Introduction to Deep Learning

Autoencoders

Andres Mendez-Vazquez

June 22, 2025

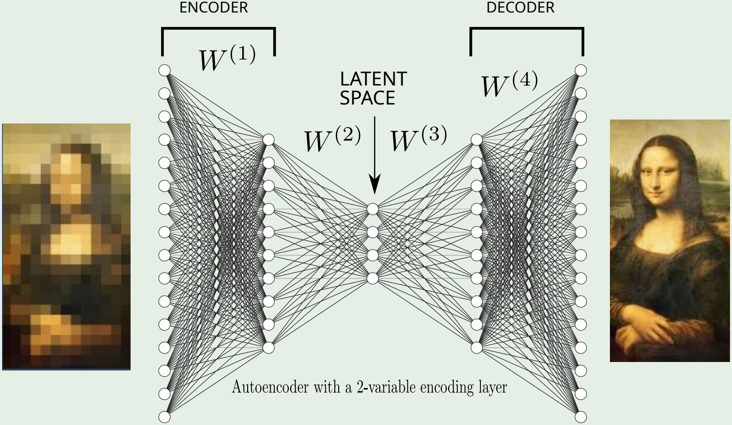
Outline

Outline

Introduction

From [1]

- "An autoencoder is a specific type of a neural network, which is mainly designed to encode the input into a compressed and meaningful representation, and then decode it back such that the reconstructed input is similar as possible to the original one."



Autoencoder with a 2-variable encoding layer

At the following work [2]

Definition

- An $n/p/n$ autonecoder is defined as a t-tuple $n, p, m, \mathbb{F}, \mathbb{G}, \mathcal{A}, \mathcal{B}, \mathcal{X}, \Delta$ where
  1. $\mathbb{F}$ and $\mathbb{G}$ are sets.
  2. n and p are positive integers with $0 < p < n$.
  3. $\mathcal{A}$ is a class of functions from $\mathbb{G}^p$ to $\mathbb{F}^n$.
  4. $\mathcal{B}$ is a class of functions from $\mathbb{F}^n$ to $\mathbb{G}^p$.
  5. $\mathcal{X} = \{x_1, ..., x_m\}$ is a set of m (training) vectors in $\mathbb{F}^n$.
  6. Δ is a dissimilarity or distortion function over $\mathbb{F}^n$.

Basically

We have that

- For any $A \in \mathcal{A}$ and $B \in \mathcal{B}$, the autoencoder transforms an input vector $x \in F^n$ into an output vector$A \circ B(x) \in F^n$

Basically

We have that

- For any $A \in \mathcal{A}$ and $B \in \mathcal{B}$, the autoencoder transforms an input vector $x \in F^n$ into an output vector $A \circ B(x) \in F^n$

Thus, we have

- The corresponding autoencoder problem is to find $A \in \mathcal{A}$ and $B \in \mathcal{B}$, that minimize the overall distortion function:

$$\min E(A, B) = \min_{A,B} \sum_{t=1}^{m} E(x_t) = \min_{A,B} \sum_{t=1}^{m} \Delta\left(B \circ A(x_t), x_t\right)$$

And there is the other case

We can have another output y_t

$$\min_{A,B} \sum_{t=1}^{m} \Delta\left(B \circ A\left(x_t\right), y_t\right)$$

And there is the other case

We can have another output y_t

$$\min_{A,B} \sum_{t=1}^{m} \Delta\left(B \circ A\left(x_t\right), y_t\right)$$

Not only that

- $p < n$ corresponds to the regime where the autoencoder tries to implement some form of compression or feature extraction.

Baldani and Hornik [3]

In 1989, they proposed an initial linear case no activation functions

$$E\left(A,B\right)=\sum_{1\leq t\leq T}\left\|y_t-BAx_t\right\|$$

Baldani and Hornik [3]

In 1989, they proposed an initial linear case no activation functions

$$E\left(A,B\right)=\sum_{1\leq t\leq T}\|y_t-BAx_t\|$$

There are several properties under these autoencoders

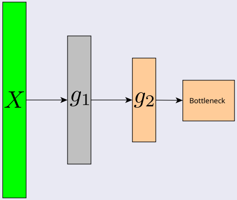
- But somethin interesting an optimal:
  - $E\left(A,B\right)$ is convex in the coefficient of B and attains minimum for B such that $BAA^T\Sigma_{XX}=A^T\Sigma_{XY}$

Outline

Basically, we have

Encoder

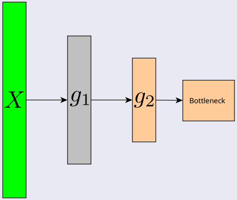
- The encoder is the part of the network which takes in the input and produces a lower Dimensional encoding



Basically, we have

Encoder

- The encoder is the part of the network which takes in the input and produces a lower Dimensional encoding



Something Notable

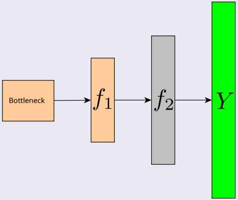
Bottleneck: It is the lower dimensional hidden layer where the encoding is produced.

- Note: Call it an embedding!!!

Then, we have that

Decoder

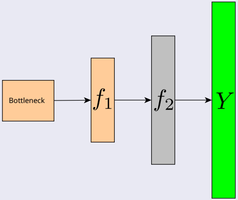
- The decoder takes in the encoding and recreates back the input.



Then, we have that

Decoder

- The decoder takes in the encoding and recreates back the input.

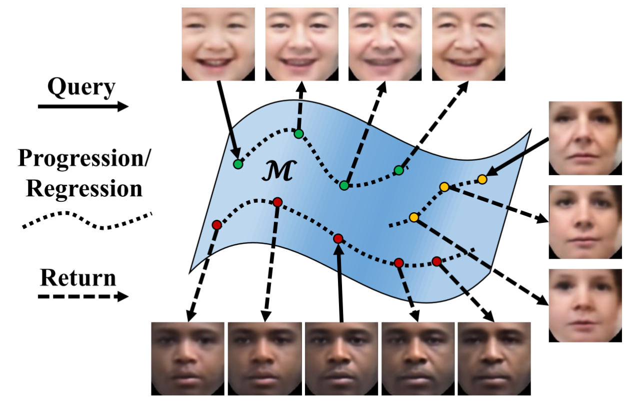


Remember the Transpose Convolution

- Yes, we can use it to implement the decoder.

Basically is Called a Manifold

Mapping x to a small dimension h from old to young



- Thanks to the Pytorch implementation by Mattan Serry, Hila Balahsan, and Dor Alt.

Autoencoders differ from General Data Compression

Autoencoders are data-specific

- i.e., only able to compress data similar to what they have been trained on

Autoencoders differ from General Data Compression

Autoencoders are data-specific

- i.e., only able to compress data similar to what they have been trained on

This is different from, say, MP3 or JPEG compression algorithm

- Which make general assumptions about "sound/images", but not about specific types of sounds/images
- Autoencoder for pictures of cats would do poorly in compressing pictures of trees
  - Because features it would learn would be cat-specific

Autoencoders differ from General Data Compression

Autoencoders are data-specific

- i.e., only able to compress data similar to what they have been trained on

This is different from, say, MP3 or JPEG compression algorithm

- Which make general assumptions about "sound/images", but not about specific types of sounds/images
- Autoencoder for pictures of cats would do poorly in compressing pictures of trees
  - Because features it would learn would be cat-specific

Autoencoders are lossy

- which means that the decompressed outputs will be degraded compared to the original inputs (similar to MP3 or JPEG compression).
- This differs from lossless arithmetic compression

And Actually, Autoncoders are Learn

Learning $g(f(\boldsymbol{x})) = \boldsymbol{x}$ everywhere is not useful

- Actually in Age Autoncoders we want old faces from young

And Actually, Autoncoders are Learn

Learning $g(f(\boldsymbol{x})) = \boldsymbol{x}$ everywhere is not useful

- Actually in Age Autoncoders we want old faces from young

Autoencoders are designed to be unable to copy perfectly

- Autoencoders learn useful properties of the data

And Actually, Autoncoders are Learn

Learning $g(f(\boldsymbol{x})) = \boldsymbol{x}$ everywhere is not useful

- Actually in Age Autoncoders we want old faces from young

Autoencoders are designed to be unable to copy perfectly

- Autoencoders learn useful properties of the data

Autoencoders learn useful properties of the data

- Being forced to prioritize which aspects of input should be copied

An important property

It can learn stochastic mappings

- Go beyond deterministic functions to mappings $p_{encoder}(h|x)$ and $p_{decoder}(x|h)$

An important property

It can learn stochastic mappings

- Go beyond deterministic functions to mappings $p_{encoder}(h|x)$ and $p_{decoder}(x|h)$

Now

- A little bit on the loss function...

Outline

Autoencoder is a feed-forward non-recurrent neural net

With an input layer, an output layer and one or more hidden layers

- Can be trained using the same techniques
- Compute gradients using back-propagation
  - Followed by minibatch gradient descent

Autoencoder is a feed-forward non-recurrent neural net

With an input layer, an output layer and one or more hidden layers

- Can be trained using the same techniques
- Compute gradients using back-propagation
  - Followed by minibatch gradient descent

Unlike feedforward networks, can also be trained using Recirculation

- Compare activations on the input to activations of the reconstructed input
- More biologically plausible than back-prop but rarely used in ML

Autoencoder training using a loss function

We have Encoder f and Decoder g

- $f : X \rightarrow h$
- $g : h \rightarrow X$
- Thus, we have

$$\arg\min_{f,g} \left\| \boldsymbol{x} - (f \circ g)\, \boldsymbol{x} \right\|^2$$

Example

One hidden layer

- Takes input $\boldsymbol{x} \in \mathbb{R}^d$
- Maps into an output $\boldsymbol{h} \in \mathbb{R}^p$

Example

One hidden layer

- Takes input $\boldsymbol{x} \in \mathbb{R}^d$
- Maps into an output $\boldsymbol{h} \in \mathbb{R}^p$

We have using an element wise activation function

- $\boldsymbol{h} = \sigma_1 \left(W\boldsymbol{x} + \boldsymbol{b} \right) \rightarrow \boldsymbol{x}' = \sigma_2 \left(W'\boldsymbol{h} + \boldsymbol{b}' \right)$

Example

One hidden layer

- Takes input $\boldsymbol{x} \in \mathbb{R}^d$
- Maps into an output $\boldsymbol{h} \in \mathbb{R}^p$

We have using an element wise activation function

- $\boldsymbol{h} = \sigma_1 \left(W\boldsymbol{x} + \boldsymbol{b} \right) \rightarrow \boldsymbol{x}' = \sigma_2 \left(W'\boldsymbol{h} + \boldsymbol{b}' \right)$

