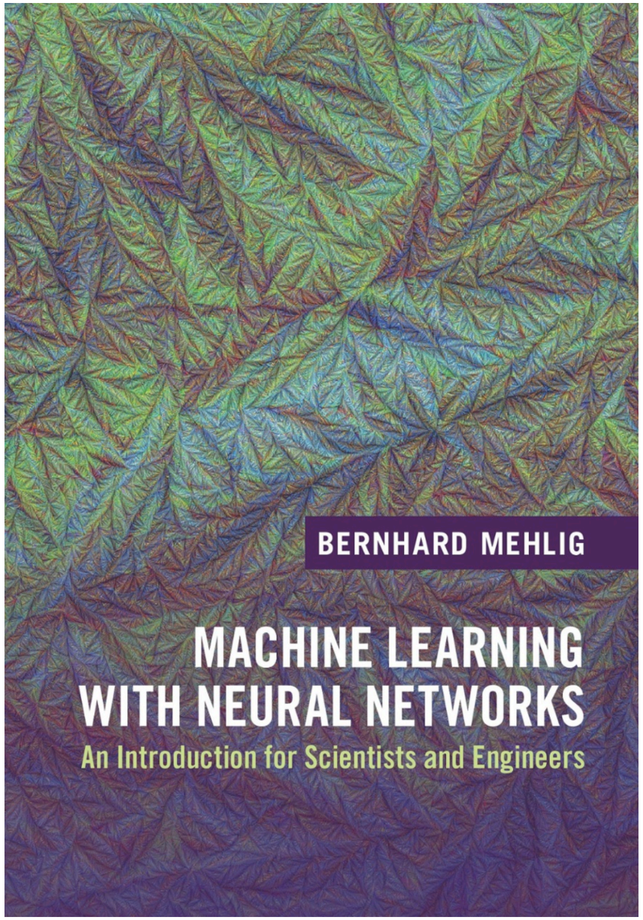
Machine learning with neural networks

B. Mehlig, Department of Physics, University of Gothenburg, Sweden

Bernhard Mehlig, *Machine learning with neural networks*

Machine learning with neural networks

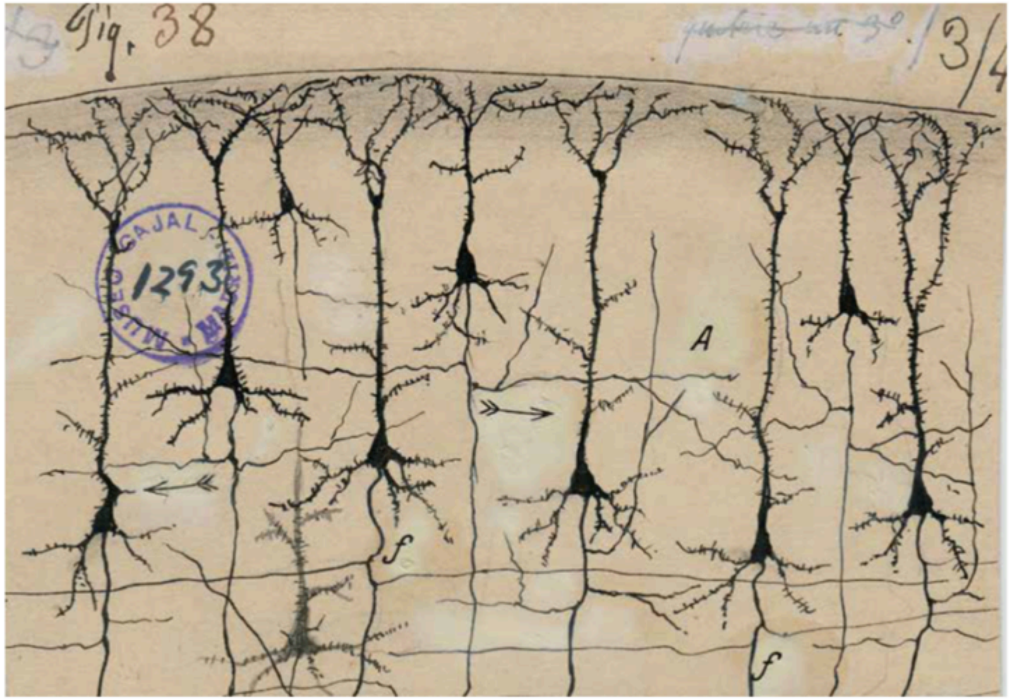
Cambridge University Press (2021)

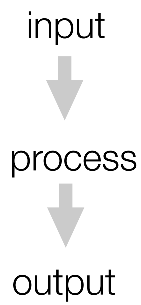


… Rather than presenting canned algorithms, this book tackles the fundamentals. As such, it is not for the faint hearted, but requires a sound background in theoretical physics, drawing on concepts such as …

Probert, *Contemporary Physics* (2022)

Neurons in the cerebral cortex



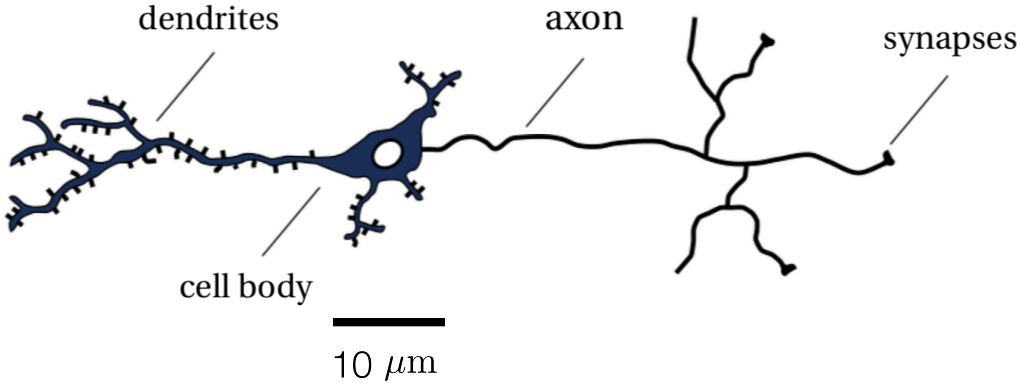


Neurons in the cerebral cortex (outer layer of the cerebrum, the largest and best developed part of the mammalian brain). Drawing by Santiago Ramón y Cajal, the Spanish neuroscientist who received the Nobel Prize in Physiology and Medicine in 1906 together with Camillo Golgi 'in recognition of their work on the structure of the nervous system'. Courtesy of the Cajal Institute, 'Cajal Legacy', Spanish National Research Council (CSIC), Madrid, Spain.

Neuron anatomy and activity

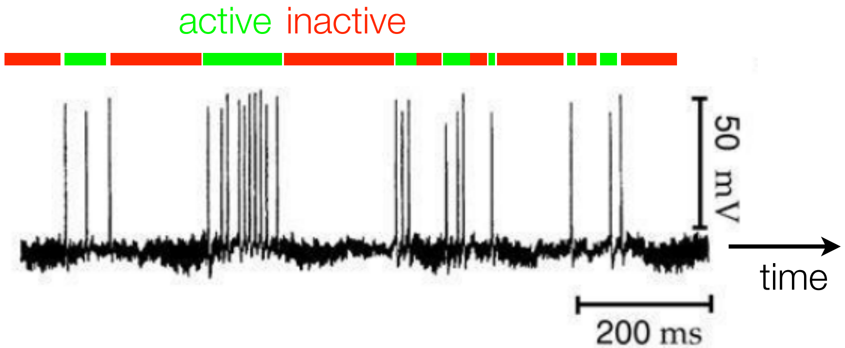
Schematic drawing of a neuron

input ➡ process ➡ output



Total length of dendrites up to $\sim$ cm .

Output of a neuron: *spike train*



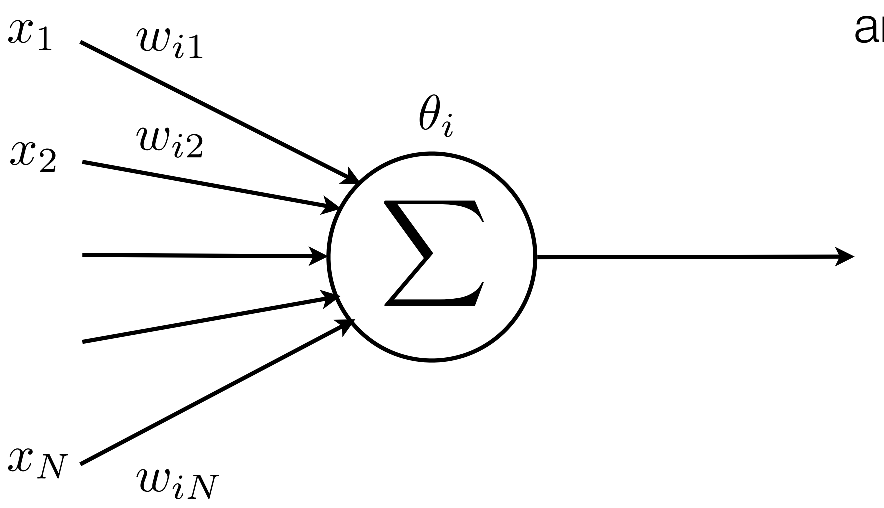
Spike train in electrosensory pyramidal neuron in fish *(Eigenmannia)*

Gabbiani & Metzner, J. Exp. Biol. **202** (1999) 1267

McCulloch-Pitts neuron

McCulloch & Pitts, Bull. Math. Biophys. **5** (1943) 115

Simple model for a neuron:



incoming signals x_j j=1,...,N

weights w_{ij} (synaptic couplings)

threshold θ_i neuron number i

output O_i

Neuron i computes weighted sum of inputs x_j with weights w_{ij}, subtracts threshold θ_i, and takes activation function:

$$O_i = g\Big(\underbrace{\sum_{j=1}^{N} w_{ij} x_j - \theta_i}_{= b_i}\Big)$$

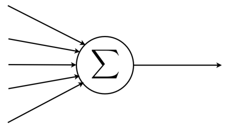
$= b_i$ (local field)

Activation function

Signal processing of *McCulloch-Pitts neuron*: weighted sum of inputs x_j with activation function $g(b_i)$:

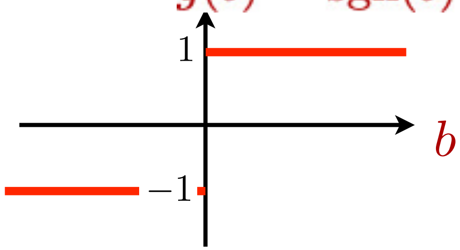
$$O_i = g\Big(\underbrace{\sum_{j=1}^{N} w_{ij}x_j - \theta_i}_{= b_i}\Big)$$

$= b_i$ (local field)



Inputs x_j
Weights w_{ij}
Threshold θ_i
Output O_i

Activation function $g(b)$:



Signum function $g(b) = \mathrm{sgn}(b)$

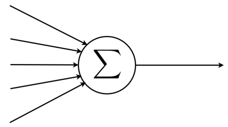
Two states: active (+1), inactive (-1).

Activation function

Signal processing of *McCulloch-Pitts neuron*: weighted sum of inputs x_j with activation function $g(b_i)$:

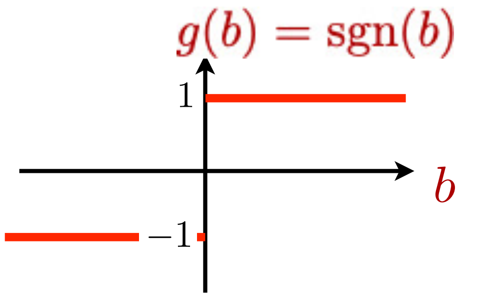
$$O_i = g\Big(\underbrace{\sum_{j=1}^{N} w_{ij} x_j - \theta_i}_{= b_i}\Big)$$

$= b_i$ (local field)

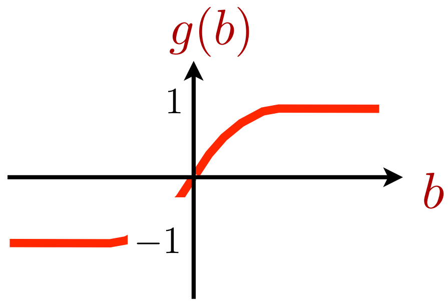


Inputs x_j
Weights w_{ij}
Threshold θ_i
Output O_i

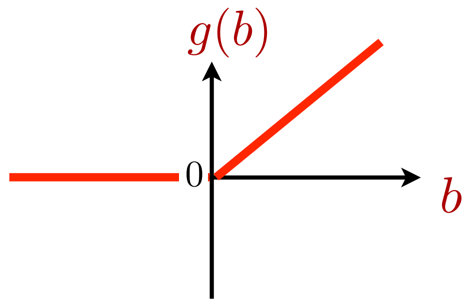
Activation function $g(b)$:



Signum function $g(b) = \mathrm{sgn}(b)$



Tangens hyperbolicus



ReLU function $\max(0, b)$

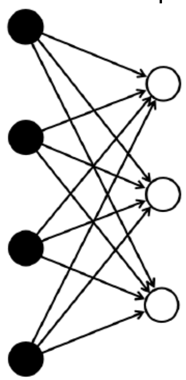
Two states: active (+1), inactive (-1).

Continuous range of state values.

