DGM Assignment 1

Judhajit Roy

1 NADE Implementation

In NADE, we use the probability chain rule to calculate the most probable pixel for every state. The equations have been referred from the paper Neural auto-regressive distribution estimation (Uria et al., 2016). Some changes in the implementation were also made since the samples generated for (0-9) states were noisy. The main implementation is to flatten the image and create 500 hidden units for the each pixel and transformed to 10 units. The units are then passed to a softmax layer to obtain probabilities for each state and update each unit with the input of the current pixel. Negative Log-Likelihood Loss is calculated for the probabilities and the flattened input image and back propagated. For sampling, the first pixel is of zero value. After the hidden units are calculated for this pixel and passed to the softmax layer, a sample is generated using a Categorical distribution for the probability values. The process continues till all 784 pixels are collected. Then, the image is reshaped and can be plotted.

The samples obtained after this implementation were noisy and loss was oscillating around 1100 therefore I added a linear layer before passing the hidden units to softmax. This increased the model complexity and resulted in better samples.

2 Results

The loss keeps decreasing with each epoch. The optimizer used ADAM with learning rate of 0.01. After 3 epochs the model keeps oscillating so the training was stopped there.

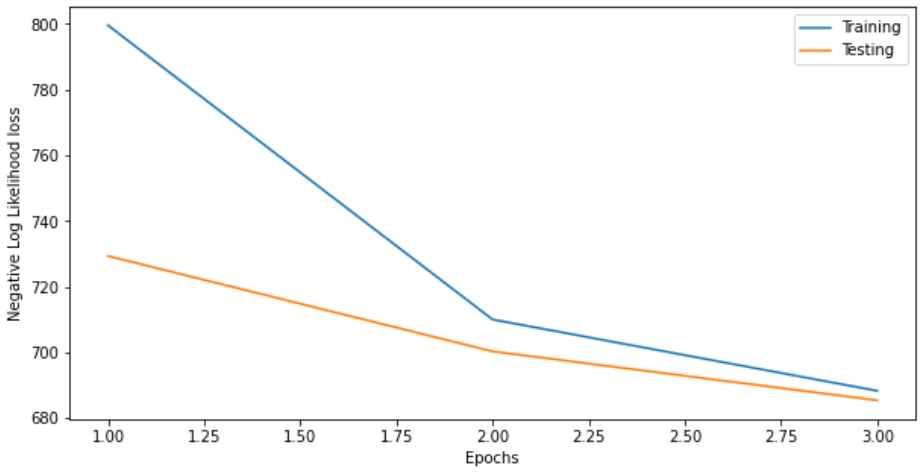

Figure 1: Loss Plot for Nade

Table 1: NADE Negative Log-Likelihood Loss

Epochs	Train Loss	Test Loss
1	800.273	753.917
2	715.891	724.280
3	690.142	709.227

