CS 444 Assignment-1

February 12, 2023

1 (Optional) Colab Setup

If you aren't using Colab, you can delete the following code cell. This is just to help students with mounting to Google Drive to access the other .py files and downloading the data, which is a little trickier on Colab than on your local machine using Jupyter.

[ ]:
```
# you will be prompted with a window asking to grant permissions
from google.colab import drive
drive.mount("/content/drive")
```

[ ]:
```
# fill in the path in your Google Drive in the string below. Note: do not␣
 ↪escape slashes or spaces
import os
datadir = "/content/assignment1"
if not os.path.exists(datadir):
  !ln -s "/content/drive/My Drive/YOUR PATH HERE/assignment1/" $datadir
os.chdir(datadir)
!pwd
```

[ ]:
```
# downloading Fashion-MNIST
import os
os.chdir(os.path.join(datadir,"fashion-mnist/"))
!chmod +x ./get_data.sh
!./get_data.sh
os.chdir(datadir)
```

2 Imports

[12]:
```
import random
import numpy as np
from data_process import get_FASHION_data, get_RICE_data
from scipy.spatial import distance
from models import Perceptron, SVM, Softmax, Logistic
from kaggle_submission import output_submission_csv
%matplotlib inline

# For auto-reloading external modules
```

```
# See http://stackoverflow.com/questions/1907993/
  ↪autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

3 Loading Fashion-MNIST

In the following cells we determine the number of images for each split and load the images. TRAIN_IMAGES + VAL_IMAGES = (0, 60000] , TEST_IMAGES = 10000

[13]:
```
# You can change these numbers for experimentation
# For submission we will use the default values
TRAIN_IMAGES = 50000
VAL_IMAGES = 10000
normalize = True
```

[115]:
```
data = get_FASHION_data(TRAIN_IMAGES, VAL_IMAGES, normalize=normalize)
X_train_fashion, y_train_fashion = data['X_train'], data['y_train']
X_val_fashion, y_val_fashion = data['X_val'], data['y_val']
X_test_fashion, y_test_fashion = data['X_test'], data['y_test']
n_class_fashion = len(np.unique(y_test_fashion))
```

4 Loading Rice

[116]:
```
# loads train / test / val splits of 80%, 20%, 20%
data = get_RICE_data()
X_train_RICE, y_train_RICE = data['X_train'], data['y_train']
X_val_RICE, y_val_RICE = data['X_val'], data['y_val']
X_test_RICE, y_test_RICE = data['X_test'], data['y_test']
n_class_RICE = len(np.unique(y_test_RICE))

print("Number of train samples: ", X_train_RICE.shape[0])
print("Number of val samples: ", X_val_RICE.shape[0])
print("Number of test samples: ", X_test_RICE.shape[0])
```

```
Number of train samples:  10911
Number of val samples:  3637
Number of test samples:  3637
```

4.0.1 Get Accuracy

This function computes how well your model performs using accuracy as a metric.

[16]:
```
def get_acc(pred, y_test):
    return np.sum(y_test == pred) / len(y_test) * 100
```

5 Perceptron

Perceptron has 2 hyperparameters that you can experiment with: - **Learning rate** - controls how much we change the current weights of the classifier during each update. We set it at a default value of 0.5, but you should experiment with different values. We recommend changing the learning rate by factors of 10 and observing how the performance of the classifier changes. You should also try adding a **decay** which slowly reduces the learning rate over each epoch. - **Number of Epochs** - An epoch is a complete iterative pass over all of the data in the dataset. During an epoch we predict a label using the classifier and then update the weights of the classifier according to the perceptron update rule for each sample in the training set. You should try different values for the number of training epochs and report your results.

You will implement the Perceptron classifier in the **models/perceptron.py**

The following code: - Creates an instance of the Perceptron classifier class - The train function of the Perceptron class is trained on the training data - We use the predict function to find the training accuracy as well as the testing accuracy

5.1 Train Perceptron on Fashion-MNIST

[570]:
```
lr = 0.1
n_epochs = 50

percept_fashion = Perceptron(n_class_fashion, lr, n_epochs)
percept_fashion.train(X_train_fashion, y_train_fashion)
```

