Suppose again we have a data set

$$\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$$

drawn from an unknown data distribution $p(\mathbf{x})$.

The goal of generative modeling is to fit a model of the data distribution such that we can synthesize new data points by sampling from the model.

In order to build a model we need a way to represent a probability density function

A common way to represent a density function is to write it as

$$p_\theta(\mathbf{x}) = \frac{\exp(-f_\theta(\mathbf{x}))}{\int \exp(-f_\theta(\mathbf{x}))d\mathbf{x}}$$

where $f_\theta : \mathbb{R}^d \to \mathbb{R}$ is some function such that the integral

$$Z_\theta = \int_{\mathbb{R}^d} \exp(-f_\theta(\mathbf{t}))d\mathbf{t}$$

is finite. The function f_θ is often called the *Energy Function* and Z_θ the *Partition Function*

The numerator is always positive and dividing by the integral assures that

$$\int p_\theta(\mathbf{x})d\mathbf{x} = 1$$

so the expression satisfies the conditions for being a valid probability density function (pdf).

Any density function $p_{\mathbf{X}}$ can be represented this way by taking

$$f(\mathbf{x}) = -\log p_{\mathbf{X}}(\mathbf{x})$$

We estimate parameters by Maximum Likelihood i.e. by computing

$$\operatorname*{argmax}_\theta \prod_{\mathbf{x}\in\mathcal{D}} p_\theta(\mathbf{x})$$

A *Markov network* is an undirected graph where the nodes are random variables

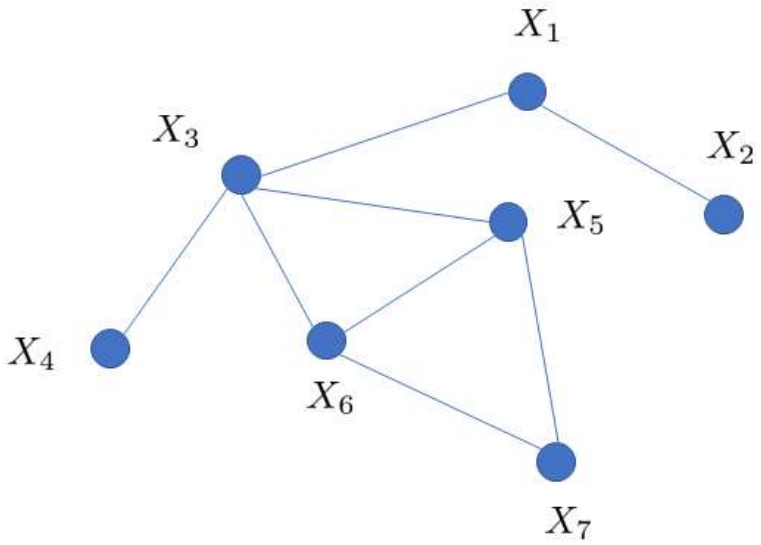

Figure 1:

The edges indicate dependencies between the random variables

We are interested in the joint distribution

$$p(X_1, X_2, \ldots, X_7)$$

Since there are no particular order and direction in the graph, ordinary conditional distributions are not so useful. Instead we shall use a concept called *factors*.

If $\mathbb{D} = \{X_i\}_{\imath \in I}$ is a set of random variables, a factor is a function $\phi(\{X_i\}_{i \in I}$ from the set of values $val(\mathbb{D}) = \prod val(X_i) \to \mathbb{R}$. For instance if each X_i takes values in $\mathbb{R}$, $val(\mathbb{D}) = \mathbb{R}^I$. If ϕ only takes positive values we say it is a positive factor.

A common way to make a factors is from an *Energy* function

$$E : val(\mathbb{D}) \to \mathbb{R}$$

and then define a factor by $\phi = e^{-E}$.

Given factors $\phi_1(X, Y)$ and $\phi_2(Y, Z)$ we define their product $\phi_3(X, Y, Z)$ as the function

$$\phi_3(x, y, z) = \phi_1(x, y)\phi_2(y, z)$$

If the factors are defined by energy functions this corresponds to

$$E_3(x, y, z) = E_1(x, y) + E_2(y, z)$$

A *clique* is a complete subgraph i.e. one where any two vertices are connected.

A maximal clique is a clique which cannot be made larger by adding any vertex.

Thus for our graph the maximal cliques are: $\mathbb{D}_1 = \{X_1, X_2\}$, $\mathbb{D}_2 = \{X_1, X_3\}$, $\mathbb{D}_3 = \{X_3, X_4\}$, $\mathbb{D}_4 = \{X_3, X_5, X_6\}$, $\mathbb{D}_5 = \{X_5, X_6, X_7\}$

A clique factorization is the product of factors, one for each maximal clique

The *Gibbs Distribution* P, parametrized by the clique factors $\{\phi_1(\mathbb{D}_1), \phi_2(\mathbb{D}_2), \ldots, \phi_K(\mathbb{D}_K)\}$ is the distribution defined by

$$P(X_1, X_2, \ldots, X_m) = \frac{1}{Z}\phi_1(\mathbb{D}_1) \cdot \phi_2(\mathbb{D}_2) \cdot \cdots \cdot \phi_K(\mathbb{D}_K)$$

where

$$Z = \int \phi_1(\mathbb{D}_1) \cdot \phi_2(\mathbb{D}_2) \cdot \cdots \cdot \phi_K(\mathbb{D}_K)$$

