Conditional Variational Auto-encoder

Team

Linan Zhang
Jintian Cai

Boston University Computer Science Department

BOSTON UNIVERSITY

Motivation

- Variational Auto-encoder Model:
  - A generative model that generate plausible looking fake samples.
  - VAE generate samples $x' \in X$ such that:

$$P(x) = \int_{z} P_{\theta}(x' \mid z) P(z)$$

  - Minimizing loss function

$$L^{V} = -D_{KL}\left(q(z \mid x) \middle| p(z)\right) + E_{q(z \mid x)} \log(p(x \mid z))$$

  - VAE could not control the model output.

Goal

- Build up an auto-encoder model that could generate specific output.
- E.g:
  - A digit number 6 rather than a random number 0-9 from MNIST dataset
  - A cute cat rather than a random animal

Conditional Variational Auto-encoder

- Kihyuk Sohn, Honglak Lee, and Xinchen Yan. 'Learning structured output representation using deep conditional generative models.' In *NIPS*, 2015.
- Carl Doersch, 'Tutorial on Variational Autoencoders', *arXiv:1606.05908, 2016*

Conditional Variational Auto-encoder

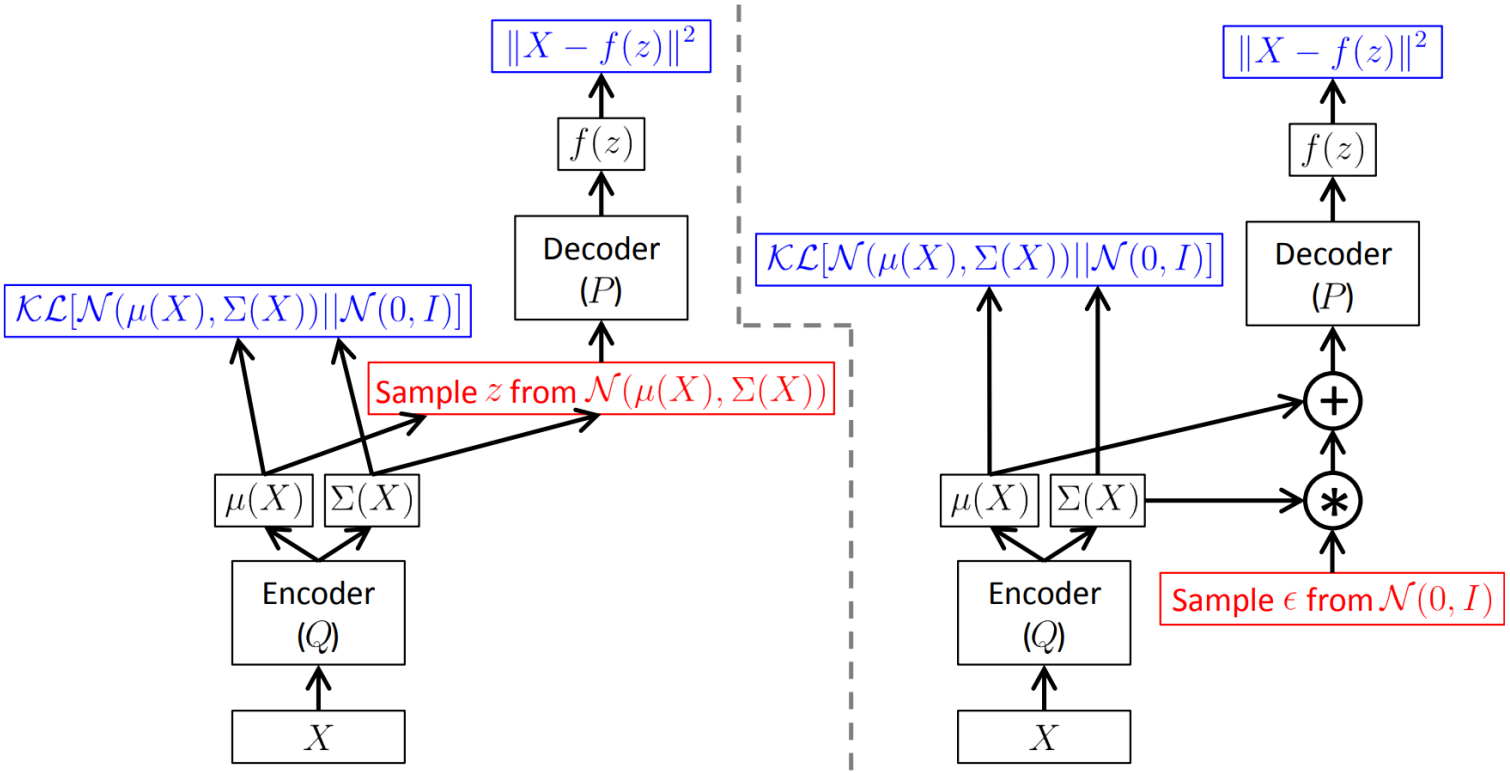
- Hole filling: given an existing image where a user has removed an unwanted object, the goal is to fill in the hole with plausible-looking pixels.
- Simply conditioning the entire generative process on an input.
- Minimizing loss function:

$$L_T = L_{KL} + L_{CE}$$

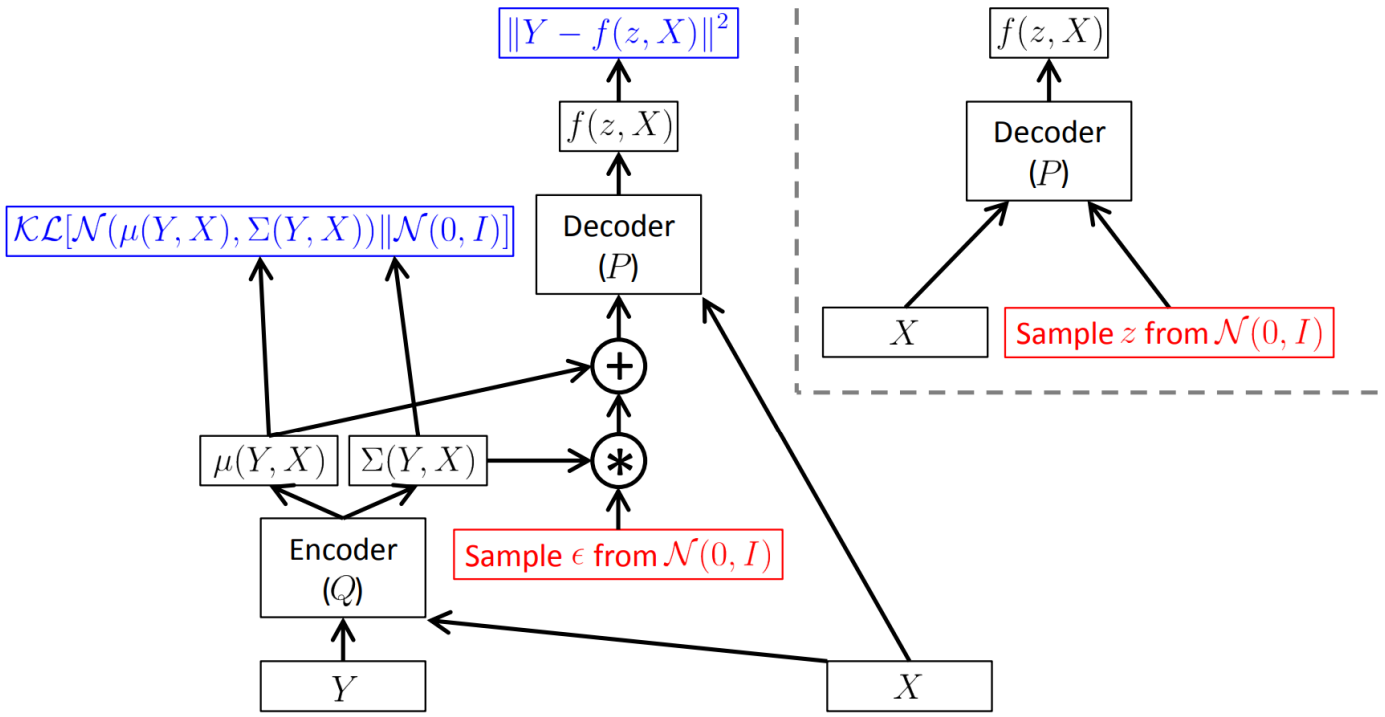
Conditional Variational Auto-encoder

- Set up $Z \sim N(0,1)$
- Define the model
  $P(y|x) = N(f(z,x),\ \sigma^2 I)$
- Minimizing loss function
  $$L^V = -D_{KL}\left(q(z|y,x)\,\middle|\,p(z)\right) + E_{q(z|y,x)}\log(p(y|z,x))$$

