# Two-layer Neural Network Workbook for CS145 Homework 3

NAME: Chuang, Kevin UID: 704769121

Please follow the notebook linearly to implement a two layer neural network.

Please print out the workbook entirely when completed.

The goal of this workbook is to give you experience with training a two layer neural network.

Import libraries and define relative error function, which is used to check results later.

In [133]:
```
mport random
mport numpy as np
rom cs145.data_utils import load_CIFAR10
mport matplotlib.pyplot as plt

matplotlib inline
lt.rcParams['figure.figsize'] = (10.0, 8.0)
load_ext autoreload
autoreload 2

ef rel_error(x, y):
   """ returns relative error """
   return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))
```
```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

## Toy example

Before loading CIFAR-10, there will be a toy example to test your implementation of the forward and backward pass.

In [134]:
```
rom lib.neural_net import TwoLayerNet
```

In [135]:
```
 Create a small net and some toy data to check your implementations.
 Note that we set the random seed for repeatable experiments.

nput_size = 4
idden_size = 10
um_classes = 3
um_inputs = 5

ef init_toy_model():
   np.random.seed(0)
   return TwoLayerNet(input_size, hidden_size, num_classes, std=1e-1)

ef init_toy_data():
   np.random.seed(1)
   X = 10 * np.random.randn(num_inputs, input_size)
   y = np.array([0, 1, 2, 2, 1])
   return X, y

et = init_toy_model()
, y = init_toy_data()
```

## Compute forward pass scores

In [136]:
```
# Implement the forward pass of the neural network.

 Note, there is a statement if y is None: return scores, which is why
 the following call will calculate the scores.
cores = net.loss(X)
rint('Your scores:')
rint(scores)
rint()
rint('correct scores:')
orrect_scores = np.asarray([
   [-1.07260209,  0.05083871, -0.87253915],
   [-2.02778743, -0.10832494, -1.52641362],
   [-0.74225908,  0.15259725, -0.39578548],
   [-0.38172726,  0.10835902, -0.17328274],
   [-0.64417314, -0.18886813, -0.41106892]])
rint(correct_scores)
rint()

 The difference should be very small. We get < 1e-7
rint('Difference between your scores and correct scores:')
rint(np.sum(np.abs(scores - correct_scores)))
```

```
our scores:
[-1.07260209  0.05083871 -0.87253915]
[-2.02778743 -0.10832494 -1.52641362]
[-0.74225908  0.15259725 -0.39578548]
[-0.38172726  0.10835902 -0.17328274]
[-0.64417314 -0.18886813 -0.41106892]]

orrect scores:
[-1.07260209  0.05083871 -0.87253915]
[-2.02778743 -0.10832494 -1.52641362]
[-0.74225908  0.15259725 -0.39578548]
[-0.38172726  0.10835902 -0.17328274]
[-0.64417314 -0.18886813 -0.41106892]]

ifference between your scores and correct scores:
.381231233889892e-08
```

## Forward pass loss

The total loss includes data loss (MSE) and regularization loss, which is,

$$L = L_{data} + L_{reg} = \frac{1}{2} \quad_{i=1} (\boldsymbol{y}_{\text{pred}} - \boldsymbol{y}_{\text{target}})^2 + \frac{\lambda}{2} \; || \quad _1||^2 + || \quad _2||^2$$

More specifically in multi-class situation, if the output of neural nets from one sample is $y_{\text{pred}} = (0.1, 0.1, 0.8)$ and $y_{\text{target}} = (0, 0, 1)$ from the given label, then the MSE error will be
$Error = (0.1 - 0)^2 + (0.1 - 0)^2 + (0.8 - 1)^2 = 0.06$

Implement data loss and regularization loss. In the MSE function, you also need to return the gradients which need to be passed backward. This is similar to batch gradient in linear regression. Test your implementation of loss functions. The Difference should be less than 1e-12.

In [84]:
```
oss, _ = net.loss(X, y, reg=0.05)
orrect_loss_MSE = 3.775701133135245 # check this number

 should be very small, we get < 1e-12
rint('Difference between your loss and correct loss:')
rint(np. sum(np.abs(loss - correct_loss_MSE)))
```

```
Difference between your loss and correct loss:
0.0
```

## Backward pass (You do not need to implemented this part) ¶

We have already implemented the backwards pass of the neural network for you. Run the block of code to check your gradients with the gradient check utilities provided. The results should be automatically correct (tiny relative error).

If there is a gradient error larger than 1e-8, the training for neural networks later will be negatively affected.

In [93]:
```
rom cs145.gradient_check import eval_numerical_gradient

