Deep Learning

Tuur Vanhoutte

September 21, 2021

Contents

1 Introduction to neural networks

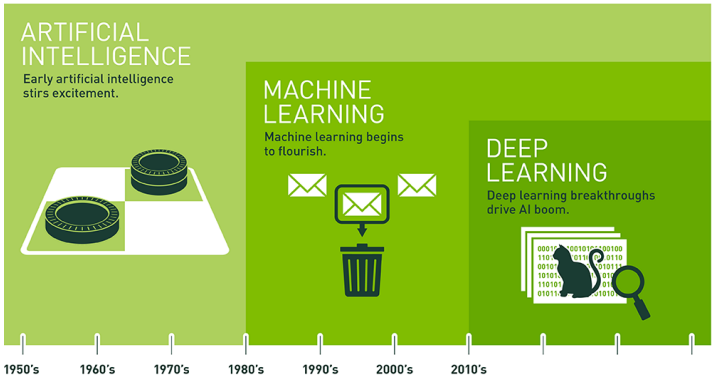


Figure 1: Timeline of AI

1.1 Why deep learning?

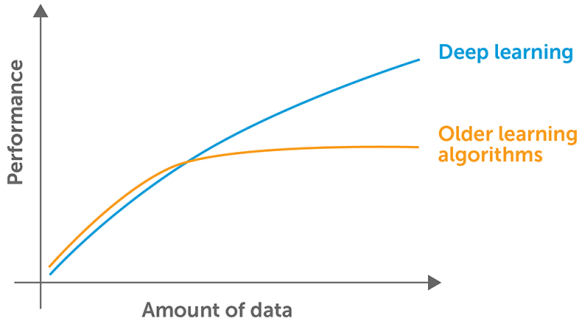


Figure 2: Performance vs amount of data

- The more data you have, the better the model
- In reality, older algorithms will plateau after a certain point
- With deep learning, this is much less of a problem
- However, neural networks are much more sensitive to overfitting: they can memorize the entire dataset
- For smaller datasets, it's often better to use traditional learning algorithms

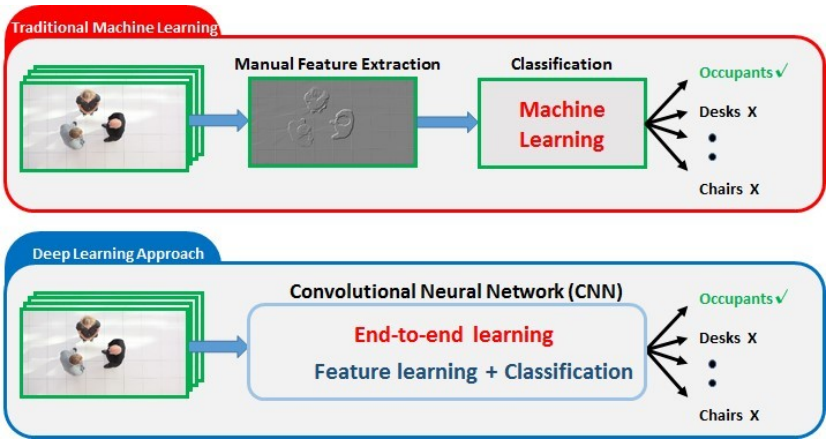
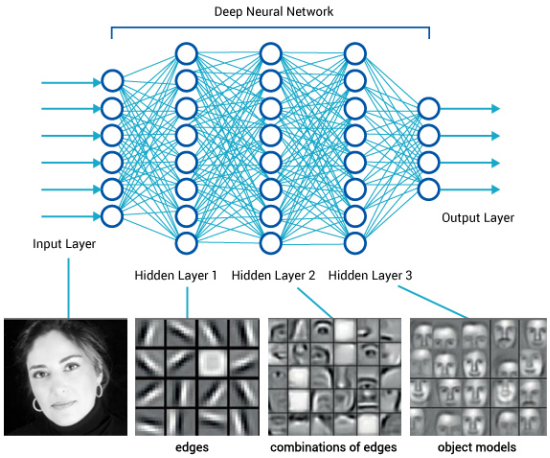


Figure 3: The difference in approach between traditional ML and Deep Learning

- With traditional ML, you first need to extract the features manually
- In the past, you had to manually mark facial features for facial recognition algorithms (eyes, mouth, ears, . . .)
- Deep learning is great for problems with non-structured data
  - Structured data = rows of data (like an Excel-sheet)
  - Non-structured = images, video, audio

Definition 1.1 *End-to-end learning is a deep learning technique where we run all raw data through a neural network. The neural network will do its own feature extraction (feature learning), and then a classifier will output predictions.*

