Curiosity project - Group 06 - Heart Disease

Submitters:
Tal Carmi, ID: 039161203
Anna Mosenzon, ID: 200320836
Rami Skolozub, ID: 316736396

Problem description:
A classification problem to distinguish between presence or absence of heart disease based on dataset contains 13 features and 1 Label column (there is a heart disease or there is no heart disease.
The target variable: Heart Disease – Yes\No distributes as follows:
54% of the observations have heart disease and 46% have no heart disease.

Part II:

4. Problem formulation with Neural Network:
   a. The Neural Network structure: the NN build from 4 layers – 1 input layers, 2 hidden layers and 1 output layer. For the hidden layer we used ReLU to add the non-linear activation function and for the output layer we used Sigmoid function. The layers of our baseline model are built from:
      i. 13 in-features (input neurons) and 5 out-features (output neurons) in addition to bias neuron.
      ii. ReLU activation function
      iii. 5 in-features (input neurons) and 5 out-features (output neurons) in addition to bias neuron.
      iv. ReLU activation function
      v. 5 in-features (input neurons) and 1 out-features (output neurons) in addition to bias neuron.
      vi. Sigmoid activation function

We used ADAM optimizer which works on the principal of gradient decent but combines also both momentum and bias correction, and the following parameters (additional information about ADAM optimizer is in the appendix):

- Number of neurons in the hidden layers = 5
- Learning rate = 0.01
- Number of iterations = 200

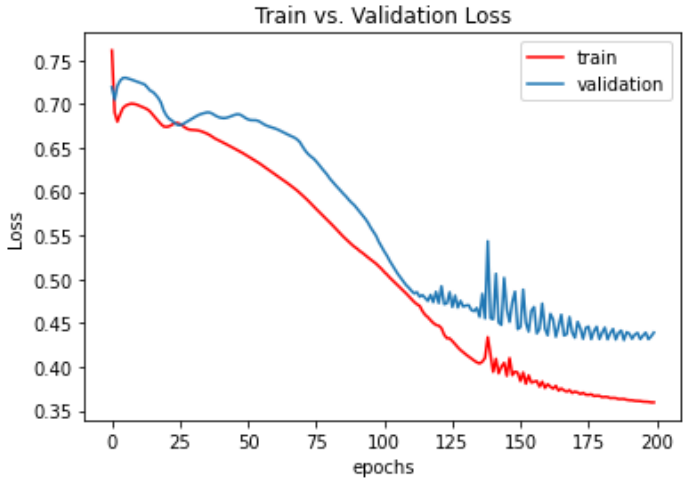
   b. After we defined a class with our NN structure, we split our data into train-validation-test datasets (80-10-10%). The 1st phase of the training occurs using forward propagation of the training data across our network. When the representation of the features of the data gets to the final layer we compare the prediction we get from the output layer to the true labels we have for the training data and we calculate the loss using our loss function.
   The loss function (learning algorithm) we used is BCE - Binary cross Entropy loss, since we are solving a classification problem and this loss function is common for this kind of problems. The loss function indicates how far was our prediction result in compare to the true label.
   Once we calculated the loss, we propagate the information backwards from the output layer to all neurons in the hidden layers, the neurons of the hidden layer only receive a fraction of the total signal of the loss, based on the relative contribution that each neuron has contributed to the original output.
   Once we propagate the information of the loss backward to our 1st layer we adjust

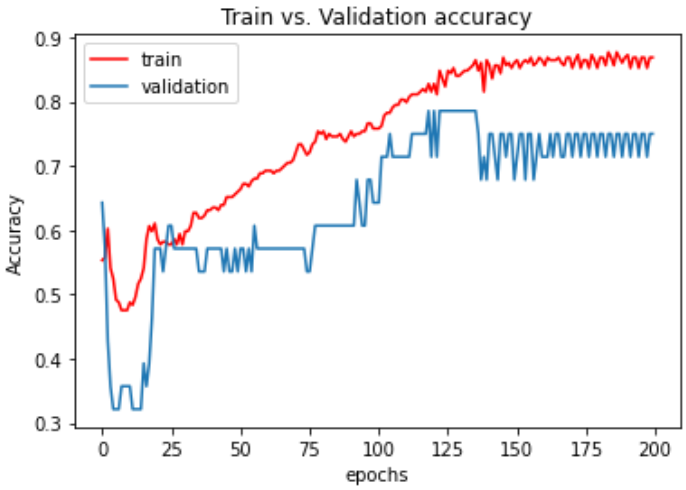
our weights using the optimizer, ADAM – which is an improved and more sophisticated version of gradient decent. The optimizer changes the weights in small increments with the help of the calculation of the gradients of the loss function, which allows us to see in which direction "to descend" towards the global minimum. – We are doing it for each epoch. In each epoch post the updates of the weights we are expecting to see improvement or Learning, as a result, our loss function output values should be decreased.