 Use numeric gradient checking to check your implementation of the backward p
ss.
 If your implementation is correct, the difference between the numeric and
 analytic gradients should be less than 1e-8 for each of W1, W2, b1, and b2.

oss, grads = net.loss(X, y, reg=0.05)

 these should all be less than 1e-8 or so
or param_name in grads:
   f = lambda W: net.loss(X, y, reg=0.05)[0]
   param_grad_num = eval_numerical_gradient(f, net.params[param_name], verbos
=False)
   print('{} max relative error: {}'.format(param_name, rel_error(param_grad_
um, grads[param_name])))
```

```
2 max relative error: 6.642863886770685e-10
2 max relative error: 2.4554844805570154e-11
1 max relative error: 7.554564328075232e-10
1 max relative error: 7.382451041178829e-10
```

## Training the network

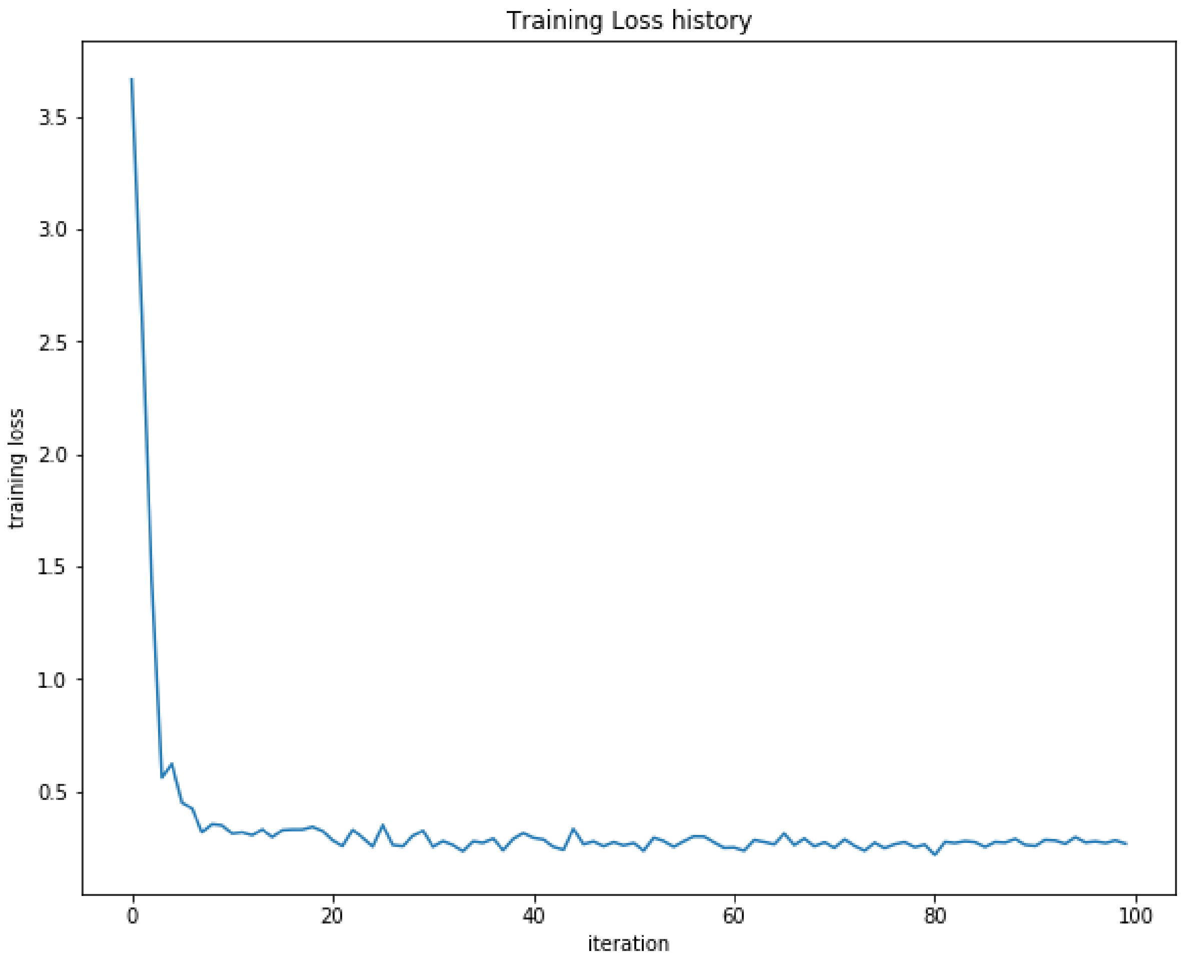
Implement neural_net.train() to train the network via stochastic gradient descent, much like the linear regression.

In [96]:
```
et = init_toy_model()
tats = net.train(X, y, X, y,
           learning_rate=1e-1, reg=5e-6,
           num_iters=100, verbose=False)

