Homework 4
601.482/682 Deep Learning
Spring 2022
Ting he

February 23, 2022

Due 11:59pm on March 9, 2022
Please submit 1) a single zip file containing your Jupyter Notebook and PDF of your Jupyter Notebook to "Homework 4 - Notebook" and 2) your written report (LaTeX generated PDF) to "Homework 4 - Report" on Gradescope

1. We learnt that a two-layer MLP can model polygonal decision boundaries. You are given data sampled from a "two-pentagons" decision boundary. Data points within the pentagons are considered "true" (labeled as 1) while data points outside are considered "false" (labeled as 0). The data can be downloaded here: https://piazza.com/class_profile/get_resource/kwo1bkotxk1ja/kzzzcr9hjax2i3.

   Please load the data via `np.load()` function and check that you obtain a numpy array with a shape of $(60000, 3)$. The first two colomns are x, y coordinates and the third one represent the labels.

   The vertices of the 2 polygons are (500, 1000), (300, 800), (400, 600), (600, 600), (700, 800) and (500, 600), (100, 400), (300, 200), (700, 200), (900, 400). We have provided some visualization functions in the Jupyter notebook. The decision boundary can be modeled with a total of $2 \times 5 + 2 + 1$ neurons with threshold activation functions.

   (a) We have set up an MLP with threshold activation by analytically obtaining weights that can model the above decision boundary.

      i. Implement the "AND gate" and "OR gate" to predict all points within the 2 polygons as the positive class. Use the `predict_output_v1()` to test your implementations. Attach the visualization to your report.
      the plot is shown in attached pic 1ai visualization of decision boundary of AND OR gate

      ii. Modify the code in `predict_output_v2()` to predict only the points in polygon 1 as the positive class. Attach the visualization to your report. *Note: You should only need to update application of the gates themselves.*
      the plot is shown in the attached pic 1aii visualization of decision boundary

   Note: We reverse-engineered the weights from the known decision boundaries but in the real world, we do not know the decision boundaries. In a more realistic scenario, we would thus need to discover (learn) the decision boundaries from training data which we will explore next.

   (b) Build an MLP using sigmoid activation functions to replace the "AND/OR" gates. You are required to use PyTorch for this question and all remaining parts of the assignment. We **strongly recommend using Google Colaboratory** for ease of debugging and modeling flexibility. Whenever we provide code in subsequent assignments, it will be optimized for this setup.[1]

   Model hyperparameters:

[1] https://pytorch.org/. If you have not programmed with PyTorch before, give yourself some time to go

i. Use 2 hidden layers, and 1 output layer with binary-cross-entropy loss [2]

ii. Use 10 nodes in the first hidden layer, 2 nodes in the second hidden layer, and 1 output node (binary classification), which mirrors the capacity of the model in 1a)

iii. Initialize the weights using Xavier Initialization

iv. You may use any combination of batch size, training epochs, and learning rate. (Note that a larger batch size typically corresponds to a larger learning rate)

v. Do not use any regularization for this problem

Use gradient descent to optimize the parameters. You should expect an average accuracy greater than 90%. **Please use the first** 50000 **points as your training set and the remaining** 10000 **as your test set. You may also want to normalize (i.e. center and scale) the data before training.**

TODO:

i. Train your model using 5 random seeds and report the mean and standard deviation of the train and test accuracy at the end of 500 epochs. Attach the plots of training loss and testing accuracy change over epochs.
after applying 5 random seeds, the mean and standard deviation of train accuracy are: 94.9 (0.79)
after applying 5 random seeds, the mean and standard deviation of test accuracy are: 94.9 (1.25)
training loss and test accuracy plots are shown in the attached pic 1bi training loss and accuracy plots

ii. In your report, attach the visualization of the test set prediction (i.e. polygon figure). **the polygon test plot is shown in the attached 1bii polygon plot**

iii. Brief discussion. Are you able to find a solution that performs almost as good as the "manual" solution in (a)? **Answer: by using MLP with 2 hidden layers, we didn't get a prefect decision boundary as question (a), the algorithm tends to determine it as a convex diagram instead of 2 polygon diagrams. Since the this MLP model is simple, it can recognize this complex diagrams.**

