CSC413 Neural Networks and Deep Learning

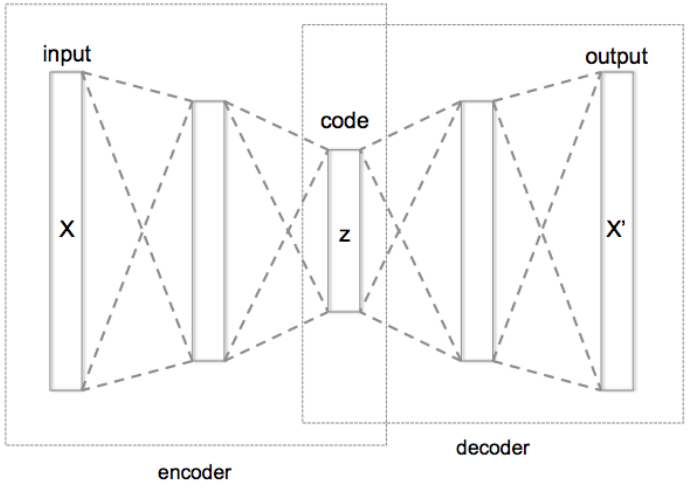
Lecture 10

March 22/25, 2021

Overview

Last week, we discussed **autoencoders**

- Encoder: maps x to a low-dimensional embedding z
- Decoder: uses the low-dimensional embedding z to reconstructs x



Let's see how much you remember!

Review Q1

What was the objective that we used to train the autoencoder?

Review Q2

If we train an autoencoder, what tasks can we accomplish with just the encoder portion of the autoencoder?

Review Q3

If we train an autoencoder, what tasks can we accomplish with mainly the decoder portion of the autoencoder?

Review Q4

What are some limitations of the autoencoder?

Autoencoder limitations

- Images are blurry (we'll talk about this next week)
- It's not certain what good values of embeddings z would be
  - Which part of the embedding space does the encoder maps data to?
  - This uncertainty means that we can't generate images without referring back to the encoder
- What should the dimension of the embeddings z be?

Autoencoder limitations

- Images are blurry (we'll talk about this next week)
- It's not certain what good values of embeddings z would be
  - Which part of the embedding space does the encoder maps data to?
  - This uncertainty means that we can't generate images without referring back to the encoder
- What should the dimension of the embeddings z be?

Is there a probabilistic version of the autoencoder model?

Could we resolve some of the issues with autoencoder, if we use a more theoretically grounded approach?

Variational Autoencoders

Mathematical Assumptions

Data $x_i \in \mathbb{R}^d$ are:

- independent, identically distributed (i.i.d)
- generated from the following joint distribution (with the true parameter θ^* unknown)

$$p_{\theta^*}(\mathbf{z}, \mathbf{x}) = p_{\theta^*}(\mathbf{z}) p_{\theta^*}(\mathbf{x}|\mathbf{z})$$

Where $\mathbf{z}$ is a low-dimensional vector (latent embedding)

- Example $\mathbf{x}$ could be an MNIST digit
- Think of $\mathbf{z}$ as encoding digit features like digit shape, tilt, line thickness, font style, etc. . .
- To generate an image, we first sample from the prior distribution $p_{\theta^*}(\mathbf{z})$ to decide on these digit features, and use $p_{\theta^*}(\mathbf{x}|\mathbf{z})$ to generate an image given those features

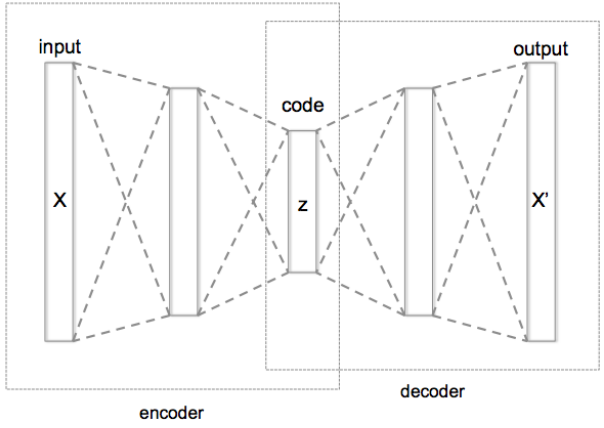
Intractability

Our data set is large, and so the following are *intractable*

- evidence $p_{\theta^*}(\mathbf{x})$
- posterior distributions $p_{\theta^*}(\mathbf{z}|\mathbf{x})$

In other words, exactly computing the distribution of $p(\mathbf{x})$ and $p(\mathbf{z}|\mathbf{x})$ using our dataset has high runtime complexity.

