GR5242 Final Project

VAE For Downstream Classification

Qiwen Gao qg2165
Xiyi Yan xy2380
Haoyu Zhang hz2563
Nuanjun Zhao nz2295

1 Abstract

Variational auto-encoder(VAE) is a useful tool for mapping data to a low-dimensional space through neural network and capture sources of variations in the data. It is frequently utilized in the task of downstream classification. In this project, our group develops VAE and applies it on the Fashion-MNIST dataset. We also train other classifiers, including VAE with logistic regression, VAE with simple CNN and multilayer CNN, then test, and compare their performances on downstream classification. We tested the classifiers and conclude that Multilayer CNN has the best testing error = 0.1135 with training size = 60,000.

2 Introduction

Variational auto-encoder(VAE) is a nonlinear dimension reduction technique. Similar to a standard auto-encoder, VAE is composed of an encoder and a decoder, and is trained to minimize the reconstruction error between the processed encoded-decoded data and the original data. In VAE, instead of encoding the inputs as single points, we take the reconstruction error in a probabilistic sense. By mapping the high-dimensional data into a low dimension through the nonlinear mapping, or the neural network, in this case, VAE is very useful for the task of downstream classification.

The data we use in this project is the Fashion-MNIST data set, which you can find on https://github.com/zalandoresearch/fashion-mnist. It consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example in this data set is associated with a label from 10 classes. We first implement VAE on the fashion-MINIST data and do sanity check for assuring that the generated VAE performs reasonably well. We check on the distribution learned by the generative model as well as the inference network. After obtaining the well performed VAE, we compare the performances of three methods: (1) VAE+logistic regression, (2) VAE+simple CNN, (3) multilayer CNN. The inputs for the first and second methods are the hidden variables that we get from the VAE, while the input for the third method is the original data. For each method, we train on a small subset of data and a large subset of data, then see how the performances vary in different data sizes. We analyze the results by comparing the test errors of the different methods in various data regimes. The details of the methods and results are discussed in the following parts as shown below.

3 Methods

3.1 Variational auto-encoder (VAE)

The first step of our project is to implement VAE. Refer to the Kingma and Welling article, in VAE, we use a neural network for the probabilistic encoder $q_{\phi}(z|x)$, which is an approximation to the posterior of the generative model $p(x, z)$. Let $p_{\phi}(x|z)$ be a multivariate Gaussian since we are using real-valued data in this project. We assume the true posterior is approximately Gaussian from an approximately diagonal covariance. The variational approximate posterior is a multivariate Gaussian with a diagonal covariance in this form:

$$log q_{\phi}(z|x^{(i)}) = log\ (z; \mu^{(i)}, \sigma^{2(i)})\ .$$

The mean, μ , and the standard deviation σ are outputs of the encoding MLP, which is a fully-connected neural network with a single hidden layer. Then we sample from the posterior $z^{(i,l)} \sim\ q_{\phi}(z|x)$, where $z^{(i,l)} = g_{\phi}(x^{(i)}, \epsilon^{(l)}) = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}$, where $\epsilon^{(l)} \sim (0,)$. Here in this model, both the prior and the posterior are Gaussian. The result of the estimator of the model and datapoint is

$$\mathcal{L}(\theta, \phi; x^{(i)}) \simeq \frac{1}{2} \sum_{j=1}^{J} (1 + log((\sigma^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma^{(i)})^2) + \frac{1}{L} \sum_{l=1}^{L} log\, p_{\theta}(x^{(i)}|z^{(i,l)})\,,$$

where $z^{(i,l)} = g_{\phi}(x^{(i)}, \epsilon^{(l)}) = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}$ and $\epsilon^{(l)} \sim (0,)$.
The decoding object $p_{\theta}(x^{(i)}|z^{(i,l)})$ is Gaussian MLP in this case.

For our VAE, we choose 20 for the latent dimension and run for 10 epochs. After implementing VAE to the fashion-MNIST data set, we perform sanity check to make sure that the model has learned the correct distribution and the inference network is correct as well.

First, we do sanity check to see if the learned distribution is correct. We generate ten original samples, each from one of ten categories for comparison. Then we create a random sample from standard Multinomial Gaussian Distribution which we consider is the prior distribution of latent variables, and run them through the decoder network. Many of the generated images are blurry or are a composition of two categories. Here we choose 5 generated samples that are most clear just to check the decoder network.