In our case the Gibbs Distribution of the Markov Network is

$$P(X_1, X_2, \ldots, X_7) = \frac{\phi_1(X_1, X_2)\phi_2(X_1, X_3)\phi_3(X_3, X_4)\phi_4(X_3, X_5, X_6)\phi_5(X_5, X_6, X_7)}{Z}$$

The factorization does imply certain independences. Indeed we have the following result:

Let $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ be disjoint sets of random variables. Then $\mathbf{X}, \mathbf{Y}$ are independent conditional on $\mathbf{Z}$ if and only if $P(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$ factors as

$$P(\mathbf{X}, \mathbf{Y}|\mathbf{Z}) = P(\mathbf{X}|\mathbf{Z})P(\mathbf{Y}|\mathbf{Z})$$

(Notation: $\mathbf{X} \perp \mathbf{Y}|\mathbf{Z}$)

In terms of factors:

$$\phi(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) = \phi_1(\mathbf{X}, \mathbf{Z})\phi_2(\mathbf{Y}, \mathbf{Z})$$

In our case the clique factorization implies

1. $X_2 \perp X_3|X_1$
2. $X_1 \perp X_4|X_3$
3. $X_4 \perp X_5, X_6|X_3$
4. $X_3 \perp X_7|X_5, X_6$

These conditions imply factorizations of the conditional probabilities

1. $P(X_2, X_3|X_1) = P(X_2|X_1)P(X_3|X_1)$
2. $P(X_1, X_4|X_3) = P(X_1|X_3)P(X_4|X_3)$
3. $P(X_4, X_5, X_6|X_3) = P(X_4|X_3)P(X_5, X_6|X_3)$
4. $P(X_3, X_7|X_5, X_6) = P(X_3|X_5, X_6)P(X_7|X_5, X_6)$

We consider an example, a model known as a *Restricted Bolzman Machine (RBM)*

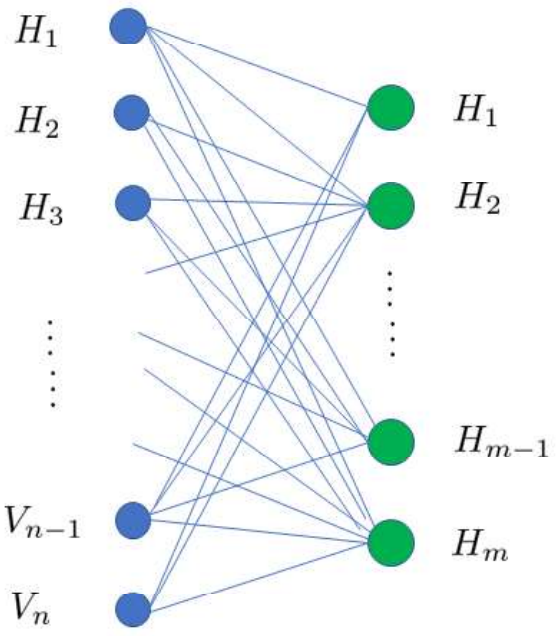

Figure 2:

This is a *bi-partite* graph. The vertices on the left are called the *visible* nodes and those on the right the *hidden* or *latent* nodes. There are no connections between two hidden nodes or between two visible nodes. Each visible node is connected to all the hidden nodes and each hidden node is connected to all the visible nodes.

The maximal cliques are $\{V_i, H_j\}_{i\in I, j\in J}$ so the Gibbs factorization is

$$P(\{V_i\}_{i\in I}, \{H_j\}_{j\in J}) = \prod_{i,j} P(V_i, H_j)$$

$V_i \perp V_j \,|\, H_k$ for all i, j, k and similarly $H_i \perp H_j \,|\, V_k$

We define an energy function E as follows: let W be an $m \times n$ matrix and let $\mathbf{b}$ and $\mathbf{c}$ be *bias vectors*.

We consider the case where both the V's and the H's are Bernoulli variables i.e. they only take the values $\{0, 1\}$.

In the following

We define

$$E(v_i, h_j) = -v_i\, w_{ij}\, h_j - b_i v_j - c_j h_j$$

Given a sample $\mathbf{v}$ of $\mathbf{V} = (V_1, V_2, \ldots, V_n)$ and a sample $\mathbf{h}$ of the latent variable $\mathbf{H} = (H_1, H_2, \ldots, H_m)$ we can write the total energy

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{v} \cdot W \cdot \mathbf{h}^T - \mathbf{b} \cdot \mathbf{v} - \mathbf{c} \cdot \mathbf{h}$$

Thus the likelihood of the sample $(\mathbf{v}, \mathbf{h})$ is

$$\frac{\exp(-E(\mathbf{v}, \mathbf{h}))}{\sum\limits_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{v}', \mathbf{h}'))}$$

Now assume we have a data set $\mathcal{D} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N\}$. We want to optimize the log-likelihood

$$\begin{aligned}
\mathcal{L}(\mathcal{D}) &= \sum_{\mathbf{v}\in\mathcal{D}} \log P(\mathbf{v}) \\
&= \sum_{\mathbf{v}\in\mathcal{D}} \log \sum_{\mathbf{h}} P(\mathbf{v}, \mathbf{h}) \\
&= \sum_{\mathbf{v}\in\mathcal{D}} \log \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h})) - N \log Z \\
&= \sum_{\mathbf{v}\in\mathcal{D}} \left(\log \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}) \right) - \log \sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{x}, \mathbf{h}))
\end{aligned}$$

