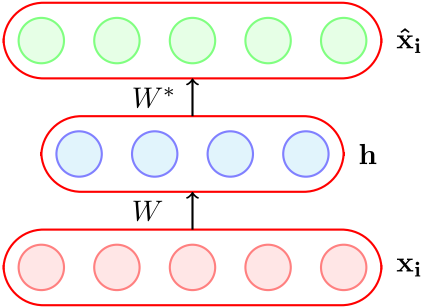
CS7015 (Deep Learning) : Lecture 21

Variational Autoencoders

Mitesh M. Khapra

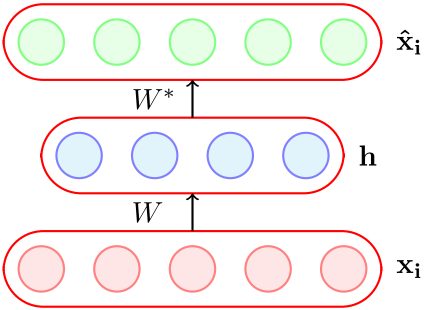
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Module 21.1: Revisiting Autoencoders



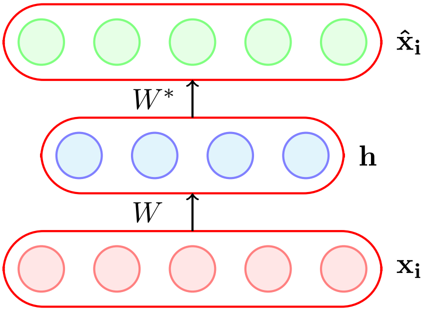
$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Before we start talking about VAEs, let us quickly revisit autoencoders



$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

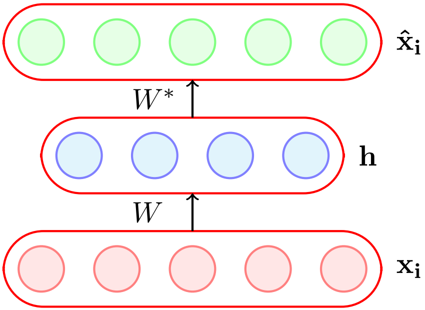
- Before we start talking about VAEs, let us quickly revisit autoencoders
- An autoencoder contains an encoder which takes the input X and maps it to a hidden representation



$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$

$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

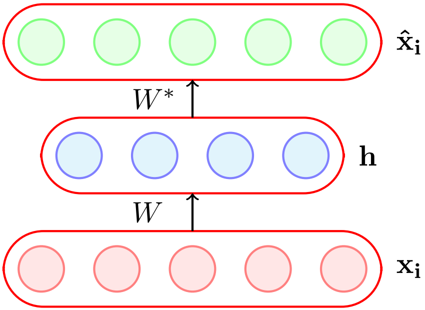
- Before we start talking about VAEs, let us quickly revisit autoencoders
- An autoencoder contains an encoder which takes the input X and maps it to a hidden representation
- The decoder then takes this hidden representation and tries to reconstruct the input from it as $\tilde{(X)}$



$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

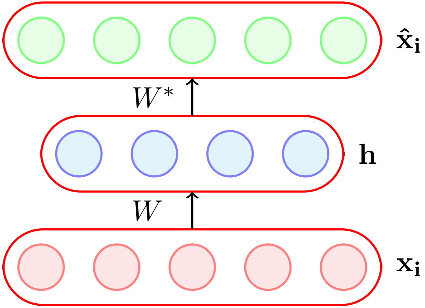
- Before we start talking about VAEs, let us quickly revisit autoencoders
- An autoencoder contains an encoder which takes the input X and maps it to a hidden representation
- The decoder then takes this hidden representation and tries to reconstruct the input from it as $\tilde(X)$
- The training happens using the following objective function

$$\min_{W,W^*,\mathbf{c},\mathbf{b}} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$



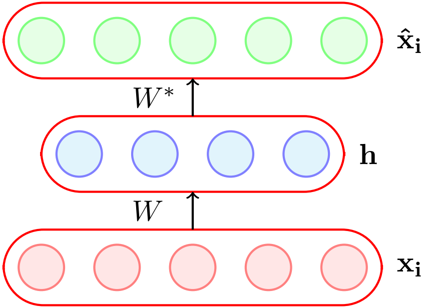
$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- But where's the fun in this ?



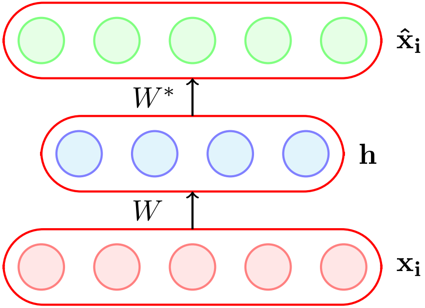
$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- But where's the fun in this ?
- We are taking an input and simply reconstructing it



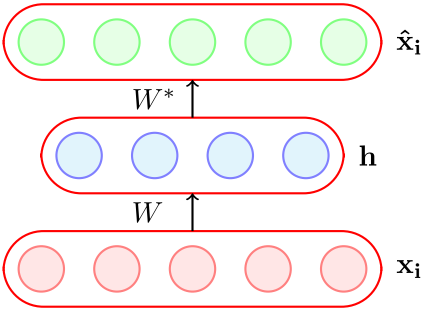
$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- But where's the fun in this ?
- We are taking an input and simply reconstructing it
- Of course, the fun lies in the fact that we are getting a good *abstraction* of the input



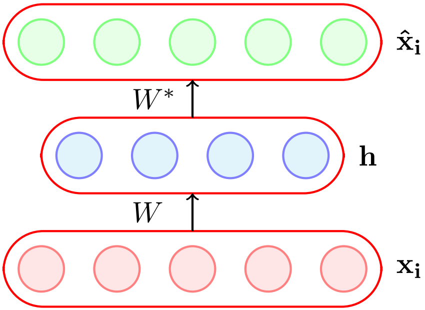
$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- But where's the fun in this ?
- We are taking an input and simply reconstructing it
- Of course, the fun lies in the fact that we are getting a good *abstraction* of the input
- But RBMs were able to do something more besides abstraction



$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

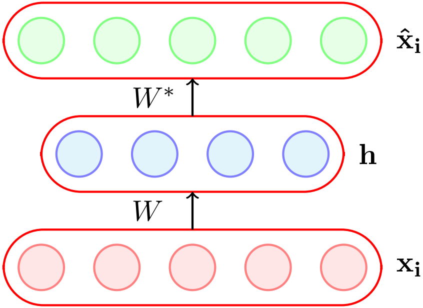
- But where's the fun in this ?
- We are taking an input and simply reconstructing it
- Of course, the fun lies in the fact that we are getting a good *abstraction* of the input
- But RBMs were able to do something more besides abstraction they were able to do *generation*)



$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$

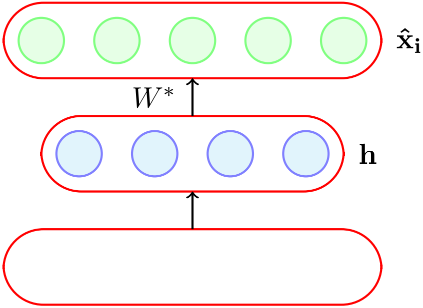
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- But where's the fun in this ?
- We are taking an input and simply reconstructing it
- Of course, the fun lies in the fact that we are getting a good *abstraction* of the input
- But RBMs were able to do something more besides abstraction they were able to do *generation*)
- Let us revisit *generation* in the context of autoencoders?



