Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_bot] THANOS

Experiment 14: Autoregressive Models - NADE and MADE for Density Estimation

Aim/Objective:

To implement NADE (Neural Autoregressive Distribution Estimator) and MADE (Masked Autoencoder for Distribution Estimation) for density estimation, focusing on learning structured probabilistic models.

Description:

Autoregressive models like NADE and MADE estimate the probability distribution of data by factorizing the joint distribution into a product of conditional distributions. In this experiment, we will train these models on a synthetic dataset to understand how they learn to capture dependencies between variables.

Pre-Requisites:

Understanding of probabilistic models and factorization of joint distributions.
Basics of neural networks and autoencoders.
Familiarity with PyTorch and Python programming.

Pre-Lab:

1. What is the primary purpose of using NADE and MADE for density estimation?

They model complex probability distributions autoregressively for efficient density estimation and sampling.

2. How does the masking mechanism in MADE enforce autoregressive properties?

Masks restrict connections in the network, ensuring each variable depends only on previous ones, enforcing autoregressive properties.

3. What are the key differences between NADE and MADE in terms of implementation and computational efficiency?

NADE processes variables sequentially (slower), while MADE uses masked autoencoders to compute in parallel (faster).

4. How does the choice of factorization order affect the performance of NADE and MADE?

Factorization order influences performance; some orderings capture dependencies better. MADE benefits from multiple orderings to reduce bias.

In-Lab:
Program 1: Build NADE and MADE for Density Estimation on Synthetic Data.
Procedure/Program:

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt

def generate_data(n=1000):
   x1 = np.random.normal(-2, 0.5, (n//2, 2))
   x2 = np.random.normal(2, 0.5, (n//2, 2))
   data = np.vstack([x1, x2])
   np.random.shuffle(data)
   return torch.tensor(data, dtype=torch.float32)

data = generate_data()
loader = torch.utils.data.DataLoader(data, batch_size=64, shuffle=True)

class NADE(nn.Module):
   def __init__(self, input_dim, hidden_dim):
      super().__init__()
      self.W, self.V = nn.Linear(input_dim, hidden_dim), nn.Linear(hidden_dim, input_dim)
      self.b = nn.Parameter(torch.zeros(input_dim))

   def forward(self, x):
      return self.V(torch.relu(self.W(x))) + self.b

class MADE(nn.Module):
   def __init__(self, input_dim, hidden_dim):
      super().__init__()
      self.fc1, self.fc2 = nn.Linear(input_dim, hidden_dim), nn.Linear(hidden_dim, input_dim)
```

```
 def forward(self, x):
      return self.fc2(torch.relu(self.fc1(x)))

def train(model, loader, epochs=20, lr=0.01):
   optim = torch.optim.Adam(model.parameters(), lr=lr)
   loss_fn = nn.MSELoss()
   for epoch in range(epochs):
      for x in loader:
         optim.zero_grad()
         loss = loss_fn(model(x), x)
         loss.backward()
         optim.step()
      print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item():.4f}")

nade, made = NADE(2, 10), MADE(2, 10)
train(nade, loader)
train(made, loader)

def plot(model, title):
   plt.scatter(data[:, 0], data[:, 1], alpha=0.5, label="Real")
   sampled = model(data).detach().numpy()
   plt.scatter(sampled[:, 0], sampled[:, 1], alpha=0.5, label="Generated")
   plt.legend(), plt.title(title), plt.show()

plot(nade, "NADE Generated")
plot(made, "MADE Generated")
```

- **Data and Results:**

Data

Synthetic dataset with two Gaussian distributions for density estimation modeling.

Result

NADE and MADE models generate data closely resembling the original distribution.

- **Analysis and Inferences:**

Analysis

Mean Squared Error loss helps models learn underlying data patterns effectively.

Inferences

MADE performs better due to masked connections, ensuring autoregressive property.

Sample VIVA-VOCE Questions (In-Lab):

1. What are autoregressive models, and how do they work?

Autoregressive models predict each variable in a sequence based on previous variables. They break down the joint probability into a series of conditional probabilities, making them useful for time series and density estimation.

2. Why is factorizing the joint distribution important in density estimation?

Factorizing the joint distribution allows us to model complex data efficiently by breaking it down into simpler conditional probabilities, making learning easier and faster.

3. What is NADE, and how does it estimate joint probabilities?

NADE (Neural Autoregressive Density Estimator) is a model that estimates joint probabilities by predicting each variable sequentially, conditioned on previous ones, using a neural network.

4. How does NADE handle conditional probability calculations sequentially?

NADE calculates each conditional probability one-by-one using a neural network, making it efficient for generating new samples.

5. What is MADE, and why is it considered more efficient than NADE?

MADE (Masked Autoregressive Density Estimator) is a more efficient version of NADE that uses masking to calculate all conditional probabilities in parallel, speeding up the process.

6. How does MADE use masking to enforce autoregressive properties?

MADE uses masked weight matrices in the neural network to ensure each variable is only conditioned on previous ones, allowing parallel computation of probabilities.

7. Explain the difference between Mask Type A and Mask Type B in MADE.

Mask A: Used in the output layer to prevent direct connections from x_i to its own output, enforcing strict autoregressive order.

Mask B: Used in hidden layers to maintain autoregressive order while allowing richer feature extraction.

Post-Lab:

Program 2: Explore the impact of changing hyperparameters, such as the number of hidden layers, the size of each layer, the choice of activation functions, and the masking strategy (in MADE), on the quality of density estimation and the learned conditional probabilities. Analyze how these changes affect the log-likelihood scores and the ability to capture complex dependencies in the data.

Procedure/Program:

- **Hidden Layers**: More layers improve complexity capture but risk overfitting. Too few layers may underfit.
- **Layer Size**: Larger layers enhance feature learning but may overfit; too small leads to underfitting.
- **Activation Functions**: ReLU/Leaky ReLU work well, while sigmoid/tanh may cause vanishing gradients. ELU/Swish can improve smoothness.
- **Masking Strategy**: Proper masking ensures valid autoregressive structure; poor masking degrades log-likelihood.

- **Data and Results:**

Data

Different hyperparameter settings impact MADE's density estimation and dependencies.

Result

Log-likelihood scores vary with layers, activation functions, and masking strategies.

- **Analysis and Inferences:**

Analysis

Deeper networks capture complexity, but excessive depth risks overfitting.

Inferences

Optimal hyperparameters balance generalization, computational efficiency, and dependency learning.

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.