1.2 The neural network



- The neurons are in one of three layers:
  - Input layer

  - Hidden layers
  - Output layer
- Every neuron is connected to every neuron of the next layer
- Every connection has a weight
- Deep Neural network $\Leftrightarrow$ multiple hidden layers

1.3 History of deep learning

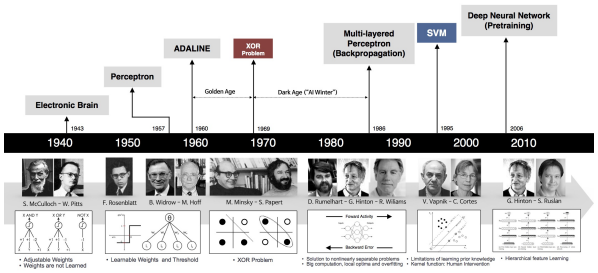


Figure 4: `https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html`

1.4 Deep learning architectures

You can change many things about the components of a neural network. Here are some examples of popular architectures. The most important ones are in **bold**:

- **Convolutional Neural Networks (CNN)**
- Capsule Network (CapsNet)
- Restricted Boltzmann Machine (RBM)
- **Autoencoder (AE)**
- Deep Belief Nets (DBN)
- **Recurrent Neural (Tensor) Network (RNTN)**
- **Long Short Term Memory (LSTM)**
- **Gated Recurrent Unit nets (GRU)**
- **Generatative Adversarial Nets (GAN)**
- ...

1.5 Biological model

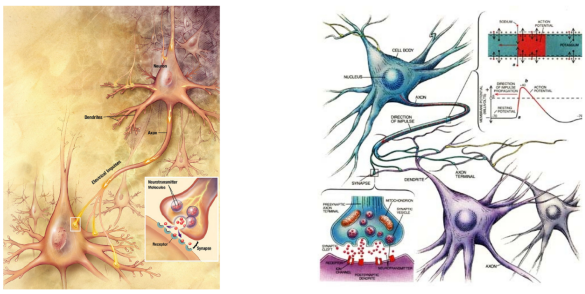


Figure 5: Anatomy of a biological neuron

- Neural networks are inspired by our brains
- A neuron has multiple connections that can send signals to other neurons
- A neuron can send a weak or strong signal

1.6 The artificial neuron

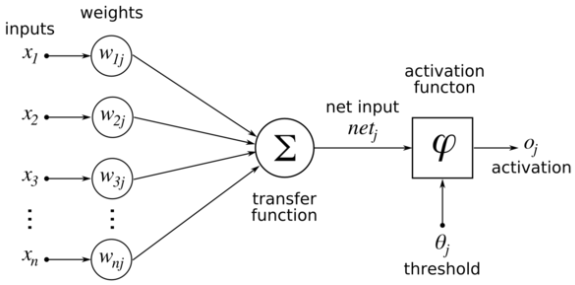


Figure 6: Conceptual model

- A neuron can receive multiple inputs
- Every input has a specified weight (= weak or strong signal)
- These inputs are multiplied by the weights, to then be used in a transfer function
- The transfer function is a mathematical formula that has a number as output
- That number is used as an input by an activation function. This functin will determine the final output
- `https://en.wikipedia.org/wiki/Activation_function`

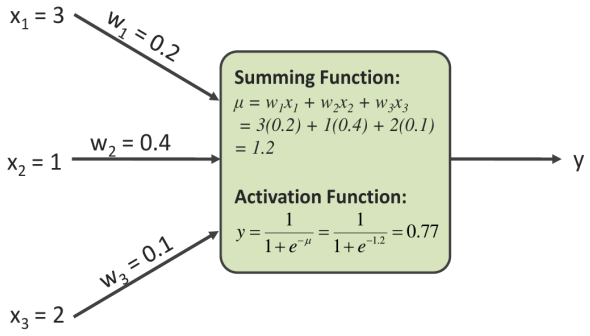


Figure 7: Example of a neuron, with a sigmoid function as activation function

2 Logistic Regression

Definition 2.1 *Logistic regression is used to determine the probability of a certain sample belonging to one of two classes.*

The name is a bit misleading: Logistic Regression is a classification technique, not regression. The reason for this name is because regression is used to calculate the classification.

