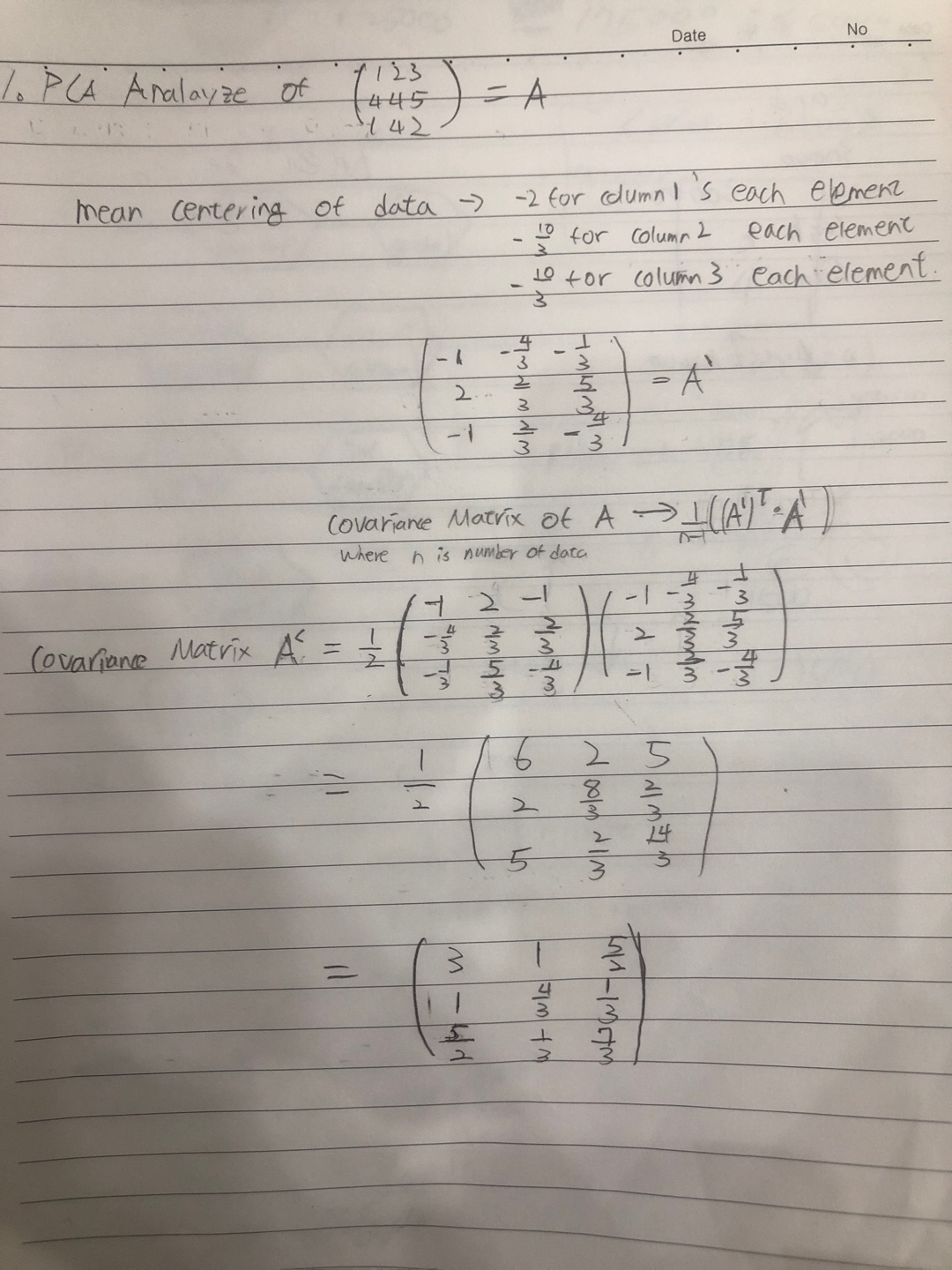
