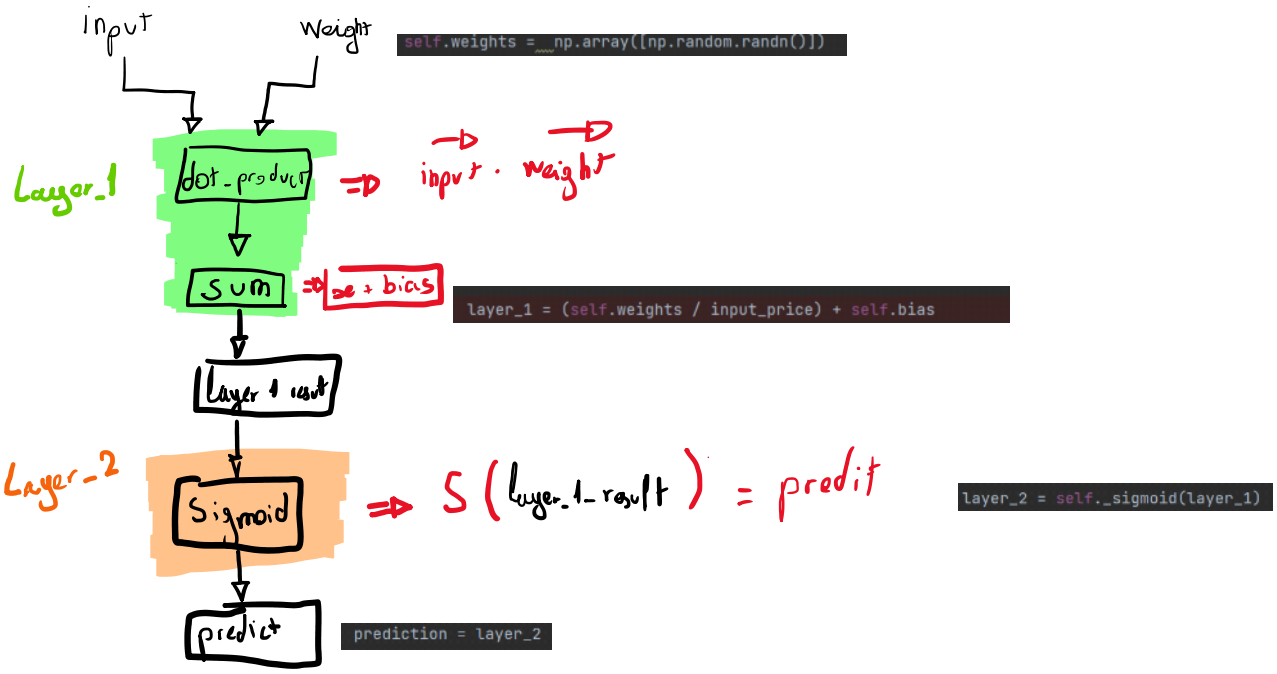
AI Home-Work

09:33

Sources: https://realpython.com/python-ai-neural-network/#neural-networks-main-concepts

This ANN has for goal to find a similar vector as the vector we are giving in input.
This is a 2 layer NN organised this way:



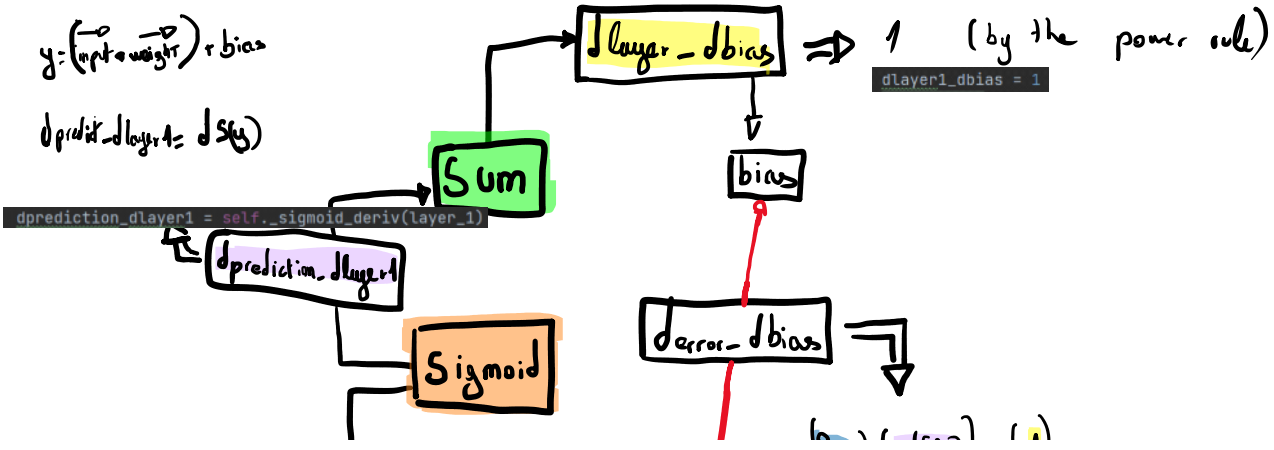
Our goal here is to compare the input to the weight with the dot produt. Then we sum this to our bias. After, we put it in the sigoid function to have a result between 0 and 1, it's easier to read for the AI. Now that we have a predition we can go furter in the learning system.