2.1 The model

Because the output is a probability, we're looking for a function h_w so the model $h_w(x)$:

$$0 \leq h_w(x) \leq 1$$

- $h_w(x)$ = estimated probability that $y = 1$ for input x
- Example: $h_w(x) = 0.80$
  - The model is 80% sure that the sample belongs to class 1
  - If you use a threshold of 50%, the model will output 1
  - If the prediction is less than that threshold of 50%, the model will output 0

2.1.1 The logistic function

$$h_w(x) = \frac{1}{1 + e^{-w^\tau x}} \tag{1}$$

- With $e =$ Euler's number = ≈ 2.718
- This is a sigmoid function: the basic form for this is $\frac{1}{1+e^{-z}}$
- In our logistic function, z equals $w^\tau x$
  - This is equal to the solution of a linear regression function
  - With $x =$ the sample values
  - With $w =$ the weights that correspond to those values

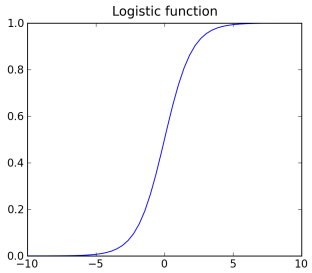


Figure 8: The logistic functie is evidently a sigmoid function or ‘S-function’

- $y = 1$ als $h_w(x) \geq 0.5 \Rightarrow w^\tau x \geq 0$
- $y = 0$ als $h_w(x) < 0.5 \Rightarrow w^\tau x < 0$

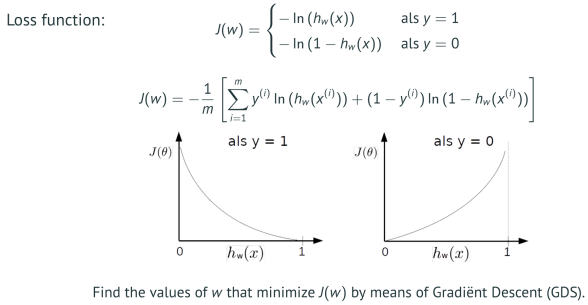


Figure 9: Minizing the binary cross-Entropy loss function

2.2 Logistic regression as a neuron

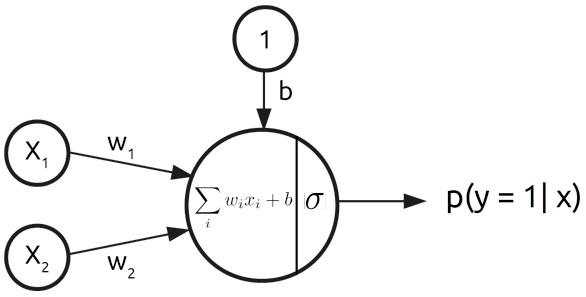


Figure 10: Logistic regression as a neuron

TODO: slides 19 - 25

3 Properties of a neural network

3.1 Feedforward neural networks

3.1.1 Properties

- 1 or more (hidden) layers.
- Each neuron is in a layer is connected to every neuron in the next layer.
- Signals travel from input to output through the hidden layers. (feedforward)
- The neural network maps inputs to desired outputs.