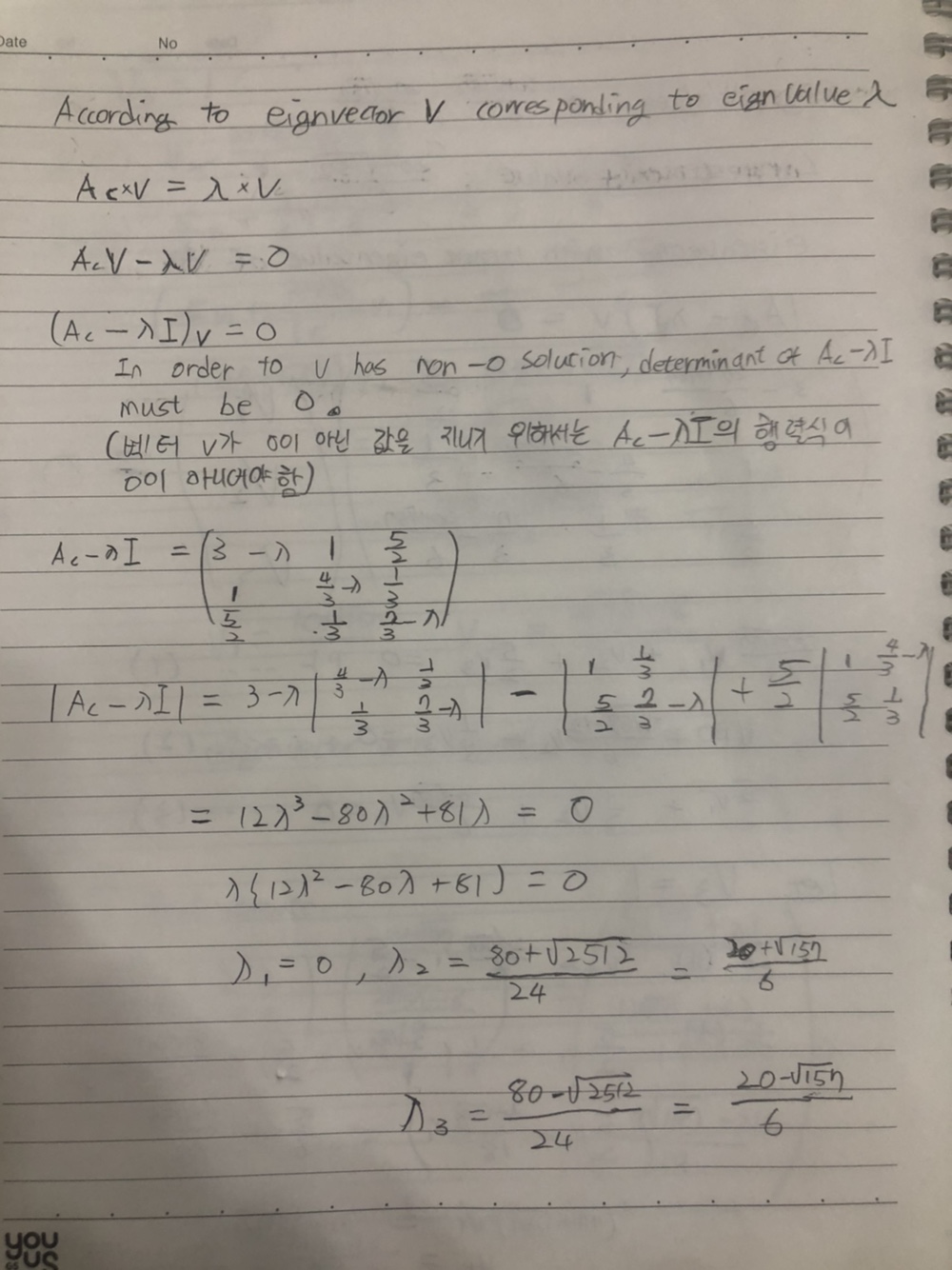