Trained to minimize reconstruction error

$$L\left(\boldsymbol{x}, \boldsymbol{x}'\right) = \left\|\boldsymbol{x} - \boldsymbol{x}'\right\| = \left\|\boldsymbol{x} - \sigma_2\left(W^t \sigma_1\left(W\boldsymbol{x} + \boldsymbol{b}\right) + \boldsymbol{b}'\right)\right\|$$

Outline

Undercomplete Autoencoder

Copying input to output sounds useless

- Imagine that we do not have interest in decoder output
- We hope $\boldsymbol{h}$ takes on useful properties

Undercomplete Autoencoder

Copying input to output sounds useless

- Imagine that we do not have interest in decoder output
- We hope $\boldsymbol{h}$ takes on useful properties

Undercomplete autoencoder

- Constrain $\boldsymbol{h}$ to have lower dimension than $\boldsymbol{x}$
- Force it to capture most salient features of training data

Autoencoder with linear decoder + MSE is PCA

We have that

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

Autoencoder with linear decoder + MSE is PCA

We have that

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

Something Notable

- Where L is a loss function penalizing $g(f(\boldsymbol{x}))$ for being dissimilar from $\boldsymbol{x}$

Autoencoder with linear decoder + MSE is PCA

We have that

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

Something Notable

- Where L is a loss function penalizing $g(f(\boldsymbol{x}))$ for being dissimilar from $\boldsymbol{x}$

Thus

- When the decoder $\boldsymbol{g}$ is linear and L is the mean squared error, an undercomplete autoencoder learns to span the same subspace as PCA

$$g\left(\boldsymbol{x}\right) = W\boldsymbol{h} + \boldsymbol{b}$$

We have

In this case

- The autoencoder trained to perform the copying task has learned the principal subspace of the training data as a side-effect

We have

In this case

- The autoencoder trained to perform the copying task has learned the principal subspace of the training data as a side-effect

Autoencoders with nonlinear f and g

- They can learn more powerful nonlinear generalizations of PCA
- Something called Capacity

The capacity of a network

The capacity of a network refers to

- The range or scope of the types of functions that the model can approximate.
- Informally, a model's capacity is its ability to fit a wide variety of functions.

Encoder/Decoder Capacity

If encoder f and decoder g are allowed too much capacity

- Autoencoder can learn to perform the copying task without learning any useful information about distribution of data

Encoder/Decoder Capacity

If encoder f and decoder g are allowed too much capacity

- Autoencoder can learn to perform the copying task without learning any useful information about distribution of data

Something Notable

- Autoencoder with a one-dimensional code and a very powerful nonlinear encoder can learn to map $x(i)$ to code i.
- The decoder can learn to map these integer indices back to the values of specific training examples

Encoder/Decoder Capacity

If encoder f and decoder g are allowed too much capacity

- Autoencoder can learn to perform the copying task without learning any useful information about distribution of data

Something Notable

- Autoencoder with a one-dimensional code and a very powerful nonlinear encoder can learn to map $x(i)$ to code i.
- The decoder can learn to map these integer indices back to the values of specific training examples

A notable problem

- Autoencoder trained for copying task fails to learn anything useful if f/g capacity is too great

Cases when Autoencoder Learning Fails

Where autoencoders fail to learn anything useful

- Capacity of encoder/decoder f/g is too high
  - Capacity controlled by depth
- Hidden code $\boldsymbol{h}$ has dimension equal to input $\boldsymbol{x}$
- Overcomplete case: where hidden code $\boldsymbol{h}$ has dimension greater than input $\boldsymbol{x}$
  - Even a linear encoder/decoder can learn to copy input to output without learning anything useful about data distribution

Outline

What is regularization

L1 Regularization

- This method adds a penalty to the loss function for the sum of the absolute values of the model weights.

What is regularization

L1 Regularization

- This method adds a penalty to the loss function for the sum of the absolute values of the model weights.

L2 Regularization

- This method adds a penalty to the loss function for the sum of the squares of the model weights.

What is regularization

L1 Regularization

- This method adds a penalty to the loss function for the sum of the absolute values of the model weights.

L2 Regularization

- This method adds a penalty to the loss function for the sum of the squares of the model weights.

Dropout

- This method randomly sets a fraction of the model's activations to zero during each training iteration.

Use regularization

Ideally

- choose code size (dimension of $\boldsymbol{h}$) small and capacity of encoder f and decoder g based on complexity of distribution modeled

Use regularization

Ideally

- choose code size (dimension of $\boldsymbol{h}$) small and capacity of encoder f and decoder g based on complexity of distribution modeled

Regularized autoencoders provide the ability to do so

- Rather than limiting model capacity by keeping encoder/decoder shallow and code size small
- They use a loss function that encourages the model to have properties other than copy its input to output=

Regularized Autoencoder Properties

Regularized AEs have properties beyond copying input to output

- Sparsity of representation
- Smallness of the derivative of the representation
- Robustness to noise
- Robustness to missing inputs

Regularized Autoencoder Properties

Regularized AEs have properties beyond copying input to output

- Sparsity of representation
- Smallness of the derivative of the representation
- Robustness to noise
- Robustness to missing inputs

Regularized autoencoder can be nonlinear and overcomplete

- But still learn something useful about the data distribution even if model capacity is great enough to learn trivial identity function

Outline

Initializing Deep Learners

Autoencoders have many interesting applications

- As data compression, visualization, etc

Initializing Deep Learners

Autoencoders have many interesting applications

- As data compression, visualization, etc

Something Notable

- Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In Advances in Neural Information Processing Systems, 2007.

They discovered something interesting

Something Notable

- They observed that autoencoders could be used as a way to “pre-train” neural networks.

They discovered something interesting

Something Notable

- They observed that autoencoders could be used as a way to "pre-train" neural networks.

With pretraining, the process of training a deep network is divided in a sequence of steps

1. Pretraining step: train a sequence of shallow autoencoders, greedily one layer at a time, using unsupervised data,
2. Fine-tuning step 1: train the last layer using supervised data,
3. Fine-tuning step 2: use backpropagation to fine-tune the entire network using supervised data.

Basically

The step 2 can be seen as the training of a Perceptron

- This can be done easily

Basically

The step 2 can be seen as the training of a Perceptron

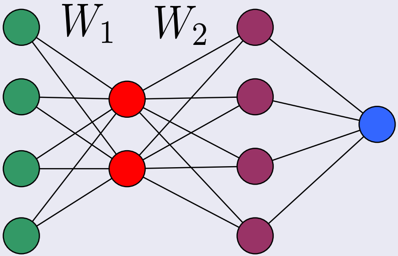
- This can be done easily

The Backpropagation of the entire network

- Also the classic procedure

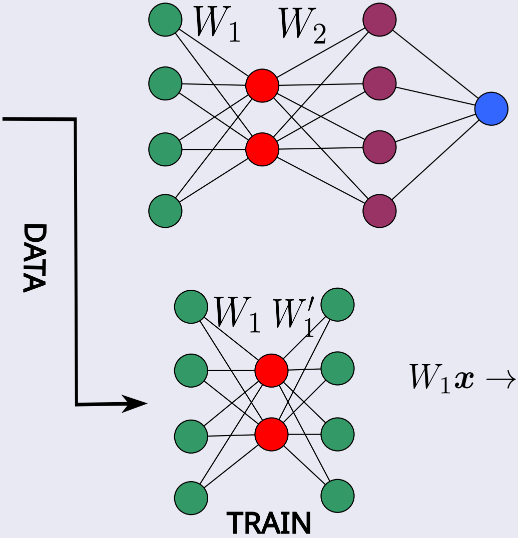
The First Step is the interesting one

We have a Feed Forward Architecture



First Pre-training, the first layer

We have

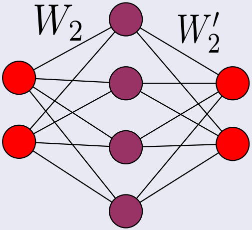


$W_1 \boldsymbol{x} \rightarrow \boldsymbol{y}$

Therefore, we have the final sub-step

We use the $\boldsymbol{y}$ to train another auto-encoder

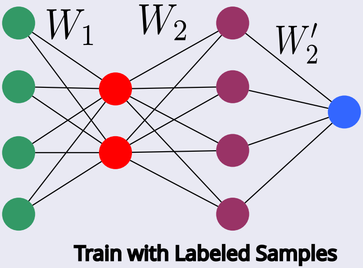
$W_1\boldsymbol{x} \rightarrow \boldsymbol{y}$



TRAIN

Finally

We have that



However

As Quality Data Sets become more widely avaible

- Google, Facebook, Microsoft and many others took charge of that...

However

As Quality Data Sets become more widely avaible

- Google, Facebook, Microsoft and many others took charge of that...

This efforts were abandoned

- Instead Transfer Learning [4] become a more popular idea
  - For initialization....

Outline

Sparse Autoencoders

Similar to the LASSO

$$\arg\min_{A,B} \sum_{t=1}^{m} \Delta\left(B \circ A\left(x_t\right), x_t\right) + \lambda \sum_{i} |a_i|$$

- where a_i is the activation at the i^{th} hidden layer and i iterates over all the hidden activation's.

Other Ways

KL-divergence

- We can assume the activation of each neuron acts as a Bernouli variable with probability p and tweak that probability.

Other Ways

KL-divergence

- We can assume the activation of each neuron acts as a Bernouli variable with probability p and tweak that probability.

For each neuron j

- The calculated empirical probability is $\hat{p}_j = \frac{1}{m} \sum_i a_i(i)$ where i iterates over the samples in the batch:

$$\arg\min_{A,B} \sum_{t=1}^{m} \Delta\left(B \circ A\left(x_t\right), x_t\right) + \sum_i KL\left(p \,\|\hat{p}\right)$$

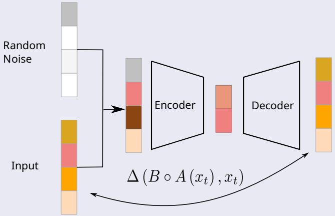
  - where the regularization term in it aims at matching p to $\hat{p}$.

Outline

Denoising Autoencoders

Note

- They can be viewed either as a regularization option, or as robust autoencoders which can be used for error correction.



Here, we have

We have that the input can be seen as

$$\widetilde{x} = x + N(0, \sigma I) \rightarrow p(\widetilde{x}|x) \sim N(x, \sigma I)$$

Here, we have

We have that the input can be seen as

$$\widetilde{x} = x + N(0, \sigma I) \to p(\widetilde{x}|x) \sim N(x, \sigma I)$$

An also

$$\widetilde{x} = \beta \odot x \to \beta \sim Ber(p)$$

Outline

Contractive Autoencoders

Here, the desire is to reduce the effect of small perturbations

- On the feature extraction process.

Contractive Autoencoders

Here, the desire is to reduce the effect of small perturbations

- On the feature extraction process.

By forcing the encoder to avoid changes

- Not important for the reconstruction by the decoder.

