Modern Generative Models: Restricted Boltzmann Machines

Based on presentation by Hung Chao
https://people.cs.pitt.edu/~milos/courses/cs3750/lectures/class22.pdf

Unsupervised Learning: use only the inputs \( x^{(i)} \) for learning
- automatically extract meaningful features for data
- Leverage the availability of unlabeled data
- Can use negative log-likelihood to learn the underlying feature

We will see 2 neural networks for unsupervised learning
- Restricted Boltzmann Machines
- Variational Autoencoders
**GENERATIVE MODELS**

- Given training data, we want to generate new samples from the same distribution

$$\text{Training data } \sim p_{\text{data}}(x)$$

$$p_{\text{model}}(x)$$

**Generative samples ~**

$$p_{\text{model}}(x)$$

Want to learn $$p_{\text{model}}(x)$$ similar to $$p_{\text{data}}(x)$$.

**Figure source: CIFAR-10 dataset (Krizhevsky and Hinton, 2009)**

---

**GENERATIVE MODELS**

- Why generative models?

  - Realistic samples for artwork, super-resolution, colorization, etc.

  - Generative models of time-series data can be used for simulation and planning (reinforcement learning applications!)

  - Training generative models can also enable inference of latent representation that can be useful as general features

**Figure source: Internet**
Many interesting theoretical results about undirected models depend on the assumption that $\forall x, \tilde{p}(x) > 0$. A convenient way to enforce this condition is to use an energy-based model where

$$\tilde{p}(x) = \exp(-E(x))$$

- Normalized probability

$$p(x) = \frac{1}{Z} \tilde{p}(x)$$

- $E(x)$ is known as the energy function

Any distribution of this form is an example of a Boltzmann distribution. For this reason, many energy-based models are called Boltzmann machines.

E.g. $E(a, b, c, d, e, f)$ can be written as

$$E_{a,b} + E_{b,c} + E_{c,d} + E_{d,e} + E_{e,f}$$

**RESTRICTED BOLTZMANN MACHINE**

- Restricted Boltzmann machines (RBMs) are undirected probabilistic graphical models containing a layer of observable variables and a single layer of latent variables.
- RBM is a bipartite graph, with no connections permitted between any variables in the observed layer or between any units in the latent layer.
Energy function: 
\[ E(x, h) = -h^T W x - c^T x - b^T h \]
\[ = - \sum_j \sum_k W_{jk} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j \]

Distribution: 
\[ p(x, h) = \exp(-E(x, h))/Z \]

The notation based on an energy function is simply an alternative to the representation as the product of factors.
RESTRICTED BOLTZMANN MACHINE

Markov network view

\[ p(x, h) = \frac{1}{Z} \prod_j \prod_k \exp(W_{j,k}h_jx_k) \]

\[ = \prod_k \exp(c_kx_k) \]

\[ \prod_j \exp(b_jh_j) \]

Unary Factors

The scalar visualization is more informative of the structure within the vectors

INFEERENCE

Conditional Distribution:

\[ p(h|x) = \prod_j p(h_j|x) \]

\[ p(h_j = 1|x) = \frac{1}{1 + \exp(-(h_j + W_j \cdot x))} \]

\[ = \text{sigm}(b_j + W_j \cdot x) \]

\[ p(x|h) = \prod_k p(x_k|h) \]

\[ p(x_k = 1|h) = \frac{1}{1 + \exp(-(c_k + h^T W_k - b_k))} \]

\[ = \text{sigm}(c_k + h^T W_k) \]
\[ p(h|x) = \frac{1}{Z} \exp \left( h^T W x + b^T h \right) \]

\[ Z = \exp \left( \sum_{h=1}^{H} h^T W x + b^T h \right) \]

\[ p(h_j = 1|x) = \frac{\exp (b_j + W^j x)}{1 + \exp (-b_j - W^j x)} = \text{sigmoid} (b_j + W^j x) \]
FREE ENERGY

What about $p(x)$?