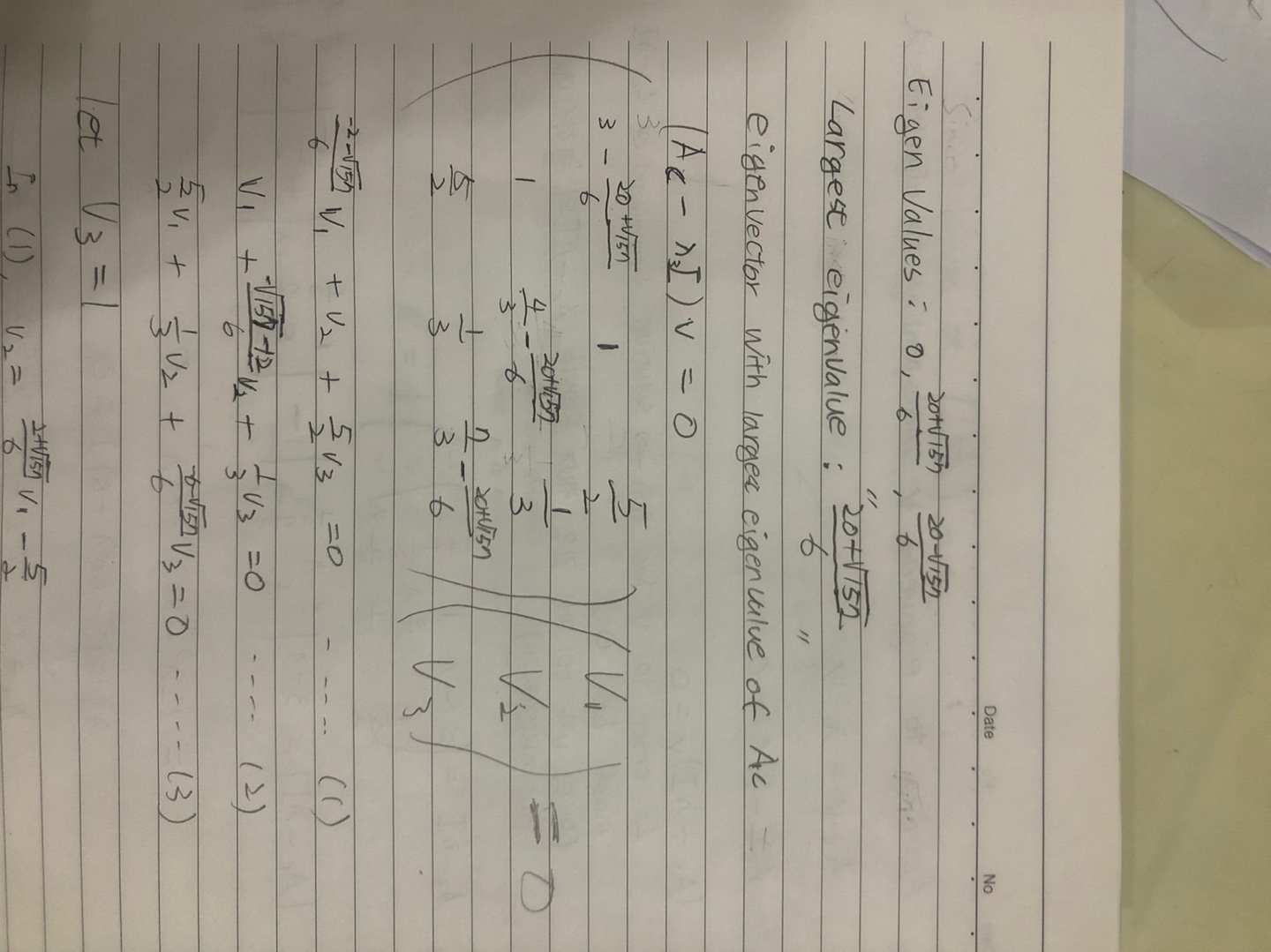
**HW #1**

