neural_network

October 8, 2020

1 CS498DL Assignment 2

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import math
from sklearn.utils import shuffle

from kaggle_submission import output_submission_csv
from models.neural_net import NeuralNetwork
from models.adam import AdamNeuralNetwork
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
```

1.1 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.

[2]:
```
# 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']
```

1.2 Train using 2 Layer 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.

[3]:
```
# Hyperparameters
input_size = 32 * 32 * 3
num_layers = 2
hidden_size = 150
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 70
batch_size = 100
learning_rate = 1e-1
learning_rate_decay = 0.95
regularization = 0.001

# 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_2sgd = np.zeros(epochs)
train_accuracy_2sgd = np.zeros(epochs)
val_accuracy_2sgd = np.zeros(epochs)


highest_accuracy = -math.inf
best_network = 0

# 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...
    batch_accuracy = []
    f_accuracy = []
    count = 0
    # https://stackoverflow.com/questions/8177079/
↪take-the-content-of-a-list-and-append-it-to-another-list
    #https://stackoverflow.com/questions/60133145/
↪neural-networks-from-scratch-problem-with-fit-method-when-i-attempt-to-use-mini
    rnd_idx = np.random.permutation(TRAIN_IMAGES)
```

```
    n_batches = TRAIN_IMAGES//batch_size
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_batch = X_train[batch_idx]
        y_batch = y_train[batch_idx]
        f_output = np.argmax(net.forward(X_batch), axis = 1)
        loss = net.backward(X_batch, y_batch, learning_rate, regularization)
        train_loss_2sgd[epoch] += loss
        batch_accuracy.extend(y_batch)
        f_accuracy.extend(f_output)
    #https://stackoverflow.com/questions/25490641/
↪check-how-many-elements-are-equal-in-two-numpy-arrays-python/25490691
    batch_array, f_array = np.array(batch_accuracy), np.array(f_accuracy)
    same_value_count = (batch_array == f_array).sum()
    train_accuracy_2sgd[epoch] = same_value_count / len(batch_accuracy)
    train_loss_2sgd[epoch] /= n_batches

    # Validation
    # No need to run the backward pass here, just run the forward pass to␣
↪compute accuracy
    batch_accuracy2 = []
    real_accuracy = []
    rnd_idx = np.random.permutation(VAL_IMAGES)
    n_batches = VAL_IMAGES // batch_size
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_val_batch = X_val[batch_idx]
        y_val_batch = y_val[batch_idx]
        f_output2 = np.argmax(net.forward(X_val_batch), axis = 1)
        batch_accuracy2.extend(y_val_batch)
        real_accuracy.extend(f_output2)

    batch_array2, real_array = np.array(batch_accuracy2), np.array(real_accuracy)
    same_value_count = (batch_array2 == real_array).sum()
    val_accuracy_2sgd[epoch] = same_value_count / len(batch_accuracy2)

    learning_rate = learning_rate * learning_rate_decay
    print("epoch:", epoch, " ", "acc:", val_accuracy_2sgd[epoch], " ", "loss:",␣
↪train_loss_2sgd[epoch])

    if val_accuracy_2sgd[epoch] > highest_accuracy:
        highest_accuracy = val_accuracy_2sgd[epoch]
        best_network = net
#print(highest_accuracy = val_accuracy_2sgd[epoch])
sgd_2_best = best_network
#print(train_accuracy, val_accuracy)
```

```
epoch: 0   acc: 0.413   loss: 1.7563669946069924
epoch: 1   acc: 0.443   loss: 1.5438423856611445
```

```
epoch: 2   acc: 0.463   loss: 1.4599814710064292
epoch: 3   acc: 0.494   loss: 1.3918972028592211
epoch: 4   acc: 0.471   loss: 1.3392911948766097
epoch: 5   acc: 0.493   loss: 1.2812650037289495
epoch: 6   acc: 0.522   loss: 1.236780642646892
epoch: 7   acc: 0.522   loss: 1.189345546707765
epoch: 8   acc: 0.519   loss: 1.1518541454140006
epoch: 9   acc: 0.521   loss: 1.104030650890289
epoch: 10   acc: 0.516   loss: 1.0581000375200234
epoch: 11   acc: 0.532   loss: 1.0293862920692654
epoch: 12   acc: 0.52   loss: 0.9959116887023791
epoch: 13   acc: 0.53   loss: 0.9554296541287884
epoch: 14   acc: 0.527   loss: 0.9271799393712058
epoch: 15   acc: 0.546   loss: 0.8933549138988446
epoch: 16   acc: 0.505   loss: 0.8642675600931842
epoch: 17   acc: 0.515   loss: 0.8332728092188175
epoch: 18   acc: 0.534   loss: 0.8019122304678645
epoch: 19   acc: 0.524   loss: 0.7765748372229494
epoch: 20   acc: 0.531   loss: 0.7468079941339683
epoch: 21   acc: 0.532   loss: 0.722425651597218
epoch: 22   acc: 0.527   loss: 0.699578797573141
epoch: 23   acc: 0.529   loss: 0.6773617496794496
epoch: 24   acc: 0.529   loss: 0.651246669071911
epoch: 25   acc: 0.514   loss: 0.6283432263374437
epoch: 26   acc: 0.528   loss: 0.6058219020988505
epoch: 27   acc: 0.521   loss: 0.5859774512133323
epoch: 28   acc: 0.536   loss: 0.5681205970182388
epoch: 29   acc: 0.519   loss: 0.5471216694465378
epoch: 30   acc: 0.527   loss: 0.5272340784789056
epoch: 31   acc: 0.529   loss: 0.5100077842787812
epoch: 32   acc: 0.539   loss: 0.4933057201546159
epoch: 33   acc: 0.525   loss: 0.4788375370565466
epoch: 34   acc: 0.523   loss: 0.4623768897108158
epoch: 35   acc: 0.529   loss: 0.45023317161971665
epoch: 36   acc: 0.519   loss: 0.4358153345809717
epoch: 37   acc: 0.519   loss: 0.4212805102898406
epoch: 38   acc: 0.526   loss: 0.40940383584342005
epoch: 39   acc: 0.532   loss: 0.39778599629098677
epoch: 40   acc: 0.525   loss: 0.38771753374881535
epoch: 41   acc: 0.522   loss: 0.37593970193022674
epoch: 42   acc: 0.526   loss: 0.36701591723990157
epoch: 43   acc: 0.523   loss: 0.3583325039232834
epoch: 44   acc: 0.527   loss: 0.34990466669919207
epoch: 45   acc: 0.519   loss: 0.34151229089079127
epoch: 46   acc: 0.529   loss: 0.3335472648297409
epoch: 47   acc: 0.518   loss: 0.3259827829548725
epoch: 48   acc: 0.519   loss: 0.3194882269351194
epoch: 49   acc: 0.527   loss: 0.3132660913254858
```