The decoder and encoder



With this assumption, we can think of the autoencoder as doing the following:

Decoder: A point approximation of the true distribution $p_{\theta^*}(\mathbf{x}|\mathbf{z})$

Encoder: Making a **point prediction** for the value of the latent vector z that generated the image x

Alternative:

- what if, instead, we try to infer the **distribution** $p_{\theta^*}(\mathbf{z}|\mathbf{x})$?

VAE Setup so far

Decoder: An approximation of the true distribution $p_{\theta^*}(\mathbf{x}|\mathbf{z})$

Encoder: An approximation of the true distribution $p_{\theta^*}(\mathbf{z}|\mathbf{x})$

Computing the encoding distribution $p_{\theta^*}(\mathbf{z}|\mathbf{x})$

Unfortunately, the true distribution $p_{\theta^*}(\mathbf{z}|\mathbf{x})$ is complex (e.g. can be multi-modal).

But can we approximate this distribution with a **simpler distribution**?

Let's restrict our estimate $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ to be a multivariate Gaussian distribution with $\phi = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$

- It suffices to estimate the mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ of $q_\phi(\mathbf{z}|\mathbf{x})$
- Let's make it simpler and assume that the covariance matrix is diagonal, $\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}_{d \times d}$

(Note: we don't have to make this assumption, but it will make computation easier later on)

VAE Setup so far

Decoder: An approximation of the true distribution $p_{\theta^*}(\mathbf{x}|\mathbf{z})$

Encoder: Predicts the mean and standard deviations of a distribution $q_\phi(\mathbf{z}|\mathbf{x})$, so that the distribution is close to the true distribution $p_{\theta^*}(\mathbf{z}|\mathbf{x})$

We want our estimate distribution to be close to the true distribution. How do we measure the difference between distributions?

Kullback-Leibler Divergence

Also called: KL Divergence, Relative Entropy

For discrete probability distributions:

$$KL[q(z) \mid\mid p(z)] = \sum_z q(z) \log \frac{q(z)}{p(z)}$$

For continuous probability distributions:

$$KL[q(z) \mid\mid p(z)] = \int q(z) \log \frac{q(z)}{p(z)} dz$$

KL Divergence Example Computation

Approximating an unfair coin with a fair coin.

- $p(z = 1) = 0.7$ and $p(z = 0) = 0.3$
- $q(z = 1) = q(z = 0) = 0.5$

KL Divergence Example Computation

Approximating an unfair coin with a fair coin.

- $p(z = 1) = 0.7$ and $p(z = 0) = 0.3$
- $q(z = 1) = q(z = 0) = 0.5$

$$
\begin{aligned}
KL[q(z) \,||\, p(z)] &= \sum_z q(z) \log \frac{q(z)}{p(z)} \\
&= q(0) \log \frac{q(0)}{p(0)} + q(1) \log \frac{q(1)}{p(1)} \\
&= 0.5 \log \frac{0.5}{0.3} + 0.5 \log \frac{0.5}{0.7} \\
&= 0.872
\end{aligned}
$$

KL Divergence is not symmetric!

Approximating an unfair coin with a fair coin.

- $p(z = 1) = 0.7$ and $p(z = 0) = 0.3$
- $q(z = 1) = q(z = 0) = 0.5$

KL Divergence is not symmetric!

Approximating an unfair coin with a fair coin.

