For the padding part: When the word is not in the pretrained model, I will embed it as np.zeros(DIM). For long files, which have more words than PADDING_SIZE, I will just embed the first PADDING_SIZE words. For short files, which have less words than PADDING_SIZE, I will embed all the words and embed the remaining parts as np.ones(DIM).

I use RNN.py uploaded by Cristian-Paul Bara as reference and modify the file.

For the final results, the best performance of my DAN model is about 74.286%. The best performance of my RNN model is about 55.000% Although it is not very high, it is quite reasonable. The performance may be further improved if I improve the padding method or the network model.

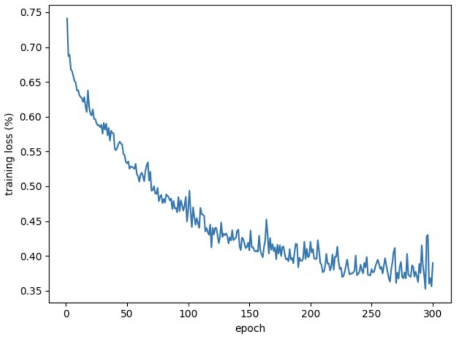
1 DAN

The configuration that leads to the best performance are shown below (number of iterations is set to 300 and even larger number of iterations will not lead to much higher performance):

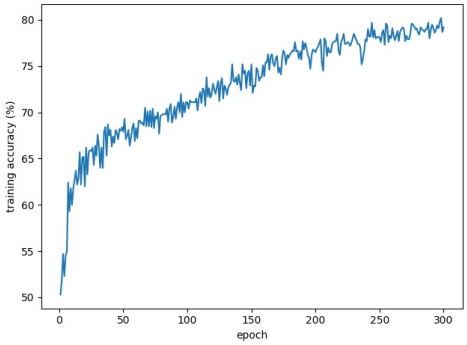
Number of Iterations	**Number of Linear Layers**	**Non-linearity**
ITERS = 300	2	tanh
Dropout Rate	**Learning Rate**	**Embedding Dimension**
DR = 0.2	LR = 0.0001	DIM = 300
Batch Size	**Padding Size**	**Optimizer**
BATCH_SIZE = 50	PADDING_SIZE = 500	Adam

My best performance has the final test accuracy of:

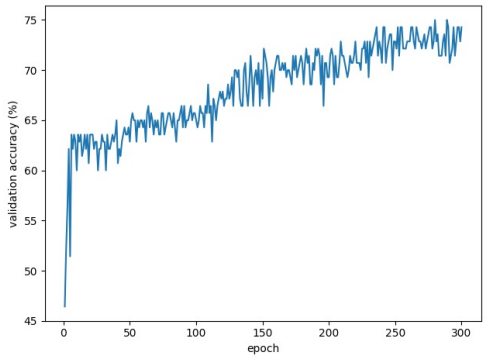
```
**********************************
The test accuracy is: 74.286%.
**********************************
```



DAN Training Loss



DAN Training Accuracy



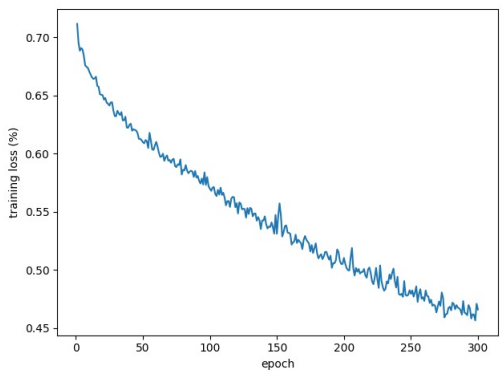
DAN Validation Accuracy

All the following results show that basically the training loss is decreasing, and the training and testing accuracy are increasing as the epoch number increases. The detailed comparisons will be shown below so that we can see why the following configuration leads to the highest testing accuracy:

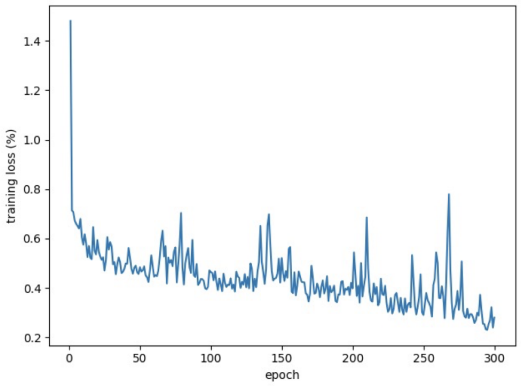
1.1 Learning Rate

I have tried many learning rates in my DAN model, which can be specified by the global variable *LR* in the python file. I will just show the results for two representative learning rates: LR = 0.0001 and LR = 0.1. Below are the comparison between the training loss, training accuracy and validation accuracy of the two functions:

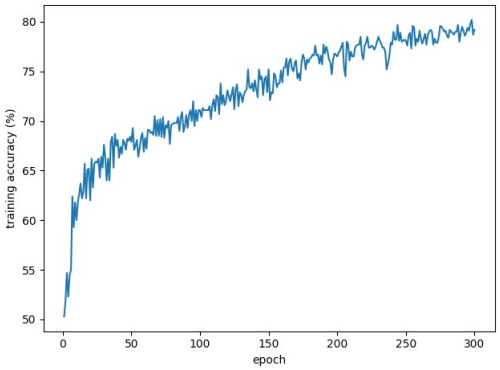
Training loss:



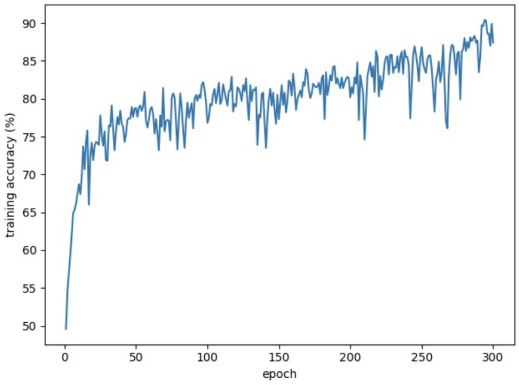
Training loss when LR = 0.0001



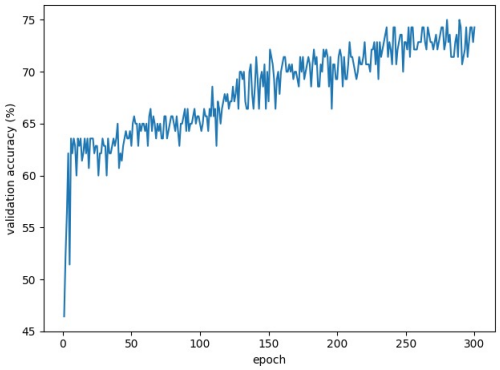
Training loss when LR = 0.1



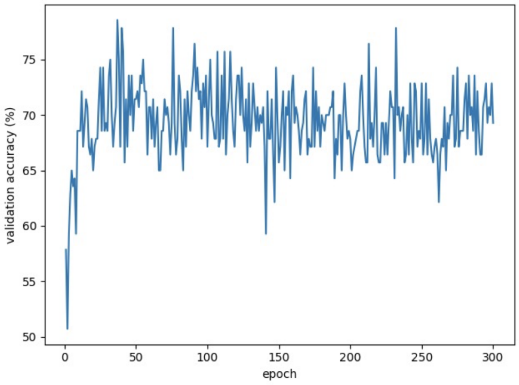
Training accuracy when LR = 0.0001



Training accuracy when LR = 0.1



Validation accuracy when LR = 0.0001



Validation accuracy when LR = 0.1

The final accuracy of the testing files for LR = 0.0001 is:

```
**********************************
  The test accuracy is: 74.286%.
**********************************
```

The final accuracy of the testing files for LR = 0.1 is:

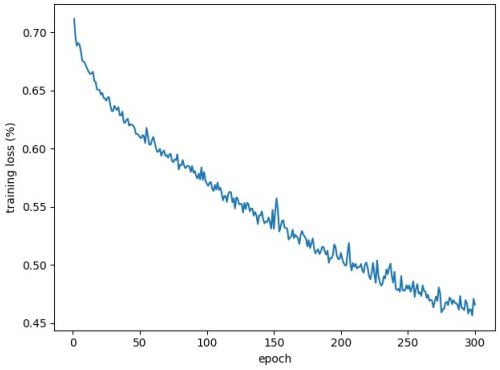
```
**********************************
  The test accuracy is: 70.000%.
**********************************
```

We can see from the above figures that when all the other conditions are the same, learning rate = 0.0001 leads to higher testing accuracy.

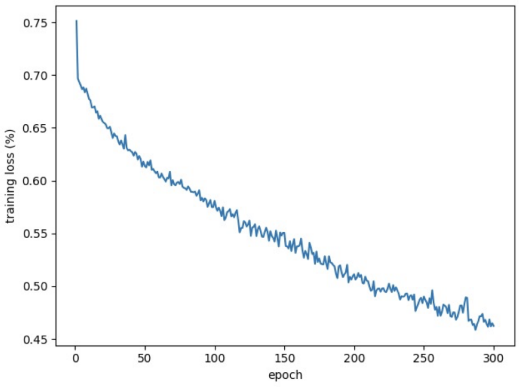
1.2 Dropout Rate

I have tried many dropout rates in my DAN model, which can be specified by the global variable *DR* in the python file. I will just show the results for two representative dropout rates: DR = 0.2 and DR = 0.6. Below are the comparison between the training loss, training accuracy and validation accuracy of the two functions:

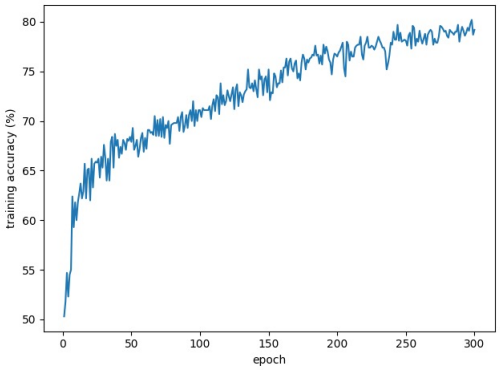
Training loss:



Training loss when DR = 0.2

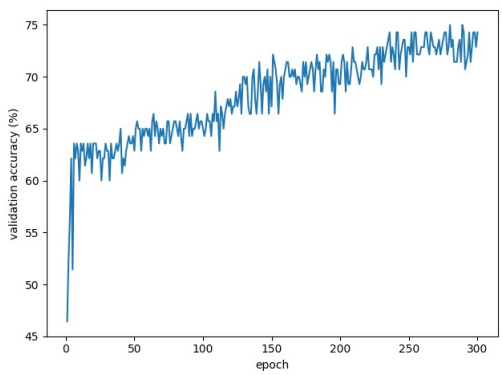


Training loss when DR = 0.6

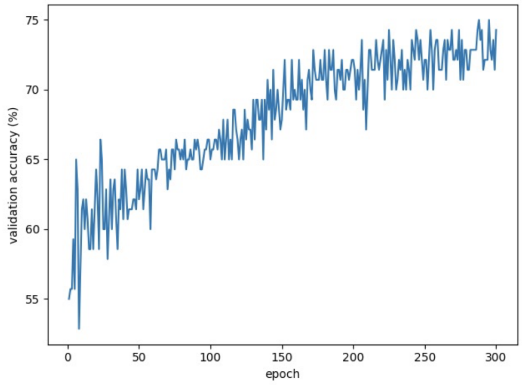


Training accuracy when DR = 0.2

Training accuracy when DR = 0.6



Validation accuracy when DR = 0.2



Validation accuracy when DR = 0.6

The final accuracy of the testing files for DR = 0.2 is:

```
**********************************
The test accuracy is: 74.286%.
**********************************
```

The final accuracy of the testing files for DR = 0.6 is:

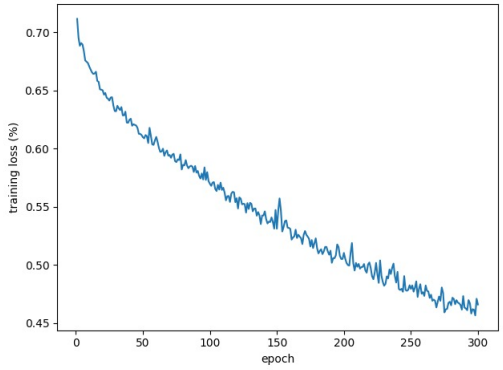
```
**********************************
The test accuracy is: 73.571%.
**********************************
```

We can see from the above figures that when all the other conditions are the same, dropout rate = 0.2 leads to higher testing accuracy.

