DYP

Assignment-No.7

TITle: Python program to show back propagation network for XOR function with binary input and output.

Aim: Write a python program to show back propagation network for XOR function with binary input and output.

Theory:

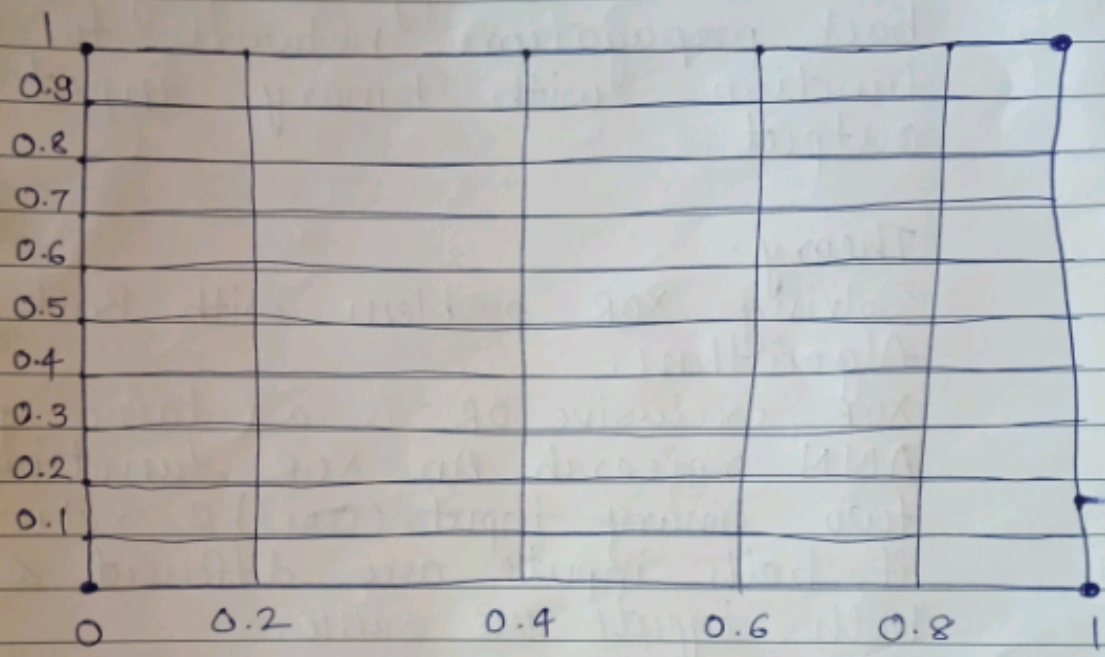
Solving XOR problem with Back propagation Algorithm:

XOR, exclusive OR is a classic problem in ANN research. An XOR function takes two binary inputs (0 or 1) & returns True if both inputs are different & false if both inputs are same.

Input 1	Input 2	Output
0	0	0
1	1	0
1	0	1
0	1	1

On the surface, XOR appears to be a very simple problem, however, Minsky and Papert showed this was a big problem for neural network architectures, known as perceptron. A limitation of this architecture is that it is only capable of seperating data points with a single

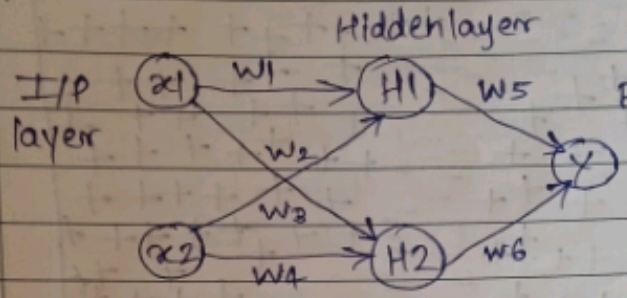
line. This is unfortunate because the XOR inputs are not linearly seperable. This is particularly visible if you plot the XOR input values to a graph.



Solution:
Backpropagation algorithm begins by comparing the actual value output by the forward propagation process to the expected value and then moves backward through the network, slightly adjusting each of the weights in a direction that reduces the size of the error by a small degree. Both forward and back propagation are re-run thousands of times on each output combination until the network can accurately predict the expected output

of the possible inputs forward propagation.