rint('Final training loss: ', stats['loss_history'][-1])

 plot the loss history
lt.plot(stats['loss_history'])
lt.xlabel('iteration')
lt.ylabel('training loss')
lt.title('Training Loss history')
lt.show()
```

```
Final training loss:  0.2680575211279022
```



## Classify CIFAR-10

Do classification on the CIFAR-10 dataset.

In [97]:
```
rom cs145.data_utils import load_CIFAR10

ef get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000):
   """
   Load the CIFAR-10 dataset from disk and perform preprocessing to prepare
   it for the two-layer neural net classifier. These are the same steps as
   we used for the SVM, but condensed to a single function.
   """
   # Load the raw CIFAR-10 data
   cifar10_dir = './cs145/datasets/cifar-10-batches-py'
   X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

   # Subsample the data
   mask = list(range(num_training, num_training + num_validation))
   X_val = X_train[mask]
   y_val = y_train[mask]
   mask = list(range(num_training))
   X_train = X_train[mask]
   y_train = y_train[mask]
   mask = list(range(num_test))
   X_test = X_test[mask]
   y_test = y_test[mask]

   # Normalize the data: subtract the mean image
   mean_image = np.mean(X_train, axis=0)
   X_train -= mean_image
   X_val -= mean_image
   X_test -= mean_image

   # Reshape data to rows
   X_train = X_train.reshape(num_training, -1)
   X_val = X_val.reshape(num_validation, -1)
   X_test = X_test.reshape(num_test, -1)

   return X_train, y_train, X_val, y_val, X_test, y_test


 Invoke the above function to get our data.
_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
rint('Train data shape: ', X_train.shape)
rint('Train labels shape: ', y_train.shape)
rint('Validation data shape: ', X_val.shape)
rint('Validation labels shape: ', y_val.shape)
rint('Test data shape: ', X_test.shape)
rint('Test labels shape: ', y_test.shape)
```

```
rain data shape:  (49000, 3072)
rain labels shape:  (49000,)
alidation data shape:  (1000, 3072)
alidation labels shape:  (1000,)
est data shape:  (1000, 3072)
est labels shape:  (1000,)
```

## Running SGD

If your implementation is correct, you should see a validation accuracy of around 15-18%.

In [98]:
```
nput_size = 32 * 32 * 3
idden_size = 50
um_classes = 10
et = TwoLayerNet(input_size, hidden_size, num_classes)

 Train the network
tats = net.train(X_train, y_train, X_val, y_val,
           num_iters=1000, batch_size=200,
           learning_rate=1e-5, learning_rate_decay=0.95,
           reg=0.1, verbose=True)

 Predict on the validation set
al_acc = (net.predict(X_val) == y_val).mean()
rint('Validation accuracy: ', val_acc)

 Save this net as the variable subopt_net for later comparison.
ubopt_net = net
```

