CS444 Assignment 2

```
import matplotlib.pyplot as plt
import numpy as np

from kaggle_submission import output_submission_csv
from models.neural_net import NeuralNetwork
from utils.data_process import get_FASHION_data
from sklearn.utils import shuffle

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0)  # set default size of plots

# 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
```

Loading Fashion-MNIST

Now that you have implemented a neural network that passes gradient checks and works on toy data, you will test your network on the Fashion-MNIST dataset.

```
# You can change these numbers for experimentation
# For submission be sure they are set to the default values
TRAIN_IMAGES = 50000
VAL_IMAGES = 10000
TEST_IMAGES = 10000

data = get_FASHION_data(TRAIN_IMAGES, VAL_IMAGES, TEST_IMAGES)
X_train, y_train = data['X_train'], data['y_train']
X_val, y_val = data['X_val'], data['y_val']
X_test, y_test = data['X_test'], data['y_test']
```

Train using SGD

To train our network we will use SGD. In addition, we will adjust the learning rate with an exponential learning rate schedule as optimization proceeds; after each epoch, we will reduce the learning rate by multiplying it by a decay rate.

You can try different numbers of layers and other hyperparameters on the Fashion-MNIST dataset below.

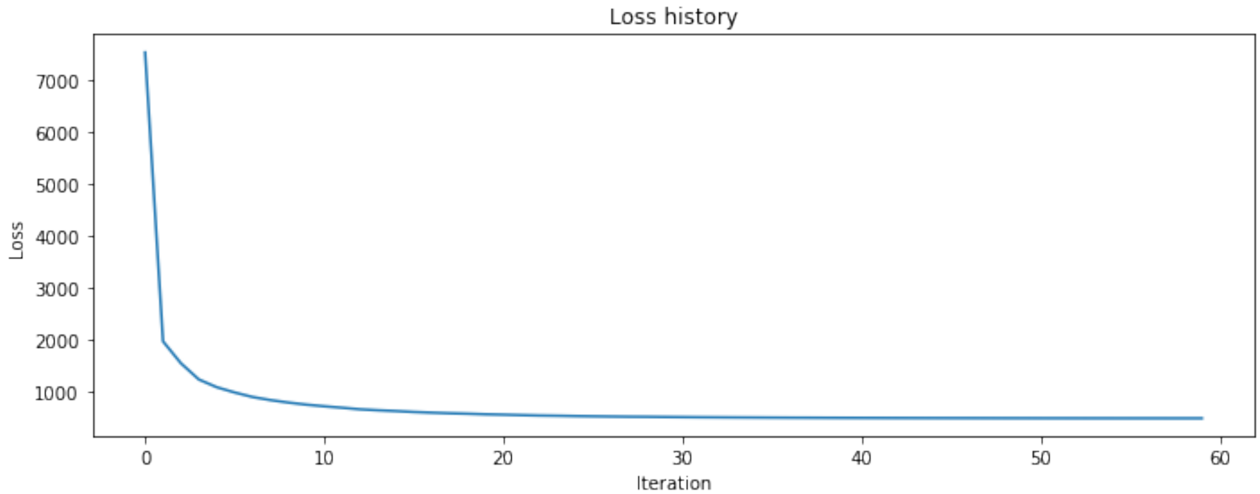
```
# Hyperparameters
input_size = 28 * 28
num_layers = 2
hidden_size = 40
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 60
batch_size = 20
learning_rate = 1.4e-3
learning_rate_decay = 0.9
regularization = 0.0001

# Initialize a new neural network model
net_2 = NeuralNetwork(input_size, hidden_sizes, num_classes, num_layers)

# Variables to store performance for each epoch
train_loss = np.zeros(epochs)
train_accuracy = np.zeros(epochs)
val_accuracy = np.zeros(epochs)

# For each epoch...
for epoch in range(epochs):
    print('epoch:', epoch)
```