Model



Inputs:

$x_1 = [1,1]^T$, $y_1 = +1$.
$x_2 = [0,0]^T$, $y_2 = +1$
$x_3 = [1,0]^T$, $y_3 = -1$
$x_4 = [0,1]^T$. $y_4 = -1$

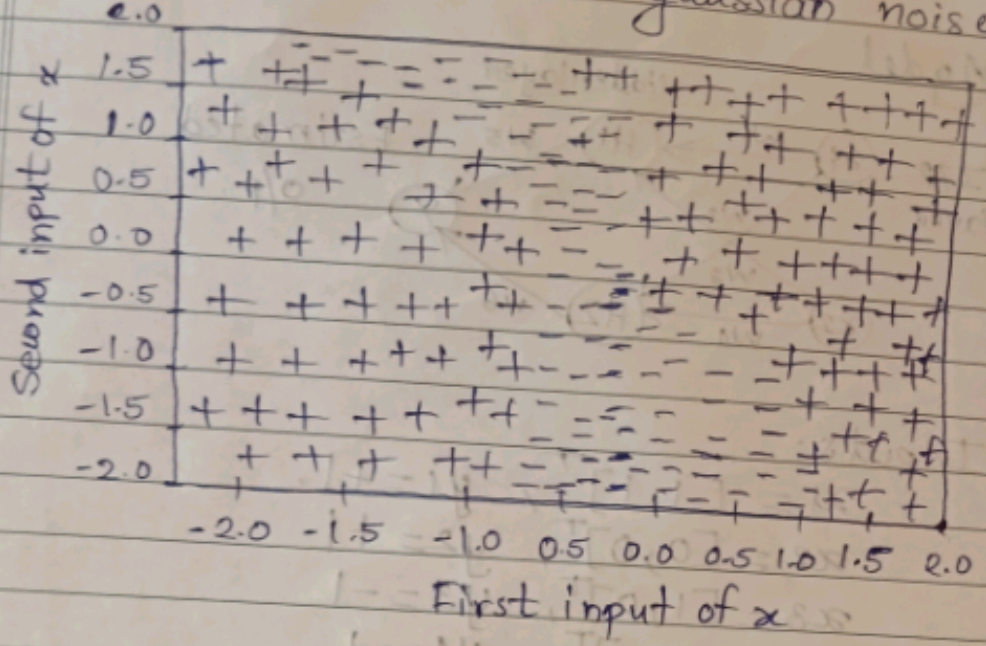
2 hidden neurons are used, each two inputs with different weights. After each forward pass, the error is back propagated.
Using sigmoid:
At hidden layer: $H1 = x_1 w_1 + x_2 w_2$.
$H2 = x_1 w_3 + x_2 w_4$.

At output layer: $\hat{y} = \sigma(H1) w_5 + \sigma(H2) w_6$.
$\sigma \rightarrow$ sigmoid function.
$\sigma(x) = 1/1+e^{-x}$.

Loss function: $\frac{1}{2}(y - \hat{y})^2$

Results:

Classification without gaussian noise:



Classification with gaussian noise:

$x_1 \sim \mu_1 = [1,1]^T$, $\Sigma_1 = \Sigma$ $y_1 = +1$

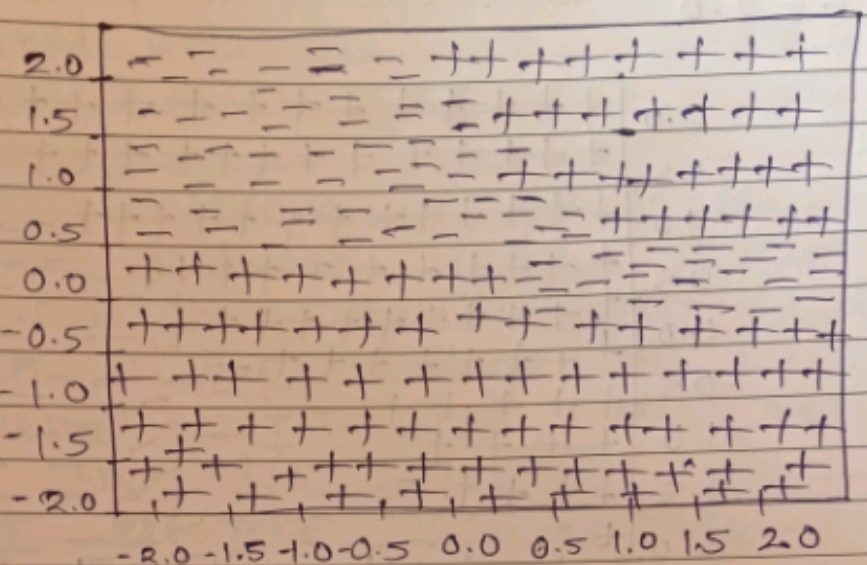
$x_2 \sim \mu_1 = [0,0]^T$, $\Sigma_2 = \Sigma$ $y_1 = +1$

$x_3 \sim \mu_1 = [\ \ ]$, $\Sigma_3 = \Sigma$ $y_1 = -1$

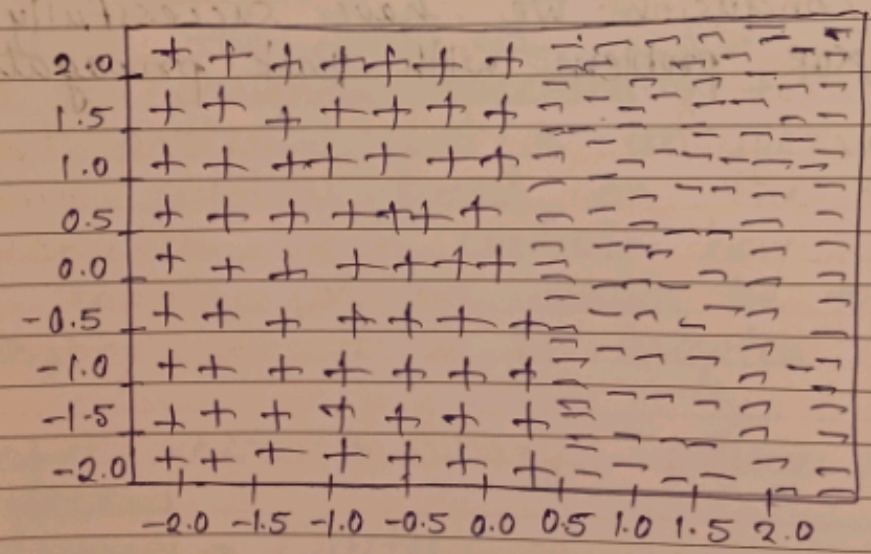
$x_4 \sim \mu_1 = [\ \ ]$, $\Sigma_4 = \Sigma$ $y_1 = -1$

