PCA, Autoencoders, and Variational Autoencoders

Ankit Pratap Singh

Iowa State University

What is PCA?

PCA is an unsupervised learning technique that is used for dimension reduction of a complex data set to reveal sometimes hidden/simplified structures that often underlie it.

Motivation [Jonathon Shlens, 2014][1]

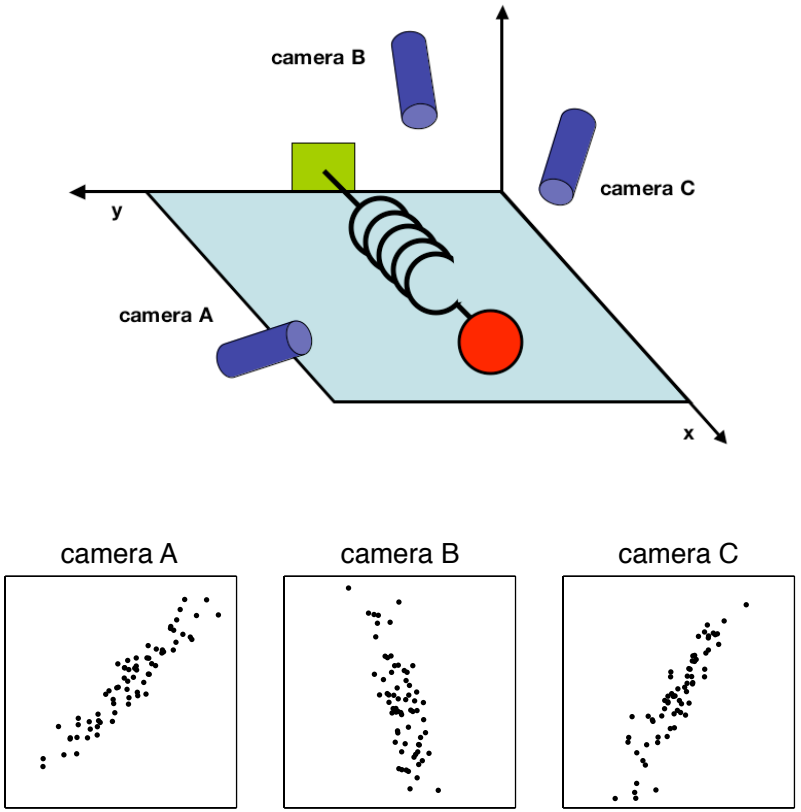

Figure 1: A toy example. The position of a ball attached to an oscillating spring is recorded using three cameras A, B and C. The position of the ball tracked by each camera is depicted in each panel below.

[1] Jonathon Shlens, A Tutorial on Principal Component Analysis

Change of Basis

$\mathbf{X}$ is $d \times n$ data matrix. Let $\mathbf{Y}$ be another $d \times n$ matrix related by a linear transformation $\mathbf{P}$.

$$\mathbf{PX} = \mathbf{Y}$$

Geometrically, $\mathbf{P}$ is a rotation and a stretch which transforms $\mathbf{X}$ into $\mathbf{Y}$.

$$\mathbf{y}_i = \begin{bmatrix} \mathbf{p_1} \cdot \mathbf{x_i} \\ \vdots \\ \mathbf{p_d} \cdot \mathbf{x_i} \end{bmatrix}$$

$\mathbf{y_i}$ is a projection on to the basis of $\{\mathbf{p_1}, \ldots, \mathbf{p_d}\}$

Good Choice of Basis **P**

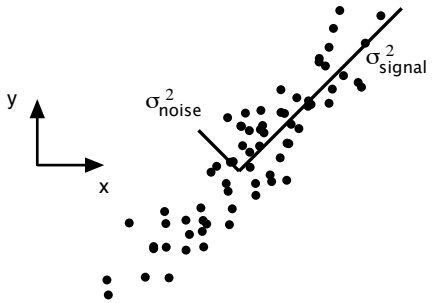

Figure 2: Simulated data of (x, y) for camera A. The signal and noise variances σ^2_{signal} and σ^2_{noise} are graphically represented by the two lines subtending the cloud of data. Note that the largest direction of variance does not lie along the basis of the recording (x_A, y_A) but rather along the best-fit line.

