Master's thesis

June 6, 2023

On variational autoencoders: theory and applications

Maksym Sevkovych

Registration number: 3330007

In collaboration with: Duckeneers GmbH

Inspector: Univ. Prof. Dr. Ingo Steinwart

Here should be a catchy abstract and maybe even a short introduction.

Contents

1 Preliminary

In order to understand the topic of variational autoencoders or even autoencoders in general, we need to consider a couple of preliminary ideas. Those ideas consist mainly of neural networks and their optimization - usually being called training. In this chapter, we will tackle the conceptional idea of how to formulate neural networks in a mathematical way and further, we will consider a couple of useful operations that neural networks are capable of doing. Lastly, we will take a look at some strategies of training neural networks.

1.1 Neural networks

Originally, the idea of neural networks originated in analysing mammal's brains. An accumulation of nodes - so called neurons, connected in a very special way that fire an electric impulse to adjacent neurons upon being triggered and transmit information that way. Scientist tried to mimic this natural architecture and replicate the human intelligence artificially. This research has been going for almost 80 years and became immensely popular recently through artificial intelligences like OpenAI's ChatGPT or Google's Bard. But what do these neural networks do? Why are they so popular? What actually is a neural network? All those are very interesting and important questions that we will find answers for.
As already mentioned, neural networks consist of single neurons that move around information upon being „triggered“. Obviously, triggering an artificial neuron can't happen the same way as neurological neurons are being triggered. Hence, we need to model the triggering of a neuron in some way. The idea is to filter information that does not exceed a certain stimulus threshold. This filter is usually being called activation function. Indeed, there are lots of ways of modelling such activation functions and it primarily depends on the specific use-case what exactly the activation function has to fulfil. Therefore, we define activation functions in the most general way possible.

Definition 1.1.1. A non-constant function $\varphi : \mathbb{R} \to \mathbb{R}$ is called an **activation function** if it is continuous.

Even though there is a zoo of different activation functions, we want to consider mainly the following ones.

Example 1.1.2. The following functions are activation functions.

Rectified linear unit (ReLU): $\varphi(t) = \max\{0, t\}$,

Leaky rectified linear unit (Leaky ReLU): $\varphi(t) = \begin{cases} \alpha t, & t \leq 0, \\ t, & t > 0. \end{cases}$

Now, having introduced activation functions we can introduce neurons.

Definition 1.1.3. Let $\varphi : \mathbb{R} \to \mathbb{R}$ be an activation function and $w \in \mathbb{R}^k$, $b \in \mathbb{R}$. Then a function $h : \mathbb{R}^k \to \mathbb{R}$ is called **φ-neuron** with weight w and bias b, if

$$h(x) = \varphi\left(\langle w, x\rangle + b\right), \quad x \in \mathbb{R}^k . \tag{1.1}$$

We call $\theta \coloneqq (w, b)$ the parameters of the neuron h.

In order to expand the architecture, we consider multiple neurons being arranged in a so called layer.

Definition 1.1.4. Let $\varphi : \mathbb{R} \to \mathbb{R}$ be an activation function and $W \in \mathbb{R}^{m \times k}$, $b \in \mathbb{R}^m$. Then a function $H : \mathbb{R}^k \to \mathbb{R}^m$ is called **φ-layer** of width m with weights W and biases b if for all $i = 1, \ldots, m$ the component function h_i of H is a φ-neuron with weight $w_i = W^\top e_i$ and bias $b_i = \langle b, e_i\rangle$, where e_i denotes the standard ONB of $\mathbb{R}^m$.
If we consider $\hat{\varphi} : \mathbb{R}^k \to \mathbb{R}$ as the component-wise mapping of $\varphi : \mathbb{R} \to \mathbb{R}$, meaning $\hat{\varphi}(v) = (\varphi(v_1), \ldots, \varphi(v_k))$, we can generalize the φ-layer $H : \mathbb{R}^k \to \mathbb{R}^m$ by

