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:

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

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

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

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 $q(z)$.)

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

We can write $ELBO(q)$ as

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

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

Variational Auto Encoders: Notation Note

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

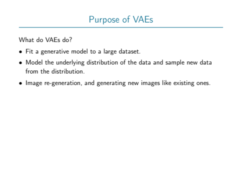
We will also add a subscript to indicate that $q(z|x)$ is parameterized by variational parameters labeled ϕ.

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.



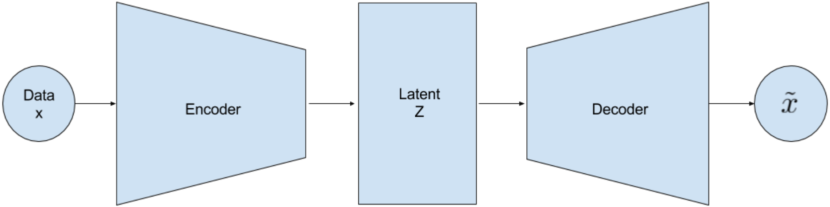
Purpose of VAEs

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.

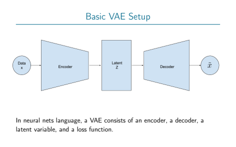
- Why not use a regular auto-encoder? They encode and decode data well, but do not define a probability distribution for the data, and do not do well at generating more data similar to the training set. VAEs can do this easily.
- VAE comes from the idea that directed graphical models can represent complex distributions, while deep neural nets can repesent arbitrarily complex functions. So we can use deep neural nets to parametersize and represent conditional distributions.
- Stochastic gradient descent makes this feasible.

Basic VAE Setup



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

Basic VAE Setup

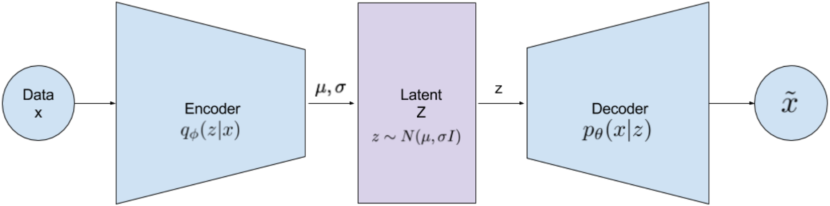


Basic VAE Setup

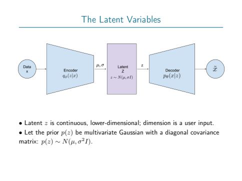
In neural nets language, a VAE consists of an encoder, a decoder, a latent variable, and a loss function.

- The encoder is a neural network with at least one hidden layer. The encoder maps the data to a distribution for the hidden variables
- The decoder is another neural network with at least one hidden layer. The decoder becomes a generative network that maps the latent variables back to the data
- Thelatent variable is actually a probability distribution, we need to sample the hidden code from its distribution to be able to generate data.

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: $p(z) \sim N(\mu, \sigma^2 I)$.

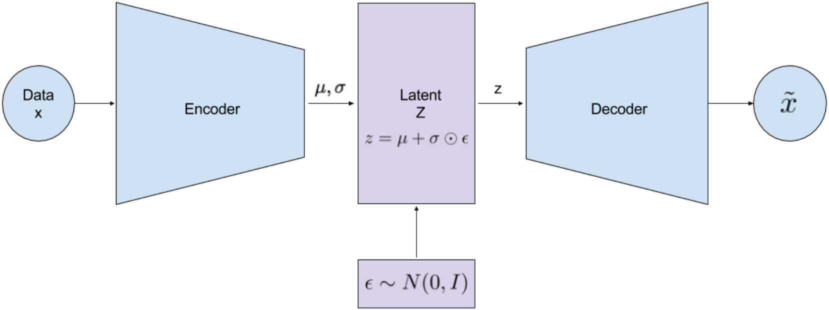


The Latent Variables

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

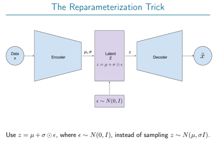
- Letting z be Gaussian means our loss function will have a nice form; we will see this later.
- So z is multivariate Normal, with a diagonal covariance matrix - independent observations.
- We can generate any distribution in d dimensions by generating normals and mapping them through a sufficiently complex function.

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



The Reparameterization Trick

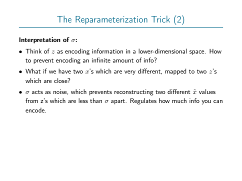
Use $z = \mu + \sigma \odot \epsilon$, where $\epsilon \sim N(0, I)$, instead of sampling $z \sim N(\mu, \sigma I)$.

- To use stochastic gradient descent with this autoencoder, we need to be able to calculate gradients w.r.t. all nodes.
- Problem: latent variables z are random variables (distributions) which are not differentiable.
- We treat the mean and variances of the distributions as simple deterministic parameters and multiply the variance by a sample from a normal noise generator to add randomness.
- By parameterizing this way, we can back-propagate the gradient to the parameters of the encoder and train the whole network with SGD.
- This procedure will allow us to learn mean and variance values for the hidden code and it's called the "re-parameterization trick".
- It is important to appreciate the importance of the fact that the whole network is now differentiable. This means that optimization techniques can now be used to solve the inference problem efficiently.