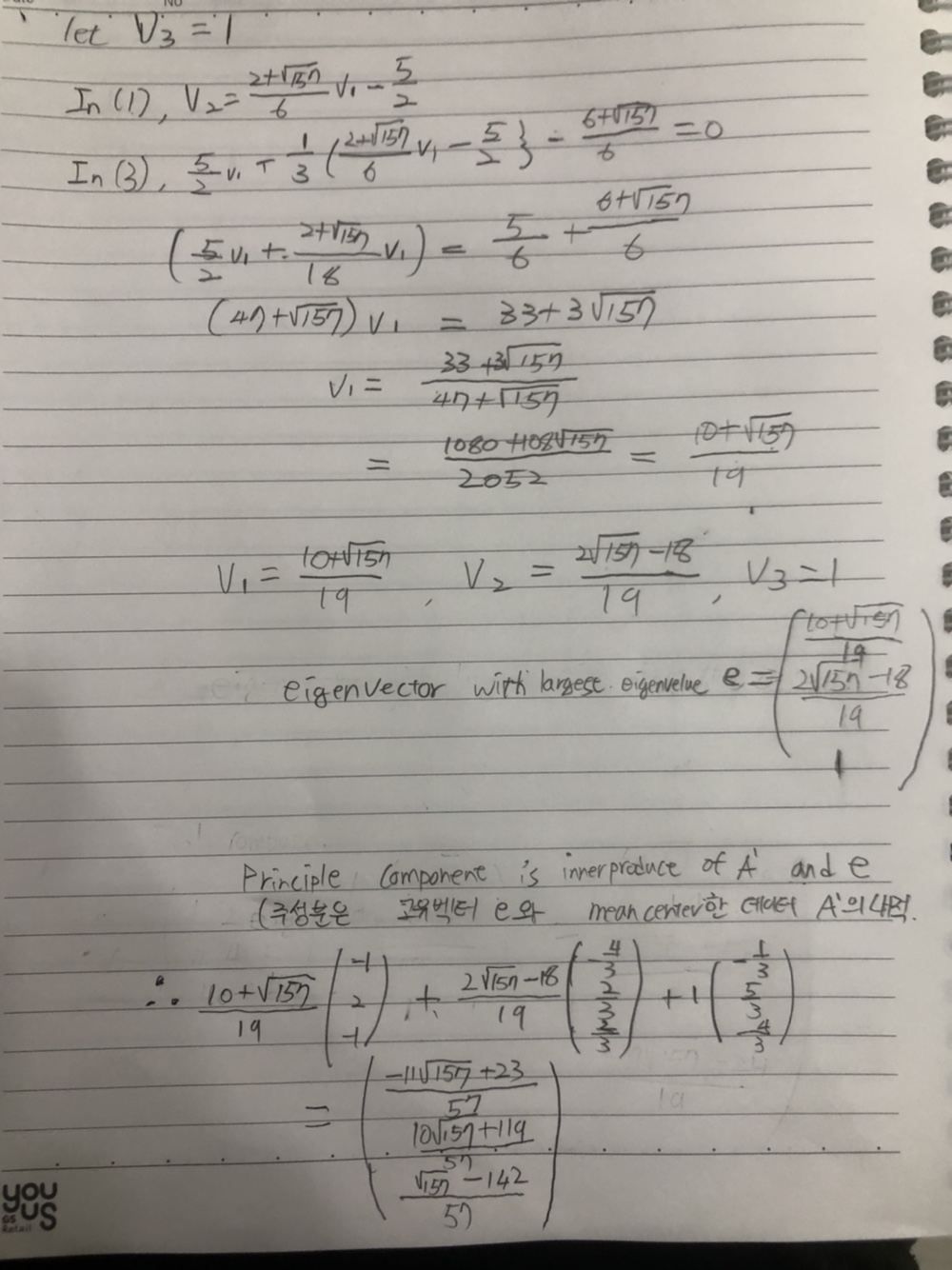
1. Manually do the principal component analysis of the following 3x3 data matrix. Each row means a data vector. Use the eigenvector with the largest eigenvalue to transform the data matrix. In other words, you PCA result should return 3x1 matrix. (30pts)

[[1,2,3]  
[4,4,5]  
[1,4,2]]



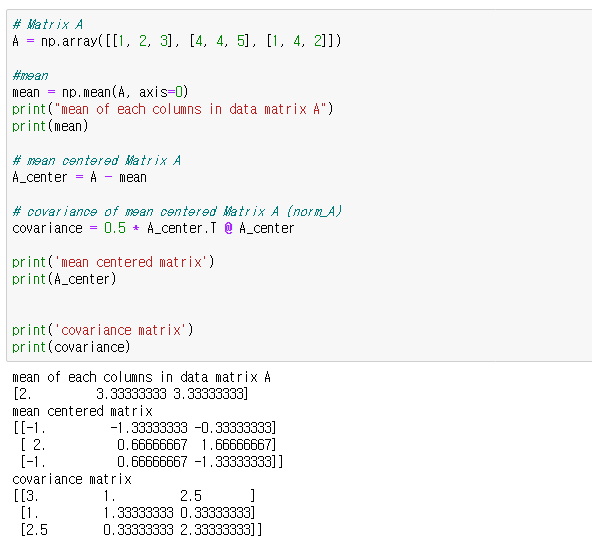




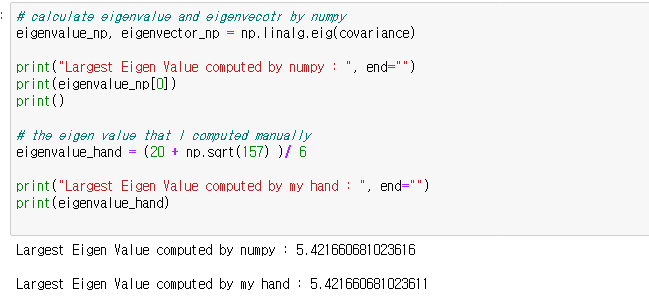


For question 1, I checked my calculation with python numpy module. However, python yielded different output values compare to my hand computation. I think it is because the eigenvalues are irrational number, which means python floating number computation is limited. Also supposition that v3 = 1 yields different eigenvector(위의 풀이에서 연립방정식 풀 때 v3=1이라 가정한 것. 고유값에 해당하는 고유벡터는 여러 개가 존재 가능.) Just in case, I attach additional solve calculating with python in following pages.

1. Calculation of mean centered matrix and covariance matrix by numpy

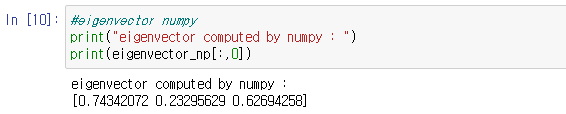


1. Calculation of eigenvalue by numpy. (and comparison with eigenvalue computed manually)

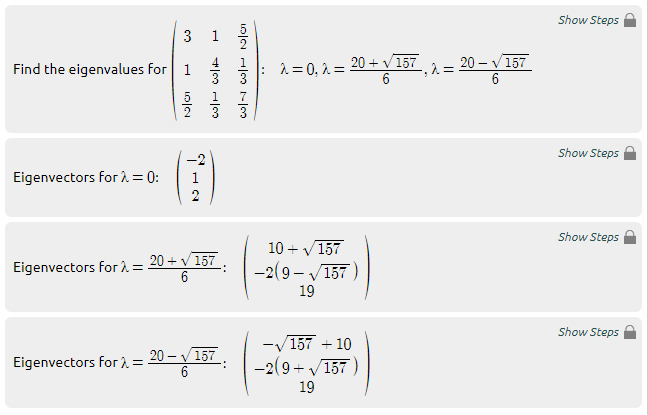


Those two values are almost same.

1. Calculation of eigenvector by numpy



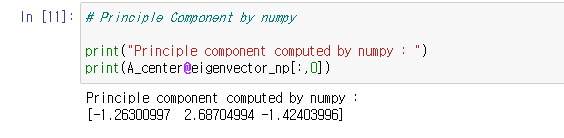
1. Calculation of eigenvector using online calculators



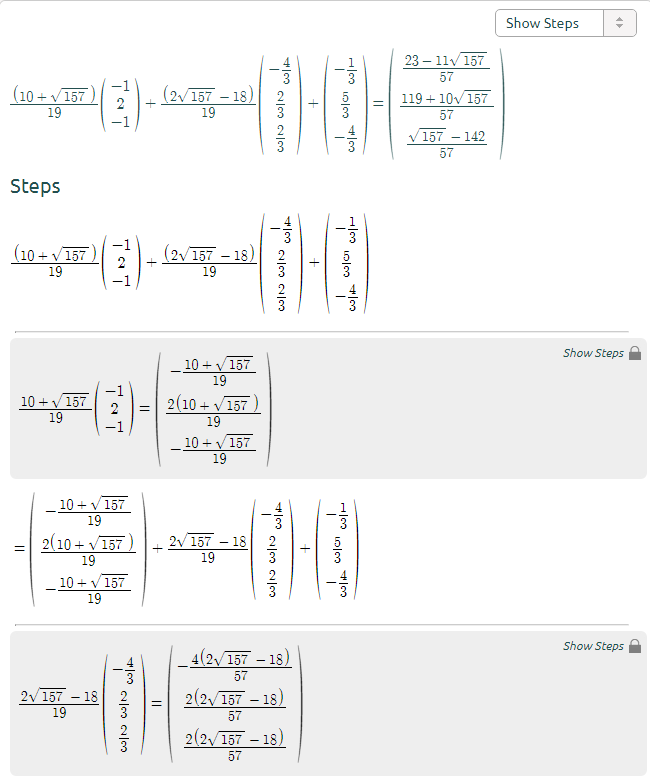
(The result of online calculator is same as my manual calculation)

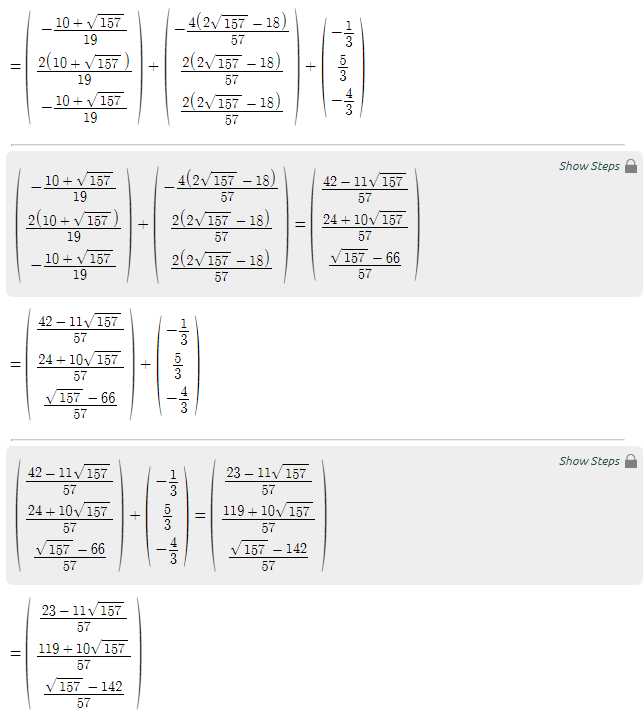
(https://www.symbolab.com/solver/matrix-eigenvectors-calculator/eigenvectors%20%5Cbegin%7Bpmatrix%7D3%261%26%5Cfrac%7B5%7D%7B2%7D%5C%5C%201%26%5Cfrac%7B4%7D%7B3%7D%26%5Cfrac%7B1%7D%7B3%7D%5C%5C%20%5Cfrac%7B5%7D%7B2%7D%26%5Cfrac%7B1%7D%7B3%7D%26%5Cfrac%7B7%7D%7B3%7D%5Cend%7Bpmatrix%7D)

1. Numpy PCA calculation



1. Online Calculator PCA calculation





<https://www.symbolab.com/solver/matrix-add-subtract-calculator/%5Cfrac%7B%5Cleft(10%2B%5Csqrt%7B157%7D%5Cright)%7D%7B19%7D%5Ccdot%5Cbegin%7Bpmatrix%7D-1%5C%5C%20%202%5C%5C%20%20-1%5Cend%7Bpmatrix%7D%20%2B%20%5Cfrac%7B%5Cleft(2%5Csqrt%7B157%7D%20-18%5Cright)%7D%7B19%7D%5Ccdot%5Cbegin%7Bpmatrix%7D-%5Cfrac%7B4%7D%7B3%7D%5C%5C%20%20%5Cfrac%7B2%7D%7B3%7D%5C%5C%20%20%5Cfrac%7B2%7D%7B3%7D%5Cend%7Bpmatrix%7D%20%2B%20%5Cbegin%7Bpmatrix%7D-%5Cfrac%7B1%7D%7B3%7D%5C%5C%20%20%5Cfrac%7B5%7D%7B3%7D%5C%5C%20%20-%5Cfrac%7B4%7D%7B3%7D%5Cend%7Bpmatrix%7D>

1. In Google Colab, please upload the attached Jupyter notebook and answer the following questions.
   1. Write a complete set of neural network parameters in the attached dense network. How many parameters do you see? (10pts)

Number of connections between the first and second layer: 2 × 32 = 64

Number of connections between the second and third layer: 32 × 6 = 192

Number of connections between the third and fourth layer: 6 × 3 = 18

Number of connections between the bias of the first layer and the neurons of the second layer (except bias of the second layer): 1 × 32 =32

