Deep Learning

Tuur Vanhoutte

September 28, 2021

Contents

1 Introduction to neural networks

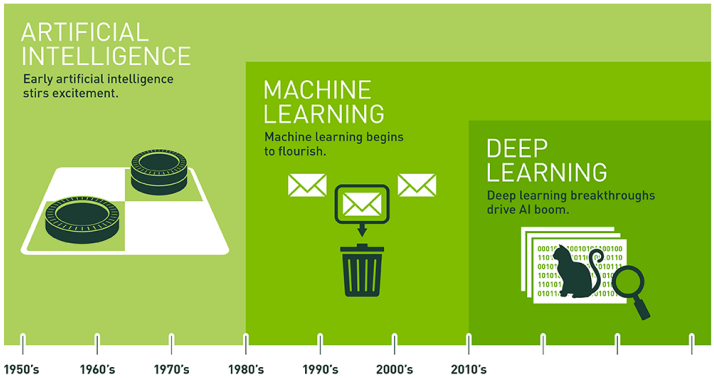


Figure 1: Timeline of AI

1.1 Why deep learning?

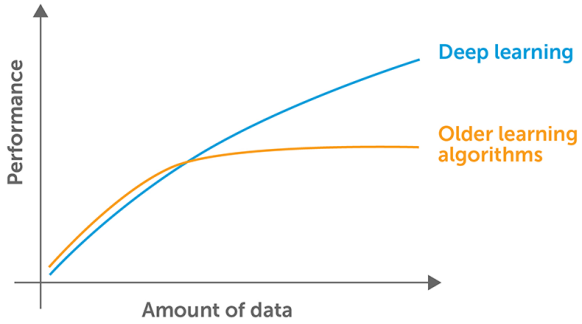


Figure 2: Performance vs amount of data

- 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

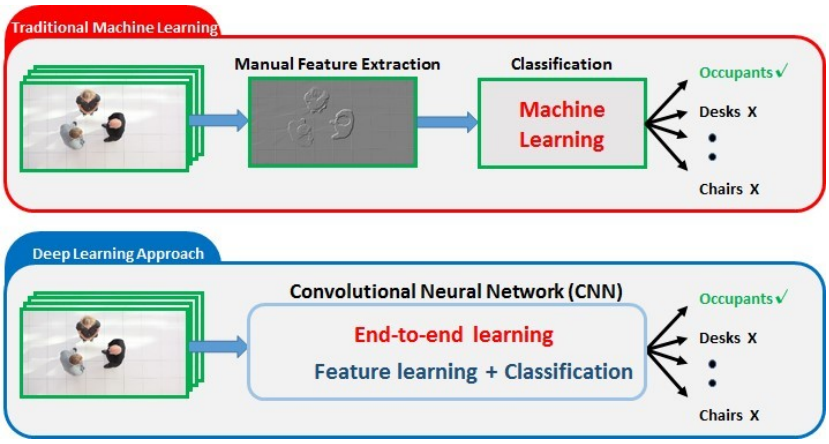
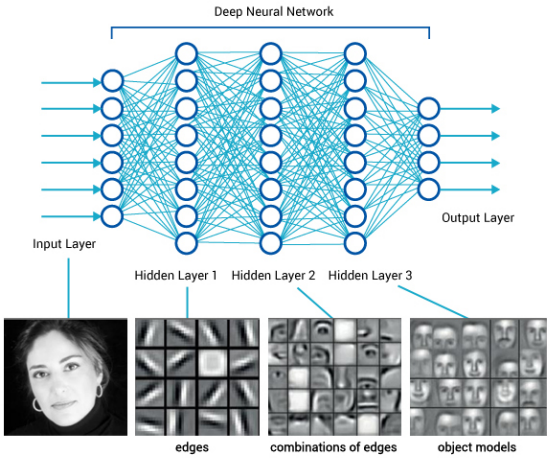