To maximize the log-likelihood with respect to the parameters $\theta = (W, \mathbf{b}, \mathbf{c})$ we need to compute the gradients (to simplify notation we just compute for a single data point $\mathbf{v}$)

$$\begin{aligned}\nabla_\theta \mathcal{L}(\mathbf{v}) &= \frac{\nabla_\theta \left(\sum_{\mathbf{h}} \exp(-E_\theta(\mathbf{v}, \mathbf{h}) \right)}{\sum_{\mathbf{h}} \exp(-E_\theta(\mathbf{v}, \mathbf{h}))} - \frac{\nabla_\theta \left(\sum_{\mathbf{v},\mathbf{h}} \exp(-E_\theta(\mathbf{v}, \mathbf{h}) \right)}{\sum_{\mathbf{v},\mathbf{h}} \exp(-E_\theta(\mathbf{v}, \mathbf{h})} \\ &= \frac{\sum_{\mathbf{h}} - \exp(-E_\theta(\mathbf{v}, \mathbf{h})) \nabla_\theta E_\theta(\mathbf{v}, \mathbf{h})}{\sum_{\mathbf{h}} \exp(-E_\theta(\mathbf{v}, \mathbf{h}))} \\ &\quad - \frac{\sum_{\mathbf{v},\mathbf{h}} - \exp(-E_\theta(\mathbf{v}, \mathbf{h})) \nabla_\theta E_\theta(\mathbf{v}, \mathbf{h})}{\sum_{\mathbf{v},\mathbf{h}} \exp(-E_\theta(\mathbf{v}, \mathbf{h}))}\end{aligned}$$

We have

$$P(\mathbf{h}|\mathbf{v}) = \frac{P(\mathbf{h}, \mathbf{v})}{P(\mathbf{v})} = \frac{\frac{1}{Z} \exp(-E(\mathbf{v}, \mathbf{h}))}{\frac{1}{Z} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))} = \frac{\exp(-E(\mathbf{h}, \mathbf{v}))}{\sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))}$$

and

$$P(\mathbf{v}, \mathbf{h}) = \frac{\exp(-E(\mathbf{v}, \mathbf{h}))}{Z}$$

Thus

$$\begin{aligned}-\sum_{\mathbf{h}} \frac{\exp(-E(\mathbf{v}, \mathbf{h}))}{\sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h})) \nabla_\theta E(\mathbf{v}, \mathbf{h})} &= -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}) \nabla_\theta E(\mathbf{h}, \mathbf{v}) \\ &= -\mathbb{E}_{P(\mathbf{h}|\mathbf{v})}(\nabla_\theta E(\mathbf{v}, \mathbf{h}))\end{aligned}$$

and

$$-\sum_{\mathbf{v},\mathbf{h}} \frac{\exp(-E(\mathbf{v}, \mathbf{h})) \nabla_\theta E(\mathbf{v}, \mathbf{h})}{\sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{x}, \mathbf{h})) d\mathbf{h} d\mathbf{x}} = -\mathbb{E}_{P(\mathbf{x},\mathbf{h})}(\nabla_\theta E(\mathbf{x}, \mathbf{h})))$$

So the negative gradient of the log-likelihood

$$-\nabla_\theta \mathcal{L}(\mathcal{D}) = \sum_{\mathbf{v} \in \mathcal{D}} \mathbb{E}_{P(\mathbf{h}|\mathbf{v})}(\nabla_\theta E(\mathbf{v}, \mathbf{h})) - N \mathbb{E}_{P(\mathbf{v},\mathbf{h})}(\nabla_\theta E(\mathbf{v}, \mathbf{h}))$$

Since the hidden variables are independent conditional on the visible variables

$$P(\mathbf{h}|\mathbf{v}) = \prod_j P(h_j|\mathbf{v})$$

Each h_j is a sample from a Bernoulli distribution hence is either 0 or 1 and so

$$\mathbb{E}_{P(\mathbf{h}|\mathbf{v})}(v_i h_j) = \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}) v_i h_j = \sum_{\mathbf{h}, h_j=1} P(\mathbf{h}|\mathbf{v}) v_i = P(h_j = 1|\mathbf{v}) v_i$$

The other term is

$$\mathbb{E}_{P(\mathbf{v},\mathbf{h})}(v_i h_j) = \sum_{\mathbf{v},\mathbf{h}} P(\mathbf{v}, \mathbf{h}) v_r h_s$$

Remark that each $\mathbf{h}$ is an allocation of m 0's and 1's and hence there are 2^m possible values for $\mathbf{h}$. Similarly there are 2^n possible values for $\mathbf{v}$ (but $\mathbf{v}$ is a fixed vector of n 0's and 1's)

Let $\sigma(z) = \dfrac{1}{1+\exp(-z)}$ be the sigmoid (or the logistic) function

Consider

$$P(h_j = 1|\mathbf{v}) = \frac{P(h_j = 1, \mathbf{v})}{P(\mathbf{v})} = \frac{\exp(-E(h_j = 1, \mathbf{v}))}{\exp(-E(h_j = 0, \mathbf{v})) + \exp(-E(h_j = 1, \mathbf{v})}$$
$$= \frac{\exp(\sum_i (v_i w_{ij} + b_i v_i) + c_j)}{\exp(\sum_i b_i v_i)) + \exp(\sum_i (v_i w_{ij} + b_i v_i) + c_j)}$$
$$= \frac{1}{\exp(-(\sum_i v_i w_{ij} + c_j)) + 1} = \sigma(\sum_i v_i w_{ij} + c_j)$$