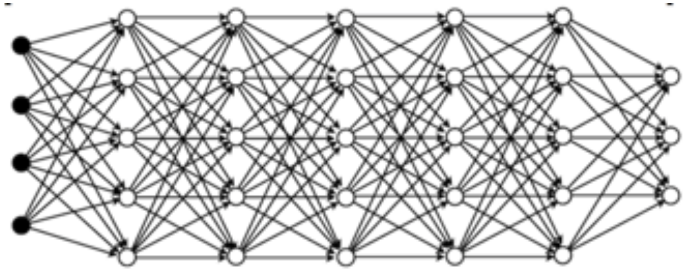
Neural nets

Rosenblatt, Psychological Review **65** (1958) 386

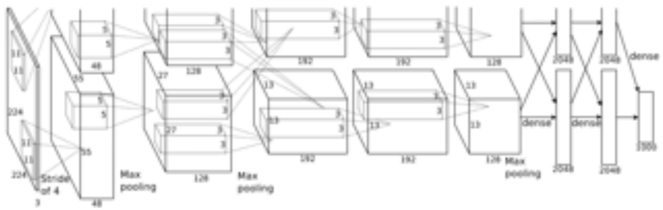
Connect neurons into networks that can perform computing tasks: for example object location and identification, speech recognition, classification, clustering,...



Simple perceptron



Deep neural net (many hidden layers).



Convolutional neural net Krizhevsky, Sutskever & Hinton (2012)

Input

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}$$

Output

$$\boldsymbol{O} = \begin{bmatrix} O_1 \\ \vdots \\ O_M \end{bmatrix}$$

Achieve this by adjusting weights and thresholds.

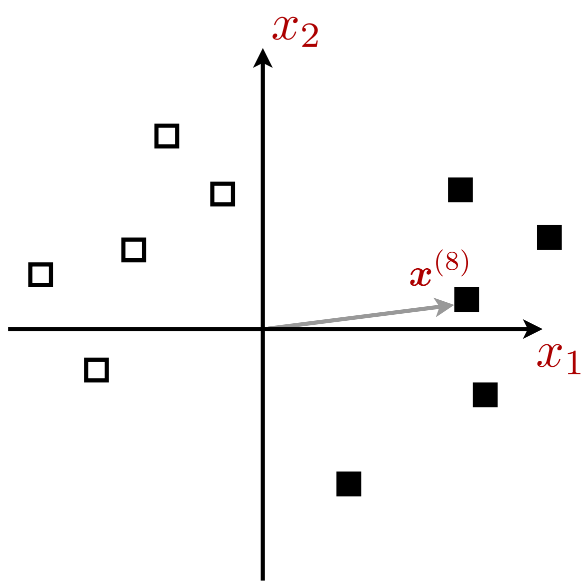
A simple classification task ($N = 2$)

Input patterns $\boldsymbol{x}^{(\mu)}$. Index $\mu = 1, \ldots, p$ labels different patterns.

Each pattern has two components, $x_1^{(\mu)}$ and $x_2^{(\mu)}$.

Arrange components into vector, $\boldsymbol{x}^{(\mu)} = \begin{bmatrix} x_1^{(\mu)} \\ x_2^{(\mu)} \end{bmatrix}$.
$\boldsymbol{x}^{(8)}$ is shown in the Figure.

Patterns fall into two classes: □ on the left, and ■ on the right.

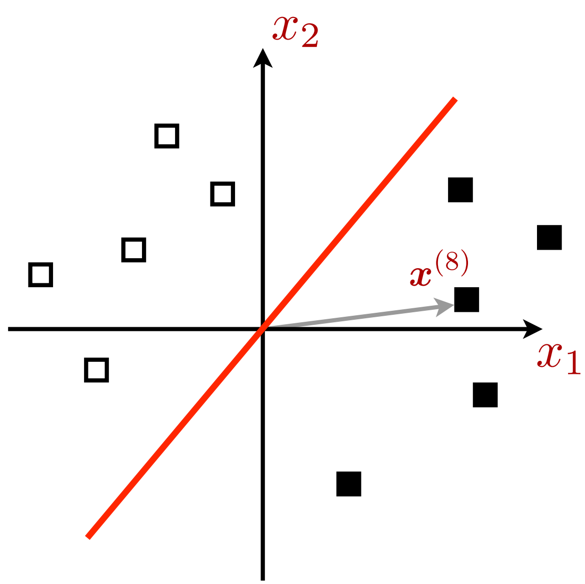


A simple classification task ($N = 2$)

Input patterns $\boldsymbol{x}^{(\mu)}$. Index $\mu = 1, \ldots, p$ labels different patterns.

Each pattern has two components, $x_1^{(\mu)}$ and $x_2^{(\mu)}$.

Arrange components into vector, $\boldsymbol{x}^{(\mu)} = \begin{bmatrix} x_1^{(\mu)} \\ x_2^{(\mu)} \end{bmatrix}$.
$\boldsymbol{x}^{(8)}$ is shown in the Figure.



Patterns fall into two classes: □ on the left, and ■ on the right.

Draw a red line (*decision boundary*) to distinguish the two types of patterns (□ and ■).

A simple classification task ($N = 2$)

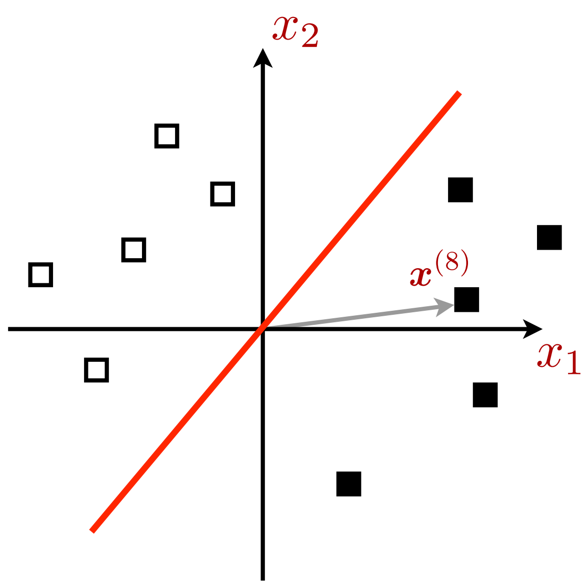
Input patterns $\boldsymbol{x}^{(\mu)}$. Index $\mu = 1, \ldots, p$ labels different patterns.

Each pattern has two components, $x_1^{(\mu)}$ and $x_2^{(\mu)}$.

Arrange components into vector, $\boldsymbol{x}^{(\mu)} = \begin{bmatrix} x_1^{(\mu)} \\ x_2^{(\mu)} \end{bmatrix}$,
$\boldsymbol{x}^{(8)}$ is shown in the Figure.

Patterns fall into two classes: □ on the left, and ■ on the right.

Draw a red line (*decision boundary*) to distinguish the two types of patterns (□ and ■).



Aim: train a neural network to compute the decision boundary. To do this, define *target values*:

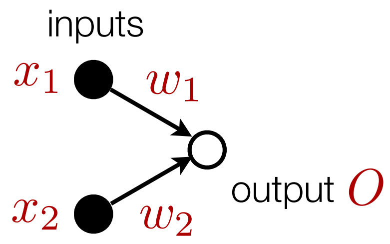
$t^{(\mu)} = 1$ for ■, and $t^{(\mu)} = -1$ for □

Training set $(\boldsymbol{x}^{(\mu)}, t^{(\mu)}), \mu = 1, \ldots, p$.

Geometrical solution

Rosenblatt, Psychological Review **65** (1958) 386
Minsky & Papert, *Perceptrons. An introduction to computational geometry*. MIT Press (1969)

Simple perceptron: one neuron. Two input terminals x_1 and x_2. Activation function $\operatorname{sgn}(b)$.

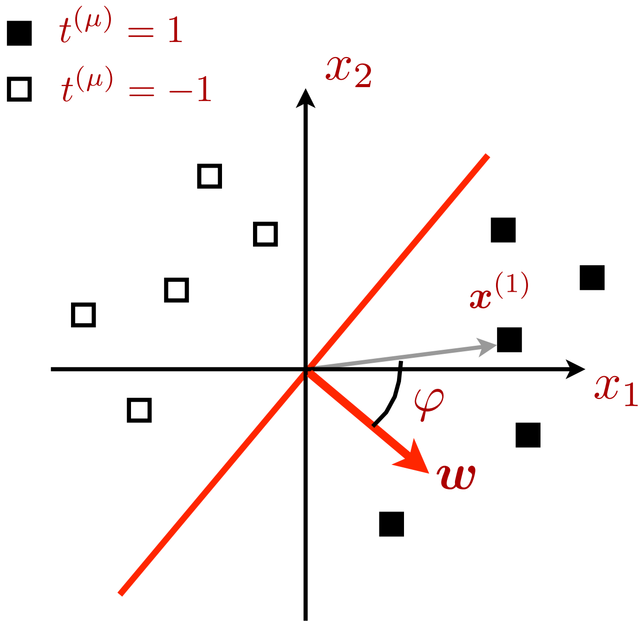


No threshold, $\theta = 0$.

Output $O^{(\mu)} = \operatorname{sgn}(w_1 x_1^{(\mu)} + w_2 x_2^{(\mu)}) = \operatorname{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}^{(\mu)})$

scalar product $\boldsymbol{w} \cdot \boldsymbol{x}^{(\mu)} = |\boldsymbol{w}|\, |\boldsymbol{x}^{(\mu)}| \cos\varphi$ ← angle between $\boldsymbol{w}$ and $\boldsymbol{x}^{(\mu)}$

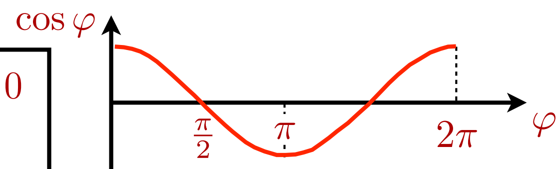
Aim: adjust the weights $\boldsymbol{w}$ so that network outputs correct target values for all patterns:



$$O^{(\mu)} = \operatorname{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}^{(\mu)}) = t^{(\mu)} \quad \text{for} \quad \mu = 1, \ldots, p$$

Solution:

define decision boundary by $\boldsymbol{w} \cdot \boldsymbol{x}^{(\mu)} = 0$
so that $\boldsymbol{w} \perp$ decision boundary.



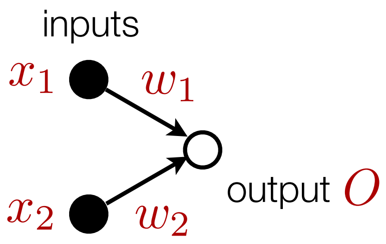
Check: $\boldsymbol{w} \cdot \boldsymbol{x}^{(1)} > 0 \Rightarrow O^{(1)} = \operatorname{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}^{(1)}) \overset{!}{=} 1$

Geometrical solution

Minsky & Papert, *Perceptrons. An introduction to computational geometry*. MIT Press (1969).

Simple perceptron: one neuron. Two input terminals x_1 and x_2. Activation function $\mathrm{sgn}(b)$.

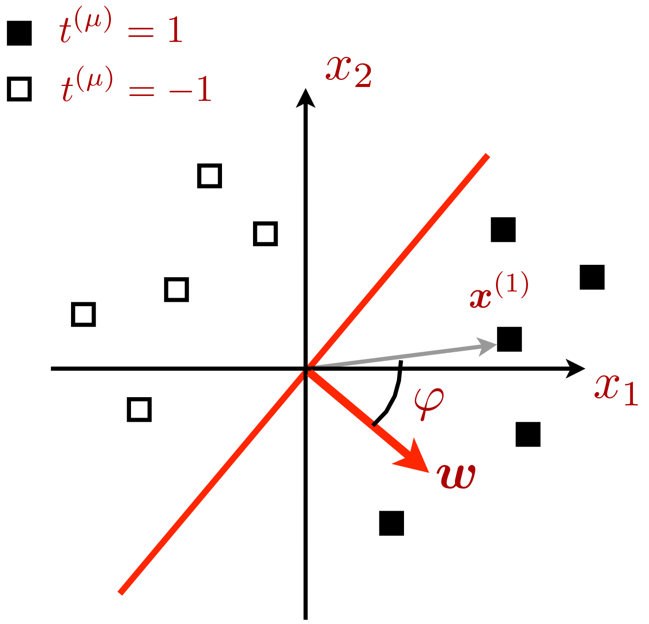
No threshold, $\theta = 0$.



Output $O^{(\mu)} = \mathrm{sgn}(w_1 x_1^{(\mu)} + w_2 x_2^{(\mu)}) = \mathrm{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}^{(\mu)})$

scalar product $\boldsymbol{w} \cdot \boldsymbol{x}^{(\mu)} = |\boldsymbol{w}|\,|\boldsymbol{x}^{(\mu)}|\,\cos\varphi$ ← angle between $\boldsymbol{w}$ and $\boldsymbol{x}^{(\mu)}$

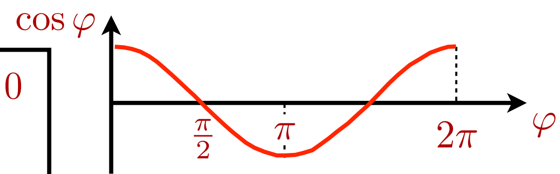
Aim: adjust the weights $\boldsymbol{w}$ so that network outputs correct target values for all patterns:



$$O^{(\mu)} = \mathrm{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}^{(\mu)}) = t^{(\mu)} \quad \text{for} \quad \mu = 1, \ldots, p$$

Solution:

define decision boundary by $\boldsymbol{w} \cdot \boldsymbol{x}^{(\mu)} = 0$
so that $\boldsymbol{w} \perp$ decision boundary.