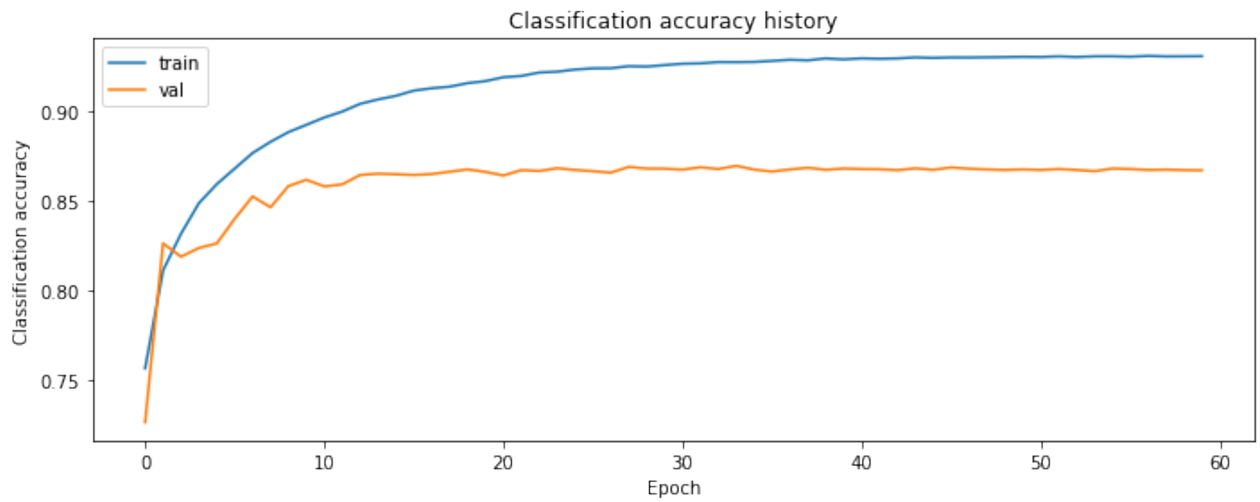
```
    # Shuffle the dataset
    X_train, y_train = shuffle(X_train, y_train)
    # Training
    # For each mini-batch...
    for batch in range(TRAIN_IMAGES // batch_size):
        # Create a mini-batch of training data and labels
        X_batch = X_train[batch*batch_size:(batch+1)*batch_size]
        y_batch = y_train[batch*batch_size:(batch+1)*batch_size]

        # Run the forward pass of the model to get a prediction and compute
        scores = net_2.forward(X_batch)
        pred = np.argmax(scores, axis=1)
        train_accuracy[epoch] += (pred == y_batch).sum()

        # Run the backward pass of the model to compute the loss, and updat
        train_loss[epoch] += net_2.backward(y_batch, regularization)
        net_2.update(learning_rate)

    # Validation
    # No need to run the backward pass here, just run the forward pass to c
    val_scores = net_2.forward(X_val)
    val_pred = np.argmax(val_scores, axis=1)
    val_accuracy[epoch] += (val_pred == y_val).sum()

    # Implement learning rate decay
    learning_rate *= learning_rate_decay


train_accuracy /= X_train.shape[0]
val_accuracy /= X_val.shape[0]
print(train_accuracy[epochs - 1])
print(val_accuracy[epochs - 1])

# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_loss)
plt.title('Loss history')
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(train_accuracy, label='train')
plt.plot(val_accuracy, label='val')
plt.title('Classification accuracy history')
```

```
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

```
epoch: 0
epoch: 1
epoch: 2
epoch: 3
epoch: 4
epoch: 5
epoch: 6
epoch: 7
epoch: 8
epoch: 9
epoch: 10
epoch: 11
epoch: 12
epoch: 13
epoch: 14
epoch: 15
epoch: 16
epoch: 17
epoch: 18
epoch: 19
epoch: 20
epoch: 21
epoch: 22
epoch: 23
epoch: 24
epoch: 25
epoch: 26
epoch: 27
epoch: 28
epoch: 29
epoch: 30
epoch: 31
epoch: 32
epoch: 33
epoch: 34
epoch: 35
epoch: 36
```

```
epoch: 37
epoch: 38
epoch: 39
epoch: 40
epoch: 41
epoch: 42
epoch: 43
epoch: 44
epoch: 45
epoch: 46
epoch: 47
epoch: 48
epoch: 49
epoch: 50
epoch: 51
epoch: 52
epoch: 53
epoch: 54
epoch: 55
epoch: 56
epoch: 57
epoch: 58
epoch: 59
0.93078
0.867
```





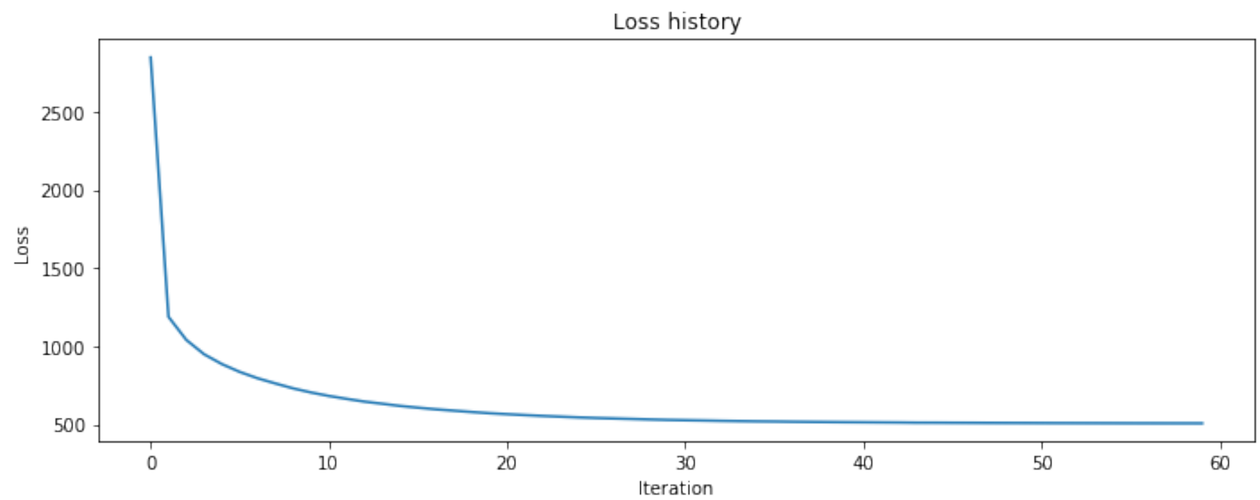

```
# Hyperparameters
input_size = 28 * 28
num_layers = 3
hidden_size = 60
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 60
batch_size = 20
learning_rate = 2e-3
learning_rate_decay = 0.9
regularization = 0.0001

