Machine Intelligence

Lecture 8: Learning II

Thomas Dyhre Nielsen

Aalborg University

Tentative course overview

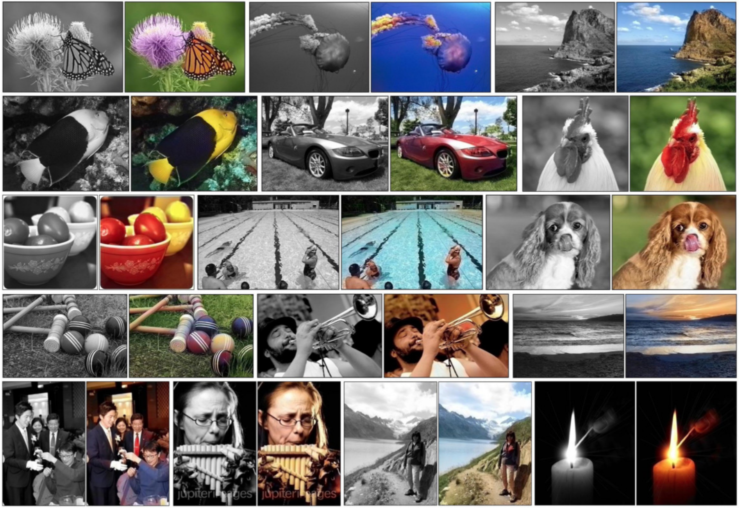
Topics:

- Introduction
- Search-based methods
- Constrained satisfaction problems
- Logic-based knowledge representation
- Representing domains endowed with uncertainty.
- Bayesian networks
- Inference in Bayesian networks
- **Machine learning**
- Planning
- Reinforcement learning
- Multi-agent systems

Neural Networks

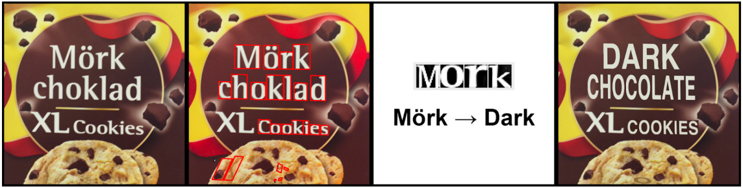
Example applications

Colorization of images



Zhang, Isola, Efros. Colorful Image Colorization. In ECCV, 2016.

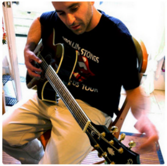
Text-image translation



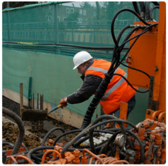
https://research.googleblog.com/2015/07/how-google-translate-squeezes-deep.html

Example applications

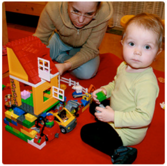
Image captioning



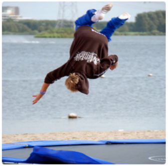
"man in black shirt is playing guitar."



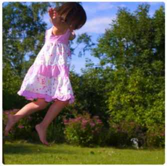
"construction worker in orange safety vest is working on road."



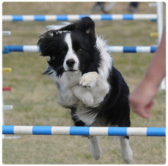
"two young girls are playing with lego toy."



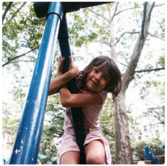
"boy is doing backflip on wakeboard."



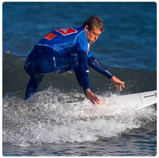
"girl in pink dress is jumping in air."



"black and white dog jumps over bar."



"young girl in pink shirt is swinging on swing."



"man in blue wetsuit is surfing on wave."

Andrej Karpathy, Li Fei-Fei, Deep Visual-Semantic Alignments for Generating Image Descriptions, CVPR 2015

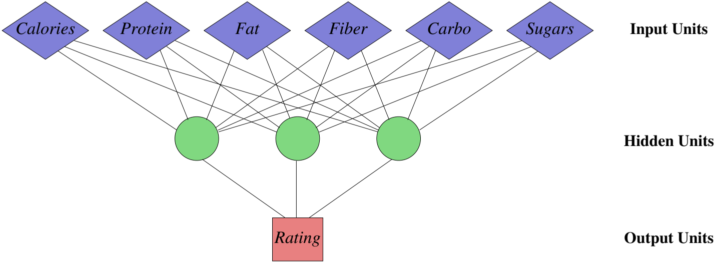
Regression: Example

Cereals Data

Name	Calories	Protein	Fat	Fiber	Carbo	Sugars	Rating
All-Bran	70	4	1	9	7	5	59.42
All-Bran_with_Extra_Fiber	50	4	0	14	8	0	93.70
Almond_Delight	110	2	2	1	14	8	34.38
Apple_Cinnamon_Cheerios	110	2	2	1.5	10.5	10	29.50
Apple_Jacks	110	2	0	1	11	14	33.17
Basic_4	130	3	2	2	18	8	37.03
Bran_Chex	90	2	1	4	15	6	49.12
Bran_Flakes	90	3	0	5	13	5	53.31
Cap_n_Crunch	120	1	2	0	12	12	18.04
Cheerios	110	6	2	2	17	1	50.76
...	...	...	...	...	...	...	...

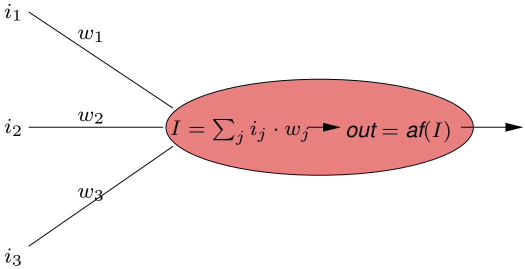
Problem: predict nutritional rating of cereal.