```
teration 0 / 1000: loss 1.0000477636152416
teration 100 / 1000: loss 0.9990569291920577
teration 200 / 1000: loss 0.9977657718586511
teration 300 / 1000: loss 0.9955998225596668
teration 400 / 1000: loss 0.9906783608102983
teration 500 / 1000: loss 0.9798023308051768
teration 600 / 1000: loss 0.9577504024675698
teration 700 / 1000: loss 0.9358526467884144
teration 800 / 1000: loss 0.9250220881544526
teration 900 / 1000: loss 0.9166617563494462
alidation accuracy:  0.171
```

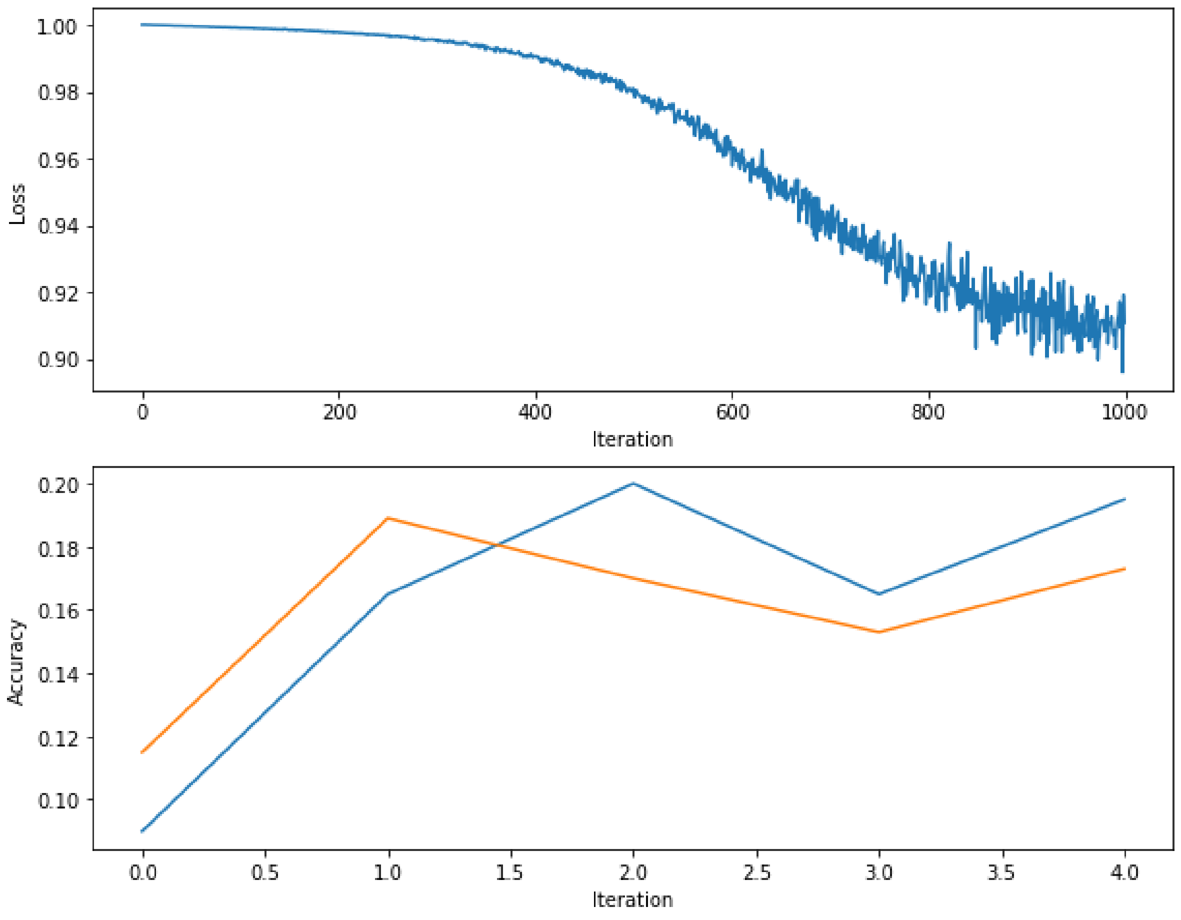
In [99]:
```
tats['train_acc_history']
```

Out[99]:
```
[0.09, 0.165, 0.2, 0.165, 0.195]
```

In [100]:
```
 Plot the loss function and train / validation accuracies
lt.subplot(2, 1, 1)
lt.plot(stats['loss_history'])
lt.xlabel('Iteration')
lt.ylabel('Loss')

lt.subplot(2, 1, 2)
lt.plot(stats['train_acc_history'], label='train')
lt.plot(stats['val_acc_history'], label='val')
lt.xlabel('Iteration')
lt.ylabel('Accuracy')

lt.show()
```



## Questions:

The training accuracy isn't great. It seems even worse than simple KNN model, which is not as good as expected.

(1) What are some of the reasons why this is the case? Take the following cell to do some analyses and then report your answers in the cell following the one below.

(2) How should you fix the problems you identified in (1)?

### Answers:

1. The hyperparameters aren't set to the best values. We have a low number of iterations and incredibly low learning rate. This means that it won't have enough iterations to train and the rate at which it learns is too small for it to make good progress.
2. We can fix this by optimizing our hyperparameters and changing them until we have a more accurate neural network.

## Optimize the neural network

Use the following part of the Jupyter notebook to optimize your hyperparameters on the validation set. Store your nets as best_net.

**Important: Think about whether you should retrain a new model from scratch every time your try a new set of hyperparameters.**

In [129]:
```
est_net = None # store the best model into this

