A Tutorial: Variational Autoencoders

Ben Herbst
MML, 10 November 2017

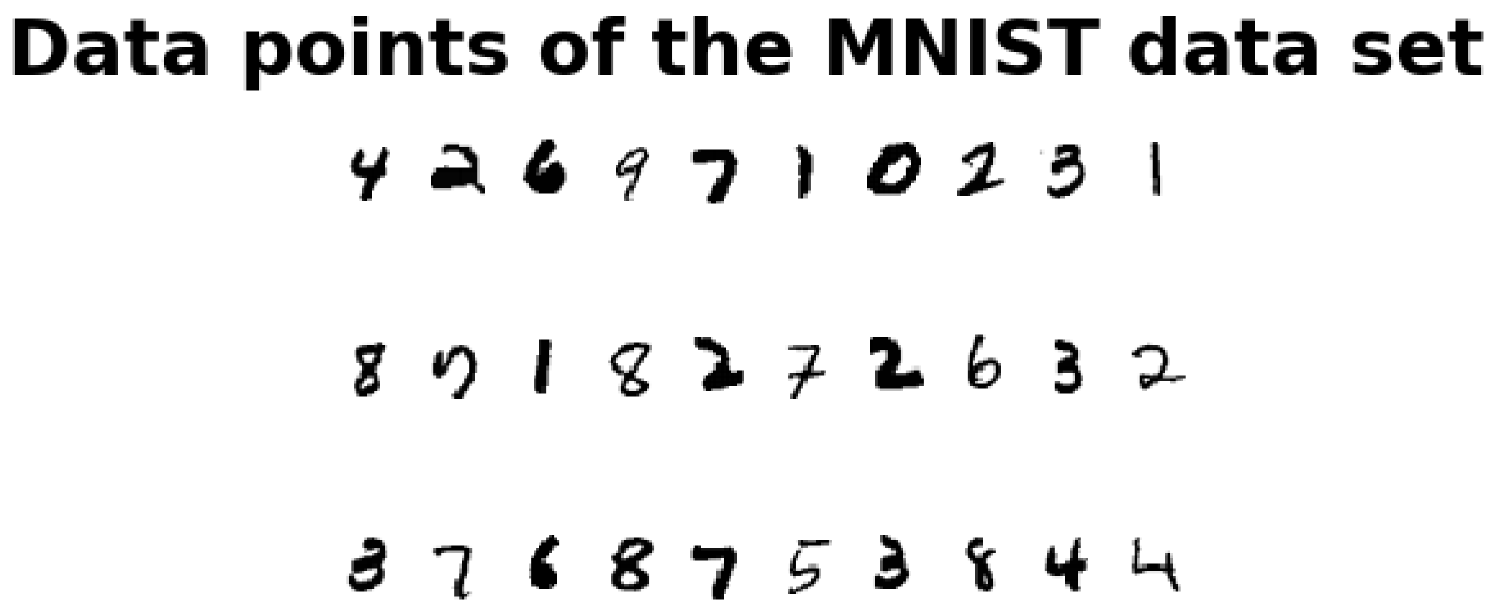
References:

- Carl Doersch. Tutorial on Variational Autoencoders
- Ulrich Paquet. Deep Learning Indaba, Wits
- Kingma and Welling. Auto-Encoding Variational Bayes
- Aurélien Géron. Hands-On Machine Learning with Scikit-Learn & Tensorflow