$$p(x) = \sum_{h \in \{0,1\}^n} p(x, h) = \sum_{h \in \{0,1\}^n} \exp(-E(x, h)) / Z$$

$$= \exp \left( c^T x + \sum_{j=1}^H \log (1 + \exp (h_j + W_j x)) \right) / Z$$

$$= \exp(-F(x)) / Z$$

Free Energy

$$p(x) = \sum_{h \in \{0,1\}^n} \exp (h^T W x + c^T x + b^T h) / Z$$

$$= \exp (c^T x) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_H \in \{0,1\}} \exp \left( \sum_j h_j W_j x + b_j h_j \right) / Z$$
\[ p(x) = \sum_{h \in \{0,1\}^H} \exp \left( h^\top W x + c^\top x + b^\top h \right) / Z \]

\[ = \exp \left( c^\top x \right) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_H \in \{0,1\}} \exp \left( \sum_{j} h_j W_{1:j} x + b_j h_j \right) / Z \]

\[ = \exp \left( c^\top x \right) \left( \sum_{h_1 \in \{0,1\}} \exp \left( h_1 W_{1} x + b_1 h_1 \right) \right) \cdots \left( \sum_{h_H \in \{0,1\}} \exp \left( h_H W_{H} x + b_H h_H \right) \right) / Z \]

\[ = \exp \left( c^\top x \right) \left( 1 + \exp \left( b_1 + W_{1} \cdot x \right) \right) \cdots \left( 1 + \exp \left( b_H + W_{H} \cdot x \right) \right) / Z \]
FREE ENERGY

\[ p(x) = \exp \left( c^\top x + \sum_{j=1}^{H} \log \left( 1 + \exp \left( b_j + W_j x \right) \right) \right) / Z \]

= \exp \left( c^\top x + \sum_{j=1}^{H} \text{softplus} \left( b_j + W_j x \right) \right) / Z

Bias of each feature
Bias the probability of x
Feature expected in x

Training

Training objective

To train an RBM, we minimize the average negative log-likelihood (NLL)

\[ \frac{1}{T} \sum_t l \left( f \left( x^{(t)} \right) \right) = \frac{1}{T} \sum_t - \log p \left( x^{(t)} \right) - \log p \left( z^{(t)} \right) - \log \left( \sum_{z} p \left( x^{(t)}, z \right) \right) = \log \left( \sum_{z} \exp \left( - E \left( x^{(t)}, z \right) \right) / Z \right) \]

We’d like to proceed by stochastic gradient descent

\[ \frac{\partial - \log p \left( x^{(t)} \right)}{\partial \theta} = E_h \left[ \frac{\partial E \left( x^{(t)}, h \right)}{\partial \theta} \right] | x^{(t)} - E_{x,h} \left[ \frac{\partial E(x,h)}{\partial \theta} \right] \]

Positive Phase
Negative Phase
Training

**Training objective**

To train an RBM, we minimize the average negative log-likelihood (NLL)

\[
\frac{1}{T} \sum_{t=1}^{T} - \log p(x^{(t)})
\]

We’d like to proceed by stochastic gradient descent

\[
\frac{\partial - \log p(x^{(t)})}{\partial \theta} = E_{\theta} \left[ \frac{\partial E(x^{(t)}, h)}{\partial \theta} \right] - E_{\theta,h} \left[ \frac{\partial E(x, h)}{\partial \theta} \right]
\]

Contrastive Divergence (CD)

(Hinton, Neural Computation, 2002)

**Idea:**
1. obtain the point \( \tilde{x} \) by Gibbs sampling
2. start sampling chain at \( x^{(t)} \)
3. replace the expectation by a point estimate at \( \tilde{x} \)

Often called negative sample
Contrastive Divergence (CD)  
(Hinton, Neural Computation, 2002)

