Report

Data Analysis with R

3th assignment — Neural Networks

People in charge:Dr.Navet, Mr.Mai Long

Student: Pedro Gomes, 017066611B

Overview

The point of the assignment was to find the best accuracy possible to know if some car network configuration would be feasible or not using neural networks, and avoiding running computational intensive algorithms.

The idea is to used the best activation and optimizations functions in the neural network to output the best accuracy. Hence, this assignment does not have many constraints, and one is free to try whatever fits best.

Information Gathering

Apart from the resources of the lecture, I made some additional reading to better understand when exactly to use each activation/optimization functions and have a closer look at the keras library:

- Choosing the right activation function: [0]
- Activation Functions in NN: [1]
- Getting started with Keras sequential model: [2]
- Types of optimization functions used in neural networks and ways to optimize gradient descent: [3]

The decisions I made about the models I chose are based on the readings above, and the lecture slides.

Setup and Notes

Before getting started, it is worth noticing that I faced some issues while trying to set the seed(0) on the tensorflow library, with the following line of code:

use_session_with_seed(0)

Error:

```
Error in py_get_attr_impl(x, name, silent) :

AttributeError: module 'tensorflow' has no attribute 'reset_default_graph'
```

As I was not the only one facing this issue Mr.Mai helped us and provided us an alternative code to accomplish the same results:

Solution

set.seed(0)

tensorflow::tf$random$set_seed(0)

Finally, there were also some issues while reading the data of the “data” and “labels” files with keras. The reason for this was that data read was not being converted to a matrix:

```
x_train <- mnist[0:8000,]
y_train <- labels[0:8000,]
x_test <- mnist[8001:9995,]
y_test <- labels[8001:9995,]
#additional code to convert data to matrices
x_train <- data.matrix(x_train)
y_train <- data.matrix(y_train)
x_test <- data.matrix(x_test)
y_test <- data.matrix(y_test)
```

Remarks:

In the R code you will find several commented models. Each model is going to have a title that matches my observations on the very same title on this report. For instance:

In the R code:

```
#SOFTMAX -- LOSS:'categorical_crossentropy' -- Optimizer: AdaDelta

...Some code for model comes here...
```

Means that you will find a title “**SOFTMAX -- LOSS:'categorical_crossentropy' -- Optimizer: AdaDelta**” in this report with the details of it. In order to easily uncomment some model select all the code about that model and click: Ctrl+Shift+c

Finally, please, in order to **re-run some model, or re-run some model with different configurations**, swipe all the variables from the last model with and re-run the whole script as a fresh run. I noticed abnormal behavior when not doing so(just re-running the model). If this does not apply in your system, ignore it.

Experiments

1. RELU -- LOSS:'categorical_crossentropy' -- Optimizer: AdaDelta

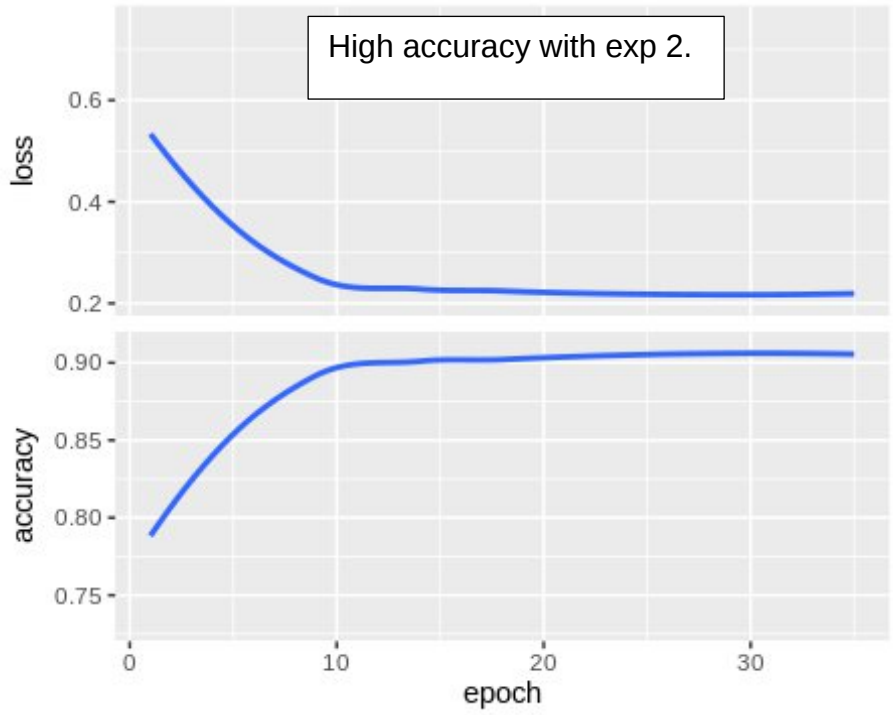
According to [0] the rule of thumb in choosing the right activation function if we are not sure which one to use is to choose the “relu” function. I had of course more information in which activation function to use, but I just wanted to give it a shot with the relu and go small at first.