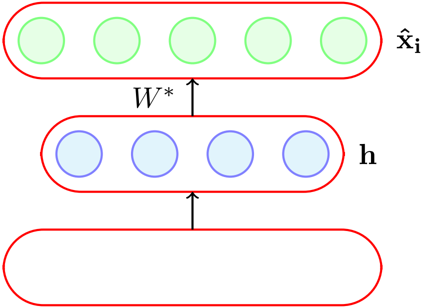
$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Can we do generation with autoencoders ?



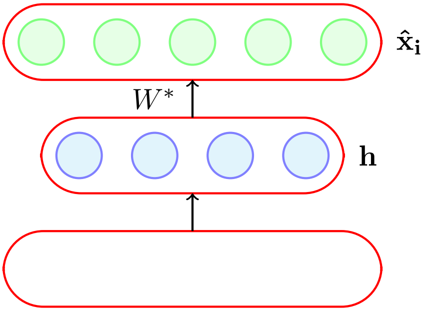
$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Can we do generation with autoencoders ?
- In other words, once the autoencoder is trained can I remove the encoder, feed a hidden representation h to the decoder and decode a $\tilde{X}$ from it ?



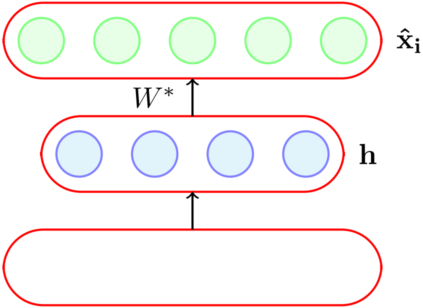
$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_{\mathbf{i}} = f(W^*\mathbf{h} + \mathbf{c})$$

- Can we do generation with autoencoders ?
- In other words, once the autoencoder is trained can I remove the encoder, feed a hidden representation h to the decoder and decode a $\tilde{X}$ from it ?
- In principle, yes, but in practice there is a problem with this approach



$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

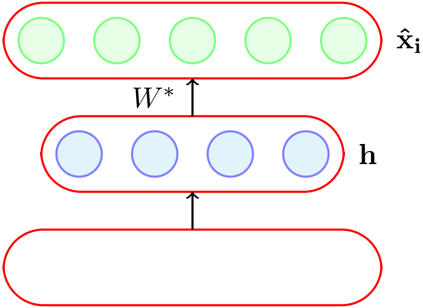
- Can we do generation with autoencoders ?
- In other words, once the autoencoder is trained can I remove the encoder, feed a hidden representation h to the decoder and decode a $\tilde{X}$ from it ?
- In principle, yes, but in practice there is a problem with this approach
- h is a very high dimensional vector and only a few vectors in this space would actually correspond to meaningful latent representations of our input



$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$

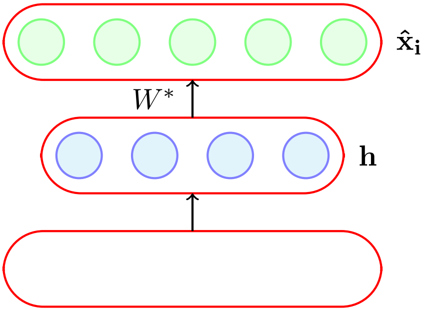
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Can we do generation with autoencoders ?
- In other words, once the autoencoder is trained can I remove the encoder, feed a hidden representation h to the decoder and decode a $\tilde{X}$ from it ?
- In principle, yes, but in practice there is a problem with this approach
- h is a very high dimensional vector and only a few vectors in this space would actually correspond to meaningful latent representations of our input
- So of all the possible value of h which values should I feed to the decoder (we had asked a similar question before: slide 67, bullet 5 of lecture 19)



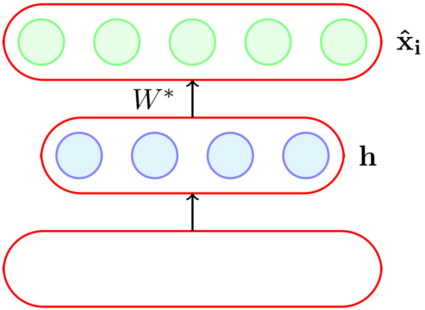
$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Ideally, we should only feed those values of h which are highly *likely*



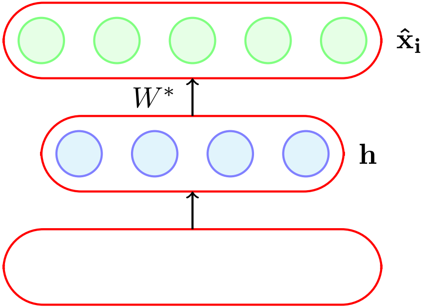
$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Ideally, we should only feed those values of h which are highly *likely*
- In other words, we are interested in sampling from $P(h|X)$ so that we pick only those h's which have a high probability



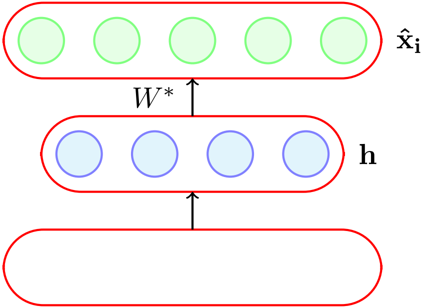
$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Ideally, we should only feed those values of h which are highly *likely*
- In other words, we are interested in sampling from $P(h|X)$ so that we pick only those h's which have a high probability
- But unlike RBMS, autoencoders do not have such a probabilistic interpretation



$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Ideally, we should only feed those values of h which are highly *likely*
- In other words, we are interested in sampling from $P(h|X)$ so that we pick only those h's which have a high probability
- But unlike RBMS, autoencoders do not have such a probabilistic interpretation
- They learn a hidden representation h but not a distribution $P(h|X)$



