Machine Learning Workshop 2

Variational Autoencoder

Jonathan Guymont

National Bank of Canada, 2019

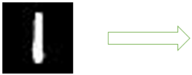
Outline

1. Autoencoders
   - An example to keep in mind
   - Autoencoders
2. Variational autoencoders
   - Generative Models
   - Latent variable models
   - Variational Lower Bound
3. Experiment
   - Unsupervised Spam Detection
   - Preprocessing
   - Decoders
   - Spam Detector

Outline

1. Autoencoders
   - An example to keep in mind
   - Autoencoders
2. Variational autoencoders
   - Generative Models
   - Latent variable models
   - Variational Lower Bound
3. Experiment
   - Unsupervised Spam Detection
   - Preprocessing
   - Decoders
   - Spam Detector

An example to keep in mind



$$\begin{bmatrix} 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \mathbf{255} & \mathbf{211} & 111 & \cdots & 0 \\ 0 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 1 & \mathbf{246} & \mathbf{244} & 4 & \cdots & 0 \\ 0 & \cdots & 12 & \mathbf{233} & \mathbf{213} & 34 & \cdots & 0 \\ 0 & \cdots & 22 & \mathbf{252} & \mathbf{234} & 2 & \cdots & 0 \\ 0 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a0 & \cdots & 4 & \mathbf{233} & \mathbf{222} & 33 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Figure: Conversion of a greyscale image to an matrix

Note: In the workshop, we will mostly work with linear layers, so we also need to flatten the image.

```
from PIL import Image
def image_to_array(image_path):
    with Image.open(image_path) as img:
        image = img.convert()
        array_image = np.asarray(image, np.float)
    return array_image
```

Outline

1. Autoencoders
   - An example to keep in mind
   - Autoencoders
2. Variational autoencoders
   - Generative Models
   - Latent variable models
   - Variational Lower Bound
3. Experiment
   - Unsupervised Spam Detection
   - Preprocessing
   - Decoders
   - Spam Detector

Autoencoders

Autoencoers are neural networks that are trained to learn how to map their input to their input. Internally, it has a hidden layer $\boldsymbol{h}$ that contains a lossy summary of the relevant features for the task.

Autoencoders

An autoencoder can be seen has a two parts network:

- Encoder function: $\boldsymbol{z} = f_\phi(\boldsymbol{x})$

Autoencoders

An autoencoder can be seen has a two parts network:

- Encoder function: $\boldsymbol{z} = f_\phi(\boldsymbol{x})$
- Decoder function: $\tilde{\boldsymbol{x}} = g_\theta(\boldsymbol{z})$

Autoencoders

An autoencoder can be seen has a two parts network:

- Encoder function: $\boldsymbol{z} = f_\phi(\boldsymbol{x})$
- Decoder function: $\tilde{\boldsymbol{x}} = g_\theta(\boldsymbol{z})$
- ϕ and θ are sets of learned parameters

Autoencoders

The simplest autoencoder is a one layer MLP:

$$
\begin{aligned}
\mathbf{z} =& \mathrm{relu}\left(\mathbf{W}_{xz}\mathbf{x} + \mathbf{b}_{xz}\right) \quad \text{[encoder]} \\
\tilde{\mathbf{x}} =& \mathrm{sigmoid}\left(\mathbf{W}_{zx}\mathbf{z} + \mathbf{b}_{zx}\right) \quad \text{[decoder]}
\end{aligned} \tag{1}
$$

Pytorch simple autoencoder

```
class Autoencoder:
        def __init__(self, **kargs):
            """constructor"""
            pass

        def encoder(self, x):
            pass

        def decoder(self, z)
            pass

        def forward(self, x):
            pass
```

