- Superscript with brackets (l): layer l
- L: Number of layers in the network.
- $n^{(l)}$: Number of nodes in layer l
- $n_x = n^{(0)}$: Input dimension
- $n_y = n^{(L)}$: Output dimension (number of classes)
- x: Array of inputs
- y: Array of *true* outputs
- $\hat{y}$: Array of *predicted* output
- w: Edge weights
- b: Node bias
- z: Linear combination of activations from previous layer
- $a^{(l)}$: Node activation in layer l.
- $a^{(0)} = x$: Input vector
- $a^{(L)} = \hat{y}$: Output vector
- Subscript j or jk: Element in vector or matrix
- m: Number of samples in the training dataset

Activation in node 3 of layer 1

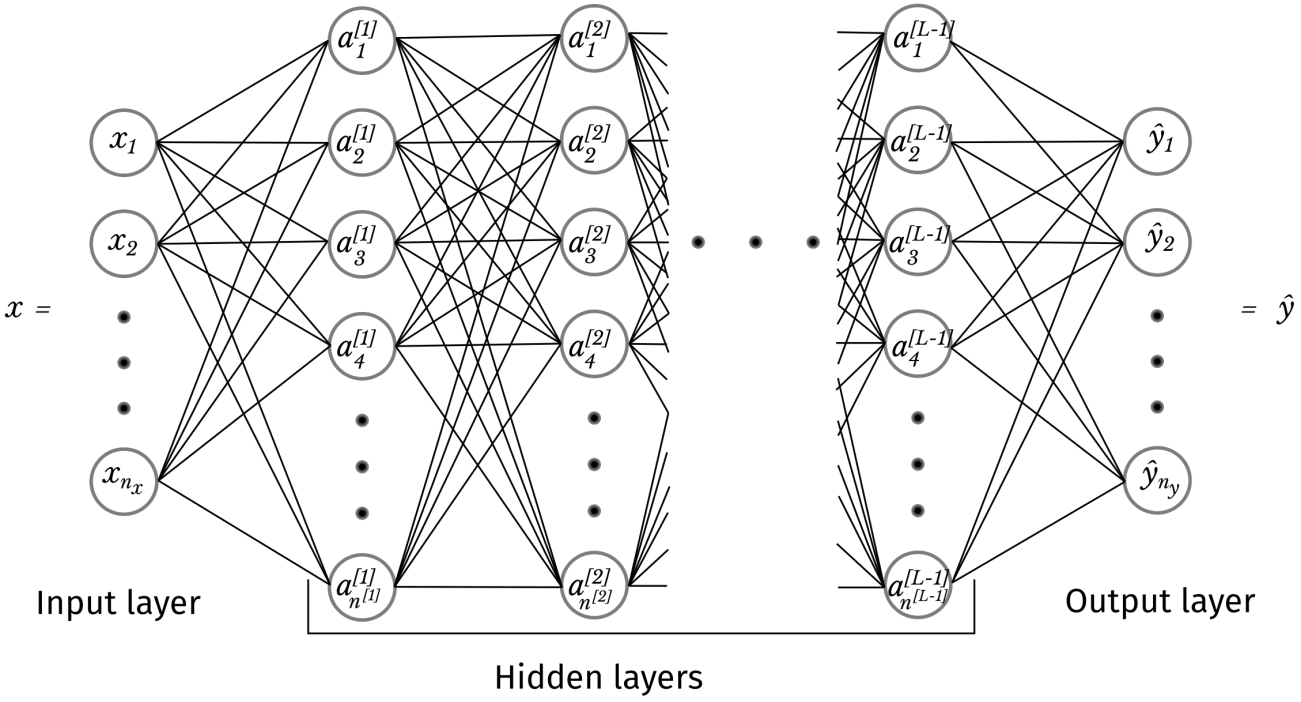


$$a_3^{(1)} = g\left(\sum_{j=1}^{n_x} w_{j3}^{(1)} x_j + b_3^{(1)}\right)$$

Activation in node 4 of layer 2

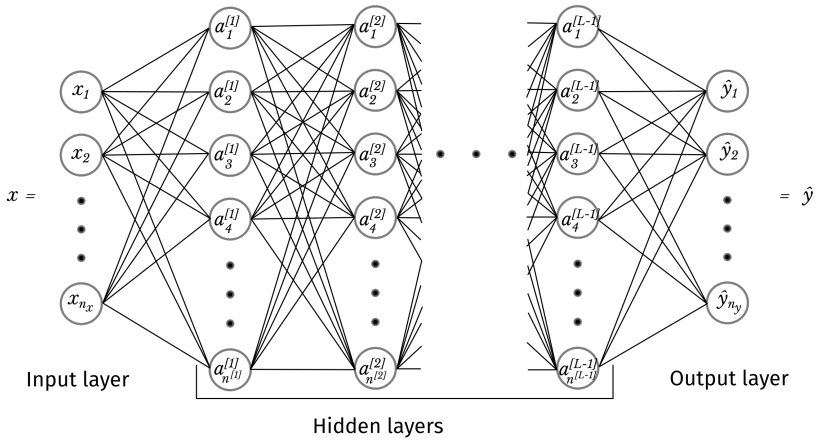


$$a_4^{(2)} = g\left(\sum_{j=1}^{n^{(1)}} w_{j4}^{(2)} a_j^{(1)} + b_4^{(2)}\right)$$

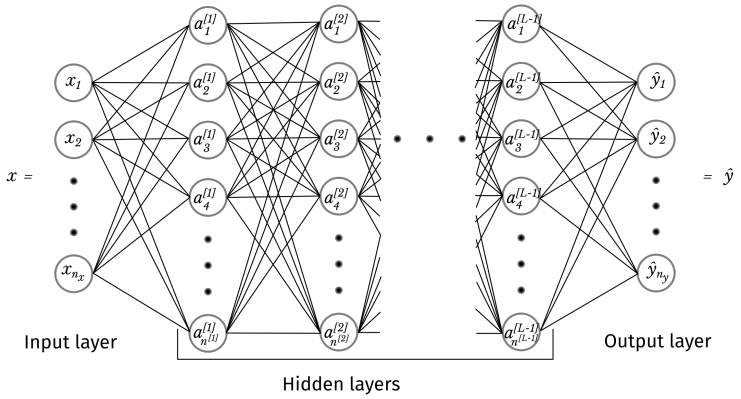


$$a_k^{(l)} = g\left(\sum_{j=1}^{n^{(l-1)}} w_{jk}^{(l)} a_j^{(l-1)} + b_k^{(l)}\right), k \in \{1, 2, \ldots, n^{(l)}\}, l \in \{1, 2, \ldots, L\}$$

Too many indices ?



$$a_k^{(l)} = g\left(\sum_{j=1}^{n^{(l-1)}} w_{jk}^{(l)} a_j^{(l-1)} + b_k^{(l)}\right)$$