# Initialize a new neural network model
net_3 = NeuralNetwork(input_size, hidden_sizes, num_classes, num_layers)

# Variables to store performance for each epoch
train_loss_S = np.zeros(epochs)
train_accuracy_S = np.zeros(epochs)
```

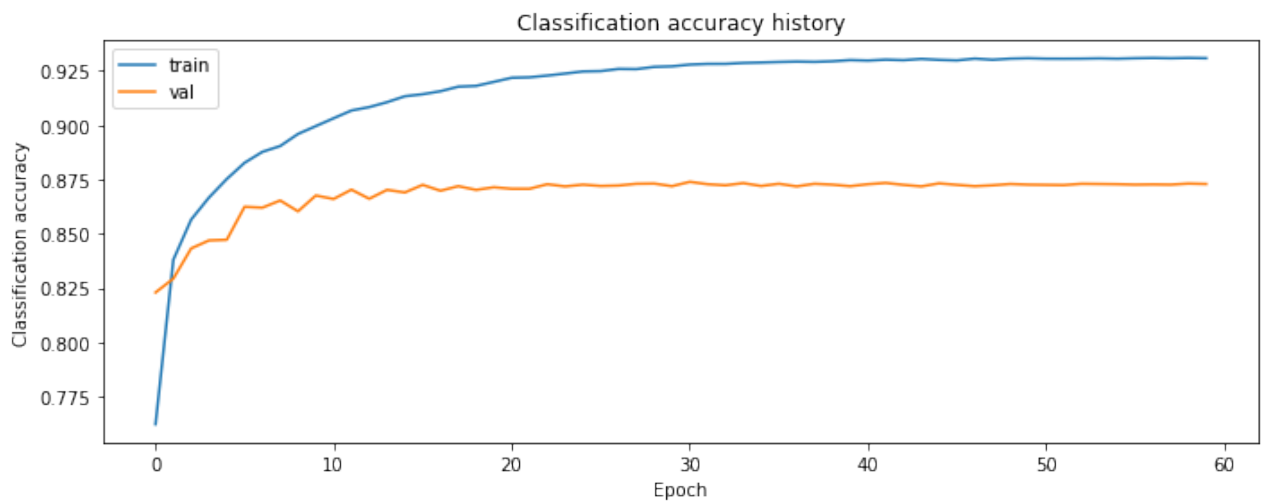
```
val_accuracy_S = np.zeros(epochs)

# For each epoch...
for epoch in range(epochs):
    print('epoch:', epoch)

    # Shuffle the dataset
    X_train, y_train = shuffle(X_train, y_train)


    # Training
    # For each mini-batch...
    for batch in range(TRAIN_IMAGES // batch_size):
        # Create a mini-batch of training data and labels
        X_batch = X_train[batch*batch_size:(batch+1)*batch_size]
        y_batch = y_train[batch*batch_size:(batch+1)*batch_size]

        # Run the forward pass of the model to get a prediction and compute
        scores = net_3.forward(X_batch)
        pred = np.argmax(scores, axis=1)
        train_accuracy_S[epoch] += (pred == y_batch).sum()

        # Run the backward pass of the model to compute the loss, and updat
        train_loss_S[epoch] += net_3.backward(y_batch, regularization)
        net_3.update(learning_rate)

    # Validation
    # No need to run the backward pass here, just run the forward pass to c
    val_scores = net_3.forward(X_val)
    val_pred = np.argmax(val_scores, axis=1)
    val_accuracy_S[epoch] += (val_pred == y_val).sum()

    # Implement learning rate decay
    learning_rate *= learning_rate_decay


train_accuracy_S /= X_train.shape[0]
val_accuracy_S /= X_val.shape[0]
print(train_accuracy_S[epochs - 1])
print(val_accuracy_S[epochs - 1])

# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_loss_S)
plt.title('Loss history')
```

```
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(train_accuracy_S, label='train')
plt.plot(val_accuracy_S, label='val')
plt.title('Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

```
epoch: 0
epoch: 1
epoch: 2
epoch: 3
epoch: 4
epoch: 5
epoch: 6
epoch: 7
epoch: 8
epoch: 9
epoch: 10
epoch: 11
epoch: 12
epoch: 13
epoch: 14
epoch: 15
epoch: 16
epoch: 17
epoch: 18
epoch: 19
epoch: 20
epoch: 21
epoch: 22
epoch: 23
epoch: 24
epoch: 25
epoch: 26
epoch: 27
epoch: 28
epoch: 29
```

```
epoch: 30
epoch: 31
epoch: 32
epoch: 33
epoch: 34
epoch: 35
epoch: 36
epoch: 37
epoch: 38
epoch: 39
epoch: 40
epoch: 41
epoch: 42
epoch: 43
epoch: 44
epoch: 45
epoch: 46
epoch: 47
epoch: 48
epoch: 49
epoch: 50
epoch: 51
epoch: 52
epoch: 53
epoch: 54
epoch: 55
epoch: 56
epoch: 57
epoch: 58
epoch: 59
0.93088
0.8729
```





Train using Adam

Next we will train the same model using the Adam optimizer. You should take the above code for SGD and modify it to use Adam instead. For implementation details, see the lecture slides. The original paper that introduced Adam is also a good reference, and contains suggestions for default values: https://arxiv.org/pdf/1412.6980.pdf

