Neural Networks and Deep Learning

Autoencoders, Variational Autoencoders

Adam Bloniarz

Department of Computer Science
adam.bloniarz@colorado.edu

April 13, 2020

University of Colorado **Boulder**

An **autoencoder** is a kind of neural network that learns to copy its input.

A deterministic autoencoder consists of two functions:

- Encoder function $\mathbf{h} = \mathbf{f}(\mathbf{x})$
- Decoder function $\mathbf{x}' = \mathbf{g}(\mathbf{h})$

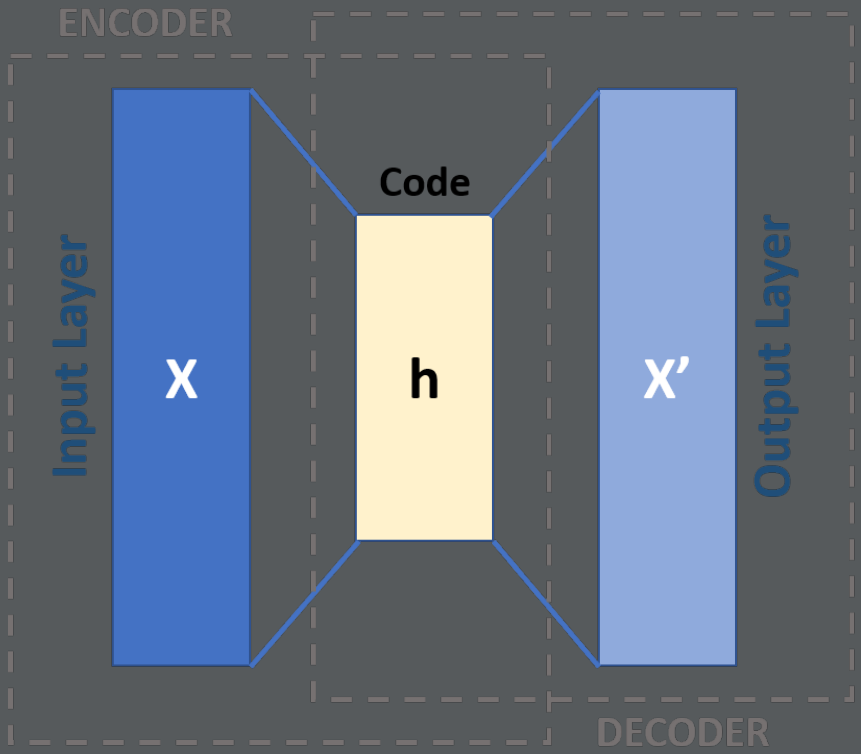
The autoencoder is trained to minimize a reconstruction error $L(\mathbf{x}, g(f(\mathbf{x})))$.

Often, the squared error loss is used:

$$L(\mathbf{x}, g(f(\mathbf{x}))) = \|\mathbf{x} - \mathbf{g}(\mathbf{f}(\mathbf{x}))\|^2$$

Undercomplete autoencoders

One strategy is to force the network to learn a low-dimensional representation of the data.



Linear undercomplete autoencoder

Consider the extreme case of representing an input with a single number. Let $\mathbf{x}_i \in \mathbb{R}^d, \mathbf{v} \in \mathbb{R}^d, \mathbf{w} \in \mathbb{R}^d$. Ignore the bias terms.

$$h_i = \mathbf{v}^T \mathbf{x}_i$$
$$\mathbf{x}'_i = \mathbf{w} h_i$$

Pack the dataset into a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$. We want to solve:

$$\min_{\mathbf{v},\mathbf{w}} \left\| \mathbf{X} - \mathbf{X}\mathbf{v}\mathbf{w}^T \right\|_F^2$$

Note that $\mathbf{X}\mathbf{v}\mathbf{w}^T$ is a rank-1 matrix. It can be shown that this problem boils down to solving

$$\min_{\mathbf{B}} \|\mathbf{X} - \mathbf{B}\|_F^2 \text{ s.t. } \mathbf{B} \text{ is rank 1.}$$

Note that $\mathbf{X}\mathbf{v}\mathbf{w}^T$ is a rank-1 matrix. It can be shown that this problem boils down to solving

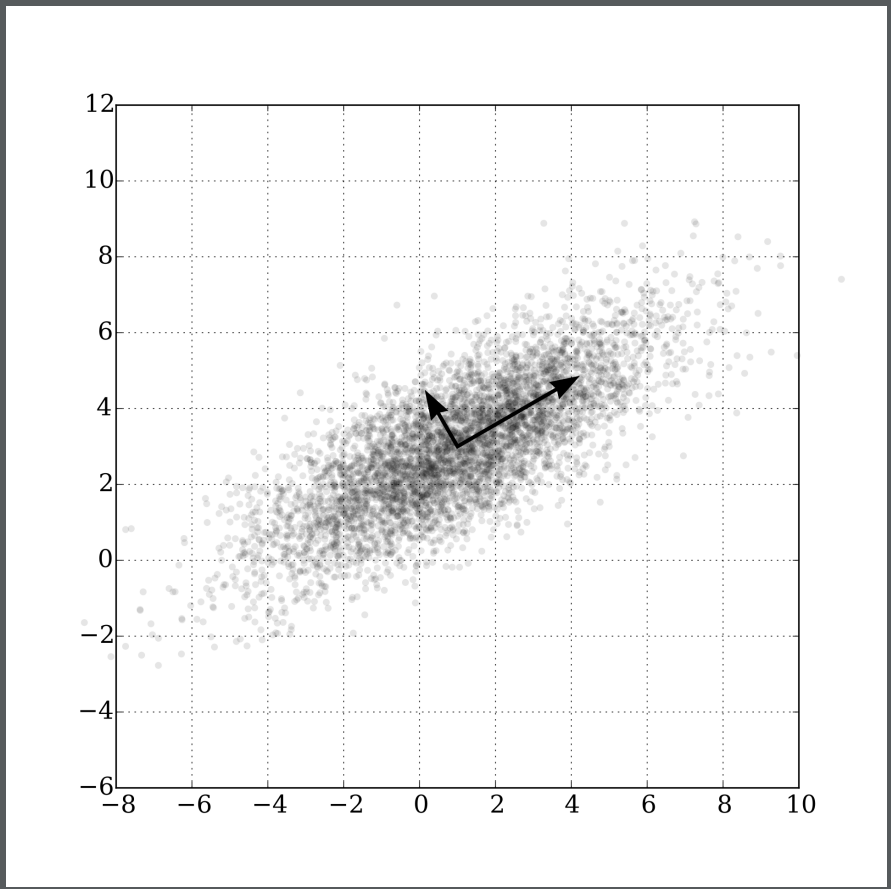
$$\min_{\mathbf{B}} \|\mathbf{X} - \mathbf{B}\|_F^2 \text{ s.t. } \mathbf{B} \text{ is rank 1.}$$

