



KOSZALIN UNIVERSITY OF TECHNOLOGY

APPLICATIONS OF ARTIFICIAL INTELLIGENCE
PROJECT REPORT

Handwritten text symbol recognition with deep neural networks

Paweł Frankowski Kacper Ochnik

supervised by
Dr. Adam Słowik

January 31, 2024

Contents

1	Introduction - Kacper	2
2	Our goal - Pawel [Done]	3
2.1	Specific objectives	3
2.2	Expected Outcomes	3
3	Decision boundary - Kacper	4
4	Weights and Biases - Pawel [Done]	5
4.1	Initialization	5
4.2	Weighted sum of inputs	5
5	Hidden layers - Kacper	6
6	Activation functions - Pawel [Done]	7
6.1	Sigmoid function	7
6.2	The derivative of the sigmoid function	7
7	Cost function - Kacper	9
8	Gradient descent - Pawel	10
9	Cost landscape - Kacper	11
10	Learning algorithm - naive approach - Pawel	12
11	Learning algorithm - calculus approach - Kacper	13
12	Learning algorithm - digit recognition - Pawel	14
13	Chain rule - Pawel/Kacper	15
14	Backpropagation - Kacper	16
15	Testing the network - Kacper	17
16	Conclusion - Pawel	18

1 Introduction - Kacper

Handwritten text symbol recognition with deep neural networks.

2 Our goal - Pawel [Done]

The primary objective of our project is to develop a handwritten text symbol recognition system using deep neural networks. We aimed to create a model capable of accurately identifying and classifying handwritten digits ranging from 0 to 9 on a matrix of 28x28 pixels.

2.1 Specific objectives

In pursuit of our overarching goal, we have identified the following specific objectives:

1. **Project setup** - Set up the project environment and install the necessary libraries and packages.
2. **Code implementation** - Write Python code to implement the deep neural network architecture. This includes developing modules for data preprocessing, model training, and evaluation.
3. **Data Collection** - Gather a comprehensive dataset of handwritten digits (0 to 9) in a 28x28 pixel matrix format from MNIST.
4. **Learning** - Train the model using the collected dataset.
5. **Optimization** - Improve the model's accuracy through optimization techniques. Accelerate computational efficiency for faster calculations.
6. **Testing** - Create testing GUI for the trained model. Evaluate the model's performance metrics.
7. **Documentation** - Write a comprehensive report documenting the project's objectives, methodology, and outcomes.

2.2 Expected Outcomes

Upon successful completion of our project, we anticipate achieving the following outcomes:

- Develop a robust deep neural network model capable of recognizing and classifying handwritten digits from 0 to 9.
- Train the model to achieve a acceptable level of accuracy.

3 Decision boundary - Kacper

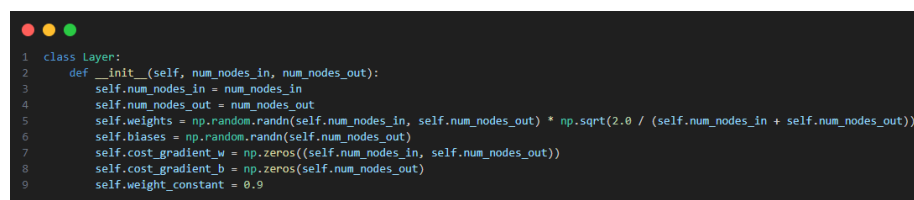
...

4 Weights and Biases - Pawel [Done]

Weights are indicators of the importance of the input in predicting the final output. Biases are essential as they allow the network to have some flexibility in fitting the data. In the training process of our neural network, the weights and biases play a crucial role in determining the model's performance. Here, we discuss the initialization, calculation, and adjustment of weights and biases in the network.

4.1 Initialization

The **Layer** class represents a single layer in the neural network. It contains information about the number of nodes in and out, weights, biases, and gradients. The neural network is initialized with random weights using the Xavier/Glorot initialization technique. This method helps in achieving better convergence during training.



