CSC420 Assignment 2

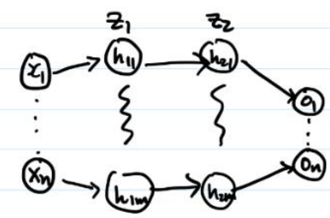
🌟 Owner's Information

First Name: Kwan Kiu
Last Name: Choy
Student No: 1005005879
UtorId: choykwan

Theoretical Part

Question 1

1)



Let z_1, z_2 be the output of the 1st and 2nd layer.
Let W_1, W_2 be the weight matrices of size $m \times n$ matrix where n is the number of the inputs and m is the number of the units of the hidden layers. Let b_1, b_2 be the biases of each hidden layer. Let $f(x) = ax + C$ be the linear activation function and I_1, I_2 be the output of the activation function

$$z_1 = W_1X + b_1$$
$$I_1 = f(z) = az + C = aW_1x + \underbrace{ab_1 + C} = aW_1x + \underbrace{b_1'}$$
$$z_2 = aW_2W_1X + W_2b_1' + b_2$$
$$I_2 = a(aW_2W_1x + W_2b_1' + b_2) + C$$
$$= a^2W_2W_1X + \underbrace{aW_2b_1' + ab_2 + C}$$
$$= \underbrace{a^2W_2W_1}_{W'}X + \underbrace{b_2'}$$

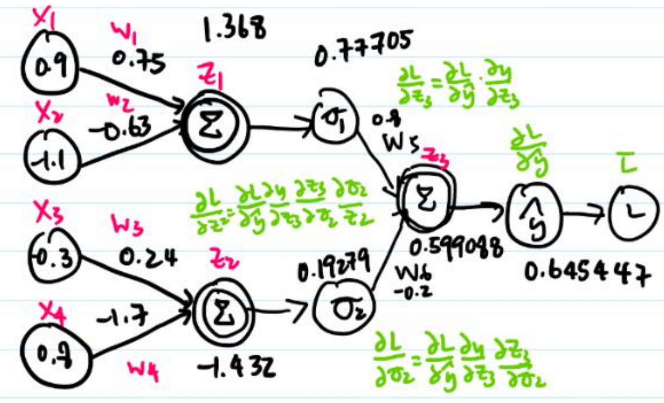
As a result, we have shown that I_2 can be expressed as $W'X + b_2'$ which means that it can be expressed as a linear function of X. The slope of the linear function is the W' matrix.

This aligns with our understanding since we are only applying a linear function on the weighted sum between the weights and the inputs at each hidden layer. We know that the composition of 2 linear function is just another linear function. No matter how many times we apply $f(x)$, the result will still be a linear function of inputs.

No matter what k is, z_1, z_2 can always be expressed as a linear function of the inputs, X. This means that back propagation will not have any effect since the derivative is constant.

Question 2

2)



Equations:

1. $L = \| 0.5 - \hat{y} \|^2$
2. $\hat{y} = \frac{1}{1+e^{-z_3}}$
3. $z_3 = w_5 \sigma_1 + w_6 \sigma_2$
4. $\sigma_1 = \frac{1}{1+e^{-z_1}}$
5. $\sigma_2 = \frac{1}{1+e^{-z_2}}$
6. $z_2 = w_3 x_3 + w_4 x_4$

Partial derivatives

$\bar{L} = 1$

$\frac{\partial L}{\partial \hat{y}} = -2 \| 0.5 - \hat{y} \|$

$\frac{\partial \hat{y}}{\partial z_3} = \frac{e^{-z_3}}{(1+e^{-z_3})^2}$

$\frac{\partial z_3}{\partial \sigma_2} = w_6$

$\frac{\partial \sigma_2}{z_2} = \frac{e^{-z_2}}{(1+e^{-z_2})^2}$

$\frac{\partial z_2}{w_3} = x_3$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial L} \cdot \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_3} \cdot \frac{\partial z_3}{\partial \sigma_2} \cdot \frac{\partial \sigma_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_3}$$

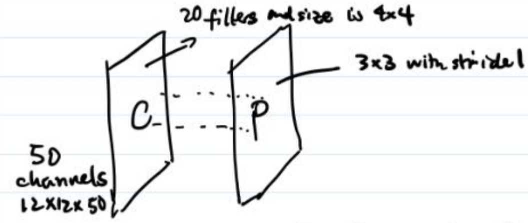
$$= -2 \| 0.5 - \hat{y} \| \frac{e^{-z_3}}{(1+e^{-z_3})^2} w_6 x_3 \frac{e^{-z_2}}{(1+e^{-z_2})^2}$$

$$= -2 \| 0.5 - 0.645451 \| \frac{e^{-0.59908}}{(1+e^{-0.599088})^2} (-0.2)(-0.3) \frac{e^{-1.432}}{(1+e^{-1.432})}$$

