Lab Course Machine Learning
— Problem Set 4—

Eugeniu Vezeteu, Alberto Gonzalo Rodriguez Salgado

July 19, 2020

Part 1: Implementation

Assignment 1 Neural Network

In this assignment we implemented $neural\_network$ class.

0.1 ReLU

The general formula of ReLU activation function is:

$$ReLU(X) = max(0, W^T * X + b)$$

We implemented ReLU version with dropout, for training $A = \delta * max(0, W^T * X + b)$ where δ is Bernoulli sampled $\delta = B(1 - p)$. For test case, $A = max(0, (1 - p) * (W^T * X + b)$

0.2 Softmax

This is the activation function of the last layer, it computes the probability of each classes. The softmax formula is:

$$A = \frac{e^{W_c * h + b_c}}{\sum_{c'} e^{W_{c'} * h + b_{c'}}}$$

0.3 Forward

In this function we pass the input through the layers and compute the model output. We computed the input of each neuron as $Z = W * X + b$ and pass it to activation function ReLU, for each layers, except the last one. On the last layer we applied softmax activation function to get the probability of each class.

0.4 Loss

In this function we computed the cross-entropy loss accordingly:

$$L(yTrue, yPred) = -(yTrue * log(yPred) + (1 - yTrue) * (1 - log(yPred)))/n$$

For the case when regularization parameter is not zero, we computed the regularization cost for each layer :

$$L_{regularization} = \lambda * \sum_l ||W_l||^2$$

$$Loss_{final} = L(yTrue, yPred) + L_{regularization}$$

0.5 Fit

In this function we update the model parameters to minimize the cost function. Our loss function is cross-entropy:

$$L = -\frac{1}{n} * (Y * log(Ypred) - (1 - Y) * log(1 - Ypred))$$

and we need compute the $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial b}$, using chain rule:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial A} * \frac{\partial A}{\partial Z} * \frac{\partial Z}{\partial W}$$

$$\frac{\partial L}{\partial W} = L' * A' * Z'$$

where:
L' - derivative of loss function
derivative of cross-entropy: $[-\frac{Y}{Ypred} + \frac{1-Y}{1-Ypred}]$
A' - is the derivative of the activation function, Z' - derivative of (W*X+b) w.r.t. W, Z' = X

and

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial A} * \frac{\partial A}{\partial Z} * \frac{\partial Z}{\partial b}$$

where A is activation function and $Z = W * X + b$. and $\frac{\partial Z}{\partial b} = \partial Z$

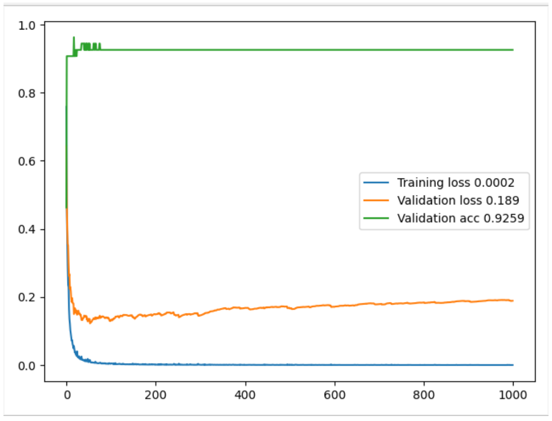


Figure 1: No regularization

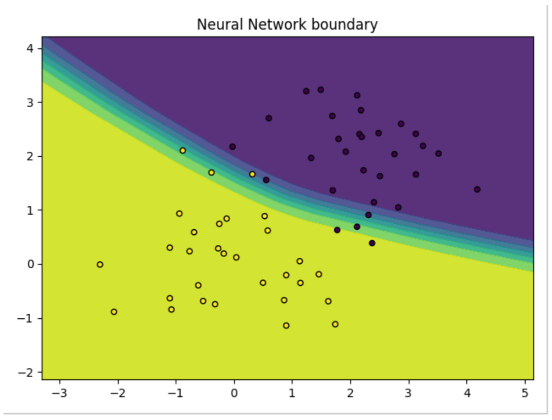


Figure 2: Boundary plot - No regularization

In figure 1 we have the neural network training loss of 0.0002 , the test loss = 0.13 , loss on validation data is 0.18. We trained the model for 1000 of epochs and a batch size of 20, without regularization.

When regularization parameter λ is not zero, compute the gradient $\frac{\partial}{\partial W} = \frac{\partial}{\partial W} + \frac{\lambda}{batch\_size} * W$

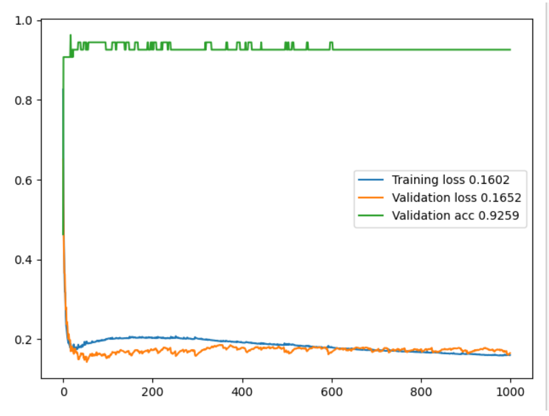


Figure 3: With $\lambda = 0.2$ regularization

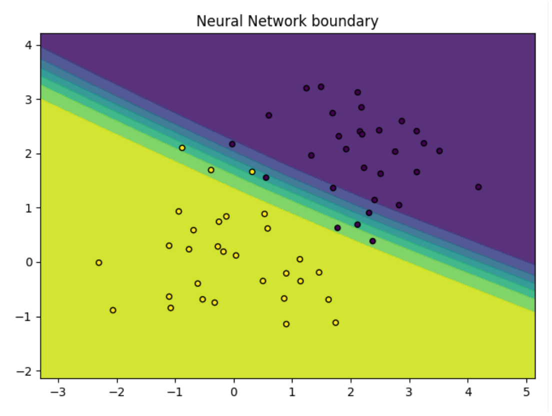


Figure 4: Boundary $\lambda = 0.2$ regularization

With regularization of $\lambda = 0.2$ we have smaller validation loss , 0.16 compared to 0.18, and bigger training loss, 0.16 compared to 0.0002. Regularization helps to avoid overfitting, see figure 1 and 3

Assignment 2 Plot boundary 2d

In this function we plotted the points as circle in different colours for each class.
The support vector were plotted as red crosses.
We computed the meshgrid (x,y), and Z as prediction of the mesh using the given parameter model, and then we plotted the contours using x, y mesh and computed Z.

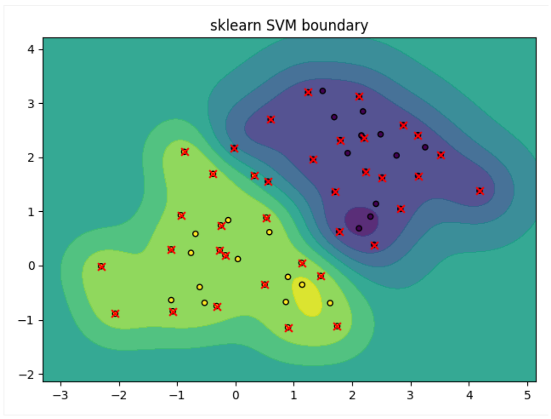


Figure 5: SVM boundary with sklearn

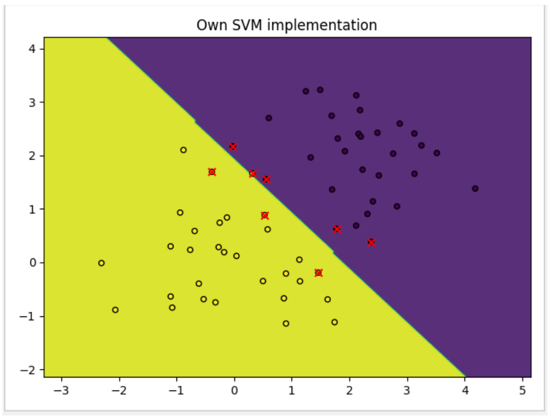


Figure 6: Own implemented SVM boundary

Assignment 3 SVM

In this assignment we had to implement the soft margin support vector machine algorithm. In soft margin SVM, we end up needing to solve a dual problem for ω and the dual variable α. In order to find the optimum α, since it is a quadratic problem, we need to solve a dual quadratic programming problem. The optimization problem we are looking to solve reads

$$max_\alpha - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j k\left(x_i, x_j\right) + \sum \alpha_i \tag{1}$$
$$s.t 0 \leq \alpha_i \leq \gamma \tag{2}$$
$$s.t \sum \alpha_i y_i = 0 \tag{3}$$