Replace the expectation by a point estimate

\[
\frac{\partial \log p (x^{(t)})}{\partial \theta} = E_h \left[ \frac{\partial E (x^{(t)}, h) | x^{(t)}}{\partial \theta} \right] - E_{x,h} \left[ \frac{\partial E (x, h)}{\partial \theta} \right]
\]

\[
E_h \left[ \frac{\partial E (x^{(t)}, h) | x^{(t)}}{\partial \theta} \right] \approx \frac{\partial E (x^{(t)}, \hat{h}(t))}{\partial \theta}
\]

\[
E_{x,h} \left[ \frac{\partial E (x, h)}{\partial \theta} \right] \approx \frac{\partial E (\hat{x}, \hat{h})}{\partial \theta}
\]
Parameter Update

Derivation of $\frac{\partial E(x, h)}{\partial \theta}$ for $\theta = W_{jk}$

$$\frac{\partial E(x, h)}{\partial W_{jk}} = \frac{\partial}{\partial W_{jk}} \left( -\sum_j W_{jk} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j \right)$$

$$= -\frac{\partial}{\partial W_{jk}} \sum_j W_{jk} h_j x_k$$

$$= -h_j x_k$$

$$\nabla_W E(x, h) = -hx^T$$
Parameter Update

Derivation of \( E_h \left[ \frac{\partial E(x, h)}{\partial \theta} | x \right] \) for \( \theta = W_{jk} \)

\[
E_h \left[ \frac{\partial E(x, h)}{\partial W_{jk}} | x \right] = E_h \left[ -h_j x_k | x \right] = \sum_{h_j \in \{0, 1\}} -h_j x_k \pi(h_j | x) \]

If we define:

\[
h(x) = \begin{pmatrix} p(h_1 = 1 | x) \\ \vdots \\ p(h_M = 1 | x) \end{pmatrix}
\]

\[
= \text{sign}(b + Wx)
\]

Then,

\[
E_h [\nabla_w E(x, h) | x] = -h(x)x^T
\]

Parameter Update

Update of \( W \)

Given \( x^{(t)} \) and \( \tilde{x} \), the learning rule of \( \theta = W \) becomes:

\[
W \leftarrow W - \alpha \left( \nabla_w - \log p(x^{(t)}) \right)
\]

\[
\leftarrow W - \alpha \left( E_h \left[ \nabla_w E \left( x^{(t)}, h \right) | x^{(t)} \right] - E_{x,h} \left| \nabla_w E(x, h) \right| \right)
\]

\[
\leftarrow W - \alpha \left( E_h \left[ \nabla_w E \left( x^{(t)}, h \right) | x^{(t)} \right] - E_h \left| \nabla_w E(x, h) | x \right| \right)
\]

\[
\leftarrow W + \alpha \left( h \left( x^{(t)} \right) \tilde{x} - h(\tilde{x}) \tilde{x}^T \right)
\]
CD-K: PSEUDOCODE

Contrastive Divergence:

1. For each training sample $\mathbf{x}^{(t)}$
   i. Generate a negative sample $\tilde{\mathbf{x}}^{(t)}$ using k steps of Gibbs sampling, starting at $\mathbf{x}^{(t)}$
   ii. Update parameters:

   $\mathbf{W} \leftarrow \mathbf{W} + \alpha \left( \mathbf{h}(\mathbf{x}^{(t)}) \mathbf{x}^{(t)\top} - \mathbf{h}(\tilde{\mathbf{x}}) \tilde{\mathbf{x}}\top \right)$
   $\mathbf{b} \leftarrow \mathbf{b} + \alpha \left( \mathbf{h}(\mathbf{x}^{(t)}) - \mathbf{h}(\tilde{\mathbf{x}}) \right)$
   $\mathbf{c} \leftarrow \mathbf{c} + \alpha \left( \mathbf{x}^{(t)} - \tilde{\mathbf{x}} \right)$

2. Go back to 1. until stopping criteria is reached

Contrastive Divergence-k

Contrastive Divergence:

- CD-k: contrastive divergence with k iterations of Gibbs sampling
- In general, the bigger k is, the less biased the estimate of the gradient will be
- In practice, k=1 works well for pre-training
Software

• Sci-kit learn
• Pydbm
  https://pypi.org/project/pydbm/
• Other self-implemented versions on github

References


[2] Hung Chao’s CS 3750 year 2018 slides
  https://people.cs.pitt.edu/~milos/courses/cs3750/lectures/class22.pdf


Modern Generative Models: Variational Autoencoders

Jun Luo
02/2020

Given training data, we want to generate new samples from the same distribution.