 ================================================================ #
 YOUR CODE HERE:
   Optimize over your hyperparameters to arrive at the best neural
   network.  You should be able to get over 45% validation accuracy.
   For this part of the notebook, we will give credit based on the
   accuracy you get.  Your score on this question will be multiplied by:
      min(floor((X - 23%)) / %22, 1)
   where if you get 50% or higher validation accuracy, you get full
   points.

   Note, you need to use the same network structure (keep hidden_size = 50)!
 ================================================================ #

 todo: optimal parameter search (you can use grid search by for-loops )

nput_size = 32 * 32 * 3 # do not change
idden_size = 50 # do not change
um_classes = 10 # do not change
est_valacc = 0 # do not change
earning_rate=1e-3
earning_rate_decay=0.95
eg=1e-5
um_iters=8000
atch_size=500
 Train the network and find best parameter:
est_net = TwoLayerNet(input_size, hidden_size, num_classes)
est_net.train(X_train, y_train, X_val, y_val, learning_rate=learning_rate, le
rning_rate_decay=learning_rate_decay,
           reg=reg, num_iters=num_iters,
           batch_size=batch_size, verbose=False)
_val_pred = best_net.predict(X_val)
est_valacc = np.mean(y_val == y_val_pred)

 Output your results
rint("== Best parameter settings ==")
rint(f"lr: {learning_rate}, lr_decay: {learning_rate_decay}, reg: {reg}, num_
ters: {num_iters}, batch_size: {batch_size}")
 print your best parameter setting here!
rint("Best accuracy on validation set: {}".format(best_valacc))
 ================================================================ #
 END YOUR CODE HERE
 ================================================================ #
```

