Australian National University

ASSIGNMENT COVER SHEET

ANU College of Engineering and Computer Science
Australian National University
Canberra ACT 0200 Australia
www.anu.edu.au
+61 2 6125 5254

Student ID	U6483489		
Student Name	Masahito Takeuchi		
Course Code	ENGN8536		
Course Name	Advanced Topics in Mechatronics Systems		
Assignment Number	Assignment1		
Due Date	Sunday, 15th September		
Date Submitted	Sunday, 22nd September	Extension Granted	

Signature タケウチ マサヒト

Assignment1: Artificial Neural Networks

Part1: Design of the network architecture

Figure.1 shows the schematic of the overall architecture for the network that includes input layer, one hidden-layer with ReLU function and output layer. Input layer has 784 neurons derived from 28*28 pixels and output layer has 10 neurons derived from 10 types of classified data. In this project, hidden layer is basically set with 128 neurons.

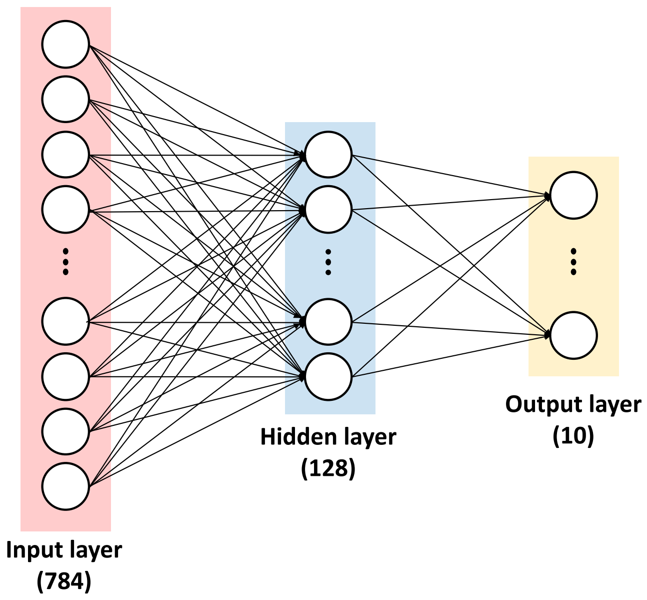


Figure.1: Network architecture

Part2: Derivation of the loss function

Let's derive the derivative of the loss function with respect to the network weights for the hidden unit and the output unit using σ that represents sigmoid function.

Assume that

$$z = w^T x + b$$
$$\hat{y} = \sigma(z)$$

We can get the objective function, the cross-entropy error, as follows.

$$L(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

Since back propagation is based on chain rules, we can obtain

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w_j} \tag{1}$$

First, we can get

$$\begin{aligned}\frac{\partial L}{\partial \hat{y}} &= \frac{\partial}{\partial \hat{y}}(-y\log(\hat{y}) - (1-y)\log(1-\hat{y})) \\ &= -y\frac{\partial}{\partial \hat{y}}(\log(\hat{y}) - (1-y)\frac{\partial}{\partial \hat{y}}\log(1-\hat{y})) \\ &= \frac{-y}{\hat{y}} + \frac{(1-y)}{(1-\hat{y})} \\ &= \frac{\hat{y}-y}{\hat{y}(1-\hat{y})}\end{aligned} \tag{2}$$

Next, we can get

$$\frac{\partial \hat{y}}{\partial z} = \frac{\partial}{\partial z}\sigma(z)$$

$$
\begin{aligned}
&= \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) \\
&= \frac{1}{(1+e^{-z})^2}\frac{\partial}{\partial z}(1+e^{-z}) \\
&= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1}{(1+e^{-z})}\frac{e^{-z}}{(1+e^{-z})} \\
&= \sigma(z)\frac{e^{-z}}{(1+e^{-z})} \\
&= \sigma(z)\left(1-\frac{1}{1+e^{-z}}\right) \\
&= \sigma(z)\big(1-\sigma(z)\big) \\
&= \hat{y}(1-\hat{y}) \qquad (3)
\end{aligned}
$$