Parameter initialization

$$\begin{aligned} \mathbf{z} =& \text{relu}\left(\mathbf{W}_{xz}\mathbf{x} + \mathbf{b}_{xz}\right) \\ \tilde{\mathbf{x}} =& \text{sigmoid}\left(\mathbf{W}_{zx}\mathbf{z} + \mathbf{b}_{zx}\right) \end{aligned} \tag{2}$$

```
class Autoencoder:
    def __init__(self, x_dim, z_dim):
        # encoder parameters \phi
        self.Wxz = xavier_init(size=[x_dim, z_dim])
        self.bxz = Variable(torch.zeros(z_dim), requires_grad=True)
        # decoder parameters \theta
        self.Wzx = xavier_init(size=[z_dim, x_dim])
        self.bzx = Variable(torch.zeros(x_dim), requires_grad=True)
```

Encoder $f_\phi(x)$

$$\mathbf{z} = \mathrm{relu}\left(\mathbf{W}_{xz}\mathbf{x} + \mathbf{b}_{xz}\right) \tag{3}$$

$$\phi = \{\mathbf{W}_{xz}, \mathbf{b}_{xz}\} \tag{4}$$

```
class Autoencoder:
        ...
    def encoder(self, x):
        z = F.relu(self.Wxz @ z + self.bxz)
        return z
```

Decoder $g_\theta(x)$

$$\mathbf{z} = \sigma\left(\mathbf{W}_{zx}\mathbf{z} + \mathbf{b}_{zx}\right) \tag{5}$$

$$\boldsymbol{\theta} = \{\mathbf{W}_{zx}, \mathbf{b}_{zx}\} \tag{6}$$

```
class Autoencoder:
        ...
    def decoder(self, z):
        x_recon = F.sigmoid(self.Wzx @ z + self.bzx)
        return x_recon
```

Forward propagation

$$\begin{aligned}\mathbf{z} &= \text{relu}\left(\mathbf{W}_{xz}\mathbf{x} + \mathbf{b}_{xz}\right) \\ \tilde{\mathbf{x}} &= \text{sigmoid}\left(\mathbf{W}_{zx}\mathbf{z} + \mathbf{b}_{zx}\right)\end{aligned} \tag{7}$$

```python
class Autoencoder:
        ...
    def forward(self, x):
        z = self.encoder(x)
        x_recon = self.decoder(z)
        return x_recon
```

Pytorch simple autoencoder

```
class Autoencoder:
    def __init__(self, x_dim, z_dim):
        # encoder parameters
        Wxz = xavier_init(size=[x_dim, z_dim])
        bxz = Variable(torch.zeros(z_dim), requires_grad=True)
        # decoder parameters
        Wzx = xavier_init(size=[h_dim, x_dim])
        bzx = Variable(torch.zeros(X_dim), requires_grad=True)

    def encoder(self, x):
        z = F.relu(self.Wxh @ x + self.bxh)
        return z

    def decoder(self, z):
        x_recon = F.sigmoid(self.Wzx @ z + self.bzx)
        return x_recon

    def forward(self, x):
        z = self.encoder(x)
        x_recon = self.decoder(z)
        return x_recon
```

Autoencoder - Loss Function

If you treat the problem like a *regression*[1], use the mean square error between the inputs and the reconstructions

$$\mathcal{L} = \sum_{i=1}^{d} (x_i - \tilde{x}_i)^2 \tag{8}$$

[1] If you do regression, you don't have to apply sigmoid in the decoder.

Training Autoencoders

Algorithm 1 Pseudocode for Stochastic Gradient Training

Require: Learning rate η
Require: Initial parameter $\boldsymbol{\omega}_0$
Require: Number of epochs T
  for $i = 1$ to T **do**
    $X = X^{train}$.copy() and $Y = Y^{train}$.copy()
    while X is not empty **do**
      Sample $\{x^{(1)}, ..., x^{(m)}\}$ from X and $\{y^{(1)}, ..., y^{(m)}\}$ from Y
      Remove samples from X and Y
      Compute gradient $\boldsymbol{g}_t = \frac{1}{m}\nabla_{\boldsymbol{\omega}} \sum_i \mathcal{L}(\tilde{x}^{(i)}, x^{(i)})$
      Apply update: $\boldsymbol{\omega}_t = \boldsymbol{\omega}_{t-1} - \eta \cdot \boldsymbol{g}_t$
    end while
  end for