Check: $\boldsymbol{w} \cdot \boldsymbol{x}^{(1)} > 0 \Rightarrow O^{(1)} = \mathrm{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}^{(1)}) = 1$

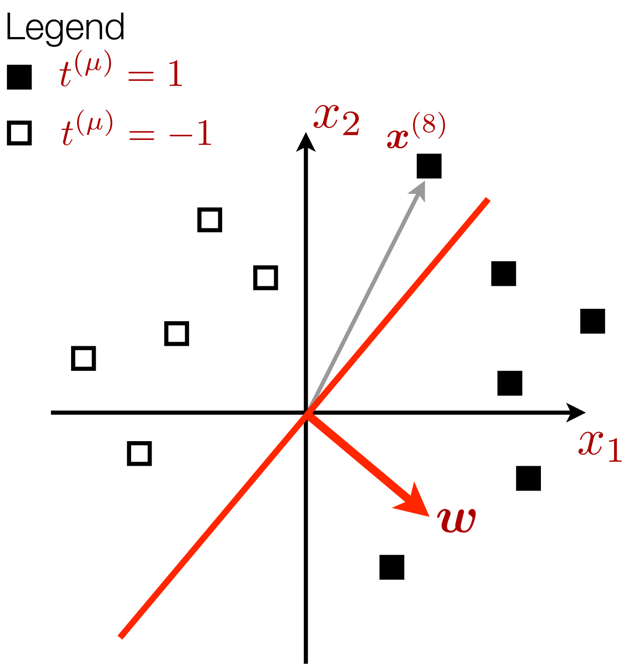
Correct since $t^{(1)} = 1$. ✔

Hebb's rule

D. O. Hebb, *The organization of behaviour: a neurospychological theory*, Wiley, New York (1949)

Now the pattern $\boldsymbol{x}^{(8)}$ is on wrong side of the red line. So $O^{(8)} = \mathrm{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}^{(8)}) \neq t^{(8)}$.

Move the red line by rotating the weight vector $\boldsymbol{w}$:



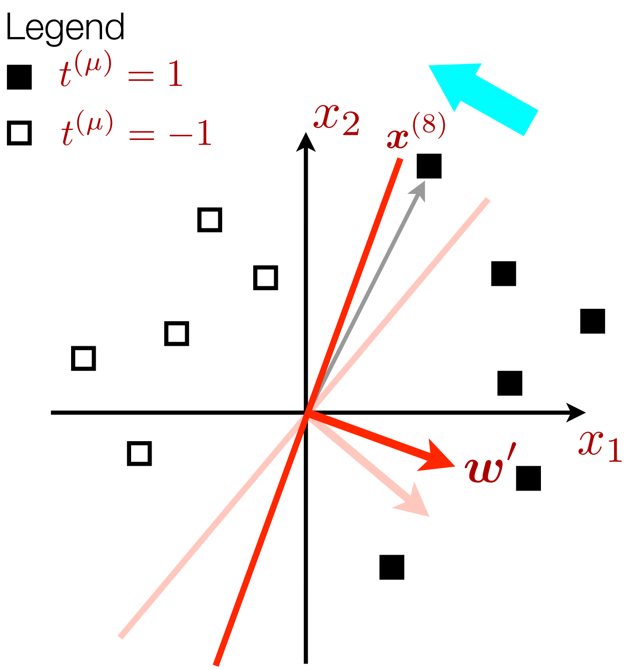
Hebb's rule

D. O. Hebb, *The organization of behaviour: a neurospychological theory*, Wiley, New York (1949)

Now the pattern $\boldsymbol{x}^{(8)}$ is on wrong side of the red line. So $O^{(8)} = \mathrm{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}^{(8)}) \neq t^{(8)}$.

Move the red line by rotating the weight vector $\boldsymbol{w}$:

$\boldsymbol{w}' = \boldsymbol{w} + \eta \boldsymbol{x}^{(8)}$ (small parameter $\eta > 0$) so that $O^{(8)} = \mathrm{sgn}(\boldsymbol{w}' \cdot \boldsymbol{x}^{(8)}) = t^{(8)}$.

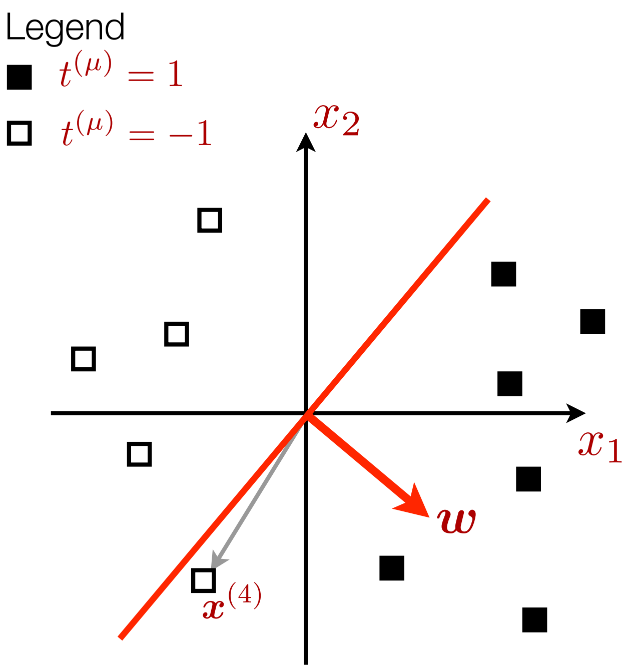


Hebb's rule

D. O. Hebb, *The organization of behaviour: a neurospychological theory*, Wiley, New York (1949)

Now the pattern $\boldsymbol{x}^{(4)}$ is on wrong side of the red line. So $O^{(4)} = \mathrm{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}^{(4)}) \neq t^{(4)}$.

Move the red line by rotating the weight vector $\boldsymbol{w}$:



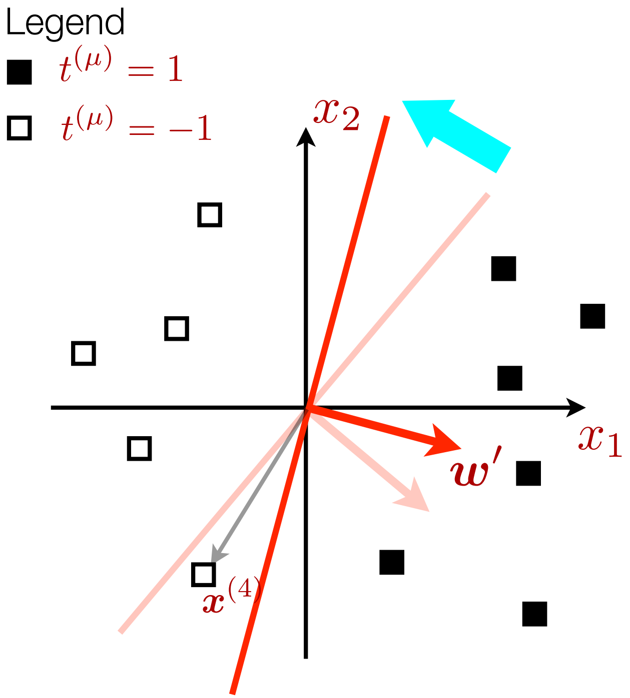
Hebb's rule

D. O. Hebb, *The organization of behaviour: a neurospychological theory*, Wiley, New York (1949)

Now the pattern $\boldsymbol{x}^{(4)}$ is on wrong side of the red line. So $O^{(4)} = \mathrm{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}^{(4)}) \neq t^{(4)}$.

Move the red line by rotating the weight vector $\boldsymbol{w}$:

$\boldsymbol{w}' = \boldsymbol{w} - \eta \boldsymbol{x}^{(4)}$(small parameter $\eta > 0$) so that $O^{(4)} = \mathrm{sgn}(\boldsymbol{w}' \cdot \boldsymbol{x}^{(4)}) = t^{(4)}$.



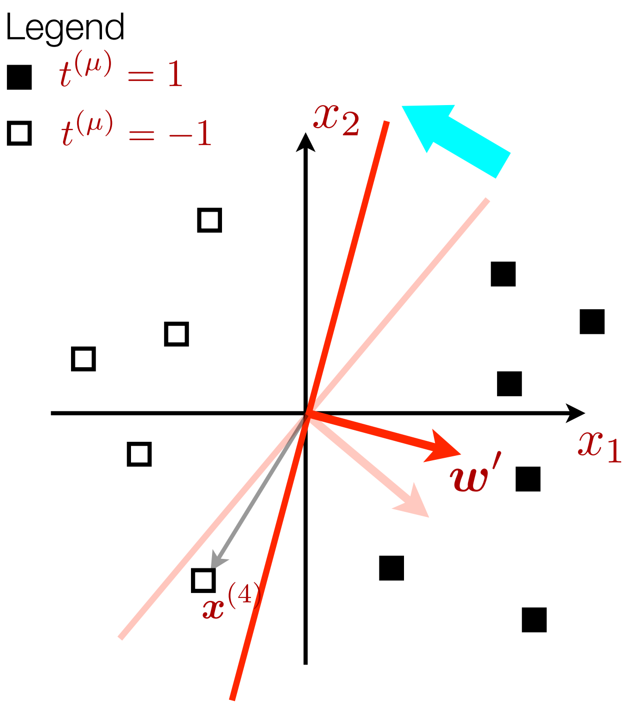
Hebb's rule

D. O. Hebb, *The organization of behaviour: a neurospychological theory*, Wiley, New York (1949)

Now the pattern $\boldsymbol{x}^{(4)}$ is on wrong side of the red line. So $O^{(4)} = \mathrm{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}^{(4)}) \neq t^{(4)}$.

Move the red line by rotating the weight vector $\boldsymbol{w}$:

$\boldsymbol{w}' = \boldsymbol{w} - \eta \boldsymbol{x}^{(4)}$(small parameter $\eta > 0$) so that $O^{(4)} = \mathrm{sgn}(\boldsymbol{w}' \cdot \boldsymbol{x}^{(4)}) = t^{(4)}$.



Note the difference in sign:

$$\boldsymbol{w}' = \boldsymbol{w} + \eta \boldsymbol{x}^{(8)} \quad \text{for} \quad t^{(8)} = 1$$

$$\boldsymbol{w}' = \boldsymbol{w} - \eta \boldsymbol{x}^{(4)} \quad \text{for} \quad t^{(4)} = -1$$

Learning rule (*Hebb's rule*)

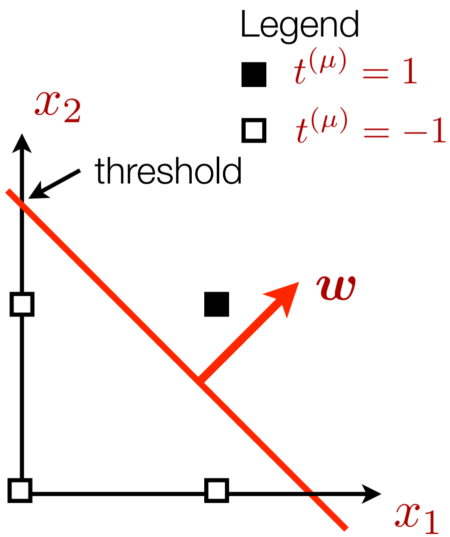
$$\boldsymbol{w}' = \boldsymbol{w} + \delta \boldsymbol{w} \quad \text{with} \quad \delta \boldsymbol{w} = \eta\, t^{(\mu)} \boldsymbol{x}^{(\mu)}$$

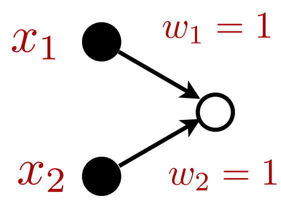
Apply learning rule many times until problem is solved.

Example - AND function

Logical AND

x_1	x_2	t
0	0	-1
1	0	-1
0	1	-1
1	1	1





Neuron computes $O^{(\mu)} = \mathrm{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}^{(\mu)} - \theta)$.

Condition for decision boundary: $\boldsymbol{w} \cdot \boldsymbol{x} - \theta = 0$.

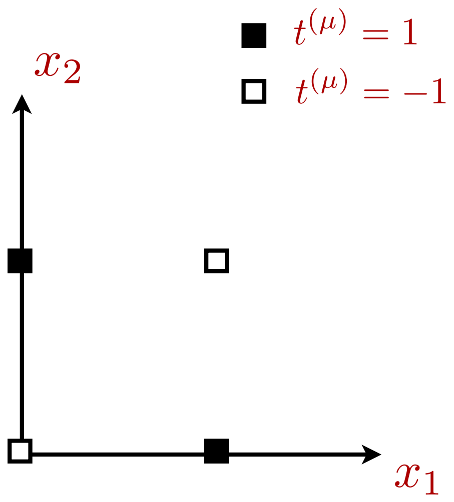
Line equation for decision boundary: $x_2 = -\frac{w_1}{w_2} x_1 + \frac{\theta}{w_2}$. ← intersection with x_2-axis

The threshold θ determines intersection of decision boundary with x_2-axis.

Example - XOR function

Logical XOR

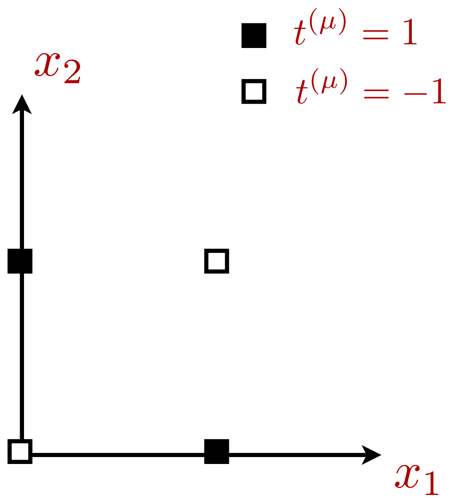
x_1	x_2	t
0	0	-1
1	0	1
0	1	1
1	1	-1



Example - XOR function

Logical XOR

x_1	x_2	t
0	0	-1
1	0	1
0	1	1
1	1	-1

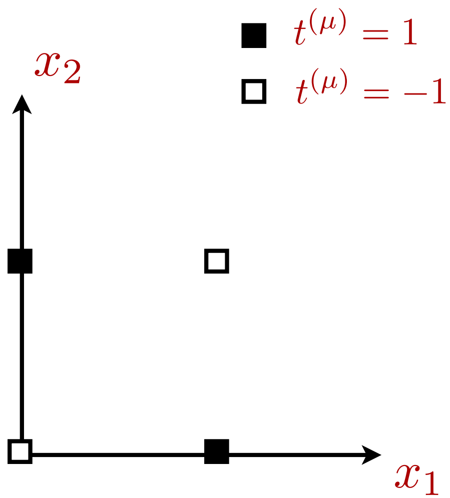


This problem is not *linearly separable* because we cannot separate ■ from □ by a single red line.