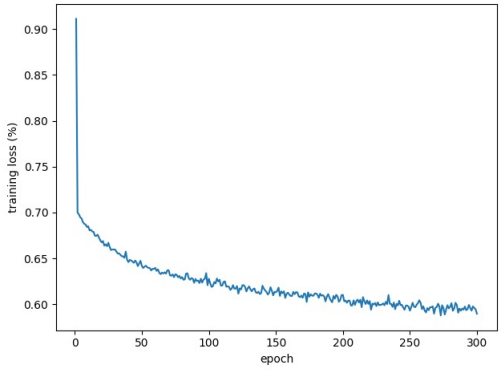
1.3 Embeddings

I have tried different embedding dimensions, which can be specified by the global variable *DIM* in the python file. I will just show the results for two representative embedding dimensions: DIM = 300 and DIM = 50. Below are the comparison between the training loss, training accuracy and validation accuracy of the two functions:

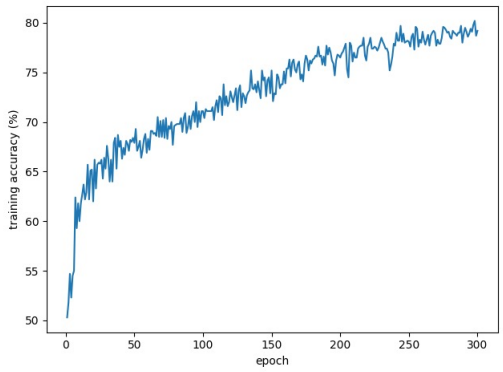
Training loss:



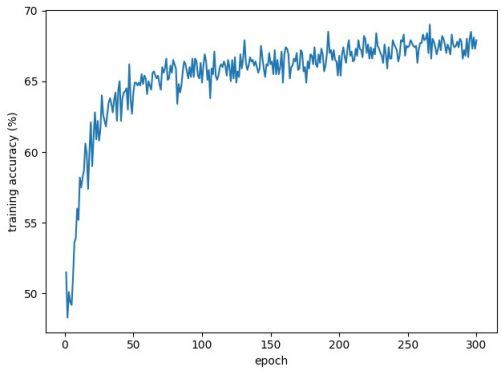
Training loss when DIM = 300



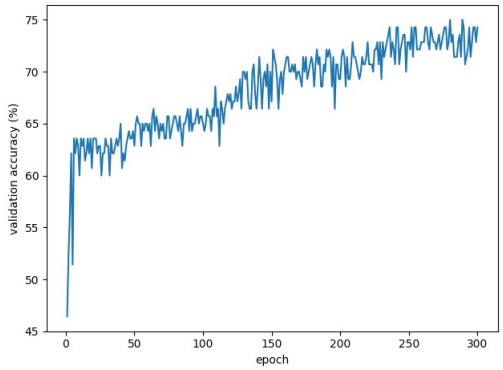
Training loss when DIM = 50



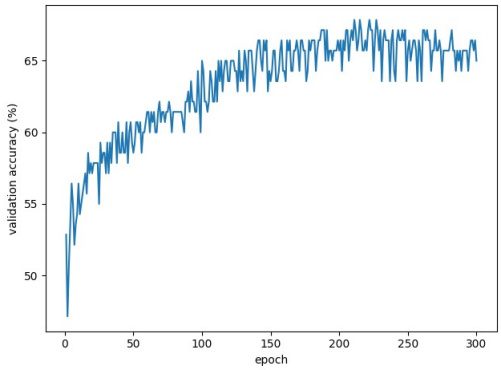
Training accuracy when DIM = 300



Training accuracy when DIM = 50



Validation accuracy when DIM = 300



Validation accuracy when DIM = 50

The final accuracy of the testing files for tanh function is:

```
**********************************
  The test accuracy is: 74.286%.
**********************************
```

The final accuracy of the testing files for relu function is:

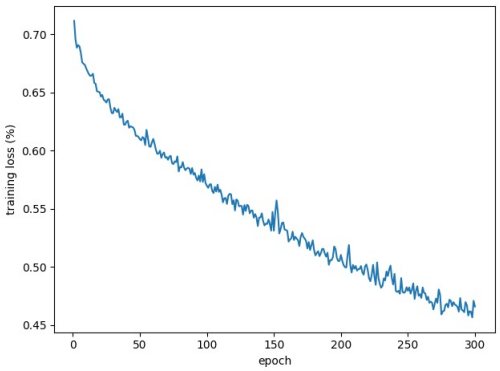
```
**********************************
  The test accuracy is: 64.286%.
**********************************
```

We can see from the above figures that when all the other conditions are the same, embedding dimension = 300 leads to higher testing accuracy.

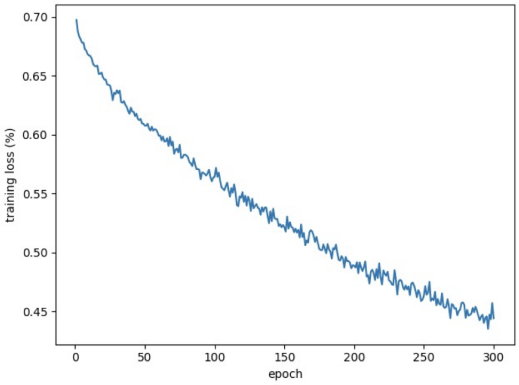
1.4 Non Linearity

I have tried **relu** and **tanh** in the forward function of my DAN model. Below are the comparison between the training loss, training accuracy and validation accuracy of the two functions:

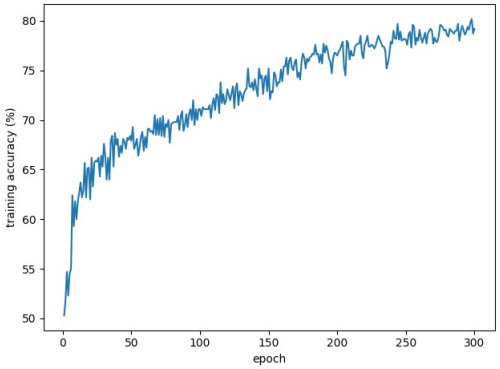
Training loss:



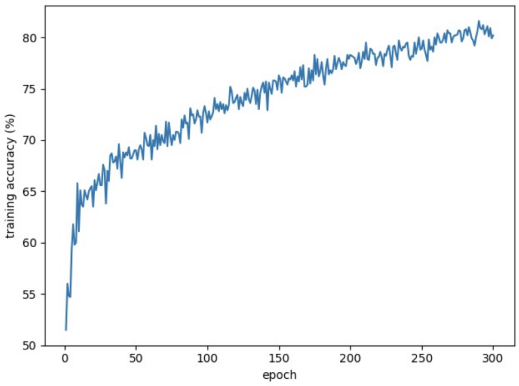
Training loss for relu



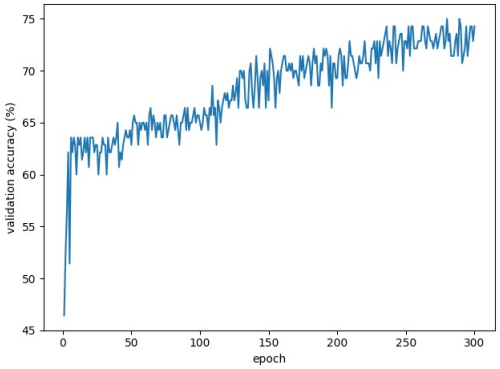
Training loss for tanh



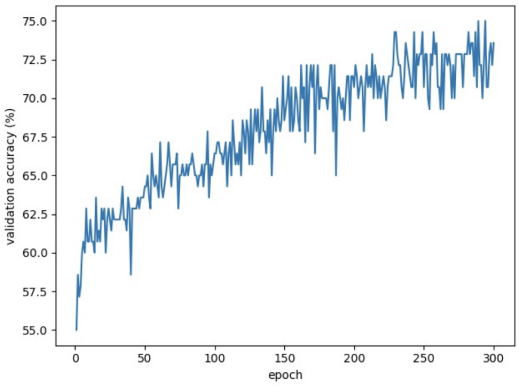
Training accuracy for relu



Training accuracy for tanh



Validation accuracy for relu



Validation accuracy for tanh

The final accuracy of the testing files for tanh function is:

```
**********************************
  The test accuracy is: 74.286%.
**********************************
```

The final accuracy of the testing files for relu function is:

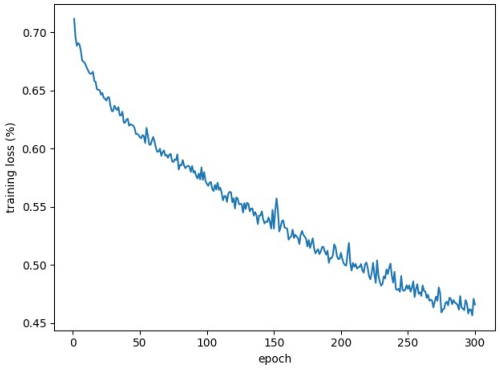
```
**********************************
  The test accuracy is: 72.143%.
**********************************
```

We can see from the above figures that when all the other conditions are the same, the tanh function leads to higher testing accuracy.

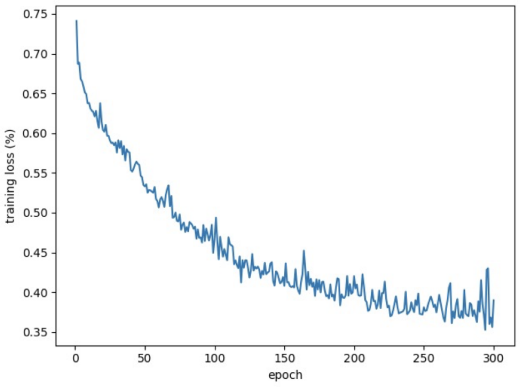
1.5 Number of Hidden Layers

I have tried 2 or 3 linear layers in the DAN class of my DAN model. Below are the comparison between the training loss, training accuracy and validation accuracy of the two functions:

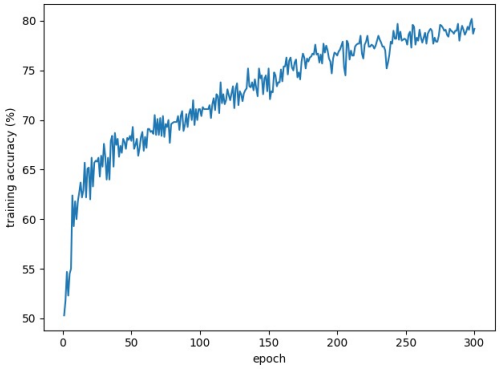
Training loss:



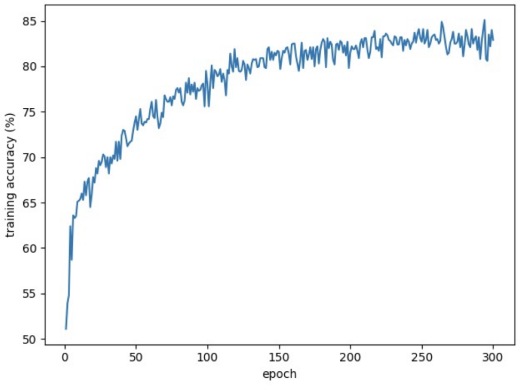
Training loss for 2 layers



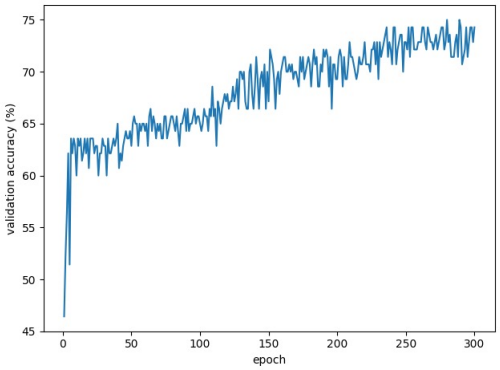
Training loss for 3 layers



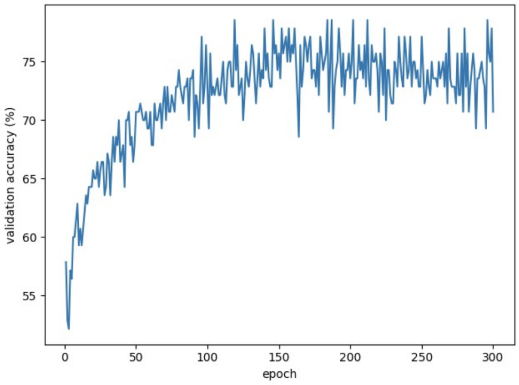
Training accuracy for 2 layers



Training accuracy for 3 layers



Validation accuracy for 2 layers



Validation accuracy for 3 layers

The final accuracy of the testing files for 2 linear layers is:

```
**********************************
The test accuracy is: 74.286%.
**********************************
```

The final accuracy of the testing files for 3 linear layers is:

```
**********************************
The test accuracy is: 72.857%.
**********************************
```

We can see from the above figures that when all the other conditions are the same, having 2 linear layers leads to higher testing accuracy.

