Variational autoencoders (and other deep generative models)

Brooks Paige

COMP0171 Week 8

Generative models and deep generative models

The deep learning models we've talked about so far in this module have been *discriminative* models.

- **Data**: pairs of instances and labels $(\mathbf{x}_i, y_i)$, for $i = 1, \ldots, N$.
- **Goal**: estimate $f_\theta(\mathbf{x}_i) \approx y_i$, or maximize $p_\theta(y_i|\mathbf{x}_i)$.

The learning task is to characterize the conditional distribution of **labels** y_i given **data** $\mathbf{x}_i$. We typically do this by estimating a parameter θ to minimize prediction error on the training set, or (equivalently) to maximize the training log-likelihood.

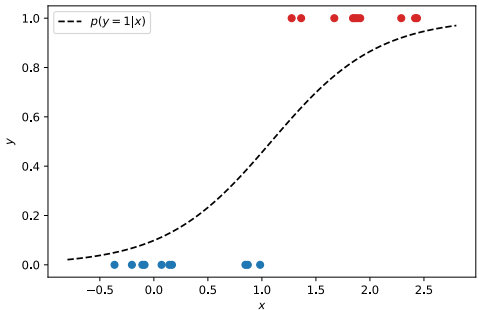
Generative models and deep generative models

In contrast, generative models also estimate the distribution of the data $\mathbf{x}_i$. This means we estimate a model for $p(\mathbf{x}_i)$, or the joint distribution $p(\mathbf{x}_i, y_i)$.

- **Data**: pairs of instances and labels $(\mathbf{x}_i, y_i)$, for $i = 1, \ldots, N$.
- **Generative model**: learn an approximation to $p_\theta(\mathbf{x}_i)$ or $p_\theta(\mathbf{x}_i, y_i)$.
- **Discriminative model**: learn an approximation to $p_\theta(y_i|\mathbf{x}_i)$.

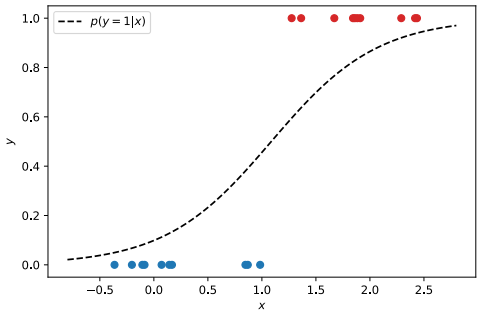
A tale of two binary linear classifiers

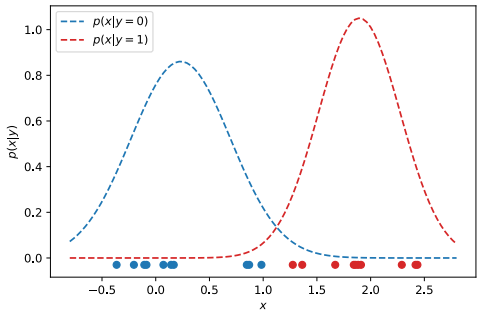
Logistic regression: learn $p(y|\mathbf{x})$
Gaussian naïve Bayes: learn $p(y)$, $p(\mathbf{x}|y=1)$, and $p(\mathbf{x}|y=0)$



A tale of two binary linear classifiers

Logistic regression: learn $p(y|\mathbf{x})$
Gaussian naïve Bayes: learn $p(y)$, $p(\mathbf{x}|y=1)$, and $p(\mathbf{x}|y=0)$

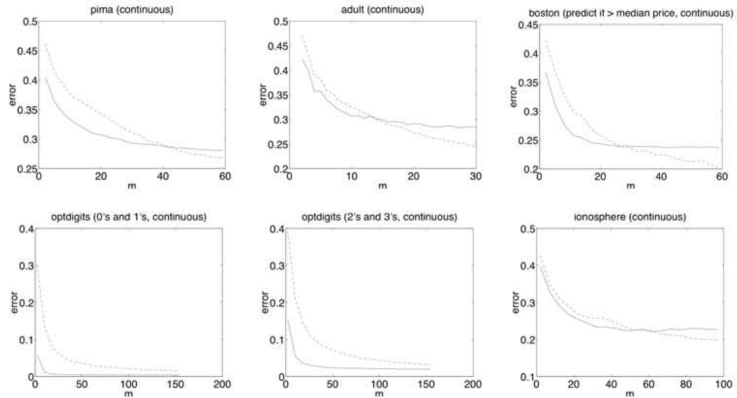




A tale of two binary linear classifiers

Logistic regression: learn $p(y|\mathbf{x})$

Gaussian naïve Bayes: learn $p(y)$, $p(\mathbf{x}|y=1)$, and $p(\mathbf{x}|y=0)$



Two distinct regimes! (**Small data** vs **Large data limit**...)

Ng & Jordan (2002)

Learning linear latent variable models

There's a long history of learning **linear** latent variable models directly from data, including

- Principal components analysis (PCA)
- Independent components analysis (ICA)
- Factor analysis
- Non-negative matrix factorization
- ...

Typically, these correspond to learning a low-rank factorization $\mathbf{X} \approx \mathbf{AB}$, where each instance $\mathbf{x}_i$ is a weighted sum of dictionary elements.

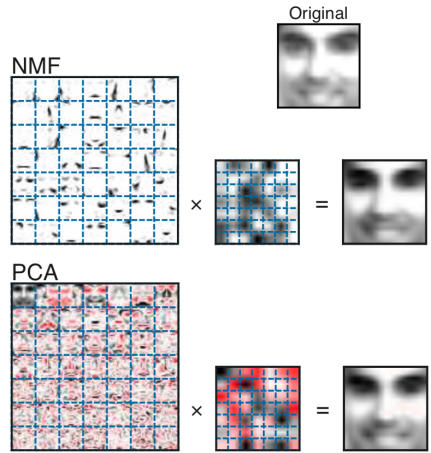


Figure: Hastie et al.

PCA as reconstruction error minimization

Suppose we have N observations $\mathbf{x}_i \in \mathbb{R}^D$ (typically assumed to be "centered", i.e. with $\frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i = \mathbf{0}$). in a design matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$.

The goal is to approximate $\mathbf{X}$ as $\hat{\mathbf{X}} \approx \mathbf{Z}\mathbf{W}^\top$, where for $K \ll D$,

$$\underbrace{\mathbf{Z} \in \mathbb{R}^{N \times K}}_{\text{"tall, skinny" matrix}} \qquad \underbrace{\mathbf{W}^\top \in \mathbb{R}^{K \times D}}_{\text{"short, wide" matrix}}.$$

PCA as reconstruction error minimization

Suppose we have N observations $\mathbf{x}_i \in \mathbb{R}^D$ (typically assumed to be "centered", i.e. with $\frac{1}{N}\sum_{i=1}^N \mathbf{x}_i = \mathbf{0}$). in a design matrix $\mathbf{X} \in \mathbb{R}^{N\times D}$.