The basis for which we are searching is not the naive basis because these directions (i.e. (x_A, y_A)) do not correspond to the directions of largest variance.

Redundancy

In case of multiple sensors record the same dynamic information.

Covariance Matrix

Assume zero mean

$$\mathbf{C_X} \equiv \frac{1}{n}\mathbf{X}\mathbf{X}^T.$$

- All off-diagonal terms in $\mathbf{C_Y}$ should be zero. Thus, $\mathbf{C_Y}$ must be a diagonal matrix. Or, said another way, $\mathbf{Y}$ is decorrelated.
- Each successive dimension in $\mathbf{Y}$ should be rank-ordered according to variance.

Find some orthonormal matrix $\mathbf{P}$ in $\mathbf{Y} = \mathbf{PX}$ such that $\mathbf{C_Y} \equiv \frac{1}{n}\mathbf{YY}^T$ is a diagonal matrix. The rows of $\mathbf{P}$ are the *principal components* of $\mathbf{X}$.

Limitation

The goal of the analysis is to decorrelate the data, or said in other terms, the goal is to remove second-order dependencies in the data. If higher order dependencies exist between the variables removing second-order dependencies is insufficient at revealing all structure in the data.

When are second order dependencies sufficient for revealing all dependencies in a data set? This statistical condition is met when the first and second order statistics are *sufficient statistics* of the data. This occurs, for instance, when a data set is Gaussian distributed.

Covariance Estimation [Roman Vershynin, 2020][2]

With probability at least $1 - 2\exp(-d)$,

$$\|\hat{\Sigma} - \Sigma^*\| \leq CK^2\sqrt{\frac{d}{n}}\|\Sigma^*\|$$

Here K is the maximum sub-Gaussian norm of $\Sigma^{*-1/2}X_i$

[2] Roman Vershynin, High-Dimensional Probability

Davis Kahan $\sin\Theta$ theorem

Let Σ^*, $\hat{\Sigma}$ be $d \times d$ symmetric matrices with $\mathbf{U}^* \in \Re^{d \times r}$, $\mathbf{U} \in \Re^{d \times r}$ being the matrices of top r eigen vectors of Σ^*, $\hat{\Sigma}$ respectively. Let $\sigma_1^* \geq ... \geq \sigma_d^*$ be the eigen values of Σ^*. If $\sigma_r^* - \sigma_{r+1}^* > 0$ and $\|\hat{\Sigma} - \Sigma^*\| \leq (1 - \frac{1}{\sqrt{2}})(\sigma_r^* - \sigma_{r+1}^*)$ then

$$\mathbf{SD}_F(\mathbf{U}, \mathbf{U}^*) \leq \frac{2\sqrt{r}\|\hat{\Sigma} - \Sigma^*\|}{\sigma_r^* - \sigma_{r+1}^*}$$

Autoencoder

An autoencoder is a type of algorithm with the primary purpose of learning an "informative" representation of the data that can be used for different applications[3] by learning to reconstruct a set of input observations well enough.

[3] Bank, D., Koenigstein, N., and Giryes, R., Autoencoders, https://arxiv.org/abs/2003.05991