```
epoch: 50   acc: 0.53   loss: 0.30714110153183893
epoch: 51   acc: 0.531   loss: 0.301757441382403
epoch: 52   acc: 0.526   loss: 0.29629948386607935
epoch: 53   acc: 0.523   loss: 0.2917078403090416
epoch: 54   acc: 0.516   loss: 0.2864945317770245
epoch: 55   acc: 0.527   loss: 0.2816806259150947
epoch: 56   acc: 0.524   loss: 0.2771672600751209
epoch: 57   acc: 0.524   loss: 0.2738857265616342
epoch: 58   acc: 0.526   loss: 0.26977208521906676
epoch: 59   acc: 0.514   loss: 0.2668132504968874
epoch: 60   acc: 0.514   loss: 0.2635206404631505
epoch: 61   acc: 0.52   loss: 0.26015206312248285
epoch: 62   acc: 0.518   loss: 0.25704643906467894
epoch: 63   acc: 0.527   loss: 0.2546543821996699
epoch: 64   acc: 0.518   loss: 0.2518163584626345
epoch: 65   acc: 0.519   loss: 0.24924726829650862
epoch: 66   acc: 0.527   loss: 0.247200100789656
epoch: 67   acc: 0.524   loss: 0.2445837835702126
epoch: 68   acc: 0.517   loss: 0.24253686323424595
epoch: 69   acc: 0.523   loss: 0.24096948788379302
```

1.3 Train using 3 Layer 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.

[4]:
```python
# Hyperparameters
input_size = 32 * 32 * 3
num_layers = 3
hidden_size = 150
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 70
batch_size = 100
learning_rate = 1e-1
learning_rate_decay = 0.95
regularization = 0.001

# 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_3sgd = np.zeros(epochs)
train_accuracy_3sgd = np.zeros(epochs)
val_accuracy_3sgd = np.zeros(epochs)
```

```
highest_accuracy = -math.inf
best_network = 0

# 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...
    batch_accuracy = []
    f_accuracy = []
    count = 0
    # https://stackoverflow.com/questions/8177079/
 ↪take-the-content-of-a-list-and-append-it-to-another-list
    #https://stackoverflow.com/questions/60133145/
 ↪neural-networks-from-scratch-problem-with-fit-method-when-i-attempt-to-use-mini
    rnd_idx = np.random.permutation(TRAIN_IMAGES)
    n_batches = TRAIN_IMAGES//batch_size
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_batch = X_train[batch_idx]
        y_batch = y_train[batch_idx]
        f_output = np.argmax(net.forward(X_batch), axis = 1)
        loss = net.backward(X_batch, y_batch, learning_rate, regularization)
        train_loss_3sgd[epoch] += loss
        batch_accuracy.extend(y_batch)
        f_accuracy.extend(f_output)
    #https://stackoverflow.com/questions/25490641/
 ↪check-how-many-elements-are-equal-in-two-numpy-arrays-python/25490691
    batch_array, f_array = np.array(batch_accuracy), np.array(f_accuracy)
    same_value_count = (batch_array == f_array).sum()
    train_accuracy_3sgd[epoch] = same_value_count / len(batch_accuracy)
    train_loss_3sgd[epoch] /= n_batches

    # Validation
    # No need to run the backward pass here, just run the forward pass to␣
 ↪compute accuracy
    batch_accuracy2 = []
    real_accuracy = []
    rnd_idx = np.random.permutation(VAL_IMAGES)
    n_batches = VAL_IMAGES // batch_size
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_val_batch = X_val[batch_idx]
        y_val_batch = y_val[batch_idx]
```

```
        f_output2 = np.argmax(net.forward(X_val_batch), axis = 1)
        batch_accuracy2.extend(y_val_batch)
        real_accuracy.extend(f_output2)

    batch_array2, real_array = np.array(batch_accuracy2), np.array(real_accuracy)
    same_value_count = (batch_array2 == real_array).sum()
    val_accuracy_3sgd[epoch] = same_value_count / len(batch_accuracy2)

    learning_rate = learning_rate * learning_rate_decay
    print("epoch:", epoch, " ", "acc:", val_accuracy_3sgd[epoch], " ", "loss:",␣
 ↪train_loss_3sgd[epoch])

    if val_accuracy_3sgd[epoch] > highest_accuracy:
        highest_accuracy = val_accuracy_3sgd[epoch]
        best_network = net
sgd_3_best = best_network
#print(train_accuracy, val_accuracy)
```

