Chapter 15

Linear Factor Models and Auto-Encoders

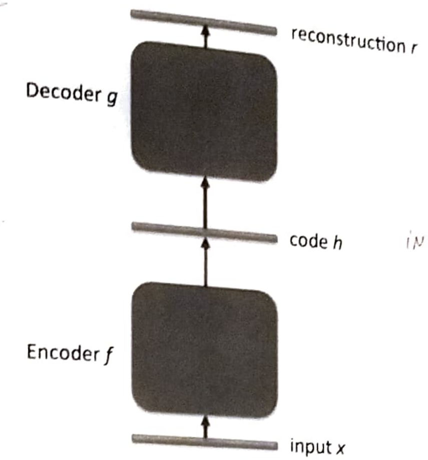
Linear factor models are generative unsupervised learning models in which we imagine that some unobserved factors $\boldsymbol{h}$ explain the observed variables $\boldsymbol{x}$ through a linear transformation. Auto-encoders are unsupervised learning methods that learn a representation of the data, typically obtained by a non-linear parametric transformation of the data, i.e., from $\boldsymbol{x}$ to $\boldsymbol{h}$, typically a feedforward neural network, but not necessarily. They also learn a transformation going backwards from the representation to the data, from $\boldsymbol{h}$ to $\boldsymbol{x}$, like the linear factor models. Linear factor models therefore only specify a parametric decoder, whereas auto-encoder also specify a parametric encoder. Some linear factor models, like PCA, actually correspond to an auto-encoder (a linear one), but for others the encoder is implicitly defined via an inference mechanism that searches for an $\boldsymbol{h}$ that could have generated the observed $\boldsymbol{x}$.

The idea of auto-encoders has been part of the historical landscape of neural networks for decades (LeCun, 1987; Bourlard and Kamp, 1988; Hinton and Zemel, 1994) but has really picked up speed in recent years. They remained somewhat marginal for many years, in part due to what was an incomplete understanding of the mathematical interpretation and geometrical underpinnings of auto-encoders, which are developed further in Chapters 17 and 20.11.

An auto-encoder is simply a neural network that tries to copy its input to its output. The architecture of an auto-encoder is typically decomposed into the following parts, illustrated in Figure 15.1: KEY IDEA

- an input, $\boldsymbol{x}$
- an encoder function f
- a "*code*" or internal representation $\boldsymbol{h} = f(\boldsymbol{x})$

1000 h codes = LIST OF CODES 1, 2, ... 3



WHAT IF INSTEAD DECODER WE USE A LOSS FUNCTION THAT ...

IN A NN THE CODE h IS ACTUALLY THE WEIGHTS OF THE N.N

Figure 15.1: General schema of an auto-encoder, mapping an input $\boldsymbol{x}$ to an output (called reconstruction) $\boldsymbol{r}$ through an internal representation or code $\boldsymbol{h}$. The auto-encoder has two components: the encoder f (mapping $\boldsymbol{x}$ to $\boldsymbol{h}$) and the decoder g (mapping $\boldsymbol{h}$ to $\boldsymbol{r}$).

- a decoder function g
- an output, also called "*reconstruction*" $\boldsymbol{r} = g(\boldsymbol{h}) = g(f(\boldsymbol{x}))$
- a loss function L computing a scalar $L(\boldsymbol{r}, \boldsymbol{x})$ measuring how good of a reconstruction $\boldsymbol{r}$ is of the given input $\boldsymbol{x}$. The objective is to minimize the expected value of L over the training set of examples $\{\boldsymbol{x}\}$.

15.1 Regularized Auto-Encoders

Predicting the input may sound useless: what could prevent the auto-encoder from simply copying its input into its output? In the 20th century, this was achieved by constraining the architecture of the auto-encoder to avoid this, by forcing the dimension of the code $\boldsymbol{h}$ to be smaller than the dimension of the input $\boldsymbol{x}$.

Figure 15.2 illustrates the two typical cases of auto-encoders: undercomplete vs overcomplete, i.e., with the dimension of the representation $\boldsymbol{h}$ respectively smaller vs larger than the input $\boldsymbol{x}$. Whereas early work with auto-encoders, just like PCA, uses the undercompleteness – i.e. a bottleneck in the sequence of layers – to avoid learning the identity function, more recent work allows overcomplete

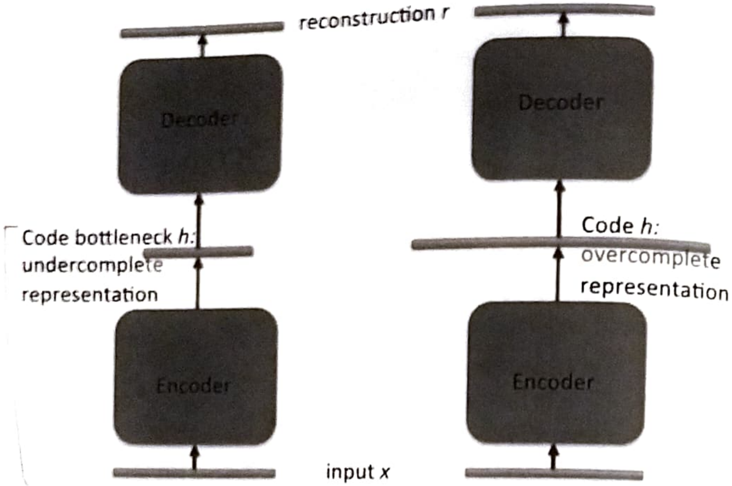


Figure 15.2: Left: undercomplete representation (dimension of code $\boldsymbol{h}$ is less than dimension of input $\boldsymbol{x}$). Right: overcomplete representation. Overcomplete auto-encoders require some other form of regularization (instead of the constraint on the dimension of $\boldsymbol{h}$) to avoid the trivial solution where $\boldsymbol{r} = \boldsymbol{x}$ for all $\boldsymbol{x}$.

representations. What we have learned in recent years is that it is possible to make the auto-encoder meaningfully capture the structure of the input distribution even if the representation is overcomplete, with other forms of constraint or regularization. In fact, once you realize that auto-encoders can capture the input distribution (indirectly, not as a an explicit probability function), you also realize that it should need more capacity as one increases the complexity of the distribution to be captured (and the amount of data available): it should not be limited by the input dimension. This is a problem in particular with the shallow auto-encoders, which have a single hidden layer (for the code). Indeed, that hidden layer size controls both the dimensionality reduction constraint (the code size at the bottleneck) and the capacity (which allows to learn a more complex distribution).

Besides the **bottleneck** constraint, alternative constraints or regularization methods have been explored and can guarantee that the auto-encoder does something useful and not just learn some trivial identity-like function:

- **Sparsity of the representation or of its derivative**: even if the intermediate representation has a very high dimensionality, the effective local dimensionality (number of degrees of freedom that capture a coordinate sys-

uld be much smaller if most of the elements
ant, such that $||\frac{\partial h_i}{\partial x}||$ is close to zero). When
ot participate in encoding local changes in
etation of this situation in terms of *mani-*
nore depth in Chapter 17. The discussion
an auto-encoder naturally tends towards
e actual factors of variation in the data.
rs" clearly fall in this category of sparse

Field, 1996) has been heavily studied
ng and feature inference mechanism.
han an auto-encoder, because it has
ric encoder, and instead uses an iterative inference
instead to compute the code. Sparse coding looks for representations that are both sparse and explain the input through the decoder. Instead of the code being a parametric function of the input, it is instead considered like free variable that is obtained through an optimization, i.e., a particular form of inference:

$$\boldsymbol{h}^* = f(\boldsymbol{x}) = \arg\min_{\boldsymbol{h}} L(g(\boldsymbol{h}), \boldsymbol{x})) + \lambda\Omega(\boldsymbol{h}) \tag{15.1}$$

where L is the reconstruction loss, f the (non-parametric) encoder, g the (parametric) decoder, $\Omega(\boldsymbol{h})$ is a sparsity regularizer, and in practice the minimization can be approximate. Sparse coding has a manifold or geometric interpretation that is discussed in Section 15.8. It also has an interpretation as a directed graphical model, described in more details in Section 19.3. To achieve sparsity, the objective function to optimize includes a term that is minimized when the representation has many zero or near-zero values, such as the L1 penalty $|\boldsymbol{h}|_1 = \sum_i |h_i|$.

- An interesting variation of sparse coding combines the freedom to choose the representation through optimization and a parametric encoder. It is called **predictive sparse decomposition** (PSD) (Kavukcuoglu *et al.*, 2008a) and is briefly described in Section 15.8.2.
- At the other end of the spectrum are simply **sparse auto-encoders**, which combine with the standard auto-encoder schema a sparsity penalty which encourages the output of the encoder to be sparse. These are described in Section 15.8.1. Besides the L1 penalty, other sparsity penalties that have been explored include the Student-t penalty (Olshausen and Field, 1996; Bergstra, 2011). TODO: should the t be in