- $p(z = 1) = 0.7$ and $p(z = 0) = 0.3$
- $q(z = 1) = q(z = 0) = 0.5$

$$
\begin{aligned}
KL[p(z) \,||\, q(z)] &= \sum_z p(z) \log \frac{p(z)}{q(z)} \\
&= p(0) \log \frac{p(0)}{q(0)} + p(1) \log \frac{p(1)}{q(1)} \\
&= 0.3 \log \frac{0.3}{0.5} + 0.7 \log \frac{0.7}{0.5} \\
&= 0.823 \\
&\neq KL[q(z) \,||\, p(z)]
\end{aligned}
$$

KL divergence Properties

The KL divergence is a measure of the difference between probability distributions.

KL divergence is an asymmetric, nonnegative measure, not a norm. It doesn't obey the triangle inequality.

KL divergence is always positive. Hint: you can show this using the inequality $ln(x) \leq x - 1$ for $x > 0$

KL Divergence: continuous example

Suppose we have two Gaussian distributions $p(x) \sim N(\mu_1, \sigma_1^2)$ and $q(x) \sim N(\mu_2, \sigma_2^2)$.

What is the KL divergence $KL[p(z) \,||\, q(z)]$?

Recall:

$$p(z; \mu_1, \sigma_1^2) = \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(z-\mu_1)^2}{2\sigma_1^2}}$$

$$\log p(z; \mu_1, \sigma_1^2) = -\log \sqrt{2\pi\sigma_1^2} - \frac{(z-\mu_1)^2}{2\sigma_1^2}$$

KL Divergence: Entropy and Cross-Entropy

We can split the KL divergence into two terms, which we can compute separately:

$$
\begin{aligned}
KL[p(z) \,||\, q(z)] &= \int p(z) \log \frac{p(z)}{q(z)} dz \\
&= \int p(z)(\log p(z) - \log q(z)) dz \\
&= \int p(z) \log p(z) dz - \int p(z) \log q(z) dz \\
&= -\text{entropy} - \text{cross-entropy}
\end{aligned}
$$

KL Divergence: continuous example, entropy computation

$$\begin{aligned}
\int p(z) \log p(z) dz &= \int p(z)(-\log \sqrt{2\pi\sigma_1^2} - \frac{(z-\mu_1)^2}{2\sigma_1^2}) dz \\
&= -\int p(z) \frac{1}{2} \log(2\pi\sigma_1^2) dz - \int p(z) \frac{(z-\mu_1)^2}{2\sigma_1^2} dz \\
&= -\frac{1}{2} \log(2\pi\sigma_1^2) \int p(z) dz - \frac{1}{2\sigma_1^2} \int p(z)(z-\mu_1)^2 dz \\
&= -\frac{1}{2} \log(2\pi\sigma_1^2) - \frac{1}{2} \\
&= -\frac{1}{2} \log(\sigma_1^2) - \frac{1}{2} \log(2\pi) - \frac{1}{2}
\end{aligned}$$

Since $\int p(z) dz = 1$ and $\int p(z)(z-\mu_1)^2 dz = \sigma_1^2$

KL Divergence: continuous example, cross-entropy computation

$$
\begin{aligned}
\int p(z) \log q(z) dz &= \int p(z)(-\log \sqrt{2\pi\sigma_2^2} - \frac{(z-\mu_2)^2}{2\sigma_2^2}) dz \\
&= -\int p(z) \frac{1}{2} \log(2\pi\sigma_2^2) dz - \int p(z) \frac{(z-\mu_2)^2}{2\sigma_2^2} dz \\
&= -\frac{1}{2} \log(2\pi\sigma_2^2) - \frac{1}{2\sigma_2^2} \int p(z)(z-\mu_2)^2 dz \\
&= \ldots \\
&= -\frac{1}{2} \log(2\pi\sigma_2^2) - \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2}
\end{aligned}
$$

Back to autoencoders: summary so far

Autoencoder:

- Decoder: point estimate of $p_{\theta^*}(\mathbf{x}|\mathbf{z})$
- Encoder: point estimate of the value of $\mathbf{z}$ that generated the image $\mathbf{x}$

VAE:

- Decoder: probabilistic estimate of $p_{\theta^*}(\mathbf{x}|\mathbf{z})$
- Encoder: probabilistic estimate of a Gaussian distribution $q_\phi(\mathbf{z}|\mathbf{x})$ that approximates the distribution $p_{\theta^*}(\mathbf{z}|\mathbf{x})$
  - In particular, our encoder will be a neural network that predicts the mean and standard deviation of $q_\phi(\mathbf{z}|\mathbf{x})$
  - We can then sample $\mathbf{z}$ from this distribution!