Pytorch Stochastic Gradient

```
def train(self, trainloader, num_epochs, learning_rate):

    for epoch in range(num_epochs):
        for inputs, targets in trainloader:
            batch_size = inputs.size(0)
            x_tilde = self.forward(x)

            loss = F.mse_loss(x, x_tilde)

            # Use autograd to compute the derivative of the loss w.r.t
            # all Tensors with requires_grad=True. After calling `loss.backward()`,
            # conv_weight.grad, dense_weight.grad, and dense_bias.grad
            # will be Tensors equal to the gradient of the loss with respect
            # to the filters of the cnn layer, the weight of the fully connected layer, and
            # the bias of the fully connected layer respectively.
            loss.backward()

            # Apply gradient descent to all the leaned parameters
            # The derivative of the loss is giving us the direction
            # where the funtion increase. Thus we go in the
            # opposite direction. Using torch.no_grad() tells pytorch
            # to not include thes operation in the computational graph.
            # Instead, gradient descent is goning to be applied `inplace`.
            with torch.no_grad():
                self.W_xz -= learning_rate * self.W_xz.grad
                self.b_xz -= learning_rate * self.b_xz.grad
                self.W_zx -= learning_rate * self.W_zx.grad
                self.b_zx -= learning_rate * self.b_zx.grad
```

Autoencoders

To summarize

- A neural network encodes x in a hidden state z of smaller dimension.
- Another neural network decodes z to reconstruct x.
- A sound loss function could be the mean square error between the inputs and their reconstructions.
- Both network can be train at the same time with stochastic subgradient method.

Outline

1. Autoencoders
   - An example to keep in mind
   - Autoencoders
2. Variational autoencoders
   - Generative Models
   - Latent variable models
   - Variational Lower Bound
3. Experiment
   - Unsupervised Spam Detection
   - Preprocessing
   - Decoders
   - Spam Detector

Generative Models

Represent the probability distribution of either $P(X, Y)$ or $P(X)$. In the case of *density estimation*, we are looking for a representation of

$$x \sim P_\theta(X)$$

For example, $x \sim \mathcal{N}(x; \boldsymbol{\mu}_{\text{mle}}, \boldsymbol{\sigma}_{\text{mle}})$

Problem: Most parametric distributions make strong (and often wrong) assumptions about the distribution.

Outline

1. Autoencoders
   - An example to keep in mind
   - Autoencoders
2. Variational autoencoders
   - Generative Models
   - Latent variable models
   - Variational Lower Bound
3. Experiment
   - Unsupervised Spam Detection
   - Preprocessing
   - Decoders
   - Spam Detector

Latent variable models

We can model the distribution of x as a function of a latent variable z

$$p(x) = \int p_\theta(x|z)p(z)dz$$

where the distribution of z is chosen. A typical choice for z is

$$z \sim \mathcal{N}(z; \mathbf{0}, \boldsymbol{I})$$

Then we can train a model to learn a good representation of $p_\theta(x|z)$ with stochastic gradient descent.

Latent variable models

$$p(x) = \int p_\theta(x|z)p(z)dz$$

Once we have a good representation of $p_\theta(x|z)$, we can sample from $p(x)$ by first sampling

$$z' \sim p(z)$$

and then sampling

$$x' \sim p(x|z')$$

Latent variable models

Problem 1: To learn $p_\theta(x|z)$ using stochastic gradient descent, we need to know a good mapping

$$f : \mathcal{Z} \times \Theta \mapsto \mathcal{X}$$

In other words when we sample $x \sim p(x|z')$ we need to know which x is likely to be generated by this particular z' in order to train the model.

Latent variable models

Solution: the prior of the latent space can be written has

$$p(z) = \int p(z|x)p(x)dx$$

During training, We can sample z by sampling

$$x' \sim p(x)$$

and then

$$z \sim p(z|x')$$

The training set comes from $p(x)$ so we can sample from it. This will reduce the space of the latent variable a lot and allow the model to learn efficiently.

Latent variable models

Problem 2: $p(z|x)$ is intractable.