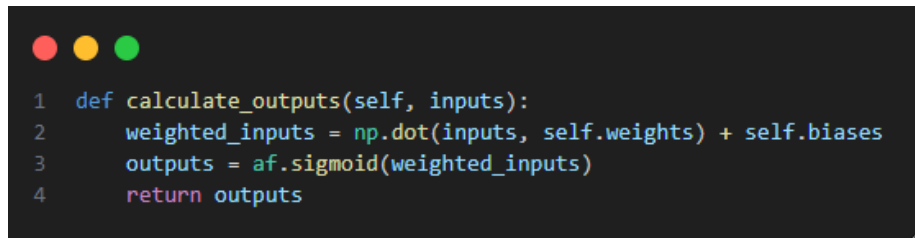
```
1 class Layer:
2     def __init__(self, num_nodes_in, num_nodes_out):
3         self.num_nodes_in = num_nodes_in
4         self.num_nodes_out = num_nodes_out
5         self.weights = np.random.randn(self.num_nodes_in, self.num_nodes_out) * np.sqrt(2.0 / (self.num_nodes_in + self.num_nodes_out))
6         self.biases = np.random.randn(self.num_nodes_out)
7         self.cost_gradient_w = np.zeros((self.num_nodes_in, self.num_nodes_out))
8         self.cost_gradient_b = np.zeros(self.num_nodes_out)
9         self.weight_constant = 0.9
```

Figure 1: Initialization of random biases and weights using Xavier/Glorot technique

When a neural network is first trained, it is first fed with input. Since it isn't trained yet, we don't know which weights should we use for each input. And so, each input is randomly assigned a weight. It will very likely give incorrect output.

4.2 Weighted sum of inputs

In the **Layer** class, the calculation of outputs begins with the computation of weighted inputs. For each node in the layer, the weighted inputs are calculated using the formula.:



```
1 def calculate_outputs(self, inputs):
2     weighted_inputs = np.dot(inputs, self.weights) + self.biases
3     outputs = af.sigmoid(weighted_inputs)
4     return outputs
```

Figure 2: The dot product of inputs and weights, representing the weighted sum of inputs

The `calculate_outputs` method calculates the dot product of inputs and weights, representing the weighted sum of inputs. The biases are then added to this sum to introduce flexibility in fitting the data. Finally, the result is passed through the sigmoid activation function to introduce non-linearity to the model, and the outputs are obtained.

5 Hidden layers - Kacper

...

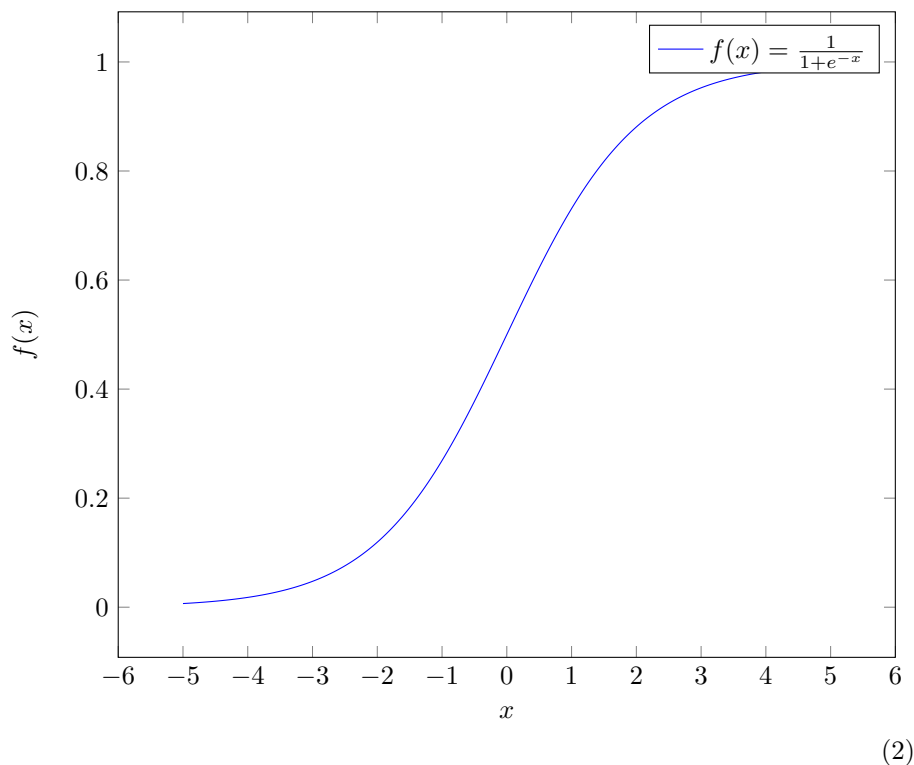
6 Activation functions - Pawel [Done]