This is solved by setting

$$\begin{aligned} \mathbf{v} = \mathbf{w} = & \text{ the first right-singular vector of } \mathbf{X}. \\ = & \text{ the leading eigenvector of } \mathbf{X}^T\mathbf{X}. \end{aligned}$$

See Theorem 3.6 in Foundations of data science: the best rank-k approximation of $\mathbf{X}$ is given by the first k singular vectors of $\mathbf{X}$.

The singular value decomposition of $\mathbf{X}$ corresponds to the principal component analysis (PCA) of $\mathbf{X}$.
PCA decomposes a signal into uncorrelated components such that for all k, the first k components account for the largest possible variance of $\mathbf{X}$.



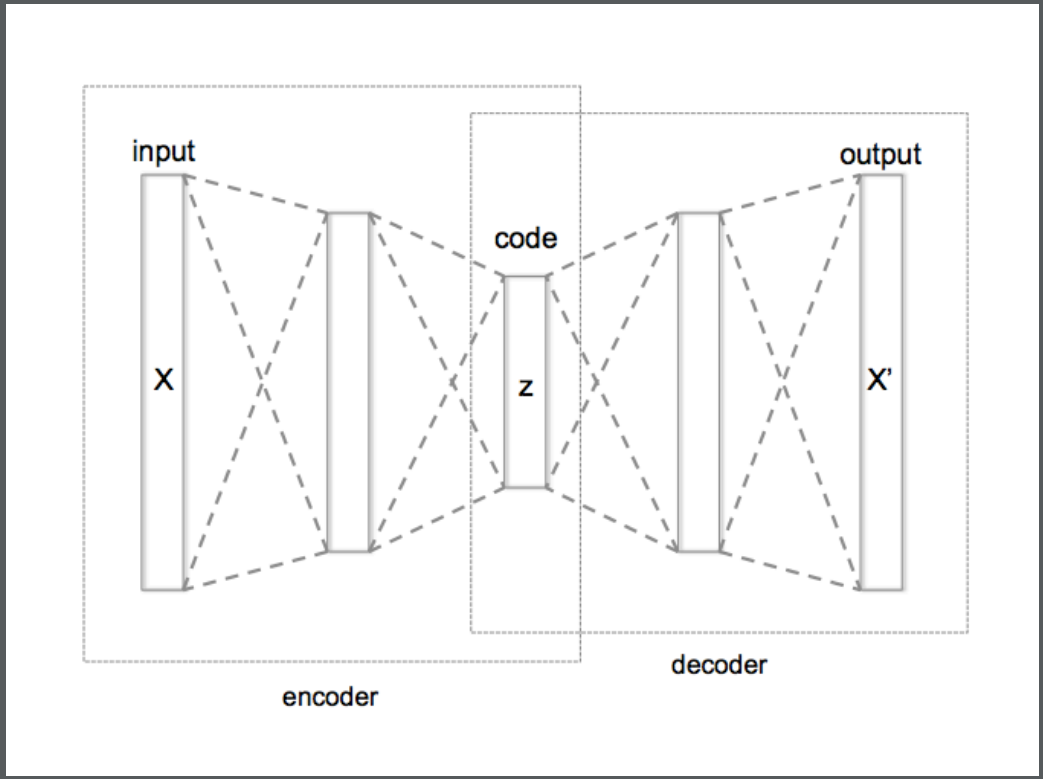
If we train a linear auto-encoder

$$\min_{\mathbf{v},\mathbf{w}} \left\| \mathbf{X} - \mathbf{X}\mathbf{v}\mathbf{w}^T \right\|_F^2$$

with standard NN tools (e.g. minibatch SGD), it will not know about eigenvalues or SVD, but if all goes well, it finds a solution such that $\mathbf{v}\mathbf{w}^T$ projects $\mathbf{X}$ into the space spanned by the first singular vector.

[Visualizing PCA on MNIST]

In general a deep autoencoder may be parametrized by a multi-layer perceptron or CNN on the encoder and decoder sides:



Need to make sure that the network does not have the capacity to learn the identity function.

Generative models

We can consider an autoencoder to be an approximation to a generative model with a hidden variable $\mathbf{z}_i$:

This graph represents a joint distribution that can be written as

$$P(\mathbf{z}_i, \mathbf{x}_i) = P(\mathbf{z}_i)P(\mathbf{x}_i \mid z_i)$$

Think of $\mathbf{z}_i$ as the underlying structure of the signal (e.g. image contents), and $\mathbf{x}_i$ as the observed signal (e.g. image pixels).

