Introduction to variational autoencoders

Abstract

Variational autoencoders are interesting generative models, which combine ideas from deep learning with statistical inference. They can be used to learn a low dimensional representation \( Z \) of high dimensional data \( X \) such as images (of e.g. faces). In contrast to standard auto encoders, \( X \) and \( Z \) are random variables. It’s therefore possible to sample \( X \) from the distribution \( P(X|Z) \), thus creating e.g. images of faces, MNIST Digits, or speech.


I will also show some code. A TensorFlow notebook can be found at: https://github.com/oduerr/dl_tutorial/blob/master/tensorflow/vae/vae_demo.ipynb
Introduction to variational autoencoders

Oliver Dürr

Datalab-Lunch Seminar Series
Winterthur, 11 May, 2016
Motivation: Generating Faces

Other examples
- random faces
- MNIST
- Speech

just google vae…

These are not part of the trainingset!

https://www.youtube.com/watch?v=XNZIN7Jh3Sg
Motivation: Generating Hand Written Digits

http://www.dpkingma.com/sgvb_mnist_demo/demo.html
Idea
Recap: Auto Encoders (‘classical’)
Recap: Linear Regression

Most people think of linear regression as points and a straight line:

\[ x = \beta_0 + z\beta_1 \]

Strange axis names, to be compatible with later notation.
Recap: Linear Regression

Statisticians additionally have $P_\theta(X \mid Z)$

Benefits of having an error model:
- How likely is a data point
- Confidence bounds
- Compare models

$P(\sigma, \beta_0, \beta_1)(x \mid z) \propto e^{-\frac{(x-\hat{x}(z))^2}{2\sigma^2}}$

$\hat{x}(z) = \beta_0 + z\beta_1$

Strange axis names, to be compatible with later notation.

credit: wikipedia
Recap: Linear Regression (as a graphical model)

Statisticians additionally have $P_{\theta}(X \mid Z)$

$\hat{x}(z) = \beta_0 + z\beta_1$

$P_{(\sigma, \beta_0, \beta_1)}(x \mid z) \propto e^{-\frac{(x-\hat{x}(z))^2}{2\sigma^2}}$

Plate notation of a graphical model to show off

See Beates talk on Causal inference with graphical models
Going from $\mathbb{R}^1$ to $\mathbb{R}^{10000}$

$X \in \mathbb{R}^{10000}$

$Z \in \mathbb{R}^n$ typically $n = 2, ..., 50$ 

“latent space”

Is $\mathbb{R}^2$ “big enough” to create images from $\mathbb{R}^{10000}$? ...

credit: wikipedia, digits: https://www.youtube.com/watch?v=gyB8RegAlQ
Manifold hypothesis

- \( \mathbf{X} \) high dimensional vector
- Data is concentrated around a low dimensional manifold
- Hope finding a representation \( \mathbf{Z} \) of that manifold.
Variational auto encoders (idea of low dim manifold)

1D
Low Dimensional representation a line

2D
High Dimensional (number of pixels)

3D

credit: http://www.deeplearningbook.org/
Variational auto encoders (idea of low dim manifold)

Examples of successful unfolding (2D $\rightarrow \mathbb{R}^{28 \times 28}$, $\mathbb{R}^{20 \times 26}$)

**MNIST:**

**Frey Face dataset:**

2000 pictures of Brendan Frey (20x26)

How did they do that?
Variational Autoencoders ("history")

Simultaneously discovered by


Alternative approach (for binary distributions)

  - Has a more information theoretic ansatz (codings length)
  - Lecture given at Nando de Freitas ML Course (University of Oxford) (a bit hand waving argument but with nice examples)

- We focus on the approach as in Kingma, Welling
**Principle Idea decoder network (graphical model)**

- We have a set of N-observations (e.g. images) \( \{x^{(1)}, x^{(2)}, \ldots, x^{(N)}\} \)
- Complex model parameterized with \( \theta \)
- There is a latent space \( z \) with

\[
z \sim p(z) \quad \text{multivariate Gaussian}
\]

\[
x | z \sim p_\theta(x | z)
\]

Wish to learn \( \theta \) from the N training observations \( x^{(i)} i=1, \ldots, N \)
The model for the decoder network

