# Neural Networks and Deep Learning – Summer Term 2019


| Team - 09 | |
|---|---|
| Name: Rajib Chandra Das<br>Matriculation Number: 1140657<br>Email: stu218517@mail.uni-kiel.de | Name: M M Mahmudul Hassan<br>Matriculation Number: 1140658<br>Email: stu218518@mail.uni-kiel.de |


## *Exercise sheet 3*

## Exercise 1 (Learning in neural networks):

### a)

- **Learning in neural networks:** In the context of neural networks, "learning" refers to specifying the organization of the network (connectivity, neuronal elements etc.) in such a way that a desired network response is achieved for a given set of input patterns (the "training set").
- **Training set:** Training set refers to input patterns and assessment of network output which indicate about success or failure.
- **Supervised Learning:** For a majority of practical neural networks we use supervised learning. Supervised learning is where we have input variables (x) and an output variable (Y) and we use an algorithm to learn the mapping function from the input to the output. The goal is to approximate the mapping function so well that when we have new input data (x) that we can predict the output variables (Y) for that data with less error.
- **Unsupervised Learning:** Unsupervised learning is where we only have input data (X) and no corresponding output variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.
- **Online (incremental) Learning:** In this learning mode, learning is done after presentation of each individual training sample to the network.
- **Offline (batch) learning:** In this learning mode, learning is done only after presentation of all training samples to the network.
- **Training Error:** The difference between the actual network output and the target output while feeding the set of training examples.
- **Generalization error:** The difference between the actual network output and the target output, based on all possible input patterns.
- **Overfitting:** Details of some training patterns are learned which are not relevant for most of the remaining patterns.
- **Underfitting:** Model / hypothesis are not detailed enough.
- **Cross-validation:** The idea of cross validation is to split the training set into two: a set of examples to train with, and a validation set. The agent trains using the new training set. Prediction on the validation set is used to determine which model to use.

b) Prevent Overfitting:

i. **Cross-validation:** Cross-validation is a powerful preventative measure against overfitting. We have to use our initial training data to generate multiple mini train-test splits. Then we have to use these splits to tune our model. In standard k-fold cross-validation, we partition the data into k subsets, called folds. Then, we iteratively train the algorithm on k-1 folds while using the remaining fold as the test set (called the "holdout fold"). Cross-validation allows us to tune hyperparameters with our original training set. This also allows us to keep our test set as a truly unseen dataset for selecting our final model.

ii. **Train with more data:** It won't work every time, but training with more data can help our algorithms detect the signal better. However if we add more noisy data, this technique won't help. That's why we should always ensure our data is clean and relevant.

iii. **Early stopping:** When we are training a learning algorithm iteratively, we can measure how well each iteration of the model performs. That's why we have to perform training until a certain number of iterations. After that point, however, the model's ability to generalize can weaken as it begins to overfit the training data. Early stopping refers stopping the training process before the learner passes that point.

iv. **Regularization:** Regularization refers to a broad range of techniques for artificially forcing our model to be simpler. The method will depend on the type of learner we are using. For example, we could prune a decision tree, use dropout on a neural network, or add a penalty parameter to the cost function in regression. Oftentimes, the regularization method is a hyperparameter as well, which means it can be tuned through cross-validation.

v. **Ensembling:** Ensembles are machine learning methods for combining predictions from multiple separate models. There are a few different methods for ensembling, but the two most common are:

- ❖ Bagging attempts to reduce the chance overfitting complex models.
  - It trains a large number of "strong" learners in parallel.
  - A strong learner is a model that's relatively unconstrained.
  - Bagging then combines all the strong learners together in order to "smooth out" their predictions.
- ❖ Boosting attempts to improve the predictive flexibility of simple models.
  - It trains a large number of "weak" learners in sequence.
  - A weak learner is a constrained model (i.e. you could limit the max depth of each decision tree).
  - Each one in the sequence focuses on learning from the mistakes of the one before it.
- ❖ Boosting then combines all the weak learners into a single strong learner.

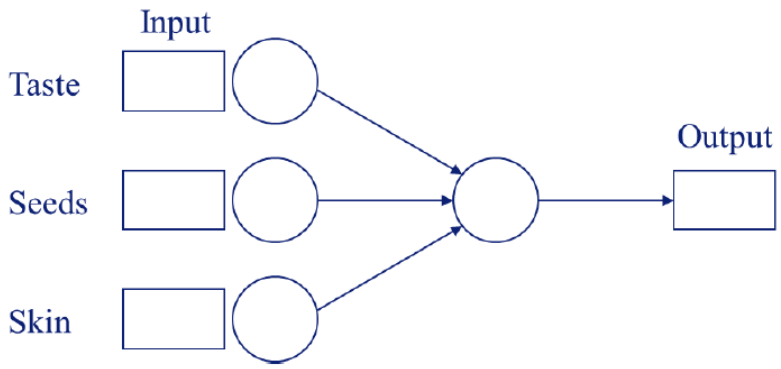
While bagging and boosting are both ensemble methods, they approach the problem from opposite directions. Bagging uses complex base models and tries to "smooth out" their predictions, while boosting uses simple base models and tries to "boost" their aggregate complexity.

## **Exercise 2** (Perceptron learning – analytical calculation):

The following table for the inputs and the target output of the perceptron:

| Fruit | Perceptron input (Features of Fruit) | | | |
|---|---|---|---|---|
| | Sweet = 1<br>Not Sweet = 0 | Seeds edible = 1<br>Not edible = 0 | Skin edible = 1<br>Not edible = 0 | Target Output:<br>Person likes = 1<br>Doesn't like = 0 |
| Banana | 1 | 1 | 0 | 1 |
| Pear | 1 | 0 | 1 | 1 |
| Lemon | 0 | 0 | 0 | 0 |
| Strawberry | 1 | 1 | 1 | 1 |
| Green Apple | 0 | 0 | 1 | 0 |

There are three (binary) input values (taste, seeds and skin) and one (binary) target output, we are given the architecture of single-layer perceptron with three inputs and one output.




So, we have three binary inputs x1, x2 and x3. To start the perceptron learning algorithm, we have to initialize the weights and the threshold. Since we have no prior knowledge on the solution, we will assume that all weights are 0 (w1 = w2 = w3 = 0) and that the threshold is θ = 1. We have to specify the learning rate η. We set η = 0.25.

We have to calculate the new weights and threshold after successively presenting a banana, pear, lemon, strawberry and a green apple to the network.

| Current Training Sample | | | | | | |
|---|---|---|---|---|---|---|
| Input, $x^{(\mu)}$ | Current Weights, w(t) | Network Output, $y^{(\mu)}$ | Target Output, $d^{(\mu)}$ | Learning Rate, η | Weight update, Δw(t) | New Weights, w(t+1) |
| $x_0 = 1$ | $w_0 = -1$ | | | 0.25 | | |
| $x_1 =$ | $w_1 =$ | | | | | |
| $x_2 =$ | $w_2 =$ | | | | | |
| $x_3 =$ | $w_3 =$ | | | | | |

We know, the equations for perceptron learning in incremental mode:

i. $w(t+1) = w(t) + \eta * (d^{(\mu)} - y^{(\mu)}) * x^{(\mu)}$,

ii. $\Delta w(t) = w(t+1) - w(t)$

iii. $y = \Theta[\sum_{i=1}^{3} x_i * w_i - \theta]$

or, $y = \Theta[\sum_{i=0}^{3} x_i * w_i]$ where $x0 = 1$ and $w0 = -\theta$

**Let's perform 1st Iteration for Banana:**

Target output, $d^{(1)} = 1$

Network output, $y^{(1)} = \Theta\,[1 * (-1) + 1 * 0 + 1 * 0 + 0 * 0] = \Theta\,[-1] = 0$

$w_0(1) = w_0(0) + \eta * (d^{(1)} - y^{(1)}) * x_0 = (-1) + 0.25 * (1 - 0) * 1 = -0.75$

$w_1(1) = w_1(0) + \eta * (d^{(1)} - y^{(1)}) * x_1 = 0 + 0.25 * (1 - 0) * 1 = 0.25$

$w_2(1) = w_2(0) + \eta * (d^{(1)} - y^{(1)}) * x_2 = 0 + 0.25 * (1 - 0) * 1 = 0.25$

$w_3(1) = w_2(0) + \eta * (d^{(1)} - y^{(1)}) * x_3 = 0 + 0.25 * (1 - 0) * 0 = 0$

$\Delta w_0(1) = w_0(1) - w_0(0) = -0.75 - (-1) = 0.25$

$\Delta w_1(1) = w_1(1) - w_1(0) = 0.25 - 0 = 0.25$

$\Delta w_2(1) = w_2(1) - w_2(0) = 0.25 - 0 = 0.25$

$\Delta w_3(1) = w_3(1) - w_3(0) = 0 - 0 = 0$

| Training Sample after 1st Iteration | | | | | | |
|---|---|---|---|---|---|---|
| Input, $x^{(\mu)}$ | Current Weights, w(t) | Network Output, $y^{(\mu)}$ | Target Output, $d^{(\mu)}$ | Learning Rate, η | Weight update, Δw(t) | New Weights, w(t+1) |
| $x_0 = 1$ | $w_0 = -1$ | 0 | 1 | 0.25 | 0.25 | -0.75 |
| $x_1 = 1$ | $w_1 = 0$ | | | | 0.25 | 0.25 |
| $x_2 = 1$ | $w_2 = 0$ | | | | 0.25 | 0.25 |
| $x_3 = 0$ | $w_3 = 0$ | | | | 0 | 0 |