```
Epoch1, acc:78.404, learning rate:0.1
Epoch11, acc:84.648, learning rate:2.6152237569239153e-06
Epoch21, acc:84.64, learning rate:2.6152237569239153e-06
Epoch31, acc:84.652, learning rate:2.6152237569239153e-06
Epoch41, acc:84.65, learning rate:2.6152237569239153e-06
```

[571]:
```
pred_percept = percept_fashion.predict(X_train_fashion)
print('The training accuracy is given by: %f' % (get_acc(pred_percept,␣
 ↪y_train_fashion)))
```

```
The training accuracy is given by: 84.642000
```

5.1.1 Validate Perceptron on Fashion-MNIST

[572]:
```
pred_percept = percept_fashion.predict(X_val_fashion)
print('The validation accuracy is given by: %f' % (get_acc(pred_percept,␣
 ↪y_val_fashion)))
```

```
The validation accuracy is given by: 83.170000
```

5.1.2 Test Perceptron on Fashion-MNIST

[573]:
```
pred_percept = percept_fashion.predict(X_test_fashion)
print('The testing accuracy is given by: %f' % (get_acc(pred_percept,␣
 ↪y_test_fashion)))
np.save("preception_fashion", percept_fashion.w)
```

```
The testing accuracy is given by: 82.360000
```

5.1.3 Perceptron_Fashion-MNIST Kaggle Submission

Once you are satisfied with your solution and test accuracy, output a file to submit your test set predictions to the Kaggle for Assignment 1 Fashion-MNIST. Use the following code to do so:

[574]:
```
output_submission_csv('kaggle/perceptron_submission_fashion.csv',␣
 ↪percept_fashion.predict(X_test_fashion))
```

5.2 Train Perceptron on Rice

[552]:
```
lr = 0.01
n_epochs = 500

percept_RICE = Perceptron(n_class_RICE, lr, n_epochs)
percept_RICE.train(X_train_RICE, y_train_RICE)
```

```
Epoch1, acc:70.8276051690954
Epoch11, acc:72.41316103015306
Epoch21, acc:60.700210796443955
Epoch31, acc:59.28879112821923
Epoch41, acc:67.77563926312895
Epoch51, acc:49.005590688296216
Epoch61, acc:45.46787645495372
Epoch71, acc:99.82586380716708
Epoch81, acc:99.8166987443864
Epoch91, acc:99.83502886994776
Epoch101, acc:99.84419393272844
Epoch111, acc:99.83502886994776
Epoch121, acc:99.87168912107049
Epoch131, acc:99.21180460086151
Epoch141, acc:99.8166987443864
Epoch151, acc:99.86252405828981
Epoch161, acc:99.83502886994776
Epoch171, acc:99.89001924663185
Epoch181, acc:99.83502886994776
Epoch191, acc:99.8166987443864
Epoch201, acc:99.85335899550913
Epoch211, acc:99.85335899550913
Epoch221, acc:99.83502886994776
```

```
Epoch231, acc:99.83502886994776
Epoch241, acc:99.83502886994776
Epoch251, acc:99.83502886994776
Epoch261, acc:99.82586380716708
Epoch271, acc:99.92667949775455
Epoch281, acc:99.73421317936028
Epoch291, acc:99.86252405828981
Epoch301, acc:99.89918430941252
Epoch311, acc:99.82586380716708
Epoch321, acc:99.82586380716708
Epoch331, acc:99.86252405828981
Epoch341, acc:99.86252405828981
Epoch351, acc:99.85335899550913
Epoch361, acc:99.85335899550913
Epoch371, acc:99.85335899550913
Epoch381, acc:99.84419393272844
Epoch391, acc:99.89918430941252
Epoch401, acc:99.93584456053523
Epoch411, acc:99.85335899550913
Epoch421, acc:99.84419393272844
Epoch431, acc:99.85335899550913
Epoch441, acc:99.82586380716708
Epoch451, acc:99.93584456053523
Epoch461, acc:99.93584456053523
Epoch471, acc:99.93584456053523
Epoch481, acc:99.93584456053523
Epoch491, acc:99.93584456053523
```

[553]:
```
pred_percept = percept_RICE.predict(X_train_RICE)
print('The training accuracy is given by: %f' % (get_acc(pred_percept,␣
 ↪y_train_RICE)))
```

