Name(s): Qi Long, Yihong Yang
NetID(s): qilong2, yihongy3
Team name on Kaggle leaderboard: Qi Long 25

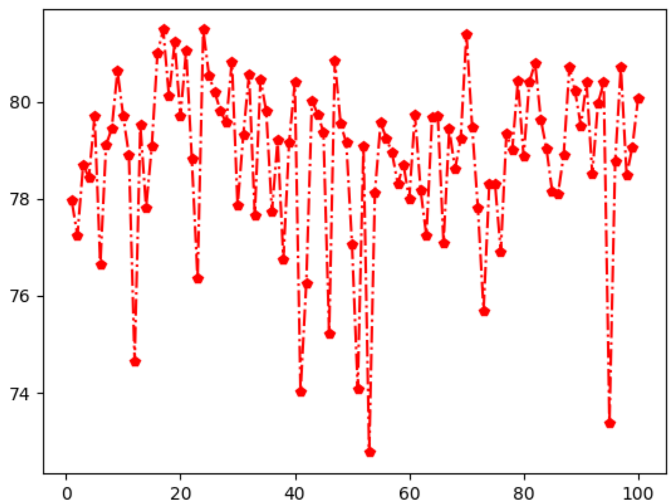
For each of the sections below, your reported test accuracy should approximately match the accuracy reported on Kaggle.

Perceptron

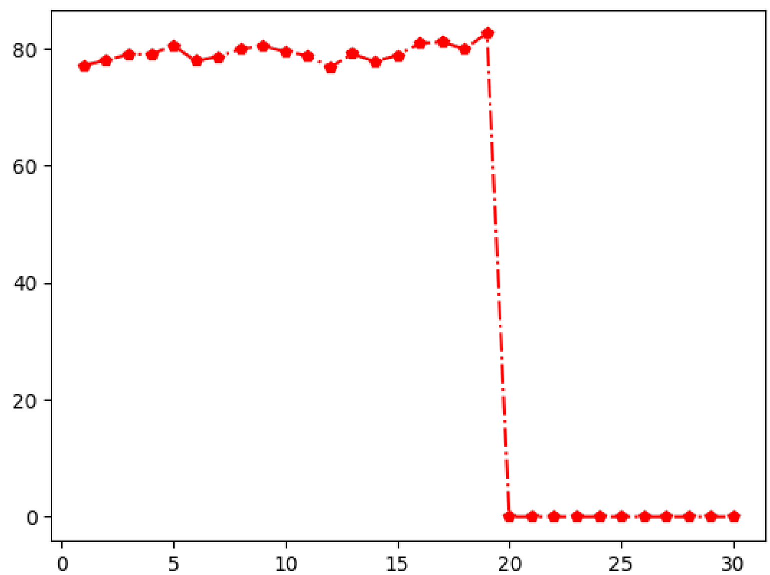
Briefly describe the hyperparameter settings you tried. In particular, you should list the different values for learning rate and number of epochs you tried. You should also mention whether adding a learning rate decay helped and how you implemented this decay. Report the optimal hyperparameter setting you found in the table below. Report your training, validation, and testing accuracy with your optimal hyperparameter setting.

Experiment mainly on Fashion-MNIST:

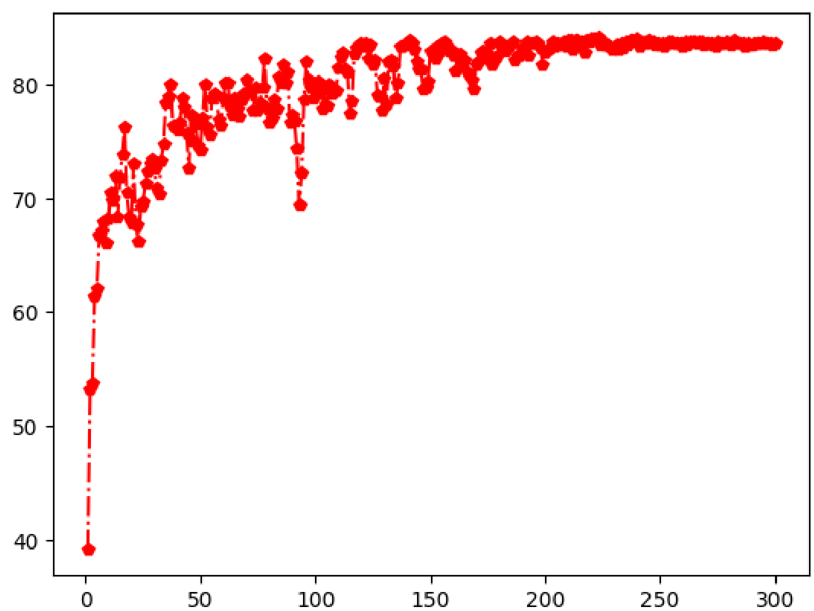
1. Learning rate:
   Lr = [5, 0.5, 0.05, 0.005, 0.0005]
   1) Starting with lr=5 and lr=0.5, the plotting is bouncing back and forth and cannot converge.



