2.5 Model shootout

In this section, we will set up the "vanilla" perceptron model in the begin, and further to use above models that we have covered so far to observe difference, in order to choose which model is better for training data and realize the real conceptions of these models. Some codes are break into two lines due to the limitation of page width to display in one line.

Calculate the accuracy

```
# Set the function and Check whether it is a bag or not
IsBag(w, b, x) = w'x + b >= 0

function accuracy(w, b, y, x)
# w: [n,]
# b: scalar
# y: entire image label [m,]
# x: entire image set [3D]
    correct = 0
    for i in 1 : length(y)
        yhat = IsBag(w, b, vec(x[:, :, i]))
        if yhat == y[i]
            correct = correct + 1
        end
    end

    correct / length(y) * 100
end
```

2.5.1 Vanilla perceptron model (BaseLine model)

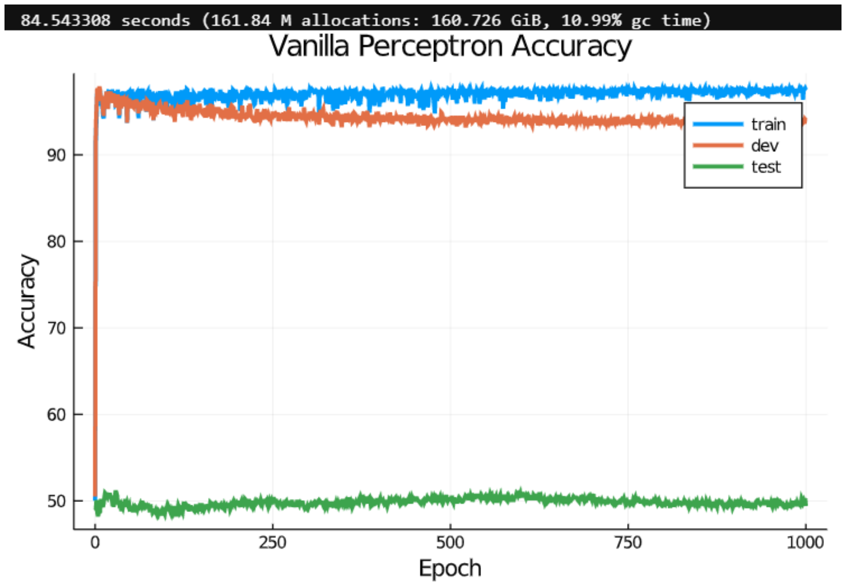
Question 1 and 2:

```
# Define the vector w dimension and scalar b
w = zeros(784)
b = 0.0

# Define the epoch and each dataset list
epochs = 1000
train_accuracy = []
dev_accuracy = []
test_accuracy = []

# Initialize the vector w and scalar b
append!(train_accuracy, accuracy(w, b, trainys, trainxs))
append!(dev_accuracy, accuracy(w, b, devys, devxs))
append!(test_accuracy, accuracy(w, b, testys, testxs))

# Calculate the execution time
@time begin
    # Training the Data
    for iteration in 1 : epochs
        for i in 1 : length(trainys)
            x = vec(trainxs[:, :, i])
            y = trainys[i]
            IsBag(w, b, x) != y || continue
            # If correct, no any update, else improve the w and b
            if y == 1
                w = w .+ x
                b = b + 1
            else
```

```
                w = w .- x
                b = b - 1
            end
        end
        # Append the data into the list
        append!(train_accuracy, accuracy(w, b, trainys, trainxs))
        append!(dev_accuracy, accuracy(w, b, devys, devxs))
        append!(test_accuracy, accuracy(w, b, testys, testxs))
    end
end
```

Accuracy plot

```
# Plot the picture
plot(0 : 1000, [train_accuracy, dev_accuracy, test_accuracy],
title = "Vanilla Perceptron Accuracy", label = ["train" "dev" "test"], lw = 3)

xlabel!("Epoch")
ylabel!("Accuracy")
```



```
84.543308 seconds (161.84 M allocations: 160.726 GiB, 10.99% gc time)
```

Figure 8: The accuracy plot on the training, validation, test set and time consuming.