$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Ideally, we should only feed those values of h which are highly *likely*
- In other words, we are interested in sampling from $P(h|X)$ so that we pick only those h's which have a high probability
- But unlike RBMS, autoencoders do not have such a probabilistic interpretation
- They learn a hidden representation h but not a distribution $P(h|X)$
- We will now look at variational autoencoders which have the same structure as autoencoders but they learn a distribution over the hidden variables

Module 21.2: Intuition behind Variational Autoencoders

- Let $\{X = x_i\}_{i=1}^{N}$ be the training data

- Let $\{X = x_i\}_{i=1}^{N}$ be the training data
- We can think of X as a random variable in R^n

- Let $\{X = x_i\}_{i=1}^{N}$ be the training data
- We can think of X as a random variable in R^n
- For example, X could be an image and the dimensions of X correspond to pixels of the image

Figure: Abstraction

- Let $\{X = x_i\}_{i=1}^{N}$ be the training data
- We can think of X as a random variable in R^n
- For example, X could be an image and the dimensions of X correspond to pixels of the image
- We are interested in learning an abstraction (i.e., given an X find the hidden representation z)

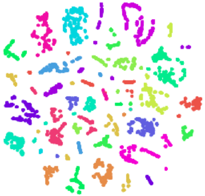


Figure: Abstraction

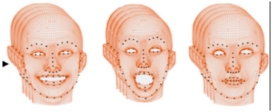


Figure: Generation

- Let $\{X = x_i\}_{i=1}^{N}$ be the training data
- We can think of X as a random variable in R^n
- For example, X could be an image and the dimensions of X correspond to pixels of the image
- We are interested in learning an abstraction (i.e., given an X find the hidden representation z)
- We are also interested in generation (i.e., given a hidden representation generate an X)

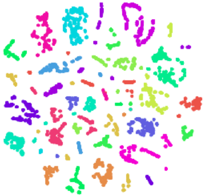


Figure: Abstraction

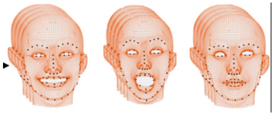
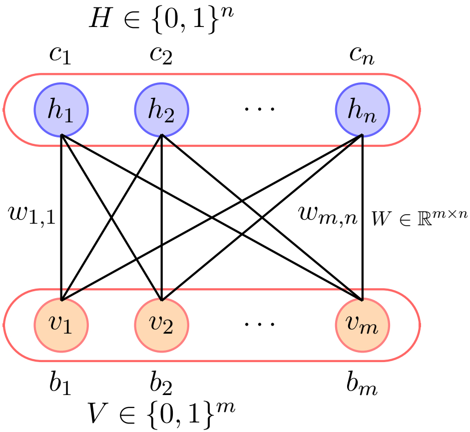
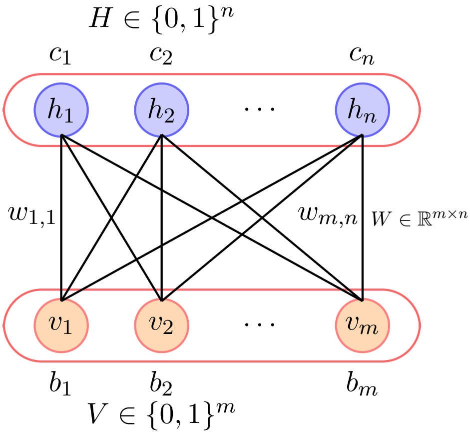


Figure: Generation

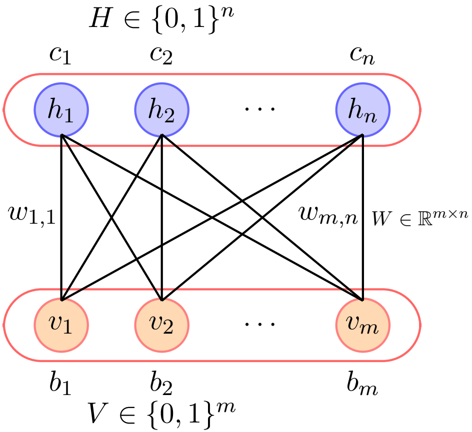
- Let $\{X = x_i\}_{i=1}^N$ be the training data
- We can think of X as a random variable in R^n
- For example, X could be an image and the dimensions of X correspond to pixels of the image
- We are interested in learning an abstraction (i.e., given an X find the hidden representation z)
- We are also interested in generation (i.e., given a hidden representation generate an X)
- In probabilistic terms we are interested in $P(z|X)$ and $P(X|z)$



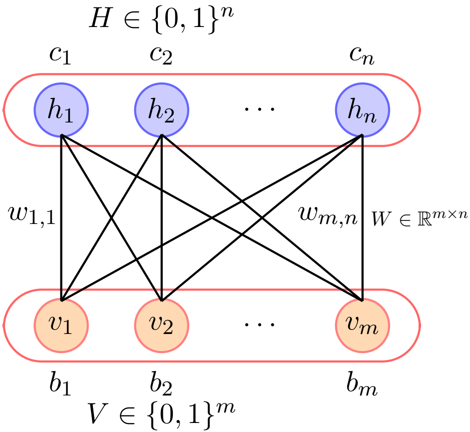
- Earlier we saw RBMs where we learnt $P(z|X)$ and $P(X|z)$ (to be consistent with the literation on VAEs we will use z instead of H and X instead of V)



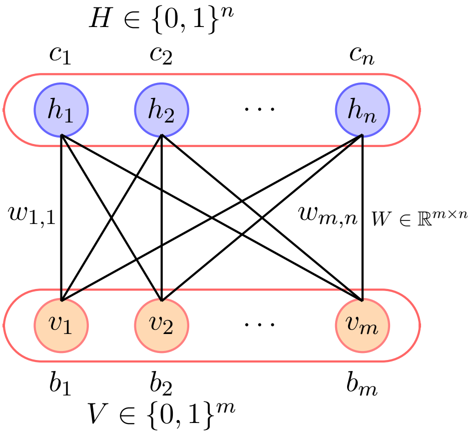
- Earlier we saw RBMs where we learnt $P(z|X)$ and $P(X|z)$ (to be consistent with the literation on VAEs we will use z instead of H and X instead of V)
- Below we list certain characteristics of RBMs



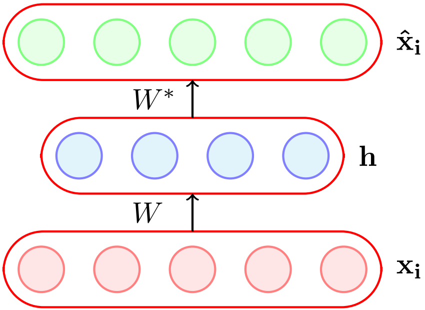
- Earlier we saw RBMs where we learnt $P(z|X)$ and $P(X|z)$ (to be consistent with the literation on VAEs we will use z instead of H and X instead of V)
- Below we list certain characteristics of RBMs
- **Structural assumptions:** We assumed certain independencies in the Markov Network