Neural Network



- Layered network of computational **units** (or *neurons*)
- Each unit has outputs of all units in preceding layer as inputs
- With each connection in the network there is an associated **weight**

Single Neuron



Two step computation:

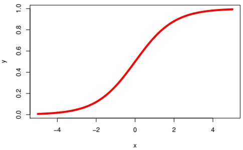
- Combine inputs as *weighted sum*
- Compute output by **activation function** of combined inputs

Activation Functions

The most common activation functions are:

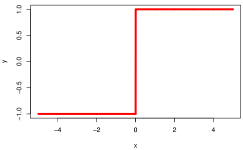
Sigmoid

$$\textit{af}(x) = 1/(1+e^{-x})$$



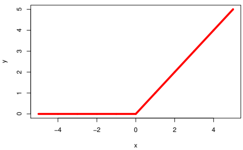
Sign

$$\textit{af}(x) = \textit{sign}(x)$$



Relu

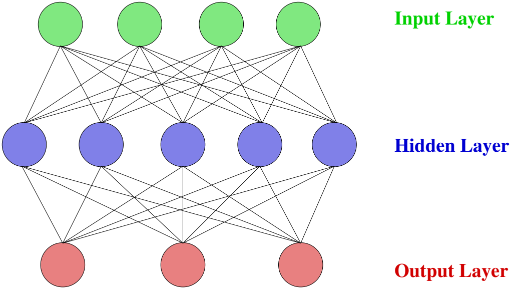
$$\textit{af}(x) = \max(0,x)$$



- If activation function is sigmoid, i.e. *out* $= \sigma(\sum_j i_j \cdot w_j)$, we also talk of *squashed linear function*.
- For the output neuron also the **identity function** is used: *af*$(x) =$ *id*$(x) = x$

Neural Networks

Neural Network Semantics



Given

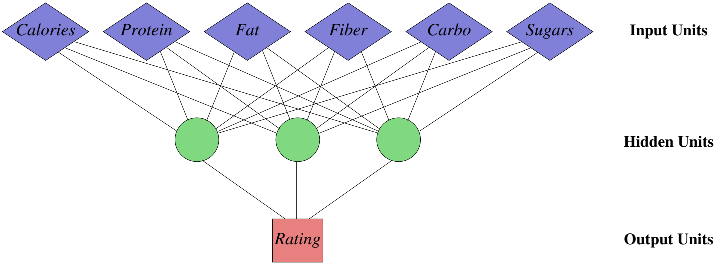
- the network structure,
- the weights associated with links/nodes,
- the activation function (usually the same for all hidden/output nodes)

a neural network with n input and k output nodes defines k real-valued functions on continuous input attributes:

$$o_i(a_1, \ldots, a_n) \in \mathbb{R} \qquad (i = 1, \ldots, k).$$

Neural Networks for Regression

Task: predict the (health-)*rating* of breakfast cereals based on their contents.
NN regression model, all predictor attributes are continuous:



Discrete Inputs

If the regression should also use discrete predictor attributes, e.g.:

Calories	*Protein*	*Sugars*	*Vitamins*	*Manufacturer*	*Rating*
70	105	8	135	Kellogs	59.3
110	80	23	99	Nabisco	43.6
...	...	...	...	...	...

Discrete Inputs

If the regression should also use discrete predictor attributes, e.g.:

Calories	*Protein*	*Sugars*	*Vitamins*	*Manufacturer*	*Rating*
70	105	8	135	Kellogs	59.3
110	80	23	99	Nabisco	43.6
...	...	...	...	...	...

replace discrete attributes with 0-1-valued <u>indicator variables</u> for their possible values:

Calories	*Protein*	*Sugars*	*Vitamins*	*M_Kellogs*	*M_Nabisco*	*M_xxx*	*Rating*
70	105	8	135	1	0	...	59.3
110	80	23	99	0	1	...	43.6
...	...	...	...	...	...	...	...

Discrete Features in general

Neural networks can also handle discrete features as inputs or outputs:

Indicator variables

- For each value x_i of X with domain $\{x_1, \ldots, x_k\}$ introduce a binary feature *X_is_*x_i with values 0,1.
- Encode input $X = x_i$ by inputs

$$\textit{X\_is\_}x_0 = 0, \ldots, \textit{X\_is\_}x_{i-1} = 0, \textit{X\_is\_}x_i = 1, \textit{X\_is\_}x_{i+1} = 0, \ldots, \textit{X\_is\_}x_k = 0$$

Discrete Features in general

Neural networks can also handle discrete features as inputs or outputs:

Indicator variables

- For each value x_i of X with domain $\{x_1, \ldots, x_k\}$ introduce a binary feature *X_is_*x_i with values 0,1.
- Encode input $X = x_i$ by inputs

$$\textit{X\_is\_}x_0 = 0, \ldots, \textit{X\_is\_}x_{i-1} = 0, \textit{X\_is\_}x_i = 1, \textit{X\_is\_}x_{i+1} = 0, \ldots, \textit{X\_is\_}x_k = 0$$

Numerical Encoding

Translate values into numbers, e.g.:

- *true*,*false* $\mapsto$ 1,0
- *low*,*medium*,*high* $\mapsto$ 0,1,2

Discrete Features in general

Neural networks can also handle discrete features as inputs or outputs:

Indicator variables

- For each value x_i of X with domain $\{x_1, \ldots, x_k\}$ introduce a binary feature *X_is_*x_i with values 0,1.
- Encode input $X = x_i$ by inputs

$$\mathit{X\_is\_}x_0 = 0, \ldots, \mathit{X\_is\_}x_{i-1} = 0, \mathit{X\_is\_}x_i = 1, \mathit{X\_is\_}x_{i+1} = 0, \ldots, \mathit{X\_is\_}x_k = 0$$

Numerical Encoding

Translate values into numbers, e.g.:

- *true*,*false* $\mapsto$ 1,0
- *low*,*medium*,*high* $\mapsto$ 0,1,2

Probably not sensible:

- *red*,*green*,*blue* $\mapsto$ 0,1,2

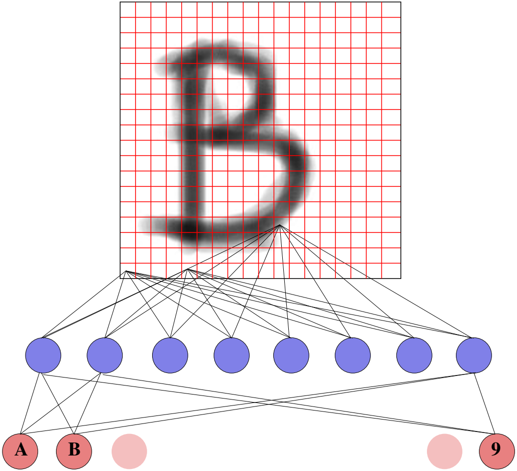
because *blue* is not “two times *green*”

Neural Networks for Classification

Task: hand-written character recognition. Predictor attributes: (continuous) grey-scale values for 18×18 grid cells. Class label: one of A,. . . ,Z,0,. . . ,9.

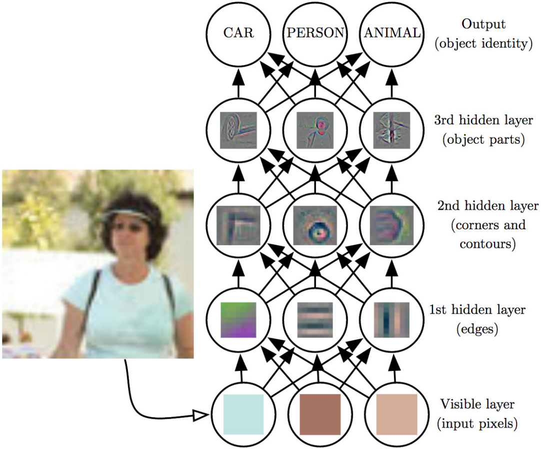
Neural Networks for Classification

Task: hand-written character recognition. Predictor attributes: (continuous) grey-scale values for 18×18 grid cells. Class label: one of A,. . . ,Z,0,. . . ,9.

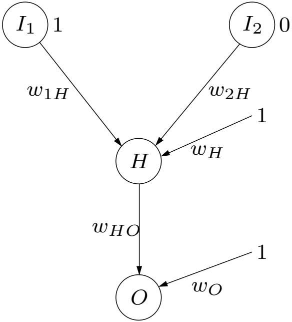


Use one output node for each class label.
Classify instance by class label associated with output node with highest output value.

Depth: repeated composition

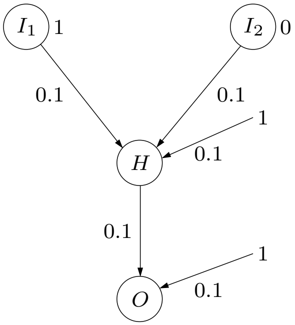


Propagation in Neural Networks



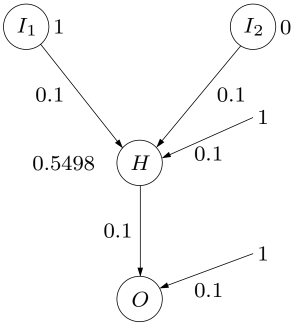
The input nodes are set to 1 and 0, respectively.

Propagation in Neural Networks



The input nodes are set to 1 and 0, respectively.

Propagation in Neural Networks

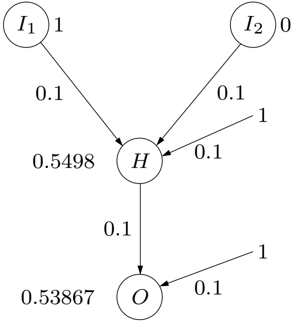


The output of of neuron H is:

$$o_H = \sigma(1 \cdot 0.1 + 0 \cdot 0.1 + 1 \cdot 0.1) = 0.5498.$$

The input nodes are set to 1 and 0, respectively.

Propagation in Neural Networks



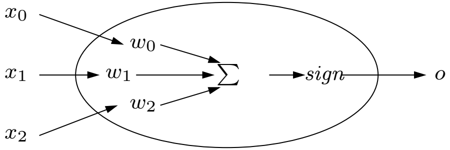
The output of of neuron O is:

$$o_H = \sigma(0.5498 \cdot 0.1 + 1 \cdot 0.1) = 0.53867.$$

The input nodes are set to 1 and 0, respectively.

Expressive power

The perceptron



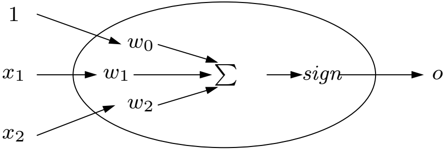
- No hidden layer
- One output neuron o
- *sign* activation function

Function computed:

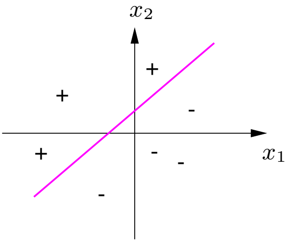
$$O(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \ldots w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

Convention: from now on assume that x_0 is an input neuron with constant input value 1. Then write $\mathbf{w} \cdot \mathbf{x}$ for $w_0 x_0 + w_1 x_1 + \ldots w_n x_n$.

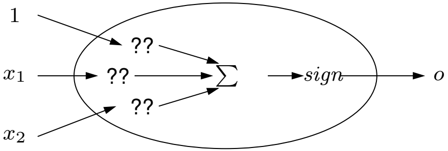
Expressive power



The decision surface of a two-input perceptron $a(x_1, x_2) = \text{sign}(x_1 \cdot w_1 + x_2 \cdot w_2 + w_0)$ is given by a straight line, separating positive and negative examples.



