Machine Learning - Homework 3

João Cardoso 81361

Mafalda Ferreira 81613

Mário Cardoso 83523

Group 3 - Alameda

Code: https://colab.research.google.com/drive/1Ha34mK_P2GYuckJK5fGdzV0Ng5nqnF9-

1 Description of the code

For this exercise three crucial blocks of code were used, each one for the corresponding implementation (linear unit with quadratic error as loss function, sigmoid unit with quadratic error as loss function and sigmoid unit with cross entropy as loss function). Each block is divided in 4 parts: the activation function, the classification function, the training block and the results block. To better evaluate our models we split the provided iris dataset in a training and test dataset. We did this utilizing sklearn train test split method in a non-stratified manner. For the first exercise, We split the dataset so that the training dataset contains 70% of observations and the test dataset contains 30%. For learning we utilized exclusively the training dataset. For the second exercise we followed a similar split strategy and we applied some pre-processing techniques, namely we replaced categorical (non-numerical) variables with appropriate numerical values. For all the tensorflow models we utilized 1 hidden layer with varying parameters.

2 Exercise 1

2.1 Formula explanation

All neurons were trained using gradient descent with the following update rule:

$$w_j^{new} = w_j^{old} + \Delta(w_j)$$

With

$$\Delta(w_j) = -\eta \times \frac{\partial E}{\partial w_j}$$

2.1.1 Linear Unit with Quadratic Error as Loss Function

Quadratic Loss Function is defined as follows:

$$E(w) = \sum_{k=1}^{N}(t_k - o_k)^2$$

Using a linear activation function:

$$o_k = \sum_{j=0}^{D} w_j \times x_j$$

To find the local minimum of the function E(w) we apply the gradient descent rule, which results in:

$$\frac{\partial E}{\partial w_j} = -\sum_{k=1}^{N}(t_k - o_k) \times x_{kj}$$

2.1.2 Sigmoid Unit with Quadratic Error as Loss Function

$$net = \sum_{j=0}^{D} w_j \times x_{kj}$$

Using the activation function:

$$o_k = \sigma(net) = \frac{1}{1 + e^{(-net)}}$$

and the loss function used for the previous unit we find the local minimum of the function E(w) by applying the gradient descent rule. This results in:

$$\frac{\partial E}{\partial w_j} = -\sum_{k=1}^{N}(t_k - o_k) \times \sigma(net_{kj}) \times (1 - \sigma(net_{kj})) \times x_{kj}$$

2.1.3 Sigmoid Unit with Cross Entropy as Loss Function

Using the same activation function defined in the section above, with the following loss function:

$$E(w) = -\sum_{k=1}^{N}(t_k log(o_k) + (1 - t_k) log(1 - o_k))$$

Applying the gradient descent rule, we obtain the following result:

$$\frac{\partial E}{\partial w_j} = -\sum_{k=1}^{N}(t_k - o_k) \times x_{kj}$$

2.2 Discussion of the results

2.2.1 Convergence Rates

Through analysis of the error for each iteration during the training process we can understand the convergence rates for each method. All of the tested methods are able to converge to a point where the error does not change significantly between training iterations. By changing the weights the rate of convergence can change significantly. While testing the sigmoid unit with Quadratic Error we realized that initializing the weights to 1 would require a significantly higher amount of training iterations for it to converge than by initializing the weights to 0. This means that the gradient descent could be finding a local minimum when it was initialized to one and would thus require more iterations to find the true minimum. Changing the learning rate also affects the rate of convergence. By using a higher learning rate we noticed that the error between iterations was rising and lowering until it eventually converged. This was a result of the considerable change of the weights' values that would occur at each training step, which meant the gradient descent was overshooting the true minimum point. However, when we used a lower learning rate the error between iterations kept going down. This is because each step, the change in weights was smaller and, as a result, the gradient descent did not overshoot the minimum.

Structure	Training Accuracy	Test Accuracy
Linear w/ Quadratic Loss	44.76%	44.44%
Sigmoid w/ Quadratic Loss	80.0%	86.66%
Sigmoid w/ Cross Entropy	79.04%	84.44%

3 Exercise 2

3.1 Discussion of the results

3.1.1 Convergence Rates

By running several models with several different parameters we noticed that all models eventually converged to a certain loss value (between 1000 and 2000 epochs). The loss values for the initial epochs varied each time we ran due to different weight initialization, but converged eventually. After selecting the best model we then calculated how the training and validation set accuracy varied over each epoch. We noticed the accuracy values were very volatile in the initial epochs but eventually converged to a similar value.

