Chapter 8
Autoencoders (AE)

8.1 Overview

So far, we have presented various deep learning models for supervised learning where output labels (e.g., heart failure diagnosis) are available in the training data. However, unlabeled data are the norm in many real-world applications. Next, we introduce the autoencoder, which is a popular unsupervised deep learning model.

In general, the autoencoder (AE) is an unsupervised and nonlinear dimensionality reduction model, which is widely used in many healthcare applications [5, 13, 94, 104, 113, 143, 144, 173]. An autoencoder maps inputs to an internal representation via an encoder and then maps the internal representation back to the input space through a decoder. The composition of encoder and decoder is called the reconstruction function. The autoencoder's objective tries to minimize the reconstruction loss, thus allowing AEs to focus on essential properties of the data while reducing the dimensionality.

Sparse autoencoders (SAE) and denoising autoencoders (DAE) are two AE variants. For SAE, the reconstruction loss is regularized via a sparsity penalty on internal representation so that the model tries to learn sparse representation. SAE was often used for unsupervised EHR phenotyping [94] or sparse EEG feature representation [100, 170, 173]. DAE introduces randomly corrupted inputs, through which the model would gain robustness towards missing data or noises. DAE was used for learning robust representations of patient phenotypes from EHRs [5, 13, 113]. In the summary, we will introduce several AEs, including stacked autoencoders, sparse autoencoders, denoising autoencoders and their healthcare applications (Table 8.1).

C. Xiao, J. Sun, *Introduction to Deep Learning for Healthcare*,
https://doi.org/10.1007/978-3-030-82184-5_8

Table 8.1 Notations for autoencoders

Notation	Definition
$\boldsymbol{x} \in \mathbb{R}^d$; X	Input vector; random variable of input data distribution
$\boldsymbol{h} \in \mathbb{R}^k$	Hidden layer
s_t	Hidden state at time t
$\boldsymbol{r}$; R	Reconstructed data; reconstructed data distribution

8.2 Autoencoders

An autoencoder is a neural network that aims to reconstruct the input data via a nonlinear dimensionality reduction. Formally, an autoencoder includes an encoder and a decoder:

- *Encoder* $f_\theta()$ transforms an input vector $\boldsymbol{x} \in \mathbb{R}^d$ into a hidden layer $\boldsymbol{h} \in \mathbb{R}^k$ (usually $k < d$):

$$\boldsymbol{h} = f_\theta(\boldsymbol{x}) = \sigma_1(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

  where $\boldsymbol{W} \in \mathbb{R}^{k\times d}$ is weight matrix, $\boldsymbol{b} \in \mathbb{R}^k$ is a bias vector and σ_1 is the input activation function.
- *Decoder* $g_{\theta'}()$ maps the hidden representation $\boldsymbol{h} \in \mathbb{R}^k$ back to a reconstructed vector

$$\boldsymbol{r} = g_{\theta'}(\boldsymbol{h}) = \sigma_2(\boldsymbol{W}'\boldsymbol{h} + \boldsymbol{b}')$$

  where $\boldsymbol{W'} \in \mathbb{R}^{d\times k}$ is weight matrix, $\boldsymbol{b'} \in \mathbb{R}^d$ is a bias vector and σ_2 the output activation function.

The procedure of encoding and decoding is depicted in Fig. 8.1. The encoder and the decoder are parameterized by $\theta = \{\boldsymbol{W}, \boldsymbol{b}\}$ and $\theta' = \{\boldsymbol{W}', \boldsymbol{b}'\}$, respectively. Note that without the nonlinear transformation σ_1, σ_2, the process becomes linear dimensionality reduction similar to Principal Component Analysis (PCA).

In general, the objective of AE is $\min L(\boldsymbol{x}, \boldsymbol{r})$, where $L(\boldsymbol{x}, \boldsymbol{r}) \propto -\log p(\boldsymbol{x}|\boldsymbol{r})$. Here $L(\cdot)$ is a loss function such as squared error or cross-entropy loss. The choice of the loss function depends on the distribution of $\boldsymbol{x}$.