$$= g\left(\boldsymbol{w}_{[:,k]}^{(l)} \cdot \boldsymbol{a}^{(l-1)} + b_k^{(l)}\right), k \in \{1, 2, \ldots, n^{[l]}\}, l \in \{1, 2, \ldots, L\}$$



$$a_k^{(l)} = g\left(\sum_{j=1}^{n^{(l-1)}} w_{jk}^{(l)} a_j^{(l-1)} + b_k^{(l)}\right)$$
$$= g\left(\boldsymbol{w}_{[:,k]}^{(l)} \cdot \boldsymbol{a}^{(l-1)} + b_k^{(l)}\right), k \in \{1, 2, \ldots, n^{[l]}\}, l \in \{1, 2, \ldots, L\}$$

output $a_k^{(l)}$ for a single neuron is $g(z_k^{(l)})$ where $z_k^{(l)}$ is the result from an inner product between

- the activation vector $\boldsymbol{a}^{(l-1)}$ of the previous layer $l-1$
- the k-th slice $\boldsymbol{w}_{[:,k]}^{(l)}$ of the weight matrix $\boldsymbol{w}_{[:,:]}^{(l)}$ of the current layer l
- added a bias term $b_k^{(l)}$ to the inner product

- $a_k^{(l)}$ is the *activation* of node k in layer l:

$$a_k^{(l)} = g\left(\sum_{j=1}^{n^{(l-1)}} w_{jk}^{(l)} a_j^{(l-1)} + b_k^{(l)}\right) = g\left(\boldsymbol{w}_{[:,k]}^{(l)} \cdot \boldsymbol{a}^{(l-1)} + b_k^{(l)}\right)$$

- $w_{jk}^{(l)}$ is the *weight* from node j in layer $l-1$ to node k in layer l.
- $b_k^{(l)}$ is the *bias* of node k in layer l.
- g is the activation function.