Figure 3: The difference in approach between traditional ML and Deep Learning

- 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

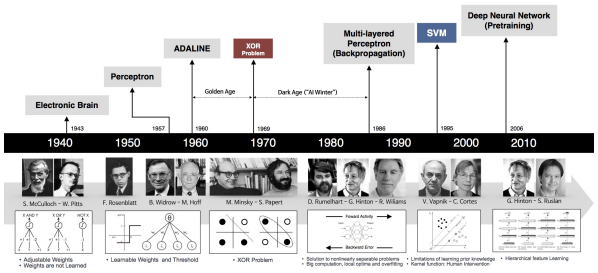


Figure 4: `https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html`

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

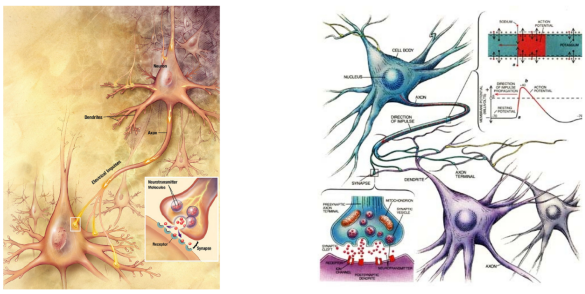


Figure 5: Anatomy of a biological neuron

- 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

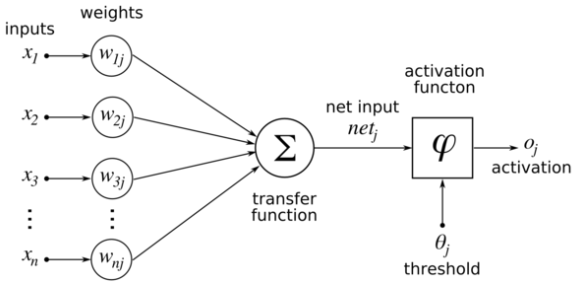


Figure 6: Conceptual model

- 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`

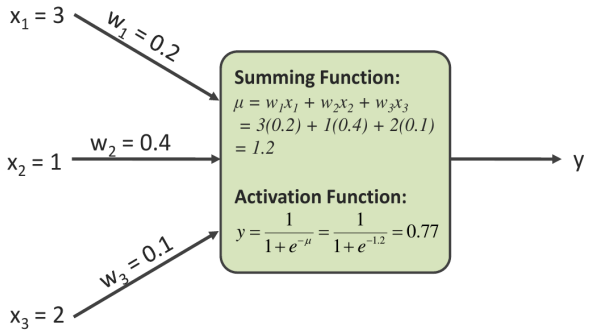


Figure 7: Example of a neuron, with a sigmoid function as activation function

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

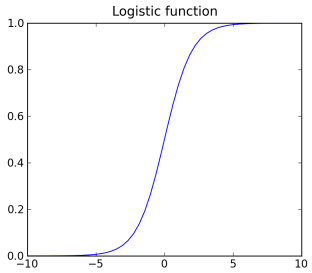


Figure 8: The logistic functie is evidently a sigmoid function or ‘S-function’

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

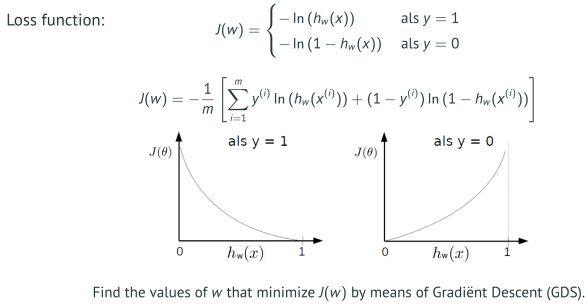


Figure 9: Minizing the binary cross-Entropy loss function

2.2 Logistic regression as a neuron

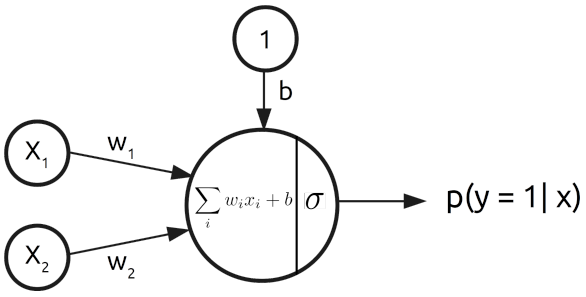
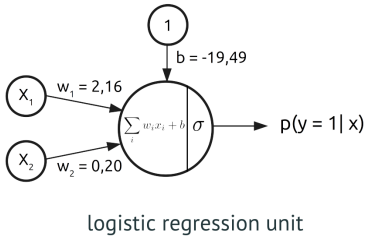


Figure 10: Logistic regression as a neuron

2.2.1 Example



Classify a person with job satisfaction $x_1 = 4$ and age $x_2 = 30$

$z = 2,16 \times 4 + 0,20 \times 30 - 19,49 = -4,85$

$p(y = 1 \mid x) = \frac{1}{1 + e^{+4,85}} = 0,0078 = 0,78\%$

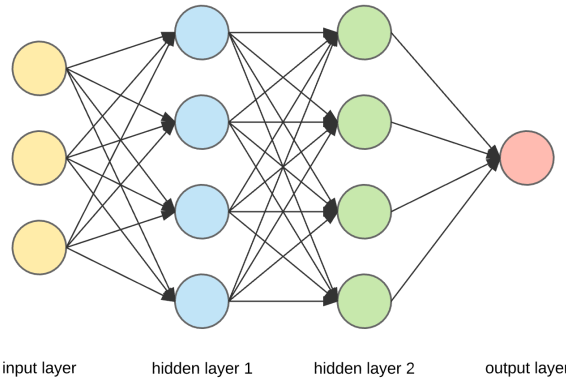


Figure 11: A neural network can be modeled as a network of logistic regression units.

2.3 Logistic regression vs neural networks

In case of non-linearly separable data:

- Use higher order features
- Neural networks make new representations of existing features.

