

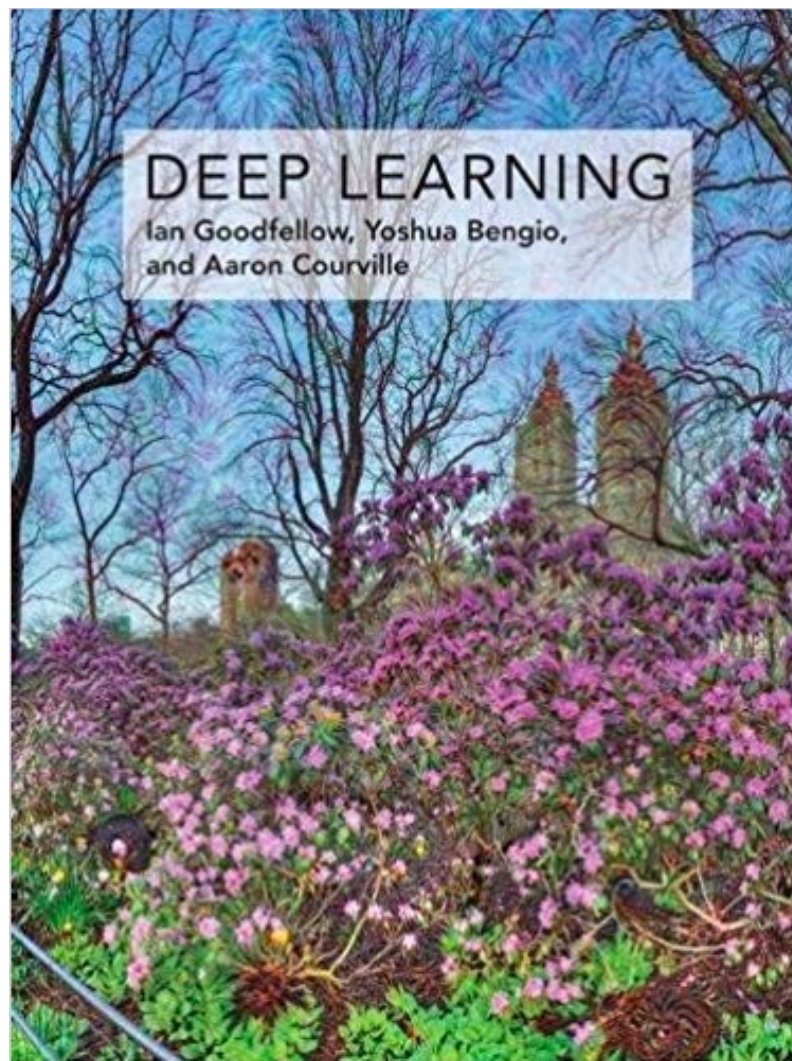


國立陽明交通大學
NATIONAL YANG MING CHIAO TUNG UNIVERSITY

Deep Learning 深度學習 Fall 2023

Autoencoders (Chapter 14)

Prof. Chia-Han Lee
李佳翰 教授



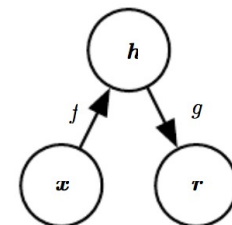


- Figure source: Textbook and Internet
- You are encouraged to buy the textbook.
- Please respect the copyright of the textbook. Do not distribute the materials to other people.



Autoencoders

- An autoencoder is a neural network that is trained to attempt to copy its input to its output.
- The network may be viewed as consisting of two parts: an encoder function $h = f(x)$ and a decoder that produces a reconstruction $r = g(h)$.
- If an autoencoder succeeds in simply learning to set $g(f(x)) = x$ everywhere, then it is not especially useful. Autoencoders are designed to be unable to learn to copy perfectly. Usually they are restricted in ways that allow them to copy only approximately, and to copy only input that resembles the training data. Because the model is forced to prioritize which aspects of the input should be copied, it often learns useful properties of the data.





