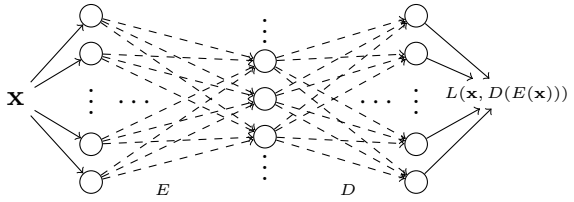
A Survey On Autoencoders

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

***Abstract*- Autoencoders are an unsupervised learning architectures in neural networks. They 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,

$$\min_{\theta} L = \nabla_{\theta} L(\mathbf{x}, D(E(\mathbf{x}))) = \nabla_{\theta} L(\mathbf{x}, \hat{\mathbf{x}}) \quad (1)$$

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 [1]. When autoencoder has non-linear activations, then autoencoder becomes more powerful and generalized in dimen-

sionality reduction, it becames non-linear version of PCA.

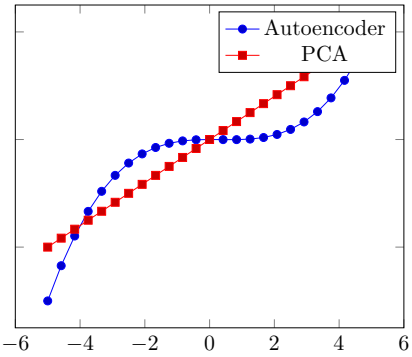


Figure 1: Encoding of PCA and Autoencoder.

2.1 Problem of Autoencoders

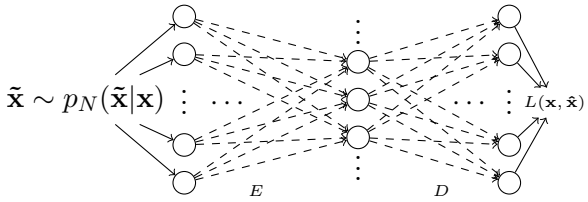
When we said that main idea behind autoencoders is to copy input to its output, the key idea is that not to copy without extracting useful informations about the distribution of the data [4]. Autoencoders are allowed too much capacity, easy to be trained to the copying the task with learning anything 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.



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

$$\min_{\theta} L = \nabla_\theta L(\mathbf{x}, D(E(\mathbf{x}))) = \nabla_\theta L(\mathbf{x}, \hat{\mathbf{x}}) \quad (2)$$

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

$$\mathbf{h} = E(\tilde{\mathbf{x}}) \quad (3)$$

$$\hat{\mathbf{x}} = D(\mathbf{h}) = D(E(\tilde{\mathbf{x}})_{\tilde{\mathbf{x}} \sim p_N(\tilde{\mathbf{x}}|\mathbf{x})}) \quad (4)$$

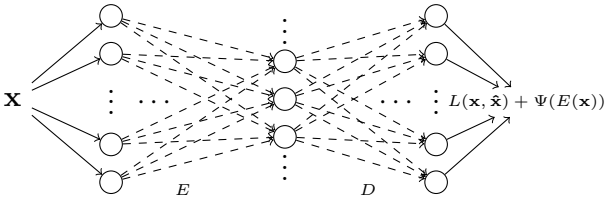
Then the minimization task is updated as

$$\min_{\theta} L = \nabla_\theta L(\mathbf{x}, D(E(\tilde{\mathbf{x}})_{\tilde{\mathbf{x}} \sim p_N(\tilde{\mathbf{x}}|x)})) = \nabla_\theta L(\mathbf{x}, \hat{\mathbf{x}}) \quad (5)$$

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 $L(\mathbf{x}, \hat{\mathbf{x}})$ [7].

3.2 Sparse Autoencoders

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



The minimization will be

$$\min_{\theta} L = \nabla_\theta L(\mathbf{x}, D(E(\mathbf{x}))) + \underbrace{\Psi(E(\mathbf{x}))}_{sparisty} = \nabla_\theta L(\mathbf{x}, \hat{\mathbf{x}}) \quad (6)$$

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

$$p(\theta|\mathbf{x}) \propto p(\mathbf{x}|\theta) \cdot p(\theta) \quad (7)$$

$$\max_{\theta} p(\theta|\mathbf{x}) = \max_{\theta} \left(\log p(\mathbf{x}|\theta) + \log p(\theta)\right) \quad (8)$$

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

$$p_{model}(\mathbf{x}, \mathbf{h}) = p_{model}(\mathbf{x}|\mathbf{h}) \cdot p_{model}(\mathbf{h}) \quad (9)$$

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

$$\log p_{model}(\mathbf{x}, \mathbf{h}) = \log p_{model}(\mathbf{x}|\mathbf{h}) + \log p_{model}(\mathbf{h}) \tag{10}$$

For simplicity, let us consider a zero-mean Laplacean prior [4]

$$Lap(h_i|\mu = 0, \lambda) = \frac{\lambda}{2} \exp(-\lambda|h_i - \mu|) \tag{11}$$

$$= \frac{\lambda}{2} \exp(-\lambda|h_i|) \tag{12}$$

Then the posterior becomes

$$\prod_i p_{model}(x|h) \cdot \prod_i p_{model}(h) \tag{13}$$

$$= \sum_i \log p_{model}(x|h) + \sum_i \log p_{model}(h) \tag{14}$$

$$= \sum_i \log p_{model}(\mathbf{x}|h) - \lambda \sum_i |h| \tag{15}$$

Other prior distributions like Student-t or Gaussian (L_1) can make an impact for sparsity [4].