Think of the two halves of the autoencoder as follows:

$$\mathbf{f}(\mathbf{x}) \approx \arg\max_{\mathbf{z}} P(\mathbf{z} \mid \mathbf{x})$$

$$\mathbf{g}(\mathbf{z}) \approx \arg\max_{\mathbf{x}'} P(\mathbf{x} \mid \mathbf{z})$$

PCA (linear autoencoder) as maximum-likelihood

Tipping, Bishop, 2002 showed that PCA provides the maximum-likelihood estimation of the parameters of a hidden variable model:

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$
$$\mathbf{x} \mid \mathbf{z} \sim \mathcal{N}(\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2 \mathbf{I})$$

This is called probabilistic PCA.

In probabilistic PCA, the 'decoder' is a linear transformation:

$$\mathbf{x} \mid \mathbf{z} \sim \mathcal{N}(\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2\mathbf{I})$$

What about a more complex transformation, such as a neural network g that produces both the mean and covariance parameters?

$$\mathbf{x} \mid \mathbf{z} \sim \mathcal{N}(g_\mu(\mathbf{z}), g_\Sigma(\mathbf{z}))$$

Fitting such a model seems hopeless:

The likelihood (evidence) is:

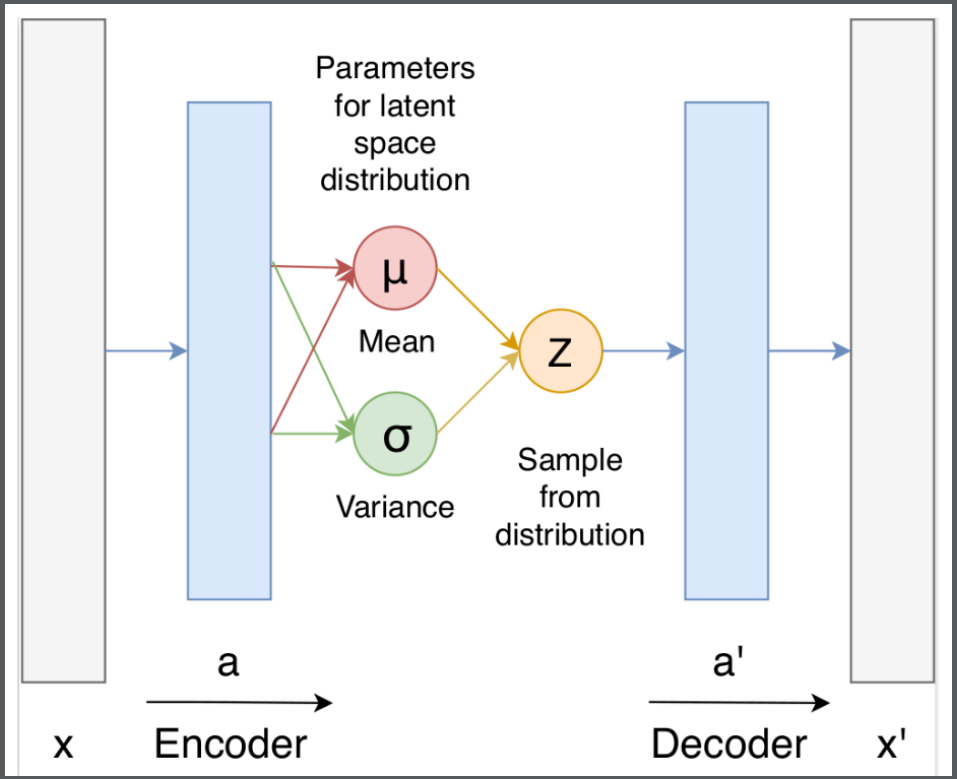
$$P(\mathbf{x}) = \int P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z}) d\mathbf{z}$$

Posterior inference (i.e. the encoder) is, by Bayes rule

$$P(\mathbf{z} \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z})}{\int P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z}) d\mathbf{z}}$$

These involve an intractable integral.

The variational autoencoder (Kingma, Welling 2014) provides a way to fit such models.



The VAE is a kind of stochastic neural network, along with a particular training procedure.

Variational inference: approximate the posterior $P(\mathbf{z} \mid \mathbf{x})$ with a *variational* distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$