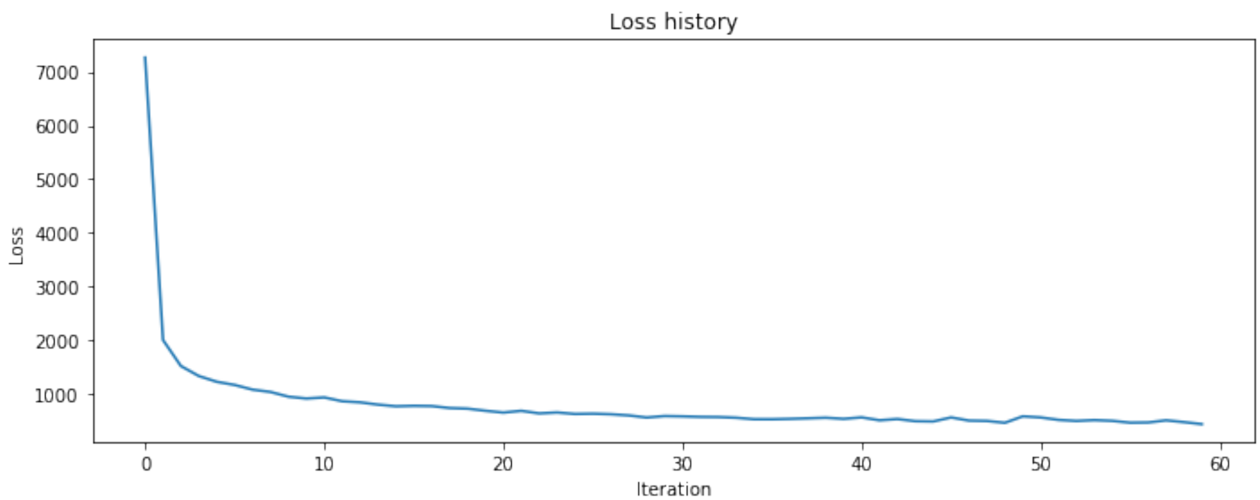
```
# TODO: implement me
# Hyperparameters
input_size = 28 * 28
num_layers = 2
hidden_size = 60
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 60
batch_size = 20
learning_rate = 2e-3
```

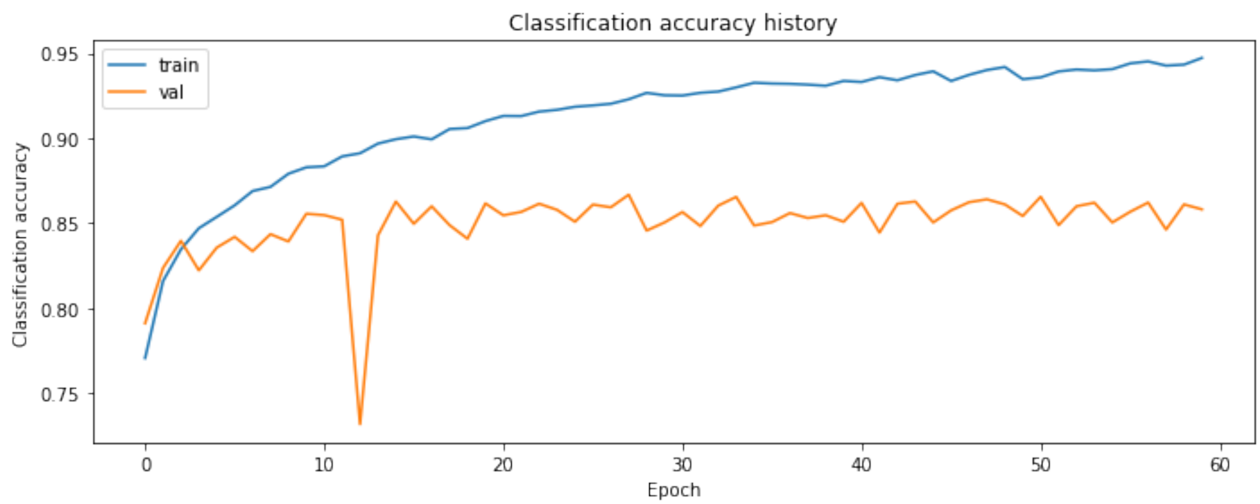
```
learning_rate_decay = 0.9
regularization = 0.0001

# Initialize a new neural network model
net_2_A = NeuralNetwork(input_size, hidden_sizes, num_classes, num_layers
)

# Variables to store performance for each epoch
train_loss_2_A = np.zeros(epochs)
train_accuracy_2_A = np.zeros(epochs)
val_accuracy_2_A = np.zeros(epochs)

# For each epoch...
for epoch in range(epochs):
    print('epoch:', epoch)

    # Shuffle the dataset
    X_train, y_train = shuffle(X_train, y_train)


    # Training
    # For each mini-batch...
    for batch in range(TRAIN_IMAGES // batch_size):
        # Create a mini-batch of training data and labels
        X_batch = X_train[batch*batch_size:(batch+1)*batch_size]
        y_batch = y_train[batch*batch_size:(batch+1)*batch_size]

        # Run the forward pass of the model to get a prediction and compute
        scores = net_2_A.forward(X_batch)
        pred = np.argmax(scores, axis=1)
        train_accuracy_2_A[epoch] += (pred == y_batch).sum()

        # Run the backward pass of the model to compute the loss, and updat
        train_loss_2_A[epoch] += net_2_A.backward(y_batch, regularization)
        net_2_A.update(lr=0.001, b1=0.9, b2=0.999, eps=1e-8, opt="Adam")

    # Validation
    # No need to run the backward pass here, just run the forward pass to c
    val_scores = net_2_A.forward(X_val)
    val_pred = np.argmax(val_scores, axis=1)
    val_accuracy_2_A[epoch] += (val_pred == y_val).sum()

    # Implement learning rate decay
    learning_rate *= learning_rate_decay
```

```
train_accuracy_2_A /= X_train.shape[0]
val_accuracy_2_A /= X_val.shape[0]
print(train_accuracy_2_A[epochs - 1])
print(val_accuracy_2_A[epochs - 1])

# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_loss_2_A)
plt.title('Loss history')
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(train_accuracy_2_A, label='train')
plt.plot(val_accuracy_2_A, label='val')
plt.title('Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