The goal is to approximate $\mathbf{X}$ as $\hat{\mathbf{X}} \approx \mathbf{Z}\mathbf{W}^\top$, where for $K \ll D$,

$$\underbrace{\mathbf{Z} \in \mathbb{R}^{N\times K}}_{\text{"tall, skinny" matrix}} \qquad \underbrace{\mathbf{W}^\top \in \mathbb{R}^{K\times D}}_{\text{"short, wide" matrix}}.$$

The standard approach includes the additional constraint that $\mathbf{W}^\top\mathbf{W} = \mathbf{I}$, and then solves the least squares problem

$$\arg\min_{\mathbf{W},\mathbf{Z}} \|\mathbf{X} - \hat{\mathbf{X}}\|_2^2 = \sum_{i=1}^N \|\mathbf{x}_i - \mathbf{W}\mathbf{z}_i\|_2^2$$

For each $\mathbf{x}_i$ we have a low-dimensional representation $\mathbf{z}_i \in \mathbb{R}^K$ with $\mathbf{x}_i \approx \mathbf{W}\mathbf{z}_i$.

Aside: finding a PCA solution

You can find the solution to this minimization problem by **alternate minimization** of $\sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{W}\mathbf{z}_i\|_2^2$:

Aside: finding a PCA solution

You can find the solution to this minimization problem by **alternate minimization** of $\sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{W}\mathbf{z}_i\|_2^2$:

1. First, for a fixed $\mathbf{W}$, compute

$$\frac{\partial}{\partial \mathbf{z}_i} \|\mathbf{x}_i - \mathbf{W}\mathbf{z}_i\|_2^2 = 2\mathbf{z}_i - 2\mathbf{W}^\top \mathbf{x}_i.$$

   Set equal to zero and solve, for $\mathbf{z}_i = \mathbf{W}^\top \mathbf{x}_i$

Aside: finding a PCA solution

You can find the solution to this minimization problem by **alternate minimization** of $\sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{W}\mathbf{z}_i\|_2^2$:

1. First, for a fixed $\mathbf{W}$, compute

$$\frac{\partial}{\partial \mathbf{z}_i}\|\mathbf{x}_i - \mathbf{W}\mathbf{z}_i\|_2^2 = 2\mathbf{z}_i - 2\mathbf{W}^\top \mathbf{x}_i.$$

   Set equal to zero and solve, for $\mathbf{z}_i = \mathbf{W}^\top \mathbf{x}_i$

2. Now, for fixed $\mathbf{Z}$, minimizing $\|\mathbf{X} - \mathbf{Z}\mathbf{W}^\top\|_2^2$ is similar to the linear regression least squares problem, with

$$\mathbf{W} = \mathbf{X}^\top \mathbf{Z}(\mathbf{Z}^\top \mathbf{Z})^{-1}.$$

Aside: finding a PCA solution

You can find the solution to this minimization problem by **alternate minimization** of $\sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{W}\mathbf{z}_i\|_2^2$:

1. First, for a fixed $\mathbf{W}$, compute

$$\frac{\partial}{\partial \mathbf{z}_i}\|\mathbf{x}_i - \mathbf{W}\mathbf{z}_i\|_2^2 = 2\mathbf{z}_i - 2\mathbf{W}^\top \mathbf{x}_i.$$

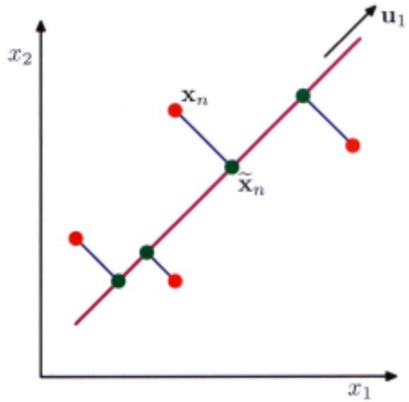
   Set equal to zero and solve, for $\mathbf{z}_i = \mathbf{W}^\top \mathbf{x}_i$

2. Now, for fixed $\mathbf{Z}$, minimizing $\|\mathbf{X} - \mathbf{Z}\mathbf{W}^\top\|_2^2$ is similar to the linear regression least squares problem, with

$$\mathbf{W} = \mathbf{X}^\top \mathbf{Z}(\mathbf{Z}^\top \mathbf{Z})^{-1}.$$

In practice it is rare to use this algorithm: instead, principal components can be found efficiently using a **singular value decomposition** (SVD) of the matrix $\mathbf{X}$.

PCA: basic example



Probabilistic view?

Probabilistic PCA

PCA also can be formulated as a probabilistic generative model:

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i|\mathbf{0}, \mathbf{I}) \qquad p(\mathbf{x}_i|\mathbf{z}_i) = \mathcal{N}(\mathbf{x}_i|\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2\mathbf{I}),$$

for $i = 1, \ldots, N$, $\mathbf{z}_i \in \mathbb{R}^K$ and $\mathbf{x}_i \in \mathbb{R}^D$, with $K < D$.

Roweis (1998), Tipping & Bishop (1999)

Probabilistic PCA

PCA also can be formulated as a probabilistic generative model:

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i|\mathbf{0}, \mathbf{I}) \qquad p(\mathbf{x}_i|\mathbf{z}_i) = \mathcal{N}(\mathbf{x}_i|\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2\mathbf{I}),$$

for $i = 1, \ldots, N$, $\mathbf{z}_i \in \mathbb{R}^K$ and $\mathbf{x}_i \in \mathbb{R}^D$, with $K < D$. This model has a tractable posterior distribution

$$p(\mathbf{z}_i|\mathbf{x}_i) = \mathcal{N}(\mathbf{z}_i|\mathbf{M}^{-1}\mathbf{W}^\top(\mathbf{x} - \boldsymbol{\mu}), \sigma^2\mathbf{M}^{-1})$$

where $\mathbf{M} = \mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I}$.

The marginal likelihood $p(\mathbf{x}_i) = \int p(\mathbf{x}_i|\mathbf{z}_i)p(\mathbf{z}_i)\mathrm{d}\mathbf{z}_i$ is also tractable (and Gaussian).