Similarly

$$P(v_i = 1|\mathbf{h}) = \frac{\exp(w_{ij} \cdot \mathbf{h}^T + b_i + \mathbf{c} \cdot \mathbf{h})}{\exp(\mathbf{c} \cdot \mathbf{h}) + \exp(w_{ij} \cdot \mathbf{h}^T + b_i + \mathbf{c} \cdot \mathbf{h}} = \sigma(\sum_j w_{ij} h_j + b_i)$$

We can sample from the joint distribution $P_\theta(\mathbf{V}, \mathbf{H})$ as follows

Start with a data point $\mathbf{v} = (v_1, v_2, \ldots, v_n)$, the v_j are 0 or 1

The distribution

$$P_\theta(H_1, H_2, \ldots, H_n | V_1 = v_1, V_2 = v_2, \ldots)$$
$$= \prod_j P_\theta(H_j | V_1 = v_1, V_2 = v_2, \ldots)$$

and each of the distributions

$$P_\theta(H_j | V_1 = v_1, V_2 = v_2, \ldots)$$

are Bernoulli with parameter

$$\sigma(\mathbf{v}W|j + c_j)$$

Now we can sample $\mathbf{h} = (h_1, h_2, \ldots, h_m)$ with each

$$h_j \sim Bernoulli(\sigma(\mathbf{v}\mathbf{W}|j + c_j))$$

The distribution

$$\begin{aligned} &P(V_1, V_2, \ldots, V_n | H_1 = h_1, H_2 = h_2, \ldots, H_m = h_m) \\ &= \prod_i P(V_i | H_1 = h_1, H_2 = h_2, \ldots, H_m = h_m) \end{aligned}$$

and each of the distributions in the product are Bernoulli with parameter

$$\sigma(\mathbf{W}_{-i} \cdot \mathbf{h} + b_i)$$

Now sample $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ with each

$$v_i \sim Bernoulli(\sigma(W_{-i} \cdot \mathbf{h} + b_i))$$

Continuing this way $(\mathbf{v}_\nu, \mathbf{h}_{\nu-1})$ will be a sample from $P_\theta(\mathbf{V}, \mathbf{H})$ for ν sufficiently large

This sampling method is called the *Gibbs Sampler*

Let P_{data} denote the true distribution of the data then

$$\mathbb{E}_{P_{data}(\mathbf{v})}(\mathbb{E}_{P_\theta(\mathbf{h}|\mathbf{v})}(\nabla_\theta E(\mathbf{v}, \mathbf{h}))) \equiv \frac{1}{N} \sum_{\mathbf{v} \in \mathcal{D}} \mathbb{E}_{P(\mathbf{h}|\mathbf{v})}(\nabla_\theta E(\mathbf{v}, \mathbf{h}))$$

Starting with the $\mathbf{v} \in \mathcal{D}$ and applying the Gibbs sampler sufficiently many times to get a pair

$$(\mathbf{v}', \mathbf{h}') \sim P_\theta$$

Thus

$$\mathbb{E}_{P(\mathbf{v},\mathbf{h})}(\nabla_\theta E(\mathbf{v}, \mathbf{h})) \equiv \frac{1}{N} \sum_{\mathbf{v} \in \mathcal{D}} \nabla_\theta E(\mathbf{v}', \mathbf{h}') \equiv \mathbb{E}_{P_{data}}(\nabla_\theta E(\mathbf{v}', \mathbf{h}'))$$

It follows that

$$\frac{1}{N} \nabla_\theta \mathcal{L}(\mathcal{D}) \equiv \mathbb{E}_{P_{data}(\mathbf{v})}(\mathbb{E}_{P_\theta(\mathbf{h}|\mathbf{v})}(\nabla_\theta E(\mathbf{v}, \mathbf{h}))) - \mathbb{E}_{P_{data}}(\nabla_\theta E(\mathbf{v}', \mathbf{h}'))$$

In practice it turns out that it is enough to take a few or even just one, Gibbs sampler step (this is *Contrastive Divergence*), so approximately

$$-\nabla_\theta \mathcal{L}(\mathcal{D}) \equiv \sum_{\mathbf{v}\in\mathcal{D}} \sum_{\mathbf{h}} P_\theta(\mathbf{h}|\mathbf{v}) \nabla_\theta E(\mathbf{v},\mathbf{h}) - \sum_{\mathbf{v}\in\mathcal{D}} \nabla_\theta E(\mathbf{v}',\mathbf{h}'))$$

We further approximate $\mathbb{E}_{P(\mathbf{h}|\mathbf{v})}(\nabla_\theta E(\mathbf{v},\mathbf{h}))$ with a single sample from $P(\mathbf{h}|\mathbf{v})$ (this is the 0'th step in the Gibbs Sampler starting with $\mathbf{v}$)

All together the Contrastive Divergence algorithm gives

$$-\nabla_\theta \mathcal{L}(\mathcal{D}) \equiv \nabla_\theta \left(\sum_{\mathbf{v}\in\mathcal{D}} E(\mathbf{v},\mathbf{h}) - E(\mathbf{v}',\mathbf{h}') \right)$$

It follows that we can use

$$-\nabla_\theta \mathcal{L}((\mathcal{D}) \equiv \left(\sum_{\mathbf{v}\in\mathcal{D}} E(\mathbf{v},\mathbf{h}) - E(\mathbf{v}',\mathbf{h}') \right)$$

as a loss function and minimize using gradient descent

1 Variational Autoencoders

1.1 Autoencoders

An Autoencoders has two parts, an *Encoder* and a *Decoder*. In the Encoder part the model outputs from a data point $\mathbf{x}$, a latent vector $\mathbf{z} = Enc(\mathbf{z})$. Typically $\dim \mathbf{z} < \dim \mathbf{x}$ so we can view $\mathbf{z}$ as a compressed version of $\mathbf{x}$.

