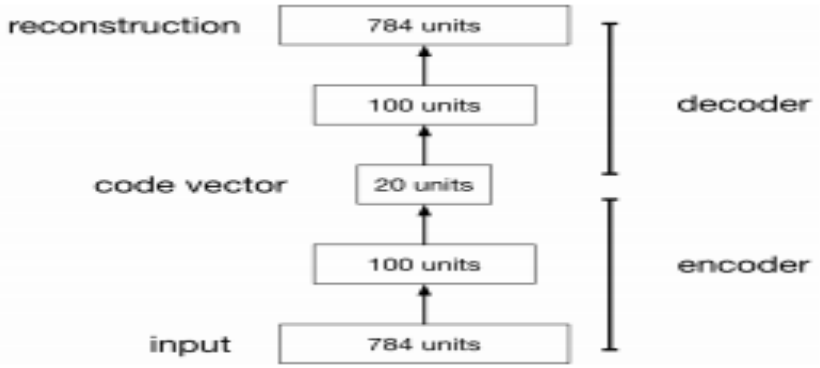
Autoencoders

Abhinav Angirekula and Alvan Arulandu

May 2020

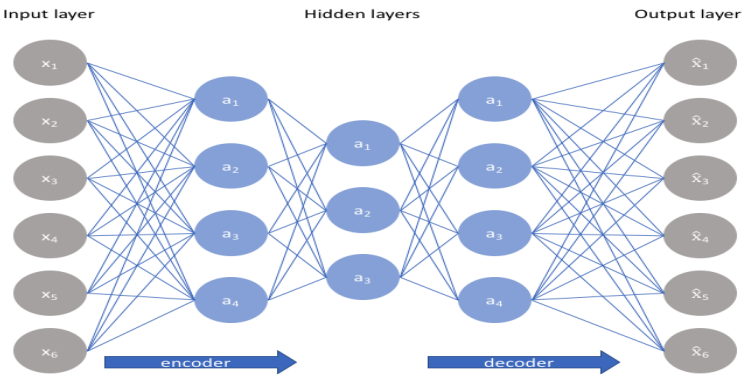
1 Introduction

An autoencoder, AE, is a neural network that trains with the intent of learning a compressed form of the input data. Basic autoencoders are trained using the same input and labels. However, this varies with the type of autoencoder. It is a type of artificial neural network used for feature learning without needing to use labels or supervised learning. However, this is what the reduction side learns, as the reconstructing side tries to learn how to generate a representation close to the original input using the reduced encoding.



2 Autoencoders

An autencoder is comprised of three main parts: the encoder(reducer), the decoder(reconstructor), and the bottleneck(latent space vector). A basic autoencoder includes a layer of fully connected neurons for each previously mentioned component. Since the output is simply the reconstructed input, the output layer has the same dimensions as the input layer.



The goal of an autoencoder is to learn to create a representation that minimizes the reconstruction loss, thus learning how to create an efficient lower-dimensional representation of the data. However, this comes with some caveats. If we simply wanted to have zero reconstruction loss, we could have our autencoder simply output the exact same image inputted. But this would mean that the bottleneck layer would not learn anything. To avoid this, it is common to constrain the hidden layer.

The most common way to do this is by constraining size: If the number of neurons in the hidden layer is less than the number of neurons in the input layer, the autencoder is dubbed "undercomplete." This guarantees that it will learn how to create a representation with less neurons, so we know that the autoencoder will learn something.

1. Encoding an image - An input image is received and its size is condensed into a smaller vector. This reduced vector represents the features of the image and can be utilized to reconstruct another image.
2. Decoding an image - Now, we simply return the image to its original size.

2.1 Loss Functions

Using the image above, we need to define a loss function that compares x_k with $\hat{x}_k$ to measure the effectiveness of our reconstruction. Using said loss function, we can use gradient descent in order to minimize it. For binary inputs, it is most popular to use a cross-entropy loss function:

$$l(f(x)) = -\sum_k (x_k log(\hat{x}_k) + (1 - x_k) log(1 - \hat{x}_k))$$

The loss function described above is minimized if x_k is equal to $\hat{x}_k$ for all values of k. For real-value inputs, it is common to squared euclidean distance (MSE):

$$l(f(x)) = \frac{1}{2}\sum_k (\hat{x}_k - x_k)^2$$

Usually, you'll use a linear activation function in tandem with this.