Roweis (1998), Tipping & Bishop (1999)

Probabilistic PCA

PCA also can be formulated as a probabilistic generative model:

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i|\mathbf{0}, \mathbf{I}) \qquad p(\mathbf{x}_i|\mathbf{z}_i) = \mathcal{N}(\mathbf{x}_i|\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2\mathbf{I}),$$

for $i = 1, \ldots, N$, $\mathbf{z}_i \in \mathbb{R}^K$ and $\mathbf{x}_i \in \mathbb{R}^D$, with $K < D$. This model has a tractable posterior distribution

$$p(\mathbf{z}_i|\mathbf{x}_i) = \mathcal{N}(\mathbf{z}_i|\mathbf{M}^{-1}\mathbf{W}^\top(\mathbf{x} - \boldsymbol{\mu}), \sigma^2\mathbf{M}^{-1})$$

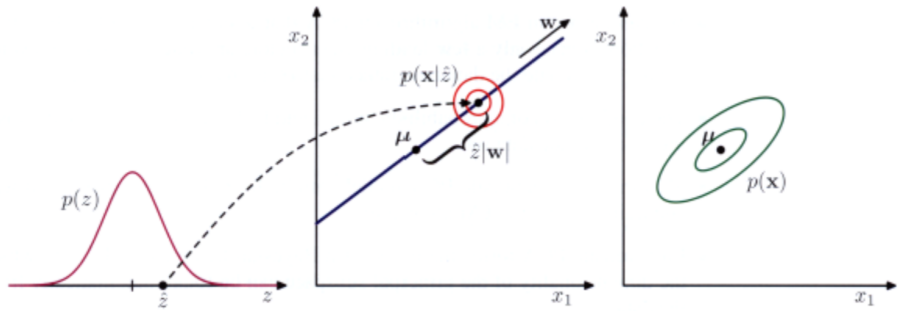
where $\mathbf{M} = \mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I}$.

The marginal likelihood $p(\mathbf{x}_i) = \int p(\mathbf{x}_i|\mathbf{z}_i)p(\mathbf{z}_i)\mathrm{d}\mathbf{z}_i$ is also tractable (and Gaussian).

Question: What happens to the posterior mean if $\mathbf{W}^\top\mathbf{W} = \mathbf{I}$ and we take $\sigma^2 \to 0$?

Roweis (1998), Tipping & Bishop (1999)

Probabilistic PCA: Generative viewpoint



This is a $K = 1$ dimensional latent space for $D = 2$ dimensional data. Data is generated by sampling a value of $\mathbf{z}$, projecting it up into 2d space with a $\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu}$, and adding noise σ^2. The rightmost plot shows the marginal density on $\mathbf{x}$.

Figure: Bishop (2006)

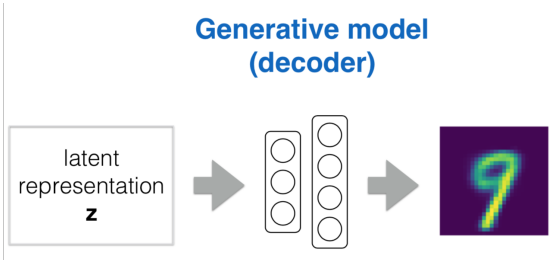
Auto-encoding variational Bayes

Variational Auto-Encoders

Now let's consider a deep generative model which has a Gaussian prior, and a likelihood defined by a deep net, e.g.

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i|\mathbf{0}, \mathbf{I}) \qquad p(\mathbf{x}_i|\mathbf{z}_i) = \mathcal{N}(\mathbf{x}_i|f_\theta(\mathbf{z}_i), \sigma^2\mathbf{I}),$$

for $i = 1, \ldots, N$. As before, assume $\mathbf{z}_i \in \mathbb{R}^K$ and $\mathbf{x}_i \in \mathbb{R}^D$, with $K < D$.



Computing the marginal likelihood

Challenge: In this model, the marginal likelihood $p(\mathbf{X})$ is completely intractable.

$$\log p(\mathbf{X}) = \sum_{i=1}^{N} \log p(\mathbf{x}_i) = \sum_{i=1}^{N} \log \int p(\mathbf{x}_i|\mathbf{z}_i, \theta) p(\mathbf{z}_i) d\mathbf{z}_i$$

In the linear model, we had closed forms for both the marginal likelihood terms $p(\mathbf{x}_i)$ and the posterior distribution $p(\mathbf{z}_i|\mathbf{x}_i)$.

- If we have a nonlinear f_θ in $p(\mathbf{x}_i|\mathbf{z}_i) = \mathcal{N}(\mathbf{x}_i|f_\theta(\mathbf{z}_i), \sigma^2\mathbf{I})$, neither of these are the case.

Learning an approximate posterior

We're going to approach this similarly to how we have used **variational Bayes** elsewhere in the module.

Main difference: instead of learning a single posterior approximating $p(\mathbf{Z}|\mathbf{X})$, we will do what is sometimes called **amortized inference**, by learning an **inference network**.

- For any input data $\mathbf{x}_i$, we will learn a "one data-point posterior" $q_\phi(\mathbf{z}_i|\mathbf{x}_i)$
- Inference entails finding a value of ϕ that makes $q_\phi(\mathbf{z}|\mathbf{x})$ "close" to the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$

This is analogous to finding the projection matrix from $\mathbf{X}$ to $\mathbf{Z}$ in the linear case, where the **same** projection happened for each i:

$$\mathbf{z}_i = \mathbf{W}^\top \mathbf{x}_i \qquad\qquad \mathbf{Z} = \mathbf{X}\mathbf{W}$$

Amortized inference networks

What is a good choice of $q_\phi(\mathbf{z}|\mathbf{x})$? The typical choice is to use a Gaussian distribution whose mean and variance are output by **another** deep network, with

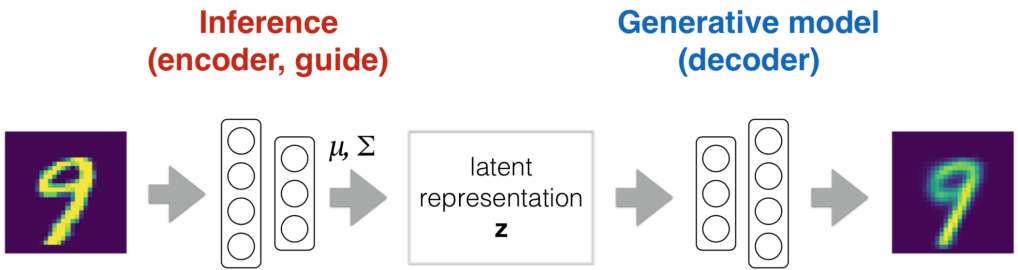
$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mu(\mathbf{x}), \mathrm{diag}(\sigma(\mathbf{x})^2))$$

where

- $\mu : \mathbb{R}^D \to \mathbb{R}^K$ is a deep net which outputs a posterior mean, and
- $\sigma : \mathbb{R}^D \to \mathbb{R}^K$ is a deep net which outputs a posterior standard deviation.