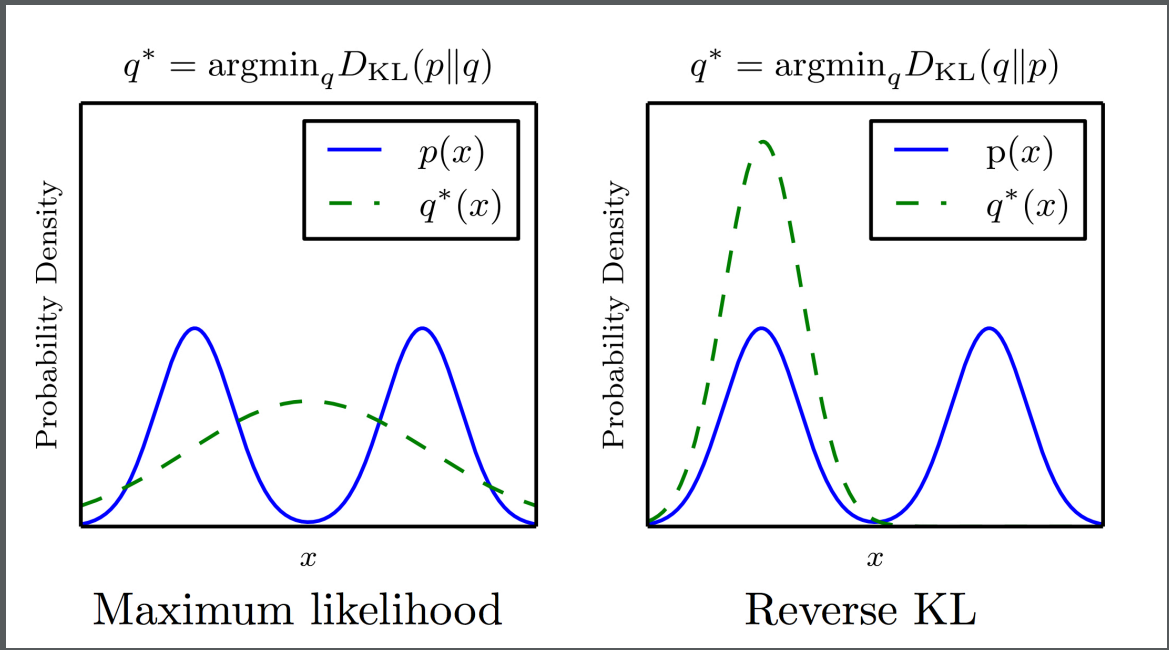
Solve an optimization problem to minimize

$$D_{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}) \| P(\mathbf{z} \mid \mathbf{x})) = E_{q_\phi(\mathbf{z}|\mathbf{x})}\left(\log \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{P(\mathbf{z} \mid \mathbf{x})}\right)$$

Kullback-Leibler divergence

Kullback-Leibler divergence is a measure of how one probability distribution differs from a second, reference distribution.

$$D_{KL}(P\|Q) = E_P \log \frac{P(x)}{Q(x)}$$



Source: Deep Learning Book, Figure 3.6

Approximating the likelihood with the evidence lower bound

We want to maximize the *marginal* likelihood $P_\theta(\mathbf{x})$, where θ parametrizes the joint distribution $P_\theta(\mathbf{z}, \mathbf{x})$, i.e. the decoder network.

Approximating the likelihood with the evidence lower bound

We want to maximize the *marginal* likelihood $P_\theta(\mathbf{x})$, where θ parametrizes the joint distribution $P_\theta(\mathbf{z}, \mathbf{x})$, i.e. the decoder network.

Can we approximate $P_\theta(\mathbf{x})$ in terms of tractable quantities?

Approximating the likelihood with the evidence lower bound

We want to maximize the *marginal* likelihood $P_\theta(\mathbf{x})$, where θ parametrizes the joint distribution $P_\theta(\mathbf{z}, \mathbf{x})$, i.e. the decoder network.

Can we approximate $P_\theta(\mathbf{x})$ in terms of tractable quantities?

By the definition of conditional probability, $P(\mathbf{z} \mid \mathbf{x}) = \frac{P(\mathbf{z},\mathbf{x})}{P(\mathbf{x})}$.

Approximating the likelihood with the evidence lower bound

We want to maximize the *marginal* likelihood $P_\theta(\mathbf{x})$, where θ parametrizes the joint distribution $P_\theta(\mathbf{z}, \mathbf{x})$, i.e. the decoder network.

Can we approximate $P_\theta(\mathbf{x})$ in terms of tractable quantities?

By the definition of conditional probability, $P(\mathbf{z} \mid \mathbf{x}) = \frac{P(\mathbf{z},\mathbf{x})}{P(\mathbf{x})}$.

Therefore, $P(\mathbf{x}) = \frac{P(\mathbf{z},\mathbf{x})}{P(\mathbf{z}|\mathbf{x})}$ holds *for all* $\mathbf{z}$

Rewrite the log-likelihood:

$$\log P(\mathbf{x}) = \log P(\mathbf{z}, \mathbf{x}) - \log P(\mathbf{z} \mid \mathbf{x})$$

Rewrite the log-likelihood:

$$\log P(\mathbf{x}) = \log P(\mathbf{z}, \mathbf{x}) - \log P(\mathbf{z} \mid \mathbf{x})$$

Because this holds for all $\mathbf{z}$, we can take an expectation over any distribution q of $\mathbf{z}$:

$$\log P(\mathbf{x}) = E_{q(\mathbf{z})} \log P(\mathbf{z}, \mathbf{x}) - E_{q(\mathbf{z})} \log P(\mathbf{z} \mid \mathbf{x})$$

We can derive a lower bound on the marginal likelihood, called the evidence lower bound (ELBO).

$$\log P(\mathbf{x}) = E_{q(\mathbf{z})} \log P(\mathbf{z}, \mathbf{x}) - E_{q(\mathbf{z})} \log P(\mathbf{z} \mid \mathbf{x})$$

We can derive a lower bound on the marginal likelihood, called the evidence lower bound (ELBO).

$$\begin{aligned}\log P(\mathbf{x}) &= E_{q(\mathbf{z})} \log P(\mathbf{z}, \mathbf{x}) - E_{q(\mathbf{z})} \log P(\mathbf{z} \mid \mathbf{x}) \\ &= E_{q(\mathbf{z})} \log P(\mathbf{z}, \mathbf{x}) - E_{q(\mathbf{z})} \log q(\mathbf{z}) + E_{q(\mathbf{z})} \log q(\mathbf{z}) \\ &\quad - E_{q(\mathbf{z})} \log P(\mathbf{z} \mid \mathbf{x})\end{aligned}$$

We can derive a lower bound on the marginal likelihood, called the evidence lower bound (ELBO).

