Instituto Superior Técnico, Universidade de Lisboa
Master of Science in Computer Science and Engineering

Deep Learning

Homework 1

André Páscoa (110817)
Francisca Silva (111006)

December, 2023

Contributions

This work was collaboratively completed by André Páscoa and Francisca Silva. Both contributors equally participated in the research, analysis, and composition of this assignment.

Question 1

Question 1 focuses on medical image classification using linear classifiers and neural networks on the OCTOMNIST dataset, evaluating the performance of Perceptron and Logistic Regression models while comparing them with a constructed multi-layer perceptron (MLP) without using neural network toolkits. This analysis aims to discern the efficacy and complexities of different models in addressing the retinal disease classification task.

1. a)

Testing the perceptron model, at epoch 20 its train accuracy was 46.54% with validation accuracy of 46.10%, and a final test accuracy of 34.22%. The resulting validation accuracies as a function of the epoch number can be seen in figure 1

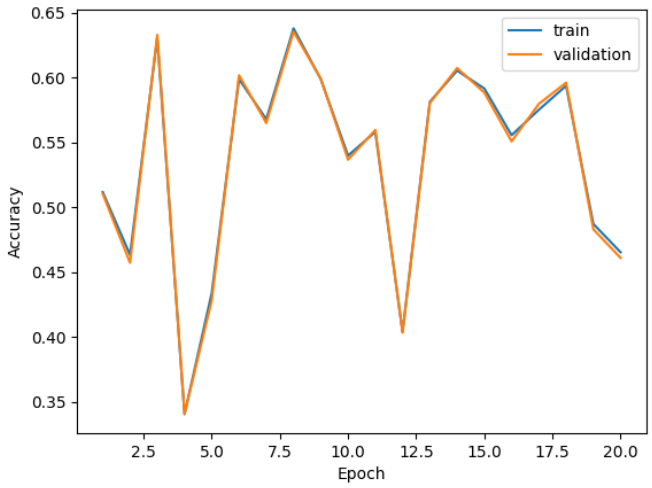

Figure 1: Accuracies for perception as a function of the epoch number

1. b)

Regarding the Logistic Regression model, the training, validation and test accuracies can be seen in table 1. The resulting train and validation accuracies can be seen in figure 2.

Comparing both plots, one can see that the model trained with learning rate 0.01 training was more unstable than the model trained with learning rate 0.001,

Table 1: Logistic Regression Accuracies

Metric	**lr=0.01**	**lr=0.001**
Train ACC	0.6609	**0.6625**
Val ACC	0.6568	**0.6639**
Final test ACC	0.5784	**0.5936**

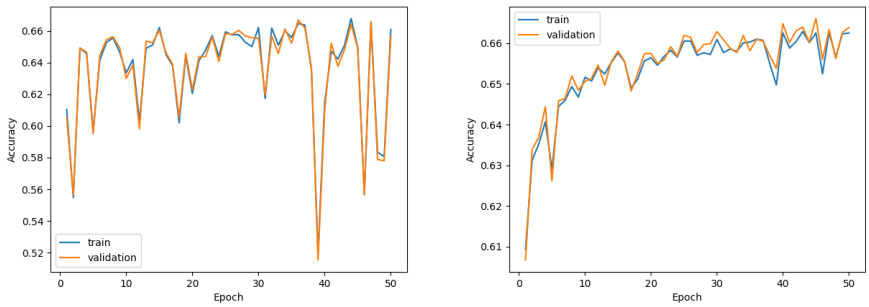

Figure 2: Accuracies for logistic regression with learning rate 0.01 and 0.001 as a function of the epoch number

this is because there was an overshooting of the minima of the loss function, due to the high learning rate.

2. a)

Since logistic regression is a linear classifier, it can only classify linearly separable data, while an MLP with a Rectified Linear Unit (ReLU) activation function and enough hidden units follows the universal approximation theorem, thus being able to approximate any function. Regarding the ease of training, logistic regression is a convex optimization problem, thus, since convex optimization with gradient descent is always guaranteed to find the global optima, it's more stable and easier to train compared to an MLP, which is a non-convex function.