Note that in practice the network for σ typically will output the log of the standard deviation, which is then exponentiated, in order to enforce a positivity constraint.

High-level diagram: autoencoder interpretation



The Kullback-Leibler divergence

To fit the **inference network**, we minimize the KL divergence

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z}.$$

This differs in two ways from the KL divergence we minimized when doing inference over our model parameters (e.g. in the first coursework or in the models from last week):

- we are not doing inference over the network parameters θ, but rather over the latent "representation" $\mathbf{z}$
- the posterior approximation takes data $\mathbf{x}$ as an input

The per-datapoint ELBO

As before, we will re-arrange this KL divergence (isolating the marginal likelihood term, to find a tractable objective):

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z}$$

The per-datapoint ELBO

As before, we will re-arrange this KL divergence (isolating the marginal likelihood term, to find a tractable objective):

$$
\begin{aligned}
D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{x}) q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}, \mathbf{z})} d\mathbf{z}
\end{aligned}
$$

The per-datapoint ELBO

As before, we will re-arrange this KL divergence (isolating the marginal likelihood term, to find a tractable objective):

$$
\begin{aligned}
D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{x}) q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}, \mathbf{z})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}, \mathbf{z})} d\mathbf{z} + \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}) d\mathbf{z}
\end{aligned}
$$

The per-datapoint ELBO

As before, we will re-arrange this KL divergence (isolating the marginal likelihood term, to find a tractable objective):

$$
\begin{aligned}
D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{x}) q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}, \mathbf{z})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}, \mathbf{z})} d\mathbf{z} + \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}) d\mathbf{z} \\
&= \underbrace{-\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]}_{\text{ELBO, } \mathcal{L}(\mathbf{x};\phi,\theta)} + \log p_\theta(\mathbf{x}).
\end{aligned}
$$

The per-datapoint ELBO

As before, we will re-arrange this KL divergence (isolating the marginal likelihood term, to find a tractable objective):

$$
\begin{aligned}
D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{x}) q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}, \mathbf{z})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}, \mathbf{z})} d\mathbf{z} + \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}) d\mathbf{z} \\
&= \underbrace{-\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]}_{\text{ELBO, } \mathcal{L}(\mathbf{x};\phi,\theta)} + \log p_\theta(\mathbf{x}).
\end{aligned}
$$

Again, this is "tractable" in the sense that it only includes the **joint distribution** $p_\theta(\mathbf{x}, \mathbf{z})$, not the posterior $p(\mathbf{z}|\mathbf{x})$ or marginal likelihood $p_\theta(\mathbf{x})$.

Per-datapoint ELBO, part 2

Re-arranging (and putting back in our subscripts!), we have

$$\log p_\theta(\mathbf{x}_i) = \mathcal{L}(\mathbf{x}_i; \phi, \theta) + D_{KL}(q_\phi(\mathbf{z}_i|\mathbf{x}_i) \| p_\theta(\mathbf{z}_i|\mathbf{x}_i)),$$

with

$$\mathcal{L}(\mathbf{x}_i; \phi, \theta) = \mathbb{E}_{q_\phi(\mathbf{z}_i|\mathbf{x}_i)} \left[\log \frac{p_\theta(\mathbf{x}_i, \mathbf{z}_i)}{q_\phi(\mathbf{z}_i|\mathbf{x}_i)} \right].$$

Fortunately, for many models the "per-datapoint" ELBO relates easily to the full ELBO:

$$\log p_\theta(\mathbf{X}) = \log \int p_\theta(\mathbf{X}, \mathbf{Z}) \mathrm{d}\mathbf{Z}$$

Per-datapoint ELBO, part 2

Re-arranging (and putting back in our subscripts!), we have

$$\log p_\theta(\mathbf{x}_i) = \mathcal{L}(\mathbf{x}_i; \phi, \theta) + D_{KL}(q_\phi(\mathbf{z}_i|\mathbf{x}_i) \| p_\theta(\mathbf{z}_i|\mathbf{x}_i)),$$

with

$$\mathcal{L}(\mathbf{x}_i; \phi, \theta) = \mathbb{E}_{q_\phi(\mathbf{z}_i|\mathbf{x}_i)} \left[\log \frac{p_\theta(\mathbf{x}_i, \mathbf{z}_i)}{q_\phi(\mathbf{z}_i|\mathbf{x}_i)} \right].$$

Fortunately, for many models the "per-datapoint" ELBO relates easily to the full ELBO:

$$\log p_\theta(\mathbf{X}) = \log \int p_\theta(\mathbf{X}, \mathbf{Z}) \mathrm{d}\mathbf{Z} = \log \prod_{i=1}^{N} \int p_\theta(\mathbf{x}_i, \mathbf{z}_i) \mathrm{d}\mathbf{z}_i$$

Per-datapoint ELBO, part 2

Re-arranging (and putting back in our subscripts!), we have

$$\log p_\theta(\mathbf{x}_i) = \mathcal{L}(\mathbf{x}_i; \phi, \theta) + D_{KL}(q_\phi(\mathbf{z}_i|\mathbf{x}_i) \| p_\theta(\mathbf{z}_i|\mathbf{x}_i)),$$

with

$$\mathcal{L}(\mathbf{x}_i; \phi, \theta) = \mathbb{E}_{q_\phi(\mathbf{z}_i|\mathbf{x}_i)} \left[\log \frac{p_\theta(\mathbf{x}_i, \mathbf{z}_i)}{q_\phi(\mathbf{z}_i|\mathbf{x}_i)} \right].$$

Fortunately, for many models the "per-datapoint" ELBO relates easily to the full ELBO:

$$\begin{aligned} \log p_\theta(\mathbf{X}) = \log \int p_\theta(\mathbf{X}, \mathbf{Z}) \mathrm{d}\mathbf{Z} &= \log \prod_{i=1}^{N} \int p_\theta(\mathbf{x}_i, \mathbf{z}_i) \mathrm{d}\mathbf{z}_i \\ &= \sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i) \end{aligned}$$

