CS498DL Assignment 2

In [37]:
```
# from getpass import getpass
# import os

# os.environ['USER'] = input('Enter the username of your Github account: ')
# os.environ['PASSWORD'] = getpass('Enter the password of your Github account: ')
# # os.environ['REPOSITORY'] = input('Enter the name of the Github repository: ')
# os.environ['GITHUB_AUTH'] = os.environ['USER'] + ':' + os.environ['PASSWORD']

# !rm -rf $REPOSITORY # To remove the previous clone of the Github repository
# !git clone https://$GITHUB_AUTH@github.com/ry323/cs498dl-mp2.git
# !git pull origin ry_dev
# os.environ['PASSWORD'] =  os.environ['GITHUB_AUTH'] = ""
# %cd cs498dl-mp2/cifar10/
# %run get_datasets
# %cd ..
# !ls
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/usr/local/lib/python3.6/dist-packages/ipykernel/kernelbase.py in _input_request(self, prompt,
 ident, parent, password)
    728             try:
--> 729                 ident, reply = self.session.recv(self.stdin_socket, 0)
    730             except Exception:

/usr/local/lib/python3.6/dist-packages/jupyter_client/session.py in recv(self, socket, mode, co
ntent, copy)
    802         try:
--> 803             msg_list = socket.recv_multipart(mode, copy=copy)
    804         except zmq.ZMQError as e:

/usr/local/lib/python3.6/dist-packages/zmq/sugar/socket.py in recv_multipart(self, flags, copy,
track)
    490         """
--> 491         parts = [self.recv(flags, copy=copy, track=track)]
    492         # have first part already, only loop while more to receive

zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.recv()

zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.recv()

zmq/backend/cython/socket.pyx in zmq.backend.cython.socket._recv_copy()

/usr/local/lib/python3.6/dist-packages/zmq/backend/cython/checkrc.pxd in zmq.backend.cython.che
ckrc._check_rc()

KeyboardInterrupt:

During handling of the above exception, another exception occurred:

KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-37-dbd0b7785e36> in <module>()
      2 import os
      3
----> 4 os.environ['USER'] = input('Enter the username of your Github account: ')
      5 os.environ['PASSWORD'] = getpass('Enter the password of your Github account: ')
      6 # os.environ['REPOSITORY'] = input('Enter the name of the Github repository: ')

/usr/local/lib/python3.6/dist-packages/ipykernel/kernelbase.py in raw_input(self, prompt)
    702             self._parent_ident,
    703             self._parent_header,
--> 704             password=False,
    705         )
    706

/usr/local/lib/python3.6/dist-packages/ipykernel/kernelbase.py in _input_request(self, prompt,
 ident, parent, password)
    732             except KeyboardInterrupt:
    733                 # re-raise KeyboardInterrupt, to truncate traceback
--> 734                 raise KeyboardInterrupt
    735             else:
```

```
    736                 break

KeyboardInterrupt:
```

In [38]:
```
import matplotlib.pyplot as plt
import numpy as np

from kaggle_submission import output_submission_csv
from models.neural_net import NeuralNetwork
from models.neural_net_adam import NeuralNetwork_adam
from utils.data_process import get_CIFAR10_data

%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 CIFAR-10

Now that you have implemented a neural network that passes gradient checks and works on toy data, you will test your network on the CIFAR-10 dataset.

In [39]:
```
# You can change these numbers for experimentation
# For submission be sure they are set to the default values
TRAIN_IMAGES = 49000
VAL_IMAGES = 1000
TEST_IMAGES = 10000

data = get_CIFAR10_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 CIFAR-10 dataset below.

2-layer SGD

In [86]:
```
# Hyperparameters
input_size = 32 * 32 * 3
num_layers = 2
hidden_size = 60 #20
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 80 #100
batch_size = 200 #200
learning_rate = 1e-2 #1e-3
learning_rate_decay = 0.95
regularization = 0.01# 0.1

# Initialize a new neural network model
net = 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
    shuffle_index = np.random.permutation(np.shape(X_train)[0])
    X = X_train[shuffle_index]
    Y = y_train[shuffle_index]

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

        # Run the forward pass of the model to get a prediction and compute the accuracy
        output = net.forward(X_batch)
        train_accuracy[epoch] += net.get_accuracy(output, y_batch)


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


    # learning rate decay
    learning_rate *= learning_rate_decay

    train_loss[epoch] /= n_batches
    train_accuracy[epoch] /= n_batches

    # Validation
    # No need to run the backward pass here, just run the forward pass to compute accuracy
    output_val = net.forward(X_val)
    val_accuracy[epoch] = net.get_accuracy(output_val, y_val)
print("2-layer SGD Train accuracy: ", train_accuracy[-1])
print("2-layer SGD Validation accuracy: ", val_accuracy[-1])
output = net.forward(X_test)
print ("2 layer sgd accuracy = ", net.get_accuracy(output, y_test))
best_2layer_sgd_prediction  = np.argmax(output, axis = 1)
```

