Gerard Castells and Mariona Carrasco

Deep Learning Methods for Medical Image Analysis (CM2003)

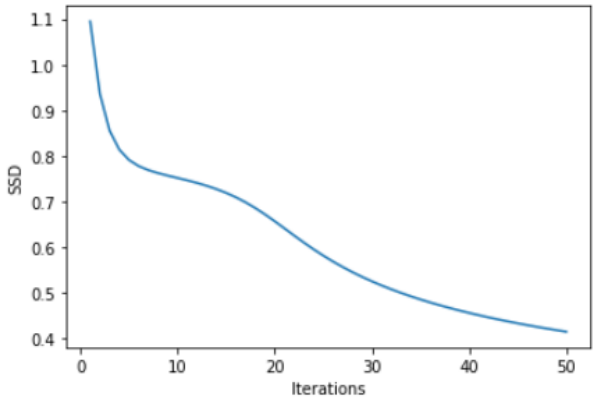
Laboratory Assignment 1

Task1

Run the above code and interpret the results. Please note the output of the model is the prediction of class labels of each of the four points. If you run the code several times, will you observe the same results? Why? Keep the parameter "n_unit=1" and increase the number of iterations starting from 10, 50, 100, 500, 2000, and compare the loss values. What can you conclude from increasing the number of iterations? Now, with a fixed value of "iterations = 1000", increase the parameter "n_unit" to 2, 5, 10 and interpret the results

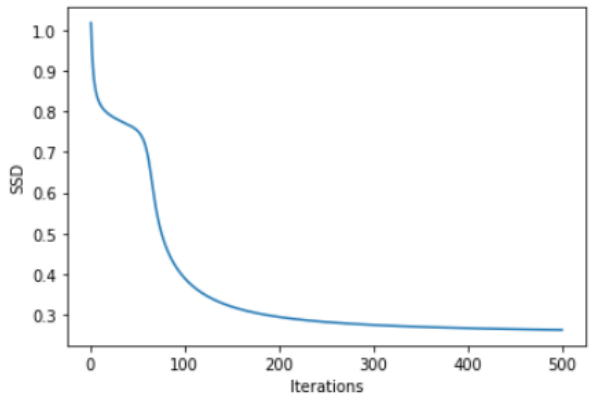
n_units=1 iterations=50

```
The target values are: [[0]
 [0]
 [0]
 [1]]
The predicted values are: [[0.06064747]
 [0.20140339]
 [0.19674166]
 [0.4249543 ]]
```



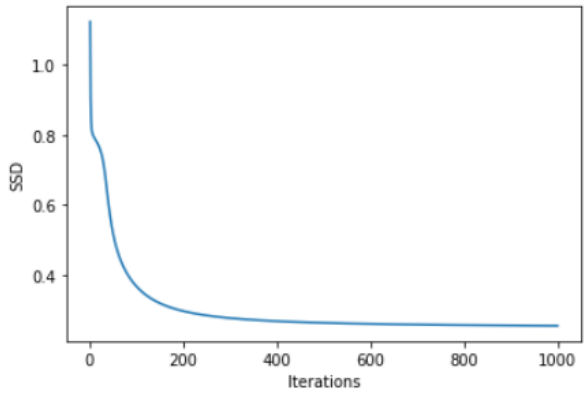
n_units=1 iterations=500

```
The target values are: [[0]
 [0]
 [0]
 [1]]
The predicted values are: [[0.00204439]
 [0.04501682]
 [0.04501682]
 [0.49116412]]
```



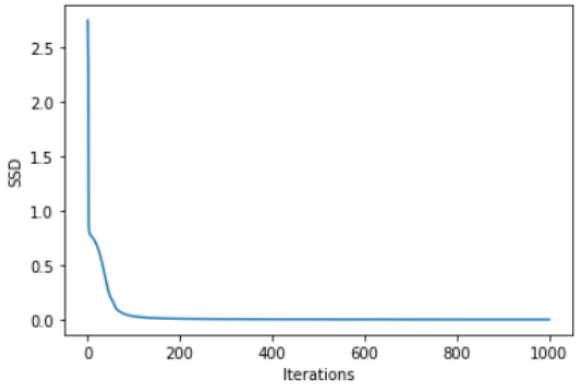
n_units=2 iterations=1000

```
The target values are: [[0]
 [0]
 [0]
 [1]]
The predicted values are: [[0.00081246]
 [0.0289807 ]
 [0.02917616]
 [0.49544832]]
```



n_units=10 iterations=1000

```
The target values are: [[0]
 [0]
 [0]
 [1]]
The predicted values are: [[1.08257063e-04]
 [1.45904350e-02]
 [1.46312860e-02]
 [9.80258155e-01]]
```