Three main components

1. Encoder
2. Latent feature representation
3. Decoder

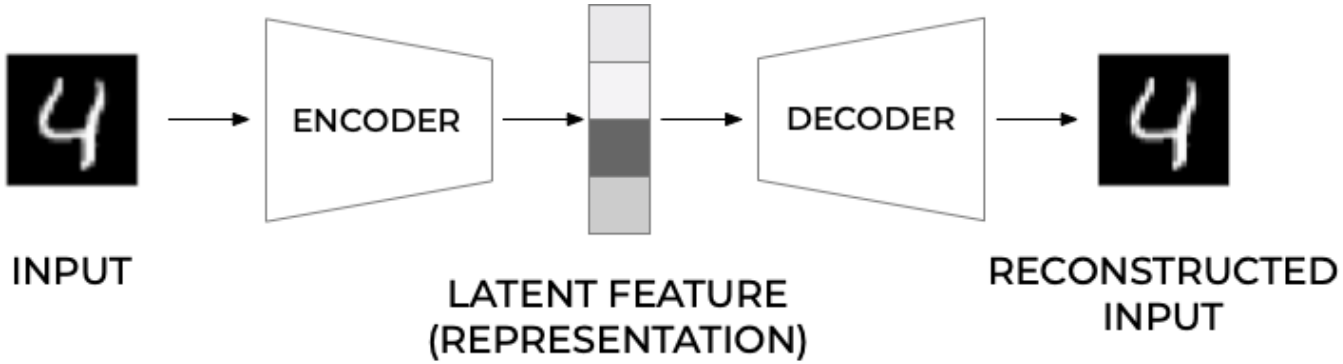

Figure 3: General structure of an autoencoder.

Latent representation can be very useful for various tasks.

1. Feature extraction
2. Classification or Clustering
3. Dimensionality reduction

In most typical architectures, the encoder and the decoder are neural networks

Encoder

The encoder can be written as a function g that will depend on some parameters

$$\mathbf{h}_i = g(\mathbf{x}_i) \tag{1}$$

Where $\mathbf{h}_i \in \mathbb{R}^q$ (the latent feature representation) is the output of the **encoder** when we evaluate it on the input $\mathbf{x}_i$.
Note that we will have $g : \mathbb{R}^n \to \mathbb{R}^q$.

Decoder

The **decoder** (and the output of the network that we will indicate with $\tilde{\mathbf{x}}_i$) can be written then as a second generic function f of the latent features

$$\tilde{\mathbf{x}}_i = f\left(\mathbf{h}_i\right) = f\left(g\left(\mathbf{x}_i\right)\right). \tag{2}$$

Where $\tilde{\mathbf{x}}_i \in \mathbb{R}^n$.

Training

Training an autoencoder simply means finding the functions $g(\cdot)$ and $f(\cdot)$ that satisfy

$$\arg\min_{f,g} < [\Delta(\mathbf{x}_i, f(g(\mathbf{x}_i))] > \tag{3}$$

where Δ indicates a measure of how the input and the output of the autoencoder differ (basically our loss function will penalize the difference between input and output) and $< \cdot >$ indicates the average over all observations.

Bottleneck

We want the autoencoder to reconstruct the input well enough. Still, at the same time, it should create a latent representation (the output of the encoder) that is useful and meaningful.

Feed Forward Autoencoder

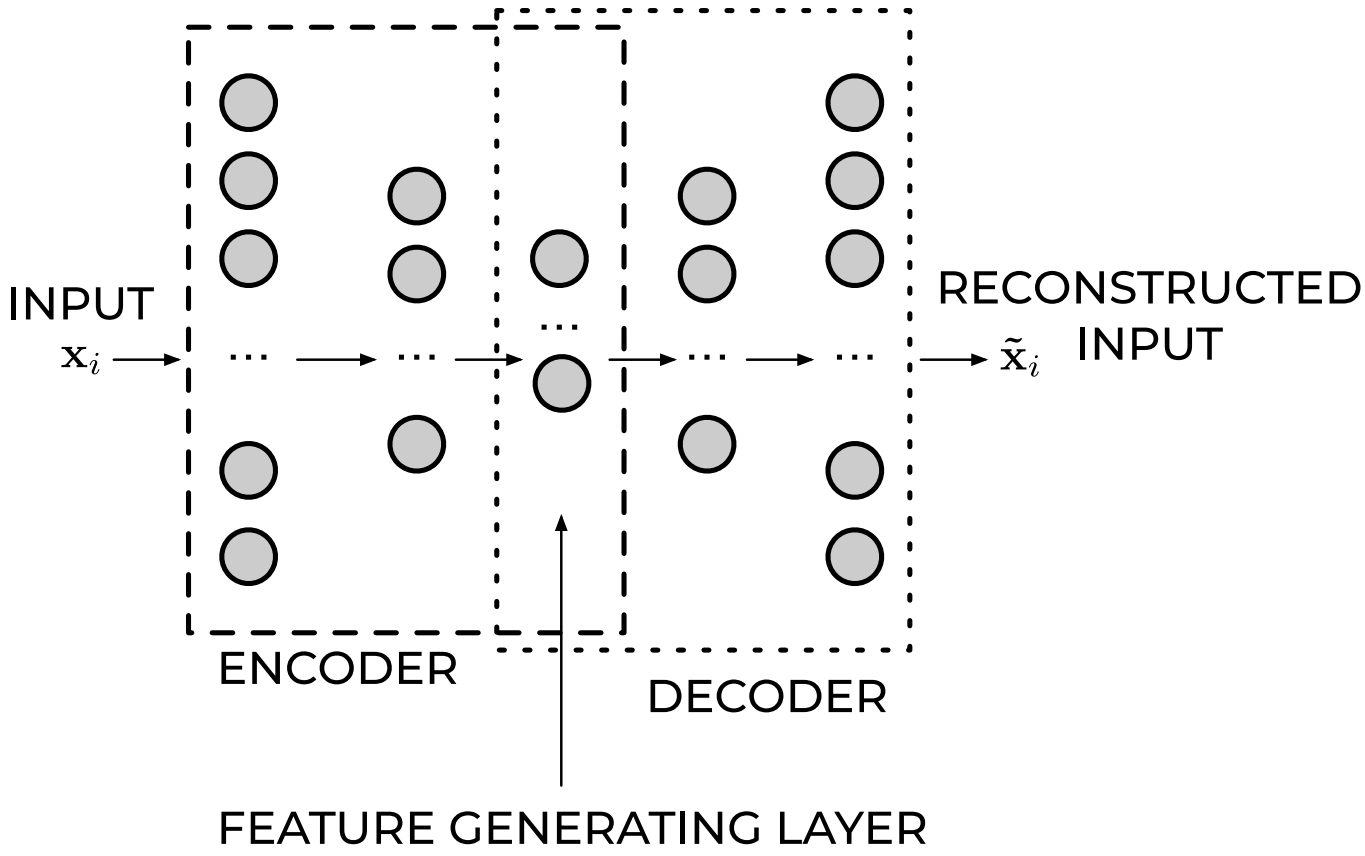

Figure 4: A typical architecture of a Feed-Forward Autoencoder. The number of neurons in the layers at first goes down as we move through the network until it reaches the middle and then starts to grow again until the last layer has the same number of neurons as the input dimensions.

Activation function of Output Layer

$$\mathrm{ReLU}\,(x) = \max\,(0, x)\,. \tag{4}$$

This choice is good when the input observations $\mathbf{x}_i$ assume a wide range of positive values.

$$\sigma\,(x) = \frac{1}{1 + e^{-x}}. \tag{5}$$

This activation function can only be used if the input observations $\mathbf{x}_i$ are all in the range $[0, 1]$ or if you have normalized them to be in that range.

Loss Functions

$$L_{\mathrm{MSE}} = \mathrm{MSE} = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|^2 \tag{6}$$

$$L_{\mathrm{CE}} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{d} [x_{i,j} \log \tilde{x}_{i,j} + (1 - x_{i,j}) \log(1 - \tilde{x}_{i,j})] \tag{7}$$

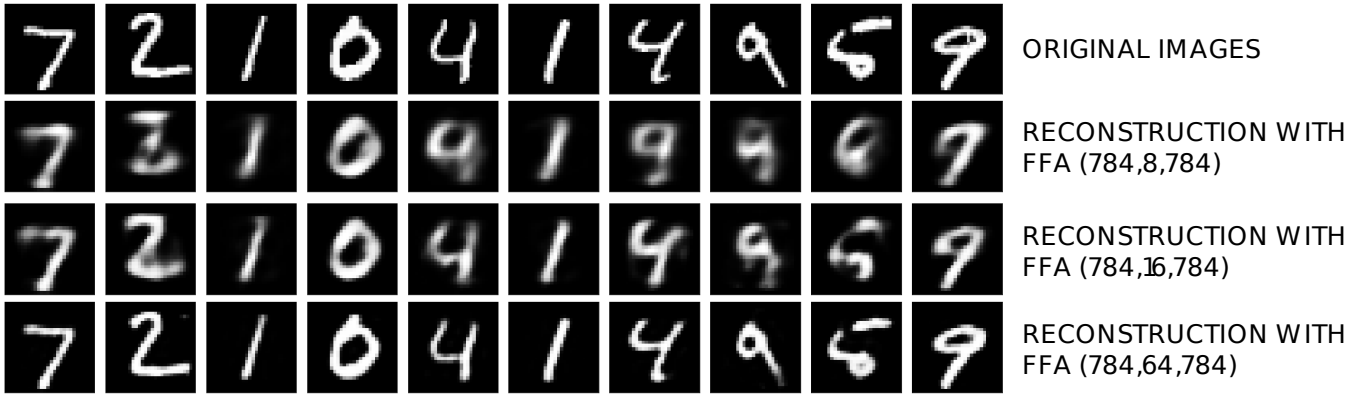

Figure 5: In the top line, you can see the original digits from the MNIST dataset. The second line of digits contains the digits reconsructed by the FFA (784,8,784), the third by the FFA (784,16,784), and the last one by the FFA (784,64,784).

Dimensionality Reduction

The encoder can reduce the number of dimensions of the input observation (n) and create a learned representation ($\mathbf{h_i}$)of the input that has a smaller dimension $q < n$. This learned representation is enough for the decoder to reconstruct the input accurately (if the autoencoder training was successful as intended).

Input Data	Accuracy	Running Time
Original data $\mathbf{x}_i \in \mathbb{R}^{784}$	96.4%	1000 sec. $\approx$ 16.6 min.
Latent Features $g(\mathbf{x}_i) \in \mathbb{R}^8$	89%	1.1 sec.

Table 1: the different in accuracy and running time when applying the kNN algorithm to the original 784 features or the 8 latent features for the MNIST dataset.

PCA as a special case

- We use a linear function for the encoder $g(\cdot)$
- We use a linear function for the decoder $f(\cdot)$
- We use the MSE for the loss function
- We normalize the inputs to

$$\hat{x}_{i,j} = \frac{1}{\sqrt{N}} \left(x_{i,j} - \frac{1}{N} \sum_{k=1}^{N} x_{k,j} \right) \tag{8}$$

The autoencoder has one main benefit from a computational point of view: it can deal with a very big amount of data efficiently since its training can be done with mini-batches, while PCA, one of the most used dimensionality reduction algorithms, needs to do its calculations using the entire dataset.

Anomaly Detection

The main idea is that since the autoencoder has only seen hand-written digits images, it will not be able to reconstruct the shoe image.
Therefore we expect this image to have the biggest reconstruction error.

Probabilistic Perspective

Assume we have a high-dimensional space $\mathcal{Z}$ containing a vector of latent variables z that we can readily sample from using a probability density function $p_\theta(z)$ defined over $\mathcal{Z}$.

Assume we have a set of deterministic functions $f(z, \theta)$, parametrized by a vector $\theta \in \Theta$, where $f : \mathcal{Z} \times \Theta \to \mathcal{X}$

Generative Model [Carl Doersch, 2016][4]

We want to maximize θ such that sampling z from $p_\theta(z)$ yields a high chance that $f(z, \theta)$ resembles the $\mathcal{X}$

[4] Carl Doersch, Tutorial on Variational Autoencoders

Generative Model

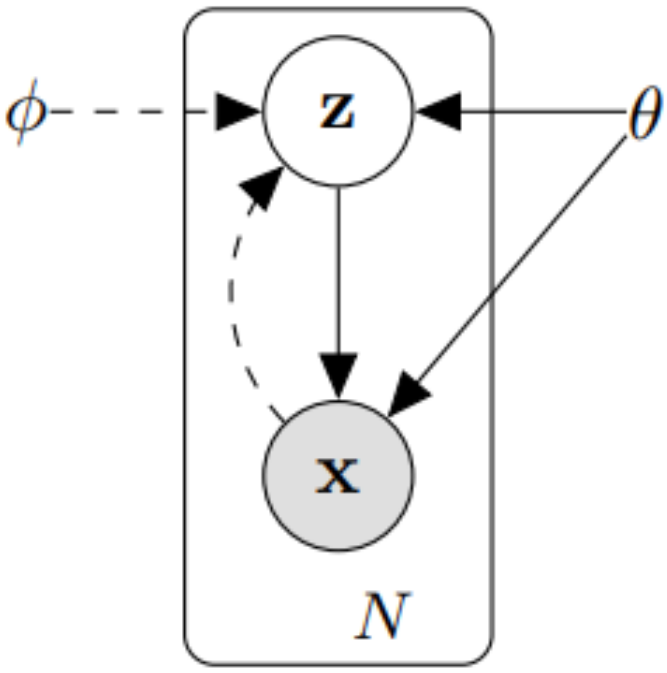

Figure 6: Generative Model

Under the generative model, we want to maximize x's probability, i.e.,

$$p_\theta(x) = \int p_\theta(x, z)dz = \int p_\theta(x|z)p_\theta(z)dz \tag{9}$$

Denoting $f(z, \theta)$ probabilistically by $p_\theta(x|z)$.

[Kingma and Welling, 2013][5]

1. If closed form of likelihood (9) can be calculated, then maximizing it with respect to θ will give us the maximum likelihood estimate of θ.
2. In practice (9) is intractable but if closed form of posterior $p_\theta(z|x)$ is known or can be efficiently approximated, then we can use EM algorithm.
3. A simple approach is taking samples z_i from $p_\theta(z)$, then take the average of their $p_\theta(x|z_i)$ values. But in practice, for most z, $p_\theta(x|z)$ will be nearly zero, and hence contribute almost nothing to the estimate $p_\theta(x)$ therefore Large Number of Samples will be required.

[5] Kingma and Welling, Auto-Encoding Variational Bayes

Variational Inference

The key idea is we need a new (approximating/surrogate) posterior $q_\phi(z|x)$ that are likely to estimate $p_\theta(x)$. However, if z is sampled from an arbitrary distribution $q_\phi(z|x)$ then how does that help us optimize $p_\theta(x)$?

Variational Inference gives the relation between $\mathbb{E}_{z\sim q_\phi}[\log p_\theta(x|z)]$ and $p_\theta(x)$

ELBO (Evidence Lower Bound)

$$\log p_\theta(x) \geq \mathcal{L}(\theta, \phi; x) = -D_{KL}(q_\phi(z|x) \| p_\theta(z)) + \mathbb{E}_{z \sim q_\phi}(\log p_\theta(x|z))$$

where

$$D_{KL}(f \| g) = \mathbb{E}_{x \sim f}\left[\log \frac{f(x)}{g(x)}\right]$$

So we optimize the lower bound.

VAE

Visualizing Autoencoder as Generative Model

1. The encoder may be seen as a function that maps input data x to a latent variable z, as indicated by the probability of detecting z given input data x, denoted by $q_\phi(z|x)$.
2. The likelihood, or probabilistic decoder $p_\theta(x|z)$, determines the strength of the association between a latent variable z and an observation x.