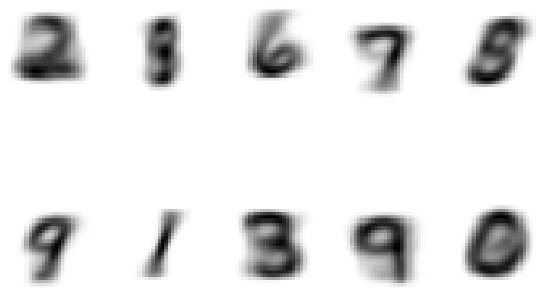
- Given *unlabelled* data:

$$\mathbf{x}^{(j)} \sim p(\mathbf{x}), \quad j = 1, \ldots, n$$

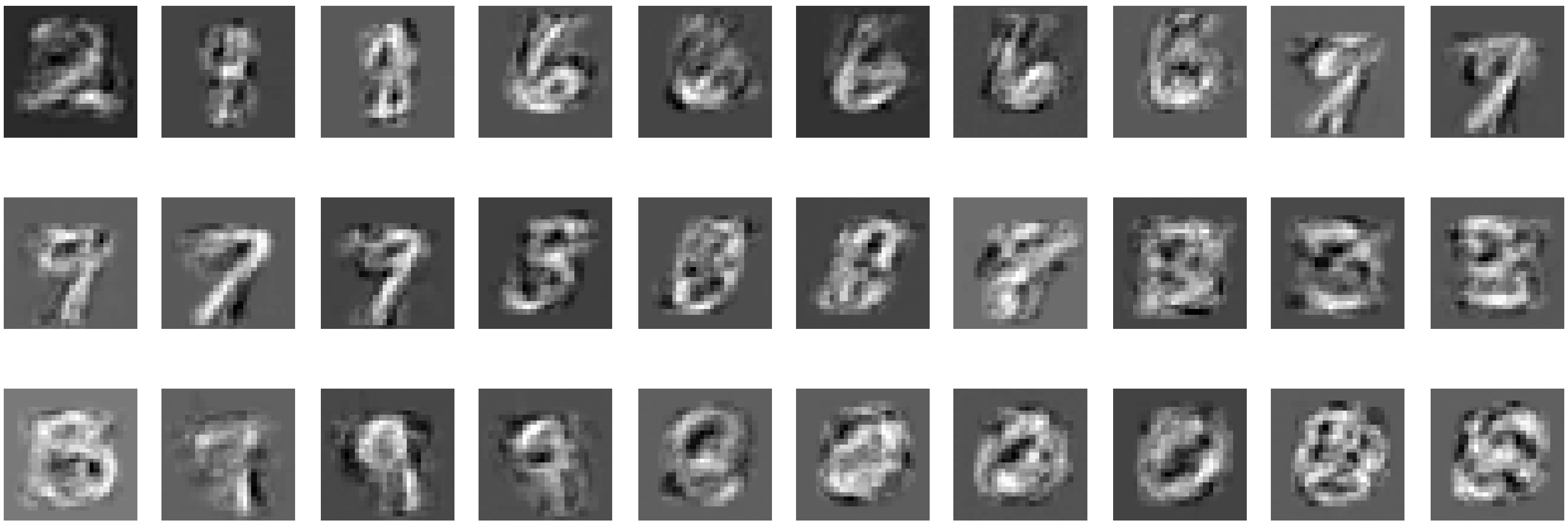
  generated by an unknown, and unknowable, distribution $p(\mathbf{x})$.
- Problem: Build a generative model from the data
- Tough — infinite number of possibilities



Mixture Component Means



GMM Generated Samples



Idea I: Latent variable

Generative models need to learn complicated dependencies between the different dimensions (pixels).
This is hard.

- With latent variable z, the model becomes

$$p(x, z) = p_\theta(x|z)p(z)$$

- Ancestral sampling

$$\begin{aligned} z^{(\ell)} &\sim p(z) \\ x^{(\ell)} &\sim p_\theta(x|z^{(\ell)}) \end{aligned}$$

- Sample the latent variable z — decide which digit to generate
- Each meaningful data value should correspond to a latent variable
- Latent variable is an efficient, compact representation of the data
- Leads to a general, powerful framework for generative models
- Ill-posed problem — Infinite number of choices for $p(x, z)$

Need to make assumptions!

How do we do it?

