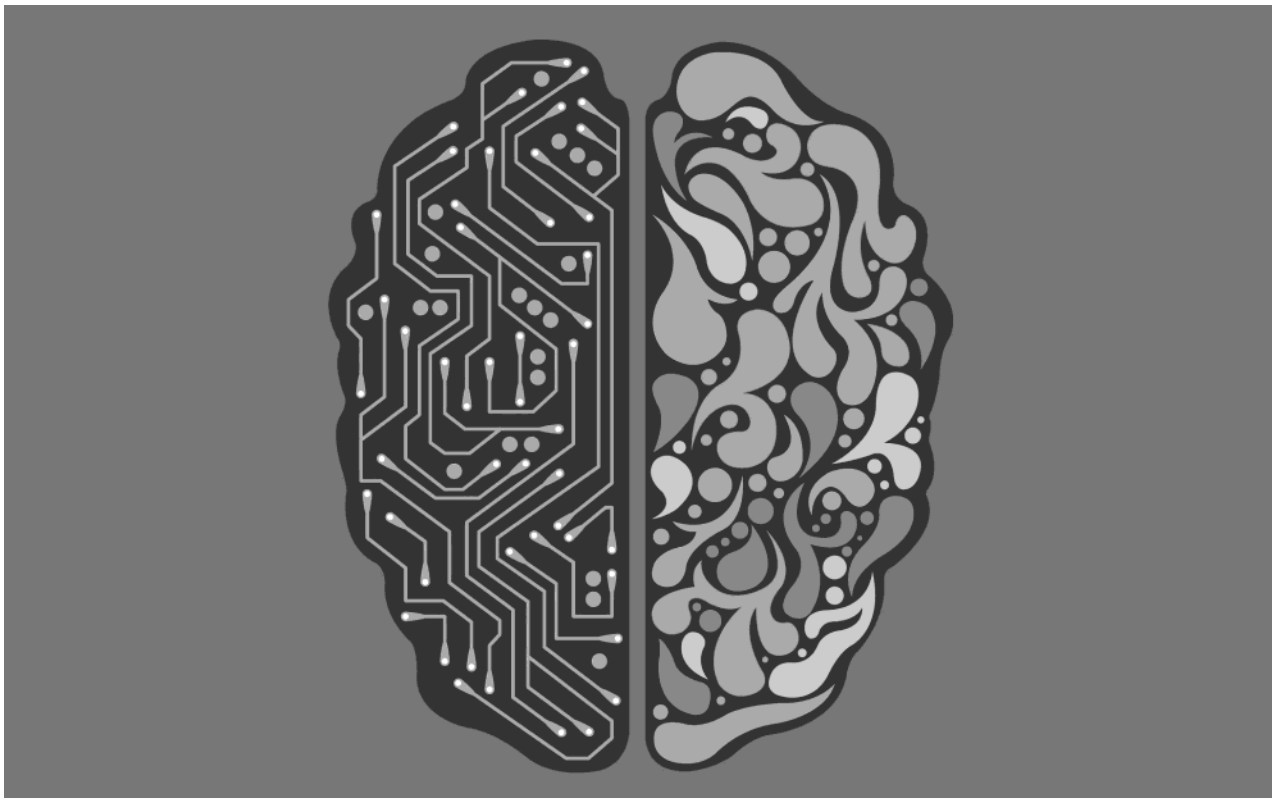
Autoencoders

2019年5月7日 11:57



Source Lists
The essay from: https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f
The code from : https://github.com/nathanhubens/Autoencoders
The essay Chinese translation from: https://zhuanlan.zhihu.com/p/34095160

Other source over the autoencoder:
Wikipedia: https://en.wikipedia.org/wiki/Autoencoder
UFLDL Tutorial Autoencoders: http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/
An MIT Press book Deep Learning Chapter14 Autoencoders:
https://www.deeplearningbook.org/contents/autoencoders.html
Building Autoencoders in Keras: https://blog.keras.io/building-autoencoders-in-keras.html
How Autoencoders work - Understanding the math and implementation:
https://www.kaggle.com/shivamb/how-autoencoders-work-intro-and-usecases

Image Retrieval with Autoencoder:
Build a simple Image Retrieval System with an Autoencoder: https://towardsdatascience.com/build-a-simple-image-retrieval-system-with-an-autoencoder-673a262b7921
Paper"Using Very Deep Autoencoders for Content-Based Image Retrieval": http://www.cs.toronto.edu/~fritz/absps/esann-deep-final.pdf
Keras Tutorial: Content Based Image Retrieval Using a Convolutional Denoising Autoencoder:
https://blog.sicara.com/keras-tutorial-content-based-image-retrieval-convolutional-denoising-autoencoder-dc91450cc511
Image Retrieval using autoencoders code: https://github.com/NarmadaBalasooriya/Image-Retrieval-using-autoencoders

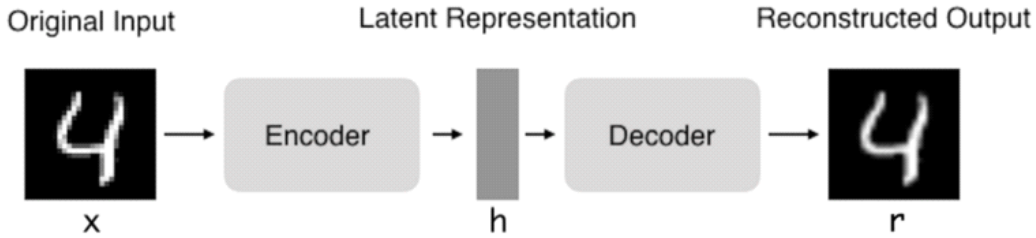
LSTM Autoencoders tutorial:
A Gentle Introduction to LSTM Autoencoders: https://machinelearningmastery.com/lstm-autoencoders/
Encoder-Decoder Long Short-Term Memory Networks: https://machinelearningmastery.com/encoder-decoder-long-short-term-memory-networks/

Paper "Unsupervised Learning of Video Representations using LSTMs": https://arxiv.org/pdf/1502.04681.pdf
Code "Unsupervised Learning of Video Representations using LSTMs":
https://github.com/mansimov/unsupervised-videos

Variational autoencoder tutorial:
Tutorial - What is a variational autoencoder?: https://jaan.io/what-is-variational-autoencoder-vae-tutorial/
Variational Autoencoders Explained: http://kvfrans.com/variational-autoencoders-explained/

Autoencoders(AE) are neural networks that aims to copy their inputs to their outputs. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation. This kind of network is composed of two parts:

1. **Encoder**: This is the part of the network that compresses the input into a laten-space representation. It can be represented by an encoding function **h = f(x)**
2. **Decoder**: This part aims to reconstruct the input from the latent space representation. It can be represented by a decoding function **r = g(h)**



The autoencoder as a whole can thus be described by the function **g(f(x)) = r** where you want **r** as close as the original input **x**.

Why copying the input to the output ?

If the only purpose of autoencoders was to copy the input to the output, they would be useless. Indeed, we hope that, by training the autoencoder to copy the input to the output, the lantent representation **h** will take on useful properties.

This can be achieved by creating constraints on the copying task. One way to obtain useful features from the autoencoder is to constrain h to have smaller dimensions than x, in this case the autoencoder is called undercomplete. By training an undercomplete representation, we force the autoencoder to learn the most salient features of the training data. If the autoencoder is given too much capacity, it can learn to perform the copying task without extracting any useful information about the distribution of the data. This can also occur if the dimension of the latent representation is the same as the input, and in the overcomplete case, where the dimension of the latent representation is greater than the input. In these cases, even a linear encoder and linear decoder can learn to copy the input to the output without learning anything useful about the data distribution. Ideally, one could train any architecture of autoencoder successfully, choosing the code dimension and the capacity of the encoder and decoder based on the complexity of distribution to be modeled.

What are autoencoders used for ?

Today **data denoising** and **dimensionality reduction** for data visualization are considered as two main interesting practical applications of autoencoders. With appropriate dimensionality and sparsity constraints, autoencoders can learn data projections that are more interesting than PCA or other basic techniques.

Autoencoders are learned automatically from data examples. It means that it is easy to train specialized instances of the algorithm that will perform well on a specific type of input and that it does not require any new engineering, only the appropriate training data.

However, autoencoders will do a poor job for image compression. As the autoencoder is trained on a given set of data, it will achieve reasonable compression results on data similar to the training set used but will be poor general-purpose image compressors. Compression techniques like JPEG will do vastly better.

Autoencoders are trained to preserve as much information as possible when an input is run through the encoder and then the decoder, but are also trained to make the new representation have various nice properties. Different kinds of autoencoders aim to achieve different kinds of properties. We will focus on four types on autoencoders.

Types of autoencoder :

In the article, the four following types of autoencoders will be described:

1. Vanilla autoencoder
2. Multilayer autoencoder
3. Convolutional autoencoder
4. Regularized autoencoder

Vanilla autoencoder
In its simplest form, the autoencoder is a three layers net, i.e. a neural net with one hidden layer. The input and output are the same, and we learn how to reconstruct the input, for example using the **adam optimizer** and the **mean squared error loss function**.

Here, we see that we have an **undercomplete autoencoder** as the hidden layer dimension(64) is smaller than the input(784). This constraint will impose our nerual net to learn a compressed representation of data.

Multilayer autoencoder
If one hidden layer is not enough, we can obviously extend the autoencoder to more hidden layers.

Now our implementation uses 3 hidden layers instead of just one. Any of the hidden layers can be picked as the feature representation but we will make the network symmetrical and use the middle-most layer.

Convolutional autoencoder
We may also ask ourselves: can autoencoders be used with Convolutions instead of Fully-connected layers ?

The answer is yes and the principle is the same, but using images (3D vectors) instead of flattened 1D vectors. The input image is downsampled to give a latent representation of smaller dimensions and force the autoencoder to learn a compressed version of the images.

Regularized autoencoder
There are other ways we can constraint the reconstruction of an autoencoder than to impose a hidden layer of smaller dimension than the input. 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. In practice, we usually find two types of regularized autoencoder: the sparse autoencoder and the denoising autoencoder.

Sparse autoencoder : 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. In this way, training to perform the copying task with a sparsity penalty can yield a model that has learned useful features as a byproduct.

Another way we can constraint the reconstruction of autoencoder is to impose a constraint in its loss. We could, for example, add a reguralization term in the loss function. Doing this will make our autoencoder learn sparse representation of data.

Notice in our hidden layer, we added an l1 activity regularizer, that will apply a penalty to the loss function during the optimization phase. As a result, the representation is now sparser compared to the vanilla

autoencoder.

Denoising autoencoder : Rather than adding a penalty to the loss function, we can obtain an autoencoder that learns something useful by changing the reconstruction error term of the loss function. This can be done by adding some noise of the input image and make the autoencoder learn to remove it. By this means, the encoder will extract the most important features and learn a robuster representation of the data.

Variational autoencoder(VAE)
Variational autoencoder models make strong assumptions concerning the distribution of latent variables. They use a variational approach for latent representation learning, which results in an additional loss component and a specific estimator for the training algorithm called the **Stochastic Gradient Variational Bayes(SGVB)** estimator. It assumes that the data is generated by a directed graphical model **p(x|z)** and that the encoder is learning an approximation **q(z|x)** to the posterior distribution **p(x|z)**. The objective of VAE has the following form:

$$\mathcal{L}(\phi, \theta, \mathbf{x}) = D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})) - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left(\log p_\theta(\mathbf{x}|\mathbf{z})\right)$$

Here, **DKL** stands for the **Kullback-Leibler divergence**. The prior over the latent variables is usually set to be the centred isotropic multivariate Gaussian p(z) = N(0,I); however, alternative configurations have been considered. Commonly, the shape of the variational and the likelihood distributions are chosen such that they are **factorized Gaussians**:

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\rho}(\mathbf{x}), \boldsymbol{\omega}^2(\mathbf{x})\mathbf{I}),$$
$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}(\mathbf{z}), \boldsymbol{\sigma}^2(\mathbf{z})\mathbf{I})),$$

This choice is justified by the simplifications that it produces when evaluating both the KL divergence and the likelihood term in variational objective defined above. VAE have been criticized because they generate blurry images. However, researchers employing this model were showing only the mean of the distributions, **u(z**), rather than a sample of the learned **Gaussian distribution**

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}(\mathbf{z}), \boldsymbol{\sigma}^2(\mathbf{z})\mathbf{I})).$$

These samples were shown to be overly noisy due to the choice of a factorized Gaussian distribution. Employing a Gaussian distribution with a full **covariance matrix**,

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}(\mathbf{z}), \boldsymbol{\Sigma}(\mathbf{z}))),$$

Could solve this issue, but is computationally intractable and numerically unstable, as it requires estimating a covariance matrix from a single data sample. However, later research showed that a restricted approach where the inverse matrix E-1(z) is sparse, could be tractably employed to generate images with high-frequency details.

Contractive autoencoder(CAE)
Contractive autoencoder adds an explicit regularizer in their objective function that forces the model to learn a function that is robust to slight variations of input values. This regularizer corresponds to the Frobenius norm of the Kacobian matrix of the encoder activations with respect to the input. The final objective function has the following form:

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') + \lambda \sum ||\nabla_x h_i||^2$$

LSTM Autoencoder(LAE)
An LSTM Autoencoder is an implementation of an autoencoder for sequence data using an Encoder-Decoder LSTM architecture. Once fit, the encoder part of the model can be used to encoder or compress sequence data that in turn may be used in **data visualizations** or as a feature vector input to a **supervised learning model**.

1. Autoencoders are a type of self-supervised learning model that can learn a compressed representation of input data.
2. LSTM Autoencoders can learn a compressed representation of sequence data and have been used on video, text, audio, and time series sequence data.

A Problem with Sequences: Sequence prediction problems are challenging, not least because the length of the input sequence can vary. This is challenging because machine learning algorithms, networks in particular, are designed to work with fixed length inputs. Another challenge with sequence data is that the temporal

ordering of the observations can make it challenging to extract features suitable for use as input to supervised learning models, often requiring deep expertise in the domain or in the field of signal processing. Many predictive modeling problems involving sequence require a prediction that itself is also a sequence. These are called sequence-to-sequence, or **seq2seq**, prediction problems.

Encoder-Decoder LSTM Models: **Recurrent neural networks**, such as the **Long Short-Term Memory**, or LSTM, network are specifically designed to support sequences of input data. They are capable of learning the complex dynamics within the temporal ordering of input sequences as well as use an internal memory to remember or use information across long input sequences. The LSTM network can be organized into an architecture called the **Encoder-Decoder LSTM** that allows the model to be used to both support variable length input sequences and to predict or output variable length output sequences. This architecture is the basis for many advances in complex sequence prediction problems such as **speech recognition** and **text translation**. In this architecture, an encoder LSTM model reads the input sequence step-by-step. After reading in the entire input sequence, the hidden state or output of this model represents an internal learned representation of the entire input sequence as a fixed-length vector. This vector is then provided as an input to the decoder model that interprets it as each step in the output sequence is generated.

What Is an LSTM Autoencoder: An LSTM Autoencoder is an implementation of an autoencoder for sequence data using an Encoder-Decoder LSTM architecture. For a given dataset of sequences, an encoder-decoder LSTM is configured to read the input sequence, encode it, decode it, and recreate it. The performance of the model is evaluated based on the model's ability to recreate the input sequence. Once the model achieves a desired level of performance recreating the sequence, the decoder part of the model may be removed, leaving just the encoder model. This model can then be used to encode input sequences to a fixed-length vector. The resulting vectors can then be used in a variety of applications, not least as a compressed representation of the sequence as an input to another supervised learning model.

Relationship with principal component analusis(PCA)
If linear activations are used, or only a single sigmoid hidden layer, then the optimal solution to an autoencoder is strongly related to principal component analysis(PCA). The weights of an autoencoder with a single hidden layer of size p(where p is less than the size of the input) span the same vector subspace as the one spanned by the first p principal components, and the output of the autoencoder is an orthogonal projection onto this subspace. The autoencoder weights are not equal to the principal components, and are generally not orthogonal, yet the principal components may be recovered from them using the singular value decomposition.

Summary
In this article, we went through the basic architecture of autoencoders. We also looked at many different types of autoencoders: vanilla, multilayer, convolutional and regularized. Each has different properties depending on the imposed constraints : either the reduced dimension of the hidden layers or another kind of penalty.