2.3.1 Example

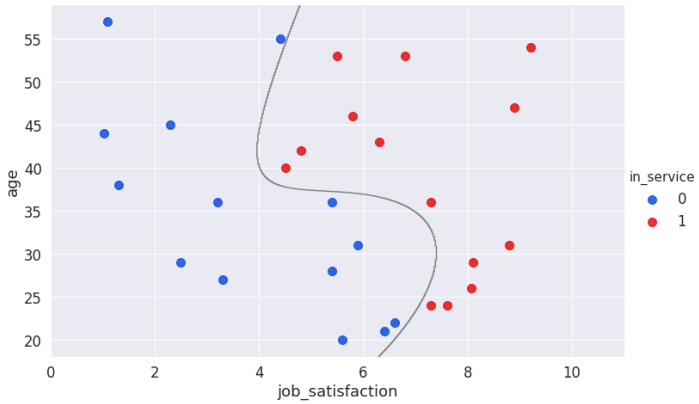


Figure 12: Decision boundary using 3rd order features with logistic regression

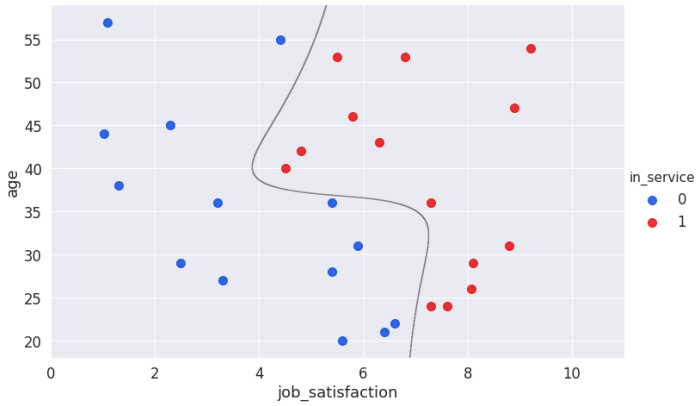
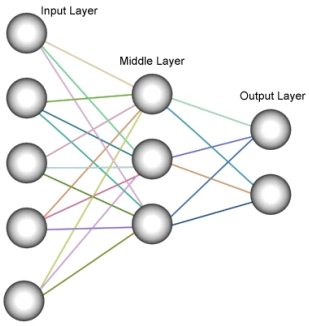


Figure 13: Decision boundary of a neural network



3 Basics of a neural network

3.1 Properties of a neural network

- Network architecture
- Learning algorithm
- Activation functions

You want the neural network to adapt and generalize to the data.

3.1.1 Common network architectures

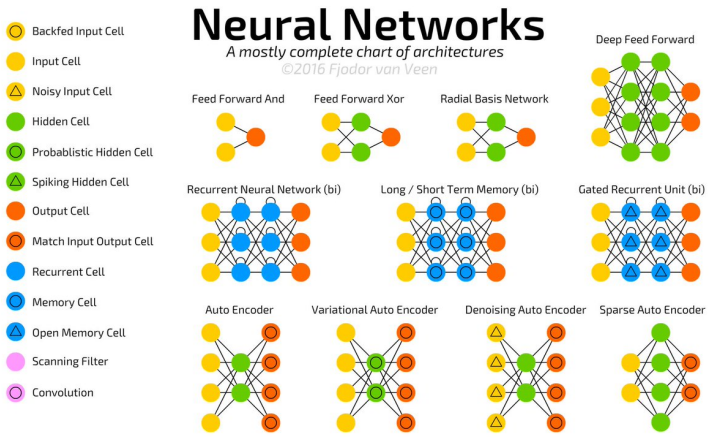


Figure 14: `http://www.asimovinstitute.org/neural-network-zoo/`

3.2 Feedforward neural networks

3.2.1 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.

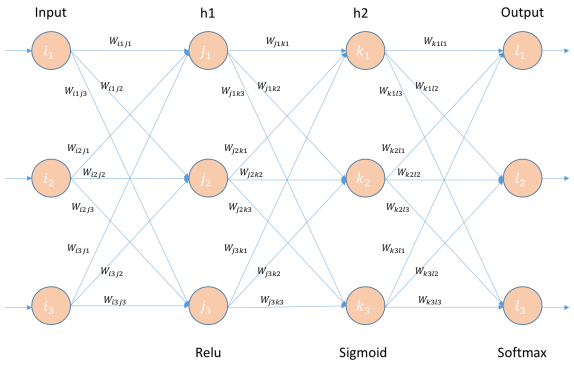


Figure 15: Feedforward neural network architecture

3.2.2 MNIST example

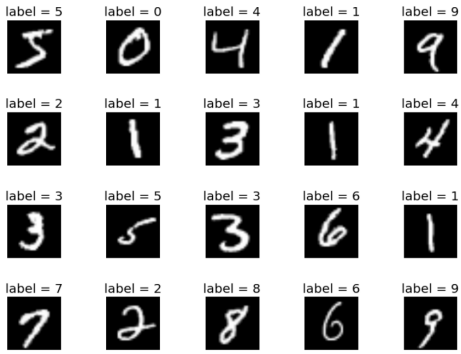


Figure 16: MNIST: dataset with labeled, handdrawn numbers

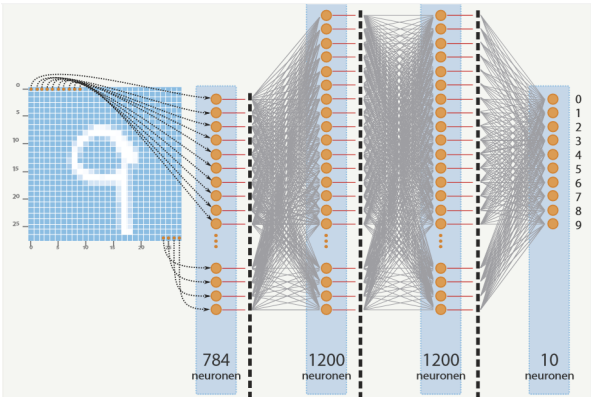


Figure 17: Simplified example of a neural network

- 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)