```
== Best parameter settings ==
lr: 0.001, lr_decay: 0.95, reg: 1e-05, num_iters: 8000, batch_size: 500
Best accuracy on validation set: 0.502
```

### Quesions

(1) What is your best parameter settings? (Output from the previous cell)

(2) What parameters did you tune? How are they changing the performance of nerural network? You can discuss any observations from the optimization.

### Answers

1. lr: 0.001, lr_decay: 0.95, reg: 1e-05, num_iters: 8000, batch_size: 500
2. I tuned the number of iterations and the batch size. By increasing the number of times it trains, I can increase the accuracy because it has more iterations to learn its weights. In addition adjusting the batch_size can go either way, but this happened to make it more accurate.

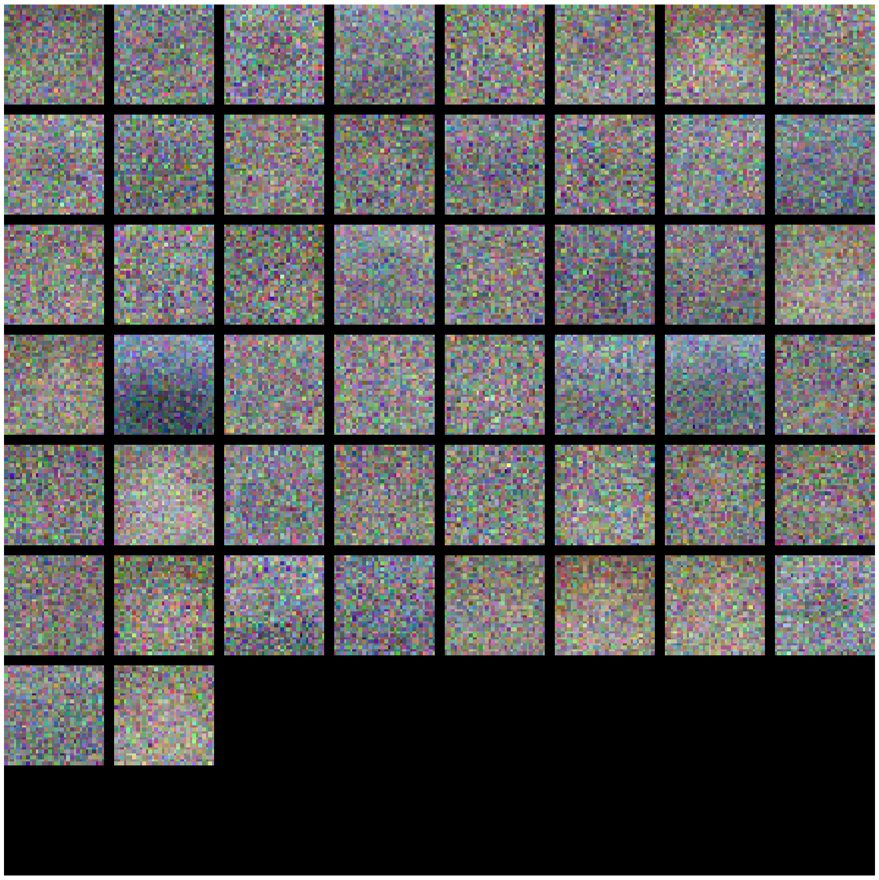
## Visualize the weights of your neural networks

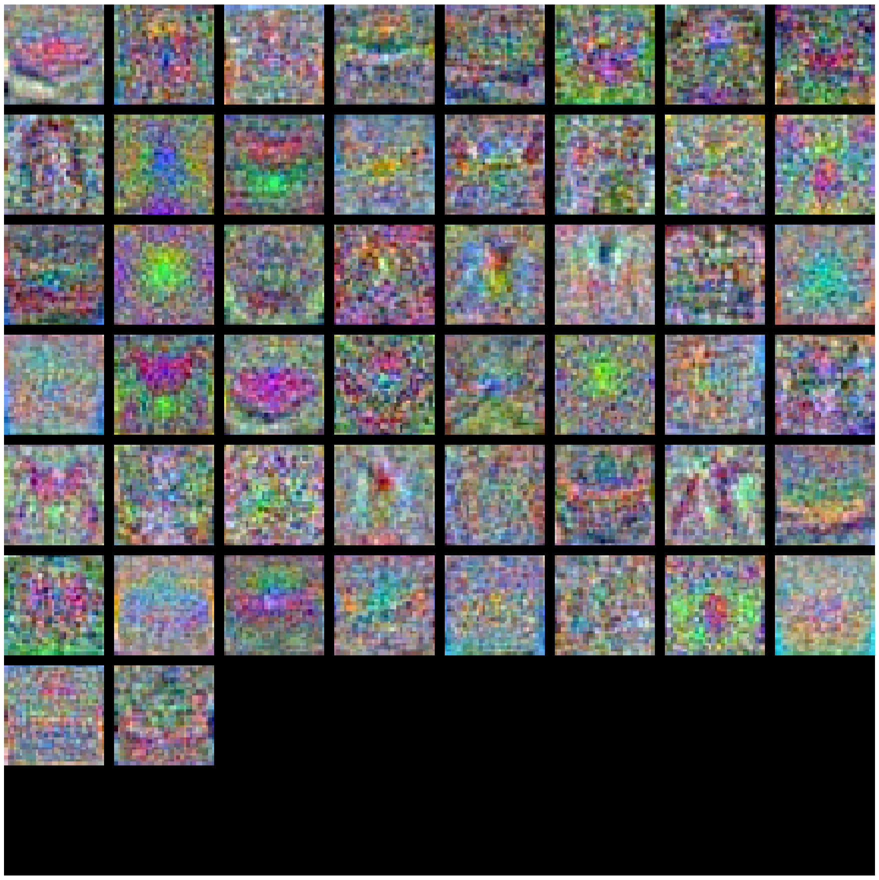
In [130]:
```
rom cs145.vis_utils import visualize_grid

 Visualize the weights of the network

ef show_net_weights(net):
   W1 = net.params['W1']
   W1 = W1.T.reshape(32, 32, 3, -1).transpose(3, 0, 1, 2)
   plt.imshow(visualize_grid(W1, padding=3).astype('uint8'))
   plt.gca().axis('off')
   plt.show()

how_net_weights(subopt_net)
how_net_weights(best_net)
```





### Questions:

What differences do you see in the weights between the suboptimal net and the best net you arrived at? What do the weights in neural networks probably learn after training?

### Answer:

The colors are brighter and you can see that certain weights are more concentrated. After training the weights are much less random and have proper clustering and patterns set to them unlike the suboptimal net. The weights learn which inputs are more important and that certain parts are more important rather than seemingly randomly in the suboptimal net.

## Evaluate on test set

In [131]:
```
est_acc = (best_net.predict(X_test) == y_test).mean()
test_acc = (subopt_net.predict(X_test) == y_test).mean()
rint('Test accuracy: ', test_acc)
```