- For illustration, z one dimensional x 2D
- Want a complex model of distribution of x given z
- Idea: NN + Gaussian (or Bernoulli) here with diagonal covariance $\Sigma$

$$x \mid z \sim N(\mu_x, \sigma^2_x)$$
Training as an autoencoder

Training use maximum likelihood of $p(x)$ given the training data

Problem:

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

Cannot be calculated:

Solution:

- MCMC (too costly)
- Approximate $p(z|x)$ with $q(z|x)$
The model for the encoder network

- A feed forward NN + Gaussian

\[ q_\phi(z \mid x) = \mathcal{N}(z; \mu_z(x), \sigma_z(x)) \]

Just a Gaussian, with diagonal covariance.
The complete auto-encoder

Learning the parameters $\phi$ and $\theta$ via backpropagation

Determining the loss function
Training: Loss Function

• What is (one of the) most beautiful idea in statistics?

• Max-Likelihood, tune $\Phi, \theta$ to maximize the likelihood

• We maximize the (log) likelihood of a given “image” $x^{(i)}$ of the training set. Later we sum over all training data (using minibatches)
Lower bound of likelihood (mathematical sleight of hand)

Likelihood, for an image $x^{(i)}$ from training set. Writing $x=x^{(i)}$ for short.

\[ L = \log (p(x)) \]

\[ = \sum_z q(z|x) \log (p(x)) \]

\[ = \sum_z q(z|x) \log \left( \frac{p(z, x)}{p(z|x)} \right) \]

\[ = \sum_z q(z|x) \log \left( \frac{p(z, x)}{q(z|x)} \frac{q(z|x)}{p(z|x)} \right) \]

\[ = \sum_z q(z|x) \log \left( \frac{p(z, x)}{q(z|x)} \right) + \sum_z q(z|x) \log \left( \frac{q(z|x)}{p(z|x)} \right) \]

\[ = L^v + D_{KL} (q(z|x) \parallel p(z|x)) \]

\[ \geq L^v \]

$D_{KL}$ KL-Divergence $\geq 0$ depends on how good $q(z|x)$ can approximate $p(z|x)$

$L^v$ “lower variational bound of the (log) likelihood” $L^v = L$ for perfect approximation
Approximate Inference (rewriting $L^y$)

\[ L^y = \sum_z q(z|x) \log \left( \frac{p(z, x)}{q(z|x)} \right) \]

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

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

\[ = -D_{KL} (q(z|x) \| p(z)) + \mathbb{E}_{q(z|x)} (\log (p(x|z))) \]

\[ = -D_{KL} (q(z|x^{(i)}) \| p(z)) + \mathbb{E}_{q(z|x^{(i)})} (\log (p(x^{(i)}|z))) \]

with \( p(z, x) = p(x|z) p(z) \)

Putting in \( x^{(i)} \) for \( x \)

Regularisation
\( p(z) \) is usually a simple prior \( N(0,1) \)

Reconstruction quality, \( \log(1) \) if \( x^{(i)} \) gets always reconstructed perfectly \( (z \text{ produces } x^{(i)}) \)

Example \( x^{(i)} \)

\[ q_\phi(z|x^{(i)}) \]

\[ p_\theta(x^{(i)}|z) \]
Calculation the regularization

Use $N(0,1)$ as prior for $p(z)$

$q(z|x^{(i)})$ is Gaussian with parameters $(\mu^{(i)}, \sigma^{(i)})$ determined by NN

$$-D_{KL} \left( q(z|x^{(i)}) \| p(z) \right) = \frac{1}{2} \sum_{j=1}^{J} \left( 1 + \log(\sigma_{z_j}^{(i)^2}) - \mu_{z_j}^{(i)^2} - \sigma_{z_j}^{(i)^2} \right)$$
Sampling to calculate $\mathbb{E}_{q(z|x^{(i)})} \left( \log(p(x^{(i)}|z)) \right)$

Approximating $\mathbb{E}_{q(z|x^{(i)})}$ with sampling form the distribution $q(z|x^{(i)})$