(c) Build an MLP with a larger capacity (increase the depth and width). What do you expect to happen to your model's ability to learn the appropriate decision boundaries when you increase the depth? Increase the width? *Note: Since this is a toy dataset, you might not actually observe differences in performance compared to your 1b solution. However, you should be able to reason about expected behavior based on our lecture discussions.*

i. Like before, train your model using 5 random seeds and report the mean and standard deviation of the train and test accuracy at the end of 500 epochs.
after applying 5 random seeds, the mean and standard deviation of train accuracy are: 96.4 (1.53)
after applying 5 random seeds, the mean and standard deviation of test accuracy are: 96.6 (1.17)

ii. Report what depth and width you used. Briefly discuss what you notice about your model performance as you change these architecture decisions. *Hint: Think about our discussions from lecture regarding model capacity.*
I chose 4 hidden layers with 20 neurons in first hidden layer, 15 neurons in second hidden layer, 10 neurons in third hidden layer and 5 neurons in fourth hidden layer
I got a better decision boundary than the previous question by using this

through some online tutorials. There are many on the internet so its hard to recommend a specific one, it would really depend on your level of comfort. You may also find it helpful to review the PyTorch resources on Piazza (https://piazza.com/jhu/spring2022/cs482682/resources).

[2] See discussion here

96.6

architecture but still this is not good as question (a). The test accuracy is xx which is higher than previous question of xx 99.9 .

iii. Attach the plots of training loss and testing accuracy change over epochs.
the training loss and testing accuracy plots are shown in the pic 1ciii training loss and testing accuracy plot

iv. Attach the visualization of the prediction for the test set (i.e. polygon figure).
the prediction plot based on test data is shown in the pic 1civ visualization plot

(d) In part (b) we had fixed the hyperparameter choices, but in part (c) you adjusted those choices for the MLP in terms of depth and width of the network. Would you consider your test results to be valid and generalizable? Please briefly discuss.
i don't think the test results are valid seems we used modified hyperparamter on test dataset and report it as our accuracy of model. We should seperate a new data in order to report accuracy of model. Also, it might not be generalizable as well seems we didn't add any regulization terms such as L1/L2 or drop out method

2. In this problem, we will explore convolutional neural networks (CNNs). We will be working with the FashionMNIST dataset.[3] This dataset only has a train-test split, therefore we will be using the last 10000 training instances as the validation set. Again, we please use PyTorch[4] for this assignment. Because training times can be quite long, you *do not* need to train your model with multiple random seeds — 1 is enough, but of course feel free to expand if you have time and are confident in your model. You should try to train your models for at least 50 epochs.

(a) Design a small CNN (e.g. 3-4 hidden layers) using the tools we learnt in class such as

i. convolution and pooling layers

ii. activation functions (other than sigmoid)

TODO:

i. **Briefly** describe your architecture in your report and explain your design choices (e.g. I used ReLU activation because...) **Please design the network architecture by yourself. This is not about performance.**
I chose the architecture with 1) convolution layer + Relu 2) another convolution layer + Relu + pooling layer 3) fully connected linear layer 4) fully connected linear layer. I chose Relu instead of sigmoid since this is more computationally efficient and converges much faster. The reason for pooling layer at second hidden layers since i want to ignore some features to make the representations smaller and more manageable. And the reason of not adding in the first layers since i think the basic representations are important by missing some at first layers may cause us to miss some important information. I chose both fully connected linear layer in third and final layers seems it can learn better than a combination of convolution layer + fully connected layer

ii. Report the train, validation, and test accuracy of your best model (e.g. if the best model was obtained at epoch 50, report the train, validation, and test accuracy at epoch 50).
At epoch 50, train accuracy is 95.78; validation accuracy is 90.19 and test accuracy is 90.02

iii. Attach the plots of training loss and validation accuracy change over epochs.
the plots are shown in 2aiii

[3]The full FashionMNIST dataset can be downloaded from the official website here: https://github.com/zalandoresearch/fashion-mnist. We provide starter code for loading and splitting the dataset in the notebook.
[4]https://pytorch.org/get-started/locally/

Note: If you achieve more than 85% accuracy on the test set, move on to the next subproblem.