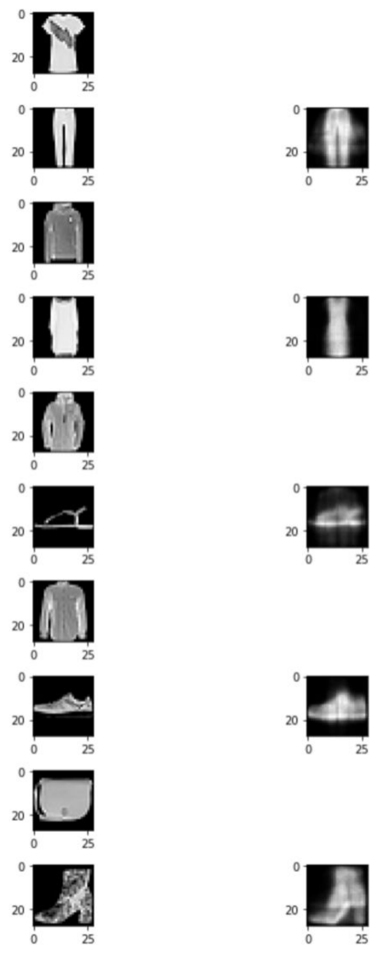


Figure 1: Actual Samples and Generated Samples

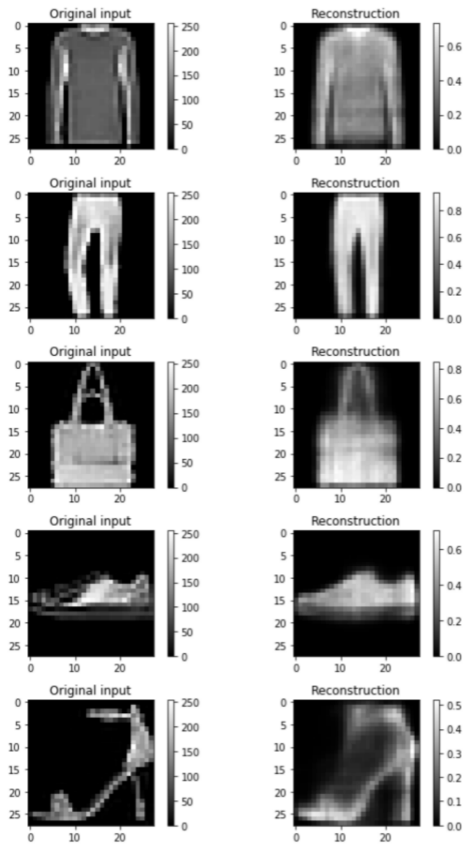


Figure 2: Actual Samples and Reconstructed Samples

These samples above (**Figure 1**) are similar to the actual ones, thus we can claim that our generative model has learned the correct distribution.

Then we do sanity check to see whether the inference network is correct or not. We generate the hidden variables from the inference network and run through them in the decoder network. The reconstructed images and original inputs look similar (**Figure 2**), so the inference network is also correct.

Now we can claim that our VAE performs reasonably well. The next thing we want to know is about its performance in the downstream classification task. As we mentioned in the introduction, we test and compare the performances of three methods. For each method, we train the classifier on different sizes of subsets of the data.

3.2 VAE+logistic regression

Keeping our VAE fixed, the first classifier we trained is logistic regression. The inputs here are the hidden variables we generate in the previous part. We use the logistic regression to predict the class labels.

3.3 VAE+simple CNN

Similarly, we keep our VAE fixed and try another more flexible model instead of fixed models like logistic regression: simple CNN, which consists of only one convolutional layer and one fully connected layer. Softmax function and one-hot-encoder is used for classification task.

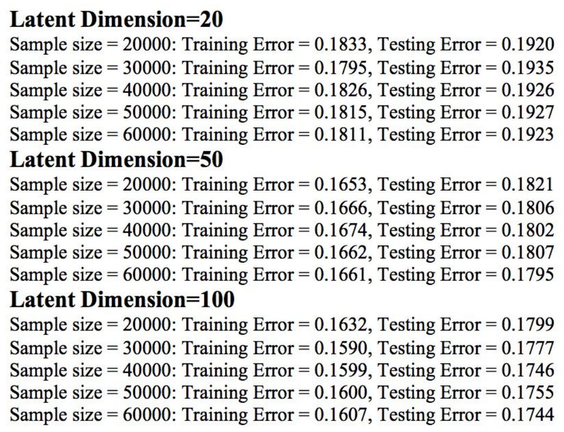
3.4 Multilayer CNN

In order to compare the process of VAE and reconstruction, here we introduce multilayer Convolutional Neural Network on our original data for comparison. Here we use $Input \rightarrow ConvLayer1 \rightarrow Poolinglayer1 \rightarrow ConvLayer2 \rightarrow PoolingLayer2$ as our architecture of neural networks.