```
epoch: 0   acc: 0.453   loss: 1.677004023009174
epoch: 1   acc: 0.488   loss: 1.45379953649261
epoch: 2   acc: 0.5   loss: 1.3503942934124455
epoch: 3   acc: 0.502   loss: 1.2714836663385856
epoch: 4   acc: 0.535   loss: 1.2072313849984577
epoch: 5   acc: 0.535   loss: 1.1474144874895063
epoch: 6   acc: 0.537   loss: 1.089623937519086
epoch: 7   acc: 0.533   loss: 1.0401858125122139
epoch: 8   acc: 0.521   loss: 0.9858961726096771
epoch: 9   acc: 0.527   loss: 0.9398279342703992
epoch: 10   acc: 0.536   loss: 0.8976653978214381
epoch: 11   acc: 0.55   loss: 0.8561262059031347
epoch: 12   acc: 0.535   loss: 0.8118510697673569
epoch: 13   acc: 0.536   loss: 0.7702320045238036
epoch: 14   acc: 0.543   loss: 0.7320647528822489
epoch: 15   acc: 0.55   loss: 0.6925576170680616
epoch: 16   acc: 0.53   loss: 0.6559131392959091
epoch: 17   acc: 0.546   loss: 0.621298259701281
epoch: 18   acc: 0.546   loss: 0.5858791756710477
epoch: 19   acc: 0.53   loss: 0.5510466674917333
epoch: 20   acc: 0.535   loss: 0.5211836212746989
epoch: 21   acc: 0.55   loss: 0.49002709488133167
epoch: 22   acc: 0.538   loss: 0.46188520469187094
epoch: 23   acc: 0.536   loss: 0.4355002620764879
epoch: 24   acc: 0.541   loss: 0.40856902617885754
epoch: 25   acc: 0.533   loss: 0.3817903786313522
epoch: 26   acc: 0.539   loss: 0.3594901382527652
epoch: 27   acc: 0.544   loss: 0.33670577119408607
epoch: 28   acc: 0.537   loss: 0.31499935217478275
```

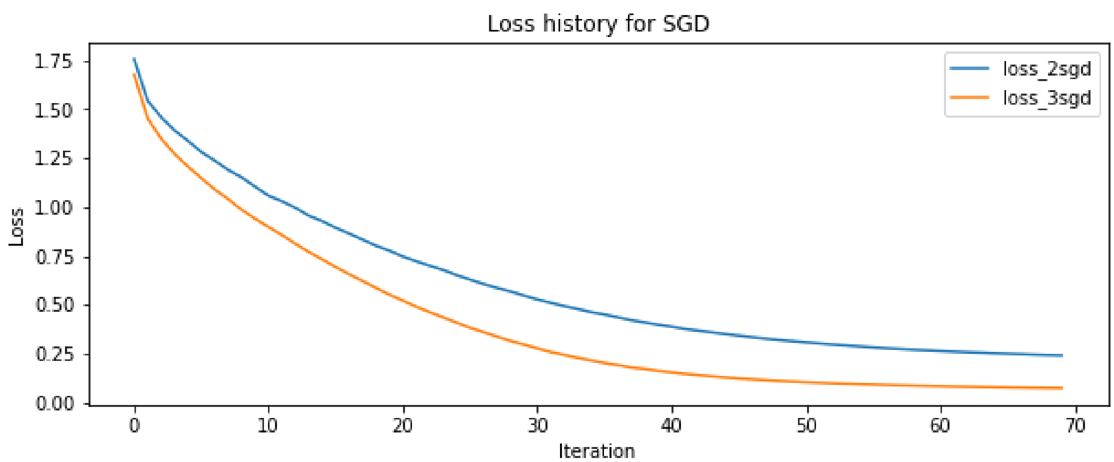
```
epoch: 29   acc: 0.524   loss: 0.2957583235544671
epoch: 30   acc: 0.539   loss: 0.2766411366768283
epoch: 31   acc: 0.527   loss: 0.2584630079169545
epoch: 32   acc: 0.528   loss: 0.24338492427828637
epoch: 33   acc: 0.535   loss: 0.22859972183867108
epoch: 34   acc: 0.532   loss: 0.215376542629848
epoch: 35   acc: 0.532   loss: 0.2016927290373801
epoch: 36   acc: 0.519   loss: 0.1914267271535204
epoch: 37   acc: 0.525   loss: 0.17908581934523832
epoch: 38   acc: 0.52   loss: 0.1714844415081876
epoch: 39   acc: 0.529   loss: 0.16169183725701966
epoch: 40   acc: 0.529   loss: 0.15443954754005157
epoch: 41   acc: 0.522   loss: 0.1466140476160938
epoch: 42   acc: 0.527   loss: 0.13991024969530755
epoch: 43   acc: 0.528   loss: 0.13404049682639296
epoch: 44   acc: 0.527   loss: 0.12834543681782018
epoch: 45   acc: 0.533   loss: 0.12323527077588133
epoch: 46   acc: 0.524   loss: 0.1188967831827395
epoch: 47   acc: 0.528   loss: 0.11445545037272246
epoch: 48   acc: 0.531   loss: 0.11053452176646422
epoch: 49   acc: 0.525   loss: 0.10728947916553591
epoch: 50   acc: 0.526   loss: 0.103857689215035
epoch: 51   acc: 0.522   loss: 0.10102401469290313
epoch: 52   acc: 0.526   loss: 0.0981508253297874
epoch: 53   acc: 0.521   loss: 0.09577580009807198
epoch: 54   acc: 0.525   loss: 0.09345375002591083
epoch: 55   acc: 0.529   loss: 0.0913514905968868
epoch: 56   acc: 0.529   loss: 0.0891952829088873
epoch: 57   acc: 0.531   loss: 0.08749426029238787
epoch: 58   acc: 0.521   loss: 0.08573575274236875
epoch: 59   acc: 0.529   loss: 0.08430591368606213
epoch: 60   acc: 0.528   loss: 0.08276065118463065
epoch: 61   acc: 0.526   loss: 0.08143878208804083
epoch: 62   acc: 0.529   loss: 0.08009480476018839
epoch: 63   acc: 0.528   loss: 0.07897881510265979
epoch: 64   acc: 0.525   loss: 0.07789992750329727
epoch: 65   acc: 0.53   loss: 0.07682062807604528
epoch: 66   acc: 0.529   loss: 0.07585787440081536
epoch: 67   acc: 0.525   loss: 0.07488379334436962
epoch: 68   acc: 0.527   loss: 0.0741097806918038
epoch: 69   acc: 0.524   loss: 0.07327738226395974
```

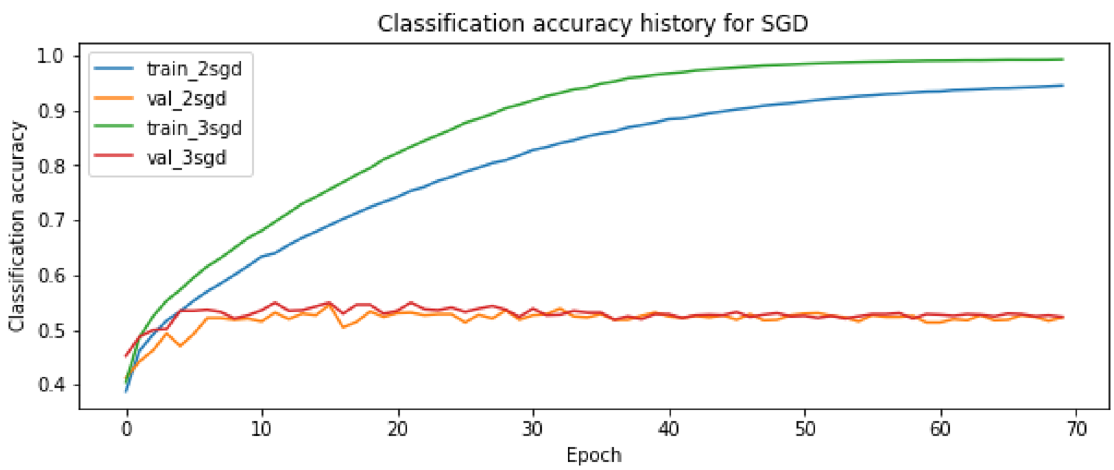
1.4 Graph loss and train/val accuracies for SGD

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.

[5]:
```
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_loss_2sgd, label = 'loss_2sgd')
plt.plot(train_loss_3sgd, label = 'loss_3sgd')
plt.title('Loss history for SGD')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()
plt.show()


plt.subplot(2, 1, 2)
plt.plot(train_accuracy_2sgd, label='train_2sgd')
plt.plot(val_accuracy_2sgd, label='val_2sgd')
plt.plot(train_accuracy_3sgd, label='train_3sgd')
plt.plot(val_accuracy_3sgd, label='val_3sgd')
plt.title('Classification accuracy history for SGD')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()
plt.show()
```





1.5 Train using 2 Layer Adam

To train our network we will use Adam. 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.

[6]:
```
# Hyperparameters
input_size = 32 * 32 * 3
num_layers = 2
hidden_size = 150
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 70
batch_size = 100
learning_rate = 1e-3
learning_rate_decay = 0.95
regularization = 0.001
beta_one = 0.9
beta_two = 0.999
epil = 1e-8

# Initialize a new neural network model
net = AdamNeuralNetwork(input_size, hidden_sizes, num_classes, num_layers)

# Variables to store performance for each epoch
train_loss_2adam = np.zeros(epochs)
train_accuracy_2adam = np.zeros(epochs)
val_accuracy_2adam = np.zeros(epochs)
```