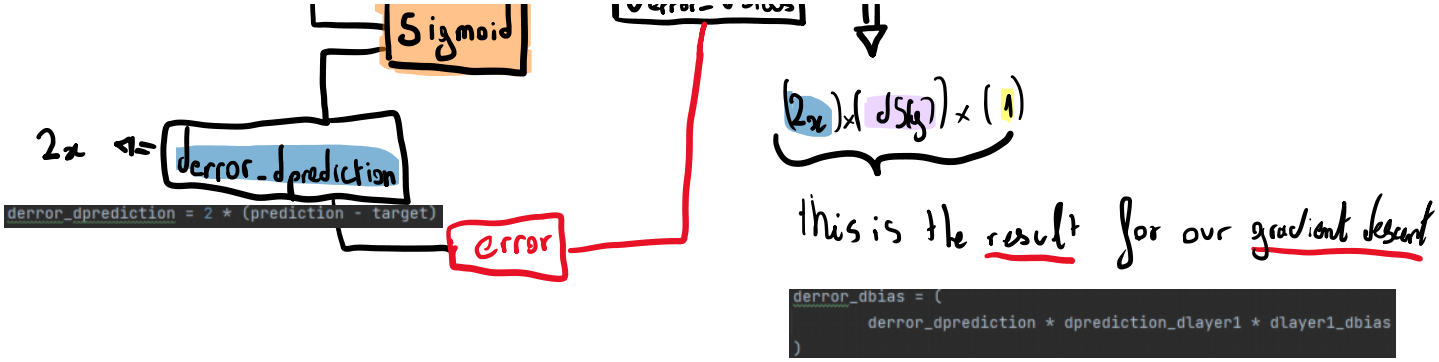
Computing the error of the prediction

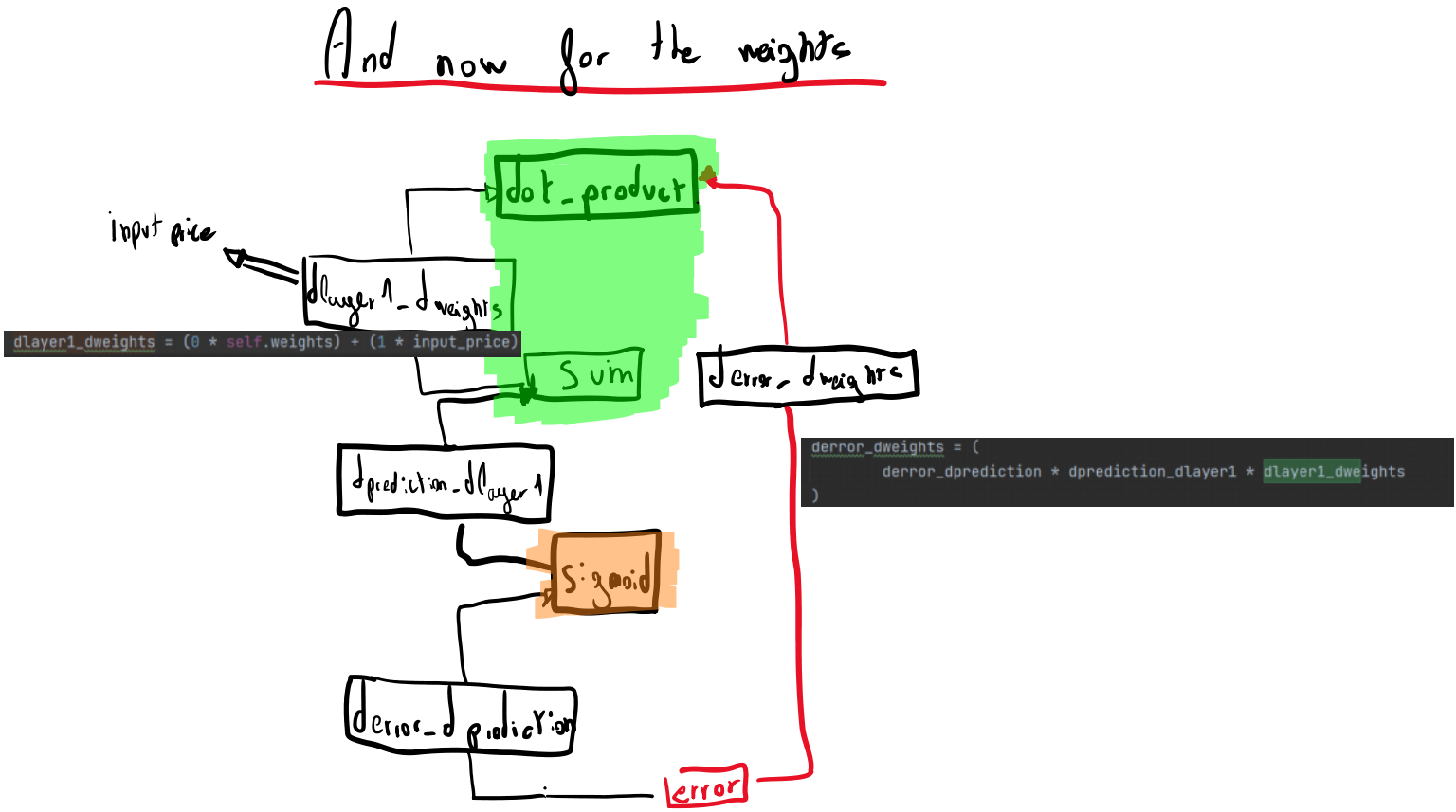
To have an error we need a targeted prediction. And then we have to do the difference between the prediction and the target to compare them.there is a lot of functions that can be used to compute the error, named cost functions or loss functions. Here we are using the mean squared error.