VAE Architecture



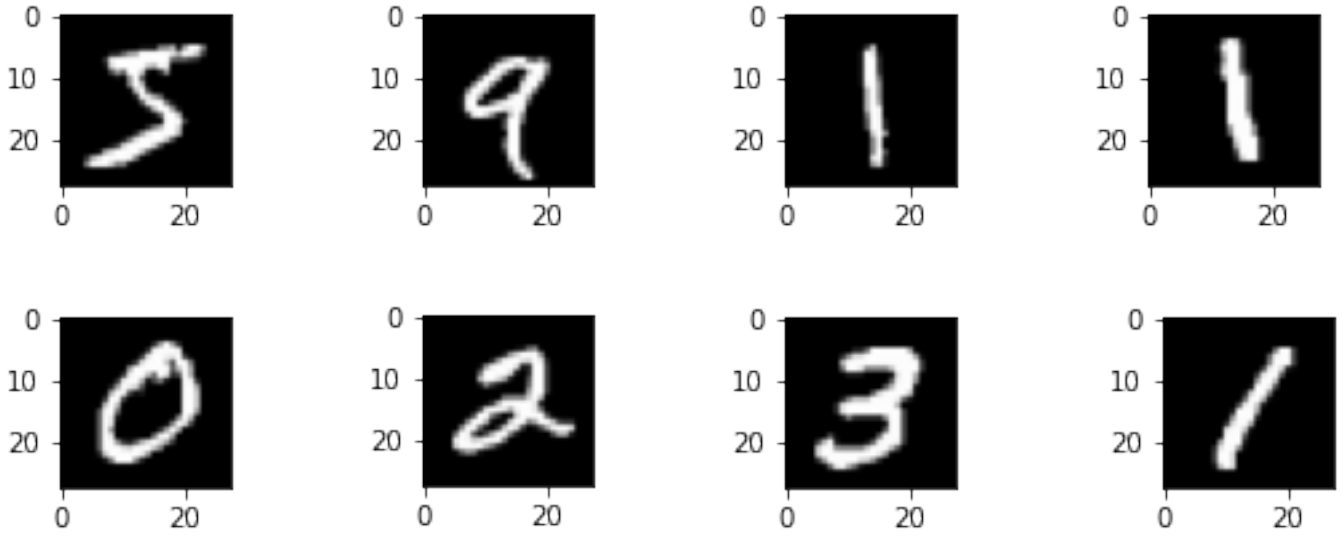
CVAE Architecture



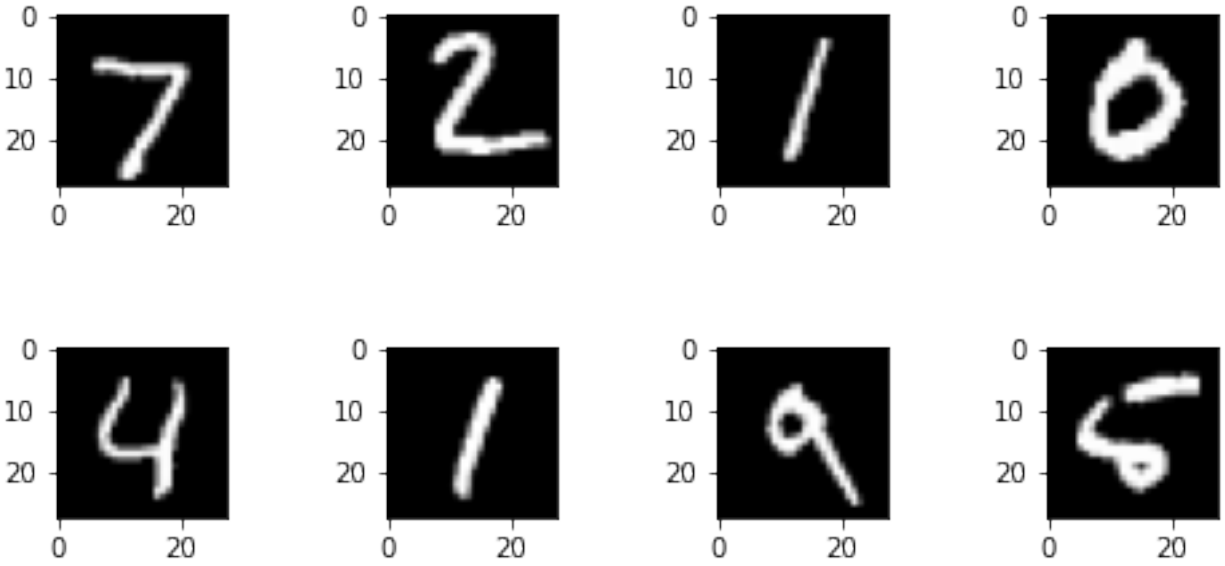
Experimental Setup

- Use the classic MNIST data from http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
- Set up batch size = 100
- Activation function: Relu(x) = max(0,x)
- Optimizer: Adam (an extending of SGD)
- Epochs = 50
- Code adapted from: https://github.com/lyeoni/pytorch-mnist-CVAE/blob/master/pytorch-mnist-CVAE.ipynb