Training data $\sim p_{\text{data}}(x)$
Generated samples $\sim p_{\text{model}}(x)$

Want to learn $p_{\text{model}}(x)$ similar to $p_{\text{data}}(x)$.

Source: CIFAR-10 dataset (Krizhevsky and Hinton, 2009)
**GENERATIVE MODELS**

- Why generative models?
  - Realistic samples for artwork, super-resolution, colorization, etc.
  - Generative models of time-series data can be used for simulation and planning (reinforcement learning applications!)
  - Training generative models can also enable inference of latent representation that can be useful as general features.

**Autoencoders (Recap)**

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data.

- $z$ usually smaller than $x$ (dimensionality reduction)
- Features vector are generally shorter than input vector to extract meaningful features
- Originally: Linear + nonlinearity (sigmoid)
- Later: Deep, fully-connected
- Later: ReLU CNN

Figure source: Internet
Autoencoders (Recap)

How to learn this feature representation?
Train such that features can be used to reconstruct original data
"Autoencoding" - encoding itself

- **Original step:** Linear + nonlinearity (sigmoid)
- **Later step:** Deep, fully-connected

Later step uses ReLU CNN for encoding.

How to learn this feature representation?
Train such that features can be used to reconstruct original data
"Autoencoding" - encoding itself

Reconstructed data

Features

Encoder

Input data

Decoder

Figure adapted from CS 231n
Autoencoders (Recap)

Train such that features can be used to reconstruct original data

Use L2 Loss Function \( \| x - \hat{x} \|^2 \)

Encoder

Input data

Features

Reconstructed data

Decoder

Reconstructed data

Input data

Use L2 Loss Function \( \| x - \hat{x} \|^2 \)

Does not need labels

Encoder: 4-layer conv
Decoder: 4-layer upconv

Encoder: 4-layer conv
Decoder: 4-layer upconv

Figure adapt from CS 231n
Autoencoders (Recap)

Encoder can be used to initialize a **supervised** model.

- **Input data** $x$
- **Encoder** $z$
- **Features** $z$
- **Decoder** $\hat{x}$
- **Reconstructed data** $\hat{x}$

**Throw away decoder**

- **Input data** $x$
- **Encoder** $z$
- **Features** $z$
- **Encoder** $x$
Autoencoders (Recap)

Encoder can be used to initialize a **supervised** model.

Train for final task (sometimes with small data).

Throw away decoder after training with reconstruction loss.

Add label and new loss function (e.g. softmax loss).

Fine-tune encoder jointly with classifier.

Train for final task (sometimes with small data).

Autoencoders can reconstruct data, and can learn features to initialize a supervised model.
Autoencoders (Recap)

Autoencoders can reconstruct data, and can learn features to initialize a supervised model.

Reconstructed data

Features

Encoder

Input data

Features capture factors of variation in training data. Can we generate new images from an autoencoder?

Variational Autoencoders

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Assume training data \( \{x^{(i)}\}_{i=1}^N \) is generated from underlying unobserved (latent) representation \( z \).

Sample from true prior \( p_\theta^\ast(z) \)

Sample from true conditional \( p_\theta^\ast(x|z^{(i)}) \)

Intuition (remember from autoencoders!): \( x \) is an image, \( z \) is latent factors used to generate
Variational Autoencoders

We want to estimate the true parameters $\theta^*$ of this generative model.

How should we represent this model?

Choose prior $p(z)$ to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

Sample from true prior $p_Z(z)$

Sample from true conditional $p_{X|Z}(x|z)$

Decoder network

Conditional $p(x|z)$ is complex (generates image) => represent with neural network
**Variational Autoencoders**

We want to estimate the true parameters $\theta^*$ of this generative model.