With $z^{(i,l)}$ $l = 1, 2, \ldots L$ sampled from $z^{(i,l)} \sim q(z|x^{(i)})$

$$L^v = -D_{KL} (q(z|x^{(i)}) \| p(z)) + \mathbb{E}_{q(z|x^{(i)})} \left( \log(p(x^{(i)}|z)) \right)$$

$$L^v \approx -D_{KL} (q(z|x^{(i)}) \| p(z)) + \frac{1}{L} \sum_{i=1}^{L} \log(p(x^{(i)}| z^{(i,l)}))$$

Example $x^{(i)}$

$$\log(p_{\theta}(x^{(i)}|z^{(i,1)})) \quad \text{where} \quad z^{(i,1)} \sim N(\mu_{Z}^{(i)}, \sigma_{Z}^{2(i)})$$

$$\ldots$$

$$\log(p_{\theta}(x^{(i)}|z^{(i,L)})) \quad \text{where} \quad z^{(i,L)} \sim N(\mu_{Z}^{(i)}, \sigma_{Z}^{2(i)})$$

$L$ is often very small (often just $L=1$)
One last trick

Backpropagation not possible through random sampling!

Sampling (reparametrization trick)

\[ z^{(i,l)} \sim N(\mu^{(i)}, \sigma^{2(i)}) \]

\[ z^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \odot \varepsilon_i \quad \varepsilon_i \sim N(0,1) \]

Writing \( z \) in this form, results in a deterministic part and noise.

Cannot back propagate through a random drawn number

\( z \) has the same distribution, but now one can back propagate.

Image from: NIPS Workshop 2015 (Kingma & Welling)
Prior $p(z) \sim N(0,1)$ and $p, q$ Gaussian, extension to $\text{dim}(z) > 1$ trivial

Cost: Regularisation

$$-D_{KL} (q(z|x^{(i)}) \| p(z)) = \frac{1}{2} \sum_{j=1}^{J} \left( 1 + \log(\sigma_{z_j}^{(i)^2}) - \mu_{z_j}^{(i)^2} - \sigma_{z_j}^{(i)^2} \right)$$

Cost: Reproduction

$$-\log(p(x^{(i)} | z^{(i)})) = \sum_{j=1}^{D} \frac{1}{2} \log(\sigma_{x_j}^2) + \frac{(x_j^{(i)} - \mu_{x_j})^2}{2\sigma_{x_j}^2}$$

We use mini batch gradient decent to optimize the cost function over all $x^{(i)}$ in the mini batch

Least Square for constant variance
Use the source Luke

Simple example 2-D distribution

Simple MNIST Example
https://github.com/oduerr/dl_tutorial/blob/master/tensorflow/vae/vae_demo.ipynb
Recent developments of VAE
Recent developments in VAE / generative models (subjective overview)

- Authors of VAE Amsterdam University and Google DeepMind teamed up and wrote a paper on semi-supervised learning:

- Karl Gregor et al. extended the (binary autoencoder) with attention
  - DRAW: A Recurrent Neural Network For Image Generation
  - https://www.youtube.com/watch?v=Zt-7MI9eKEo

- Adversial networks as a non-statistical way to generate high dimensional data
  - Play a game:
    - Fist network invents some data $P(X)$ to fool second network
    - Second network tells if first network is a liar.
Semisupervised learning

Slide: Kingma, Rezendem Nohamed, Welling
Semisupervised learning

VAEs are SOTA on semi-supervised learning on MNIST

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AtlasRBF (Pitelis et al., 2014)</td>
<td>8.10% (±0.95)</td>
</tr>
<tr>
<td>Deep Generative Model (M1+M2) (Kingma et al., 2014)</td>
<td>3.33% (±0.14)</td>
</tr>
<tr>
<td>Virtual Adversarial (Miyato et al., 2015)</td>
<td>2.12%</td>
</tr>
<tr>
<td>Ladder (Rasmus et al., 2015)</td>
<td>1.06% (±0.37)</td>
</tr>
<tr>
<td>Auxiliary Deep Generative Model (1 MC)</td>
<td>2.25% (± 0.08)</td>
</tr>
<tr>
<td>Auxiliary Deep Generative Model (10 MC)</td>
<td>0.96% (± 0.02)</td>
</tr>
</tbody>
</table>

That’s 10 per class!

“Improving Semi-Supervised Learning with Auxiliary Deep Generative Models”
[Maaløe, Sønderby, Sønderby and Winter, 2015]
Thank you, questions?