Variational Autoencoders

Kingma et al. (2014)

JE Starling, 10-2017

Fall 2017

Review of Variational Inference

Setting:

x is our data.
z is our latent variable.
Joint density is $p(x,z) = p(x|z) \cdot p(z)$

Goal: Approximate the posterior $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$,
where marginal $p(x) = \int p(x|z)p(z)dz$ is intractible.

Strategy: We pose a family of approximations, Q, and choose a member of that family, $q(z) \in Q$, to minimize $KL\left[q(z) \; || \; p(z|x)\right]$.

Review of Variational Inference (2)

We want to find the best approximation:

$$q^*(z) = \underset{q(z) \in Q}{\arg\min}\, KL\left[q(z) \;||\; p(z|x)\right]$$

This objective is intractible because it involves $p(x)$:

$$\begin{aligned} KL\left[q(z) \;||\; p(z|x)\right] &= E_{q(z)}\left[\log\left(\frac{q(z)}{p(z|x)}\right)\right] \\ &= E_{q(z)}\left[\log\left(q(z)\right)\right] - E_{q(z)}\left[\log\left(p(x,z)\right)\right] + \log\left(p(x)\right) \end{aligned}$$

Review of Variational Inference (3)

We can maximize another quantity which is equivalent to minimizing the KL divergence. (The $\log(p(x))$ term is constant wrt $q(z)$.)

$$KL\left[q(z) \,||\, p(z|x)\right] = \underbrace{E_{q(z)}\left[\log(q(z))\right] - E_{q(z)}\left[\log(p(x,z))\right]}_{-ELBO(q)} + \log(p(x))$$

We can write $ELBO(q)$ as

$$\begin{aligned} ELBO(q) &= E_{q(z)}\left[\log(q(z))\right] - E_{q(z)}\left[\log(p(x,z))\right] \\ &= E_{q(z)}\left[\log(p(z))\right] + E_{q(z)}\left[\log(p(x|z))\right] - E_{q(z)}\left[\log(q(z))\right] \\ &= E_{q(z)}\left[\log(p(x|z))\right] - KL\left[q(z) \,||\, p(z)\right] \end{aligned}$$

The $ELBO(q)$ is a lower bound on $\log(p(x))$.

Variational Auto Encoders: Notation Note

Going forward, we will write $q(z)$ as $q(z|x)$.

We will also add a subscript to indicate that $q(z|x)$ is parameterized by variational parameters labeled ϕ.

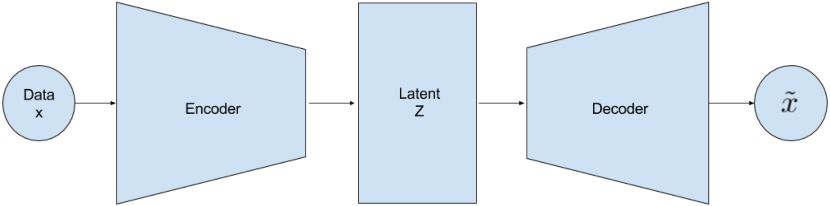
We write: $q_\phi(z|x)$

Purpose of VAEs

What do VAEs do?

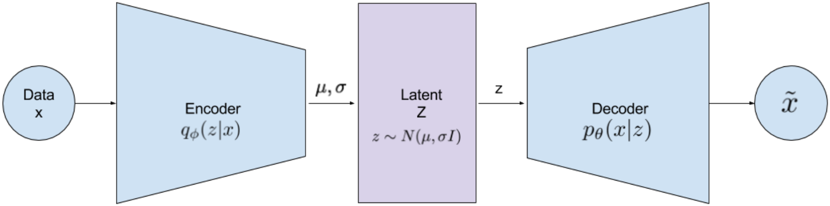
- Fit a generative model to a large dataset.
- Model the underlying distribution of the data and sample new data from the distribution.
- Image re-generation, and generating new images like existing ones.

Basic VAE Setup



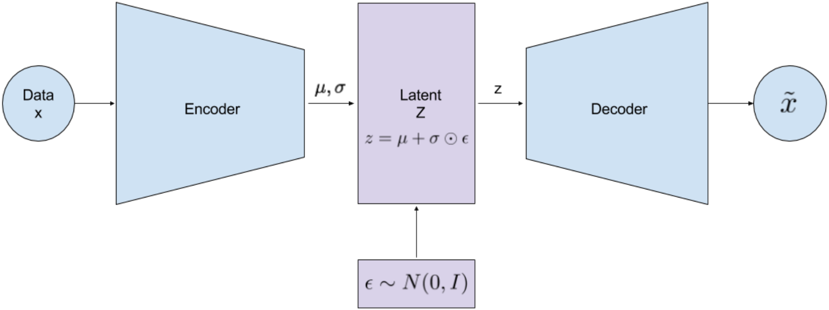
In neural nets language, a VAE consists of an encoder, a decoder, a latent variable, and a loss function.