```
Test accuracy:  0.496
```

### Questions:

(1) What is your test accuracy by using the best NN you have got? How much does the performance increase compared with kNN? Why can neural networks perform better than kNN?

(2) Do you have any other ideas or suggestions to further improve the performance of neural networks other than the parameters you have tried in the homework?

### Answers:

1. The best test accuracy I have is .496 which is much better than .274 of KNN. KNN are just a lazy version of Neural Networks and don't learn anything, instead they only look at the training data to predict the output. Neural networks will instead try to generate an equation basically to base off the output.
2. We could train the model on more data and train it for longer. Either new separate pieces of data, or put it all into one big cluster of data and train the neural network on that. We could also use a different type of loss function which could boost our accuracy.

## Bonus Question: Change MSE Loss to Cross Entropy Loss

This is a bonus question. If you finish this (cross entropy loss) correctly, you will get **up to 20 points** (add up to your HW3 score).

Note: From grading policy of this course, your maximum points from homework are still 25 out of 100, but you can use the bonus question to make up other deduction of other assignments.

Pass output scores in networks from forward pass into softmax function. The softmax function is defined as,

$$p_j = \sigma(z_j) = \frac{e^{z_j}}{{}_{c=1}^{C} e^{z_c}}$$

After softmax, the scores can be considered as probability of $j$-th class.

The cross entropy loss is defined as,

$$L = L_{\text{CE}} + L_{reg} = \frac{1}{\ } {}_{i=1} \log(p_{i,j}) + \frac{\lambda}{2} \ ||\ \ {}_1||^2 + ||\ \ {}_2||^2$$

To take derivative of this loss, you will get the gradient as,

$$\frac{\partial L_{\text{CE}}}{\partial o_i} = p_i - y_i$$

More details about multi-class cross entropy loss, please check http://cs231n.github.io/linear-classify/ (http://cs231n.github.io/linear-classify/) and more explanation (https://deepnotes.io/softmax-crossentropy) about the derivative of cross entropy.

Change the loss from MSE to cross entropy, you only need to change you `MSE_loss(x,y)` in `TwoLayerNet.loss()` function to `softmax_loss(x,y)`.

**Now you are free to use any code to show your results of the two-layer networks with newly-implemented cross entropy loss. You can use code from previous cells.**

In [137]:
```
est_net = None # store the best model into this

 ================================================================ #
 YOUR CODE HERE:
   Optimize over your hyperparameters to arrive at the best neural
   network.  You should be able to get over 45% validation accuracy.
   For this part of the notebook, we will give credit based on the
   accuracy you get.  Your score on this question will be multiplied by:
      min(floor((X - 23%)) / %22, 1)
   where if you get 50% or higher validation accuracy, you get full
   points.

   Note, you need to use the same network structure (keep hidden_size = 50)!
 ================================================================ #

 todo: optimal parameter search (you can use grid search by for-loops )

nput_size = 32 * 32 * 3 # do not change
idden_size = 50 # do not change
um_classes = 10 # do not change
est_valacc = 0 # do not change
earning_rate=1e-3
earning_rate_decay=0.95
eg=1e-5
um_iters=8000
atch_size=500
 Train the network and find best parameter:
est_net = TwoLayerNet(input_size, hidden_size, num_classes)
est_net.train(X_train, y_train, X_val, y_val, learning_rate=learning_rate, le
rning_rate_decay=learning_rate_decay,
           reg=reg, num_iters=num_iters,
           batch_size=batch_size, verbose=False)
_val_pred = best_net.predict(X_val)
est_valacc = np.mean(y_val == y_val_pred)

 Output your results
rint("== Best parameter settings ==")
rint(f"lr: {learning_rate}, lr_decay: {learning_rate_decay}, reg: {reg}, num_
ters: {num_iters}, batch_size: {batch_size}")
 print your best parameter setting here!
rint("Best accuracy on validation set: {}".format(best_valacc))
 ================================================================ #
 END YOUR CODE HERE
 ================================================================ #
```

```
== Best parameter settings ==
lr: 0.001, lr_decay: 0.95, reg: 1e-05, num_iters: 8000, batch_size: 500
Best accuracy on validation set: 0.515
```

As you can see, with softmax_loss, our accuracy on the validation set with the same parameters is better than without MSE loss.