- $a_k^{(l)}$ is the *activation* of node k in layer l:

$$a_k^{(l)} = g\left(\sum_{j=1}^{n^{(l-1)}} w_{jk}^{(l)} a_j^{(l-1)} + b_k^{(l)}\right) = g\left(\boldsymbol{w}_{[:,k]}^{(l)} \cdot \boldsymbol{a}^{(l-1)} + b_k^{(l)}\right)$$

- All w and b are "trainable", and will be adjusted according to some optimization routine.
- By convention:
  - $a_k^{(0)} = x_k$ and $a_k^{(L)} = \hat{y}_k$
  - The network have L layers (we do not include the input layer in the count) and $L-1$ hidden layers.

- Applying an activation function at the last layer will in general not provide outputs that can be interpreted as probabilities.
- For 2-class classification, we could have applied the logistic sigmoid function instead of an activation function to produce probabilities from the single output neuron.
- We will instead apply a generalisation of the logistic sigmoid function called the softmax function:

$$s(x)_k = \frac{e^{x_k}}{\sum_{i=1}^{n} e^{x_i}}$$

- This function converts the linearly mapped values in the output layer, $z_1, \ldots, z_{n_y}$, to predicted probabilities.

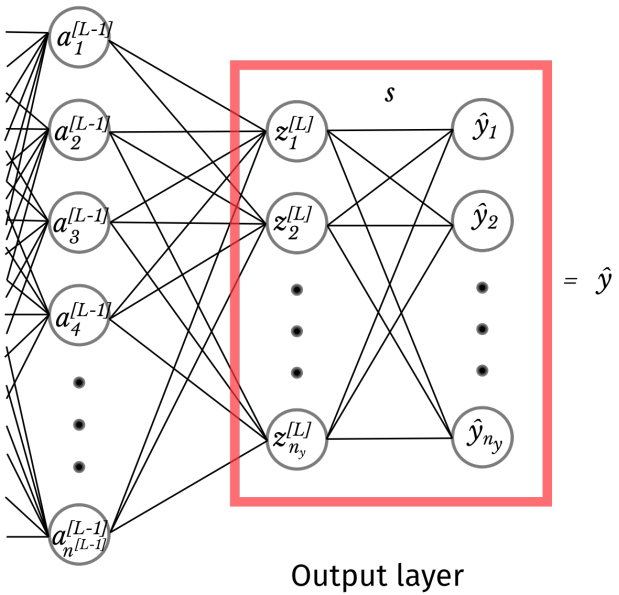
In the output layer we have:

$$\begin{aligned} a_k^{(L)} &= s(z^{(L)})_k \\ &= \hat{y}_k \end{aligned}$$

for:

$$\begin{aligned} k &= 1, \ldots, n_y, \\ &= 1, \ldots, n^{(L)}. \end{aligned}$$

$z^{(L)}$ are called *logits*, and $\hat{y}$ will be interpreted as the predicted probabilities.



Output layer

Implementing the softmax function

Numerical instability can be a problem when using the softmax function, because of the exponential function and division.

Two common "tricks" that can help:

1. Shift exponential arguments to max zero:

$$s(x)_k = \frac{e^{x_k}}{\sum_{i=1}^{n} e^{x_i}} = \frac{e^{x_k - \max(x)}}{\sum_{i=1}^{n} e^{x_i - \max(x)}}$$

2. Take logarithm and exponentiate it to get rid of division:

$$t(x)_k = \log s(x)_k = x_k - \log \sum_{i=1}^{n} e^{x_i}$$

$$s(x)_k = e^{t(x)_k}$$

It is possible to combine these two "tricks".

The derivation and definition of cross-entropy loss can be extended to any model's predicted probabilities $\hat{y}_i \in [0,1]^{n_y}$ associated with input sample $x_i \in \mathbb{R}^{n_x}$ with true class label $y_i \in [0,1]^{n_y}$:

Definition: Cross-entropy loss

The cross-entropy loss between two probability distribution vectors $(y_1, \ldots, y_i, \ldots, y_{n_y})$ and $(\hat{y}_1, \ldots, \hat{y}_i, \ldots, \hat{y}_{n_y})$ is defined as:

$$L(\hat{y}_i, y_i) = -\sum_{k=1}^{n_y} (y_i)_k \log(\hat{y}_i)_k$$

For one-hot labels, i.e. each y_i is 0 in all elements except one where it is 1, the cross-entropy loss of a data point (x_i, y_i) is the negative logarithm of the predicted probability of the ground-truth class:

$$L(\hat{y}_i, y_i) = -\log(\hat{y}_i)_k \text{ where } y_i = e_k$$

- As for the logistic regression model, we can use gradient descent to optimise the cross-entropy loss function:

$$\Theta^* = \text{argmin}_\Theta \left\{ -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{n_y} (y_i)_k \log(\hat{y}_i)_k \right\}.$$