Expressive power

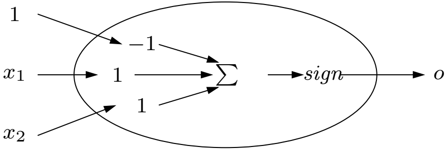


		x_1	
		-1	1
x_2	-1	-1	-1
	1	-1	1

$x_1 \wedge x_2$

Can the perceptron represent the Boolean function?

Expressive power

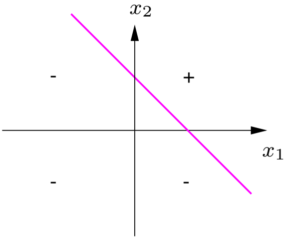


		x_1	
		-1	1
x_2	-1	-1	-1
	1	-1	1

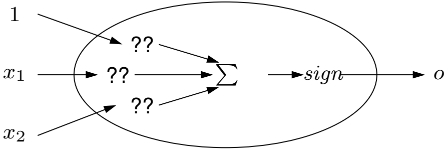
$x_1 \wedge x_2$

This perceptron specifies the decision surface:

$$-1 + 1 \cdot x_1 + 1 \cdot x_2 = 0$$



Perceptron examples

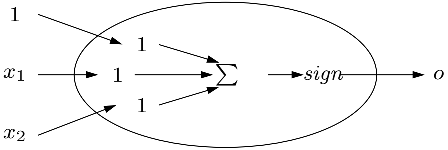


		x_1	
		-1	1
x_2	-1	-1	1
	1	1	1

$x_1 \vee x_2$

Can the perceptron represent the Boolean function?

Perceptron examples

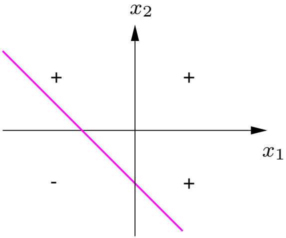


		x_1	
		-1	1
x_2	-1	-1	1
	1	1	1

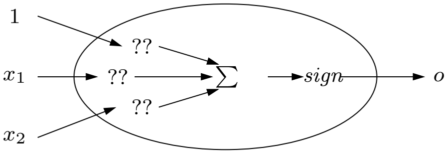
$x_1 \vee x_2$

This perceptron specifies the decision surface:

$$1 + 1 \cdot x_1 + 1 \cdot x_2 = 0$$



Expressive power

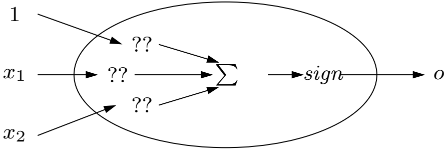


		x_1	
		-1	1
x_2	-1	-1	1
	1	1	-1

x_1 xor x_2

Can the perceptron represent the Boolean function?

Expressive power



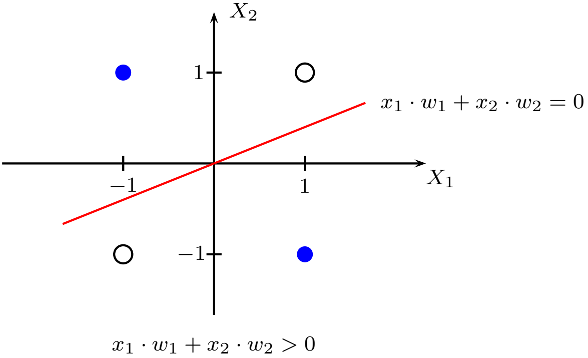
		x_1	
		-1	1
x_2	-1	-1	1
	1	1	-1

x_1 xor x_2

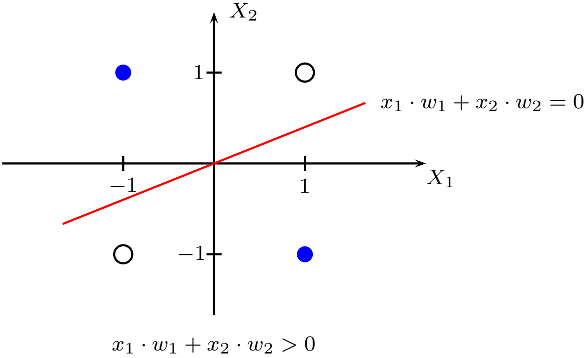
We cannot specify any values for the weights so that the perceptron can represent the “xor” function:

The examples are not linear separable!

Linear separable


X_2
1
-1
X_1
-1
1
$x_1 \cdot w_1 + x_2 \cdot w_2 = 0$
$x_1 \cdot w_1 + x_2 \cdot w_2 > 0$

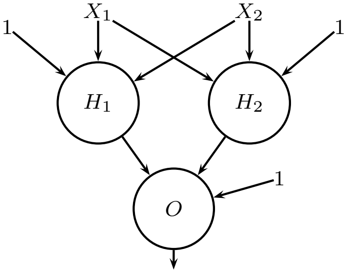
Linear separable


X_2
1
$x_1 \cdot w_1 + x_2 \cdot w_2 = 0$
-1
1
X_1
-1
$x_1 \cdot w_1 + x_2 \cdot w_2 > 0$

Only linearly separable functions can be computed by a single perceptron.

XOR again

Can multiple neurons help?

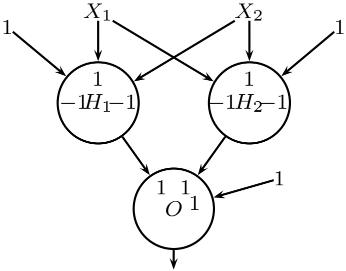


		X_1	
		-1	1
X_2	-1	-1	1
	1	1	-1

X_1 xor X_2

XOR again

Can multiple neurons help?