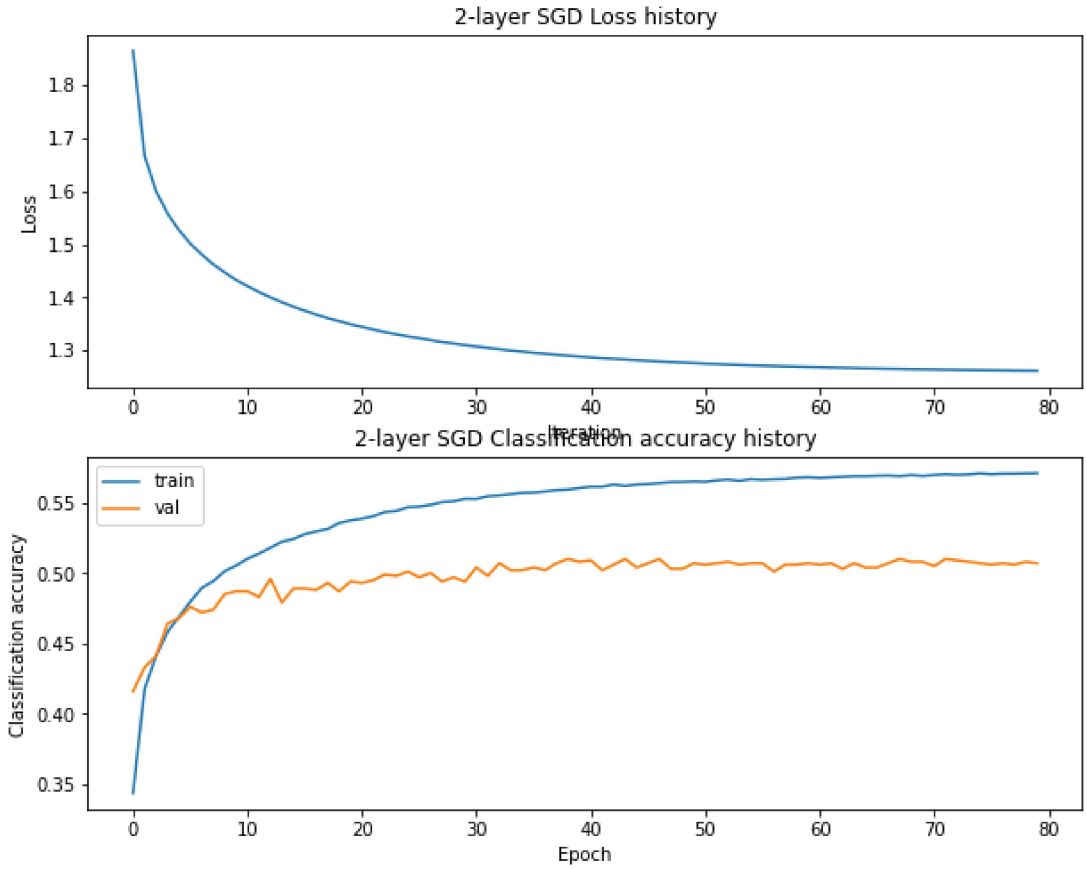
```
epoch: 0
epoch: 1
epoch: 2
epoch: 3
epoch: 4
epoch: 5
epoch: 6
epoch: 7
epoch: 8
epoch: 9
epoch: 10
epoch: 11
epoch: 12
epoch: 13
epoch: 14
epoch: 15
epoch: 16
epoch: 17
epoch: 18
epoch: 19
epoch: 20
epoch: 21
epoch: 22
epoch: 23
epoch: 24
epoch: 25
epoch: 26
epoch: 27
epoch: 28
epoch: 29
epoch: 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
epoch: 60
epoch: 61
epoch: 62
epoch: 63
epoch: 64
epoch: 65
epoch: 66
epoch: 67
epoch: 68
epoch: 69
epoch: 70
epoch: 71
epoch: 72
epoch: 73
epoch: 74
epoch: 75
epoch: 76
epoch: 77
epoch: 78
epoch: 79
2-layer SGD Train accuracy:  0.5531020408163266
2-layer SGD Validation accuracy:  0.5
2 layer sgd accuracy =  0.5037
```