Training Set



Test Set



Neural Network

Architecture

```
# encoder part
self.fc1 = nn.Linear(x_dim + c_dim, h_dim1)
self.fc2 = nn.Linear(h_dim1, h_dim2)
self.fc31 = nn.Linear(h_dim2, z_dim)
self.fc32 = nn.Linear(h_dim2, z_dim)
# decoder part
self.fc4 = nn.Linear(z_dim + c_dim, h_dim2)
self.fc5 = nn.Linear(h_dim2, h_dim1)
self.fc6 = nn.Linear(h_dim1, x_dim)
```

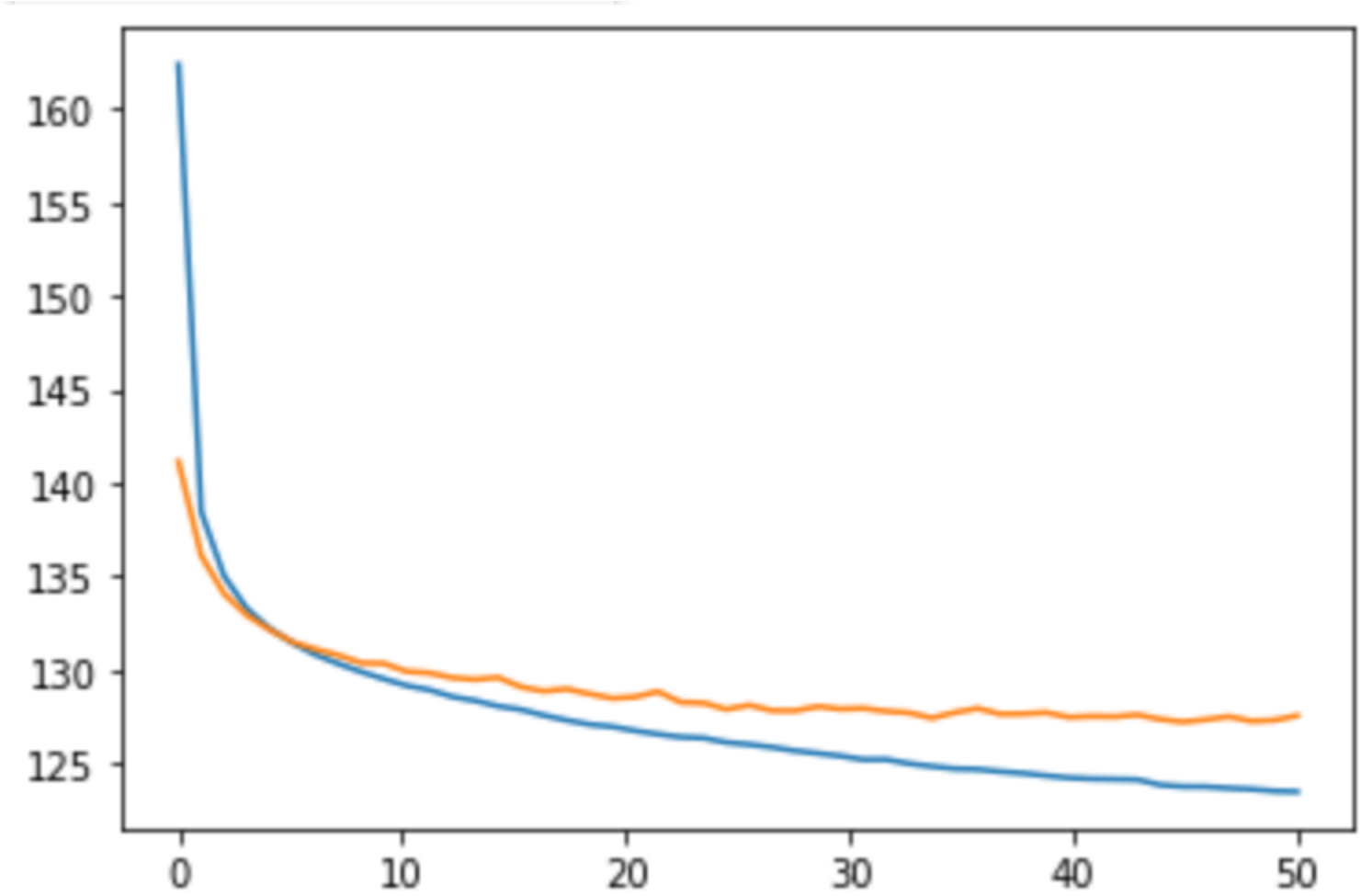
Experimental Results

Final Training & Test loss

```
Train Epoch: 50 [0/60000 (0%)]  Loss: 111.991074
Train Epoch: 50 [10000/60000 (17%)]     Loss: 122.893184
Train Epoch: 50 [20000/60000 (33%)]     Loss: 122.242119
Train Epoch: 50 [30000/60000 (50%)]     Loss: 120.423447
Train Epoch: 50 [40000/60000 (67%)]     Loss: 127.631328
Train Epoch: 50 [50000/60000 (83%)]     Loss: 121.239258
====> Epoch: 50 Average loss: 123.3046
====> Test set loss: 127.5173
```

Experimental Results

Overall Training & Test loss



Experimental Results

Test loss

```
Train Epoch: 48 [0/60000 (0%)]  Loss: 118.912588
Train Epoch: 48 [10000/60000 (17%)]     Loss: 130.376650
Train Epoch: 48 [20000/60000 (33%)]     Loss: 123.381592
Train Epoch: 48 [30000/60000 (50%)]     Loss: 120.482500
Train Epoch: 48 [40000/60000 (67%)]     Loss: 126.858691
Train Epoch: 48 [50000/60000 (83%)]     Loss: 131.365947
====> Epoch: 48 Average loss: 123.5206
====> Test set loss: 127.4892
Train Epoch: 49 [0/60000 (0%)]  Loss: 129.250117
Train Epoch: 49 [10000/60000 (17%)]     Loss: 127.192578
Train Epoch: 49 [20000/60000 (33%)]     Loss: 119.742520
Train Epoch: 49 [30000/60000 (50%)]     Loss: 122.451836
Train Epoch: 49 [40000/60000 (67%)]     Loss: 124.438359
Train Epoch: 49 [50000/60000 (83%)]     Loss: 122.721084
====> Epoch: 49 Average loss: 123.4475
====> Test set loss: 127.6638