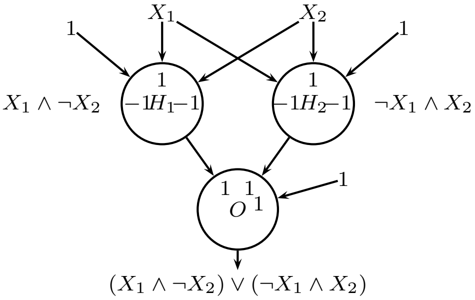


		X_1	
		-1	1
X_2	-1	-1	1
	1	1	-1

X_1 xor X_2

XOR again

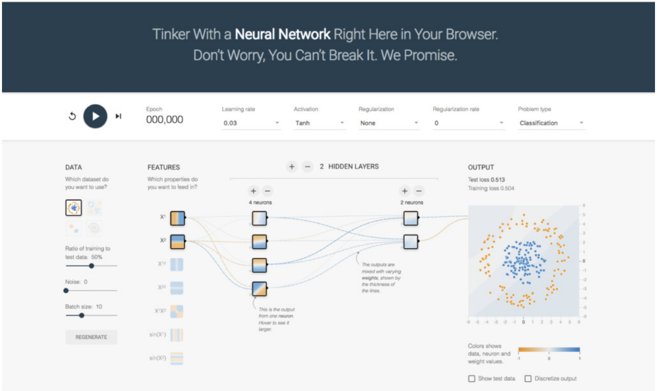
Can multiple neurons help?



		X_1	
		-1	1
X_2	-1	-1	1
	1	1	-1

X_1 xor X_2

Playground



Tinker With a **Neural Network** Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.

Epoch 000,000

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

FEATURES

Which properties do you want to feed in?

2 HIDDEN LAYERS

4 neurons

2 neurons

This is the output from one neuron. Hover to see it larger.

The outputs are mixed with varying weights, shown by the thickness of the lines.

OUTPUT

Test loss 0.513

Training loss 0.504

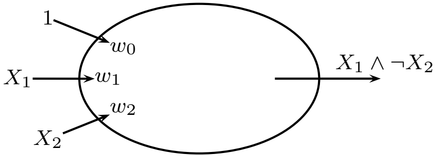
Colors shows data, neuron and weight values.

□ Show test data □ Discretize output

playground.tensorflow.org

The task of learning

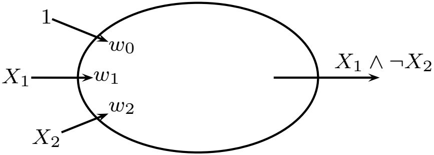
Learning weights and threshold I



		X_1	
		-1	1
X_2	-1	-1	1
	1	-1	-1

$X_1 \wedge \neg X_2$

Learning weights and threshold I



		X_1	
		-1	1
X_2	-1	-1	1
	1	-1	-1

$X_1 \wedge \neg X_2$

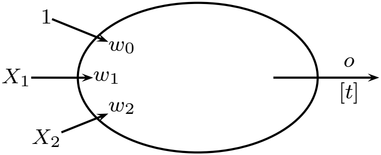
We have:

- $\mathcal{D} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$ input vectors (cases).
- $\mathbf{t} = (-1, 1, -1, -1)$ vector of target outputs.
- $\mathbf{w} = (w_0, w_1, w_2)$ vector of current parameters.
- $\mathbf{o} = (o_1, o_2, o_3, o_4)$ vector of current outputs.

We request:

- $\mathbf{w}^*$ parameters yielding $\mathbf{o} = \mathbf{t}$.

Learning weights and threshold I



Error: $E = t - o$

		X_1	
		-1	1
X_2	-1	-1	1
	1	-1	-1

$X_1 \wedge \neg X_2$

Weight updating procedure

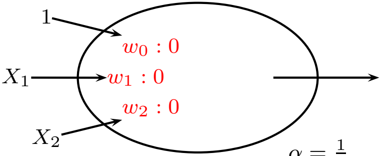
The weights are updated as follows:

- $E > 0 \Rightarrow o$ shall be increased $\Rightarrow \mathbf{x} \cdot \mathbf{w}$ up $\Rightarrow \mathbf{w} := \mathbf{w} + \alpha E \mathbf{x}$
- $E < 0 \Rightarrow o$ shall be decreased $\Rightarrow \mathbf{x} \cdot \mathbf{w}$ down $\Rightarrow \mathbf{w} := \mathbf{w} + \alpha E \mathbf{x}$

α is called the learning rate.

With $\mathcal{D}$ linearly separable and α not too large this procedure will converge to a correct set of parameters.

Example: $X_1 \wedge X_2$



$\alpha = \frac{1}{4}$

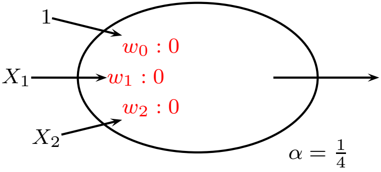
$$o = \begin{cases} 1 & \text{if } w_1 x_1 + w_2 x_2 + w_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$

		X_1	
		-1	1
X_2	-1	-1	-1
	1	-1	1

$X_1 \wedge X_2$

Cases:	$(1,-1,-1)$	$(1,1,-1)$	$(1,-1,1)$	$(1,1,1)$
$\mathbf{t}$:	-1	-1	-1	1
o:				
E:				

Example: $X_1 \wedge X_2$



$$o = \begin{cases} 1 & \text{if } w_1x_1 + w_2x_2 + w_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$