$$H(x) = \hat{\varphi}(Wx + b), \quad x \in \mathbb{R}^k . \tag{1.2}$$

A visual representation of a neural network can be found in figure 1.1

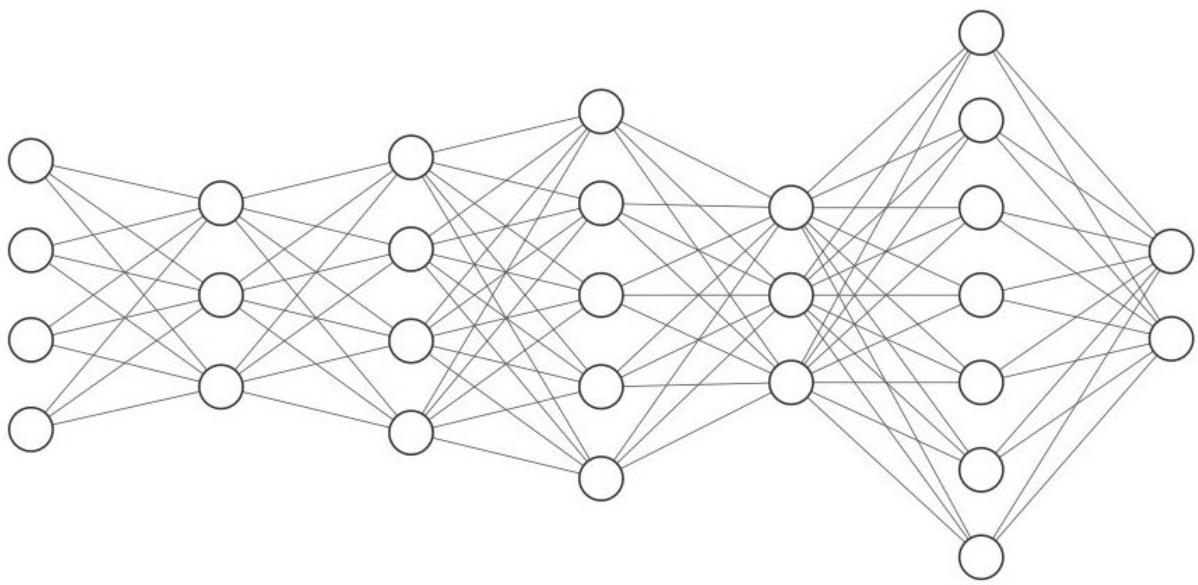

Figure 1.1: A neural network with input $x \in \mathbb{R}^4$ and output $y \in \mathbb{R}^2$. The five hidden layers have dimensions 3, 4, 5, 3 and 7 respectively. The graphic was generated with http://alexlenail.me/NN-SVG/index.html

1.2 Training of neural networks

2 Autoencoders

Now, having introduced the basics of neural networks in Chapter 1 we can consider a specific architecture of a neural network, a so called autoencoder neural network, or short: autoencoders. The conceptional idea of autoencoders is to take a given input, compress (usually called encore) the input to a given size and afterwards, expand (usually called decode) it as close as possible to the original representation again. Such an architecture is widely used in different areas. For example on social media platforms - where users send images to one another. Instead of sending the original image, which size might very well be a couple of megabytes, the image is being encoded first and sent in the compressed representation. Afterwards, the recipient decodes the image to its original representation. This way one has only to transmit the encoded representation, which usually is smaller by magnitudes. Another very important application of autoencoders is in the Machine Learning field. Most state of the art Machine Learning models are using autoencoders, since it is way more efficient to first encode the data and then fit the model on the encoded data. This is quite straight-forward, considering the same argument as in the previous use-case - the encoded data being smaller by magnitudes. This way firstly, processing the samples can happen much faster compared to the non-encoded data samples and secondly, it makes storing data (on the drive and in memory) much more efficient.
The conceptional idea of autoencoders is now clear, but how exactly would one formulate such an architecture mathematically? This is the central question we want to answer in this chapter.

2.1 Mathematical formulation of autoencoders

As already mentioned, the input data is firstly being encoded, and afterwards it is being decoded. Hence, we can divide these two steps into separate architectures - the encoder and the decoder. We will formulate these two steps separately, but we will realise that the architecture is basically analogous. In figure 2.1 we can take a look at a visual example of an autoencoder architecture.