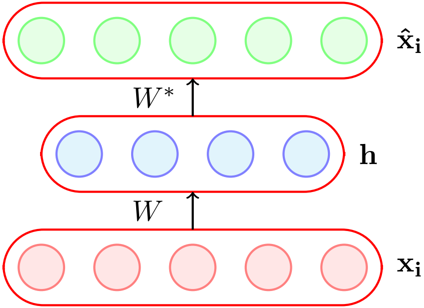
- Earlier we saw RBMs where we learnt $P(z|X)$ and $P(X|z)$ (to be consistent with the literation on VAEs we will use z instead of H and X instead of V)
- Below we list certain characteristics of RBMs
- **Structural assumptions:** We assumed certain independencies in the Markov Network
- **Computational:** When training with Gibbs Sampling we have to run the Markov Chain for many time steps which is expensive



- Earlier we saw RBMs where we learnt $P(z|X)$ and $P(X|z)$ (to be consistent with the literation on VAEs we will use z instead of H and X instead of V)
- Below we list certain characteristics of RBMs
- **Structural assumptions:** We assumed certain independencies in the Markov Network
- **Computational:** When training with Gibbs Sampling we have to run the Markov Chain for many time steps which is expensive
- **Approximation:** When using Contrastive Divergence, we approximate the expectation by a point estimate

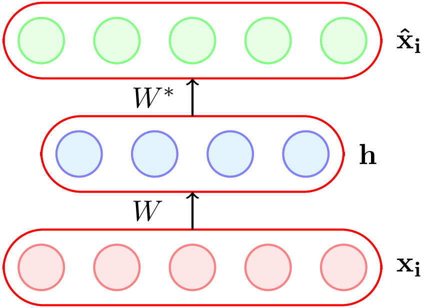


- We will now look at Variational Autoencoders which also learn $P(z|X)$ and $P(X|z)$ but using a very different approach



- We will now look at Variational Autoencoders which also learn $P(z|X)$ and $P(X|z)$ but using a very different approach
- Recall that in RBMS we learned the parameters by solving the following optimization problem

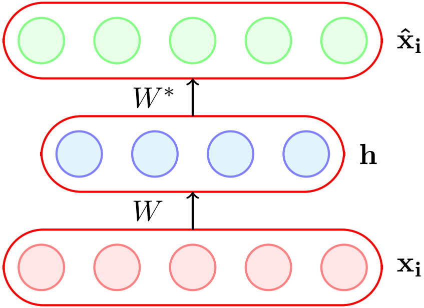
$$maximize \prod_{i=1}^{N} P(X = x_i)$$



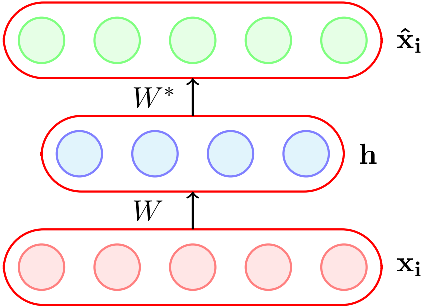
- We will now look at Variational Autoencoders which also learn $P(z|X)$ and $P(X|z)$ but using a very different approach
- Recall that in RBMS we learned the parameters by solving the following optimization problem

$$maximize \prod_{i=1}^{N} P(X = x_i)$$

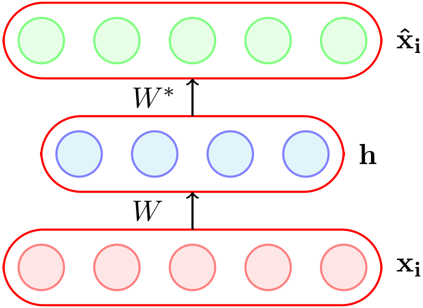
- In VAEs also we will start with the same objective function but use a different trick to solve the optimization problem (we will get there soon)



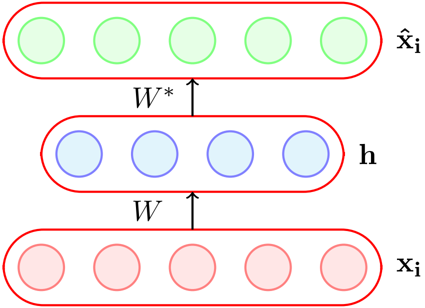
- Goal 1: Learn a distribution over the latent variables ($P(z|X)$)



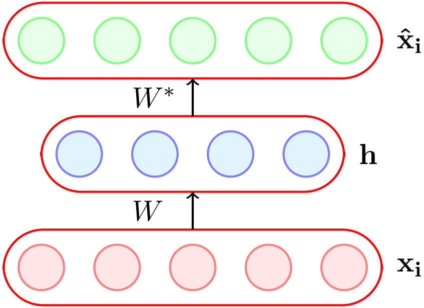
- Goal 1: Learn a distribution over the latent variables ($P(z|X)$)
- Goal 2: Learn a distribution over the visible variables ($P(X|z)$)



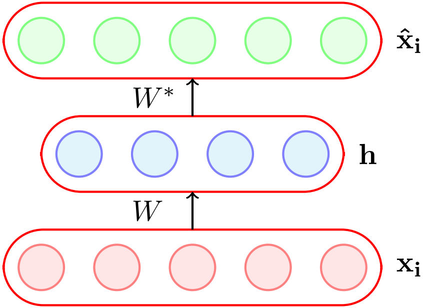
- Goal 1: Learn a distribution over the latent variables ($P(z|X)$)
- Goal 2: Learn a distribution over the visible variables ($P(X|z)$)
- Suppose we assume a certain form for the distribution $P(z|X)$



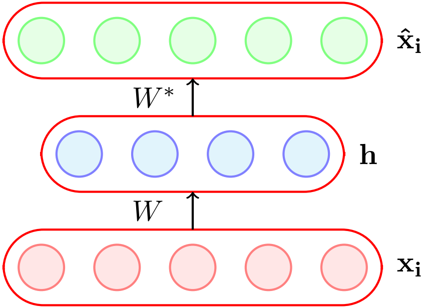
- Goal 1: Learn a distribution over the latent variables ($P(z|X)$)
- Goal 2: Learn a distribution over the visible variables ($P(X|z)$)
- Suppose we assume a certain form for the distribution $P(z|X)$
- For example, let us assume that $P(z|X)$ is a Normal distribution