In [82]:

```
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_loss)
plt.title('2-layer SGD 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('2-layer SGD Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()
plt.show()
```



3-layer SGD

In [56]:
```
# Hyperparameters
input_size = 32 * 32 * 3
num_layers = 3
hidden_size = 60 #20
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 80 #100
batch_size = 200
learning_rate = 1e-2 #1e-3
learning_rate_decay = 0.95
regularization = 0.01# 0.1

# Initialize a new neural network model
net3 = NeuralNetwork(input_size, hidden_sizes, num_classes, num_layers)

# Variables to store performance for each epoch
train_loss3 = np.zeros(epochs)
train_accuracy3 = np.zeros(epochs)
val_accuracy3 = np.zeros(epochs)

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

    # Shuffle the dataset
    shuffle_index = np.random.permutation(np.shape(X_train)[0])
    X = X_train[shuffle_index]
    Y = y_train[shuffle_index]

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

        # Run the forward pass of the model to get a prediction and compute the accuracy
```

```
        output = net3.forward(X_batch)
        train_accuracy3[epoch] += net3.get_accuracy(output, y_batch)


        # Run the backward pass of the model to update the weights and compute the loss
        loss= net3.backward(X_batch, y_batch,learning_rate , regularization)
        train_loss3[epoch] += loss


    # learning rate decay
    learning_rate *= learning_rate_decay

    train_loss3[epoch] /= n_batches
    train_accuracy3[epoch] /= n_batches

    # Validation
    # No need to run the backward pass here, just run the forward pass to compute accuracy
    output_val = net3.forward(X_val)
    val_accuracy3[epoch] = net3.get_accuracy(output_val, y_val)

print ("Batch size: ", batch_size)
print("Learning rate: ", learning_rate)
print("Hidden layer size:", hidden_size)
print ("Regularization coefficient: ", regularization)

print("3-layer SGD Train accuracy: ", train_accuracy3[-1])
print("3-layer SGD Validation accuracy: ", val_accuracy3[-1])
output = net3.forward(X_test)
print ("3 layer sgd accuracy = ", net3.get_accuracy(output, y_test))
best_3layer_sgd_prediction  = np.argmax(output, axis = 1)
```