```python
highest_accuracy = -math.inf
best_network = 0

# 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...
    batch_accuracy = []
    f_accuracy = []
    count = 0
    # https://stackoverflow.com/questions/8177079/
↪take-the-content-of-a-list-and-append-it-to-another-list
    #https://stackoverflow.com/questions/60133145/
↪neural-networks-from-scratch-problem-with-fit-method-when-i-attempt-to-use-mini
    rnd_idx = np.random.permutation(TRAIN_IMAGES)
    n_batches = TRAIN_IMAGES//batch_size
    time = 0
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_batch = X_train[batch_idx]
        y_batch = y_train[batch_idx]
        f_output = np.argmax(net.forward(X_batch), axis = 1)
        time += 1
        loss = net.backward(X_batch, y_batch, learning_rate, time, beta_one,␣
↪beta_two, epil, regularization)
        train_loss_2adam[epoch] += loss
        batch_accuracy.extend(y_batch)
        f_accuracy.extend(f_output)
    #https://stackoverflow.com/questions/25490641/
↪check-how-many-elements-are-equal-in-two-numpy-arrays-python/25490691
    batch_array, f_array = np.array(batch_accuracy), np.array(f_accuracy)
    same_value_count = (batch_array == f_array).sum()
    train_accuracy_2adam[epoch] = same_value_count / len(batch_accuracy)
    train_loss_2adam[epoch] /= n_batches

    # Validation
    # No need to run the backward pass here, just run the forward pass to␣
↪compute accuracy
    batch_accuracy2 = []
    real_accuracy = []
    rnd_idx = np.random.permutation(VAL_IMAGES)
    n_batches = VAL_IMAGES // batch_size
    for batch_idx in np.array_split(rnd_idx, n_batches):
```

```
        X_val_batch = X_val[batch_idx]
        y_val_batch = y_val[batch_idx]
        f_output2 = np.argmax(net.forward(X_val_batch), axis = 1)
        batch_accuracy2.extend(y_val_batch)
        real_accuracy.extend(f_output2)

    batch_array2, real_array = np.array(batch_accuracy2), np.array(real_accuracy)
    same_value_count = (batch_array2 == real_array).sum()
    val_accuracy_2adam[epoch] = same_value_count / len(batch_accuracy2)

    learning_rate = learning_rate * learning_rate_decay
    print("epoch:", epoch, " ", "acc:", val_accuracy_2adam[epoch], " ", "loss:",␣
 ↪train_loss_2adam[epoch])

    if val_accuracy_2adam[epoch] > highest_accuracy:
        highest_accuracy = val_accuracy_2adam[epoch]
        best_network = net
sgd_2_best_adam = best_network
print(highest_accuracy)
#print(train_accuracy, val_accuracy)
```

```
epoch: 0   acc: 0.455   loss: 1.781848481853594
epoch: 1   acc: 0.507   loss: 1.4215156700911995
epoch: 2   acc: 0.502   loss: 1.3271816608662605
epoch: 3   acc: 0.506   loss: 1.2755329343779374
epoch: 4   acc: 0.485   loss: 1.2290071052401057
epoch: 5   acc: 0.51   loss: 1.19420813474105
epoch: 6   acc: 0.529   loss: 1.154756255361247
epoch: 7   acc: 0.53   loss: 1.1247534765364682
epoch: 8   acc: 0.529   loss: 1.0950878174020258
epoch: 9   acc: 0.5   loss: 1.0688436459006527
epoch: 10   acc: 0.52   loss: 1.0432271910549251
epoch: 11   acc: 0.519   loss: 1.0191682381363585
epoch: 12   acc: 0.532   loss: 0.9936166995061683
epoch: 13   acc: 0.519   loss: 0.9760135965351561
epoch: 14   acc: 0.523   loss: 0.9517511006939997
epoch: 15   acc: 0.529   loss: 0.9292873752617823
epoch: 16   acc: 0.535   loss: 0.9118295479212173
epoch: 17   acc: 0.526   loss: 0.894332710401648
epoch: 18   acc: 0.535   loss: 0.8768767273358347
epoch: 19   acc: 0.516   loss: 0.8595064917616229
epoch: 20   acc: 0.538   loss: 0.8459411864814331
epoch: 21   acc: 0.537   loss: 0.8292626498421413
epoch: 22   acc: 0.539   loss: 0.8139482023532268
epoch: 23   acc: 0.539   loss: 0.7992217949414239
epoch: 24   acc: 0.529   loss: 0.7882500648581292
epoch: 25   acc: 0.523   loss: 0.7749307679800699
```

```
epoch: 26   acc: 0.531   loss: 0.7615660727751893
epoch: 27   acc: 0.534   loss: 0.7534513010837615
epoch: 28   acc: 0.529   loss: 0.7405757699037835
epoch: 29   acc: 0.543   loss: 0.7291112188397215
epoch: 30   acc: 0.528   loss: 0.7186323274840272
epoch: 31   acc: 0.55   loss: 0.70983000527058
epoch: 32   acc: 0.54   loss: 0.7007016337439341
epoch: 33   acc: 0.526   loss: 0.6928355958573388
epoch: 34   acc: 0.519   loss: 0.6832453685174167
epoch: 35   acc: 0.542   loss: 0.6767654610983624
epoch: 36   acc: 0.515   loss: 0.6680194073307174
epoch: 37   acc: 0.535   loss: 0.6611657172185718
epoch: 38   acc: 0.536   loss: 0.6569975052432504
epoch: 39   acc: 0.53   loss: 0.6478375762047177
epoch: 40   acc: 0.527   loss: 0.6422494132459349
epoch: 41   acc: 0.525   loss: 0.6360803037534456
epoch: 42   acc: 0.535   loss: 0.6309443951003592
epoch: 43   acc: 0.531   loss: 0.6260101047374886
epoch: 44   acc: 0.535   loss: 0.619851770481345
epoch: 45   acc: 0.526   loss: 0.6153586822405722
epoch: 46   acc: 0.531   loss: 0.6104508262444146
epoch: 47   acc: 0.534   loss: 0.6063335428559846
epoch: 48   acc: 0.526   loss: 0.6026869237499377
epoch: 49   acc: 0.532   loss: 0.5991283438611519
epoch: 50   acc: 0.526   loss: 0.5941353287779787
epoch: 51   acc: 0.527   loss: 0.5918242225920124
epoch: 52   acc: 0.531   loss: 0.5873000644489106
epoch: 53   acc: 0.525   loss: 0.5842291593354041
epoch: 54   acc: 0.532   loss: 0.5808482525748033
epoch: 55   acc: 0.534   loss: 0.5786273168354464
epoch: 56   acc: 0.534   loss: 0.5756837558770033
epoch: 57   acc: 0.533   loss: 0.5728897270341072
epoch: 58   acc: 0.536   loss: 0.5703073632336896
epoch: 59   acc: 0.533   loss: 0.5678374946187219
epoch: 60   acc: 0.54   loss: 0.5658667916195199
epoch: 61   acc: 0.531   loss: 0.5631288553140954
epoch: 62   acc: 0.534   loss: 0.5611535719637392
epoch: 63   acc: 0.528   loss: 0.5595607072817882
epoch: 64   acc: 0.528   loss: 0.5575766365714184
epoch: 65   acc: 0.529   loss: 0.5558428471341006
epoch: 66   acc: 0.534   loss: 0.5539401323140312
epoch: 67   acc: 0.525   loss: 0.5525525410563583
epoch: 68   acc: 0.531   loss: 0.55075918248299
epoch: 69   acc: 0.535   loss: 0.5495847779907489
0.55
```

1.6 Train using 3 Layer Adam

To train our network we will use Adam. 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.

[14]:
```
# Hyperparameters
input_size = 32 * 32 * 3
num_layers = 3
hidden_size = 150
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 70
batch_size = 100
learning_rate = 1e-3
learning_rate_decay = 0.95
regularization = 0.001
beta_one = 0.9
beta_two = 0.999
epil = 1e-8