3.2.3 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]
 [1 0 0 0 0 0 0]
 [0 0 0 0 1 0 0]
 [0 1 0 0 0 0 0]
 [0 0 0 0 0 0 1]
 [0 0 1 0 0 0 0]
 [0 1 0 0 0 0 0]
 [0 0 0 1 0 0 0]
 [0 1 0 0 0 0 0]
 [0 0 0 0 1 0 0]]
```

Figure 18

3.3 Multiclass classification

How to classify with more than 2 classes?

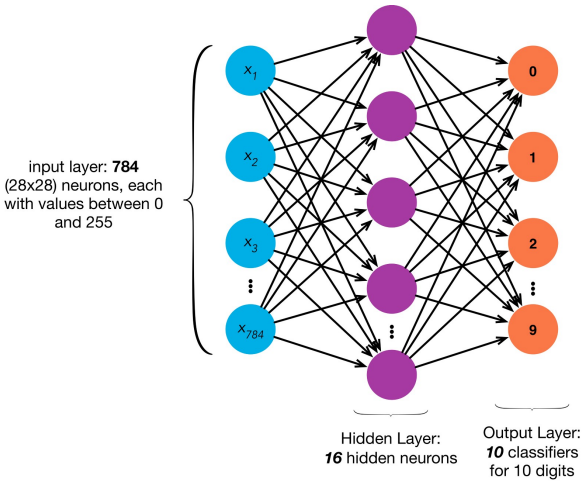


Figure 19

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

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

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

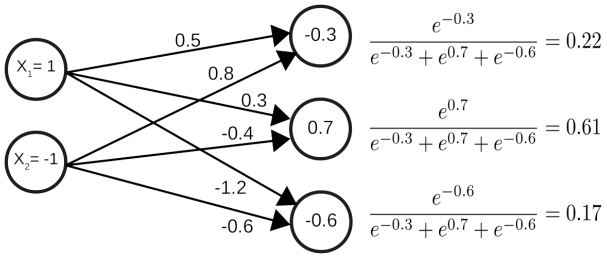


Figure 20: Softmax function

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 21: Output computation in vector notation

3.3.1 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.

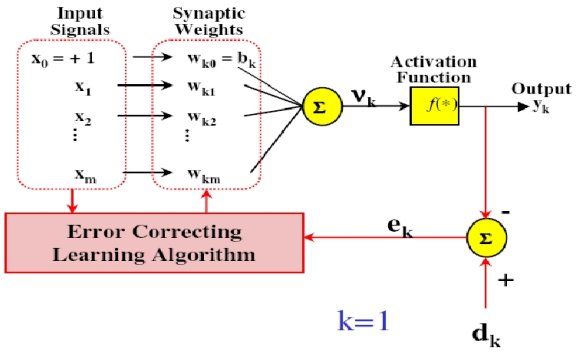


Figure 22

- 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 ⇒ **backpropagation**

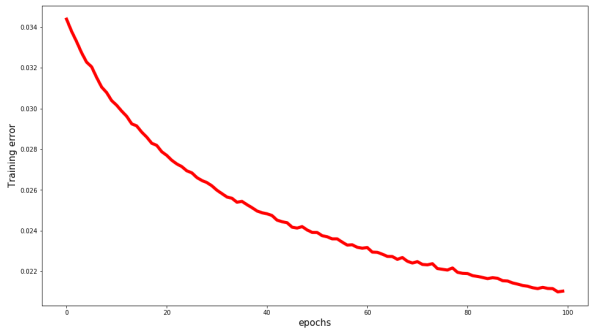


Figure 23: The error function (=loss) in function of the amount of training epochs

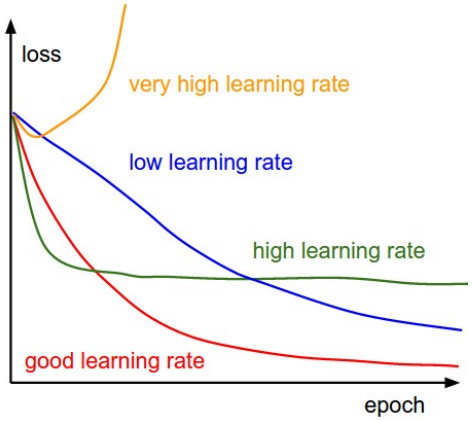
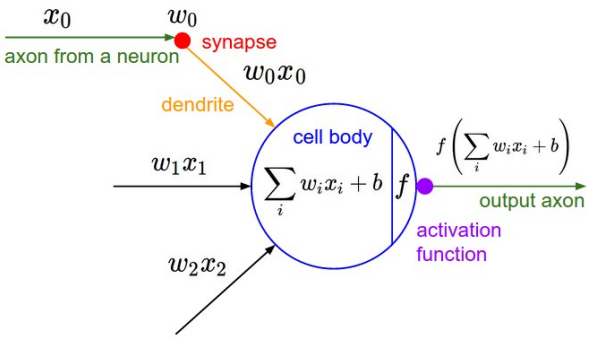


Figure 24: Impact of the learning rate on the loss. During training, the learning rate keeps changing.

3.4 Activation function

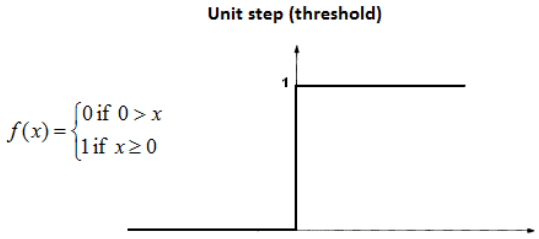


Definition 3.3 *The activation function of a neuron defines the output of that neuron using a mathematical function. The inputs are multiplied by their weight, and are then summed up by the transfer function. The output of that transfer function is used as an input by the activation function.*

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 $+\infty$

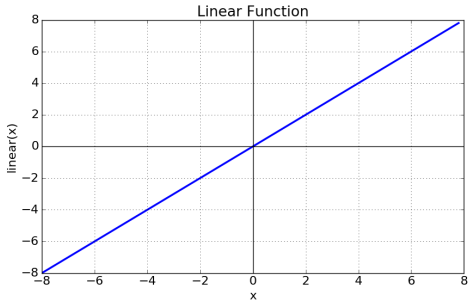
3.4.1 Step function



- Output is 1 when the value > 0
- Output is 0 when the value < 0
- Downsides:
  - Binary: can only output yes or no (100% of 0%)
  - Problems with multiple classes. What if multiple classes output more than 0?
  - Hard to train: backpropagation does not work (the derivative is 0)
- Use:

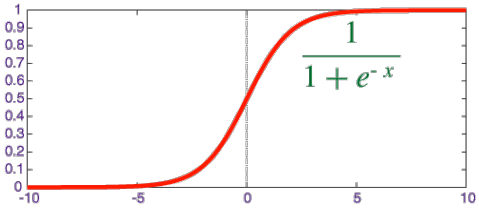
- Do not use the step function

3.4.2 Linear function (Adaline)



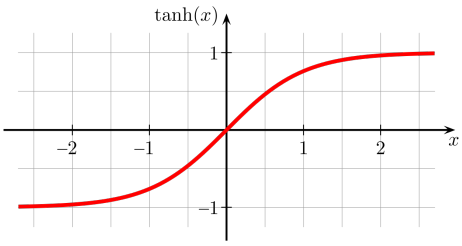
- Output is proportional to the input
- Downsides:
  - It doesn't matter how many layers you use, in the end, the output of stacked lineair layers
  - The derivative is constant: does not have a relation to the input
- Use:
  - Input layer
  - Output layer with regressie

3.4.3 Sigmoid function



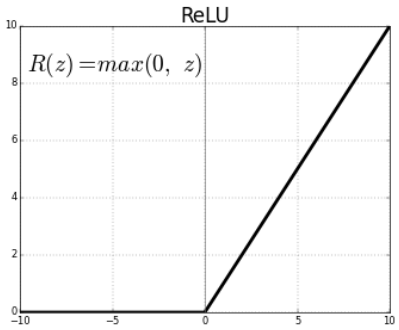
- Non-linear
- The output will always be between 0 and +1