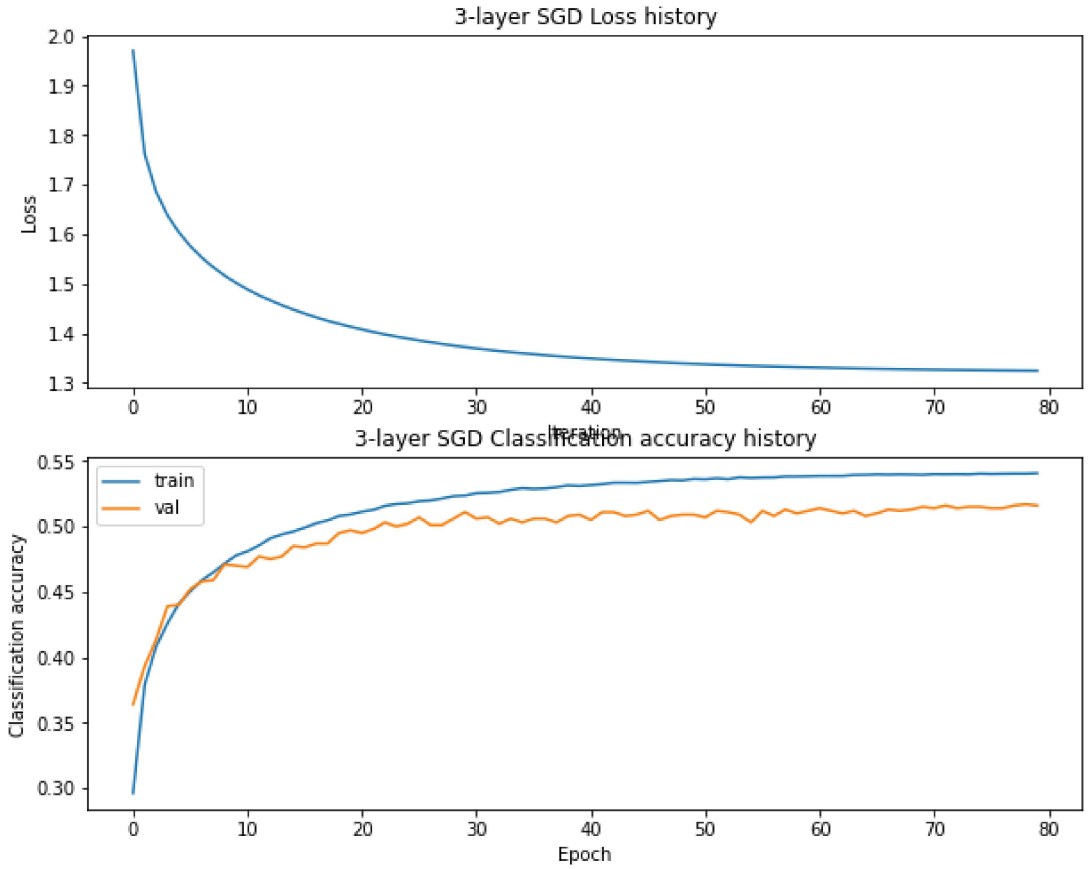
```
epoch: 0
epoch: 1
epoch: 2
epoch: 3
epoch: 4
epoch: 5
epoch: 6
epoch: 7
epoch: 8
epoch: 9
epoch: 10
epoch: 11
epoch: 12
epoch: 13
epoch: 14
epoch: 15
epoch: 16
epoch: 17
epoch: 18
epoch: 19
epoch: 20
epoch: 21
epoch: 22
epoch: 23
epoch: 24
epoch: 25
epoch: 26
epoch: 27
epoch: 28
epoch: 29
epoch: 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
epoch: 60
epoch: 61
epoch: 62
epoch: 63
epoch: 64
epoch: 65
epoch: 66
epoch: 67
epoch: 68
epoch: 69
epoch: 70
epoch: 71
epoch: 72
epoch: 73
epoch: 74
epoch: 75
epoch: 76
epoch: 77
epoch: 78
epoch: 79
Batch size:  200
Learning rate:  0.00016515374385013573
Hidden layer size: 60
Regularization coefficient:  0.01
3-layer SGD Train accuracy:  0.5408163265306122
3-layer SGD Validation accuracy:  0.516
3 layer sgd accuracy =  0.5001
```

In [65]:

```
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_loss3)
plt.title('3-layer SGD Loss history')
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(train_accuracy3, label='train')
plt.plot(val_accuracy3, label='val')
plt.title('3-layer SGD Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()
plt.show()
```



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

2-layer Adam

In [68]:
```
# TODO: implement me
# Hyperparameters
input_size = 32 * 32 * 3
num_layers = 2
hidden_size = 60 #20
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 80 #100
batch_size = 200
learning_rate = 1e-3 #1e-3
learning_rate_decay = 0.95
regularization = 0.01
beta1 = 0.9
beta2 = 0.999

# Initialize a new neural network model
net_adam = NeuralNetwork_adam(input_size, hidden_sizes, num_classes, num_layers)

# Variables to store performance for each epoch
train_loss_adam = np.zeros(epochs)
train_accuracy_adam = np.zeros(epochs)
val_accuracy_adam = np.zeros(epochs)

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

    # Shuffle the dataset
```

```
    # idxs = np.random.choice(TRAIN_IMAGES, batch_size, replace=True)
    shuffle_index = np.random.permutation(np.shape(X_train)[0])
    X = X_train[shuffle_index]
    Y = y_train[shuffle_index]

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

        # Run the forward pass of the model to get a prediction and compute the accuracy
        output = net_adam.forward(X_batch)
        train_accuracy_adam[epoch] += net_adam.get_accuracy(output, y_batch)

        # Run the backward pass of the model to update the weights and compute the loss
        loss = net_adam.backward(X_batch, y_batch,learning_rate , regularization, beta1, beta2)
        train_loss_adam[epoch] += loss

        # Validation
        # No need to run the backward pass here, just run the forward pass to compute accuracy
        output = net_adam.forward(X_batch)
        val_accuracy_adam[epoch] += net_adam.get_accuracy(output, y_batch)

    train_loss_adam[epoch] /= batch_size
    train_accuracy_adam[epoch] /= batch_size
    val_accuracy_adam[epoch] /= batch_size

print ("Batch size: ", batch_size)
print("Learning rate: ", learning_rate)
print("Hidden layer size:", hidden_size)
print ("Regularization coefficient: ", regularization)
print ("beta1 ", beta1)
print("beta2: ", beta2)

print("2-layer Adam Train accuracy: ", train_accuracy_adam[-1])
print("2-layer Adam Validation accuracy: ", val_accuracy_adam[-1])
output = net_adam.forward(X_test)
print ("2 layer Adam accuracy = ", net_adam.get_accuracy(output, y_test))
best_2layer_adam_prediction  = np.argmax(output, axis = 1)
```