The Decoder part inputs the latent $\mathbf{z}$ and outputs a vector $Dec(\mathbf{z}) = \widetilde{\mathbf{x}}$ with $\dim \widetilde{\mathbf{x}} = \dim \mathbf{x}$ and the model is trained so that $\widetilde{\mathbf{x}}$ is close to the original data point $\mathbf{x}$.

Thus an Autoencoder can be thought of as a way to compress the original data to lower dimension such that the original data can be reconstructed from the compressed version.

Autoencoders are quite simple to code.

```
comp_dim = 8
```

```
Encoder = nn.Sequential(
            nn.Linear(784,256),
            nn.ReLU(),
            nn.Linear(256,64),
            nn.Tanh(),
            nn.Linear(64,comp_dim))

Decoder = nn.Sequential(
            nn.Linear(comp_dim,64),
            nn.Tanh(),
            nn.Linear(64,256),
            nn.Tanh(),
            nn.Linear(256,784),
            nn.Sigmoid()
)

```

Figure 3:

Here the Encoder compresses the 728-dimensional image to an 8-dimensional vector just using 3 `nn.Linear` layers, a `nn.ReLU()` and a `nn.Tanh()`, activations.

The Decoder inputs an 8-dimensional vector and sends it through 3, `nn.Linear` layers and two `nn.Tanh()` activation layers. The last layer is a `nn.Sigmoid()` layer that takes the 728-dimensional output of the last linear layer and applies the *Sigmoid* function that squeezes all coordinates between 0 and 1

Here is the original image and the reconstructed image. Given the severe compression ($728 \to 8$) the reconstructed image is quite close to the original.

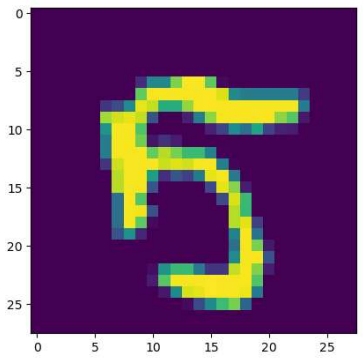

Figure 4: Original

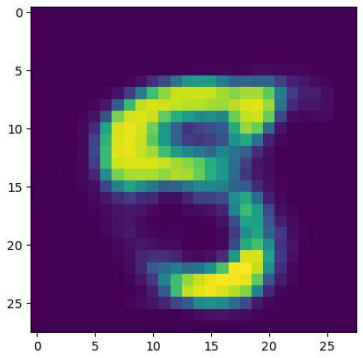

Figure 5: Reconstructed

The compression and reconstruction are both deterministic i.e. there is no randomness.

1.2 Variational Autoencoders

The *Variational Autoencoder* introduces randomness in both Encoder and Decoder.

The idea is that the data are samples from some conditional distribution

$$p(\mathbf{x}|\mathbf{z})$$

The Encoder generates the distribution of the latent variable $\mathbf{z}$ dependent on the data point

$$q_\phi(\mathbf{z}|\mathbf{x})$$

The Decoder generates the reconstruction distribution

$$p_\theta(\mathbf{x}|\mathbf{z})$$

where

$$\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$$

is a sample from the latent variable distribution.

ϕ and θ denote the parameters of the Encoder and Decoder networks resp.

Here is how it works: we choose parametrized distributions q_θ and p_ϕ and we then optimize the parameters θ and ϕ such that for each $\mathbf{x}$ in the data set, we sample $\mathbf{z}$ from the distribution $q_\phi(\mathbf{z}|\mathbf{x})$ and then maximize the log-likelihood

$$\log p_\theta(\mathbf{x})$$

Now let's write

$$\begin{aligned}\log p_\phi(\mathbf{x}) &= \log p_\phi(\mathbf{x}) \int q_\theta(\mathbf{z}|\mathbf{x}) d\mathbf{z} = \int \log p_\phi(\mathbf{x}) q_\theta(\mathbf{z}|\mathbf{x}) d\mathbf{z} \\ &= \int \log \frac{p_\phi(\mathbf{x}) p_\phi(\mathbf{z}|\mathbf{x})}{p_\phi(\mathbf{z}|\mathbf{x})} q_\theta(\mathbf{z}|\mathbf{x}) d\mathbf{z} = \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}\left(\log \frac{p_\phi(\mathbf{x},\mathbf{z})}{p_\phi(\mathbf{z}|\mathbf{x}))}\right) \\ &= \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}\left(\log \frac{p_\phi(\mathbf{x},\mathbf{z}) q_\theta(\mathbf{z}|\mathbf{x})}{p_\phi(\mathbf{z}|\mathbf{x}) q_\theta(\mathbf{z}|\mathbf{x})}\right) \\ &= \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}\left(\log \frac{p_\phi(\mathbf{x},\mathbf{z})}{q_\theta(\mathbf{z}|\mathbf{x})}\right) + \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}\left(\log \frac{q_\theta(\mathbf{z}|\mathbf{x})}{p_\phi(\mathbf{z}|\mathbf{x})}\right)\end{aligned}$$

The second term we can write

$$\mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}\left(\log \frac{q_\theta(\mathbf{z}|\mathbf{x})}{p_\phi(\mathbf{z}|\mathbf{x})}\right) = -\mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}\left(\log \frac{p_\phi(\mathbf{z}|\mathbf{x})}{q_\theta(\mathbf{z}|\mathbf{x})}\right) = D_{KL}(q_\theta(\mathbf{z}|\mathbf{x})||p_\phi(\mathbf{z}|\mathbf{x}))$$