$$\begin{aligned}
\log P(\mathbf{x}) &= E_{q(\mathbf{z})} \log P(\mathbf{z}, \mathbf{x}) - E_{q(\mathbf{z})} \log P(\mathbf{z} \mid \mathbf{x}) \\
&= E_{q(\mathbf{z})} \log P(\mathbf{z}, \mathbf{x}) - E_{q(\mathbf{z})} \log q(\mathbf{z}) + E_{q(\mathbf{z})} \log q(\mathbf{z}) \\
&\quad - E_{q(\mathbf{z})} \log P(\mathbf{z} \mid \mathbf{x}) \\
&= E_{q(\mathbf{z})} \log P(\mathbf{z}, \mathbf{x}) - E_{q(\mathbf{z})} \log q(\mathbf{z}) + D_{KL}\left(q(\mathbf{z}) \| P(\mathbf{z} \mid \mathbf{x})\right)
\end{aligned}$$

Using the fact that $P(\mathbf{z}, \mathbf{x}) = P(\mathbf{z})P(\mathbf{x} \mid \mathbf{z})$:

$$\log P(\mathbf{x}) = E_{q(\mathbf{z})} \log P(\mathbf{z}, \mathbf{x}) - E_{q(\mathbf{z})} \log q(\mathbf{z}) + D_{KL}(q(\mathbf{z}) \| P(\mathbf{z} \mid \mathbf{x}))$$

Using the fact that $P(\mathbf{z}, \mathbf{x}) = P(\mathbf{z})P(\mathbf{x} \mid \mathbf{z})$:

$$\begin{aligned}\log P(\mathbf{x}) &= E_{q(\mathbf{z})} \log P(\mathbf{z}, \mathbf{x}) - E_{q(\mathbf{z})} \log q(\mathbf{z}) + D_{KL}(q(\mathbf{z}) \| P(\mathbf{z} \mid \mathbf{x})) \\ &= E_{q(\mathbf{z})} \log P(\mathbf{x} \mid \mathbf{z}) + E_{q(\mathbf{z})} \log P(\mathbf{z}) - E_{q(\mathbf{z})} \log q(\mathbf{z}) \\ &\quad + D_{KL}\left(q(\mathbf{z}) \| P(\mathbf{z} \mid \mathbf{x})\right)\end{aligned}$$

Using the fact that $P(\mathbf{z}, \mathbf{x}) = P(\mathbf{z})P(\mathbf{x} \mid \mathbf{z})$:

$$\begin{aligned}
\log P(\mathbf{x}) &= E_{q(\mathbf{z})} \log P(\mathbf{z}, \mathbf{x}) - E_{q(\mathbf{z})} \log q(\mathbf{z}) + D_{KL}(q(\mathbf{z}) \| P(\mathbf{z} \mid \mathbf{x})) \\
&= E_{q(\mathbf{z})} \log P(\mathbf{x} \mid \mathbf{z}) + E_{q(\mathbf{z})} \log P(\mathbf{z}) - E_{q(\mathbf{z})} \log q(\mathbf{z}) \\
&\quad + D_{KL}\left(q(\mathbf{z}) \| P(\mathbf{z} \mid \mathbf{x})\right) \\
&= \underbrace{E_{q(\mathbf{z})} \log P(\mathbf{x} \mid \mathbf{z}) - D_{KL}\left(q(\mathbf{z}) \| P(\mathbf{z})\right)}_{\text{ELBO: } \mathcal{L}(q)} + \underbrace{D_{KL}\left(q(\mathbf{z}) \| P(\mathbf{z} \mid \mathbf{x})\right)}_{\text{Intractable, but } \geq 0}
\end{aligned}$$

Using the fact that $P(\mathbf{z}, \mathbf{x}) = P(\mathbf{z})P(\mathbf{x} \mid \mathbf{z})$:

$$\begin{aligned}
\log P(\mathbf{x}) &= E_{q(\mathbf{z})} \log P(\mathbf{z}, \mathbf{x}) - E_{q(\mathbf{z})} \log q(\mathbf{z}) + D_{KL}(q(\mathbf{z}) \| P(\mathbf{z} \mid \mathbf{x})) \\
&= E_{q(\mathbf{z})} \log P(\mathbf{x} \mid \mathbf{z}) + E_{q(\mathbf{z})} \log P(\mathbf{z}) - E_{q(\mathbf{z})} \log q(\mathbf{z}) \\
&\quad + D_{KL}\left(q(\mathbf{z}) \| P(\mathbf{z} \mid \mathbf{x})\right) \\
&= \underbrace{E_{q(\mathbf{z})} \log P(\mathbf{x} \mid \mathbf{z}) - D_{KL}\left(q(\mathbf{z}) \| P(\mathbf{z})\right)}_{\text{ELBO: } \mathcal{L}(q)} + \underbrace{D_{KL}\left(q(\mathbf{z}) \| P(\mathbf{z} \mid \mathbf{x})\right)}_{\text{Intractable, but } \geq 0}
\end{aligned}$$

We parametrize $q(\mathbf{z}) = q_\phi(\mathbf{z} \mid \mathbf{x})$ using a neural network to map $\mathbf{x}$ to a distribution over $\mathbf{z}$, and maximize the ELBO w.r.t. ϕ and θ.

Reminder: θ parametrizes the distribution $P(\mathbf{z}, \mathbf{x})$, i.e. the decoder network.