VAE Objective

But how do we train a VAE?

We want to maximize the likelihood of our data:

$\log p(x) = \log \int p(x|z)p(x)dz$

And we want to make sure that the distributions $q(z|x)$ and $p(z|x)$ are close:

- We want to minimize $KL[q(\mathbf{z}|\mathbf{x}) \,||\, p(\mathbf{z}|\mathbf{x})]$
- This is a measure of encoder quality

In other words, we want to maximize

$$-KL[q(\mathbf{z}|\mathbf{x}) \,||\, p(\mathbf{z}|\mathbf{x})] + \log p(x)$$

How can we optimize this quantity in a tractable way?

VAE: Evidence Lower-Bound

$$
\begin{aligned}
KL[q(\mathbf{z}|\mathbf{x}) \,||\, p(\mathbf{z}|\mathbf{x})] &= \int q(\mathbf{z}|\mathbf{x}) \log \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} dz \\
&= \mathbb{E}_q[\log \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})}] \\
&= \mathbb{E}_q[\log q(\mathbf{z}|\mathbf{x})] - \mathbb{E}_q[\log p(\mathbf{z}|\mathbf{x})] \\
&= \mathbb{E}_q[\log q(\mathbf{z}|\mathbf{x})] - \mathbb{E}_q[\log p(\mathbf{z}, \mathbf{x})] + \mathbb{E}_q[\log p(\mathbf{x})] \\
&= \mathbb{E}_q[\log q(\mathbf{z}|\mathbf{x})] - \mathbb{E}_q[\log p(\mathbf{z}, \mathbf{x})] + \log p(\mathbf{x})
\end{aligned}
$$

We'll define the **evidence lower-bound**:

$$
\text{ELBO}_q(\mathbf{x}) = \mathbb{E}_q[\log p(\mathbf{z}, \mathbf{x}) - \log q(\mathbf{z}|\mathbf{x})]
$$

So we have

$$
\log p(\mathbf{x}) - KL[q(\mathbf{z}|\mathbf{x}) \,||\, p(\mathbf{z}|\mathbf{x})] = \text{ELBO}_q(\mathbf{x})
$$

Optimizing the ELBO

The ELBO gives us a way to estimate the gradients of $\log p(\mathbf{x}) - KL[q(\mathbf{z}|\mathbf{x}) \,||\, p(\mathbf{z}|\mathbf{x})]$

How?

$$\text{ELBO}_q(\mathbf{x}) = \mathbb{E}_q[\log p(\mathbf{z}, \mathbf{x}) - \log q(\mathbf{z}|\mathbf{x})]$$

- The right hand side of this expression is an expectation over $z \sim q(z|x)$
- To estimate the ELBO, we can **sample** from the distribution $z \sim q(z|x)$, and compute the terms inside.
- We can estimate gradients in the same way—this is called a **Monte-Carlo gradient estimator**

Monte Carlo Estimation

(This notation is unrelated to other slides: $p(z)$ is just a univariate Gaussian distribution, and $f_\phi(z)$ is a function parameterized by ϕ)

Suppose we want to optimize an objective $\mathcal{L}(\phi) = \mathbb{E}_{z\sim p(z)}[f_\phi(z)]$ where $p(z)$ is a normal distribution.

