HOMEWORK-06

Srihitha Maradi
11706365

Question-1:-

Let's create a base neural network to illustrate the XOR function. when inputs differs, XOR holds true. other logical operators, such as AND, OR and NOT can be used by express XOR.

First, we need to express the XOR function in terms of others logical operations. XOR can be expressed as (A and (NOT B)) OR ((NOT A) And B) i.e.,

XOR (A, B) = (A AND NOT B) OR (NOT A AND B)

The second step is to create the neural network that can represent the function. we will need a hidden layer with two neuron (one for each term in the OR operation) and an output layer with a single neuron.

Hidden layer:- At least 2 nodes with ReLU activation function.

Output layer:- one node with sigmoid activation function

And then, weight need to be assigned. The weights are assigned to connections in network the weights for the connections from the input layer to the hidden layer should be set to implement the AND and NOT operations and the weights for the connection from the hidden layer to output layer should be set to OR operation.

Input to hidden layer:-

Neuron 1 : [20, -20] , bias -10

Neuron 2 : [-20, 20], bias : 10

Hidden to output layer :-

Neuron : [20, +20], bias : - 10

And the neural network that can represent the XOR function has a hidden layer with two neurons using ReLU function and an output layer with one neuron using sigmoid function. The weights for the connection from the input layer to the hidden layer are [20, -20] and [-20, 20] with biases of -10 & 10 respectively. The weights for the connections from the hidden layers to the output layer are [20, + 20] with the bias of minus 10.

And then the 4th step is to compile the model. And then the model is trained based on XOR data that is.

x - train = [[0, 0, 0], [1, 0], [1, 1]]

y - train = [0, 1, 1, 0]

model fit (x- train, y- train, epoch = 5000, verbose = 0]

And then, the trained model is predicted.

Predictions = model. predict (x-train),

After that predication are displayed.

Question-2:

Initalize the filter weights and bias randomly. for each position in the input, compute the dot product between the input and the filter weight s, then add the bias.

Apply a non-linear activation function to the result of the dot production and bias addiction.

Repeat steps 2 and 3 for each filter in the layer
The hyper parameters which we use are:

1) Input_data: The input one_dimension array.
2) Kernel: the one - dimensonal convolutional kernal
3) stride: The step size used when sliding the kernel over the input.
4) A padding:- The number of zero's added the input data before applying the conulution.

And the pseudocode for ConvID, can be written in this way also:-

class con1D(num-filters, k, stride = 1):
create w[i,f] for each 0<i<k, 0<f< num-filters
initalize to 0.

Method output (Input):-

Inputs

Input is x array (It is one dimensional array)

Create out [x,f] for each $0 \le x <$ leng (input) $-k+1$, $0 \le f <$ num_filters.

for each x : $0 \le x <$ len (input) $- k + 1$ do

for each f : $0 \le f \le$ num-filters do

for each f : $0 \le g < k$ do

out [x,f] f = input [x+i] * w[i,f]

return out

Method Back prop (error):

Input:

error is len [input] - k + 1 * num_filters array

Create error [x,f] for each $0 \le x <$ len (Input), $0 \le f <$ num-filters initialized to 0.

for each x : $0 \le x <$ len (Input) $- k + 1$ do

for each f : $0 \le f <$ num_filters do

for each i : $0 \le f < k$ do.

d [i,f] f = input [x+i] * error [x,f]

error [x+i,f) f = error [x,f] * w [i,f]

return error.

Method updata (learning-rate):

updating the weights & biases

for each f : 0 ≤ i < k do

for each f : 0 ≤ f < num-filters do

w[i,f] = learning * d[i,f]