Since we will be using the cvxopt package functions to solve the quadratic optimization problem, we had to pass the optimization problem in a very specific vectorized form to the function. The cvxopt qp solver needs the optimization problen as follows:

$$min_x \frac{1}{2}\boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x} \tag{4}$$
$$s.t\boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h} \tag{5}$$
$$s.t\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{6}$$

Note that the qp function requires a minimzation problem and we are trying to maximize a function. Therefore, since it is the same problem, we can choose to minimize the next problem where we just multiply by -1 the optimization problem:

$$min_\alpha \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j k\left(x_i, x_j\right) - \sum \alpha_i \tag{7}$$
$$s.t 0 \leq \alpha_i \leq \gamma \tag{8}$$
$$s.t \sum \alpha_i y_i = 0 \tag{9}$$

When vectorizing the optimization problem, we found out the next variables

- $\boldsymbol{P} = \left(\boldsymbol{Y} \cdot \boldsymbol{Y}^T\right) * \boldsymbol{K}$, where * represents an element wise array multiplication and $\boldsymbol{K}$ is the kernel matrix.
- $\boldsymbol{q}$ = - $\boldsymbol{e}$ where $\boldsymbol{e}$ is a nx1 ones array. This corresponds to the right term of the optimization function where we are summing over all α_i.

- $\boldsymbol{G}$ is a matrix composed of two identity matrixes $\boldsymbol{I}$ and $-\boldsymbol{I}$ that are vertically stacked together. The first one corresponds to the condition that $\alpha_i < \gamma$ and the second one to the condition that $\alpha_i > 0$. Note that the original equation reads $-\alpha_i < 0$ so that we have the form $\boldsymbol{G\alpha} < \boldsymbol{h}$ and due to the -1 it is the same as $\alpha_i > 0$.

- $\boldsymbol{h}$ accordingly consists of two column vectors, the first one being a vector $[\gamma, \gamma ..., \gamma]$ with n components according to the first condition and the second one is a zeros vector $\mathbf{0}$ with n components as well describing the first condition as well.

- $\boldsymbol{A} = \boldsymbol{Y^T}$ (1xn row vector) since we need to multiply Y with each α. This corresponds to the second condition.

- $\boldsymbol{b} = 0$ is a scalar corresponding to the second condition.

After solving the quadratic problem, we obtain one α_i value for each data point x_i. Therefore, we have an nx1 array $\boldsymbol{\alpha}$. In SVM only the support vectors will affect the prediction and since the $\boldsymbol{\alpha}$ vector tends to be sparse, we do not need to use most of the data points later on for prediction, we just need to use the support vectors which are the datapoints which have an $\alpha_i \geq tol$ (e.g tol=1^{-3}). By doing so, we end up having a data matrix $\boldsymbol{X_{sv}}$ containing the support vectors and a label matrix Y_{sv} containing the labels of the support vectors.

Since b is the same for all data points, we can compute b as follows:

$$b = y_0 - \sum_i \boldsymbol{\alpha_{sv}}_i * \boldsymbol{Y_{sv}}_i * \boldsymbol{K_{sv}}_i \tag{10}$$

Last but not leas, as stated before, we just only need to use the support vectors and its labels for prediction. For a new data point $\boldsymbol{x}$, we can predict its classification as follows:

$$y_{pred} = \sum \boldsymbol{K} * \boldsymbol{\alpha_{sv}} * \boldsymbol{Y_{sv}} + b \tag{11}$$

In order to get a class prediction of -1 or 1, we can just apply the sign function:

$$y_{pred,class} = sign\left(y_{pred}\right) \tag{12}$$

Part 2: Application

Assignment 4 SVM QP on easy 2d dataset

On this assignment we used SVM QP with gaussian kernel. To find the optimal parameters C and δ we used cross-validation with k-fold = 10 number of repetitions = 2.

Parameters candidates for CV are presented in Table 1

Table 1: Searching space for CV

Method	**kernel**	**kernel parameter**	**regularization**
CV	gaussian	[.1, .5, .9, 1., 2., 3.]	np.linspace(1, 500, num=20) + None

The original training and testing dataset is plotted in figure 16.
The best fit, overfit and underfit parameters are presented in table 2

Table 2: Fit parameters

Fit	**kernel**	**kernel parameter**	**regularization**
Best fit	gaussian	2	263.63
Overfit	gaussian	.01	10000
Underfit	gaussian	7	None

The plot result of best fit parameters are in figure 8 and 9.