We can **estimate** $\mathcal{L}(\phi)$ by sampling $z_i \sim p(z)$ and computing

$$\begin{aligned}\mathcal{L}(\phi) &= \mathbb{E}_{z\sim p(z)}[f_\phi(z)] \\ &= \int_z p(z) f_\phi(z) dz \\ &\approx \frac{1}{N}\sum_{i=1}^{N} f_\phi(z_i)\end{aligned}$$

Monte Carlo Gradient Estimation

Likewise, if we want to estimate $\nabla_\phi \mathcal{L}$, we can sample $z_i \sim p(z)$ and compute

$$\begin{aligned}
\nabla_\phi \mathcal{L} &= \nabla_\phi \mathbb{E}_{z \sim p(z)}[f_\phi(z)] \\
&= \nabla_\phi \int_z p(z) f_\phi(z) dz \\
&\approx \nabla_\phi \frac{1}{N} \sum_{i=1}^{N} f_\phi(z_i) \\
&= \frac{1}{N} \sum_{i=1}^{N} \nabla_\phi f_\phi(z_i)
\end{aligned}$$

The reparamaterization trick

$\text{ELBO}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{z}, \mathbf{x}) - \log q_\phi(\mathbf{z}|\mathbf{x})]$

Problem: typical Monte-Carlo gradient estimator with samples $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ has very high variance

Reparameterization trick: instead of sampling $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ express $\mathbf{z} = g_\phi(\epsilon, \mathbf{x})$ where g is deterministic and only ϵ is stochastic.

In practise, the reparameterization trick is what makes the VAE encoder deterministic. When running a VAE forward pass:

1. We get the means and standard deviations from the VAE
2. We sample from $\mathcal{N}(\mathbf{0}, \mathbf{I})$
3. We use the samples from step 2 to get a sample from $q(z)$ obtained from step 1

VAE: Summary so far

Decoder: estimate of $p_{\theta^*}(\mathbf{x}|\mathbf{z})$

Encoder: estimate of a Gaussian distribution $q_\phi(\mathbf{z}|\mathbf{x})$ that approximates the distribution $p_{\theta^*}(\mathbf{z}|\mathbf{x})$

- Encoder is a NN that predicts the mean and standard deviation of $q_\phi(\mathbf{z}|\mathbf{x})$
- Use the **reparameterization trick** to sample from this distribution

The VAE objective is equal to the evidence lower-bound:

$$\log p(\mathbf{x}) - KL[q(\mathbf{z}|\mathbf{x}] \,||\, p(\mathbf{z}|\mathbf{x})) = \text{ELBO}_q(\mathbf{x})$$

Which we can estimate using Monte Carlo

$$\text{ELBO}_q(\mathbf{x}) = \mathbb{E}_q[\log p(\mathbf{z}, \mathbf{x}) - \log q(\mathbf{z}|\mathbf{x})]$$

VAE: Summary so far

But given a value $z \sim q(z|x)$, how can we compute

$$\log p(\mathbf{z}, \mathbf{x}) - \log q(\mathbf{z}|\mathbf{x})$$

. . . or its derivative with respect to the neural network parameters?

We need to do some more math to write this quantity in a form that is easier to estimate.

VAE: a simpler form

$$\mathsf{ELBO}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{z},\mathbf{x}) - \log q_\phi(\mathbf{z}|\mathbf{x})]$$

VAE: a simpler form

$$\begin{aligned} \mathsf{ELBO}_{\theta,\phi}(\mathbf{x}) &= \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{z},\mathbf{x}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_\theta(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \end{aligned}$$

VAE: a simpler form

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

VAE: a simpler form

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

VAE: a simpler form

$$
\begin{aligned}
\text{ELBO}_{\theta,\phi}(\mathbf{x}) &= \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{z},\mathbf{x}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\
&= \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_\theta(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\
&= \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{z}) + \log q_\phi(\mathbf{z}|\mathbf{x})] \\
&= \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \text{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z})) \\
&= \text{decoding quality} - \text{encoding regularization}
\end{aligned}
$$

Both terms can be computed easily if we make some simplifying assumptions

Let's see how...

Computing Decoding Quality

In order to estimate this quantity

$$\mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}|\mathbf{z})]$$

. . . we need to make some assumptions about the distribution $p_\theta(\mathbf{x}|\mathbf{z})$.