Per-datapoint ELBO, part 2

Re-arranging (and putting back in our subscripts!), we have

$$\log p_\theta(\mathbf{x}_i) = \mathcal{L}(\mathbf{x}_i; \phi, \theta) + D_{KL}(q_\phi(\mathbf{z}_i|\mathbf{x}_i) \| p_\theta(\mathbf{z}_i|\mathbf{x}_i)),$$

with

$$\mathcal{L}(\mathbf{x}_i; \phi, \theta) = \mathbb{E}_{q_\phi(\mathbf{z}_i|\mathbf{x}_i)} \left[\log \frac{p_\theta(\mathbf{x}_i, \mathbf{z}_i)}{q_\phi(\mathbf{z}_i|\mathbf{x}_i)} \right].$$

Fortunately, for many models the "per-datapoint" ELBO relates easily to the full ELBO:

$$\begin{aligned} \log p_\theta(\mathbf{X}) = \log \int p_\theta(\mathbf{X}, \mathbf{Z}) \mathrm{d}\mathbf{Z} &= \log \prod_{i=1}^{N} \int p_\theta(\mathbf{x}_i, \mathbf{z}_i) \mathrm{d}\mathbf{z}_i \\ &= \sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i) \geq \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}_i; \phi, \theta) \end{aligned}$$

Per-datapoint ELBO, part 2

Re-arranging (and putting back in our subscripts!), we have

$$\log p_\theta(\mathbf{x}_i) = \mathcal{L}(\mathbf{x}_i; \phi, \theta) + D_{KL}(q_\phi(\mathbf{z}_i|\mathbf{x}_i) \| p_\theta(\mathbf{z}_i|\mathbf{x}_i)),$$

with

$$\mathcal{L}(\mathbf{x}_i; \phi, \theta) = \mathbb{E}_{q_\phi(\mathbf{z}_i|\mathbf{x}_i)} \left[\log \frac{p_\theta(\mathbf{x}_i, \mathbf{z}_i)}{q_\phi(\mathbf{z}_i|\mathbf{x}_i)} \right].$$

Fortunately, for many models the "per-datapoint" ELBO relates easily to the full ELBO:

$$\begin{aligned} \log p_\theta(\mathbf{X}) = \log \int p_\theta(\mathbf{X}, \mathbf{Z}) \mathrm{d}\mathbf{Z} &= \log \prod_{i=1}^{N} \int p_\theta(\mathbf{x}_i, \mathbf{z}_i) \mathrm{d}\mathbf{z}_i \\ &= \sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i) \geq \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}_i; \phi, \theta) =: \mathcal{L}(\mathbf{X}; \phi, \theta) \end{aligned}$$

The evidence lower bound (ELBO)

What about θ? Again,

$$\log p_\theta(\mathbf{x}) = \mathcal{L}(\mathbf{x}; \phi, \theta) + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})).$$

For maximum likelihood estimation of θ, we can (approximately) maximize $\log p_\theta(\mathbf{x})$ by maximizing the ELBO $\mathcal{L}(\mathbf{x}; \phi, \theta) \leq \log p_\theta(\mathbf{x})$; as a surrogate.

- We saw this before for hyperparameter optimization — but now, **we are optimizing all of our neural network parameters** θ! We are only being "Bayesian" about the representation $\mathbf{z}$.
- **Note**: This lower bound is tight when the KL divergence is zero, i.e. if we have learned the correct posterior!

Optimizing the ELBO

So far, we have seen that maximizing the ELBO

$$\mathcal{L}(\mathbf{x};\phi,\theta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{reconstruction term}} - \underbrace{D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))}_{\text{regularization term}}$$

is sensible for estimating both

- the generative parameters θ, and
- the inference network parameters ϕ.

Optimizing the ELBO

So far, we have seen that maximizing the ELBO

$$\mathcal{L}(\mathbf{x};\phi,\theta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{reconstruction term}} - \underbrace{D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))}_{\text{regularization term}}$$

is sensible for estimating both

- the generative parameters θ, and
- the inference network parameters ϕ.

Let's look at the **reconstruction term** in the ELBO, $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]$.

- Gradients ∇_θ: not a problem,

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

- Gradients ∇_ϕ: **reparameterization trick**

Reparameterization trick for the encoder

Essentially, we need to make sure we can reparameterize the encoder distribution $q_\phi(\mathbf{z}|\mathbf{x})$. Define a function $g_\phi(\epsilon, \mathbf{x})$ and distribution $p(\epsilon)$ such that if

$$\epsilon \sim p(\epsilon), \qquad \tilde{\mathbf{z}} = g_\phi(\epsilon, \mathbf{x})$$

then $\tilde{\mathbf{z}} \sim q_\phi(\mathbf{z}|\mathbf{x})$.

Then, for example, a Gaussian encoder

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mu(\mathbf{x}), \mathrm{diag}(\sigma(\mathbf{x})^2)).$$

would have

$$p(\epsilon) = \mathcal{N}(0, \mathbf{I})$$
$$\tilde{\mathbf{z}} = g_\phi(\epsilon, \mathbf{x}) = \mu(\mathbf{x}) + \sigma(\mathbf{x}) \odot \epsilon.$$

Kingma & Welling, 2014

Autodiff and SGD (the usual!)

Now we can compute the gradient (using **autodiff**)

$$\begin{aligned}\nabla_{\phi,\theta}\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] &= \nabla_{\phi,\theta}\mathbb{E}_{p(\epsilon)}[\log p_\theta(\mathbf{x}|\tilde{\mathbf{z}} = g_\phi(\epsilon, \mathbf{x}))] \\ &= \mathbb{E}_{p(\epsilon)}[\nabla_{\phi,\theta} \log p_\theta(\mathbf{x}|\tilde{\mathbf{z}} = g_\phi(\epsilon, \mathbf{x}))]\end{aligned}$$

Kingma & Welling, 2014

Autodiff and SGD (the usual!)

Now we can compute the gradient (using **autodiff**)

$$\begin{aligned}\nabla_{\phi,\theta}\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] &= \nabla_{\phi,\theta}\mathbb{E}_{p(\epsilon)}[\log p_\theta(\mathbf{x}|\tilde{\mathbf{z}} = g_\phi(\epsilon, \mathbf{x}))] \\ &= \mathbb{E}_{p(\epsilon)}[\nabla_{\phi,\theta} \log p_\theta(\mathbf{x}|\tilde{\mathbf{z}} = g_\phi(\epsilon, \mathbf{x}))]\end{aligned}$$

Note that if $p_\theta(\mathbf{x}|\mathbf{z})$ is Gaussian, e.g. $\mathcal{N}(\mathbf{x}|f_\theta(\mathbf{z}), \sigma^2\mathbf{I})$, then this term involves differentiating the squared error loss

$$\log p_\theta(\mathbf{x}|\tilde{\mathbf{z}} = g_\phi(\epsilon, \mathbf{x})) = -\frac{1}{2\sigma^2}\|\mathbf{x} - f_\theta(g_\phi(\epsilon, \mathbf{x}))\|_2^2 + \text{const.}$$

This recovers the (probabilistic) PCA setting if f_θ and g_ϕ are linear functions.