```
The training accuracy is given by: 99.908349
```

5.2.1 Validate Perceptron on Rice

[554]:
```
pred_percept = percept_RICE.predict(X_val_RICE)
print('The validation accuracy is given by: %f' % (get_acc(pred_percept,␣
 ↪y_val_RICE)))
```

```
The validation accuracy is given by: 99.807534
```

5.2.2 Test Perceptron on Rice

[555]:
```
pred_percept = percept_RICE.predict(X_test_RICE)
print('The testing accuracy is given by: %f' % (get_acc(pred_percept,␣
 ↪y_test_RICE)))
```

```
The testing accuracy is given by: 99.780038
```

6 Support Vector Machines (with SGD)

Next, you will implement a "soft margin" SVM. In this formulation you will maximize the margin between positive and negative training examples and penalize margin violations using a hinge loss.

We will optimize the SVM loss using SGD. This means you must compute the loss function with respect to model weights. You will use this gradient to update the model weights.

SVM optimized with SGD has 3 hyperparameters that you can experiment with: - **Learning rate** - similar to as defined above in Perceptron, this parameter scales by how much the weights are changed according to the calculated gradient update. - **Epochs** - similar to as defined above in Perceptron. - **Regularization constant** - Hyperparameter to determine the strength of regularization. In this case it is a coefficient on the term which maximizes the margin. You could try different values. The default value is set to 0.05.

You will implement the SVM using SGD in the **models/svm.py**

The following code: - Creates an instance of the SVM classifier class - The train function of the SVM class is trained on the training data - We use the predict function to find the training accuracy as well as the testing accuracy

6.1 Train SVM on Fashion-MNIST

[514]:
```
lr = 0.1
n_epochs = 30
reg_const = 0.001#0.05

svm_fashion = SVM(n_class_fashion, lr, n_epochs, reg_const)
svm_fashion.train(X_train_fashion, y_train_fashion)
```

```
Epoch1, acc:81.072
Epoch11, acc:84.786
Epoch21, acc:84.788
```

[515]:
```
pred_svm = svm_fashion.predict(X_train_fashion)
print('The training accuracy is given by: %f' % (get_acc(pred_svm,␣
 ↪y_train_fashion)))
```

```
The training accuracy is given by: 84.794000
```

6.1.1 Validate SVM on Fashion-MNIST

[516]:
```
pred_svm = svm_fashion.predict(X_val_fashion)
print('The validation accuracy is given by: %f' % (get_acc(pred_svm,␣
 ↪y_val_fashion)))
```

```
The validation accuracy is given by: 82.940000
```

6.1.2 Test SVM on Fashion-MNIST

[517]:
```
pred_svm = svm_fashion.predict(X_test_fashion)
print('The testing accuracy is given by: %f' % (get_acc(pred_svm,␣
 ↪y_test_fashion)))

np.save("svm_fashion", svm_fashion.w)
```

```
The testing accuracy is given by: 82.070000
```

6.1.3 SVM_Fashion-MNIST Kaggle Submission

Once you are satisfied with your solution and test accuracy output a file to submit your test set predictions to the Kaggle for Assignment 1 Fashion-MNIST. Use the following code to do so:

[518]:
```
output_submission_csv('kaggle/svm_submission_fashion.csv', svm_fashion.
 ↪predict(X_test_fashion))
```

6.2 Train SVM on Rice

[519]:
```
lr = 0.001
n_epochs = 50
reg_const = 0.05

svm_RICE = SVM(n_class_RICE, lr, n_epochs, reg_const)
svm_RICE.train(X_train_RICE, y_train_RICE)
```