If we make the assumption that $p_\theta(\mathbf{x}|\mathbf{z})$ is a normal distribution centered around some pixel intensity, then optimizing $p_\theta(\mathbf{x}|\mathbf{z})$ is equivalent to optimizing the *square loss*!

That is, $p_\theta(\mathbf{x}|\mathbf{z})$ tells us how intense a pixel could be, but that pixel could be a bit darker/lighter, following a normal distribution).

Bonus: A traditional autoencoder is optimizing this same quantity!

Computing Encoding Quality

This KL divergence computes the difference in distribution between two distributions:

$$\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \;||\; p_\theta(\mathbf{z}))$$

- $q_\phi(\mathbf{z}|\mathbf{x})$ is a normal distribution that approximates $p_\theta(\mathbf{z}|\mathbf{x})$
- $p_\theta(\mathbf{z})$ is the **prior distribution on z**
  - distribution of z when we don't know anything about $\mathbf{x}$ or any other quantity

Since $\mathbf{z}$ is a *latent* variable, not actually observed in the real word, we can choose $p_\theta(\mathbf{z})$

- we choose $p_\theta(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$

. . . and we know how to compute the KL divergence of two Gaussian distributions!

Interpretation

The VAE objective

$$\mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}))$$

has an extra regularization term that the traditional autoencoder does not.

This extra regularization term pushes the values of $\mathbf{z}$ to be closer to 0!

MNIST results

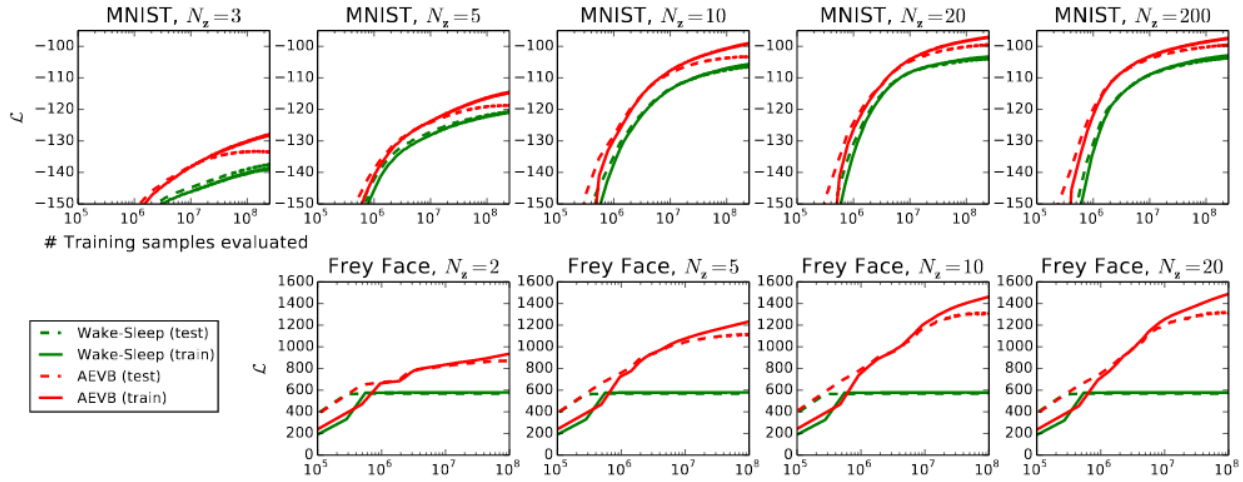
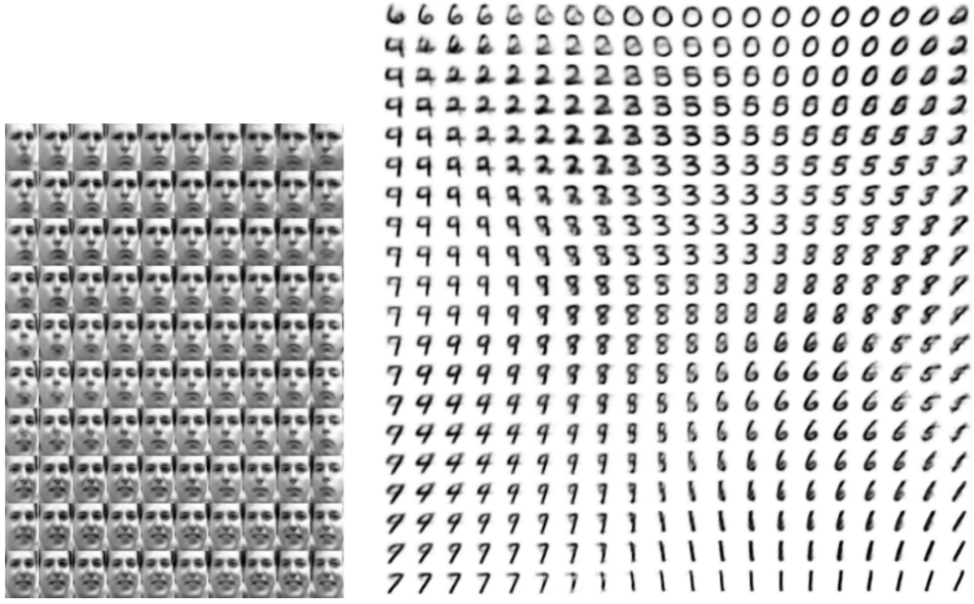