- However, we have a lot more parameters this time:

$$\begin{aligned}
\Theta &= \{w_{jk}^{[l]}, b_k^{[l]} : j \in \{1, \ldots, n^{[l-1]}\}, k \in \{1, \ldots, n^{[l]}\}, l \in \{1, \ldots, L\}\} \\
&= \{\boldsymbol{w}_{[:,k]}^{[l]}, b_k^{[l]}, k \in \{1, \ldots, n^{[l]}\}, l \in \{1, \ldots, L\}\} \\
&= \{\boldsymbol{w}^{[l]}, \boldsymbol{b}^{[l]}, l \in \{1, \ldots, L\}\}
\end{aligned}$$

- How to do this in practice will be the subject of a later lecture.

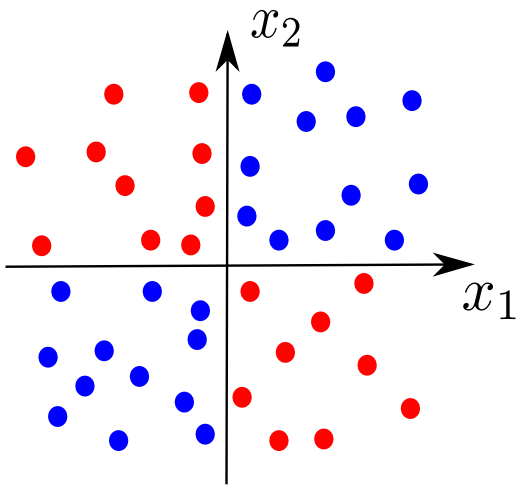
Key takeaways:

- Logistic regression is an exemplary model for a single neuron network, that is, a 1-layer neural network (0 hidden layers) with one node in the output layer.
- Stacking multiple layers gives a deep neural network.
- Each neuron (in hidden layers) in dense feedforward neural networks receives input from all neurons in the previous layer and sends its activation (output) to all neurons in the next layer.
- Note: The particular choice of network architecture is often subordinate to the selecting appropriate loss function, algorithm to updating weights, data augmentation and more.

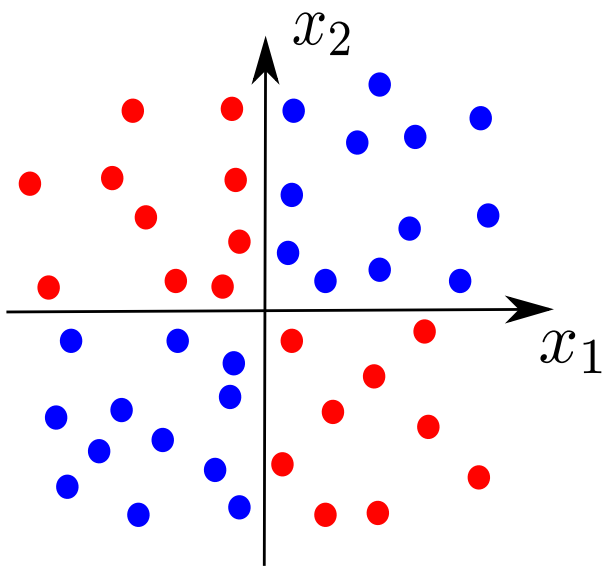
Outline

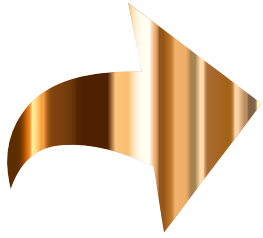
1. Classification by logistic regression
2. Artificial neurons
3. Neural networks
4. Example: The XOR problem
5. More on representability