Basically

Apply regularization into the hidden layer of the encoder

- Into the Jacobian matrix of the Hidden Layers of the Encoder

Basically

Apply regularization into the hidden layer of the encoder

- Into the Jacobian matrix of the Hidden Layers of the Encoder

Fromally, $J_{ji} = \nabla_{x_i} h_j \left(x_i \right)$

- Thus changes at the nodes h_j of a layer h.

Thus, we have

We try to minimize a Ridge regression on the Jacobian

$$\arg\min_{A,B} E\left(\Delta\left(x, B \circ A\left(x\right)\right)\right) + \lambda \left\| J_A\left(x\right)\right\|_2^2$$

Outline

The Paper

Around 2015 [5]

- An extraordinary network came to be
  - "U-Net: Convolutional Networks for Biomedical Image Segmentation" by Olaf Ronneberger, Philipp Fischer, and Thomas Brox

The Paper

Around 2015 [5]

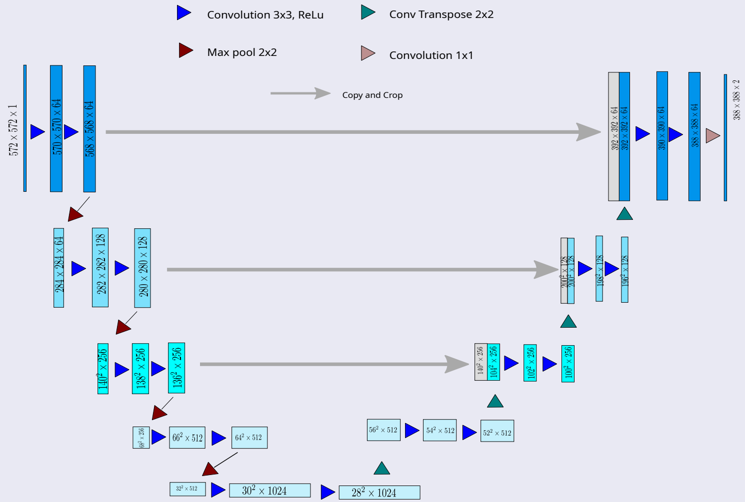
- An extraordinary network came to be
  - "U-Net: Convolutional Networks for Biomedical Image Segmentation" by Olaf Ronneberger, Philipp Fischer, and Thomas Brox

It won

1. The Grand Challenge for Computer-Automated Detection of Caries in Bitewing Radiography at ISBI 2015,
2. The Cell Tracking Challenge at ISBI 2015

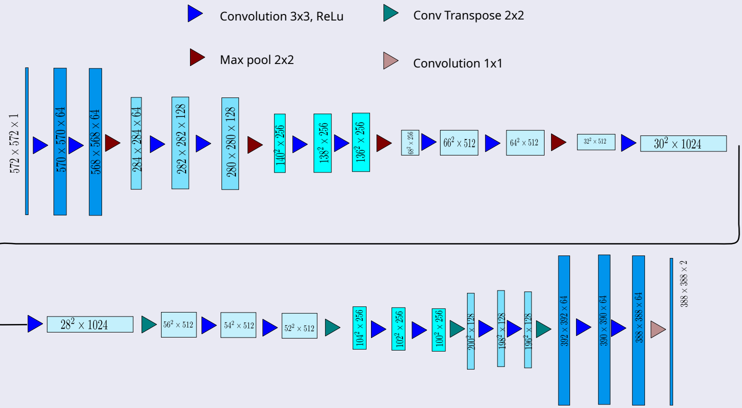
The Segmentation Architecture

We have (Imagen from [5])



Similarity with an Autoencoder

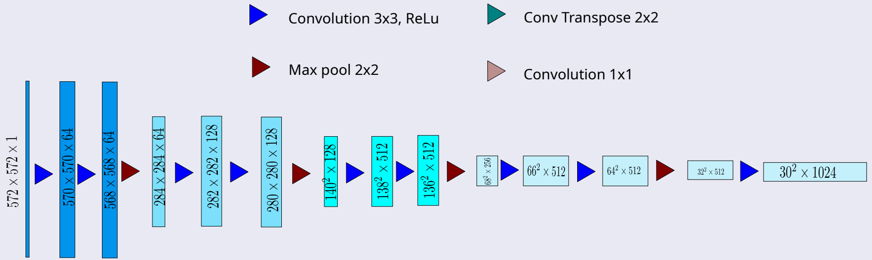
If we remove the extra passing of information, we finish with an autoencoder



Outline

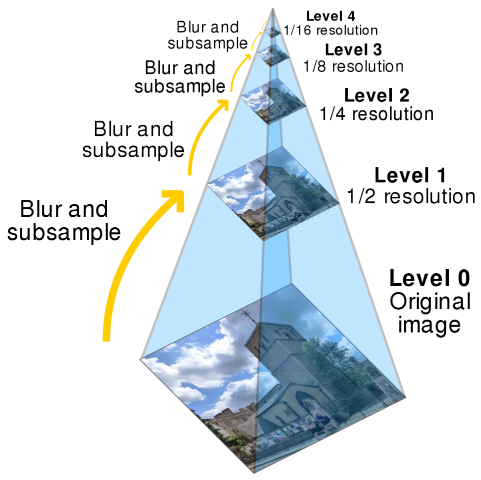
We have

At the encoder part



If you think about this

It is our Image Pyramid of old computer vision



Clearly

We can train this pyramide/encoder with data

- Yes our old backpropagation a.k.a automatic differentiation

Clearly

We can train this pyramide/encoder with data

- Yes our old backpropagation a.k.a automatic differentiation

At the center of this is the Convolution for multiple filters

$$Y_i^{(l)}(x,y) = B_i^{(l)}(x,y) + \sum_{j=1}^{m_1^{(l-1)}} \sum_{u=-ks}^{ks} \sum_{v=-ks}^{ks} Y_j^{(l-1)}(i-u, j-v)\, K_{ij}^{(l)}(u,v)$$

Basically

We take the j filter of the C_{out} dimension

- Then for each filter j you apply the convolution to each $Image_k$

Basically

We take the j filter of the C_{out} dimension

- Then for each filter j you apply the convolution to each $Image_k$

Then you simply add the images result of such convolution

- And Add Pointwise the bias $b_{C_{out_j}}$

Outline

The Variational Autoencoders [6]

It is based on the Maximum a Posteriori Idea under a Random Process

- A value $x^{(i)}$ is generated from some likelihood distribution $p_{\Theta}(x|z)$ with a prior $p_{\Theta}(z)$

$$p_{\Theta}(z|x) \approx p_{\Theta}(x|z)\, p_{\Theta}(z)$$

The Variational Autoencoders [6]

It is based on the Maximum a Posteriori Idea under a Random Process

- A value $x^{(i)}$ is generated from some likelihood distribution $p_\Theta(x|z)$ with a prior $p_\Theta(z)$

$$p_\Theta(z|x) \approx p_\Theta(x|z)\, p_\Theta(z)$$

We ask $p_\Theta(x|z)$ and $p_\Theta(z)$

1. They are coming from parametric families
2. They are differentiable almost everywhere w.r.t. Θ and z

However

When we have the following case

- The case where the integral of the marginal likelihood is intractable

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

However

When we have the following case

- The case where the integral of the marginal likelihood is intractable

$$p_\Theta(x) = \int p_\Theta(x|z)\, p_\Theta(z)\, dz$$

Also the **posterior density** is also intractable (No Expectation Maximization can be used)

$$p_\Theta(z|x) = \frac{p_\Theta(x|z)\, p_\Theta(z)}{p_\Theta(x)}$$

However

When we have the following case

- The case where the integral of the marginal likelihood is intractable

$$p_{\Theta}\left(x\right)=\int p_{\Theta}\left(x|z\right)p_{\Theta}\left(z\right)dz$$

Also the **posterior density** is also intractable (No Expectation Maximization can be used)

$$p_{\Theta}\left(z|x\right)=\frac{p_{\Theta}\left(x|z\right)p_{\Theta}\left(z\right)}{p_{\Theta}\left(x\right)}$$

These intractability's are quite common and appear in cases of moderately complicated likelihood functions

- For example Neural Networks...

More problems with sampling Bayesian Estimators

Gibbs Samplers, Metropolis-Hastings

- Have a serious problem, they are difficult to parallelize making them unusable for Deep Learning Applications.

More problems with sampling Bayesian Estimators

Gibbs Samplers, Metropolis-Hastings

- Have a serious problem, they are difficult to parallelize making them unusable for Deep Learning Applications.

We need another way to estimate the posterior of a Bayesian method

- Variational Bayes is such alternative

More problems with sampling Bayesian Estimators

Gibbs Samplers, Metropolis-Hastings

- Have a serious problem, they are difficult to parallelize making them unusable for Deep Learning Applications.

We need another way to estimate the posterior of a Bayesian method

- Variational Bayes is such alternative

This is done by approximate $p(x|z)$ by the use of a tractable $q(x)$

- This is done by the use a trick in Variational Bayes.

Outline

We start with something simple

The classic

- The marginal likelihood is composed of a sum over the marginal likelihoods of individual datapoints:

$$\log p_\theta \left(\boldsymbol{x}_1, ..., \boldsymbol{x}_N\right) = \sum_{i=1}^{N} \log p_\theta \left(\boldsymbol{x}_i\right)$$

We start with something simple

The classic

- The marginal likelihood is composed of a sum over the marginal likelihoods of individual datapoints:

$$\log p_\theta\left(\boldsymbol{x}_1, ..., \boldsymbol{x}_N\right) = \sum_{i=1}^{N} \log p_\theta\left(\boldsymbol{x}_i\right)$$

Therefore, if we can maximize each $\log p_\theta\left(\boldsymbol{x}_i\right)$

- We maximize all $\log p_\theta\left(\boldsymbol{x}_1, ..., \boldsymbol{x}_N\right)$

Outline

The Variational Bound using the Kullback-Leibler (KL) Divergence

The best Variational Bayes approximation $q^* \in \mathcal{Q}$ is found by minimizing against the true $p(x)$

$$q^* = \arg\min_{q^* \in \mathcal{Q}} \left\{ KL\left(q \,\|p\right) = \int q(x) \log\left(\frac{q(x)}{p(x)}\right) dx \right\}$$

Properties of KL Divergence

Non-Negativity

- We have that $KL\left(q\,\|p\right)\geq 0$
- Equality holds if and only if : $q=p$ almost surely (i.e., the distributions are identical).

Properties of KL Divergence

Non-Negativity

- We have that $KL\left(q\,\|p\right) \geq 0$
- Equality holds if and only if : $q = p$ almost surely (i.e., the distributions are identical).

Convexity in q

- The KL divergence is convex in q (for fixed p).
- This ensures that the optimization problem in variational inference (minimizing KL divergence) has a unique global minimum, making it tractable for gradient-based methods.

