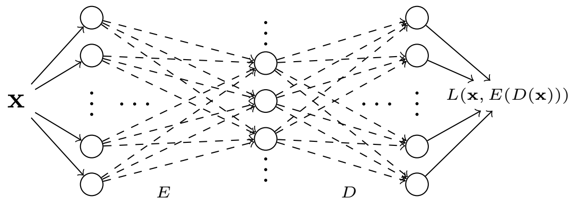
A Survey On Autoencoders

BILICI, M. Şafak
safakk.bilici.2112@gmail.com

***Abstract-* Autoencoders are an unsupervised learning architectures in neural networks. Theys are commonly used in Deep Learning tasks; such as generative models, anomaly detection, dimensionality reduction. In this article, we will evaluate theoretical approaches of Autoencoders and see it's extensions.**

1 Introduction

Autoencoders are an unsupervised learning method. They map the input data into lower dimensional space with encoder E, and then maps into same space that have same dimension of input data with decoder D.



The main idea behind Autoencoders is to attempt to copy its input to its output. The input layer is fed with input vector $\mathbf{x}$ and the loss is calculated at output layer between $\mathbf{x}$ and $E(D(\mathbf{x}))$, in other words the loss is $L(\mathbf{x}, E(D(\mathbf{x})))$. It measures difference between our original input and the consequent reconstruction. We named the middle layer, that is connection between encoder E and decoder D, as the "bottleneck". We can denote our output of bottleneck as $\mathbf{h} = E(\mathbf{x})$ and denote our output as $\hat{\mathbf{x}} = D(\mathbf{h}) = D(E(\mathbf{x}))$. We can define our encoder and decoder as conditional probability density function that are $p_{encoder}(\mathbf{h}|\mathbf{x})$ and $p_{decoder}(\hat{\mathbf{x}}|\mathbf{h})$.
The loss function is named reconstruction loss which is $L(\hat{\mathbf{x}}, \mathbf{x})$. We can treat the process as a feedforward networks; the loss can be minimized via mini-batch statistics following gradients computed by backpropagation algorithm,

$$\min_{\theta} L = \nabla_{\theta} L(\mathbf{x}, D(E(\mathbf{x}))) = \nabla_{\theta} L(\mathbf{x}, \hat{\mathbf{x}})$$

The bottleneck is the key of the effectiveness of Autoencoders. We map our input vector to bottleneck: the bottleneck keeps the 'latent informations' of input $\mathbf{x}$. The network represents input but in lower dimensions. In other words, it behaves like a approximative compression algorithm. The encoding parameters are learned in training process. Then we map bottleneck information $\mathbf{h}$ into same dimension as input $\mathbf{x}$. Then, this procedure can be seen as approximative extracting compressed latent information.

2 Undercomplete Autoencoders

The simplest idea behind autoencoders is the decreasing the number of nodes through the hidden layers before bottleneck. An autoencoder that has dimension less than the input $\mathbf{x}$ is called undercomplete autoencoder. When we minimize the reconstruction error, autoencoder learns to represent latent attributes of input data with lower dimensions than input $\mathbf{x}$'s. This procedure is same as in Principal Component Analysis (PCA) but in non-linear way. When decoder is linear and the loss $L(\hat{\mathbf{x}}, \mathbf{x})$ is the L^2 error, an autocomplete autoencoder learns to span the same subspace as PCA. When autoencoder has non-linear activations, then autoencoder becomes more powerful and generalized in dimensionality reduction, it becames non-linear version of PCA.

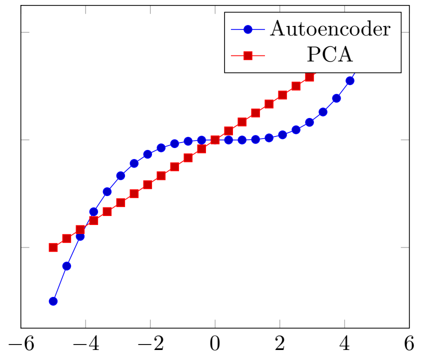


Figure 1: Encoding of PCA and Autoencoder.

2.1 Problem of Autoencoders

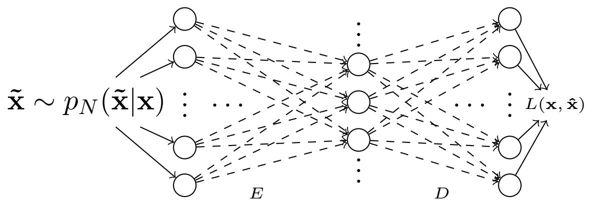
When we said that main idea behind autoencoders is to copy input to its output, the key idea is that not to copy without extracting useful informations about the distribution of the data. Autoencoders are allowed too much capacity, easy to be trained to the copying the task with learning anything useful about the dataset. So we need to penalize those autoencoders.