Sample from true conditional $p_{\theta^*}(x|z^{(0)})$

Sample from true prior $p_{\theta^*}(z)$

How to train the model?

Decoder network

Data likelihood

$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

Similar as Restricted Boltzman Machines, here, we maximize the data likelihood.

**Variational Autoencoders**

Data likelihood:

$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

Simple Gaussian Prior

Decoder neural network
Variational Autoencoders

Data likelihood: 
\[ p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz \]

Intractable to compute \( p(x|z) \) for every \( z \)

Variational Autoencoders

Data likelihood: 
\[ p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz \]

Posterior density also intractable: 
\[ p_{\theta}(z|x) = \frac{p_{\theta}(x|z)p_{\theta}(z)}{p_{\theta}(x)} \]

Because of intractable data likelihood
Variational Autoencoders

Data likelihood: \[ p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz \]

Posterior density also intractable: \[ p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x) \]

Solution: In addition to decoder network modeling \( p_\theta(x|z) \), define additional encoder network \( q_\phi(z|x) \) that approximates \( p_\theta(z|x) \)

Will see that this allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize

Variational Autoencoders

Since we’re modeling probabilistic generation of data, encoder and decoder networks are probabilistic

Encoder network \( q_\phi(z|x) \) (parameters \( \phi \))
Decoder network \( p_\theta(x|z) \) (parameters \( \theta \))

\[ \mu_{z|x}, \Sigma_{z|x} \quad \mu_{x|z}, \Sigma_{x|z} \]
Variational Autoencoders

Since we’re modeling probabilistic generation of data, encoder and decoder networks are probabilistic.

Sample $z|x$ from $z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$

Sample $x|z$ from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$

Encoder network $q_\phi(z|x)$ (parameters $\phi$)

Decoder network $p_\theta(x|z)$ (parameters $\theta$)

Taking expectation over $z$ (using encoder network) will be helpful later on.

Variational Autoencoders

Now let us see the (log) data likelihood again with encoder and decoder:

$$\log p(x) = \mathbb{E}_{z \sim q_\phi(x)} \left[ \log p_{\theta}(x|z) \right]$$

($p_{\theta}(x|z)$ does not depend on $z$.)
Variational Autoencoders

Now let us see the (log) data likelihood again with encoder and decoder

\[
\log p_d (x^{(i)}) = E_{x \sim p(x|z^{(i)})} \left[ \log p_d (x^{(i)}) \right] \quad (p_d (x^{(i)}) \text{Does not depend on } Z)
\]

\[
= E_z \left[ \log \frac{p_d (x^{(i)}|z^{(i)}) p_d (z)}{p_d (z|x^{(i)})} \right] \quad \text{(Bayes' rule)}
\]
Variational Autoencoders

Now let us see the (log) data likelihood again with encoder and decoder
\[
\log p_{\text{data}}(x^{(i)}) = E_{z \sim q(z|x^{(i)})} \left[ \log p_{\text{data}}(x^{(i)}) \right] \quad (p_{\text{data}}(x^{(i)}) \text{Does not depend on } Z)
\]
\[
= E_{z} \left[ \log \frac{p_{\text{data}}(x^{(i)}|z)}{p_{\text{data}}(z|x^{(i)})} \right] \quad (\text{Bayes' rule})
\]
\[
= E_{z} \left[ \log \frac{p_{\text{data}}(x^{(i)}|z)}{p_{\text{data}}(z|x^{(i)})} \right] + E_{z} \left[ \log \frac{q_{z}(z|x^{(i)})}{p_{\text{data}}(z|x^{(i)})} \right] \quad (\text{Multiply by 1})
\]
\[
= E_{z} \left[ \log p_{\text{data}}(x^{(i)}|z) \right] - E_{z} \left[ \log \frac{q_{z}(z|x^{(i)})}{p_{\text{data}}(z|x^{(i)})} \right] + E_{z} \left[ \log \frac{q_{z}(z|x^{(i)})}{p_{\text{data}}(z|x^{(i)})} \right] \quad (\text{Logarithm})
\]
Variational Autoencoders

Now let us see the (log) data likelihood again with encoder and decoder

\[
\log p(x) = E_{z \sim q(z|x)} \left[ \log p(x|z) \right] = E_z \left[ \log \frac{p(x|z) q(z|x)}{p(z|x)} \right]
\]

(Bayes' rule)

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!

\[
z = \mu + \sigma \odot \epsilon
\]

Decoder network gives \(p(x|z)\) can compute estimate of this term through sampling. (Sampling differentiable through reparam. trick, see paper.)