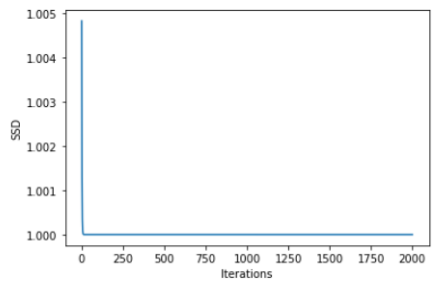
When running the code several times with the same number of iterations, the result keep changing. That's because the starting values for the weights are generated randomly. This makes that with a small number of iterations the variations are bigger. When increasing the number of iterations the loss values decrease until reaching a value around 0.2. However, by increasing the parameter "n_units" it decreases the loss value more quickly than with iterations. Then the predicted values are closer to the target values.

Task 2

Repeat task1 for XOR logic operator. For fixed values of parameters (iterations=2000, and n_unit=1), which of the AND or XOR operators has lower loss values? Why? Increase the number of neurons in the hidden layer (n_unit) to 2, 5, 10, 50. Does increasing the number of neurons improve the results? Why?

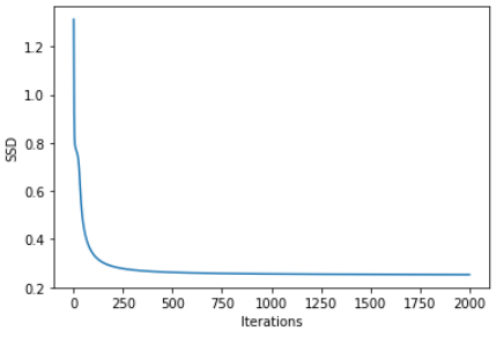
OR n_units=1 iterations=2000

```
The target values are: [[1]
 [0]
 [0]
 [1]]
The predicted values are: [[0.49978746]
 [0.4997848 ]
 [0.49976962]
 [0.49976736]]
1/1 [==============================] - 0s 42ms/step
The predicted class labels are: [[0.43264404]
 [0.484165  ]
 [0.75357354]
 [0.32289505]]
```



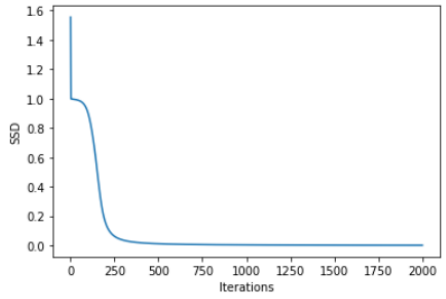
AND n_units=1 iterations=2000

```
The target values are: [[0]
 [0]
 [0]
 [1]]
The predicted values are: [[4.68545384e-04]
 [1.83896605e-02]
 [1.83896605e-02]
 [4.98115895e-01]]
```



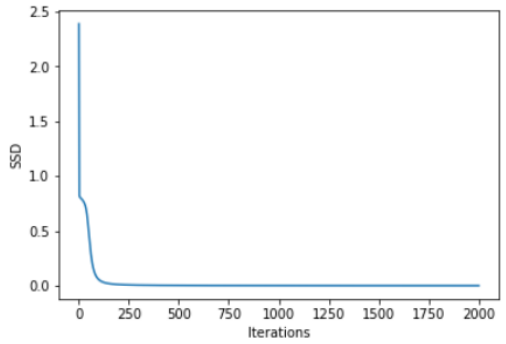
OR n_units=5 iterations=2000

```
The target values are: [[1]
 [0]
 [0]
 [1]]
The predicted values are: [[0.99167152]
 [0.01999751]
 [0.0196261 ]
 [0.97625814]]
1/1 [==============================] - 0s 38ms/step
The predicted class labels are: [[0.5280707 ]
 [0.59449804]
 [0.5428668 ]
 [0.3837045 ]]
```



AND n_units=5 iterations=2000

```
The target values are: [[0]
 [0]
 [0]
 [1]]
The predicted values are: [[6.25639646e-05]
 [9.89677384e-03]
 [9.70812569e-03]
 [9.85229485e-01]]
```



We can see that AND has lower loss values than XOR (0.99).

- AND operator is easy to learn by the neural network, because it is linearly separable and simple logical operation. It outputs a 1 only when both input values are 1 and outputs 0 for all other input combinations.
- XOR operator is more difficult to learn because it is a more complex logical operation and not linearly separable. It outputs a 1 only when exactly one of the input values is 1 and outputs 0 for all other input combinations. So,it requires a more complex representation.