   2) Then try lr = 0.05 and lr 0.005, with fixed epochs, lr=0.005 has the best accuracy.
   (Plotting includes early stop)



3) When lr=0.0005, the accuracy takes over 200 epoches to converge.



2. Learning rate decay:
   Adding learning rate decay is helpful. It made the accuracy stable at above 0.84 instead of stable at around 0.80. We interpret it as when model get closer to optimum, it can still make minor progress instead of bouncing around optimum.
   We implemented it by multiplying the rate each epoch. (`self.lr *= 0.95`)
3. Epochs:
   After all the other hyperparameters set, we finally optimize training epochs. Starting with epochs=100, after several runs, our experiment simply set the threshold 0.82 validation dataset accuracy to stop training when model's accuracy decreases and falls below 0.82, so that model can early stop to avoid overfitting. It stops at n_epoch = 19 (as shown above).
4. Other details:

To rise accuracy from around 0.80 to above baseline, other tricks have been tried out.

1) Seed when initialize w: (Make experiment more convenient)
```
np.random.seed(100)
self.w = np.random.random(D+1)
```
2) Add a bias term:
```
self.w = np.random.random(D+1)
xi = np.append(1, X_use_new[i])
```
3) Shuffle dataset:
```
### shuffle training set
permutation = np.random.permutation(X_train.shape[0])
X_use_new = X_use[permutation]
y_use_new = y_use[permutation]
```

These show improvement on overall accuracy, so we keep them.

RICE DATASET

Optimal hyperparameters:	`lr = 0.000001` `n_epochs = 8`
Training accuracy:	0.99706718
Validation accuracy:	0.99670058
Test accuracy:	0.99615067

Fashion-MNIST DATASET

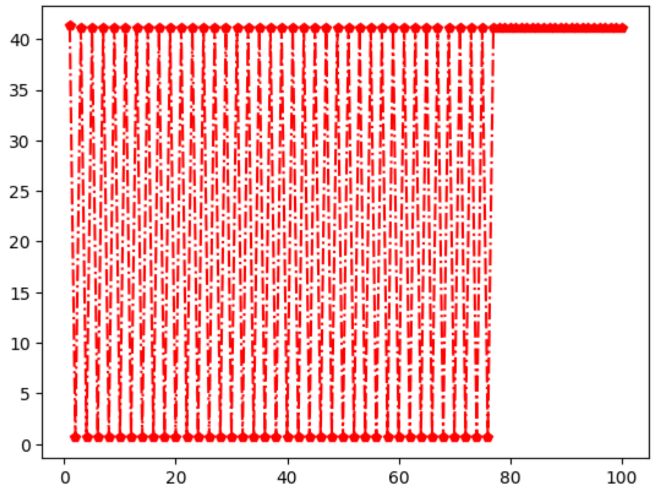
Optimal hyperparameters:	`lr = 0.005` `n_epoches = 19`
Training accuracy:	0.84734000
Validation accuracy:	0.82530000
Test accuracy:	0.81820000

SVM

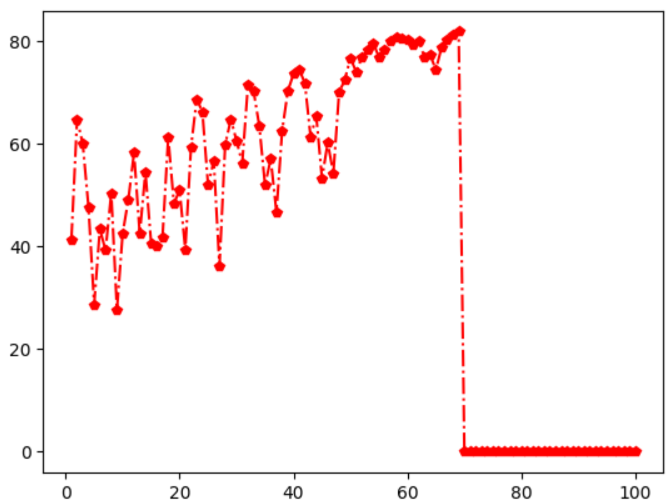
Describe the hyperparameter tuning you tried for learning rate, number of epochs, and regularization constant. Report the optimal hyperparameter setting you found in the table below. Also report your training, validation, and testing accuracy with your optimal hyperparameter setting.