- Downsides:
  - Vanishing gradient descent has a higher performance than adaptive techniques problem: problematic for neural networks with many hidden layers
- Use:
  - Is not used that much anymore for hidden layers
  - Sometimes used for output layer with classification problems

3.4.4 Hyperbolic tangent (tanh)



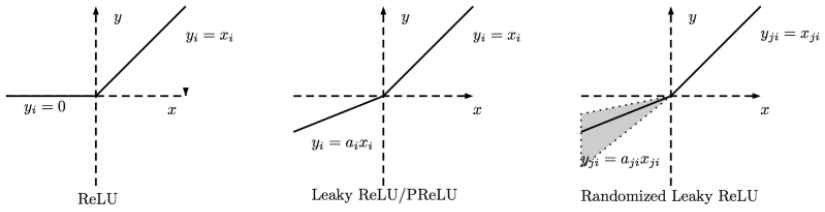
- Non-linear
- The output is always between -1 and +1
- Centered around 0
- Downsides:
  - Vanishing gradient problem is still present
- Use:
  - Is not used that much anymore, except for specific architectures

3.4.5 Rectified Linear Unit (ReLu)



- Non-linear
- Every function can be approached using a combination of ReLu functions
- Computationally efficient
- Downsides:
  - Sparse activation: many activations become 0
  - $\Rightarrow$ 'Dead' neurons cannot be activated again
- Use:
  - For hidden layers

3.4.6 Leaky ReLu



- Variant on ReLu
- Neurons do not 'die': there is still a low gradient when $x < 0$
- Downsides:
  - More parameters to train
- Gebruik:
  - For hidden layers

3.4.7 Conclusion

Hidden layers:

- First try ReLu
- Try Leaky ReLu
- Usually no Sigmoid or Tanh.

Output layer:

- Lineair for regression problems.
- Softmax / Sigmoid for classification problems

Softmax is a generalisation of Sigmoid: $\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$

3.5 Underfitting and overfitting

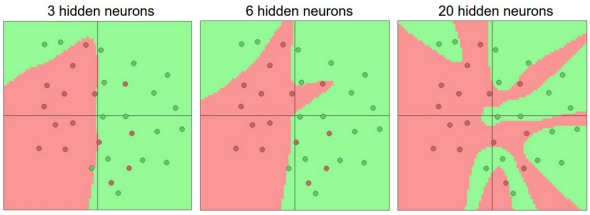
Definition 3.4 (Underfitting) *Underfitting occurs when an AI can not model the training data, and cannot generalize new data.*

- *The model is too 'simple'*
- *Model with high bias*
- $\Rightarrow$ *the score with training data and the score with test data are both low*

Definition 3.5 (Overfitting) *Overfitting occurs when an AI models the training data too well, and cannot generalize new data.*

- *The model is too 'complex'*
- *Random noise in the data gets picked up and remembered by the AI*
- *Model with a high variance*

- $\Rightarrow$ *the score with training data is higher than the score with test data*



- Too large neural network: overfitting
- Too small neural network: underfitting

TODO: L1 & L2 regularization

3.6 Regularisation

Definition 3.6 (Regularisation) *Method to find a good balance between underfitting and overfitting.*

We'll discuss 2 regularisation methods:

- Ridge regression (L2 regularisation)
- Lasso regression (L1 regularisation)

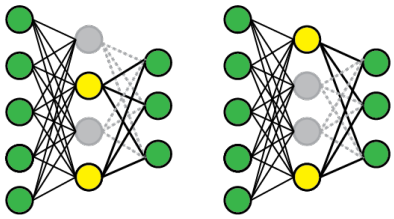
3.6.1 Regularisation with L2

TODO

3.6.2 Regularisation with L1

TODO

3.6.3 Dropout