1. If $\boldsymbol{x} \in \mathbb{R}^d$ is a real-value vector following normal distribution, that is $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{r}, \sigma^2\boldsymbol{I})$, the squared loss $L(\boldsymbol{x}, \boldsymbol{r}) = ||\boldsymbol{x} - \boldsymbol{r}||^2$ is preferred.
2. If $\boldsymbol{x}$ is binary-valued, the decoder needs to produce a $\boldsymbol{r} \in \{0, 1\}^d$. Thus a sigmoid activation will typically be used in the decoder. This yields the choice of cross-entropy loss: $L(\boldsymbol{x}, \boldsymbol{r}) = -\sum_i [\boldsymbol{x}_i \log \boldsymbol{r}_i + (1 - \boldsymbol{x}_i) \log(1 - \boldsymbol{r}_i)]$.

In general, the formulation of autoencoders is an empirical risk optimization.

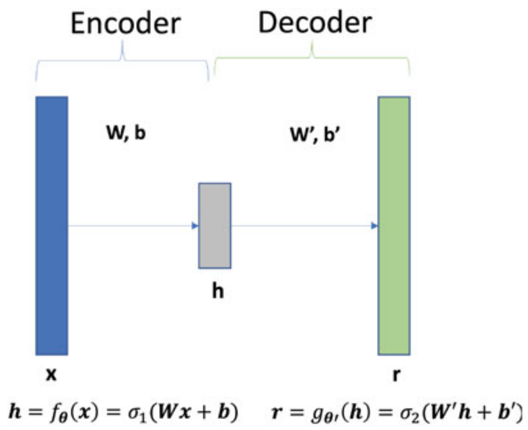
Encoder
Decoder
W, b
W′, b′
h
x
r
$h = f_\theta(x) = \sigma_1(Wx + b)$
$r = g_{\theta'}(h) = \sigma_2(W'h + b')$

Fig. 8.1 Autoencoder model illustration

$$\hat{R}(f_\theta, g_{\theta'}, \boldsymbol{x}) = \sum_{i=1}^{n} L(\boldsymbol{x}_i, g_{\theta'}(f_\theta(\boldsymbol{x}_i)))$$

where $\boldsymbol{x}_i$ is the i-th input data point and $g_{\theta'}(f_\theta(\boldsymbol{x}_i))$ the reconstruction of $\boldsymbol{x}_i$.

To avoid overfitting or to enhance sparsity, we can add regularization on the model parameters θ or θ'.

$$\hat{R}_\lambda(f_\theta, g_{\theta'}, \boldsymbol{x}) = \sum_{i=1}^{n} L(\boldsymbol{x}_i, g_{\theta'}(f_\theta(\boldsymbol{x}_i))) + \lambda\Omega. \tag{8.1}$$

where Ω is the regularization function and $\lambda \geq 0$ is the regularization weight. For example, L1 regularization $\Omega = \sum |f_\theta(\boldsymbol{x}_i)|$ to impose sparsity constraint on the latent activations, or Kullback–Leibler divergence to penalize excessive activations which we will discuss below.

8.3 Sparse Autoencoders

Sparse autoencoders (SAE) [115] aim at imposing a sparsity constraint on the hidden layer of AEs. The idea is to have fewer neurons activated when applying to the input. The SAE constrains the neurons to be inactive most of the time (i.e., most neurons' output is 0). Specifically, we assume the activation function of hidden units is sigmoid, which means the output is between 0 and 1. We also denote $\hat{\rho}_j$ be the average activation of hidden unit j (averaged over all n data points) such that

$$\hat{\rho}_j = \frac{1}{n}\sum_{i=1}^{n} h_j[i]$$

where $h_j[i]$ represents the j-th hidden unit of the input data point i.

The SAE tries to enforce the constraint $\hat{\rho}_j = \rho$ where ρ is a sparsity parameter. For example, we can set the sparse parameter to be 0.05, which means each hidden unit is activated on 5% of the data points and is zero on 95% of the remaining data points.