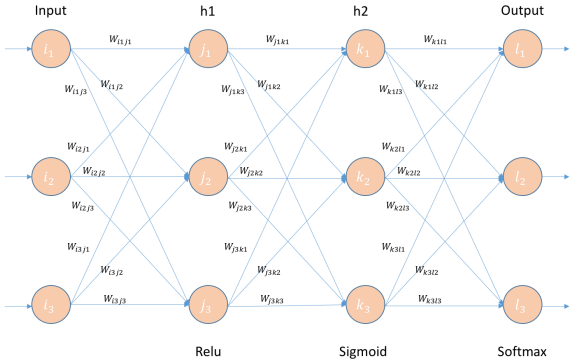


Figure 11: Feedforward neural network architecture

3.1.2 MNIST example

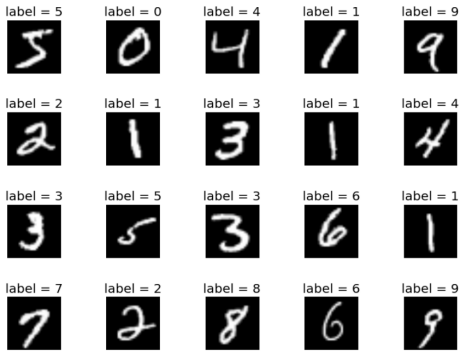


Figure 12: MNIST: dataset with labeled, handdrawn numbers

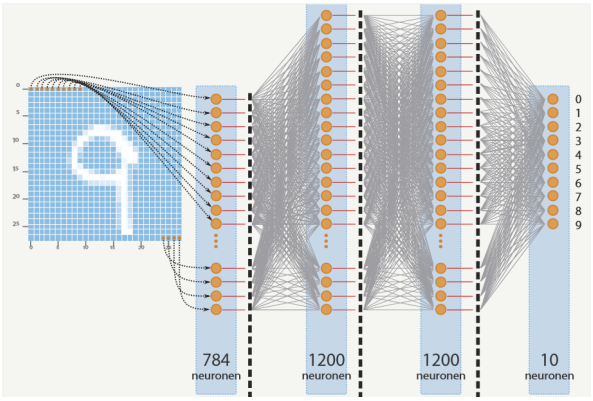


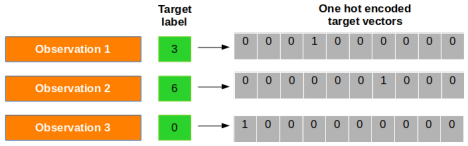
Figure 13: Simplified example of a neural network

- The numbers are drawn on a 28x28 grid (=784 input neurons)
- Every output (0-9) has a certain chance
- That chance determines how certain the model is that the given input is that number
- The chance is calculated using **backpropagation** (see later)

3.1.3 One-hot encoding

Definition 3.1 *One-hot encoding is a technique to transform categorical features or targets to a more suitable format. In a one-hot encoded target vector, the index of 1 represents the target label. All other values are 0.*

This is also used in neural networks.




```
# one-hot encoding class labels

from keras.utils import np_utils

y_train[:10]
```

```
array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4], dtype=uint8)
```

```
y_train_OneHotEncoding = np_utils.to_categorical(y_train)
y_train_OneHotEncoding[:10]
```

```
array([[ 0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.],
       [ 0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.]])
```

```
from sklearn import preprocessing
print(y_train)

lb = preprocessing.LabelBinarizer()
lb.fit(y_train)
y_train = lb.transform(y_train)

print(y_train)
```

```
[5, 0, 4, 1, 9, 2, 1, 3, 1, 4]
[[0 0 0 0 0 1 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1]
 [0 0 1 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]]
```

Figure 14

3.2 Multiclass classification

How to classify with more than 2 classes?

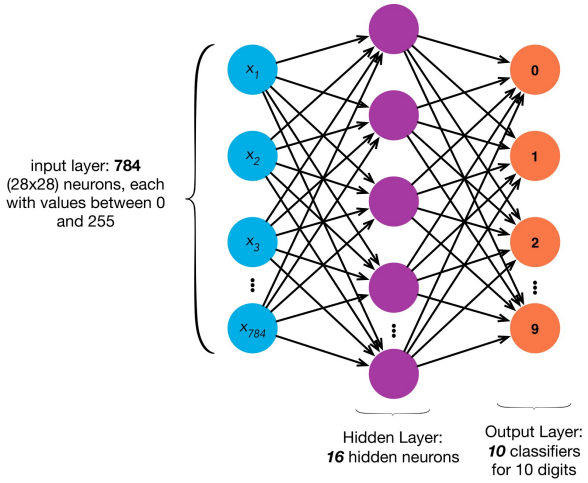


Figure 15

- Sigmoid functions only allow for binary classification
- We need a different function for multiclass classification with K classes
- $\Rightarrow$ **Softmax function**

$$P(y = 1 \mid x) = \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{x^T w_k}}$$

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

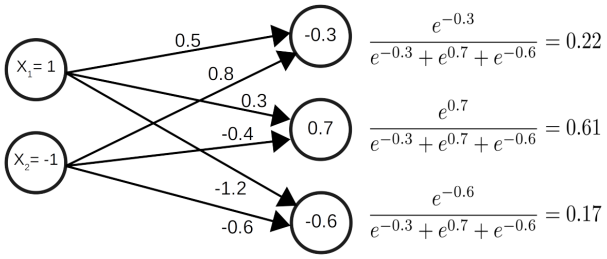


Figure 16: Softmax function

TODO: more info about softmax

$$x = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \text{ en } W = \begin{bmatrix} 0.5 & 0.3 & -1.2 \\ 0.8 & -0.4 & -0.6 \end{bmatrix}$$

$$b = 0$$

$$P(y = 1 \mid x) = \text{softmax}\left(W^T x + b\right)$$

Figure 17: Output computation in vector notation

3.2.1 Backpropagation

Definition 3.2 *Backpropagation is the most popular technique to train neural networks. It works by calculating an error (=loss) for every prediction, and to adjust the weights based on that error.*

The goal is to decrease the error as much as possible.

