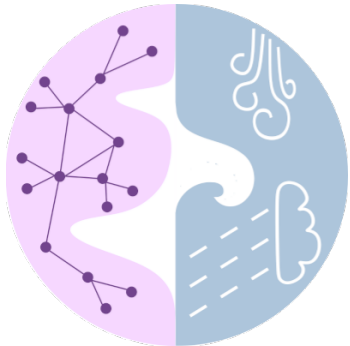
Introduction to Neural Networks



Jakob Schlör

University of Tübingen

November 30, 2022

Content

Part I: A Little Bit of Theory

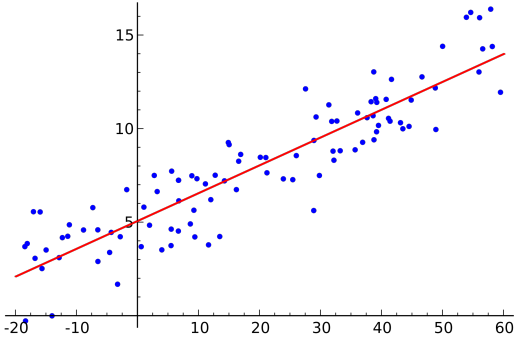
1. Linear regression
2. Neural Perceptron
3. Multilayer Perceptron (MLP)
4. Training an MLP

Part II: Jupyter Notebook

1. Linear Regression

As always ... we start with **linear regression**:

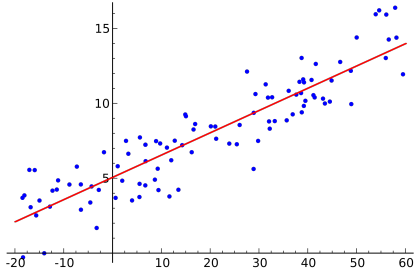
$$\hat{y}(\mathbf{w}, x) = \sum_{j=1}^{M} w_j x_j + w_0$$



1. Linear Regression

Fitting a linear model to observations $\{y(x_1), \dots, y(x_N)\}$ by

1. Define model $\hat{y}(\mathbf{w}, x) = \sum_{j=1}^{M} w_j x_j + w_0$



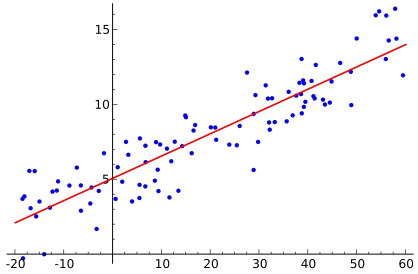
1. Linear Regression

Fitting a linear model to observations $\{y(x_1), \dots, y(x_N)\}$ by

1. Define model $\hat{y}(\mathbf{w}, x) = \sum_{j=1}^{M} w_j x_j + w_0$

2. Pick loss function, e.g. MSE

$$L(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \left(\hat{y}\left(x_n, \mathbf{w}\right) - y_o(x_n)\right)^2$$



1. Linear Regression

Fitting a linear model to observations $\{y(x_1), \dots, y(x_N)\}$ by

1. Define model $\quad \hat{y}(\mathbf{w}, x) = \sum_{j=1}^{M} w_j x_j + w_0$

2. Pick loss function, e.g. MSE

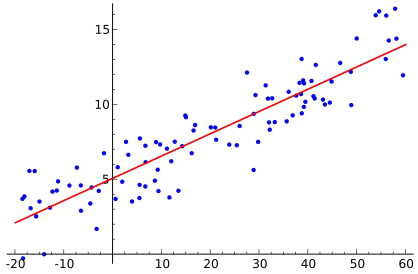
$$L(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \left(\hat{y}\left(x_n, \mathbf{w}\right) - y_o(x_n)\right)^2$$

3. Minimize loss function wrt. to parameters $\mathbf{w}$,

$\arg\min_{\mathbf{w}} L(\mathbf{w})$.