To train a VAE, we need to optimize

$$\mathcal{L}(\theta, \phi; \mathbf{x}_i) = E_{q_\phi(\mathbf{z}|\mathbf{x}_i)} \left[\log P(\mathbf{x}_i \mid \mathbf{z})\right] - D_{KL} \left(q_\phi(\mathbf{z} \mid \mathbf{x}_i) \| P(\mathbf{z})\right)$$

To train a VAE, we need to optimize

$$\mathcal{L}(\theta, \phi; \mathbf{x}_i) = E_{q_\phi(\mathbf{z}|\mathbf{x}_i)} \left[\log P(\mathbf{x}_i \mid \mathbf{z})\right] - D_{KL} \left(q_\phi(\mathbf{z} \mid \mathbf{x}_i) \| P(\mathbf{z})\right)$$

For this, we need to calculate a gradient of the form

$$\nabla_\phi E_{q_\phi(\mathbf{z}|\mathbf{x}_i)} \left[h(\mathbf{z})\right]$$

for some function h.

To train a VAE, we need to optimize

$$\mathcal{L}(\theta, \phi; \mathbf{x}_i) = E_{q_\phi(\mathbf{z}|\mathbf{x}_i)} \left[\log P(\mathbf{x}_i \mid \mathbf{z})\right] - D_{KL} \left(q_\phi(\mathbf{z} \mid \mathbf{x}_i) \| P(\mathbf{z})\right)$$

For this, we need to calculate a gradient of the form

$$\nabla_\phi E_{q_\phi(\mathbf{z}|\mathbf{x}_i)} \left[h(\mathbf{z})\right]$$

for some function h. Note that exact calculation of the gradient is intractable.

Can we use Monte Carlo to approximate $\nabla_\phi E_{q_\phi(\mathbf{z}|\mathbf{x}_i)}[h(\mathbf{z})]$?

Can we use Monte Carlo to approximate $\nabla_\phi E_{q_\phi(\mathbf{z}|\mathbf{x}_i)}[h(\mathbf{z})]$?

This would mean we are drawing samples from a distribution **whose parameters we are trying to optimize.**

Can we use Monte Carlo to approximate $\nabla_\phi E_{q_\phi(\mathbf{z}|\mathbf{x}_i)}[h(\mathbf{z})]$?

This would mean we are drawing samples from a distribution **whose parameters we are trying to optimize.**

Solution: Reparameterization trick

Reparameterization trick

Recall that the encoder network f uses a neural network to spit out parameters of a Gaussian:

$$q_\phi(\mathbf{z} \mid \mathbf{x}_i) = \mathcal{N}(f_\phi^\mu(\mathbf{x}_i), f_\phi^\Sigma(\mathbf{x}_i))$$

Sampling $\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{x}_i)$ is equivalent to the following:

$$\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$$
$$\mathbf{z} = \sqrt{f_\phi^\Sigma(\mathbf{x}_i)}\boldsymbol{\epsilon} + f_\phi^\mu(\mathbf{x}_i)$$

Note: $f_\phi^\Sigma(\mathbf{x}_i)$ is a diagonal covariance matrix, so the above square root is taken element-wise.

Reparameterization trick

This removes the parameters from the sampling distribution!

We can approximate

$$\nabla_\phi E_{q_\phi(\mathbf{z}|\mathbf{x}_i)}\left[h(\mathbf{z})\right] \approx \frac{1}{L}\sum_{l=1}^{L}\left[\nabla_\phi h\left(\sqrt{f_\phi^\Sigma(\mathbf{x}_i)}\boldsymbol{\epsilon}_l + f_\phi^\mu(\mathbf{x}_i)\right)\right]$$

Reparameterization Trick

The reparameterization trick allows backpropagatation through (or around) samples from a random distribution.

