Introduction to Machine Learning

Project Phase 2

OOD Detection and Generation with NADE

Instructor: Dr. S. Amini

Farnoosh Choogani - 402100691

Department of Electrical Engineering

Sharif University of Technology

Winter 2026

Contents

Theoretical Questions

I Question 1: Parameter Efficiency

1. Tabular Method

For a binary image with D pixels (binary random variables), there are 2^D possible unique configurations or states. To fully represent the joint distribution without any assumptions (a tabular method), we must assign a specific probability to every possible state.

However, because the sum of probabilities for all possible states must equal 1 ($\sum_{\mathbf{x}} p(\mathbf{x}) = 1$), the last probability is determined by the others. Therefore, the number of independent parameters required is:

$$2^D - 1$$

2. NADE Model Parameters

NADE parametrizes the conditional probabilities using a neural network with shared weights, rather than storing a value for every state. Based on the architecture defined in the project with input dimension D and hidden size H:

- **Input weights $\mathbf{W} \in \mathbb{R}^{H \times D}$**: Requires $H \times D$ parameters.
- **Output weights $\mathbf{V} \in \mathbb{R}^{D \times H}$**: Requires $D \times H$ parameters.
- **Hidden bias $\mathbf{c} \in \mathbb{R}^{H}$**: Requires H parameters.
- **Output bias $\mathbf{b} \in \mathbb{R}^{D}$**: Requires D parameters.

Summing these components gives the total number of free parameters in NADE:

$$N_{params} = (H \times D) + (D \times H) + H + D = 2HD + H + D$$

3. Scaling Comparison

Comparing the computational complexity with respect to the input dimension D:

- **Tabular Method:** Scales **exponentially** ($O(2^D)$). This leads to the "curse of dimensionality." For high-dimensional data like MNIST where $D = 784$, storing 2^{784} parameters is computationally impossible.
- **NADE:** Scales **linearly** ($O(D)$), assuming the hidden size H is fixed or scales linearly with D. By utilizing parameter sharing and the chain rule factorization, NADE reduces the complexity from exponential to linear, making it feasible to model complex high-dimensional distributions.

II Question 2: The Computational "Trick"

1. Equation for a_{d+1}:

In NADE, we recursively compute the pre-activation for each pixel using the previous pixel's pre-activation. For the d-th pixel, the pre-activation is:

$$a_d = W_{<d}x_{<d} + c$$

Where:

- $W_{<d}$ refers to the weights corresponding to all pixels before the d-th pixel.
- $x_{<d}$ is the input vector for all pixels before the d-th pixel.
- c is the bias term.

To compute a_{d+1} for the next pixel, we can use the following update rule:

$$a_{d+1} = a_d + W_d x_d$$

Where:

- W_d is the weight vector corresponding to the d-th pixel.
- x_d is the value of the d-th pixel.

2. Computational Complexity:

- **With Recursive Update:** The recursive update method ensures that for each pixel, we are simply adding the next term in the sequence. This results in a time complexity of $O(1)$ for each pixel. Since there are D pixels in the image, the total time complexity for a forward pass is:

  $$O(D)$$

  This is efficient because we only need to perform one addition per pixel, avoiding repeated matrix multiplications.

- **Without Recursive Update:** If we did not use the recursive update, we would need to compute the full matrix-vector multiplication for each pixel. For the d-th pixel, this would involve calculating a dot product between the weight matrix W and the input vector $x_{<d}$, resulting in $O(D)$ time complexity for each pixel. Since we would need to do this for every pixel, the total time complexity would be:

  $$O(D^2)$$

  This approach is much slower due to the repeated computation of matrix-vector multiplications for each conditional probability.

III Question 3: Maximum Likelihood and Loss Function

1. Equivalence of Negative Log-Likelihood and BCE Loss:

The joint probability of the binary vector x can be factored as:

$$p(x) = \prod_{d=1}^{D} p(x_d|x_{<d})$$

Taking the negative log of the joint probability:

$$-\log p(x) = -\log \left(\prod_{d=1}^{D} p(x_d|x_{<d}) \right) = -\sum_{d=1}^{D} \log p(x_d|x_{<d})$$

Since $p(x_d|x_{<d})$ is modeled by the predicted probability $\hat{x}_d = p(x_d = 1|x_{<d})$, we have:

$$\log p(x_d|x_{<d}) = x_d \log \hat{x}_d + (1 - x_d) \log(1 - \hat{x}_d)$$

Thus, the negative log-likelihood of the entire vector x becomes:

$$-\log p(x) = -\sum_{d=1}^{D} \left[x_d \log \hat{x}_d + (1 - x_d) \log(1 - \hat{x}_d) \right]$$

This is exactly the Binary Cross-Entropy (BCE) loss function for each pixel. Therefore, minimizing the negative log-likelihood of $p(x)$ is equivalent to minimizing the sum of the BCE losses for all pixels in the image.

2. Why Binarize the MNIST Data?

The MNIST dataset consists of grayscale images with pixel values in the range [0, 255]. NADE works with binary data, so we binarize the images by setting pixel values greater than 0.5 to 1, and others to 0. This is necessary because NADE uses Binary Cross-Entropy (BCE) loss, which assumes each pixel is either 0 or 1. If we used continuous values, BCE wouldn't be suitable, and we would need a different loss function, like Mean Squared Error (MSE), which is used for continuous outputs.

IV Question 4: Sampling vs. Scoring

1. Algorithm to Sample a New Image from NADE:

To generate a new image from a trained NADE model, we follow these steps:

- **Start with the First Pixel:** We randomly initialize the first pixel x_1 (either 0 or 1).
- **Generate Each Pixel Sequentially:** For each subsequent pixel x_d (from 2 to D), we calculate the conditional probability $p(x_d = 1|x_{<d})$ based on the previous pixels. This probability is given by $\hat{x}_d$, and we sample the new pixel by comparing $\hat{x}_d$ to a random number. If the random number is less than $\hat{x}_d$, we set $x_d = 1$; otherwise, $x_d = 0$.
- **Repeat for All Pixels:** We continue this process until we have generated all D pixels, creating a complete new image x_{new}.

Each pixel is generated based on the values of the pixels before it, following NADE's autoregressive nature.

2. Why Calculating Probability $p(x)$ is Faster Than Sampling:

Calculating the probability $p(x)$ of an existing image is much faster because it can be done in parallel. We can compute the conditional probability for all pixels at once, making the process efficient and parallelizable.

In contrast, sampling requires generating each pixel one by one, in a strict sequence. To generate each pixel, we need to wait for the previous pixel to be generated, making this process inherently sequential and slower.