```
epoch: 0
epoch: 1
epoch: 2
epoch: 3
epoch: 4
epoch: 5
epoch: 6
epoch: 7
epoch: 8
epoch: 9
epoch: 10
epoch: 11
epoch: 12
epoch: 13
epoch: 14
epoch: 15
epoch: 16
epoch: 17
epoch: 18
epoch: 19
```

```
epoch: 20
epoch: 21
epoch: 22
epoch: 23
epoch: 24
epoch: 25
epoch: 26
epoch: 27
epoch: 28
epoch: 29
epoch: 30
epoch: 31
epoch: 32
epoch: 33
epoch: 34
epoch: 35
epoch: 36
epoch: 37
epoch: 38
epoch: 39
epoch: 40
epoch: 41
epoch: 42
epoch: 43
epoch: 44
epoch: 45
epoch: 46
epoch: 47
epoch: 48
epoch: 49
epoch: 50
epoch: 51
epoch: 52
epoch: 53
epoch: 54
epoch: 55
epoch: 56
epoch: 57
epoch: 58
epoch: 59
0.94726
0.8582
```





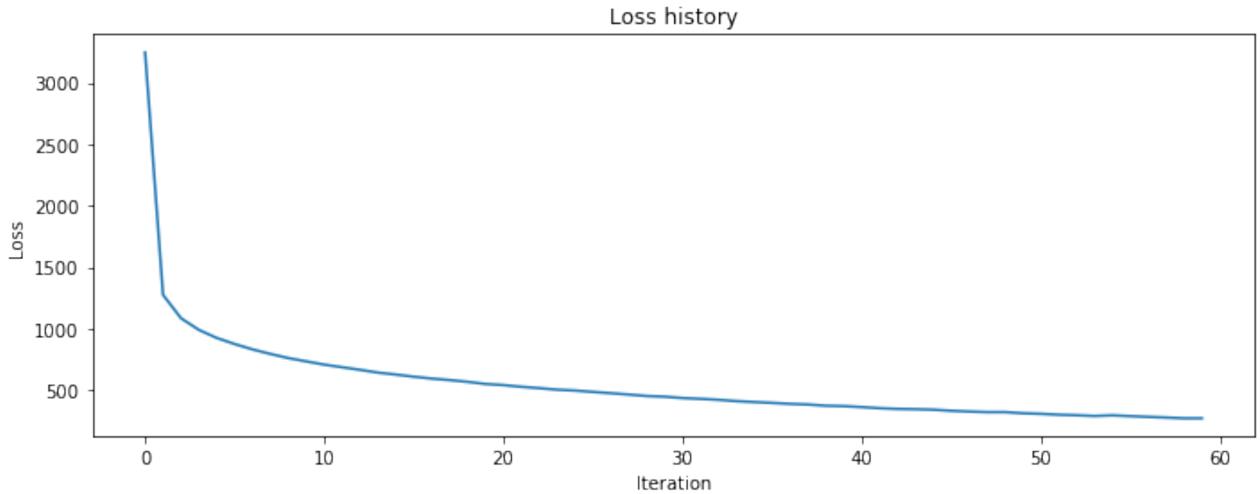

```
# TODO: implement me
# Hyperparameters
input_size = 28 * 28
num_layers = 3
hidden_size = 60
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 60
batch_size = 20
learning_rate = 2e-3
learning_rate_decay = 0.9
regularization = 0.0001