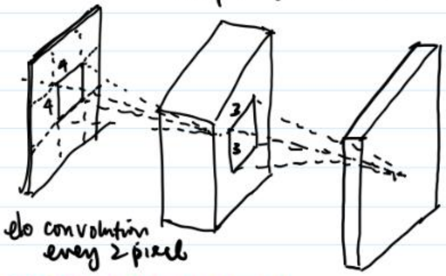
$$= 6.2158 \times 10^{-4}$$

Question 3

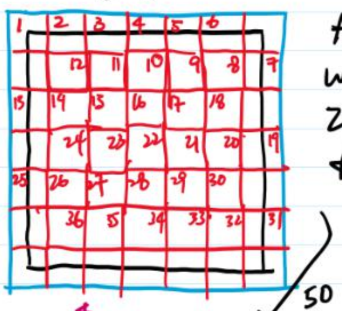
3)



Output of C has 20 channels



do convolution every 2 pixels



After convolution, we have an image of 6x6
we do convolution 36 times, and there are 20 filters in total.

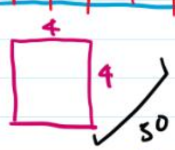
Each convolution:

→ 50x16 multiplication

→ 50x16-1 addition

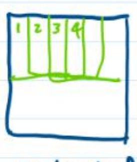
Total number of operations in convolution layer:

$20 \times 36 \times (50 \times 16 + 50 \times 16 - 1) = 1151280.$



Total number of operations in max pooling layer.

$16 \times 8 \times 20 = 2560$



Output of convolution

16 max pooling operations

In max pooling, there are 9 pixels per filter so each pooling accounts for 8 FLOPS

↳ Total number of operations in the network without bias:

$1151280 + 2560 = 1153840$ //

↳ Total number of operations in network with bias:

Each convolution operation

- 50x16 multiplications
- 50x16 additions

$20 \times 36 \times (50 \times 16 + 50 \times 16) + 16 \times 8 \times 20 = 1154560$ //

Question 4

4) First layer: Input $\rightarrow$ C1

We know that the filter size can be computed by the size of the input tensor W_1 and the output tensor W_2.

$W_2 = (W_1 - F)/S + 1$ F-filter size S-stride size

$28 = (32 - F)/1 + 1$

$27 = 32 - F$

$F = 5$

The number of trainable parameters per feature map in C_1:
total number pixels in filter + bias

$= 5 \times 5 + 1 = 26$

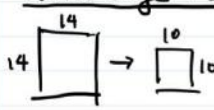
We have 6 feature maps in C_1, so total for this layer:

$26 \times 6 = 156$

Second layer: $C_1 \rightarrow S_2$

Assuming that the subsampling operation is performed by a max pooling or an average pooling layer, there would not be any trainable parameters in this layer.

Third layer: $S_2 \rightarrow C_3$



using formula before,

$W_2 = (W_1 - F)/S + 1$

$10 = (14 - F)/1 + 1$

$9 = 14 - F$

So the filter needed $F = 5$

in the convolution operation is a 5×5 filter.

Each of the feature map would have its own bias, and therefore the total number of trainable parameters would be

$6 \times (5 \times 5 + 1) \times 16$

$= 6 \times 26 \times 16 = 2496$

Forth layer $C_3 \rightarrow S_4$

Assuming that the subsampling operation is performed by a max pooling or an average pool (fixed pooling method) there are no trainable parameters.

Fifth layer: $S_4 \rightarrow C_5$

We know the filter is 5×5 and then there's going to be only 1 bias term for each feature map in the C5 layer

$(16 \times 5 \times 5 + 1)120 = 48120$

Sixth layer: $C_5 \rightarrow F_6$

There's 1 bias for each pixel in F6 layer, so the total number of trainable parameters:

$84(120 + 1) = 10164$

Seventh layer $F_6 \rightarrow$ Gaussian

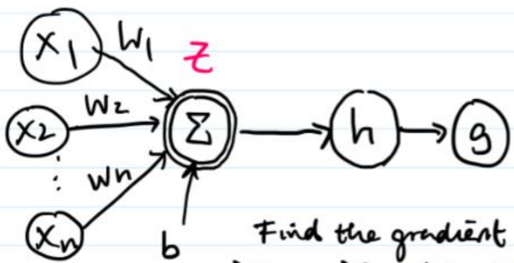
There's 1 bias for each pixel in the gaussian layer, so:
the number of trainable parameters:

$10(84 + 1) = 850$

$\boxed{\text{Total}}$: $156 + 2496 + 48120 + 10164 + 850 = 61786$ ∎

Question 5

5)



Forward Pass

1) $g = h$
2) $h = \frac{1}{1+e^{-z}}$
3) $z = w_1 x_1 + w_2 x_2 + \ldots w_n x_n + b$