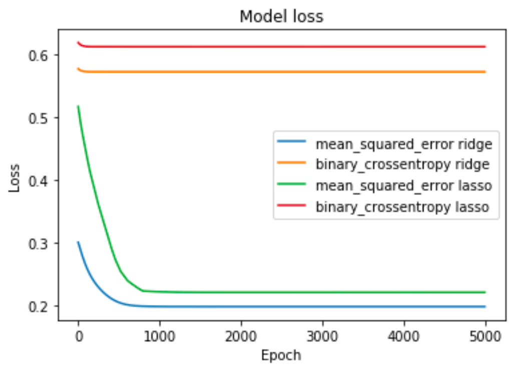
3.1.2 Method to select best method

Based on the calculated figures and loss values we compared each model to see which model offered the lowest training loss values. We noticed some models had converged to very similar results, namely the model whose activation function was relu, with mean squared error loss and l2 regularization, and another model with the same parameters but using leakyRelu showed almost identical results. The model using the same parameters but sigmoid activation also displayed very similar results. Given these results we decided to select relu with the parameters mentioned above as the best model, and analyzed how the training and test accuracy varied over epochs. We also calculated the final training and test loss, aswell as the final training and test accuracy.

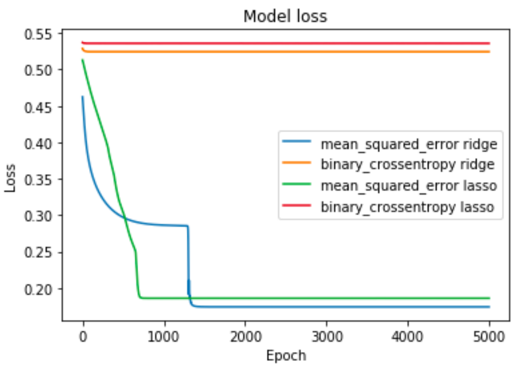
3.1.3 Regularization

Regularization is used to dampen the effect of variance and to do so without inflicting too much of a change in bias. It achieves this by adding a shrinkage quantity to the loss function that also needs to be minimized. This will punish coefficients with high values and in Lasso regularization will also be able to set coefficients to zero, effectively working as a feature selector. After running several models with different values for both regularization types and found that Ridge regression consistently outperformed Lasso regression. This could be a result of Lasso regression filtering out correlated features which would lead to some information loss given the small number of features in our dataset. Lasso regression would be more applicable when there are a considerable number of features, so that it can make use of its feature selection property without losing too much information in the process.

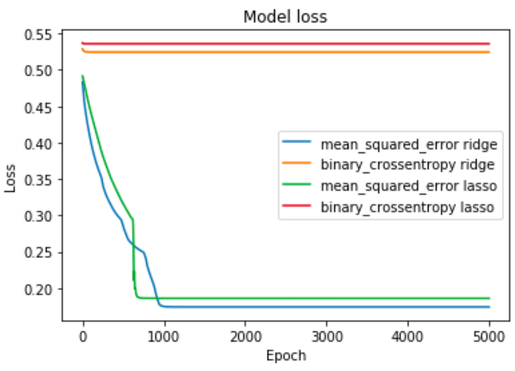
3.1.4 Sigmoid Activation Function



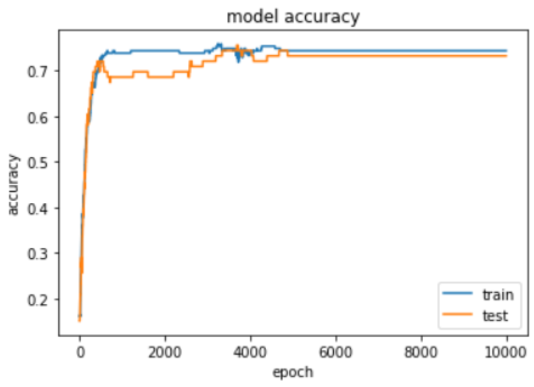
3.1.5 Relu Activation Function



3.1.6 Leaky Relu Activation Function



3.1.7 Best Model Accuracy Comparison


model accuracy
accuracy
epoch
train
test