Figure 2: Comparison of our AEVB method to the wake-sleep algorithm, in terms of optimizing the lower bound, for different dimensionality of latent space ($N_{\mathbf{z}}$). Our method converged considerably faster and reached a better solution in all experiments. Interestingly enough, more latent variables does not result in more overfitting, which is explained by the regularizing effect of the lower bound. Vertical axis: the estimated average variational lower bound per datapoint. The estimator variance was small (< 1) and omitted. Horizontal axis: amount of training points evaluated. Computation took around 20-40 minutes per million training samples with a Intel Xeon CPU running at an effective 40 GFLOPS.

Figure 1: image

Frey Faces results



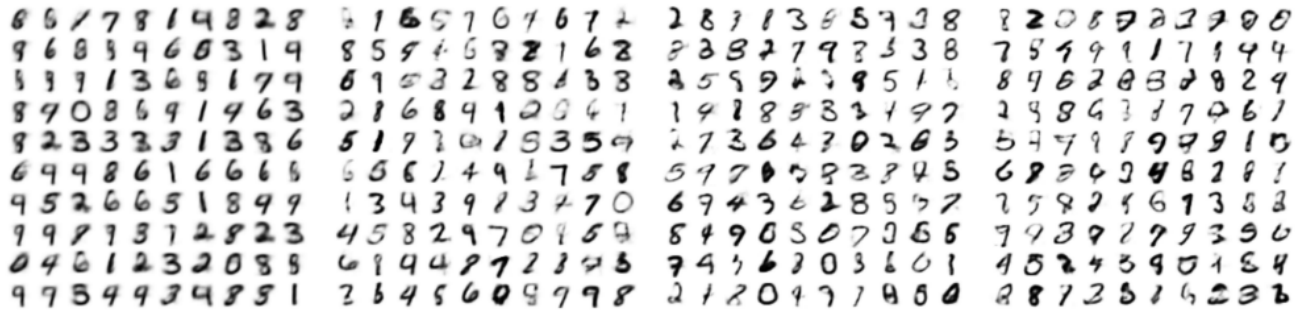
(a) Learned Frey Face manifold

(b) Learned MNIST manifold

Figure 4: Visualisations of learned data manifold for generative models with two-dimensional latent space, learned with AEVB. Since the prior of the latent space is Gaussian, linearly spaced coordinates on the unit square were transformed through the inverse CDF of the Gaussian to produce values of the latent variables $\mathbf{z}$. For each of these values $\mathbf{z}$, we plotted the corresponding generative $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ with the learned parameters $\boldsymbol{\theta}$.

Figure 2: image

Dimension of latent variables



(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Figure 5: Random samples from learned generative models of MNIST for different dimensionalities of latent space.

Figure 3: image