Example - XOR function

Logical XOR

x_1	x_2	t
0	0	-1
1	0	1
0	1	1
1	1	-1



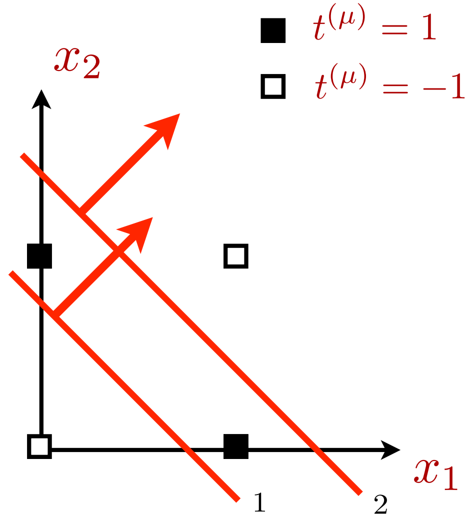
This problem is not *linearly separable* because we cannot separate ■ from □ by a *single* red line.

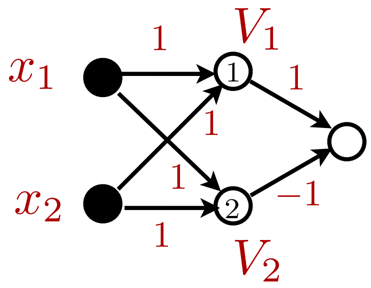
Solution: use *two* red lines.

Example - XOR function

Logical XOR

x_1	x_2	t
0	0	-1
1	0	1
0	1	1
1	1	-1





layer of hidden neurons V_1 and V_2 (neither input nor output)

all *hidden weights* equal to 1

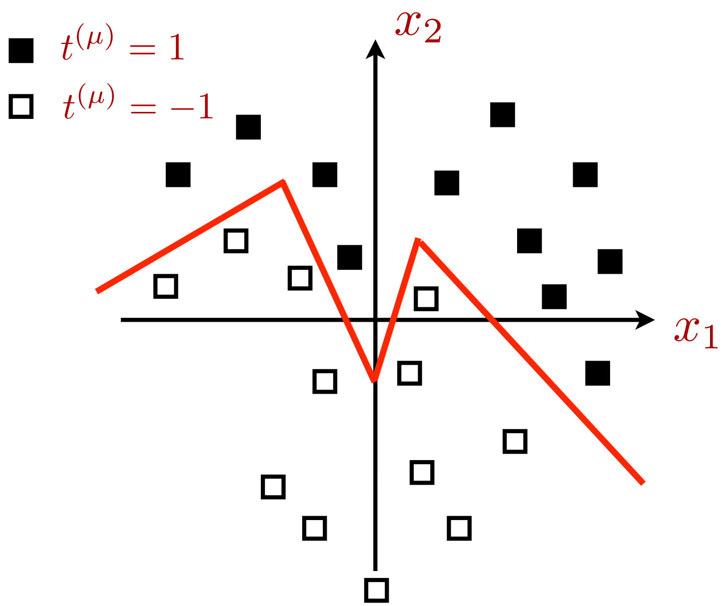
Two *hidden* neurons, each one defines one red line.

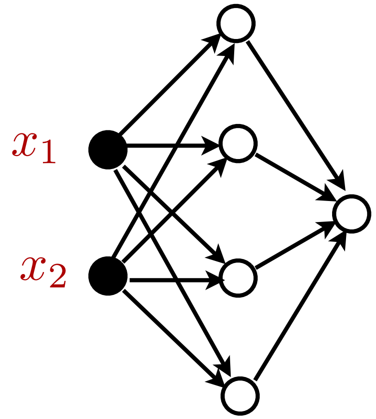
We need a third neuron to process the output of the hidden neurons.
It computes $O = \text{sgn}(V_1 - V_2 - 1)$

One reason why we need hidden neurons ⇨ deep nets ⇨ deep learning
(with many hidden layers)

Non-(linearly) separable problems

Solve problems that are not linearly separable with a hidden layer of neurons

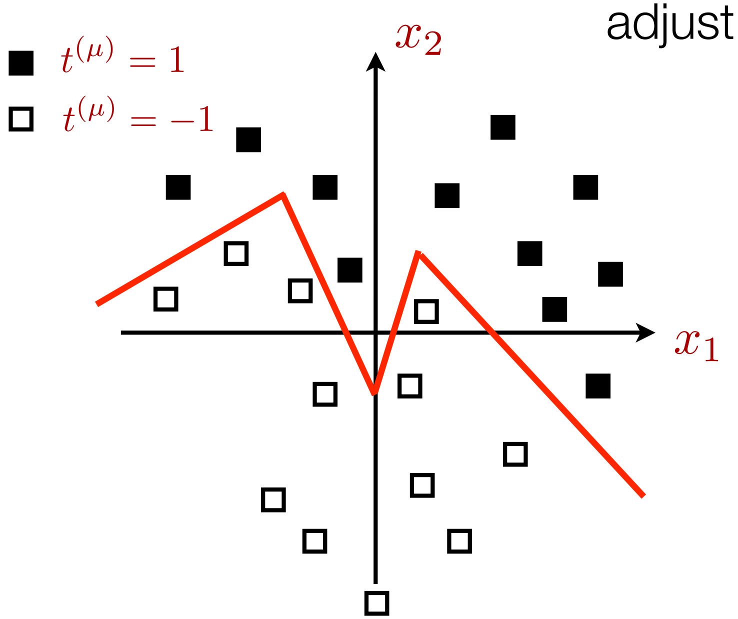


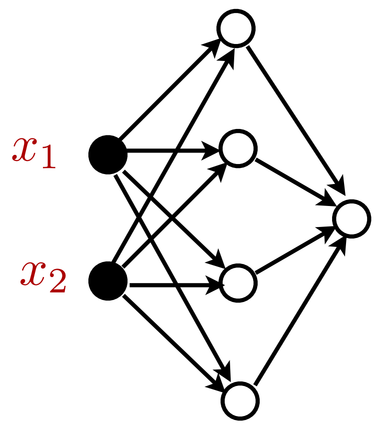


Four hidden neurons - one for each red line segment. Move the red lines into the correct configuration by repeatedly using Hebb's rule until the problem is solved (a fifth neuron assigns regions and solves the classification problem).

Training

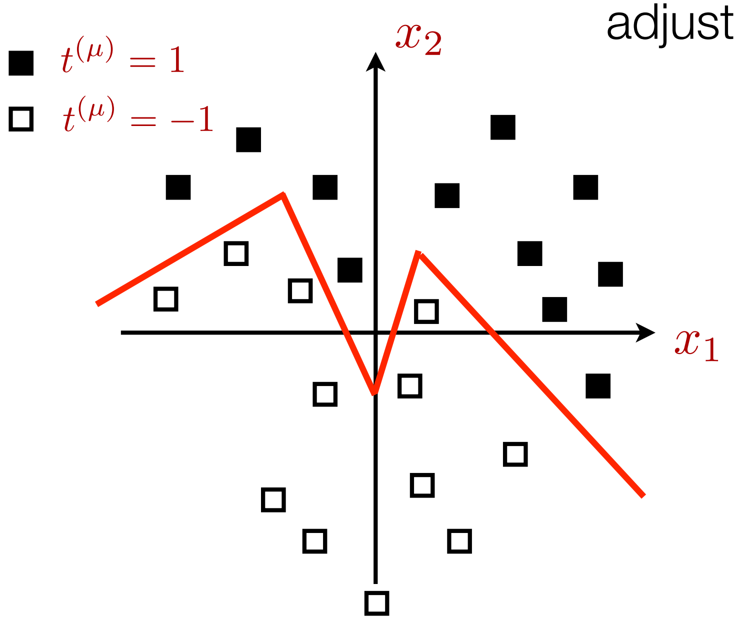
Train the network on a *training set* $(\boldsymbol{x}^{(\mu)}, t^{(\mu)}), \mu = 1, \ldots, p$: move red lines into the correct configuration by repeatedly applying Hebb's rule to adjust all weights. Usually many iterations necessary.

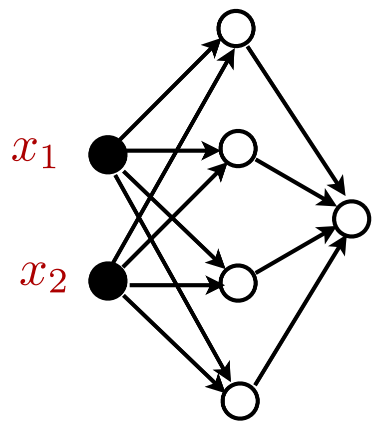




Training

Train the network on a *training set* $(\boldsymbol{x}^{(\mu)}, t^{(\mu)}), \mu = 1, \ldots, p$: move red lines into the correct configuration by repeatedly applying Hebb's rule to adjust all weights. Usually many iterations necessary.



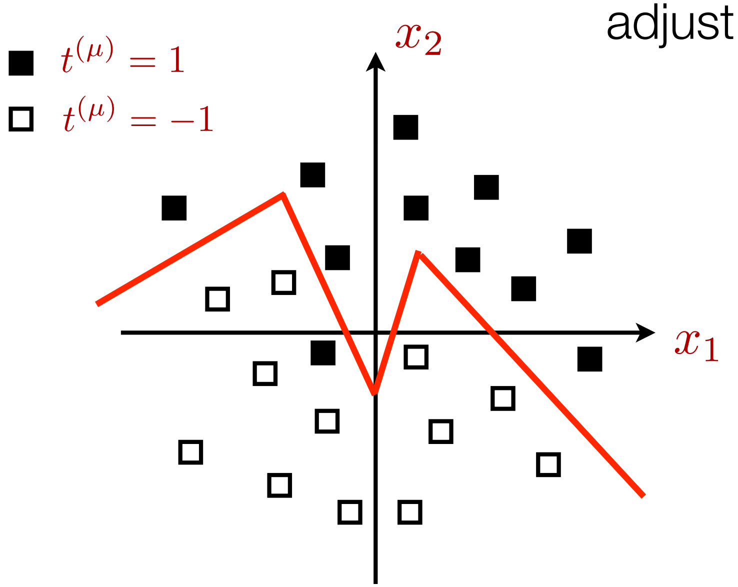


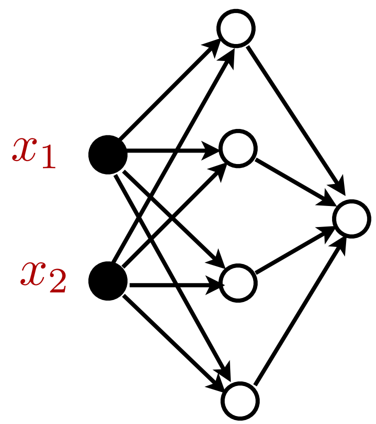
Training with Hebb's rule $\delta w_{mn} = \eta t_m^{(\mu)} x_n^{(\mu)}$. Better: $\delta w_{mn} = \eta (t_m^{(\mu)} - O_m^{(\mu)}) x_n^{(\mu)}$. Almost the same, but converges.

Once all red lines are in the right place, apply network to a new data set. If the training set was reliable, then the network has learnt to classify the new data, it has learnt to *generalise*.

Training

Train the network on a *training set* $(\boldsymbol{x}^{(\mu)}, t^{(\mu)}), \mu = 1, \ldots, p$: move red lines into the correct configuration by repeatedly applying Hebb's rule to adjust all weights. Usually many iterations necessary.

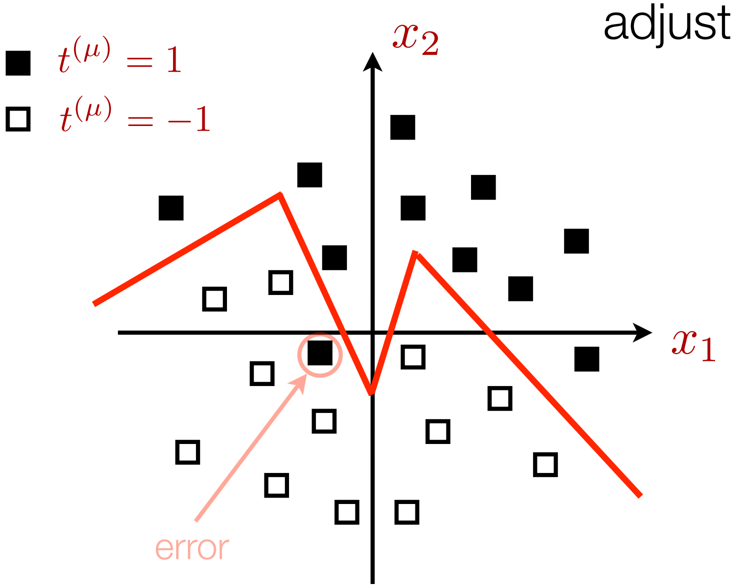


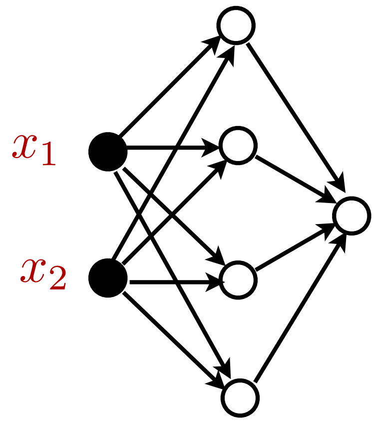


Once all red lines are in the right place (all weights determined), apply network to a new data set. If the training set was reliable, then the network has learnt to classify the new data, it has learnt to *generalise*.