Kingma & Welling, 2014

Mini-batch gradient estimation

The last missing step is considering mini-batch gradient updates. Since

$$\log p_\theta(\mathbf{X}) = \sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i) \geq \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}_i; \phi, \theta)$$

then we can use a minibatch estimator

$$\sum_{i=1}^{N} \mathcal{L}(\mathbf{x}_i; \phi, \theta) \approx \frac{N}{M} \sum_{j=1}^{M} \mathcal{L}(\mathbf{x}_j; \phi, \theta),$$

where $\mathbf{x}_j$ are M random samples from the dataset.

Summary

A **variational autoencoder** is

- a deep generative model for data $\mathbf{x}$, with a latent variable $\mathbf{z}$
- ... which takes the form

$$p_\theta(\mathbf{X}, \mathbf{Z}) = \prod_{i=1}^{N} p_\theta(\mathbf{x}_i|\mathbf{z}_i)p(\mathbf{z}_i)$$

- ... trained jointly with an approximation to the posterior

$$q_\phi(\mathbf{z}_i|\mathbf{x}_i) \approx p_\theta(\mathbf{z}_i|\mathbf{x}_i)$$

- ... by maximizing an ELBO.

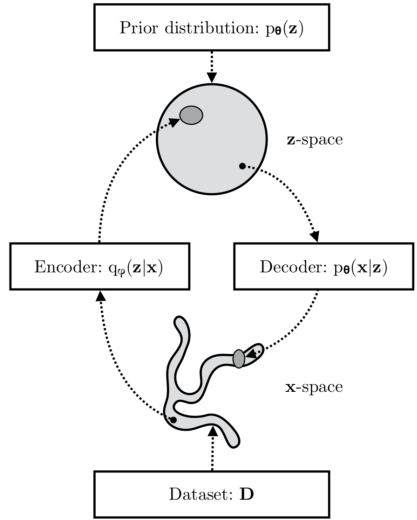


Figure: Kingma & Welling (2019)

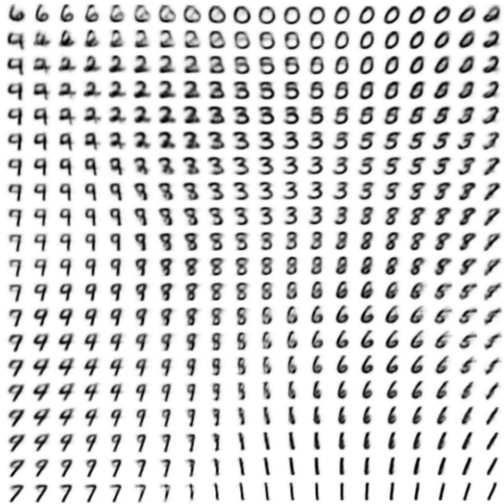
Examples

The "hello world" of deep generative models

MNIST dataset.

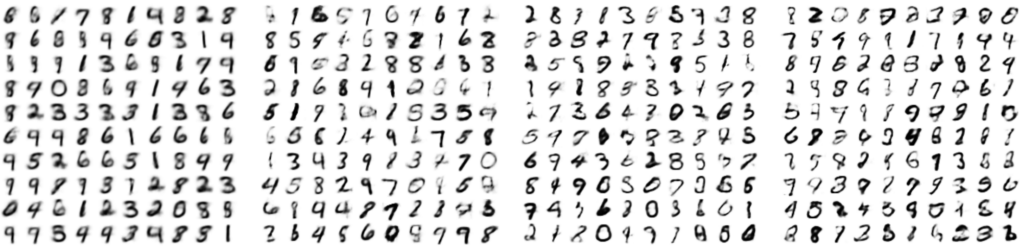
- **Encoder**: feed-forward network, Gaussian posterior
- **Decoder**: feed-forward network, Bernoulli or Gaussian likelihood
- 2-dimensional latent space $\mathbf{z}$

Plot shows the mean of the decoder for different values of the latent $\mathbf{z}$.



Kingma & Welling (2014)

MNIST: latent space comparison



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

- Non-obvious question: how big should the latent space be?

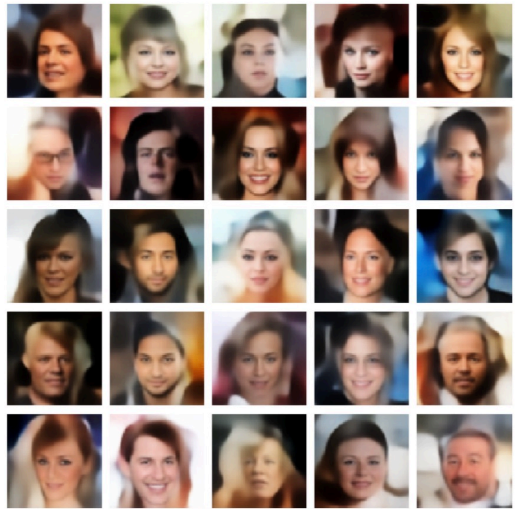
Kingma & Welling (2014)

Convolutional networks

CelebA dataset.

- **Encoder**: Fully convolutional network, Gaussian posterior
- **Decoder**: Fully convolutional network, Gaussian likelihood

Note: recent papers with different architectures and likelihoods can produce more photorealistic random samples.



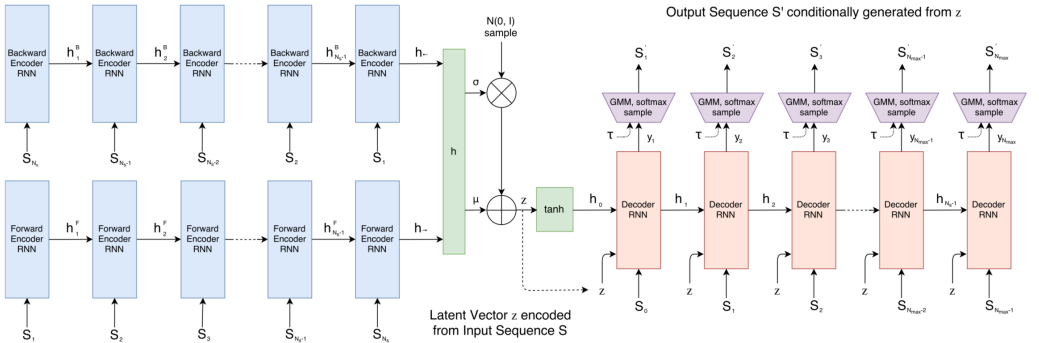
Lamb et al. (2016)

Perturbations in latent space

- **Uses**: We can identify directions in the latent space that correspond to properties, and manipulate them

White (2016)

A sequence model: SketchRNN



- **Encoder**: bi-directional LSTM, Gaussian posterior
- **Decoder**: LSTM that outputs parameters of a distribution over 5-tuples $(\Delta i, \Delta j,$ "pen up", "pen down", "stop"$)$

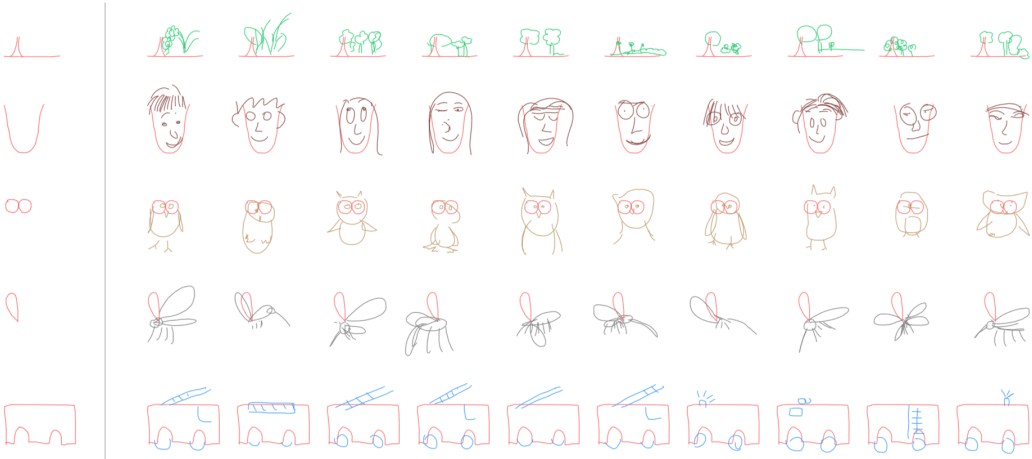
Ha & Eck (2017)

A sequence model: SketchRNN

- **Uses**: Reconstruction, generation, latent space manipulation, ...

Ha & Eck (2017)

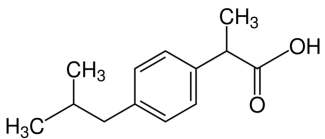
A sequence model: SketchRNN



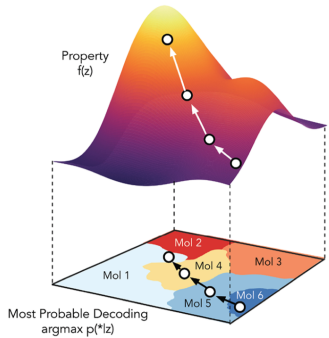
- **Uses**: Automatic completion of partial sketches

Ha & Eck (2017)

Sequence model for molecules



Ibuprofen: CC(C)CC1=CC=C(C=C1)C(C)C(=O)O



- Molecules are complex, discrete objects, but can be described succinctly as strings in a formal language
- **Uses**: Lift discrete optimization to a continuous latent space

Gomez-Bombarelli et al. (2018)

Application:

Semi-supervised learning

Learning with labels

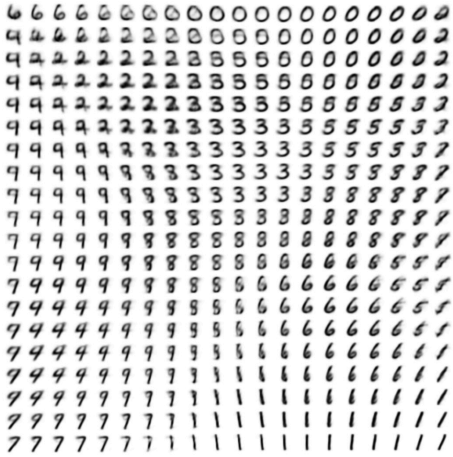
So far, we have only considered unlabelled data $\mathbf{x}$.
Question: What should we do with labeled pairs $(\mathbf{x}_i, y_i)$?

Learning with labels

Let's take MNIST as a running example.

- Images of handwritten digits $\mathbf{x}_i$
- Digit labels $y_i \in 0, \ldots, 9$
- The 2d learned latent space is shown again on the right.

Reminder: this was trained on an unsupervised model, that only had access to images $\mathbf{x}_i$!



Defining a joint generative model

We're going to define a generative model over the joint distribution $\mathbf{x}, y$, with a latent variable $\mathbf{z}$.

- **Question**: How should we factorize $p_\theta(\mathbf{x}, y, \mathbf{z})$?

Defining a joint generative model

We're going to define a generative model over the joint distribution $\mathbf{x}, y$, with a latent variable $\mathbf{z}$.

- **Question**: How should we factorize $p_\theta(\mathbf{x}, y, \mathbf{z})$?
- How do we actually draw images? Generally, it's helpful to mirror the real-world causal direction as much as possible.

Defining a joint generative model

We're going to define a generative model over the joint distribution $\mathbf{x}, y$, with a latent variable $\mathbf{z}$.

- **Question**: How should we factorize $p_\theta(\mathbf{x}, y, \mathbf{z})$?
- How do we actually draw images? Generally, it's helpful to mirror the real-world causal direction as much as possible.

A reasonable choice:

$$p_\theta(\mathbf{x}, y, \mathbf{z}) = p_\theta(\mathbf{x}|y, \mathbf{z})p(\mathbf{z})p(y)$$

"First, independently pick a digit y, and a latent style vector $\mathbf{z}$. Then, go draw the corresponding character $\mathbf{x}$."

Generative model and inference networks

We'll mostly follow the same structure used before, but now with an extra random variable y.

Generative model:

$$
\begin{aligned}
p(\mathbf{z}) &= \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}) \\
p(y) &= \text{Discrete}(y|\boldsymbol{\pi}) \\
p_\theta(\mathbf{x}|y, \mathbf{z}) &= f(\mathbf{x}; y, \mathbf{z}, \theta)
\end{aligned}
$$

where $\boldsymbol{\pi}$ is a prior probability vector, and f is an appropriate probability density (e.g. Gaussian) with parameters given by a nonlinear transformation of $\mathbf{z}, y$.

Generative model and inference networks

For the inference network, we also need to choose a factorization.

Approximate posterior:

$$q(y|\mathbf{x}) = \text{Discrete}(y|\pi_\phi(\mathbf{x}))$$
$$q(\mathbf{z}|\mathbf{x}, y) = \mathcal{N}(\mathbf{z}|\mu_\phi(\mathbf{x}, y), \text{diag}(\sigma_\phi(\mathbf{x}, y)^2))$$

where π_ϕ, μ_ϕ, and σ_ϕ are all deep networks.

