Autoencoders

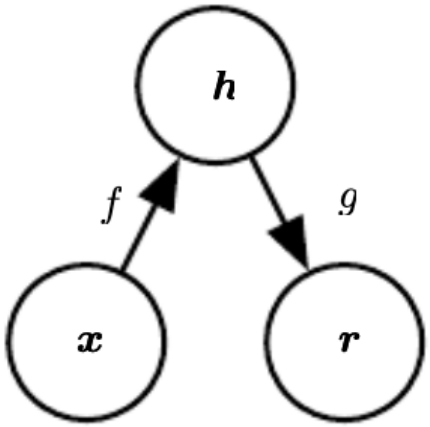
Denis Derkach, Maksim Artemev, Artem Ryzhikov

CS HSE faculty, Generative Models, spring 2020

Contents

Autoencoders

General idea



Two parts of the network:

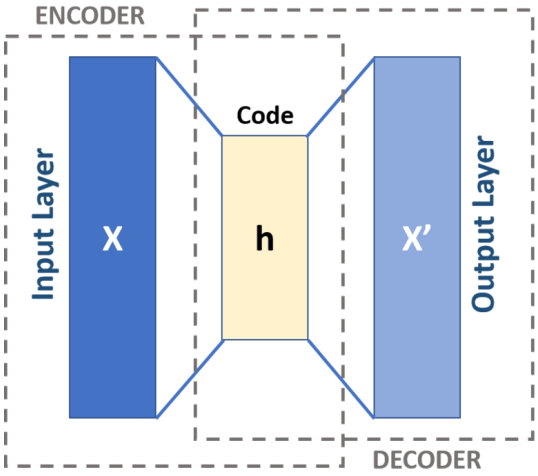
- encoder $h = f(x)$;
- decoder $r = g(h)$

Generally, we want to find a transformation

$$g(f(x)) = x$$

Which is fairly useless.

Hidden representation



- Interesting part is in the middle.
- Make it difficult to learn identity function.
- Reduce the width of hidden layer.
- Or add noise to the input layer.

Simple Autoencoder

For binary data $x \in \mathbb{R}^D$ and a one-layer network we have:

$$h(x) = g(b + Wx);$$
$$\hat{x} = sigmoid(c + Vh(x)).$$

Here g is the hidden layer nonlinear activation function, W and V are network input-to-hidden and hidden-to-output weights, respectively, and b and c are the bias terms.
A typical loss is cross-entropy:

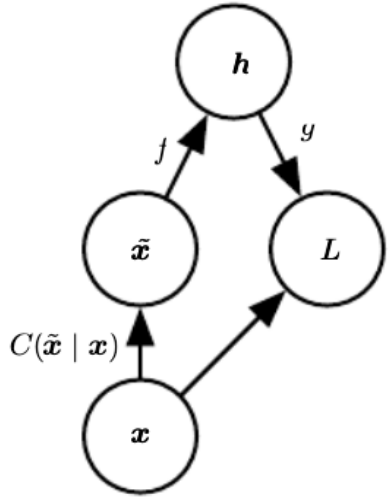
$$l(x) = \sum_{d=1}^{D} -x_d \log(\hat{x}_d) - (1 - x_d) \log(1 - \hat{x}_d).$$

Simple Autoencoder: Discussion

$$l(x) = \sum_{d=1}^{D} -x_d \log(\hat{x}_d) - (1 - x_d) \log(1 - \hat{x}_d).$$

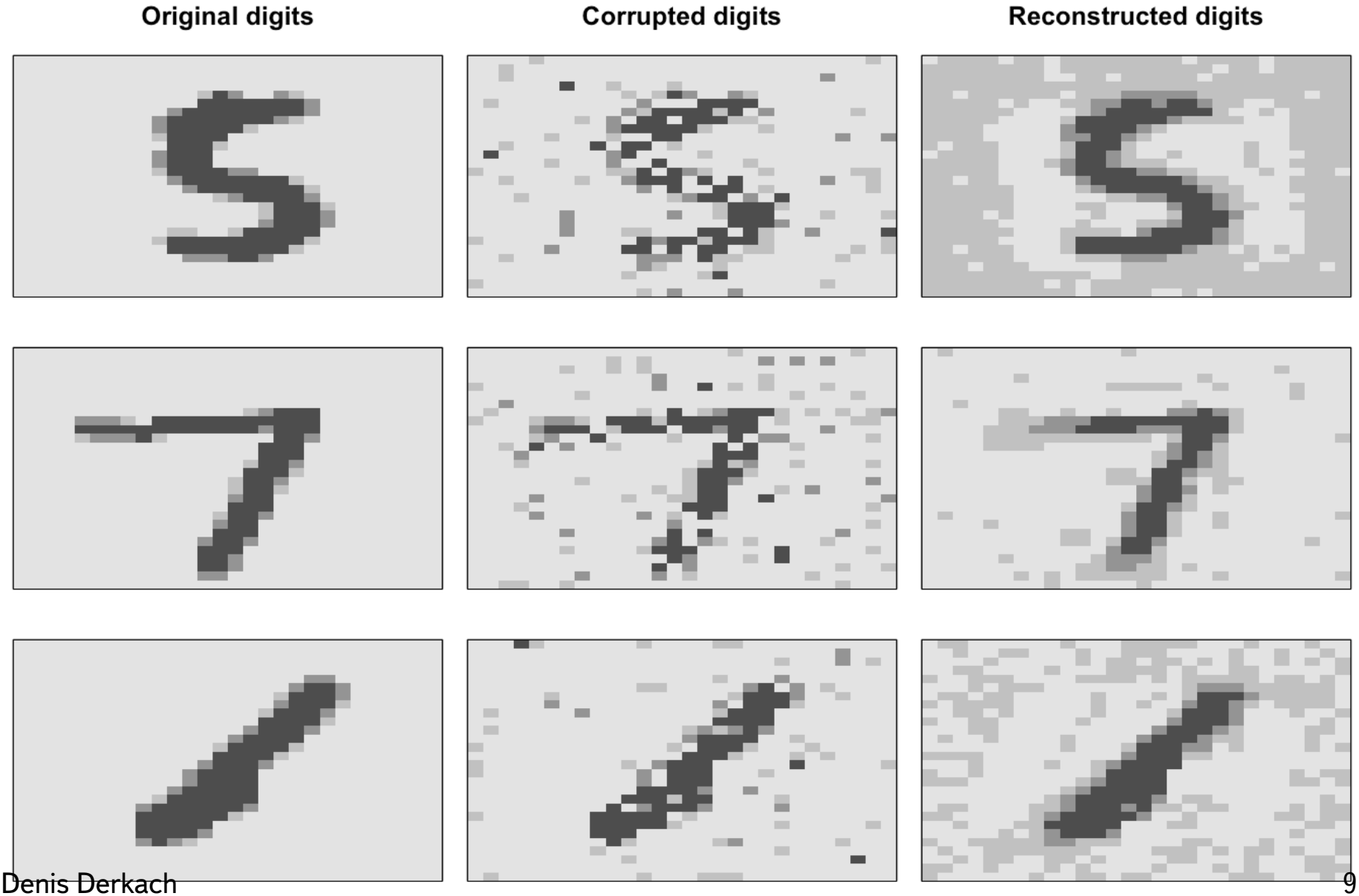
We can interpret $\mathbb{P}(x_d = 1) = \hat{x}_d$, which leads to consideration cross-entropy as a negative log probability. However, $e^{-l(x)}$ does not sum to 1: for perfect model $x_1 \to \hat{x}_1$ and $x_2 \to \hat{x}_2$, which brings $\mathbb{P}(x_1 = 1) = 1$ and $\mathbb{P}(x_d = 2) = 1$.
Thus we cannot create a true generative model out of vanilla autoencoder.