Lastly, we can get

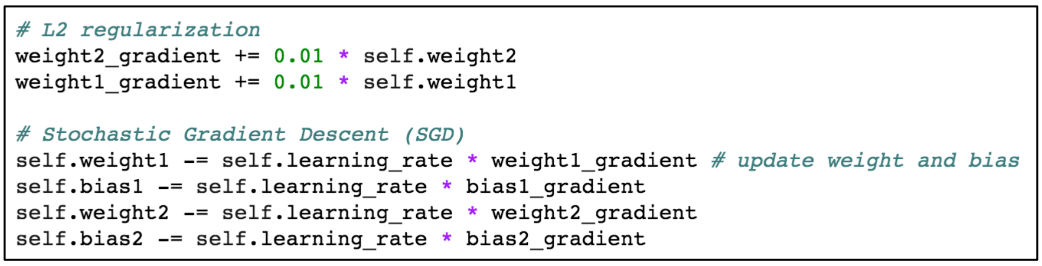
$$
\begin{aligned}
\frac{\partial z}{\partial w_j} &= \frac{\partial}{\partial w_j}(w^T x + b) \\
&= \frac{\partial}{\partial w_j}(w_0 x_0 + \dots + w_n x_n + b) \\
&= \frac{\partial}{\partial w_j}(w_n x_n) \\
&= w_n \qquad (4)
\end{aligned}
$$

Therefore, we can derive

$$
\begin{aligned}
\frac{\partial L}{\partial w_j} &= \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w_j} \qquad (1) \\
&= (\hat{y}-y)w_n \qquad (\because\ equation\ (2), (3), (4))
\end{aligned}
$$

Part3: Building the neural network from scratch

As mentioned before, hidden layer is basically set with 128 neurons. This program includes forward / backward propagation for the network. Also, stochastic gradient descent (SGD) and the concept of L2 regularization are implemented in this program as Figure.2 shows. The detailed code are implemented by Main.ipynb, function.py and network.py in the folder of Part3 in this assignment1 submission folder.



```
# L2 regularization
weight2_gradient += 0.01 * self.weight2
weight1_gradient += 0.01 * self.weight1

# Stochastic Gradient Descent (SGD)
self.weight1 -= self.learning_rate * weight1_gradient # update weight and bias
self.bias1 -= self.learning_rate * bias1_gradient
self.weight2 -= self.learning_rate * weight2_gradient
self.bias2 -= self.learning_rate * bias2_gradient
```

Part4: Report of the performance of the network

The detailed code are implemented and rebuilt by several codes using PyTorch, in the folder of Part4.

(a) The loss function on the training set

Based on the Part3 code, the loss function on the training set is described in Figure.3.

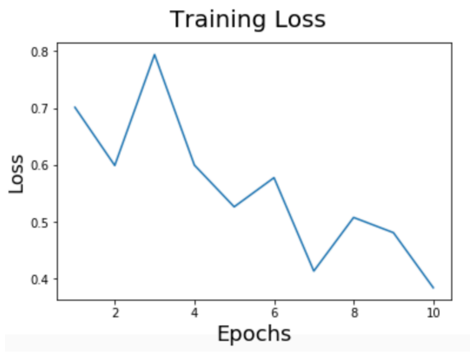


Figure.3: The loss function on the training set

(b) The loss function on the validation set

Based on the Part3 code, the loss function on the training set is described in Figure.4.