Furthermore

We actually cannot minimize the KL divergence exactly,

- but we can minimize a function that is equal to it up to a constant

Furthermore

We actually cannot minimize the KL divergence exactly,

- but we can minimize a function that is equal to it up to a constant

An application of the Jensen's inequality for probability distributions

- When f is concave

$$f\left(E\left[x\right]\right) \geq E\left[f\left(x\right)\right]$$

Using it we get the maximizing the Evidence Lower Bound (ELBO)

Going back to the problem of maximizing $\log p_{\theta}\left(\boldsymbol{x}_1, ..., \boldsymbol{x}_N\right)$

$$
\begin{aligned}
\log p\left(x\right) &= \log \int_{z} p\left(x, z\right) \\
&= \log \int_{z} p\left(x, z\right) \frac{q\left(z\right)}{q\left(z\right)} \\
&= \log \left[E_q\left[\frac{p\left(x, Z\right)}{q\left(z\right)}\right]\right] \\
&\geq E_q\left[\log\left(\frac{p\left(x, Z\right)}{q\left(z\right)}\right)\right] \\
&= E_q\left[\log p\left(x, Z\right)\right] - E_q\left[\log q\left(Z\right)\right]
\end{aligned}
$$

From minimization to maximization

Minimizing KL is equivalent to maximizing the lower bound on $\log p(x)$

$$ELBO(q) = E_q[\log p(x, Z)] - E_q[\log q(Z)]$$

- The Expectation Lower Bound (ELBO) which also appears at the Expectation Maximization

We can see this when looking at the Posterior probability

Note first that

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

We can see this when looking at the Posterior probability

Note first that

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

We can see how the ELBO is inserted into the KL divergence

$$\begin{aligned}
KL\left(q\left(z\right)\|p\left(z|x\right)\right) &= E_q\left[\log\frac{q\left(Z\right)}{p\left(Z|x\right)}\right]\\
&= E_q\left[\log q\left(Z\right)\right]-E_q\left[\log p\left(Z|x\right)\right]\\
&= E_q\left[\log q\left(Z\right)\right]-E_q\left[\log\frac{p\left(Z,x\right)}{p\left(x\right)}\right]\\
&= -\left(\underbrace{E_q\left[\log p\left(Z,x\right)\right]-E_q\left[\log q\left(Z\right)\right]}_{ELBO}\right)+\log p\left(x\right)\\
&= -E_q\left[\log\frac{p\left(Z,x\right)}{q\left(Z\right)}\right]+\log p\left(x\right)
\end{aligned}$$

Therefore

If we want to minimize the $KL\left(q\left(z\right)\|p\left(z|x\right)\right)$

- We need to maximize the term $E_q\left[\log p\left(x,Z\right)\right]-E_q\left[\log q\left(Z\right)\right]$

Outline

Thus, we have

Something Notable

- The join likelihood is composed of a sum over the marginal likelihoods of individual datapoints

$$\log p_{\Theta}\left(z_1, z_2, z_3, ..., z_N\right) = \sum_{i=1}^{N} \log p_{\Theta}\left(z_i\right)$$

Thus, we have

Something Notable

- The join likelihood is composed of a sum over the marginal likelihoods of individual datapoints

$$\log p_\Theta\left(z_1, z_2, z_3, ..., z_N\right) = \sum_{i=1}^{N} \log p_\Theta\left(z_i\right)$$

Typically, this family does not contain the true posterior

- because the hidden variables are dependent.
  - A the Gaussian mixture model all of the cluster assignments z_i are dependent on each other and the cluster locations $\mu_{1:K}$ given the data $x_{1:n}$.
  - These dependencies are often what makes the posterior difficult to work with.

Some Notation

$p\left(z_j \middle| z_{-j}, x\right)$

- $p\left(z_j \middle| z_1, ..., z_{j-1}, z_j, ..., z_m, x\right) = p\left(z_j \middle| z_{-j}, x\right)$

Now, we optimize the ELBO for this factorized distribution

First, recall the chain rule and use it to decompose the joint

$$p\left(z_{1:m}, x_{1:n}\right) = p\left(x_{1:n}\right) \prod_{j=1}^{m} p\left(z_j | z_{1:j-1}, x_{1:n}\right)$$

- Notice that the z variables can occur in any order in this chain.

Now, we optimize the ELBO for this factorized distribution

First, recall the chain rule and use it to decompose the joint

$$p(z_{1:m}, x_{1:n}) = p(x_{1:n}) \prod_{j=1}^{m} p(z_j | z_{1:j-1}, x_{1:n})$$

- Notice that the z variables can occur in any order in this chain.

Second, decompose the entropy of the variational distribution,

$$E[\log q(z_{1:m})] = \sum_{j=1}^{m} E_j [\log q(z_j)]$$

Now, we optimize the ELBO for this factorized distribution

First, recall the chain rule and use it to decompose the joint

$$p\left(z_{1:m}, x_{1:n}\right) = p\left(x_{1:n}\right)\prod_{j=1}^{m} p\left(z_j | z_{1:j-1}, x_{1:n}\right)$$

- Notice that the z variables can occur in any order in this chain.

Second, decompose the entropy of the variational distribution,

$$E\left[\log q\left(z_{1:m}\right)\right] = \sum_{j=1}^{m} E_j\left[\log q\left(z_j\right)\right]$$

We know from the previous $KL\left(q\left(z\right)\|p\left(z|x\right)\right) = -E_q\left[\log \frac{p(Z,x)}{q(Z)}\right] + \log p\left(x\right)$

$$KL\left(q\left(z\right)\|p\left(z|x\right)\right) = \log p\left(x\right) - \left[E_q\left[\log p\left(Z,x\right) - \log q\left(Z\right)\right]\right]$$

We have then

We know that

$$p\left(z_j | z_{1:j-1}, x_{1:n}\right) = \frac{p\left(z_{1:j}, x_{1:n}\right)}{p\left(z_{1:j-1}, x_{1:n}\right)}$$

We have then

We know that

$$p\left(z_j|z_{1:j-1},x_{1:n}\right)=\frac{p\left(z_{1:j},x_{1:n}\right)}{p\left(z_{1:j-1},x_{1:n}\right)}$$

Then if we multiply with $\frac{p(z_{1:j-1},x_{1:n})}{p(z_{1:j-1},x_{1:n})}$ the numerator can go out and seen as a constant

$$\mathcal{L}=\log p\left(x_{1:n}\right)-\sum_{j=1}^{m}\left(E\left[\log\frac{p\left(z_{1:j},x_{1:n}\right)}{p\left(z_{1:j-1},x_{1:n}\right)}\right]+\underbrace{E\left[p\left(z_{1:j-1},x_{1:n}\right)\right]}_{\text{const with respect to } z_j}-E_j\left[\log q\left(z_j\right)\right]\right)$$

Finally, we get

A final equation

$$\mathcal{L} \approx \log p\left(x_{1:n}\right) - \underbrace{\sum_{j=1}^{m}\left(E\left[\log p\left(z_j|z_{1:j-1}, x_{1:n}\right)\right] - E_j\left[\log q\left(z_j\right)\right]\right)}_{KL}$$

Now, Consider the ELBO as a function of $q(z_j)$

Employ the chain rule with the variable z_k as the last variable in the list

- This leads to the objective function

$$\begin{aligned}\mathcal{L} =& E\left[\log p\left(z_j|z_{-j},x\right)\right] - E_j\left[\log q\left(z_j\right)\right] + constant\\ \approx& E\left[\log p\left(z_j|z_{-j},x\right)\right] - E_j\left[\log q\left(z_j\right)\right]\end{aligned}$$

Now, Consider the ELBO as a function of $q(z_j)$

Employ the chain rule with the variable z_k as the last variable in the list

- This leads to the objective function

$$\begin{aligned}\mathcal{L} =& E\left[\log p\left(z_j|z_{-j},x\right)\right] - E_j\left[\log q\left(z_j\right)\right] + constant\\ \approx& E\left[\log p\left(z_j|z_{-j},x\right)\right] - E_j\left[\log q\left(z_j\right)\right]\end{aligned}$$

Write this objective as a function of $q(z_k)$

$$\mathcal{L}\left[q\left(z_j\right)\right] \approx \int q\left(z_j\right) E_{q_{-j}}\left[\log p\left(z_j|z_{-j},x\right)\right] dz_j - \int q\left(z_j\right)\log q\left(z_j\right) dz_j$$

Therefore, we want

To maximize this quantity

$$\arg\max_{q_j} \left\{ \int q\left(z_j\right) E_{q_{-j}}\left[\log p\left(z_j|z_{-j},x\right)\right] dz_j - \int q\left(z_j\right) \log q\left(z_j\right) dz_j \right\}$$

Now Optimize

Take the derivative with respect to $q\left(z_j\right)$

$$\frac{d\mathcal{L}\left[q\left(z_j\right)\right]}{dq\left(z_j\right)} = E_{q_{-j}}\left[\log p\left(z_j|z_{-j},x\right)\right] - \log q\left(z_j\right) - \underbrace{\frac{q\left(z_j\right)}{q\left(z_j\right)}}_{1} = 0$$

Now Optimize

Take the derivative with respect to $q(z_j)$

$$\frac{d\mathcal{L}[q(z_j)]}{dq(z_j)} = E_{q_{-j}}[\log p(z_j|z_{-j}, x)] - \log q(z_j) - \underbrace{\frac{q(z_j)}{q(z_j)}}_{1} = 0$$

This (and Lagrange multipliers) leads to an coordinate ascent for $q(z_j)$

$$q^*(z_j) \propto \exp\left\{E_{q_{-j}}[\log p(z_j|z_{-j}, x)]\right\}$$

Now Optimize

Take the derivative with respect to $q(z_j)$

$$\frac{d\mathcal{L}[q(z_j)]}{dq(z_j)} = E_{q_{-j}}\left[\log p(z_j|z_{-j},x)\right] - \log q(z_j) - \underbrace{\frac{q(z_j)}{q(z_j)}}_{1} = 0$$

This (and Lagrange multipliers) leads to an coordinate ascent for $q(z_j)$

$$q^*(z_j) \propto \exp\left\{E_{q_{-j}}\left[\log p(z_j|z_{-j},x)\right]\right\}$$

Since the denominator of the conditional does not depend on z_j

$$q^*(z_j) \propto \exp\left\{E_{q_{-j}}\left[\log p(z_j,z_{-j},x)\right]\right\}$$

Some Remarks

We have that

- This coordinate ascent procedure convergences to a local maximum.
- The coordinate ascent update for $q\left(z_j\right)$ only depends on the other, fixed approximations $q\left(z_k\right), k \neq j$.

Some Remarks

We have that

- This coordinate ascent procedure convergences to a local maximum.
- The coordinate ascent update for $q\left(z_j\right)$ only depends on the other, fixed approximations $q\left(z_k\right), k \neq j$.

In addition

- While this determines the optimal $q\left(z_j\right)$, it is still not specified.
- Depending on what form we use, the coordinate update $q\left(z_j\right)$ might not be easy to work with.