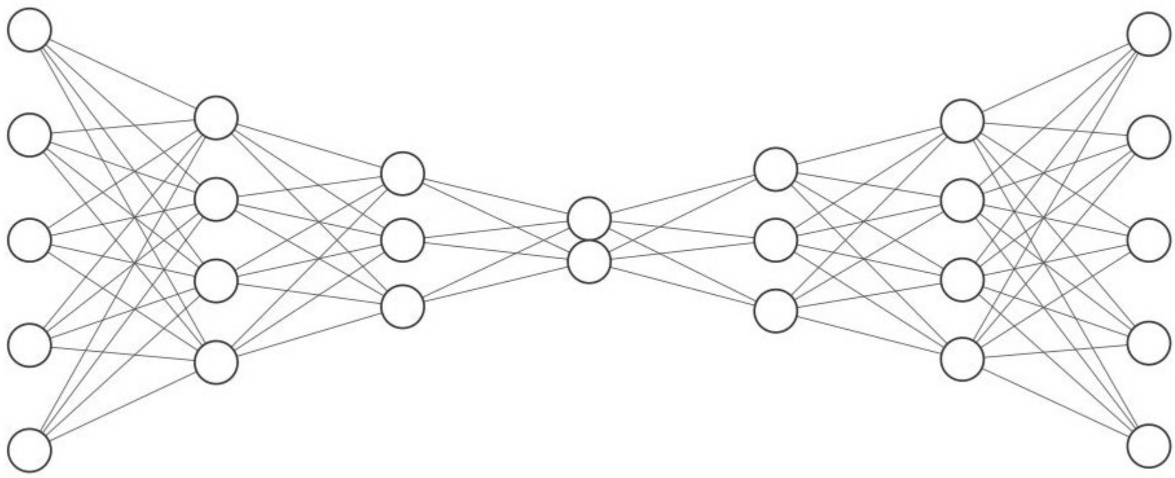

Figure 2.1: An autoencoder neural network with input and output $x, y \in \mathbb{R}^5$. The five hidden layers have dimensions 4, 3, 2, 3 and 4 respectively. Hence, the bottleneck dimension is 2 in this example. The graphic was generated with http://alexlenail.me/NN-SVG/index.html

If we divide the autoencoder as described above, we obtain the encoder and decoder as we can see in figures 2.2 and 2.3 respectively.