Training

Train the network on a *training set* $(\boldsymbol{x}^{(\mu)}, t^{(\mu)}), \mu = 1, \ldots, p$: move red lines into the correct configuration by repeatedly applying Hebb's rule to adjust all weights. Usually many iterations necessary.

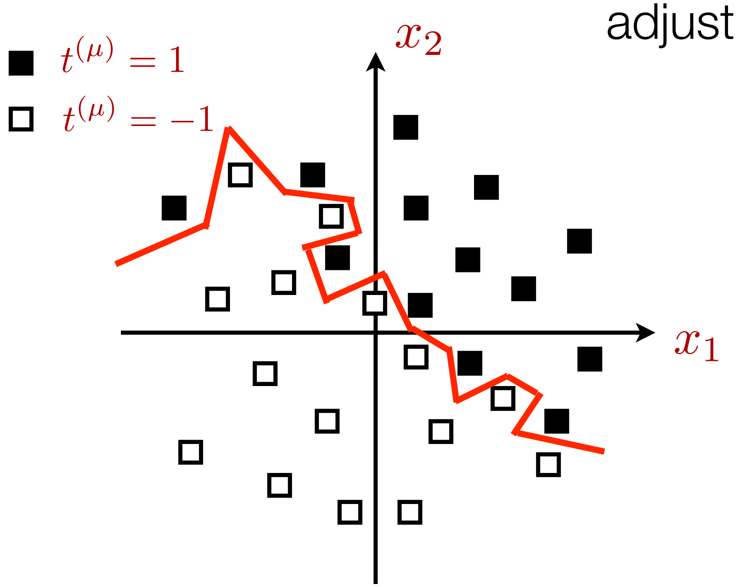




A small number of errors is acceptable. It is often not meaningful to try to fine-tune very precisely.

Overfitting

Train the network on a *training set* $(\boldsymbol{x}^{(\mu)}, t^{(\mu)}), \mu = 1, \ldots, p$: move red lines into the correct configuration by repeatedly applying Hebb's rule to adjust all weights. Usually many iterations necessary.



training data set

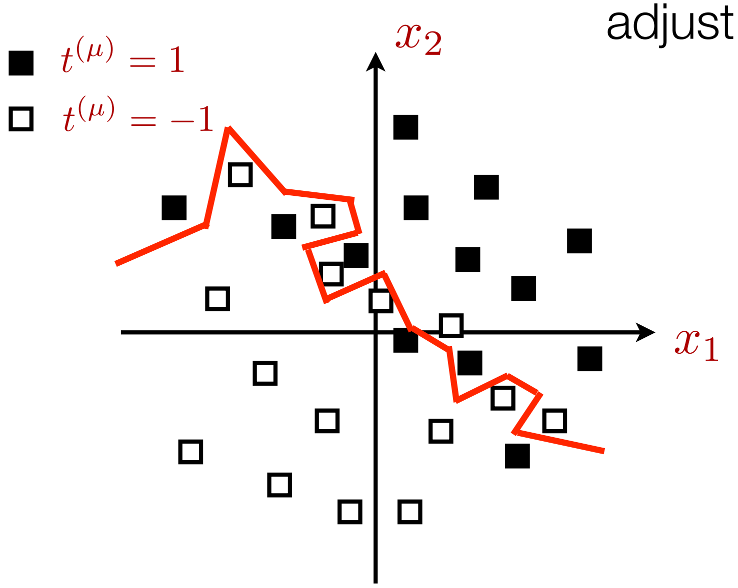
Here: used 15 hidden neurons to fit decision boundary very precisely.

Too many free parameters: network fits fine details specific for training set, but lack general meaning (*overfitting*).

A different sample from the same input distribution might look quite different in detail, inputs shifted randomly by a different realisation of input noise.

Overfitting

Train the network on a *training set* $(\boldsymbol{x}^{(\mu)}, t^{(\mu)}), \mu = 1, \ldots, p$: move red lines into the correct configuration by repeatedly applying Hebb's rule to adjust all weights. Usually many iterations necessary.



test data set

Here: used 15 hidden neurons to fit decision boundary very precisely.

Too many free parameters: network fits fine details specific for training set, but lack general meaning (*overfitting*).

A different sample from the same input distribution might look quite different in detail, inputs shifted randomly by a different realisation of input noise.

Networks usually have many parameters (weights and thresholds) ⇨ overfitting can be substantial problem.

How many hidden layers?

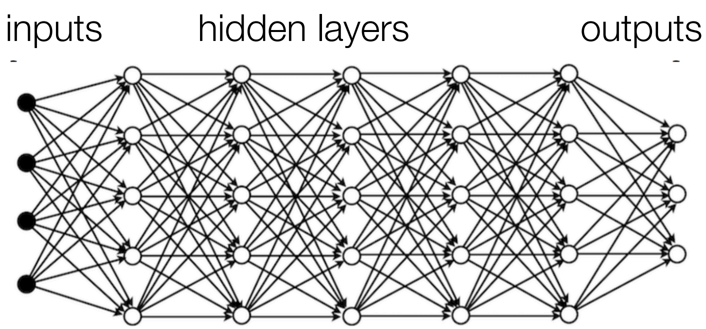
All Boolean functions with N inputs can be trained/learned with a single hidden layer.

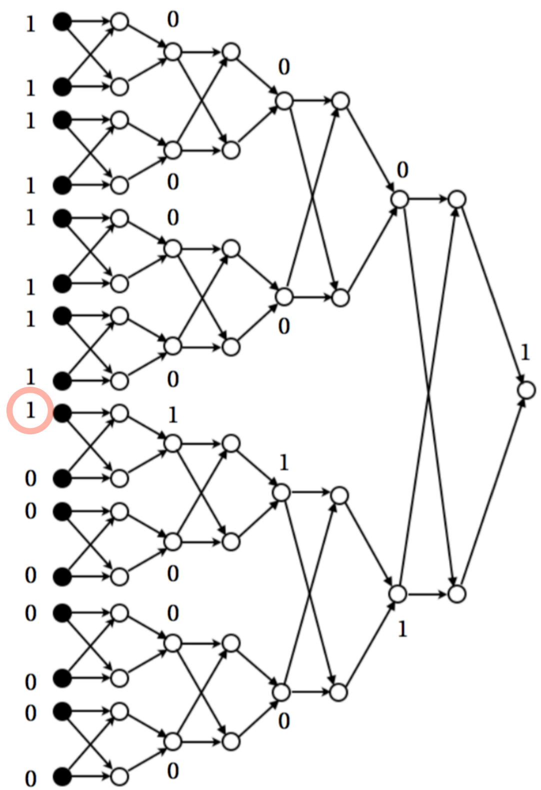
Proof by construction. Requires 2^N neurons in hidden layer.

For large N, this architecture is not practical because the number of neurons increases exponentially with N.

Example: parity function. More efficient layout: build network using XOR units. Requires only $\sim N$ units.

But such *deep networks* are in general hard to train.





Parity function: $O = 1$ if odd number of inputs equal 1, otherwise $O = 0$.

Convolutional networks

Convolutional network. Each neuron in first hidden layer is connected to a small region of inputs. Here 3×3 pixels.

Slide the region over input image. Use *same* weights for all hidden neurons. Update V_{ij}

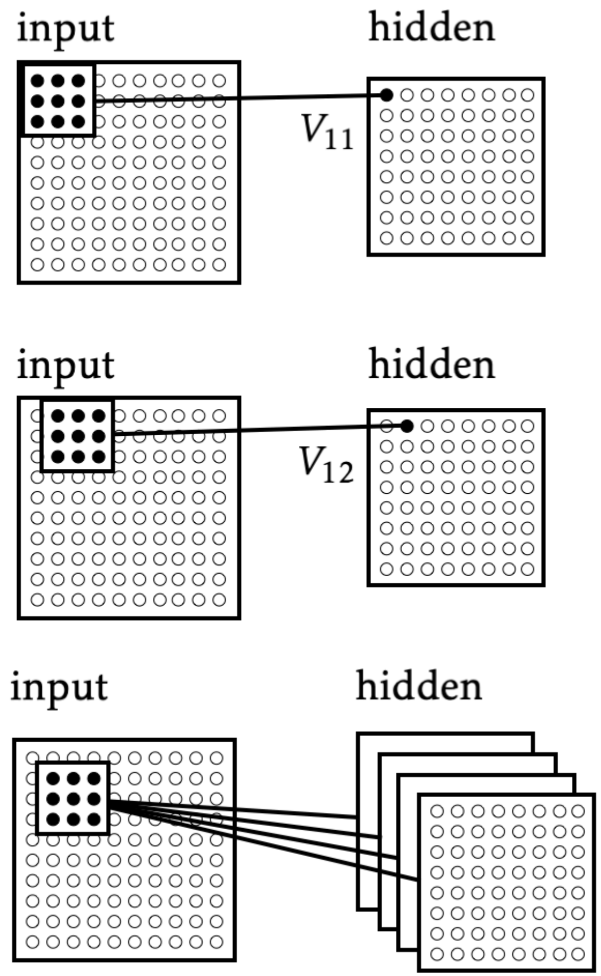
$$V_{ij} = g\Big(\sum_{p=1}^{3}\sum_{q=1}^{3} w_{pq} x_{p+i-1,q+j-1} - \theta\Big)$$

- detects the *same local feature* everywhere in input image (edge, corner,...). *Feature map*.

- the form of the sum is called *convolution*

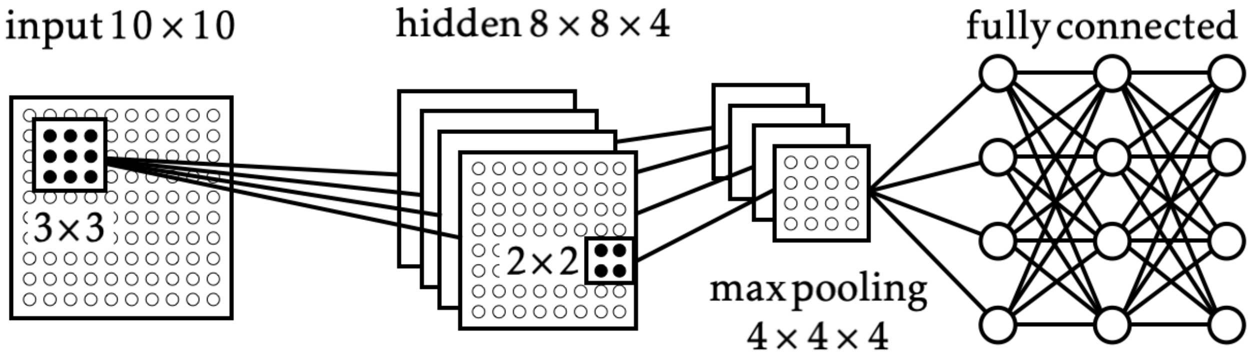
- less *overfitting* because fewer weights

- use several feature maps to detect different features in input image

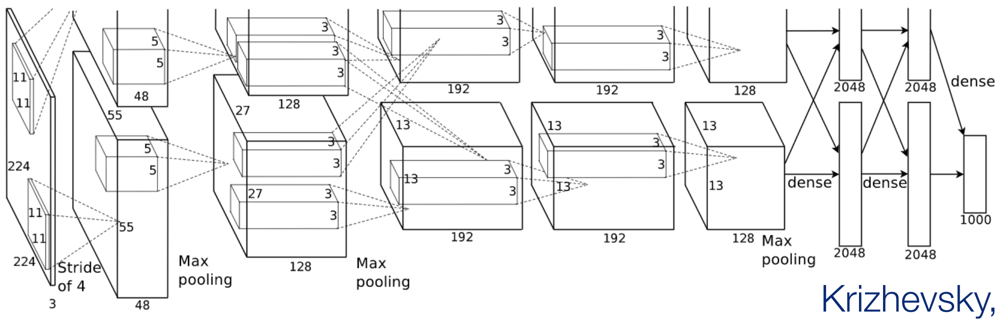


Convolutional networks

- *max-pooling*: take maximum of V_{mn} over small region (2×2) to reduce # of parameters



- add fully connected lavers to learn more abstract features



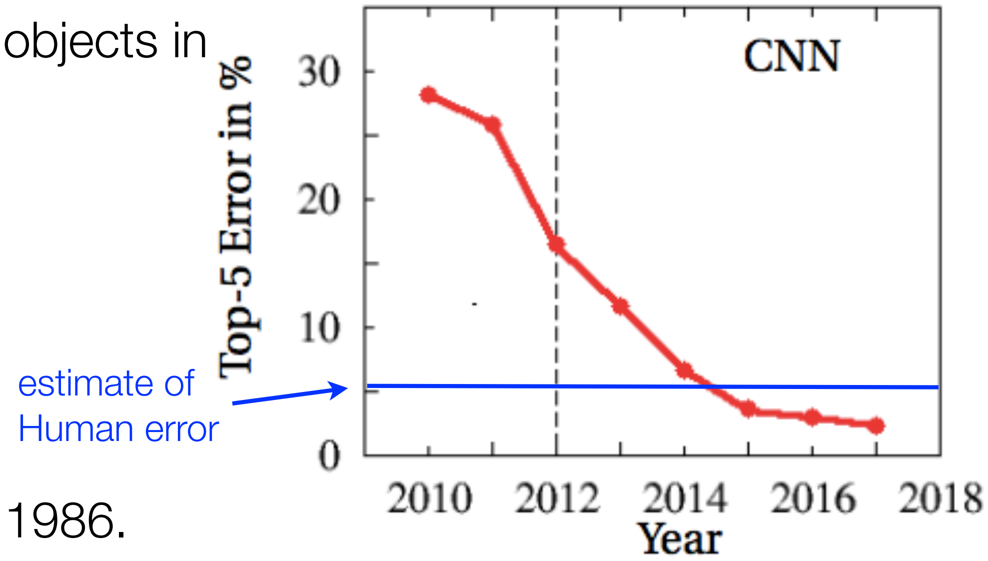
Krizhevsky, Sutskever & Hinton (2012)

Object location and classification

Deep convolutional nets locate and classify objects in images with very high accuracy.

