Fundamentals of AI

Name: Chebelli Pavan Kumar

ID: 11691712

Homework 6: Neural Networks & Deep Learning

Q1) To represent the XOR (Exclusive-or) function with a neural network using a sigmoid output activation & a single hidden layer with Relu activation you can use following structure.

Input Layers : Two neurons, one for each input, representing the two boolean values (0 (or) 1)

Hidden Layers : You will need a minimum of two neurons in the hidden layers to represent XOR.

→ Simplied diagram of neural network:

Input Layer:	Hidden Layer :-	Output Layer :
[x_1]	[Relu]	[sigmoid]
[x_2]	[Relu]	[y]

[w_1] [w_2]

→ Now, let's consider how to set the weights to the network to represent XOR, XOR can be represented as a combination of Logical operations, specially using AND, OR & NOT operation.

XOR = (x_1 AND (NOT x_2)) OR ((NOT x_1) AND x_2)

S → To create a neural network to represent this express

you can use the following weights and biases:

→ For the hidden layer:

- Neuron 1 weights: [1, -1]
- Neuron 1 bias: 0
- Neuron 2 weights: [-1, 1]
- Neuron 2 bias: 0

→ For the output layer:

- output neuron weights: [1, 1]
- output neuron bias: -1

→ with these weights and biases, the network will correctly represent the XOR function as defined by the logical expression above. It will produce the following outputs:

- (0, 0) → output ≈ 0 (false)
- (0, 1) → output ≈ 1 (True)
- (1, 0) → output ≈ 1 (True)
- (1, 1) → output ≈ 0 (false)

→ These weights & biases create a network that performs the XOR operation using the given logical operations, as specified in the hint.

Q2) Here's the pseudo code for a one-dimentional convolutional layer (Conv1D) based on the structure provided for the two-dimensional version (Conv2D):

b) Python code.

```
class Conv1D (K):
    create w[i] for each 0<=i<k, initialize randomly
    create d[i] for each 0<=i<k, initialize to 0.
    method output (input)
        Inputs
            input is yd array
        create out [y] for each 0<=y<yd-K+1
        for each y:0<=y<yd-K+1 do
            out[y]: sigma (i=0 to K-1) input [y+i]*w[i]
        return out
    method Backprop(error)
        Inputs
            error is yd-K+1 array
        create ierror [y] for each 0<=y<yd, init to 0
        for each y:0<=y<yd-K+1 do
            for each i:0<=i<K do
                d[i]:=d[i]+input[y+i]*error[y]
                ierror [y+i]:=ierror [y+i]+error[y]*w[i]
        return ierror
```

method update (learning rate, batch-size)

for each i do

w[i] := w[i] - learning rate / batch_size * d[i]

d[i] := 0

→ This pseudocode defines a one-dimensional convolution layer (Conv1D) with similar operations to the two-dimensional

Hyper-parameters:

① KERNEL SIZE (K)*: The kernal size (K) is defined at the beginning of the Conv1D class. To extend the pseudocode to allow for variable kernel sizes, we can add a constructor (or) an initialization method to set the kernel size when creating an instance of the Conv1D layer.

```
class Conv1D:
    def _init_(self, kernel-size):
        self.K = kernel size
        self.w = [random-inites for _ in range(self.K)]
        self.d = [0] * self.K
    # Rest of the pseudocode remain the same
```

→ with this modification, you can create Conv1D Layers with different kernel sizes by passing the deserred kernel-size when creating an instance.

2) Activation function: The pseudocode doesn't specify the activation function, which is a crucial hyperparameter is neural network i.e. include an activation function, you can add an activation function variable & apply it to output of convolution operation.

```
class Conv1D:
    def _init_ (self, kernal=size, activation):
        self.k = kernel_size
        self.activation = activation
        self.w = [random_init() for _ in range(self.k)]
        self.d = [0] * self.k

    def output (self, input):
        out = [0] * (len(input) - self.k+1)
        for y in range (len(input) - self.k+1):
            convolution_result = sum(input [y+i] * self.w[i]
                                     for i in range (self.k))
            out [y] = self.activation (convolution_result)

    return out

# Rest of the pseudocode remains the same.
```

→ with this modification, you can specify the activation function when creating a Conv1D layer.

→ These extensions allows you to control the kernel size & the activation function, making our Conv1D Layer more versatile & consistent with typical deep learning libraries like keras & python.