Contents

1 Neural networks

The idea of a **N**eural **N**etwork (NN) is to attempt to capture the human thinking process in machine code. For this purpose, a network architecture connects some input (e.g. a picture) to an output (e.g. digits 0-9). Much like in a human brain, the architecture consists of multiple smaller chunks, neurons and layers, which connect in some way to form an emergent intelligent system.

As described, neural network do not yet hold the abilities to achieve their designated tasks, and can hardly be called intelligent. They need to be trained. This is done by presenting an example input (called training data) to the network. The network output is compared to the desired output for the given input via some loss function. During training, the network attempts to minimize this loss function. How it is minimized is often a design choice, and in general will depend on the network architecture, which in turn is influenced by the type of data and kind of task the NN should accomplish.

In the following several network architectures which are relevant for this work are detailed. The most simple option of a **D**ense NN (DNN) is given in **??** in order to introduce several key concepts. **C**onvolutional NNs (CNNs) used for example in image recognition are explained in **??**. Lastly **R**ecurrent NNs (RNNs) that find an application in time series analysis are shown in **??**

1.1 Dense neural network

Dense neural networks are subdivided into layers, which themselve consist of individual neurons. Each neuron conglomerates information from a previous layer according to some weights w_{jk} and a bias b_j and propagates it through some nonlinear activation function $\sigma^{(i)}$. That is, the propagation of an input to the output layer through intermediate, hidden layers $\mathcal{L}^{(i)}$ can be described with the below matrix form:

$$\mathcal{L}_j^{(i)} = \sum_{k=0}^{n^{(i-1)}} \sigma^{(i)} \left(w_{jk} \mathcal{L}_j^{(i-1)} + b_j \right). \tag{1.1}$$

In **??** $\mathcal{L}_j^{(i)}$ is the value of neuron j in layer i, and $n^{(i)}$ is a reference to the number of neurons in layer i. The activation function achieves two important goals. First, it limits the numerical value neurons can have. This ensures numerical stability during training, and is typically achieved by choosing a sigmoidal activation function. Secondly, the nonlinearity of the activation function ensures that the propagation function of the entire network cannot mathematically be reduced to a single layer, as this disallows the network to learn nonlinearly separable patterns [**russell2010artificial**].

Important to note is the fact that the usage of one densely connected layer does not restrict the network architecture to consist solely of such layers. In fact, the network architectures discussed here and in the following section can all be used interchangably. This is a common practice in model building [**szegedy2015going**, **krizhevsky2017imagenet**].

1.2 Convolutional neural network

Convolutional neural networks introduce convolutional layers, which aggregate information from nearby input values. Nearby in this case referring for example to proximate pixels in a 2D image, or neighbouring voxels in a 3D scan. Even succesive inputs in a 1-dimensional time series can be convoluted. In general, the working principle of a convolutional layer can be extended to an arbitrary input shape and size, but will be representatively explained here for a two-dimensional, image-like input.

The convolution in a single layer is done by one or several filters, matrices, that are scalar-multiplied to subchunks of the input data. An example visualization is offered in **??**. In the demonstration, a mock filter designed like

$$\mathcal{F} = \begin{pmatrix} +1 & -1 \\ -1 & +1 \end{pmatrix}$$

is swept across the entire image area. During the iteration, areas of the image that resemble the filter will result in a large (relative to the input data) positive scalar value.

In this fashion, an image can be efficiently scanned for specific patterns (lines, edges, etc.) with just few parameters. Namely, these are the numerical values in the filter matrices, which can be optimized during training. The resulting output of a single layer containing different filters can interatively be propagated into subsequent convolutional layers (searching for curves, corners, etc.) until full-scale image detection of complex structures becomes possible.

1.3 Recurrent neural network

Recurrent neural networks not only propagate an input value forward through their architecture, but also introduce cyclic connections between distinct layers. For example, in the simplest conceptional case there exists a feedback loop that connects the output of the last network layer to the input layer of the first network (c.f. **??**). Other feedback configurations are of course also possible, and generally preferred, as they elegantly eliminate the problem of vanishing/exploding gradients [**hochreiter1991untersuchungen**].

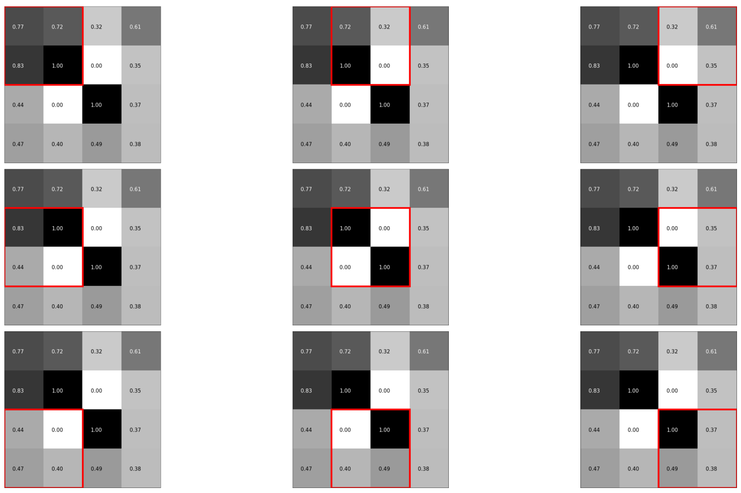

Figure 1.1: asdasd

`imgs/simple_RNN.png`

(a) simple RNN

`imgs/LSTM.png`

(b) LSTM

a) An example RNN architecture, with one layer, where the network output of one step is used as an additional input for the next step. Image taken with changes from [**NN-images**]. **b)** The architecture of a single LSTM layer relies on multiple gates, that update the cell state, or memory of the layer. From [**NN-images**]. .

Moreover, due to the cyclic connections in the network architecture, RNNs are espically qualified for time-series analysis. i.e. where temporally sucessive inputs are highly correlated. A popular example is the **L**ong **S**hort-**T**erm **M**emory (LSTM) architecture, which is visualized in **??** .

The output of a single LSTM layer at timestep $t \neq 0$ is calculated from two variables, the hidden state h_t, as well as the cell state C_t.

1.4 Other architectures