Denoising Autoencoders as Generative model

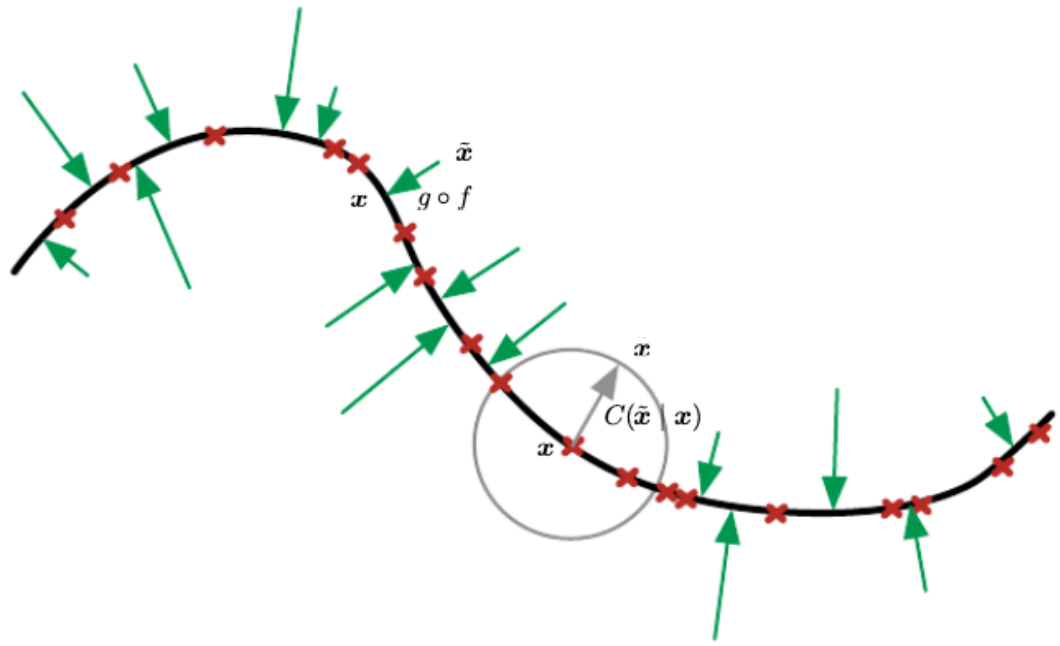


- Artificially add noise to sample: $\tilde{x} \sim C(\tilde{x}|x)$;
- Reconstruct $\mathbb{P}(x|\tilde{x})$
- Uses $L = -\log p_{dec}(x|h = f(\tilde{x}))$.

(De-)Noising MNIST



Signal Manifold



The true signal is always situated on a manifold inside the $\mathbb{R}^D$ space. Denoising autoencoder is trained to map a corrupted data point $\tilde{x}$ back to the original data point x.

Making Probabilistic Model

We thus can construct a Markov chain:

- $X_t \sim P_\theta(X|\tilde{X}_{t-1})$;
- $\tilde{X}_t \sim C(\tilde{X}|X_t)$.

And sample new data points from it.
While we were not expecting this, we constructed a sampling algorithm (in fact, we were constructing pseudo-likelihood).

Y. Bengio, Generalized Denoising Auto-Encoders as Generative Models, NIPS-13

DAE: Results

Figure 4: *Successive samples generated by Markov chain associated with the trained DAEs according to the plain sampling scheme (left) and walkback sampling scheme (right). There are less "spurious" samples with the walkback algorithm.*

Y. Bengio, Generalized Denoising Auto-Encoders as Generative Models, NIPS-13

Autoregressive Models: Reminder

We use product rule:

$$p(x_1, \ldots, x_D) = \prod_{d=1}^{D} p(x_d | x_{<d}).$$

The loss to use in this case looks somewhat different:

$$l(x) = \sum_{d=1}^{D} -x_d \log(p(x_d = 1 | x_{<d})) - (1 - x_d) \log(p(x_d = 0 | x_{<d}))$$

and becomes a a valid negative log probability.