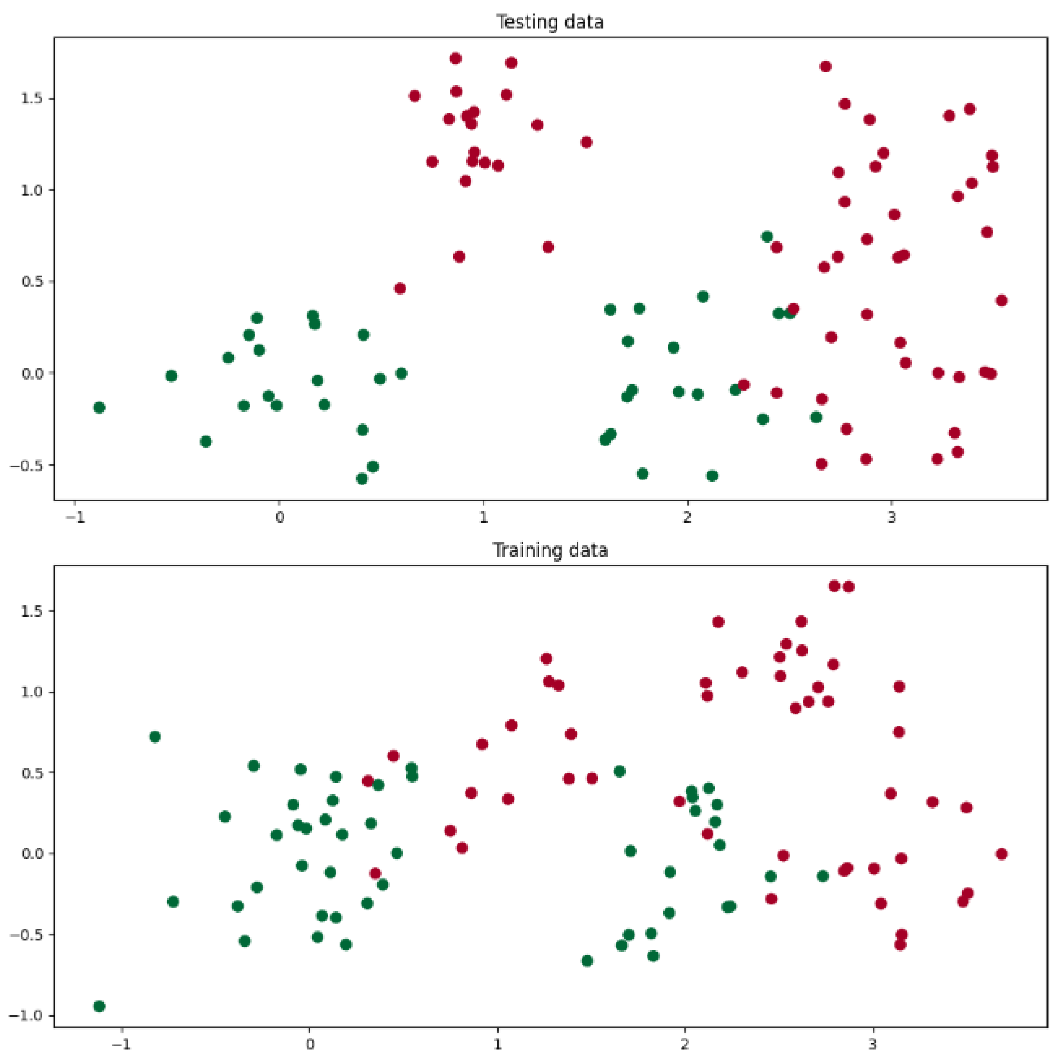


Figure 7: Original training testing dataset

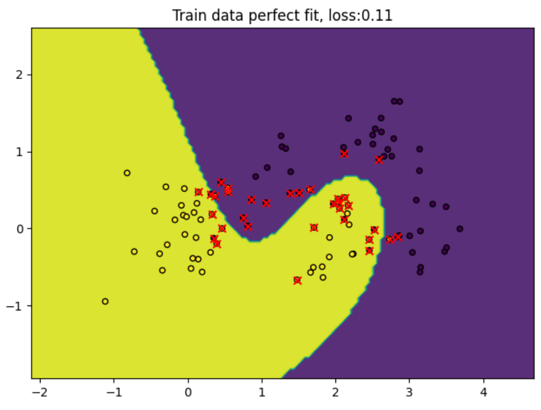


Figure 8: Best fit training data, loss = 0.11

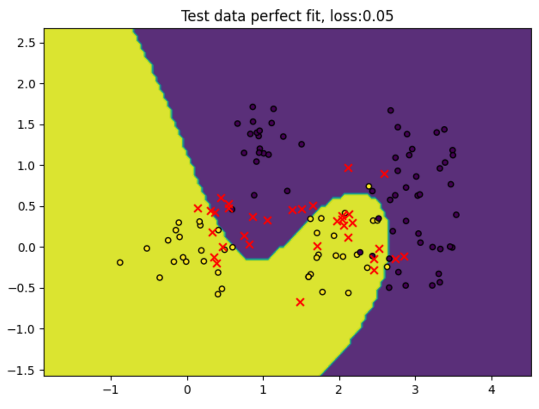


Figure 9: Best fit testing data, loss = 0.05

In figure 10 and 11 we plotted the underfitting case with linear kernel, just to see how linear kernel performs on this data.

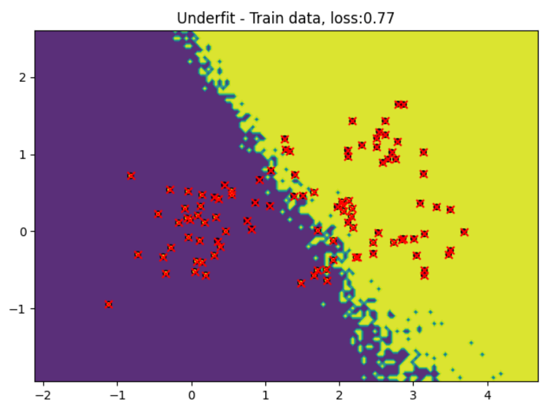


Figure 10: Underfit training data with linear kernel, loss = 0.77

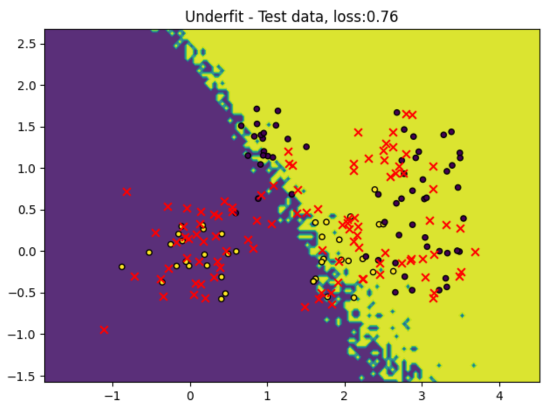


Figure 11: Underfit testing data with linear kernel, loss = 0.76

In figures 12 and 13 we plotted the underfitting result for training data and test data using gaussian kernel.

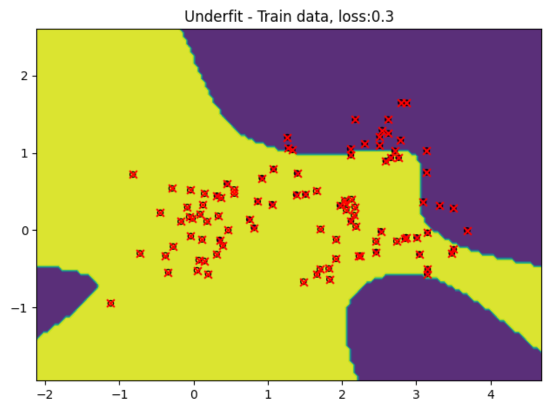


Figure 12: Underfit training data, loss = 0.3

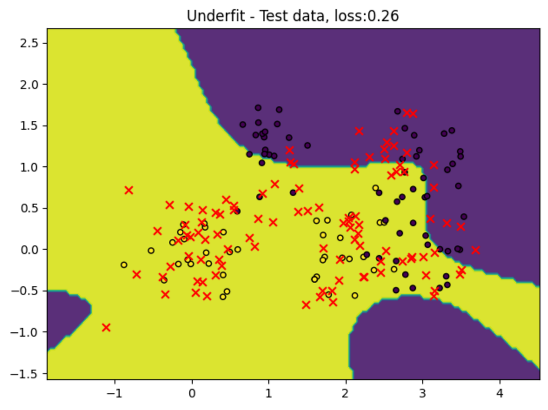


Figure 13: Underfit testing data, loss = 0.26

The overfit case is presented in figures 14 and 15.

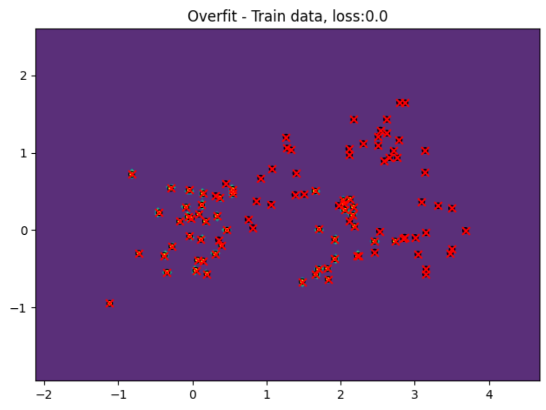


Figure 14: Overfit training data, loss = 0.0

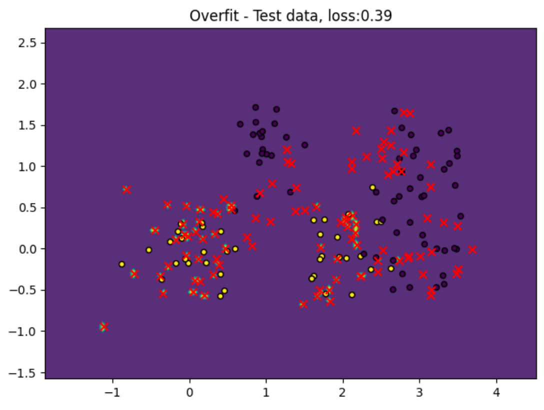


Figure 15: Overfit testing data, loss = 0.39

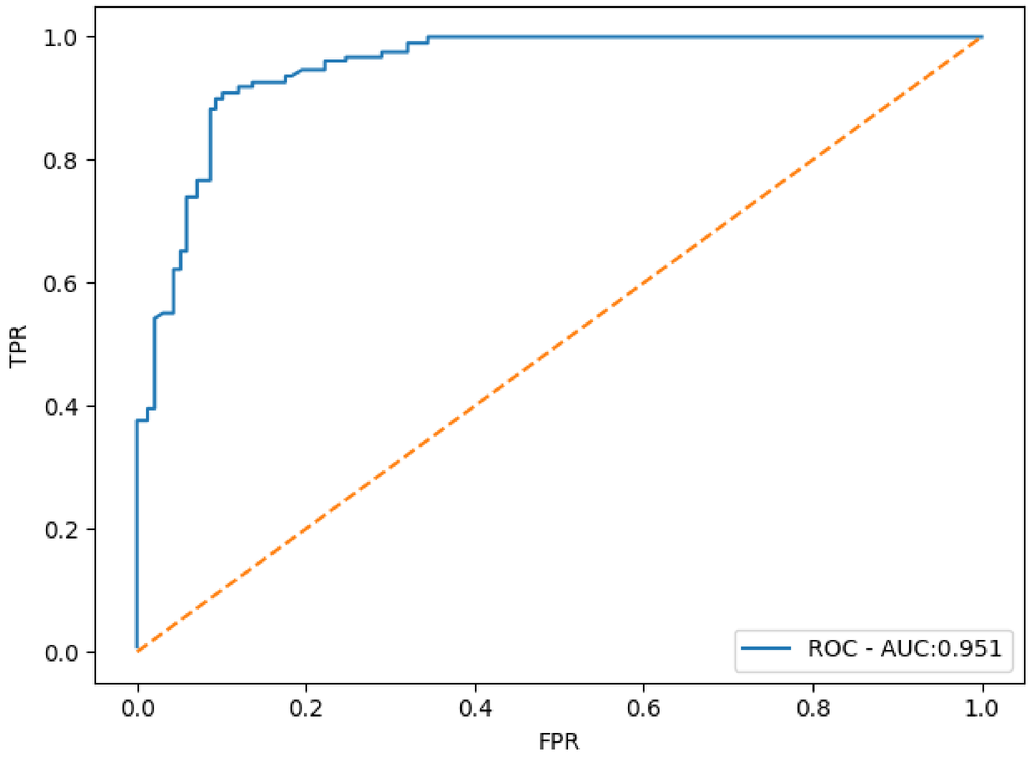


Figure 16: AUC = 0.95

0.6 Conclusion

- **Well fit** - small training loss and small validation loss
- **Overfit** - small training loss, big validation loss
- **Underfit** - big training loss, big validation loss

Assignment 5 Iris dataset

In this assignment we predicted iris dataset with SVM and with Neural Network.
On the first step we plotted the dataset using seaborn, to get an idea of how the data looks, figure 17.

