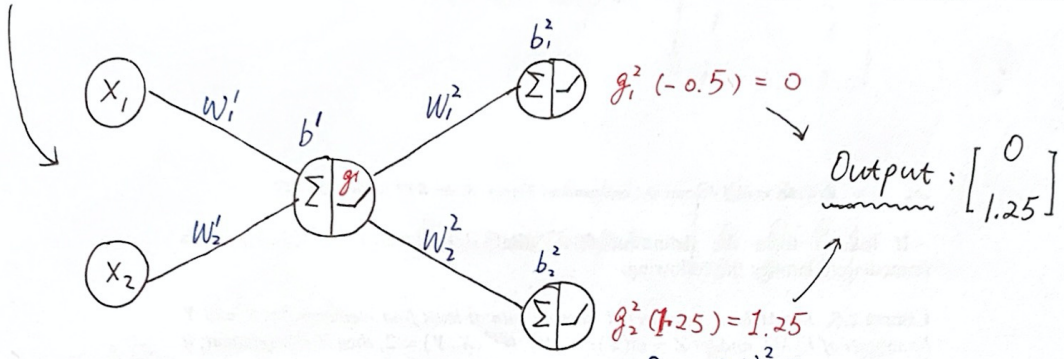
Assignment 3 - DATA448-20S1(C)

Name: Zhen Huang **ID:** 74093323 **Name:** Wen Zhang **ID:** 89352953

Q1 A

Input: $X = \begin{bmatrix} 1.25 \\ -0.5 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

Diagram of the Network



$W' = [0 \quad 1] = [w_1' \quad w_2']$ $\quad W^2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} w_1^2 \\ w_2^2 \end{bmatrix}$

$b' = [1]$ $\quad b^2 = \begin{bmatrix} -0.5 \\ 0.75 \end{bmatrix}$

W' is $(1, 2)$ so in the first hidden layer there are:
1 unit & 2 inputs.

Layer1: $h^1 = g(W^1X + b^1) = g\left([0.1]\begin{bmatrix} 1.25 \\ -0.5 \end{bmatrix} + [1]\right) = g(-0.5 + 1)$

$= g(0.5) = 0.5 \quad (g(z) = \max(0, z))$

W^2 is $(2, 1)$ so the output layer has 2 outputs & 1 input

Layer2: $h^2 = g(W^2X + b^2) = g\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix}[0.5] + \begin{bmatrix} -0.5 \\ 0.75 \end{bmatrix}\right)$