2 RNN

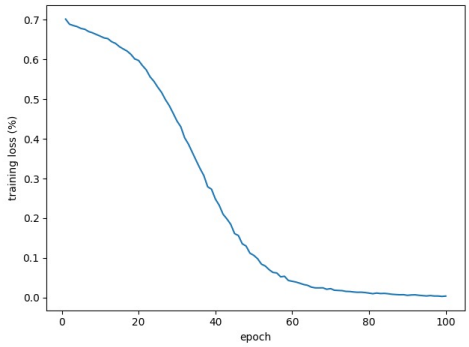
For the RNN part, it takes much longer time to train the model, so I will only present comparisons when the number of iterations is 10. Although it is not very large, we can see the basic tendency. But the results with the best performance will be shown using ITERS = 100. The configuration that leads to the best performance are shown below:

Number of Iterations	**Number of Linear Layers**	**Non-linearity**
ITERS = 50	3	relu
Dropout Rate	**Learning Rate**	**Embedding Dimension**
DR = 0.2	LR = 0.0001	DIM = 300
Batch Size	**Padding Size**	**Optimizer**
BATCH_SIZE = 50	PADDING_SIZE = 500	Adam

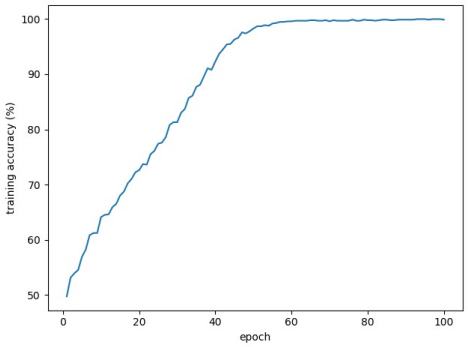
Generating the below plots takes quite a long time, so I change the ITERS to 50 in the code since after epoch = 50, the three values don't change much.

My best performance has the final test accuracy of:

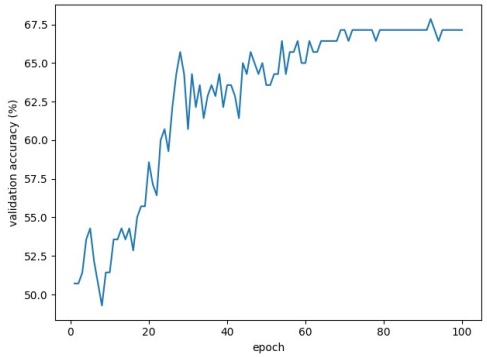
```
**********************************
The test accuracy is: 55.000%.
**********************************
```



RNN Training Loss



RNN Training Accuracy



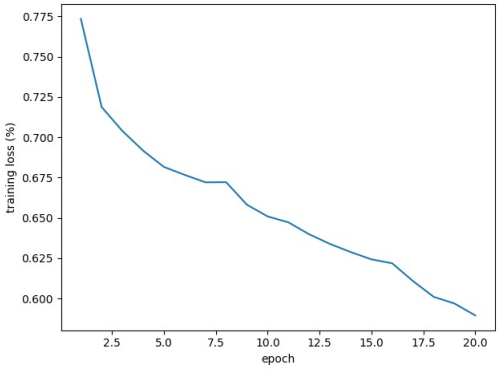
RNN Validation Accuracy

All the following results show that basically the training loss is decreasing, and the training and testing accuracy are increasing as the epoch number increases. The detailed comparisons will be shown below so that we can see why the following configuration leads to the highest testing accuracy:

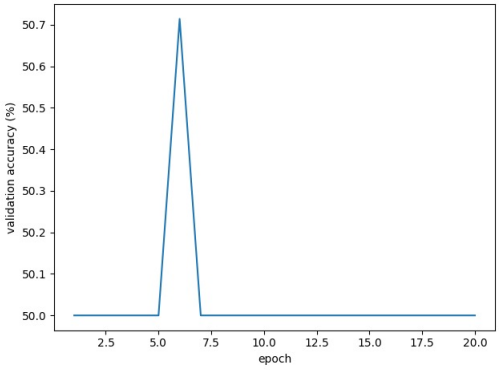
2.1 Learning Rate

I have tried many learning rates in my RNN model, which can be specified by the global variable *LR* in the python file. I will just show the results for two representative learning rates: LR = 0.0001 and LR = 0.1. Below are the comparison between the training loss, training accuracy and validation accuracy of the two functions:

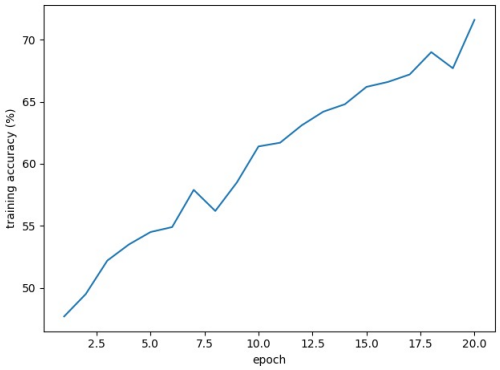
Training loss:



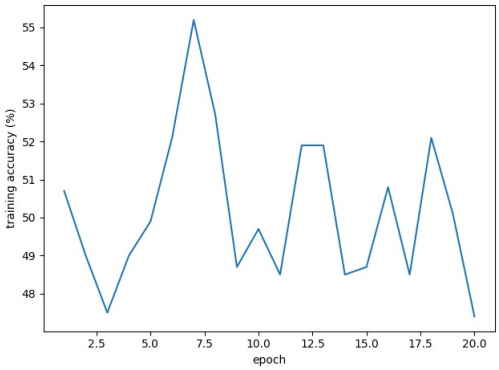
Training Loss for LR = 0.0001



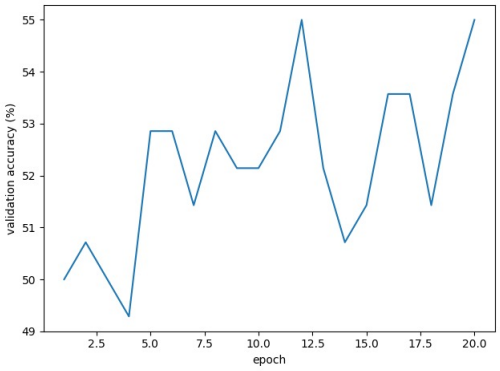
Training Loss for LR = 0.1



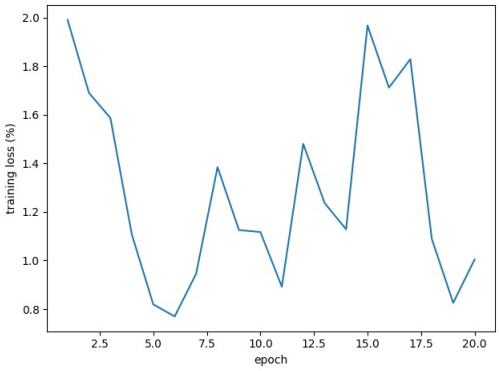
Training Accuracy for LR = 0.0001



Training Accuracy for LR = 0.1



Validation Accuracy for LR = 0.0001



Validation Accuracy for LR = 0.1

The final accuracy of the testing files for LR = 0.0001 is:

```
**********************************
  The test accuracy is: 52.857%.
**********************************
```