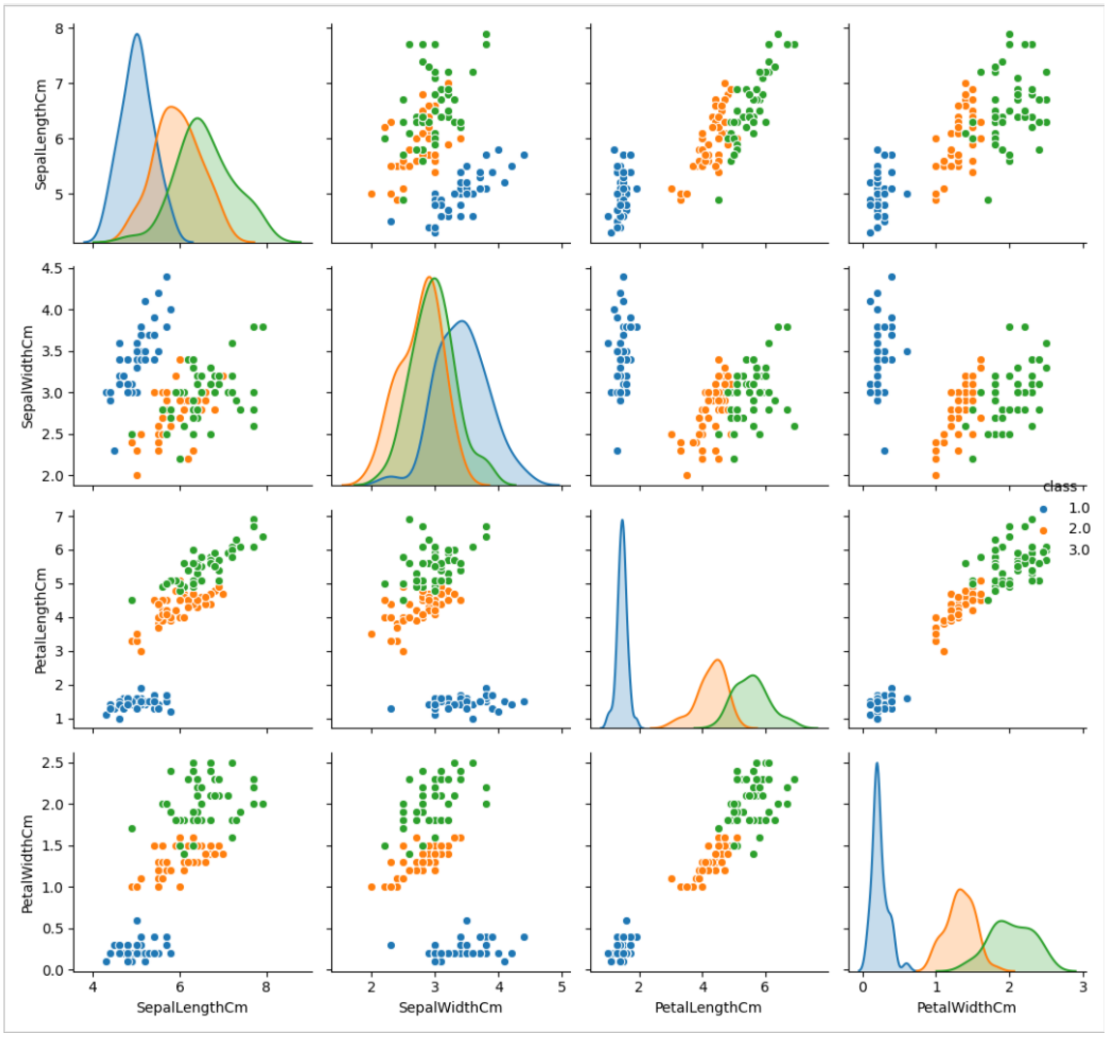


Figure 17: Iris dataset

- **SVM** - here we transformed the data to be one vs other. We have 3 classes, so we have :(1,2) vs 3, (1,3) vs 2, (2,3) vs 1. We used cross-validation and searched for kernel = (linear, polynomial, gaussian), kernelparameter = [1,2,3]. We splitted the dataset into 0.7 training, 0.3 testing.

  We transformed the y data, $y = \pm 1$.
  Case (2,3) VS 1, we have accuracy = 1.0

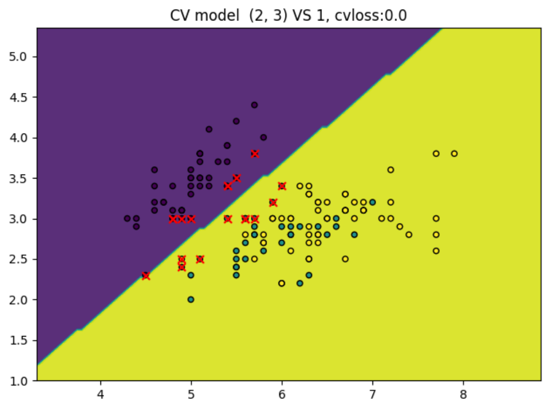


Figure 18: SVM , polynomial kernel, kernelparam=1, C=1, loss=0.0

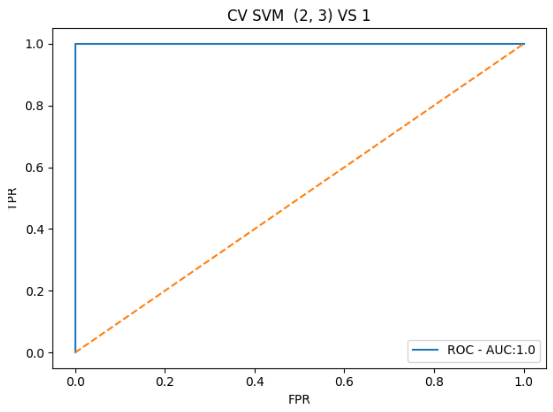


Figure 19: (2,3) vs 1, AUC:1.0

From figures 18 and 19 we notice that class (2,3) and 1 are linearly separable.

Case (1,3) VS 2

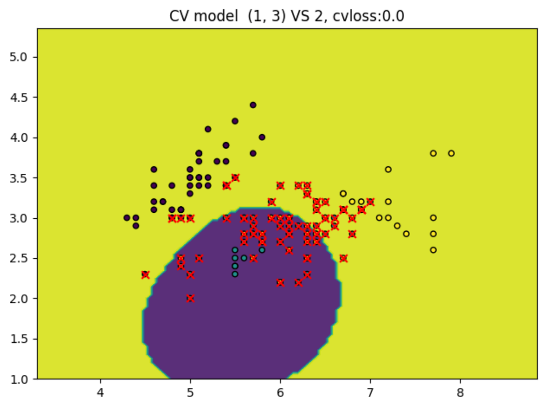


Figure 20: SVM , gaussian kernel, kernel-param=1, C=1, loss=0.07

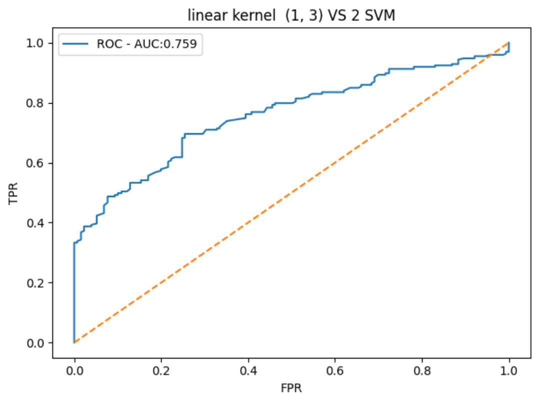


Figure 21: (1,3) vs 2, AUC:0.75

We notice that linear kernel is poor for case (1,3) VS 2, we conclude that classes 2 and 3 aren't linearly separable.

