ACML - Backpropagation

Dimitrios Gagatsis(6258651), Christos Kaparakis(6258164)

November 3, 2020

1 How To Use This

1. Run the **backpropagation.py** python file.
2. Run **backpropagation(inputs,weights_l1_l2,weights_l2_l3, bias_inp, bias_hidden, learning_rate, lambda_value)**

The parameters are:

- **inputs:** the input examples.
- **weights_l1_l2:** the weights of the nodes in the input layer
  Default values set to random values near zero.
- **weights_l2_l3:** the weights of the nodes in the hidden layer
  Default values set to random values near zero.
- **bias_inp:** the bias associated with the hidden layer
  Default values set to 1.
- **bias_hidden:** the bias associated with the output layer
  Default values set to 1.

- **learning_rate:** learning rate α
  Default value set to **2**
- **lambda_value:** weight decay parameter λ
  Default value set to **0.00001**

These parameters can be changed and allow experimentation with the network.

2 Introduction

The implementation of the back-propagation algorithm to train a Neural Network, as well as the interpretation of the Neural Network are the goals for the first assignment in the course Advanced Concepts of Machine Learning. This report aims to inform the user for the training and execution of the Neural Network.

3 Description of the Neural Network

The delivered Network is a simple ("vanilla") Neural Network with 8 input nodes of 0's and 1's, 1 hidden layer consisting of 3 nodes, and an output layer with 8 output nodes.

The assignment's objective is to train the network by using the back-propagation algorithm so that the output is exactly the same as the input.

For example:
input:(1,0,0,0,0,0,0,0)
output:(1,0,0,0,0,0,0,0)

4 Learning Performance

7801 iterations are required for the system to reach convergence.

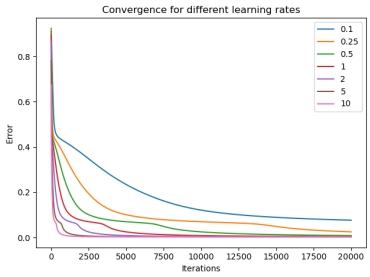
As stated in the previous section, the desired output is 1's and 0's. The Neural Network is not fully successful in this particular detail, since it outputs values close to 0 or values close to 1.

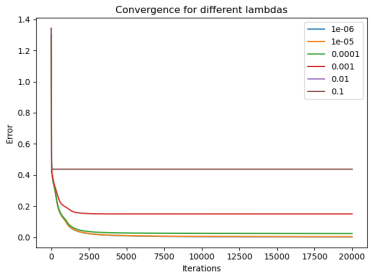
Even in that way, by reducing the values close to 0 to 0, and increasing the values close to 1 to 1, we manage to succeed in predicting the correct outcome.

For the system to converge, we set a threshold value of 0.005 for the mean squared error of our network.

5 Choosing the parameters

In order for our Neural Network to produce better results, we can experiment with the parameters learning_rate (α) and lambda_value (λ).





These parameters determine the weights and biases values when they are updated and how fast the output converges to input. After repetitions, we noticed that the larger the learning_rate (without being excessive due to a possible verfit) and the smaller the lambda_value, the faster the error is detected and the faster our system converges to the target value.

6 Interpretation of the learned weights

The weights are the coefficients that determine the node values in every layer. Also, they are dependent on learning_rate and lambda_value and determine to a large extent how effective they are.

The purpose of the weights is that they show how the nodes of a layer influence the values of the nodes in the next layer.

For example:
(3.64321468e+00, 5.00656746e+00, 3.64387728e-03) are the weights of the first node for the nodes of the hidden layer. This is the Neural Network moving the information from 8 nodes to three.
(-8.41531644, -6.8093618 , -6.01462414, -8.65427908, 6.12263062, 5.06103631, 3.78991223, 6.9334295) are the weights of the first node of the hidden layer for the output nodes.

As was stated earlier, the final output does not give 0's or 1's but decimal numbers close to them. The weights from the hidden layer to the output layer show how the nodes of the hidden layer influence the probability for every node in the output layer to be 1. The values of every node in the output layer, are the probabilities for how close to 1 these nodes are.