Autoencoders

- Modern autoencoders have generalized the idea of an encoder and a decoder **beyond deterministic functions** to **stochastic mappings** $p_{\text{encoder}}(\mathbf{h}|\mathbf{x})$ and $p_{\text{encoder}}(\mathbf{x}|\mathbf{h})$.
- Traditionally, autoencoders were used for **dimensionality reduction** or **feature learning**. Recently, **theoretical connections between autoencoders and latent variable models** have brought autoencoders to the forefront of **generative modeling**.
- Autoencoders may be thought of as being **a special case of feedforward networks** and **may be trained with all the same techniques**.



Undercomplete autoencoders

- Copying the input to the output may sound useless, but we are typically not interested in the output of the decoder. Instead, we hope that training the autoencoder to perform the input copying task will result in h taking on useful properties.
- One way to obtain useful features from the autoencoder is to constrain h to have a smaller dimension than x . An autoencoder whose code dimension is less than the input dimension is called undercomplete.
- Learning an undercomplete representation forces the autoencoder to capture the most salient features of the training data.



Undercomplete autoencoders

- The learning process is described simply as **minimizing a loss function**

$$L(\mathbf{x}, g(f(\mathbf{x}))), \quad (14.1)$$

where L is a loss function penalizing $g(f(\mathbf{x}))$ for being dissimilar from \mathbf{x} , such as the mean squared error.

- When the decoder is linear and L is the mean squared error, an undercomplete autoencoder learns to span the **same subspace as PCA**. In this case, an autoencoder trained to perform the copying task has **learned the principal subspace of the training data** as a side effect.
- Autoencoders with nonlinear encoder functions f and nonlinear decoder functions g can thus learn a more powerful **nonlinear generalization of PCA**.



Undercomplete autoencoders

- Unfortunately, if the encoder and decoder are **allowed too much capacity**, the autoencoder can learn to perform the copying task without extracting useful information about the distribution of the data.
- If the hidden code is allowed to have dimension equal to the input, and in the overcomplete case in which the hidden code has dimension greater than the input, even a linear encoder and a linear decoder can learn to copy the input to the output without learning anything useful about the data distribution.



Regularized autoencoders

- Rather than limiting the model capacity by keeping the encoder and decoder shallow and the code size small, regularized autoencoders use a loss function that encourages the model to have other properties besides the ability to copy its input to its output.
- These properties include sparsity of the representation, smallness of the derivative of the representation, and robustness to noise or to missing inputs.
- A regularized autoencoder can be nonlinear and overcomplete but still learn something useful about the data distribution, even if the model capacity is great enough to learn a trivial identity function.



Sparse autoencoders

- A **sparse autoencoder** is simply an autoencoder whose training criterion involves a **sparsity penalty $\Omega(\mathbf{h})$ on the code layer \mathbf{h}** , in addition to the reconstruction error:

$$L(\mathbf{x}, g(f(\mathbf{x}))) + \Omega(\mathbf{h}), \quad (14.2)$$

where $g(\mathbf{h})$ is the decoder output, and typically we have $\mathbf{h} = f(\mathbf{x})$, the encoder output.

- **Sparse autoencoders** are typically used to **learn features for another task, such as classification**.
- An autoencoder that has been regularized to be sparse must respond to **unique statistical features** of the dataset it has been trained on, rather than simply acting as an identity function.



Sparse autoencoders

- We can think of the entire sparse autoencoder framework as approximating likelihood training of a generative model that has latent variables.
- Suppose we have a model with visible variables \mathbf{x} and latent variables \mathbf{h} , with an explicit joint distribution $p_{\text{model}}(\mathbf{x}, \mathbf{h}) = p_{\text{model}}(\mathbf{h}) p_{\text{model}}(\mathbf{x}|\mathbf{h})$. We refer to $p_{\text{model}}(\mathbf{h})$ as the model's prior distribution over the latent variables, representing the model's beliefs prior to seeing \mathbf{x} .
- It provides a different reason for why the features learned by the autoencoder are useful: they describe the latent variables that explain the input.