Better than humans?
cs.stanford.edu/people/karpathy/ilsvrc

Goodfellow, Bengio & Courville (2016)



Convolutional nets have been around since 1986.

Patterns (T and C) detected by convolutional net Rumelhart, Hinton & Williams (1986)

It was always thought that deep nets are very difficult to train (overfitting, slow learning)

Why does it suddenly work so well?

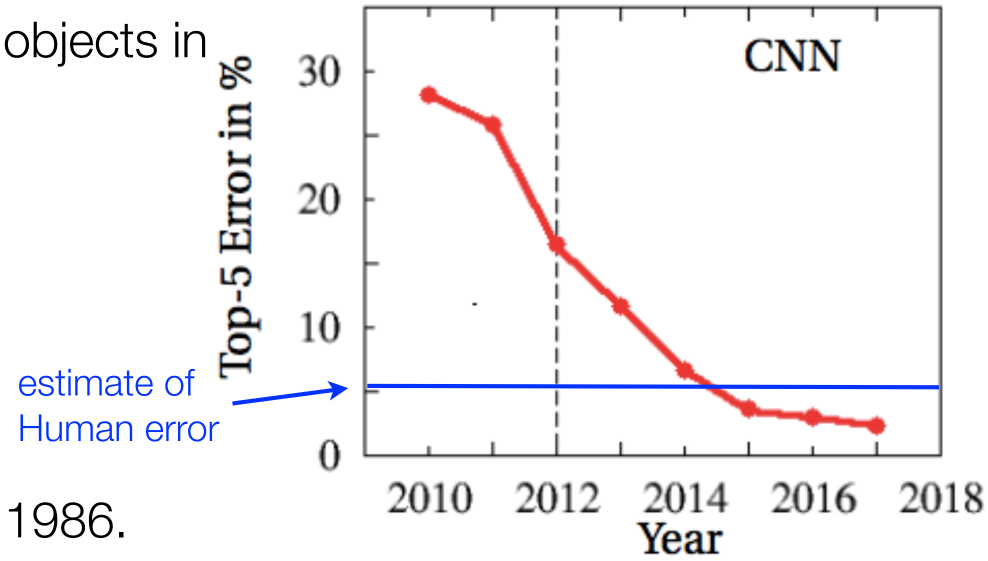
Object location and classification

Goodfellow, Bengio & Courville (2016)

Deep convolutional nets locate and classify objects in images with very high accuracy.

Better than humans?

cs.stanford.edu/people/karpathy/ilsvrc



Convolutional nets have been around since 1986.

Patterns (T and C) detected by convolutional net Rumelhart, Hinton & Williams (1986)

It was always thought that deep nets are very difficult to train (overfitting, slow learning)

Why does it suddenly work so well?

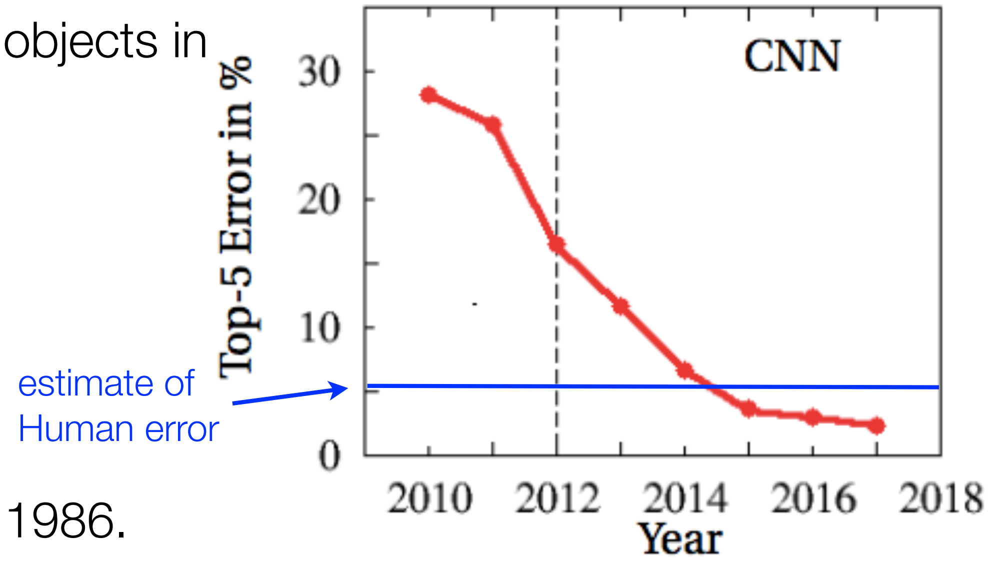
Object location and classification

Goodfellow, Bengio & Courville (2016)

Deep convolutional nets locate and classify objects in images with very high accuracy.

Better than humans?

cs.stanford.edu/people/karpathy/ilsvrc



Convolutional nets have been around since 1986.

Patterns (T and C) detected by convolutional net Rumelhart, Hinton & Williams (1986)

It was always thought that deep nets are very difficult to train (overfitting, slow learning)

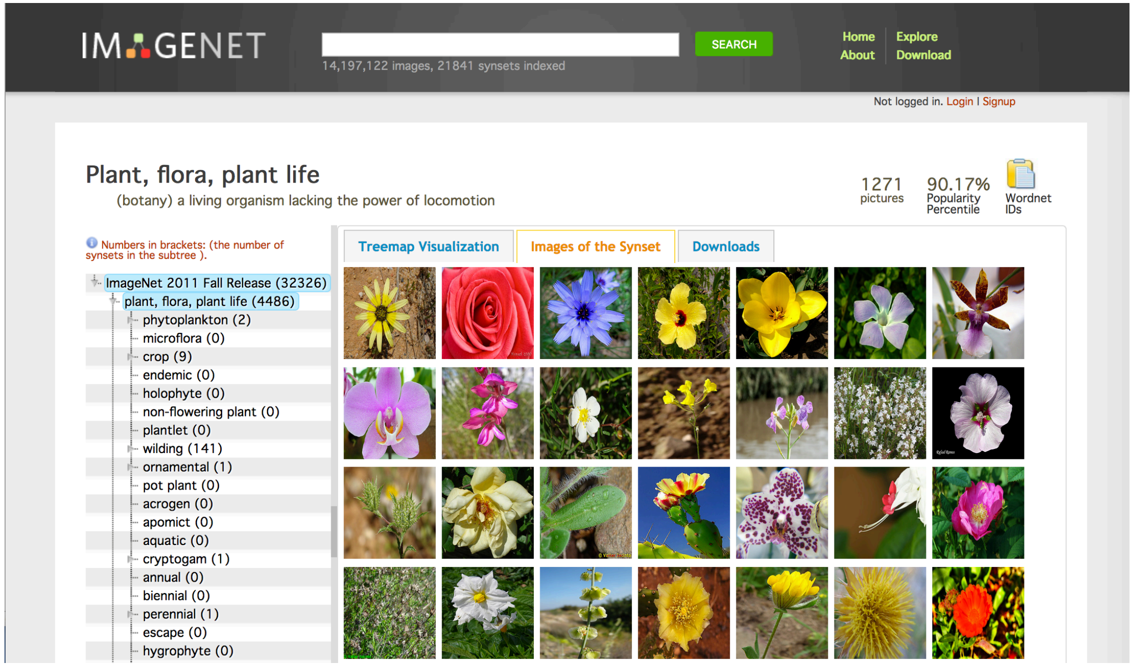
Why does it suddenly work so well? Mainly: better training sets (larger sets and more accurate targets).

Data sets: ImageNet

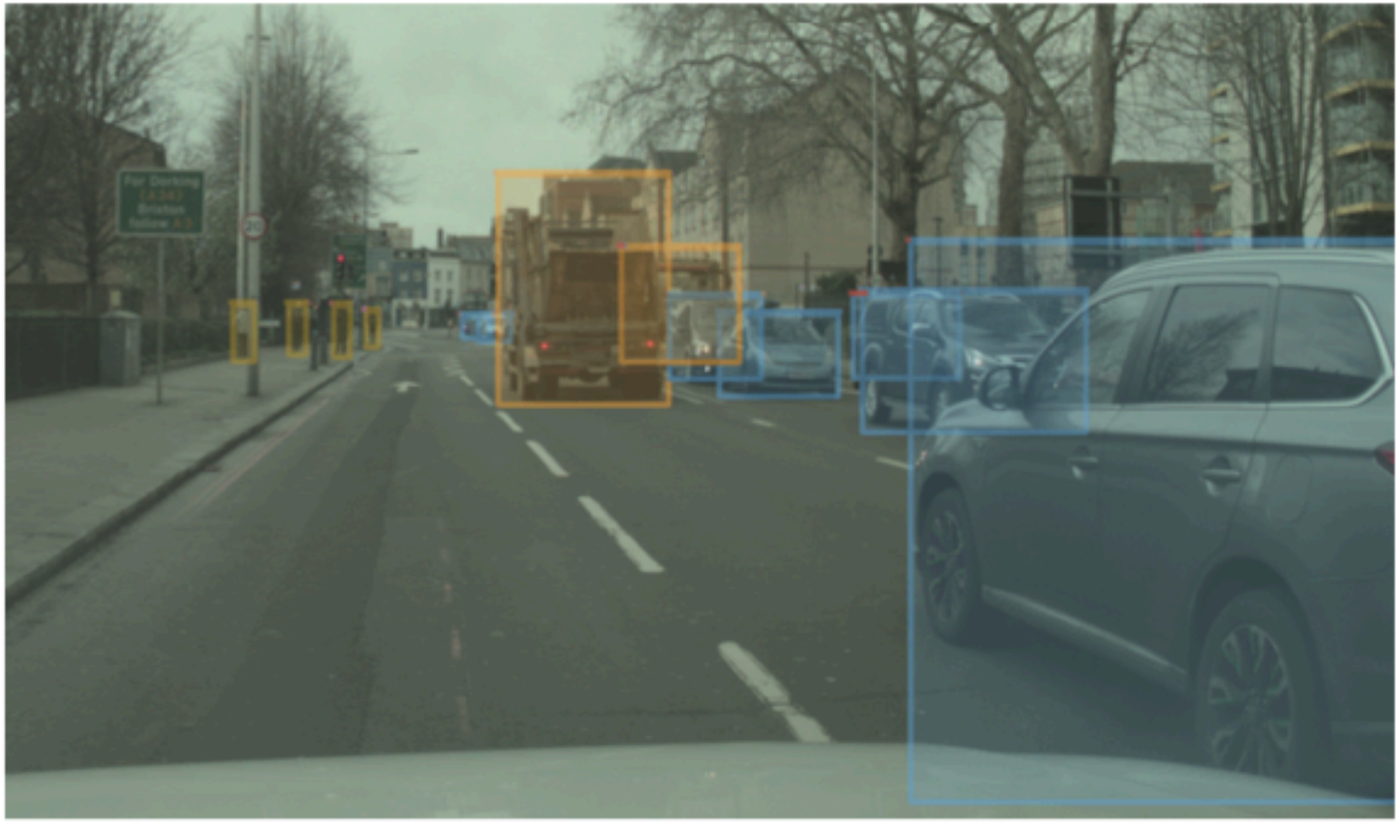
image-net.org

Must collect and *manually* annotate huge amounts of data.
Expensive. Ethical questions. Classifcation errors.

One of the first large data sets used for training large convolutional networks was created in the public domain: Imagenet with $> 10^7$ images in 10^3 categories.



Data sets: autonomous driving



Object recognition using a deep convolutional network. Shown is a frame from a movie recorded by a data-collection vehicle of the company Zenseact. The neural net recognises pedestrians, cars, and lorries, and localises them in the image by bounding boxes. Copyright © Zenseact AB 2020. Reproduced with permission.

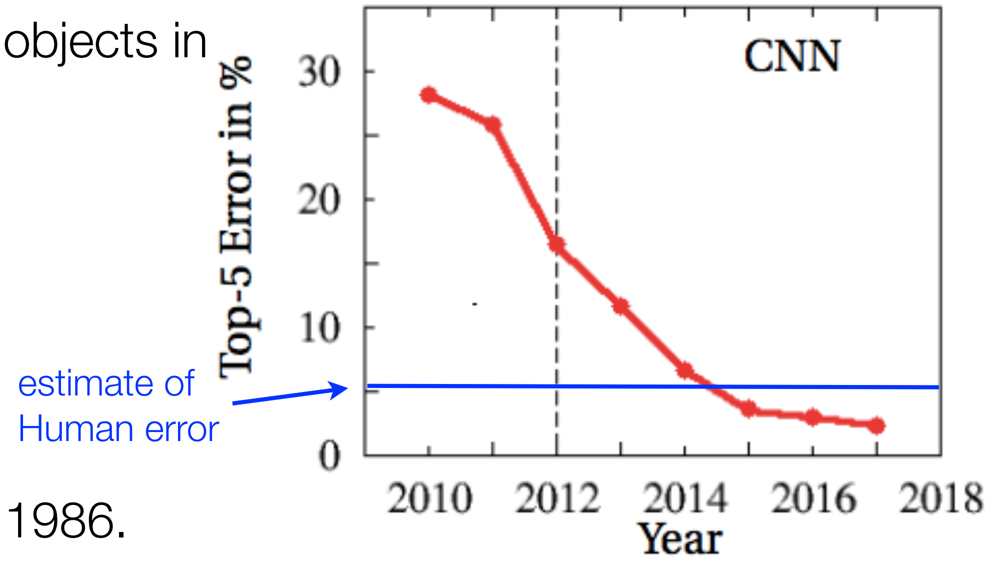
Object location and classification

Goodfellow, Bengio & Courville (2016)

Deep convolutional nets locate and classify objects in images with very high accuracy.