# Initialize a new neural network model
net_3_A = NeuralNetwork(input_size, hidden_sizes, num_classes, num_layers
)

# Variables to store performance for each epoch
```

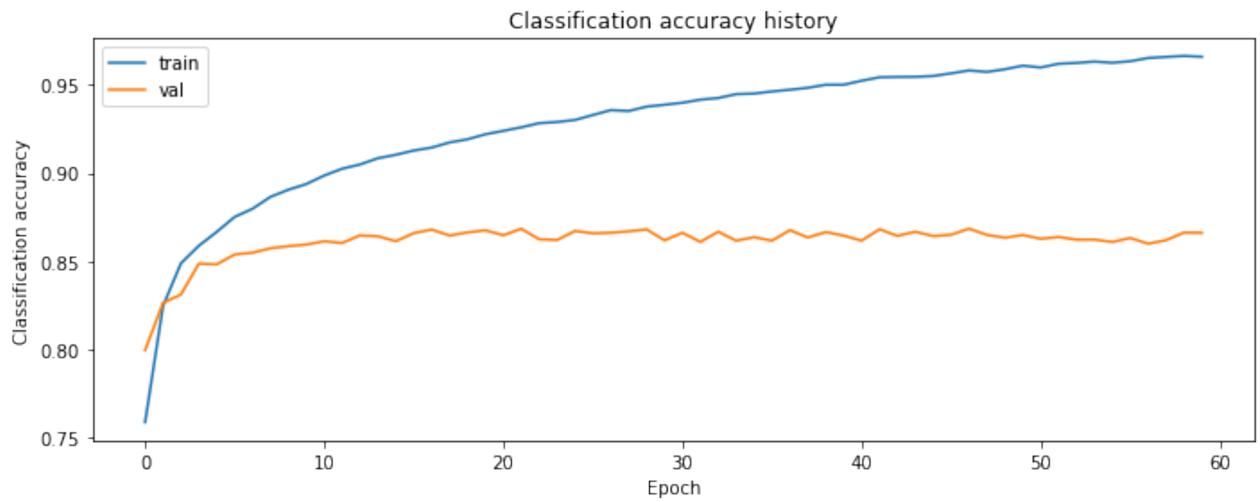
```
train_loss_3_A = np.zeros(epochs)
train_accuracy_3_A = np.zeros(epochs)
val_accuracy_3_A = np.zeros(epochs)

# For each epoch...
for epoch in range(epochs):
    print('epoch:', epoch)

    # Shuffle the dataset
    X_train, y_train = shuffle(X_train, y_train)


    # Training
    # For each mini-batch...
    for batch in range(TRAIN_IMAGES // batch_size):
        # Create a mini-batch of training data and labels
        X_batch = X_train[batch*batch_size:(batch+1)*batch_size]
        y_batch = y_train[batch*batch_size:(batch+1)*batch_size]

        # Run the forward pass of the model to get a prediction and compute
        scores = net_3_A.forward(X_batch)
        pred = np.argmax(scores, axis=1)
        train_accuracy_3_A[epoch] += (pred == y_batch).sum()

        # Run the backward pass of the model to compute the loss, and updat
        train_loss_3_A[epoch] += net_3_A.backward(y_batch, regularization)
        net_3_A.update(lr=0.001, b1=0.9, b2=0.999, eps=1e-8, opt="Adam")

    # Validation
    # No need to run the backward pass here, just run the forward pass to c
    val_scores = net_3_A.forward(X_val)
    val_pred = np.argmax(val_scores, axis=1)
    val_accuracy_3_A[epoch] += (val_pred == y_val).sum()

    # Implement learning rate decay
    learning_rate *= learning_rate_decay


train_accuracy_3_A /= X_train.shape[0]
val_accuracy_3_A /= X_val.shape[0]
print(train_accuracy_3_A[epochs - 1])
print(val_accuracy_3_A[epochs - 1])

# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
```

```
plt.plot(train_loss_3_A)
plt.title('Loss history')
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(train_accuracy_3_A, label='train')
plt.plot(val_accuracy_3_A, label='val')
plt.title('Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

```
epoch: 0
epoch: 1
epoch: 2
epoch: 3
epoch: 4
epoch: 5
epoch: 6
epoch: 7
epoch: 8
epoch: 9
epoch: 10
epoch: 11
epoch: 12
epoch: 13
epoch: 14
epoch: 15
epoch: 16
epoch: 17
epoch: 18
epoch: 19
epoch: 20
epoch: 21
epoch: 22
epoch: 23
epoch: 24
epoch: 25
epoch: 26
epoch: 27
```

```
epoch: 28
epoch: 29
epoch: 30
epoch: 31
epoch: 32
epoch: 33
epoch: 34
epoch: 35
epoch: 36
epoch: 37
epoch: 38
epoch: 39
epoch: 40
epoch: 41
epoch: 42
epoch: 43
epoch: 44
epoch: 45
epoch: 46
epoch: 47
epoch: 48
epoch: 49
epoch: 50
epoch: 51
epoch: 52
epoch: 53
epoch: 54
epoch: 55
epoch: 56
epoch: 57
epoch: 58
epoch: 59
0.96604
0.8663
```