(b) Now, try to improve your model's performance by including additional architecture elements. You should implement *at least two* of the following:

i. dropout
ii. batch normalization
iii. data augmentation
iv. different optimizers (other than vanilla stochastic gradient descent)

Try to improve your accuracy over the model in 2(a). Your new model should perform at least as well as your previous model.
TODO:

i. Describe your new model additions and explain your design choices (e.g. I added dropout because...).
I added drop out of third fully connected linear layer seems in the fully connected linear layer we have more parameters than convolution layer and also in the higher layer we can mask some parameters out to get a more generalized result by adding this regularization. I also added batch normalization at first and second convolution neuron nets to eliminate covariate shift.

ii. Report the train, validation, test accuracy of your best model.
At epoch 50, I got training accuracy at 95.9, validation accuracy at 89.5 and test accuracy at 89.5 which the validation accuracy and test accuracy are slightly less than previous question. My guess is that previous model is good enough, so there is not many room for improvements and seems we only run the model once, the result might not be stable as previous one using 5 random seed so they are slightly lower even with adding more features.

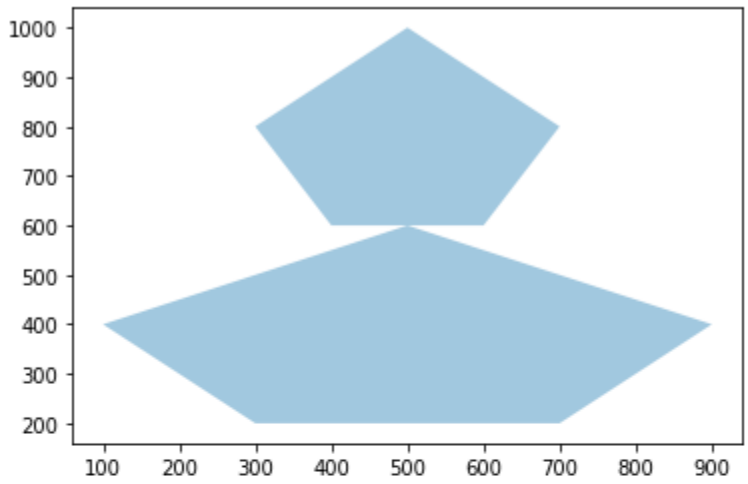
iii. Attach the plots of training loss and validation accuracy change over epochs.
The plot was shown in 2biii

(c) Search the internet for models that others have built on FashionMNIST. This can be from research papers or even something like Kaggle. Briefly describe one technique or architecture decision which you found interesting (remember to cite your sources). It can be an extension of an existing method, and you don't need to cover the paper's results for that method. As a rough guide, anything from approximately 50-100 words of prose is sufficient.

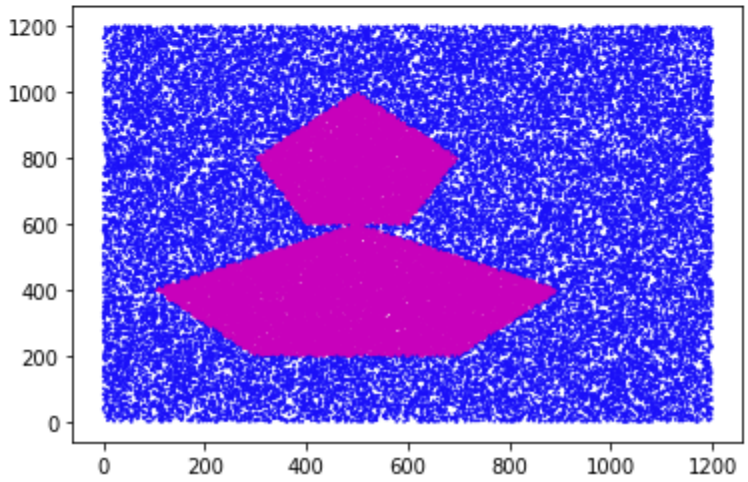
i found a kaggle solution with accuracy of 0.93. It used data augmentation method to add variations of training dataset plus 4 convolution hidden layers with batch normalization and 2 fully connected linear layers. original link: https://www.kaggle.com/fuzzywizard/fashion-mnist-cnn-keras-accuracy-93/notebook.