Outline

We have that

Thus, from $\log p_{\Theta}\left(z_1, z_2, z_3, ..., z_N\right) = \sum_{i=1}^{N} \log p_{\Theta}\left(z_i\right)$,and knowing that KL is positive

$$\log p_{\Theta}\left(z_i\right) \geq -\int_{\mathcal{Z}} q_{\phi}\left(z\right) \log\left(\frac{p_{\Theta}\left(x, z_i\right)}{q_{\phi}\left(z\right)}\right) dx + \mathcal{T}\left(\Theta, \phi | z_i\right)$$

We have that

Thus, from $\log p_\Theta(z_1, z_2, z_3, \ldots, z_N) = \sum_{i=1}^{N} \log p_\Theta(z_i)$,and knowing that KL is positive

$$\log p_\Theta(z_i) \geq -\int_{\mathcal{Z}} q_\phi(z) \log\left(\frac{p_\Theta(x, z_i)}{q_\phi(z)}\right) dx + \mathcal{T}(\Theta, \phi | z_i)$$

The First Term has

- $\int_{\mathcal{Z}} q_\phi(z) \log\left(\frac{p_\Theta(x,z_i)}{q_\phi(z)}\right) dx = D_{KL}(p_\Theta(x, z_i) || q_\phi(z))$ is the KL Divergence of the approximate of the true posterior

We have that

Thus, from $\log p_\Theta(z_1, z_2, z_3, ..., z_N) = \sum_{i=1}^N \log p_\Theta(z_i)$,and knowing that KL is positive

$$\log p_\Theta(z_i) \geq -\int_{\mathcal{Z}} q_\phi(z) \log\left(\frac{p_\Theta(x, z_i)}{q_\phi(z)}\right) dx + \mathcal{T}(\Theta, \phi | z_i)$$

The First Term has

- $\int_{\mathcal{Z}} q_\phi(z) \log\left(\frac{p_\Theta(x,z_i)}{q_\phi(z)}\right) dx = D_{KL}(p_\Theta(x, z_i) || q_\phi(z))$ is the KL Divergence of the approximate of the true posterior

The Second Term

- $\mathcal{T}(\Theta, \phi | z_i)$ the variational lower bound on the marginal likelihood z_i

Now given that the $D_{KL}\left(p_{\Theta}\left(x,z_i\right)||q_{\phi}\left(z\right)\right)$ is positive

We can say

$$\log p_{\Theta}\left(z_i\right) \geq \mathcal{T}\left(\Theta,\phi|z_i\right) = E_{q_{\phi}(x|z)}\left[-\log q_{\phi}\left(x|z\right) + \log p_{\Theta}\left(z\right)\right]$$

Now given that the $D_{KL}\left(p_{\Theta}\left(x, z_i\right) || q_{\phi}\left(z\right)\right)$ is positive

We can say

$$\log p_{\Theta}\left(z_i\right) \geq \mathcal{T}\left(\Theta, \phi | z_i\right) = E_{q_{\phi}(x|z)}\left[-\log q_{\phi}\left(x|z\right) + \log p_{\Theta}\left(z\right)\right]$$

Actually is a KL on p_{Θ} and q_{ϕ}

$$-\log q_{\phi}\left(x|z\right) + \log p_{\Theta}\left(z\right) = \log\left(\frac{p_{\Theta}\left(z\right)}{q_{\phi}\left(x|z\right)}\right)$$

Now given that the $D_{KL}\left(p_{\Theta}\left(x, z_i\right) || q_{\phi}\left(z\right)\right)$ is positive

We can say

$$\log p_{\Theta}\left(z_i\right) \geq \mathcal{T}\left(\Theta, \phi | z_i\right) = E_{q_{\phi}(x|z)}\left[-\log q_{\phi}\left(x|z\right) + \log p_{\Theta}\left(z\right)\right]$$

Actually is a KL on p_{Θ} and q_{ϕ}

$$-\log q_{\phi}\left(x|z\right) + \log p_{\Theta}\left(z\right) = \log\left(\frac{p_{\Theta}\left(z\right)}{q_{\phi}\left(x|z\right)}\right)$$

Actually, we can simplify this last function by dropping the term $-\log q_{\phi}\left(x|z\right)$ because is already in the first term

$$\mathcal{T}\left(\Theta, \phi | z_i\right) \propto E_{q_{\phi}(x|z)}\left[\log p_{\Theta}\left(z\right)\right]$$

Therefore, we can generate a loss function combining both terms $\mathcal{L}(\Theta, \phi|z)$

As the following give that they are KL divergences i.e. maximization of this minimize the divergences

$$\mathcal{L}(\Theta, \phi|z_i) = -KL\left(p_{\Theta}(x, z)\,||q_{\phi}(z)\right) + E_{q_{\phi}(x|z_i)}\left[\log p_{\Theta}(z_i)\right]$$

Therefore, we can generate a loss function combining both terms $\mathcal{L}(\Theta, \phi|z)$

As the following give that they are KL divergences i.e. maximization of this minimize the divergences

$$\mathcal{L}(\Theta, \phi|z_i) = -KL\left(p_\Theta(x, z)\,||q_\phi(z)\right) + E_{q_\phi(x|z_i)}\left[\log p_\Theta(z_i)\right]$$

Thus, we want to derive $\mathcal{L}(\Theta, \phi|z_i)$

- w.r.t. both the variational parameters ϕ and generative parameters Θ.

Problem

Problem is that MCMC is not enough

- The usual (naïve) Monte Carlo gradient estimator for this type of problem is:

$$
\begin{aligned}
\nabla_{\phi} E_{q_{\phi}(x)}\left(\log p_{\Theta}\left(z_i\right)\right) &= E\left(\nabla_{q_{\phi}(x)} q_{\phi}\left(x\right) \log p_{\Theta}\left(z_i\right)\right) \\
&\approx \frac{1}{L}\sum_{l=1}^{L} \nabla_{q_{\phi}(x)} q_{\phi}\left(x_l\right) \log p_{\Theta}\left(z_i|x_l\right) \\
&\approx E_{q_{\phi}(x_l)}\left[\log p_{\Theta}\left(z|x_l\right)\right]
\end{aligned}
$$

Problem with this first attempt of using this for maximization

Here, we have that $x \sim q_{\phi}(x|z_i)$, yes z_i is a hidden variable

- This gradient estimator exhibits exhibits very high variance and non practical

Outline

First, explicit re-parameterization Gradients

Suppose we would like to optimize an expectation $E_{q_\phi(x)}\left[f\left(x\right)\right]$

- A continuously differentiable function $f\left(z\right)$ w.r.t. the parameters of the distribution.

First, explicit re-parameterization Gradients

Suppose we would like to optimize an expectation $E_{q_\phi(x)}\left[f\left(x\right)\right]$

- A continuously differentiable function $f\left(z\right)$ w.r.t. the parameters of the distribution.

We assume that we can find a standardization function $S_\phi\left(x\right) = \epsilon \sim q\left(x\right)$

- Such that you can do the following $x = S_\phi^{-1}\left(\epsilon\right)$

First, explicit re-parameterization Gradients

Suppose we would like to optimize an expectation $E_{q_{\phi}(x)}\left[f\left(x\right)\right]$

- A continuously differentiable function $f\left(z\right)$ w.r.t. the parameters of the distribution.

We assume that we can find a standardization function $S_{\phi}\left(x\right) = \epsilon \sim q\left(x\right)$

- Such that you can do the following $x = S_{\phi}^{-1}\left(\epsilon\right)$

For example, for a Gaussian distribution $N\left(\mu, \sigma^2\right)$

- We can use the following function $S_{\mu,\sigma}\left(x\right) = \frac{x-\mu}{\sigma} \sim N\left(0, 1\right)$ then we have we can sample $\epsilon \sim N\left(0, 1\right)$

$$S_{\mu,\sigma}^{-1}\left(\epsilon\right) = \sigma \cdot \epsilon + \mu$$

Therefore, we can express the $E_{q_\phi(x)}\left[f\left(x\right)\right]$

As follow

$$E_{q_\phi(x)}\left[f\left(x\right)\right] = E_{q(\epsilon)}\left[f\left(S_\phi^{-1}\left(\epsilon\right)\right)\right]$$

Therefore, we can express the $E_{q_\phi(x)}\left[f\left(x\right)\right]$

As follow

$$E_{q_\phi(x)}\left[f\left(x\right)\right] = E_{q(\epsilon)}\left[f\left(S_\phi^{-1}\left(\epsilon\right)\right)\right]$$

This allows us to compute the gradient of the expectation as the expectation of the gradients by rule chain:

$$\begin{aligned}\nabla_\phi E_{q_\phi(x)}\left[f\left(x\right)\right] &= E_{q(\epsilon)}\left[\nabla_\phi f\left(S_{\mu,\sigma}^{-1}\left(\epsilon\right)\right)\right]\\ &= E_{q(\epsilon)}\left[\nabla_z f\left(S_{\mu,\sigma}^{-1}\left(\epsilon\right)\right)\nabla_\phi S_\phi^{-1}\left(\epsilon\right)\right]\end{aligned}$$

Therefore

An alternative was proposed to avoid the inverse of $S_{\phi}\left(x\right)$

- For this, we first we express the classic stuff

$$\nabla_{\phi}E_{q_{\phi}(x)}\left[f\left(x\right)\right]=E_{q_{\phi}(x)}\left[\nabla_{x}f\left(x\right)\nabla_{\phi}x\right]$$
$$\nabla_{\phi}x=\nabla_{\phi}S_{\phi}^{-1}\left(\epsilon\right)\Big|_{\epsilon=S_{\phi}(x)}$$

Therefore

An alternative was proposed to avoid the inverse of $S_\phi(x)$

- For this, we first we express the classic stuff

$$\nabla_\phi E_{q_\phi(x)}[f(x)] = E_{q_\phi(x)}[\nabla_x f(x) \nabla_\phi x]$$
$$\nabla_\phi x = \nabla_\phi S_\phi^{-1}(\epsilon)\Big|_{\epsilon=S_\phi(x)}$$

It was proposed to compute $\nabla_\phi x$ by implicit differentiation

- Implicit differentiation makes use of the chain rule to differentiate a function which cannot be explicitly expressed in the form $y = f(x)$

$$\frac{df(y(x))}{dx} = \frac{df}{dy} \times \frac{dy}{dx}$$

Then

We apply the total gradient to the equality $S_{\phi}(x) = \epsilon$

- We finish with the following given that $S_{\phi}(x)$ depends on ϕ directly by the subscript and from x indirectly. Thus, $x = x(\phi)$:

$$\frac{dS_{\phi}(x)}{d\phi} = \frac{d\epsilon}{d\phi}$$

$$\frac{dS_{\phi}(x)}{dx} \times \frac{dx}{d\phi} + \frac{dS_{\phi}(x)}{d\phi} = 0$$

Then, we have that

Therefore, we have that

$$\frac{dx}{d\phi} = -\left(\frac{dS_{\phi}\left(x\right)}{dx}\right)^{-1} \times \frac{dS_{\phi}\left(x\right)}{d\phi}$$

Then, we have that

Therefore, we have that

$$\frac{dx}{d\phi} = -\left(\frac{dS_{\phi}\left(x\right)}{dx}\right)^{-1} \times \frac{dS_{\phi}\left(x\right)}{d\phi}$$

This expression for the gradient only requires differentiating the standardization function

- Basically we get the value and invert it, we do not need an explicit version of the inverse function, then derive it.