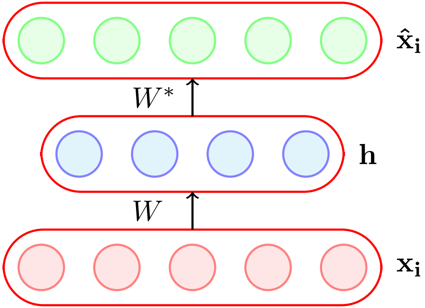
- Goal 1: Learn a distribution over the latent variables ($P(z|X)$)
- Goal 2: Learn a distribution over the visible variables ($P(X|z)$)
- Suppose we assume a certain form for the distribution $P(z|X)$
- For example, let us assume that $P(z|X)$ is a Normal distribution
- In RBMs, we had assumed a certain structure (independencies) for the joint distribution whereas here we are assuming a certain family form for the distribution $P(z|X)$



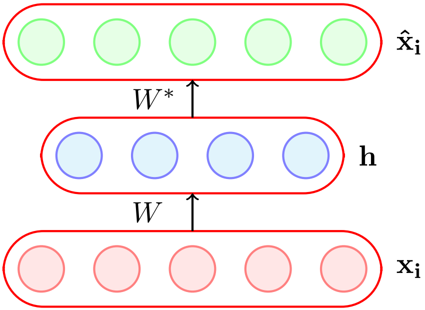
- Now given the assumption that $P(z|X)$ is a Normal distribution can you think of adapting the autoencoder architecture to learn the distribution $P(z|X)$ instead of learning z



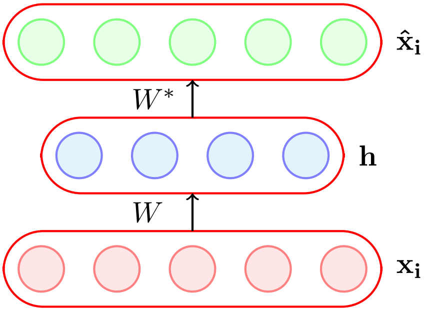
- Now given the assumption that $P(z|X)$ is a Normal distribution can you think of adapting the autoencoder architecture to learn the distribution $P(z|X)$ instead of learning z
- What do we mean when we say we want to learn a distribution? We mean that we want to learn the parameters of the distribution



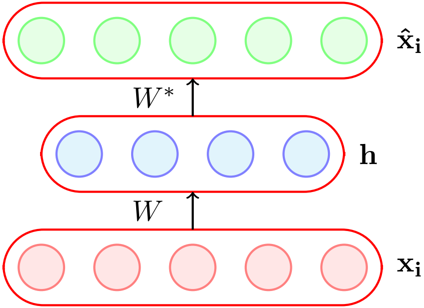
- Now given the assumption that $P(z|X)$ is a Normal distribution can you think of adapting the autoencoder architecture to learn the distribution $P(z|X)$ instead of learning z
- What do we mean when we say we want to learn a distribution? We mean that we want to learn the parameters of the distribution
- What are the parameters of a Normal Distribution? Mean (μ) and Variance (σ)



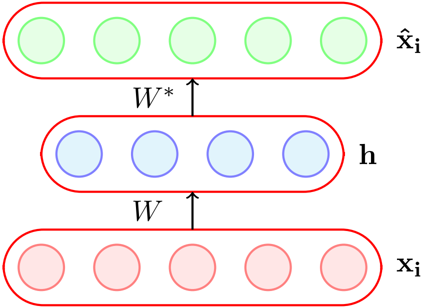
- Now given the assumption that $P(z|X)$ is a Normal distribution can you think of adapting the autoencoder architecture to learn the distribution $P(z|X)$ instead of learning z
- What do we mean when we say we want to learn a distribution? We mean that we want to learn the parameters of the distribution
- What are the parameters of a Normal Distribution? Mean (μ) and Variance (σ)
- So now what would you want the embedding generated by the encoder to represent ?



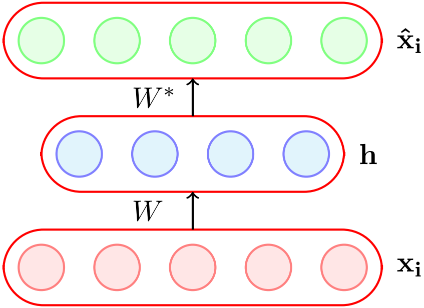
- Now given the assumption that $P(z|X)$ is a Normal distribution can you think of adapting the autoencoder architecture to learn the distribution $P(z|X)$ instead of learning z
- What do we mean when we say we want to learn a distribution? We mean that we want to learn the parameters of the distribution
- What are the parameters of a Normal Distribution? Mean (μ) and Variance (σ)
- So now what would you want the embedding generated by the encoder to represent ?
- Well, we could design the encoder to predict the mean and variance of $P(z|X)$



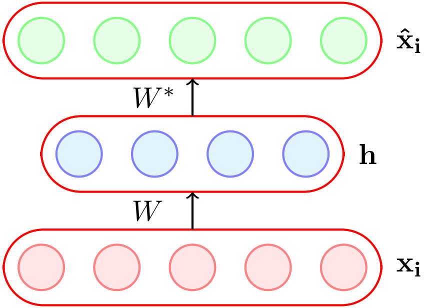
- But how will we train the encoder ?



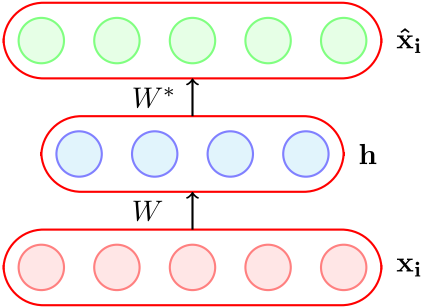
- But how will we train the encoder ?
- What is the loss function that we will use ?



- But how will we train the encoder ?
- What is the loss function that we will use ?
- More specifically, how will we ensure that the μ and σ that the encoder produces are the true μ and σ of $P(z|X)$



- But how will we train the encoder ?
- What is the loss function that we will use ?
- More specifically, how will we ensure that the μ and σ that the encoder produces are the true μ and σ of $P(z|X)$
- We don't even know what the true μ and σ are then how to we guide/train the model to produce these μ and σ



- But how will we train the encoder ?
- What is the loss function that we will use ?
- More specifically, how will we ensure that the μ and σ that the encoder produces are the true μ and σ of $P(z|X)$
- We don't even know what the true μ and σ are then how to we guide/train the model to produce these μ and σ
- **Spoiler Alert:** We will be making some assumptions but to see why these assumptions make sense we will first revisit the concept of latent variables

- Consider images of handwritten digits where visible random variables are the pixels of the image

- Consider images of handwritten digits where visible random variables are the pixels of the image
- And we can think of several latent variables (z): the digit, size, angle, thickness, position and so on

- Consider images of handwritten digits where visible random variables are the pixels of the image
- And we can think of several latent variables (z): the digit, size, angle, thickness, position and so on
- It is reasonable to argue that once we select a z, the image is actually a deterministic function of z