5. Problem programming and the solution:
   Attached Jupiter notebook with the full code.
   We built a class of our neural network as described in section 4.

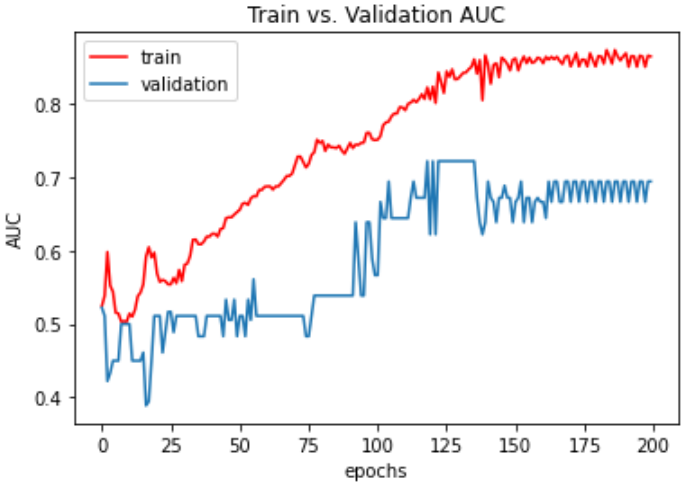
```
Sequential(
  (0): Linear(in_features=13, out_features=5, bias=True)
  (1): ReLU()
  (2): Linear(in_features=5, out_features=5, bias=True)
  (3): ReLU()
  (4): Linear(in_features=5, out_features=1, bias=True)
  (5): Sigmoid()
)
```

The outputs we preformed to estimate our network are the loss vs. epochs – we expect the train loss to decrease over the epochs and would like to stop our training when validation loss starts to increase (overfitting). In this case we'll usually put a stopping condition to stop after 5 epochs in a row that our loss increases but in our dataset we achieved pretty good results in a reasonable amount of epochs without getting to over fitting as can be seen in the graph outputs. Since we split the train-validation-test randomly we have got each time a bit different output, but the concept stayed the same.
As for accuracy, we see the same phenomena as in the loss output but in the opposite way. At 1st post few epochs that our accuracy is getting down till the gradients stabilized a bit and the weights got less shifts, but than our accuracy was getting higher as a function of epochs. The AUC graph of course shows similar behavior of the accuracy from other POV (different metric).





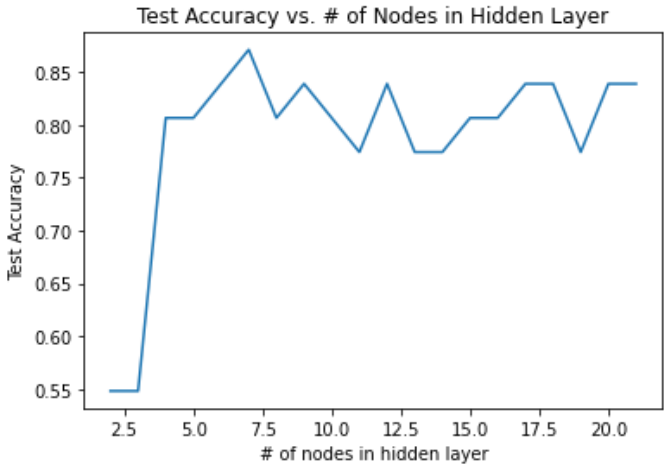
- Train Accuracy 86.89%
- Validation Accuracy 75.00%
- Test accuracy: 80.645%