Experiment mainly on Fashion-MNIST:

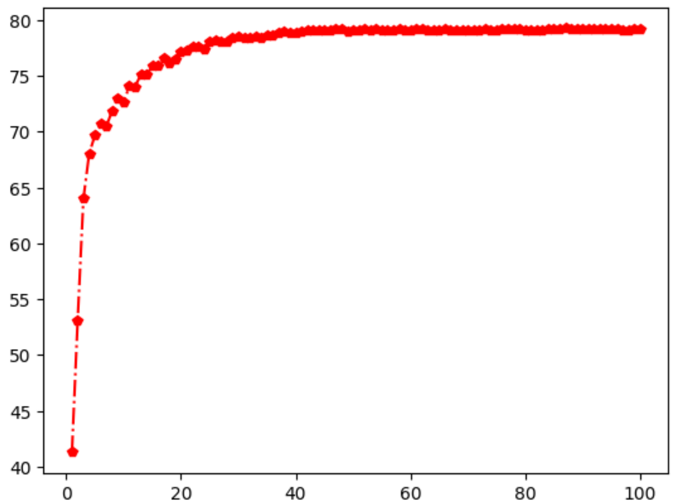
1. Learning rate:
   Lr = [5000, 500, 50, 5, 0.5]
   1) Starting with lr=5 and lr=0.5, it takes over 100 epochs to reach a good accuracy.
   2) Then try lr = 50 and lr=500, with fixed epochs, lr=0.005 has the best accuracy.
   3) When lr=0.0005, the accuracy takes over 200 epoches to converge.
   (*Figures are similar to Perceptron Learning rate experiments*)
2. Learning rate decay:
   Adding learning rate decay is helpful, as discussed in Perceptron experiments.
   We implemented it by multiplying the rate each epoch. (`self.lr *= 0.99`)
3. Regularization constant:
   Reg_const = [1, 0.1, 0.01, 0.001, 0.0001, 0.00001]
   1) For reg_const = 1, it was too large that the accuracy became stable at a low level.



   2) For reg_const = 0.001, the plotting shows a good result. (early stop applied)



   3) For reg_const = 0.00001, the training was slow and became stable at a non-optimal accuracy (<0.8).



4. Batch_size = 1024 works fine, changing it to 512 or others make little difference.
5. Epochs:
   After all the other hyperparameters set, we finally optimize training epochs. Starting with epochs=100, after several runs, our experiment simply set the threshold 0.817 validation dataset accuracy to stop training when model's accuracy decreases and falls below 0.817, so that model can early stop to avoid overfitting. It stops at n_epoch = 69. (as shown above)
6. Other details:
   To rise accuracy from around 0.80 to above baseline, other tricks have been tried out.
   1) Seed when initialize w: (Make experiment more convenient)
   ```
   np.random.seed(100)
   self.w = np.random.random(D)
   ```
   2) Standardize RICE dataset: center data and scale down by maximum.
   ```
   X_mean = np.mean(X_train, axis=0)
   X_use = X_train - X_mean
   X_use_max = np.max(np.absolute(X_use), axis=0)
   X_use = X_use / X_use_max
   ```
   3) Mini-batch SGD: required by assignment.
   ```
   ### Step 2: fetch mini-batch
    row_idxs = np.arange(N)
    np.random.shuffle(row_idxs)
    X_mini = X_use[row_idxs[0:batch_size], :]
    y_mini = y_train[row_idxs[0:batch_size]]
   ```
   These show improvement on overall accuracy, so we keep them.

RICE DATASET

Optimal hyperparameters:	`lr = 0.5`
	`n_epochs = 30`
	`batch_size = 1024`
	`reg_const = 0.001`

Training accuracy:	1.00000000
Validation accuracy:	0.99917514
Test accuracy:	1.00000000

Fashion-MNIST DATASET

Optimal hyperparameters:	`lr = 500`
	`n_epoches = 69`
	`batch_size = 1024`
	`reg_const = 0.001`

Training accuracy:	0.82584000
Validation accuracy:	0.81820000
Test accuracy:	0.81370000

Softmax

Once again, describe the hyperparameter tuning you tried for learning rate, number of epochs, and regularization constant. Report the optimal hyperparameter setting you found in the table below. Also report your training, validation, and testing accuracy with your optimal hyperparameter setting.

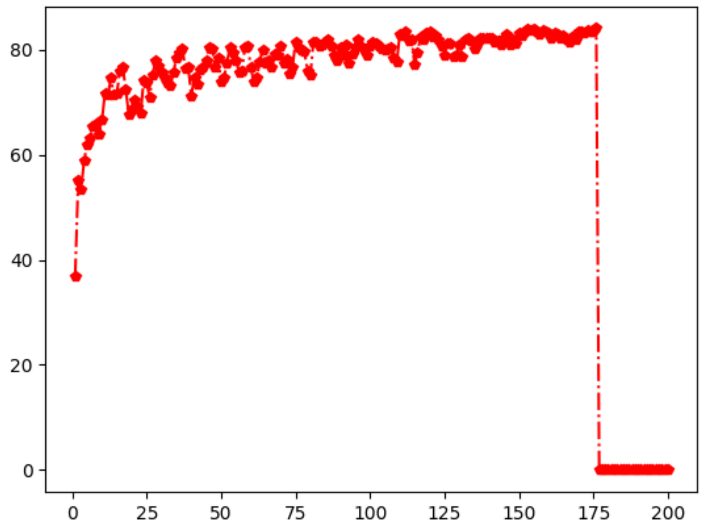
Experiment mainly on Fashion-MNIST:

1. Learning rate:
   Lr = [5, 4, 3.5, 3, 2, 1, 0.5]
   1) Starting with lr=5 and lr=0.5, it works fine.
   2) Since the baseline is not achieved, try further small changes. Lr=3.5 works the best.
   (*Figures are similar to Perceptron Learning rate experiments*)
2. Learning rate decay:
   Adding learning rate decay is helpful, as discussed in Perceptron experiments.
   We implemented it by multiplying the rate each epoch. (`self.lr *= 0.99`)
3. Regularization constant:
   Reg_const = [1, 0.1, 0.01, 0.001, 0.0001, 0.00001]
   1) For reg_const = 1, it was too large that the accuracy became stable at a low level.
   2) For reg_const = 0.001, the plotting shows a good result. (early stop applied)
   3) For reg_const = 0.00001, the training was slow and became stable at a non-optimal accuracy.
   (*Figures are similar to SVM Regularization constant experiments*)
4. Batch_size:
   1) Starting with Batch_size = 1024, it works fine.

2) Since the baseline is not achieved, try further changes. Batch_size = 1500 works the best.

5. Epochs:

After all the other hyperparameters set, we finally optimize training epochs. Starting with epochs=200, after several runs, our experiment simply set the threshold 0.84 validation dataset accuracy to stop training when model's accuracy decreases and falls below 0.84, so that model can early stop to avoid overfitting. It stops at n_epoch = 176.



6. Other details:

To rise accuracy from around 0.80 to above baseline, other tricks have been tried out.

1) Seed when initialize w: (Make experiment more convenient)

```
np.random.seed(100)
self.w = np.random.random(D)
```

2) Standardize RICE dataset: center data and scale down by maximum.

```
X_mean = np.mean(X_train, axis=0)
X_use = X_train - X_mean
X_use_max = np.max(np.absolute(X_use), axis=0)
X_use = X_use / X_use_max
```

3) Mini-batch SGD: required by assignment.

```
### Step 2: fetch mini-batch
row_idxs = np.arange(N)
np.random.shuffle(row_idxs)
X_mini = X_use[row_idxs[0:batch_size], :]
y_mini = y_train[row_idxs[0:batch_size]]
```

These show improvement on overall accuracy, so we keep them.

RICE DATASET

Optimal hyperparameters:	lr = 0.0005 n_epochs = 10 reg_const = 0.001 batch_size = 1500

Training accuracy:	0.90834937
Validation accuracy:	0.90184218
Test accuracy:	0.92438823

Fashion-MNIST DATASET

Optimal hyperparameters:	`lr = 3.5`
	`n_epoches = 176`
	`reg_const = 0.001`
	`batch_size = 1500`

Training accuracy:	0.84958000
Validation accuracy:	0.84020000
Test accuracy:	0.83030000

Logistic

Once again, describe the hyperparameter tuning you tried for learning rate, number of epochs, and threshold. Report the optimal hyperparameter setting you found in the table below. Also report your training, validation, and testing accuracy with your optimal hyperparameter setting.

Experiment on RICE:

1. Learning rate:
   Lr = [5, 0.5, 0.05]
   1) Starting with lr=5 and lr=0.5, lr works the best, reaching good results within 5 epoches.
   (*Figures are similar to Perceptron Learning rate experiments*)
2. Learning rate decay:
   Adding learning rate decay is helpful, as discussed in Perceptron experiments.
   We implemented it by multiplying the rate each epoch. (`self.lr *= 0.95`)
3. Threshold:
   1) Starting from thred = 0.5, it reaches a good result.
   2) When thred is changed to be closer to 0 or 1, it takes more epochs to reach a good result.
4. Epochs:
   After all the other hyperparameters set, we finally optimize training epochs. n_epoches = 10 can already reach a stable accuracy.
5. Other details:
   To rise accuracy from around 0.80 to above baseline, other tricks have been tried out.
   1) Seed when initialize w: (Make experiment more convenient)

```
np.random.seed(100)
self.w = np.random.random(D)
```

These show improvement on overall accuracy, so we keep them.

RICE DATASET

Optimal hyperparameters:	`learning_rate = 0.5`
	`n_epochs = 10`
	`threshold = 0.5`

Training accuracy:	0.97204656
Validation accuracy:	0.97360462
Test accuracy:	0.97360462