**Let's perform 2nd Iteration for Pear:**

Target output, $d^{(2)} = 1$

Network output, $y^{(2)} = \Theta\,[1 * (-0.75) + 1 * 0.25 + 0 * 0.25 + 1 * 0] = \Theta\,[-0.50] = 0$

$w_0(2) = w_0(1) + \eta * (d^{(2)} - y^{(2)}) * x_0 = (-0.75) + 0.25 * (1 - 0) * 1 = -0.50$

$w_1(2) = w_1(1) + \eta * (d^{(2)} - y^{(2)}) * x_1 = 0.25 + 0.25 * (1 - 0) * 1 = 0.50$

$w_2(2) = w_2(1) + \eta * (d^{(2)} - y^{(2)}) * x_2 = 0.25 + 0.25 * (1 - 0) * 0 = 0.25$

$w_3(2) = w_2(1) + \eta * (d^{(2)} - y^{(2)}) * x_3 = 0 + 0.25 * (1 - 0) * 1 = 0.25$

$\Delta w_0(2) = w_0(2) - w_0(1) = -0.50 - (-0.75) = 0.25$

$\Delta w_1(2) = w_1(2) - w_1(1) = 0.50 - 0.25 = 0.25$

$\Delta w_2(2) = w_2(2) - w_2(1) = 0.25 - 0.25 = 0$

$\Delta w_3(2) = w_3(2) - w_3(1) = 0.25 - 0 = 0.25$

| Training Sample after 2nd Iteration | | | | | | |
|---|---|---|---|---|---|---|
| Input, $x^{(\mu)}$ | Current Weights, w(t) | Network Output, $y^{(\mu)}$ | Target Output, $d^{(\mu)}$ | Learning Rate, η | Weight update, Δw(t) | New Weights, w(t+1) |
| $x_0 = 1$ | $w_0 = -1$ | 0 | 1 | 0.25 | 0.25 | -0.50 |
| $x_1 = 1$ | $w_1 = 0$ | | | | 0.25 | 0.50 |
| $x_2 = 1$ | $w_2 = 0$ | | | | 0 | 0.25 |
| $x_3 = 0$ | $w_3 = 0$ | | | | 0.25 | 0.25 |

**Let's perform 3rd Iteration for Lemon:**

Target output, $d^{(3)} = 0$

Network output, $y^{(3)} = \Theta\,[1 * (-0.50) + 0 * 0.50 + 0 * 0.25 + 0 * 0.25] = \Theta\,[-0.50] = 0$

$w_0(3) = w_0(2) + \eta * (d^{(3)} - y^{(3)}) * x_0 = (-0.50) + 0.25 * (0 - 0) * 0 = -0.50$

$w_1(3) = w_1(2) + \eta * (d^{(3)} - y^{(3)}) * x_1 = 0.50 + 0.25 * (0 - 0) * 0 = 0.50$

$w_2(3) = w_2(2) + \eta * (d^{(3)} - y^{(3)}) * x_2 = 0.25 + 0.25 * (0 - 0) * 0 = 0.25$

$w_3(3) = w_2(2) + \eta * (d^{(3)} - y^{(3)}) * x_3 = 0.25 + 0.25 * (0 - 0) * 0 = 0.25$

$\Delta w_0(3) = w_0(3) - w_0(2) = -0.50 - (-0.50) = 0$

$\Delta w_1(3) = w_1(3) - w_1(2) = 0.50 - 0.50 = 0$

$\Delta w_2(3) = w_2(3) - w_2(2) = 0.25 - 0.25 = 0$

$\Delta w_3(3) = w_3(3) - w_3(2) = 0.25 - 0.25 = 0$

| Training Sample after 3rd Iteration | | | | | | |
|---|---|---|---|---|---|---|
| Input, $x^{(\mu)}$ | Current Weights, w(t) | Network Output, $y^{(\mu)}$ | Target Output, $d^{(\mu)}$ | Learning Rate, η | Weight update, Δw(t) | New Weights, w(t+1) |
| $x_0 = 1$ | $w_0 = -1$ | 0 | 0 | 0.25 | 0 | -0.50 |
| $x_1 = 1$ | $w_1 = 0$ | | | | 0 | 0.50 |
| $x_2 = 1$ | $w_2 = 0$ | | | | 0 | 0.25 |
| $x_3 = 0$ | $w_3 = 0$ | | | | 0 | 0.25 |

**Let's perform 4th Iteration for Strawberry:**

Target output, $d^{(4)} = 1$

Network output, $y^{(4)} = \Theta\,[1 * (-0.50) + 1 * 0.50 + 1 * 0.25 + 1 * 0.25] = \Theta\,[0.50] = 1$

$w_0(4) = w_0(3) + \eta * (d^{(4)} - y^{(4)}) * x_0 = (-0.50) + 0.25 * (1 - 1) * 1 = -0.50$

$w_1(4) = w_1(3) + \eta * (d^{(4)} - y^{(4)}) * x_1 = 0.50 + 0.25 * (1 - 1) * 1 = 0.50$

$w_2(4) = w_2(3) + \eta * (d^{(4)} - y^{(4)}) * x_2 = 0.25 + 0.25 * (1 - 1) * 1 = 0.25$

$w_3(4) = w_2(3) + \eta * (d^{(4)} - y^{(4)}) * x_3 = 0.25 + 0.25 * (1 - 1) * 1 = 0.25$

$\Delta w_0(4) = w_0(4) - w_0(3) = -0.50 - (-0.50) = 0$

$\Delta w_1(4) = w_1(4) - w_1(3) = 0.50 - 0.50 = 0$

$\Delta w_2(4) = w_2(4) - w_2(3) = 0.25 - 0.25 = 0$

$\Delta w_3(4) = w_3(4) - w_3(3) = 0.25 - 0.25 = 0$

| Training Sample after 4th Iteration | | | | | | |
|---|---|---|---|---|---|---|
| Input, $x^{(\mu)}$ | Current Weights, w(t) | Network Output, $y^{(\mu)}$ | Target Output, $d^{(\mu)}$ | Learning Rate, η | Weight update, Δw(t) | New Weights, w(t+1) |
| $x_0 = 1$ | $w_0 = -1$ | 1 | 1 | 0.25 | 0 | -0.50 |
| $x_1 = 1$ | $w_1 = 0$ | | | | 0 | 0.50 |
| $x_2 = 1$ | $w_2 = 0$ | | | | 0 | 0.25 |
| $x_3 = 0$ | $w_3 = 0$ | | | | 0 | 0.25 |

**Let's perform 5th Iteration for Green Apple:**

Target output, $d^{(5)} = 0$

Network output, $y^{(5)} = \Theta\,[1 * (-0.50) + 0 * 0.50 + 0 * 0.25 + 1 * 0.25] = \Theta\,[-0.25] = 0$

$w_0(5) = w_0(4) + \eta * (d^{(5)} - y^{(5)}) * x_0 = (-0.50) + 0.25 * (0 - 0) * 1 = -0.50$

$w_1(5) = w_1(4) + \eta * (d^{(5)} - y^{(5)}) * x_1 = 0.50 + 0.25 * (0 - 0) * 0 = 0.50$

$w_2(5) = w_2(4) + \eta * (d^{(5)} - y^{(5)}) * x_2 = 0.25 + 0.25 * (0 - 0) * 0 = 0.25$

$w_3(5) = w_2(4) + \eta * (d^{(5)} - y^{(5)}) * x_3 = 0.25 + 0.25 * (0 - 0) * 1 = 0.25$