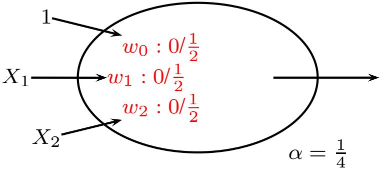
		X_1	
		-1	1
X_2	-1	-1	-1
	1	-1	1

$X_1 \wedge X_2$

Cases:	$(1,-1,-1)$	$(1,1,-1)$	$(1,-1,1)$	$(1,1,1)$
$\mathbf{t}$:	-1	-1	-1	1
o:	-1	-1	-1	-1
E:	0	0	0	$\underline{2}$

$$w := \begin{pmatrix}0\\0\\0\end{pmatrix} + \frac{1}{4}\cdot 2 \cdot \begin{pmatrix}1\\1\\1\end{pmatrix} = \begin{pmatrix}\frac{1}{2}\\\frac{1}{2}\\\frac{1}{2}\end{pmatrix}$$

Example: $X_1 \wedge X_2$



$\alpha = \frac{1}{4}$

$$o = \begin{cases} 1 & \text{if } w_1 x_1 + w_2 x_2 + w_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$

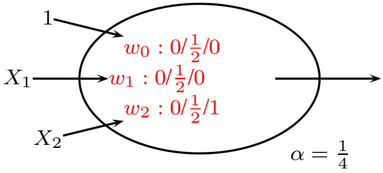
		X_1	
		-1	1
X_2	-1	-1	-1
	1	-1	1

$X_1 \wedge X_2$

Cases:	$(1,-1,-1)$	$(1,1,-1)$	$(1,-1,1)$	$(1,1,1)$
$\mathbf{t}$:	-1	-1	-1	1
o:	-1	1	1	1
E:	0	$\underline{-2}$	-2	0

$$w := \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} + \frac{1}{4} \cdot -2 \cdot \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Example: $X_1 \wedge X_2$



$$o = \begin{cases} 1 & \text{if } w_1x_1 + w_2x_2 + w_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$

		X_1	
		-1	1
X_2	-1	-1	-1
	1	-1	1

$X_1 \wedge X_2$

<u>Cases:</u>	$(1,-1,-1)$	$(1,1,-1)$	$(1,-1,1)$	$(1,1,1)$
$\mathbf{t}$:	-1	-1	-1	1
o:	-1	-1	1	1
E:	0	0	$\underline{-2}$	0

ETC... for a finite number of steps

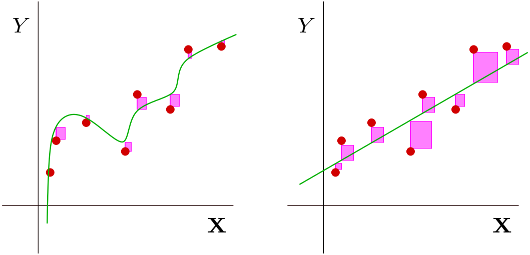
Sum of Squared Errors

Given:

- data (training examples): $(\mathbf{x}_i, y_i)$ $(i = 1, \ldots, N)$, with
  - $\mathbf{x}_i$: value of input features $\mathbf{X}$
  - y_i: value of output feature Y
- a neural network that computes outputs $o_i = \mathit{out}(\mathbf{x}_i)$

define **sum of squares error**

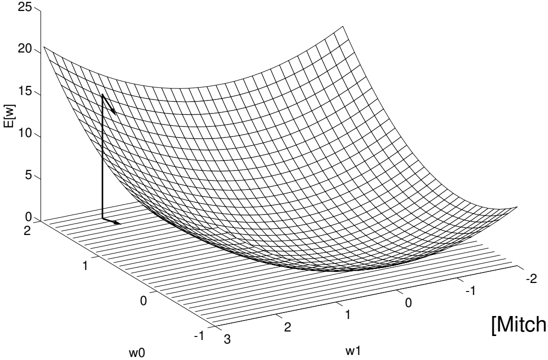
$$\mathit{SSE} = \sum_{i=1}^{N} (y_i - o_i)^2$$



Set of training examples (red dots), function defined by neural network (green), squared errors for each training example (area of magenta squares).

Minimizing SSE

For a given dataset, the SSE is a smooth, convex function of the weights.
Example for $n = 2$ (and a linear activation function):



[Mitchell, Fig.4.4]

$\rightsquigarrow$ weights $\mathbf{w}$ that minimize $E(\mathbf{w})$ can be found by gradient descent.

Gradient Descent Learning

The gradient is the vector of partial derivatives:

$$\nabla E[\mathbf{w}] = \left(\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n} \right)$$

The partial derivatives are (with a linear activation function):

$$\frac{\partial E}{\partial w_k} = \sum_{i=1}^{N} (t_i - \mathbf{w} \cdot \mathbf{x}_i)(-x_{i,k}).$$

Gradient descent rule:

```
Initialize $\mathbf{w}$ with random values
repeat
    $\mathbf{w} := \mathbf{w} - \eta \nabla E(\mathbf{w})$
until $\nabla E(\mathbf{w}) \approx \mathbf{0}$
```

(η is again a small constant, the learning rate).

Properties:

- The procedure converges to the weights $\mathbf{w}$ that minimize the SSE (if η is small enough).

Stochastic Gradient Descent

Variation of gradient descent: instead of following the gradient computed from the whole dataset:

$$\frac{\partial E}{\partial w_i} = \sum_{k=1}^{N} (t_k - \mathbf{w} \cdot \mathbf{x}_k)(-x_{k,i}),$$

iterate through the data instances one by one, and in one iteration follow the gradient defined by a single data instance $(\mathbf{x}_k, t_k)$:

$$\frac{\partial E}{\partial w_i} = (t_k - \mathbf{w} \cdot \mathbf{x}_k)(-x_{k,i}),$$

The Task of Learning

Given: structure and activation functions. To be learned: weights.
Goal: given the training examples

Input				Output			
X_1	X_2	$\ldots$	X_n	X_1	X_2	$\ldots$	X_m
$x_{1,1}$	$x_{2,1}$	$\ldots$	$x_{n,1}$	$t_{1,1}$	$t_{2,1}$	$\ldots$	$t_{m,1}$
$x_{1,2}$	$x_{2,2}$	$\ldots$	$x_{n,2}$	$t_{1,2}$	$t_{2,2}$	$\ldots$	$t_{m,2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$x_{1,N}$	$x_{2,N}$	$\ldots$	$x_{n,N}$	$t_{1,N}$	$t_{2,N}$	$\ldots$	$t_{m,N}$

Find the weights that minimize the *sum of squared errors (SSE)*

$$\sum_{i=1}^{N} \sum_{j=1}^{m} (t_{j,i} - o_{j,i})^2,$$

where $o_{j,i}$ is the value of the jth output neuron for the ith data instance.