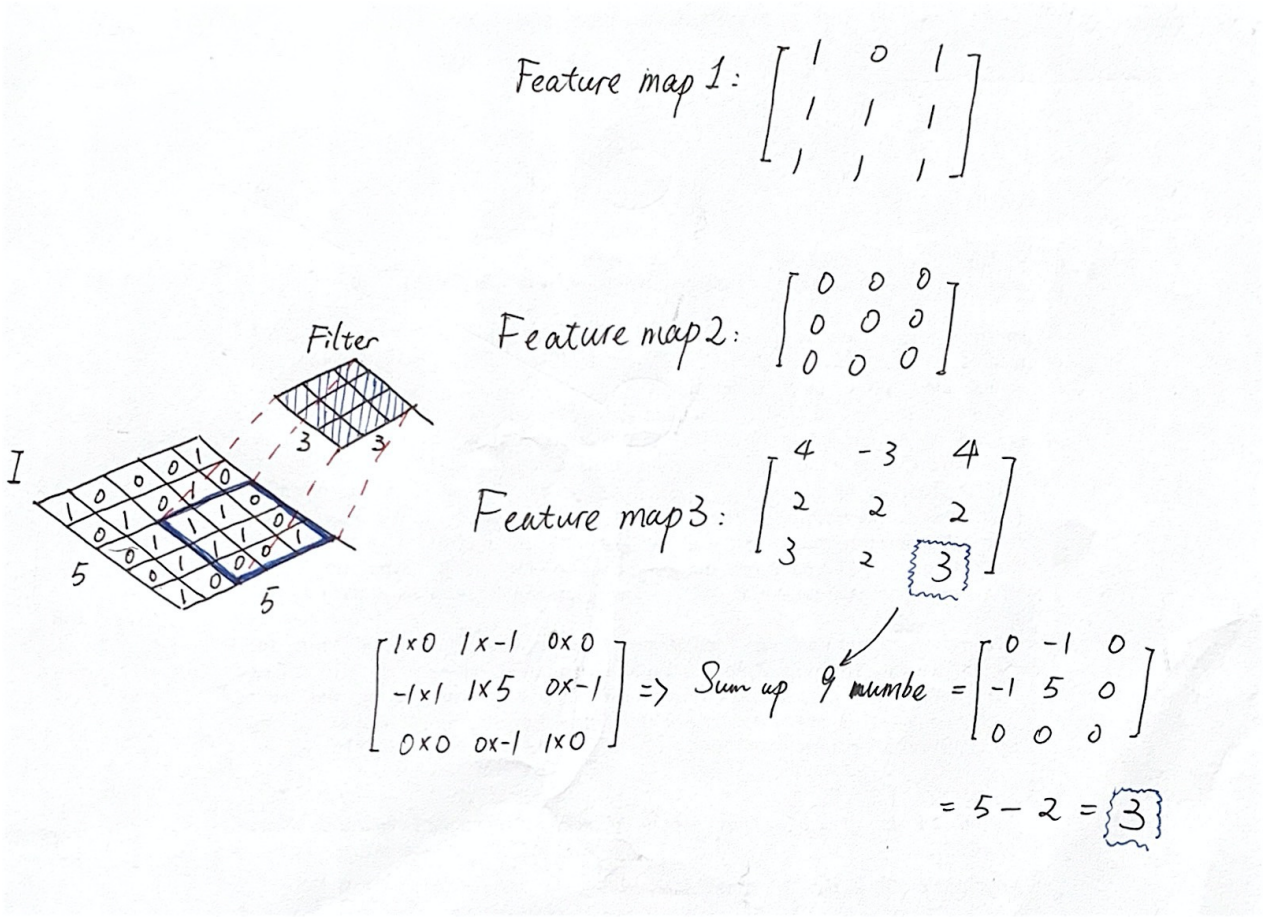
$= g\left(\begin{bmatrix} 0 \\ 0.5 \end{bmatrix} + \begin{bmatrix} -0.5 \\ 0.75 \end{bmatrix}\right) = g\left(\begin{bmatrix} -0.5 \\ 1.25 \end{bmatrix}\right)$

$h_1^2 = g(-0.5) = 0$,

$h_2^2 = g(1.25) = 1.25$

The output is $\begin{bmatrix} 0 \\ 1.25 \end{bmatrix}$, which of dimension 2×1.

Q1 B



Q2

Data Structure

There is a API in Keras to load Fashion-MNIST dataset in a similar way to MNIST. The dataset has 60,000 images for training and 10,000 images for testing, each image has 28*28 pixels with values between 0-255 of each pixel. The images are labelled 0-9 which represent ten classes of fashion wear and accessories.

Pre-process

For each image, the values of pixels are stored as an 28*28 nparray, we need to flatten the array which length is 784. Then, the image values can be input into the neural network later.

The values also need to be scaled by divided 255. This step can potentially improve the performance of gradients descent in our model when handling image classification problems.

Basic model

A simple one hidden layer neural network is built in the initial exploration. There are 32 neurons in the hidden layer using sigmoid as the activation function. The output layer should have ten neurons representing ten classes, and the activation function is softmax which is suitable for the multi-classification problem.

The test loss is 0.51, test accuracy is 0.823, which is quite well for the initial model, and it is better than a null model.

