ML – HW 2
Name: **Peter Garamvoelgyi** Student ID: **2018280070**

1.1. MLE for Multinomial

$$P(\mathbf{x}|\boldsymbol{\mu}) = \frac{n!}{\prod_i x_i!}\prod_i \mu_i{}^{x_i} \qquad i = 1, \dots, d$$

where

$$x_i \in \mathbb{N};\ \sum_i x_i = n;\ \ 0 < \mu_i < 1;\ \sum_i \mu_i = 1$$

Derivation based on the slides (p. 56 of *Logistic Regression*):

$$\begin{aligned} P(\mathbf{x}|\boldsymbol{\mu}) = \frac{n!}{\prod_i x_i!}\prod_i \mu_i{}^{x_i} &= \frac{n!}{\prod_i x_i!} \cdot \exp\left(\sum_i x_i \ln \mu_i\right) = \\ &= \frac{n!}{\prod_i x_i!} \cdot \exp\left(\left(\sum_{i=1}^{d-1} x_i \ln \mu_i\right) + \left(n - \sum_{i=1}^{d-1} x_i\right) \ln\left(1 - \sum_{i=1}^{d-1} \mu_i\right)\right) \\ &= \frac{n!}{\prod_i x_i!} \cdot \exp\left(\left(\sum_{i=1}^{d-1} x_i \ln \frac{\mu_i}{1 - \sum_{j=1}^{d-1} \mu_j}\right) + n \cdot \ln\left(1 - \sum_{i=1}^{d-1} \mu_i\right)\right) \end{aligned}$$

Exponential family representation:

$$\boldsymbol{\eta} = \left[\ln \frac{\mu_i}{\mu_d}; 0\right] \qquad T(\mathbf{x}) = \mathbf{x} \qquad h(\mathbf{x}) = \frac{n!}{\prod_i x_i!}$$
$$A(\boldsymbol{\eta}) = -n \cdot \ln\left(1 - \sum_{i=1}^{d-1} \mu_i\right) = -n \cdot \ln\left(\sum_{i=1}^{d} e^{\boldsymbol{\eta}_i}\right)$$

For *iid* data, log-likelihood is

$$\mathcal{L}(\boldsymbol{\eta}; D) = \sum_n \log h(\mathbf{x}_n) + \left(\boldsymbol{\eta}^T \sum_n T(\mathbf{x}_n)\right) - NA(\boldsymbol{\eta})$$

Take derivatives and set to zero:

$$\nabla_{\boldsymbol{\eta}}\, \mathcal{L}(\boldsymbol{\eta}; D) = \sum_n T(\mathbf{x}_n) - N\, \nabla_{\boldsymbol{\eta}} A(\boldsymbol{\eta}) = 0 \qquad \Longrightarrow \qquad \nabla_{\boldsymbol{\eta}}\, A(\boldsymbol{\eta}) = \frac{1}{N}\sum_n T(\mathbf{x}_n)$$

Thus

$$\widehat{\boldsymbol{\mu}}_{MLE} = \frac{1}{N}\sum_n \mathbf{x}_n$$

1.2. EM for Mixture of Multinomials

We choose a latent topic for each document:

$$P(c_d = k) = \pi_k \quad k = 1, \dots, K \quad \sum_k \pi_k = 1$$

… to each topic we get a categorical distribution over our vocabulary:

$$\boldsymbol{\mu}_k = (\mu_{1k}, \dots, \mu_{Wk})$$

… for which we get the multinomial distribution

$$P(d \mid c_d = k) = \frac{n_d!}{\prod_{w=1}^{W} T_{dw}!} \prod_{w=1}^{W} \mu_{wk}^{T_{dw}} \qquad n_d = \sum_{w=1}^{W} T_{dw}$$

In summary

$$P(d) = \sum_{k=1}^{K} P(d \mid c_d = k) P(c_d = k) = \frac{n_d!}{\prod_{w=1}^{W} T_{dw}!} \sum_{k=1}^{K} \pi_k \prod_{w=1}^{W} \mu_{wk}^{T_{dw}}$$