Find the gradient of g with respect to the inputs

$\boxed{1}$

$$\frac{\partial g}{\partial x_1} = \frac{\partial g}{\partial g} \cdot \frac{\partial g}{\partial h} \cdot \frac{\partial h}{\partial z} \cdot \frac{\partial z}{\partial x_1}$$

$$= 1 \cdot 1 \cdot \frac{e^{-z}}{(1+e^{-z})^2} \cdot w_1$$

$$= \frac{w_1 e^{-z}}{(1+e^{-z})^2}$$

Since we know that z can be fined by using g using $z = -\ln(\frac{1}{g} - 1)$.

$$\text{So } \frac{\partial g}{\partial x_1} = \frac{w_1 e^{\ln(\frac{1}{g}-1)}}{(1+e^{\ln(\frac{1}{g}-1)})^2}$$

$\boxed{2}$

$$\frac{\partial g}{\partial x_2} = \frac{\partial g}{\partial g} \cdot \frac{\partial g}{\partial h} \cdot \frac{\partial h}{\partial z} \cdot \frac{\partial z}{\partial x_2}$$

$$= 1 \cdot 1 \cdot \frac{e^{-z}}{(1+e^{-z})^2} w_2$$

$$= \frac{w_2 e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{w_2 e^{\ln(\frac{1}{g}-1)}}{(1+e^{\ln(\frac{1}{g}-1)})^2}$$

So $\forall i \in \mathbb{N}, 1 \leq i < n$

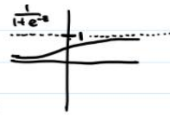
$$\frac{\partial g}{\partial x_i} = \frac{\partial g}{\partial g} \frac{\partial g}{\partial h} \cdot \frac{\partial h}{\partial z} \frac{\partial z}{\partial x_i}$$

$$= \frac{w_i e^{\ln(\frac{1}{g}-1)}}{(1+e^{\ln(\frac{1}{g}-1)})^2}$$

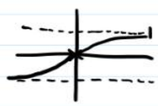
As a result, we have shown that the gradient of g does not depend on any of the input x_i.

Question 6

6a) Logistic Regression: $\frac{1}{1+e^{-w^Tx+b}}$



Tanh(x): $\frac{1-e^{-2x}}{1+e^{-2x}}$



The shape of the two functions are very similar but the range of Tanh(x) function is from -1 to 1 whereas it is 0 to 1 for the logistic

6b) $\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$

Let $f(x)$ represents the logistic regression function such that $f(x) = \frac{1-e^{-x}}{1+e^{-x}}$

$$\tanh(x) = (1-e^{-2x}) \cdot \frac{1}{(1+e^{-2x})}$$
$$= (1-e^{-2x}) \cdot f(2x)$$
$$= \left(2 - \frac{1}{f(2x)}\right) \cdot f(2x)$$
$$= 2f(2x) - 1$$
$$\frac{d}{dx}\tanh(x) = \frac{d}{dx}(2f(2x) - 1)$$
$$= 2f'(2x) \cdot 2$$
$$= 4f'(2x)$$

Derivative of logistic regression:

$$f(x) = \frac{1}{1+e^{-z}}$$
$$f'(x) = \frac{e^{-z}}{(1+e^{-z})^2}$$
$$= \left[\frac{1}{f(x)} - 1\right] f^2(x)$$
$$= f(x) - f^2(x)$$

Continuing with $\frac{d}{dx}\tanh(x)$,
$$= 4f'(2x)$$
$$= 4(f(2x) - f^2(2x))$$
$$= 4f(2x) - 4f^2(2x)$$

As a result, we know that the gradient of the tanh(x) can be expressed as a function of the logistic regression function.

6c) Tanh(x):

↳ It is symmetric around 0. If the mean of the inputs is 0, then it gives network to adjust the weights more flexibly as they don't have to all increase or all decrease together, which makes the backpropagation process more efficient. It centers data so that it's easier to train.

Logistic(x)

The range of the tanh(x) is larger than that of the logistic regression, as shown in (a). As a result, leads to larger derivatives (due to the shape of the curve and the large range), this means that convergence will be reached faster.

Implementation Part

Task 1: Data Preprocessing

```
def step_1_load_data(gaussian=False, normalized=True):
    """
    Preprocessed data and load the data such that trainloaders,
validloaders
    and testloaders can be created for the rest of the codes to use.
    :param gaussian: Boolean to indicates whether we want gaussian blur or
not
    in the data preprocessing stage
    :return:
    1) trainloader: loader to load training data
    2) validloader: Loader to load validation data
    3) testloader: loader to load testing data
    """
    # Define a transform to normalize the data
    transform = transforms.Compose([
        transforms.Grayscale(num_output_channels=1),
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,))])
    if gaussian:
        transform = transforms.Compose([
            transforms.Grayscale(num_output_channels=1),
            transforms.GaussianBlur(3, sigma=0.7),
            transforms.ToTensor(),
            transforms.Normalize((0.5,), (0.5,))])
    if not normalized:
        transform = transforms.Compose([
            transforms.Grayscale(num_output_channels=1),
            transforms.ToTensor()])
    trainset = datasets.ImageFolder(root=TRAIN_DATA_PATH,
transform=transform)
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
                                              shuffle=True)

    testset = datasets.ImageFolder(root=TEST_DATA_PATH,
transform=transform)
    testloader = torch.utils.data.DataLoader(testset, batch_size=64,
                                             shuffle=True)
```

```
    validset = datasets.ImageFolder(root=VALID_DATA_PATH,
transform=transform)
    validloader = torch.utils.data.DataLoader(validset, batch_size=64,
                                              shuffle=True)
    return trainloader, testloader, validloader
```

My preprocessing pipeline:

1) Convert the images with 3 channels (RGB) into a single channel image (grayscale)
    - Using images with 3 channels would be problematic during training process, therefore I decided to convert it to grayscale images so that it is easier to train the network
2) Convert the images to tensors
    - This is a step that we must include in the preprocessing stage as stated in the assignment
3) Normalize the images with mean and std to be 0.5, which will normalize image in the range of [-1, 1]

	Training loss	Valid loss	Train accuracy	Valid accuracy
Without Normalization	1.53001837151	1.54884378623	0.93473333333	0.91499999904
With Normalization	1.5088002970	1.541611072540	0.9542	0.92399999952

Since both the training accuracies and validation accuracies improve with normalization, I decided to include that into my preprocessing pipeline.

Apart from the three steps mentioned in my data preprocessing state, I have also tried whether applying a gaussian blur to smoothen the image will improve the accuracy. The results are displayed below.

	Training loss	Valid loss	Train accuracy	Valid accuracy
With Gaussian	1.52371720536	1.54690929985	0.94166666666	0.918
Without Gaussian	1.50880029703	1.541611072540	0.9542	0.92399999952

Since, the validation loss is actually larger when a gaussian filter is applied, therefore I decided not to include gaussian blurring as part of the preprocessing stage.

Task 2: Testing learning rates and Optimizer

For this task, I have used 4 different optimizers and 5 different learning rates to test the training, validation and testing accuracies.

The 4 different Optimizers:

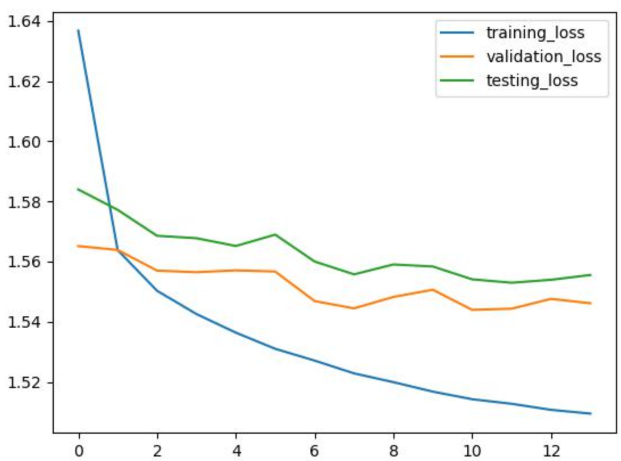
1) SGD
2) Adam
3) RMSProp
4) Adadelta

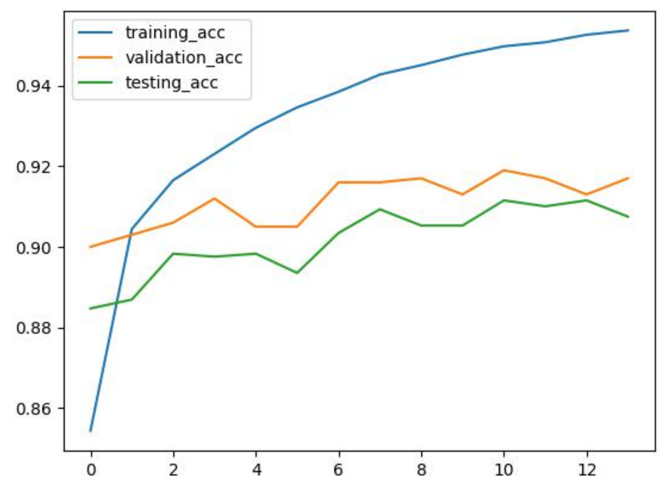
The 5 different learning rates:

1) 0.001
2) 0.01
3) 0.05
4) 0.1
5) 0.5

The table below shows the best results (lowest validation loss) for **each optimizer**.

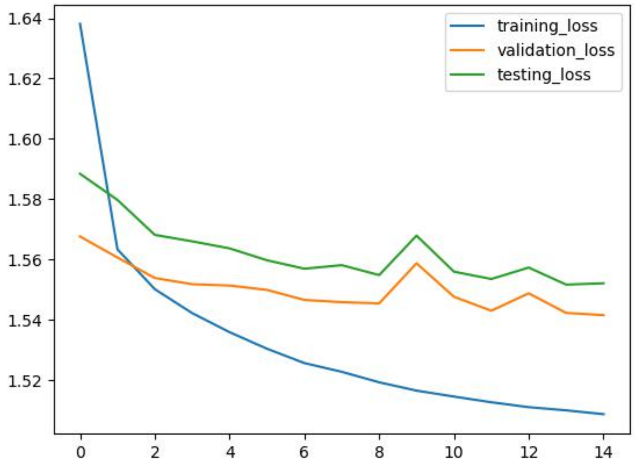
1) Optimizer: Adadelta, Learning rate: 0.5, n_epochs = 14 (early stopped)





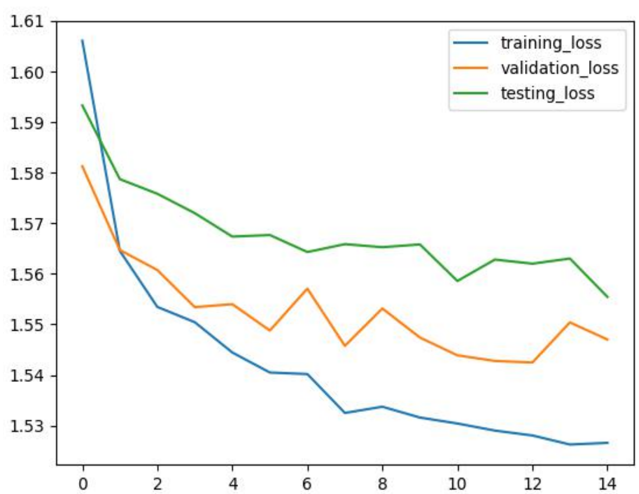
	Training	**Validation**	**Test**
Final Losses	1.509453068033854	1.546137075424194	1.555530561224479
Final Accuracies	0.953666666666666	0.917000000953674	0.907488986434040

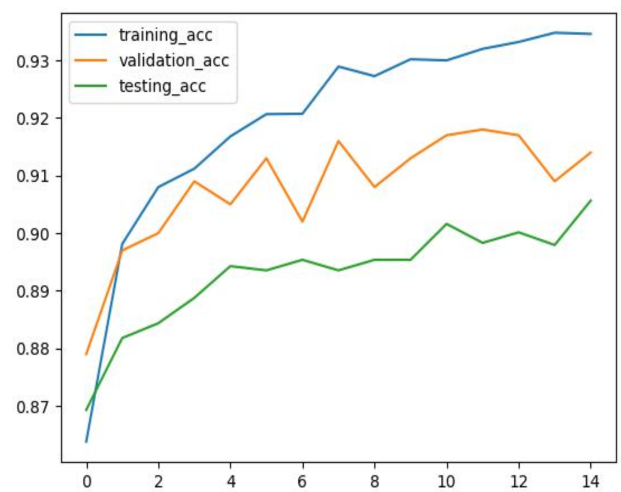
2) **Optimizer: RMSProp, Learning rate: 0.5, n_epochs = 14 (early stopped)**



	Training	Validation	Test
Final Losses	1.508800297037760	1.5416110725402832	1.552166826399412
Final Accuracies	0.9542	0.923999999523162	0.9115271661950278

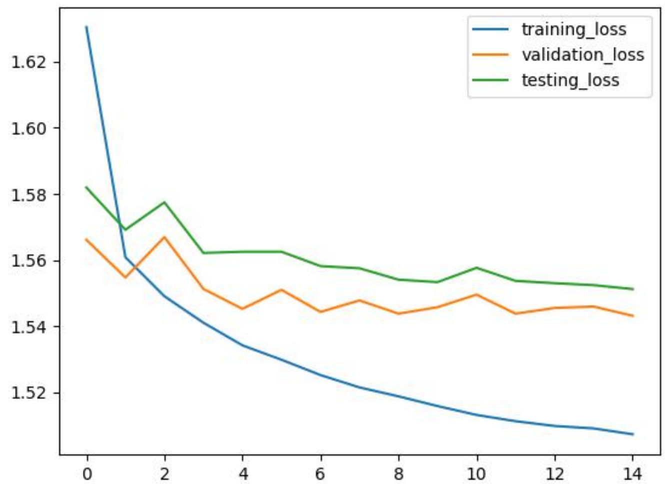
3) Optimizer: Adam, Learning rate: 0.001, n_epochs = 15

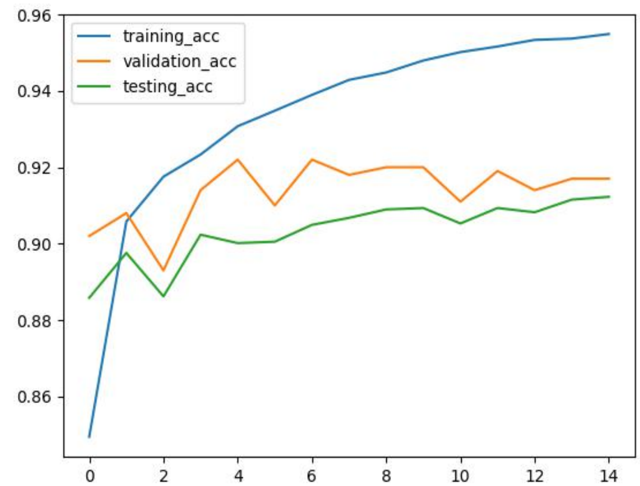




	Training	Validation	Test
Final Losses	1.526564672915140	1.547003357887268	1.55542909678908
Final Accuracies	0.9346	0.914000000476837	0.90565345089516

4) Optimizer: SGD, Learning rate: 0.5, n_epochs= 14 (early stopped)





	Training	Validation	Test
Final Losses	1.507383525848388	1.543181458473205	1.551257100574420
Final Accuracies	0.954866666634877	0.916999999046325	0.91226138014800

- ➔ Therefore, if we summarize the data, we know that using the RMSProp optimizer, a learning rate of 0.5 with early stopping gives us the best set of results which produce the minimum validation errors.

Task 3: Testing the Number of Hidden Units

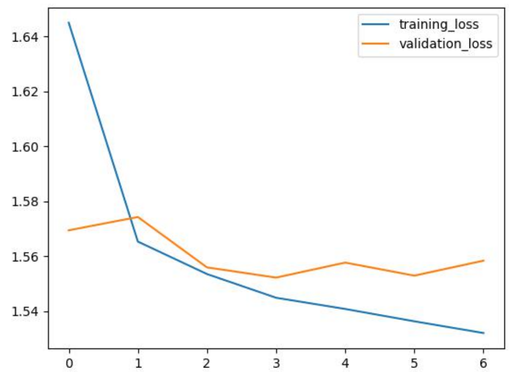
Using the best set of parameters in task 2 (optimizer: RMSProp, learning rate: 0.5), I ran experiments on the model with different hidden units: 100, 500, 1000. The below are the summarized results.

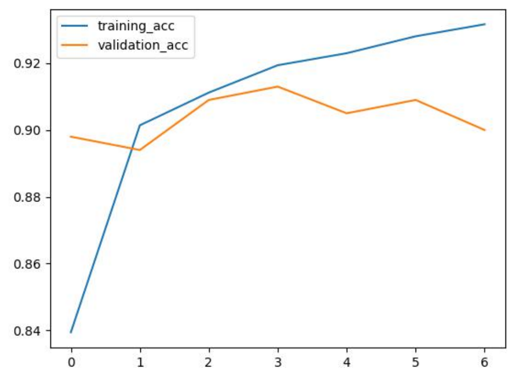
Hidden Units	Final Training Loss	Final Validation Loss	Final Training Accuracy	Final Validation Accuracy
100	1.5208186177	1.543632783889	0.944266666634	0.928
500	1.5171704068	1.545582385063	0.946933333301	0.9199999995231628
1000	1.5133933462	1.54627965354	0.950333333333	0.92

As you can see here, the network with 100 hidden units still has the lowest validation loss, and best validation accuracies. **It seems that the lower the number of hidden units, the smaller the validation loss and hence the better the final results.**

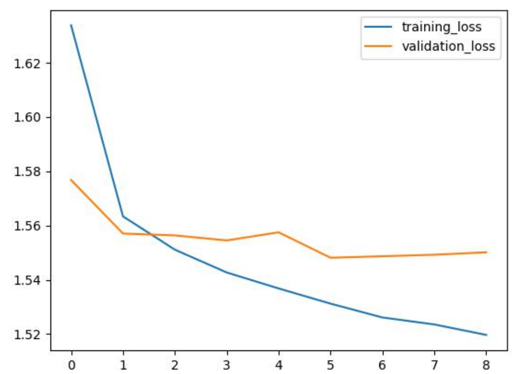
Hidden Units	Loss	Accuracy

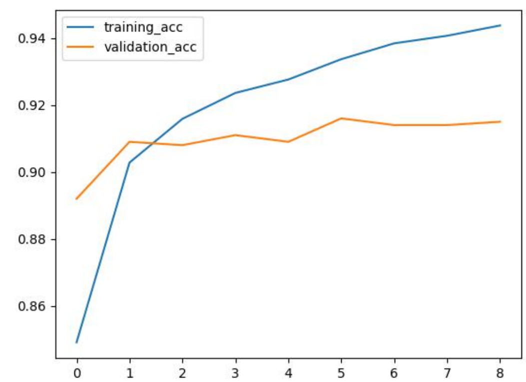
100



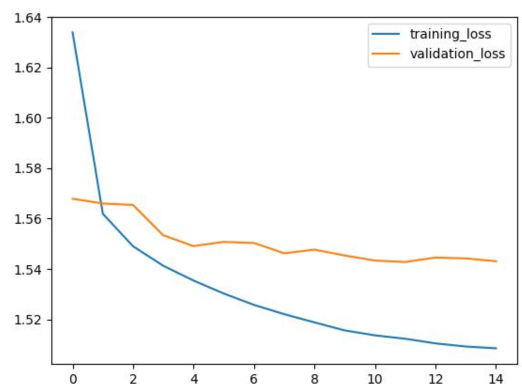


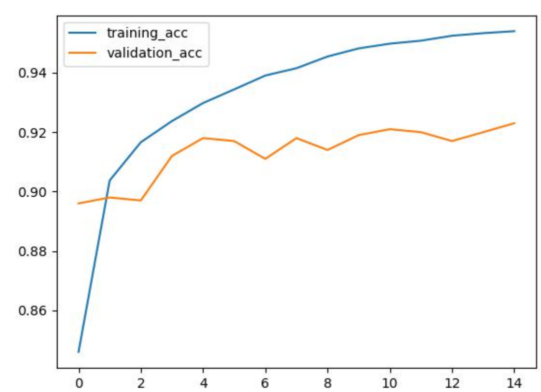
500





1000





Therefore, I ran the model with 100 units on the testing set, and the below are the results that I obtained.

Testing loss	**Testing Accuracies**
1.5678207690495227	0.8939060206455282

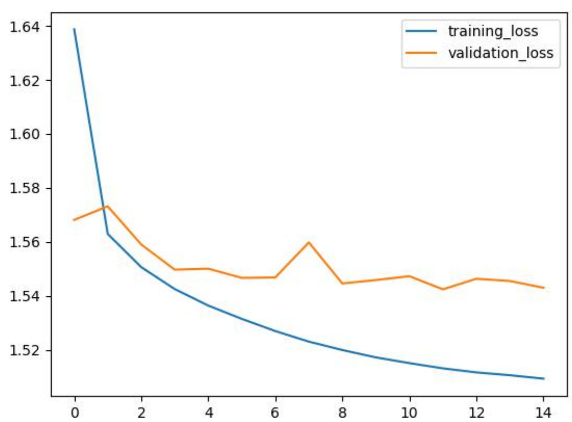
Task 4: Testing the Number of Layers

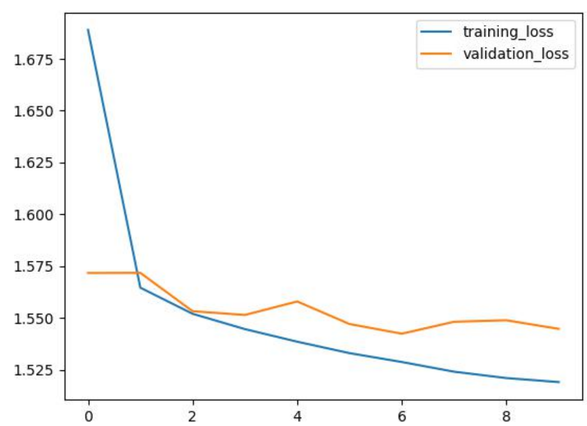
Using the best set of parameters in task 2 (optimizer: SGD, learning rate: 0.5), I ran experiments on both the one layer model (in task 2) and the new two layer model. The below are the summarized results.

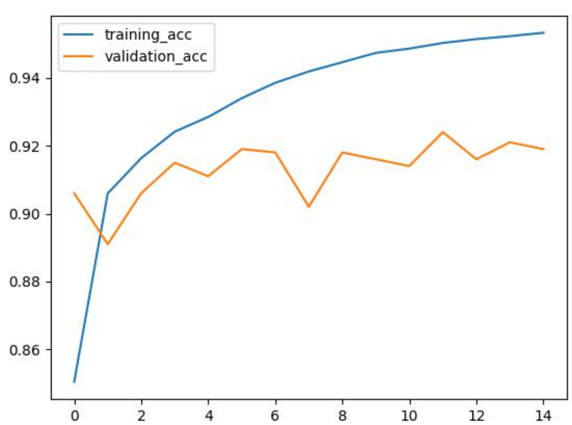
	Training Loss	**Valid Loss**	**Training Accuracy**	**Valid Accuracy**
Two Layer	1.518992037455	1.5446843318	0.94359999996821	0.9180000004768
One Layer	1.509279940032	1.5429654560	0.9532666666666	0.919

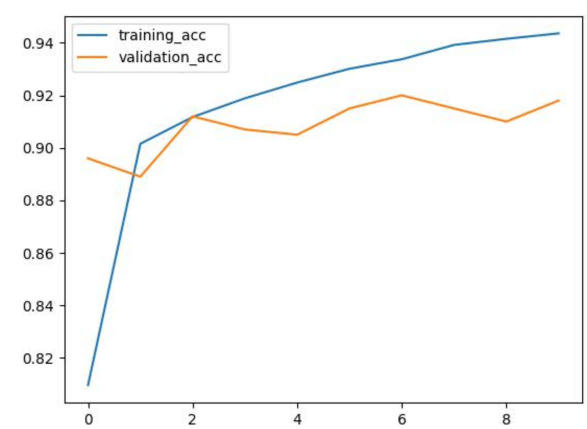
The two layer model and the one layer model have very similar validation loss. However, the one layer model has a slightly lower validation loss compared to the two layer models.

One Layer **Two Layer**









	Testing loss	Testing Accuracies
Two Layer	1.5676022591219774	0.8957415567970836
One Layer	1.56034973119324	0.9034508075483045

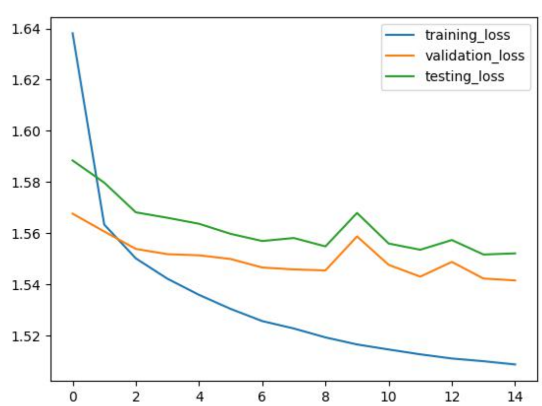
From this table, we can see that the one layer model does in fact have a higher testing accuracy than the two layer model, which aligns with the training and validation results generated before.

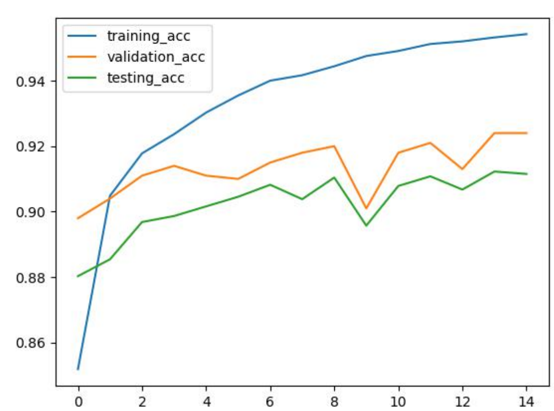
Task 5: Testing the Dropout

Using the best set of parameters in task 2 (optimizer: SGD, learning rate: 0.5), I ran an experiment on the model we used in task 2 with a dropout layer. The below are the summarized results.

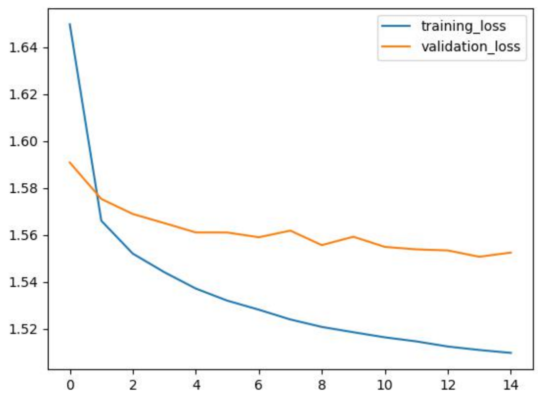
	Training loss	Valid loss	Training Accuracy	Valid Accuracy
With Dropout	1.509791118939	1.5525431972	0.9528666666348	0.911160058474
Without Dropout (From task 2)	1.508800297037	1.5416110725	0.9542	0.923999999523

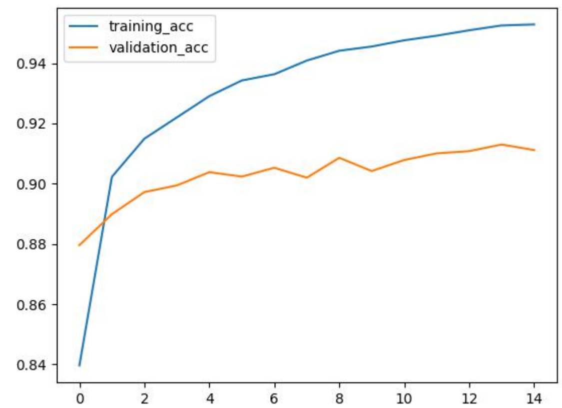
Without Dropout





With Dropout





Even though the final validation accuracy from the model with dropout is slightly higher than that of the model without dropout, the results from the plots are interesting. From the four plots above, it is very clear that the validation curve in the model with a dropout layer is smoother. There are less fluctuations (bumps) in the validation curve, and hence prevent the model from overfitting.