```
Epoch1, acc:64.73283841994318
Epoch11, acc:90.81660709375859
Epoch21, acc:99.71588305379892
Epoch31, acc:99.87168912107049
Epoch41, acc:99.89001924663185
```

[520]:
```
pred_svm = svm_RICE.predict(X_train_RICE)
print('The training accuracy is given by: %f' % (get_acc(pred_svm,␣
 ↪y_train_RICE)))
```

```
The training accuracy is given by: 99.890019
```

6.2.1 Validate SVM on Rice

[521]:
```
pred_svm = svm_RICE.predict(X_val_RICE)
print('The validation accuracy is given by: %f' % (get_acc(pred_svm,␣
 ↪y_val_RICE)))
```

```
The validation accuracy is given by: 99.890019
```

6.3 Test SVM on Rice

[522]:
```
pred_svm = svm_RICE.predict(X_test_RICE)
print('The testing accuracy is given by: %f' % (get_acc(pred_svm, y_test_RICE)))
```

```
The testing accuracy is given by: 99.835029
```

7 Softmax Classifier (with SGD)

Next, you will train a Softmax classifier. This classifier consists of a linear function of the input data followed by a softmax function which outputs a vector of dimension C (number of classes) for each data point. Each entry of the softmax output vector corresponds to a confidence in one of the C classes, and like a probability distribution, the entries of the output vector sum to 1. We use a cross-entropy loss on this sotmax output to train the model.

Check the following link as an additional resource on softmax classification: http://cs231n.github.io/linear-classify/#softmax

Once again we will train the classifier with SGD. This means you need to compute the gradients of the softmax cross-entropy loss function according to the weights and update the weights using this gradient. Check the following link to help with implementing the gradient updates: https://deepnotes.io/softmax-crossentropy

The softmax classifier has 3 hyperparameters that you can experiment with: - **Learning rate** - As above, this controls how much the model weights are updated with respect to their gradient. - **Number of Epochs** - As described for perceptron. - **Regularization constant** - Hyperparameter to determine the strength of regularization. In this case, we minimize the L2 norm of the model weights as regularization, so the regularization constant is a coefficient on the L2 norm in the combined cross-entropy and regularization objective.

You will implement a softmax classifier using SGD in the **models/softmax.py**

The following code: - Creates an instance of the Softmax classifier class - The train function of the Softmax class is trained on the training data - We use the predict function to find the training accuracy as well as the testing accuracy

7.1 Train Softmax on Fashion-MNIST

[415]:
```
lr = 0.01
n_epochs = 50
reg_const = 0 #0.01

softmax_fashion = Softmax(n_class_fashion, lr, n_epochs, reg_const)
softmax_fashion.train(X_train_fashion, y_train_fashion)
```

```
Epoch1  , acc:85.1320, loss:13.6801
Epoch11 , acc:85.1260, loss:13.6756
Epoch21 , acc:85.1260, loss:13.6756
Epoch31 , acc:85.1260, loss:13.6757
Epoch41 , acc:85.1260, loss:13.6758
```

[416]:
```
pred_softmax = softmax_fashion.predict(X_train_fashion)
print('The training accuracy is given by: %f' % (get_acc(pred_softmax,␣
 ↪y_train_fashion)))
```

```
The training accuracy is given by: 85.126000
```

7.1.1 Validate Softmax on Fashion-MNIST

[417]:
```
pred_softmax = softmax_fashion.predict(X_val_fashion)
print('The validation accuracy is given by: %f' % (get_acc(pred_softmax,␣
 ↪y_val_fashion)))
```

```
The validation accuracy is given by: 84.210000
```

7.1.2 Testing Softmax on Fashion-MNIST

[418]:
```
pred_softmax = softmax_fashion.predict(X_test_fashion)
print('The testing accuracy is given by: %f' % (get_acc(pred_softmax,␣
 ↪y_test_fashion)))
np.save("softmax_weights", softmax_fashion.w)
```