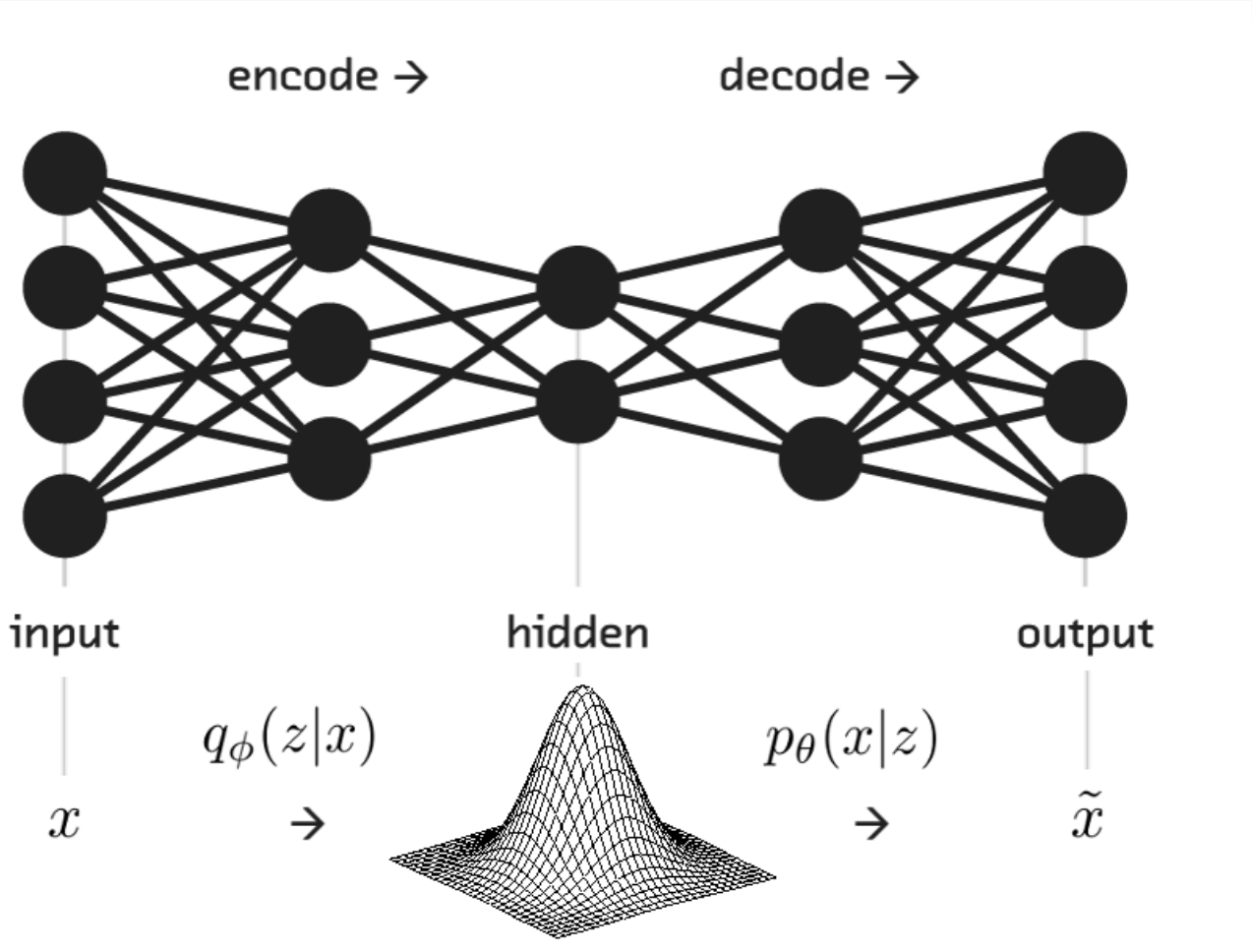

Figure 7: Neural Network Structure of Variational Autoencoder

How to implement it?

Prior on z is assumed to be standard multivariate normal distribution $\mathcal{N}(0, I)$ of desired latent size.