# Initialize a new neural network model
net = AdamNeuralNetwork(input_size, hidden_sizes, num_classes, num_layers)

# Variables to store performance for each epoch
train_loss_3adam = np.zeros(epochs)
train_accuracy_3adam = np.zeros(epochs)
val_accuracy_3adam = np.zeros(epochs)


highest_accuracy = -math.inf
best_network = 0

# 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...
    batch_accuracy = []
    f_accuracy = []
    count = 0
    # https://stackoverflow.com/questions/8177079/
 ↪take-the-content-of-a-list-and-append-it-to-another-list
```

```
    #https://stackoverflow.com/questions/60133145/
↪neural-networks-from-scratch-problem-with-fit-method-when-i-attempt-to-use-mini
    rnd_idx = np.random.permutation(TRAIN_IMAGES)
    n_batches = TRAIN_IMAGES//batch_size
    time = 0
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_batch = X_train[batch_idx]
        y_batch = y_train[batch_idx]
        f_output = np.argmax(net.forward(X_batch), axis = 1)
        time += 1
        loss = net.backward(X_batch, y_batch, learning_rate, time, beta_one,␣
↪beta_two, epil, regularization)
        train_loss_3adam[epoch] += loss
        batch_accuracy.extend(y_batch)
        f_accuracy.extend(f_output)
    #https://stackoverflow.com/questions/25490641/
↪check-how-many-elements-are-equal-in-two-numpy-arrays-python/25490691
    batch_array, f_array = np.array(batch_accuracy), np.array(f_accuracy)
    same_value_count = (batch_array == f_array).sum()
    train_accuracy_3adam[epoch] = same_value_count / len(batch_accuracy)
    train_loss_3adam[epoch] /= n_batches

    # Validation
    # No need to run the backward pass here, just run the forward pass to␣
↪compute accuracy
    batch_accuracy2 = []
    real_accuracy = []
    rnd_idx = np.random.permutation(VAL_IMAGES)
    n_batches = VAL_IMAGES // batch_size
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_val_batch = X_val[batch_idx]
        y_val_batch = y_val[batch_idx]
        f_output2 = np.argmax(net.forward(X_val_batch), axis = 1)
        batch_accuracy2.extend(y_val_batch)
        real_accuracy.extend(f_output2)

    batch_array2, real_array = np.array(batch_accuracy2), np.array(real_accuracy)
    same_value_count = (batch_array2 == real_array).sum()
    val_accuracy_3adam[epoch] = same_value_count / len(batch_accuracy2)

    learning_rate = learning_rate * learning_rate_decay
    print("epoch:", epoch, " ", "acc:", val_accuracy_3adam[epoch], " ", "loss:",␣
↪train_loss_3adam[epoch])

    if val_accuracy_3adam[epoch] > highest_accuracy:
        highest_accuracy = val_accuracy_3adam[epoch]
        best_network = net
```

```python
sgd_3_best_adam = best_network
print(highest_accuracy)
#print(train_accuracy, val_accuracy)
```

```
epoch: 0   acc: 0.443   loss: 1.695785622456197
epoch: 1   acc: 0.482   loss: 1.4349419417617093
epoch: 2   acc: 0.498   loss: 1.337647648172673
epoch: 3   acc: 0.509   loss: 1.2636183772759182
epoch: 4   acc: 0.51   loss: 1.208515871832974
epoch: 5   acc: 0.519   loss: 1.1555652605012536
epoch: 6   acc: 0.524   loss: 1.1099358302035622
epoch: 7   acc: 0.526   loss: 1.0691541169977619
epoch: 8   acc: 0.525   loss: 1.0302290742688571
epoch: 9   acc: 0.539   loss: 0.9955487992393077
epoch: 10   acc: 0.519   loss: 0.9589991222603809
epoch: 11   acc: 0.529   loss: 0.9246341263139255
epoch: 12   acc: 0.537   loss: 0.8977564416436331
epoch: 13   acc: 0.524   loss: 0.8676460375672775
epoch: 14   acc: 0.542   loss: 0.838806086528868
epoch: 15   acc: 0.521   loss: 0.8138471652958539
epoch: 16   acc: 0.523   loss: 0.787427493895793
epoch: 17   acc: 0.553   loss: 0.7637115132717165
epoch: 18   acc: 0.526   loss: 0.7421986144563922
epoch: 19   acc: 0.535   loss: 0.7197181957468192
epoch: 20   acc: 0.536   loss: 0.6972338024747221
epoch: 21   acc: 0.534   loss: 0.6791092822887249
epoch: 22   acc: 0.52   loss: 0.6607535171627347
epoch: 23   acc: 0.538   loss: 0.6420326299914598
epoch: 24   acc: 0.535   loss: 0.6227775324514834
epoch: 25   acc: 0.523   loss: 0.607008924831012
epoch: 26   acc: 0.525   loss: 0.5928696041234346
epoch: 27   acc: 0.52   loss: 0.5774938586144541
epoch: 28   acc: 0.535   loss: 0.5634221850152258
epoch: 29   acc: 0.536   loss: 0.5505526434679565
epoch: 30   acc: 0.533   loss: 0.5388774524582731
epoch: 31   acc: 0.532   loss: 0.5260930796628566
epoch: 32   acc: 0.529   loss: 0.5153074824454947
epoch: 33   acc: 0.54   loss: 0.5046332809115942
epoch: 34   acc: 0.523   loss: 0.4944624123586097
epoch: 35   acc: 0.521   loss: 0.4849217970528342
epoch: 36   acc: 0.516   loss: 0.4749705861681502
epoch: 37   acc: 0.527   loss: 0.46735566480727947
epoch: 38   acc: 0.532   loss: 0.45670911866997477
epoch: 39   acc: 0.517   loss: 0.44936412533368064
epoch: 40   acc: 0.522   loss: 0.44212804147397816
epoch: 41   acc: 0.526   loss: 0.4348712669618535
epoch: 42   acc: 0.53   loss: 0.4280145822256825
```