Autoregressive Models: More Interpretations

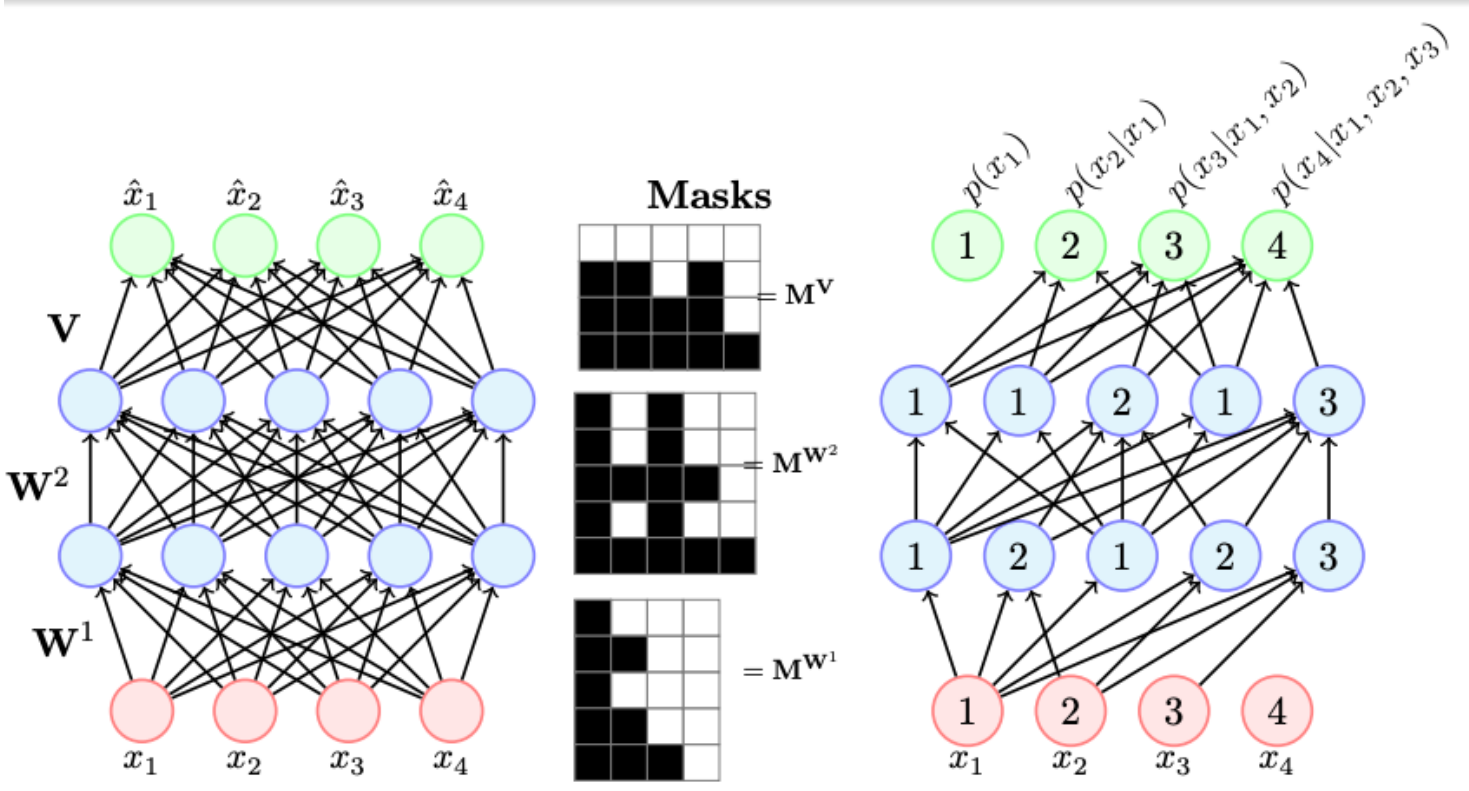
- Each output of the network $\hat{x}_d$ represents the probability distribution $p(x_d|x_{<d}$.
- Each output x_d can only have connections (recursively) to smaller indexed inputs $x_{<d}$ and not any of the other ones.

In this view of the autoencoder, we are sequentially predicting (i.e. regressing) each dimension of the data using its previous values, hence this is called the autoregressive property of autoencoders.

Brian Keng's blog

Masked Autoencoder: Intuition

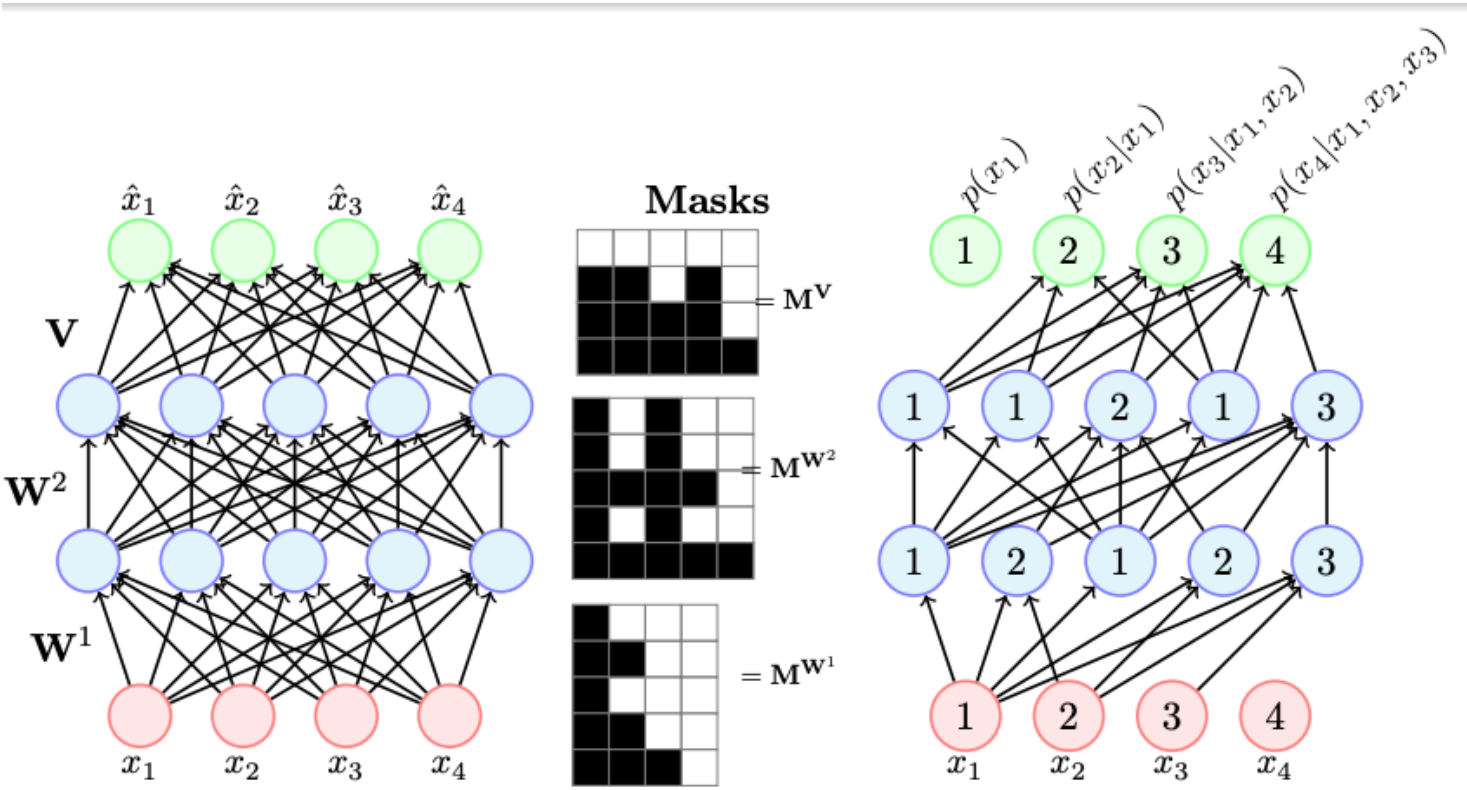
We can drop the connections that are not interesting to us.