Log-likelihood:

$$\mathcal{L}(\boldsymbol{\pi}, \boldsymbol{\mu}) = \log \prod_{d=1}^{D} P(d \mid \boldsymbol{\pi}, \boldsymbol{\mu}) = \sum_{d=1}^{D} \log \left(\frac{n_d!}{\prod_{w=1}^{W} T_{dw}!} \sum_{k=1}^{K} \pi_k \prod_{w=1}^{W} \mu_{wk}^{T_{dw}} \right)$$

Adding the constraints (Lagrange):

$$L(\boldsymbol{\pi}, \boldsymbol{\mu}) = \mathcal{L}(\boldsymbol{\pi}, \boldsymbol{\mu}) - \lambda_1 \left(\sum_{k=1}^{K} \pi_k - 1 \right) - \lambda_2 \left(\sum_{w=1}^{W} \mu_{wk} - 1 \right)$$

Let us take the derivatives:

$$\frac{\partial L}{\partial \pi_k} = \sum_{d=1}^{D} \frac{\pi_k \prod_{w=1}^{W} \mu_{wk}^{T_{dw}}}{\sum_{j=1}^{K} \pi_j \prod_{w=1}^{W} \mu_{wj}^{T_{dw}} x} \cdot \frac{1}{\pi_k} - \lambda_1$$

$$\frac{\partial L}{\partial \mu_{wk}} = \sum_{d=1}^{D} \frac{\pi_k \prod_{w=1}^{W} \mu_{wk}^{T_{dw}}}{\sum_{j=1}^{K} \pi_j \prod_{w=1}^{W} \mu_{wj}^{T_{dw}}} \cdot \frac{T_{dw}}{\mu_{wk}} - \lambda_2$$

Let us introduce

$$\gamma(z_{dk}) = \frac{\pi_k \prod_{w=1}^{W} \mu_{wk}^{T_{dw}}}{\sum_{j=1}^{K} \pi_j \prod_{w=1}^{W} \mu_{wj}^{T_{dw}}} \quad \rightarrow \quad \sum_{k=1}^{K} \gamma(z_{dk}) = 1$$

The derivatives become:

$$\frac{\partial L}{\partial \pi_k} = \frac{1}{\pi_k}\sum_{d=1}^{D}\gamma(z_{dk}) - \lambda_1 = 0 \quad \rightarrow \quad \pi_k = \frac{1}{\lambda_1}\sum_{d=1}^{D}\gamma(z_{dk})$$

$$\frac{\partial L}{\partial \mu_{wk}} = \frac{1}{\mu_{wk}}\sum_{d=1}^{D}\gamma(z_{dk})T_{dw} - \lambda_2 = 0 \quad \rightarrow \quad \mu_{wk} = \frac{1}{\lambda_2}\sum_{d=1}^{D}\gamma(z_{dk})T_{dw}$$

Plugging these back into the constraints we get:

$$\sum_{k=1}^{K}\pi_k = 1 \quad \rightarrow \quad \lambda_1 = \sum_{k=1}^{K}\sum_{d=1}^{D}\gamma(z_{dk}) \quad \rightarrow \quad \pi_k = \frac{\sum_{d=1}^{D}\gamma(z_{dk})}{\sum_{j=1}^{K}\sum_{d=1}^{D}\gamma(z_{dj})} = \frac{\sum_{d=1}^{D}\gamma(z_{dk})}{D}$$

$$\sum_{w=1}^{W}\mu_{wk} = 1 \quad \rightarrow \quad \lambda_2 = \sum_{w=1}^{W}\sum_{d=1}^{D}\gamma(z_{dk})T_{dw} \quad \rightarrow \quad \mu_{wk} = \frac{\sum_{d=1}^{D}\gamma(z_{dk})T_{dw}}{\sum_{w=1}^{W}\sum_{d=1}^{D}\gamma(z_{dk})T_{dw}}$$