Train Epoch: 50 [0/60000 (0%)]  Loss: 111.991074
Train Epoch: 50 [10000/60000 (17%)]     Loss: 122.893184
Train Epoch: 50 [20000/60000 (33%)]     Loss: 122.242119
Train Epoch: 50 [30000/60000 (50%)]     Loss: 120.423447
Train Epoch: 50 [40000/60000 (67%)]     Loss: 127.631328
Train Epoch: 50 [50000/60000 (83%)]     Loss: 121.239258
====> Epoch: 50 Average loss: 123.3046
====> Test set loss: 127.5173
```

Demos

Comparing to VAE

Cite from :
https://github.com/lyeoni/pytorch-mnist-VAE

Github Page

CS523

An application of CVAE on MINIST dataset

author: Linan Zhang, Jintian Cai

email: {lz1018,jtcai}@bu.edu

This project focuses on the Conditional Variantional Auto-encoder, a member of VAE family that is enabled to sampling new data conditional on a specific input.

- Traditional VAE model generates plausible looking fake samples while it could not control the model output.
- The goal of this project aims to build up an auto-encoder model that could generate specific output.

Conditional Variational Auto-encoder (CVAE)

In a one-sentence description, CVAE proceeds as conditioning the entire generative process on an input with the loss function as a sumation of KL divergence and cross-entropy loss.

- Dependencies
    - pytorch
    - matlabplotlib
    - keras

Reference:

[1] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. ‘Learning structured output representation using deep conditional generative models.’ *In NIPS*, 2015.
[2] Carl Doersch, ‘Tutorial on Variational Autoencoders’, *arXiv:1606.05908*, 2016
[3] Deng, L., 'The mnist database of handwritten digit images for machine learning research.' *IEEE Signal Processing Magazine*, 29(6), 141–142, 2012.
[4] https://github.com/lyeoni/pytorch-mnist-CVAE
[5] https://github.com/lyeoni/pytorch-mnist-VAE

Thank you!

QUESTIONS?