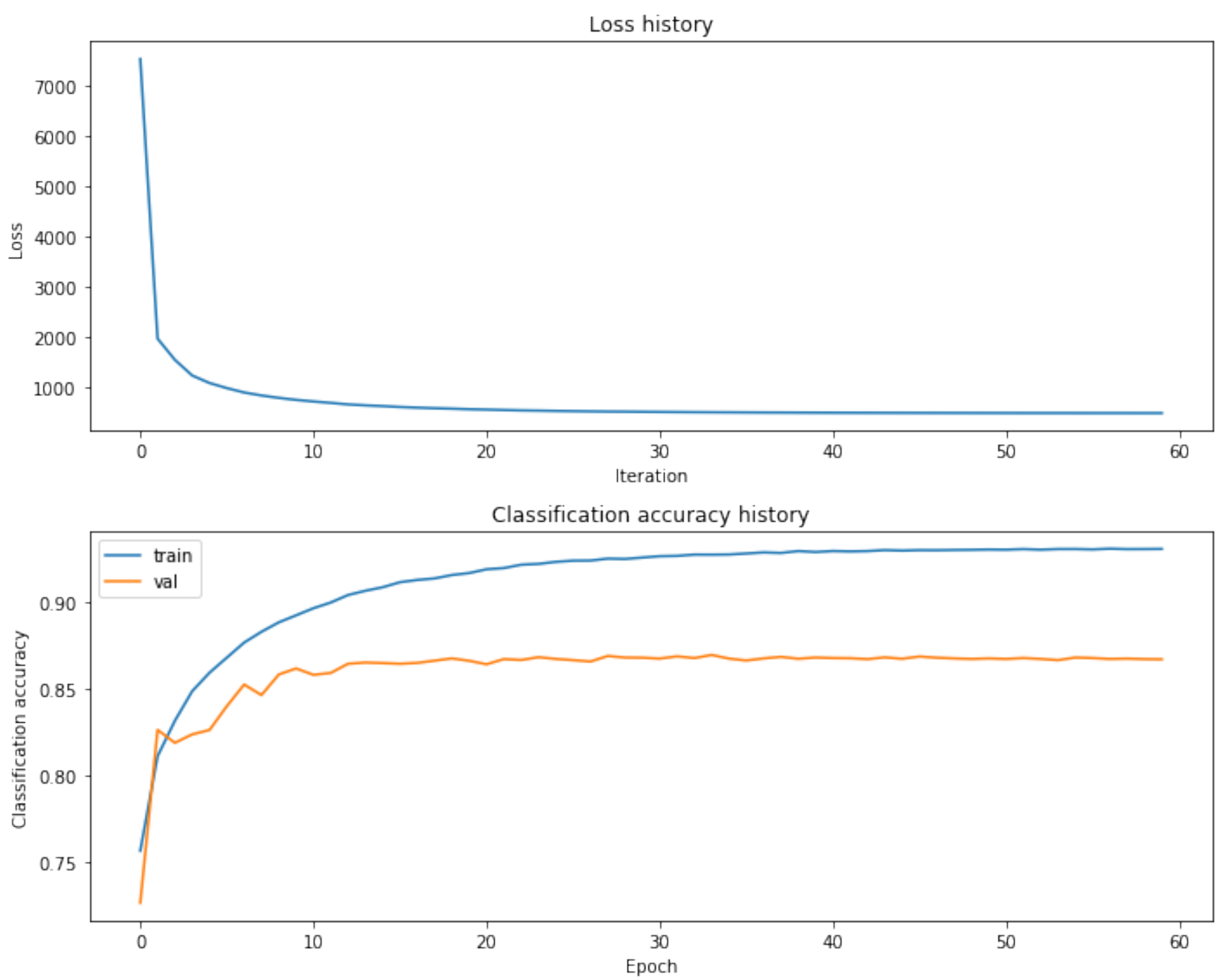




Graph loss and train/val accuracies

Examining the loss graph along with the train and val accuracy graphs should help you gain some intuition for the hyperparameters you should try in the hyperparameter tuning below. It should also help with debugging any issues you might have with your network.

```
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_loss)
plt.title('Loss history')
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(train_accuracy, label='train')
plt.plot(val_accuracy, label='val')
plt.title('Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```



Hyperparameter tuning

Once you have successfully trained a network you can tune your hyparameters to increase your accuracy.

Based on the graphs of the loss function above you should be able to develop some intuition about what hyperparameter adjustments may be necessary. A very noisy loss implies that the learning rate might be too high, while a linearly decreasing loss would suggest that the learning rate may be too low. A large gap between training and validation accuracy would suggest overfitting due to a large model without much regularization. No gap between training and validation accuracy would indicate low model capacity.

You will compare networks of two and three layers using the different optimization methods you implemented.

The different hyperparameters you can experiment with are:
- **Batch size**: We recommend you leave this at 200 initially which is the batch size we used.
- **Number of iterations**: You can gain an intuition for how many iterations to run by checking when the validation accuracy plateaus in your train/val accuracy graph.
- **Initialization** Weight initialization is very important for neural networks. We used the initialization `W = np.random.randn(n) / sqrt(n)` where `n` is the input dimension for layer corresponding to `W`. We recommend you stick with the given initializations, but you may explore modifying these. Typical initialization practices: http://cs231n.github.io/neural-networks-2/#init
- **Learning rate**: Generally from around 1e-4 to 1e-1 is a good range to explore according to our implementation.
- **Learning rate decay**: We recommend a 0.95 decay to start.
- **Hidden layer size**: You should explore up to around 120 units per layer. For three-layer network, we fixed the two hidden layers to be the same size when obtaining the target numbers. However, you may experiment with having different size hidden layers.
- **Regularization coefficient**: We recommend trying values in the range 0 to 0.1.

Hints:
- After getting a sense of the parameters by trying a few values yourself, you will likely want to write a few for-loops to traverse over a set of hyperparameters.
- If you find that your train loss is decreasing, but your train and val accuracy start to decrease rather than increase, your model likely started minimizing the regularization term. To prevent this you will need to decrease the regularization coefficient.

Run on the test set

When you are done experimenting, you should evaluate your final trained networks on the test set.