How do we reach this?

from M. Khapra Lectures

Masked Autoencoder: Intuition

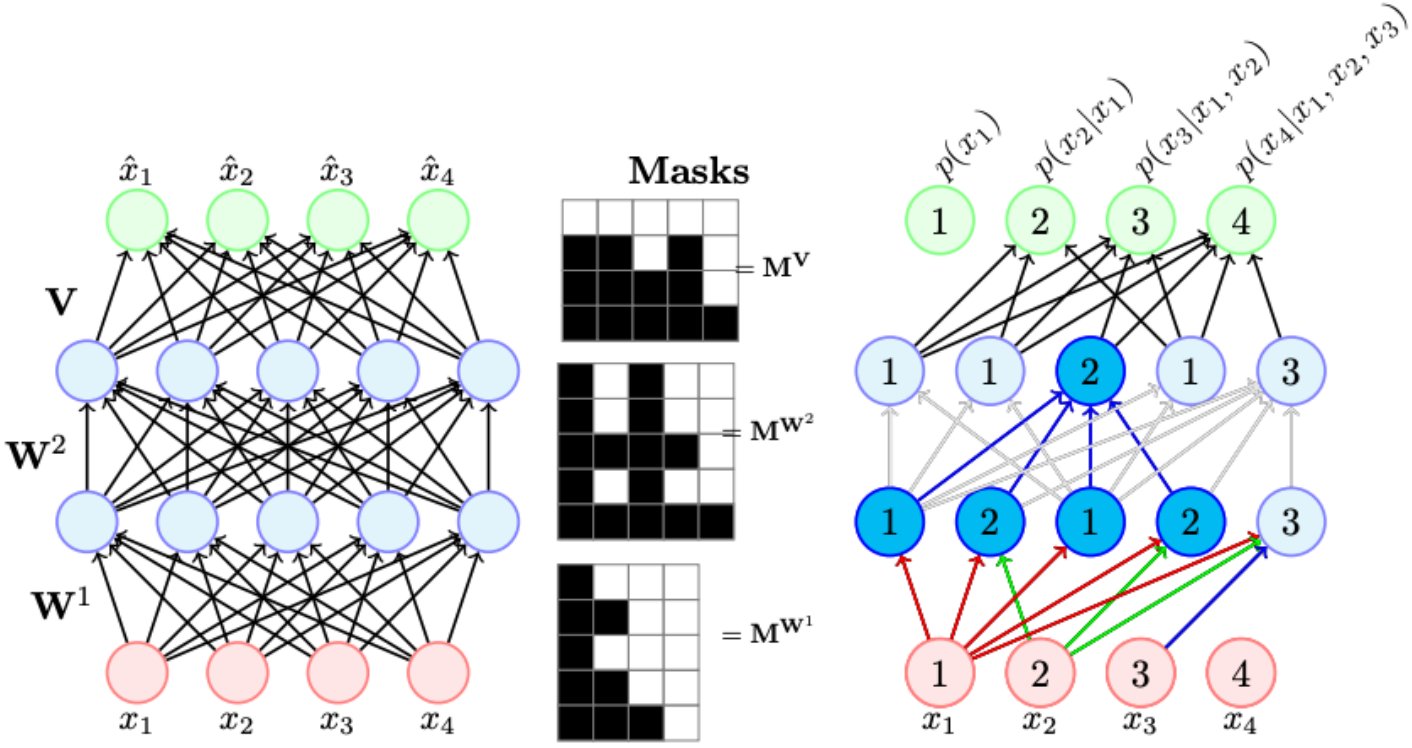


How do we reach this?

- randomly uniformly assign a number $\{1; D-1\}$ to each node.
- we than only keep connections to the nodes with numbers smaller than this node.

from Stanford S. Ermon and A. Grover Write-up

Masked Autoencoder: Intuition

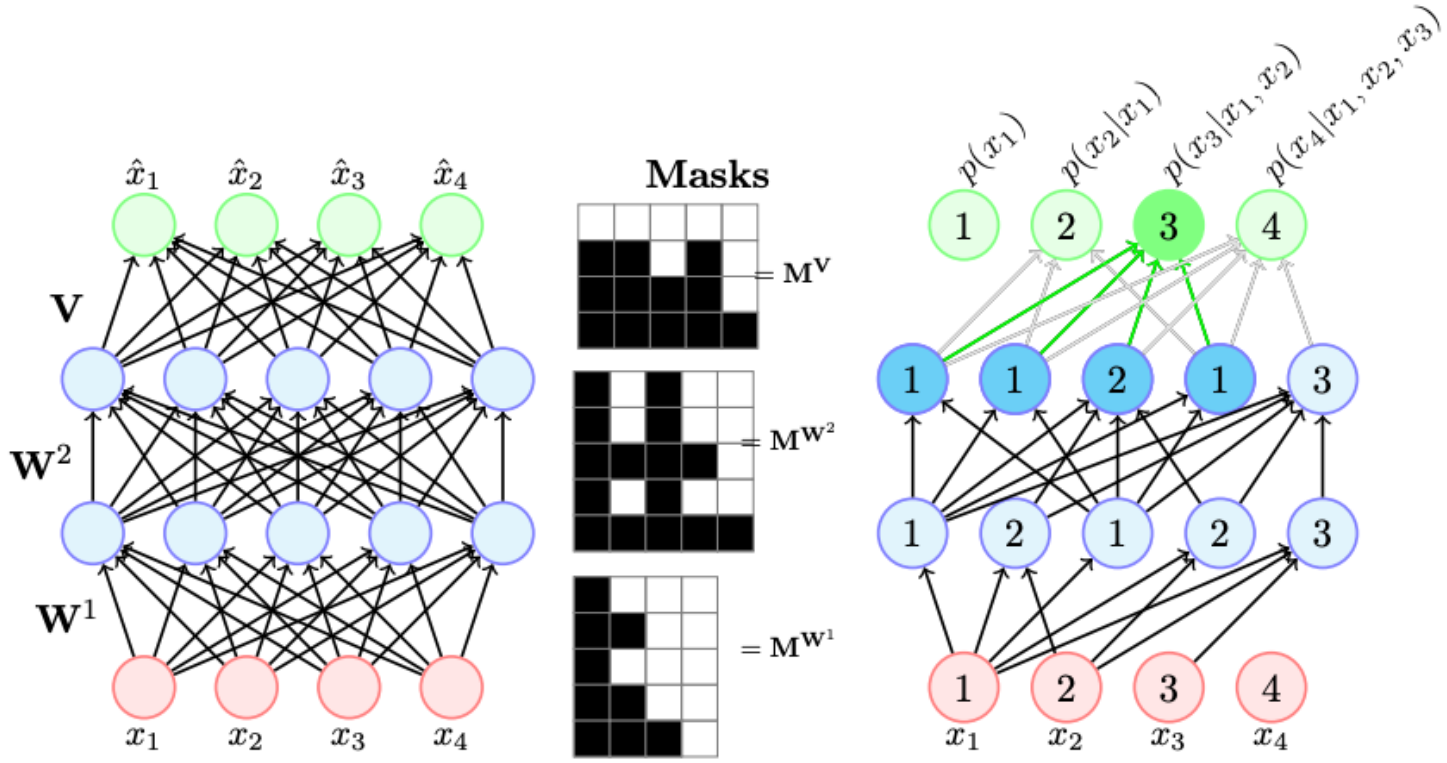


How do we reach this?

- randomly uniformly assign a number $\{1; D-1\}$ to each node.
- we than only keep connections to the nodes with numbers smaller than this node.

from M. Khapra Lectures

Masked Autoencoder: Intuition

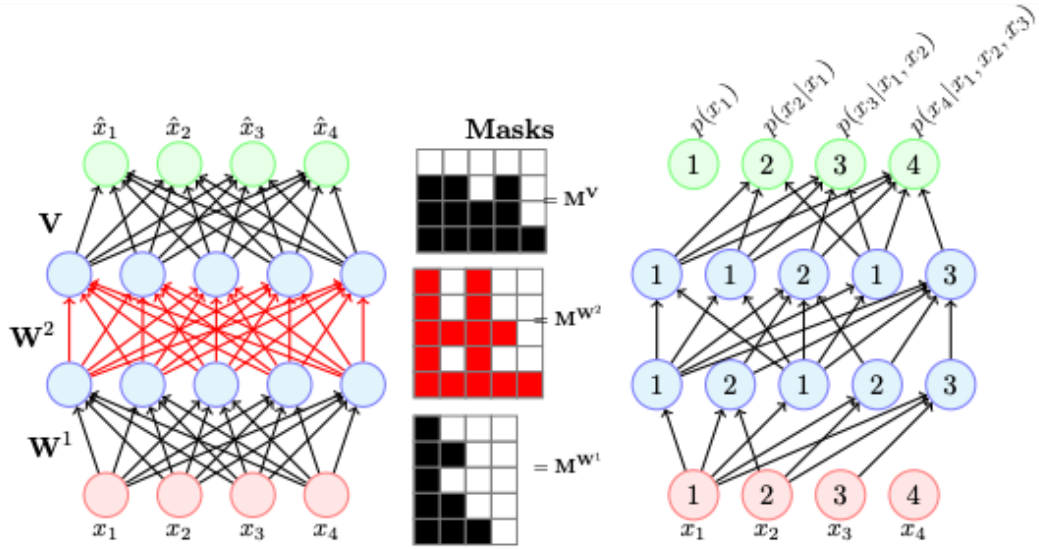


How do we reach this?

- randomly uniformly assign a number $\{1; D-1\}$ to each node.
- we than only keep connections to the nodes with numbers smaller than this node.
- in the last layer we only connections to lower numbers.

Masked Autoencoder: Intuition

Finally, we can write out the mask.



- For example we can apply the following mask at layer 2

$$\begin{bmatrix} W^2_{11} & W^2_{12} & W^2_{13} & W^2_{14} & W^2_{15} \\ W^2_{21} & W^2_{22} & W^2_{23} & W^2_{24} & W^2_{25} \\ W^2_{31} & W^2_{32} & W^2_{33} & W^2_{34} & W^2_{35} \\ W^2_{41} & W^2_{42} & W^2_{43} & W^2_{44} & W^2_{45} \\ W^2_{51} & W^2_{52} & W^2_{53} & W^2_{54} & W^2_{55} \end{bmatrix} \odot \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

from M. Khapra Lectures

MADE: Masked Autoencoder for Distribution Estimation

This is a simple modification to our standard neural networks. Consider a one hidden layer autoencoder with input:

$$h(x) = g(b + (W \odot M^W)x);$$
$$\hat{x} = sigmoid(c + (V \odot M^V))h(x)).$$

Where we added M, the masks that were obtained in the previous slide.
We can also sample over some possible configurations of masks, to have the stability of the training or add a direct connection:

$$\hat{x} = sigmoid(c + (V \odot M^V))h(x) + A \odot M^A x$$

M. Germain et al., MADE: Masked Autoencoder for Distribution Estimation, ICML'15

MADE: Sampling

Sampling requires some efforts (as usual for autoregressive.

1. Randomly generate vector x, set $i = 1$.
2. Feed x into autoencoder and generate outputs for the network, set $p = \hat{x}_i$.
3. Sample from a Bernoulli distribution with parameter p, set input $x_i = Bernoulli(p)$
4. Increment i and repeat steps 2-4 until $i > D$.

MADE: results

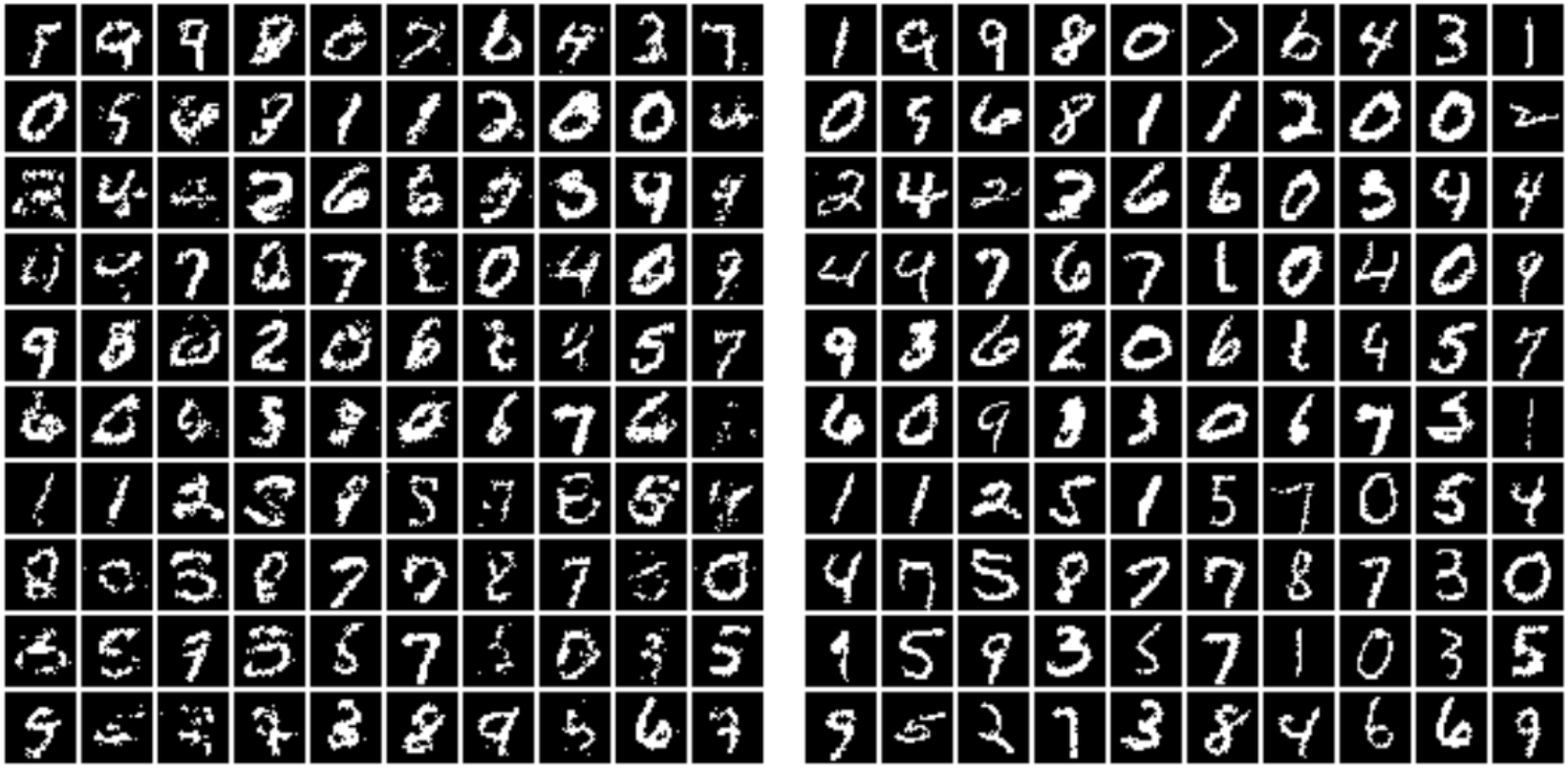


Figure 3. Left: Samples from a 2 hidden layer MADE. Right: Nearest neighbour in binarized MNIST.

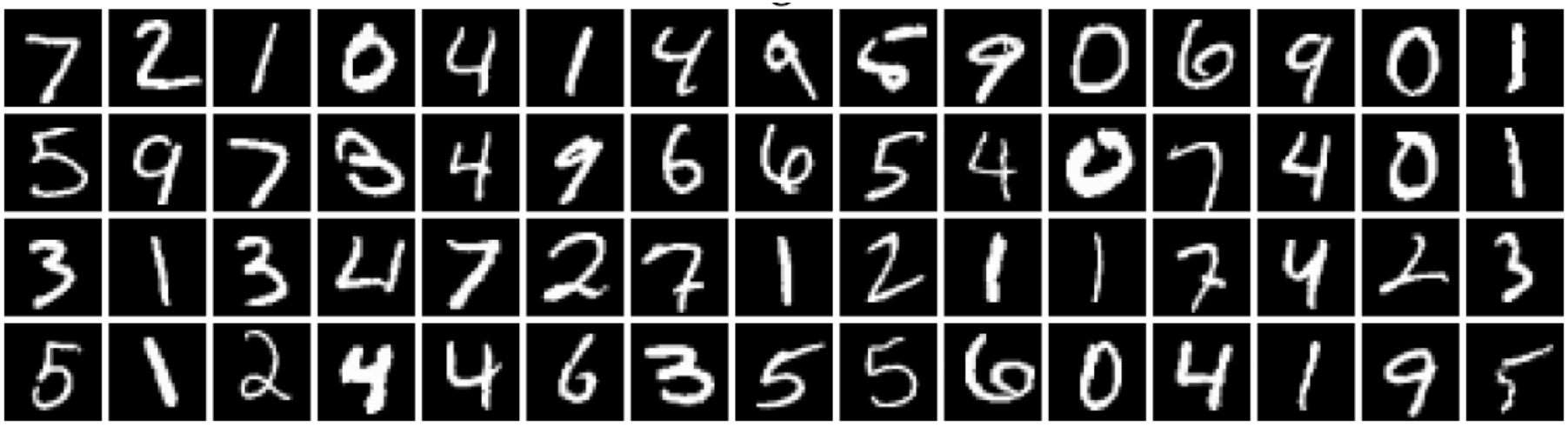
M. Germain et al., MADE: Masked Autoencoder for Distribution Estimation, ICML'15

Summary so far

- Autoencoders are able to find hidden representations of the observables.
- Vanilla Autoencoders are not suitable for generative moding.
- Denoising Autoencoders can produce a sampling model.
- Masked Autoencoders, inspired by autoregressive models, are generative.

Latent Variable Models

Latent Variables - MNIST

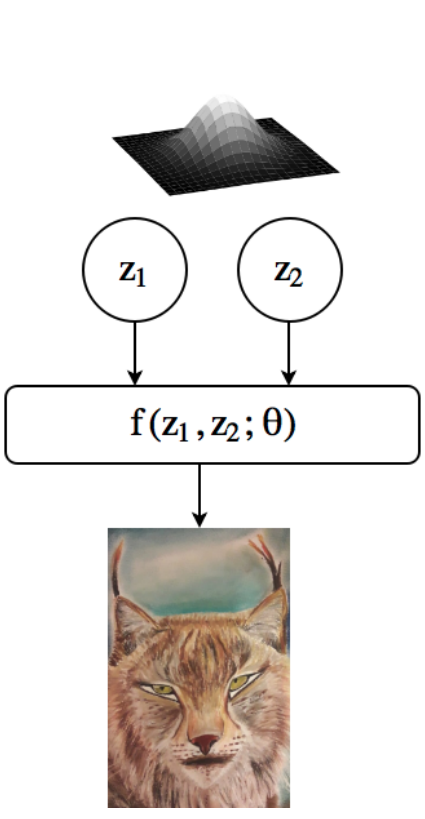


What is common in these images? How can they be characterised?

Latent Variables - Celeb

Variability due to eye and hair color, gender, race. However, unless images are annotated, these factors of variation are not explicitly available (latent).

Latent Variable Models: Motivation



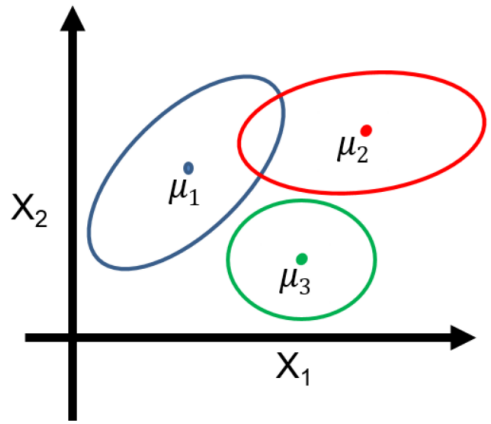
- We only observe the X in the data sample. Latent observables Z are hidden.
- Knowing latent variable space correctly can be beneficial as $p(x|z)$ can be described simpler than $p(x)$.
- Hard to find the real Z space manually, we need unsupervised learning.

Semih Akbayrak's blog

Mixture of Gaussians: a Shallow Latent Variable Model

In fact, we can consider GMM as a latent variable model:

- $z \sim Categorical(1, , K)$
- $p(x|z = k) = N(\mu_k, \Sigma_k)$

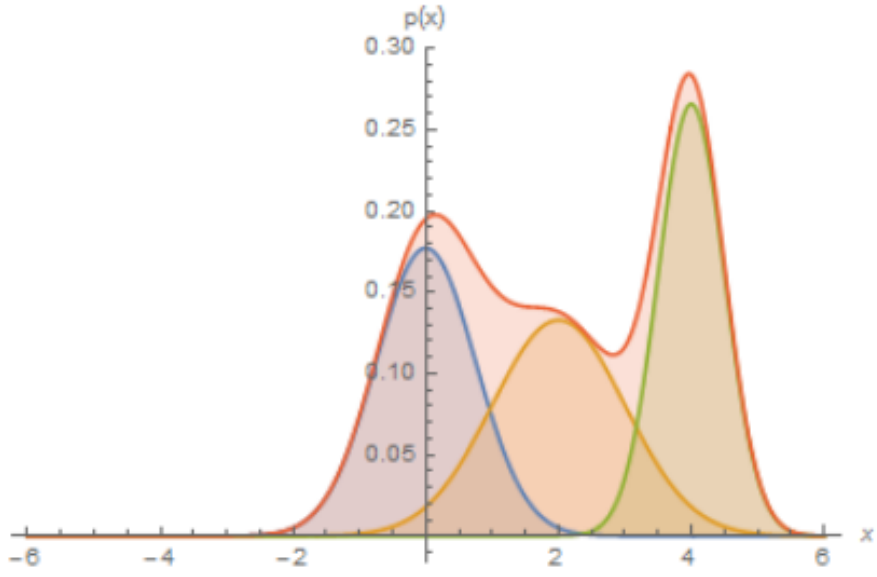


Generative process

1. Pick a mixture component k by sampling z.
2. Generate a data point by sampling from that Gaussian.

GMM: combination

Combine simple models into a more complex and expressive one.



$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{z}} p(\mathbf{z}) p(\mathbf{x} \mid \mathbf{z}) = \sum_{k=1}^{K} p(\mathbf{z} = k) \underbrace{\mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)}_{\text{component}}$$

Generating samples from $p(x)$

What if we have a continuous latent observable? We will need a better understanding of how to calculate integral:

$$p(x;\theta) = \int_z p(x,z;\theta)dz.$$

We need a good estimation of this integral.

Finding parameters θ

How do we find the parameters θ? The most natural way is to maximise the log likelihood:

$$\log \prod_{i=1}^{D} p(x^i; \theta) = \sum_{x \in D} \log p(x; \theta) = \sum_{x \in D} \log \sum_{z} p(x, z; \theta).$$

In the following we will need to:

- define a loss function to optimize
- calculate gradients of the loss function with respect to the parameters θ.
- apply a variant of stochastic gradient descent.

from UCLA DL lectures buy Kao

Maximizing the likelihood

Consider one example, x. Then to maximize the likelihood, we have to calculate:

$$p(x;\theta) = \int_z p(x,z;\theta)dz = \int_z p(x;\theta|z)p(z)dz$$

There are a few cases when we know how to do this integral.

- If $x|z$ and z are normal, and further if x is a linear function of z, we can use EM algorithm.

In general, if x is a nonlinear function of z then the distribution, $p(x,z;\theta)$ is not something we can easily write analytically. Thus, in general, calculating $p_\theta(x)$ is intractable and we thus cannot directly maximize the likelihood.

Estimating likelihood: Naive Monte-Carlo

We can use the same algorithm as we have seen in GMM:

$$p(x;\theta) = \sum_{allz} p(x,z;\theta) = |Z|\mathbb{E}_{z\sim Uniform(z)}(p(x,z;\theta)).$$

For this, we can fix the distribution followed by z.
This is really only tractable when x is relatively low-dimensional. In case of image, we run into the curse of dimensionality, where we need to grab many samples to get an accurate view of x.

Wrap-up

We have problems:

- We can't calculate $p(x)$.
- Hence, we can't write the maximum-likelihood objective.

We need a different approach or objective function. Idea:

- If we can't write likelihood, let's instead derive a lower bound on it.
- If the lower bound is tractable, then we can optimize the parameters with respect to the lower bound.
- If we are making the lower bound larger, we are making the likelihood larger.

Deriving the Evidence Lower BOund

With this intuition, we're prepared to derive the ELBO. For the sake of simplicity, we'll assume here $\mathbf{x}$ is a sample, $\mathbf{x}^{(i)}$. (i.e., for these slides, we'll calculate the ELBO for one example $\mathbf{x}^{(i)}$, but I will drop the superscript (i) for the sake of brevity.)

First, we note that:

$$\log p_\theta(\mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x})$$

We can do this because $\log p_\theta(\mathbf{x})$ has no dependence on $\mathbf{z}$; and so by the linearity of the expectation, $\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}) = \log p_\theta(\mathbf{x})\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} 1$, which is $\log p_\theta(\mathbf{x})$. For shorthand, we'll let $\mathbb{E}_{\mathbf{z}}$ denote $\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}$.

Deriving the ELBO

We continue from here:

$$
\begin{aligned}
\log p_\theta(\mathbf{x}) &= \mathbb{E}_{\mathbf{z}} \log p_\theta(\mathbf{x}) \\
&\overset{(a)}{=} \mathbb{E}_{\mathbf{z}} \log \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})} \\
&\overset{(b)}{=} \mathbb{E}_{\mathbf{z}} \log \left(\frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})} \frac{q(\mathbf{z}|\mathbf{x})}{q(\mathbf{z}|\mathbf{x})} \right) \\
&\overset{(c)}{=} \mathbb{E}_{\mathbf{z}} \log p(\mathbf{x}|\mathbf{z}) - \mathbb{E}_{\mathbf{z}} \log \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})} + \mathbb{E}_{\mathbf{z}} \log \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} \\
&\overset{(d)}{=} \mathbb{E}_{\mathbf{z}} \log p(\mathbf{x}|\mathbf{z}) - \mathrm{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) + \mathrm{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) \\
&\geq \mathbb{E}_{\mathbf{z}} \log p(\mathbf{x}|\mathbf{z}) - \mathrm{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))
\end{aligned}
$$

where we used the following facts: (a) is the chain rule for probability, (b) is multiplication by 1 expressed as $q(\mathbf{z}|\mathbf{x})/q(\mathbf{z}|\mathbf{x})$, (c) is expanding the logarithm and using the linearity of the expectation, (d) is using the definition of KL divergence. The final inequality comes from the fact that

$$\mathrm{KL}(q(\mathbf{z}|x)||p(\mathbf{z}|x) \geq 0$$

Notice that this term encapsulates our penalty in estimating the log-likelihood of the data from using $q(\mathbf{z}|\mathbf{x})$ to approximate $p(\mathbf{z}|\mathbf{x})$

Deriving the ELBO

This is then our lower bound on the log-likelihood:

$$\begin{aligned} \log p_\theta(\mathbf{x}) &\geq \mathbb{E}_{\mathbf{z}} \log p(\mathbf{x}|\mathbf{z}) - \mathrm{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \\ &= \mathcal{L}_{\mathrm{vae}}(\mathbf{x}) \end{aligned}$$

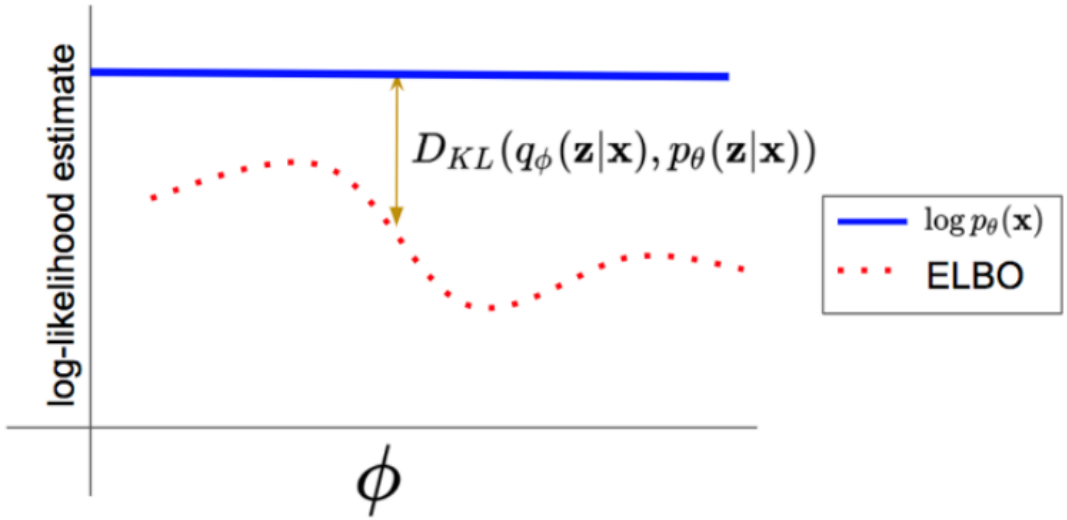
This is great news, because we now have a loss function (the lower bound, $\mathcal{L}_{\mathrm{vae}}(\mathbf{x})$) that is tractable in the following manner:

- The term $\mathbb{E}_{\mathbf{z}} \log p(\mathbf{x}|\mathbf{z})$ can be approximated from data using a minibatch, almost exactly like we do for the softmax loss. We take some minibatch of m examples $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}$, and for each $\mathbf{x}^{(i)}$, we calculate $q(\mathbf{z}|\mathbf{x}^{(i)})$, sample $\mathbf{z}$ from it, and then calculate $\log p(\mathbf{x}^{(i)}|\mathbf{z})$.
- The KL divergence has a closed form when $q()$ and $p()$ are Gaussian distributions. That is,

$$\mathrm{KL}(\mathcal{N}(\mu_0, \Sigma_0), \mathcal{N}(\mu_1, \Sigma_1)) = \frac{1}{2}\left[\mathrm{tr}(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T \Sigma_1^{-1}(\mu_1 - \mu_0) - d + \log \frac{\det \Sigma_1}{\det \Sigma_0}\right]$$

Graphical representation

In general, q can also be dependent on some parameter set ϕ. The final optimised boundary can be visualised:



Our next goal is learn how to fit ϕ and θ simultanously in variational autoencoder.