Solution: use an approximation $q_\phi(z|x)$

To summarize

- Sample $x \sim D_n$
- Sample $z \sim q_\phi(z|x)$
- Sample $\tilde{x} \sim p_\theta(x|z)$

The sets of parameters to learn are ϕ and θ and they should be learn such that the marginal likelihood $p(x)$ is maximized.

Latent variable models

Before looking at how we can train this model efficiently, let's take a closer look at how it works.

Probabilistic Encoder $q_\phi(z|x)$

Example: Gaussian MLP as encoder

- $\boldsymbol{\mu}_z,\ \log \boldsymbol{\sigma}_z^2 = f(\mathbf{x}; \boldsymbol{\phi})$
  - $\mathbf{h} = \mathrm{relu}\left(\mathbf{W}_{xh}\mathbf{x} + \mathbf{b}_{xh}\right)$
  - $\boldsymbol{\mu} = \mathbf{W}_{hz}^{(1)}\mathbf{h} + \mathbf{b}_{hz}^{(1)}$
  - $\log \boldsymbol{\sigma}^2 = \mathbf{W}_{hz}^{(2)}\mathbf{h} + \mathbf{b}_{hz}^{(2)}$
- $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_z, \boldsymbol{\sigma}_z^2)$
- $\phi = \{\mathbf{W}_{hz}^{(1)}, \mathbf{b}_{hz}^{(1)}, \mathbf{W}_{hz}^{(2)}, \mathbf{b}_{hz}^{(2)}, \mathbf{W}_{xh}, \mathbf{b}_{xh}\}$

```
class VAE:
    ...
    def encoder(self, x):
        # Encoder network. Return the parameter of q(z|x)
        h = relu(self.Wxh @ x + self.bxh)
        mu = self.Whz_mu @ h + self.bhz_mu
        logvar = self.Whz_var @ h + self.bhz_var
        reture mu, logvar
```

Sampling $z \sim q_\phi(z|x)$

Example: Sampling $\mathbf{z}$ from $q_\phi(\mathbf{z}|\mathbf{x})$

- $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z)$
  - $\boldsymbol{\mu}_z,\ \log \boldsymbol{\sigma}_z^2 = f(\mathbf{x}; \phi)$
  - $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  - $\mathbf{z} = \boldsymbol{\mu}_z + \boldsymbol{\sigma}_z \odot \boldsymbol{\epsilon}$

```
class VAE:
    ...
    def _sample_z(self, mu, logvar):
        epsilon = Variable(torch.randn(mu.size()))
        sigma = torch.exp(logvar / 2)
        return mu + sigma * epsilon
```

Probabilistic Decoder $p_\theta(x|z)$

Example: Gaussian MLP as decoder

- $\boldsymbol{\mu}_x,\ \log \boldsymbol{\sigma}_x^2 = g(\mathbf{x}; \boldsymbol{\theta})$
  - $\mathbf{h} = \mathrm{relu}\left(\mathbf{W}_{zh}\mathbf{x} + \mathbf{b}_{zh}\right)$
  - $\boldsymbol{\mu}_x = \mathbf{W}_{hz}^{(1)}\mathbf{h} + \mathbf{b}_{hz}^{(1)}$
  - $\log \boldsymbol{\sigma}_x^2 = \mathbf{W}_{hz}^{(2)}\mathbf{h} + \mathbf{b}_{hz}^{(2)}$
- $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_x, \log \boldsymbol{\sigma}_x^2)$
- $\boldsymbol{\theta} = \{\mathbf{W}_{zh}, \mathbf{b}_{zh}, \mathbf{W}_{zx}, \mathbf{b}_{zx}\}$

```python
class VAE:
    ...
    def decoder(self, z):
        # Decoder network. Reconstruct the input from
        # the latent variable z
        h = relu(self.Wzh @ z + self.bzh)
        mu_x = self.Whx_mu @ h + self.bhx_mu
        logvar_x = self.Whx_var @ h + self.bhx_var
        return mu_x, logvar_x
```

Latent variable models

To summarize:

- Sample $x \sim D_n$
- Sample $z \sim q_\phi(z|x)$
- Sample $\tilde{x} \sim p_\theta(x|z)$

The sets of parameters to learn are ϕ and θ and they should be learn such that the marginal likelihood $p(x)$ is maximized.