4 Variational Autoencoders

Variational autoencoders (VAEs) are a deep learning technique for learning latent representations. Variational Autoencoders also can be used for generative tasks, they let us design complex generative models of data, and fit them to large datasets [2]. Variational Autoencoders perform efficient approximate inference and learning with directed probabilistic models whose continuous latent variables and/or parameters have intractable prior distributions [5].

4.1 Latent Space Models

Let's define our prior distribution. Our dataset is sampled form $x \sim p(X)$ and the latent variable is sampled from $z \sim p(Z)$. If we define our deterministic neural network as $f : Z \times \Theta \to X$ [3], (Θ is our parameter space), then $p(X)$ will be

$$p(X) = \int p(X|z;\theta)p(z)dz \tag{16}$$

where $p(X|z;\theta)$ is another form of our deterministic neural network function.

4.2 Setting Up The Objective

Let's define $q_\phi(z|x)$ as our probabilistic encoder. It represents, the probability of z when x is known. Second, $p_\theta(x|z)$ as our probabilistic decoder. It represents, the probability of x when z is known [5]. The motivation behind Variational Autoencoders is to produce a output distribution which is equivalent (good reconstructing) to the input distribution. This task can be achieved with Kullback-Leibler Divergence defined as

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} \tag{17}$$

$$= \mathbb{E}[\log p(x) - \log q(x)] \tag{18}$$

The output distribution has a prior on z, which makes our true posterior as

$$p(z|x) = \frac{p(x|z)p(z)}{\int p(x|z)p(z)dz} = \frac{p(x|z)p(z)}{p(x)} \tag{19}$$

$$\log p(z|x) = \log p(x|z) + \log p(z) - \log p(x) \tag{20}$$

From this point of view KL-Divergence of the approximate from the true posterior [5] will be defined as

$$D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)})) \tag{21}$$

$$= \mathbb{E}_{q_\phi(z|x)}[\log q_\phi(z|x^{(i)}) - \log p_\theta(z|x^{(i)})] \tag{22}$$

$$= \mathbb{E}_{q_\phi(z|x)}[\log q_\phi(z|x^{(i)}) - \log p(x^{(i)}|z) - \log p(z)] + \log p_\theta(x^{(i)}) \tag{23}$$

Putting Equation 21 and Equation 23 together simplifies,

$$\log p_\theta(x^{(i)}) - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)})) \tag{24}$$

$$= D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z)) - \mathbb{E}_{q_\phi(z|x^{(i)})}[\log p_\theta(x^{(i)}|z)] \tag{25}$$

Where Equation 25 references variational lower bound $\mathcal{L}(\theta, \phi; x^{(i)})$ [5] [6] which is $\log p(x^{(i)}) \geq \mathcal{L}(\theta, \phi; x^{(i)})$. We want to differentiate and optimize this lower bound both on decoder parameters θ and encoder parameters ϕ.
Therefore, we have 2 objective to deal with. This two objectives can be written as

$$\mathbb{E}_X[\log p_\theta(x^{(i)}) - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)}))] \tag{26}$$

$$= \mathbb{E}_X[D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z)) - \mathbb{E}_{q_\phi(z|x^{(i)})}[\log p_\theta(x^{(i)}|z)]] \quad (27)$$

When we take the gradient of this Equation 27, gradient moves into the expectations. This means, we can average the gradient of Equation 24 over stochastically choosen samples of X and z, and the result converges to the gradient of Equation 26-27 [3].

The first one from Equation 27 is the KL-Divergence

$$D_{KL}(\mathcal{N}(\mu(\mathbf{x}), \Sigma(\mathbf{x}))||\mathcal{N}(0, I)) \quad (28)$$

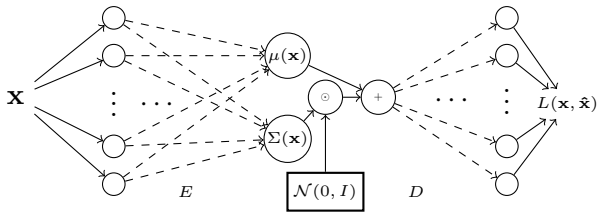
where $\mathcal{N}(\mu(\mathbf{x}), \Sigma(\mathbf{x}))$ corresponds to $q_\phi(z|x^{(i)})$. The latter objective is

$$\mathbb{E}_{q_\phi(z|x^{(i)})}[\log p_\theta(x^{(i)}|z)] \quad (29)$$

which is proportional to Gaussian distribution. The problem is $\mathbb{E}_{q_\phi(z|x^{(i)})}[\log p_\theta(x^{(i)}|z)]$ is dependent on both parameters ϕ and θ. We need to backpropagate the error through a layer that samples z from $q_\phi(z|x^{(i)})$ which is a non-continuous operation and has no gradient [3]. So, what we did in Equation 28 is solution for that, re-parameterization trick. The Equation 27 becomes

$$\mathbb{E}_X[D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z)) - \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[\log p_\theta(x^{(i)}|z = \mu(\mathbf{x}) + \Sigma^{\frac{1}{2}}(\mathbf{x}) * \epsilon)]] \quad (30)$$

This simple re-parameterization trick can be visualized:



Since the sampling operator doesn't have a gradient, this procedure can be seen like encoding the input data with its mean and standard deviation representations intuitively. Now we can backpropagate through the mean and standard deviation, and learn them.

The next thing we need to do is solving the KL-Divergence term, since we know the term $\mathbb{E}_{q_\phi(z|x^{(i)})}[\log p_\theta(x^{(i)}|z)]$ is proportional to Gaussian Distribution.

4.2.1 Setting KL-Divergence

In our problem, KL-Divergence is simply in Gaussian case [5]. Set $p_\theta(z) = \mathcal{N}(0, I)$ and $q_\phi(z|x^{(i)})$ Gaussian, $z \in \mathbb{R}^K$, $\boldsymbol{\mu}$ is variational mean and $\boldsymbol{\sigma}$ is variational standard deviation. The KL-Divergence can be found with subtraction of two integrals in closed form,

$$D_{KL}(q_\phi(z)||p_\theta(z)) \quad (31)$$

$$= -\int q_\theta(z) \log p_\phi(z) dz + \int q_\theta(z) \log q_\theta(z) dz \quad (32)$$

Where

$$\int q_\phi(z) \log p_\theta(z) dz \quad (33)$$

$$= \int \mathcal{N}(z; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(z; 0, \boldsymbol{I}) \quad (34)$$

$$= -\frac{K}{2} \log(2\pi) - \frac{1}{2} \sum_{k=1}^{K} (\mu_k^2 + \sigma_k^2) \quad (35)$$

and

$$\int q_\phi(z) \log p_\phi(z) dz \quad (36)$$

$$= \int \mathcal{N}(z; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(z; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) dz \quad (37)$$

$$= -\frac{K}{2} \log(2\pi) - \frac{1}{2} \sum_{k=1}^{K} (1 + \log \sigma_j^2) \quad (38)$$

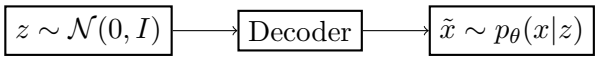
To sum up, the KL-Divergence has the form

$$D_{KL}(q_\phi(z)||p_\theta(z)) \quad (39)$$

$$= \int q_\theta(z)(\log q_\theta(z) - \log p_\phi(z)) dz \quad (40)$$

$$= \frac{1}{2} \sum_{j=1}^{J} \left(1 + \log \sigma_j^2 - \mu_j^2 - \sigma_j^2\right) \quad (41)$$

4.3 Inference Time In VAE



Since we showed the training time with last encoder-decoder figure, test time a little bit different from train time forward equation. The main idea behind the Variational Autoencoders is at the train time, the probabilistic encoder encodes data into two representative variable: the mean and s.d. We sample from this mean and s.d and forward it to probabilistic decoder.

Since the sampling hasn't got gradient operator, we defined a re-parameterization technique. The KL-Divergence in the loss is the trick about VAEs: to explain in intuitive way, we enforce our dataset's distribution into a zero mean, $\boldsymbol{I}$ s.d Gaussian distribution. By doing that, at the test time, we can generate new samples from a single Gaussian noise!

References

[1] Charu C. Aggarwal. *Neural Networks and Deep Learning*. Berlin, Heidelberg: Springer International Publishing, 2018. ISBN: 978-3-319-94463-0.

[2] Jaan Altosaar. *Understanding Variational Autoencoders from two perspectives: deep learning and graphical models*. `https://jaan.io/what-is-variational-autoencoder-vae-tutorial/`.

[3] Carl Doersch. *Tutorial on Variational Autoencoders*. 2016. arXiv: `1606.05908 [stat.ML]`.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[5] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: `1312.6114 [stat.ML]`.

[6] *The variational auto-encoder*. `https://ermongroup.github.io/cs228-notes/extras/vae/`.

[7] Pascal Vincent et al. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion". In: *J. Mach. Learn. Res.* 11 (Dec. 2010), pp. 3371–3408. ISSN: 1532-4435.