Questions: Why infer y first? Why does $\mathbf{z}$ depend on both $\mathbf{x}$ and y?

Kingma et al. (2014)

Semi-supervised setting

In the **semi-supervised setting**, we assume that the labels y are known for some instances, but not for others.

This effectively partitions the dataset into

- N^s image, label pairs $(\mathbf{x}_i, y_i)$
- N^u unlabelled images $\mathbf{x}_j$.

We'll treat these two sets of data separately, looking at their per-datapoint contributions to an ELBO.

Kingma et al. (2014)

Supervised ELBO

Let's start with the **supervised case**, focusing only on instances $\mathbf{x}$ where y is known, with

$$\mathcal{L}(\mathbf{x}, y; \phi, \theta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},y)}[\log p_\theta(\mathbf{x}, y, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}, y)].$$

This ELBO is essentially identical to the one we derived before, aside from the fact that our observed "data" now also includes y.

$$\mathcal{L}(\mathbf{x}, y; \phi, \theta) \leq \log p_\theta(\mathbf{x}, y)$$

Kingma et al. (2014)

Unsupervised ELBO

In the **unsupervised case**, we do not know y, and need to

$$\mathcal{U}(\mathbf{x}; \phi, \theta) = \mathbb{E}_{q_\phi(y|\mathbf{x})q_\phi(\mathbf{z}|\mathbf{x},y)}[\log p_\theta(\mathbf{x}, y, \mathbf{z}) - \log q_\phi(y|\mathbf{x}) - \log q_\phi(\mathbf{z}|\mathbf{x}, y)].$$

This ELBO is **also essentially identical** to the one we derived before, except now our latent space also includes y.

$$\mathcal{U}(\mathbf{x}; \phi, \theta) \leq \log p_\theta(\mathbf{x})$$

Kingma et al. (2014)

Semi-supervised objective function

Putting these together, we can define an overall objective function

$$\mathcal{J} = \sum_{i=1}^{N^s} \mathcal{L}(\mathbf{x}_i, y_i; \phi, \theta) + \sum_{j=1}^{N^u} \mathcal{U}(\mathbf{x}_j; \phi, \theta).$$

There's just one thing odd about this objective: the supervised terms do not include the quantity we would traditionally call a classifier,

$$q_\phi(y|\mathbf{x}) = \text{Discrete}(y|\pi_\phi(\mathbf{x})),$$

which means the classifier network $\pi_\phi(\mathbf{x})$ is *only* estimated using the unsupervised data $\mathbf{x}_i$.

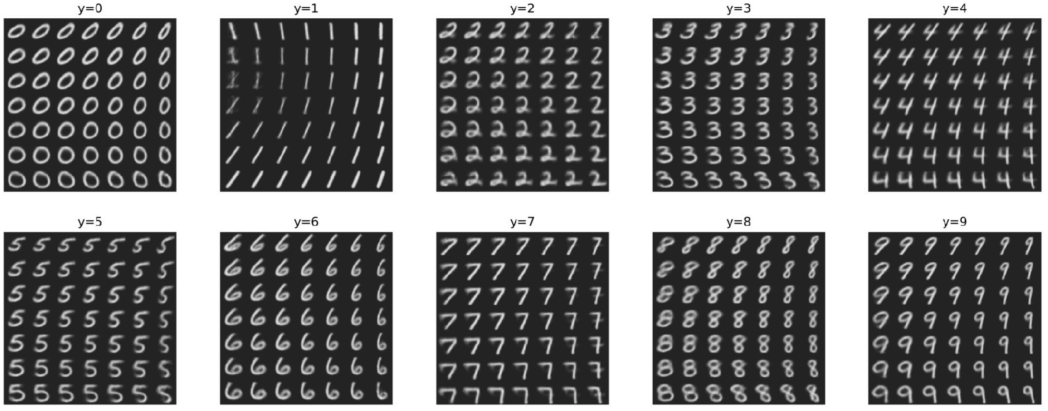
This counterintuitive behaviour can be avoided by adding in an additional term

$$\tilde{\mathcal{J}} = \mathcal{J} + \alpha \sum_{i=1}^{N^s} \log q_\phi(y_i|\mathbf{x}_i).$$

Check-in

- **Question**: Compared to the previous VAE, what do we think the latent space $\mathbf{z}$ will look like?
- **Question**: What is the difference between training this model when fully supervised, and learning ten independent VAEs (one per class)?

Class-conditional simulation

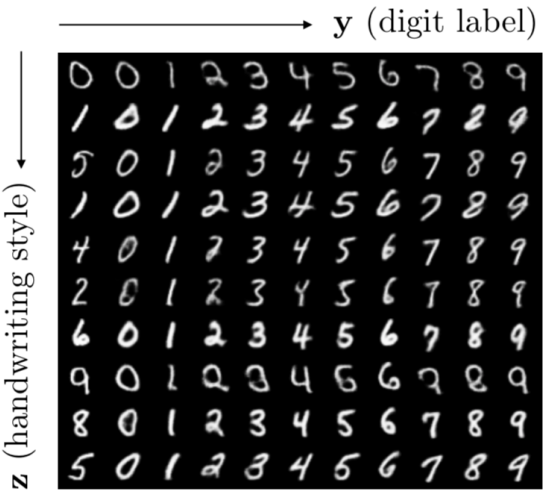


- First: the generated images do represent the target class.
- Second: the latent factor $\mathbf{z}$ seems to capture other independent variation in a consistent manner.

Kingma et al. (2014)

Style transfer, or visual analogies

- Holding the random variable $\mathbf{z}$ constant while modifying y transfers *style* across classes
- Each row corresponds to a fixed value of $\mathbf{z}_i$
- Each column past the first shows the mean $p_\theta(\mathbf{x}|y_k, \mathbf{z}_i)$, for $k = 0, \ldots, 9$



Kingma et al. (2014)

Street view house numbers data

- Same thing, on a potentially more impressive dataset
- First column shows real data, the rest show label-conditional reconstructions

Kingma et al. (2014)

How good is the classifier?

- Kingma et al. (2014) report 3.33% test error on MNIST when using only 100 labels, i.e. 10 labels per class...
- ...and down to 2.18% error for 3000 labels.
- Later work, Siddharth et al. (2017), uses different network architectures and reaches 1.57% test error on 3000 total labels, which is fairly comparable to looking at the full dataset.

The full MNIST dataset has 60k examples.

SVHN classification performance tells a similar story.