E-step: estimate the responsibilities (initialization is crucial)

$$\gamma(z_{dk}) = \frac{\pi_k \prod_{w=1}^{W}\mu_{wk}^{T_{dw}}}{\sum_{j=1}^{K}\pi_j \prod_{w=1}^{W}\mu_{wj}^{T_{dw}}}$$

M-step: re-estimate the parameters

$$\pi_k = \frac{\sum_{d=1}^{D}\gamma(z_{dk})}{D}$$

$$\mu_{wk} = \frac{\sum_{d=1}^{D}\gamma(z_{dk})T_{dw}}{\sum_{w=1}^{W}\sum_{d=1}^{D}\gamma(z_{dk})T_{dw}}$$

2.1. PCA – Minimum Error Formulation

Given a set of complete orthonormal basis

$$\boldsymbol{\mu}_i{}^T \boldsymbol{\mu}_j = \delta_{ij} \qquad i,j = 1,\dots,p$$

… each data point can be represented as

$$\boldsymbol{x}_n = \sum_{i=1}^{p} \alpha_{ni}\, \boldsymbol{\mu}_i \qquad \alpha_{ni} = \boldsymbol{x}_n{}^T \boldsymbol{\mu}_i$$

A low-dimensional approximation of these data points can be represented as

$$\tilde{\boldsymbol{x}}_n = \sum_{i=1}^{d} z_{ni}\boldsymbol{\mu}_i + \sum_{i=d+1}^{p} b_i\, \boldsymbol{\mu}_i$$

… where z_{ni} are unique to every data point but b_i are common.

The best approximation is to minimize the error

$$J = \frac{1}{N}\sum_{n=1}^{N} \|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2 = \frac{1}{N}\sum_{n=1}^{N} \left\| \sum_{i=1}^{p} (\boldsymbol{x}_n{}^T \boldsymbol{\mu}_i)\, \boldsymbol{\mu}_i - \left(\sum_{i=1}^{d} z_{ni}\boldsymbol{\mu}_i + \sum_{i=d+1}^{p} b_i\, \boldsymbol{\mu}_i \right) \right\|^2 =$$
$$= \frac{1}{N}\sum_{n=1}^{N} \left\| \sum_{i=1}^{d} (\boldsymbol{x}_n{}^T \boldsymbol{\mu}_i - z_{ni})\, \boldsymbol{\mu}_i + \sum_{i=d+1}^{p} (\boldsymbol{x}_n{}^T \boldsymbol{\mu}_i - b_i)\, \boldsymbol{\mu}_i \right\|^2$$

… by choosing the appropriate $\boldsymbol{\mu}_i$, z_{ni}, and b_i values.

Let us first compute the derivatives w.r.t. these terms:

$$\frac{\partial J}{\partial z_{mj}} = \frac{\partial J}{\partial z_{mj}} \frac{1}{N}\sum_{n=1}^{N} \|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2 \qquad j = 1,\dots,d$$

First, the derivatives of all terms of the outer sum except for $n = m$ are 0.

$$\frac{\partial J}{\partial z_{mj}} \frac{1}{N}\sum_{n=1}^{N} \|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2 = \frac{1}{N}\frac{\partial J}{\partial z_{mj}} \left\| \sum_{i=1}^{d} (\boldsymbol{x}_n{}^T \boldsymbol{\mu}_i - z_{ni})\, \boldsymbol{\mu}_i + \sum_{i=d+1}^{p} (\boldsymbol{x}_n{}^T \boldsymbol{\mu}_i - b_i)\, \boldsymbol{\mu}_i \right\|^2$$

Second, the derivatives of all terms of the inner sums except for $i = j$ are 0.