$\Delta w_0(5) = w_0(5) - w_0(4) = -0.50 - (-0.50) = 0$

$\Delta w_1(5) = w_1(5) - w_1(4) = 0.50 - 0.50 = 0$

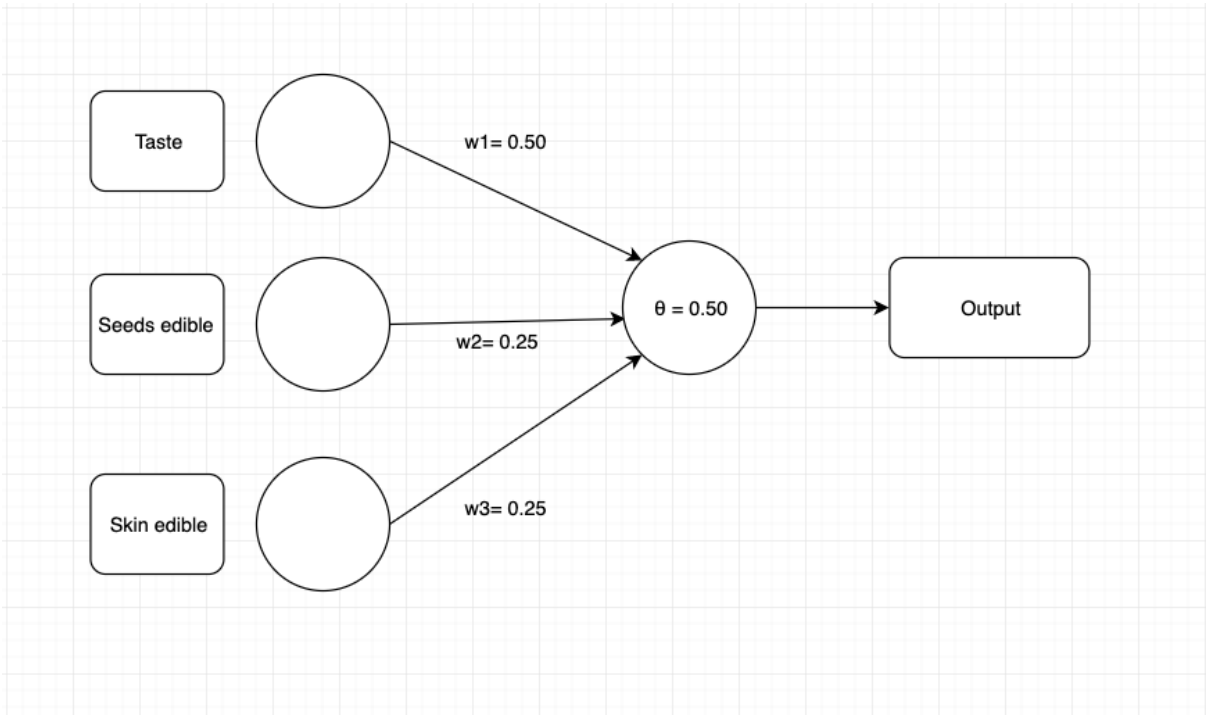
$\Delta w_2(5) = w_2(5) - w_2(4) = 0.25 - 0.25 = 0$

$\Delta w_3(5) = w_3(5) - w_3(4) = 0.25 - 0.25 = 0$

| Training Sample after 5th Iteration | | | | | | |
|---|---|---|---|---|---|---|
| Input, $x^{(\mu)}$ | Current Weights, w(t) | Network Output, $y^{(\mu)}$ | Target Output, $d^{(\mu)}$ | Learning Rate, η | Weight update, Δw(t) | New Weights, w(t+1) |
| $x_0 = 1$ | $w_0 = -1$ | 0 | 0 | 0.25 | 0 | -0.50 |
| $x_1 = 1$ | $w_1 = 0$ | | | | 0 | 0.50 |
| $x_2 = 1$ | $w_2 = 0$ | | | | 0 | 0.25 |
| $x_3 = 0$ | $w_3 = 0$ | | | | 0 | 0.25 |

After performing 5 training samples we can assume that if we design a single layer perceptron with $w_1 = 0.50$, $w_2 = 0.25$, $w_3 = 0.25$, $\theta = 0.50$ then we can get expected network output.

Finalized version of Single layer perceptron is given bellow:




## Exercise 3 (**Single-layer perceptron, gradient learning**):

a) **logistic()** activation function is given as

logistic(h) $= \frac{1}{1+e^{-h}}$

Output of the perceptron $= \Theta$ [y – 0.5]

Inputs : x1 and x2

Weights: w1 and w2

Now, postsynaptic potential, h $= x_0 * w_0 + x_1 * w_1 + x_2 * w_2$

$= w_0 + x_1 * w_1 + x_2 * w_2$ [Because, $x_0 = 1$]

We have to prove that, $x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2}$ this equation is the straight line in two dimensional space to separate the inputs.

The binary output of the perceptron $= \Theta$ [y – 0.5]

$= \Theta$ [logistic(h) – 0.5]

$= \Theta [\frac{1}{1+e^{-h}} - 0.5]$

$= \Theta [\frac{1}{1+e^{-h}} - \frac{1}{2}]$

$= \Theta [\frac{1}{1+e^{-h}} - \frac{1}{2}]$

Now, as the straight line is the boundary of two class, the parameter of the Heaviside function should be 0 in that situation.

For the boundary line we must have, Θ[0]

Therefore, $\frac{1}{1+e^{-h}} - \frac{1}{2} = 0$

⇨ $\frac{1}{1+e^{-h}} = \frac{1}{2}$

⇨ $1 + e^{-h} = 2$

⇨ $e^{-h} = 1$

⇨ -h = ln(1)

⇨ -h = 0

⇨ $-(w_0 + x_1 * w_1 + x_2 * w_2) = 0$

⇨ $x_2 * w_2 = -x1 * w_1 - w_0$

⇨ $x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}$

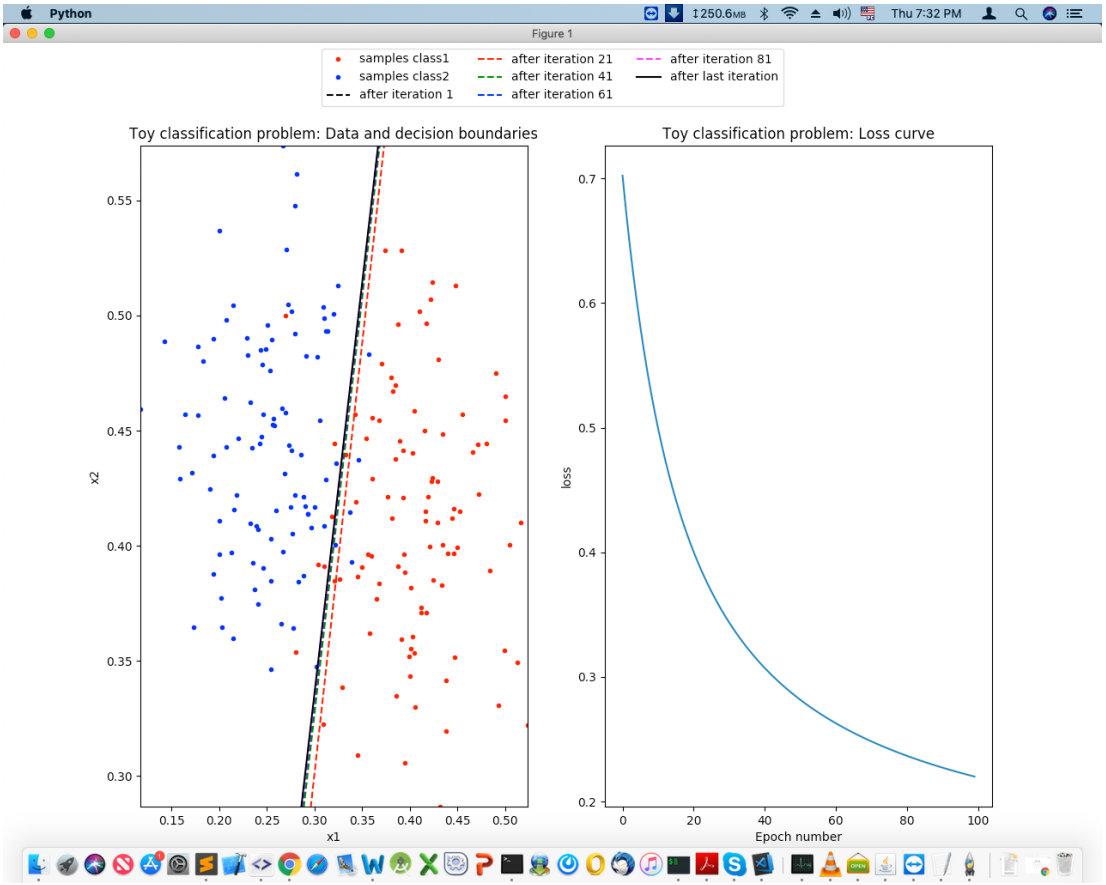
(Proved)