- Train AUC: 87%
- Validation AUC: 69%

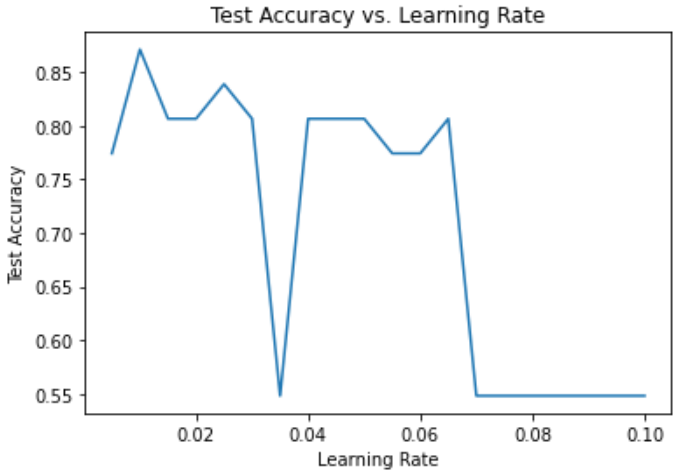
6. Parameters sensitivity analysis: we change 3 of our hyper-parameters to find the best model and we compared them using the accuracy metric:
    - Number of neurons in the hidden layers
    - Learning rate
    - Number of epochs

    Number of neurons in the hidden layers:



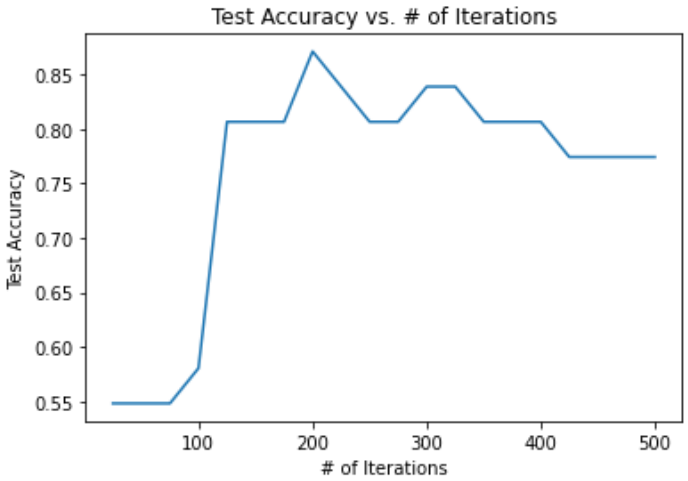
Best test accuracy with 7 nodes in hidden layers: 87.097%

    Learning rate:



Best test accuracy with 0.01 learning rate and 7 nodes in hidden layers: 87.097%

Number of iterations:

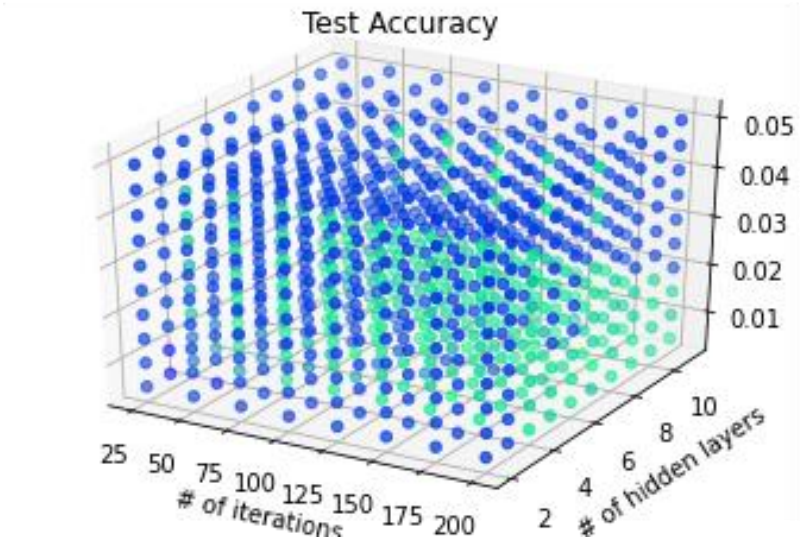


Best test accuracy with 200 iterations, 0.01 learning rate and 7 nodes in hidden layers: 87.097%

In addition to the analysis above, we ran a grid search on the three parameters checking all combination of the following values:

- Number of neurons in the hidden layers: 2-11
- Learning rate: 0.005-0.05
- Number of iterations: 25-200

In the graph below the "greener" the dot is the model achieve better accuracy.
Since it's hard to see the details in this visualization – we decided to take the models that gave us the best results and plot just them so we can better spot the differences in the hyper parameters and compare them using the metrics of AIC&BIC (section7).