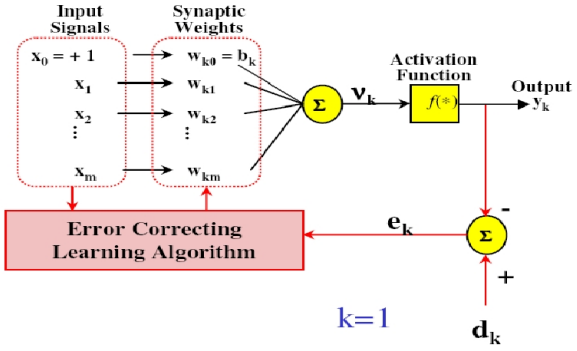


Figure 18

- η is the learning rate
- $0 < m \cdot \eta < 1$ with m the number of inputs
- Error: $e_k(n) = d_k(n) - y_k(n)$
  - Minimize the error $e_k(n)$
  - $\epsilon(n) = \frac{1}{2} \sum_k e_k^2(n)$
  - $\Delta W_{kj}(n) = \eta \cdot e_k(n) \cdot x_j(n)$
  - Adjust the weight: $\Delta W_{kj}(n+1) = W_{kj}(n) + \Delta W_{kj}(n)$

`https://becominghuman.ai/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c`

Using the MNIST example:

- In the output layer, the neural network assigns a probability to every number
- The prediction is made based on the highest chance.
- Suppose a certain number received too high a probability, and the true number too low a probability:
  - The neural network will calculate the differences (=error)
  - Next time, the neural network will try to keep the error as low as possible
  - It does this by repeatedly adjusting the weights
  - This process happens through all layers, from back to front $\Rightarrow$ **backpropagation**

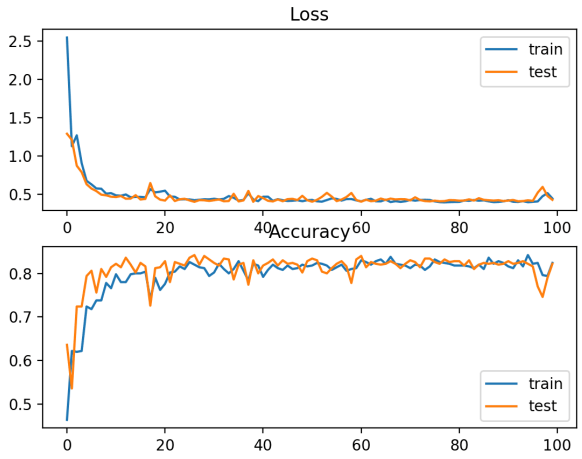


Figure 19: The error function (=loss) in functie of the amount of training epochs

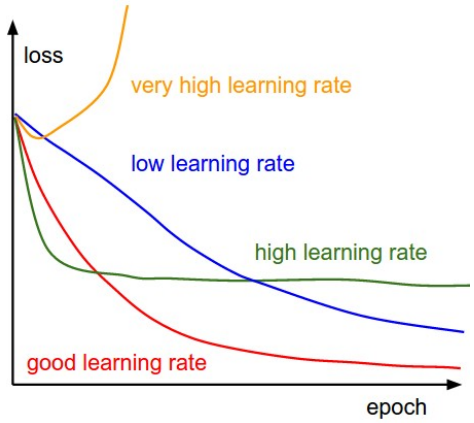
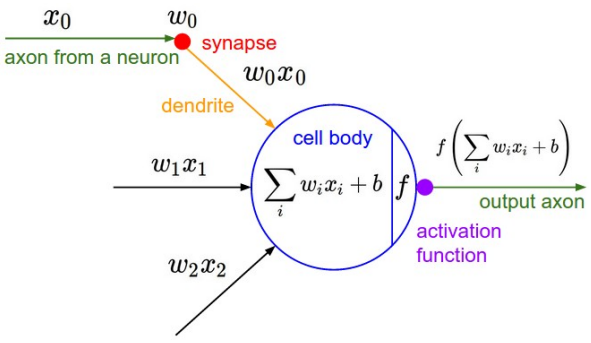


Figure 20: Impact of the learning rate to the loss. During training, the learning rate keeps changing.

3.3 Activation function



Definition 3.3 *The activation function of a neuron defines the output of that neuron using a mathematical function. The inputs are multiplied by their weight, and are then summed up by the transfer function. The output of that transfer function is used as an input by the activation function.*

There are various functions that each have a specific output range (possible y-values):

- Step function: 0 or 1
- Linear function: $-\infty$ to $+\infty$
- Sigmoid function: 0 to 1
- Hyperbolic tangent (tanh): -1 to 1
- ReLu: 0 to $+\infty$
- Leaky ReLu: $-\infty$ to $+\infty$