## b) 1st Setting (Default):

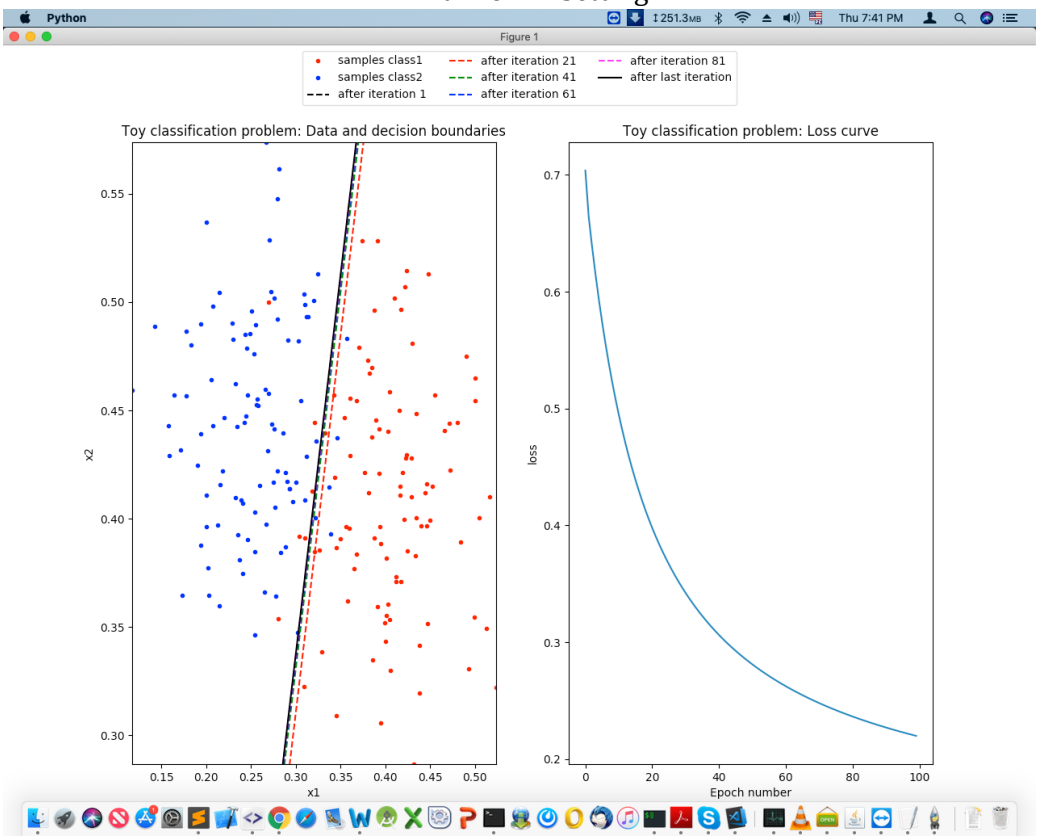
numEpochs = 100

net = MLPClassifier(activation='logistic', hidden_layer_sizes=(), batch_size=1,
learning_rate='constant', learning_rate_init=0.1,
shuffle=False, max_iter=1, warm_start=True, momentum = 0.0,
nesterovs_momentum=False, solver='sgd', verbose=True,
validation_fraction=0.0, alpha=0.0, tol=0.00001) # default for tol: 0.0001

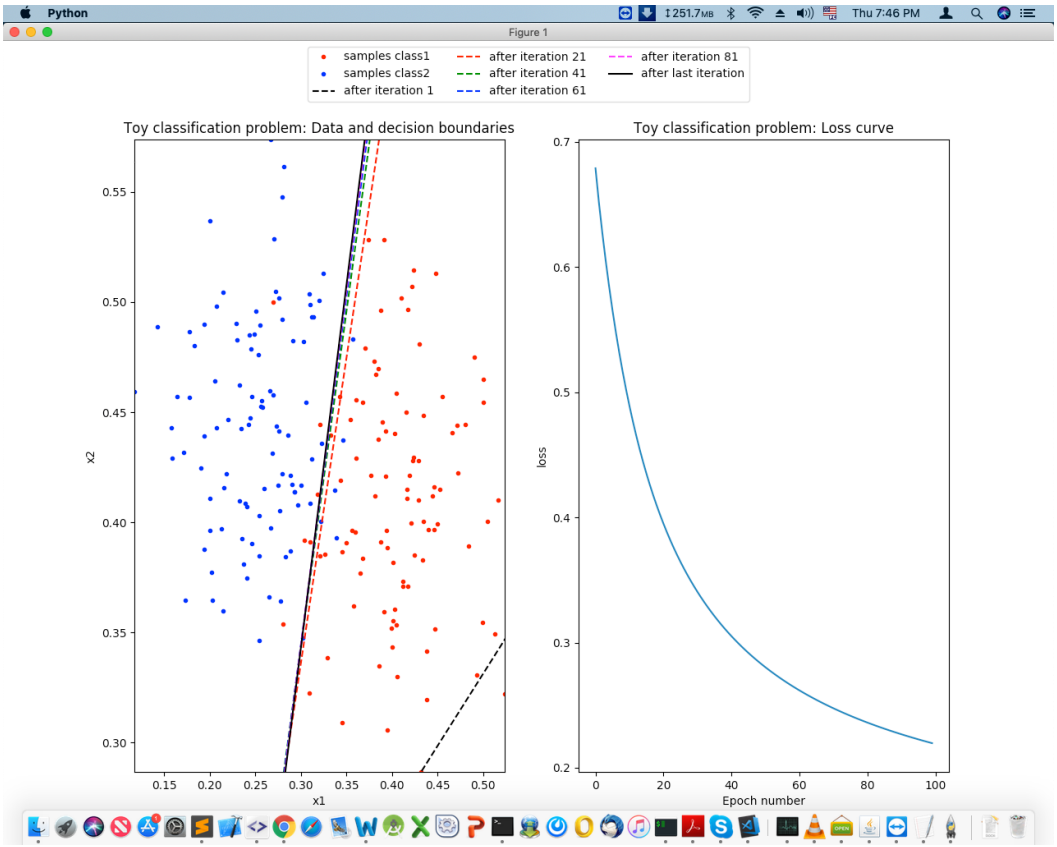
1st run for 1st Setting:



2nd run for 1st Setting:



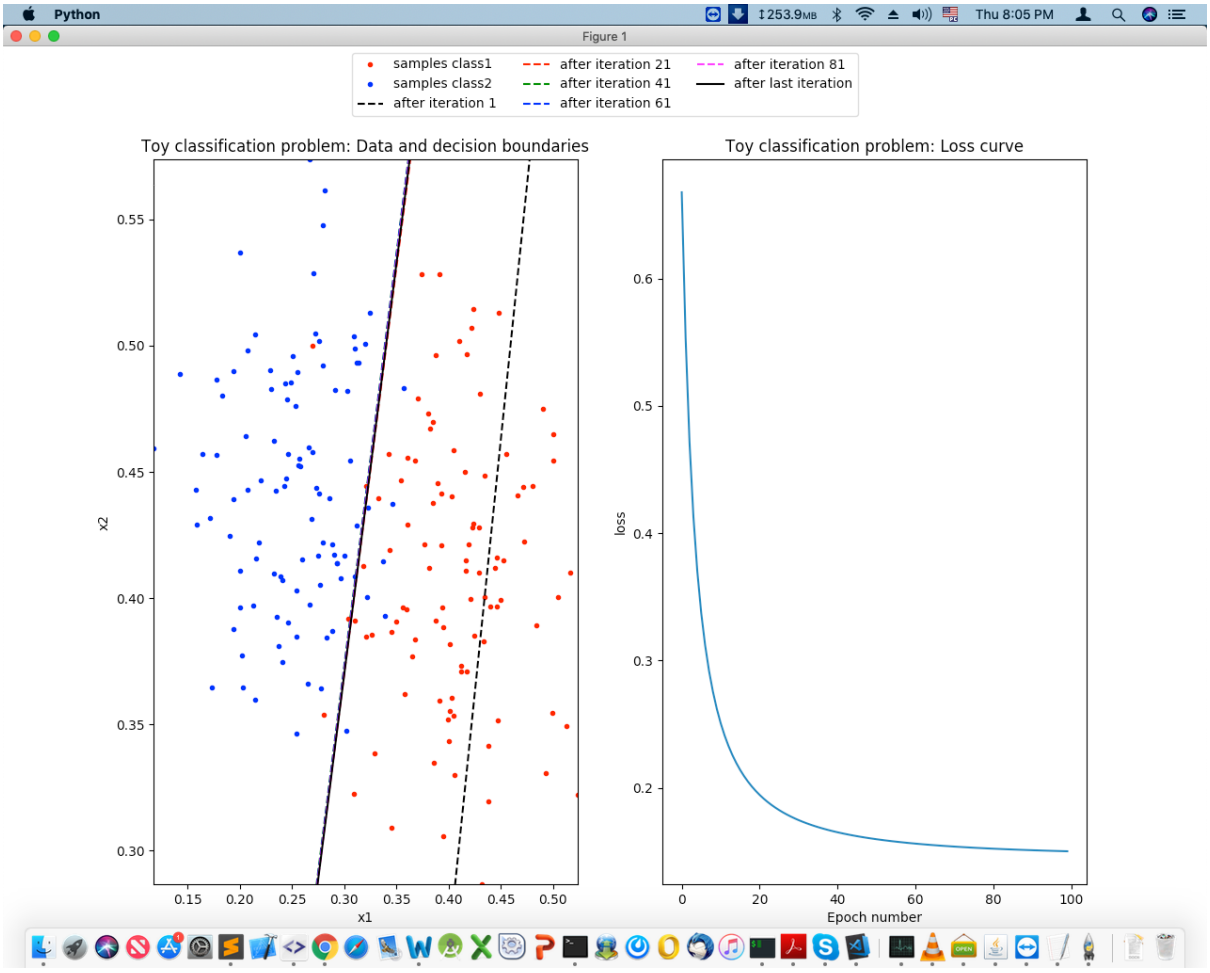

3rd run for 1st Setting:



**Findings from Setting 1:** Number of Errors approximately 11. We are getting different outputs for the same setting.

## 2nd Setting (Changing solver = adam):

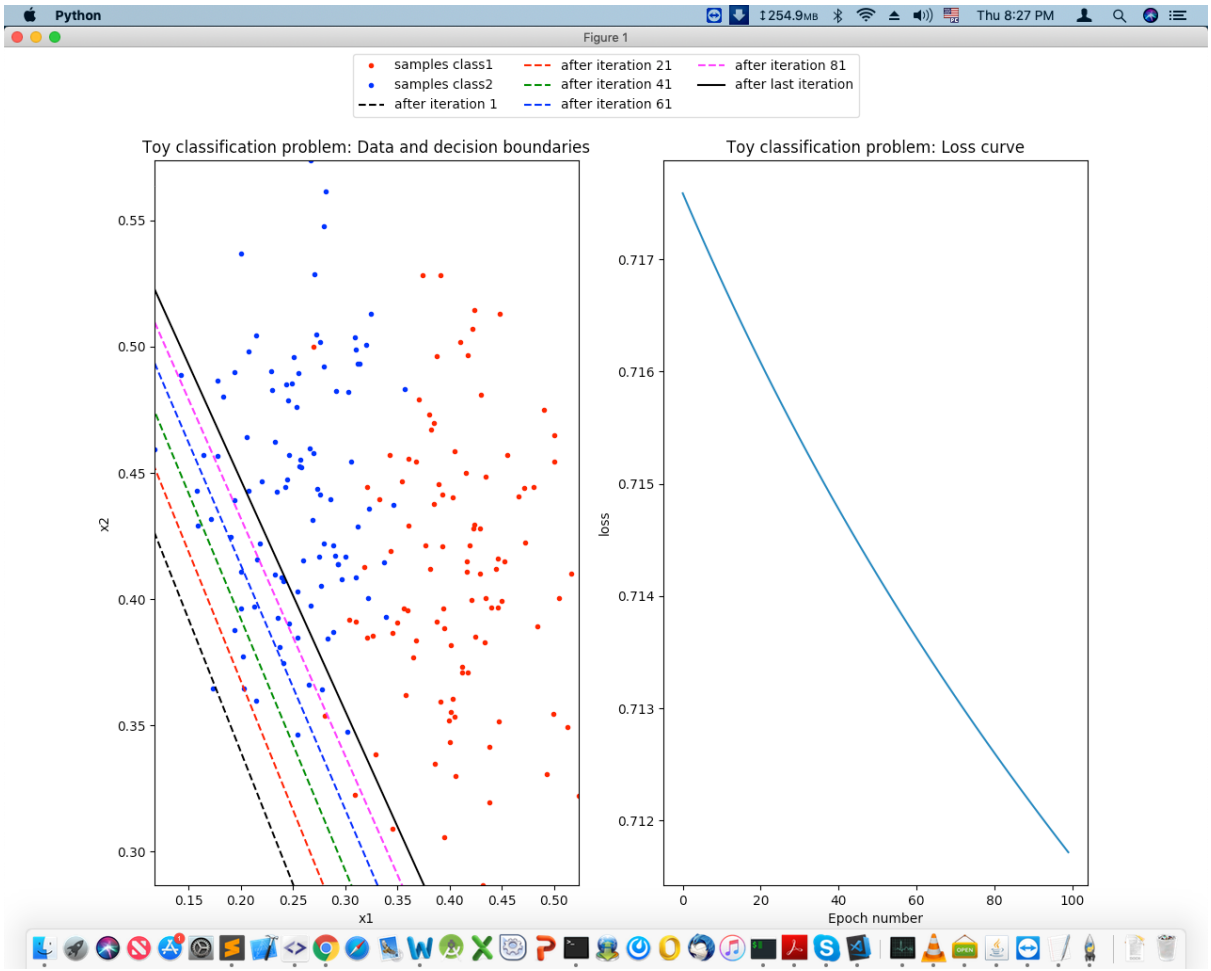
numEpochs = 100
net = MLPClassifier(activation='logistic', hidden_layer_sizes=(), batch_size=1,
learning_rate='constant', learning_rate_init=0.1,
shuffle=False, max_iter=1, warm_start=True, momentum = 0.0,
nesterovs_momentum=False, solver=' adam ', verbose=True,
validation_fraction=0.0, alpha=0.0, tol=0.00001) # default for tol: 0.0001




**Findings from Setting 2**: Number of Errors 11. After 20 iterations the loss reduced from 0.66775000 to 0.19756450.

## 3rd Setting (Changing learning rate):

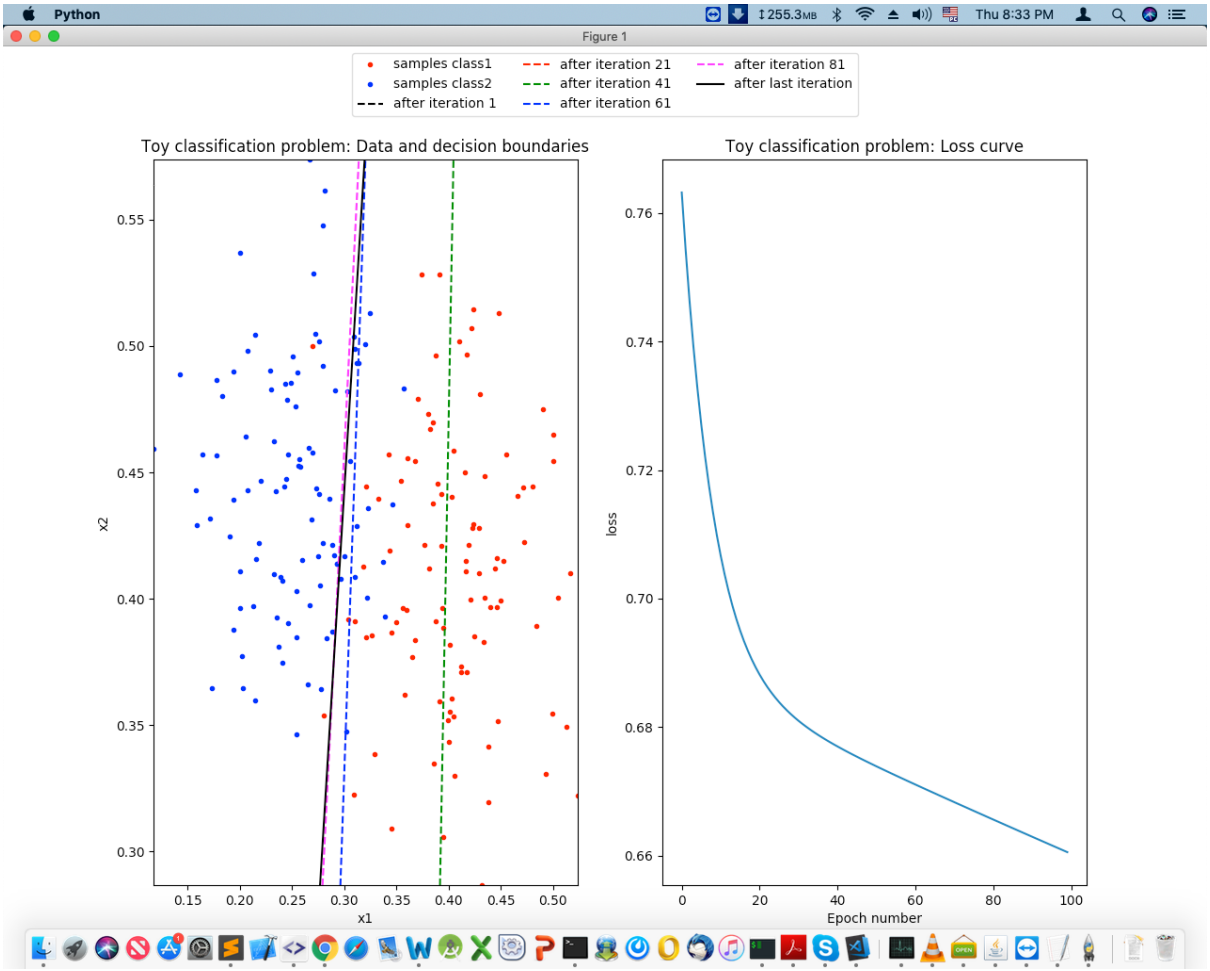
numEpochs = 100
net = MLPClassifier(activation='logistic', hidden_layer_sizes=(), batch_size=1,
learning_rate='constant', learning_rate_init=0.0001,
shuffle=False, max_iter=1, warm_start=True, momentum = 0.0,
nesterovs_momentum=False, solver=' sgd ', verbose=True,
validation_fraction=0.0, alpha=0.0, tol=0.00001) # default for tol: 0.0001




**Findings from Setting 3**: Number of Errors 128. If we set learning_rate too low, then we will get so many binary errors.