Outline

1. Autoencoders
   - An example to keep in mind
   - Autoencoders
2. Variational autoencoders
   - Generative Models
   - Latent variable models
   - Variational Lower Bound
3. Experiment
   - Unsupervised Spam Detection
   - Preprocessing
   - Decoders
   - Spam Detector

Training VAE

We need $p_\theta(x|z)$ to be such that the marginal likelihood $p(x)$ is maximized.

In other words, the lost function should be

$$\mathcal{L}(x; \phi, \theta) = -\log p(x)$$

Kullback-Leibler divergence

Kullback-Leibler divergence

Let $p(x)$ and $q(x)$ be probability measures over a set X, and $q(x)$ is absolutely continuous with respect to $p(x)$, then the Kullback-Leibler divergence from p to q is defined as

$$\begin{aligned}\mathcal{D}_{KL}[q(x)||p(x)] =& \mathrm{E}_{x\sim q(x)}[\log q(x) - \log p(x)] \\ =& \int q(x)(\log q(x) - \log p(x))dx \\ =& \int q(x)\log q(x)dx - \int q(x)\log p(x)dx\end{aligned}$$

Gibbs' inequality

$$\mathcal{D}_{KL}[q(x)||p(x)] \geq 0$$

Variational Lower Bound

Corollary

The marginal log-likelihood $\log p(x)$ is lower bounded by the variational lower bound i.e.

$$\log p(x) \geq \mathrm{E}_{z \sim q(z|x)} \log p(x|z) - \mathcal{D}_{KL}[q(z|x)||p(z)]$$

Hence, our loss function is

$$\mathcal{L}(x; \phi, \theta) = \mathrm{E}_{z \sim q(z|x)} \log p(x|z) - \mathcal{D}_{KL}[q(z|x)||p(z)]$$

Proof of the Variational Lower Bound

Proof.

First, let's compute the Kullback-Leibler divergence $\mathcal{D}_{KL}$ between $p(z|x)$ and $q(z|x)$

$$\begin{aligned}\mathcal{D}_{KL}[q(z|x)||p(z|x)] =& \mathrm{E}_{z\sim q(z|x)}[\log q(z|x) - \log p(z|x)] \\ =& \mathrm{E}_{z\sim q(z|x)}[\log q(z|x) - \log p(x|z) - \log p(z) + \log p(x)] \\ =& \log p(x) + \mathrm{E}_{z\sim q(z|x)}[\log q(z|x) - \log p(z)] - \mathrm{E}_{z\sim q(z|x)} \log p(x|z) \\ =& \log p(x) - \mathcal{D}_{KL}[q(z|x)||p(z|x)] - \mathrm{E}_{z\sim q(z|x)} \log p(x|z)\end{aligned}$$

Sending all the terms on the right except $\log p(x)$ gives

$$\log p(x) = \mathrm{E}_{z\sim q(z|x)} \log p(x|z) - \mathcal{D}_{KL}[q(z|x)||p(z)] + \mathcal{D}_{KL}[q(z|x)|| \log p(z|x)]$$

Finally, because of Gibbs' inequality, we have

$$\boxed{\log p(x) \geq \mathrm{E}_{z\sim q(z|x)} \log p(x|z) - \mathcal{D}_{KL}[q(z|x)||p(z)]}$$

□

Solution of $\mathcal{D}_{KL}[q(z|x)||p(z)]$

Lemma

Let $z \in \mathbb{R}^J$ be a standard multivariate Gaussian distribution and let $z|x$ be a multivariate Gaussian distribution with mean $\boldsymbol{\mu}_z$ and standard deviation $\boldsymbol{\sigma}_z$. Then the KL-divergence from $p(z)$ to $q_\phi(z|x)$ is

$$\mathcal{D}_{KL}[q(z|x)||p(z)] = \frac{1}{2}\sum_{j=1}^{J} \mu_j^2 + \sigma_j^2 - 1 - \log\sigma_j^2$$

```
def kl_divergence(mu, log_sigma):
    sigma = torch.exp(log_sigma)
    return .5 * torch.sum(mu**2 + sigma**2 - 1 - 2*log_sigma, axis=1)
```

Proof for the Solution of $\mathcal{D}_{KL}[q(z|x)||p(z)]$

Proof.