For linear regression an analytic solution exists:

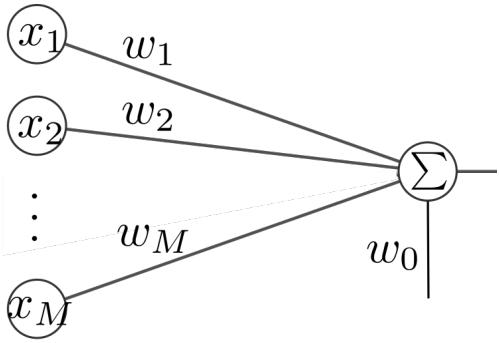
$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial \hat{y}(x,\mathbf{w})} \frac{\partial \hat{y}(x,\mathbf{w})}{\partial \mathbf{w}} \stackrel{!}{=} 0$$



1. Linear Regression

Linear regression can be illustrated as a graph:

$$\hat{y}(\mathbf{w}, x) = \sum_{j=1}^{M} w_j x_j + w_0$$

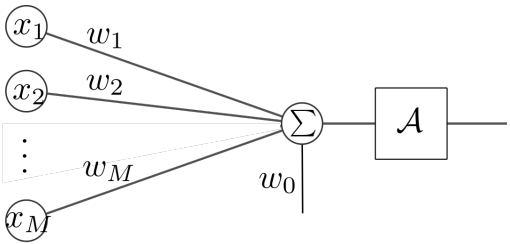


2. Neural Perceptron

Building block of Fully Connected Neural Networks (FCNN)

- Instead of linear combination add non-linear function

$$y = \mathcal{A}\left(\sum_j x_j \cdot w_j + w_0\right)$$



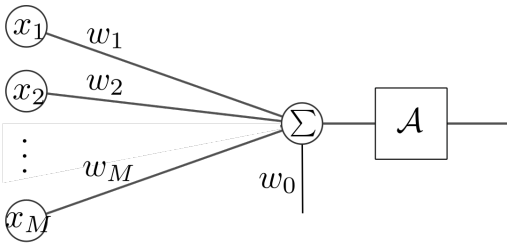
[1] P. Koehn, Introduction to Neural Networks (2022)

2. Neural Perceptron

Building block of Fully Connected Neural Networks (FCNN)

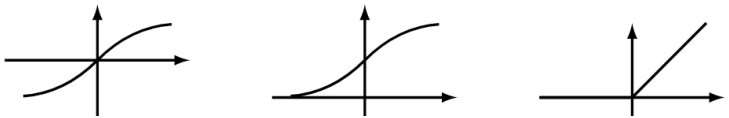
- Instead of linear combination add non-linear function

$$y = \mathcal{A}\left(\sum_j x_j \cdot w_j + w_0\right)$$



- ***f*** called the activation function
- Common activation functions: [1]

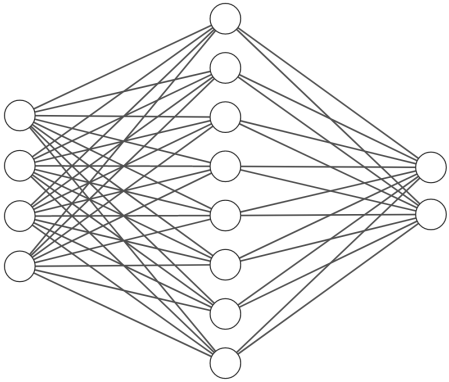
$\tanh(x)$ $\quad$ $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$ $\quad$ $\text{relu}(x) = \max(0,x)$



[1] P. Koehn, Introduction to Neural Networks (2022)

3. Multilayer Perceptron

Combining perceptrons into Neural Network.



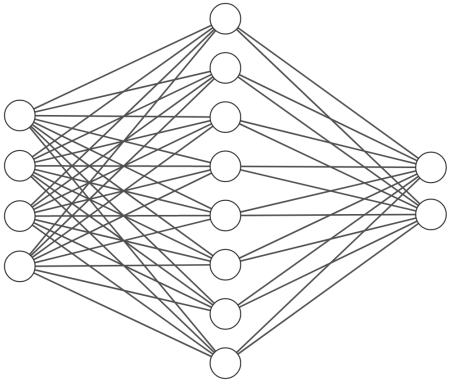
[a] http://alexlenail.me/NN-SVG/

3. Multilayer Perceptron

Combining perceptrons into Neural Network.