In order to compare the effect of different sample sizes on test errors of different classifiers, we tried 20000, 30000, 40000, 50000, 60000 for training. For each classifier we also tried 20 (which is the dimension we used to build our VAE model), 50 and 100. The testing size are the same (10000) for each of them.

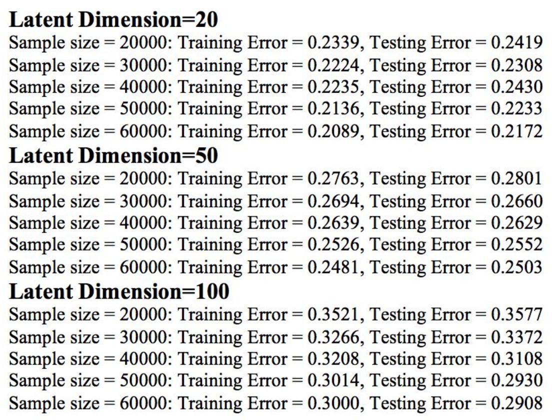
4 Results

The results of VAR + Logistic Regression Classifier, VAE + Simple CNN and a Multilayer CNN are shown below. VAE+Simple CNN and Multiplayer CNN are both run for 5 epochs.



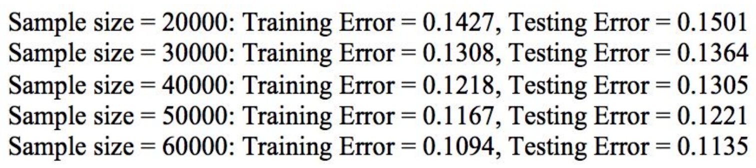
```
Latent Dimension=20
Sample size = 20000: Training Error = 0.1833, Testing Error = 0.1920
Sample size = 30000: Training Error = 0.1795, Testing Error = 0.1935
Sample size = 40000: Training Error = 0.1826, Testing Error = 0.1926
Sample size = 50000: Training Error = 0.1815, Testing Error = 0.1927
Sample size = 60000: Training Error = 0.1811, Testing Error = 0.1923
Latent Dimension=50
Sample size = 20000: Training Error = 0.1653, Testing Error = 0.1821
Sample size = 30000: Training Error = 0.1666, Testing Error = 0.1806
Sample size = 40000: Training Error = 0.1674, Testing Error = 0.1802
Sample size = 50000: Training Error = 0.1662, Testing Error = 0.1807
Sample size = 60000: Training Error = 0.1661, Testing Error = 0.1795
Latent Dimension=100
Sample size = 20000: Training Error = 0.1632, Testing Error = 0.1799
Sample size = 30000: Training Error = 0.1590, Testing Error = 0.1777
Sample size = 40000: Training Error = 0.1599, Testing Error = 0.1746
Sample size = 50000: Training Error = 0.1600, Testing Error = 0.1755
Sample size = 60000: Training Error = 0.1607, Testing Error = 0.1744
```

Figure 3: VAE+Logistic Regression



```
Latent Dimension=20
Sample size = 20000: Training Error = 0.2339, Testing Error = 0.2419
Sample size = 30000: Training Error = 0.2224, Testing Error = 0.2308
Sample size = 40000: Training Error = 0.2235, Testing Error = 0.2430
Sample size = 50000: Training Error = 0.2136, Testing Error = 0.2233
Sample size = 60000: Training Error = 0.2089, Testing Error = 0.2172
Latent Dimension=50
Sample size = 20000: Training Error = 0.2763, Testing Error = 0.2801
Sample size = 30000: Training Error = 0.2694, Testing Error = 0.2660
Sample size = 40000: Training Error = 0.2639, Testing Error = 0.2629
Sample size = 50000: Training Error = 0.2526, Testing Error = 0.2552
Sample size = 60000: Training Error = 0.2481, Testing Error = 0.2503
Latent Dimension=100
Sample size = 20000: Training Error = 0.3521, Testing Error = 0.3577
Sample size = 30000: Training Error = 0.3266, Testing Error = 0.3372
Sample size = 40000: Training Error = 0.3208, Testing Error = 0.3108
Sample size = 50000: Training Error = 0.3014, Testing Error = 0.2930
Sample size = 60000: Training Error = 0.3000, Testing Error = 0.2908
```