$$\frac{1}{N}\frac{\partial J}{\partial z_{mj}} \left\| \sum_{i=1}^{d} (\boldsymbol{x}_m{}^T \boldsymbol{\mu}_i - z_{mi})\, \boldsymbol{\mu}_i + \sum_{i=d+1}^{p} (\boldsymbol{x}_m{}^T \boldsymbol{\mu}_i - b_i)\, \boldsymbol{\mu}_i \right\|^2 = \frac{2}{N} \cdot \left(\boldsymbol{x}_m{}^T \boldsymbol{\mu}_j - z_{mj}\right)\boldsymbol{\mu}_j \cdot \left(-\boldsymbol{\mu}_j\right)$$

$$\boldsymbol{y} = \sum_{i=1}^{d} (\boldsymbol{x}_m{}^T \boldsymbol{\mu}_i - z_{mi})\, \boldsymbol{\mu}_i + \sum_{i=d+1}^{p} (\boldsymbol{x}_m{}^T \boldsymbol{\mu}_i - b_i)\, \boldsymbol{\mu}_i$$

$$\boldsymbol{y_1} = (\boldsymbol{x}_m{}^T \boldsymbol{\mu}_i - z_{mi})$$

$$\boldsymbol{y_{d+1}} = (\boldsymbol{x}_m{}^T \boldsymbol{\mu}_i - z_{mi})$$

Making use of orthonormality and setting the derivative to 0, we get:

$$z_{mj} = \boldsymbol{x}_m{}^T \boldsymbol{\mu}_j \quad m = 1, \dots, N; \;\; j = 1, \dots, d$$

Similarly, for b_i:

$$\frac{\partial J}{\partial b_j} = \frac{\partial J}{\partial b_j} \frac{1}{N} \sum_{n=1}^{N} \|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2 \qquad j = d+1, \dots, p$$

Here, we can similarly eliminate the inner sum (but not the outer one):

$$\frac{\partial J}{\partial b_j} \frac{1}{N} \sum_{n=1}^{N} \|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2 = \frac{\partial J}{\partial b_j} \frac{1}{N} \sum_{n=1}^{N} \left\| \sum_{i=1}^{d} (\boldsymbol{x}_n{}^T \boldsymbol{\mu}_i - z_{ni})\, \boldsymbol{\mu}_i + \sum_{i=d+1}^{p} (\boldsymbol{x}_n{}^T \boldsymbol{\mu}_i - b_i)\, \boldsymbol{\mu}_i \right\|^2 =$$

$$= \frac{2}{N} \sum_{n=1}^{N} \left(\boldsymbol{x}_n{}^T \boldsymbol{\mu}_j - b_j\right) \boldsymbol{\mu}_j \cdot \left(-\boldsymbol{\mu}_j\right) = \frac{2}{N} \sum_{n=1}^{N} \left(b_j - \boldsymbol{x}_n{}^T \boldsymbol{\mu}_j\right) =$$

$$= 2b_j - 2\boldsymbol{\mu}_j \cdot \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}_n{}^T = 2b_j - 2\boldsymbol{\mu}_j \cdot \overline{\boldsymbol{x}}^T$$

Setting the derivative to 0, we get:

$$b_j = \overline{\boldsymbol{x}}^T \boldsymbol{\mu}_j \quad j = d+1, \dots, p$$

Substituting these, we can get the displacement vectors:

$$\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n = \sum_{i=1}^{d} (\boldsymbol{x}_n{}^T \boldsymbol{\mu}_i - z_{ni})\, \boldsymbol{\mu}_i + \sum_{i=d+1}^{p} (\boldsymbol{x}_n{}^T \boldsymbol{\mu}_i - b_i)\, \boldsymbol{\mu}_i = \sum_{i=d+1}^{p} (\boldsymbol{x}_n{}^T \boldsymbol{\mu}_i - \overline{\boldsymbol{x}}^T \boldsymbol{\mu}_i)\, \boldsymbol{\mu}_i$$

The error formula becomes

$$J = \frac{1}{N} \sum_{n=1}^{N} \|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2 = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=d+1}^{p} (\boldsymbol{x}_n{}^T \boldsymbol{\mu}_i - \overline{\boldsymbol{x}}^T \boldsymbol{\mu}_i)^2 = \sum_{i=d+1}^{p} \boldsymbol{\mu}_i{}^T S \boldsymbol{\mu}_i$$

… where

$$S = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \overline{\boldsymbol{x}})(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^T$$