The Latent Variables



- Latent z is continuous, lower-dimensional; dimension is a user input.
- Let the prior $p(z)$ be multivariate Gaussian with a diagonal covariance matrix: $p(z) \sim N(\mu, \sigma^2 I)$.

The Reparameterization Trick



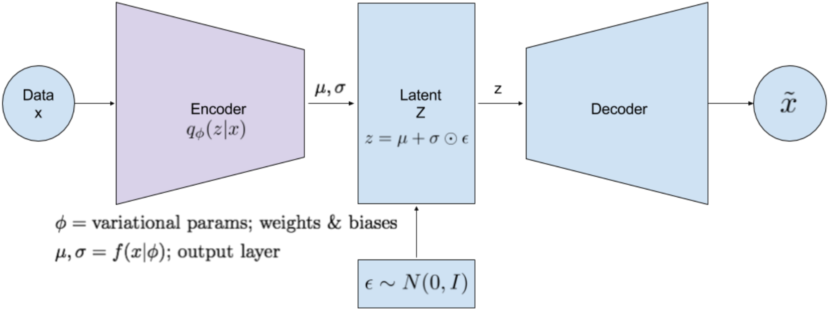
Use $z = \mu + \sigma \odot \epsilon$, where $\epsilon \sim N(0, I)$, instead of sampling $z \sim N(\mu, \sigma I)$.

The Reparameterization Trick (2)

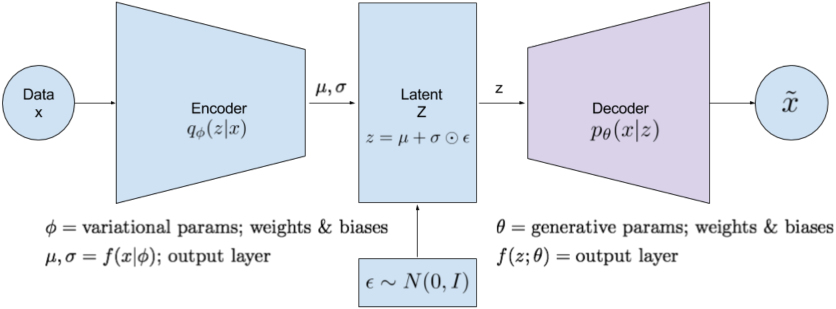
Interpretation of σ:

- Think of z as encoding information in a lower-dimensional space. How to prevent encoding an infinite amount of info?
- What if we have two x's which are very different, mapped to two z's which are close?
- σ acts as noise, which prevents reconstructing two different $\tilde{x}$ values from z's which are less than σ apart. Regulates how much info you can encode.

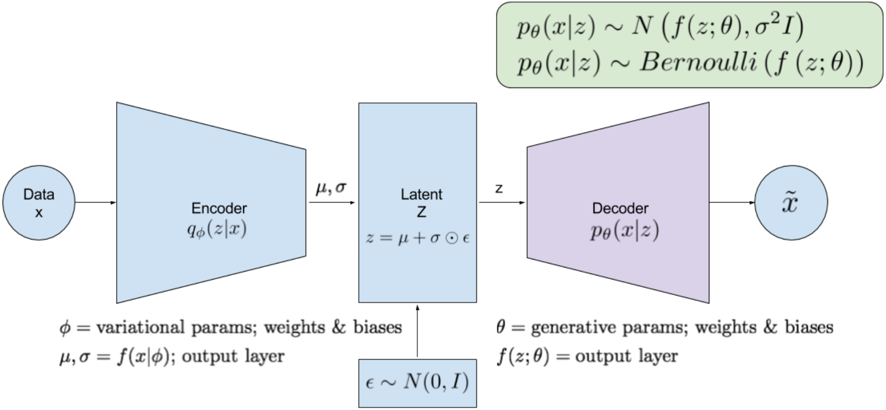
The Encoder


Data
x
Encoder
$q_\phi(z|x)$
μ, σ
Latent
Z
$z = \mu + \sigma \odot \epsilon$
z
Decoder
$\tilde{x}$
$\phi =$ variational params; weights & biases
$\mu, \sigma = f(x|\phi)$; output layer
$\epsilon \sim N(0, I)$

The Decoder


Data
x
Encoder
$q_\phi(z|x)$
μ, σ
Latent
Z
$z = \mu + \sigma \odot \epsilon$
z
Decoder
$p_\theta(x|z)$
$\tilde{x}$
ϕ = variational params; weights & biases
$\mu, \sigma = f(x|\phi)$; output layer
$\epsilon \sim N(0, I)$
θ = generative params; weights & biases
$f(z;\theta)$ = output layer