Figure 4: VAE+Simple CNN



```
Sample size = 20000: Training Error = 0.1427, Testing Error = 0.1501
Sample size = 30000: Training Error = 0.1308, Testing Error = 0.1364
Sample size = 40000: Training Error = 0.1218, Testing Error = 0.1305
Sample size = 50000: Training Error = 0.1167, Testing Error = 0.1221
Sample size = 60000: Training Error = 0.1094, Testing Error = 0.1135
```

Figure 5: Multilayer CNN

From the result of *VAE + Logistic Regression* (**Figure 3**), which is cross validated with 5 folds, we can find that with the latent dimension of 100, the model has the best performance. In general, smaller subset of data (20000) leads to a little bit higher testing error: 0.1799 compared with testing error of original dataset (60000) of data: 0.1744. In our attempt, we also find that as sample size increases, testing error does not decrease stably.

From the result of *VAE + Simple CNN* (**Figure 4**), the results indicate that smaller latent dimension(20) has the best performance. In terms of effect of sample size, original dataset (60,000) has better performance (testing error = 0.2172) compared with smaller sample size(20,000): testing error = 0.2419. Similar to logistic regression, the testing error does not always decrease with larger sample size.

From the result of *Multilayer CNN* (**Figure 5**), sample size of 2000 (testing error = 0.1501) has worse performance compared with original sample size (testing error = 0.1135). In this case, testing error decreases steadily with increasing sample size.

Comparing three different models, Multilayer CNN has the best testing error = 0.1135 with training size = 60,000. VAE + Logistic Regression has the second best performance with latent dimension = 100 and sample size = 60,000, achieving testing error = 0.1744. VAE + Simple CNN has the worst performance with latent dimension = 20 and sample size = 60,000, achieving the testing error = 0.2172.

5 Conclusion

From the result, we can generate a fair amount of desired images. From those reconstructed images, the encoder pipeline can roughly mimic the original pattern. The reconstructed images are blurry but can show rough shapes of corresponding original images.

All of the three classification methods tend to have better performance with larger training size; although in both VAE + Logistic Regression and VAE + Simple CNN, the testing error is not always smaller with a larger training size. For example, with latent dimension of 100 and training size 50000, the testing error of logistic regression is larger than with 40000 training samples. Moreover, it seems that with a small data regime, the dimension reduction of VAE does not help with improving accuracy, since multiplayer CNN still has a lower testing error.

Latent dimension somehow has influence on the neural network models. Larger latent dimension in VAE + logistic regression performs better, while larger latent dimension in VAE + simple CNN does not improve. We conclude that the latent dimension of 20 (which we used to construct VAE model) may be a better choice since as latent dimension increases from 20 to 100, the testing error of VAE + simple CNN was increased by 0.8-0.9, while testing error of VAE + Logistic Regression only decreases by 0.01-0.02. Furthermore, a high latent dimension may lead to overfitting and longer computational time.

6 Further Discussion

Since there are a few results do not match our expectations, we would like to propose a few further discussions.

1. We expect that with larger sample size, the accuracy of our model should also increase. However, as noted in previous discussion, in the case of VAE + Logistic Regression and VAE + Simple CNN, the decrease of testing error is not stable. This may be due to mismatching of dimension of latent space and size of the training sample. Also, more epochs and more sample sizes can be tested for this problem.
2. Modify the network architecture and try other activation functions such as sigmoid function so that VAE can work more efficiently. Moreover, our current version of multilayer CNN only consists of two layers, so we can try more layers to see if it can improve accuracy.
3. Try with dataset which contains images with higher resolution to make better reconstruction quality.

7 Reference

1. Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. arXiv preprint arXiv:1312.6114, 2013.

2. Iftekher Mamun. A Simple CNN: Multi Image Classifier：
https://towardsdatascience.com/a-simple-cnn-multi-image-classifier-31c463324fa
3. Github links: https://github.com/MorvanZhou/tutorials/tree/master/kerasTUT
https://github.com/bojone/vae