On the other hand, when increasing the number of neurons in the hidden layer, we can see that the results improve. Obtaining outputs closer to the target and also reducing the loss value.

Task 3

In the above code, change the parameter "n_unit" as 1, 10 and interpret the observed results.

Using Tensorflow code is faster (0,35ms/step) and more accurate to the target values. In addition, tensorflow and numpy obtain quite similar results.

Task 4

Review the following data loader code and find out how it works. Run it to load the training and test data.

```
Reading: 0/1000 of train images
Reading: 100/1000 of train images
Reading: 200/1000 of train images
Reading: 300/1000 of train images
Reading: 400/1000 of train images
Reading: 500/1000 of train images
Reading: 600/1000 of train images
Reading: 700/1000 of train images
Reading: 800/1000 of train images
Reading: 900/1000 of train images
Reading: 0/200 of train images
Reading: 100/200 of train images
```

Task 5

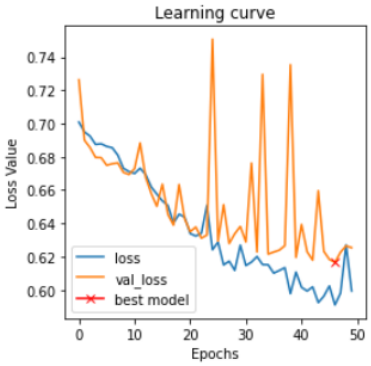
Develop a 4-layers MLP. If you call the number of neurons in the first fully-connected layer as "base_dense", this 4-layers MLP should contain "base_dense", "base_dense//2", and "base_dense//4" as the number of neurons in the first 3 layers respectively. The activation function of all those neurons should be set as "Relu". However, in the last layer (4th layer), choose a proper number of neurons as well as activation function(s) that fit the binary classification task. Develop your model as a function, and remember to, first, import all the required layers/tools from tensorflow.

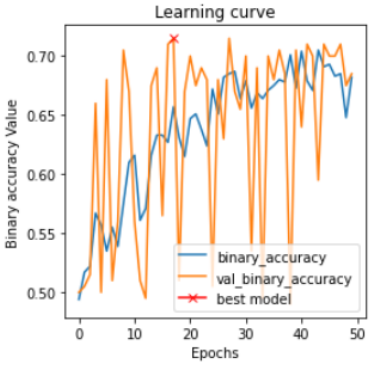
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 128, 128, 1)]     0

 flatten_31 (Flatten)        (None, 16384)             0

 dense_135 (Dense)           (None, 64)                1048640

 dense_136 (Dense)           (None, 32)                2080

 dense_137 (Dense)           (None, 16)                528

 dense_138 (Dense)           (None, 1)                 17

=================================================================
Total params: 1,051,265
Trainable params: 1,051,265
Non-trainable params: 0
```