Gain by depths:

- Each layer is a processing step
- Multiple processing step allows to approx any function
- Metaphor: NN and computing



[a] http://alexlenail.me/NN-SVG/

4. Training a Neural Network

1. Define NN $\hat{y} = f(\mathbf{w}, x)$
2. Pick loss function, e.g. MSE $L(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} (\hat{y}(x_n, \mathbf{w}) - y_o(x_n))^2$
3. Minimize loss function wrt. to parameters $\mathbf{w}$, $\arg\min_{\mathbf{w}} L(\mathbf{w})$

How do we adjust the weights without analytic solution?

- Gradient descent
- Back-propagation

4. Training a Neural Network

Gradient descent and back-propagation

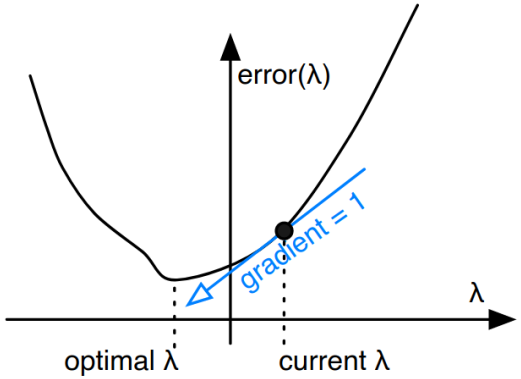
Gradient descent:

- Error is a function of the weights

$$L(\mathbf{w}) \propto \left|\hat{y}\left(x_n, \mathbf{w}\right) - y_o(x_n)\right|$$

- Compute gradients to move towards error minimum

$$\mathbf{w_{i+1}} = \mathbf{w_i} - \alpha \cdot \frac{\partial L}{\partial \mathbf{w_i}}$$



[a] P. Koehn, Introduction to Neural Networks (2022)

4. Training a Neural Network

Gradient descent and back-propagation

Gradient descent:

- Error is a function of the weights

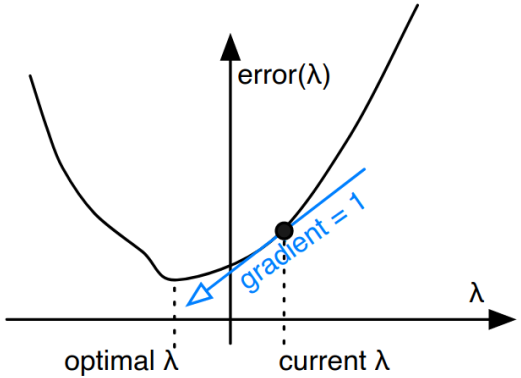
$$L(\mathbf{w}) \propto \left|\hat{y}\left(x_n, \mathbf{w}\right) - y_o(x_n)\right|$$

- Compute gradients to move towards error minimum

$$\mathbf{w_{i+1}} = \mathbf{w_i} - \alpha \cdot \frac{\partial L}{\partial \mathbf{w_i}}$$

Back-propagation:

- Adjust weights of last layer first
- Propagate error back to each previous layer
- Adjust weights of hidden layers



[a] P. Koehn, Introduction to Neural Networks (2022)

Summary

- Neural perceptron: Linear model with non-linear function
- MLPs are simply stacked perceptrons
- Theoretically: MLPs can approximate any function

Part II: Jupyter Notebook

python™

https://github.com/jakob-schloer/tutorials

Back-propagation

Reminder:

$$y(h, \mathbf{w}) = \mathcal{A}(\underbrace{\sum_j h_j \cdot w_j}_{s}), \qquad L(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} (y(x_n, \mathbf{w}) - y_o(x_n))^2$$

1. Derivative of error of last layer with regard to one weight w_k

   $$\frac{dL}{dw_k} = \frac{dL}{dy} \frac{dy}{ds} \frac{ds}{dw_k} = (y - y_o) y' h_k$$

2. Hidden layer error

   $$\delta = (y - y_o) y'$$

3. Update weights $\quad w_k^{(i+1)} = w_k^{(i)} - \alpha \, \delta \, h_k$