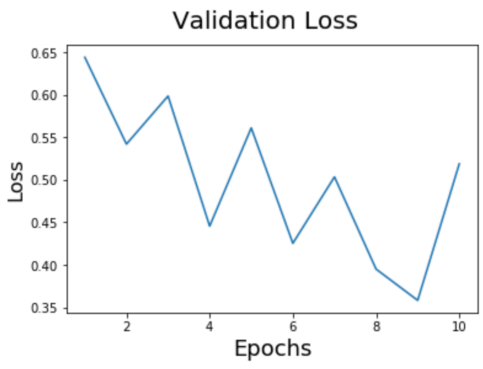
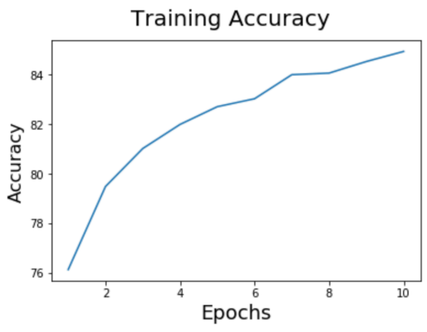


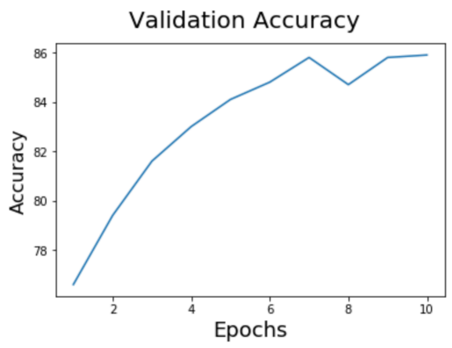
Figure.4: The loss function on the validation set

(c) Comparison of the learning rate

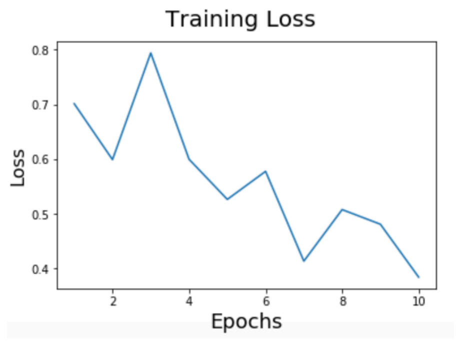
Different learning rate in SGD are compared in Figure 5, Figure.6 and Figure.7, respectively.



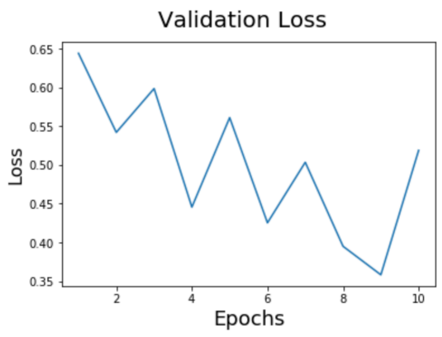
(a) Training accuracy



(b) Validation accuracy

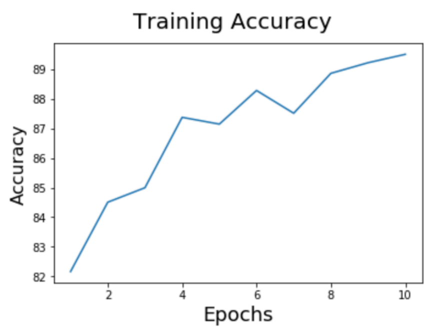


(c) Training loss

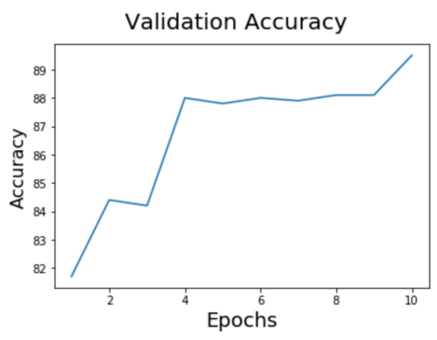


(d) Validation loss

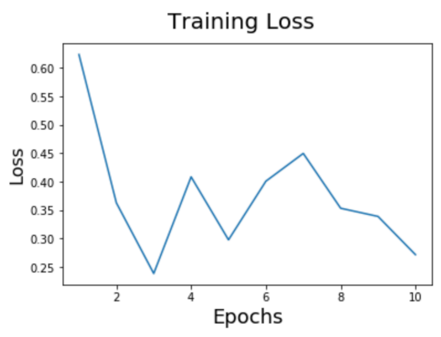
Figure.5: SGD (learning rate=0.01)