Basicaly it's just a square of our result: $f(x)=x^2$
And x is the difference betwenn our predict and the target "(predict - target)"

And now we have to know how to undersand and use this to reduce the error it's doing.
If we look further more in the function we can see that there is only one minimal value that the result can take.
To know wich way to go we will need to interest ourselves to the method called "Gradient descent".
To use it we have to derivate our function, so df(x) = 2x
If 2x is positive we need to reduce x in order to reduce the error.
If 2x is negative we have to add to this vaue in order to reduce the error.
But we also need to look the dervatives of the entire function we are using in order to update our paraeters.
So we will use the Chain Rule.
The goal here is to look the partial derivatives and to put them all together.







Updating the parameters

This part is not hard or even verry mathematical. We are just substracting the derivative we gat with the value of the repective variable. And we are factorising it by the learning rate to ensure the derivative is not too high or too low in some cases.

```
self.bias = self.bias - (derror_dbias * self.learning_rate)
self.weights = self.weights - (
        derror_dweights * self.learning_rate
)
```

Training with some data

This part is explain line by line on the souces i gave. So i will talk only about how it works and how to make it work well. It's very important to have a coherent dataset and to atribute the good taget to those datas. Beacause the system woks by doing many iterations and adjusting the bias and weight in function of the data he gets. And the more iteration it's doing the batter it will get on those datas. But if the datas are wrong it's gonna train to be wrong. In the program, mathplot lib is used to visualise the cumlative erros the network is doing for each itteration. You can test and have some fun moving the data to see how it reacts.

```python
class NeuralNetwork:
    # ...

    def train(self, input_vectors, targets, iterations):
        cumulative_errors = []
        for current_iteration in range(iterations):
            # Pick a data instance at random
            random_data_index = np.random.randint(len(input_vectors))

            input_vector = input_vectors[random_data_index]
            target = targets[random_data_index]

            # Compute the gradients and update the weights
            derror_dbias, derror_dweights = self._compute_gradients(
                input_vector, target
            )

            self._update_parameters(derror_dbias, derror_dweights)

            # Measure the cumulative error for all the instances
            if current_iteration % 100 == 0:
                cumulative_error = 0
                # Loop through all the instances to measure the error
                for data_instance_index in range(len(input_vectors)):
                    data_point = input_vectors[data_instance_index]
                    target = targets[data_instance_index]

                    prediction = self.predict(data_point)
                    error = np.square(prediction - target)

                    cumulative_error = cumulative_error + error
                cumulative_errors.append(cumulative_error)

        return cumulative_errors
```

- **Line 8** picks a random instance from the dataset.
- **Lines 14 to 16** calculate the partial derivatives and return the derivatives for the bias and the weights. They use `_compute_gradients()`, which you defined earlier.
- **Line 18** updates the bias and the weights using `_update_parameters()`, which you defined in the previous code block.
- **Line 21** checks if the current iteration index is a multiple of `100`. You do this to observe how the error changes every `100` iterations.
- **Line 24** starts the loop that goes through all the data instances.
- **Line 28** computes the `prediction` result.
- **Line 29** computes the `error` for every instance.
- **Line 31** is where you accumulate the sum of the errors using the `cumulative_error` variable. You do this because you want to plot a point with the error for *all* the data instances. Then, on line 32, you append the `error` to `cumulative_errors`, the array that stores the errors. You'll use this array to plot the graph.

À partir de l'adresse <https://realpython.com/python-ai-neural-network/#neural-networks-main-concepts>