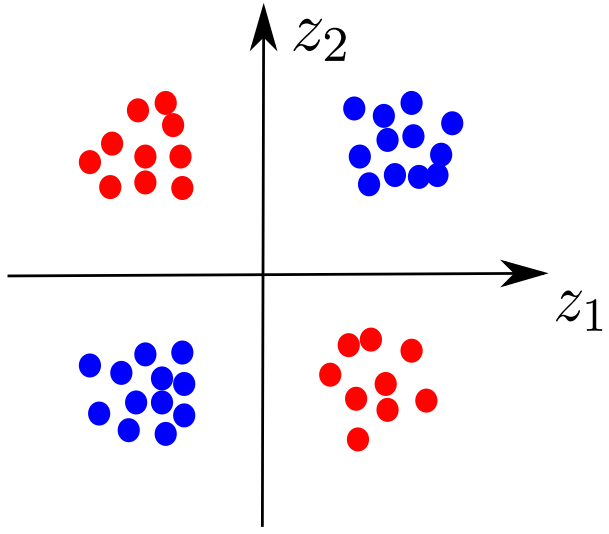
To illustrate how concatenation of neurons allows learning non-linear mappings, consider separating red from blue samples in the following example:



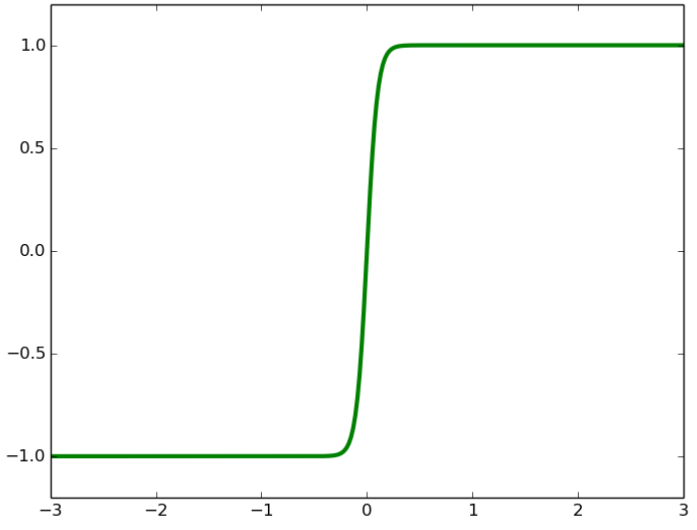
Samples lie in quadrants around coordinated $(\pm 1, \pm 1)$.