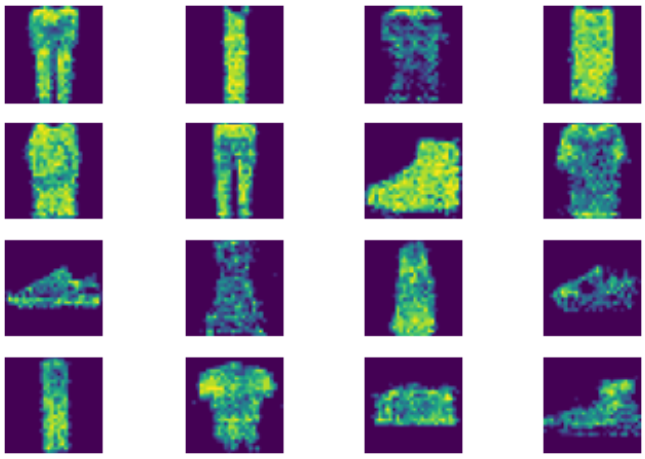

Figure 2: Samples obtained using Nade

3 PixelRNN Implementation

In PixelRNN, we pass the flattened image to a LSTM layer and obtain the value of the next pixel for each pixel. In the paper PixelRNN (van den Oord and Kalchbrenner, 2016), there are multiple implementations of modifying this approach using convolution layers and customized recurrent layers like Row LSTM and Diagonal LSTM. In this implementation, I passed the zero as the initial state and then the each pixel of the image to LSTM Cell to obtain the hidden state. In the next iteration, the next pixel is the input with the obtained hidden state. Also, the hidden state is passed to a linear layer to obtain 10 hidden units. These hidden units are passed to log softmax function to obtain log probabilities of the 10 states.For sampling, since we have 10 states I used categorical distribution to sample. This sample is the input for the next iteration. I referred the paper Pixel-Based LSTM Generative Model (Phon-Amnuaisuk et al., 2018) to get an insight for this implementation.

The samples obtained after this implementation have a clear background and foreground but are a bit distorted.

4 Results

The loss keeps decreasing with each epoch. The optimizer used ADAM with learning rate of 0.001. After 3 epochs the model keeps oscillating so the training was stopped there.

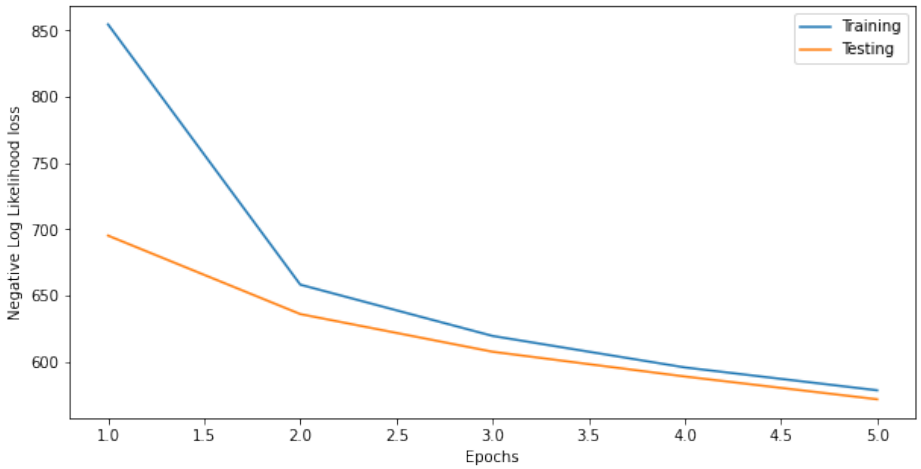

Figure 3: Loss Plot for PixelRNN

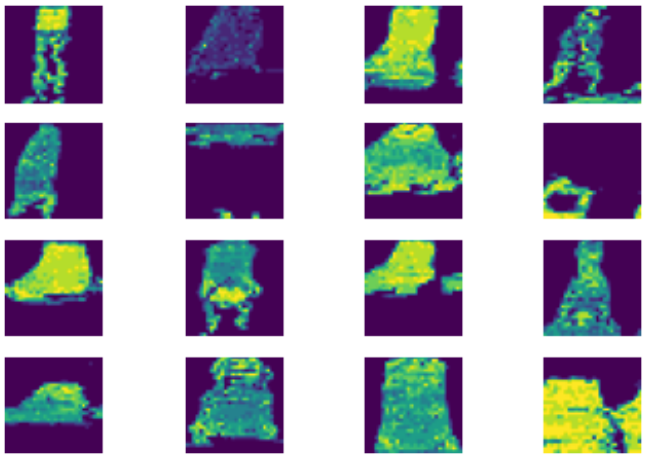

Figure 4: Samples obtained using PixelRNN

Table 2: PixelRNN Negative Log-Likelihood Loss

Epochs	Train Loss	Test Loss
1	851.273	708.917
2	660.891	655.280
3	632.142	629.227
4	615.891	612.280
5	600.142	597.227

References

Phon-Amnuaisuk, S., Salleh, N. D. H. M., and Woo, S.-L. (2018). Pixel-based lstm generative model. In *International Conference on Computational Intelligence in Information System*, pages 203–212. Springer.

Uria, B., Côté, M.-A., Gregor, K., Murray, I., and Larochelle, H. (2016). Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220.

van den Oord, A. and Kalchbrenner, N. (2016). Pixel rnn. In *ICML*.