Case (1,2) VS 3, accuracy 0.95

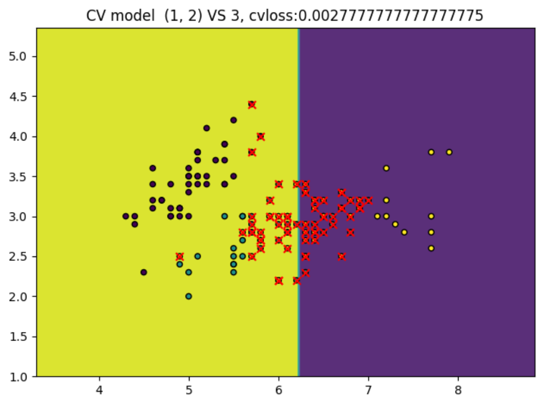


Figure 22: SVM , polynomial kernel, kernel-param=1, C=1, loss=0.002

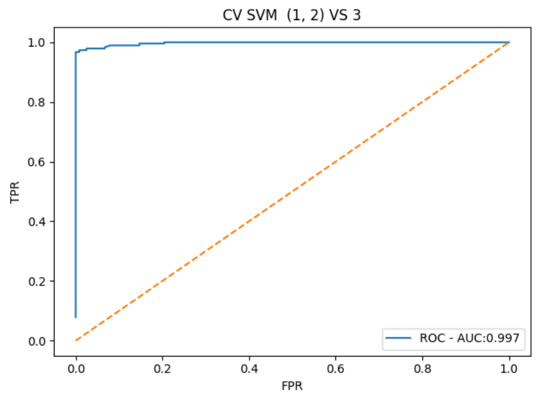


Figure 23: (1,2) vs 3, AUC:0.99

From figures 20 and 22 we notice that class 2 and 3 aren't linearly separable.

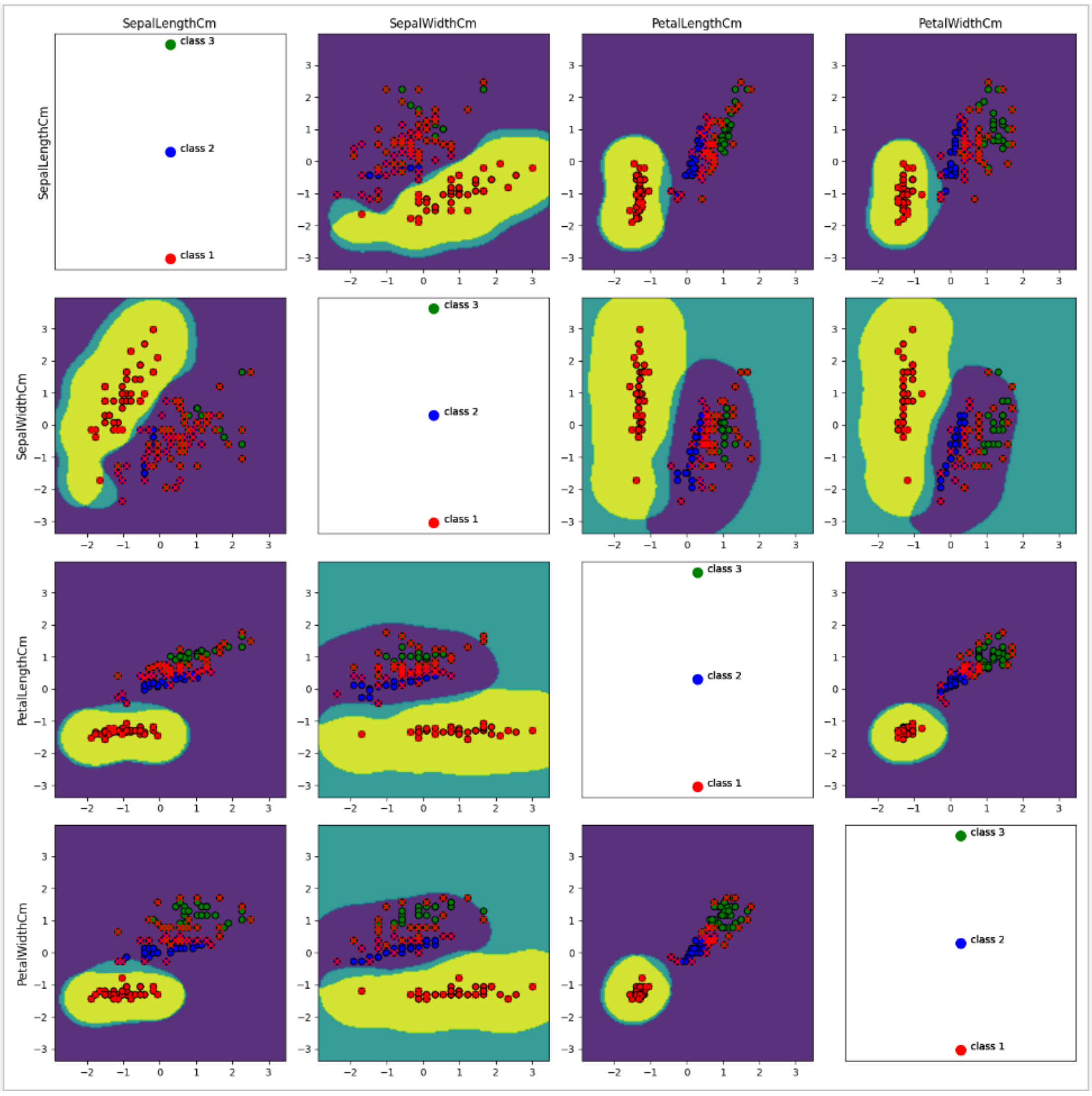


Figure 24: Iris dataset - predicted with SVM

In figure 24 is presented prediction of the Iris dataset using SVM.

- **Neural Network** - Here we firstly normalize the dataset, $X \in (0, 1)$.
  We transformed Y vector to one hot encoding. Since old Y = 1,2 or 3, current Y_i has 3 dimensions, an array of zeros and 1 on the index of position of the class, for example $OldY = 2$, current Y = [0,1,0], has 1 on the index 2.
  We created 5 different model, for each model we increased the hidden layers number, see sheet4.py

  We trained all models for 50 epochs, with batch size = 10, and learning rate = 0.1
  For each model we plotted the validation loss, validation accuracy (see 25) and ROC see 26.