2. b)

Testing the Multi Layer Perceptron with ReLU activations model, its train accuracy was 79.71% with validation accuracy of 78.77% and final test accuracy of 76.56%. The resulting train loss, and the train and validation accuracies as a function of the epoch number can be seen in figure 3

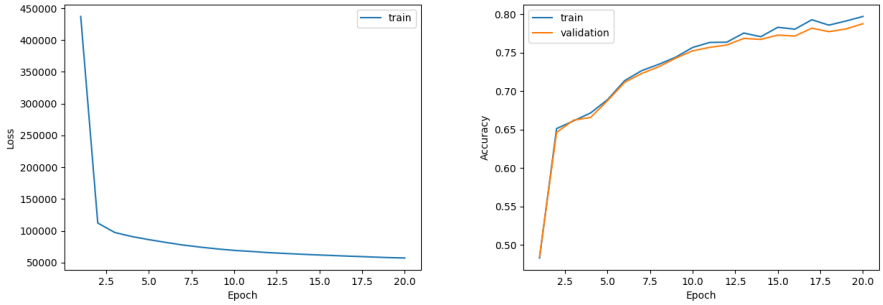

Figure 3: Train loss and accuracies for MLP as a function of the epoch number

Question 2

This question focuses on medical image classification using an Automatic Differentiation (autodiff) toolkit. After manually implementing gradient backpropagation in the previous question, this task involves implementing the same system using a deep learning framework with autodiff, in this case PyTorch. The goal was to construct a logistic regression model using stochastic gradient descent and fine-tune hyperparameters for optimal validation accuracy, followed by building a feed-forward neural network while analyzing its performance variations with different regularization methods, hyperparameter configurations and batch sizes.

1.

The best model was the one trained with a learning rate of 0.01. The model obtained a train accuracy of 93.70%, validation accuracy of 65.35% and 62% test accuracy. Its training loss and validation accuracy as a function of epoch number can be seen in 4

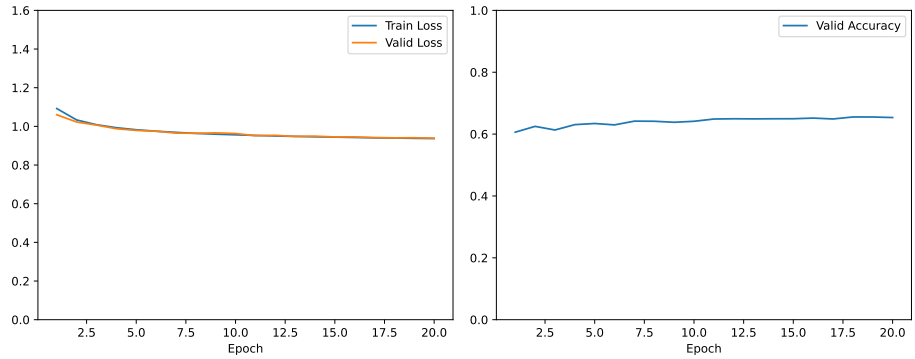

Figure 4: Train loss and accuracies for Logistic Regression as a function of the epoch number

2. a)

After training both models, the training and validation losses along with the validation accuracy for the model trained with batch size 1024 can be seen in figure 5. For the model trained with batch size 16, these are displayed in figure 6.

The best test accuracy of 78.83% was achieved by the model trained with batch size 16.

Analyzing both sets of plots 5 and 6, our hypothesis for the cause of the higher training loss in the training run with batch size 1024 is that it does not do as many passes through the data nor update its weights as frequently as the one with batch size 16, thus, for the same amount of epochs, it is slower to converge, even though the gradient is more precise. Since PyTorch is able to parallelize batch training with the GPU, as long as the batch can fit inside GPU VRAM, then it should be faster to train for the same amount of epochs

compared to a smaller batch size that needs to do more weight updates. Our model's execution times are aligned with this expectation, where the model trained with batch size 1024 took 32.52 seconds while the one trained with batch size 16 took 153.10 seconds.

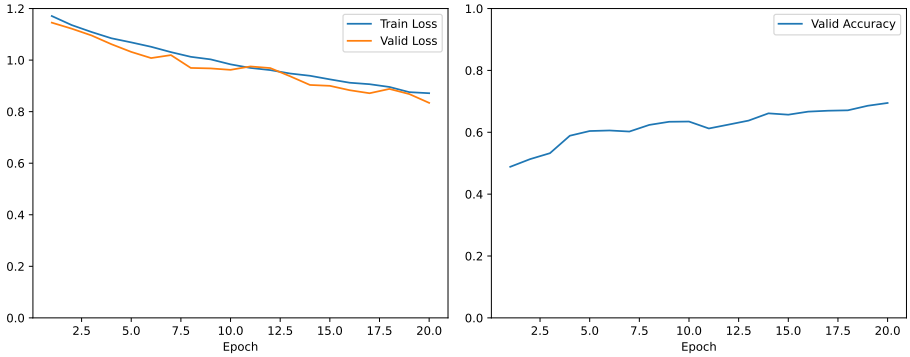

Figure 5: Train, validation loss and validation accuracy for MLP trained with batch size 1024