Better than humans?

cs.stanford.edu/people/karpathy/ilsvrc



Convolutional nets have been around since 1986.

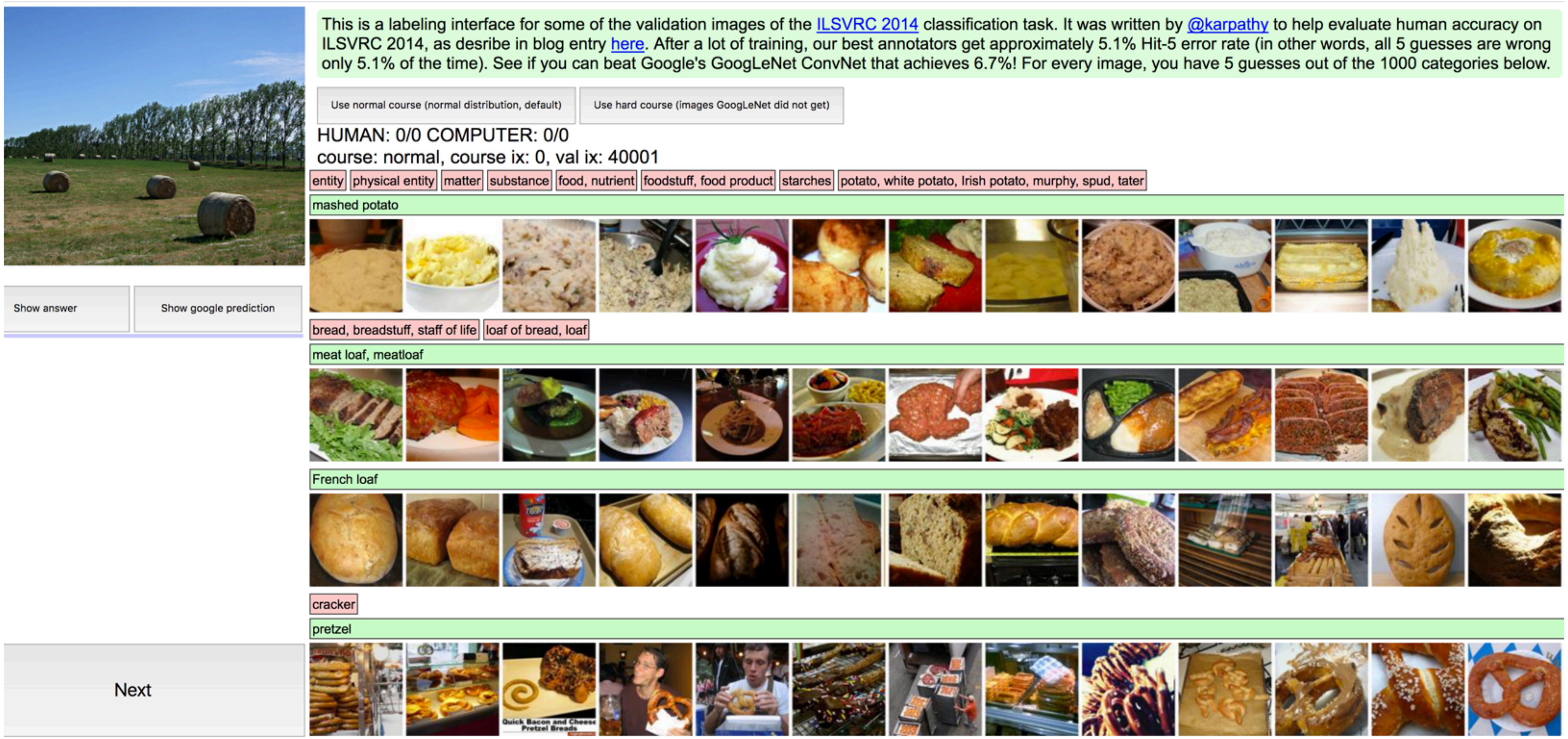
Patterns (T and C) detected by convolutional net Rumelhart, Hinton & Williams (1986)

It was always thought that deep nets are very difficult to train (overfitting, slow learning)

Why does it suddenly work so well? Mainly: better training sets (larger sets and more accurate targets).

Better than humans?

cs.stanford.edu/people/karpathy/ilsvrc



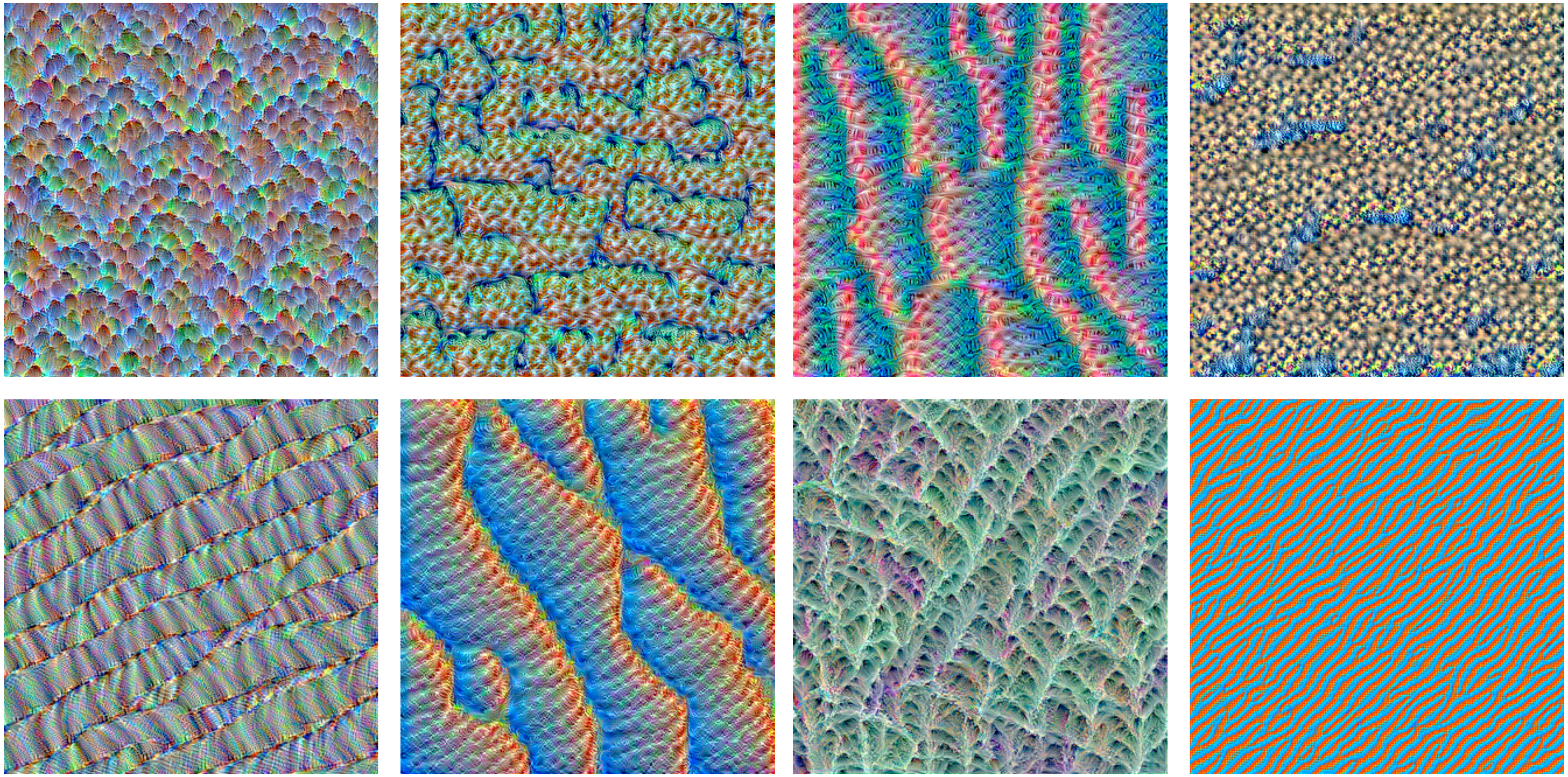
Manual annotation of training data

xkcd.com/1897

What do the hidden layers learn?

Mehlig, *Machine learning with neural networks*, Cambridge University Press (2021)

To which patterns do hidden neurons react most strongly?



Patterns that give largest activations of hidden neurons in residual convolutional network trained on the imagenet data base. F. M. Graetz https://towardsdatascience.com

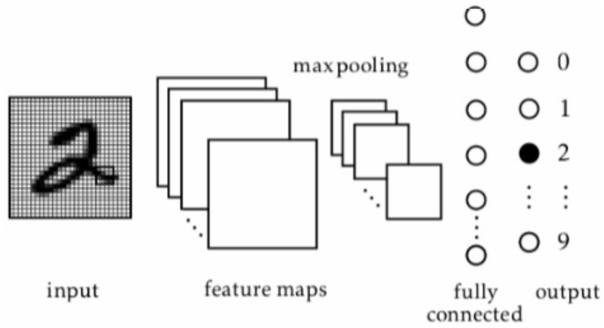
Recognising hand-written digits

LeCun, Cortes, Burges (2012)

THE MNIST DATABASE

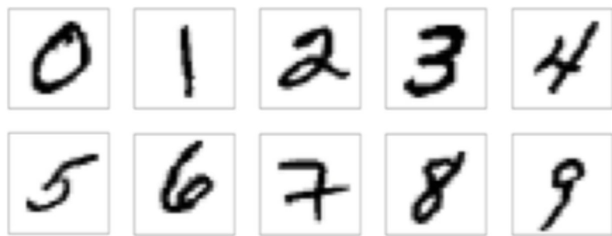
of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

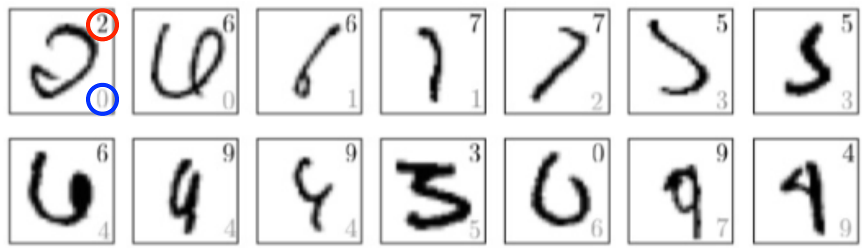


Training set: 60 000 hand-written digits, test set: 10 000 digits.

Convolutional network trained on training set classifies digits in test set with high accuracy. Best result: only 23 out of 10 000 wrong.
Ciresan, Meyer & Schmidhuber, arxiv:1202.2745



Some MNIST digits misclassified by a state-of-the-art network. Oleksandr Balabanov

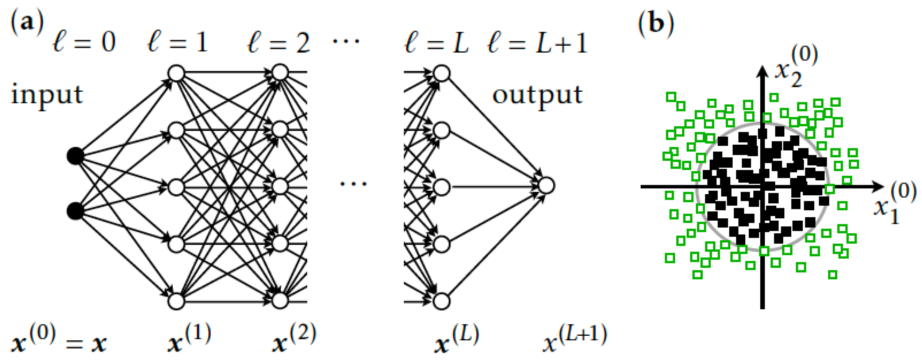


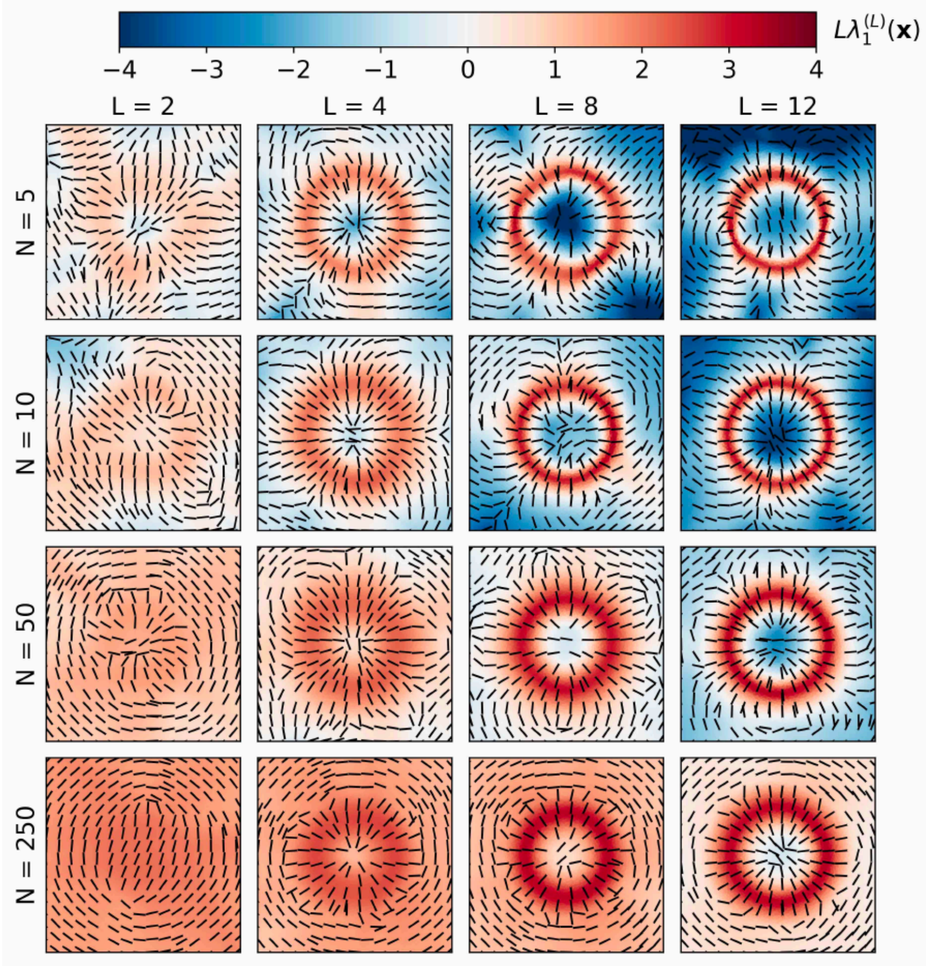
target
output

Lagrangian coherent structures

Storm, Linander, Bec, Gustavsson & Mehlig, Phys. Rev. Lett. (2024)

Neural networks are discrete dynamical systems, the layer index ℓ plays the role of time.