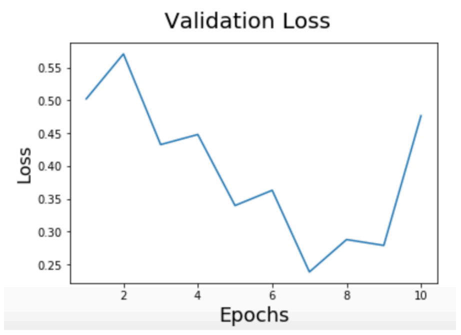
(a) Training accuracy



(b) Validation accuracy

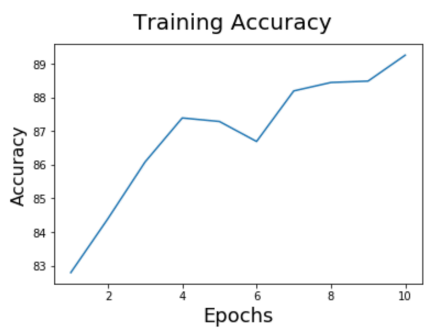


(c) Training loss

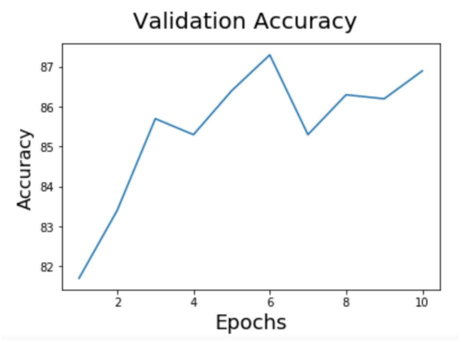


(d) Validation loss

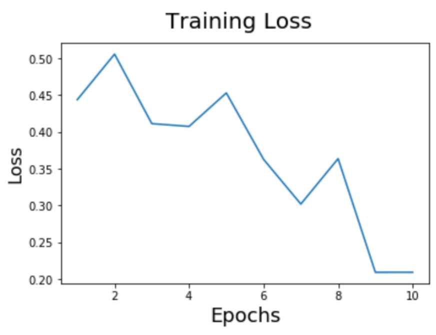
Figure.6: SGD (learning rate=0.1)



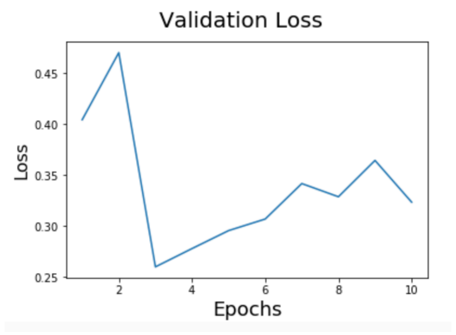
(a) Training accuracy



(b) Validation accuracy



(c) Training loss

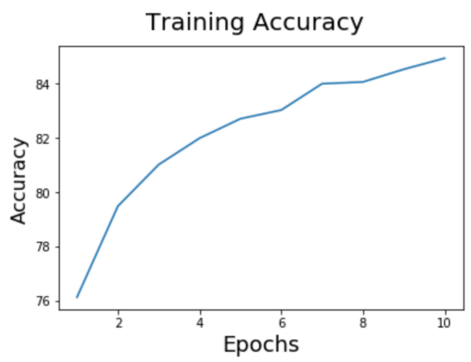


(d) Validation loss

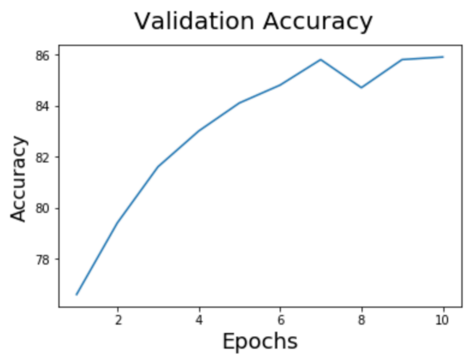
Figure.7: SGD (learning rate=0.3)

Learning rate is a hyperparameter that controls how much to change the model with regards to an error each time when the model weights are updated. If the rate is too small, it may result in a long training process that could get stuck. On the other hand, if the rate is too large, it may result in learning too fast or an unstable training process. In this project, as shown in the above figures, learning rate=0.3 performs the best accuracy and loss score.