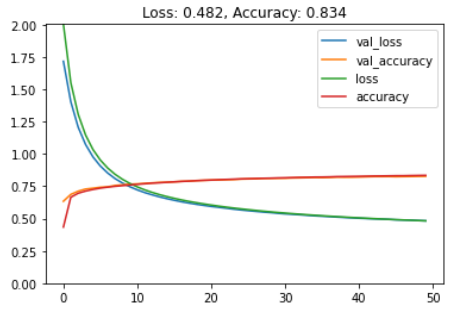
As the figure shown below, there is no big gap between validation and test metrics which suggest no overfitting here.

The "wider" neuron network, such as 64 and 128 neurons in one hidden layer, is also compared with the basic model. The metrics are not improved. Thus, the basic model can be a baseline for the succeeding attempts.

Table 1 Basic model structure

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 32)                25120
_________________________________________________________________
dense_19 (Dense)             (None, 10)                330
=================================================================
Total params: 25,450
Trainable params: 25,450
Non-trainable params: 0
_________________________________________________________________
```

```
Test loss: 0.51
Test accuracy: 0.823
<Figure size 576x432 with 0 Axes>
```



Params

Params in each layer relat to the number of the inputs, neurons and outputs. In the basic model, Input layer has 784 flattened pixel values. The first hidden layer has 25120 params (784 *32 = 25088 weights; 25088 + 32 bias = 25120). The output layer has 10 nodes * 32 inputs from hidden layer, thus 330 params in this layer (320 weights + 10 bias).

Compile and Fit

The sequencial layers need to be compile usually with optimizer, loss function and metrics. Here I choose sgd, categorical_crossentropy and accuracy respectively.

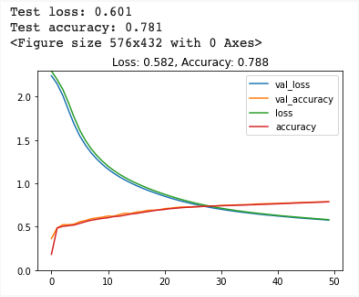
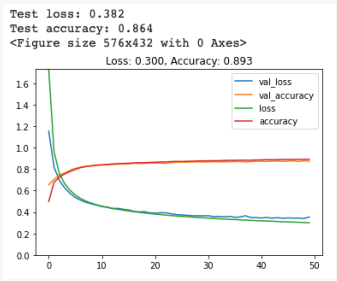
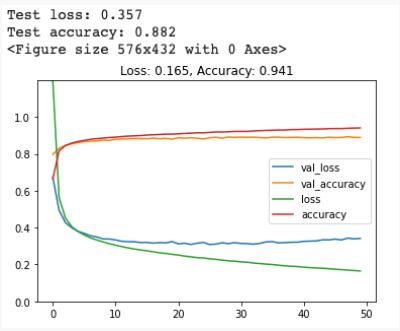
The model is trained by use fit function, three hyperparameters pre-assigned due to early stage. The batch size is 128, epochs is 50 and validation set is 10% of the training dataset. Mini-batch helps to speed up the training process (default 32) without too many noise in gradient descent. Here we update weights by 54000/128=421 times per epoch.

Deeper Neuron Network

By increasing the number of neurons to 64 and adding one more hidden layers with 32 neurons, a deeper neuron network did not performs better than the basic model at the begining.

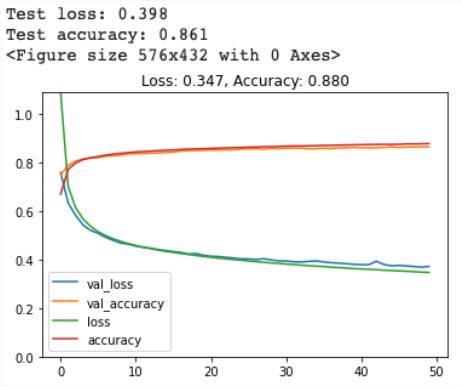
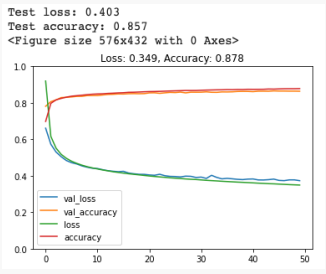
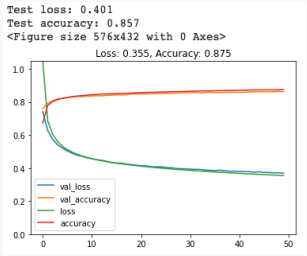
Optimizer and learning rate

The batch size, epochs and validation split keep the same as in basic model to make a fair comparison between models and the effect of optimizers and learning rate. In the early stage at the 10th epoch, the validation loss of Adam model has lower than 0.4.

Table 2 Comparison with optimizers		
SGD(lr=0.01)	SGD(lr=0.1)	Adam(lr=0.001)
		
The loss and accuracy are not flat at end of 50 epochs. It suggests the model need more training to improve. But it also can be optimised by changing learning rate.	The faster learning rate speeds up the converge, and the deeper model has lower loss and higher accuracy than basic model without obvious overfitting.	Adm is a stochasitc gradient descent method. The accuracy hits 0.95. However, there is a siginificatn gap between test and validation loss which means the model overfits the data.

Activation Function

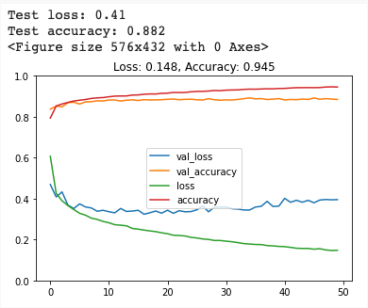
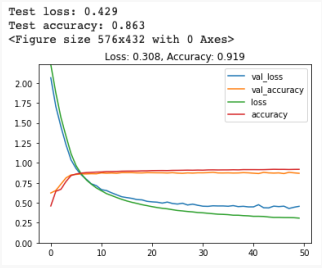
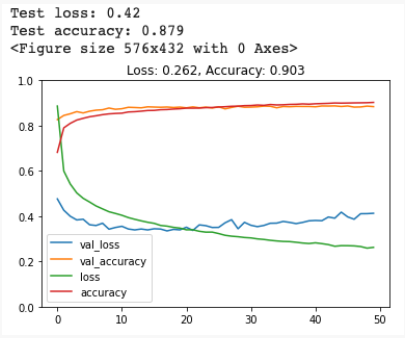
The activation function can also affect the model performance. ReLU has a potential advantage over sigmoid in terms of converge and back propogation. SELU and Tanh also have similar performance with one hidden layer and 128 nodes, both functions are better than sigmoid and sightly not as good as ReLU.