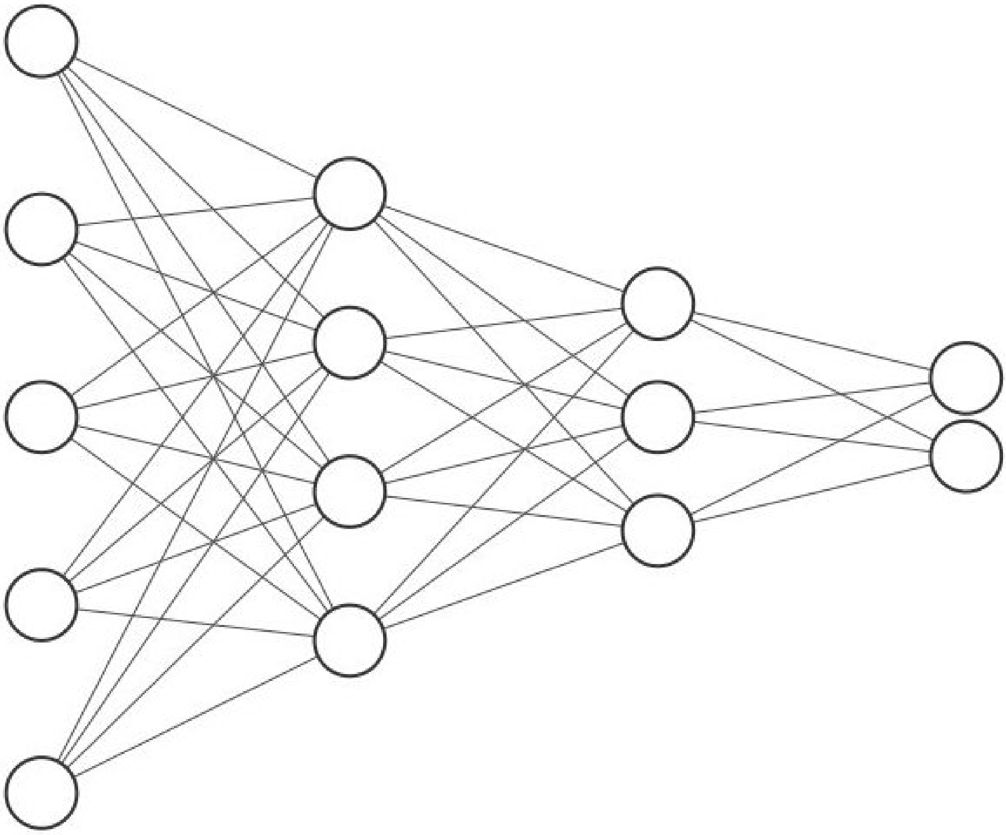

Figure 2.2: An encoder neural network with input $x \in \mathbb{R}^5$ and output $y \in \mathbb{R}^2$. The two hidden layers have dimensions 4 and 3. Hence, the encoder reduces the data dimensionality from 5 to 2 dimension. The graphic was generated with http://alexlenail.me/NN-SVG/index.html

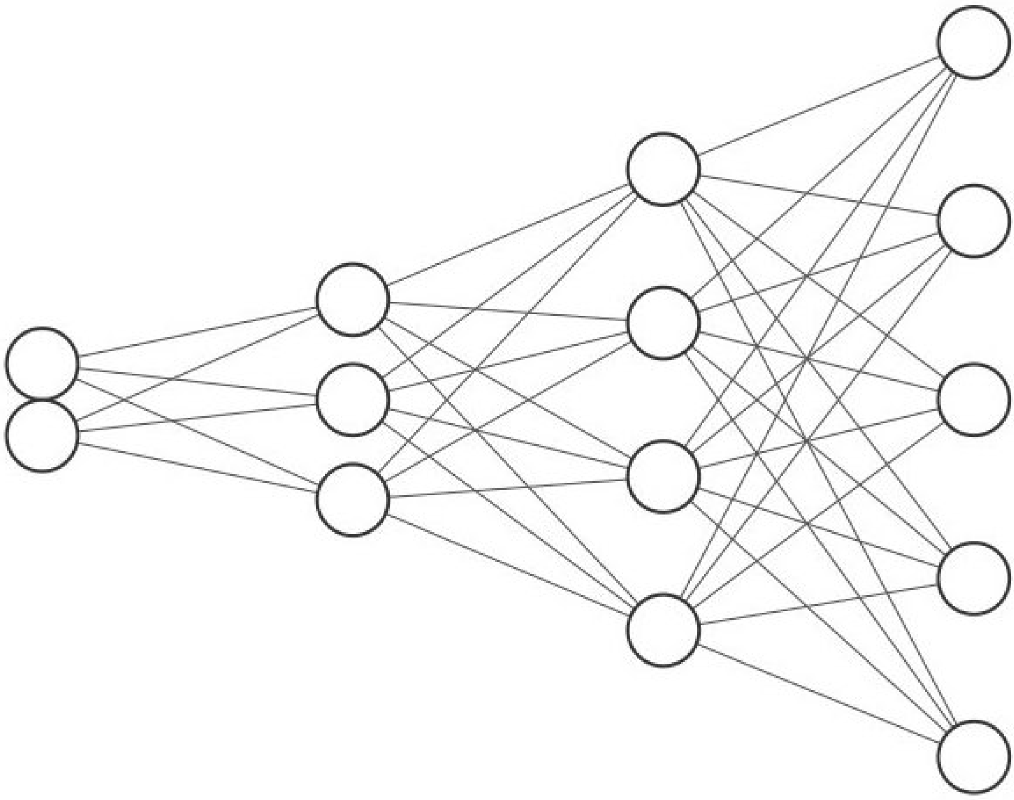

Figure 2.3: A decoder neural network with input $x \in \mathbb{R}^2$ and output $y \in \mathbb{R}^5$. The two hidden layers have dimensions 3 and 4. Hence, the decoder expands the data dimensionality from 2 to 5 dimensions. The graphic was generated with http://alexlenail.me/NN-SVG/index.html

Stuttgart, June 6, 2023