Our final task is to find the base

$$\min_{\boldsymbol{\mu}_i} J$$

The general solution is

$$S\boldsymbol{\mu}_i = \lambda_i \boldsymbol{\mu}_i$$

3.1. Reinforcement Learning – Value Iteration

$$Q_k(s,a) \leftarrow R(s,a) + \gamma \cdot \sum_{s'} P(s' \mid s,a) \cdot \max_{a'} Q(s',a')$$

```
Q(Non,      Send) =  -1 + 0.9·(0.1 ·(-1) + 0.85·35 + 0.05·50) =  27.935
Q(Non,   No-send) =   0 + 0.9·(0.95·(-1) + 0.05·35 + 0.0 ·50) =   0.72
Q(Light,    Send) =  15 + 0.9·(0.1 ·(-1) + 0.2 ·35 + 0.7 ·50) =  52.71
Q(Light, No-Send) =  20 + 0.9·(0.2 ·(-1) + 0.75·35 + 0.05·50) =  45.695
Q(Heavy,    Send) =  80 + 0.9·(0.05·(-1) + 0.15·35 + 0.8 ·50) = 120.68
Q(Heavy, No-Send) = 100 + 0.9·(0.1 ·(-1) + 0.2 ·35 + 0.7 ·50) = 137.71
```

4. Deep Generative Models – Class-conditioned VAE

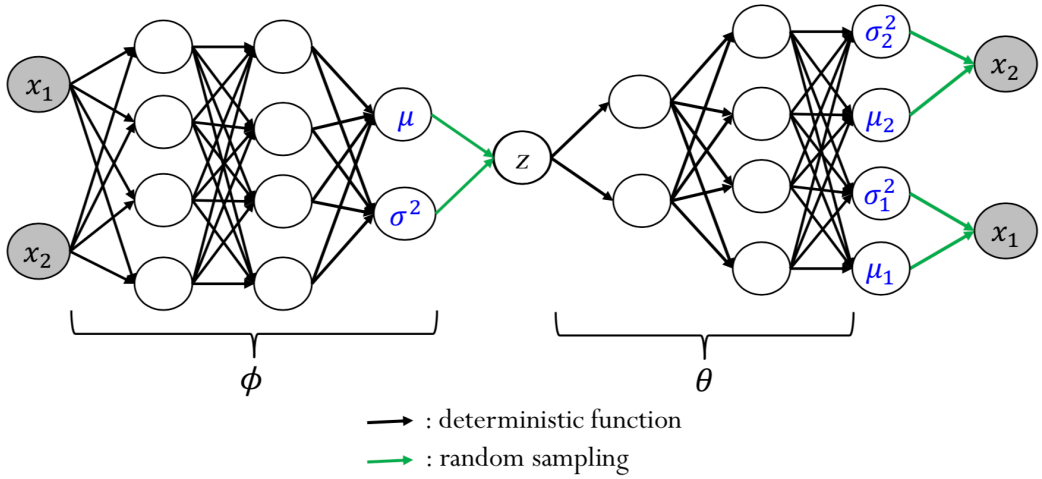
The main idea behind Variational Auto-Encoders (VAE) is to introduce a latent variable that helps us approximate the true distribution that generates our input data.

We model the distribution of the latent variable using a normal distribution and minimize the difference between it and the true distribution, captured by the KL-divergence.

Variational lower bound:

$$\log p_\theta(\mathbf{x}) = KL\left(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x})\right) + E_{q_\phi(\mathbf{z}|\mathbf{x})}\left[-\log q_\phi(\mathbf{z}|\mathbf{x}) + \log p_\theta(\mathbf{x},\mathbf{z})\right]$$
$$\geq -KL\left(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z})\right) + E_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right]$$

In its essence $q_\phi(\mathbf{z}|\mathbf{x})$ represents the encoding network, while $p_\theta(\mathbf{x}|\mathbf{z})$ represents the decoding network. This structure is also often referred to as p-q network.