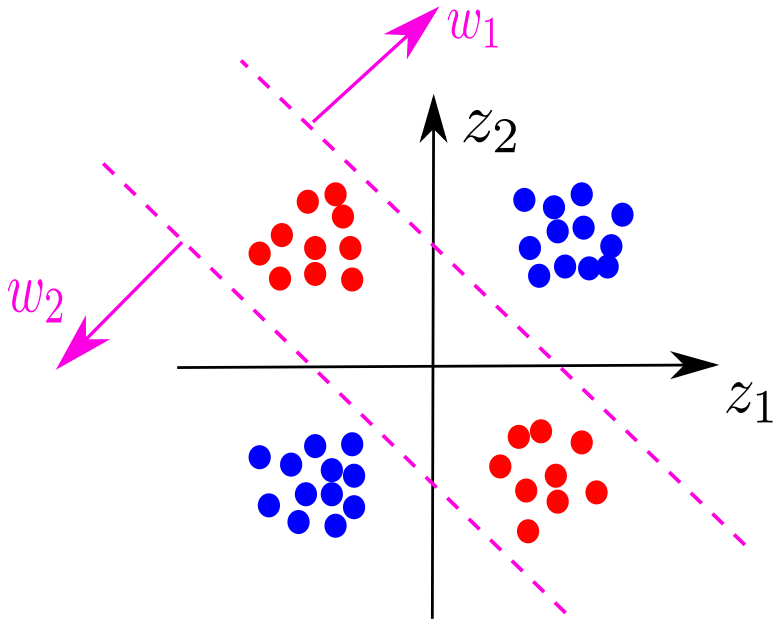




$$z_1 = tanh(10x_1)$$
$$z_2 = tanh(10x_2)$$

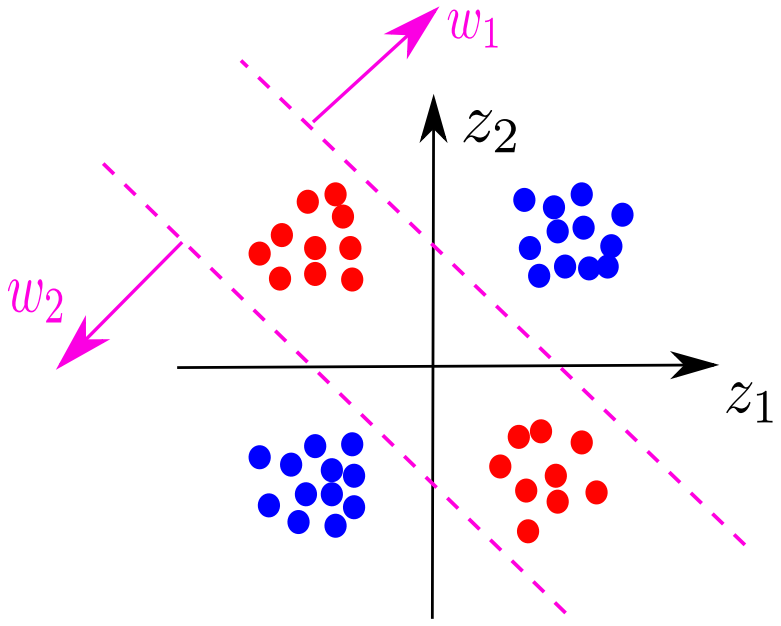


- *tanh* pushes points towards $(\pm 1, \pm 1)$

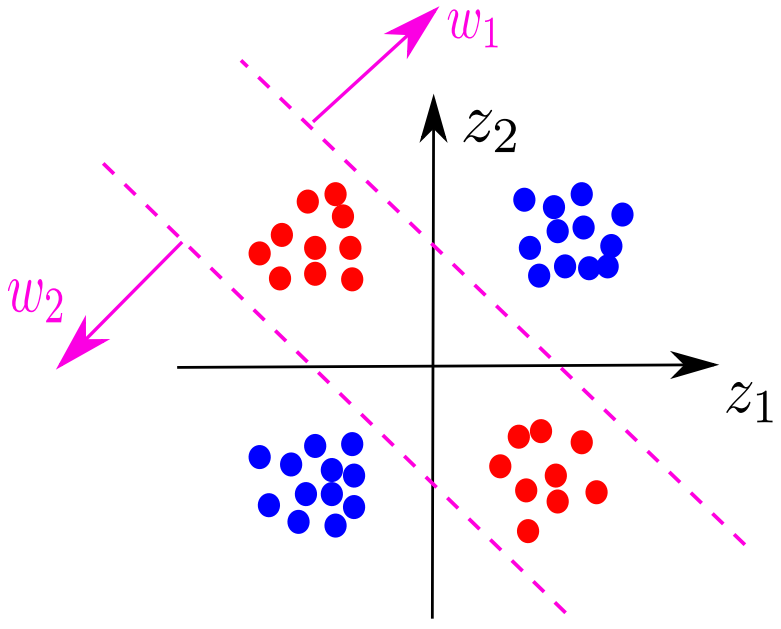


Idea:

- For proper choice of w_1 and b_1, $g_1 = \sigma(w_1 z + b_1)$ has high values ≈ 1 in the upper right $z \approx (1, 1)$ and low values ≈ 0 elsewhere.
- For proper choice of w_2 and b_2, $g_2 = \sigma(w_2 z + b_2)$ has high values ≈ 1 in the lower left $z \approx (-1, -1)$ and low values ≈ 0 elsewhere.
- Then, $g_1 + g_2$ will be ≈ 1 in the upper right and lower left corners.
- In the middle zone, $g_1 + g_2$ will be ≈ 0.



- $w_1 = ?,\ w_2 = ?$

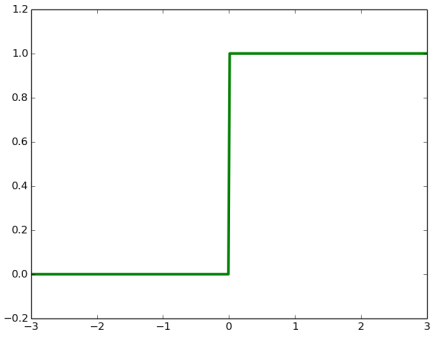


- If $w_1 = (1, 1)$, $w_2 = (-1, -1)$, then $w_1 \cdot z \approx 0$, $w_2 \cdot z \approx 0$ for middle zone, while $w_1 \cdot z \approx 2$ in the upper right corner and $w_2 \cdot z \approx 2$ in the lower left corner.
- Could then choose $-2 < b_1 < 0$ and $-2 < b_2 < 0$.
- Final output: $f(x) = \sigma(c(g_1 + g_2) + cb_3)$