Pic 1ai visualization

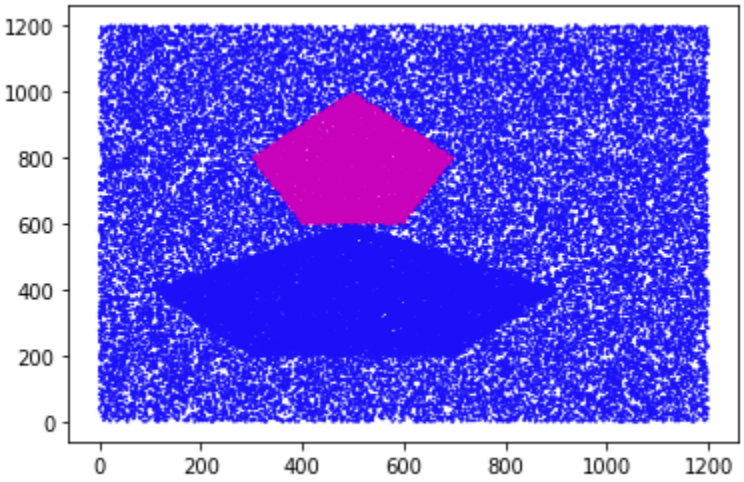
```
[ ]  1  visualize_polygons(p0,p1)
```



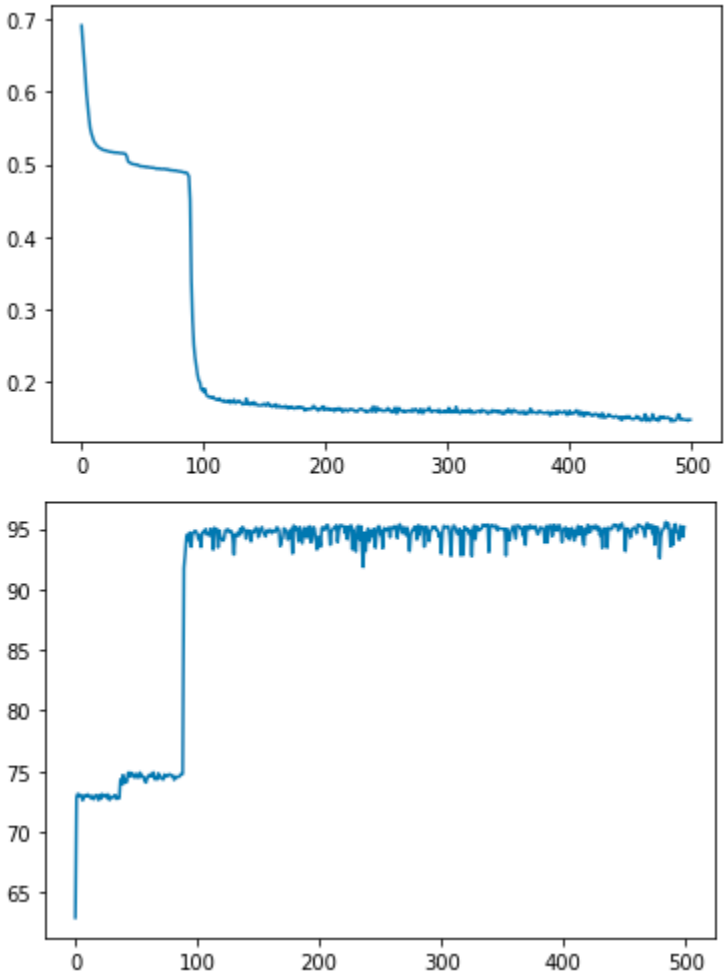

```
[ ]  1  visualize_datapoints(X, Y)
```