It follows that we can express the log-likelihood as

$$\log p_\phi(\mathbf{x}) = \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}\left(\log \frac{p_\phi(\mathbf{x},\mathbf{z})}{q_\theta(\mathbf{z}|\mathbf{x})}\right) + D_{KL}(q_\theta(\mathbf{z}|\mathbf{x})||p_\phi(\mathbf{z}|\mathbf{x}))$$

The first term $\mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}\left(\log \frac{p_\phi(\mathbf{x},\mathbf{z})}{q_\theta(\mathbf{z}|\mathbf{x})}\right)$ is known as the $ELBO$ (**E***vidence***L***ower***BO***und*) ($\log p_\phi(\mathbf{x})$ is known as the *evidence*)

Since the KL-divergence is always ≥ 0 we have

$$\log p_\phi(\mathbf{x}) \geq ELBO$$

so we will want to maximize the $ELBO$ i.e. to find

$$\underset{\theta,\phi}{\operatorname{argmax}}\ \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}\left(\log \frac{p_\phi(\mathbf{x},\mathbf{z})}{q_\theta(\mathbf{z}|\mathbf{x})}\right)$$

Rewriting

$$p_\phi(\mathbf{x},\mathbf{z}) = p_\phi(\mathbf{x}|\mathbf{z})p(\mathbf{z})$$

where $p(\mathbf{z})$ is some prior distribution of $\mathbf{z}$ which we typically choose as $\mathcal{N}(0, I)$, we get

$$\begin{aligned}ELBO &= \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}(\log p_\phi(\mathbf{x}|\mathbf{z})) - \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}(q_\theta(\mathbf{z}|\mathbf{x})) + \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}(\log p(\mathbf{z})) \\ &= \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}(\log p_\phi(\mathbf{x}|\mathbf{z})) + (\mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}(\log p(\mathbf{z}))) - \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}(\log q_\theta(\mathbf{x}|\mathbf{z}))) \\ &= \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}(\log p_\phi(\mathbf{x}|\mathbf{z})) + \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}\left(\log \frac{\log p(\mathbf{z})}{\log q_\theta(\mathbf{z}|\mathbf{x})}\right) \\ &= \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}(\log p_\phi(\mathbf{x}|\mathbf{z})) - D_{KL}(q_\theta(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))\end{aligned}$$

The $\log p_\phi(\mathbf{x}|\mathbf{z})$ can be viewed as a reconstruction log likelihood dependent on a specific value of the latent variable $\mathbf{z}$ and so we want to maximize the expectation over all the values of $\mathbf{z}$

$$\mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}(\log p_\phi(\mathbf{x}|\mathbf{z}))$$

The KL-divergence term

$$D_{KL}(q_\theta(\mathbf{z}|\mathbf{x})||p_\phi(\mathbf{z}))$$

which we want to minimize can be viewed as a regularizer which tries to keep the latent distribution $q_\theta(\mathbf{z}|\mathbf{x})$ close to the prior $p(\mathbf{z})$

To do gradient ascent to maximize the $ELBO$ we need to estimate the gradients

$$\begin{aligned}\nabla_{\phi,\theta} ELBO &= \nabla_{\phi,\theta}\mathbb{E}_{q_\theta}\left(\log p_\phi(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z})\right) - \log q_\theta(\mathbf{z}|\mathbf{x})) \\ &= \nabla_{\phi,\theta}\mathbb{E}_{q_\theta}\left(\log p_\phi(\mathbf{x},\mathbf{z}) - \log q_\theta(\mathbf{z}|\mathbf{x})\right)\end{aligned}$$

We are now back in to the situation from section 1.2.

There is no problem computing ∇_ϕ since we can differentiate with respect to ϕ under $\mathbb{E}_{q_\theta}$

$$\nabla_\phi \mathbb{E}_{q_\theta}\left(\log p_\phi(\mathbf{x},\mathbf{z}) - \log q_\theta(\mathbf{z}|\mathbf{x})\right) = \mathbb{E}_{p_\theta}(\nabla_\phi(\log p_\phi(\mathbf{x},\mathbf{z})))$$

In order to differentiate with respect to θ we resort to the 'reparametrization trick'. This means that we cannot arbitrarily choose the distribution output by the Encoder.

In practice this means that we let the Encoder output vectors $\mu_\theta(\mathbf{x})$ and $\sigma_\theta(\mathbf{x})$ and let the latent variable

$$\mathbf{z} \sim q_\theta(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu_\theta(\mathbf{x}), \sigma^2_\theta(\mathbf{x})I)$$

i.e. q_θ is a multi-variate Gaussian with diagonal covariance matrix.

Then we can change variables

$$\mathbf{z} = \mu_\theta(\mathbf{x}) + \sigma_\theta(\mathbf{x})\,\underline{\varepsilon}$$

where $\underline{\varepsilon} \sim \mathcal{N}(0, I)$ so

$$\mathbb{E}_{q_\theta}(\log q_\theta(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{\mathcal{N}(0,I)}(\log(\mu_\theta(\mathbf{x})) + \sigma_\theta(\mathbf{x})\,\underline{\varepsilon}))$$

So we can estimate gradients by Monte Carlo and do gradient ascent to update the parameters θ and ϕ

This is done pretty much automatically by the pyro class `SVI`

In the code example we again use the $MNIST$ data set.