According to the official keras documentation [2], the loss: “categorical_crossentropy” is good when we have **several features**(different classes, for instance: color, shape, height of some fruit). This is the case in our dataset, since we have: “nb.of.audio.flows; nb.of.video.flows; ...”. However, the final output is a **binary one**: Either the network is feasible or it is not. Hence, it would make sense to use “binary_crossentropy” at least once, I took this in consideration for the next tries.

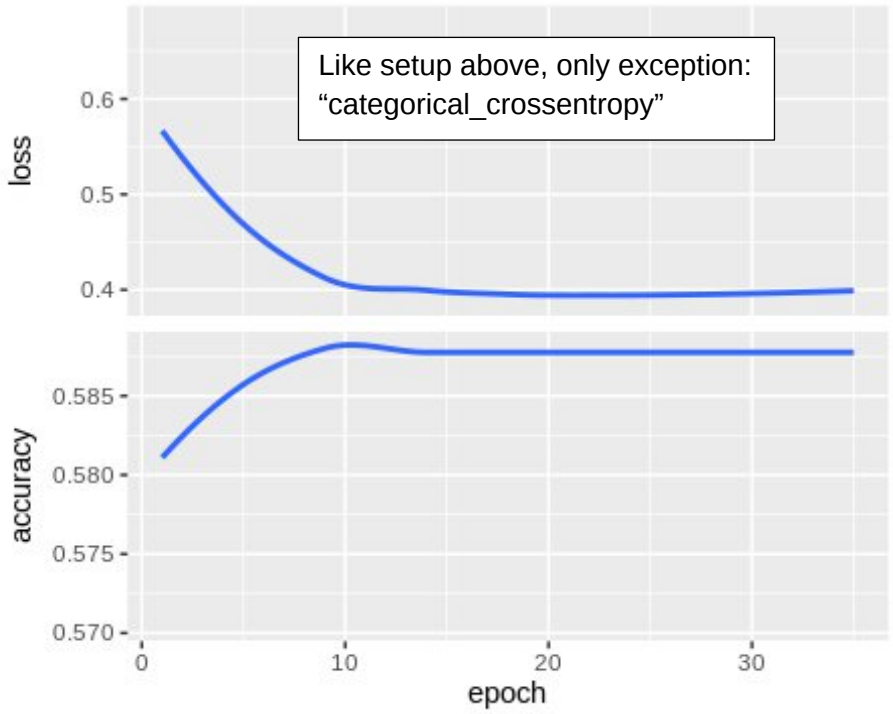
We can see that the results are not very satisfying. Furthermore the loss does not even appear. I doubt that the accuracy would stay that stable all over the 35 epochs. I tried to run this experiment a couple of times, with the same results. My setup seems to be correct, I do not think this is an error because of a setup fault. Thus, I decided to try another experiment.

2. Sigmoid-- LOSS:”binary_crossentropy' -- Optimizer: Adam

According to [0, 1] the sigmoid function is adapted to situations where we have a binary output, I wanted to give it a try. Also my optimization function could have been better. According to [3] the adam function: “*...works well in practice and outperforms other Adaptive techniques.*” . Adaptive techniques are good if there features to classify that are very sparse from each other (sparse data). Trying both adam and adadelta I was able to see that adam outputs better accuracy results by more than 25%! The adam had a learning rate of 0.05 (see the R code for more details).