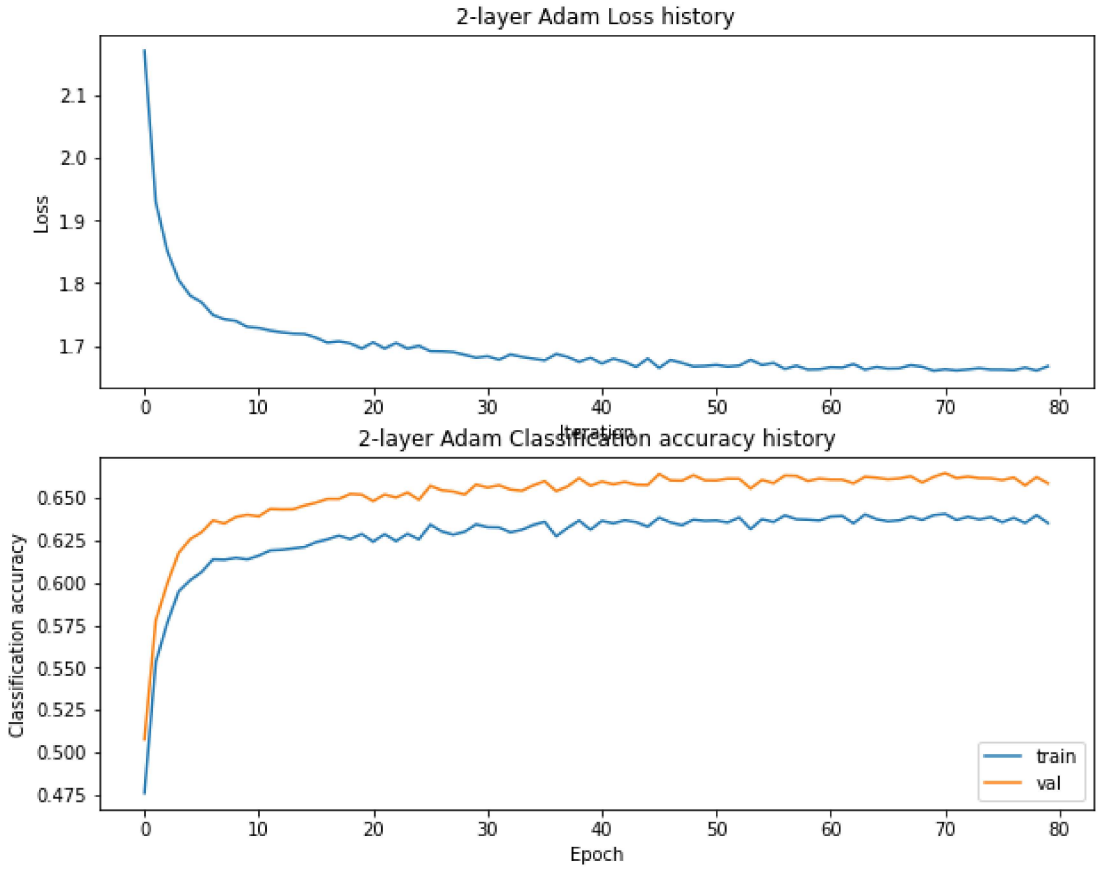
```
epoch: 0
epoch: 1
epoch: 2
epoch: 3
epoch: 4
epoch: 5
epoch: 6
epoch: 7
epoch: 8
epoch: 9
epoch: 10
epoch: 11
epoch: 12
epoch: 13
epoch: 14
epoch: 15
epoch: 16
epoch: 17
epoch: 18
epoch: 19
epoch: 20
epoch: 21
epoch: 22
epoch: 23
epoch: 24
epoch: 25
epoch: 26
epoch: 27
epoch: 28
epoch: 29
epoch: 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
epoch: 60
epoch: 61
epoch: 62
epoch: 63
epoch: 64
epoch: 65
epoch: 66
epoch: 67
epoch: 68
epoch: 69
epoch: 70
epoch: 71
epoch: 72
epoch: 73
epoch: 74
epoch: 75
epoch: 76
epoch: 77
epoch: 78
epoch: 79
Batch size:  200
Learning rate:  0.001
Hidden layer size: 60
Regularization coefficient:  0.01
beta1  0.9
beta2:  0.999
2-layer Adam Train accuracy:  0.6352499999999996
2-layer Adam Validation accuracy:  0.6587500000000004
2 layer Adam accuracy =  0.4965
```