```
best_2layer_sgd_prediction = np.argmax(net_2.forward(X_test), axis=1)
best_3layer_sgd_prediction = np.argmax(net_3.forward(X_test), axis=1)
best_2layer_adam_prediction = np.argmax(net_2_A.forward(X_test), axis=1)
best_3layer_adam_prediction = np.argmax(net_3_A.forward(X_test), axis=1)
```

Kaggle output

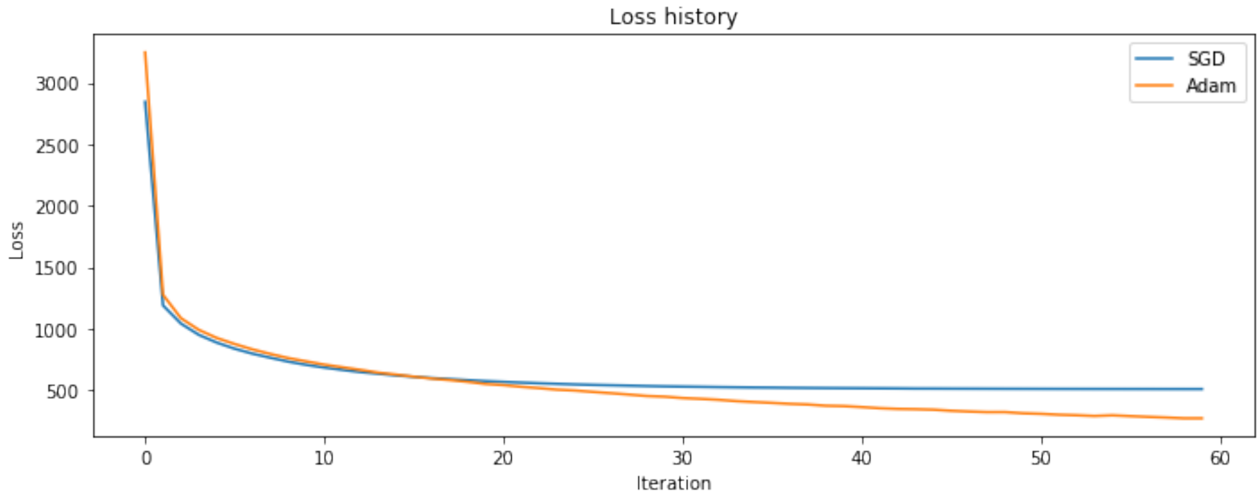
Once you are satisfied with your solution and test accuracy, output a file to submit your test set predictions to the Kaggle for Assignment 2 Neural Network. Use the following code to do so:

```
output_submission_csv('./nn_2layer_sgd_submission.csv', best_2layer_sgd_p
rediction)
output_submission_csv('./nn_3layer_sgd_submission.csv', best_3layer_sgd_p
rediction)
output_submission_csv('./nn_2layer_adam_submission.csv', best_2layer_adam
_prediction)
output_submission_csv('./nn_3layer_adam_submission.csv', best_3layer_adam
_prediction)
```

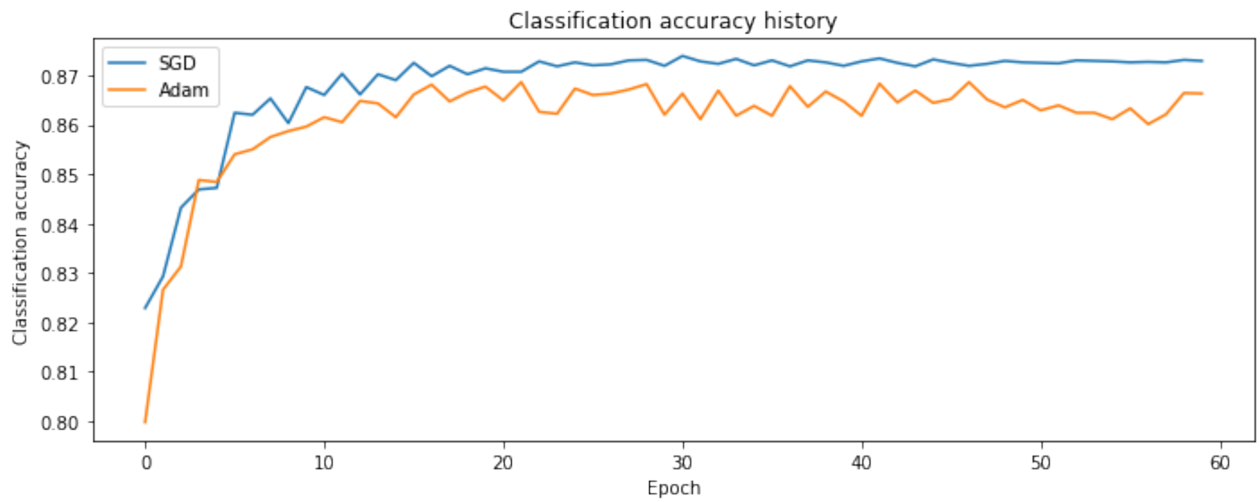
Compare SGD and Adam

Create graphs to compare training loss and validation accuracy between SGD and Adam. The code is similar to the above code, but instead of comparing train and validation, we are comparing SGD and Adam.

```
# TODO: implement me
plt.subplot(2, 1, 1)
plt.plot(train_loss_S, label='SGD')
plt.plot(train_loss_3_A, label='Adam')
plt.title('Loss history')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(val_accuracy_S, label='SGD')
plt.plot(val_accuracy_3_A, label='Adam')
plt.title('Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```



Loss history
SGD
Adam
3000
2500
2000
1500
1000
500
Loss
0
10
20
30
40
50
60
Iteration



Classification accuracy history
SGD
Adam
0.87
0.86
0.85
0.84
0.83
0.82
0.81
0.80
Classification accuracy
0
10
20
30
40
50
60
Epoch