According to the definition of the KL-divergence we have

$$\mathcal{D}_{KL}[q_\phi(z|x)||p(z)] = \int q_\phi(z|x)\log q_\phi(z|x)dx - \int q_\phi(z|x)\log p(z))dx$$

Solving the first integral gives

$$\begin{aligned}\int q(z|x)\log q(z|x)dz &= \int \mathcal{N}(z;\mu,\sigma)\log\mathcal{N}(z;\mu,\sigma)\\ &= -\frac{J}{2}\log 2\pi - \frac{1}{2}\sum_{j=1}^{J}(1+\log\sigma_j^2)\end{aligned} \tag{9}$$

Now solving the second integral

$$\begin{aligned}\int q(z|x)\log p(z)dz &= \int \mathcal{N}(z;\mu,\sigma)\log\mathcal{N}(z;0,I)\\ &= -\frac{J}{2}\log 2\pi - \frac{1}{2}\sum_{j=1}^{J}(\mu_j^2+\sigma_j^2)\end{aligned} \tag{10}$$

where J is the dimension of z. Finally, subtracting (10) from (9) gives

$$\mathcal{D}_{KL}[q(z|x)||p(z)] = \frac{1}{2}\sum_{j=1}^{J}\mu_j^2+\sigma_j^2-1-\log\sigma_j^2$$

□

VAE Loss Function

The loss function for a standard Gaussian latent variable and a multivariate Gaussian posterior distribution on x is given by

$$\begin{aligned}\mathcal{L}(x;\phi,\theta) &= -\operatorname{E}_{z\sim q(z|x)}\log p(x|z) + \mathcal{D}_{KL}[q(z|x)||p(z)] \\ &= -\log \mathcal{N}(x;\boldsymbol{\mu}_x,\boldsymbol{\sigma}_x) + \frac{1}{2}\sum_{j=1}^{J}\mu_j^2+\sigma_j^2-1-\log\sigma_j^2\end{aligned} \tag{11}$$

Choice of $p_\theta(x|z)$

In our example, we used $p_\theta(x|z) = \mathcal{N}(x; \boldsymbol{\mu}_z, \boldsymbol{\sigma}_z)$. In practice you can use any distribution you want.

Some important aspects to consider:

- The image of the distribution must correspond to the input space.

Choice of $p_\theta(x|z)$

In our example, we used $p_\theta(x|z) = \mathcal{N}(x; \boldsymbol{\mu}_z, \boldsymbol{\sigma}_z)$. In practice you can use any distribution you want.

Some important aspects to consider:

- The image of the distribution must correspond to the input space.
- Choose a distribution that makes sense based on your knowledge of the problem.

Choice of $p_\theta(x|z)$

In our example, we used $p_\theta(x|z) = \mathcal{N}(x; \boldsymbol{\mu}_z, \boldsymbol{\sigma}_z)$. In practice you can use any distribution you want.

Some important aspects to consider:

- The image of the distribution must correspond to the input space.
- Choose a distribution that makes sense based on your knowledge of the problem.
- If the domain is continuous on the real number, a normal distribution can make sense.

Choice of $p_\theta(x|z)$

In our example, we used $p_\theta(x|z) = \mathcal{N}(x; \boldsymbol{\mu}_z, \boldsymbol{\sigma}_z)$. In practice you can use any distribution you want.

Some important aspects to consider:

- The image of the distribution must correspond to the input space.
- Choose a distribution that makes sense based on your knowledge of the problem.
- If the domain is continuous on the real number, a normal distribution can make sense.
- If the domain is discrete, choose a discrete distribution like a Bernoulli or a Poisson.