In [69]:

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

plt.subplot(2, 1, 2)
plt.plot(train_accuracy_adam, label='train')
plt.plot(val_accuracy_adam, label='val')
plt.title('2-layer Adam Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()
plt.show()
```



3-layer Adam

In [36]:
```
# TODO: implement me
# Hyperparameters
input_size = 32 * 32 * 3
num_layers = 3
hidden_size = 60
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 80 #100
batch_size = 200
learning_rate = 1e-3 #1e-3
learning_rate_decay = 0.95
regularization = 0.01
beta1 = 0.9
beta2 = 0.999

# Initialize a new neural network model
net_adam3 = NeuralNetwork_adam(input_size, hidden_sizes, num_classes, num_layers)

# Variables to store performance for each epoch
train_loss_adam3 = np.zeros(epochs)
train_accuracy_adam3 = np.zeros(epochs)
val_accuracy_adam3 = np.zeros(epochs)

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

    # Shuffle the dataset
    # idxs = np.random.choice(TRAIN_IMAGES, batch_size, replace=True)
    shuffle_index = np.random.permutation(np.shape(X_train)[0])
    X = X_train[shuffle_index]
    Y = y_train[shuffle_index]

    # Training
    # For each mini-batch...
    # Create a mini-batch of training data and labels
    n_batches = TRAIN_IMAGES // batch_size
    for batch in range(n_batches):
```

```
        # Create a mini-batch of training data and labels
        X_batch = X[batch*batch_size : batch*batch_size + batch_size]
        y_batch = Y[batch*batch_size : batch*batch_size + batch_size]

        # Run the forward pass of the model to get a prediction and compute the accuracy
        output = net_adam3.forward(X_batch)
        train_accuracy_adam3[epoch] += net_adam3.get_accuracy(output, y_batch)

        # Run the backward pass of the model to update the weights and compute the loss
        loss = net_adam3.backward(X_batch, y_batch,learning_rate , regularization, beta1, beta2)
        train_loss_adam3[epoch] += loss

        # Validation
        # No need to run the backward pass here, just run the forward pass to compute accuracy
        output = net_adam3.forward(X_batch)
        val_accuracy_adam3[epoch] += net_adam3.get_accuracy(output, y_batch)


    train_loss_adam3[epoch] /= batch_size
    train_accuracy_adam3[epoch] /= batch_size
    val_accuracy_adam3[epoch] /= batch_size

print ("Batch size: ", batch_size)
print("Learning rate: ", learning_rate)
print("Hidden layer size:", hidden_size)
print ("Regularization coefficient: ", regularization)
print ("beta1 ", beta1)
print("beta2: ", beta2)