The details (number of iterations, learning rate and number of nodes in the hidden layers) of each model are in the appendix.

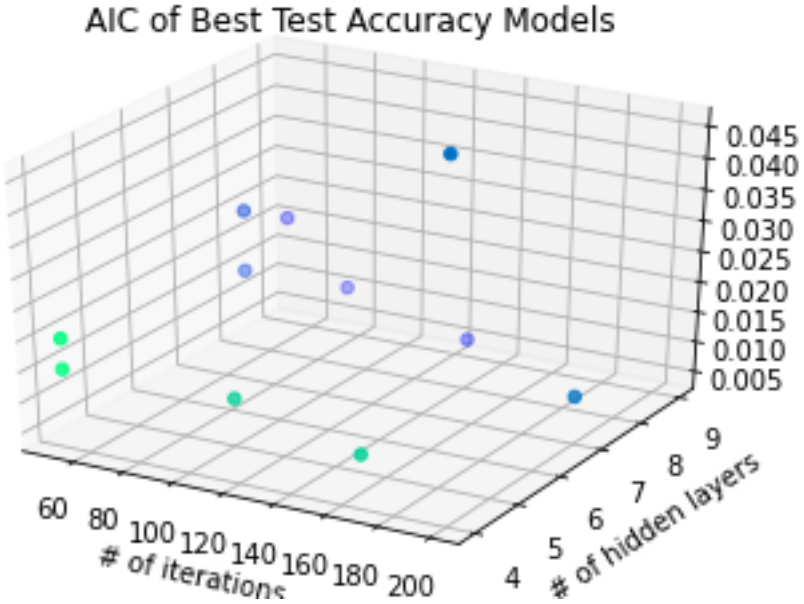
7. The best test accuracy we have got was: 87.097% and we achieved it with 11 different models. In order to determine which model is the best we used AIC and BIC measures:

$$AIC = nln(E) + 2k$$

$$BIC = nln(E) + kln(n)$$

k – number of parameters
n – number of data points
E – network error

We took the 11 models and compared in similar way, this time the "greener" the dot the better AIC measure the model achieved (the BIC results are correlated):



We can see that even for only 50 iterations, 4 nodes and 0.02 learning rate we achieve the same accuracy with more parameters and more complexity.

The best model is with the following parameters:

# of iterations	Learning rate	Nodes in hidden layers	Accuracy	AIC	BIC
50	0.02	4	87.1%	134.370	250.523

The AIC and BIC measures of each model are in the appendix.

Appendix

ADAM optimizer:

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

Taken from: Stanford CS231n

Published as a conference paper at ICLR 2015

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Grid search details:

- new best test accuracy with 25 iterations, 0.005 learning rate and 2 nodes in hidden layers: 54.839000000000006%
- finished checking 2 hiddenLayers with learning rate 0.005
- finished checking 3 hiddenLayers with learning rate 0.005
- new best test accuracy with 25 iterations, 0.005 learning rate and 4 nodes in hidden layers: 61.29%