Outline

1. Autoencoders
   - An example to keep in mind
   - Autoencoders
2. Variational autoencoders
   - Generative Models
   - Latent variable models
   - Variational Lower Bound
3. Experiment
   - Unsupervised Spam Detection
   - Preprocessing
   - Decoders
   - Spam Detector

Unsupervised Spam Detection

- SMS Spam Collecion Data Set[2]
  - Number of instances: 5574
  - Number of spams: 747 ($\sim 13\%$)
- Spam detection
  - Estimate the distribution of all the text messages
  - Evaluate the density of each text message
  - Text messages with low density are classified as spam

[2] https://archive.ics.uci.edu/ml/datasets/sms+spam+collection

Data

- Examples of **non spams**
  - *What are you doing? how are you?*
  - *Ok lar... Joking wif u oni...*
- Examples of **spams**
  - *Your mobile No 07808726822 was awarded a L2,000 Bonus Caller Prize on 02/09/03! This is our 2nd attempt to contact you! Call 0871-872-9758 as soon as possible.*

Outline

1. Autoencoders
   - An example to keep in mind
   - Autoencoders
2. Variational autoencoders
   - Generative Models
   - Latent variable models
   - Variational Lower Bound
3. Experiment
   - Unsupervised Spam Detection
   - Preprocessing
   - Decoders
   - Spam Detector

Bag of words

- $x^{(i)} =$ *What you doing?how are you?* (original sentence)
- $x^{(i)} =$ *what you do how be you* (after preprocessing)
- $x^{(i)} = [0 \cdots 0\ 1\ 0 \cdots\ 0\ 2\ 0 \cdots 0]$ (vector representation)

```
x = 'What you doing?how are you?'

x = preprocess(x)
print(x)
>>> 'what you do how be you'

x = vectorize(x)
print(x)
>>> [0 0 0 ... 0 0 0]
```

Bag of words

- Bag of words
  - All examples in the corpus have the same length.
  - The value of a position in the vector is equal to the frequency of the corresponding word in the example.
  - $x^{(i)} = [0 \cdots 0\ 1\ 0 \cdots\ 0\ 2\ 0 \cdots 0]$
- Binary bag of words
  - All examples in the corpus have the same length.
  - Each element in the vector is either 0 or 1.
  - $x^{(i)} = [0 \cdots 0\ 1\ 0 \cdots\ 0\ 1\ 0 \cdots 0]$

Outline

1. Autoencoders
   - An example to keep in mind
   - Autoencoders
2. Variational autoencoders
   - Generative Models
   - Latent variable models
   - Variational Lower Bound
3. Experiment
   - Unsupervised Spam Detection
   - Preprocessing
   - Decoders
   - Spam Detector

Bernoulli Decoder $p_\theta(x|z)$

- $\boldsymbol{\gamma} = g(z; \boldsymbol{\theta})$
  - $\mathbf{h} = \text{relu}\left(\mathbf{W}_{zh}\mathbf{z} + \mathbf{b}_{zh}\right)$
  - $\boldsymbol{\gamma} = \sigma\left(\mathbf{W}_{hx}\mathbf{h} + \mathbf{b}_{hx}\right)$
- $p_\theta(\mathbf{x}|\mathbf{z}) = \prod_i \gamma_i^{x_i}(1-\gamma_i)^{(1-x_i)}$

```
class VAE:
    ...
    def decoder(self, z):
        h = relu(self.W_zh @ z + self.b_zh)
        gamma = F.sigmoid(self.W_hx @ h + self.b_hx)
        reture gamma
```

Poisson Decoder $p_\theta(x|z)$

- $\boldsymbol{\lambda} = g(\mathbf{z}; \boldsymbol{\theta})$
  - $\mathbf{h} = \mathrm{relu}\left(\mathbf{W}_{zh}\mathbf{z} + \mathbf{b}_{zh}\right)$
  - $\log \boldsymbol{\lambda} = \tanh\left(\mathbf{W}_{hx}\mathbf{h} + \mathbf{b}_{hx}\right)$
- $p_\theta(\mathbf{x}|\mathbf{z}) = \prod_i e^{-\lambda_i} \frac{\lambda_i^{x_i}}{x_i!}$

```
class VAE:
    ...
    def decoder(self, z):
        h = relu(self.W_zh @ z + self.b_zh
        loglambda = F.tanh(self.W_hx @ h + self.b_hx)
        reture loglambda
```