$$\Sigma = \begin{bmatrix} 6 & 0 \\ 0 & 6 \end{bmatrix}$$

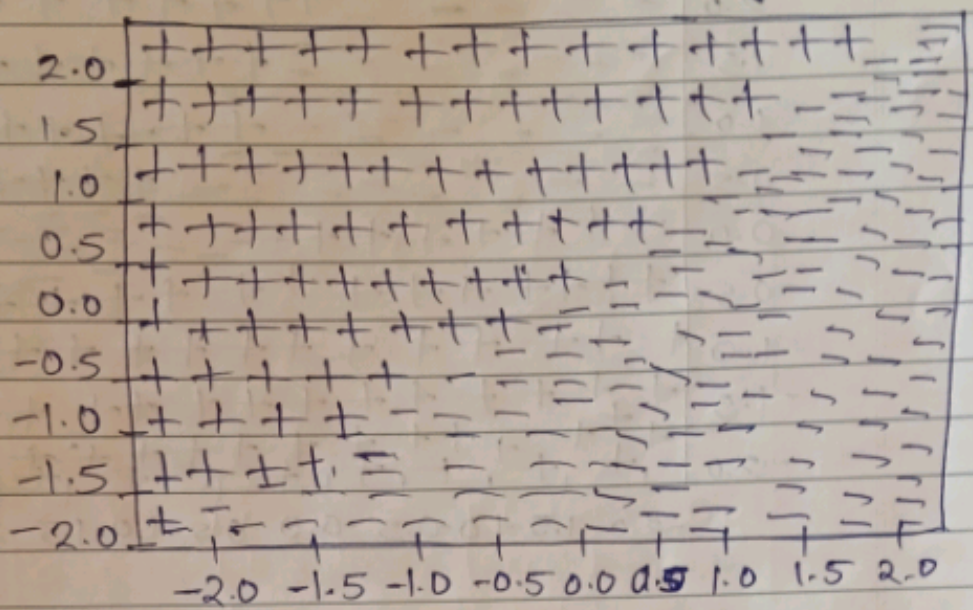
$6 = 0.5$


2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
-2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0

$\sigma = 1.0$


2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
-2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0

.6 = 2.0



Conclusion: We have successfully implemented XOR problem with Back propagation algorithm.

DYP

Assignment - No.

Title: Python program to illustrate ART Neural network.

Theory:

Adaptive Resonance Theory is a type of neural network model developed to address the stability-plasticity dilemma, which is the challenge of learning new information (plasticity) while retaining old knowledge (stability).

ART networks are particularly effective for pattern recognition and clustering tasks and can adaptively classify input patterns into categories.

ART1 is the simplest version of ART and is designed specifically for binary input patterns. It is unsupervised and forms clusters based on the vigilance parameter, which controls how similiar an input pattern must be to a category for it to be accepted.

1) Input vector → Binary pattern to be clustered.
2) Categories → Clusters of similiar patterns formed during learning.
3) Vigilance parameter → Threshold that determines minimum similiarity required for an input to be grouped.
4) Feedforward & feedback weights → Used to compute match between input vectors and

Category templates.

Working mechanism:

1) Initialization:
Feedforward and feedback weights are initialized to 1s.
The network starts with no categories and adds them as needed.

2) Pattern Matching:
Each new input pattern is compared to existing category templates using a match function.
The similiarity is calculated as the dot product between input and output weight factor, normalized by the number of active features in the input.

3) Vigilance Test:
If the similiarity is greater than the vigilance parameter, the pattern is accepted into the category.
If not, a new category is created to accommodate new pattern.

4) Learning
When a pattern is accepted into a category, weights are adjusted to better represent the input while maintaining the binary nature of the data.

DYP

Advantages of ART1:

1) Stability
2) Plasticity.
3) Online learning
4) Control.

Applications:

1) Clustering binary data.
2) Pattern recognition & classification.
3) Anomaly detection.
4) Medical diagnostics.

Conclusion:

DYP

Assignment

Title: Program for creating a back propagation feed forward neural network

Aim: Write a python program for creating a back propagation feed forward neural network.

Theory:

Backpropagation Neural Network:

Backpropagation is a widely used algorithm for training feed forward neural networks. It computes the gradient of the loss function with respect to the network weights. It is very efficient rather than naively directly computing the gradient concerning each weight. This efficiency makes it possible to use gradient methods to train multi-layer networks and update weights to minimize loss; variants such as gradient descent or stochastic gradient descent are used.

Backpropagation Algorithm:

① Input vector x is fed forward through randomly initialized weights W.

② Outputs of all neurons (input → hidden → output layer) are computed.

③ Calculate error:

Error = Desired Output - Actual Output.

④ Backpropagate error from output layer to hidden layer and adjust weights

⑤ Repeat steps until the error is minimized.

Key Parameters:

x = Input vector.
t = target vector.
α = Learning rate.
$\delta k, \delta j$ = Errors at output and hidden layers.
v_{0j}, w_{0k} = Biases at hidden and output layers.

Training Algorithm:

① Initialize weights to small random values.

② Repeat until stopping condition:
For each training input:
Feedforward: Compute activations using weighted sums and activation functions.
Backpropagation:
Calculate output error:
$\delta k = (t_k - y_k) * (f(y_{ink}))$.