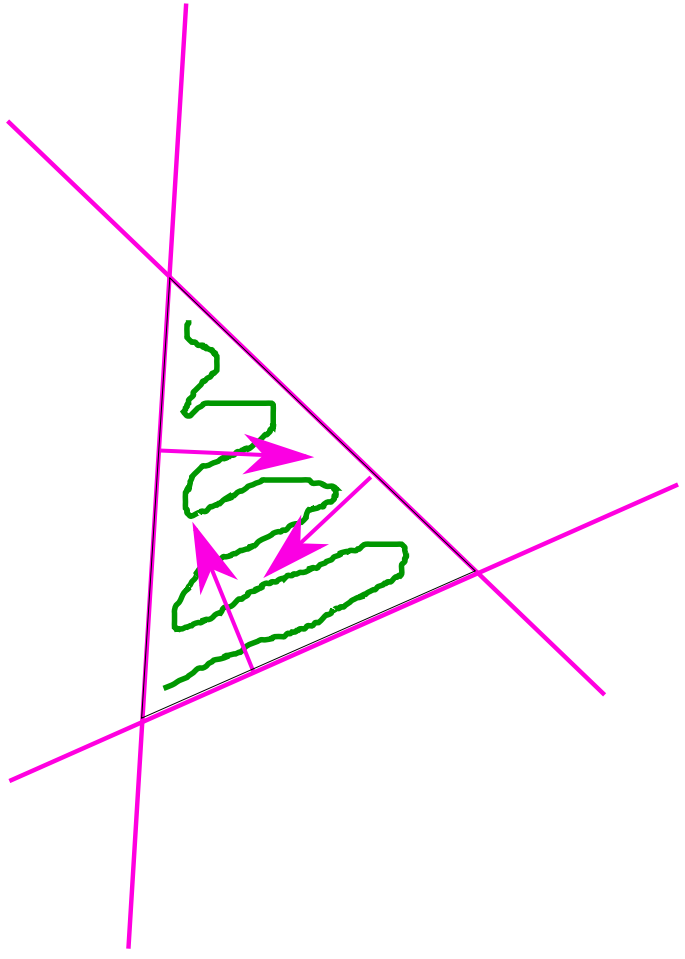
Outline

1. Classification by logistic regression
2. Artificial neurons
3. Neural networks
4. Example: The XOR problem
5. More on representability

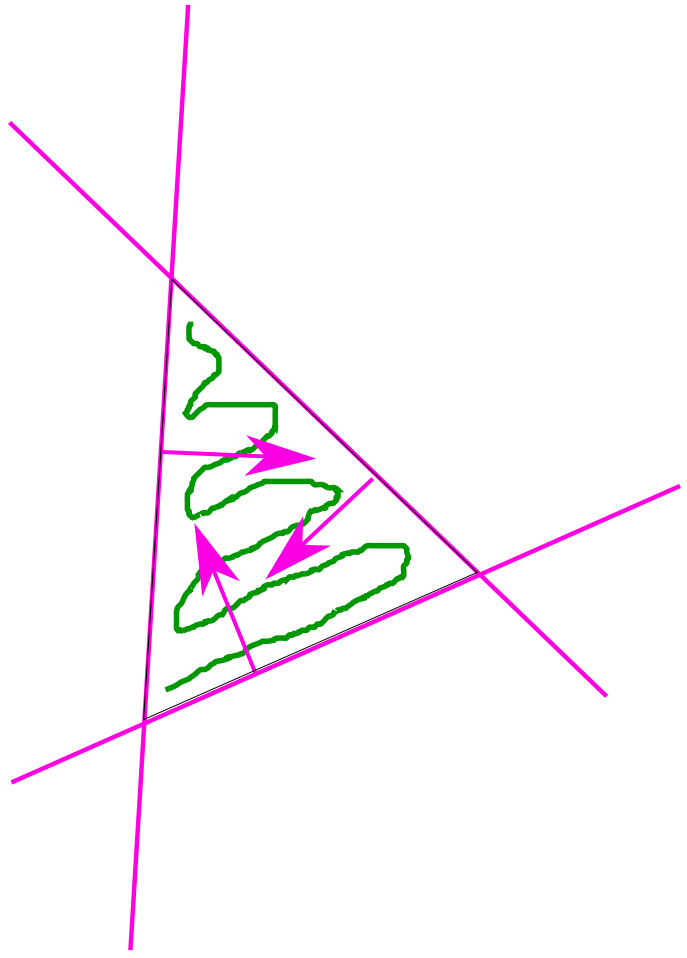
- Use threshold activation (for simplicity):



- A 2-layer network with n neurons in the hidden layer, and a thresholded sum in the second layer can represent any convex polygonal shape with n edges.
- With $n \to \infty$ neurons in the first layer, any convex shape can be **approximately** encoded.



