Inhaltsverzeichnis

1 Introduction **2**

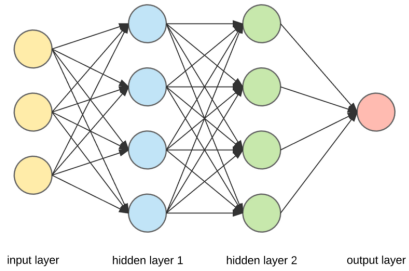
2 Artificial Neural Networks **2**

3 Autoencoders **5**

1 Introduction

2 Artificial Neural Networks

The study of artificial neural networks is motivated by their similarity to successfully working biological systems, which - in comparison to the overall system - consist of very simple but numerous nerve cells that work massively in parallel and (which is probably one of the most significant aspects) have the capability to learn.
There is no need to explicitly program a neural network. For instance, it can learn from training samples or by means of encouragement - with a carrot and a stick, so to speak.[1]



They consist of an input layer, multiple hidden layers, and an output layer. Every node in one layer is connected to every other node in the next layer. We make the network deeper by increasing the number of hidden layers.[2]

Technically a neural network consists of simple processing units, the neurons, and directed, weighted connections between those neurons. The strength of a connection (or the connecting weight) between two neurons i and j is referred to as $w_{i,j}$.
So the weights can be implemented in a square weight matrix W or, optionally, in a weight vector W. Data are transferred between neurons via connections with the connecting weight being either excitatory or inhibitory.

[1] D. Kriesel, A brief introduction to Neural Networks, http://www.dkriesel.com/media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf, p. 4

[2] https://medium.com/towards-data-science/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

For a neuron j the propagation function receives the outputs of the other neurons, which are connected to j, and transforms them in consideration of the connecting weights $w_{i,j}$ into the network input that can be further processed by the activation function. Here the weighted sum is very popular: The multiplication of the output of each neuron i by $w_{i,j}$, and the summation of the results.
Based on the model of nature every neuron is, to a certain extent, at all times active. The activation state a_j from a neuron j indicates the extent of the neuron's activity and results from the activation function.[3] Typically it can take any value between 0 and 1 or between -1 and 1.[4]
Another important parameter is the threshold value, which marks the position of the maximum gradient value of the activation function.[5].

Lets take a formal look at the activation function:

$$a_j^i = \sigma(\sum_k (w_{jk}^i * a_k^{i-1}) + b_j^i)$$

where
σ is the activation function
a_j^i represents the activation value of the j-th neuron in the i-th layer
w_{jk}^i is the weight from the k-th neuron in the i-1-th layer to the j-th neuron in the i-th layer
b_j^i is the bias of the jth neuron in the ith layer, where bias = threshold * -1

There is a couple of different activation functions. Each of it has its own strengths and weaknesses. When you know the function you are trying to approximate has certain characteristics, you can choose an activation function which will approximate the function faster leading to faster training process. Otherwise it may be usefull to try out to try out some.

Popular activation functions are:

- the sigmoid or logit function[6]:

[3]D. Kriesel, A brief introduction to Neural Networks, p. 33 ff.
[4]http://neuralnetworksanddeeplearning.com/chap1.html
[5]D. Kriesel, A brief introduction to Neural Networks, p. 36 ff.
[6]http://neuralnetworksanddeeplearning.com/chap1.html

$$\sigma(z) = \frac{1}{1 + exp(-z)}$$

- the relu function:[7]

$$\sigma(z) = max(z, 0)$$

- the tanh [8]

$$\sigma(z) = tanh(z) = \frac{2}{1 + exp(-2z)}$$

- the cos function

$$\sigma(z) = cos(z)$$

The next important feature is the learning algorithm that is used to change and thereby train the neural network, so that the network produces a desired output for a given input.[9].

We will work with back propagation. To implement this, we need a differentiable activation function and an error function Err, which receives all n weights as arguments and assigning them to the output error, i.e.being n -dimensional. Backpropagation is a gradient descent procedure. On Err a point of small error or even a point of the smallest error is sought by means of the gradient descent. Thus backpropagation trains the weights of the neural network.[10].

[7]https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

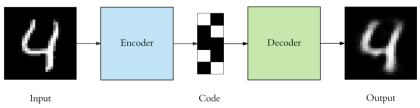
[8]https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[9]D. Kriesel, A brief introduction to Neural Networks, p. 39

[10]D. Kriesel, A brief introduction to Neural Networks, p. 86

3 Autoencoders

An Autoencoder is a specific type of a neural network which is trying to learn an approximation to the identity function, so as to give an output that is similar to the input. The identity function seems a particularly trivial function to be trying to learn; but by placing constraints on the network, such as by limiting the number of hidden units, we can discover interesting structure about the data.[11]
It consists of three components: encoder, code and decoder. The encoder compresses the input and produces the code, the decoder then reconstructs the input only using this code.