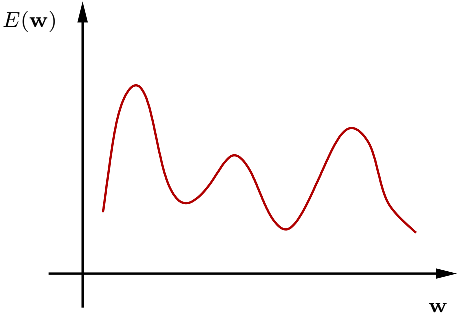
Gradient Descent for Multilayer NN

As for perceptron with SSE error:

- Error is smooth function of weights $\mathbf{w}$
- Can use gradient descent to optimize weights

but:

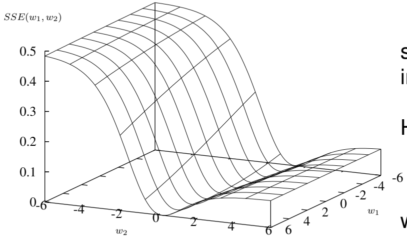
- Error no longer convex, can have multiple local minima:



- Partial derivatives more difficult to compute

Learning

Basic principle: SSEis a differentiable function of the weights (for differentiable activation functions such as the sigmoid function!). Use *gradient descent* to optimize SSE:



$$\nabla SSE(\mathbf{w}) = \left(\frac{\partial SSE}{\partial w_0}, \ldots, \frac{\partial SSE}{\partial w_n} \right)$$

specifies the direction of steepest increase in SSE.

Hence, our new training rule becomes:

$$w_i := w_i + \Delta w_i,$$

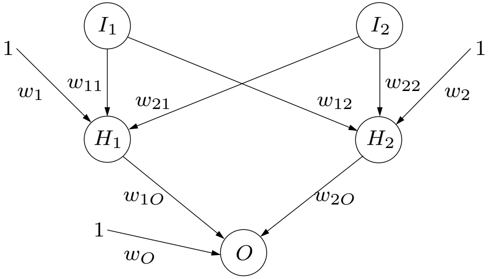
where

$$\Delta w_i = -\eta \frac{\partial SSE}{\partial w_i}$$

In practice: use the *back propagation algorithm* (approximation of gradient descent)

The Principle of Back Propagation

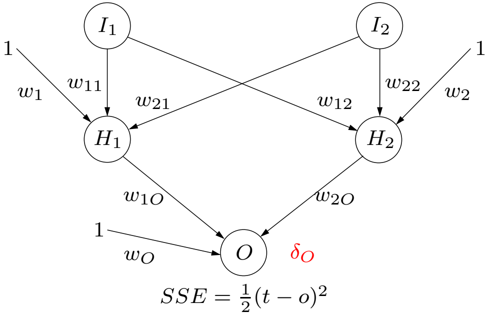
Training examples provide target values for only network outputs, so no target values are directly available for indicating the error of the hidden units' values.



$$SSE = \frac{1}{2}(t-o)^2$$

The Principle of Back Propagation

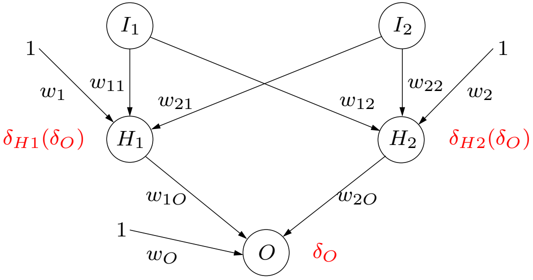
Training examples provide target values for only network outputs, so no target values are directly available for indicating the error of the hidden units' values.



Idea: Calculate an error term δ_h for a hidden unit by taking the weighted sum of the error terms, δ_k, for each output units it influences.

The Principle of Back Propagation

Training examples provide target values for only network outputs, so no target values are directly available for indicating the error of the hidden units' values.



$$SSE = \frac{1}{2}(t - o)^2$$

Idea: Calculate an error term δ_h for a hidden unit by taking the weighted sum of the error terms, δ_k, for each output units it influences.

Updating Rules

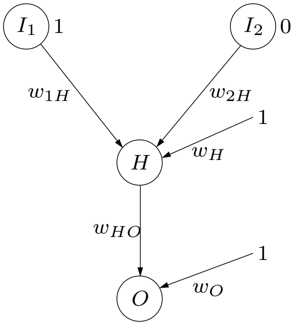
When using a sigmoid activation function we can derive the following updating rule:

$$w_{ij}^{\text{new}} := w_{ij}^{\text{current}} + \underset{\underset{\text{learning rate}}{\uparrow}}{\eta} \cdot \underset{\underset{\text{error term}}{\uparrow}}{\delta_j} \cdot \underset{\underset{\text{input}}{\uparrow}}{x_{ij}},$$

where

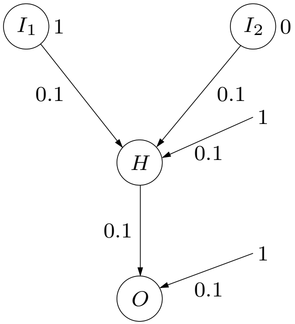
$$\delta_j = \begin{cases} o_j(1-o_j)(t-o_j) & \text{for output nodes,} \\ o_j(1-o_j)\sum_{k=1}^{m} w_{jk}\delta_k & \text{for hidden nodes.} \end{cases}$$

Back Propagation Example



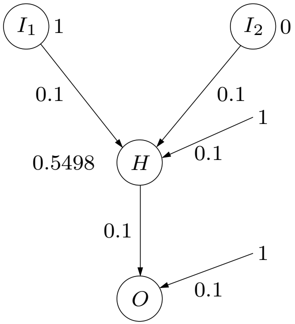
Assume that we have the training example $(I_1 = 1, I_2 = 0, O = 1)$.

Back Propagation Example



Assume that we have the training example $(I_1 = 1, I_2 = 0, O = 1)$.

Back Propagation Example

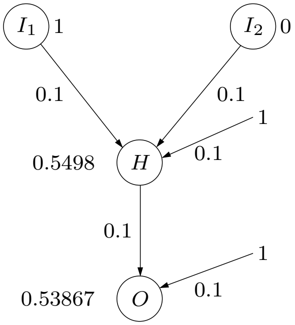


The output of of neuron H is:

$$o_H = \sigma(1 \cdot 0.1 + 0 \cdot 0.1 + 1 \cdot 0.1) = 0.5498.$$

Assume that we have the training example $(I_1 = 1, I_2 = 0, O = 1)$.

Back Propagation Example

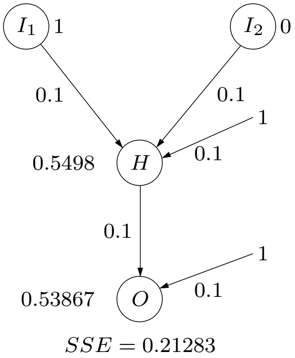


The output of of neuron O is:

$$o_H = \sigma(0.5498 \cdot 0.1 + 1 \cdot 0.1) = 0.53867.$$

Assume that we have the training example $(I_1 = 1, I_2 = 0, O = 1)$.

Back Propagation Example

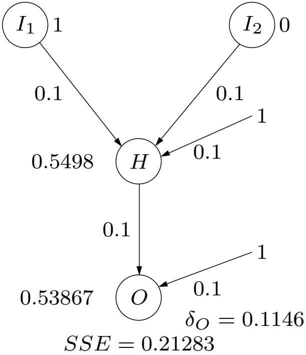


The SSEvalue is:

$$SSE = (1 - 0.53867)^2 = 0.21283.$$

Assume that we have the training example $(I_1 = 1, I_2 = 0, O = 1)$.

Back Propagation Example



The error term for node O is:

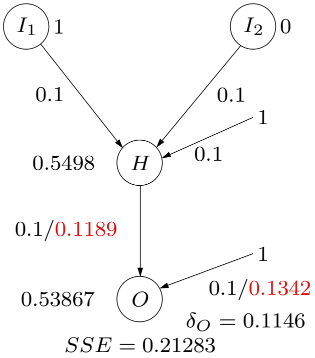
$\delta_O = 0.53867 \cdot (1 - 0.53867) \cdot (1 - 0.53867) = 0.1146.$

Recall:

$$\delta_O = o_j(1 - o_j)(O - o_j).$$

Assume that we have the training example $(I_1 = 1, I_2 = 0, O = 1)$.

Back Propagation Example



The updated weights are:

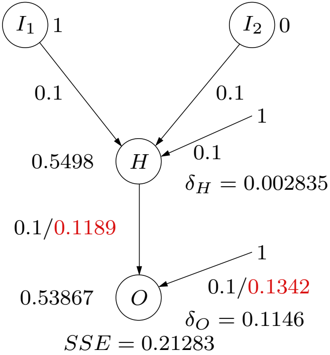
$$
\begin{aligned}
w_O &= 0.1 + [0.3 \cdot 0.1146 \cdot 1] = 0.1342, \\
w_{HO} &= 0.1 + [0.3 \cdot 0.1146 \cdot 0.5498] = 0.1189.
\end{aligned}
$$

Recall:

$$w_{ij}^{\text{new}} := w_{ij}^{\text{current}} + [\eta \cdot \delta_j \cdot x_{ij}].$$

Assume that we have the training example $(I_1 = 1, I_2 = 0, O = 1)$.

Back Propagation Example



The error term for node H is:

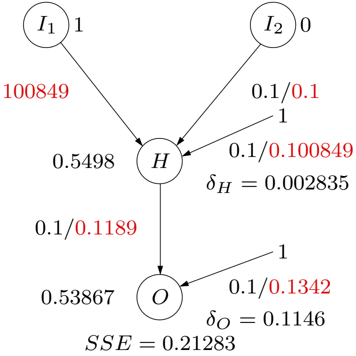
$\delta_H = 0.5498{\cdot}(1{-}0.5498){\cdot}0.1{\cdot}0.1146 = 0.002836.$

Recall:

$$\delta_H = o_j(1 - o_j)\sum_{k=1}^{m} w_{jk}\delta_k.$$

Assume that we have the training example $(I_1 = 1, I_2 = 0, O = 1)$.

Back Propagation Example



The updated weights are:

$$\begin{aligned} w_{1H} &= 0.1 + [0.3 \cdot 0.00283 \cdot 1] = 0.100849, \\ w_{2H} &= 0.1 + [0.3 \cdot 0.00283 \cdot 0] = 0.1, \\ w_{H} &= 0.1 + [0.3 \cdot 0.00283 \cdot 1] = 0.100849 \end{aligned}$$

Recall:

$$w_{ij}^{\text{new}} := w_{ij}^{\text{current}} + [\eta \cdot \delta_j \cdot x_{ij}].$$

Assume that we have the training example $(I_1 = 1, I_2 = 0, O = 1)$.