# Simple neural network in C

Omar El Nahhas, 214316IV

## I. Introduction

The goal of this assignment is to develop a simple neural network in C to perform binary classification of human satisfaction based on temperature, humidity and air quality. All functions have been created as a result from Lab 1-4 and are adapted to this homework assignment.

## II. Methodology

The neural network layer architecture was predefined as having two hidden layers with 5 and 4 nodes respectively, see Fig. 1.

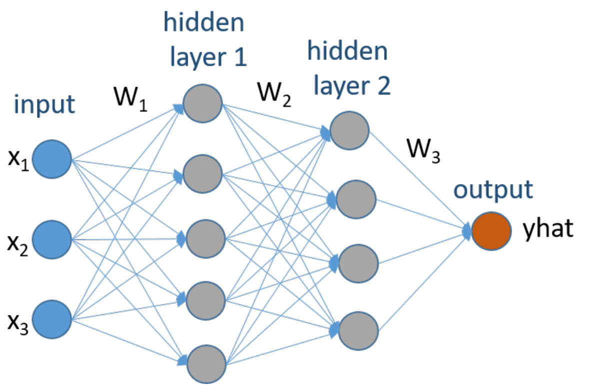

Fig. 1: The proposed 3-5-4-1 neural network architecture.

A 3D data set was created to execute the experiments, containing 40 data samples with equally divided classes (20/20) within the following ranges:

- **Temperature**: $-70 \leq x_1 \leq 60$
- **Humidity**: $0 \leq x_2 \leq 100$
- **Air quality**: $0 \leq x_3 \leq 500$

### A. Data preparation

Before feeding the data into the neural network, the data should be prepared. The three data features mentioned above all have a separate range: the maximum temperature is relatively small compared to the maximum air quality. Also, the temperature range goes below zero, whereas the other features have a minimum of 0. Therefore, the data should be normalized per feature, scaling every feature between 0 and 1. The normalization for every feature separately and the output is done using Eq. 1, with $i$ denoting the feature number and $j$ the sample number.

$$N(x_{i,j}) = \frac{x_{i,j} - min(x_i)}{max(x_i) - min(x_i)} \tag{1}$$

### B. Forward propagation

After normalizing the data, every data sample is propagated forward through the neural network to calculate parameters $z1, z2, z3, a1, a2$ and $\hat{y}$. The nodes in the hidden layers use Leaky ReLu (instead of the prescribed ReLu) and the output layer uses Sigmoid as activation function. Leaky ReLu is used to prevent vanishing gradients, see Eq. 2. The activation functions allow for non-linearity to be introduced in the neural network.

$$f(x) = \begin{cases} 0.1x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2}$$

### C. Backward propagation

After having propagated all 40 data samples forward through the network and obtaining $z1, z2, z3, a1, a2$ and $\hat{y}$, backward propagation is implemented to calculate parameters $\delta w1, \delta w2, \delta w3, \delta b1, \delta b2$ and $\delta b3$ with respect to the to-be-minimized cost function, which is defined in Eq. 3. A margin $\epsilon$ is added to $\hat{y}$ to calculate the natural log, to avoid undefined behaviour when $\hat{y}$ is equal to precisely 0 or 1.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^i \ln(\hat{y}^i + \epsilon) + (1 - y^i)\ln(1 - \hat{y}^i + \epsilon) \tag{3}$$

### D. Validation

After performing steps B and C sequentially multiple times in a row (i.e. epochs), the model's performance can be validated. The metrics used to determine performance within this assignment are:

- **Cost**: The steady decrease of cost to 0 over time (Eq. 3).
- **Training accuracy**: The number of correctly classified samples from the training data.
- **Validation accuracy**: The number of correctly classified samples from the unseen validation data.

Sometimes, the cost function can approach zero, while the model does not get better at its classification job. Then, an alternative to measure the internal working of the model is Gradient Checking, which is numerically checking the derivatives computed for debugging reasons. The implementation of this function is seen in Eq. 4, with $\epsilon$ indicating the step size of the gradient. This is not implemented in this assignment.

$$\frac{\delta}{\delta\theta}J(\theta) \approx \lim_{x \to 0} \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon} \tag{4}$$

## III. Results

Through trial and error, the best hyper-parameters were selected for the neural network. Table I shows a highlight of the combination of parameters and their results. All 40 data points are used for training and 10 new data points are used as validation of the model. Not having 100% accuracy on the training data is a good sign, meaning the model is not over-fitting on the small sample of data but is rather generalizing. This is reflected on the classification accuracy of the unseen validation data. As a result of trial and error, a learning rate of 0.100 with 16 epochs yields a classification accuracy of 80%. The downside of using these parameters is that it turns the cost function into NaN, which means that the cost computation still reaches undefined answers, despite the margins inside the natural log. This behaviour should be researched in further studies.

TABLE I: Model parameters with validation metrics.

| LR | Epochs | Cost | Training acc. | Validation acc. |
|---|---|---|---|---|
| 0.350 | 8 | 0.585 | 27/40 | 6/10 |
| 0.350 | 16 | 1.658 | 24/40 | 3/10 |
| 0.100 | 8 | 3.223 | 23/40 | 5/10 |
| 0.100 | 16 | NaN | 26/40 | 8/10 |
| 0.010 | 16 | 6.121 | 23/40 | 5/10 |
| 0.001 | 16 | NaN | 9/40 | 5/10 |

A learning rate of 0.350 and 8 epochs result in a cost approaching zero, while having an acceptable accuracy for training data (67.5%) and validation data (60%). The high learning rate and the low amount of epochs are rather unusual, but do however yield the best results within the scope of the performed experiments. Using the same learning rate, but performing 16 epochs, it can be observed that the loss is higher and the performance of the model is worse on both the training and validation data. Therefore, it can be concluded that those parameters cause the model to go near a local/global minimum, but overshoot it on following epochs because of the large value of the learning rate.

When altering the architecture of the network, e.g. changing the hidden layers from {5,4} to {3,3}, the network has a higher accuracy on the training data, but does not see an increase in the validation accuracy. Finding the optimal network architecture is out of scope for this assignment.

## IV. Issues

Besides some slight issues of passing an integer where a double should have been put and wrongly normalizing the data, the biggest confusion/issue that arose was regarding the dimensions of matrices. Following the formulas from the lecture slides, a bug was fixed last week regarding the function "Linear_backward". The newly given formula is Eq. 5.

$$\delta W = \frac{1}{m} * \delta Z * A_{prev}^T \tag{5}$$

$$(SAMPLESxHID2)(SAMPLESxHID1)^T \tag{6}$$

$\delta W_2$ has the dimension of HID2 x HID1, resulting in a $4x5$ matrix. $\delta Z_2$ has the dimensions of SAMPLES x HID2 resulting in a $40x4$ matrix, and $A_1$ has the dimensions of SAMPLES x HID1 resulting in a $40x5$ matrix. However, this leads to an impossible matrix multiplication seen in Eq. 6, which cannot lead to the desired dimensions of $\delta W_2$. As a result, instead of using the newly given formula, the formula in Eq. 7 was used for the "Linear_backward" function.

$$\delta W = \frac{1}{m} * \delta Z^T * A_{prev} \tag{7}$$

$$(SAMPLESxHID2)^T(SAMPLESxHID1) \tag{8}$$

Using Eq. 7 results in the matrix multiplication dimensions shown in Eq. 8, now equal to the desired dimensions of $\delta W_2$: HID2 x HID1. Although the matrix dimensions are now correct as a result of altering the formula, it does change the calculations of $\delta W$ and can heavily influence the working of the backward propagation step of the neural network.