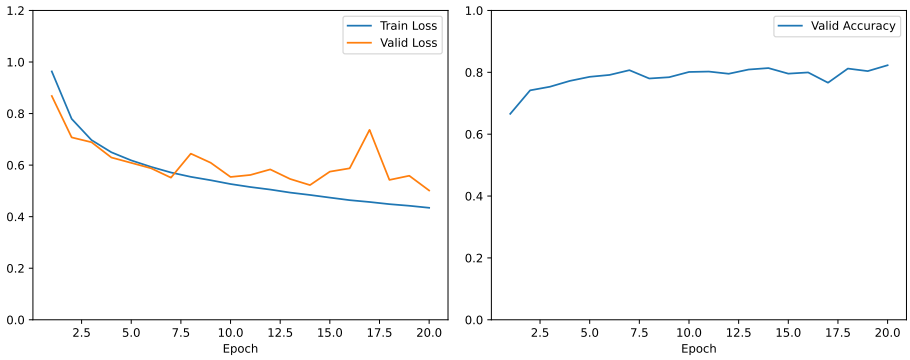

Figure 6: Train, validation loss and validation accuracy for MLP trained with batch size 16

2. b)

Training the model with learning rates 1, 0.1, 0.01 and 0.001, we found that the model trained with learning rate 0.1 achieved the best test accuracy of 78.83% and its training, validation loss and validation accuracy can be seen in plots 7. The worst model was the one trained with learning rate 1.0, and as before its plots can be seen in 8. We propose that the high learning rate causes the model to constantly overshoot, which makes it diverge, while the model trained with learning rate 0.1 does not overshoot and thus is able to converge. The other models trained with even smaller learning rates (i.e. 0.01 and 0.001) were not yet able to converge, not because of overshooting, but because they converge slowly.

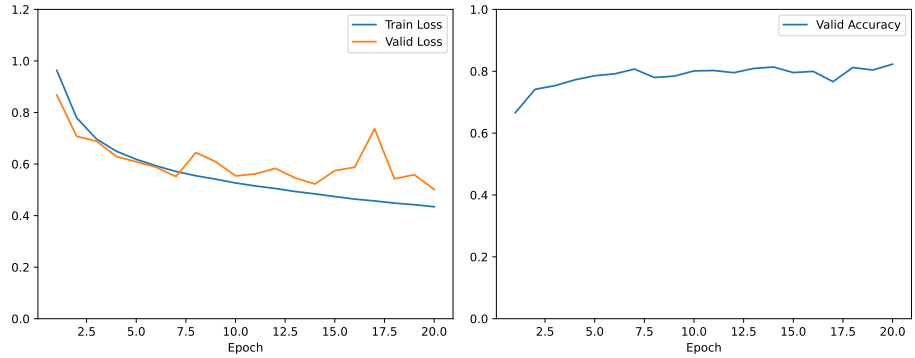

Figure 7: Train loss and accuracies for MLP as a function of the epoch number for best learning rate of 0.1

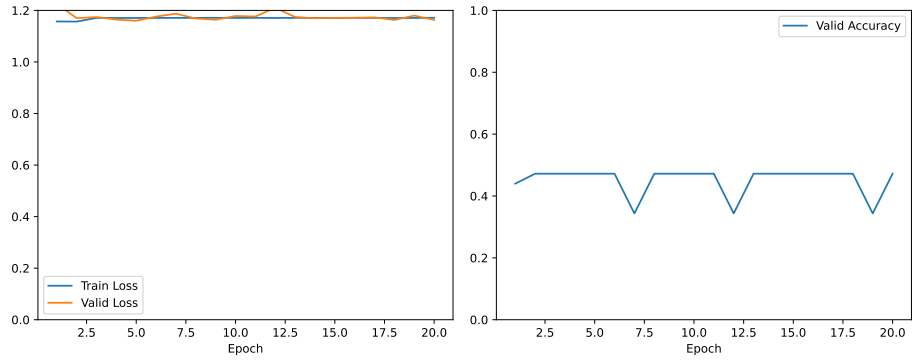

Figure 8: Train loss and accuracies for MLP as a function of the epoch number for worse learning rate of 1.0

2. c)