Example

Univariate Normal distribution $N\left(\mu, \sigma^2\right)$

- We have the standardization function $S_{\mu,\sigma}\left(x\right) = \frac{x-\mu}{\sigma} = \epsilon \sim N\left(0, 1\right)$

Example

Univariate Normal distribution $N\left(\mu,\sigma^2\right)$

- We have the standardization function $S_{\mu,\sigma}\left(x\right)=\frac{x-\mu}{\sigma}=\epsilon\sim N\left(0,1\right)$

We have then, if we use the derivatives

$$\frac{dx}{d\mu}=-\left(\frac{dS_{\mu,\sigma}\left(x\right)}{dz}\right)^{-1}\times\frac{dS_{\mu,\sigma}\left(x\right)}{d\mu}=-\frac{-\frac{1}{\sigma}}{\frac{1}{\sigma}}=1$$

$$\frac{dx}{d\sigma}=-\left(\frac{dS_{\mu,\sigma}\left(x\right)}{dz}\right)^{-1}\times\frac{dS_{\mu,\sigma}\left(x\right)}{d\sigma}=\frac{-\frac{(z-\mu)}{\sigma^2}}{\frac{1}{\sigma}}=\frac{(z-\mu)}{\sigma}\sim N\left(0,1\right)$$

If we compare against inverting and deriving

We have the following inverse of the function

$$z = S_{\mu,\sigma}^{-1}(x) = \mu + \sigma\epsilon$$

If we compare against inverting and deriving

We have the following inverse of the function

$$z = S^{-1}_{\mu,\sigma}(x) = \mu + \sigma\epsilon$$

Then, we have that with $\epsilon \sim N(0,1)$

$$\frac{dz}{d\mu} = 1, \qquad \frac{dz}{d\sigma} = \epsilon$$

Outline

We have that

The essential parameterization trick is quite simple

- Let x be a continuous random variable and $x \sim q_{\phi}(x|z)$ a conditional distribution

We have that

The essential parameterization trick is quite simple

- Let x be a continuous random variable and $x \sim q_{\phi}(x|z)$ a conditional distribution

It is possible to extend $S_{\phi}(x) = \epsilon$

$$g_{\phi,z}(x) = \epsilon$$

- with $\epsilon \sim p(x)$ and $g_{\phi}(\cdot)$ is some vector valued function parameterized by ϕ, z

This parameterization is useful for our case

It can be used to rewrite an expectation w.r.t $q_\phi(x|z)$

- Such that the Monte Carlo estimate of the expectation is differentiable w.r.t. ϕ

This parameterization is useful for our case

It can be used to rewrite an expectation w.r.t $q_\phi\left(x|z\right)$

- Such that the Monte Carlo estimate of the expectation is differentiable w.r.t. ϕ

We want the following

- Given the mapping $g_{\phi,z}\left(x\right)=\epsilon$ (Here, x is a vector and $dx=\prod_i dx_i$ are infinitesimals)

$$q_\phi\left(x|z\right)\approx g_{\phi,z}\left(x\right)$$

Now, we can do the following

Given that the error is a small variation, we can say that in an small area

$$q_{\phi}\left(x|z\right)\prod_{i}dx_{i}=p\left(x\right)\prod_{i}d\epsilon_{i}$$

Now, we can do the following

Given that the error is a small variation, we can say that in an small area

$$q_\phi(x|z)\prod_i dx_i = p(x)\prod_i d\epsilon_i$$

Therefore

$$\int q_\phi(x|z)\, f(x)\, dx = \int p(x)\, f(x)\, dx = \int p(x)\, f(g_{\phi,z}(x))\, dx$$

We can then construct then a variational estimation

We have that

$$\int q_{\phi}\left(x|z\right) f\left(x\right) dx = \frac{1}{L}\sum_{l=1}^{L} f\left(g_{\phi,z}\left(x\right)\right) \text{ with } \epsilon_{l} \sim p\left(x\right)$$

Therefore, we have

Using the previous ideas

$$\nabla_{\phi} E_{q_{\phi}(x)} \left[f \left(g_{\phi,z} \left(x \right) \right) \right] = E_{q_{\phi}(x)} \left[\nabla_{x} f \left(g_{\phi,z} \left(x \right) \right) \frac{dx}{d\phi, z} \right]$$

$$\frac{dx}{d\phi, z} = - \left(\frac{dg_{\phi,z} \left(x \right)}{dx} \right)^{-1} \times \frac{dg_{\phi,z} \left(x \right)}{d\phi, z}$$

Outline

Therefore

We have that

$$\widetilde{\mathcal{L}}^{A}\left(\Theta,\phi|z_i\right)\approx\mathcal{L}\left(\Theta,\phi|z_i\right)$$

Therefore

We have that

$$\widetilde{\mathcal{L}}^{A}\left(\Theta,\phi|z_i\right)\approx\mathcal{L}\left(\Theta,\phi|z_i\right)$$

Thus, we use the estimation with $x_{i,l}=g_{\phi}\left(\epsilon_{i,l}\right)$ and $\epsilon_l\sim p\left(\epsilon\right)$

$$\widetilde{\mathcal{L}}^{A}\left(\Theta,\phi|z_i\right)=\frac{1}{L}\sum_{l=1}^{L}\left[\log p_{\Theta}\left(z_i,x_{i,l}\right)-\log q_{\phi}\left(x_{i,l}|z_i\right)\right]$$

Therefore

We have that

$$\widetilde{\mathcal{L}}^{A}\left(\Theta, \phi|z_i\right) \approx \mathcal{L}\left(\Theta, \phi|z_i\right)$$

Thus, we use the estimation with $x_{i,l} = g_{\phi}\left(\epsilon_{i,l}\right)$ and $\epsilon_l \sim p\left(\epsilon\right)$

$$\widetilde{\mathcal{L}}^{A}\left(\Theta, \phi|z_i\right) = \frac{1}{L}\sum_{l=1}^{L}\left[\log p_{\Theta}\left(z_i, x_{i,l}\right) - \log q_{\phi}\left(x_{i,l}|z_i\right)\right]$$

Actually, if $p\left(x\right)$ is seen as a normalization factor

$$\log\left(\frac{p_{\Theta}\left(z|x\right)}{q_{\phi}\left(x\right)}\right) = \log\left(\frac{p_{\Theta}\left(z, x\right)}{p\left(z\right)q_{\phi}\left(x\right)}\right) \approx \log\left(\frac{p_{\Theta}\left(z, x\right)}{q_{\phi}\left(x|z\right)}\right)$$

In this was, we have

We can say that

$$\log\left(\frac{p_\Theta\left(z_i, x_{i,l}\right)}{q_\phi\left(x_{i,l}|z_i\right)}\right) \approx \log p_\Theta\left(z_i, x_{i,l}\right) - \log q_\phi\left(x_{i,l}|z_i\right)$$

Final Algorithm

Minibatch version of the Auto-Encoding VB (AEVB) algorithm

- Initialize Parameters Θ, ϕ
- Repeat
- $\quad X^M \leftarrow$ Random Mini-batch of M data points
- $\quad \epsilon \leftarrow$ Random samples from noise distribution $p(\epsilon)$
- $\quad g \leftarrow \nabla_{\Theta,\phi} \widetilde{\mathcal{L}}^A (\Theta, \phi | z_i)$ (Gradient Mini-batch estimator)
- $\quad \Theta, \phi$ update parameters using the gradient
- return Θ, ϕ

Outline

The original variational auto-encoder

It is a continuous latent variable model.

- The model is intended to learn a latent space $\mathcal{Z} = \mathbb{R}^t$ using a given set of samples $\{x_n\} \subseteq \mathcal{Y} = \mathbb{R}^d$
- Such that $t \ll d$

The original variational auto-encoder

It is a continuous latent variable model.

- The model is intended to learn a latent space $\mathcal{Z} = \mathbb{R}^t$ using a given set of samples $\{x_n\} \subseteq \mathcal{Y} = \mathbb{R}^d$
- Such that $t \ll d$

Therefore

- The model consists of the generative model $p(x|z)$ given a fixed prior $p(z)$

Bernoulli MLP as decoder

In this case, let $p_\Theta(x|z)$ be a multivariate Bernoulli

- The probability is calculated using a fully connected neural network with a single layer

$$\log p(x|z) = \sum_{i=1}^{D} x_i \log y_i + (1 - x_i) \log (1 - y_i)$$

  - where $\boldsymbol{y} = f_\sigma (W_2 \tanh (W_1 z + b_1) + b_2)$ with f_σ is the sigmoidal function
  - with $\Theta = \{W_1, W_2, b_1, b_2\}$ are the weight and biases of the MLP

Gaussian MLP as encoder

We have the following function

$$
\begin{aligned}
\log p\left(z|x\right) =& \log N\left(\mu, \sigma^2 I\right) \\
\text{where } \mu =& W_4 h + b_4 \\
\log \sigma^2 =& W_5 h + b_5 \\
h =& \tanh\left(W_3 x + b_3\right)
\end{aligned}
$$

We have then

Let the prior over isotropic multivariate Gaussian

$$p_{\Theta}(z) = N(0, I)$$

- Note that in this case, the prior lacks parameters.

We have then

Let the prior over isotropic multivariate Gaussian

$$p_\Theta(z) = N(0, I)$$

- Note that in this case, the prior lacks parameters.

We let $p_\Theta(x|z)$

- Bernoulli whose distribution parameters are computed from z with a MLP

We have then

Let the prior over isotropic multivariate Gaussian

$$p_{\Theta}(z) = N(0, I)$$

- Note that in this case, the prior lacks parameters.

We let $p_{\Theta}(x|z)$

- Bernoulli whose distribution parameters are computed from z with a MLP

Note

- Note the true posterior $p_{\Theta}(z|x)$ is in this case intractable.