The Encoder Layer takes as input a batch B of images $B \times 1 \times 28 \times 28$ image, flattens to $B \times 1 \times 784$.

This goes through a linear Layer `fc1` and then a `Softplus` activation layer ($Softplus(\mathbf{u}) = \log(1 + \exp(\mathbf{u}))$)

Then the output of the activation layer is branched, one branch going to a linear layer, `fc21` which outputs the mean of the latent variable distribution $q_\theta(\mathbf{z}|\mathbf{x})$. The other branch going to another linear layer `fc22`, after applying exp , this is the diagonal in the covariance matrix of $q_\theta(\mathbf{z}|\mathbf{x})$.

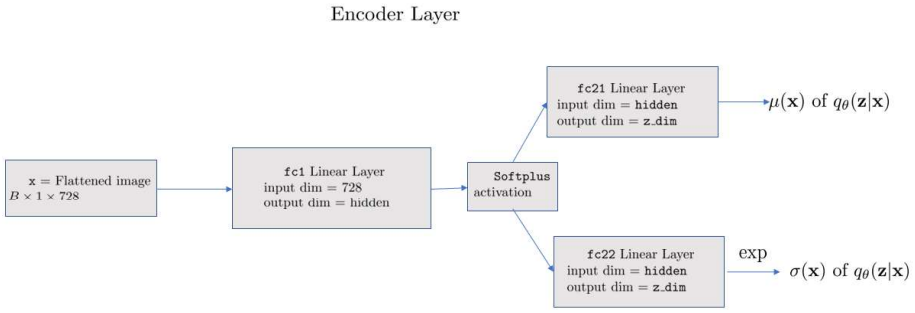

Figure 6:

```python
class Encoder(nn.Module):
    def __init__(self, z_dim, hidden_dim):
        super().__init__()
        # setup the three linear transformations used
        self.fc1 = nn.Linear(784, hidden_dim)
        self.fc21 = nn.Linear(hidden_dim, z_dim)
        self.fc22 = nn.Linear(hidden_dim, z_dim)
        # setup the non-linearities
        self.softplus = nn.Softplus()

    def forward(self, x):
        # define the forward computation on the image x
        # first shape the mini-batch to have pixels in the rightmost dimension
        x = x.reshape(-1, 784)
        # then compute the hidden units
        hidden = self.softplus(self.fc1(x))
        # then return a mean vector and a (positive) square root covariance
        # each of size batch_size x z_dim
        z_loc = self.fc21(hidden)
        z_scale = torch.exp(self.fc22(hidden))
        return z_loc, z_scale
```

Figure 7:

The Decoder Layer takes a sample from the distribution of the latent variable $q_\theta(\mathbf{z}|\mathbf{x})$ and sends it through, first a linear layer `fc1`, an activation layer, `Softplus()`, another linear layer `fc 2` and finally applies a `sigmoid()` function, so the outputs are between 0 and 1

```python
class Decoder(nn.Module):

    def __init__(self,z_dim,hidden_dim):
        super().__init__()
        self.fc1 = nn.Linear(z_dim,hidden_dim)
        self.fc21 = nn.Linear(hidden_dim,28*28)
        self.softplus = nn.Softplus()
        self.sigmoid = nn.Sigmoid()

    def forward(self,z):

        hidden = self.softplus(self.fc1(z))
        loc_img = self.sigmoid(self.fc21(hidden))

        return loc_img
```

Figure 8:

The idea here is that the distribution $p_\phi(\mathbf{x}|\mathbf{z})$ is a product of Bernoulli distributions, one for each of the 784 pixels, and the output from the decoder, `loc_img`, is the vector of means for the these Bernoulli distributions.

The reconstructed image is the 28×28 tensor of samples from the Bernoulli distributions so a tensor of 0's and 1's

The reconstructed image actually looks better if instead of sampling the Bernoullis we just use the means as gray scale values.

We set the dimension of the latent variable to `z_dim` $= 50$.

The `model` registers the decoder object `decoder = Decoder(50,400.cuda()` in the `pyro.param_store` and defines the prior $p_{prior}(\mathbf{z}) = \mathcal{N}(0, I)$

Then we sample a $\mathbf{z}$ from the prior. The decoder outputs the means `loc_im` of the Bernoulli distributions, one for each of the 784 pixels. Su the likelihood is

$$p(\mathbf{x}|\mathbf{z}) = \prod_{x_i \in \mathbf{x}} Bernoulli(x_i; loc\_im_i)$$

The `guide` registers the `encoder = Encoder(50,400)` as a `pyro.module` and then defines the Variational approximation $q(\mathbf{z}|\mathbf{x})$ to the posterior

$$p_{post}(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p_{prior}(\mathbf{z})}{p(\mathbf{x})}$$

as

$$\mathcal{N}(\texttt{z\_loc}, \texttt{z\_scale})$$

where `z_loc,z_scale = encoder(x)`

Then we can run the training loop, minimizing $-ELBO$

The trained model can then be used to generate samples from the posterior i.e. the latent variables $\mathbf{z}$ and then generate images from the likelihood

$$p(\mathbf{x}|\mathbf{z})$$

Basically what we are doing is to generate new parameters $\mathbf{z}$ for the likelihood $p(\mathbf{x}|\mathbf{z})$ and then generate a sample from this distribution

We start with an image $\mathbf{x}$, sends it through the `encoder`, which outputs `z_loc,z_scale = encoder(x)`

Then generate a number of samples $\{\mathbf{z}_k\}$ from $q_\theta = \mathcal{N}(\texttt{z\_loc}, \texttt{z\_scale})$

`samples` = `dist.Normal(z_loc, z_scale).sample(15)`
`loc_img` = `decoder(samples)`

The `decoder` will send each of the $\mathbf{z}_k$'s to a 784 dimensional vector of means of Bernoulli distributions.