The final accuracy of the testing files for LR = 0.1 is:

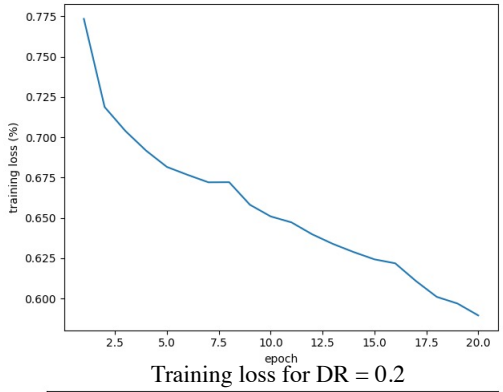
```
**********************************
  The test accuracy is: 50.241%.
**********************************
```

We can see from the above figures that when all the other conditions are the same, learning rate = 0.0001 leads to higher testing accuracy.

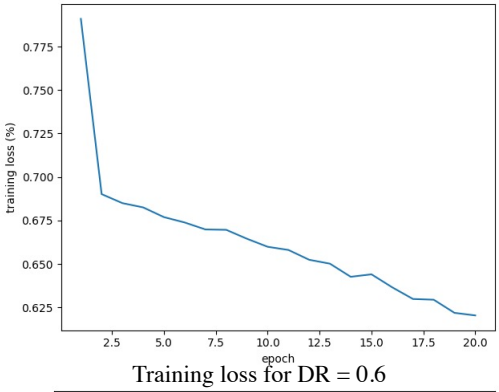
2.2 Dropout Rate

I have tried many dropout rates in my RNN model, which can be specified by the global variable *DR* in the python file. I will just show the results for two representative dropout rates: DR = 0.2 and DR = 0.6. Below are the comparison between the training loss, training accuracy and validation accuracy of the two functions:

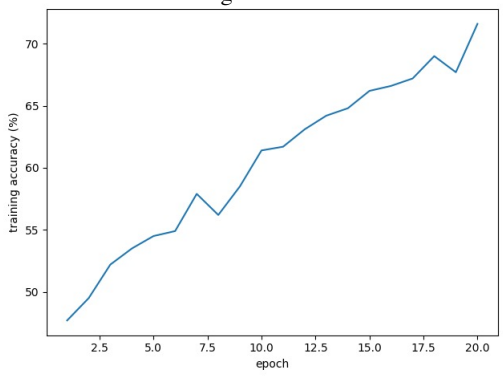
Training loss:



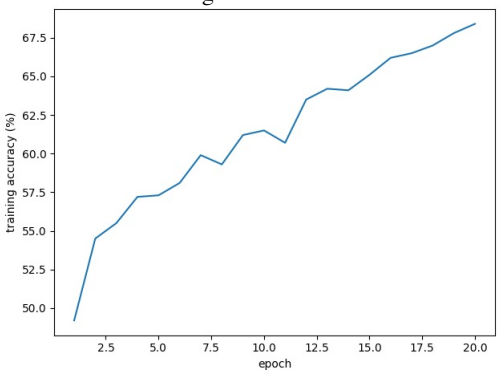
Training loss for DR = 0.2



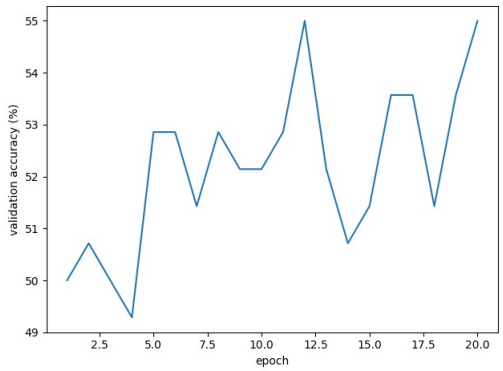
Training loss for DR = 0.6



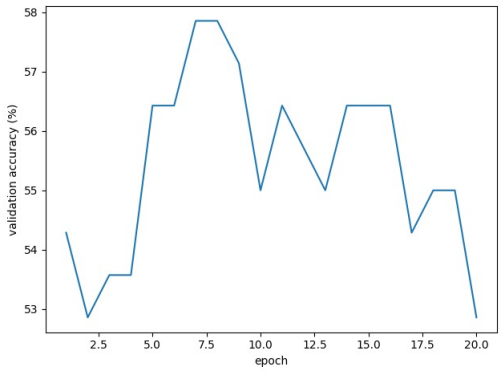
Training Accuracy for DR = 0.2



Training Accuracy for DR = 0.6



Validation Accuracy for DR = 0.2



Validation Accuracy for DR = 0.6

The final accuracy of the testing files for DR = 0.2 is:

```
**********************************
The test accuracy is: 52.857%.
**********************************
```

The final accuracy of the testing files for DR = 0.6 is:

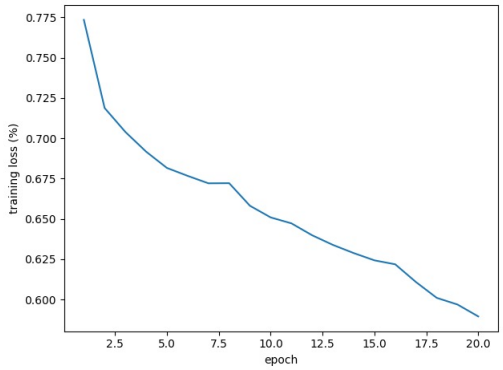
```
**********************************
The test accuracy is: 51.137%.
**********************************
```

We can see from the above figures that when all the other conditions are the same, dropout rate = 0.2 leads to higher testing accuracy.