Now for our estimation

We have then the following

$$\log q_{\phi}\left(\boldsymbol{z}|\boldsymbol{x}\right) = \log N\left(\boldsymbol{\mu}, \sigma^2 I\right)$$

- where the mean and s.d. of the approximate posterior, $\boldsymbol{\mu}$ and σ , are outputs of the encoding MLP, i.e. nonlinear functions of data point $\boldsymbol{x}$ and the variational parameters ϕ

Now for our estimation

We have then the following

$$\log q_{\phi}\left(\boldsymbol{z}|\boldsymbol{x}\right) = \log N\left(\boldsymbol{\mu}, \sigma^2 I\right)$$

- where the mean and s.d. of the approximate posterior, $\boldsymbol{\mu}$ and σ , are outputs of the encoding MLP, i.e. nonlinear functions of data point $\boldsymbol{x}$ and the variational parameters ϕ

Then, we sample $z_{i,l} \sim q_{\phi}\left(z|x_i\right)$ using the fact that we have a diagonal covariance $\sigma^2 I$

$$z_{i,l} = \mu_i + \sigma_i \odot \epsilon_l \text{ and } \epsilon_l \sim N\left(0, I\right)$$

Then, we have that

In this model both

- $p_{\Theta}(z)$ and $q_{\phi}(z|x_i)$ are Gaussian

Then, we have that

In this model both

- $p_{\Theta}(z)$ and $q_{\phi}(z|x_i)$ are Gaussian

The resulting estimator for this model comes from the classic where the l is the meaning of multiple samples of z_i

$$\mathcal{L}(\Theta,\phi|x_i) = D_{KL}\left(q_{\phi}(z|x_i)\,||p_{\Theta}(z)\right) + \underbrace{\frac{1}{L}\sum_{l=1}^{L} -\log p_{\theta}(x_i|z_{i,l})}_{E[-\log p_{\theta}(x_i|z_i)]}$$

- Basically the negative of $-\int_{\mathcal{Z}} q_{\phi}(z)\log\left(\frac{p_{\Theta}(x,z_i)}{q_{\phi}(z)}\right)dx + \mathcal{T}(\Theta,\phi|z_i)$ which we want to maximize but when minimizing we take the negative.

Therefore

We have that $q_\phi\left(z|x_i\right) = N\left(z|\mu\left(x\right), \sigma^2\left(x\right)I\right)$

- Where at the encoder

$$
\begin{aligned}
\mu =& W_4 x + b_4 \\
\log\sigma^2 =& W_5 x + b_5
\end{aligned}
$$

Therefore

We have that $q_{\phi}\left(z|x_i\right) = N\left(z|\mu\left(x\right), \sigma^2\left(x\right)I\right)$

- Where at the encoder

$$\begin{aligned} \mu =& W_4 x + b_4 \\ \log \sigma^2 =& W_5 x + b_5 \end{aligned}$$

The prior of $p\left(z\right) \approx N\left(z|0, I\right)$

- Then, the KL divergence is

$$D_{KL}\left(q_{\phi}\left(z|x\right)||p_{\Theta}\left(z\right)\right) = E_{q_{\phi}\left(z|x\right)}\left[\log \frac{q_{\phi}\left(z|x\right)}{p_{\Theta}\left(z\right)}\right]$$

Therefore

We have that $q_\phi(z|x_i) = N(z|\mu(x), \sigma^2(x) I)$

- Where at the encoder

$$\begin{aligned}\mu &= W_4 x + b_4\\ \log\sigma^2 &= W_5 x + b_5\end{aligned}$$

The prior of $p(z) \approx N(z|0, I)$

- Then, the KL divergence is

$$D_{KL}(q_\phi(z|x) || p_\Theta(z)) = E_{q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\Theta(z)}\right]$$

Expand the logarithm

$$\log \frac{q_\phi(z|x)}{p_\Theta(z)} = \log q_\phi(z|x) - \log p_\Theta(z)$$

Given that we have Gaussian's

Assume the encoder outputs a diagonal Gaussian distribution

$$\log q_{\phi}\left(\boldsymbol{z}|\boldsymbol{x}\right) = -\frac{1}{2}\left(d\log 2\pi + d\log\sigma^{2} + \frac{\left\|\boldsymbol{z}-\boldsymbol{\mu}\right\|^{2}}{\sigma^{2}}\right)$$

$$p_{\Theta}\left(\boldsymbol{z}\right) = -\frac{1}{2}\boldsymbol{z}^{T}\boldsymbol{z} - \frac{d}{2}\log 2\pi$$

Given that we have Gaussian's

Assume the encoder outputs a diagonal Gaussian distribution

$$\log q_{\phi}\left(\boldsymbol{z}|\boldsymbol{x}\right) = -\frac{1}{2}\left(d\log 2\pi + d\log\sigma^{2} + \frac{\left\|\boldsymbol{z}-\boldsymbol{\mu}\right\|^{2}}{\sigma^{2}}\right)$$

$$p_{\Theta}\left(\boldsymbol{z}\right) = -\frac{1}{2}\boldsymbol{z}^{T}\boldsymbol{z} - \frac{d}{2}\log 2\pi$$

Putting all together, we have

$$D_{KL}\left(q_{\phi}\left(z|x_{i}\right)||p_{\Theta}\left(z\right)\right) = E_{q_{\phi}\left(z|x_{i}\right)}\left[-\frac{1}{2}d\log 2\pi - \frac{1}{2}d\log\sigma^{2} - \frac{1}{2}\frac{\left\|\boldsymbol{z}-\boldsymbol{\mu}\right\|^{2}}{\sigma^{2}} + \frac{1}{2}z^{T}z + \frac{d}{2}\log 2\pi\right]$$

We have the following equality's on the Expectation

We have the following

$$E_{q_\phi(z|x_i)}\left[\|\boldsymbol{z}-\boldsymbol{\mu}\|^2\right] = E_{q_\phi(z|x_i)}\left[tr\left([\boldsymbol{z}-\boldsymbol{\mu}]\,[\boldsymbol{z}-\boldsymbol{\mu}]^T\right)\right] = tr\left(\sigma^2 I\right) = d\sigma^2$$

$$E_{q_\phi(z|x_i)}\left[z^T z\right] = \mu^T\mu + tr\left(\sigma I\right) = \mu^T\mu + d\sigma^2$$

Therefore, we apply the expected value

We have the following

$$D_{KL}\left(q_{\phi}\left(z|x_i\right)||p_{\Theta}\left(z\right)\right) = -\frac{1}{2}\log 2\pi - \frac{1}{2}\log\sigma^2 - \frac{1}{2}\frac{d\sigma^2}{\sigma^2} + \frac{1}{2}\mu^T\mu + \frac{1}{2}d\sigma^2 + \frac{d}{2}\log 2\pi$$

$$\approx -\frac{1}{2}d\log\sigma^2 - \frac{d}{2} + \frac{1}{2}\sum_{i=1}^{d}\mu_i^2 + \frac{1}{2}d\sigma^2$$

Therefore, we apply the expected value

We have the following

$$D_{KL}\left(q_{\phi}\left(z|x_i\right)||p_{\Theta}\left(z\right)\right) = -\frac{1}{2}\log 2\pi - \frac{1}{2}\log\sigma^2 - \frac{1}{2}\frac{d\sigma^2}{\sigma^2} + \frac{1}{2}\mu^T\mu + \frac{1}{2}d\sigma^2 + \frac{d}{2}\log 2\pi$$

$$\approx -\frac{1}{2}d\log\sigma^2 - \frac{d}{2} + \frac{1}{2}\sum_{i=1}^{d}\mu_i^2 + \frac{1}{2}d\sigma^2$$

Which can be seen as with $\sigma_i^2 = \sigma^2$

$$D_{KL}\left(q_{\phi}\left(z|x_i\right)||p_{\Theta}\left(z\right)\right) = -\frac{1}{2}\sum_{i=1}^{d}\left(\log\sigma_i^2 + 1 - \frac{1}{2}\mu_i^2 + \sigma_i^2\right)$$

Therefore, we apply the expected value

We have the following

$$D_{KL}\left(q_\phi\left(z|x_i\right)||p_\Theta\left(z\right)\right) = -\frac{1}{2}\log 2\pi - \frac{1}{2}\log\sigma^2 - \frac{1}{2}\frac{d\sigma^2}{\sigma^2} + \frac{1}{2}\mu^T\mu + \frac{1}{2}d\sigma^2 + \frac{d}{2}\log 2\pi$$

$$\approx -\frac{1}{2}d\log\sigma^2 - \frac{d}{2} + \frac{1}{2}\sum_{i=1}^{d}\mu_i^2 + \frac{1}{2}d\sigma^2$$

Which can be seen as with $\sigma_i^2 = \sigma^2$

$$D_{KL}\left(q_\phi\left(z|x_i\right)||p_\Theta\left(z\right)\right) = -\frac{1}{2}\sum_{i=1}^{d}\left(\log\sigma_i^2 + 1 - \frac{1}{2}\mu_i^2 + \sigma_i^2\right)$$

Finally, what we want to minimize this the ELBO will be maximized

$$\mathcal{L}\left(\Theta,\phi|x_i\right) = \underbrace{-\frac{1}{2}\sum_{i=1}^{d}\left(1+\log\left(\sigma_i^2\right)-\left(\mu_i\right)^2-\left(\sigma_i\right)^2\right)}_{D_{KL}\left(q_\phi(z|x_i)||p_\Theta(z)\right)} + \underbrace{\frac{1}{L}\sum_{l=1}^{L}-\log p_\theta\left(x_i|z_{i,l}\right)}_{E[-\log p_\theta(x|z)]}$$

Now what about the real Loss function?

We can do the following if we think that $E\left[\log p_{\theta}\left(x|z\right)\right]$ is the reconstruction log of the data i.e $x \sim p_{\theta}\left(x|z\right)$

- We can see that $p_{\theta}\left(x|z\right)$, if it is modeled as a Bernoulli distribution because you rebuild it or not

$$p_{\theta}\left(x|z\right) = p^{x}\left(1-p\right)^{1-x}$$

Now what about the real Loss function?

We can do the following if we think that $E\left[\log p_{\theta}\left(x|z\right)\right]$ is the reconstruction log of the data i.e $x \sim p_{\theta}\left(x|z\right)$

- We can see that $p_{\theta}\left(x|z\right)$, if it is modeled as a Bernoulli distribution because you rebuild it or not

$$p_{\theta}\left(x|z\right) = p^{x}\left(1-p\right)^{1-x}$$

Where we use $p = \sigma\left(z\right)$ the output of the decoder

$$p_{\theta}\left(x|z\right) = \sigma\left(z\right)^{x}\left(1-\sigma\left(z\right)\right)^{1-x} \Rightarrow -\log p_{\theta}\left(x|z\right) = -x\log\left(\sigma\left(z\right)\right) - \left(1-x\right)\log\left(1-\sigma\left(z\right)\right)$$

Now what about the real Loss function?

