# CHAPTER 15

# Appendices

## 1 Unsupervised learning with autoencoders

So far we have studied learning to predict labels, such as for classification or regression. What if you do not have labels? As introduced in Section 1.2, another type of machine learning, *unsupervised learning*, can be used to discover patterns in data when you don't have labels. Some examples of this are to discover and characterize underlying factors of variation in data, which can aid in scientific discovery, to compress data for efficient storage or communication, or to use as a pre-processing step prior to supervised learning, reducing the amount of data that is needed to learn a good classifier or regressor.

*Autoencoders* are a family of unsupervised learning algorithms that obtain these insights by seeking to learn compressed versions of the original data. Assume that we have input data $\mathcal{D} = \{x^{(1)}, \dots, x^{(n)}\}$, where $x^{(i)} \in \mathbb{R}^d$. The algorithms will output a new dataset $\mathcal{D}_{\text{out}} = \{a^{(1)}, \dots, a^{(n)}\}$, where $a^{(i)} \in \mathbb{R}^k$ with $k < d$. We can think about $a^{(i)}$ as the new *representation* of data point $x^{(i)}$. For example, in Fig. 15.1 we show the learned representations of a dataset of MNIST digits with $k = 2$. We see, after inspecting the individual data points, unsupervised learning has automatically grouped together images of the same digit.

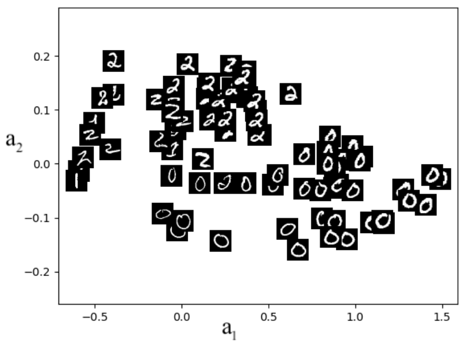

Figure 15.1: Compression of digits dataset into two dimensions. The input $x^{(i)}$, an image of a handwritten digit, is shown at the new low-dimensional representation $(a_1, a_2)$.

Formally, an autoencoder consists of two functions, a vector-valued *encoder* $g : \mathbb{R}^d \rightarrow \mathbb{R}^k$ which deterministically maps the data to the representation space $a \in \mathbb{R}^k$ and a *decoder* $h : \mathbb{R}^k \rightarrow \mathbb{R}^d$ which maps the representation space back into the original data space. The basic architecture of an autoencoder is shown in Figure 15.2. In this example, the original $d$-dimensional input is compressed into $k = 3$ dimensions via the encoder $g(x; W^1) = f_1({W^1}^T x)$ with $W^1 \in \mathbb{R}^{d \times k}$ (the non-linearity $f_1$ is applied to each dimension of the vector). To recover (an approximation to) the original instance, we then apply the decoder $h(a; W^2) = f_2({W^2}^T a)$, where $f_2$ denotes a different non-linearity (activation function). In general, both the decoder and the encoder could involve multiple layers, as opposed to the single layer shown here. Learning seeks parameters $W^1$ and $W^2$ such that the reconstructed instances, $h(g(x^{(i)}; W^1); W^2)$, are close to the original input $x^{(i)}$.

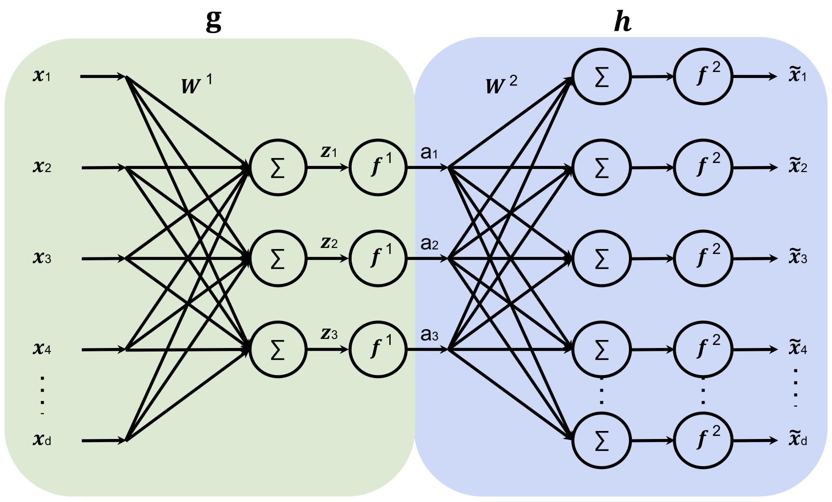

Figure 15.2: Autoencoder structure, showing the encoder (left half, light green), and the decoder (right half, light blue), encoding inputs $x$ to the representation $a$, and decoding the representation to produce $\tilde{x}$, the reconstruction. In this specific example, the representation ($a_1$, $a_2$, $a_3$) only has three dimensions.

We learn autoencoders using the same tools we previously used for supervised learning, namely (stochastic) gradient descent of a multi-layer neural network to minimize a loss function. All that remains is to specify the loss function $L(\tilde{x}, x)$, which tells us how to measure the discrepancy between the reconstruction $\tilde{x} = h(g(x; W^1); W^2)$ and the original input $x$. For example, for continuous-valued $x$ it might make sense to use squared loss, i.e. $L(\tilde{x}, x) = \sum_{j=1}^{d}(x_j - \tilde{x}_j)^2$. Learning then seeks to optimize the parameters of $h$ and $g$ so as to minimize the reconstruction error, measured according to this loss function:

Alternatively, you could think of this as *multi-task learning*, where the goal is to predict each dimension of $x$. One can mix-and-match loss functions as appropriate for each dimension's data type.

$$\min_{W^1, W^2} \sum_{i=1}^{n} L\left(h(g(x^{(i)}; W^1); W^2), x^{(i)}\right)$$

What makes a good representation? Notice that, without further constraints, it is always possible to perfectly reconstruct the input. For example, we could let $k = d$ and $h$ and $g$ be the identity functions. In this case, we would not obtain any compression of the data. To learn something useful, we must create a *bottleneck* by making $k$ to be much smaller than $d$. This forces the learning algorithm to seek transformations that describe the original data using as simple a description as possible. Thinking back to the digits dataset, for example, an example of a compressed representation might be the digit label (i.e., 0–9), rotation, and stroke thickness. Of course, there is no guarantee that the learning algorithm will discover precisely this representation. After learning, we can inspect the learned representations, such as by artificially increasing or decreasing one of the dimensions (e.g. $a_1$) and seeing how it affects the output $h(a)$, to try to better understand what it has learned.

We close by mentioning that even linear encoders and decoders can be very powerful. In this case, rather than minimizing the above objective with gradient descent, a technique called *principal components analysis* can be used to obtain a closed-form solution to the optimization problem using a singular value decomposition. More broadly, there are many types of unsupervised learning. Later in the semester we will learn about *clustering*, where $k = 1$ and $a_i$ takes a discrete number indicating the cluster in which each data point is assigned.