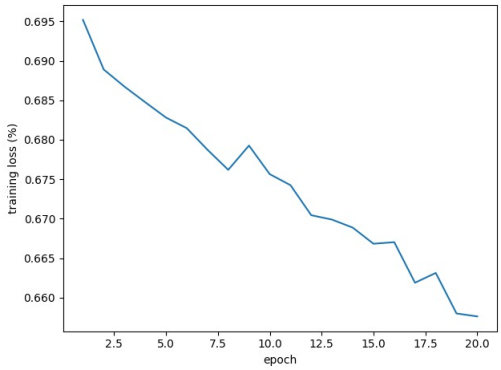
2.3 Embeddings

I have tried different embedding dimensions, which can be specified by the global variable *DIM* in the python file. I will just show the results for two representative embedding dimensions: DIM = 300 and DIM = 50. Below are the comparison between the training loss, training accuracy and validation accuracy of the two functions:

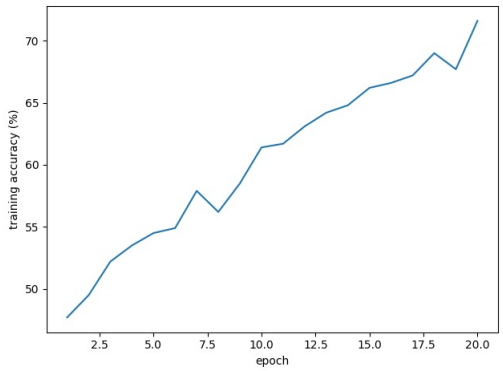
Training loss:



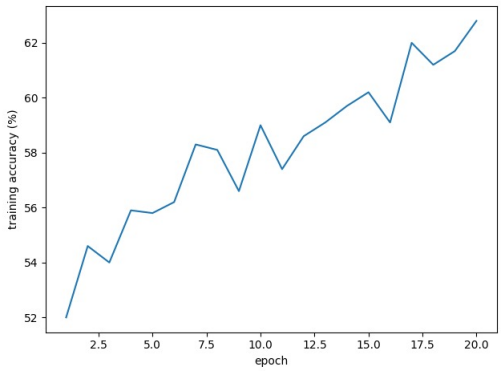
Training loss for DIM = 300



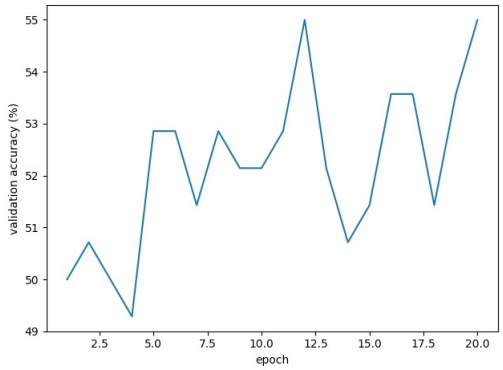
Training loss for DIM = 50



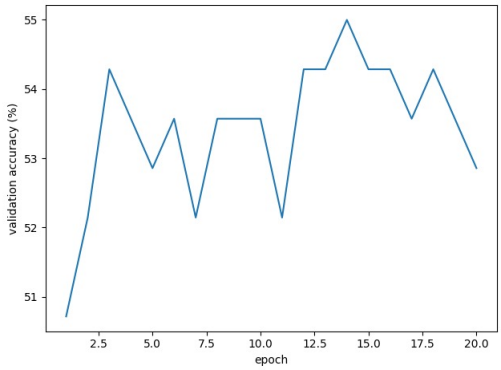
Training Accuracy for DIM = 300



Training Accuracy for DIM = 50



Validation Accuracy for DIM = 300



Validation Accuracy for DIM = 50

The final accuracy of the testing files for DIM = 300 is:

```
**********************************
The test accuracy is: 52.857%.
**********************************
```

The final accuracy of the testing files for DIM = 50 is:

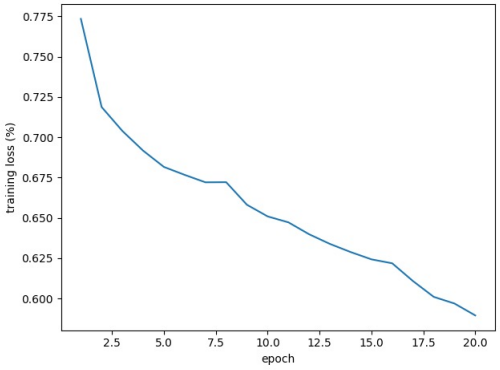
```
**********************************
The test accuracy is: 50.714%.
**********************************
```

We can see from the above figures that when all the other conditions are the same, embedding dimension = 300 leads to higher testing accuracy.

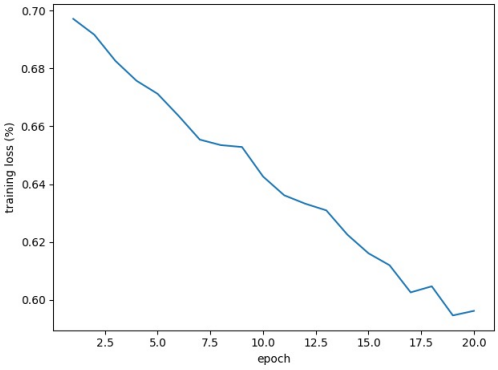
2.4 Non Linearity

I have tried **relu** and **tanh** in the forward function of my RNN model. Below are the comparison between the training loss, training accuracy and validation accuracy of the two functions:

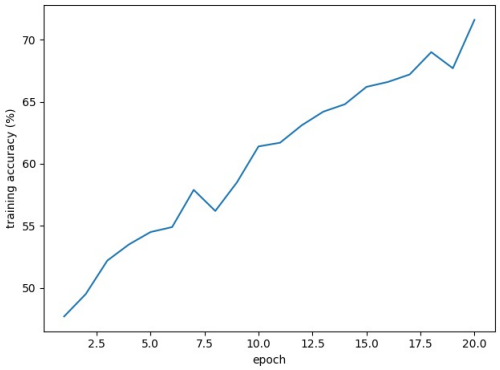
Training loss:



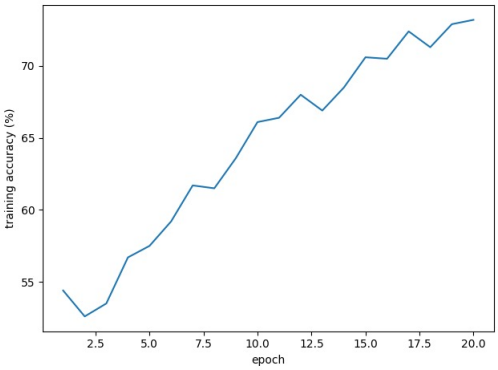
Training Loss for relu



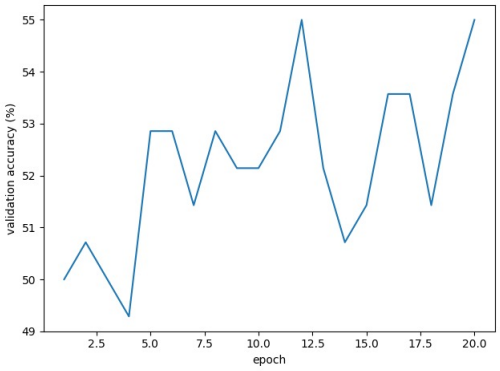
Training Loss for tanh



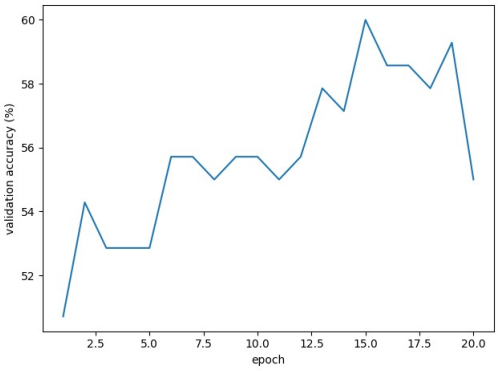
Training Accuracy for relu



Training Accuracy for tanh



Validation Accuracy for relu



Validation Accuracy for tanh

The final accuracy of the testing files for tanh function is:

```
**********************************
The test accuracy is: 52.857%.
**********************************
```

The final accuracy of the testing files for relu function is:

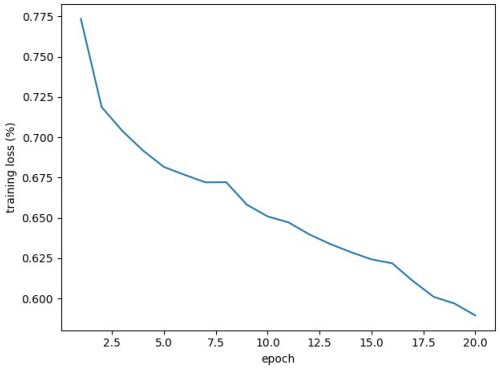
```
**********************************
The test accuracy is: 51.429%.
**********************************
```

We can see from the above figures that when all the other conditions are the same, the tanh function leads to higher testing accuracy. However, the number of iterations is too small. If we make more iterations, we will find that relu function has a higher accuracy. So we use relu function.

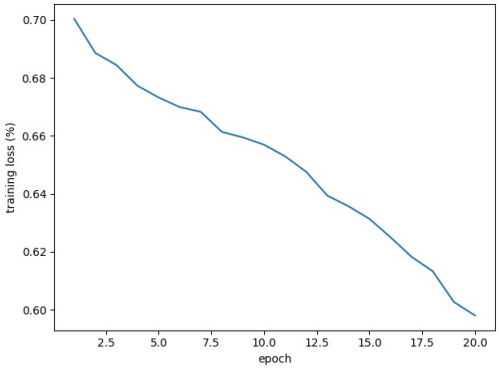
2.5 Number of Hidden Layers

I have tried 2 or 3 linear layers in the RNN class of my RNN model. Below are the comparison between the training loss, training accuracy and validation accuracy of the two functions:

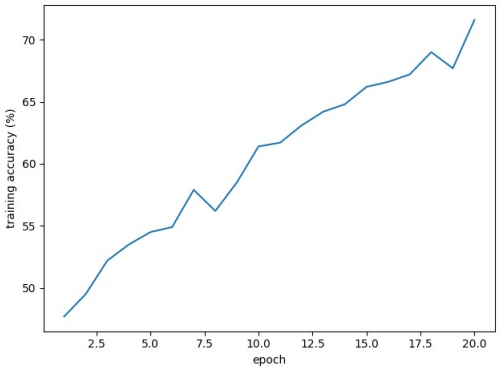
Training loss:



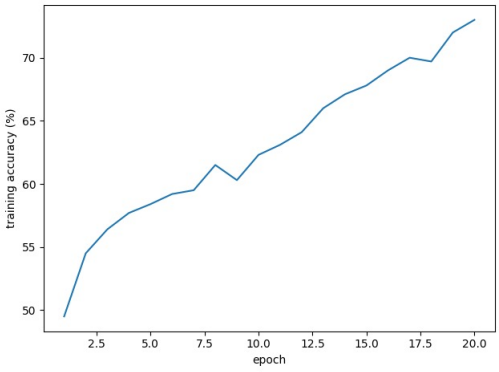
Training Loss for 2 layers



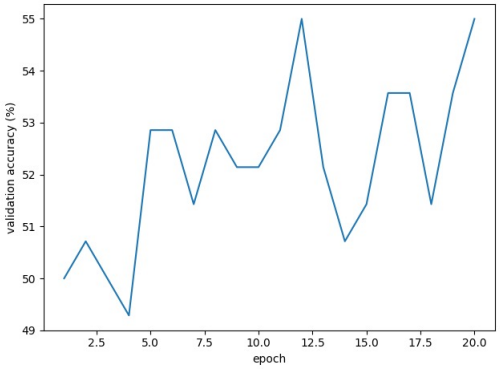
Training Loss for 3 layers



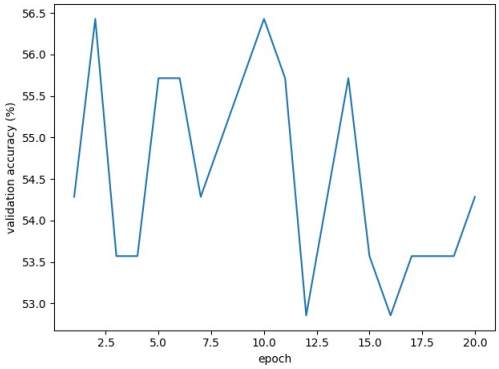
Training Accuracy for 2 layers



Training Accuracy for 3 layers



Validation Accuracy for 2 layers



Validation Accuracy for 3 layers

The final accuracy of the testing files for 2 linear layers is:

```
**********************************
  The test accuracy is: 52.857%.
**********************************
```

The final accuracy of the testing files for 3 linear layers is:

```
**********************************
  The test accuracy is: 55.000%.
**********************************
```

We can see from the above figures that when all the other conditions are the same, having 3 linear layers leads to higher testing accuracy.