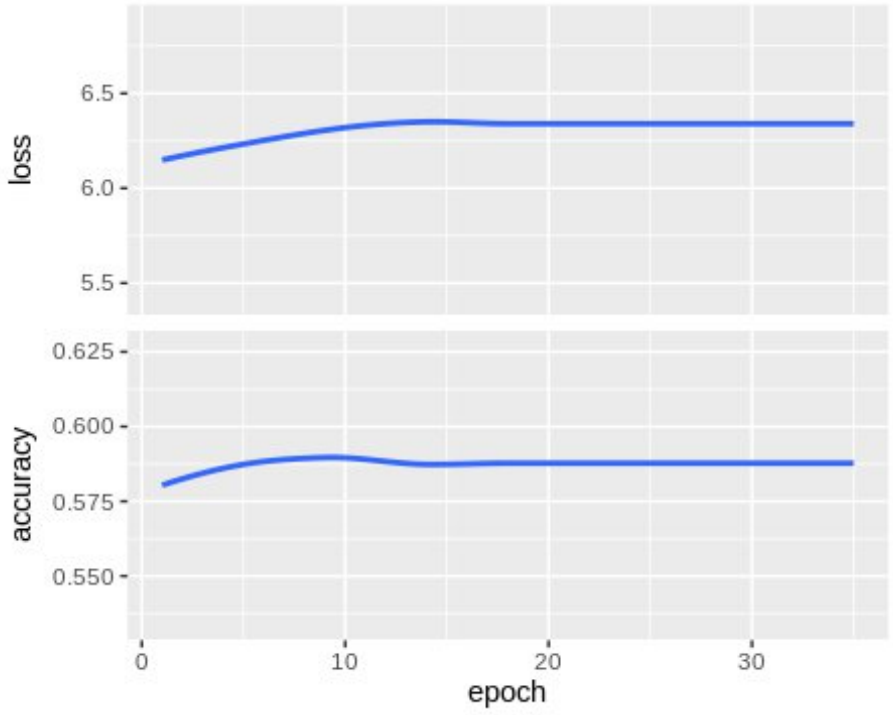


It is worth mentioning that with a fresh run(all variables swopped with graphics as well), and with **loss as “categorical_crossentropy”** I had poorer results, in the scale of minus 25%-30% of accuracy.

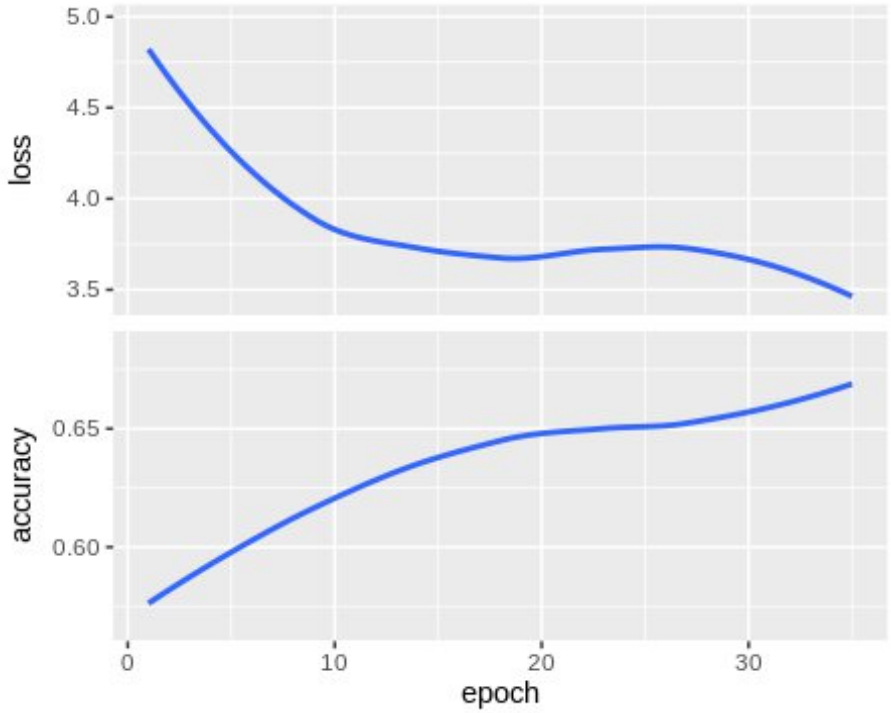


3. Tanh -- LOSS:'categorical_crossentropy' -- Optimizer: Adam

According to [0, 1] the tanh function should in theory perform better than the sigmoid function, mainly because: “... *inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph”* . Even if there are no negative values in our dataset, I wanted to give it a try.