- new best test accuracy with 50 iterations, 0.005 learning rate and 4 nodes in hidden layers: 70.968%
- new best test accuracy with 75 iterations, 0.005 learning rate and 4 nodes in hidden layers: 83.871%
- finished checking 4 hiddenLayers with learning rate 0.005
- new best test accuracy with 150 iterations, 0.005 learning rate and 5 nodes in hidden layers: 87.09700000000001%
- finished checking 5 hiddenLayers with learning rate 0.005
- finished checking 6 hiddenLayers with learning rate 0.005
- finished checking 7 hiddenLayers with learning rate 0.005
- finished checking 8 hiddenLayers with learning rate 0.005
- finished checking 9 hiddenLayers with learning rate 0.005
- finished checking 10 hiddenLayers with learning rate 0.005
- finished checking 11 hiddenLayers with learning rate 0.005
- finished checking 2 hiddenLayers with learning rate 0.01
- finished checking 3 hiddenLayers with learning rate 0.01
- finished checking 4 hiddenLayers with learning rate 0.01
- finished checking 5 hiddenLayers with learning rate 0.01
- finished checking 6 hiddenLayers with learning rate 0.01
- finished checking 7 hiddenLayers with learning rate 0.01
- finished checking 8 hiddenLayers with learning rate 0.01
- finished checking 9 hiddenLayers with learning rate 0.01
- finished checking 10 hiddenLayers with learning rate 0.01
- finished checking 11 hiddenLayers with learning rate 0.01
- finished checking 2 hiddenLayers with learning rate 0.015
- finished checking 3 hiddenLayers with learning rate 0.015
- finished checking 4 hiddenLayers with learning rate 0.015
- finished checking 5 hiddenLayers with learning rate 0.015
- finished checking 6 hiddenLayers with learning rate 0.015
- finished checking 7 hiddenLayers with learning rate 0.015
- finished checking 8 hiddenLayers with learning rate 0.015
- finished checking 9 hiddenLayers with learning rate 0.015
- finished checking 10 hiddenLayers with learning rate 0.015
- finished checking 11 hiddenLayers with learning rate 0.015
- finished checking 2 hiddenLayers with learning rate 0.02
- finished checking 3 hiddenLayers with learning rate 0.02
- finished checking 4 hiddenLayers with learning rate 0.02
- finished checking 5 hiddenLayers with learning rate 0.02
- finished checking 6 hiddenLayers with learning rate 0.02
- finished checking 7 hiddenLayers with learning rate 0.02
- finished checking 8 hiddenLayers with learning rate 0.02
- finished checking 9 hiddenLayers with learning rate 0.02
- finished checking 10 hiddenLayers with learning rate 0.02
- finished checking 11 hiddenLayers with learning rate 0.02
- finished checking 2 hiddenLayers with learning rate 0.025
- finished checking 3 hiddenLayers with learning rate 0.025
- finished checking 4 hiddenLayers with learning rate 0.025
- finished checking 5 hiddenLayers with learning rate 0.025
- finished checking 6 hiddenLayers with learning rate 0.025

- finished checking 7 hiddenLayers with learning rate 0.025
- finished checking 8 hiddenLayers with learning rate 0.025
- finished checking 9 hiddenLayers with learning rate 0.025
- finished checking 10 hiddenLayers with learning rate 0.025
- finished checking 11 hiddenLayers with learning rate 0.025
- finished checking 2 hiddenLayers with learning rate 0.030000000000000002
- finished checking 3 hiddenLayers with learning rate 0.030000000000000002
- finished checking 4 hiddenLayers with learning rate 0.030000000000000002
- finished checking 5 hiddenLayers with learning rate 0.030000000000000002
- finished checking 6 hiddenLayers with learning rate 0.030000000000000002
- finished checking 7 hiddenLayers with learning rate 0.030000000000000002
- finished checking 8 hiddenLayers with learning rate 0.030000000000000002
- finished checking 9 hiddenLayers with learning rate 0.030000000000000002
- finished checking 10 hiddenLayers with learning rate 0.030000000000000002
- finished checking 11 hiddenLayers with learning rate 0.030000000000000002
- finished checking 2 hiddenLayers with learning rate 0.034999999999999996
- finished checking 3 hiddenLayers with learning rate 0.034999999999999996
- finished checking 4 hiddenLayers with learning rate 0.034999999999999996
- finished checking 5 hiddenLayers with learning rate 0.034999999999999996
- finished checking 6 hiddenLayers with learning rate 0.034999999999999996
- finished checking 7 hiddenLayers with learning rate 0.034999999999999996
- finished checking 8 hiddenLayers with learning rate 0.034999999999999996
- finished checking 9 hiddenLayers with learning rate 0.034999999999999996
- finished checking 10 hiddenLayers with learning rate 0.034999999999999996
- finished checking 11 hiddenLayers with learning rate 0.034999999999999996
- finished checking 2 hiddenLayers with learning rate 0.04
- finished checking 3 hiddenLayers with learning rate 0.04
- finished checking 4 hiddenLayers with learning rate 0.04
- finished checking 5 hiddenLayers with learning rate 0.04
- finished checking 6 hiddenLayers with learning rate 0.04
- finished checking 7 hiddenLayers with learning rate 0.04
- finished checking 8 hiddenLayers with learning rate 0.04
- finished checking 9 hiddenLayers with learning rate 0.04
- finished checking 10 hiddenLayers with learning rate 0.04
- finished checking 11 hiddenLayers with learning rate 0.04
- finished checking 2 hiddenLayers with learning rate 0.045
- finished checking 3 hiddenLayers with learning rate 0.045
- finished checking 4 hiddenLayers with learning rate 0.045
- finished checking 5 hiddenLayers with learning rate 0.045
- finished checking 6 hiddenLayers with learning rate 0.045
- finished checking 7 hiddenLayers with learning rate 0.045
- finished checking 8 hiddenLayers with learning rate 0.045
- finished checking 9 hiddenLayers with learning rate 0.045
- finished checking 10 hiddenLayers with learning rate 0.045
- finished checking 11 hiddenLayers with learning rate 0.045
- finished checking 2 hiddenLayers with learning rate 0.049999999999999996
- finished checking 3 hiddenLayers with learning rate 0.049999999999999996
- finished checking 4 hiddenLayers with learning rate 0.049999999999999996
- finished checking 5 hiddenLayers with learning rate 0.049999999999999996