The Decoder



$p_\theta(x|z) \sim N\left(f(z;\theta), \sigma^2 I\right)$
$p_\theta(x|z) \sim Bernoulli\left(f\left(z;\theta\right)\right)$
Data
x
Encoder
$q_\phi(z|x)$
μ, σ
Latent
Z
$z = \mu + \sigma \odot \epsilon$
z
Decoder
$p_\theta(x|z)$
$\tilde{x}$
ϕ = variational params; weights & biases
$\mu, \sigma = f(x|\phi)$; output layer
$\epsilon \sim N(0, I)$
θ = generative params; weights & biases
$f(z;\theta)$ = output layer

Training the Model

Model parameters (ϕ, θ) are trained using stochastic gradient descent with back-propogation.

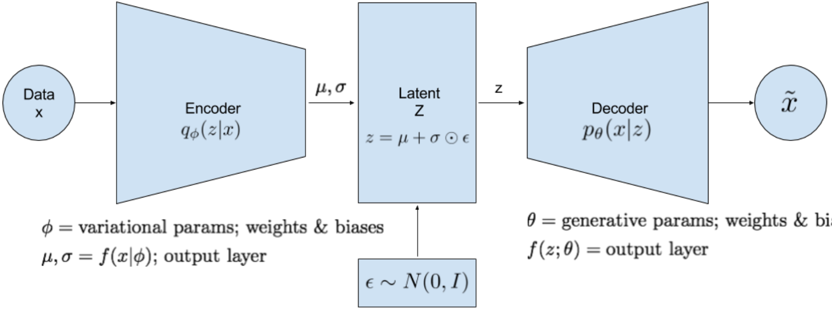
- (ϕ, θ) are learned jointly.
- With step size ρ, encoder and decoder parameters are updated as

$$\phi_{\text{new}} = \phi_{\text{old}} - \rho \frac{\partial L_i(\phi, \theta, x_i)}{\partial \phi}$$
$$\theta_{\text{new}} = \theta_{\text{old}} - \rho \frac{\partial L_i(\phi, \theta, x_i)}{\partial \theta}$$

where $L_i(\phi, \theta, x_i)$ is a loss function.

The Loss Function

$$p_\theta(x|z) \sim N\left(f(z;\theta), \sigma^2 I\right)$$
$$p_\theta(x|z) \sim Bernoulli\left(f\left(z;\theta\right)\right)$$



ϕ = variational params; weights & biases
$\mu, \sigma = f(x|\phi)$; output layer

θ = generative params; weights & biases
$f(z;\theta)$ = output layer

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

Tasks we can accomplish after training

(1) Visualize our latent space and draw samples from it.

(2) Efficient approximate marginal inference of x:

- Reconstruct data sets we have seen.
- Generate new data sets that are like one we have already seen.
- De-noise input data sets.

Pseudo-code

```
network= {

  # encoder
  encoder_x = Input_layer(size=input_size, input=data)
  encoder_h = Dense_layer(size=hidden_size, input= encoder_x)

  # the re-parameterized distributions that are inferred from data
  z_mean = Dense(size=number_of_distributions, input=encoder_h)
  z_variance = Dense(size=number_of_distributions, input=encoder_h)
  epsilon= random(size=number_of_distributions)

  # decoder network needs a sample from the code distribution
  z_sample= z_mean + exp(z_variance / 2) * epsilon

  #decoder
  decoder_h = Dense_layer(size=hidden_size, input=z_sample)
  decoder_output = Dense_layer(size=input_size, input=decoder_h)
}

cost={
  reconstruction_loss = input_size * crossentropy(data, decoder_output)
  kl_loss = - 0.5 * sum(1 + z_variance - square(z_mean) - exp(z_variance))
  cost_total= reconstruction_loss + kl_loss
}

stochastic_gradient_descent(data, network, cost_total)
```

Sampling of papers

- Variational Autoencoder for Deep Learning of Images, Labels and Captions
- Alternative priors for Deep Generative Models
- Least Squares VAE with Regularization
- Stein VAE
- Ladder VAEs

MNIST Example (on github)

In the example on github:

- Uses the MNIST data set.
- Uses a gaussian encoder and a bernoulli decoder.
- Both neural nets have two hidden layers.
- Uses drop-outs (tf.nn.dropout) to avoid overfitting.
- Results following used 2-dimensional z.

MNIST Example (2)

Python code has four scripts.

- **mnist_data.py** For downloading and formatting MNIST data. (Also included in TensorFlow.)
- **run_main.py** The main file; processes inputs, trains network using SGD, calls plot function.
- **vae.py** Contains functions for the encoder, decoder, and overall network.
- **plot_utils.py** Creates output plots.