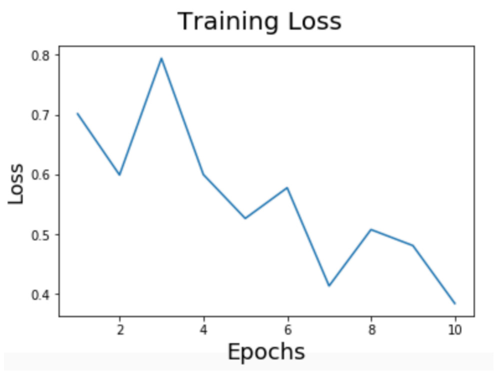
(d) Implementation of a momentum term
Momentum term in SGD helps accelerate gradients vectors in the right directions, thus leading to faster converging. Figure.8 shows the result of each accuracy and loss in SGD without momentum, while Figure.9 shows the result of each accuracy and loss in SGD with momentum. Comparing each model, SGD with momentum represents better accuracy and loss than SGD without momentum.



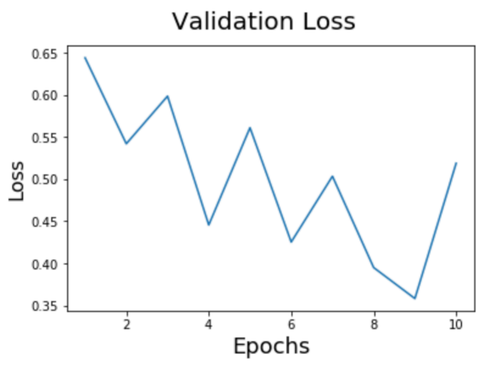
(a) Training accuracy



(b) Validation accuracy

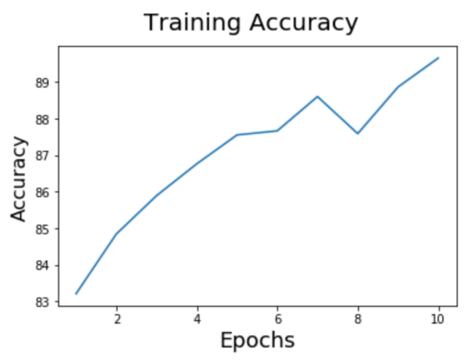


(c) Training loss

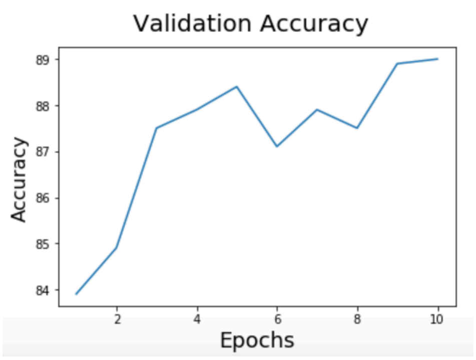


(d) Validation loss

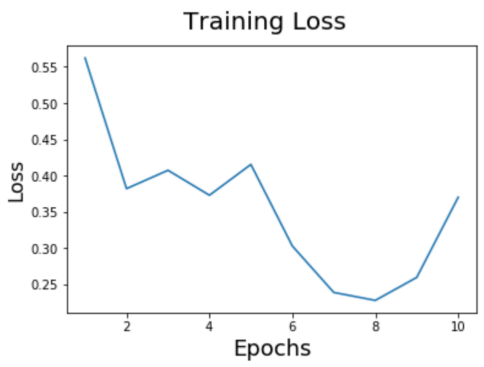
Figure.8: SGD without momentum (learning rate=0.01)