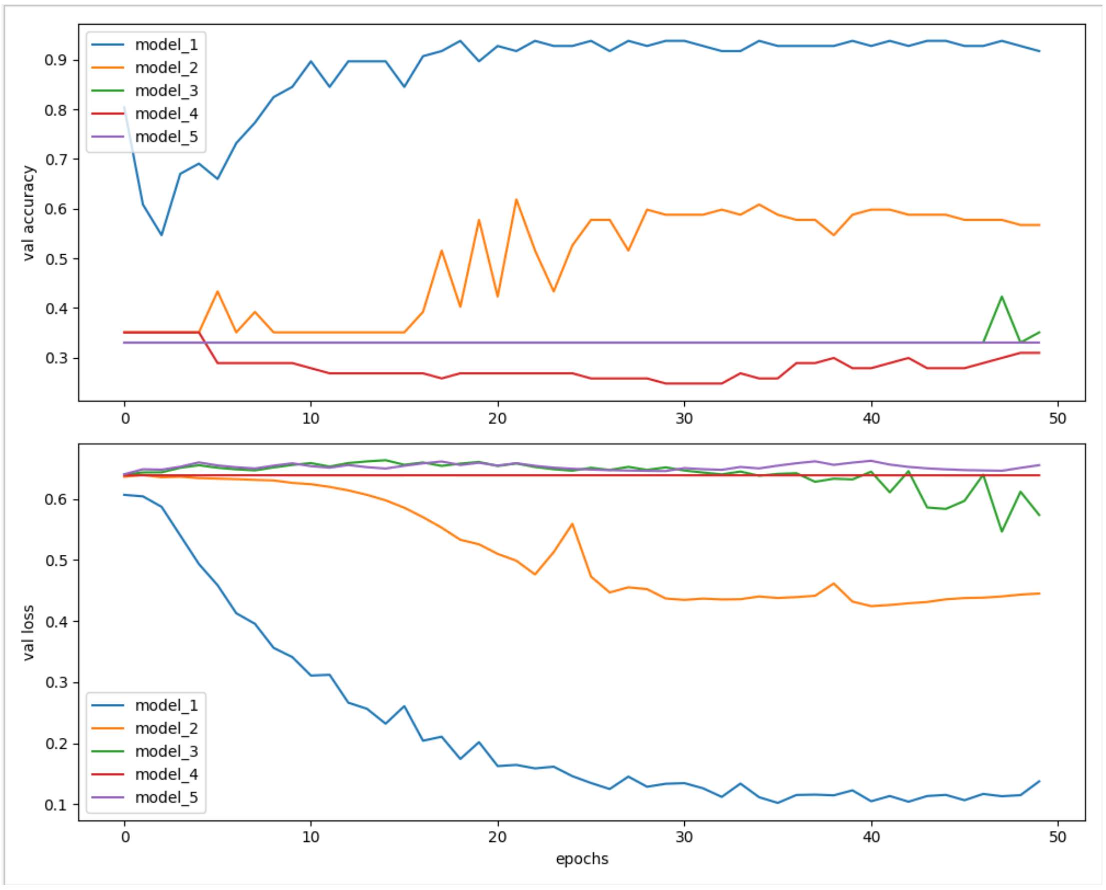


Figure 25: Neural Network models

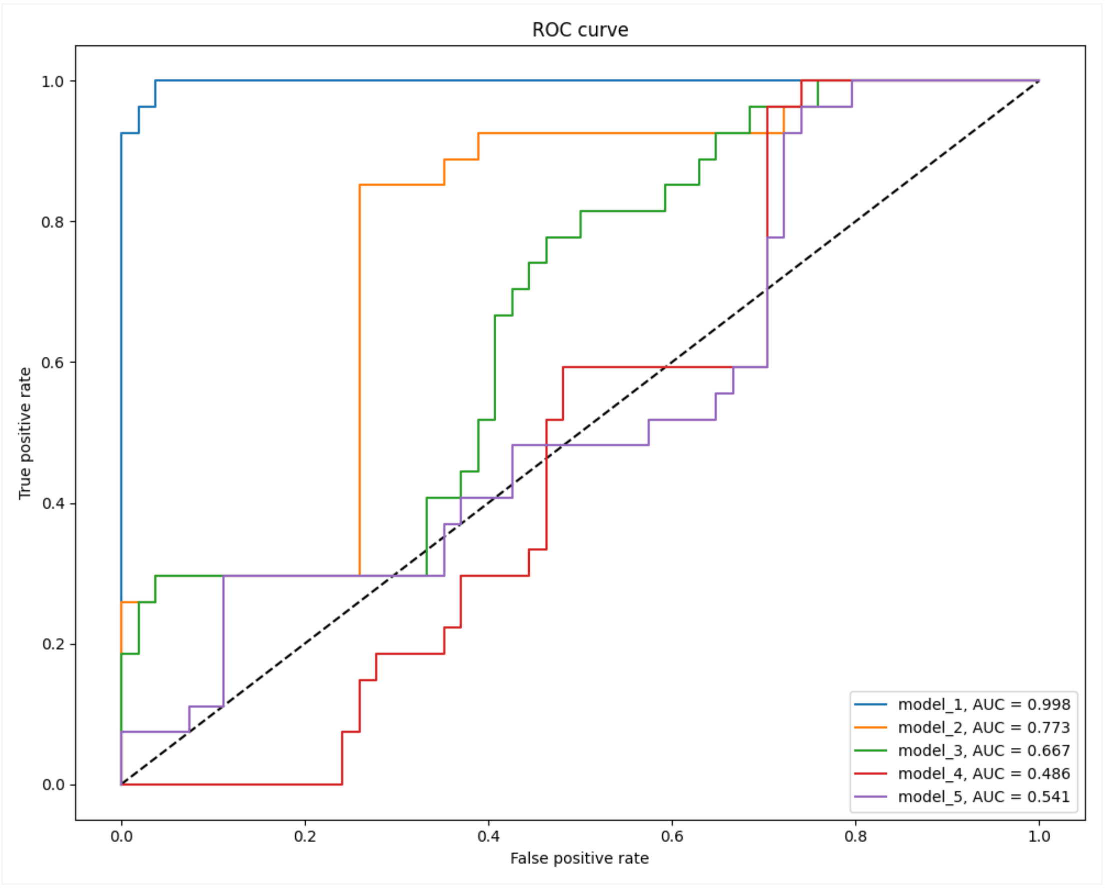


Figure 26: Neural Network AUC

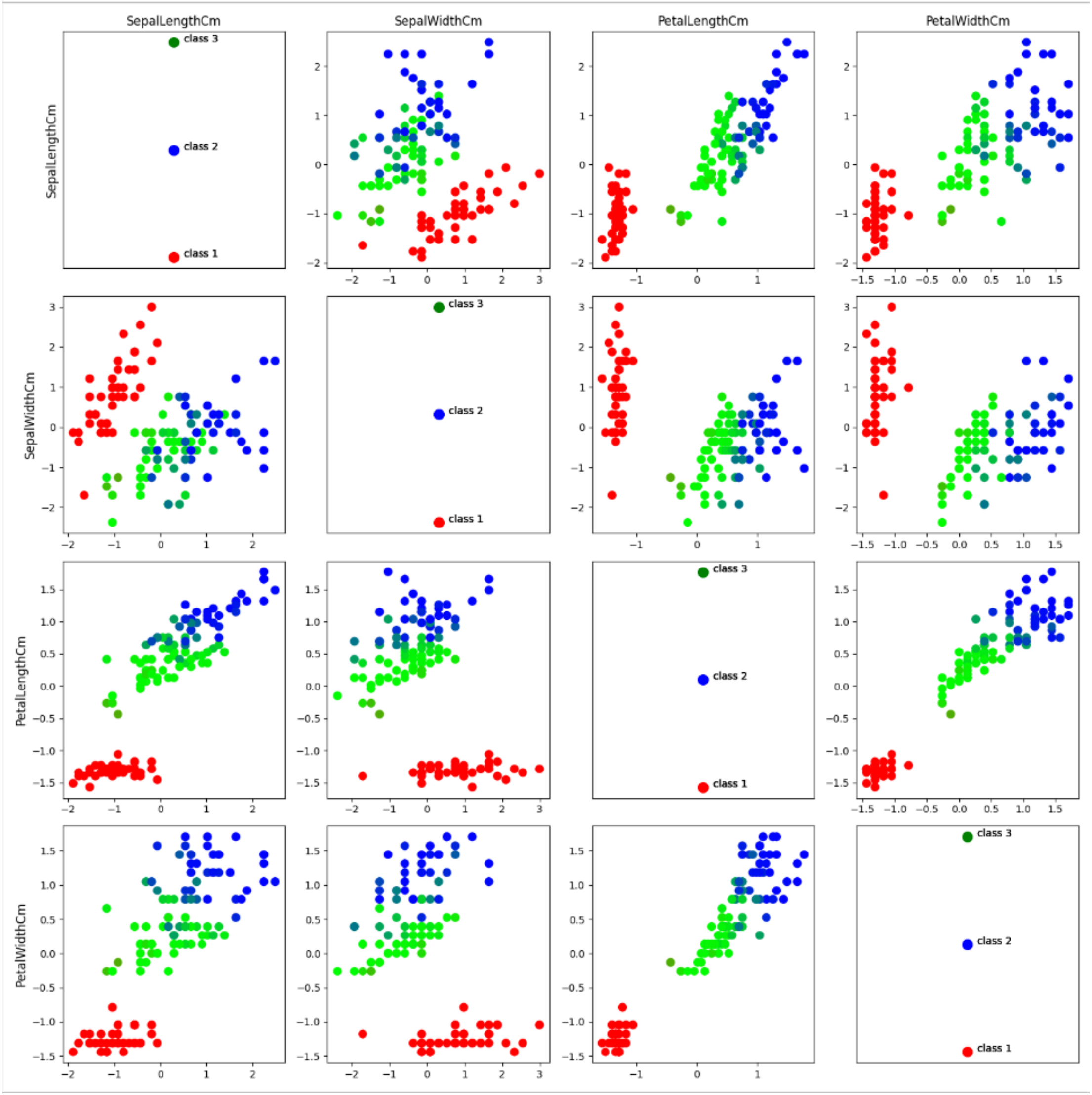


Figure 27: Iris dataset - predicted with NN

In figure 27 is presented prediction of the Iris dataset using the best model found above (model 1), from this plot we notice that clasess 2 and 3 aren't linearly separable.

Assignment 6 MLP and SVM on the MNIST dataset

In this assignment, we used the SVM and a MLP/neural network to classify as best as possible the digists 0-9 from the MNIST dataset. The MNIST dataset consists on a train set with 60.000 digits images and a test set with 10.000 digits images. Each image has $28x28 = 784$ pixels.