Table 3 Comparison with activations		
ReLU (Basic model)	SELU (Basic model)	Tanh(Basic model)
		
ReLU improves the basic model performance, which suggests that ReLU is suitable for this multi-classification problem over sigmoid.	Scaled Exponential Linear Unit (SELU) gets similar performance and also improves over sigmoid.	Hyperbolic tangent activation function also has an average preformance of three methods.

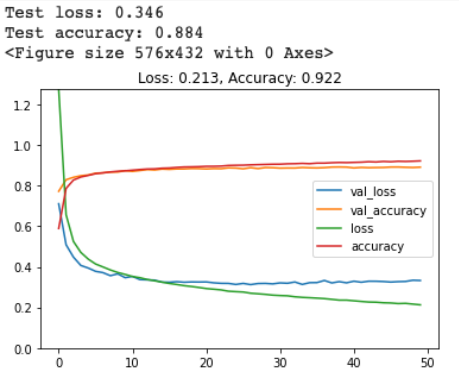
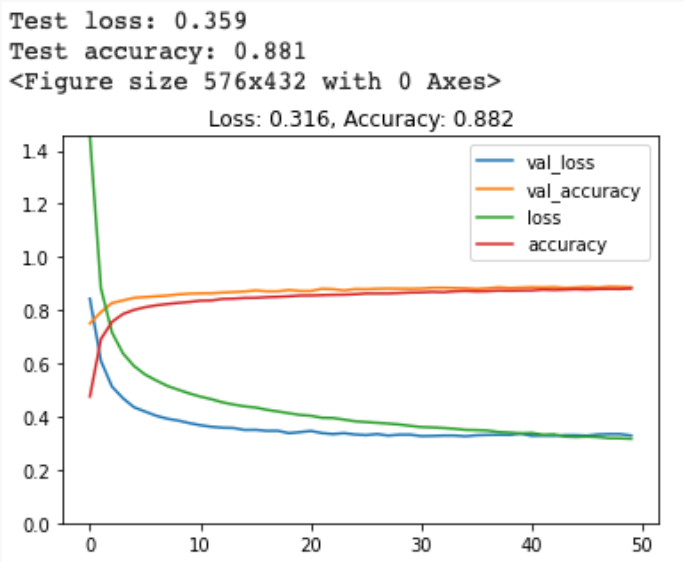
Dropout and Regularisation

Reducing the number of epochs can also avoid overfitting by stopping the training process early before it happens. The less epochs can reduce training time which is another advantage. When the validation loss no longer decreases after 10 or 20 epochs, the training process can be called to stop.

The other ways to prevent overfitting include data augmentation (kind of resampling in machine learning) and regularization.

Table 4 Overfitting in Deeper constructer		
ReLU (Deeper) 1st layer: 64 units 2nd layer:32 units	ReLU (Deeper with L2 Regularisation) 1st layer: 64 units 2nd layer:32 units l2 = 0.01@2nd layer outputs	ReLU (Deeper with dropout = 0.5) 1st layer: 64 units 2nd layer:32 units dropout = 0.5
		
Deeper constructure has lower loss, however an overfitting emerges. There is a big gap between validation and test loss.	ReLU has a faster learning speed than sigmoid as its accuracy breach over 0.8 before 10 epochs. L2 regular	With more strict dropout, the overfitting reliefs a bit, but cannot be overcome. This suggest more sensible model to be investigated.

Dropout can prevent overfitting in a deeper neural network which is similar with regularisation in predicting model. Dropout is no a hidden layer but randomly choosing some nodes in the previous hidden layer.

Table 5 Dropout effects in different optimizers	
Adam(lr=0.001 Dropout(0.2)	Nadam(lr=0.001) Dropout(0.2)
	
Adding one dropout slightly reduce the overfitting of previous model. The gap between test and validation loss narrowed.	Another optimizer Nadam is Adam with Nesterov momentum. The model has similar performance which suggests the potential maximum capacity of the constructer we build.

Convolutional Neuron Network

Although we do not confront the problem of too many parameters in previous model, the convolutional neuron network is still worth to try in this study. Because CNN has been proved efficient in many pracitcal tasks with a deep structure of multiple convolutional layer and maxpooling.

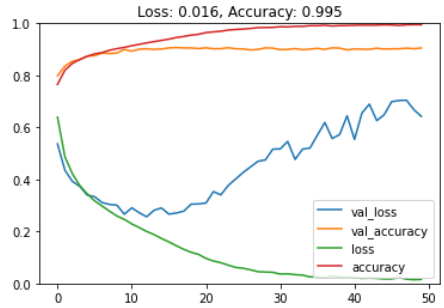
The parameters in a Conv2d layer is different with the previous hidden layers in previous model. For example, in the first Conv2d layer, 320 parameters is 32 filter * 9 weighs in each kernel * 1 grey channel + 32 bias for each filters. The max pooling has no parameters, in this model its size (2, 2) shrinks the image size from 26*26 to 13*13.

Table 5 CNN model structure

Layer (type)	Output Shape	Param #
conv2d_17 (Conv2D)	(None, 28, 28, 32)	320
conv2d_18 (Conv2D)	(None, 26, 26, 32)	9248
max_pooling2d_5 (MaxPooling2	(None, 13, 13, 32)	0
conv2d_19 (Conv2D)	(None, 13, 13, 64)	18496
conv2d_20 (Conv2D)	(None, 11, 11, 64)	36928
max_pooling2d_6 (MaxPooling2	(None, 5, 5, 64)	0
flatten_2 (Flatten)	(None, 1600)	0
dense_36 (Dense)	(None, 512)	819712
dense_37 (Dense)	(None, 10)	5130

Total params: 889,834
Trainable params: 889,834
Non-trainable params: 0