We can do the following if we think that $E\left[\log p_{\theta}\left(x|z\right)\right]$ is the reconstruction log of the data i.e $x \sim p_{\theta}\left(x|z\right)$

- We can see that $p_{\theta}\left(x|z\right)$, if it is modeled as a Bernoulli distribution because you rebuild it or not

$$p_{\theta}\left(x|z\right) = p^{x}\left(1-p\right)^{1-x}$$

Where we use $p = \sigma\left(z\right)$ the output of the decoder

$$p_{\theta}\left(x|z\right) = \sigma\left(z\right)^{x}\left(1-\sigma\left(z\right)\right)^{1-x} \Rightarrow -\log p_{\theta}\left(x|z\right) = -x\log\left(\sigma\left(z\right)\right) - \left(1-x\right)\log\left(1-\sigma\left(z\right)\right)$$

Therefore when you take a minibatch or the $E\left[\log p_{\theta}\left(x_i|z_i\right)\right]$

$$E\left[-\log p_{\theta}\left(x|z\right)\right] = BCE\left[\sigma\left(z\right), x\right]$$

Therefore the final Loss function is

Two parts the divergence and the binary cross entropy

$$\mathcal{L}\left(\Theta, \phi | x_i\right) = \underbrace{-\frac{1}{2}\sum_{i=1}^{d}\left(1 + \log\left(\sigma_i^2\right) - \left(\mu_i\right)^2 - \left(\sigma_i\right)^2\right)}_{D_{KL}\left(q_\phi(z|x_i)||p_\Theta(z)\right)} + \underbrace{BCE\left[\sigma\left(z\right), x\right]}_{E[-\log p_\theta(x|z)]}$$

Therefore the final Loss function is

Two parts the divergence and the binary cross entropy

$$\mathcal{L}\left(\Theta,\phi|x_i\right) = \underbrace{-\frac{1}{2}\sum_{i=1}^{d}\left(1+\log\left(\sigma_i^2\right)-\left(\mu_i\right)^2-\left(\sigma_i\right)^2\right)}_{D_{KL}\left(q_\phi(z|x_i)||p_\Theta(z)\right)} + \underbrace{BCE\left[\sigma\left(z\right),x\right]}_{E[-\log p_\theta(x|z)]}$$

A last comment is the following, this model is not exactly a generative model but

- the $q(z|x)$ (simple and tractable posteriors) must close enough to $N(0,I)$.
- We can sample from $N(0,I)$ to get a input for the decoder and get some generated sample.

Given the inception of the Variational part

It is coming from the work of [7]

- Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg (2006)

Given the inception of the Variational part

It is coming from the work of [7]

- Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg (2006)

Variational Autoencoder (VAE)

- VAE are generative models that attempt to describe data generation through a probabilistic distribution.

Outline

Outline

Generative Model

At the encoder part

- The posterior distribution $q_{\phi}(x|z_i)$ which is derived by the encoder

Generative Model

At the encoder part

- The posterior distribution $q_{\phi}\left(x|z_i\right)$ which is derived by the encoder

Thus, we have

- This is regularized towards a continuous and complete distribution in the shape of the predefined prior of the latent variables $p_{\Theta}\left(x\right)$.

Therefore

Once trained

- One can simply samples random variables from the the same prior, and feed it to the decoder.

Therefore

Once trained

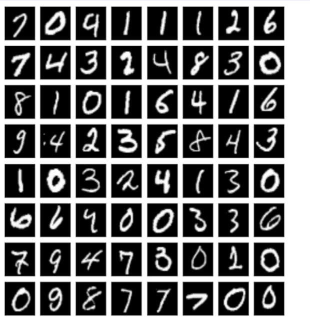
- One can simply samples random variables from the the same prior, and feed it to the decoder.

Since the decoder was trained generate x from $p_{\Theta}(x_i|z)$

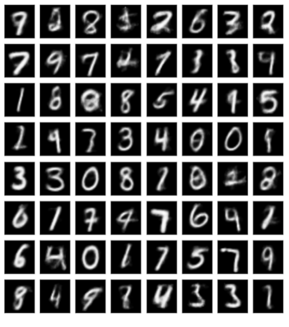
- It would generate a meaningful generated sample

Example

Original Images vs the generated



Sample from the original MNIST dataset



VAE generated MNIST IMAGES

Outline

Consider N images $\{X_l\}_{l=1}^{N}$, with $X_l \in \mathbb{R}^{N_x \times N_y \times C}$

We introduce the decoder

- To introduce the image decoder (generative model) in its simplest form, we first consider a decoder with $L = 2$ layers.

Code Entry $\Downarrow$

$$Layer\ 2{:}\widetilde{S}^{n,2} = \sum_{k_2=1}^{K_2} D^{k_2,2} * S^{n,k_2,2} \tag{1}$$

$$Unpool\ {:}S^{n,1} = \text{unpool}\left(\widetilde{S}^{n,2}\right) \tag{2}$$

$$Layer\ 1{:}\widetilde{S}^{n,1} = \sum_{k_1=1}^{K_1} D^{k_1,2} * S^{n,k_1,2} \tag{3}$$

$$\text{Data Generation:} X_l \sim N\left(\widetilde{S}^{n,1}, \alpha_0^{-1} I\right) \tag{4}$$

Notation

The following are 3D tensors

- $D^{k_l,l}$
- $S^{n,l}$
- $\widetilde{S}^{n,l}$

Notation

The following are 3D tensors

- $D^{k_l,l}$
- $S^{n,l}$
- $\widetilde{S}^{n,l}$

2D Activation Maps

- $S^{n,k,l}$ as slices of 3D tensors $S^{n,l}$

Notation

The following are 3D tensors

- $D^{k_l,l}$
- $S^{n,l}$
- $\widetilde{S}^{n,l}$

2D Activation Maps

- $S^{n,k,l}$ as slices of 3D tensors $S^{n,l}$

Finally

- $D^{k,l} * S^{n,k,l} =$ each of the K_{l-1} "slices" of $D^{k_l,l}$ is convolved with the spatially-dependent $S^{n,k,l}$

Meaning for equation (4)

It indicates

1. $E[X_l] = \widetilde{S}^{n,1}$
2. $X_l - E(X_l)$ is iid zero mean Gaussina with preccision α_0

The Stochastic Unpooling

$S^{n,k_l,l}$ is partitioned into contiguous $p_x \times p_y$ pooling blocks

- $z_{i,j}^{n,k,1} \in \{0,1\}^{p_x p_y}$ be a vector of $p_x p_y - 1$ zeros and a single one.

The Stochastic Unpooling

$S^{n,k_l,l}$ is partitioned into contiguous $p_x \times p_y$ pooling blocks

- $z_{i,j}^{n,k,1} \in \{0,1\}^{p_x p_y}$ be a vector of $p_x p_y - 1$ zeros and a single one.

$z_{i,j}^{n,k,1}$ corresponds to pooling bloock (i,j) in $S^{n,k_1,1}$

- The location of the non-zero element of $z_{i,j}^{n,k,1}$ identifies the location of the single non-zero element in the corresponding pooling block of $S^{n,k_1,1}$.

Therefore

The non-zero element in pooling block (i, j)

- $S^{n,k_1,1}$ is set to $\widetilde{S}^{n,k_1,2}_{i,j}$

Therefore

The non-zero element in pooling block (i, j)

- $S^{n,k_1,1}$ is set to $\widetilde{S}_{i,j}^{n,k_1,2}$

Thus, we have the following multinomial for the unpooling

$$z_{i,j}^{n,k_1,1} \sim \text{Mult}\left(1; \frac{1}{p_x p_y}, ..., \frac{1}{p_x p_y}\right)$$

Finally, the Encoder

Something Notable

$$
\begin{aligned}
&\text{Code Entry} \Downarrow \\
Layer\ 1 &: \widetilde{C}^{n,k_1,1} = X_l *_s F^{k_1,1} & (5) \\
Pool\ &: C^{n,1} \sim pool & (6) \\
Layer\ 2 &: \widetilde{C}^{n,k_2,2} = C^{n,1} *_s F^{k_2,2} & (7) \\
\text{Data Generation} &: s_n \sim N\left(\mu_0 \widetilde{C}^{n,2}, diag\left(\sigma_\phi\left[\widetilde{C}^{n,2}\right]\right)\right) & (8)
\end{aligned}
$$

This was trained in a semisupervised way

By using a loss function

$$\mathcal{L}_{\phi,\alpha,\psi}\left(X,Y\right) = \xi\left\{E_{q_{\phi}(s|X)}\left[\log p_{\psi}\left(Y|s\right)\right]\right\} + \underbrace{E_{q_{\phi}(s,z|X)}\left[\log p_{\alpha}\left(X,s,z\right) - \log q_{\phi}\left(s,z|X\right)\right]}_{\mathcal{U}_{\phi,\alpha}(X)}$$

This was trained in a semisupervised way

By using a loss function

$$
\begin{aligned}
\mathcal{L}_{\phi,\alpha,\psi}(X,Y) = & \xi \left\{ E_{q_\phi(s|X)} \left[\log p_\psi (Y|s) \right] \right\} \\
& + \underbrace{E_{q_\phi(s,z|X)} \left[\log p_\alpha (X,s,z) - \log q_\phi (s,z|X) \right]}_{\mathcal{U}_{\phi,\alpha}(X)}
\end{aligned}
$$

The lower bound for the entire dataset is then

$$
\mathcal{J}_{\phi,\alpha,\psi} = \sum_{(X,Y)\in\mathcal{D}_c} \mathcal{L}_{\phi,\alpha,\psi}(X,Y) + \sum_{X\in\mathcal{D}_u} \mathcal{U}_{\phi,\alpha}(X)
$$

Benchmark for Classic

We have

Table 1: Classification error (%) and testing time (ms per image) on benchmarks.

Method	MNIST		CIFAR-10		CIFAR-100		Caltech 101		Caltech 256	
	test error	test time	test error	test time	test error	test time	test error	test time	test error	test time
Gibbs [8]	0.37	3.1	8.21	10.4	34.33	10.4	12.87	50.4	29.50	52.3
MCEM [8]	0.45	0.8	9.04	1.1	35.92	1.1	13.51	8.8	30.13	8.9
VAE-d	0.42	**0.007**	10.74	**0.02**	37.96	**0.02**	14.79	**0.3**	32.18	**0.3**
VAE (Ours)	0.38	**0.007**	8.19	**0.02**	35.01	**0.02**	11.99	**0.3**	29.33	**0.3**

Method	ImageNet 2012			ImageNet Pretrained for			
				Caltech 101		Caltech 256	
	top-1 error	top-5 error	test time	test error	test time	test error	test time
MCEM [8]	37.9	16.1	14.4	6.85	14.1	22.10	14.2
VAE (Ours)	38.2	15.7	**1.0**	6.91	**0.9**	22.53	**0.9**

Other examples

We have a long list

- Autoencoders for classification
- Autoencoders for clustering
- Autoencoders for anomaly detection
- Autoencoders for recommendation systems

- D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *CoRR*, vol. abs/2003.05991, 2020.
- P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning* (I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, eds.), vol. 27 of *Proceedings of Machine Learning Research*, (Bellevue, Washington, USA), pp. 37–49, PMLR, 02 Jul 2012.
- P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima," *Neural networks*, vol. 2, no. 1, pp. 53–58, 1989.
- S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.

D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013.

C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*.
Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.