2.2 Disadvantages

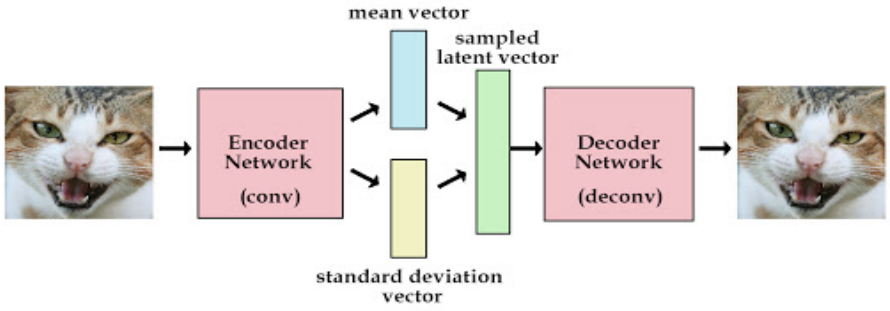
Unfortunately, compared to Generative Adversarial Networks, autoencoders are less efficient in the reconstruction of images. As the images inputted increase in complexity, the autoencoder's reconstructions become blurrier and blurrier, often due to the encoder's inability to properly create a reduced representation that captures enough hidden features due to the aforementioned complexity of the original image. Additionaly, autoencoders may end up reconstructing very similar data despite the code vector if the encoder is not trained properly.

2.3 Regularized Autoencoders

Regularized autoencoders are autoencoders in which the input and output are not the same. These types of autoencoders have been commonly used in image-denoising, neural inpainting, and image compression/decompression algorithms.

3 Variational Autoencoders

Variational autoencoders, VAEs, are a subset of autoencoders that better address the pitfalls mentioned above with respect to training. Instead of immediately passing the encoder's output to the decoder, these autoencoders utilize a random sampling from a distribution of the encoder's output.



One step in a VAE's training process:
1. An input vector is fed to the encoder.
2. The encoder outputs a latent space vector.
3. The latent space vector is used to calculate mean and standard deviation vectors, with the same dimensionality.
4. The vectors are randomly sampled from, yielding a random N-dimensional vector.
5. This vector is then fed into the decoder that attempts to reconstruct the initial input vector.

3.1 VAE Loss

For the following calculations let x be the input to the encoder, z be the sampled latent vector.

$$L(\theta, \phi; x, z) = E_{q\phi(z|x)}[log_{p\theta}(x|z)] - D_{KL}(q_\theta(z|x)||p(z))$$

Net Loss: $L(\theta, \phi; x, z)$
Reconstruction Loss: $E_{q\phi(z|x)}[log_{p\theta}(x|z)]$
KL-Divergence Loss (maintaining normal distribution): $D_{KL}(q_\theta(z|x)||p(z))$

This loss can be easier estimated via a Monte Carlo estimate.

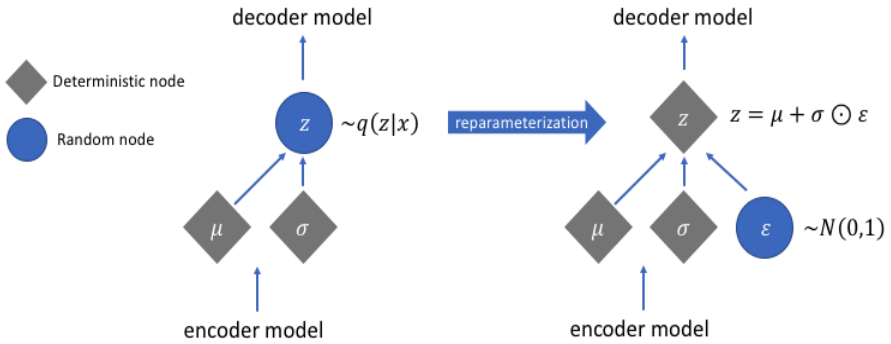
$$ELBO = log(p(x|z)) + log(p(z)) - log(p(z|x))$$

Loss of the latent vector given the input: $log(p(x|z))$
Loss of the sampled latent vector from the mean and standard deviation: $log(p(z))$
Loss of the final output given the sampled latent vector: $log(p(z|x))$

During the training process, a VAE attempts to maximize the ELBO, evidence lower bound. For more information on VAE loss see [1] and [3].