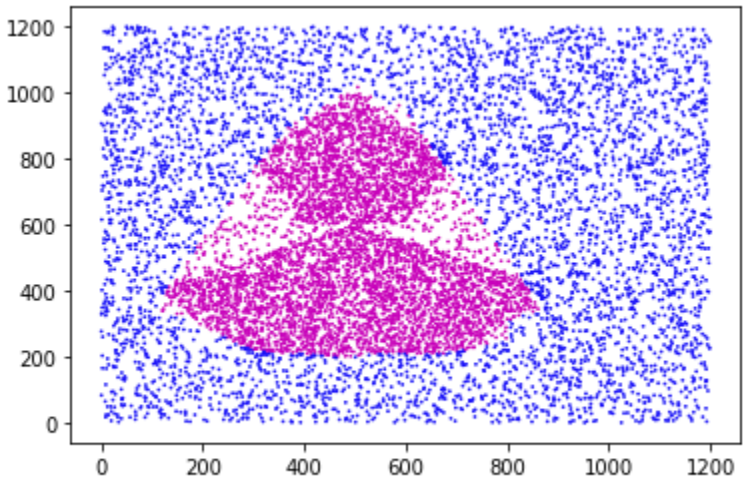
Pic 1aii visualization



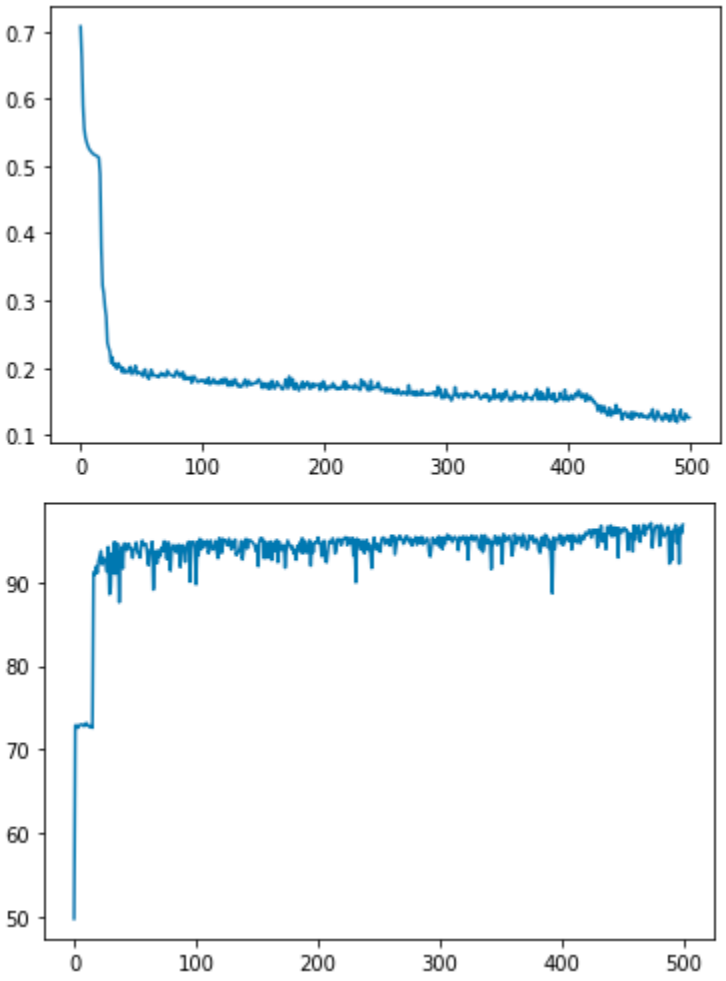
Pic 1bi


0.7
0.6
0.5
0.4
0.3
0.2
0
100
200
300
400
500
95
90
85
80
75
70
65
0
100
200
300
400
500