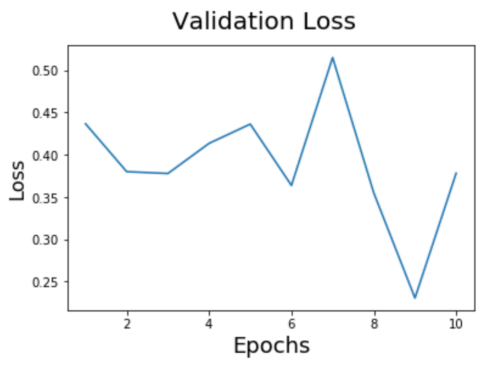
(a) Training accuracy



(b) Validation accuracy



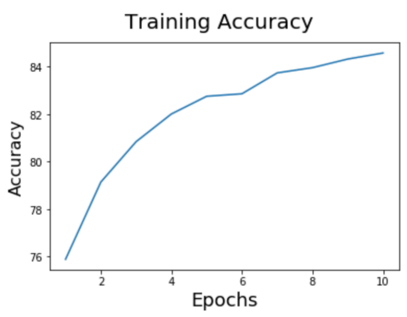
(c) Training loss



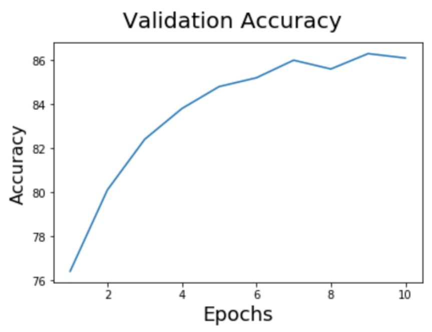
(d) Validation loss

Figure.9: SGD with momentum (learning rate=0.01)

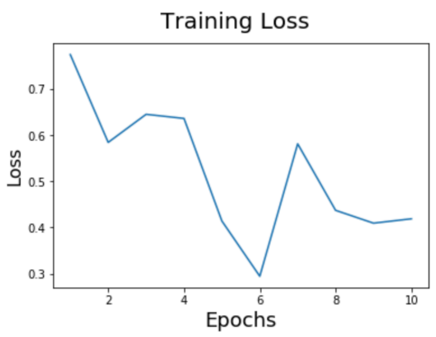
(e) Comparison of the number of hidden units
Figure.8 shows the model with 128 neurons in hidden layer unit. Figure.10, Figure.11 and Figure.12 show the result of different the number of hidden units.



