**Neural Network for Logic Gates: Report**

**Model Architecture**

The neural network model for logic gates is a simple feedforward neural network with the following architecture:

**- Input Layer:** 2 neurons corresponding to the two binary inputs of the logic gate.

**- Hidden Layer:** No hidden layers, making this a single-layer perceptron.

**- Output Layer:** 1 neuron producing a single output, which is then passed through a Sigmoid activation function to produce a value between 0 and 1.

This model is sufficient for learning basic logic gates like AND and OR but has limitations for non-linearly separable functions like XOR.

**class LogicGateNN(nn.Module):**

**def \_\_init\_\_(self):**

**super(LogicGateNN, self).\_\_init\_\_()**

**self.fc = nn.Linear(2, 1)**

**self.sigmoid = nn.Sigmoid()**

**def forward(self, x):**

**x = self.fc(x)**

**x = self.sigmoid(x)**

**return x**

**Training Process**

1. **Data Preparation:**

- For each logic gate (AND, OR, XOR), the input-output pairs are predefined.

- The inputs are the combinations of binary values [0, 1], and the targets are the respective gate outputs.

**2. Loss Function:**

- Binary Cross-Entropy Loss (`nn.BCELoss`) is used as the loss function because the outputs are binary.

**3. Optimizer:**

- Stochastic Gradient Descent (SGD) with a learning rate of 0.1 is used to update the model parameters.

**4. Training Loop:**

- The model is trained for 10,000 epochs.

- In each epoch, the following steps are performed:

- Forward pass to compute the model's output.

- Loss computation.

- Backward pass to compute the gradients.

- Parameter update using the optimizer.

**def train\_gate\_model(model, inputs, targets):**

**criterion = nn.BCELoss()**

**optimizer = optim.SGD(model.parameters(), lr=0.1)**

**epochs = 10000**

**for epoch in range(epochs):**

**optimizer.zero\_grad()**

**outputs = model(inputs)**

**loss = criterion(outputs, targets)**

**loss.backward()**

**optimizer.step()**

**if (epoch + 1) % 1000 == 0:**

**print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')**

**Results**

- The model successfully learns the AND and OR gates as these functions are linearly separable.

- The XOR gate poses a challenge because it is not linearly separable. A single-layer perceptron cannot perfectly learn the XOR function. However, the training process still attempts to minimize the loss over the given epochs.

**Visualisation Interface**

The interactive interface is created using Gradio, allowing users to visualize and interact with the trained logic gate models. Here are the steps to use the interface:

**1. Launching the Interface:**

- The interfaces for AND, OR, and XOR gates are created and launched using `gr.Interface`.

**# Create and launch interfaces for AND, OR, and XOR gates**

**and\_iface = create\_gradio\_interface("AND")**

**or\_iface = create\_gradio\_interface("OR")**

**xor\_iface = create\_gradio\_interface("XOR")**

**and\_iface.launch()**

**or\_iface.launch()**

**xor\_iface.launch()**

```

**2. Using the Interface:**

- Each interface has two radio buttons for the binary inputs (0 or 1).

- Select the desired inputs and click the submit button to see the predicted output.

- The model's decision boundary is visualized in a plot, showing how the neural network classifies the input space.

**Instructions for Visualization**

1. **Select Inputs:** Use the radio buttons to choose the inputs for the logic gate (either 0 or 1 for both inputs).

**2. Submit:** Click the submit button to see the prediction from the neural network.

**3. View Output:** The predicted output (0 or 1) will be displayed.

**4. Decision Boundary:** Observe the decision boundary plot which visually represents how the model separates the input space based on the learned logic function.