Calculate hidden layer error:
$\delta_j = f'(z_{inj}) * \Sigma(\delta_k * w_{jk})$.

Update weights and biases:
Output layer:
$\Delta w_{jk} = \alpha * \delta_k * z_j$.
$\Delta w_{ok} = \alpha * \delta_k$.

Hidden layer:
$\Delta v_{ij} = \alpha * \delta_j * x_i$
$\Delta v_{oj} = \alpha * \delta_j$.

Stopping conditions:
Error threshold reached.
Max number of epochs.

Need for backpropagation:
1) Efficient & simple.
2) Fast learning for multiple layer networks.
3) No prior knowledge needed.

Conclusion:
Backpropagation efficiently trains feed-forward neural networks by minimizing the output error through iterative weight updates.

DYP

Assignment-No.

Title: Design and implement a Hopfield Network in python to store 4 vectors.

Theory:

A Hopfield Network, introduced by Dr. John Hopfield in 1982, is a recurrent neural network used for autoassociation and optimization. It contains a single layer of fully connected neurons where each neurons output serves as input to others (excluding itself). The network operates iteratively and converges to stable states, making it useful for pattern recognition and noise correction.

Discrete Hopfield Network:

1) Works with discrete patterns: binary (0, 1) or bipolar (+1, -1).

2) Has symmetrical weights ($w_{ij} = w_{ji}$) and no self-connections ($w_{ii} = 0$).

3) Neurons influence each other through excitatory or inhibitory connections.

4) Network stabilizes by updating neurons until no further changes occur.

Training algorithm:

Uses Hebbian learning.
For bipolar patterns:
$W = \sum_{p=1}^{P} s^p (s^p)^T$.

Set diagonal weights to zero.

Testing algorithm:

1) Initialize weights using training algorithm.
2) Set the initial state to input pattern.
3) Update each neuron using:
   $net_i = \sum w_{ij} x_j$.
   $y_i = \text{sign}(net_i - \theta_i)$.
4) Repeat until network stabilizes.

Continuous Hopfield Network:
Differs by using continuous activation values between 0 and 1, making it suitable for optimization tasks.

Conclusion:
Successfully designed and implemented a Hopfield Network capable of storing and recalling 4 patterns.

Assignment No.

Title: How to train a Neural Network with Tensorflow/Pytorch and evaluation of logistic regression using Tensorflow.

Problem Statement: How to train a neural network with Tensorflow/Pytorch and evaluation of logistic regression using Tensorflow.

Theory:

① Tensorflow:
Free, open source ML & AI library developed by Google. It is widely used for building and training deep neural network and supports languages like Python, Javascript, C++, and Java. Tensorflow 2.0 was released in 2019.

② Pytorch:
Open-source ML framework based on Python and the Torch library. It's popular in research for building deep learning models and offers ease of use, dynamic computation graphs, and support for over 200 mathematical operations.

③ Regression:
Regression is used to model the

the relationship between features & outcome. It helps predict continuous values like age, salary or experience.

④ Neural Network:
Set of algorithms inspired by the human brain that detect patterns in data. Consists of layers of interconnected neurons to classify or predict outputs.

⑤ Logistic Regression:
Used for binary classification, logistic regression estimates the probability of a class. It maps input features to outputs using a sigmoid function and is useful for problems like predicting win/loss or yes/no outcomes.

Algorithm:

Step 1: Import modules.

Step 2: Load MNIST dataset.

Step 3: Set hyperparameters.

Step 4: Shuffle & batch data.

Step 5: Initialize weights & biases.

Step 6: Define & cost function.

Step 7 : Set optimizer and accuracy metric.

Step 8 : Update weights in training.

Step 9 : Run training loop.

Step 10 : Test model accuracy.

Conclusion:
Implemented a Neural Network with Tensorflow / Pytorch and evaluated logistic Regression performance.

DYP

Assignment No.

Title: Implementation of CNN using Tensorflow / Pytorch.

Problem Statement: Implement a CNN using Tensorflow / Pytorch.