Outline

1. Autoencoders
   - An example to keep in mind
   - Autoencoders
2. Variational autoencoders
   - Generative Models
   - Latent variable models
   - Variational Lower Bound
3. Experiment
   - Unsupervised Spam Detection
   - Preprocessing
   - Decoders
   - Spam Detector

Anomaly detection

Algorithm 2 Pseudocode for Anomaly detection

Require: Density Threshold $\mathcal{T}$

Require: Sets of trained parameters ϕ and θ

Sample $\mathbf{z}^{(1)}, ..., \mathbf{z}^{(1)}$ from $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$

Compute $p(\mathbf{x}) \approx \frac{1}{L}\sum_{i=1}^{L} p_\theta(\mathbf{x}|\mathbf{z}^{(i)})$

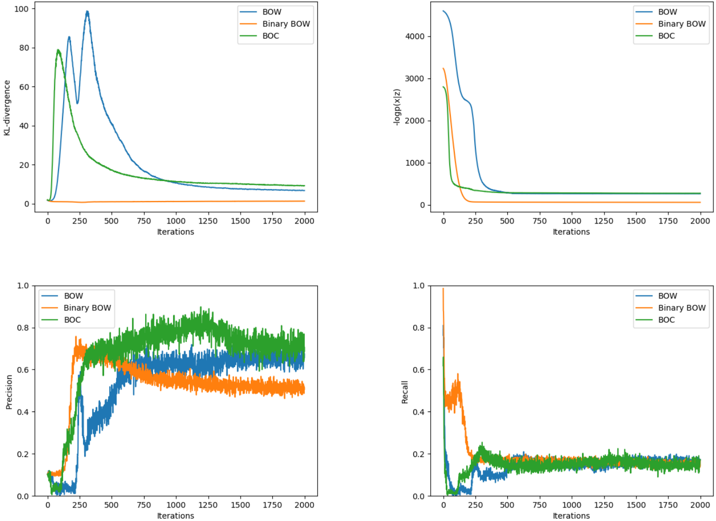
If $p(\mathbf{x}) < \mathcal{T}$, then $\mathbf{x}$ is classified as an outlier

Anomaly detection

Algorithm 3 Pseudocode to find the density threshold

Require: Percentile threshold k
Require: Trained parameters ϕ and θ
  for $i = 1$ to N **do**
    Sample $\mathbf{z}^{(1)}, ..., \mathbf{z}^{(1)}$ from $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$
    Compute $p(\mathbf{x}^{(i)}) \approx \frac{1}{L}\sum_{i=1}^{L} p_\theta(\mathbf{x}|\mathbf{z}^{(i)})$
  end for
  $\mathcal{T} = p(\boldsymbol{x}^{(\tau)})$ where $|\{\boldsymbol{x} : p(\boldsymbol{x}) < p(\boldsymbol{x}^{(\tau)})\}| = k \cdot N$

Results


KL-divergence
Iterations
-logp(x|z)
Precision
Recall
BOW
Binary BOW
BOC
0
250
500
750
1000
1250
1500
1750
2000

Results

Table: Test results of the spam detection and the modelization of the text distribution. The test data represent 50% of the entire data set. The confidence interval are due to the randomness caused by the sampling of the latent variable $\mathbf{z}$ when evaluating the probability.

model	precision	recall	$\log p(\mathbf{x}\|\mathbf{z})$	$\mathcal{D}_{KL}$
BOW	0.62 ± 0.006	0.16 ± 0.004	-258.62 ± 0.33	8.27 ± 2.580
Binary BOW	0.64 ± 0.003	0.22 ± 0.003	-48.89 ± 0.03	1.25 ± 0.000
BOC	0.79 ± 0.005	0.26 ± 0.004	-279.95 ± 0.41	14.17 ± 4.560