Training with the three configurations, we found that the best test accuracy of 79.40% was achieved by the model with dropout regularization. The best configuration with regard to validation accuracy was the model trained with dropout, achieving 85.88%, while the worst one was the one trained with l2 regularization, achieving 84.07% validation accuracy. Plots for the train, validation loss, and validation accuracy for the model trained with dropout can be seen in plots 9 and for the l2 regularization in 10. Regarding the test accuracy, the models with dropout and l2 regularization performed the best, with 79.40% and 77.13% test accuracy, while the one without regularization only achieved 76.56%. Thus, we can infer that the model is generalizing better when regularization is used. Regarding the validation accuracy, even though the model trained with l2 regularization achieved worse final validation accuracy, the difference in validation accuracy is small. The model trained without regularization suffered from overfitting, as can be seen in plots 11, where the validation loss and training loss contain a larger gap than l2 regularization and an even greater gap than dropout regularization.

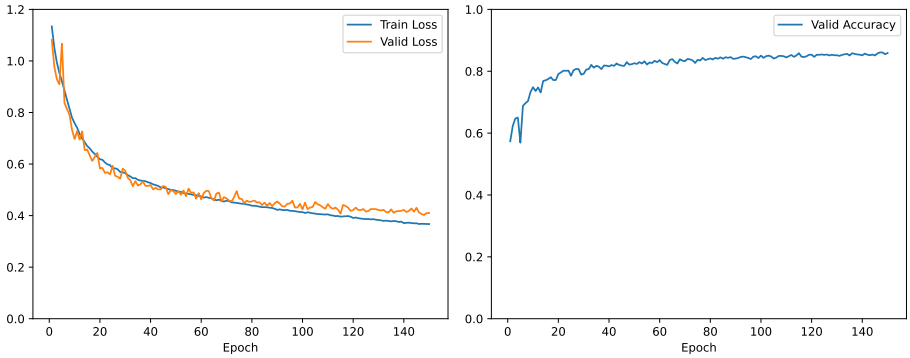

Figure 9: Train loss and accuracies as a function of the epoch number for the model trained with dropout p=0.2

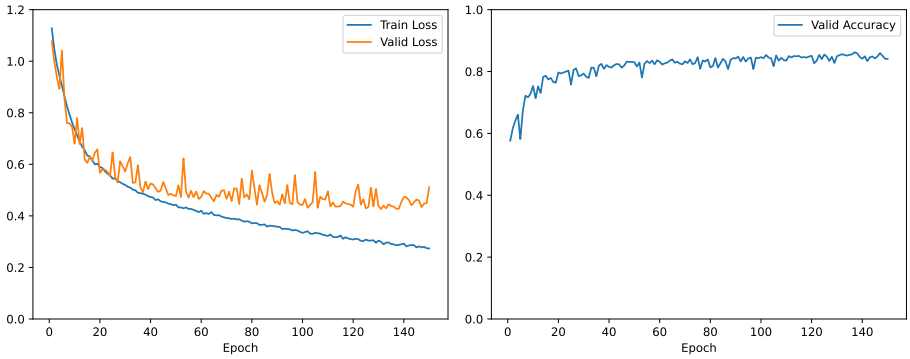

Figure 10: Train loss and accuracies as a function of the epoch number for the model trained with l2 regularization = 0.0001

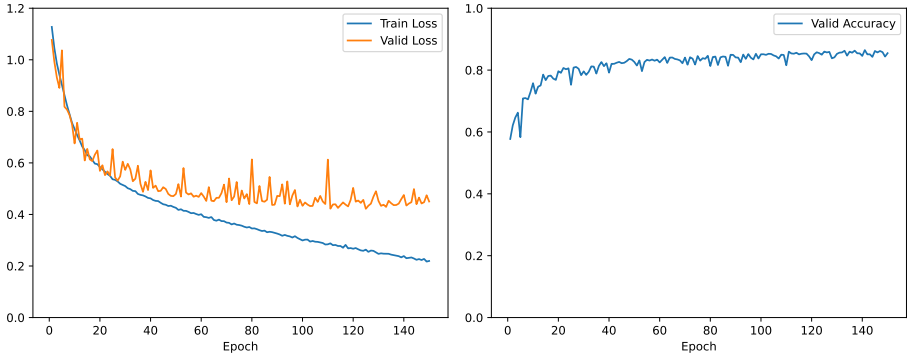

Figure 11: Train loss and accuracies as a function of the epoch number for the model trained with no regularization

Question 3

The last question requires an analysis of the limitations and robustness of multilayer perceptrons to compute whether the result of summing all the inputs (1 or -1) is within a specified range ($A \leq sum(x) \leq B$).

a)

To show that the function above cannot generally be computed with a single perceptron, a simple counter-example is assuming D = 2, A = -1, and B = 1, and computing all possible outcomes.