Condition of stop training

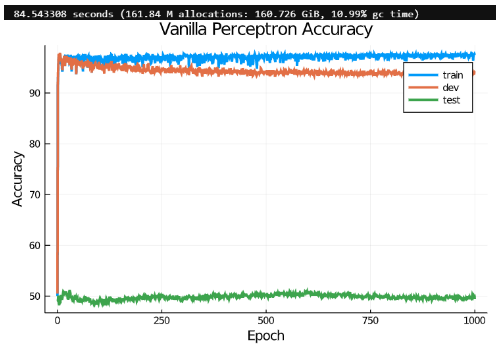
Because of the plot result, we can notice that the validation set(dev) is still bumpy when execute one thousand iterations, hence, we can make a reasonable speculation that the validation set wouldn't converge and still be bumpy when the epoch is larger than 1000. Hence, if we want to determine when to stop training, we just try to make any reasonable assumptions. For instance, we can take the average every 100 epochs (1 -> 100, 101 -> 200, 201 -> 300, ..., 901 -> 1000) and obtain mean accuracy, when the previous one is larger than the latter one, we can stop to train data.

2.5.2 Analyze, compare and explain which is best model

Question 3 and 4:

Define the conditions

In order to select the best model we have covered so far, we use the above models and define the same epochs(1000), time to train data, observe and compare the each models' efficiency. Finally, we choose the fourth model to be our best model in this assignment, some reasons illustrating my perspective will be shed light on as follows.



```
84.543308 seconds (161.84 M allocations: 160.726 GiB, 10.99% gc time)
```

Figure 9: BaseLine Model
Time consuming: 84.54sec

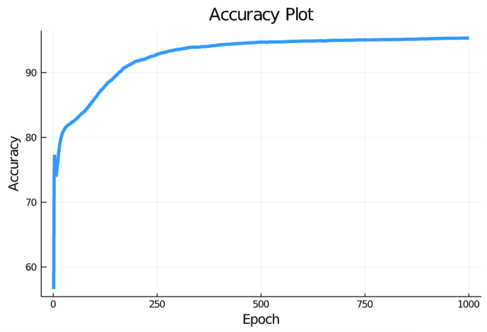


Figure 10: Mean Squared Loss Model
Time consuming: 92.36sec

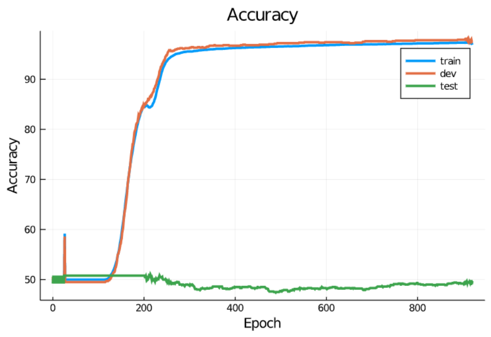


Figure 11: Multiple Layer Model
Time consuming: 38802.47sec

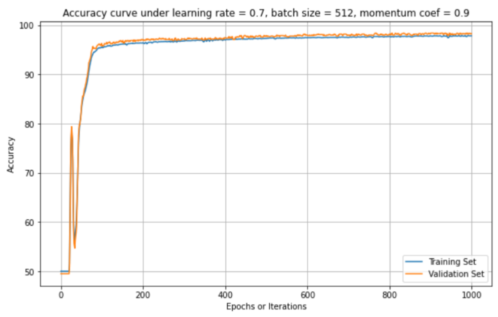


Figure 12: Hyper Parameter Model
Time consuming: 58.89sec

BaseLine Model (Parameter: N / A):

Because of the plot result, we can notice that the validation set(dev) is still bumpy when execute one thousand iterations, hence, we can make a reasonable speculation that the validation set wouldn't converge and still be bumpy when the epoch is larger than 1000. In addition, even though the BaseLine Model training and predicting efficiency are fast, and don't need to set an extra parameter in the model, it still have many disadvantages. Actually, this model is suitable to small size data, if the data are larger than this assignment, it may not obtain the higher accuracy. Moreover, it can't execute in non-linear situation and the model have higher probability to overfit. It is not a stable algorithm.

Mean squared-loss logistic regression Model (Parameter: Learning Rate: 0.3):

We can use the MSE(Mean Square error) to evaluate the variation of the data. When the MSE become smaller than we can obtain the higher accuracy. In addition, the Mean squared-loss logistic regression Model is similar to BaseLine Model. It also can't execute in non-linear situation and it is suitable to small size data. In addition to the previous statements, according to the plot result, we can notice that the accuracy of Mean squared-loss logistic is smaller than other models, besides, it is also trapped in a local extreme. However, compared to BaseLine Model, when the epoch is at 1000, the Model can reach convergence. But, if we want to receive higher accuracy, we need to execute more iterations.