```
The testing accuracy is given by: 82.970000
```

7.1.3 Softmax_Fashion-MNIST Kaggle Submission

Once you are satisfied with your solution and test accuracy output a file to submit your test set predictions to the Kaggle for Assignment 1 Fashion-MNIST. Use the following code to do so:

[419]:
```
output_submission_csv('kaggle/softmax_submission_fashion.csv', softmax_fashion.
 ↪predict(X_test_fashion))
```

7.2 Train Softmax on Rice

[481]:
```
lr = 0.001
n_epochs = 500
reg_const = 0.000#0.05

softmax_RICE = Softmax(n_class_RICE, lr, n_epochs, reg_const)
softmax_RICE.train(X_train_RICE, y_train_RICE)
```

```
Epoch1  , acc:99.9267, loss:0.0668
Epoch11 , acc:99.8259, loss:0.1567
Epoch21 , acc:99.8350, loss:0.1518
Epoch31 , acc:99.8350, loss:0.1519
Epoch41 , acc:99.8259, loss:0.1404
Epoch51 , acc:99.8442, loss:0.1435
Epoch61 , acc:99.8442, loss:0.1396
Epoch71 , acc:99.8625, loss:0.1190
```

```
Epoch81 , acc:99.8809, loss:0.1097
Epoch91 , acc:99.8809, loss:0.1097
Epoch101        , acc:99.8809, loss:0.1097
Epoch111        , acc:99.8809, loss:0.1039
Epoch121        , acc:99.8900, loss:0.1013
Epoch131        , acc:99.8900, loss:0.1013
Epoch141        , acc:99.8900, loss:0.1013
Epoch151        , acc:99.8900, loss:0.1013
Epoch161        , acc:99.8900, loss:0.1013
Epoch171        , acc:99.8900, loss:0.1013
Epoch181        , acc:99.8900, loss:0.0947
Epoch191        , acc:99.8900, loss:0.0955
Epoch201        , acc:99.8900, loss:0.0949
Epoch211        , acc:99.8900, loss:0.0964
Epoch221        , acc:99.8900, loss:0.0953
Epoch231        , acc:99.8900, loss:0.0952
Epoch241        , acc:99.8900, loss:0.0948
Epoch251        , acc:99.9083, loss:0.0724
Epoch261        , acc:99.9083, loss:0.0694
Epoch271        , acc:99.9175, loss:0.0709
Epoch281        , acc:99.9175, loss:0.0730
Epoch291        , acc:99.9175, loss:0.0754
Epoch301        , acc:99.9175, loss:0.0760
Epoch311        , acc:99.9175, loss:0.0760
Epoch321        , acc:99.9175, loss:0.0760
Epoch331        , acc:99.9175, loss:0.0760
Epoch341        , acc:99.9175, loss:0.0742
Epoch351        , acc:99.9175, loss:0.0679
Epoch361        , acc:99.9175, loss:0.0683
Epoch371        , acc:99.9267, loss:0.0675
Epoch381        , acc:99.9267, loss:0.0675
Epoch391        , acc:99.9267, loss:0.0675
Epoch401        , acc:99.9267, loss:0.0675
Epoch411        , acc:99.9267, loss:0.0675
Epoch421        , acc:99.9267, loss:0.0675
Epoch431        , acc:99.9267, loss:0.0675
Epoch441        , acc:99.9267, loss:0.0675
Epoch451        , acc:99.9267, loss:0.0675
Epoch461        , acc:99.9267, loss:0.0675
Epoch471        , acc:99.9267, loss:0.0675
Epoch481        , acc:99.9267, loss:0.0675
Epoch491        , acc:99.9267, loss:0.0675
```

[482]:
```
pred_softmax = softmax_RICE.predict(X_train_RICE)
print('The training accuracy is given by: %f' % (get_acc(pred_softmax,␣
 ↪y_train_RICE)))
```