Ridges of large maximal finite-time Lyapunov exponent (local stretching)

$$\lambda_1^{(L)}(\boldsymbol{x}^{(0)}) = \frac{1}{L}\log\left|\frac{\delta\boldsymbol{x}^{(L)}}{\delta\boldsymbol{x}^{(0)}}\right|$$

on decision boundary

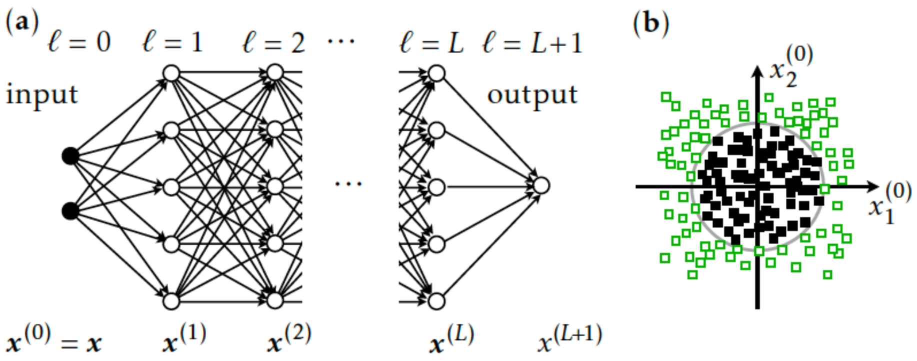
Lagrangian coherent structures.

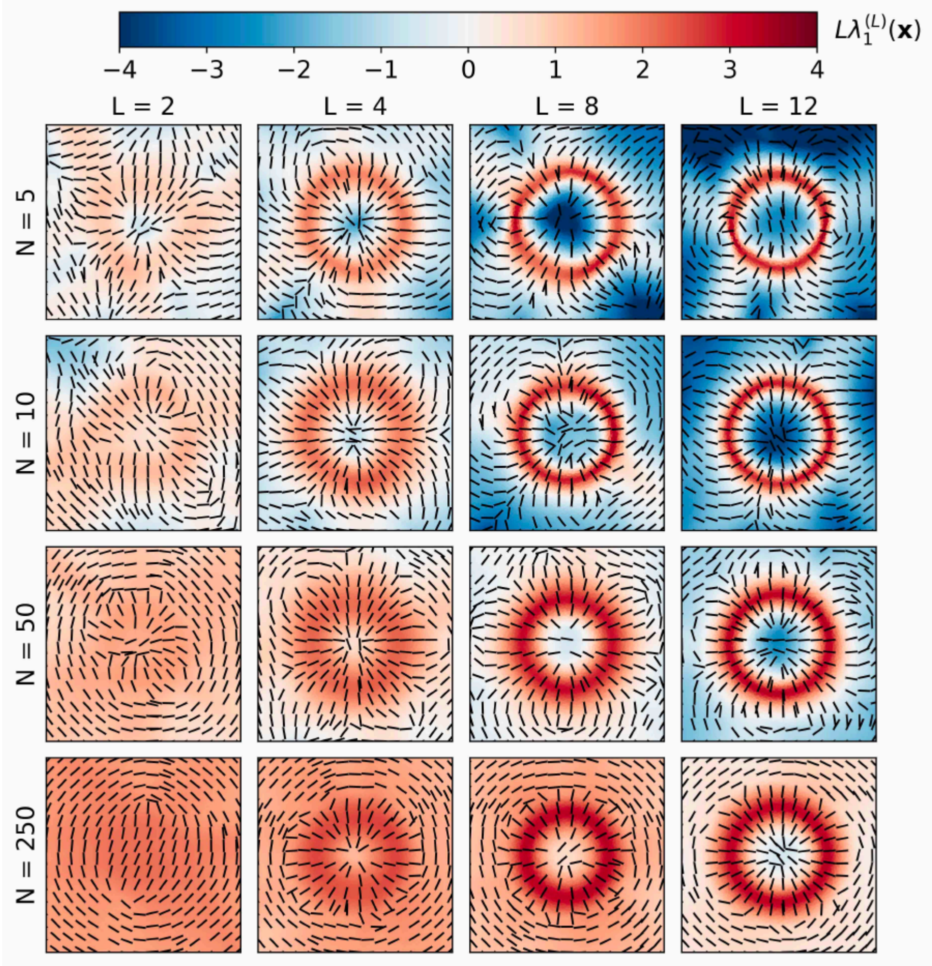
Haller, Ann. Rev. Fluid Mech. **47** (2015) 137

Lagrangian coherent structures

Storm, Linander, Bec, Gustavsson & Mehlig, Phys. Rev. Lett. (2024)

Neural networks are discrete dynamical systems, the layer index ℓ plays the role of time.





Ridges of large maximal finite-time Lyapunov exponent (local stretching)

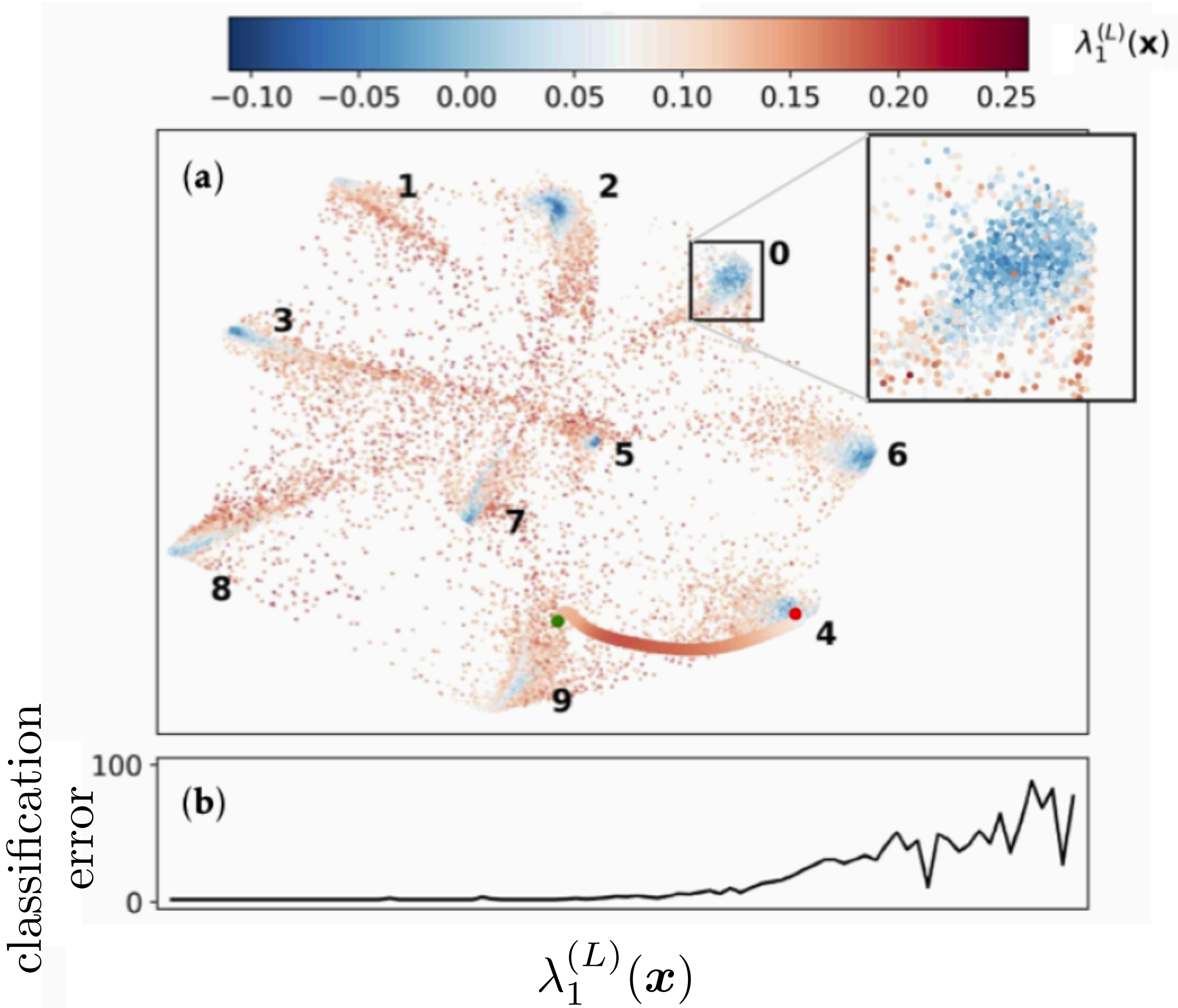
$$\lambda_1^{(L)}(\boldsymbol{x}^{(0)}) = \frac{1}{L} \log \left| \frac{\delta \boldsymbol{x}^{(L)}}{\delta \boldsymbol{x}^{(0)}} \right|$$

on decision boundary.
Lagrangian coherent structure = ridge of large $\lambda_1^{(L)}(\boldsymbol{x}^{(0)})$.

Haller, Ann. Rev. Fluid Mech. **47** (2015) 137

Lagrangian coherent structures

Storm, Linander, Bec, Gustavsson & Mehlig, Phys. Rev. Lett. (2024)



Handwritten digits

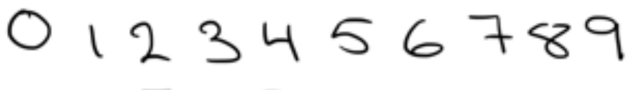
Nonlinear Projection of 28^2-dimensional input space to two dimensions.

red = decision boundary
= ridge of large $\lambda_1^{(L)}(\boldsymbol{x})$

Input deformations

Convolutional network trained on the MNIST data set (60 000 digits) classifies digits in a MNIST test set with high accuracy. Only 23 out of 10 000 wrong.

Ciresan, Meyer & Schmidhuber, arxiv:1202.2745

But the neural net fails on our own digits! 

Oleksandr Balabanov
arxiv:1901.05639

Neural nets excel at learning patterns in a given input distribution very precisely.
But they may fail if inputs come from a different distribution.

In this case a neural net may classify an input with high confidence ($O_i^{(\mu)} \approx \delta_{ij}$) but its prediction may nevertheless be wrong.

When can we be certain that the prediction is correct?

Fundamental problem: estimate uncertainty.

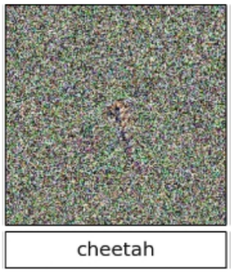
Further problems

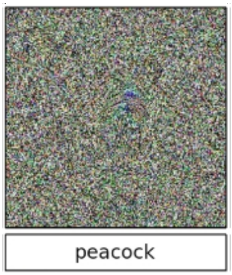
Convolutional nets do not *understand* what they see in the same way as Humans do.

original image correctly classified as *car*

slightly distorted image classified as *ostrich*

Convolutional net misclassifies slightly distorted image with high confidence Szegedy *et al.* arxiv:1312.6199





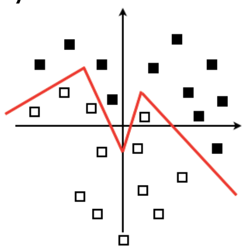
Convolutional net misclassifies noise with 99.6% certainty.

Nguyen *et al.* arxiv:1412.61897

classified as *leopard*

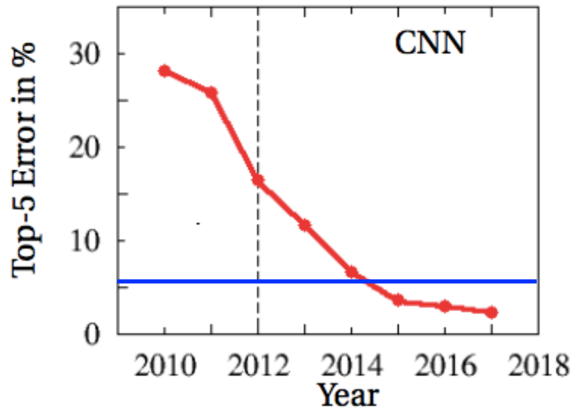
Khurshudov, blog (2015)

The nearest decision boundary is very close. Not intuitive but possible in high-dimensional input space.



Translational-invariant nature of convolutional layout makes it difficult to learn global features.

Conclusions



Artificial neural networks were already studied in the 80ies.

In the last few years: *deep-learning revolution* due to
- better training sets
- better hardware (GPUs, dedicated chips)

Applications: google, facebook, tesla, medical sciences,….

But: neural nets do not *understand* what they learn in the way Humans do, and the nets learn in a different way - which we do not fully understand.

Are neural networks *intelligent*?

What are the challenges we face as machine learning is more widely adopted in the natural sciences, and in society?

The algorithms get better and better (better training). What are the key risks?