3.2 Reparameterization Trick

Random sampling, however, impedes the process of stochastic gradient descent. Due to the random sampling of a vector from the latent space, back-propagation is not able to pass from the decoder to the encoder. This issue is resolved using the reparameterization trick.



Reparameterization breaks up the single stochastic node of random sampling into multiple parts. Let a random sample be generated as follows: $Z = \mu + \sigma \cdot \epsilon$. Note that μ and σ are learned variables and are capable of running back propagation. However, ϵ is simply a sample from a normal distribution. So, the reparameterization trick cleverly breaks apart the sampling

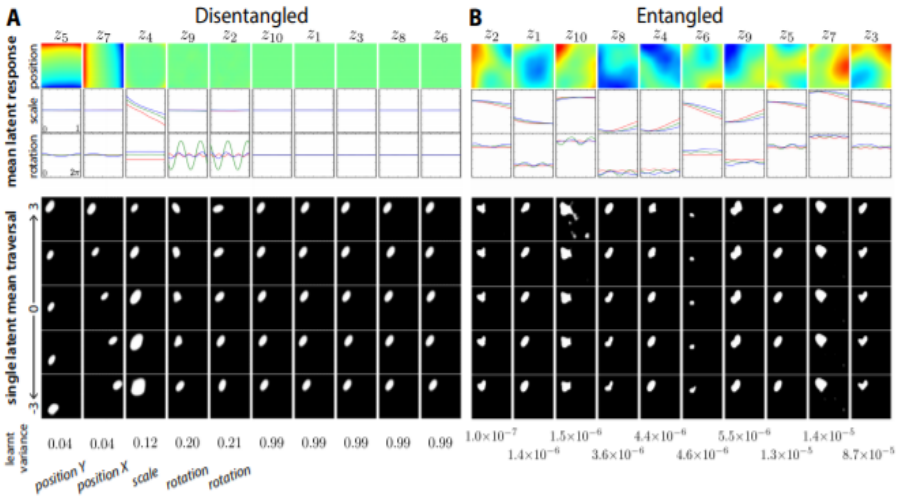
operation into multiple parts so that the ϵ node is the only stochastic node. This allows back propagation to flow from the decoder back to the encoder. For more information see [1].

3.3 Disentangled variational autoencoders

Disentagled variational autoencoders are a subset within variational autoencoders that provide better control over random sampling. These networks do this by using a hyperparameter, β, to manipulate the effect of KL loss on the overall loss. For the following calculations let x be the input to the encoder, z be the sampled latent vector.

Disentangled KL-Divergence Loss: $\beta D_{KL}(q_\theta(z|x)||p(z))$

Please note that the value of β heavily impacts the performance of the autoencoder. See [2] for more information on disentangled implementation.

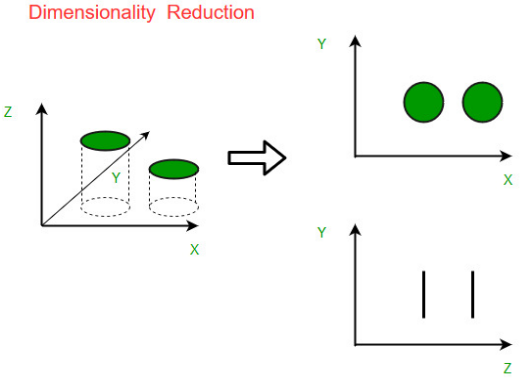


4 Applications

Autoencoders can be used for a plethora of endeavors, such as noise removal, image colorization, etc. For example, inputting a noise-ridden image, the autoencoder learns the hidden/latent features of the image to reconstruct an image without noise. In such an example, we can establish the reconstruction error as the difference between the pixel values of the "true" image and the output image. However, most of these autoencoders use multiple convolutional layers instead of just fully connected one.

Other applications include the following:

1. Dimensionality Reduction:

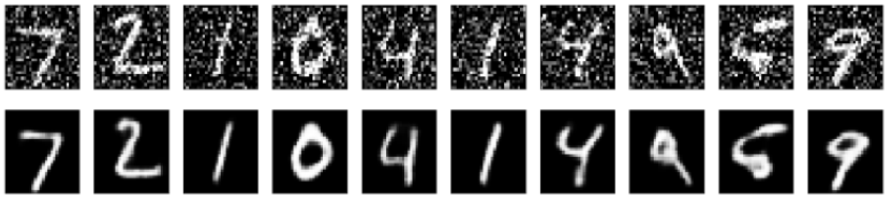


Dimensionality reduction can be used to make classification tasks less cumbersome by decreasing the number of features.