Autoencoders are mainly a dimensionality reduction (or compression) algorithm with a couple of important properties:

- Data-specific: Autoencoders are only able to meaningfully compress data similar to what they have been trained on. Since they learn features specific for the given training data, they are different than a standard data compression algorithm like gzip. For example, we can't expect an autoencoder trained on handwritten digits to compress landscape photos.

- Lossy: The output of the autoencoder will not be exactly the same as the input, it will be a close but degraded representation. If you want lossless compression they are not the way to go.

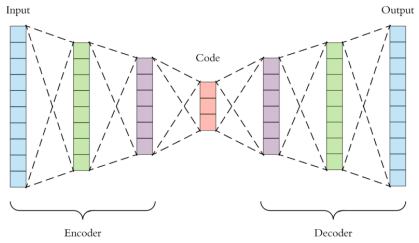
- Unsupervised: Unsupervised learning means that the training set only consists of input patterns, the network tries by itself to detect similarities and to generate pattern classes.[12]
  Autoencoders are considered an unsupervised learning technique since they don't need explicit labels to train on. But to be more precise they are self-supervised because they generate their own labels from the training data.

[11] http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/
[12] D. Kriesel, A brief introduction to Neural Networks, p. 52

[13]

To build an autoencoder three things are needed: an encoding method, decoding method, and a loss function to measure the information loss between the compressed representation of your data and the decompressed representation.[14]

An autoencoder can be visualized as follows:



First the input passes through the encoder, which is an artificial neural network, to produce the code. The decoder, which has the similar artificial neural network structure, then produces the output only using the code. The goal is to get an output identical with the input. The only requirement is that the dimensionality of the input and output needs to be the same. Anything in the middle can be played with.

Four hyperparameters need to be set before training an autoencoder:

- Code size: number of nodes in the middle layer. Smaller size results in more compression.
- Number of layers: the autoencoder can be as deep as we like. In the figure above we have 2 layers in both the encoder and decoder, without considering the input and output.
- Number of nodes per layer: we work here with stacked autoencoders, so called because the layers are stacked one after another. The number of nodes per layer decreases with each subsequent layer of the encoder,

[13]https://medium.com/towards-data-science/applied-deep-learning-part-3-autoencoders-1c083af4d798

[14]https://medium.com/towards-data-science/applied-deep-learning-part-3-autoencoders-1c083af4d798

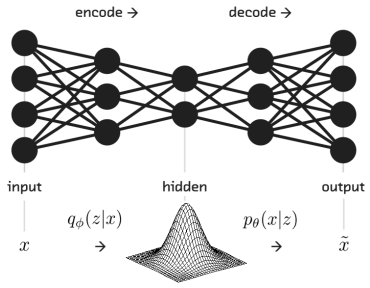
and increases back in the decoder. Also the decoder is symmetric to the encoder in terms of layer structure.

- Loss function: for example we can use mean squared error (mse), but of course other functions are possible too.

Autoencoders then can then be trained in the same way as artificial neuronal networks.[15]

Two interesting versions of autoencoders are:

- Variational Autoencoders: VAEs put a probabilistic spin on the basic autoencoder paradigm, treating their inputs, hidden representations, and reconstructed outputs as probabilistic random variables within a directed graphical model. With this Bayesian perspective, the encoder becomes a variational inference network, mapping observed inputs to (approximate) posterior distributions over latent space, and the decoder becomes a generative network, capable of mapping arbitrary latent coordinates back to distributions over the original data space.[16]



- Generative Adversarial Networks (GANs): there are two main components of a GAN – Generator Neural Network and Discriminator Neural Network. The Generator Network takes an random input and tries to generate a sample of data. It then generates a data which is then

[15] https://medium.com/towards-data-science/applied-deep-learning-part-3-autoencoders-1c083af4d798

[16] http://blog.fastforwardlabs.com/2016/08/22/under-the-hood-of-the-variational-autoencoder-in.html

fed into a discriminator network. The task of Discriminator Network is to take input either from the real data or from the generator and try to predict whether the input is real or generated. Now the training of GAN is done as a fight between generator and discriminator.[17]

[17]https://www.analyticsvidhya.com/blog/2017/06/introductory-generative-adversarial-networks-gans/