- finished checking 6 hiddenLayers with learning rate 0.049999999999999996
- finished checking 7 hiddenLayers with learning rate 0.049999999999999996
- finished checking 8 hiddenLayers with learning rate 0.049999999999999996
- finished checking 9 hiddenLayers with learning rate 0.049999999999999996
- finished checking 10 hiddenLayers with learning rate 0.049999999999999996
- finished checking 11 hiddenLayers with learning rate 0.049999999999999996

AIC and BIC comparison:

- 87.1% accuracy with 150 iterations, 0.005 learning rate and 5 nodes in hidden layers. Loss 0.4216243326663971, # of params: 106, aic: 185.22714233398438, bic: 337.22979736328125
- 87.1% accuracy with 125 iterations, 0.005 learning rate and 9 nodes in hidden layers. Loss 0.4613324701786041, # of params: 226, aic: 428.01727294921875, bic: 752.098388671875
- 87.1% accuracy with 100 iterations, 0.01 learning rate and 5 nodes in hidden layers. Loss 0.454110711812973, # of params: 106, aic: 187.5281524658203, bic: 339.53082275390625
- 87.1% accuracy with 200 iterations, 0.01 learning rate and 7 nodes in hidden layers. Loss 0.4339844584465027, # of params: 162, aic: 298.12286376953125, bic: 530.4287719726562
- 87.1% accuracy with 75 iterations, 0.01 learning rate and 9 nodes in hidden layers. Loss 0.452497661113739, # of params: 226, aic: 427.4178466796875, bic: 751.4989624023438
- 87.1% accuracy with 50 iterations, 0.015 learning rate and 4 nodes in hidden layers. Loss 0.42171838879585266, # of params: 81, aic: 135.2340545654297, bic: 251.3870086669922
- 87.1% accuracy with 50 iterations, 0.015 learning rate and 8 nodes in hidden layers. Loss 0.4535236060619354, # of params: 193, aic: 361.4880676269531, bic: 638.24755859375
- 87.1% accuracy with 50 iterations, 0.02 learning rate and 4 nodes in hidden layers. Loss 0.41012972593307495, # of params: 81, aic: 134.37026977539062, bic: 250.52322387695312
- 87.1% accuracy with 50 iterations, 0.02 learning rate and 9 nodes in hidden layers. Loss 0.494573712348938, # of params: 226, aic: 430.1741638183594, bic: 754.2553100585938
- 87.1% accuracy with 50 iterations, 0.025 learning rate and 8 nodes in hidden layers. Loss 0.42323973774909973, # of params: 193, aic: 359.345703125, bic: 636.105224609375
- 87.1% accuracy with 150 iterations, 0.045 learning rate and 7 nodes in hidden layers. Loss 0.46060797572135925, # of params: 162, aic: 299.96856689453125, bic: 532.2744750976562