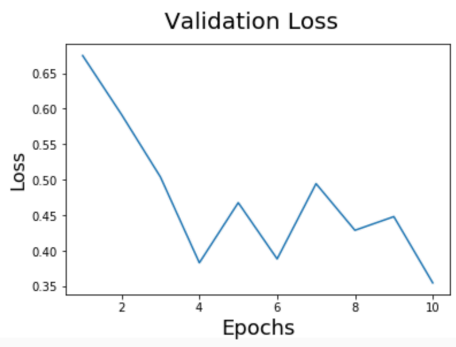
(a) Training accuracy



(b) Validation accuracy

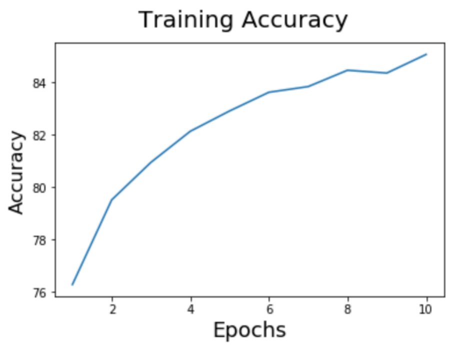


(c) Training loss

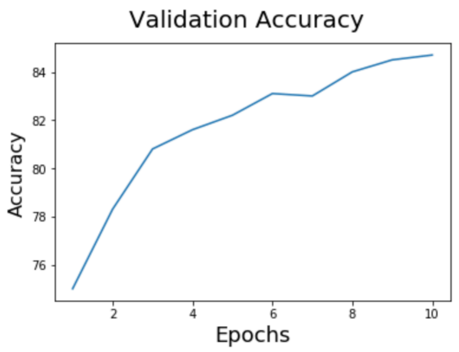


(d) Validation loss

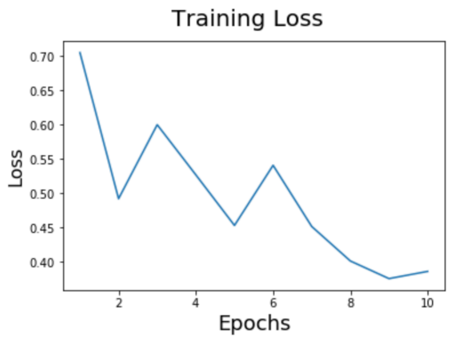
Figure.10: 64 neurons in hidden layer



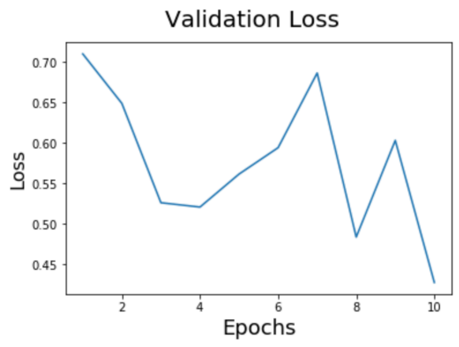
(a) Training accuracy



(b) Validation accuracy

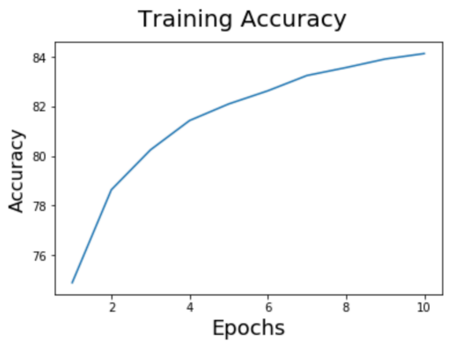


(c) Training loss

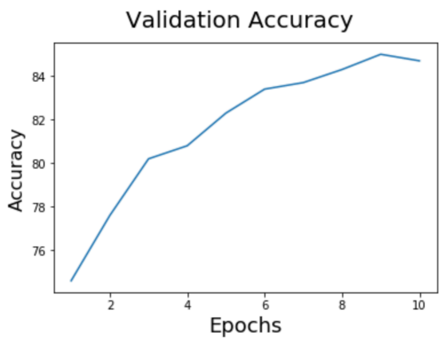


(d) Validation loss

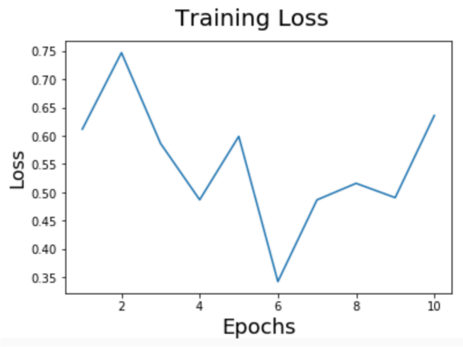
Figure.11: 256 neurons in hidden layer



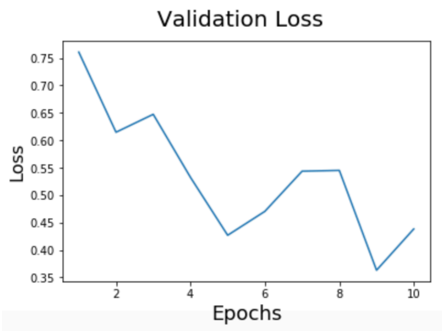
(a) Training accuracy



(b) Validation accuracy



(c) Training loss



(d) Validation loss

Figure.12: 32 neurons in hidden layer

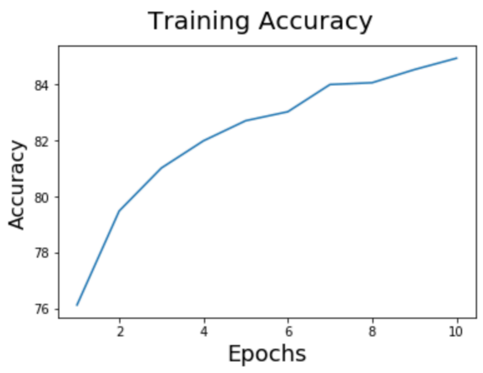
As shown the figures, looking at the training accuracy, the more units the hidden layer has, the higher training accuracy the network has. However, this tendency does not correspond to the validation accuracy. Therefore, it can be said that it is challenging to find the optimized number of units in a hidden layer.