Activation functions play a crucial role in neural networks, introducing non-linearity to the model and allowing it to learn complex relationships in the data. Here, we describe various activation functions implemented in the `activation functions` module.

6.1 Sigmoid function

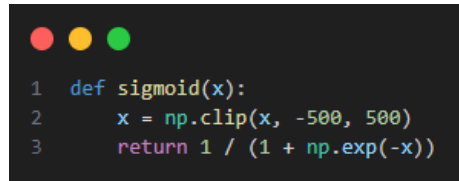
The sigmoid function squashes its input to the range $(0, 1)$, making it suitable for binary classification problems.

$$f(x) = \frac{1}{1 + e^{-x}} \quad (1)$$



6.2 The derivative of the sigmoid function

The derivative of the sigmoid function is used in the backpropagation algorithm to calculate the gradients of the cost function with respect to the weights and biases. The derivative of the sigmoid function is given by the formula:

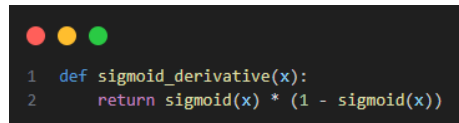
A code editor window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains three lines of Python code: a function definition for sigmoid(x) that clips x to the range [-500, 500] and returns 1 / (1 + np.exp(-x)).

```
1 def sigmoid(x):  
2     x = np.clip(x, -500, 500)  
3     return 1 / (1 + np.exp(-x))
```

Figure 3: Sigmoid function

Limiting the values of x to prevent overflow. Overflow refers to a situation where the exponential function in the sigmoid formula produces very large positive values, causing numerical instability in calculations. The exponential function, especially when dealing with large positive inputs, can lead to floating-point overflow, which means the result becomes too large to be represented within the numerical precision of the system.

$$f'(x) = f(x) \cdot (1 - f(x)) \quad (3)$$

A code editor window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains two lines of Python code: a function definition for sigmoid_derivative(x) that returns sigmoid(x) * (1 - sigmoid(x)).

```
1 def sigmoid_derivative(x):  
2     return sigmoid(x) * (1 - sigmoid(x))
```

Figure 4: The derivative of the sigmoid function

7 Cost function - Kacper

...

8 Gradient descent - Pawel

The cost gradients for weights and biases are initialized with zeros, and during training, these gradients will be updated based on the backpropagation algorithm.

```
1 class Layer:
2     def __init__(self, num_nodes_in, num_nodes_out):
3         self.num_nodes_in = num_nodes_in
4         self.num_nodes_out = num_nodes_out
5         self.weights = np.random.randn(self.num_nodes_in, self.num_nodes_out) * np.sqrt(2.0 / (self.num_nodes_in + self.num_nodes_out))
6         self.biases = np.random.randn(self.num_nodes_out)
7         self.cost_gradient_w = np.zeros((self.num_nodes_in, self.num_nodes_out))
8         self.cost_gradient_b = np.zeros(self.num_nodes_out)
9         self.weight_constant = 0.9
```

Figure 5: Initialization a vector of zeros with the same length as the bias vector

This matrix will be used to accumulate the gradients of the cost function with respect to the weights during the backpropagation process.

9 Cost landscape - Kacper

...

10 Learning algorithm - naive approach - Pawel

...

11 Learning algorithm - calculus approach - Kacper

...

12 Learning algorithm - digit recognition - Pawel

...

13 Chain rule - Pawel/Kacper

...

14 Backpropagation - Kacper

...

15 Testing the network - Kacper

...

16 Conclusion - Pawel

This is the conclusion of the document.