\(p(x|z)\) intractable (saw earlier), can't compute this KL term. But we know KL divergence always \(\geq 0\).

\[\log p(x) - E_{z \sim q(z|x)} \left[ \log p(x|z) \right] \text{ (Multiply by 1)}\]

\[= E_z \left[ \log \frac{p(x|z) q(z|x)}{p(z|x)} \right] + E_z \left[ \log \frac{q(z|x)}{p(z|x)} \right] \text{ (Logarithm)}\]

\[= E_z \left[ \log p(x|z) \right] - D_{KL} \left( q_\theta \mid \mid p_\theta \right) + D_{KL} \left( q_\theta \mid \mid p_\theta \right) \geq 0\]

Tractable lower bound which we can take gradient of and optimize! \(p(x|z)\) differentiable, KL term differentiable.)
Variational Autoencoders

Now let us see the (log) data likelihood again with encoder and decoder

\[ \log p_x(x^{(i)}) = \mathbb{E}_{x\sim q_x(z|x^{(i)})} \left[ \log p_x(x^{(i)}|z) \right] \quad (p_x(x^{(i)}) \text{ does not depend on } \mathcal{Z}) \]

\[ = \mathbb{E}_z \left[ \log \frac{p_x(x^{(i)}|z) p_x(z)}{p_x(z|x^{(i)})} \right] \quad (\text{Bayes' rule}) \]

\[ = \mathbb{E}_z \left[ \log \frac{p_x(x^{(i)}|z) p_x(z)}{p_x(z|x^{(i)})} q_x(z|x^{(i)}) \right] \quad (\text{Multiply by 1}) \]

\[ = \mathbb{E}_z \left[ \log \frac{p_x(x^{(i)}|z) p_x(z)}{p_x(z|x^{(i)})} \right] - \mathbb{E}_z \left[ \log q_x(z|x^{(i)}) \right] + \mathbb{E}_z \left[ \log \frac{q_x(z|x^{(i)})}{p_x(z|x^{(i)})} \right] \quad (\text{Logarithm}) \]

\[ = \mathbb{E}_z \left[ \log p_x(x^{(i)}|z) - D_{KL} \left( q_x(z|x^{(i)}) \parallel p(z|x^{(i)}) \right) + D_{KL} \left( q_x(z|x^{(i)}) \parallel p(y(z)) \right) \right] \]

\[ \log p_x(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi) \quad \mathcal{L}(x^{(i)}, \theta, \phi) \geq 0 \]

**Variational Lower Bound**

\[ \theta^*, \phi^* = \arg \max \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi) \]

**Training:** Maximize lower bound

---

Variational Autoencoders

Now let us see the (log) data likelihood again with encoder and decoder

\[ \log p_x(x^{(i)}) = \mathbb{E}_{x\sim p_x(z|x^{(i)})} \left[ \log p_x(x^{(i)}|z) \right] \quad (p_x(x^{(i)}) \text{ does not depend on } \mathcal{Z}) \]

\[ = \mathbb{E}_z \left[ \log \frac{p_x(x^{(i)}|z) p_x(z)}{p_x(z|x^{(i)})} \right] \quad (\text{Bayes' rule}) \]

\[ = \mathbb{E}_z \left[ \log \frac{p_x(x^{(i)}|z) p_x(z)}{p_x(z|x^{(i)})} q_x(z|x^{(i)}) \right] \quad (\text{Multiply by 1}) \]