## 4th Setting (Changing learning rate again):

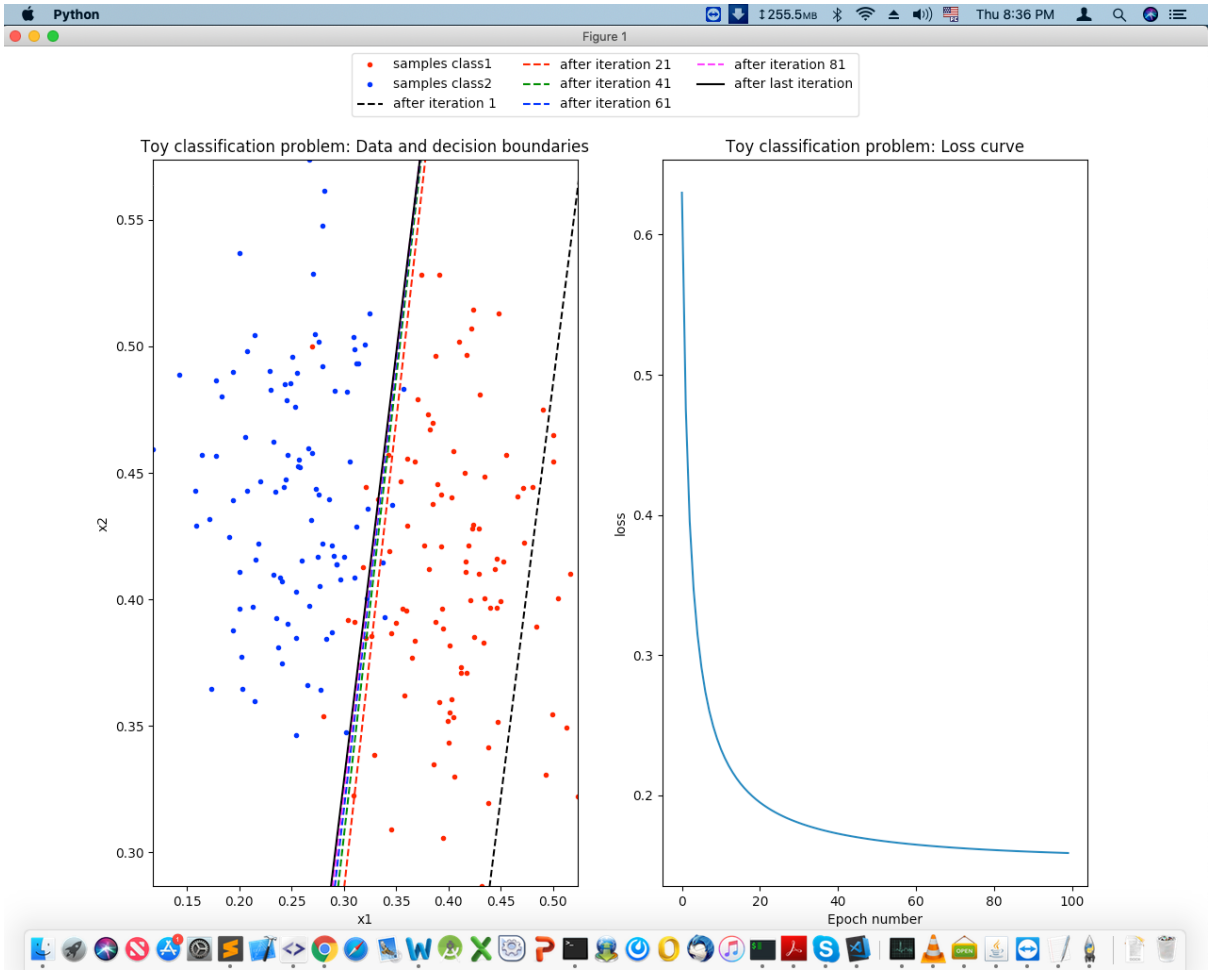
numEpochs = 100
net = MLPClassifier(activation='logistic', hidden_layer_sizes=(), batch_size=1,
learning_rate='constant', learning_rate_init=0.001,
shuffle=False, max_iter=1, warm_start=True, momentum = 0.0,
nesterovs_momentum=False, solver=' sgd ', verbose=True,
validation_fraction=0.0, alpha=0.0, tol=0.00001) # default for tol: 0.0001




**Findings from Setting 4**: Number of Errors 20. Still we didn't find better solution than the default setting.

## 5th Setting (Increase learning rate):

numEpochs = 100
net = MLPClassifier(activation='logistic', hidden_layer_sizes=(), batch_size=1,
learning_rate='constant', learning_rate_init=1,
shuffle=False, max_iter=1, warm_start=True, momentum = 0.0,
nesterovs_momentum=False, solver=' sgd ', verbose=True,
validation_fraction=0.0, alpha=0.0, tol=0.00001) # default for tol: 0.0001

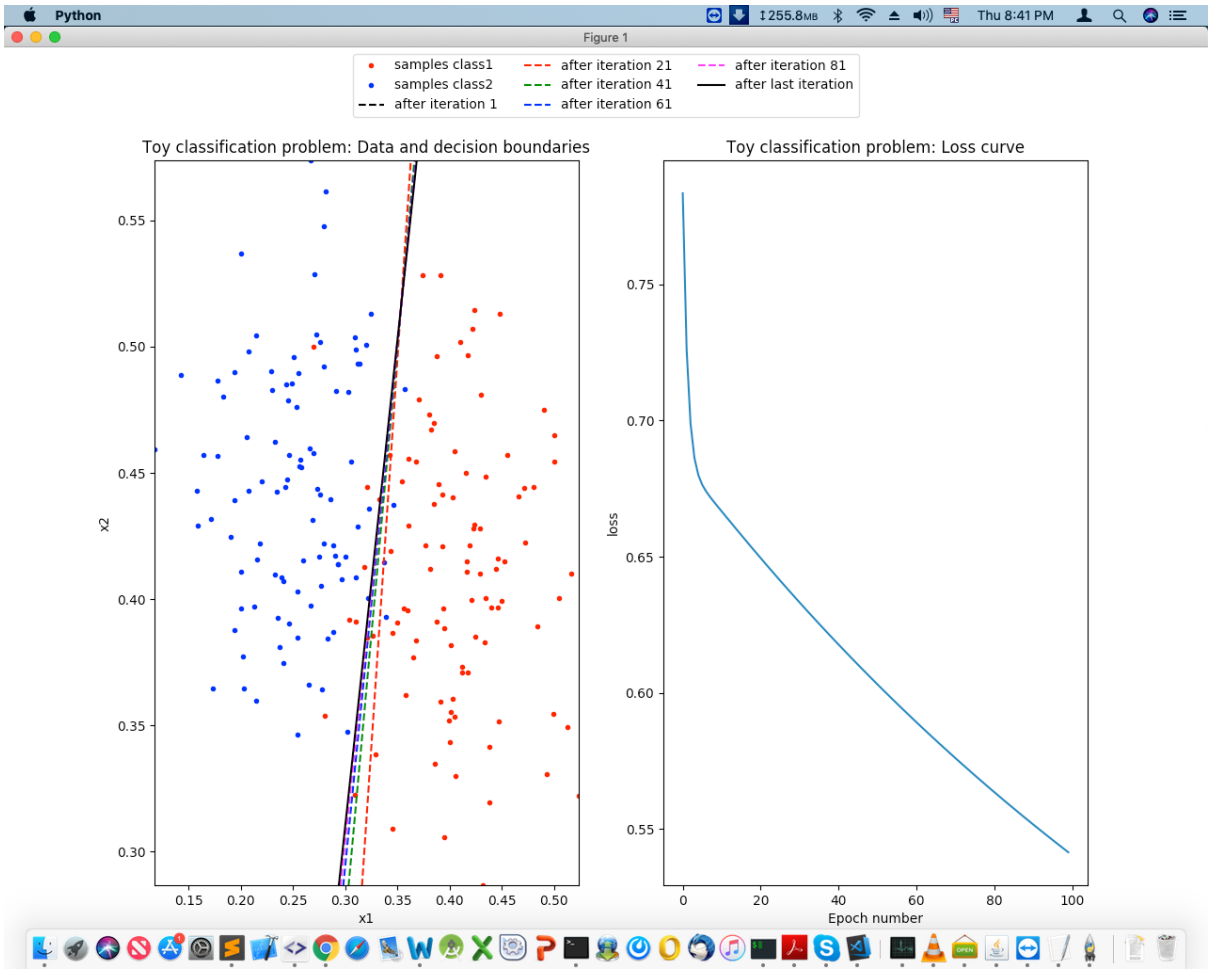



**Findings from Setting 5**: After several runs we found that, number errors are mostly 14. Loss reduces fast but still it's not the best setting.

## 6th Setting (Changing batch size):

numEpochs = 100

net = MLPClassifier(activation='logistic', hidden_layer_sizes=(), batch_size=16,
learning_rate='constant', learning_rate_init=0.1,
shuffle=False, max_iter=1, warm_start=True, momentum = 0.0,
nesterovs_momentum=False, solver=' sgd ', verbose=True,
validation_fraction=0.0, alpha=0.0, tol=0.00001) # default for tol: 0.0001