Since we are dealing with images of digits which tend to be sparse (many pixels are zero), we had to scale the training data so that each pixel $\in [0.1, 1]$ by multiplying each pixel with 0.99/255 and adding 0.1. Therefore, we avoid 0 values as inputs which might prevent some wights from being updated.

In the neural network case, we considered for computational reasons and available resources two models. The first model consists on

- 1 Input layer with 784 neurons
- 1 Hidden layer with 100 neurons
- 1 Output layer with 10 neurons

The second model consists on

- 1 Input layer with 784 neurons
- 2 Hidden layers with 100 neurons each

- 1 Output layer with 10 neurons

In the first model, there are $784 * 100 + 100 * 10 = 79.400$ weights being optimized. In the second model, there are $784 * 100 + 100 * 100 + 100 * 10 = 89.400$ weights being optimized. We can clearly observe that since we are using so many parameters to make a prediction, neural networks easily tend to over-fit the data. For this reason we used two regularization methods:

- Drop out, randomly setting some weights in each layer to zero controlled by the parameter p
- Weight decay, which is ridge regularization controlled by the parameter λ

Moreover, the learning rate has to be adjusted too. We used a *ReLu* activation function for all layers except for the last output layer which uses the $softmax$ activation function which produces values we can interpret as probabilities.

In order to determine the best parameters combination for both models, we performed 5 fold cross validation by using the same CV function as in Assignment 3. Because of computational reasons, we only used 1 repetition per fold. For both models, we considered the next parameters :

- $\lambda \in \left[0, 1^{-4}, 1^{-2}\right]$
- $p \in [0, 0.1, 0.5]$
- $learning\ rate \in [0.01, 0.1]$

After performing 5 fold CV, we found out that

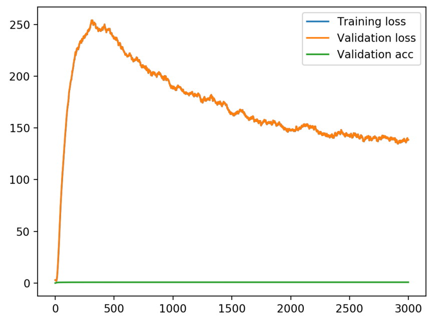
- Using no regularization techniques the neural network tends to over-fit (high validation error)
- The best parameter for the learning rate seems to be 0.1 which is a typical choice in literature
- The best weight decay value for λ seems to be 1^{-4}
- The best dropout value for p seems to be 0.1

Since we only performed CV using 1 repetition, we decided to plot the train-test loss and accuracy curves as well as the 100 weights vectors of the first layer for different cases. Nevertheless, we will see that the best test accuracy is achieved for the model with the hyper parameters the CV computed.

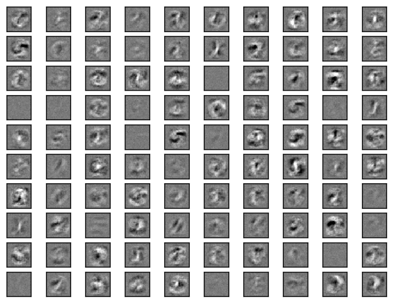
In order to optimize the weights, we used PyTorch built in optimizet SGD which uses stochastic gradient descent. In the same way, we used PyTorch built in function aurograd to compute the gradients.

In all cases, we used a 168 batch size. This means that the neural network updates the weights every 168 images. We used 3000 training steps. Therefore, the neural network is trained with $168 * 3000 = 504.000$ images that correspond to $504.000/60.000 = 8.4$ epochs. After training each model. we computed the accuracy on the test set which contains 10.000 images.

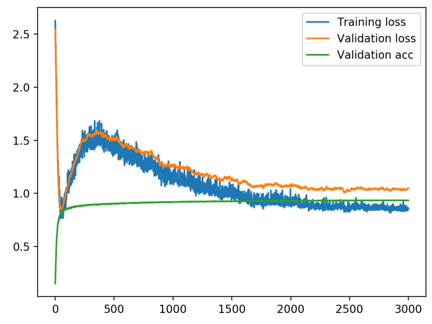
The next figures show the result we obtained for the first model (1 Input Layer + 1 Hidden Layer + 1 Output Layer)



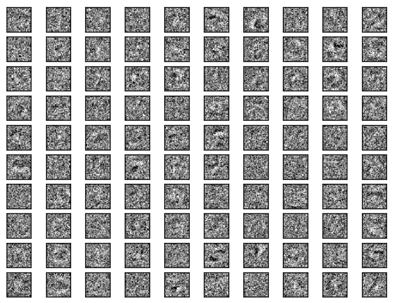
(a) $\lambda = 1^{-2}, p = 0, lr = 0.1$



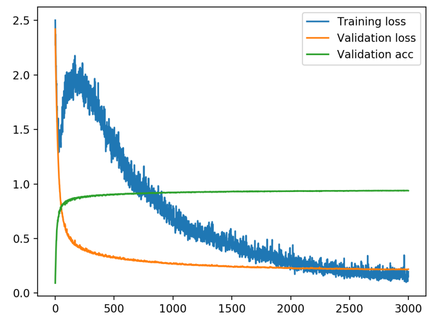
(b) $\lambda = 1^{-2}, p = 0, lr = 0.1$



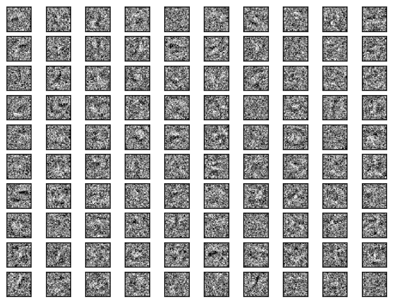
(c) $\lambda = 1^{-4}, p = 0, lr = 0.1$



(d) $\lambda = 1^{-4}, p = 0, lr = 0.1$



(e) $\lambda = 1^{-4}, p = 0.1, lr = 0.1$



(f) $\lambda = 1^{-4}, p = 0.1, lr = 0.1$

Figure 28: Train-Validation-Accuracy and the first 100 weight vectors of layer 1

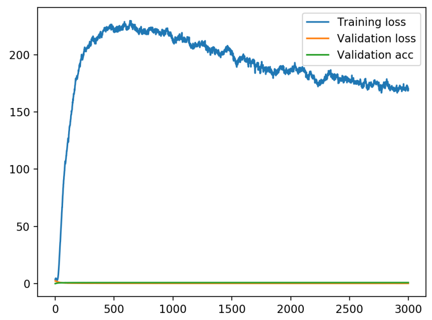
From figure 28 we can observe the following:

- When using a larger weight decay regularization $\lambda = 1^{-2}$, the validation accuracy tends to 0.93. (see Fig 28a) shows that the train error seems to be high while the validation error is low. This happens due to the fact that we are plotting the train error which includes the weight decay term with the highest λ of all considered cases. Nevertheless, the test accuracy is 92% and the test loss is 0.2787. Moreover, we can clearly recognize from Fig 28b that each neuron weights represent in some way a number. This is a similar result as when plotting the principal components after performing PCA on the MNIST dataset as we did in assignment 1.

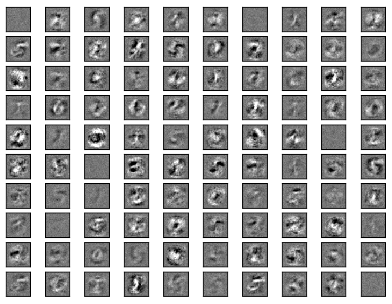
- When using a lower weight decay $\lambda = 1^{-4}$, we can observe from figure 28c that we have much lower train loss values due to the fact that λ has a lower value. Moreover, we recognize that both the train loss curve and specially the validation loss curve oscillate but still decrease. We believe this is a common result of using Stochastic gradient descent. The validation accuracy tends to 0.94 and we obtained a better test accuracy of 93.6% and test loss of 0.223. Furthermore, we can see from figure 29d that the weight vectors still represent numbers, but seem to be noisier.

- When using the same lower weight decay $\lambda = 1^{-4}$ and a dropout rate of $p = 0.1$, we obtained the results shown in figure 28e. Both the train and validation error decrease and the validation accuracy goes against 0.94. This model is the best of all ones since we obtained a test accuracy of 94.16% and a corresponding loss of 0.20. The weight vectors plot (see Fig 28f) shows a similar result as in the second scenario.