- Consider images of handwritten digits where visible random variables are the pixels of the image
- And we can think of several latent variables (z): the digit, size, angle, thickness, position and so on
- It is reasonable to argue that once we select a z, the image is actually a deterministic function of z
- For example, once the digit, size, angle thickness, position, *etc.* are fixed we know exactly what the image should look like

- Consider images of handwritten digits where visible random variables are the pixels of the image
- And we can think of several latent variables (z): the digit, size, angle, thickness, position and so on
- It is reasonable to argue that once we select a z, the image is actually a deterministic function of z
- For example, once the digit, size, angle thickness, position, *etc.* are fixed we know exactly what the image should look like
- Of course, we are assuming that we have enough latent variables to capture all characteristics of handwritten digits

- Of course, even though deterministic, the image (or visible variables) may still be a complex function of the hidden variables z

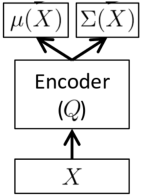
- Of course, even though deterministic, the image (or visible variables) may still be a complex function of the hidden variables z
- In others words, $X = f(z; \theta)$ where z is a complex function and θ are the parameters of this complex function

- Of course, even though deterministic, the image (or visible variables) may still be a complex function of the hidden variables z
- In others words, $X = f(z;\theta)$ where z is a complex function and θ are the parameters of this complex function
- Can you think of how you could model such a complex function ?

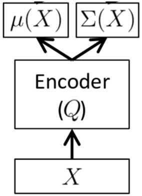
- Of course, even though deterministic, the image (or visible variables) may still be a complex function of the hidden variables z
- In others words, $X = f(z;\theta)$ where z is a complex function and θ are the parameters of this complex function
- Can you think of how you could model such a complex function ?
- Well, we could use a deep neural network which is good at approximating arbitrary complex functions (recall Universal Approximation Theorem)

- Of course, even though deterministic, the image (or visible variables) may still be a complex function of the hidden variables z
- In others words, $X = f(z; \theta)$ where z is a complex function and θ are the parameters of this complex function
- Can you think of how you could model such a complex function ?
- Well, we could use a deep neural network which is good at approximating arbitrary complex functions (recall Universal Approximation Theorem)
- With this intuition, we will now look at the full picture and discuss the objective function

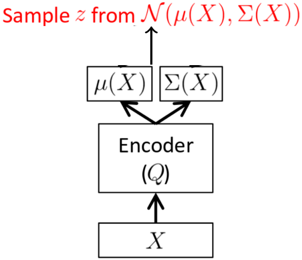
- First, the task of the encoder is to predict the parameters (μ, σ) of $P(z|X)$

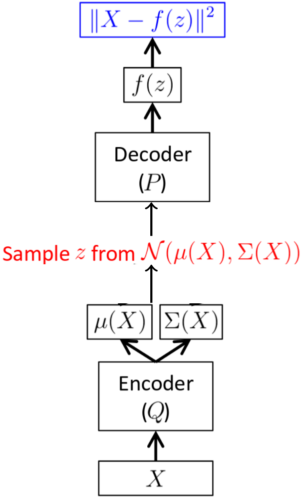


- First, the task of the encoder is to predict the parameters (μ, σ) of $P(z|X)$
- So given a training sample X, the encoder will first produce $\mu(X), \sigma(X)$

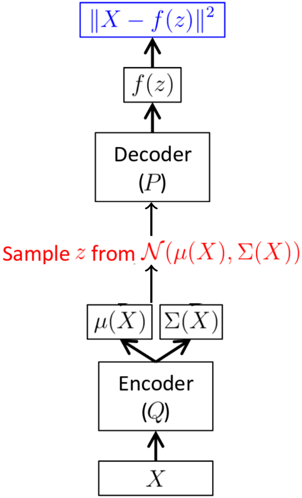


- First, the task of the encoder is to predict the parameters (μ, σ) of $P(z|X)$
- So given a training sample X, the encoder will first produce $\mu(X), \sigma(X)$
- We will now sample a hidden variable z from the distribution $N(\mu(X), \sigma(X))$



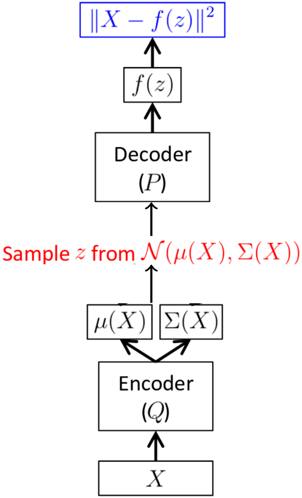


- First, the task of the encoder is to predict the parameters (μ, σ) of $P(z|X)$
- So given a training sample X, the encoder will first produce $\mu(X), \sigma(X)$
- We will now sample a hidden variable z from the distribution $N(\mu(X), \sigma(X))$
- The job of the decoder is to then reconstruct X from this sampled z



- Given this setup what should be the loss function for the decoder

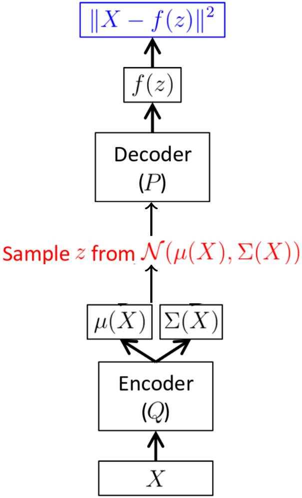
$$||X - f(z;\theta)||^2$$



- Given this setup what should be the loss function for the decoder

$$||X - f(z;\theta)||^2$$

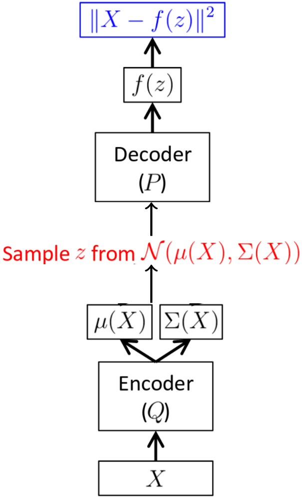
- But what about the encoder? What kind of loss function would ensure μ and σ that the encoder produces are the true μ and σ of $P(z|X)$?