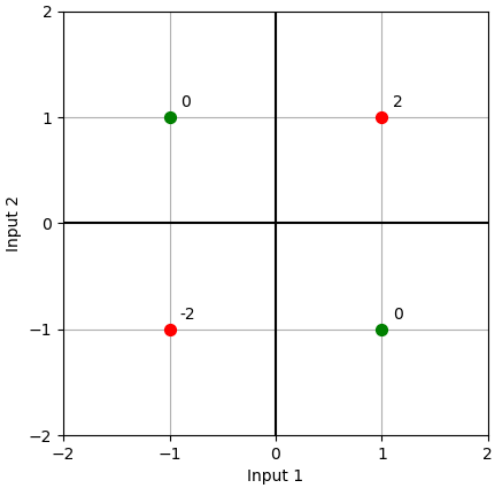

Figure 12: Points representing all possible outcomes of the problem for 2 inputs, and A=-1, B=1

As visible in figure 12, the result of the sum is not linearly divisible, which is all a single perceptron can compute.

b)

To answer this question, there will be an explanation of the purpose of each hidden unit, taking into account how to mitigate the appearance of perturbations in the network. Since the problem requires the network to check if a number is in an interval, each hidden unit must perform an inequality, as explained below.

First hidden unit

This unit must check if the sum of all inputs is bigger than A, accounting for an infinitesimal error in each unit, for which the sum will be represented by $\frac{e1}{e2}$, given both e1 and e2 are integers. This is represented in the following inequality.

$$\sum_{i=1}^{D} x_i + \frac{e1}{e2} \geq A \Leftrightarrow e2 \times \sum_{i=1}^{D} x_i + e1 \geq e2 \times A \Leftrightarrow \sum_{i=1}^{D} e2 \times x_i + e1 - e2 \times A \geq 0$$

Since all the weights and biases must be integers, everything is multiplied by e2, and the sign activation function performs the inequality, returning a boolean value.

This way all the weights for this unit should be $e2$ and the bias $e1 - e2 \times A$.

Second hidden unit

This unit must check if the sum is smaller than B, also accounting for an error given by $\frac{e3}{e4}$, which results in the inequality below.

$$\sum_{i=1}^{D} x_i + \frac{e3}{e4} \leq B \Leftrightarrow \sum_{i=1}^{D} -e4 \times x_i - e3 + e4 \times B \geq 0$$

Once more, everything is multiplied by e4, and the sign activation function returns whether the condition was met.

This way all the weights for this unit must be $-e4$ and the bias $e4 \times B - e3$.

Added error

After theorizing about the requirements of the network, there must be an actual value for each of the added variables representing possible error. An example for a possible input sequence could be:

$$1.1 \quad 0.9 \quad -1.1 \quad 1 \quad -0.9 \quad -0.9 \quad ...$$

Due to the formulation of the problem, and excluding the option to round the inputs, before they enter the network, to the integer they represent, some assumptions must be made to maintain feasibility in the solution. For example, if there were an input like this:

$$0.8 \quad 0.8 \quad 0.8 \quad 0.8 \quad 0.8 \quad -1$$

For a B = 3, the sum of this input would be 3, and the network would return 1, but the sum expected should be 4, resulting in -1. Because of this, it must be assumed that the errors are not only small, but equally distributed errors in the signal, meaning their sum will always tend to 0. Since all inputs are integers, there is an interval of 1 between all possible sum results, which means a reasonable division between results could be by the middle, therefore, a threshold of 0.5. Given that inside the interval it's irrelevant to add this constraint, it only needs to appear in the limits A and B. This way, the network must accept all values that are between this interval:

$$A - 0.5 \leq \sum_{i=1}^{D} x_i \leq B + 0.5$$

$$\sum_{i=1}^{D} 2 \times x_i + 1 - 2 \times A \geq 0 \qquad \sum_{i=1}^{D} -2 \times x_i + 1 + 2 \times B \geq 0$$

Thus, the values for e1, e2, e3 and e4 must be:

$$e1 = 1 \qquad e2 = 2 \qquad e3 = -1 \qquad e4 = 2$$

Final results

After processing the hidden units, the result should be positive, only if both units return 1. For this, the final weights must be 1 for both units, with a bias of -1. In conclusion, the suggested network is depicted in figure 13, with all the weights and biases.