Theory:
Convolutional Neural Networks are deep learning models designed for processing image, audio, or speech inputs. They automatically learn features using layers like:

Convolutional layers:
Extract local features using filters.

Pooling layers:
Reduce dimensionality and retain key features.

Fully connected layers:
Perform final classification.

CNN's gradually recognize features from edges to complete objects, making them highly effective in image-related tasks.

DYP

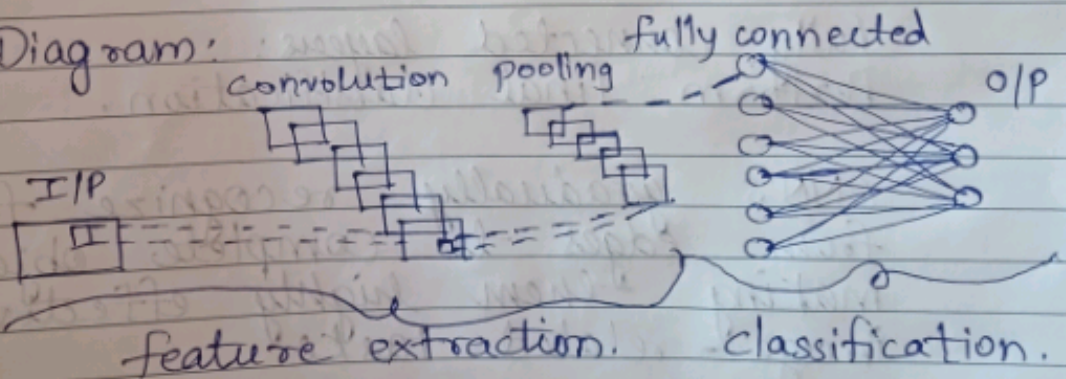
Advantages of CNN's:

1. Detect complex patterns in images/ audio.
2. Invariant to translation and rotation.
3. No manual features extraction.
4. Scalable and accurate with large datasets.

Disadvantages of CNN's:

1. High computational cost.
2. Needs large labeled datasets.
3. Risks of overfitting.
4. Limited interpretability.

Diagram:



DYP

Algorithm:

1. Import Tensorflow/Pytorch modules and datasets.
2. Define the run_cnn() function with parameters and placeholders.
3. Load and reshape image data.
4. Build convolutional and pooling layers.
5. Flatten output for the fully connected layer.
6. Add softmax activation and optimizer.
7. Define accuracy metrics and initialize variables.

Conclusion:
CNN was successfully implemented using Tensorflow/Pytorch for effective image classification.

DYP

Assignment No.

Title: Implementation of MNIST handwritten character Design using Pytorch, Keras, Tensorflow.

Problem Statement: Build a digital recognition system using MNIST dataset and implement it with Pytorch, Keras, and Tensorflow.

Theory:
The MNIST dataset consists of 60,000 training and 10,000 testing grayscale images (28×28 pixels) of handwritten digits (0-9). It is a standard benchmark in deeps learning and image classification

- Tensorflow: Open-source library for machine learning.
- Keras: High-level API built on Tensorflow for fast prototyping.
- Pytorch: Python based deep learning framework developed by facebook.

Implementation steps:

① Dataset: Use MNIST digit images in grayscale (28×28 px)

② Import Libraries: Use Tensorflow/ Keras and Pytorch along with PIL for image handling.

③ Build CNN model:
- Import dataset and layers (Conv2D, Dense, MaxPooling2D, Flatten, Dropout)
- Preprocess & reshape data.
- Convert labels to categorical.

④ Train the Model:
- Set batch size, epochs, and optimizer (eg. Adadelta).
- Compile & train model.
- Save the model for future use.

⑤ Digit Prediction:
- Define a function to load and preprocess images.
- Resize, grayscale, normalize, reshape and predict.

⑥ Build GUI application:
- Use TKinter to create a drawing canvas.
- Buttons: Recognize (to predict) and clear (to reset canvas).
- Run the application using a main loop.

Conclusion: Implemented MNIST handwritten character detection using Pytorch, Keras etc.