- Given this setup what should be the loss function for the decoder

$$||X - f(z;\theta)||^2$$

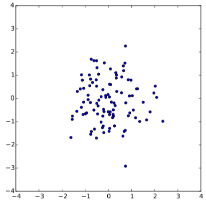
- But what about the encoder? What kind of loss function would ensure μ and σ that the encoder produces are the true μ and σ of $P(z|X)$?
- Well if we knew the true $P(z)$ then we could have minimized the KL divergence between $Q(z|X)$ and $P(z)$

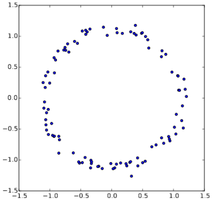


- Given this setup what should be the loss function for the decoder

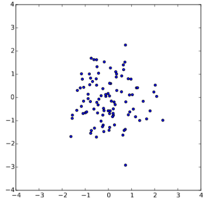
$$||X - f(z;\theta)||^2$$

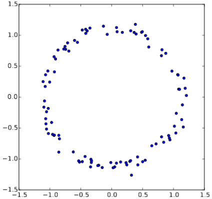
- But what about the encoder? What kind of loss function would ensure μ and σ that the encoder produces are the true μ and σ of $P(z|X)$?
- Well if we knew the true $P(z)$ then we could have minimized the KL divergence between $Q(z|X)$ and $P(z)$
- But we don't know what $P(z)$ is so we will make an assumption that $P(z)$ is $N(0, I)$



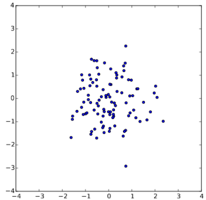


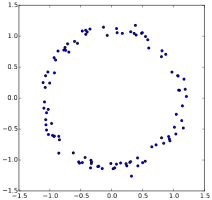
- Isn't this is very strong assumption ?



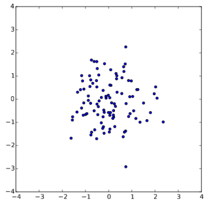


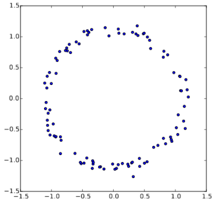
- Isn't this is very strong assumption ?
- For example, in the 2-dimensional case how can we be sure that P(z) is a normal distribution and not any other distribution



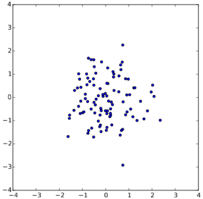


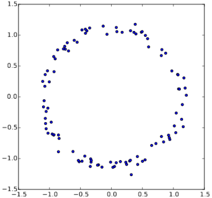
- Isn't this is very strong assumption ?
- For example, in the 2-dimensional case how can we be sure that P(z) is a normal distribution and not any other distribution
- The key insight here is that any distribution in d dimensions can be generated by the following steps



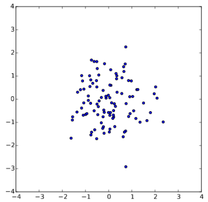


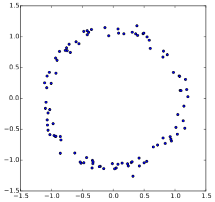
- Isn't this is very strong assumption ?
- For example, in the 2-dimensional case how can we be sure that P(z) is a normal distribution and not any other distribution
- The key insight here is that any distribution in d dimensions can be generated by the following steps
- Step 1: Start with a set of d variables that are normally distributed (that's exactly what we are assuming for $P(z)$)





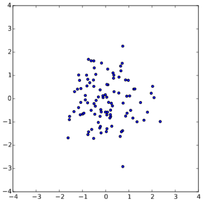
- Isn't this is very strong assumption ?
- For example, in the 2-dimensional case how can we be sure that P(z) is a normal distribution and not any other distribution
- The key insight here is that any distribution in d dimensions can be generated by the following steps
- Step 1: Start with a set of d variables that are normally distributed (that's exactly what we are assuming for $P(z)$)
- Step 2: Mapping these variables through a sufficiently complicated function (that's exactly what the first few layers of the decoder can do)

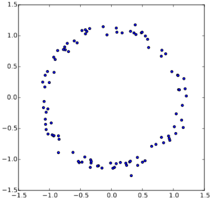




- In particular, note that in the adjoining example if z is 2-D and normally distributed then $g(z)$ is roughly ring shaped (giving us the distribution in the bottom figure)

$$g(z) = \frac{z}{10} + \frac{z}{||z||}$$



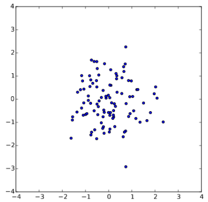


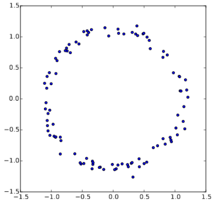
- In particular, note that in the adjoining example if z is 2-D and normally distributed then $g(z)$ is roughly ring shaped (giving us the distribution in the bottom figure)

$$g(z) = \frac{z}{10} + \frac{z}{||z||}$$

- In other words,

$$g(z) = \theta_1 z + \theta_2 z$$





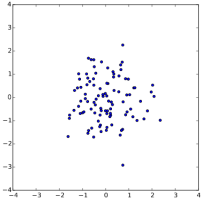
- In particular, note that in the adjoining example if z is 2-D and normally distributed then $g(z)$ is roughly ring shaped (giving us the distribution in the bottom figure)

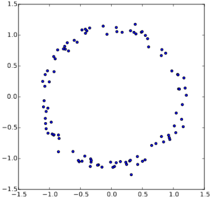
$$g(z) = \frac{z}{10} + \frac{z}{||z||}$$

- In other words,

$$g(z) = \theta_1 z + \theta_2 z$$

- The initial layers of the decoder could learns their weights such that the output is $g(z, \theta)$





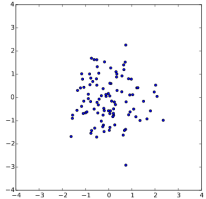
- In particular, note that in the adjoining example if z is 2-D and normally distributed then $g(z)$ is roughly ring shaped (giving us the distribution in the bottom figure)

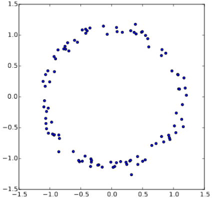
$$g(z) = \frac{z}{10} + \frac{z}{||z||}$$

- In other words,

$$g(z) = \theta_1 z + \theta_2 z$$

- The initial layers of the decoder could learns their weights such that the output is $g(z, \theta)$
- The above argument suggests that even if we start with normally distributed variables the initial layers of the encoder could learn a complex transformation of these variables say $f(z, \theta_1)$ if required





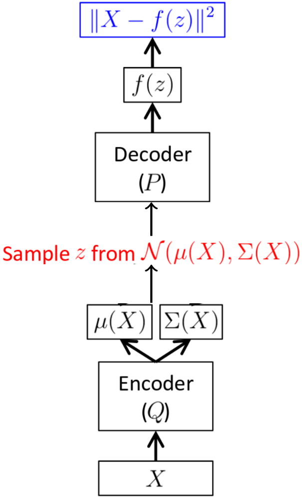
- In particular, note that in the adjoining example if z is 2-D and normally distributed then $g(z)$ is roughly ring shaped (giving us the distribution in the bottom figure)

$$g(z) = \frac{z}{10} + \frac{z}{||z||}$$

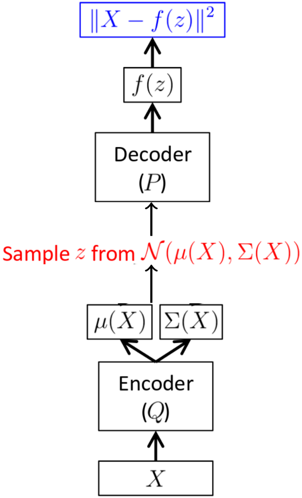
- In other words,

$$g(z) = \theta_1 z + \theta_2 z$$

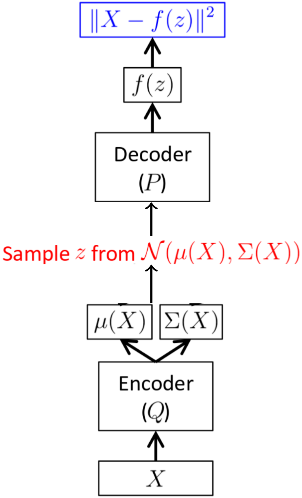
- The initial layers of the decoder could learns their weights such that the output is $g(z, \theta)$
- The above argument suggests that even if we start with normally distributed variables the initial layers of the encoder could learn a complex transformation of these variables say $f(z, \theta_1)$ if required
- The objective function of the decoder will ensure that an appropriate transformation of z is learnt to reconstruct X



- So now that we are convinced that it is okay to assume P(z) is N(0, I) then the objective function of the encoder is straightforward

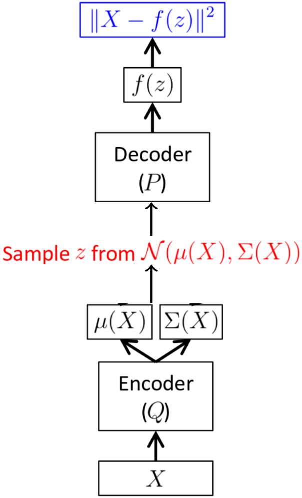


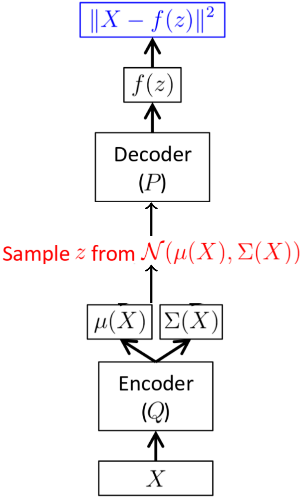
- So now that we are convinced that it is okay to assume P(z) is N(0, I) then the objective function of the encoder is straightforward
- We simply need to minimize $KL[N(\mu(X), \sigma(X)), N(0, I)]$



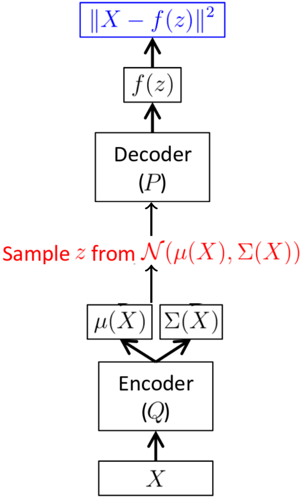
- So now that we are convinced that it is okay to assume P(z) is N(0, I) then the objective function of the encoder is straightforward
- We simply need to minimize $KL[N(\mu(X), \sigma(X)), N(0, I)]$
- That completes the full picture and we will summarize the discussion on the next slide

- Encoder: Generates the μ and σ of $Q(z|X)$



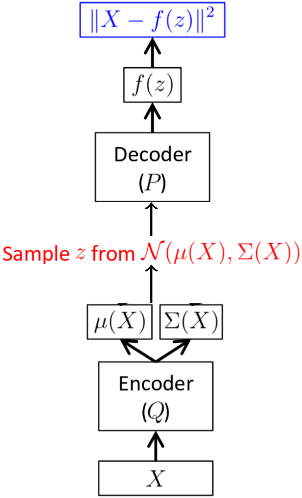


- Encoder: Generates the μ and σ of $Q(z|X)$
- Decoder: Samples from $Q(z|X)$ and reconstructs X



- Encoder: Generates the μ and σ of $Q(z|X)$
- Decoder: Samples from $Q(z|X)$ and reconstructs X
- Encoder loss:

$$min \quad KL[N(\mu(X), \Sigma(X)), N(0, I)]$$

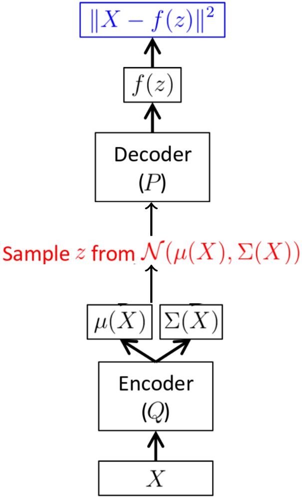


- Encoder: Generates the μ and σ of $Q(z|X)$
- Decoder: Samples from $Q(z|X)$ and reconstructs X
- Encoder loss:

$$min \quad KL[N(\mu(X), \Sigma(X)), N(0, I)]$$

- Decoder loss:

$$min \quad ||X - f(z; \theta)||^2$$



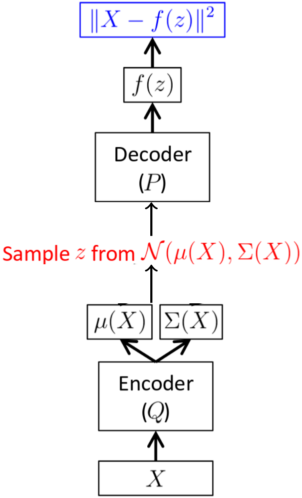
- Encoder: Generates the μ and σ of $Q(z|X)$
- Decoder: Samples from $Q(z|X)$ and reconstructs X
- Encoder loss:

$$min \quad KL[N(\mu(X), \Sigma(X)), N(0, I)]$$

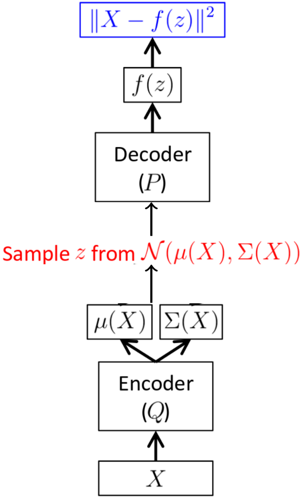
- Decoder loss:

$$min \quad ||X - f(z; \theta)||^2$$

- There are still a few pieces missing and we will get back to them later



- This was a very simplistic (non-rigorous) introduction to VAEs



- This was a very simplistic (non-rigorous) introduction to VAEs
- We will now do a more rigorous discussion on the Math behind VAEs