Denoising autoencoders

- Rather than adding a penalty Ω to the cost function, we can obtain an autoencoder that learns something useful by **changing the reconstruction error term of the cost function**.
- A **denoising autoencoder (DAE)** instead minimizes

$$L(\mathbf{x}, g(f(\tilde{\mathbf{x}}))), \quad (14.9)$$

where $\tilde{\mathbf{x}}$ is a copy of \mathbf{x} that has been corrupted by some **form of noise**. Denoising autoencoders must therefore **undo this corruption rather than simply copying input**.



Denoising autoencoders

- Denoising training forces f and g to implicitly learn the structure of $p_{\text{data}}(\mathbf{x})$. Denoising autoencoders provide another example of how useful properties can emerge as a byproduct of minimizing reconstruction error.
- They are an example of how overcomplete, high-capacity models may be used as autoencoders as long as care is taken to prevent them from learning the identity function.



Regularizing by penalizing derivatives

- Another strategy for regularizing an autoencoder is to use a penalty Ω , as in sparse autoencoders,

$$L(\mathbf{x}, g(f(\mathbf{x}))) + \Omega(\mathbf{h}, \mathbf{x}), \quad (14.10)$$

but with a different form of Ω :

$$\Omega(\mathbf{h}, \mathbf{x}) = \lambda \sum_i \|\nabla_{\mathbf{x}} h_i\|^2. \quad (14.11)$$

- This forces the model to learn a function that does not change much when \mathbf{x} changes slightly. Because this penalty is applied only at training examples, it forces the autoencoder to learn features that capture information about the training distribution. An autoencoder regularized in this way is called a **contractive autoencoder (CAE)**.



Contractive autoencoders

- The **contractive autoencoder** introduces an **explicit regularizer** on the code $\mathbf{h} = f(\mathbf{x})$, encouraging the derivatives of f to be as small as possible:

$$\Omega(\mathbf{h}) = \lambda \left\| \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right\|_F^2. \quad (14.18)$$

- The penalty $\Omega(\mathbf{h})$ is the **squared Frobenius norm** (sum of squared elements) **of the Jacobian matrix of partial derivatives** associated with the encoder function.



Contractive autoencoders

- There is a connection between the denoising autoencoder and the contractive autoencoder: It was shown that in the limit of small Gaussian input noise, the denoising reconstruction error is equivalent to a contractive penalty on the reconstruction function that maps x to $r = g(f(x))$.
- Denoising autoencoders make the reconstruction function resist small but finite-sized perturbations of the input.
- Contractive autoencoders make the feature extraction function resist infinitesimal perturbations of the input.



Contractive autoencoders

- The name contractive arises from the way that **the CAE warps space**. Specifically, because the CAE is trained to resist perturbations of its input, it is encouraged to **map a neighborhood of input points to a smaller neighborhood of output points**. We can think of this as **contracting the input neighborhood to a smaller output neighborhood**.



Representational power, layer size and depth

- Autoencoders are often trained with only a single layer encoder and a single layer decoder. However, **using deep encoders and decoders offers many advantages.**
- An autoencoder with a single hidden layer is able to represent the identity function along the domain of the data arbitrarily well. However, we are not able to enforce arbitrary constraints, such as that the code should be sparse.
- Experimentally, **deep autoencoders yield much better compression** than corresponding shallow or linear autoencoders.



Applications of autoencoders

- Autoencoders have been successfully applied to dimensionality reduction and information retrieval tasks.
- Lower-dimensional representations can improve performance on many tasks, such as classification. Many forms of dimensionality reduction place semantically related examples near each other. The hints provided by the mapping to the lower-dimensional space aid generalization.
- Models of smaller spaces consume less memory and runtime.



Applications of autoencoders

- Information retrieval is the task of finding entries in a database that resemble a query entry. This task derives the usual benefits from dimensionality reduction that other tasks do, but also derives the additional benefit that search can become extremely efficient in certain kinds of low-dimensional spaces.
- Specifically, if we train the dimensionality reduction algorithm to produce a code that is low-dimensional and binary, then we can store all database entries in a hash table that maps binary code vectors to entries. This hash table allows us to perform information retrieval by returning all database entries that have the same binary code as the query.