Definition 3.7 *Dropout (or 'Dilution') is a regularisation technique to prevent overfitting in neural networks. This happens by disabling a certain percentage of neurons in the network. Other neurons must then take over from the disabled neurons.*

- Randomly switch off a certain percentage of neurons in a layer.
- Active neurons have to take over.
- Hidden layers are forced to learn the general underlying structure.
- Emulates an ensemble of neural networks.

- Can be responsible for the fact that the validation accuracy is higher than the training accuracy.
- Dropout only happens during training time. When making predictions, the full network is used.

`https://en.wikipedia.org/wiki/Dilution_(neural_networks)`

3.6.4 Noise injection

Definition 3.8 *Noise injection makes a neural network **more robust against small changes** in the weights*

- Without noise injection: train(X , Y)
- With noise injection: train(X+noise , Y)

Instead of adding noise to the input, add noise to the weights w:

$Y = f(X, W + noise), noise\ N(0, kleine\sigma)$

3.7 Gradient Descent modes

3.7.1 Full gradient descent

- Batch size = full training set
- Also called batch mode
- Not always suited for large data sets

3.7.2 Stochastic gradient descent

- Minimize the costfunction on a sample by sample basis
- Batch size = 1 (training) sample.
- In the long term the error will decrease.
- The learning curve can be very erratic.

3.7.3 Batch gradient descent (mini-batch)

- Batch size is larger than 1 and smaller than the size of the training set.
- Happy medium.
- Less erratic than with stochastic gradient descent.
- Also called mini-batch.

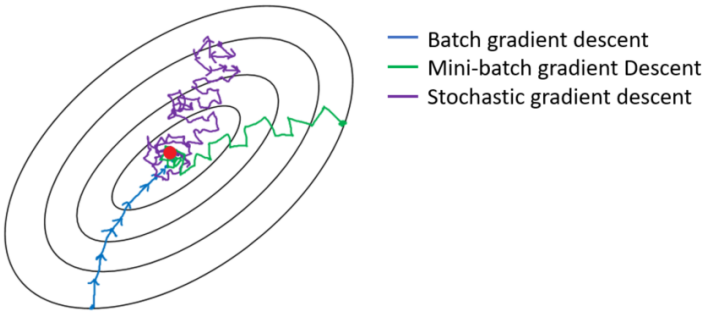


Figure 25: Path of the training loss

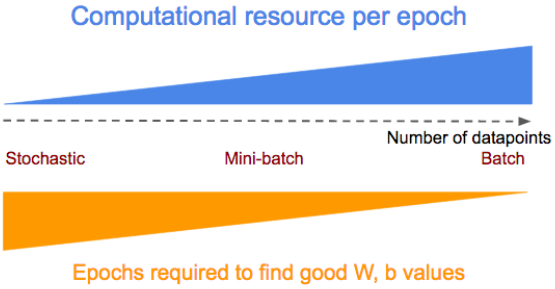


Figure 26: Resources

3.8 Momentum & adaptive learning rates

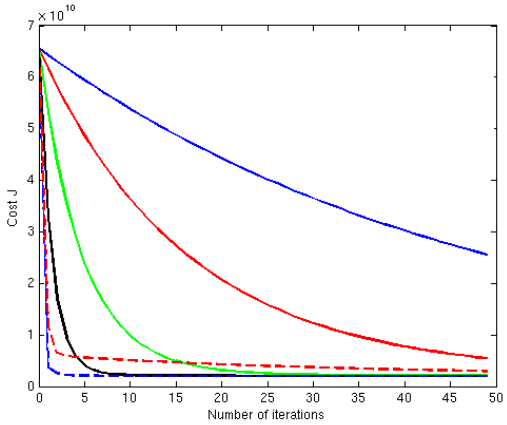


Figure 27: TODO

3.8.1 Goal

- speeding up gradient descent.

- Momentum is the most important factor
  - = The force that keeps an object moving or keeps an event developing after it has started.
  - Less erratic movements of the training loss.
  - Updating the weights: $\theta t = \theta t - 1 + \mu v t - 1 - \eta g t$

3.8.2 Nesterov momentum

Make a jump in the direction of the momentum, compute the gradient and adjust the trajectory.

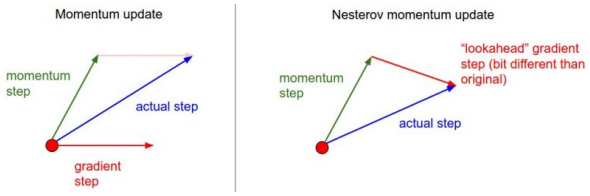


Figure 28: Nesterov momentum

3.8.3 Variable learning rates

1. Step Decay:
   - Decrease the learning rate every x iterations.
   - Example: every 100 iterations the learning is cut in half.