(f) Best performing model
Several models are tested and each result is described in Table.1.

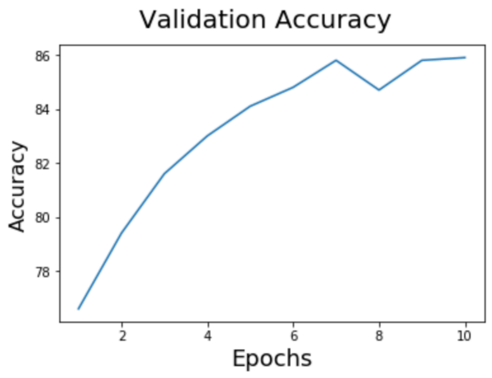
Table.1: Performance comparison in different network models

Test No.	Hidden layer units	SGD learning rate	Mometum	Training accuracy	Validation accuracy	Training loss	Validation loss	Note
1	128	0.01	Yes	89.7	89.0	0.23	0.23	
2	128	0.01	No	85.0	85.9	0.38	0.36	Initial model
3	128	0.1	No	89.5	89.5	0.24	0.24	Best model
4	128	0.3	No	89.2	86.9	0.21	0.26	
5	64	0.01	No	84.6	86.1	0.29	0.35	
6	256	0.01	No	85.1	84.7	0.37	0.43	
7	32	0.01	No	84.1	85.0	0.34	0.36	
8	64	0.01	Yes	89.0	88.3	0.27	0.28	
9	64	0.1	No	89.1	87.1	0.23	0.28	

Looking at accuracy, the model of No.3 (hidden layer neurons=128, SGD learning rate=0.1, no momentum term) has the highest accuracy. Figure.13 shows each result of initial model which has 85.9% accuracy in validation, while Figure.14 shows each result of best performance model which has 89.5% accuracy in validation.

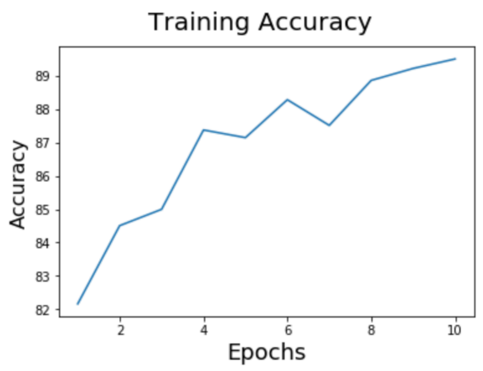


(a) Training accuracy

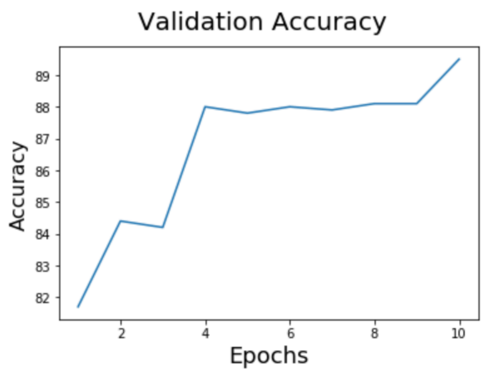


(b) Validation accuracy

Figure.13: Initial model
(hidden layer neurons=128, SGD learning rate=0.01, no momentum term)



(a) Training accuracy



(b) Validation accuracy

Figure.14: Best performance model
(hidden layer neurons=128, SGD learning rate=0.1, no momentum term)