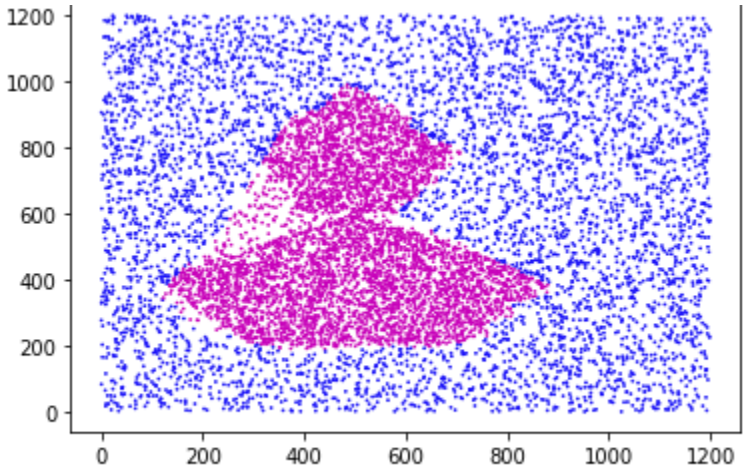
Pic 1bii



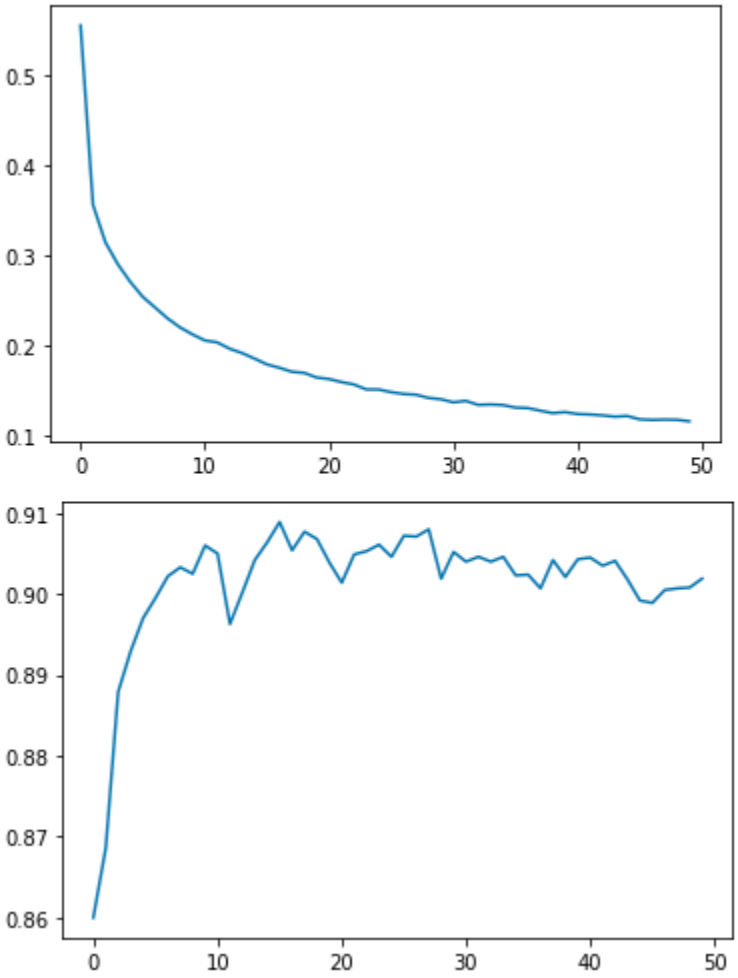
Pic 1ciii


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
100
200
300
400
500
90
80
70
60
50
0
100
200
300
400
500

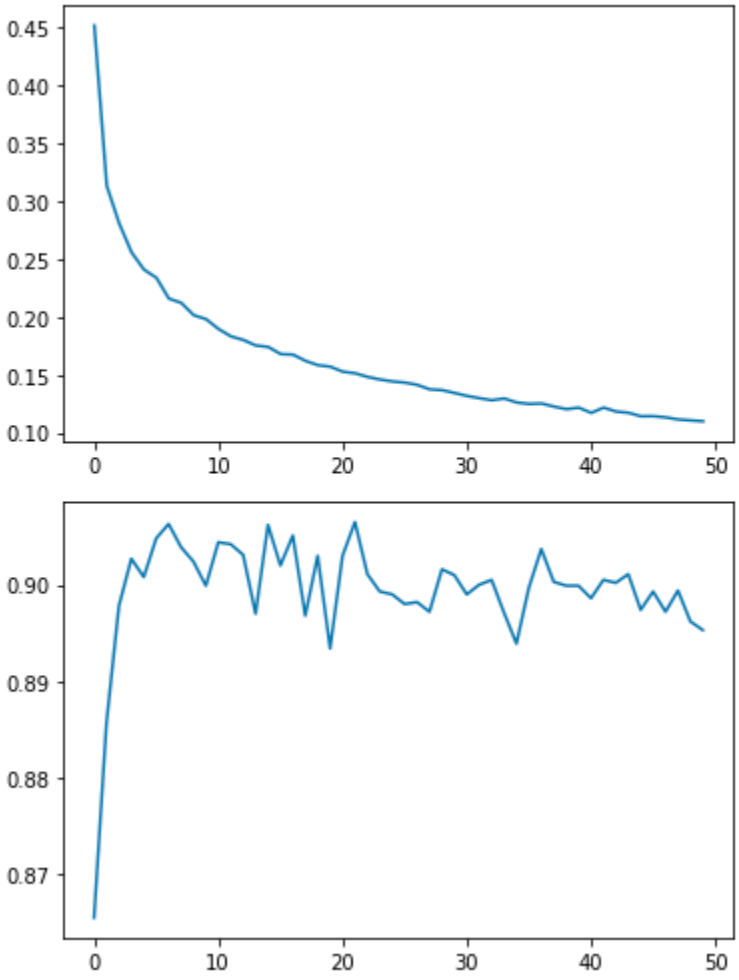
Pic 1civ



Pic 2aiii


0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50
0.91
0.90
0.89
0.88
0.87
0.86
0
10
20
30
40
50

Pic 2biii


0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
10
20
30
40
50
0.90
0.89
0.88
0.87
0
10
20
30
40
50