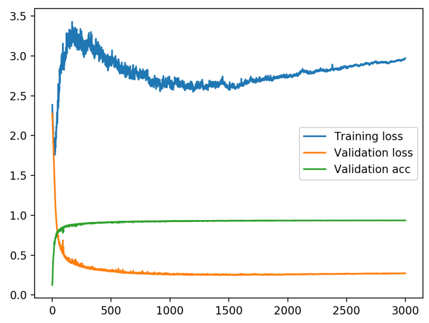
The next figure shows the result we obtained for the second model (1 Input Layer + 2 Hidden layers with each 100 Neurons + 1 Output layer with 10 neurons)



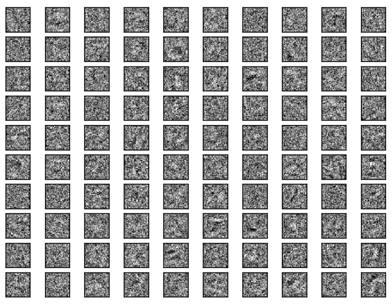
(a) $\lambda = 1^{-2}, p = 0, lr = 0.1$



(b) $\lambda = 1^{-2}, p = 0, lr = 0.1$



(c) $\lambda = 1^{-4}, p = 0, lr = 0.1$



(d) $\lambda = 1^{-4}, p = 0, lr = 0.1$

Figure 29: Train-Validation-Accuracy and the first 100 weight vectors of layer 1

From figure 29 we can observe the following:

- We obtained similar results e.g test accuracy and test loss as in the first model: 91.8% accuracy, 0.25 loss and 93% accuracy, 0.225 loss on both networks with $\lambda = 1^{-2}$ and $\lambda = 1^{-4}$ respectively. Therefore, adding a new layer which adds computation time did not result in a results improvement.

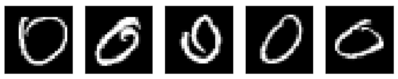
Next, we performed the same classification task on the MNIST data set by using the SVM algorithm. Since the SVM algorithm is built for binary classification tasks, we performed one vs rest classification. In the Mnist case, since we have 10 possible classes, we need to train 10 svm algorithms which each solve a binary classifcation problem. As we saw in the previous exercises, we need to find an optimum kernel, an optimum kernel parameter and an optimum regularization parameter C. As in the neural network case, we performed 5 fold cross validation to find the best hyperparameters for each digit model.

The 5-fold cross validation was performed with the following parameters

- $C \in [0.1, 0.01, 0.001]$
- Gaussian kernel and polynomial kernel
- Gaussian width $\in np.log(-2, 2, 5)$ and polynomial degree $\in [2, 3, 4]$

In general, the best parameter combination we ended up using for all digits was the polynomial kernen with degree 3 and a regularization $C = 0.001$. For each digit, we computed the test accuracy as well as the true positive rate. Moreover, we plotted for each digit five randomly choosen support vectors for the positive and negative class respectively.

Digit 0


(a) 5 randomly choosen Support vectors positive class

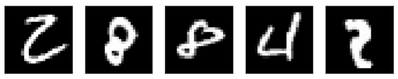
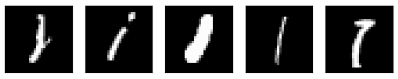

(b) 5 randomly choosen Support vectors negative class

Figure 30: Digit 0

- Train accuracy : 100%
- Test accuracy : 99.73%
- Test true positive rate: 98.46%

Digit 1


(a) 5 randomly chosen Support vectors positive class

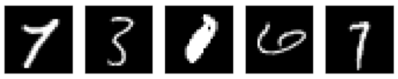
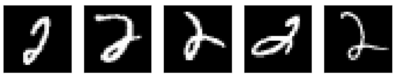

(b) 5 randomly chosen Support vectors negative class

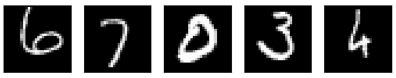
Figure 31: Digit 1

- Train accuracy : 99.7%
- Test accuracy : 99.8%
- Test true positive rate: 98.9%

Digit 2



(a) 5 randomly chosen Support vectors positive class

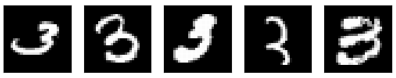


(b) 5 randomly chosen Support vectors negative class

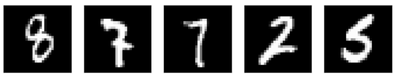
Figure 32: Digit 2

- Train accuracy : 99.9%
- Test accuracy : 99.5%
- Test true positive rate: 97.2%

Digit 3



(a) 5 randomly chosen Support vectors positive class

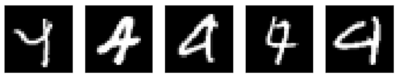


(b) 5 randomly chosen Support vectors negative class

Figure 33: Digit 3

- Train accuracy : 99.9%
- Test accuracy : 99.5%
- Test true positive rate: 97.4%

Digit 4


(a) 5 randomly chosen Support vectors positive class

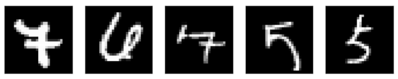
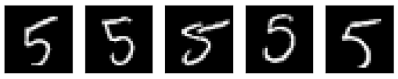

(b) 5 randomly chosen Support vectors negative class

Figure 34: Digit 4

- Train accuracy : 99.9%
- Test accuracy : 99.6%
- Test true positive rate: 97.9%

Digit 5


(a) 5 randomly chosen Support vectors positive class

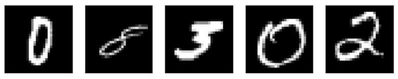
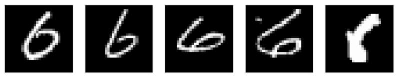

(b) 5 randomly chosen Support vectors negative class

Figure 35: Digit 5

- Train accuracy : 99.9%
- Test accuracy : 99.7%
- Test true positive rate: 97.9%

Digit 6


(a) 5 randomly chosen Support vectors positive class

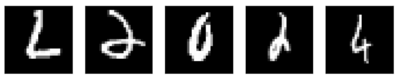
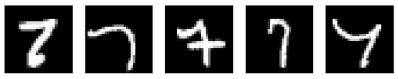

(b) 5 randomly chosen Support vectors negative class

Figure 36: Digit 6

- Train accuracy : 99.9%
- Test accuracy : 99.6%
- Test true positive rate: 97.5%

Digit 7


(a) 5 randomly chosen Support vectors positive class

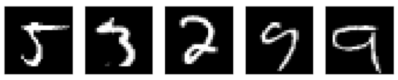
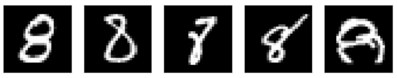

(b) 5 randomly chosen Support vectors negative class

Figure 37: Digit 7

- Train accuracy : 99.9%
- Test accuracy : 99.5%
- Test true positive rate: 96.7%

Digit 8


(a) 5 randomly chosen Support vectors positive class

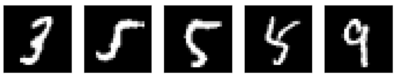
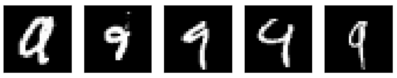

(b) 5 randomly chosen Support vectors negative class

Figure 38: Digit 8

- Train accuracy : 99.9%
- Test accuracy : 99.3%
- Test true positive rate: 96.5%

Digit 9


(a) 5 randomly chosen Support vectors positive class

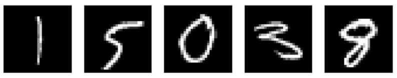

(b) 5 randomly chosen Support vectors negative class

Figure 39: Digit 9

- Train accuracy : 99.9%
- Test accuracy : 99.3%
- Test true positive rate: 95.8%

One vs Rest Classification

After training the correspondent 10 SVM models, we performed ones vs rest classification on the test data set. For each data point, we computed its correspondent output from each digit model and classified the data point according to the highest model value. Therefore, we have a $Y_{pred} \in R^{n \times 10}$ matrix and just need to look for the maximum position along each row.

The main problem with the one vs rest classification approach is data imbalance. In our case, since the training Mnist data set consists of 6000 images per digit, each model is trained with 90% negative class data points and 10% positive class data points. We see there is a clearly imbalance in each models binary training data. Consequently, a binary model can always predict -1 for all training or test inputs and still get 90% accuracy. For this reason, we decided to compute the true positive rate which gives a more vision on how good the model is predicting the positive examples. This data imbalance does not happen when using one vs one classification since we would train models for e.g digit 1 vs digit 0 and have a 50% vs 50% positive-negative examples distribution. On the other side, we would en up training 45 SVM models which is much more computationally expensive.