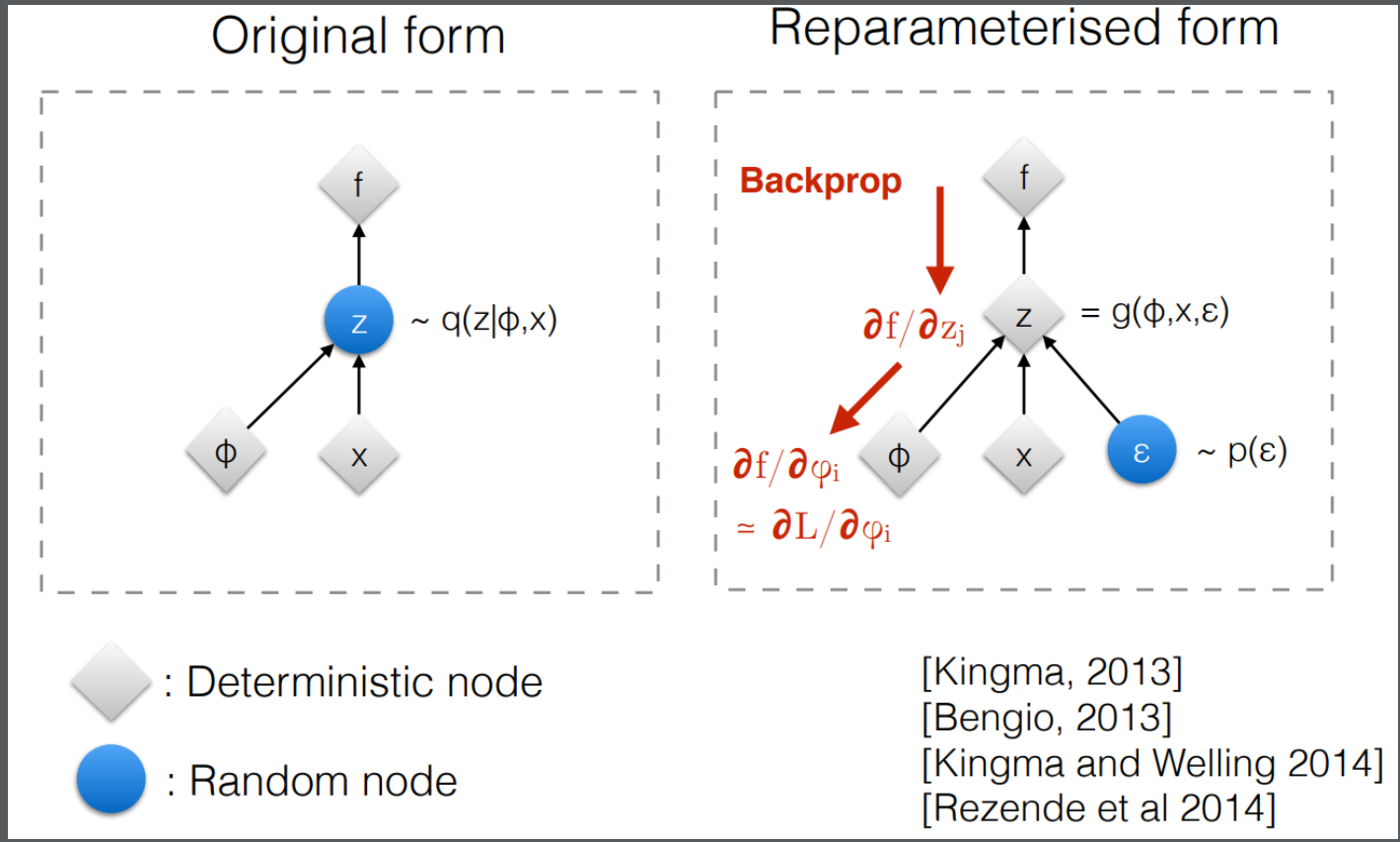


Figure: Source: Kingma, 2015

$$\mathcal{L}(\theta, \phi; \mathbf{x}_i) = \underbrace{E_{q_\phi(\mathbf{z}|\mathbf{x}_i)}\left[\log P(\mathbf{x}_i \mid \mathbf{z})\right]}_{\text{Expected reconstruction quality}} - \underbrace{D_{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}_i) \| P(\mathbf{z})\right)}_{\text{Divergence between approx posterior and the prior}}$$

$$\mathcal{L}(\theta, \phi; \mathbf{x}_i) = \underbrace{E_{q_\phi(\mathbf{z}|\mathbf{x}_i)}\left[\log P(\mathbf{x}_i \mid \mathbf{z})\right]}_{\text{Expected reconstruction quality}} - \underbrace{D_{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}_i) \| P(\mathbf{z})\right)}_{\text{Divergence between approx posterior and the prior}}$$

To train a VAE, we optimize a Monte Carlo estimate of the expectation via the reparameterization trick.

$$\mathcal{L}(\theta, \phi; \mathbf{x}_i) = \underbrace{E_{q_\phi(\mathbf{z}|\mathbf{x}_i)}\left[\log P(\mathbf{x}_i \mid \mathbf{z})\right]}_{\text{Expected reconstruction quality}} - \underbrace{D_{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}_i) \| P(\mathbf{z})\right)}_{\text{Divergence between approx posterior and the prior}}$$

To train a VAE, we optimize a Monte Carlo estimate of the expectation via the reparameterization trick.

In some cases, $D_{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}_i) \| P(\mathbf{z})\right)$ is analytically tractable and we don't need to use Monte Carlo.