- Need both the prior $p(z)$ as well as the likelihood $p_\theta(x|z)$.
- Choose prior

$$p(z) = \mathcal{N}(z|0, I)$$

  - $p(z)$ has no parameters to learn!
  - Learn $p_\theta(x|z)$ from the data

- How is this even possible?
- Each sample $z \sim p(z)$ has to correspond to a meaningful observation

This is exactly what we are going to do!

Learning from data

- Relate $p_\theta(x|z)$ to (unknown) model $p(x)$

$$\begin{aligned} p(x) &= \int p_\theta(x|z)p(z)dz \\ &= \mathbb{E}_{p(z)}\left[p_\theta(x|z)\right] \end{aligned}$$

- Since we have samples from $p(x)$, perhaps we can calculate likelihood, or

$$\mathbb{E}_{p(x)}\left[\mathbb{E}_{p(z)}\left[p_\theta(x|z)\right]\right]$$

Progress: A relationship between latent variable z and the data produced by $p(x)$

- Cannot relate any latent sample to any specific observation — this is unsupervised learning
- Need to relate two distributions, $p(z)$ and $p(x|z)$

CHOOSING $p_\theta(x|z)$ — WOW??

- For binary images, **pixel-wise**, choose Bernoulli distribution

$$p_\theta(x|z) = \rho^x (1-\rho)^{1-x}, \quad x \in \{0, 1\}$$

- For grayscale images, **pixel-wise**, choose

$$p_\theta(x|z) = \begin{cases} \frac{\ln\left(\frac{\rho}{1-\rho}\right)}{2\rho - 1} \rho^x (1-\rho)^{1-x}, & \rho \neq \frac{1}{2} \\ 1, & \rho = \frac{1}{2} \end{cases}, \quad x \in [0, 1]$$

- Gausian, **pixel-wise**

$$p_\theta(x|z) = \mathcal{N}(x|\mu, \sigma^2)$$

WHAT HAPPENED TO z, AND WHAT EXACTLY ARE THE PARAMETERS?

Idea II: $\rho = f_\phi(z)$

Recap

- Want to sample from latent variable

$$\begin{aligned} z^{(\ell)} &\sim p(z) \\ x^{(\ell)} &\sim p_\theta(x|z^{(\ell)}) \end{aligned}$$

- Specify $p(z) = \mathcal{N}(z|0, I)$.
- Specify $p_\theta(x|z)$, pick one
- Somehow this should allow us to produce more samples — approximate $p(x)$

Map z to x through a deterministic function

- Choosing $p_\theta(x|z)$ as Bernoulli: $\rho = f_\phi(z)$, i.e. $p_\theta(x|z) = \mathrm{Be}\left(x|\rho(z)\right)$
- Choosing $p_\theta(x|z)$ Gaussian: $\mu = \mu_\phi(z), \quad \sigma = \sigma_\phi(z)$.

$z \sim \mathcal{N}(z|0, I)$ $\quad$ $x = z/10 + z/\,||z||$

Good idea to map $p(z)$ to $p(x|z)$ through deterministic function

HARD TO DIRECTLY LEARN THE NONLINEAR MAP

- Need to fully, and simultaneously sample from both spaces — hard
- Recall: Not possible to relate sample $z \sim p(z)$ with any observation
- Hard to find good objective function

GO BY THIS INDIRECTLY

Idea III: Sample from posterior

Posterior: $p(z|x)$

- Conditioning on known samples of x, constrains the distributions over the latent variable z

But, $p(z|x)$ is not tractable!!

We need a miracle!

Idea IV: Approximation of posterior

Approximate $p(z|x)$ which is intractable with $q_\phi(z|x)$ which can be computed

Error in the approximation, Kullback-Leibler divergence

$$\begin{aligned} \mathrm{KL}[q_\phi(z|x)||p(z|x)] &= \mathbb{E}_{q_\phi(z|x)}\left[\ln q_\phi(z|x) - \ln p(z|x)\right] \\ &= -\int q_\phi(z|x) \ln \left[\frac{p(z|x)}{q_\phi(z|x)}\right] dz \\ &\geq 0 \end{aligned}$$

- Equality iff $q_\phi(z|x) = p(z|x)$.
- Apply Bayes' rule

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

The miracle: The Evidence Lower Bound (ELBO)

Applying Bayes' rule to the KL-divergence

$$\mathrm{KL}[q_\phi(z|x)||p(z|x)] = \ln p(x) - \mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(x|z)] + \mathrm{KL}[q_\phi(z|x)||p(z)]$$

or

$$\begin{aligned} \ln p(x) &= \underbrace{\mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(x|z)] - \mathrm{KL}[q_\phi(z|x)||p(z)]}_{\text{ELBO}} + \mathrm{KL}[q_\phi(z|x)||p(z|x)] \\ &= \underbrace{\mathrm{KL}[q_\phi(z|x)||p(x,z)]}_{\text{ELBO}} + \mathrm{KL}[q_\phi(z|x)||p(z|x)] \end{aligned}$$

- Recall: Started with

$$p(x) = \mathbb{E}_{p(z)}[p_\theta(x|z)]$$

Working The miracle

Applying Bayes' rule to the KL-divergence

$$\begin{aligned}\ln p(x) &= \text{ELBO} + \text{KL}[q_\phi(z|x)||p(z|x)] \\ &\geq \text{ELBO}\end{aligned}$$

- In order to minimize $\text{KL}[q_\phi(z|x)||p(z|x)]$

MAXIMIZE

$$\text{ELBO} = \mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(x|z)] - \text{KL}[q_\phi(z|x)||p(z)]$$

This does the following:

I. Pushes $q_\phi(z|x)$ closer to intractable $p(z|x)$

II. Pushes $q_\phi(z|x)$ closer to $p(z)$ — want to sample from $p(z)$!

III. Maximize the expected value of $\ln p_\theta(x|z)$ — maximum likelihood

Dissecting ELBO

Minimize negative ELBO, i.e. *minimize*

$$-\mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(x|z)] + \mathrm{KL}[\ln q_\phi(z|x)||p(z)]$$

- Choose $p(z) = \mathcal{N}(z|0, I)$
- Choose $q_\phi(z|x) = \mathcal{N}\left(z|\mu(x), \sigma^2(x)\right)$ — yet to be explained

- Straightforward calculation shows that

$$\mathrm{KL}[\ln q_\phi(z|x)||p(z)] = \frac{1}{2}\left[-1 - \ln \sigma^2(x) + \sigma^2(x) + \mu^2(x)\right]$$

- Use Monte Carlo sampling to evaluate $-\mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(x|z)] = -\int q_\phi(z|x) \ln p_\theta(x|z) dz$
  For $z^{(\ell)} \sim q_\phi(z|x), \quad \ell = 1, \ldots, m,$

$$-\mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(x|z)] \approx \frac{1}{m}\sum_{\ell=1}^{m} p_\theta\left(x|z^{(\ell)}\right)$$

- In practice use $m = 1$

Putting it all together

Given an observation $\mathbf{x}$, do

- Encoder: $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(z|\mu(\mathbf{x}), \sigma^2(\mathbf{x})\right)$
- Sample: $\mathbf{z}^{(\ell)} \sim q_\phi(\mathbf{z}|\mathbf{x})$
- Decoder: $\rho = \rho(\mathbf{z})$
- Cross entropy: $-\ln p_\theta(x|\mathbf{z}) = -x \ln \rho(\mathbf{z}) - (1-x)\ln(1-\rho(\mathbf{z}))$
  (This is for each dimension of $\mathbf{x}$, e.g. for each pixel)
- The vector ρ is the output image

In each case we need to learn a nonlinear function, that

I. maps $\mathbf{x}^{(k)} \sim p(\mathbf{x})$ to $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(k)})$

II. maps $\mathbf{z}^{(\ell)} \sim q_\phi(\mathbf{z}|\mathbf{x})$ to $\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z}^{(\ell)})$

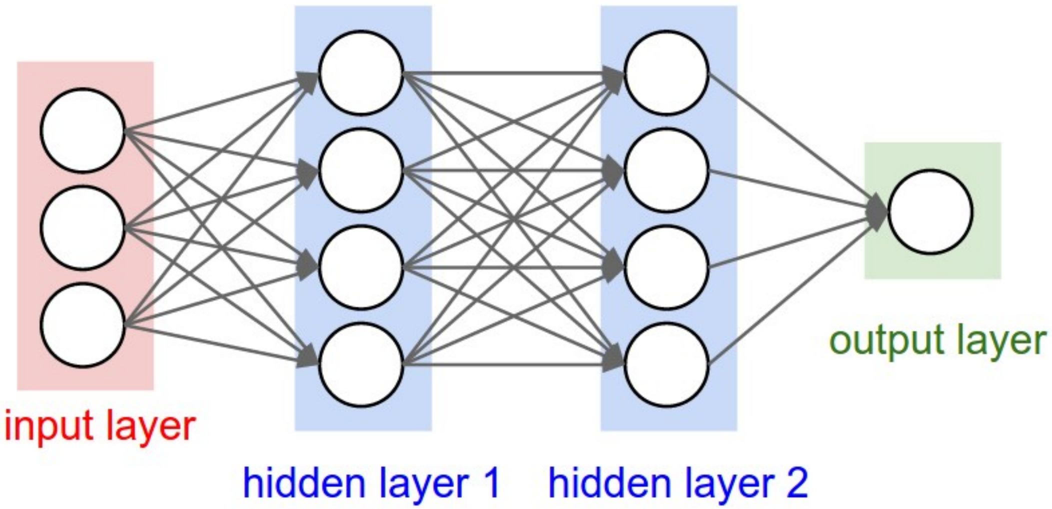
Quite miraculously this also maps $z \sim \mathcal{N}(z|0, I)$ to $x \sim p(x)$

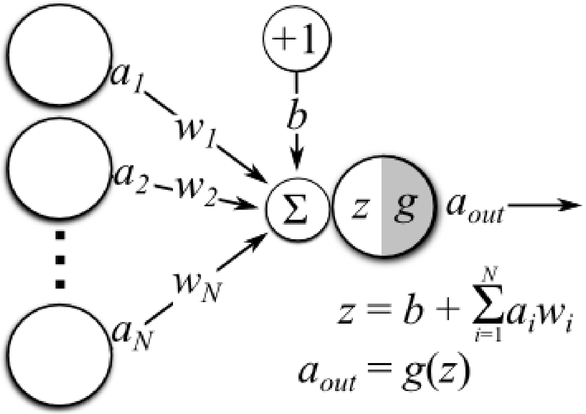
Idea V: Neural network

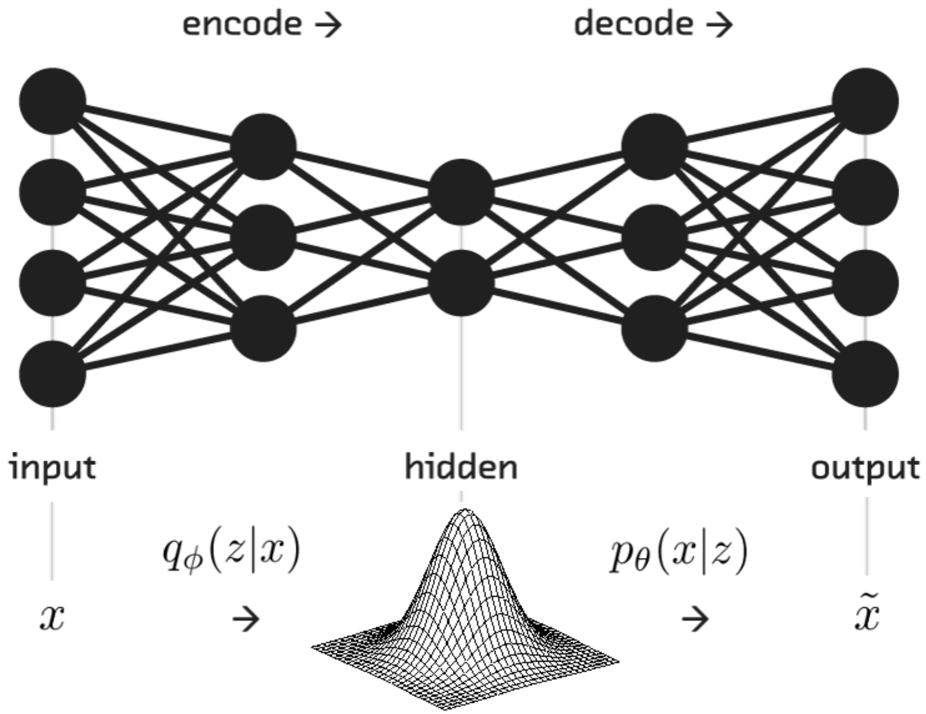
- Make use of the universal function approximator property of neural networks
- Use a neural network to calculate the function that

I. maps $x^{(k)} \sim p(x)$ to $z \sim q_\phi(z|x^{(k)})$

II. maps $z^{(\ell)} \sim q_\phi(z|x^{(k)})$ to $x \sim p_\theta(x|z^{(\ell)})$.







- Objective function

$$C(\phi) = \underbrace{-x \ln \rho_\phi(z^{(\ell)}) - (1-x)\ln(1-\rho_\phi(z^{(\ell)}))}_{\text{reconstruction loss}} + \underbrace{\frac{1}{2}\left[-1 - \ln\sigma^2_\phi(x) + \sigma^2_\phi(x) + \mu^2_\phi(x)\right]}_{\text{latent loss}}$$

  Take mean over all pixels, over each mini-batch

- Solve using gradient descent

Big Problem!!

Big Problem

- Sampling

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

  is no good.

- Another look at the objective function

$$C(\phi) = -x \ln \rho_\phi(z^{(\ell)}) - (1-x)\ln(1-\rho_\phi(z^{(\ell)})) + \frac{1}{2}\left[-1 - \ln \sigma^2_\phi(x) + \sigma^2_\phi(x) + \mu^2_\phi(x)\right]$$

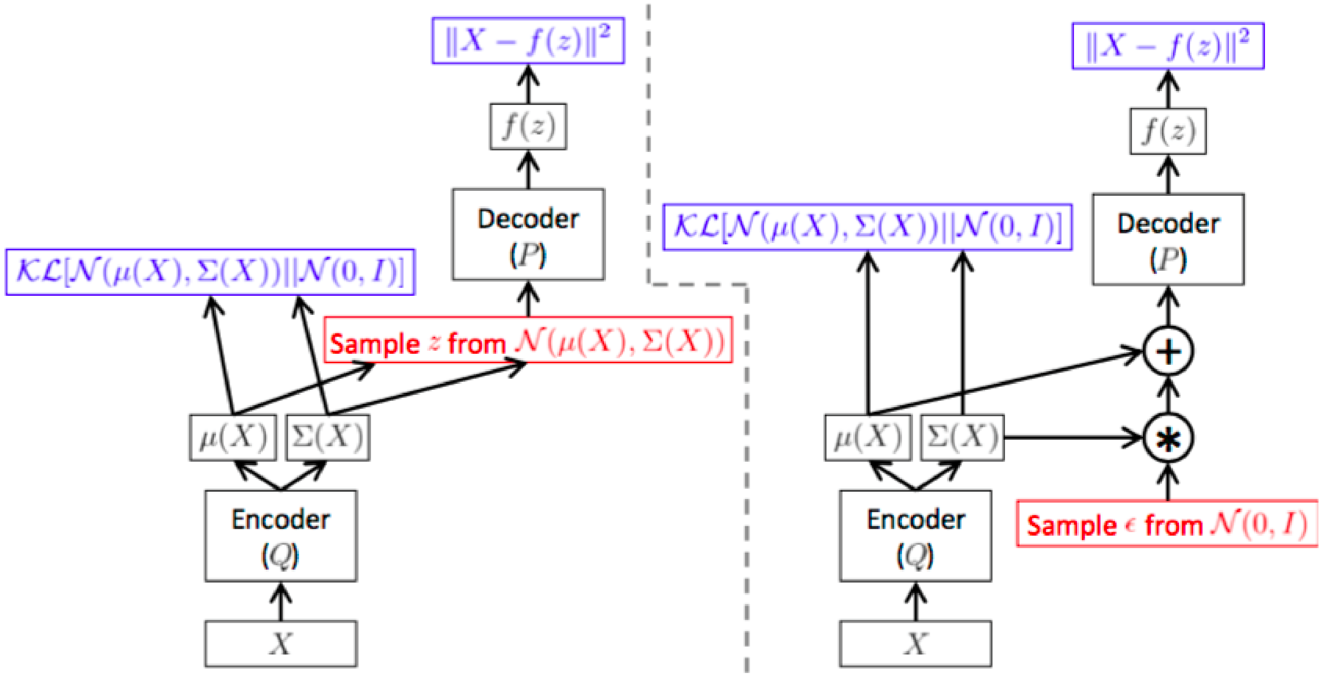
  The samples $z^{(\ell)}$ depend on the parameters that we need to optimize.

But they are hidden from the objective function!!
Cannot take a gradient through random samples

Idea VI: Reparameterization Trick

Sample $\epsilon^{(\ell)} \sim \mathcal{N}(\epsilon|0,1)$ and set

$$z^{(\ell)} = \mu_\phi(x) + \epsilon^{(\ell)}\sigma_\phi(x)$$



Tensorflow code: Autoencoder

```
n_inputs = 28 * 28
n_hidden1 = 500
n_hidden2 = 500
n_hidden3 = 20  # codings
n_hidden4 = n_hidden2
n_hidden5 = n_hidden1
n_outputs = n_inputs
learning_rate = 0.001

X = tf.placeholder(tf.float32, [None, n_inputs])
hidden1 = my_dense_layer(X, n_hidden1)
hidden2 = my_dense_layer(hidden1, n_hidden2)
hidden3_mean = my_dense_layer(hidden2, n_hidden3, activation=None)
hidden3_gamma = my_dense_layer(hidden2, n_hidden3, activation=None)
noise = tf.random_normal(tf.shape(hidden3_gamma), dtype=tf.float32)
hidden3 = hidden3_mean + tf.exp(0.5 * hidden3_gamma) * noise
hidden4 = my_dense_layer(hidden3, n_hidden4)
hidden5 = my_dense_layer(hidden4, n_hidden5)
logits = my_dense_layer(hidden5, n_outputs, activation=None)
outputs = tf.sigmoid(logits)
```

Tensorflow code: Cost function

```
xentropy = tf.nn.sigmoid_cross_entropy_with_logits(labels=X, logits=logits)
reconstruction_loss = tf.reduce_sum(xentropy)
latent_loss = 0.5 * tf.reduce_sum(
    tf.exp(hidden3_gamma) + tf.square(hidden3_mean) - 1 - hidden3_gamma)

loss = reconstruction_loss + latent_loss

optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)
```

Tensorflow code: Train and generate digits

```
n_digits = 60
n_epochs = 50
batch_size = 150

with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        n_batches = mnist.train.num_examples // batch_size
        for iteration in range(n_batches):
            X_batch, y_batch = mnist.train.next_batch(batch_size)
            sess.run(training_op, feed_dict={X: X_batch})
        loss_val, reconstruction_loss_val, latent_loss_val = sess.run([loss,
         reconstruction_loss, latent_loss], feed_dict={X: X_batch})


    codings_rnd = np.random.normal(size=[n_digits, n_hidden3])
    outputs_val = outputs.eval(feed_dict={hidden3: codings_rnd})
```

Example: generating digits