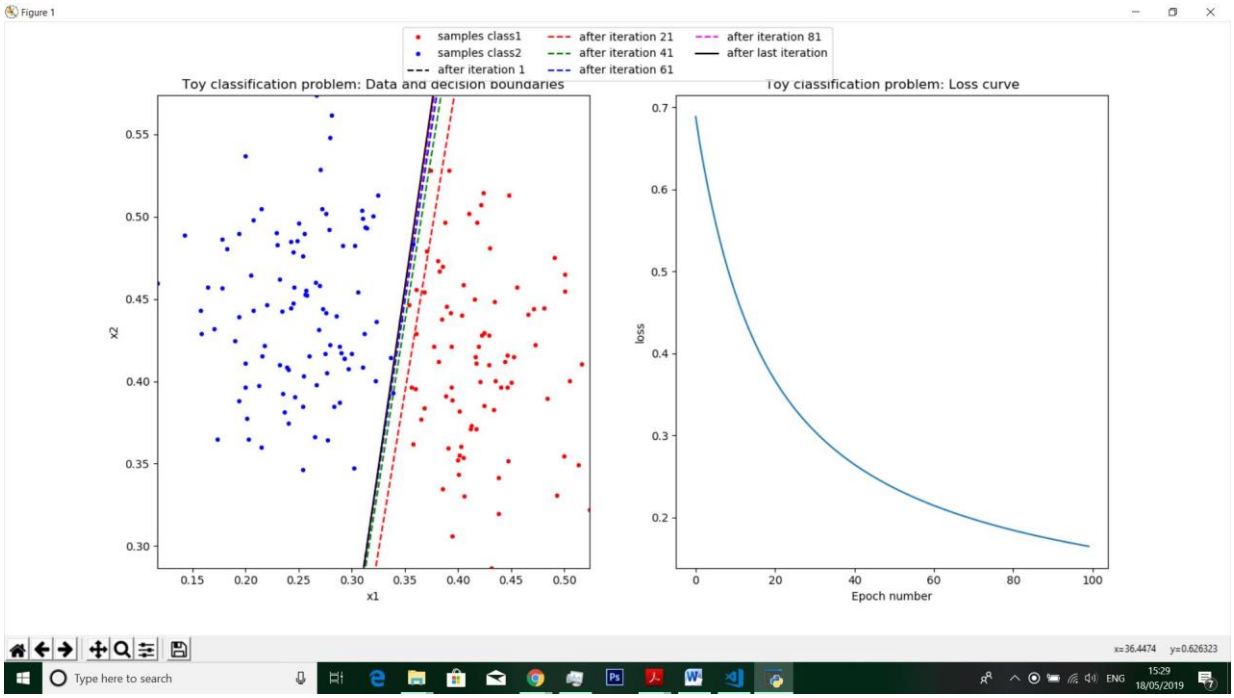

**Findings from Setting 6**: After several runs we found that, number errors are mostly 9 to 13.

**c)** Now we have changed input and target files .

**Setting (Default):**

numEpochs = 100

net = MLPClassifier(activation='logistic', hidden_layer_sizes=(), batch_size=1,
learning_rate='constant', learning_rate_init=0.1,
shuffle=False, max_iter=1, warm_start=True, momentum = 0.0,
nesterovs_momentum=False, solver='sgd', verbose=True,
validation_fraction=0.0, alpha=0.0, tol=0.00001) # default for tol: 0.0001




Findings: After several runs we have found that,

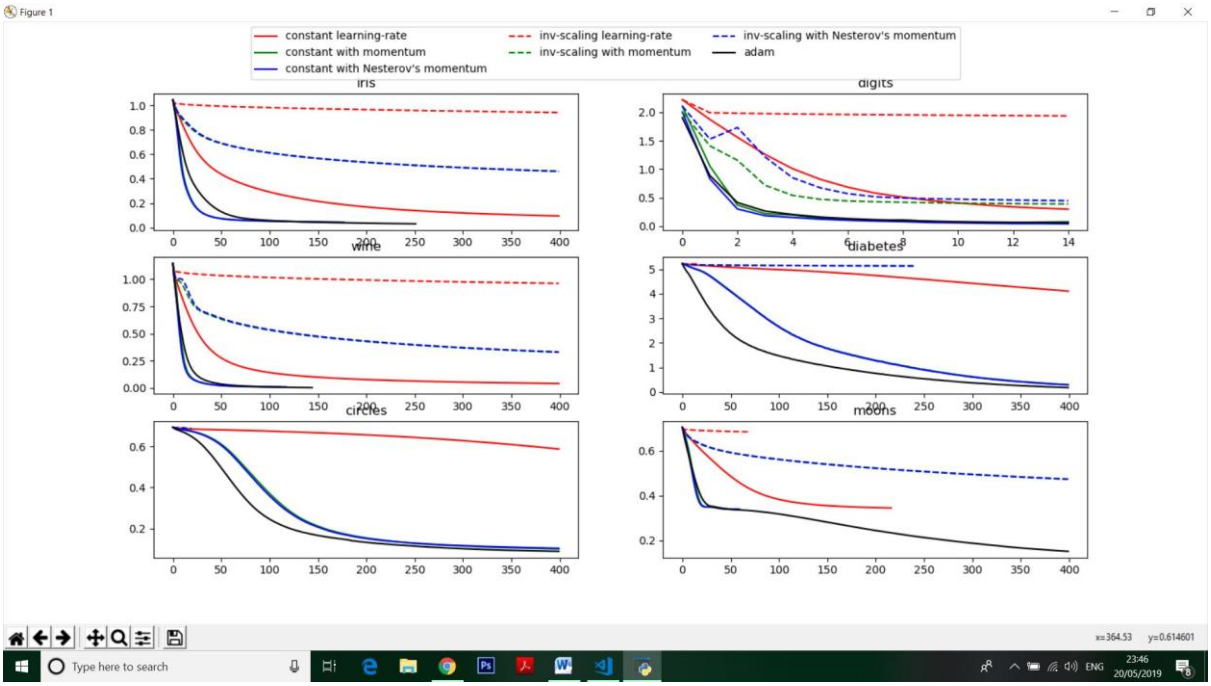
i. The number of binary errors is mostly 3 or 4.
ii. Less loss than previous sample.

**d)** Problem specification is not clear to us.

**e)** We have changed the input and target as specified for XOR function.
Findings: After several runs using default settings, we found that the loss is not decreasing. And the training loss did not improve more than tol=0.000010 for 10 consecutive epochs and learning didn't improved. The loss is always larger than 0.7. This is also another proof that XOR cannot be represented by single layer perceptron.

## Exercise 4 (Multi-layer perceptron and backpropagation – small datasets):

a) We got following output after running this script on default setting:




Output result:

| Dataset | Training Sample | Test Sample | Network input | Network output | Network Layer |
|---|---|---|---|---|---|
| Iris | 105 | 45 | 4 | 3 | 3 |

| Training Category | Training Set Score | Training Set Loss | Test Set Score |
|---|---|---|---|
| Constant Learning Rate | 0.952381 | 0.095458 | 0.977778 |
| Constant with momentum | 0.980952 | 0.043713 | 0.977778 |
| Constant Nesterov's | 0.980952 | 0.043353 | 0.977778 |
| Inv-scaling learning rate | 0.657143 | 0.941146 | 0.555556 |
| Inv-scaling with momentum | 0.866667 | 0.459645 | 0.777778 |
| Inv-scaling with Nesterov | 0.866667 | 0.460321 | 0.777778 |
| Adam | 0.990476 | 0.030960 | 0.977778 |

| Dataset | Training Sample | Test Sample | Network input | Network output | Network Layer |
|---|---|---|---|---|---|
| Digits | 1257 | 540 | 64 | 10 | 3 |

| Training Category | Training Set Score | Training Set Loss | Test Set Score |
|---|---|---|---|
| Constant Learning Rate | 0.9435116 | 0.298351 | 0.925926 |
| Constant with momentum | 0.985680 | 0.075761 | 0.940741 |

| | | | |
|---|---|---|---|
| Constant Nesterov's | 0.996818 | 0.040823 | 0.964815 |
| Inv-scaling learning rate | 0.566428 | 1.936591 | 0.494444 |
| Inv-scaling with momentum | 0.911695 | 0.388572 | 0.888889 |
| Inv-scaling with Nesterov | 0.910103 | 0.446801 | 0.890741 |
| Adam | 0.993636 | 0.060121 | 0.970370 |

| Dataset | Training Sample | Test Sample | Network input | Network output | Network Layer |
|---|---|---|---|---|---|
| Wine | 124 | 54 | 13 | 3 | 3 |

| Training Category | Training Set Score | Training Set Loss | Test Set Score |
|---|---|---|---|
| Constant Learning Rate | 1.000000 | 0.042257 | 1.000000 |
| Constant with momentum | 1.000000 | 0.009941 | 1.000000 |
| Constant Nesterov's | 1.000000 | 0.010130 | 1.000000 |
| Inv-scaling learning rate | 0.661290 | 0.962229 | 0.703704 |
| Inv-scaling with momentum | 0.967742 | 0.329851 | 0.962963 |
| Inv-scaling with Nesterov | 0.967742 | 0.331924 | 0.962963 |
| Adam | 1.000000 | 0.004960 | 1.000000 |

| Dataset | Training Sample | Test Sample | Network input | Network output | Network Layer |
|---|---|---|---|---|---|
| Diabetes | 309 | 133 | 10 | 181 | 3 |

| Training Category | Training Set Score | Training Set Loss | Test Set Score |
|---|---|---|---|
| Constant Learning Rate | 0.103560 | 4.106060 | 0.007519 |
| Constant with momentum | 0.993528 | 0.298405 | 0.007519 |
| Constant Nesterov's | 0.987055 | 0.293299 | 0.007519 |
| Inv-scaling learning rate | 0.009709 | 5.213435 | 0.000000 |
| Inv-scaling with momentum | 0.012945 | 5.132862 | 0.007519 |
| Inv-scaling with Nesterov | 0.012945 | 5.132669 | 0.007519 |
| Adam | 0.993528 | 0.182448 | 0.000000 |

| Dataset | Training Sample | Test Sample | Network input | Network output | Network Layer |
|---|---|---|---|---|---|
| Circles | 70 | 30 | 2 | 1 | 3 |