```
The training accuracy is given by: 99.926679
```

7.2.1 Validate Softmax on Rice

[483]:
```
pred_softmax = softmax_RICE.predict(X_val_RICE)
print('The validation accuracy is given by: %f' % (get_acc(pred_softmax,␣
 ↪y_val_RICE)))
```

```
The validation accuracy is given by: 99.862524
```

7.2.2 Testing Softmax on Rice

[484]:
```
pred_softmax = softmax_RICE.predict(X_test_RICE)
print('The testing accuracy is given by: %f' % (get_acc(pred_softmax,␣
 ↪y_test_RICE)))
np.save("softmax_rice", softmax_RICE.w)
```

```
The testing accuracy is given by: 99.890019
```

8 Logistic Classifier

The Logistic Classifier has 2 hyperparameters that you can experiment with: - **Learning rate** - similar to as defined above in Perceptron, this parameter scales by how much the weights are changed according to the calculated gradient update. - **Number of Epochs** - As described for perceptron. - **Threshold** - The decision boundary of the classifier.

You will implement the Logistic Classifier in the **models/logistic.py**

The following code: - Creates an instance of the Logistic classifier class - The train function of the Logistic class is trained on the training data - We use the predict function to find the training accuracy as well as the testing accuracy

8.0.1 Training Logistic Classifer

[176]:
```
learning_rate = 0.1
n_epochs = 300
threshold = 0.5

lr = Logistic(learning_rate, n_epochs, threshold)
lr.train(X_train_RICE, y_train_RICE)
```

```
/home/kaiwenjon/Documents/Spring2023/Deep-Learning-for-
CV/spring2023/MP1/assignment1/assignment1/models/logistic.py:32: RuntimeWarning:
overflow encountered in exp
  return 1 / (1 + np.exp(-z))
```

```
Epoch1, acc:77.00485748327375
Epoch11, acc:54.30299697552928
Epoch21, acc:82.49473008890111
Epoch31, acc:81.56905874805243
Epoch41, acc:61.5067363211438
```

```
Epoch51, acc:52.27751810099899
Epoch61, acc:98.23114288332874
Epoch71, acc:99.87168912107049
Epoch81, acc:99.8166987443864
Epoch91, acc:99.82586380716708
Epoch101, acc:98.42360920172302
Epoch111, acc:99.47759142150123
Epoch121, acc:99.9083493721932
Epoch131, acc:98.51525982952984
Epoch141, acc:99.87168912107049
Epoch151, acc:99.86252405828981
Epoch161, acc:99.87168912107049
Epoch171, acc:99.89001924663185
Epoch181, acc:99.91751443497388
Epoch191, acc:99.87168912107049
Epoch201, acc:99.92667949775455
Epoch211, acc:99.92667949775455
Epoch221, acc:99.76170836770231
Epoch231, acc:99.74337824214096
Epoch241, acc:98.52442489231052
Epoch251, acc:99.92667949775455
Epoch261, acc:99.69755292823756
Epoch271, acc:99.92667949775455
Epoch281, acc:99.23013472642288
Epoch291, acc:99.89001924663185
```

[157]:
```
pred_lr = lr.predict(X_train_RICE)
print('The training accuracy is given by: %f' % (get_acc(pred_lr,␣
  ↪y_train_RICE)))
```

```
The training accuracy is given by: 99.908349
```

8.0.2 Validate Logistic Classifer

[158]:
```
pred_lr = lr.predict(X_val_RICE)
print('The validation accuracy is given by: %f' % (get_acc(pred_lr,␣
  ↪y_val_RICE)))
```

```
The validation accuracy is given by: 99.890019
```

8.0.3 Test Logistic Classifier

[159]:
```
pred_lr = lr.predict(X_test_RICE)
print('The testing accuracy is given by: %f' % (get_acc(pred_lr, y_test_RICE)))
```

```
The testing accuracy is given by: 99.945010
```

[575]:
```
2460+1489+950
```

[575]: 4899

[ ]: