Artificial Intelligence

Neural Networks

Lesson 7:
Training Multi-layer Perceptrons

Vincenzo Piuri

Università degli Studi di Milano

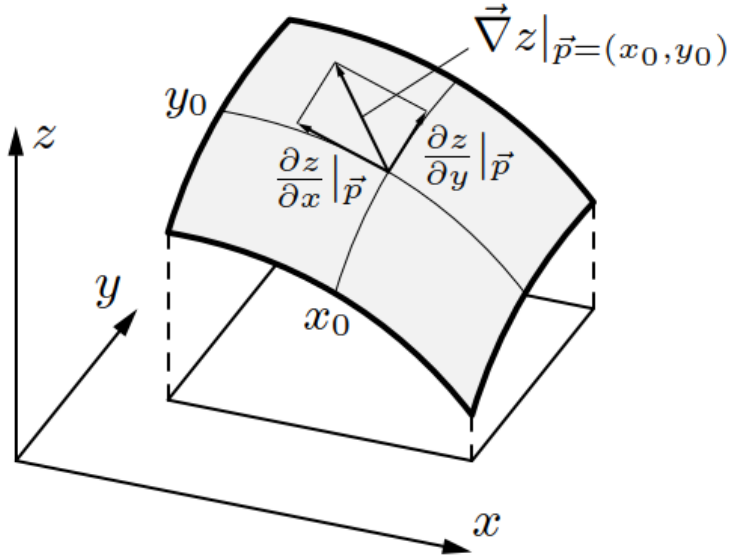
Contents

- Gradient descent
- Error backpropagation
- Number of hidden neurons
- Cross validation

Gradient Descent (1)

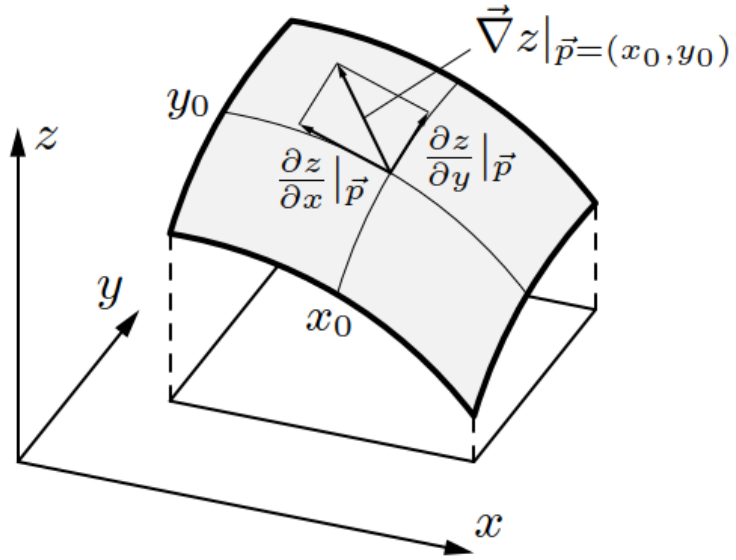
- Logistic regression works only for two-layer perceptrons
- General approach: **gradient descent**
  - Necessary to have differentiable activation and output functions

Gradient Descent (2)



- Gradient of a real-valued function $z = f(x, y)$ at a point (x_0, y_0)

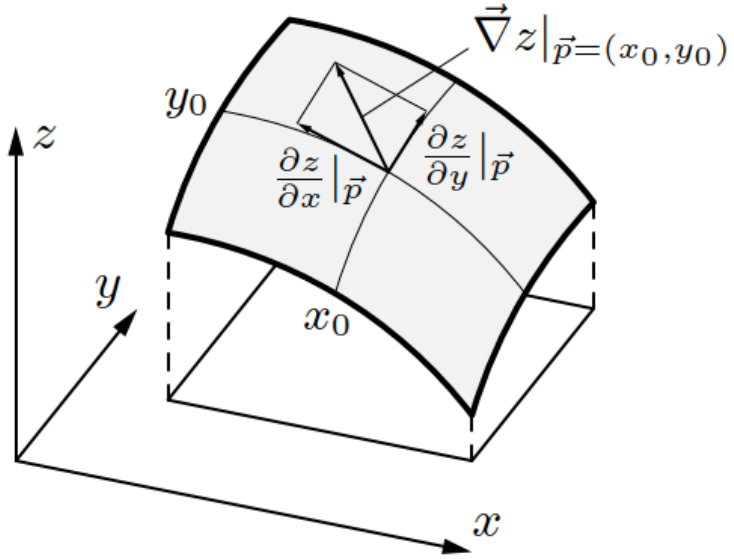
Gradient Descent (3)



- Approach the minimum of the error function in small steps

$$e = \sum_{l \in L_{\text{fixed}}} e^{(l)} = \sum_{v \in U_{\text{out}}} e_v = \sum_{l \in L_{\text{fixed}}} \sum_{v \in U_{\text{out}}} e_v^{(l)},$$

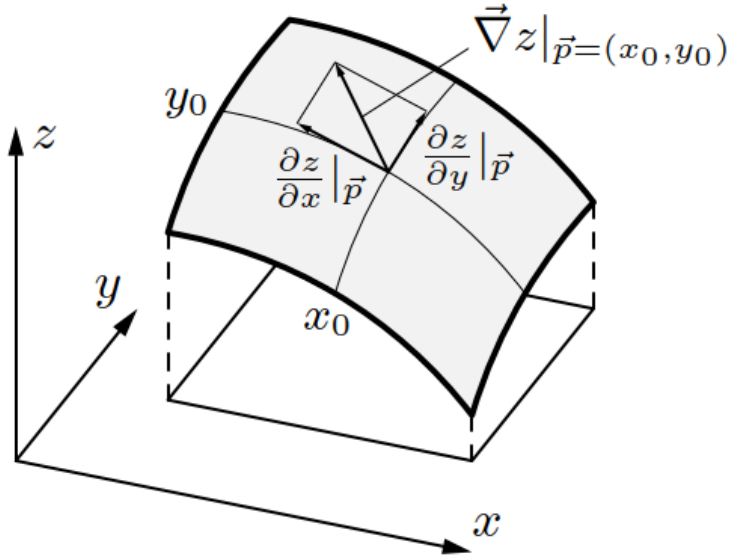
Gradient Descent (4)



- Form gradient to determine the direction of the step

$$\vec{\nabla}_{\vec{w}_u} e = \frac{\partial e}{\partial \vec{w}_u} = \left(-\frac{\partial e}{\partial \theta_u}, \frac{\partial e}{\partial w_{up_1}}, \ldots, \frac{\partial e}{\partial w_{up_n}} \right).$$

Gradient Descent (5)



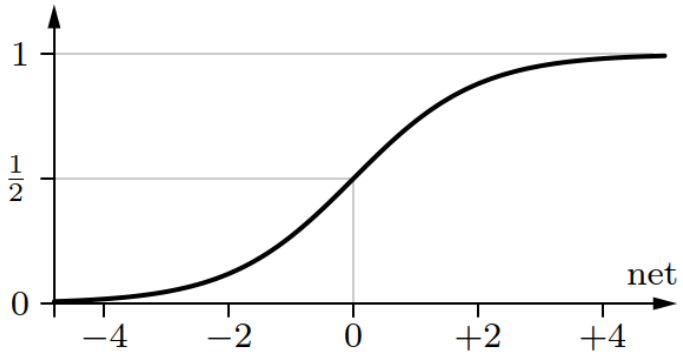
- Exploit the sum over the training patterns

$$\vec{\nabla}_{\vec{w}_u} e = \frac{\partial e}{\partial \vec{w}_u} = \frac{\partial}{\partial \vec{w}_u} \sum_{l \in L_{\text{fixed}}} e^{(l)} = \sum_{l \in L_{\text{fixed}}} \frac{\partial e^{(l)}}{\partial \vec{w}_u}.$$

Gradient Descent (6)

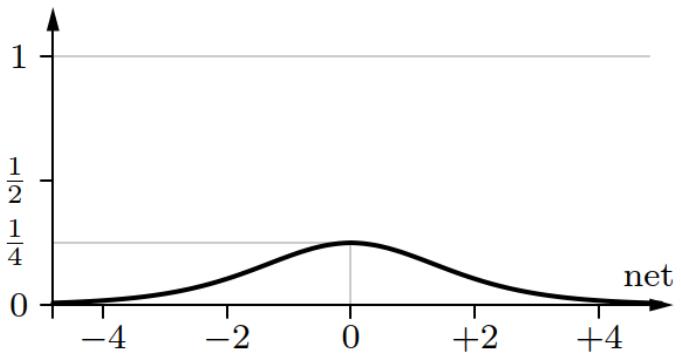
logistic activation function:

$$f_{\text{act}}(\text{net}_u^{(l)}) = \frac{1}{1 + e^{-\text{net}_u^{(l)}}}$$



derivative of logistic function:

$$f'_{\text{act}}(\text{net}_u^{(l)}) = f_{\text{act}}(\text{net}_u^{(l)}) \cdot (1 - f_{\text{act}}(\text{net}_u^{(l)}))$$



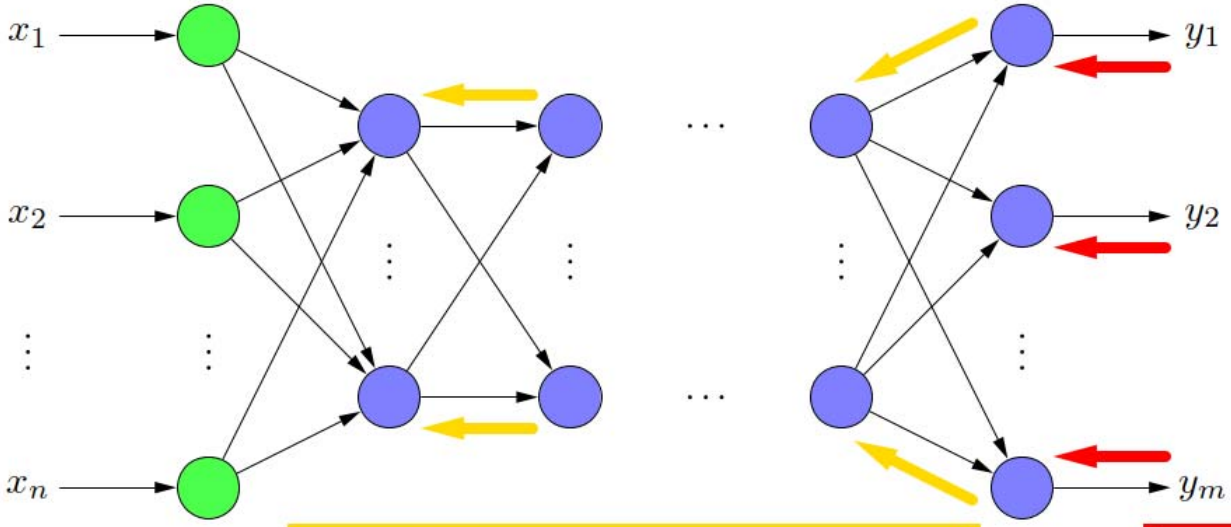
- If logistic activation function f_{act} is used, the weight changes are proportional to f'_{act}
  - Weight changes (and thus the training speed) are highest close to 0
  - Far from 0, gradient becomes small and training slow

Error Backpropagation (1)

$\forall u \in U_{\text{in}}$:
$\text{out}_u^{(l)} = \text{ext}_u^{(l)}$

forward propagation:

$\forall u \in U_{\text{hidden}} \cup U_{\text{out}}$:
$\text{out}_u^{(l)} = \left(1 + \exp\left(-\sum_{p \in \text{pred}(u)} w_{up}\, \text{out}_p^{(l)}\right)\right)^{-1}$



- logistic activation function
- implicit bias value

backward propagation:

$\forall u \in U_{\text{hidden}}$:
$\delta_u^{(l)} = \left(\sum_{s \in \text{succ}(u)} \delta_s^{(l)} w_{su}\right) \lambda_u^{(l)}$

error factor:

$\forall u \in U_{\text{out}}$:
$\delta_u^{(l)} = \left(o_u^{(l)} - \text{out}_u^{(l)}\right) \lambda_u^{(l)}$

activation derivative:

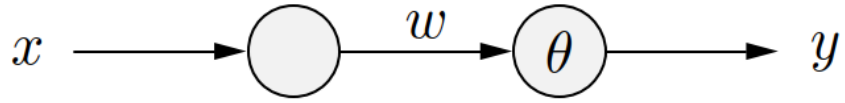
$\lambda_u^{(l)} = \text{out}_u^{(l)} \left(1 - \text{out}_u^{(l)}\right)$

weight change:

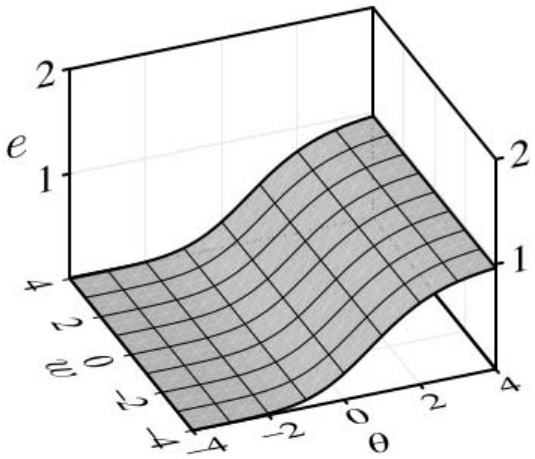
$\Delta w_{up}^{(l)} = \eta\, \delta_u^{(l)}\, \text{out}_p^{(l)}$

Error Backpropagation (2)

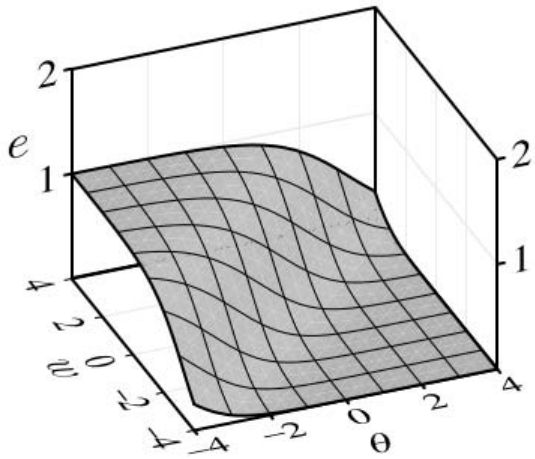
- Gradient descent training for the negation $\neg x$



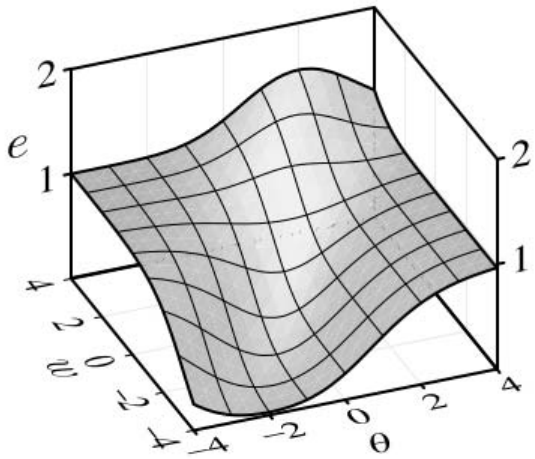
x	y
0	1
1	0



error for $x = 0$



error for $x = 1$



sum of errors

Error Backpropagation (2)

- Error cannot vanish completely due to the properties of the logistic function

epoch	θ	w	error
0	3.00	3.50	1.307
20	3.77	2.19	0.986
40	3.71	1.81	0.970
60	3.50	1.53	0.958
80	3.15	1.24	0.937
100	2.57	0.88	0.890
120	1.48	0.25	0.725
140	−0.06	−0.98	0.331
160	−0.80	−2.07	0.149
180	−1.19	−2.74	0.087
200	−1.44	−3.20	0.059
220	−1.62	−3.54	0.044

Online Training

epoch	θ	w	error
0	3.00	3.50	1.295
20	3.76	2.20	0.985
40	3.70	1.82	0.970
60	3.48	1.53	0.957
80	3.11	1.25	0.934
100	2.49	0.88	0.880
120	1.27	0.22	0.676
140	−0.21	−1.04	0.292
160	−0.86	−2.08	0.140
180	−1.21	−2.74	0.084
200	−1.45	−3.19	0.058
220	−1.63	−3.53	0.044

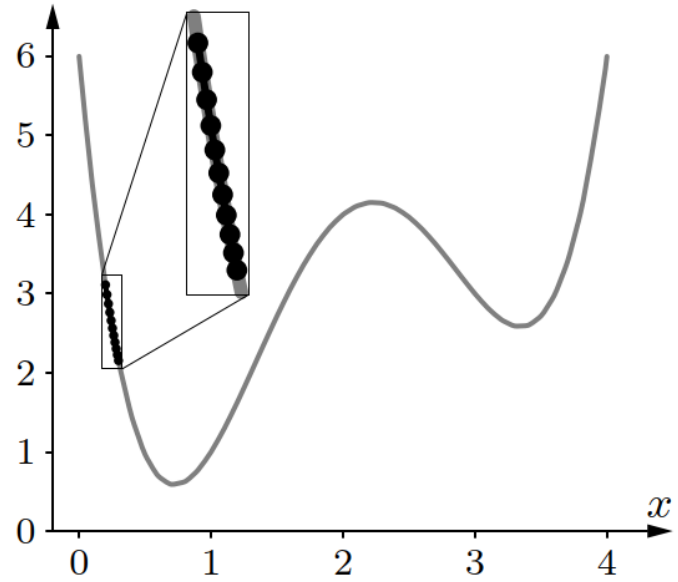
Batch Training

Error Backpropagation (3)

- Gradient descent with initial value 0.2 and learning rate 0.001

Example function: $f(x) = \frac{5}{6}x^4 - 7x^3 + \frac{115}{6}x^2 - 18x + 6,$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	0.200	3.112	−11.147	0.011
1	0.211	2.990	−10.811	0.011
2	0.222	2.874	−10.490	0.010
3	0.232	2.766	−10.182	0.010
4	0.243	2.664	−9.888	0.010
5	0.253	2.568	−9.606	0.010
6	0.262	2.477	−9.335	0.009
7	0.271	2.391	−9.075	0.009
8	0.281	2.309	−8.825	0.009
9	0.289	2.233	−8.585	0.009
10	0.298	2.160		

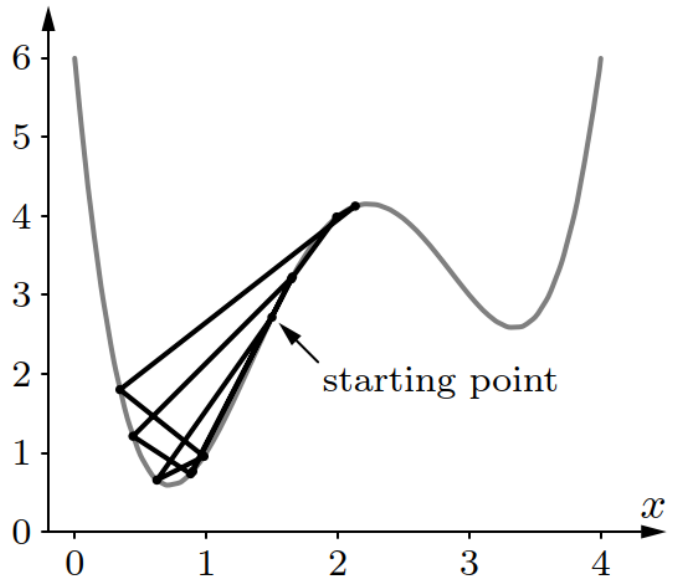


Error Backpropagation (4)

- Gradient descent with initial value 1.5 and learning rate 0.25

Example function: $f(x) = \frac{5}{6}x^4 - 7x^3 + \frac{115}{6}x^2 - 18x + 6,$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	1.500	2.719	3.500	−0.875
1	0.625	0.655	−1.431	0.358
2	0.983	0.955	2.554	−0.639
3	0.344	1.801	−7.157	1.789
4	2.134	4.127	0.567	−0.142
5	1.992	3.989	1.380	−0.345
6	1.647	3.203	3.063	−0.766
7	0.881	0.734	1.753	−0.438
8	0.443	1.211	−4.851	1.213
9	1.656	3.231	3.029	−0.757
10	0.898	0.766		



Stuck in Local Minimum

Empiric approach to increase chances to reach the global minimum (or a good local minimum):

- Repeat learning with different starting points
- Choose the best solution

Variants of Gradient Descend (1)

- Goals:
  - Learning rate
  - Length of the learning step

- **Manhattan training**: sign of the gradient
- **Flat spot elimination**: lifting of the derivative of the activation function
- **Momentum term**: add a fraction of the preceeding weight change to the gradient step
- **Self-adaptative Error Backpropagation**: independent learning rate for each parameter, adapte according to previous and current gradient

Variants of Gradient Descend (2)

- **Resilient Error Backpropagation**: mixes Manhattan training and self-adaptive backpropagation
- **Quick Propagation**: approximates the error function at the current location by a parabola and jumps to the apex
- **Weight Decay**: reduce the weight to prevent get stuck in the saturation region

Number of Hidden Neurons (1)

- For a single hidden layer the following **rule of thumb** is popular:
  - Number of hidden neurons =
    (number of inputs + number of outputs) / 2

Number of Hidden Neurons (2)

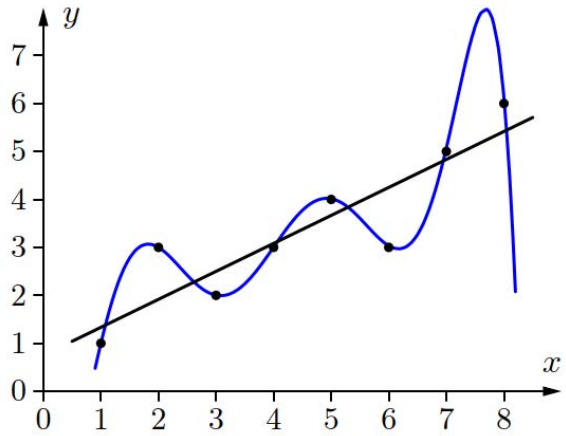
- **Underfitting**: if the number of neurons in the hidden layer is too small, the multi-layer perceptron may not be able to capture the structure of the relationship between inputs and outputs precisely enough due to a lack of parameters.

Number of Hidden Neurons (3)

- **Overfitting**: with a larger number of hidden neurons a multi-layer perceptron may adapt not only to the regular dependence between inputs and outputs, but also to the accidental specifics (errors and deviations) of the training data set.

Number of Hidden Neurons (4)

- Avoid overfitting: select the model that best fits the data, taking the model complexity into account.
  - The more complex the model, the better it usually fits the data.



black line:
regression line
(2 free parameters)

blue curve:
7th order regression polynomial
(8 free parameters)

  - The blue curve fits the data points perfectly, but it is not a good model.

Cross Validation (1)

- **Training and validation data**
  - Randomly split the given data into two subsets: training data and validation data
  - Train multi-layer perceptrons with different numbers of hidden neurons on the training data and evaluate them on the validation data
  - Repeat the random split of the data and training/evaluation many times and average the results
  - Choose the number of hidden neurons with the best average error

Cross Validation (2)

- **N-fold cross validation**
  - The given data set is split into n parts or subsets (also called folds) of about equal size
  - The relative frequency of the output values in the subsets/folds represent as well as possible the relative frequencies of these values in the data set as a whole (**stratification**)
  - Out of these n data subsets (or folds) n pairs of training and validation data set are formed by using 1 fold as a validation and $n - 1$ folds for training

Cross Validation (3)

- **Stopping criterion**
  - During training the performance is evaluated after each epoch on validation data
  - The error on training data set should always decrease, while the error on validation data set should start increasing when overfitting
  - At this moment training is terminated
  - This method stops the training of a complex network early, rather than adjust the complexity of the network to the “correct” level