2. Exponential Decay:
   - $\eta(t) = A \cdot e^{-kt}$
3. 1/t Decay:
   - $\eta(t) = \frac{A}{kt+1}$
   - Slower than exponential decay
4. Babysitting method:
   - Train for a couple of epochs and look what's going on.
   - If necessary, change the learning rate.
   - **Warning:** can be stuck on a plateau

3.8.4 Adaptive learning rates

1. Adagrad:
   - The cost of each parameter (weight) is different: steep in one direction, flat in the other direction.
   - Adjust the learing rate of each individual parameter based on how much it has been adjusted in the past = cache.
   - $\theta(t) = \theta(t-1) - \eta \frac{\Delta J}{\sqrt{cache+e}}$
2. RMSprop:

- Adagrad is too agressive in lowering the learning rate.
- Decrease the cache with every update
- cache = decay x cache + (1 - decay) x gradient^2
- Decay is typically 0.99 or 0.999

3. Adam:
   - Most popular optimizer in deep learning
   - Combines RMSprop with momentum
   - TODO: formulas
   - **Remark:** experiments show that regular gradient descent has a higher performance than adaptive techniques

3.9 Weight initialization

- Observation: the deeper the better. More layers are better than more neurons per layer
- Problems of deep learning: vanishing gradients and exploding gradients

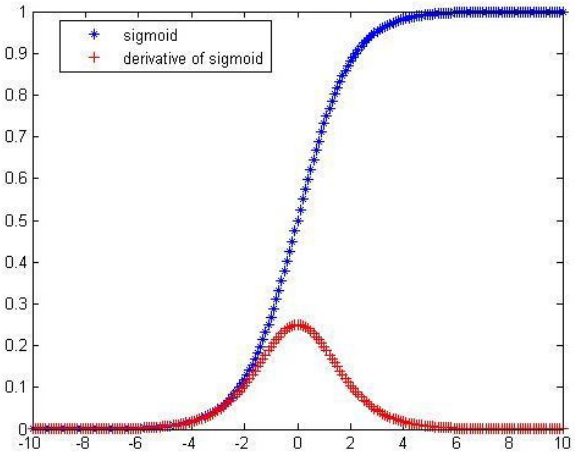


Figure 29: TODO

3.9.1 Solutions to the vanishing gradient and exploding gradient problem

- Greedy layer-wise unsupervised pre-training.
- ReLu.
- Good initialization of the weights. (random)

3.9.2 Types of initializations

1. Constant: leads to symmetry. Has to be avoided!
2. Random:
   - Uses the np.random functions

- Other: Glorot uniform, Lecun (uniform distribution)
- Bias can be initialized to 0

3.10 Batch normalization

- We like to normalize our data.
- A lot of activation functions have a limited active range.
- Makes sure that each layer learn a bit more independently from the other layers.
- Regularization effect: less prone to overfitting.
- Adds a small amount of noise to each layer. Less need for dropout.
- **Normalize each layer of the neural network**

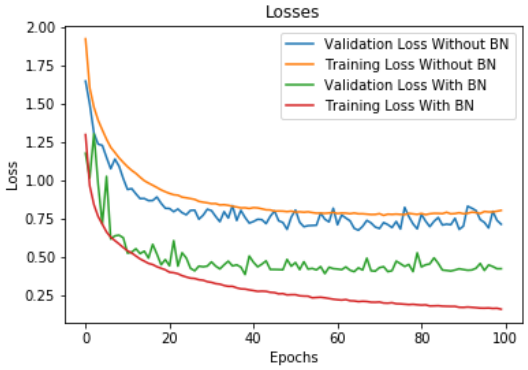


Figure 30: Training loss with & without batch normailzation

4 Keras

4.1 What is Keras

- Neural Network library written in Python.
- Built on top of Tensorflow and Theano.
- Easy to use.
- Modular.
- Makes it possible to build complex neural networks

4.2 The sequential model

- Stacking layers on top of each other.
- `https://keras.io/#getting-started-30-seconds-to-keras`

```
1 import tensorflow as tf
2 from tf.keras.models import Sequential
3 model = Sequential()
```

```
4
5  from tf.keras.layers import Dense, Activation
6  model.add(Dense(units=30, input_dim=10))
7  model.add(Activation('relu'))
8  model.add(Dense(units=5))
9  model.add(Activation('softmax'))
```

⇒ 10 inputs | 30 hidden layer ReLu units | 5 softmax outputs

4.2.1 Compiling, training and making predictions

```
1  model.compile(loss='categorical_crossentropy',
2                optimizer='sgd',
3                metrics=['accuracy'])
4
5  model.compile(loss=keras.losses.categorical_crossentropy,
6                optimizer = keras.optimizers.SGD(lr=0.01,momentum=0.9, nesterov=True)
7              )
8  model.fit(x_train, y_train, epochs=5, batch_size=32)
9  classes = model.predict(x_test, batch_size=128)
```

4.3 Parameters of the sequential model

4.3.1 Activation functions

`https://keras.io/activations/`

Available activation functions:

- softmax
- relu
- sigmoid
- tanh
- linear
- `https://keras.io/layers/advanced-activations/`

4.3.2 Learning rate optimizers

`https://keras.io/optimizers/`

- SGD + Nesterov (Stochastic Gradient Descent)
- RMSProp: more applicable to recurrent networks
- Adagrad
- Adam
- Adamax

4.3.3 Loss function

`https://keras.io/losses/#available-loss-functions`

- The loss function to be optimized during the training process.
- Error losses: (usually for regression)
  - mean_squared_error(y_true, y_pred)
  - mean_absolute_error(y_true, y_pred)
  - mean_absolute_percentage_error(y_true, y_pred)
  - mean_squared_logarithmic_error(y_true, y_pred)
- Hinge losses: (usually for Support Vector Machines (SVM))
  - squared_hinge(y_true, y_pred)
  - hinge(y_true, y_pred)
  - categorical_hinge(y_true, y_pred)
- Class losses: (usually for classification)
  - binary_crossentropy(y_true, y_pred)
    - for **binary classification or multilabel classification**: problems where a sample can belong to more than one class
    - With sigmoid
  - categorical_crossentropy(y_true, y_pred)
    - For **multiclass classification** with softmax where a data sample can only belong to one class.
    - With softmax

4.4 Metrics

Used for performance evaluation

- Regression metrics:
  - Mean Squared Error: mean_squared_error of mse
  - Mean Absolute Error: mean_absolute_error, mae
  - Mean Absolute Percentage Error: mean_absolute_percentage_error, mape
  - Cosine Proximity: cosine
- Classification metrics:
  - Binary Accuracy: binary_accuracy, acc
  - Categorical Accuracy: categorical_accuracy, acc
  - Top k Categorical Accuracy: top_k_categorical_accuracy
  - Cosine Proximity

4.5 Hyperparameter tuning

- number of neurons per layer
- dropout rate
- batchsize
- optimizer
- weight initialization
- learningrate
- activation function
- loss function
- weight decay
- ...

We often use Grid Search, Random search and Bayes Optimization to find the best parameters.

4.6 Imbalanced data

4.6.1 Problem of imbalanced data

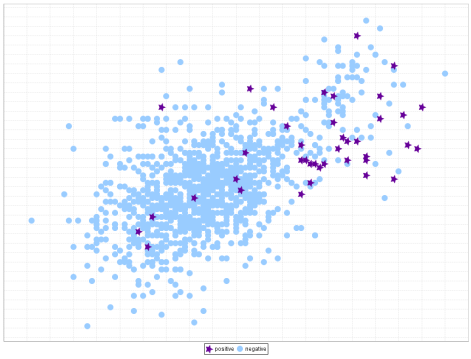


Figure 31: We have very few data points in the dataset of one class compared to the other

4.6.2 Solutions

- Collect more data
- Undersampling & Oversampling

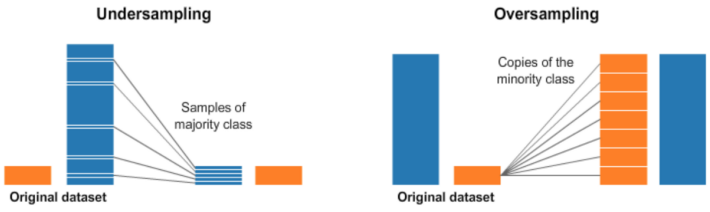


Figure 32: Undersampling & Oversampling

- Data augmentation