Empirical objective:

$$\tilde{\mathcal{L}}_{VAE}(\mathbf{x},\theta,\phi) = -KL\left(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z})\right) + \frac{1}{L}\sum_{l=1}^{L} \log p_\theta\left(\mathbf{x}|\mathbf{z}^{(l)}\right)$$

where

$$\mathbf{z}^{(l)} \sim g_\phi\left(\mathbf{x}, \epsilon^{(l)}\right), \quad \epsilon^{(l)} \sim N(\mathbf{0}, \boldsymbol{I})$$

(Making use of the *reparameterization trick* to make our network deterministic and trainable.)

In Class-conditioned VAE (CVAE), we not only want to look at the data but also the associated labels. Thus, we need to adjust our objective function as follows

$$
\begin{aligned}
\log p_\theta(\mathbf{x}|y) &= E_{q_\phi(\mathbf{z}|\mathbf{x},y)}\big[\log p_\theta(\mathbf{x},\mathbf{z}\mid y) - \log q_\phi(\mathbf{z}\mid\mathbf{x},y)\big] + KL\left(q_\phi(\mathbf{z}\mid\mathbf{x},y)\,||\,p_\theta(\mathbf{z}\mid\mathbf{x},y)\right)\\
&\geq E_{q_\phi(\mathbf{z}|\mathbf{x},y)}\big[\log p_\theta(\mathbf{x},\mathbf{z}\mid y) - \log q_\phi(\mathbf{z}\mid\mathbf{x},y)\big]\\
&= E_{q_\phi(\mathbf{z}|\mathbf{x},y)}\big[\log p_\theta(\mathbf{x}\mid\mathbf{z},y) + \log p_\theta(\mathbf{z}\mid y) - \log q_\phi(\mathbf{z}\mid\mathbf{x},y)\big]\\
&= E_{q_\phi(\mathbf{z}|\mathbf{x},y)}[\log p_\theta(\mathbf{x}\mid\mathbf{z},y)] - KL\big[q_\phi(\mathbf{z}\mid\mathbf{x},y)\,||\,p_\theta(\mathbf{z}\mid y)\big]
\end{aligned}
$$

Empirical lower bound:

$$
\tilde{\mathcal{L}}_{VAE}(\mathbf{x},\mathrm{y},\theta,\phi) = -KL\big[q_\phi(\mathbf{z}\mid\mathbf{x},y)\,||\,p_\theta(\mathbf{z}\mid y)\big] + \frac{1}{L}\sum_{l=1}^{L}\log p_\theta\big(\mathbf{x}|\mathbf{z}^{(l)},y\big)
$$

where

$$
\mathbf{z}^{(l)}\sim g_\phi\big(\mathbf{x},\mathrm{y},\epsilon^{(l)}\big),\quad \epsilon^{(l)}\sim N(\mathbf{0},\boldsymbol{I})
$$

The implementation is based on the example VAE code provided. The main idea is that we provide the labels (y) to the decoding network. In code this simply means concatenating our samples from z with y. This way the P-network learns to reconstruct specific numbers based on the latent distribution and the labels provided. (It could be considered to provide the labels to the Q-network as well but this does not seem necessary.)

The Q-network part looks like this:

```
@zs.reuse_variables(scope="q_net")
def build_q_net(x, z_dim):
    bn = zs.BayesianNet()
    h = tf.layers.dense(x, 500, activation=tf.nn.relu)
    h = tf.layers.dense(h, 500, activation=tf.nn.relu)
    z_mean = tf.layers.dense(h, z_dim)
    z_logstd = tf.layers.dense(h, z_dim)
    bn.normal("z", z_mean, logstd=z_logstd, group_ndims=1)
    return bn
```

Explanation: We create a neural network with two dense hidden layers with ReLU activation and feed our images into it. The output of this network is the mean and standard deviation used for modelling the latent distribution.