Multiple Layer Model (Parameter: Hidden1: 80, Hidden2: 80, Learning Rate: 0.65):

In fact, Multiple Layer Model have many advantages, such as it can deal with non-linear questions, have parallel processing and stronger learning capacities. Whereas, even though we obtain the higher accuracy in this model, it still have many disadvantages. Firstly, it is pretty hard to define the correct hidden1, hidden2 and learning rate parameters. For the purpose of getting higher accuracy, we need to adjust different combinations. In addition, the Multiple Layer needs to take a long time to train data. When the hidden1, hidden2 parameters are too large, further to cause the learning efficiency be pretty slow. Third, sometimes the Multiple Layer Model will be trapped in a local extreme.

Hyper Parameter Model (Parameter: Learning Rate: 0.7, Batch Size: 512, Momentum Coefficient: 0.9):

In fact, we couldn't know that which kinds of parameters are have more impact on the neural network, as a result, we use the random method to receive more parameters and take them to test model to improve efficiency. In addition, it doesn't need to take extra time to adjust parameters. According to the plot result, even though the result have a little bumpy when the epoch is at 1000, however, it can receive the highest accuracy and smallest time consuming compared with other models. Therefore, we select this model to be our "best" model that we have covered so far.

2.5.3 Best model Loss plot

Question 5:

The loss of this model with "good" hyperparameters on both training and validation sets can be viewed in the following figure.

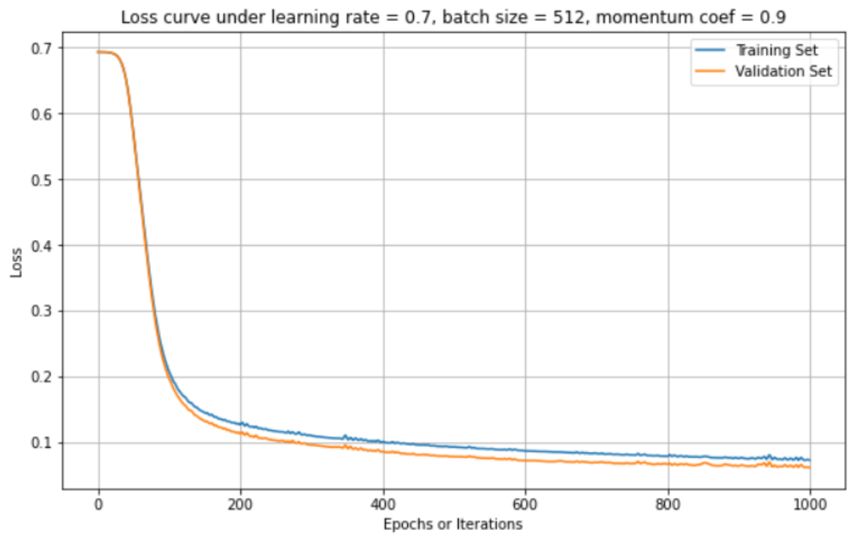


Figure 13: The loss plot on the training and validation set.

2.5.4 Best model Accuracy plot

Question 6:

The accuracy of this model with "good" hyperparameters on both training and validation sets can be viewed in the following figure.

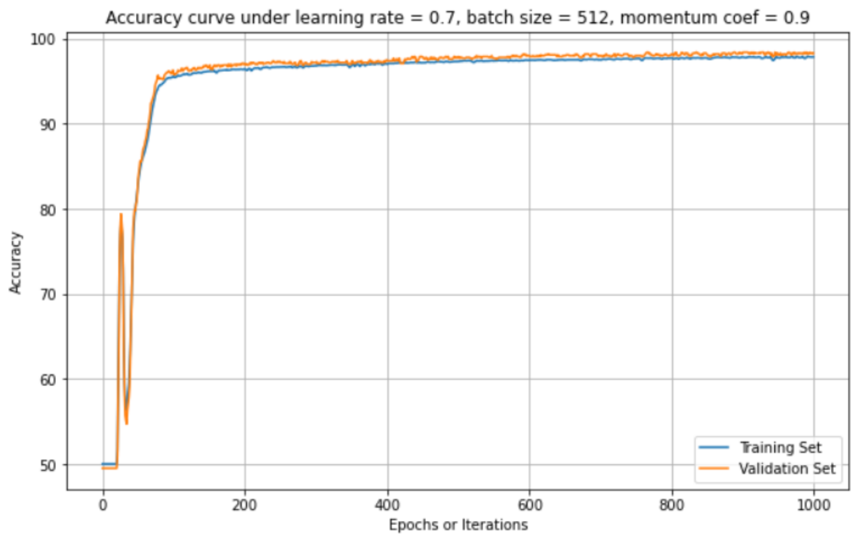


Figure 14: The accuracy plot on the training and validation set.

2.5.5 Best model Accuracy

Question 7:

The best accuracy for validation set is 98.4%, at the same epoch, the accuracy for training set is 97.7%

References

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.