```
Epoch 45/50
63/63 [==============================] - 0s 3ms/step - loss: 0.6097 - binary_accuracy: 0.6700 - val_loss: 0.6095 - val_binar
y_accuracy: 0.6750
Epoch 46/50
63/63 [==============================] - 0s 3ms/step - loss: 0.5865 - binary_accuracy: 0.6970 - val_loss: 0.6052 - val_binar
y_accuracy: 0.7200
Epoch 47/50
63/63 [==============================] - 0s 3ms/step - loss: 0.5821 - binary_accuracy: 0.6940 - val_loss: 0.6037 - val_binar
y_accuracy: 0.6950
Epoch 48/50
63/63 [==============================] - 0s 3ms/step - loss: 0.5807 - binary_accuracy: 0.7090 - val_loss: 0.6030 - val_binar
y_accuracy: 0.7100
Epoch 49/50
63/63 [==============================] - 0s 4ms/step - loss: 0.5819 - binary_accuracy: 0.6980 - val_loss: 0.6348 - val_binar
y_accuracy: 0.6450
Epoch 50/50
63/63 [==============================] - 0s 4ms/step - loss: 0.5914 - binary_accuracy: 0.6800 - val_loss: 0.6046 - val_binar
y_accuracy: 0.7150
```

Task 6

The values of loss and accuracy metrics are saved within the variable "clf_hist". You can use the following code to visualize the loss curves:



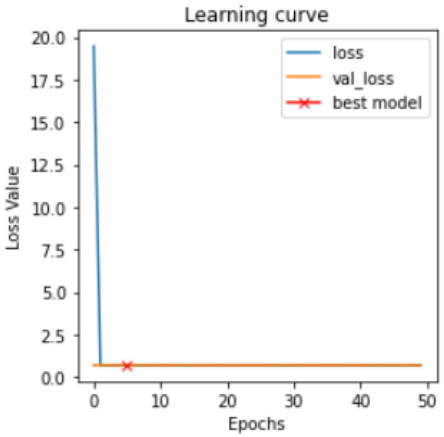


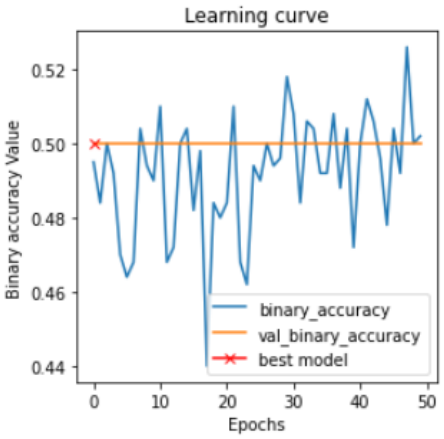
How do you interpret the observed values of loss and accuracy values?

We can see that by increasing the number of epochs, the loss value decreases and the accuracy increases. That's because every epoch means a complete pass through the entire training dataset. In other words, it's the number of times the model has seen and learned from the entire training dataset. So as this number increases the error is less, and accuracy is higher.

Is the number of epochs enough to make a good decision about model performance?

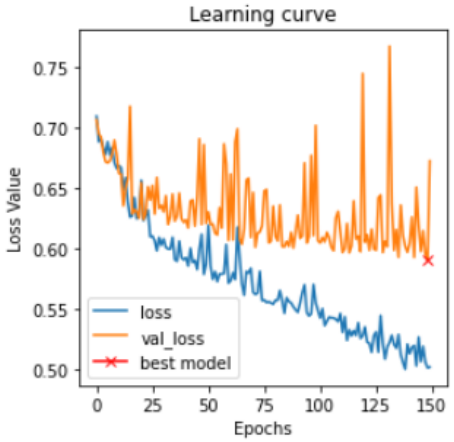
We consider that it's not good enough as long as the oscillation of the loss and accuracy values is still reasonably big. In order to make a good decision about model performance we should increase the number of epochs in order to obtain stable values.





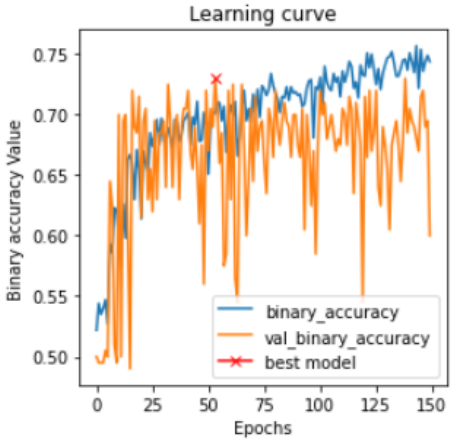
For the same number of epochs, reduce the learning rate parameter to 0.1 and interpret the results.

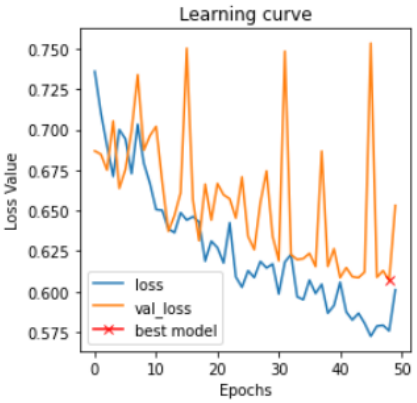
Reducing the learning rate to 0.1 makes the values of loss and accuracy values become stable with less number of epochs, but in a worse value. The loss value gets stable at a 0.69, which is higher than with the previous learning rate, and the binary accuracy value gets stable at 0.5, which is less than before. The fact that using a higher learning rate value produces worse performance is because the size of the steps is bigger, which leads to overshooting the optimal solution.



Now increase the number of epochs to 150 with LR=0.0001. Does this model have enough capacity to yield acceptable results?

The model has not enough capacity to yield acceptable results because the loss and accuracy values are still oscillating so it has not enough stability to accept it.





Increase the "base_dense" parameter to 256 and compare the results with the case of "base_dense=64". Is increasing the model capacity helpful to improve the model performance? Why?

Increasing the model capacity improves the model performance because we can see higher binary accuracy and lower loss values. So it is better to increase the "base_dense" to have better results.