```
epoch: 43   acc: 0.518   loss: 0.42176413913112726
epoch: 44   acc: 0.524   loss: 0.41585553552184956
epoch: 45   acc: 0.52   loss: 0.4096427615124636
epoch: 46   acc: 0.519   loss: 0.40406487696911075
epoch: 47   acc: 0.527   loss: 0.39899573873757155
epoch: 48   acc: 0.514   loss: 0.39341209966365626
epoch: 49   acc: 0.535   loss: 0.38927909343304684
epoch: 50   acc: 0.526   loss: 0.3844557162836823
epoch: 51   acc: 0.525   loss: 0.3801241809623601
epoch: 52   acc: 0.533   loss: 0.3764500853599727
epoch: 53   acc: 0.525   loss: 0.3724487310741941
epoch: 54   acc: 0.528   loss: 0.36889221466797584
epoch: 55   acc: 0.523   loss: 0.36557442128421136
epoch: 56   acc: 0.525   loss: 0.361997894084862
epoch: 57   acc: 0.531   loss: 0.35836412516543265
epoch: 58   acc: 0.535   loss: 0.3559525103520796
epoch: 59   acc: 0.518   loss: 0.35328963406948205
epoch: 60   acc: 0.523   loss: 0.350248227089357
epoch: 61   acc: 0.522   loss: 0.3477245600699434
epoch: 62   acc: 0.526   loss: 0.34542030967086956
epoch: 63   acc: 0.53   loss: 0.3428098360003228
epoch: 64   acc: 0.526   loss: 0.34071922942417454
epoch: 65   acc: 0.528   loss: 0.33866178830843063
epoch: 66   acc: 0.519   loss: 0.3370056623940969
epoch: 67   acc: 0.52   loss: 0.33486720605681053
epoch: 68   acc: 0.531   loss: 0.33307852575230334
epoch: 69   acc: 0.518   loss: 0.33191046827364284
0.553
```

1.7 Graph loss and train/val accuracies for Adam

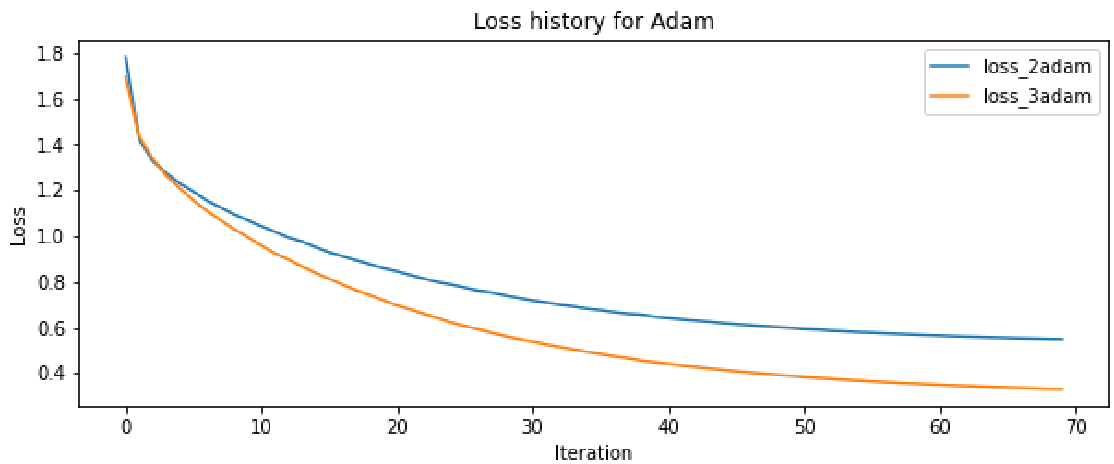
Examining the loss graph along with the train and val accuracy graphs should help you gain some intuition for the hyperparameters you should try in the hyperparameter tuning below. It should also help with debugging any issues you might have with your network.

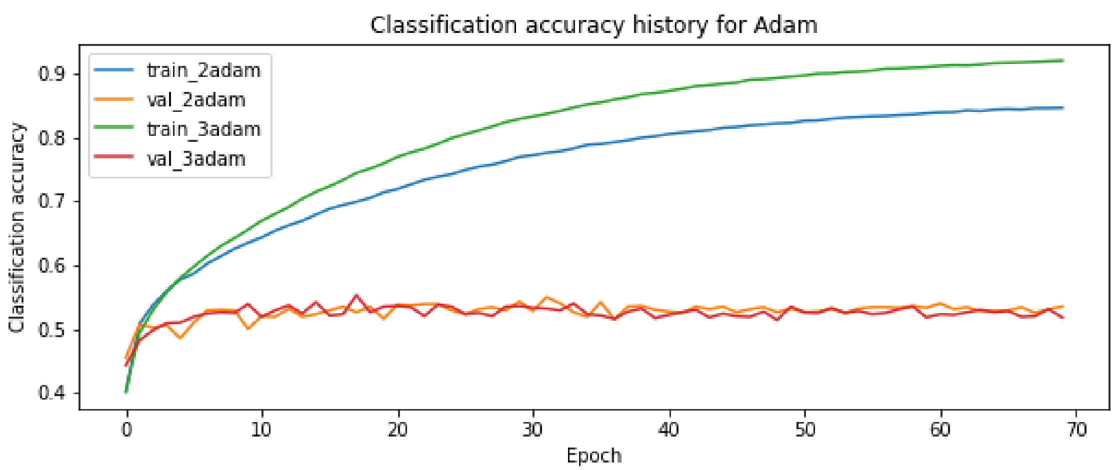
[15]:
```
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_loss_2adam, label='loss_2adam')
plt.plot(train_loss_3adam, label='loss_3adam')
plt.title('Loss history for Adam')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()
plt.show()



plt.subplot(2, 1, 2)
plt.plot(train_accuracy_2adam, label='train_2adam')
```

```
plt.plot(val_accuracy_2adam, label='val_2adam')
plt.plot(train_accuracy_3adam, label='train_3adam')
plt.plot(val_accuracy_3adam, label='val_3adam')
plt.title('Classification accuracy history for Adam')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()
plt.show()
```





1.8 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.

[20]:
```
# Hyperparameters Tuning
input_size = 32 * 32 * 3
num_layers = 2
hidden_size = 25
hidden_sizes = [hidden_size] * (num_layers - 1)
num_classes = 10
epochs = 50
batch_size = 100
learning_rate = 1e-1
learning_rate_decay = 0.95
regularization = 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_2sgd_tune = np.zeros(epochs)
train_accuracy_2sgd_tune = np.zeros(epochs)
val_accuracy_2sgd_tune = np.zeros(epochs)
```

```python
highest_accuracy = -math.inf
best_network = 0

# 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...
    batch_accuracy = []
    f_accuracy = []
    count = 0
    # https://stackoverflow.com/questions/8177079/
 ↪take-the-content-of-a-list-and-append-it-to-another-list
    #https://stackoverflow.com/questions/60133145/
 ↪neural-networks-from-scratch-problem-with-fit-method-when-i-attempt-to-use-mini
    rnd_idx = np.random.permutation(TRAIN_IMAGES)
    n_batches = TRAIN_IMAGES//batch_size
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_batch = X_train[batch_idx]
        y_batch = y_train[batch_idx]
        f_output = np.argmax(net.forward(X_batch), axis = 1)
        loss = net.backward(X_batch, y_batch, learning_rate, regularization)
        train_loss_2sgd_tune[epoch] += loss
        batch_accuracy.extend(y_batch)
        f_accuracy.extend(f_output)
    #https://stackoverflow.com/questions/25490641/
 ↪check-how-many-elements-are-equal-in-two-numpy-arrays-python/25490691
    batch_array, f_array = np.array(batch_accuracy), np.array(f_accuracy)
    same_value_count = (batch_array == f_array).sum()
    train_accuracy_2sgd_tune[epoch] = same_value_count / len(batch_accuracy)
    train_loss_2sgd_tune[epoch] /= n_batches

    # Validation
    # No need to run the backward pass here, just run the forward pass to␣
 ↪compute accuracy
    batch_accuracy2 = []
    real_accuracy = []
    rnd_idx = np.random.permutation(VAL_IMAGES)
    n_batches = VAL_IMAGES // batch_size
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_val_batch = X_val[batch_idx]
        y_val_batch = y_val[batch_idx]
        f_output2 = np.argmax(net.forward(X_val_batch), axis = 1)
```