\[ = \mathbb{E}_z \left[ \log \frac{p_x(x^{(i)}|z) p_x(z)}{p_x(z|x^{(i)})} \right] - \mathbb{E}_z \left[ \log q_x(z|x^{(i)}) \right] + \mathbb{E}_z \left[ \log \frac{q_x(z|x^{(i)})}{p_x(z|x^{(i)})} \right] \quad (\text{Logarithm}) \]

\[ = \mathbb{E}_z \left[ \log p_x(x^{(i)}|z) - D_{KL} \left( q_x(z|x^{(i)}) \parallel p(z|x^{(i)}) \right) + D_{KL} \left( q_x(z|x^{(i)}) \parallel p(y(z)) \right) \right] \]

\[ \log p_x(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi) \quad \mathcal{L}(x^{(i)}, \theta, \phi) \geq 0 \]

**Variational Lower Bound**

\[ \theta^*, \phi^* = \arg \max \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi) \]

**Training:** Maximize lower bound

---

**Reconstruct the input data**

Make approximate posterior distribution close to prior
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

\[ \mathbb{E}_z \left[ \log p(x|z) \right] - D_{KL} (q(z|x) \| p(z)) \]

Let’s look at computing the bound (forward pass) for a given minibatch of input
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

\[ \mathbb{E}_q[\log p_{\theta}(x|z)] - \mathbb{D}_KL (q_{\phi}(z|x) \parallel p(z)) \]

Encoder network

\[ q_{\phi}(z|x) \quad \text{(parameters } \phi) \]

Input data

Make approximate posterior distribution close to prior

Sample \( z|x \sim \mathcal{N}(\mu_{xz}, \Sigma_{xz}) \)

Encoder network

\[ q_{\phi}(z|x) \quad \text{(parameters } \phi) \]

Input data
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

\[ \mathbb{E}_q [ \log p (x | z) ] - D_{KL} ( q \| p^{(z)}) \]

Input data

Encoder network

\( q_\phi (z | x) \) (parameters \( \phi \))

Sample \( z \mid x \sim \mathcal{N} (\mu_{1|x}, \Sigma_{z|x}) \)

Make approximate posterior distribution close to prior

Decoder network

\( p_\theta (x | z) \) (parameters \( \theta \))

Sample \( x \mid z \sim \mathcal{N} (\mu_{1|z}, \Sigma_{x|z}) \)

Maximize likelihood of original input being reconstructed

Output data

\( \hat{x} \)
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

\[
\mathbb{E}_q \left[ \log p (x | z) \right] - \mathbb{D}_{KL} \left( q (z | x) \parallel p (z) \right)
\]

Input data

Encoder network

Decoder network (parameters \( \theta \))

Sample \( z \sim \mathcal{N} (\mu_{z|x}, \Sigma_{z|x}) \)

Make approximate posterior distribution close to prior

For every minibatch of input data: compute this forward pass, and then backprop!

Maximize likelihood of original input being reconstructed

VAE: Generate data

Use decoder network. Now sample \( z \) from prior!

Sample \( x|z \sim \mathcal{N} (\mu_{x|z}, \Sigma_{x|z}) \)

Decoder network

Decoder network (parameters \( \theta \))

Sample \( z \) from \( z \sim \mathcal{N} (\mu_{z|x}, \Sigma_{z|x}) \)

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014
VAE: Generate data

Use decoder network. Now sample z from prior!

Sample $x|z$ from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$

Decoder network
$p_{\theta}(x|z)$
(parameters $\theta$)

Sample $z|x$ from $z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$
Use decoder network. Now sample $z$ from prior!

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Different dimensions of $z$ encode interpretable factors of variation

Diagonal prior on $z$ $\rightarrow$ independent latent variables

Degree of smile

$\text{Vary } Z_1$

$\text{Vary } Z_2$

Head pose

$\text{Also good feature representation that can be computed using } q_\phi(z|x)$
Softwares

- Vae – PyPI
  - https://pypi.org/project/vae/
- Deep learning platforms such as TensorFlow and PyTorch

References


[2] CS 3750 Previous slides