| Training Category | Training Set Score | Training Set Loss | Test Set Score |
|---|---|---|---|
| Constant Learning Rate | 0.857143 | 0.586886 | 0.800000 |
| Constant with momentum | 0.942857 | 0.101080 | 0.866667 |
| Constant Nesterov's | 0.957143 | 0.104257 | 0.866667 |
| Inv-scaling learning rate | 0.514286 | 0.690762 | 0.466667 |
| Inv-scaling with momentum | 0.628571 | 0.687026 | 0.666667 |
| Inv-scaling with Nesterov | 0.528571 | 0.687378 | 0.600000 |
| Adam | 0.957143 | 0.089005 | 0.866667 |

| Dataset | Training Sample | Test Sample | Network input | Network output | Network Layer |
|---|---|---|---|---|---|
| Moons | 70 | 30 | 2 | 1 | 3 |

| Training Category | Training Set Score | Training Set Loss | Test Set Score |
|---|---|---|---|
| Constant Learning Rate | 0.842857 | 0.344432 | 0.866667 |
| Constant with momentum | 0.828571 | 0.339465 | 0.900000 |
| Constant Nesterov's | 0.828571 | 0.338865 | 0.900000 |
| Inv-scaling learning rate | 0.485714 | 0.684865 | 0.533333 |
| Inv-scaling with momentum | 0.814286 | 0.472911 | 0.833333 |
| Inv-scaling with Nesterov | 0.814286 | 0.473382 | 0.833333 |
| Adam | 0.942857 | 0.150447 | 0.933333 |

b) After running this script using default setting we have found following graphs:

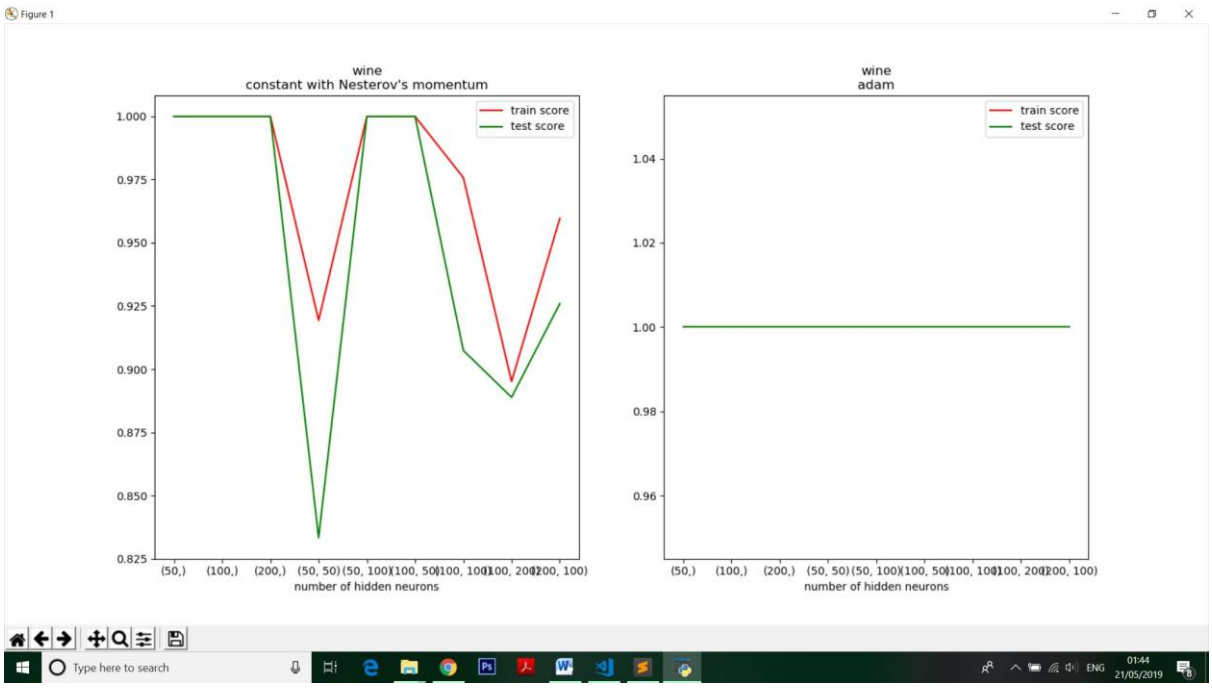



Figure 1
wine
constant with Nesterov's momentum
train score
test score
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
(50,) (100,) (200,) (50, 50) (50, 100) (100, 50) (100, 100) (100, 200) (200, 100)
number of hidden neurons
wine
adam
train score
test score
1.04
1.02
1.00
0.98
0.96
(50,) (100,) (200,) (50, 50) (50, 100) (100, 50) (100, 100) (100, 200) (200, 100)
number of hidden neurons
Type here to search
ENG
01:44
21/05/2019

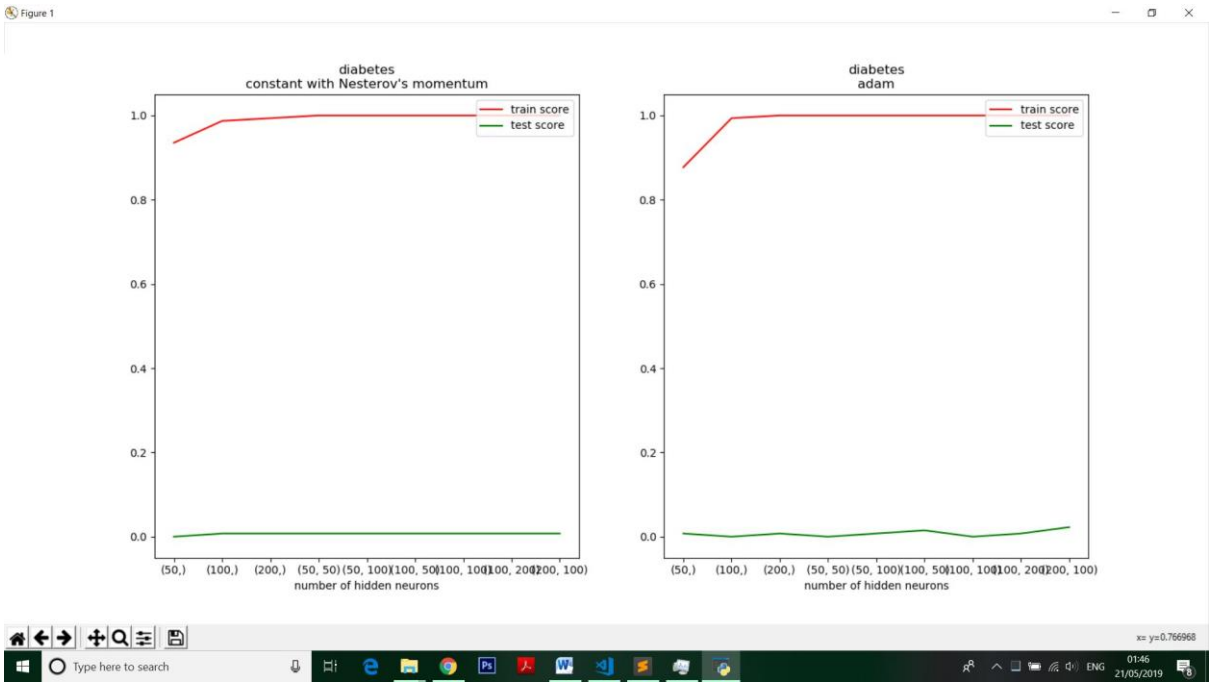


Figure 1
diabetes
constant with Nesterov's momentum
train score
test score
1.0
0.8
0.6
0.4
0.2
0.0
(50,) (100,) (200,) (50, 50) (50, 100) (100, 50) (100, 100) (100, 200) (200, 100)
number of hidden neurons
diabetes
adam
train score
test score
1.0
0.8
0.6
0.4
0.2
0.0
(50,) (100,) (200,) (50, 50) (50, 100) (100, 50) (100, 100) (100, 200) (200, 100)
number of hidden neurons
x= y=0.766968
Type here to search
ENG
01:46
21/05/2019


Figure 1
circles
constant with Nesterov's momentum
train score
test score
0.95
0.90
0.85
0.80
(50,) (100,) (200,) (50, 50) (50, 100) (100, 50) (100, 100) (100, 200) (200, 100)
number of hidden neurons
circles
adam
train score
test score
0.96
0.94
0.92
0.90
0.88
(50,) (100,) (200,) (50, 50) (50, 100) (100, 50) (100, 100) (100, 200) (200, 100)
number of hidden neurons
x= y=0.955748
Type here to search
ENG
01:46
21/05/2019


Figure 1
moons
constant with Nesterov's momentum
train score
test score
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.76
(50,) (100,) (200,) (50, 50) (50, 100) (100, 50) (100, 100) (100, 200) (200, 100)
number of hidden neurons
moons
adam
train score
test score
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
(50,) (100,) (200,) (50, 50) (50, 100) (100, 50) (100, 100) (100, 200) (200, 100)
number of hidden neurons
x= y=0.879111
Type here to search
ENG
01:47
21/05/2019