```
        batch_accuracy2.extend(y_val_batch)
        real_accuracy.extend(f_output2)

    batch_array2, real_array = np.array(batch_accuracy2), np.array(real_accuracy)
    same_value_count = (batch_array2 == real_array).sum()
    val_accuracy_2sgd_tune[epoch] = same_value_count / len(batch_accuracy2)

    learning_rate = learning_rate * learning_rate_decay
    print("epoch:", epoch, " ", "acc:", val_accuracy_2sgd_tune[epoch], " ",␣
 ↪"loss:", train_loss_2sgd_tune[epoch])

    if val_accuracy_2sgd_tune[epoch] > highest_accuracy:
        highest_accuracy = val_accuracy_2sgd_tune[epoch]
        best_network = net
#print(highest_accuracy = val_accuracy_2sgd[epoch])
sgd_2_best_tune = best_network
#print(train_accuracy, val_accuracy)
```

```
epoch: 0   acc: 0.442   loss: 1.8008444256987126
epoch: 1   acc: 0.437   loss: 1.611029937709001
epoch: 2   acc: 0.455   loss: 1.557684436982191
epoch: 3   acc: 0.465   loss: 1.5194882445381626
epoch: 4   acc: 0.46   loss: 1.487650026603168
epoch: 5   acc: 0.477   loss: 1.464897963941532
epoch: 6   acc: 0.479   loss: 1.4435274695184899
epoch: 7   acc: 0.494   loss: 1.421720912199226
epoch: 8   acc: 0.471   loss: 1.405974528093507
epoch: 9   acc: 0.469   loss: 1.3883570471806888
epoch: 10   acc: 0.478   loss: 1.3760785631736878
epoch: 11   acc: 0.476   loss: 1.362307455376832
epoch: 12   acc: 0.466   loss: 1.3520588663381863
epoch: 13   acc: 0.473   loss: 1.3422022081620555
epoch: 14   acc: 0.48   loss: 1.3288786230150236
epoch: 15   acc: 0.484   loss: 1.3206803529549533
epoch: 16   acc: 0.467   loss: 1.3128390576432816
epoch: 17   acc: 0.49   loss: 1.302610722987751
epoch: 18   acc: 0.486   loss: 1.2946451837291582
epoch: 19   acc: 0.486   loss: 1.2874666959732848
epoch: 20   acc: 0.479   loss: 1.2781390320584836
epoch: 21   acc: 0.475   loss: 1.2715903308143137
epoch: 22   acc: 0.489   loss: 1.2621895831402514
epoch: 23   acc: 0.489   loss: 1.2574908820048958
epoch: 24   acc: 0.48   loss: 1.2526596733617423
epoch: 25   acc: 0.496   loss: 1.2445091814835614
epoch: 26   acc: 0.486   loss: 1.2383504420328693
epoch: 27   acc: 0.472   loss: 1.231774455634069
epoch: 28   acc: 0.481   loss: 1.2265772303853162
```

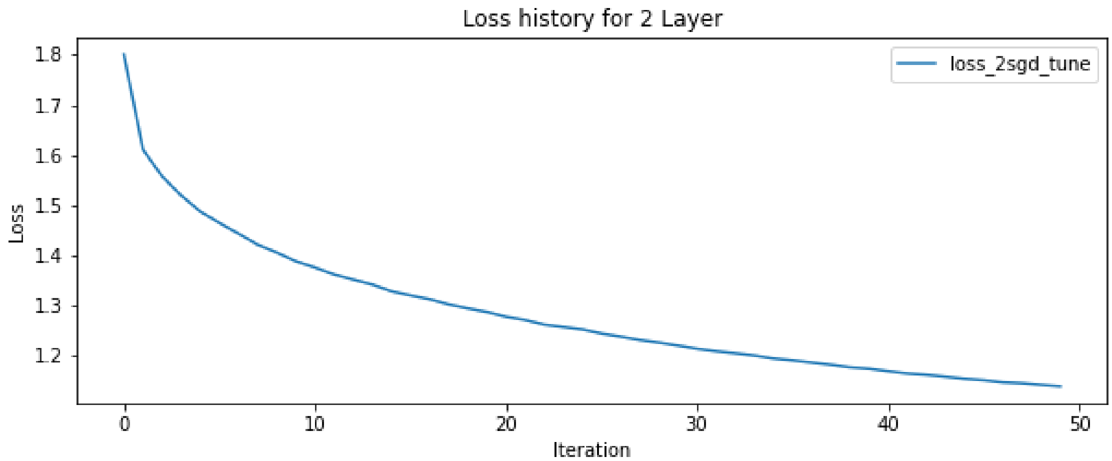
```
epoch: 29   acc: 0.488   loss: 1.2206010136872816
epoch: 30   acc: 0.481   loss: 1.214059339534931
epoch: 31   acc: 0.48   loss: 1.20922372582806
epoch: 32   acc: 0.49   loss: 1.2052462793108731
epoch: 33   acc: 0.478   loss: 1.2008297663977063
epoch: 34   acc: 0.475   loss: 1.1949063440052996
epoch: 35   acc: 0.498   loss: 1.1911228617620973
epoch: 36   acc: 0.477   loss: 1.1867454599291403
epoch: 37   acc: 0.49   loss: 1.1824755758406962
epoch: 38   acc: 0.491   loss: 1.177316755723951
epoch: 39   acc: 0.475   loss: 1.1744851048010752
epoch: 40   acc: 0.484   loss: 1.1693162729161617
epoch: 41   acc: 0.486   loss: 1.1648293416665758
epoch: 42   acc: 0.481   loss: 1.1622548887285458
epoch: 43   acc: 0.488   loss: 1.1584692725388686
epoch: 44   acc: 0.481   loss: 1.154465576847213
epoch: 45   acc: 0.473   loss: 1.1516646826346773
epoch: 46   acc: 0.475   loss: 1.147423874490345
epoch: 47   acc: 0.475   loss: 1.1455399725879165
epoch: 48   acc: 0.484   loss: 1.1422660960441715
epoch: 49   acc: 0.477   loss: 1.1393723519596815
```

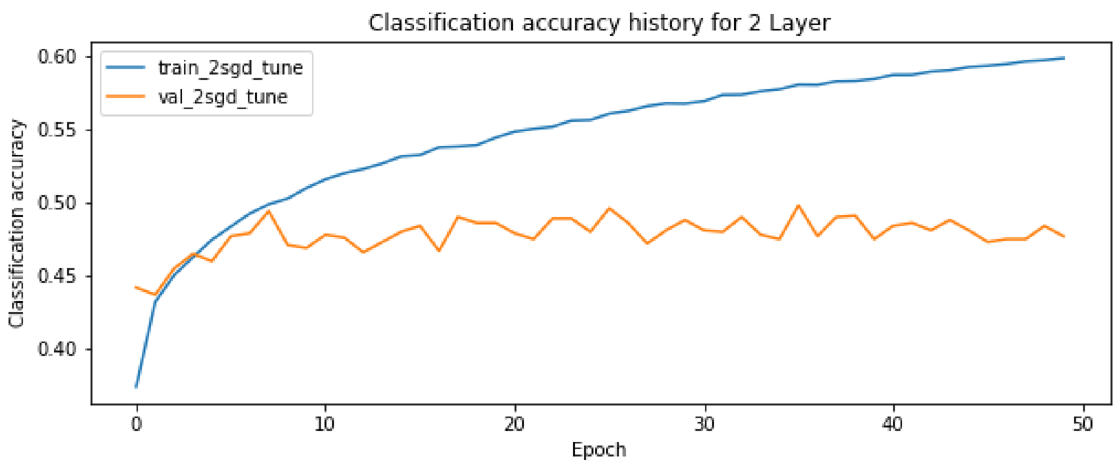
1.9 Graph loss and train/val accuracies for 2 Layer after tuning

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.

[21]:
```python
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_loss_2sgd_tune, label='loss_2sgd_tune')
plt.title('Loss history for 2 Layer')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()
plt.show()