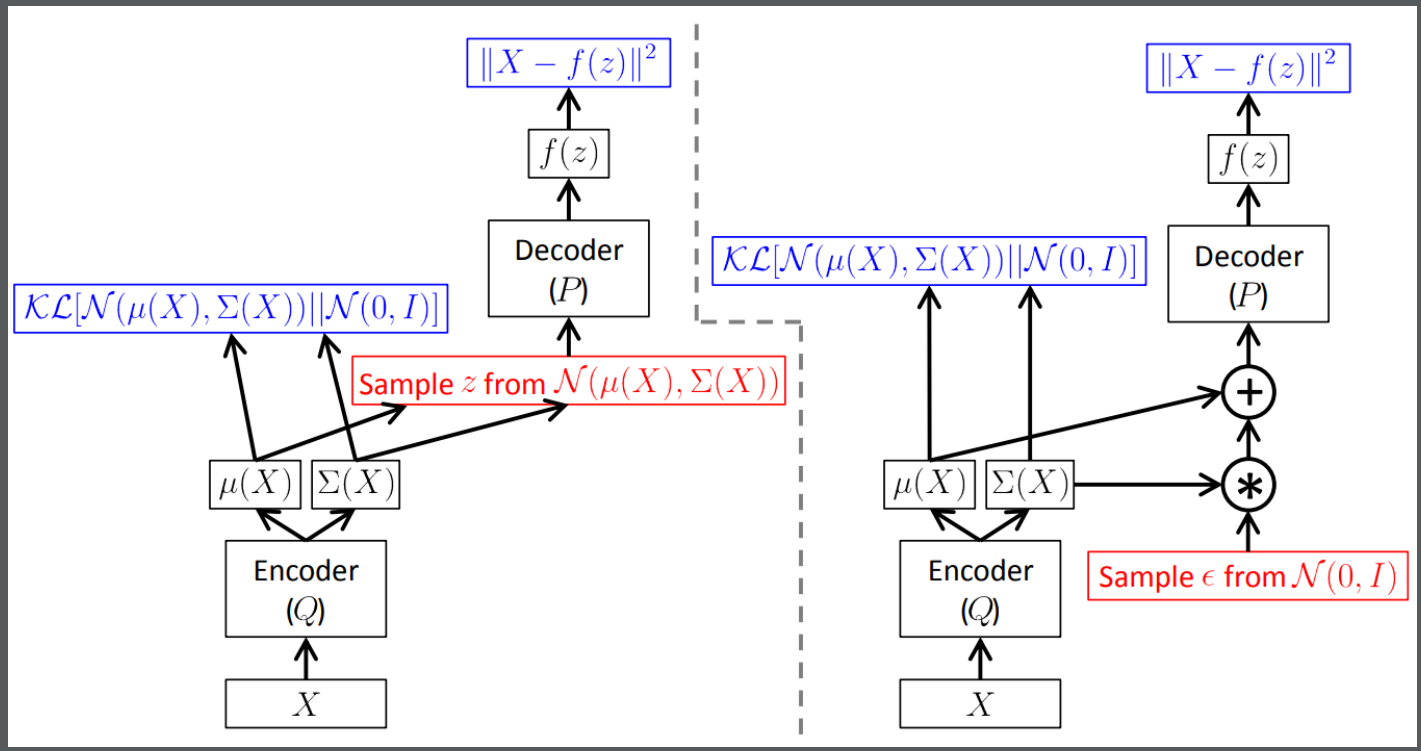


Figure: A training-time variational autoencoder implemented as a feedforward neural network, where $P(X|z)$ is Gaussian. Left is without the reparameterization trick, and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior of these networks is identical, but backpropagation can be applied only to the right network. Source: Doersch, 2016

Tensorflow tutorials on VAEs:

- Using standard tensorflow
- Using tensorflow probability

Auto-Encoding Variational Bayes

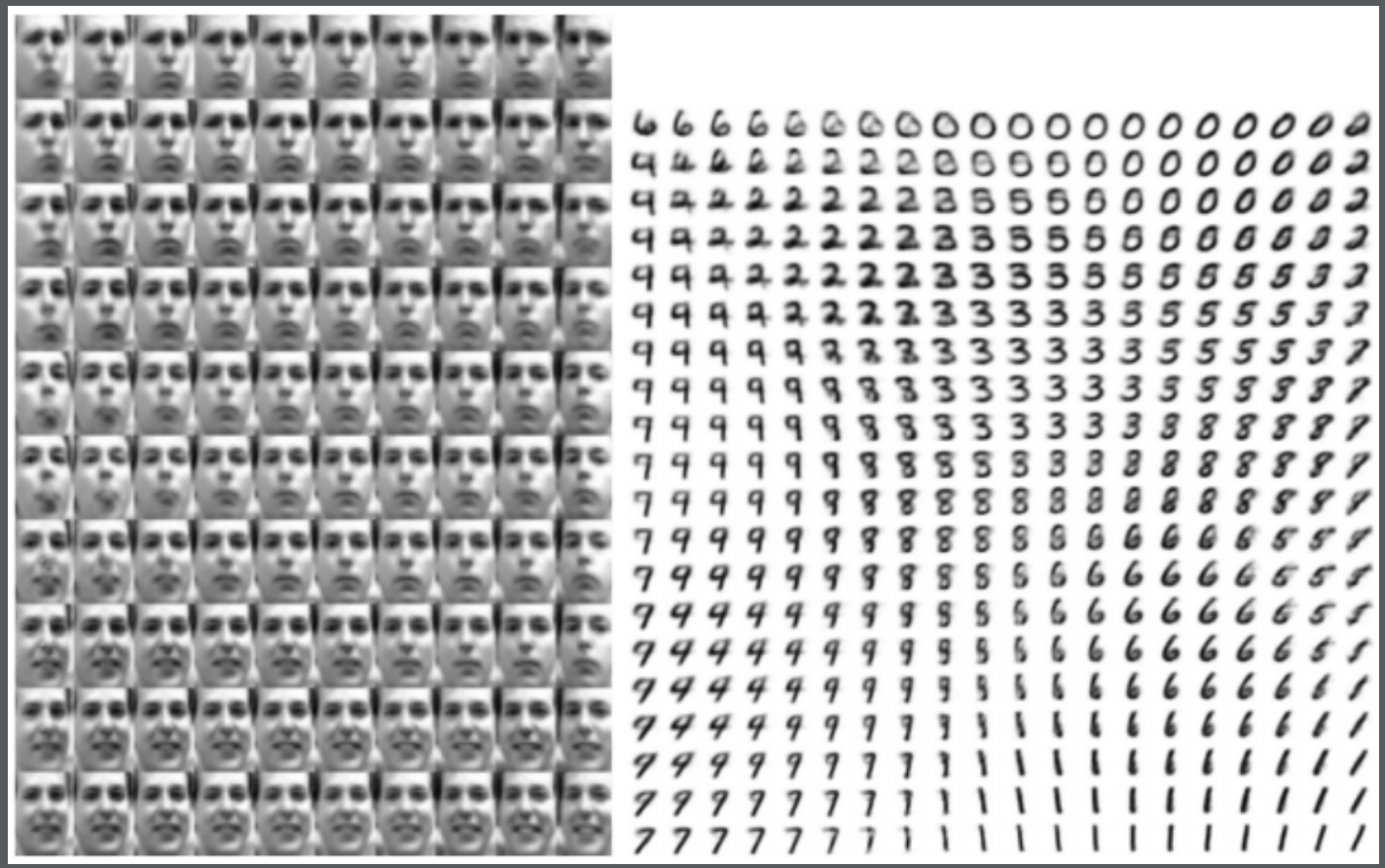


Figure: Examples of 2-D coordinate systems for high-dimensional manifolds, learned by a variational autoencoder. Source: Kingma and Welling, 2013