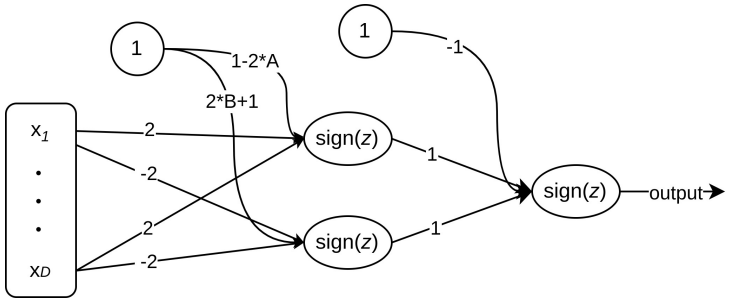

Figure 13: Representation of the network for question b).

c)

If the weights and bias between the input and the hidden units are maintained, and the activation function is changed for a rectified linear unit activation, the results of each unit would represent the distance to each limit, towards the inside of the interval, if the value of the sum is positive, and 0 if that value is on the limit or outside the interval. Some examples of this can be seen in figure 14.

To verify if a number is inside the interval, the sum of these distances to the limits must be equal to the distance between the limits. Since all negative values are turned to 0 by the activation function, if this sum of distances is bigger than the interval, the sum is either above or below the interval. This way, and assuming a sign activation function at the end, the weights and bias must result in a 0 if the sum is the correct value, and negative values if it is outside the interval. This is explained by the inequality below, given that $u1$ is the result of the first unit, and $u2$ is the result of the second.

$$u1 + u2 \leq (B + 0.5) - (A - 0.5) \Leftrightarrow B - A + 1 - u1 - u2 \geq 0$$

The result on the left side of the final inequality could never be greater than 0, only equal to or less than, as explained before. This results in the weights and bias for the connections between the hidden layer and the output. Finally, the proposed network is pictured in figure 15.

As the neural network in question b), this one is also robust to infinitesimal perturbation of the inputs, since the weights and biases that enforced this requirement were not changed.

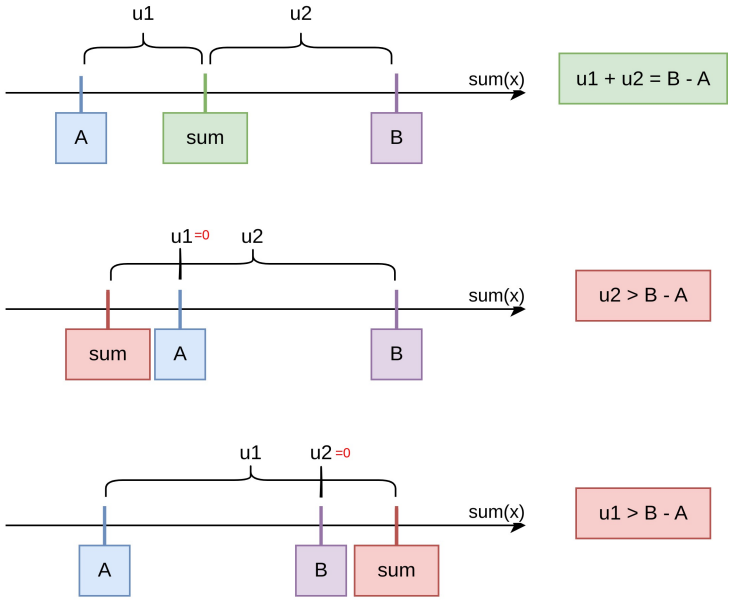

Figure 14: Examples of the results of the hidden units for different values of the sum.

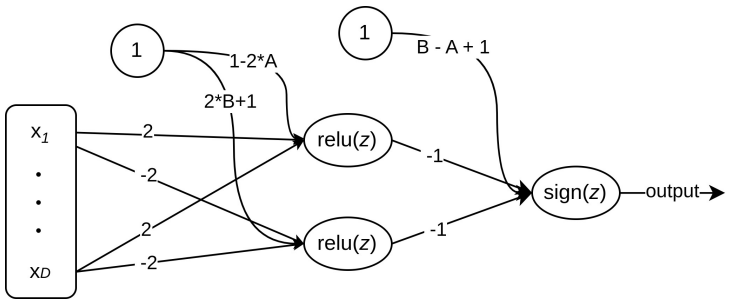

Figure 15: Representation of the network for question c).