Number of connections between the bias of the second layer and the neurons of the third layer: 1 × 6 = 6

Number of connections between the bias of the third layer and the neurons of the fourth layer: 1 × 3 = 3

Total number of parameters : 64 + 192 + 18 + 32 + 6 + 3 = 315

* 1. What are hyperparameters in the attached code? (10pts)

Number of hidden layers and units : 2 hidden layers( 32, 6 units each)

Learning rate : 0.001

Numbers of Epochs : 800

Activation function : ReLU

Batch size : 50

Loss function criterion: CrossEntropyLoss

Initializing weight and bias : (optimizer settings)

* 1. Why does the final linear layer have an output of size 3? (10pts)

Because this model uses supervised dataset which classified into three output classes(underdone, soft, hard)

* 1. What is the role of ReLU in the codes? (10pts)

ReLU is one of the activation functions. At the end of each neurons of the models, ReLU is attached. ReLU normalizes the output of each neuron in scale of between 1 and 0. It determines whether the output of a neuron should be activated or not(turned on and off) by considering aspects of model`s rule or certain threshold.

Moreover, ReLU is classified as a Non-Linear Activation function. It allows backpropagation because it has a derivative function. Also, it allows putting multiple layers between input and output.

Because ReLU is comparably simpler than other non-linear activation functions it has advantage of faster and efficient computation speed.

* 1. Try at least three modifications of the model by changing i) the number of layers, ii) the output sizes (dimensionalities) of layers, and/or iii) utilizing other non-linear activations instead of ReLU. For these three modifications, you can decide how and what to change from the model. For each modification, you should train for 800 epochs and take a note of the best test accuracy (not the accuracy from the last epoch but the best accuracy you have observed during the entire training process). Repeat this 10 times and report the average score of those 10 best accuracies. Training includes some randomness, and you may see different results every training. Therefore, repeating multiple times and reporting their mean best accuracy is required. You should report in the following format. I do not require accuracy improvements for HW1. (30pts)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Modifications that I made | The best accuracies for each 10 trials | average score of 10 best accuracies | Name of .ipnyb file |
| Original | No modifications | [0.9, 0.9, 0.9, 0.9, 0.9,0.9,0.9,0.9,0.9,0.9] | 90% | HW1-original.ipynb |
| Mod#1 | Add two more hidden layers.       self.fc1 = nn.Linear(num\_in, 64)          self.fc2 = nn.Linear(64, 32)          self.fc3 = nn.Linear(32, 16)          self.fc4 = nn.Linear(16,6)          self.fc5 = nn.Linear(6,3) | [0.9, 0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9] | 90% | HW1-mod1.ipynb |
| Mod#2 | Change activation function (sigmoid)      def forward(self, x):          x = F.sigmoid(self.fc1(x))          x = F.sigmoid(self.fc2(x))          x = self.fc3(x) | [0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9] | 90% | HW1-mod2.ipynb |
| Mod#3 | Changes the output sizes (dimensionalities) of layers from 3 to 12  self.fc1 = nn.Linear(num\_in, 32)          self.fc2 = nn.Linear(32, 6)          self.fc3 = nn.Linear(6, 12) | [0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9] | 90% | HW1-mod3 |

* 3 modified model has been tested 10 times for each, however all of them commonly yielded 90% of average score of 10 best accuracies.
* For each modified model of each test trial, I manually initialized both model implementation codes and test codes.

**<References>**

**(PCA, eigenvalue, eigenvectors)**

https://www.youtube.com/watch?v=YEdscCNsinU&ab\_channel=%EA%B3%B5%EB%8F%8C%EC%9D%B4%EC%9D%98%EC%88%98%ED%95%99%EC%A0%95%EB%A6%AC%EB%85%B8%ED%8A%B8

<http://wwwf.imperial.ac.uk/metric/metric_public/matrices/eigenvalues_and_eigenvectors/eigenvalues2.html>

<https://darkpgmr.tistory.com/105>

**(online calculator)**

**https://www.symbolab.com/**

**(number of parameters)**

https://towardsdatascience.com/number-of-parameters-in-a-feed-forward-neural-network-4e4e33a53655

**(activation function(ReLU))**

https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/\

**(hyper parameters)**

https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a

https://en.wikipedia.org/wiki/Hyperparameter\_(machine\_learning)