plt.subplot(2, 1, 2)
plt.plot(train_accuracy_2sgd_tune, label='train_2sgd_tune')
plt.plot(val_accuracy_2sgd_tune, label='val_2sgd_tune')
plt.title('Classification accuracy history for 2 Layer')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()
plt.show()
```





1.10 Run on the test set

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

[22]:
```
best_2layer_sgd_prediction = np.argmax(sgd_2_best.forward(X_test), axis = 1)
best_3layer_sgd_prediction = np.argmax(sgd_3_best.forward(X_test), axis = 1)
best_2layer_adam_prediction = np.argmax(sgd_2_best_adam.forward(X_test), axis = 1)
best_3layer_adam_prediction = np.argmax(sgd_3_best_adam.forward(X_test), axis = 1)
```

1.11 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:

[23]:
```
output_submission_csv('kaggle/nn_2layer_sgd_submission.csv',␣
↪best_2layer_sgd_prediction)
output_submission_csv('kaggle/nn_3layer_sgd_submission.csv',␣
↪best_3layer_sgd_prediction)
output_submission_csv('kaggle/nn_2layer_adam_submission.csv',␣
↪best_2layer_adam_prediction)
output_submission_csv('kaggle/nn_3layer_adam_submission.csv',␣
↪best_3layer_adam_prediction)
```

1.12 Compare SGD and Adam

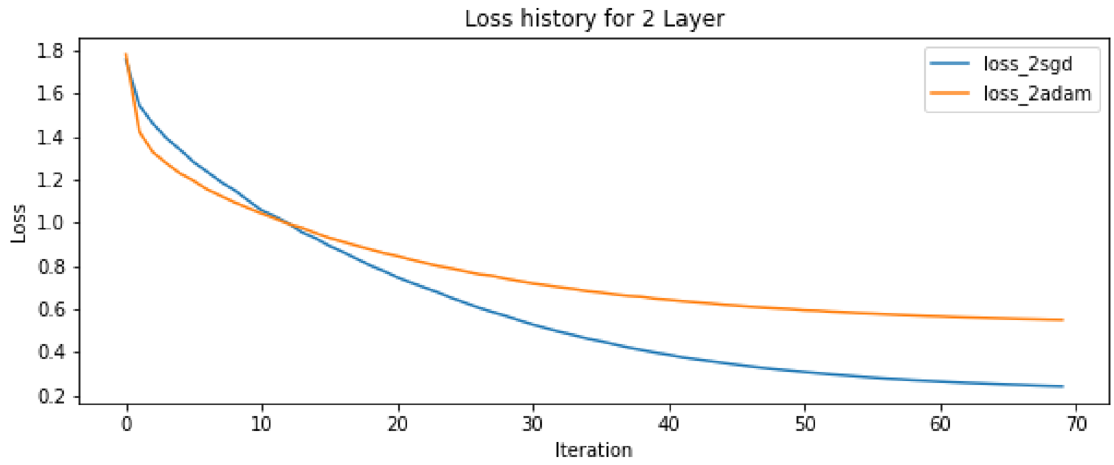
Create graphs to compare training loss and validation accuracy between SGD and Adam. The code is similar to the above code, but instead of comparing train and validation, we are comparing SGD and Adam.

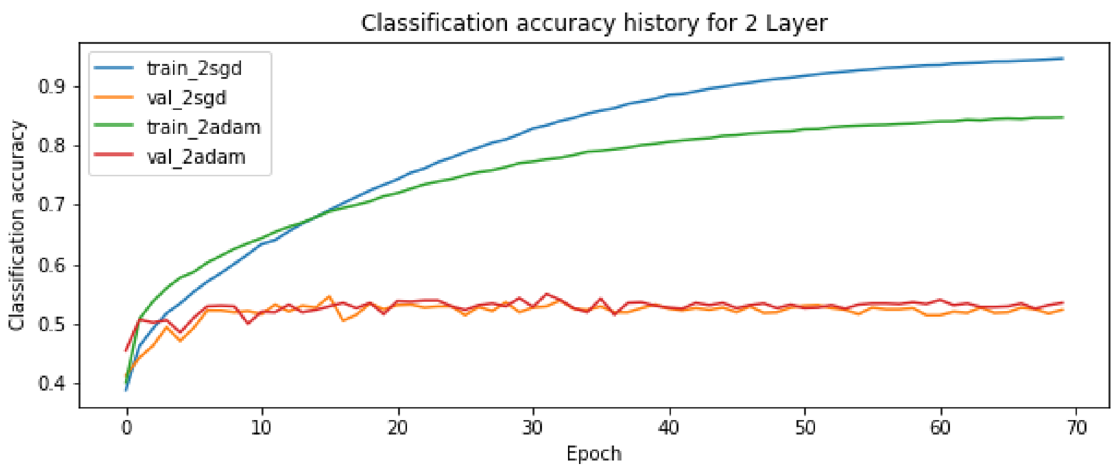
1.13 Graph loss and train/val accuracies for 2 Layer

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.

[24]:
```
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_loss_2sgd, label='loss_2sgd')
plt.plot(train_loss_2adam, label='loss_2adam')
plt.title('Loss history for 2 Layer')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()
plt.show()


plt.subplot(2, 1, 2)
plt.plot(train_accuracy_2sgd, label='train_2sgd')
plt.plot(val_accuracy_2sgd, label='val_2sgd')
plt.plot(train_accuracy_2adam, label='train_2adam')
plt.plot(val_accuracy_2adam, label='val_2adam')
plt.title('Classification accuracy history for 2 Layer')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()
plt.show()
```





1.14 Graph loss and train/val accuracies for 3 Layer

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.

[25]:
```
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_loss_3sgd, label='loss_3sgd')
plt.plot(train_loss_3adam, label='loss_3adam')
plt.title('Loss history for 3 Layer')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()
```

```
plt.show()

plt.subplot(2, 1, 2)
plt.plot(train_accuracy_3sgd, label='train_3sgd')
plt.plot(val_accuracy_3sgd, label='val_3sgd')
plt.plot(train_accuracy_3adam, label='train_3adam')
plt.plot(val_accuracy_3adam, label='val_3adam')
plt.title('Classification accuracy history for 3 Layer')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()
plt.show()
```