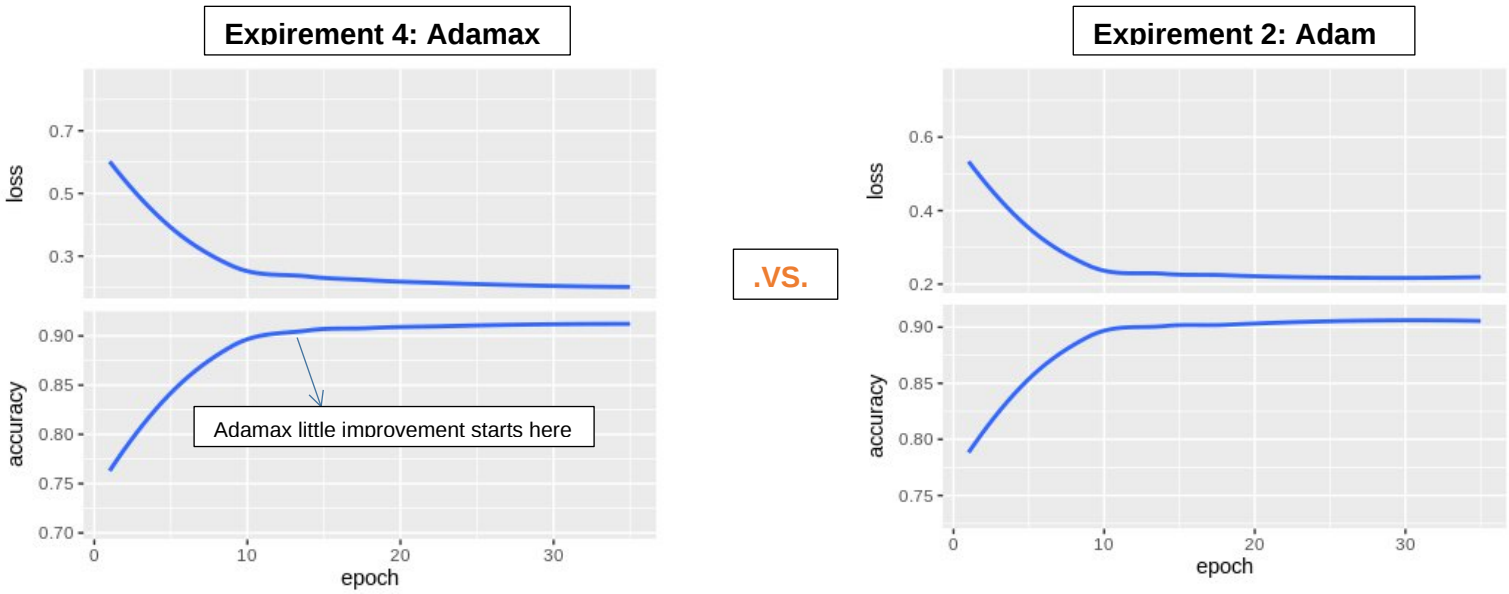


Unfortunately, the results are not very satisfying comparing with the sigmoid function. **After using the optimizer function: "*Adamax*" instead of "*Adam*"** I was able to notice an improvement in both accuracy and loss:



It is true that we have less linear results, but they are more accurate. Giving the fact that the Adamax function kinda improvement my results here, I wanted to try it with the experiment: **2. Sigmoid-- LOSS:"binary_crossentropy' -- Optimizer: Adam**, since I already had a high accuracy I wanted to get a bit closer to 100%.

4. Sigmoid-- LOSS:"binary_crossentropy' -- Optimizer: Adamax



If we have a closer look at the line of the accuracy, we can see that there is a very little improvement with Adamax. The loss is a bit higher, but not significant enough to not consider using the Adamax optimization function.

I tried to normalize the input data for train sets, with the following function:

##########################Code Block################################

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
mnist <- as.data.frame(lapply(mnist, normalize))
```

##

But the accuracy did not change.

Final Notes:

I tried more than 9 different experiments, some of theme with different configurations. Only the 4 above made sense to put in the report and in the R code. All the others provided very low accuracy or resulted in errors.

After my experiments, **I conclude** that the: **4. Sigmoid-- LOSS:”binary_crossentropy' -- Optimizer: Adamax** is the best setup to obtain a high accuracy.

References

[0] : https://www.geeksforgeeks.org/activation-functions-neural-networks/

[1] : https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[2] : https://keras.io/getting-started/sequential-model-guide/

[3] : https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f