XOR Neural Network Learning Plan

1. Full Formatted Summary (with Explanations)

Building and Training a Neural Network for XOR – Key Steps and Concepts

1. Data Representation

- Inputs and outputs for XOR are represented as lists of lists:
- `inputs = [[0,0], [0,1], [1,0], [1,1]]`
- `outputs = [[0], [1], [1], [0]]`

2. Network Structure

- The network has three layers:
- **Input Layer:** 2 neurons (`x1` , `x2`)
- **Hidden Layer:** 2 neurons (`h1` , `h2`)
- **Output Layer:** 1 neuron (`o`)
- Every input connects to every hidden neuron; every hidden neuron connects to the output neuron.
- Each **weight** and **bias** is unique to its connection/neuron.

3. Forward Pass

- For each input:
- Compute **weighted sums** for each hidden neuron:
  - `z1 = w1_1*x1 + w1_2*x2 + b1`
  - `z2 = w2_1*x1 + w2_2*x2 + b2`
- Apply sigmoid activation to each weighted sum:
  - `h1 = sigmoid(z1)`
  - `h2 = sigmoid(z2)`
- Compute output neuron's weighted sum:
  - `zo = v1*h1 + v2*h2 + bo`
- Apply sigmoid activation to output sum:
  - `o = sigmoid(zo)`

4. Error Calculation

- Calculate the error for each output:
- `error = actual_output - predicted_output`

5. Backpropagation (Calculating Contributions)

- Compute the **gradient at the output neuron**:
- `go = error * sigmoid_derivative(output)`
- Propagate the error back to the hidden neurons:
- For hidden neuron 1: `hidden1_error = go * v1`

- For hidden neuron 2: `hidden2_error = go * v2`
- Compute **sigmoid derivative** at each hidden neuron.
- Compute **gradients for each hidden neuron**:
- `gh1 = hidden1_error * sigmoid_derivative(h1)`
- `gh2 = hidden2_error * sigmoid_derivative(h2)`
- These gradients show how much each neuron (and their associated weights and biases) contributed to the output error.

6. Weight and Bias Update

- Update each variable using its calculated gradient (contribution) and the learning rate:
- `new_weight = old_weight + learning_rate * gradient * input`
- `new_bias = old_bias + learning_rate * gradient * 1`
- This is done for **all weights and biases** in both layers.

7. Repeat Training

- Repeat the process (forward, error, backprop, update) for every sample, for multiple epochs.
- This allows the network to learn and minimize error on the XOR dataset.

Extra Insights:

- **No weights or biases are shared between neurons or layers;** each is updated independently.
- The **gradient's sign** tells you the direction to update each parameter to reduce error.
- This process splits up and "assigns blame" for the error across all weights and biases, so each can be nudged in the best direction.

---

2. Step-by-Step Checklist

Step 1: Initialize

- Randomly set all weights and biases:
- Input-to-hidden weights (`w1_1` , `w1_2` , `w2_1` , `w2_2`)
- Hidden biases (`b1` , `b2`)
- Hidden-to-output weights (`v1` , `v2`)
- Output bias (`bo`)
- Set the learning rate and number of epochs.

Step 2: For Each Epoch Repeat steps 3–7 for each epoch.

Step 3: For Each Training Pair Loop over all `(inputs, expected_output)` pairs.

Step 4: Forward Pass

- Compute hidden layer activations:

- For each hidden neuron:
  - `z = w1*x1 + w2*x2 + b`
  - `h = sigmoid(z)`
- Compute output neuron activation:
- `zo = v1*h1 + v2*h2 + bo`
- `o = sigmoid(zo)`

Step 5: Error Calculation

- `error = expected_output - o`

Step 6: Backpropagation

- Compute gradient at output neuron:
- `go = error * sigmoid_derivative(o)`
- Compute hidden-to-output gradients:
- For each hidden neuron:
  - `hidden_error = go * v`
  - `gh = hidden_error * sigmoid_derivative(h)`
- These give you the contributions from each variable.

Step 7: Update Weights and Biases

- Output weights and bias:
- `v_new = v_old + learning_rate * go * h`
- `bo_new = bo_old + learning_rate * go * 1`
- Hidden weights and biases:
- `w_new = w_old + learning_rate * gh * x`
- `b_new = b_old + learning_rate * gh * 1`

Step 8: Repeat

- Continue until all pairs and all epochs are complete.

Step 9: Evaluate

- After training, test your network on all XOR input pairs to see if it learned the correct output.

---

3. Arithmetic Steps – Your Reasoning, Rewritten for Clarity

Step 1: Forward Pass - Calculate the equations z1 and z2, which represent the weighted sums for each hidden neuron. - These are simply the sum of each input times its weight, plus the respective bias. - To introduce non-linearity, apply the activation function (sigmoid) to each weighted sum, giving you the outputs of the hidden neurons (h1 and h2). - Next, calculate the output neuron's weighted sum using the hidden neuron outputs and their respective weights (v1, v2), plus the output bias. - Apply the activation function again to get the final prediction. - **Clarification:** The weights for the hidden layer and the output

layer are *not* the same. Each connection (input to hidden, hidden to output) has its own unique weight. Likewise, each neuron (hidden and output) has its own bias.

Step 2: Error Calculation - Compute the error by subtracting the predicted output from the actual (target) output: - `error = expected_output - predicted_output` - This measures how far off the prediction was.

Step 3: Backpropagation – Calculating Gradients and Contributions - Calculate the derivative of the sigmoid activation function at the output. - Multiply this derivative by the error to get the gradient at the output neuron. This gradient indicates both the direction and magnitude for adjusting the weights and bias to reduce error. - A positive gradient means increasing the parameter will increase the error, so you want to adjust in the opposite direction; a negative gradient means the opposite. - To understand how much each hidden neuron contributed to the output error, multiply the output gradient by the connecting weights (v1, v2). This shows how the error "flows back" through the network. - Do the same for each bias: calculate its gradient to know its contribution to the error. - Repeat this process for all parameters—every weight and bias in both the hidden and output layers—so you know precisely how much each contributed to the final error.

Step 4: Updating Weights and Biases - After finding each variable's contribution (the gradient), update the weights and biases: - `new_value = old_value + learning_rate * gradient * input` (for weights) - `new_bias = old_bias + learning_rate * gradient` (since bias's input is always 1) - Repeat this whole process for every input/output pair, and for as many epochs as necessary.

Step 5: Repeating the Process - Continue looping through forward pass, error calculation, backpropagation, and parameter updates, until the network learns the correct outputs for all XOR cases.