To achieve this goal, the SAE adds a penalty term to the AE objective to penalize $\hat{\rho}_j$ that deviates significantly from ρ, and the penalty term that is mostly chosen is based on the Kullback–Leibler (KL) divergence $\mathcal{D}_{\mathrm{KL}}$ between $\hat{\rho}_j$ and ρ as given below. The total (KL) divergence $\mathcal{D}_{\mathrm{KL}}$ between $\hat{\rho}_j$ and ρ across all k hidden units is given by

$$\sum_{j=1}^{k} \mathcal{D}_{KL}(\rho||\hat{\rho}_j) = \sum_{j=1}^{k}(\rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j})$$

where k is the number of neurons in this hidden layer. Since the KL divergence reaches its minimum of 0 when $\hat{\rho}_j = p$ and grows to ∞ as $\hat{\rho}_j$ approaches 0 or 1, this penalty term based on KL can effectively enforce $\hat{\rho}_j$ to be close to ρ.

With the aforementioned penalty term, the new objective for SAE is given below.

$$\arg\min_{\theta,\theta'} \frac{1}{n}\sum_{i=1}^{n} L(\boldsymbol{x}_i, \boldsymbol{r}_i) + \gamma \sum_{j=1}^{k} \mathcal{D}_{KL}(\rho||\hat{\rho}_j) \tag{8.2}$$

where γ is the regularization weight and $\sum_{j=1}^{k} \mathcal{D}_{KL}(\rho||\hat{\rho}_j)$ is the sparsity regularization on the hidden units.

8.4 Stacked Autoencoders

When multiple autoencoders are stacked together such that outputs of each layer are used as inputs of the next layer, we have the stacked autoencoders. Formally, consider an autoencoder with K layers. Stacked autoencoders perform the encoding step by running each layer's encoding step in forward order and perform decoding by running each layer's decoding step in backward order. Given K layers of encoders and decoders each, we have the following formulation:

- *Encoder* of the first layer $f_\theta^{(1)}()$ transforms an input vector $\boldsymbol{x} \in \mathbb{R}^d$ into a hidden layer $\boldsymbol{h}^{(1)}$.

$$\boldsymbol{h}^{(1)} = f_\theta^{(1)}(\boldsymbol{x}) = \sigma_1(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}).$$

  Then the encoder of the k-th layer $f_\theta^{(k)}()$ transforms the output from the previous layer $\boldsymbol{h}^{(k-1)}$ into next hidden layer $\boldsymbol{h}^{(k)}$:

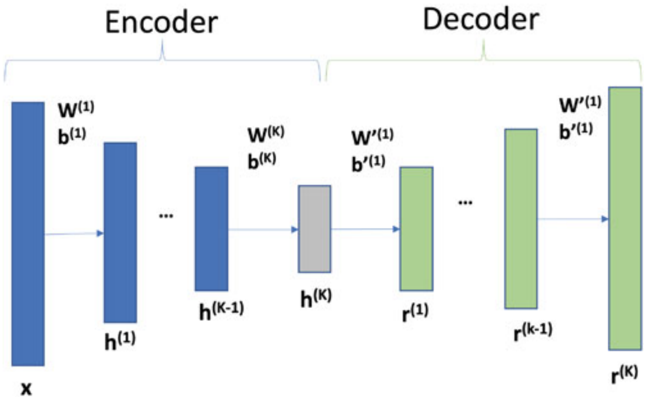

Fig. 8.2 Stacked autoencoder model of K layers

$$\boldsymbol{h}^{(k)} = f_{\theta}^{(k)}(\boldsymbol{x}) = \sigma_1(\boldsymbol{W}^{(k)}\boldsymbol{h}^{(k-1)} + \boldsymbol{b}^{(k)}).$$

- The initial input to the first layer decoder is the hidden representation from the K-th layer encoder, i.e., $\boldsymbol{r}^{(0)} = \boldsymbol{h}^{(K)}$. In general, the *decoder* of the k-th layer $g_{\theta'}^{(k)}()$ maps the hidden representation from the previous reconstruction layer $\boldsymbol{r}^{(k-1)}$ back to the next reconstruction vector:

$$\boldsymbol{r}^{(k)} = g_{\theta'}^{(k)}(\boldsymbol{r}^{(k-1)}) = \sigma_2(\boldsymbol{W'}^{(k)}\boldsymbol{r}^{(k-1)} + \boldsymbol{b'}^{(k)}).$$

8.5 Denoising Autoencoders

In [163], the authors proposed denoising autoencoders (DAE), which can learn more robust representation against noisy input. The idea is to add noises to the original input $\boldsymbol{x}$ to obtain corrupted version $\tilde{\boldsymbol{x}}$ and then try to reconstruct clean uncorrupted input $\boldsymbol{x}$ from noisy input $\tilde{\boldsymbol{x}}$ (Fig. 8.3).

- First, we corrupt the initial input $\boldsymbol{x}$ to obtain $\tilde{\boldsymbol{x}}$ by means of a stochastic mapping $\tilde{\boldsymbol{x}} \sim q_D(\tilde{\boldsymbol{x}}|\boldsymbol{x})$. Different types of noises can be created: (1) random Gaussian noises at all locations, (2) masking noises (a small number of locations are set to 0), (3) salt-and-pepper noises (a small number of locations are set to 0 or 1 at random).
- Then we pass the corrupted input $\tilde{\boldsymbol{x}}$ through a standard encoder:

$$\boldsymbol{h} = f_{\theta}(\tilde{\boldsymbol{x}}) = \sigma(\boldsymbol{W}\tilde{\boldsymbol{x}} + \boldsymbol{b})$$

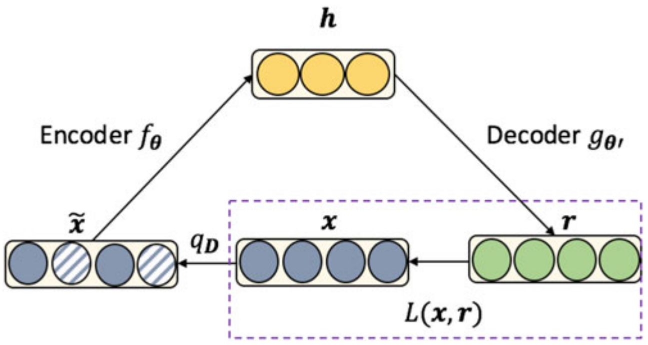
$\boldsymbol{h}$
Encoder $f_{\boldsymbol{\theta}}$
Decoder $g_{\boldsymbol{\theta'}}$
$\tilde{\boldsymbol{x}}$
q_D
$\boldsymbol{x}$
$\boldsymbol{r}$
$L(\boldsymbol{x},\boldsymbol{r})$

Fig. 8.3 A schematic representation of denoising autoencoders. The input data is stochastically corrupted via q_D to $\tilde{\boldsymbol{x}}$. It then be mapped to $\boldsymbol{h}$ via encoder f_θ and tries to reconstruct $\boldsymbol{x}$ via decoder $g_{\theta'}$ and produces reconstruction $\boldsymbol{r}$. The reconstruction error is measured by $L(\boldsymbol{x}, \boldsymbol{r})$

- Finally, the decoder will try to map $\boldsymbol{h}$ back to a reconstructed vector that is close to the original uncorrupted input $\boldsymbol{x}$:

$$\boldsymbol{r} = g_{\theta'}(\boldsymbol{h}) = \sigma(\boldsymbol{W}'\boldsymbol{h} + \boldsymbol{b}')$$

  Note that denoising autoencoders try to minimize the reconstruction loss between the original $\boldsymbol{x}$ and the reconstruction $\boldsymbol{r}$ from the corrupted $\tilde{\boldsymbol{x}}$.

8.6 Case Study: "Deep Patient" via Stacked Denoising Autoencoders

Problem How to derive a general-purpose feature representation from EHR data? Authors of "Deep Patient" compared several unsupervised feature construction methods, including Stacked Denoising Autoencoders (SDA) for supporting predictive modeling tasks [113].

Data In "deep patient", patient representations were learned based on the EHR data from the Mount Sinai hospital and evaluated using disease classification and prediction tasks. In particular, the dataset includes over 700k patients with up to 12 years of patient history. The raw data include structured data such as diagnoses (ICD-9 codes), medications, procedures, lab tests, and unstructured clinical text. The clinical notes are processed with Open Biomedical Annotator [132]. They used the number of occurrences of all the events in their EHR data as the raw feature values. All the feature values were then normalized to lie between zero and one to reduce the data variance while preserving zero entries.

Table 8.2 Disease classification results in terms of area under the ROC curve (AUC-ROC), accuracy and F-score. The difference with the corresponding second best measurement is statistically significant

Patient representation	ROC-AUC	Accuracy	F-1
Raw feature	0.659	0.805	0.084
PCA	0.696	0.879	0.104
GMM	0.632	0.891	0.072
K-means	0.672	0.887	0.093
ICA	0.695	0.882	0.101
Deep patient	**0.773**	**0.929**	**0.181**

The bold values indicate the best performance numbers across all methods

Method A 3-layer stacked denoising autoencoders were used to process input feature vectors of patients in an unsupervised manner. In particular, 500 hidden units per layer are used for each layer. Then 5% random input locations were set to 0 for each layer. This can be viewed as simulating the presence of missed components in the EHRs (e.g., medications or diagnoses not recorded in the patient records), thus assuming that the input clinical data is a degraded or "noisy" version of the actual clinical situation. Sigmoid activations are used. Next, the learned encoding function is applied to the clean input, and the resulting hidden layer is the final patient representation. As a result, each patient was represented by a dense vector of 500 features.

Result The output of the stacked denoising autoencoder was compared with other well-known feature learning algorithms and demonstrated utility in various tasks. In particular, the following algorithms were used as baselines, including principal component analysis (PCA) with 100 principal components, k-means clustering with 500 clusters, Gaussian mixture model with 200 mixtures and full the covariance matrix, and independent component analysis with 100 principal components.

Then random forest classifiers were implemented to predict the probability that patients might develop a certain disease given their current clinical status using the learned features on a separate dataset of 200,000 patients. The training and evaluation follow a one-vs-all strategy. Prediction result performance was listed in Table 8.2. The main finding is that Deep Patient (i.e., DAE) significantly outperformed the unsupervised methods in the classification tasks. This result confirmed the DAE as an effective method for feature engineering from EHR data.

8.7 Case Study: Learning from Noisy, Sparse, and Irregular Clinical Data

Problem The study in [94] targeted unbiased deep phenotype discovery using large-scale EHR data. The paper argues that standard deep learning on its own cannot reliably learn compact longitudinal features from the noisy, sparse, and irregular observations typically existed in EHRs. To bridge the gap, this work first applied Gaussian process regression to transform the irregularly sampled lab results

(i.e., serum uric acid in this case) into a continuous longitudinal probability density over time and then learn phenotypes using sparse autoencoder (SAE) over the output of Gaussian process regression.

Data The data used in this work was extracted from Vanderbilt's Synthetic Derivative, a deidentified mirror of their production EHR. This mirror contains over 15 years of longitudinal clinical data on over 2 million individuals. The authors identified 4368 records of individuals with either gout or acute leukemia, but not both. We extracted the full sequence of uric acid values and measurement times from each record and associated it with the appropriate disease label. The disease label served as the reference standard for downstream evaluation but was not used in the feature learning. Roughly a third of the records were set aside as a final test set.

Method The paper proposed a framework consisting of the following two steps: the transformation step and the feature learning step. The transformation step assumed that there was an unobserved source function for each individual that represented the true uric acid concentration over time, and considered each uric acid sequence to be a set of possibly noisy samples taken from that source function. The authors use Gaussian process regression to learn the continuous uric acid function from limited samples of the lab results.

A sliding window of 30 days was applied on daily resampled data to construct patches in the feature learning step. A two-layer stacked sparse autoencoder was then trained on patches of 30-day long where the inputs are 30-dimensional vectors. The reason for using sparse autoencoder is to learn sparse features that will be used as new phenotypes.

To evaluate the effectiveness of the proposed method, three tasks were evaluated: (1) the face validity of the learned features as low-level, high-resolution phenotypes, (2) their ability to illuminate unknown disease population subtypes, and (3) their accuracy in distinguishing between disease phenotype signatures known to exist in the data but which were unknown to the feature learning algorithm.

Result To assess the quality of the learned features, they were evaluated using a supervised classification task unknown to the feature-learning algorithm. Since the features were not optimized for the evaluation task, the task serves as a generalized performance test. They were compared with expert features, which were optimized for the specific task. The selected learning task was to distinguish the known gout vs. leukemia phenotypes using only the uric acid sequences. The selected classification algorithm was logistic regression because a simple linear classifier is more likely to illuminate differences in feature quality than a more sophisticated algorithm.

Four supervised classifiers were trained for comparison: (1) a classifier using first-layer learned features, (2) a classifier using second-layer learned features, (3) a gold-standard classifier using expert-engineered features, (4) a baseline classifier using the sequence means as the only input feature. The first two classifiers evaluated the learned features. The gold-standard classifier was intended to estimate the upper-bound performance for the task. The baseline classifier was intended to

Table 8.3 Unsupervised features were as powerful as expert engineered features in distinguishing uric acid sequences from gout vs. leukemia

Classifier	AUC (training)	AUC [CI] (test)
First-layer learned features	0.969	0.972 [0.968, 0.979]
Second-layer learned features	0.965	0.972 [0.968, 0.979]
Expert engineered features	0.968	0.974 [0.966, 0.981]
Baseline (sequence mean only)	0.922	0.932 [0.922, 0.944]

establish how well a single basic feature can do on the task. Gout and leukemia disease labels were used as the class labels for all classifiers.

The performance was evaluated using the area under the Receiver Operating Characteristic curve (AUC) on a held-out test set. The experiment shows that features generated using Gaussian process regression followed by sparse autoencoders can achieve similar predictive performance as the expert features shown in Table 8.3.

8.8 Exercises

1. What are the main difference between autoencoder and principal component analysis?
2. What is the main difference between autoencoder and denoising autoencoder?
3. What is the most analogous method to Autoencoders?
   (a) K-means clustering
   (b) Principal component analysis
   (c) Support vector machine
   (d) Hierarchical clustering
4. Which of the following is NOT true about autoencoders?
   (a) It is an unsupervised method.
   (b) It is a lossless compression technique.
   (c) It is a dimensionality reduction method.
   (d) It is a feedforward neural network.
5. What is NOT true about sparse autoencoder?
   (a) It introduces sparsity in the latent code $\mathbf{h}$.
   (b) Sigmoid activation function is used to produce latent code $\mathbf{h}$.
   (c) The same loss function to the standard autoencoder is used for sparse autoencoder.
   (d) Sparsity level on each dimension of h needs to be specified.

6. What is NOT true about denoising autoencoder?
    (a) It adds random noise to the original input before applying the autoencoder model.
    (b) Its loss function is between reconstruction and the original input x without adding random noise.
    (c) It is more expensive to train because of random noise added to the original input.
    (d) It is more robust against noises thanks to the introduction of corrupted input.
7. What are the model parameters first learned in a stacked autoencoder as illustrated in Fig. 8.2?
8. What is NOT true about stacked autoencoder?
    (a) It applies multiple encoders first, then applies the corresponding decoders in reverse orders.
    (b) It is a deep neural network of 2K layers where K is the number of autoencoders.
    (c) It is trained in an end-to-end fashion as a single model using backpropagation.
    (d) It is trained sequentially as K separate autoencoders.
9. What other model is used before applying autoencoder in the phenotype discovery paper [94]?
10. Question 8 What is the variant of autoencoder model used in deep patient paper [113]?