Encoder Network (ϕ parameters): The encoder network takes an input x and compresses it into a latent representation using sequence of layers decreasing in size. Final layer outputs a Multivariate Normal Diagonal distribution using the mean, standard deviation values from previous layer i.e., if latent size is d then layer just before final layer should have size $2 * d$ which will correspond to d location and d standard deviation values. You can construct more complex final layer with non-zero covariance terms correspondingly increasing the previous layer size. ϕ **includes all the weights and biases of the dense layers in the encoder.**

Define the Encoder

```
latent_size = 2
event_shape = (28,28)
encoder = Sequential([
       Flatten(input_shape=event_shape),
       Dense(256, activation='relu'),
       Dense(128, activation='relu'),
       Dense(64, activation='relu'),
       Dense(32, activation='relu'),
       Dense(2*latent_size),
       tfpl.DistributionLambda(
           lambda t: tfd.MultivariateNormalDiag(
              loc=t[..., :latent_size],
              scale_diag=tf.math.exp(t[..., latent_size:])
            )
          )
])
```

- latent_size $= 2$: This defines the size of latent space $\mathcal{Z}$.
- event_shape $= (28, 28)$: Shape of the input data (an image).
- Several Dense layers with ReLU activation.
- Dense(2*latent_size): This layer outputs a vector of size $2 * \textit{latent\_size}$ (which would be 4 in this case). This is because for each dimension in the latent space, the network will produce two values: one for the mean (loc) and one for the standard deviation (scale_diag).
- Final layer outputs a Multivariate Normal Diagonal distribution using the mean (loc), standard deviation (scale_diag) values from previous layer.

Decoder network (θ parameters): The decoder network takes a point from the latent space by sampling from final layer of encoder (Multivariate Normal Diagonal distribution) and reconstructs an output image. It is parameterized by θ, which includes all the weights and biases of the dense layers in the decoder.

Define the Decoder

```
decoder = Sequential([
 Dense(32, activation='relu'),
 Dense(64, activation='relu'),
 Dense(128, activation='relu'),
 Dense(256, activation='relu'),
 Dense(
  tfpl.IndependentBernoulli.params_size(event_shape)),
 tfpl.IndependentBernoulli(event_shape)
])
```

- Dense Layers: These layers gradually increase the dimensionality of the data, effectively reversing the compression done by the encoder.
- Dense layer tfpl.IndependentBernoulli.params_size(event_shape) computes the necessary parameters for a Bernoulli distribution for each 28x28 pixel, adapting the model for binary (black or white) image reconstruction.
- 'tfpl.IndependentBernoulli(event_shape)': layer then takes these parameters and forms a set of Bernoulli distributions, one for each pixel. This setup models each pixel's intensity as a probability of being white, leading to a probabilistic reconstruction of the image.

Optimization

We need to minimize negative ELBO

$$D_{KL}(q_\phi(z|x)\|p_\theta(z)) - E_{z\sim q_\phi}(\log p_\theta(x|z))$$

Where first we will encode x in the encoder and get the encoding distribution $q_\phi(z|x)$, then we will sample $z \sim q_\phi(z|x)$ from this distribution and pass to decoder which will output $p_\theta(x|z)$. Assuming independence of $x_i's$ we can write loss as

$$\sum_i D_{KL}(q_\phi(z_i|x_i)\|p_\theta(z_i)) - E_{z_i\sim q_\phi(z_i|x_i)}(\log p_\theta(x_i|z_i))$$

Gradient Flow: The gradients flow back from the output of the decoder through the decoder network (affecting θ), through the latent space, and back into the encoder network (affecting ϕ). The parameters are updated in a way that minimizes both the reconstruction loss $-E_{z\sim q_\phi}(\log p_\theta(x|z))$ and the KL divergence $D_{KL}(q_\phi(z|x)\|p_\theta(z))$.

Generative Task: Post Training

Generation of new x can be done by first sampling from $p_\theta(z)$, then passing it to decoder which will output the parameters for these $z's$. Which are the output of layer before distribution layer (Dense layer tfpl.IndependentBernoulli.params_size(event_shape)).

'tfpl.IndependentBernoulli(event_shape)': layer then takes these parameters and forms a set of Bernoulli distributions ($p_\theta(x|z)$), one for each pixel from which we can sample new x.