- Adding a third layer allows the neural network to approximately encode any union of convex shapes.
- However, being able to (approximately) represent does not imply that this would be the result when learning from finite data.



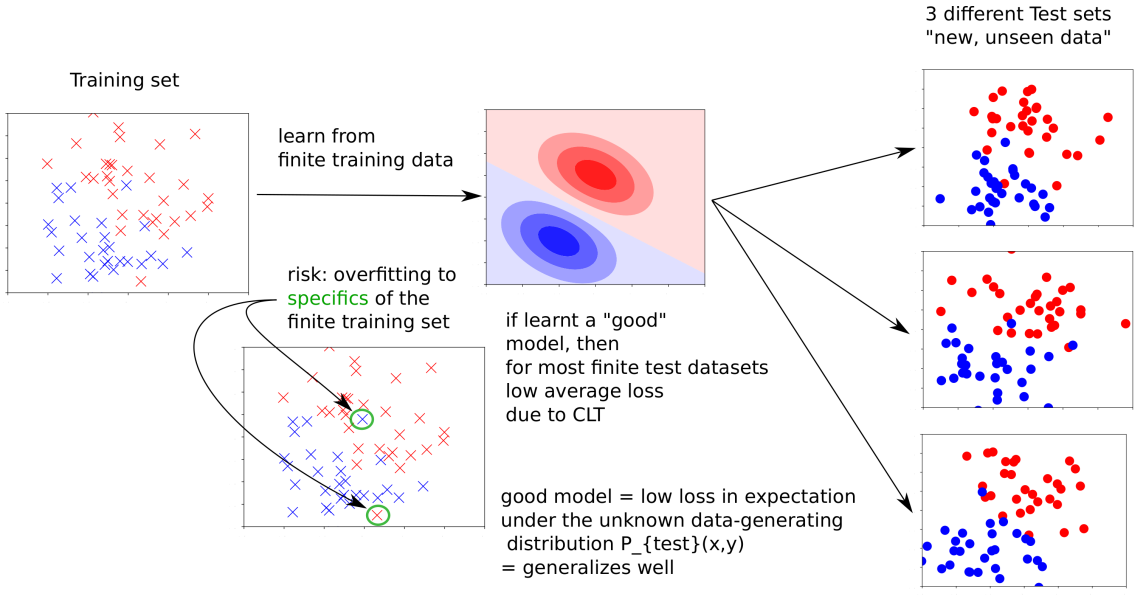
Universal approximation theorem (a version)

Let $g(\cdot)$ be a continuous function on a m-dimensional hypercube $[0,1]^m$. Let $a(\cdot)$ be a non-constant, bounded, continuous (activation) function. Then $g(\cdot)$ can be approximated arbitrarily well, that is, for every maximal deviation $\epsilon > 0$, there exists a set of weights u_i, w_i and biases b_i such that:

$$\forall x \in [0,1]^m : \ |g(x) - (\sum_{i=1} u_i a(w_i x + b_i) + b)| < \epsilon$$

Neural networks with one hidden layer and two layers of weights can approximate any smooth function on a compact hypercube. If there is a good algorithm for learning the parameters from finite data, we are done!

- Universal approximation theorem was the arch-seducer of neural network research.
- Kolmogorov (1957), Hornik (1989), Cybenko (1989) and others: Neural network can approximate *any continuous function on a compact region*.
- However, ability to approximate any function $\neq$ able to learn any function well from finite training data (... overfitting)



Right because you can represent any shape, you will easily reserve a blue area for the blue point in the training data on the wrong side. This results in overfitting. need to restrict models to prevent this!!!

- convolution layers as locally restricted linear operators
- better ways to apply gradients (momentum, Adam, etc)
- regularizations on gradient flow

Key learning goals

- Understand logistic regression and why it is a classification algorithm and a 1-layer neural network.
- Know the structure and components of dense feedforward neural networks.
- Be able to do forward pass by hand for dense feedforward neural networks.
- Understand the cross-entropy loss.
- Have an intuition about representability of simple neural networks.

Questions?