3 Regularizations

As we said, autoencoders are allowerd too much capacity. Regularized autoencoders can give us the task that find the latent features of input, instead of copying the input. There are lot of regularization methods to prevent copying task such as Sparse Autoencoders or Denoising Autoencoders.

3.1 Denoising Autoencoders

We can achieve the task that learning useful informations about data by adding some noise to input data.



$\tilde{\mathbf{x}} \sim p_N(\tilde{\mathbf{x}}|\mathbf{x})$
$L(\mathbf{x}, \hat{\mathbf{x}})$
E
D

In other words, we can achive it by changing the reconstruction loss. We defined our minimization,

$$\min_\theta L = \nabla_\theta L(\mathbf{x}, D(E(\mathbf{x}))) = \nabla_\theta L(\mathbf{x}, \hat{\mathbf{x}})$$

To perform the denoising, the input $\mathbf{x}$ is corrupted into $\tilde{\mathbf{x}}$ through stochastic mapping of $\tilde{\mathbf{x}} \sim p_N(\tilde{\mathbf{x}}|x)$. Then the noisy (corrupted) input is used for encoding and decoding parts

$$\mathbf{h} = E(\tilde{\mathbf{x}})$$

$$\hat{\mathbf{x}} = D(\mathbf{h}) = D(E(\tilde{\mathbf{x}})_{\tilde{\mathbf{x}} \sim p_N(\tilde{\mathbf{x}}|x)})$$

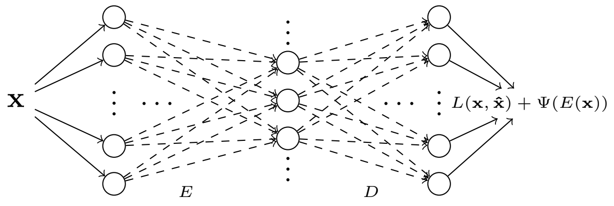
Then the minimization task is updated as

$$\min_\theta L = \nabla_\theta L(\mathbf{x}, D(E(\tilde{\mathbf{x}})_{\tilde{\mathbf{x}} \sim p_N(\tilde{\mathbf{x}}|x)})) = \nabla_\theta L(\mathbf{x}, \hat{\mathbf{x}})$$

The denoising can be seen as forcing our model to learn latent features of our input by adding noise to input then penalizing with reconstruction loss $L(\mathbf{x}, \hat{\mathbf{x}})$.

Sparse Autoencoders

The sparsity simply comes from adding a shrinkage method to reconstruction loss like in machine learning tasks.



The minimization will be

$$\min_\theta L = \nabla_\theta L(\mathbf{x}, D(E(\mathbf{x}))) + \underbrace{\Psi(E(\mathbf{x}))}_{sparisty} = \nabla_\theta L(\mathbf{x}, \hat{\mathbf{x}})$$

This procedure can be interpreted as Bayesian inference with terms of posterior, likelihood and prior: *posterior* $\propto$ *likelihood* $\times$ *prior*. From this point maximizing the likelihood is equivalent to maximizing the posterior

$$\max_\theta p(\theta, \mathbf{x}) = \max_\theta p(\mathbf{x}, \theta) \cdot p(\theta)$$

$$= \max_\theta \log p(\mathbf{x}|\theta) + \log p(\theta)$$

Now we can write our joint distribution in terms of prior of $\mathbf{h}$ and it's factor seeing $\mathbf{x}$

$$p_{model}(\mathbf{x}, \mathbf{h}) = p_{model}(\mathbf{x}|h) \cdot p_{model}(\mathbf{h})$$

From this point, we can rewrite our likelihood and propose a prior distribution.

$$\log p_{model}(x, h) = \log p_{model}(\mathbf{x}|\mathbf{h}) \cdot p_{model}(\mathbf{h})$$

For simplicity, let us consider a zero-mean Laplacean prior

$$Lap(h_i|\mu = 0, \lambda) = \frac{\lambda}{2}\exp(-\lambda|h_i - \mu|)$$

$$= \frac{\lambda}{2}\exp(-\lambda|h_i|)$$

Then the posterior becomes

$$\prod p_{model}(\mathbf{x}|h) \cdot \prod p_{model}(h)$$

$$= \sum \log p_{model}(\mathbf{x}|h) + \sum \log p_{model}(h)$$

$$= \sum \log p_{model}(\mathbf{x}|h) - \lambda \sum |h|$$

Other prior distributions like Student-t, Gaussian (L_1) can make an impact for sparsity.