The P-network part looks like this:

```
@zs.meta_bayesian_net(scope="gen", reuse_variables=True)
def build_gen(y, x_dim, z_dim, n):
    bn = zs.BayesianNet()
    z = bn.normal("z", tf.zeros([n, z_dim]), std=1., group_ndims=1)
    input = tf.concat([z, y], axis=1)
    h = tf.layers.dense(input, 500, activation=tf.nn.relu)
    h = tf.layers.dense(h, 500, activation=tf.nn.relu)
    x_logits = tf.layers.dense(h, x_dim)
    x_mean = bn.deterministic("x_mean", tf.sigmoid(x_logits))
    x = bn.bernoulli("x", x_logits, group_ndims=1, dtype=tf.float32)
    return bn
```

Explanation: We take a sample from the latent variable and concatenate it with the labels, using the result as the input for our decoding neural network.

The input labels need one-hot encoding:

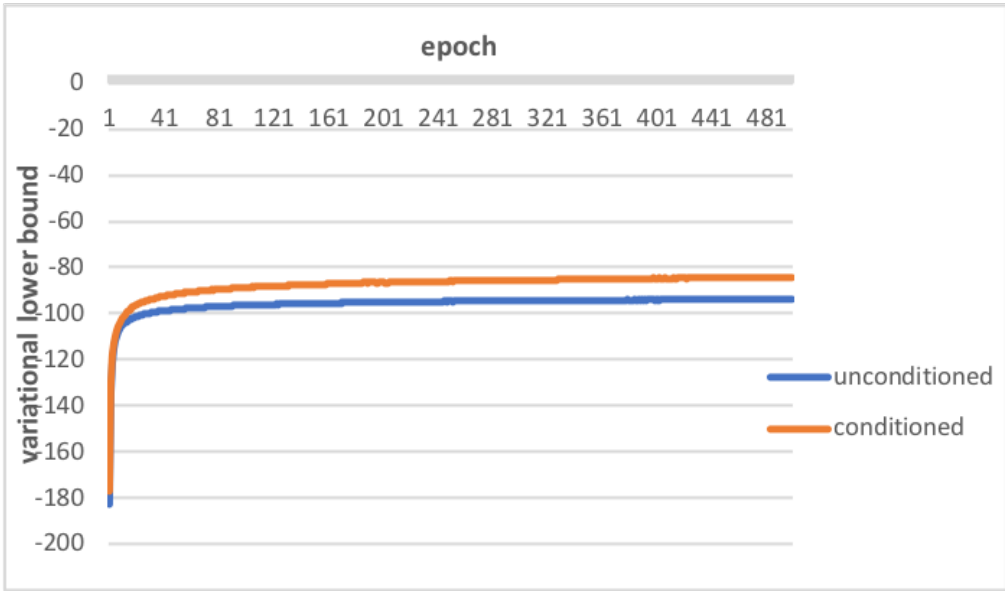
```
y_dim = 10
t_train_one_hot = np.zeros((t_train.size, y_dim))
t_train_one_hot[np.arange(t_train.size), t_train] = 1
```

And we also need to adjust the image generation part so that we get images for specific labels:

```
for i in range(0, y_dim):
    current_y = np.zeros(y_dim)
    current_y[i] = 1.
    current_y = np.tile(current_y, (100, 1))
    images = sess.run(x_gen, feed_dict={n: 100, y: current_y})
    name = os.path.join("results", "vae.epoch.{}.number.{}.png".format(epoch, i))
    save_image_collections(images, name)
```

The rest of the code has only minor changes (defining a placeholder for the labels and passing it appropriately).

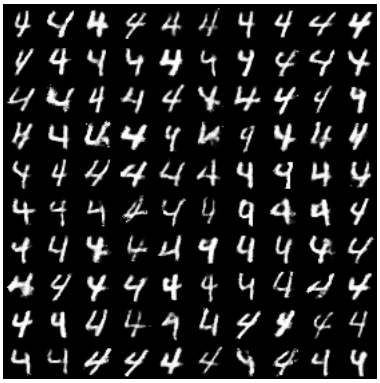
The following chart shows how the variational lower bound changes over the epochs:



It is clear that the conditional version delivers better results.

Below are some examples for number 4 in epoch 10, 50, and 500:





(Some more examples are submitted along with this document.)