print("3-layer Adam Train accuracy: ", train_accuracy_adam3[-1])
print("3-layer Adam Validation accuracy: ", val_accuracy_adam3[-1])
output = net_adam3.forward(X_test)
print ("3 layer Adam accuracy = ", net_adam3.get_accuracy(output, y_test))
best_2layer_adam_prediction  = np.argmax(output, axis = 1)
```

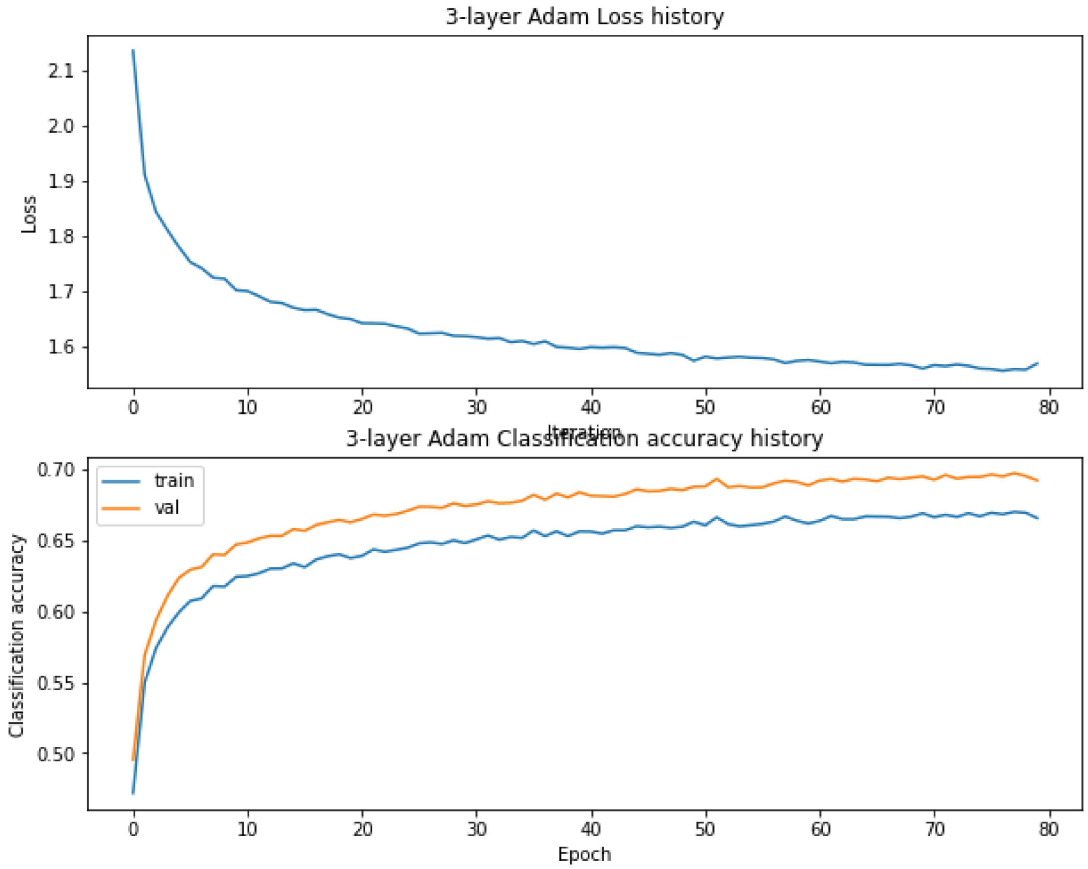
```
epoch: 0
epoch: 1
epoch: 2
epoch: 3
epoch: 4
epoch: 5
epoch: 6
epoch: 7
epoch: 8
epoch: 9
epoch: 10
epoch: 11
epoch: 12
epoch: 13
epoch: 14
epoch: 15
epoch: 16
epoch: 17
epoch: 18
epoch: 19
epoch: 20
epoch: 21
epoch: 22
epoch: 23
epoch: 24
epoch: 25
epoch: 26
epoch: 27
epoch: 28
epoch: 29
epoch: 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
epoch: 60
epoch: 61
epoch: 62
epoch: 63
epoch: 64
epoch: 65
epoch: 66
epoch: 67
epoch: 68
epoch: 69
epoch: 70
epoch: 71
epoch: 72
epoch: 73
epoch: 74
epoch: 75
epoch: 76
epoch: 77
epoch: 78
epoch: 79
Batch size:  200
Learning rate:  0.001
Hidden layer size: 60
Regularization coefficient:  0.01
beta1  0.9
beta2:  0.999
3-layer Adam Train accuracy:  0.665575
3-layer Adam Validation accuracy:  0.6921750000000003
3 layer Adam accuracy =  0.5117
```

In [70]:

```
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_loss_adam3)
plt.title('3-layer Adam Loss history')
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(train_accuracy_adam3, label='train')
plt.plot(val_accuracy_adam3, label='val')
plt.title('3-layer Adam Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()
plt.show()
```



In [ ]:

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 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.

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:

In [75]:
```
# %cd cs498dl-mp2/
# !ls
```