2. Image Compression/Decompression:

Image compression is the process of reducing the size of a graphics image file without resorting to reducing the quality of said image to an intolerable level.

3. Image Denoising:

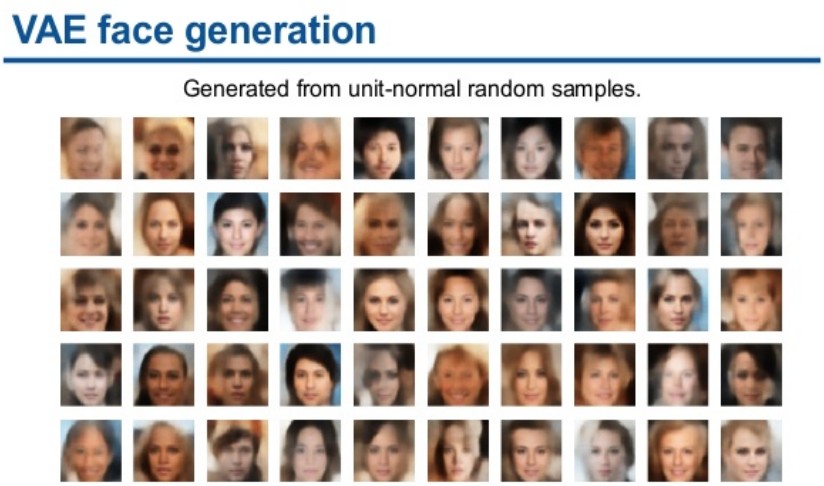


Amongst the fundamental challenges in the world of computer vision, the process of image denoising is one that entails the removal or minimizing of noise in a noise-ridden image.

4. Image Coloration:

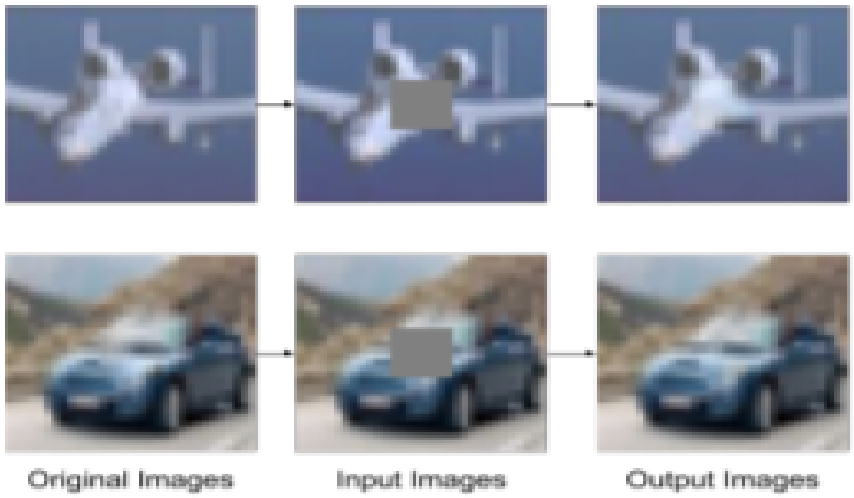
The goal of image coloration is to receive an image that is in black and white, or perhaps one that is lacking in proper coloring in some other manner, and to color it in such a way that would be realistic and reminiscent of what the actual subject depicted in the image would look like if not for the distinct lack of any hues.

5. Image Generation:



Above are examples of images of faces that were generated via the implementation of a variational autoencoder.

6. Neural Inpainting:



Nerual Inpainting is a partial image generation technique. This process aims to teach the autoencoder to use features found in the surrounding sections of the image to reconstruct the missing portion.

5 Works Cited

[1]https://www.tensorflow.org/tutorials/generative/cvae
[2]https://arxiv.org/abs/1606.05579
[3]https://arxiv.org/abs/1312.6114
[4]https://www.cs.mcgill.ca/ jpineau/comp551/Lectures/16DeepLearning-share.pdf