Sampling an image by sampling each pixel value (0 or 1) from the appropriate Bernoulli, generates an image.

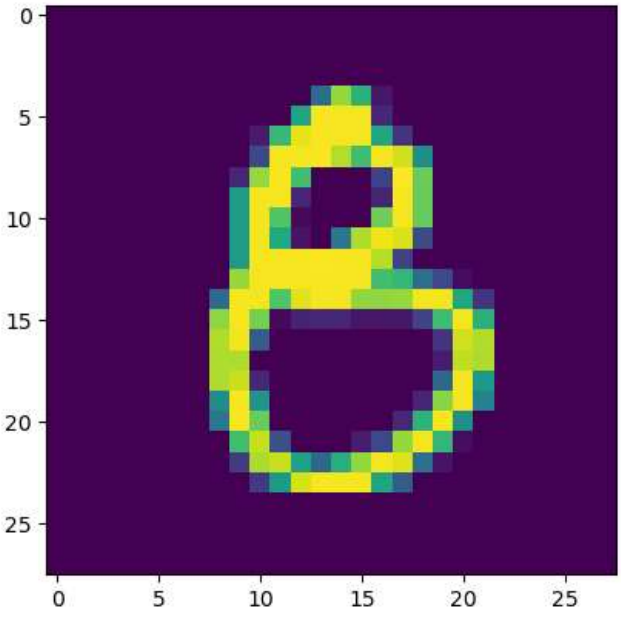

Figure 9: Input Image

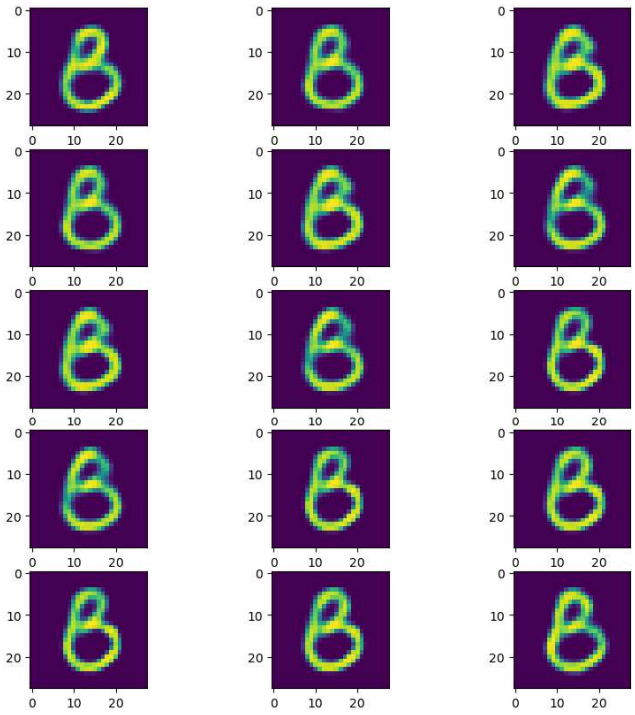

Figure 10: Generated Samples

If we sample from the Bernoullis we get purely black and white images

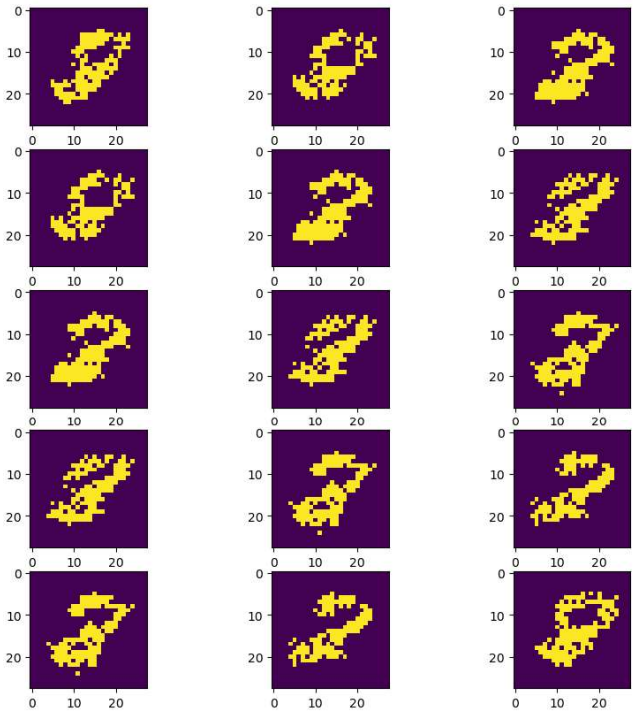

Figure 11: Samples from the Bernoulli Distributions

We can visualize a VAE by fig. 31

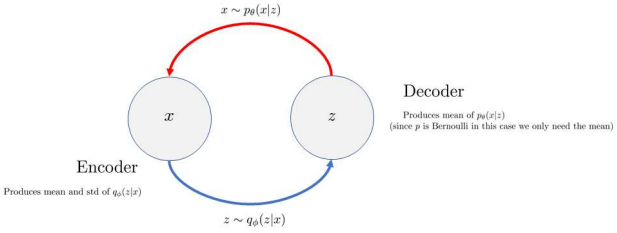

Figure 12: VAE