```
[Errno 2] No such file or directory: 'cs498dl-mp2/'
/content/cs498dl-mp2
cifar10                       neural_network.ipynb           __pycache__
develop_neural_network.ipynb  nn_2layer_adam_submission.csv  utils
kaggle_submission.py          nn_2layer_sgd_submission.csv
models                        nn_3layer_sgd_submission.csv
```

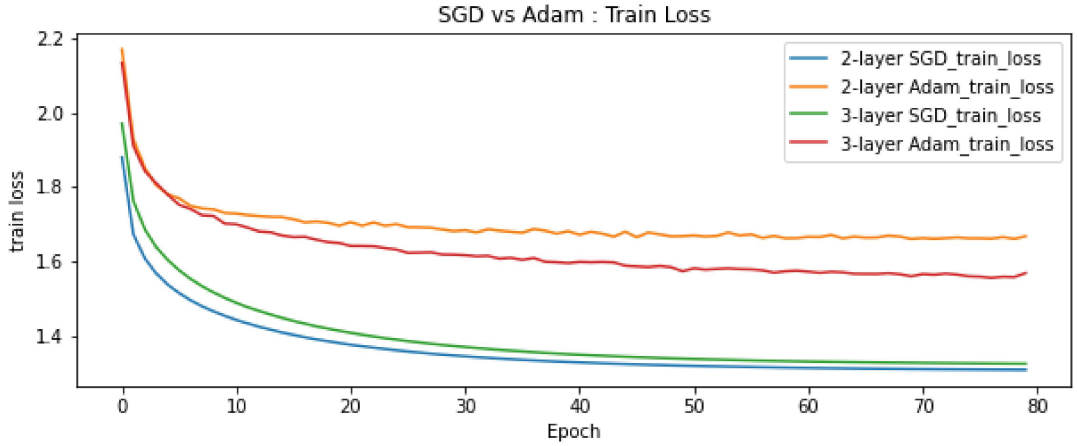
In [76]:
```
output_submission_csv('./nn_2layer_sgd_submission.csv', best_2layer_sgd_prediction)
output_submission_csv('./nn_3layer_sgd_submission.csv', best_3layer_sgd_prediction)
output_submission_csv('./nn_2layer_adam_submission.csv', best_2layer_adam_prediction)
output_submission_csv('./nn_3layer_adam_submission.csv', best_3layer_adam_prediction)
```

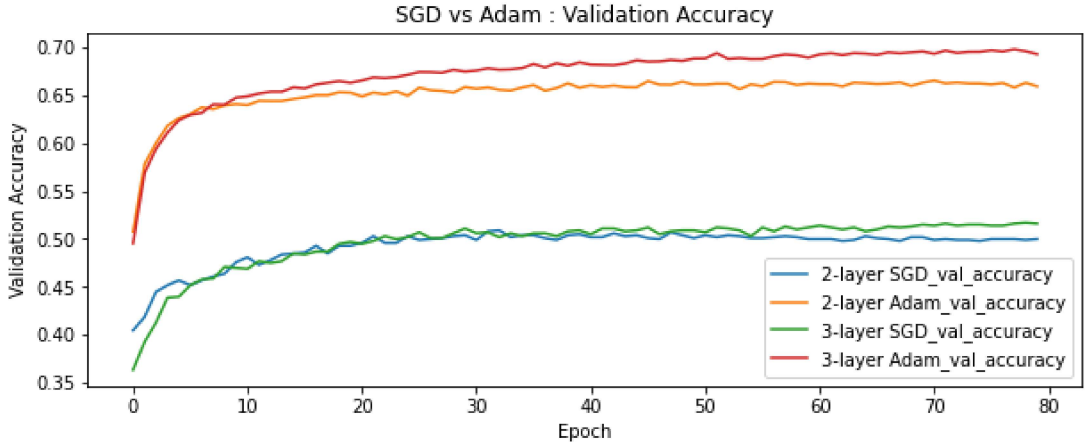
Compare SGD and Adam

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.

In [87]:
```
# TODO: implement me
plt.subplot(2, 1, 1)
plt.plot(train_loss, label='2-layer SGD_train_loss')
plt.plot(train_loss_adam, label='2-layer Adam_train_loss')
plt.plot(train_loss3, label='3-layer SGD_train_loss')
plt.plot(train_loss_adam3, label='3-layer Adam_train_loss')
plt.title('SGD vs Adam : Train Loss')
plt.xlabel('Epoch')
plt.ylabel('train loss')
plt.legend()
plt.show()

plt.subplot(2, 1, 2)
plt.plot(val_accuracy, label='2-layer SGD_val_accuracy')
plt.plot(val_accuracy_adam, label='2-layer Adam_val_accuracy')
plt.plot(val_accuracy3, label='3-layer SGD_val_accuracy')
plt.plot(val_accuracy_adam3, label='3-layer Adam_val_accuracy')
plt.title('SGD vs Adam : Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Validation Accuracy')
plt.legend()
plt.show()
```


SGD vs Adam : Train Loss
train loss
Epoch
2-layer SGD_train_loss
2-layer Adam_train_loss
3-layer SGD_train_loss
3-layer Adam_train_loss


SGD vs Adam : Validation Accuracy
Validation Accuracy
Epoch
2-layer SGD_val_accuracy
2-layer Adam_val_accuracy
3-layer SGD_val_accuracy
3-layer Adam_val_accuracy