```
Test loss: 3.12e+02
Test accuracy: 0.72
<Figure size 576x432 with 0 Axes>
```



Optimizer: Adam(lr = 0.001)
Batch size: 256
Epochs: 50
Validation: 10%

Obviously, the model overfits in the early stage. But the potential improvement is also shown in the figure that validation and test accuracy hits 0.9 at arond 10th epoch. The training loss and accuracy are quite well in the complex CNN while obvious overfitting the data. The performance of CNN model shows the possibility outperforming the previous full connect deep neuron network. Dropout can also be implemented in CNN to reduce the complexity of the model.

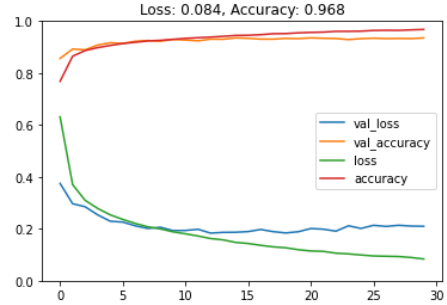
As shown in Table 6, after several tries, three layers of dropout, at ration 0.2, 0.2 and 0.5, are added into the model. The other hyperparameters are as the same except epochs 30 to call an early stop for the training process. The figure at the right side shows the overfitting still emerges at epochs 10, but the gap between test and validation accuracy shrinked.

Table 6 CNN model with Dropout

Layer (type)	Output Shape	Param #
conv2d_36 (Conv2D)	(None, 28, 28, 32)	320
conv2d_37 (Conv2D)	(None, 26, 26, 32)	9248
max_pooling2d_15 (MaxPooling	(None, 13, 13, 32)	0
dropout_5 (Dropout)	(None, 13, 13, 32)	0
conv2d_38 (Conv2D)	(None, 13, 13, 64)	18496
conv2d_39 (Conv2D)	(None, 11, 11, 64)	36928
max_pooling2d_16 (MaxPooling	(None, 5, 5, 64)	0
dropout_6 (Dropout)	(None, 5, 5, 64)	0
flatten_8 (Flatten)	(None, 1600)	0
dense_23 (Dense)	(None, 512)	819712
dropout_7 (Dropout)	(None, 512)	0
dense_24 (Dense)	(None, 10)	5130

Total params: 889,834
Trainable params: 889,834
Non-trainable params: 0

