MLP Documentation (Focus on BackPropagation)

Group Members:

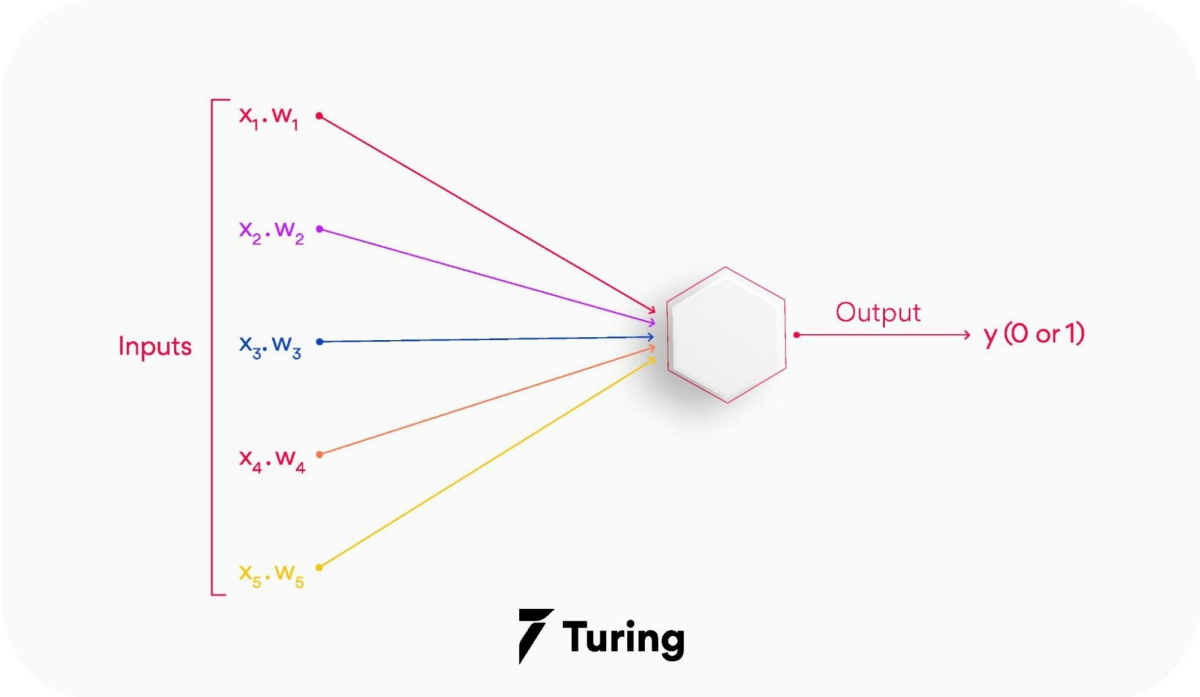
- Amadin Ahmed
- Yash Amin

Github LinK:

https://github.com/amadinahmed/460_pro

MLP

1. `Activation Funtions`(sigmoid, tanh, and ReLU): These are used to introduce non-linearity in the model.
2. `feedforward`: Implements the forward pass of the neural network, calculating the output of each layer and feeding it as input to the next layer.



1. `output-error`: Calculates the error between the actual and expected outputs of the network.
2. `propagate-error`: Propagates the error backward through the network, calculating the error for each layer.
3. `backprop`: Implements the backpropagation algorithm, calculating the error gradients for the weights and biases.
4. `adjust-weights`: Updates the weights in the network based on the calculated gradients.
5. `adjust-parameters`: Updates the weights and biases in the network using the gradients.
6. `gradient-descent`: Performs the gradient descent optimization algorithm for updating the network parameters.
7. `parameters-in-net`: Calculates the total number of parameters (weights and biases) in the network.

https://github.com/karpathy/micrograd

BackPropagation

Backpropagation is an essential part of training neural networks as it helps compute the gradients of the loss function with respect to each weight and bias by propagating the errors backward through the network.

In the code, the `backprop` function is responsible for implementing the Backpropagation algorithm:

```
(define (backprop weights outputs expected-output)
  (let loop ([errors (list)]
             [prev-error (output-error (first (last outputs)) (second (last outputs)) expected-output)]
             [outputs (reverse (drop-right (reverse outputs) 1))]
             [weights (reverse (rest weights))])
    (if (empty? outputs)
        (reverse (cons prev-error errors))
        (let ([error (matrix-map * (matrix* (first weights) prev-error) (matrix-map activation~ (first (fi
rst outputs))))])
          (loop (cons prev-error errors) error (rest outputs) (rest weights))))))
```

1. Calculate the cost function, C(w)
2. Calculate the gradient of C(w) with respect to (w.r.t) all the weights, **w**, and biases, **b,** in your neural network (NN)
3. Adjust the **w** and **b** proportional to the size of their gradients.

Feed Forwarding

The `feedforward` function implements the forward pass of the neural network, calculating the output of each layer and feeding it as input to the next layer. The function takes as input an `input` matrix, which represents the input to the neural network, and `weights` and `biases` lists, which represent the weights and biases for each layer of the network.

```
(define (feedforward input weights biases)
  (if (empty? weights)
      (list)
      (let* ([z (matrix+ (matrix* (first weights) input) (first biases))]
             [out (matrix-map activation z)])
```

```
        (cons (list z out)
              (feedforward out (rest weights) (rest biases))))))
```

The function first checks if the `weights` list is empty, in which case an empty list is returned. Otherwise, the function computes the output of the first layer by taking the dot product of the input matrix with the first set of weights and adding the first set of biases. This output is then passed through the activation function to obtain the output of the first layer. The function then recursively computes the output of each subsequent layer by taking the dot product of the output of the previous layer with the corresponding set of weights and adding the corresponding set of biases. This output is then passed through the activation function to obtain the output of the current layer. The function returns a list of tuples, where each tuple contains the `z` and `out` values for a given layer, representing the input to the activation function and the output of the activation function, respectively.

Gradient Descent:

The `backprop` function is called within the `gradient-descent` function:

```
(define (gradient-descent weights biases inputs expected-output (n 1))
  (let* ([outputs (feedforward inputs weights biases)]
         [errors (backprop weights outputs expected-output)]
         ...))
```

During the gradient descent optimization process, the `gradient-descent` function calculates the outputs using the `feedforward` function, then computes the errors for the weights and biases using the `backprop` function. The calculated errors are used to update the weights and biases in the `adjust-parameters` function, allowing the network to learn and minimize the difference between its predictions and the expected output.

Checklist:

- ~~Changing the Activation functions~~

```
(let-values ([(trained-weights trained-biases) (gradient-descent weights biases inputs expected-output 256)])
  (display (second (last (feedforward inputs trained-weights trained-biases)))) (newline))
```

- ~~Have a function to define new weights~~
- ~~Things to take a look into:~~
    - ~~spring model k times x~~