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.



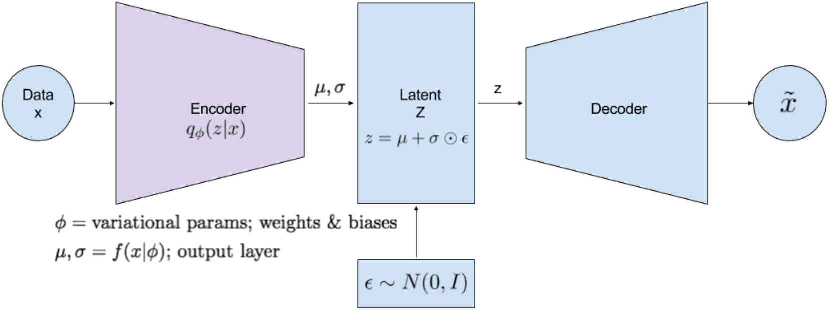
The Reparameterization Trick (2)

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.

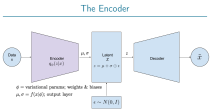
- This lets us avoid deciding by hand what each dimension of z encodes.
- VAEs assume there is no simple interpretation of the dimensions of z.
- So we can draw samples of z from a standard multivariate Gaussian.

The Encoder



ϕ = variational params; weights & biases

$\mu, \sigma = f(x|\phi)$; output layer

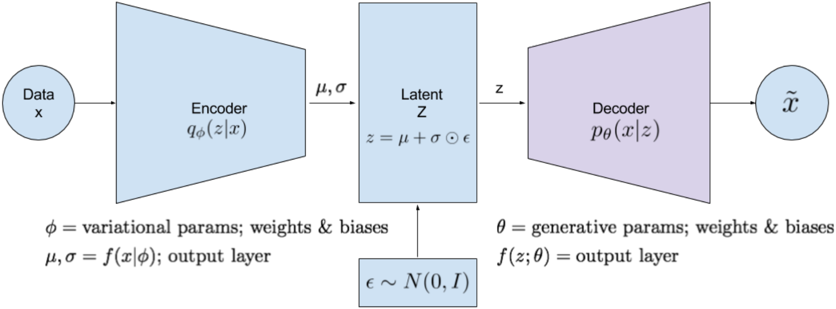


The Encoder

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

- This is the first neural net.
- Both the encoder and decoder should have same number of neurons; want to reconstruct same picture or similar.
- Encoder outputs the parameters for the latent gaussian.
- Last time with regular VI, this $q_\theta(z|x)$ was fit using CAVI, or other methods. This time, we are tuning the ϕ weights and biases, and obtaining the μ and σ which define z as the output layer.

The Decoder



Data x

Encoder $q_\phi(z|x)$

μ, σ

Latent Z

$z = \mu + \sigma \odot \epsilon$

z

Decoder $p_\theta(x|z)$

$\tilde{x}$

$\epsilon \sim N(0, I)$

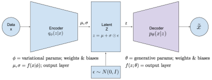
ϕ = variational params; weights & biases

$\mu, \sigma = f(x|\phi)$; output layer

θ = generative params; weights & biases

$f(z;\theta)$ = output layer

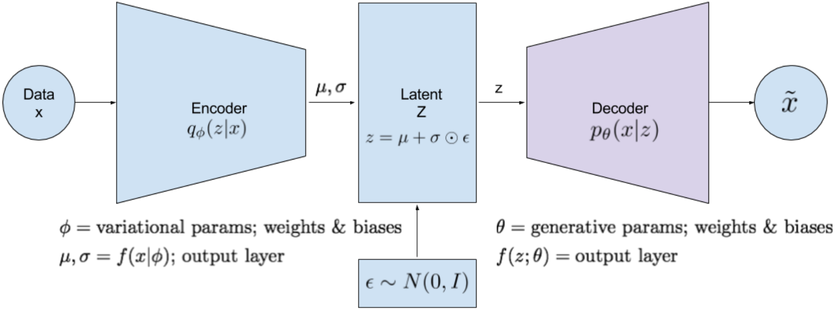
The Decoder


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 is another neural net.
- θ are the weights and biases; we call them the 'generative parameters'.
- The output layer parameterizes the distribution we will use to reconstruct the x's.

The Decoder

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



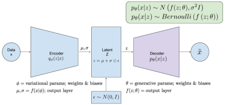
$\phi =$ variational params; weights & biases

$\mu, \sigma = f(x|\phi)$; output layer

$\theta =$ 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

- The output layer is the $f(z|\theta)$, some function of the z's and the neural net.
- We then use that function output to generate from our choice of $p_\theta(x|z)$.
- Bernoulli and Gaussian are common choices.
- Bernoulli would be for black-white images; pixels are black or white.
- Gaussian would represent images with more scale in the pixels.
- Key: $p_\theta(x|z)$ must be computable, and continuous in θ.