```
Test loss: 0.221
Test accuracy: 0.933
<Figure size 576x432 with 0 Axes>
```



Optimizer: Adam(lr = 0.001)
Batch size: 256
Epochs: 30
Validation: 10%

In fact, there are only ten classes in Fashion MINIST dataset and CNN can potentially handle over hundreds classes problems. Thus the model is easy to overfit. With a very strict construct and dropout, the overfitting improved. Compared with the construct above, only two convolutional layers kept and followed by a 0.5 dropout layer, the number of neurons also reduced to keep a "parsimonious" CNN model. The test accuracy hits 0.927 which is quite good at this stage.

Table 6 A Parsimonious CNN

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_64 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
dropout_44 (Dropout)         (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_65 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
dropout_45 (Dropout)         (None, 11, 11, 64)        0
_________________________________________________________________
flatten_21 (Flatten)         (None, 7744)              0
_________________________________________________________________
dense_49 (Dense)             (None, 128)               991360
_________________________________________________________________
dropout_46 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_50 (Dense)             (None, 10)                1290
=================================================================
Total params: 1,011,466
Trainable params: 1,011,466
Non-trainable params: 0
_________________________________________________________________
```

```
Test loss: 0.209
Test accuracy: 0.927
<Figure size 576x432 with 0 Axes>
```