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

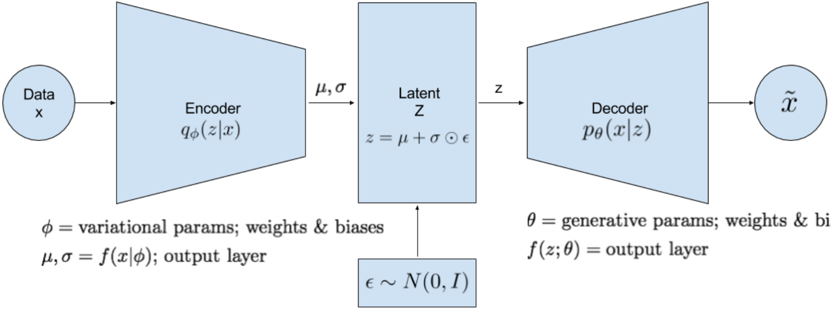
$$\phi_{\text{new}} = \phi_{\text{old}} - \rho \frac{\partial L_i(\phi, \theta, x_i)}{\partial \phi}$$

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

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

The Loss Function

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



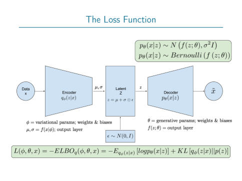
$\phi =$ variational params; weights & biases

$\mu, \sigma = f(x|\phi)$; output layer

$\theta =$ generative params; weights & biases

$f(z;\theta) =$ output layer

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



The Loss Function

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

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

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

- Our old friend, the ELBO, ie the lower bound on the evidence (data given model).
- We said earlier that maximizing the ELBO is equivalent to minimizing the KL divergence between the true posterior and our approximation.
- From the perspective of autoencoders, the ELBO function can be seen as the sum of the reconstruction cost of the input plus the regularization terms.
- Maximizing the ELBO minimizes the error distance. We can think of this as gradient ascent, or gradient descent to minimize the negative ELBO. This whole algorithm is called "autoencoding variational Bayes".

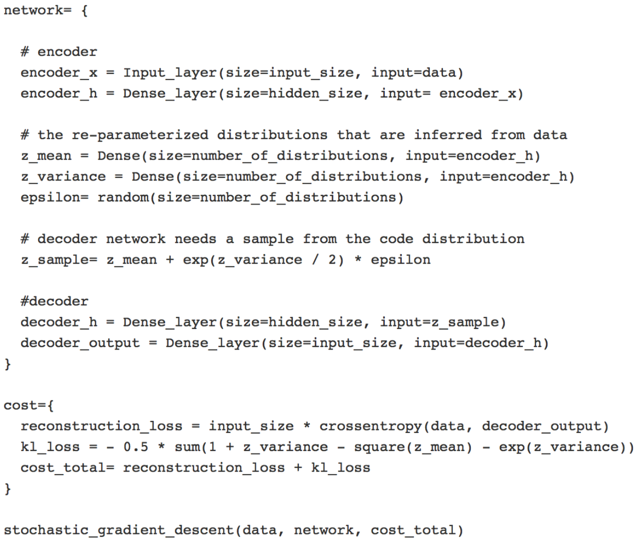
Tasks we can accomplish after training

(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.

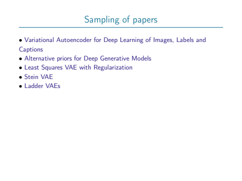
Pseudo-code



```
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



Sampling of papers

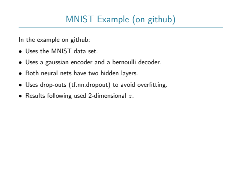
- Variational Autoencoder for Deep Learning of Images, Labels and Captions
- Alternative priors for Deep Generative Models
- Least Squares VAE with Regularization
- Stein VAE
- Ladder VAEs

- VAE for Deep Learning of Images: A novel variational autoencoder is developed to model images, as well as associated labels or captions. The Deep Generative Deconvolutional Network (DGDN) is used as a decoder of the latent image features, and a deep Convolutional Neural Network (CNN) is used as an image encoder; the CNN is used to approximate a distribution for the latent DGDN features/code.
- Alternative Priors: Gives good overview of tons of stuff going on, including non-gaussian.
- Ladder VAEs: Variational autoencoders are powerful models for unsupervised learning. However deep models with several layers of dependent stochastic variables are difficult to train which limits the improvements obtained using these highly expressive models. We propose a new inference model, the Ladder Variational Autoencoder, that recursively corrects the generative distribution by a data dependent approximate likelihood in a process resembling the recently proposed Ladder Network.

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 (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.

- Dropout refers to ignoring neurons during the training phase of certain set of neurons which is chosen at random. By “ignoring”, I mean these units are not considered during a particular forward or backward pass.

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.