Assignment 2: Test Your Neural Network on a Toy Dataset

Use this notebook to build your neural network by first implementing the following functions in the python files under assignment2/algorithms,

1. linear.py
2. relu.py
3. softmax.py
4. loss_func.py

First you will be testing your 2 layer neural network implementation on a toy dataset.

In [65]:

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

from layers.sequential import Sequential
from layers.linear import Linear
from layers.relu import ReLU
from layers.softmax import Softmax
from layers.loss_func import CrossEntropyLoss
from utils.optimizer import SGD

%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-ipy
%load_ext autoreload
%autoreload 2
```

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

We will use the class Sequential as implemented in the file assignment2/layers/sequential.py to build a layer by layer model of our neural network. Below we initialize the toy model and the toy random data that you will use to develop your implementation.

In [66]:
```
# Create a small net and some toy data to check your implementations.
# Note that we set the random seed for repeatable experiments.

input_size = 4
hidden_size = 10
num_classes = 3 # Output
num_inputs = 10 # N


def init_toy_model():
    np.random.seed(0)
    l1 = Linear(input_size, hidden_size)
    l2 = Linear(hidden_size, num_classes)

    r1 = ReLU()
    r2 = ReLU()
    softmax = Softmax()
    return Sequential([l1, r1, l2, softmax])

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

net = init_toy_model()
X, y = init_toy_data()
```

Forward Pass: Compute Scores (20%)

Implement the forward functions in Linear, Relu and Softmax layers and get the output by passing our toy data X The output must match the given output scores

In [67]:
```
scores = net.forward(X)
print('Your scores:')
print(scores)
print()
print('correct scores:')
correct_scores = np.asarray([[0.33333514, 0.33333826, 0.33332661],
 [0.3333351,  0.33333828, 0.33332661],
 [0.3333351,  0.33333828, 0.33332662],
 [0.3333351,  0.33333828, 0.33332662],
 [0.33333509, 0.33333829, 0.33332662],
 [0.33333508, 0.33333829, 0.33332662],
 [0.33333511, 0.33333828, 0.33332661],
 [0.33333512, 0.33333827, 0.33332661],
 [0.33333508, 0.33333829, 0.33332662],
 [0.33333511, 0.33333828, 0.33332662]])
print(correct_scores)

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

```
Your scores:
[[0.33333514 0.33333826 0.33332661]
 [0.3333351  0.33333828 0.33332661]
 [0.3333351  0.33333828 0.33332662]
 [0.3333351  0.33333828 0.33332662]
 [0.33333509 0.33333829 0.33332662]
 [0.33333508 0.33333829 0.33332662]
 [0.33333511 0.33333828 0.33332661]
 [0.33333512 0.33333827 0.33332661]
 [0.33333508 0.33333829 0.33332662]
 [0.33333511 0.33333828 0.33332662]]

correct scores:
[[0.33333514 0.33333826 0.33332661]
 [0.3333351  0.33333828 0.33332661]
 [0.3333351  0.33333828 0.33332662]
 [0.3333351  0.33333828 0.33332662]
 [0.33333509 0.33333829 0.33332662]
 [0.33333508 0.33333829 0.33332662]
 [0.33333511 0.33333828 0.33332661]
 [0.33333512 0.33333827 0.33332661]
 [0.33333508 0.33333829 0.33332662]
 [0.33333511 0.33333828 0.33332662]]
Difference between your scores and correct scores:
8.799388495628335e-08
```

Forward Pass: Compute loss given the output scores from the previous step (10%)

Implement the forward function in the loss_func.py file, and output the loss value. The loss value must match the given loss value.

In [68]:
```
Loss = CrossEntropyLoss()
loss = Loss.forward(scores,y)
correct_loss = 1.098612723362578
print(loss)
# should be very small, we get < 1e-12
print('Difference between your loss and correct loss:')
print(np.sum(np.abs(loss - correct_loss)))
```

```
1.0986127233625778
Difference between your loss and correct loss:
2.220446049250313e-16
```

Backward Pass (40%)

Implement the rest of the functions in the given files. Specifically, implement the backward function in all the 4 files as mentioned in the files. Note: No backward function in the softmax file, the gradient for softmax is jointly calculated with the cross entropy loss in the loss_func.backward function.

You will use the chain rule to calculate gradient individually for each layer. You can assume that this calculated gradeint then is passed to the next layers in a reversed manner due to the Sequential implementation. So all you need to worry about is implementing the gradient for the current layer and multiply it will the incoming gradient (passed to the backward function as dout) to calculate the total gradient for the parameters of that layer.

In [69]:
```
# No need to edit anything in this block ( 20% of the above 40% )
net.backward(Loss.backward())

gradients = []
for module in net._modules:
    for para, grad in zip(module.parameters, module.grads):
        assert grad is not None, "No Gradient"
        #Print gradients of the linear layer
        print(grad.shape)
        gradients.append(grad)

# Check shapes of your gradient. Note that only the linear layer has paramete
#(4, 10) -> Layer 1 W
#(10,)   -> Layer 1 b
#(10, 3) -> Layer 2 W
#(3,)    -> Layer 2 b
```

```
(4, 10)
(10,)
(10, 3)
(3,)
```

In [70]:
```
# No need to edit anything in this block ( 20% of the above 40% )
# Now we check the values for these gradients. Here are the values for these
# difference, you must get difference < 1e-10
grad_w1 = np.array([[-6.24320917e-05,  3.41037180e-06, -1.69125969e-05,
         2.41514079e-05,  3.88697976e-06,  7.63842314e-05,
        -8.88925758e-05,  3.34909890e-05, -1.42758303e-05,
        -4.74748560e-06],
       [-7.16182867e-05,  4.63270039e-06, -2.20344270e-05,
        -2.72027034e-06,  6.52903437e-07,  8.97294847e-05,
        -1.05981609e-04,  4.15825391e-05, -2.12210745e-05,
         3.06061658e-05],
       [-1.69074923e-05, -8.83185056e-06,  3.10730840e-05,
         1.23010428e-05,  5.25830316e-05, -7.82980115e-06,
         3.02117990e-05, -3.37645284e-05,  6.17276346e-05,
        -1.10735656e-05],
       [-4.35902272e-05,  3.71512704e-06, -1.66837877e-05,
         2.54069557e-06, -4.33258099e-06,  5.72310022e-05,
        -6.94881762e-05,  2.92408329e-05, -1.89369767e-05,
         2.01692516e-05]])
grad_b1 = np.array([-2.27150209e-06,  5.14674340e-07, -2.04284403e-06,  6.088
                    3.92085824e-06, -5.40772636e-06,  2.93354593e-06, -3.1456

grad_w2 = np.array([[ 1.28932983e-04,  1.19946731e-04, -2.48879714e-04],
       [ 1.08784150e-04,  1.55140199e-04, -2.63924349e-04],
       [ 6.96017544e-05,  1.42748410e-04, -2.12350164e-04],
       [ 9.92512487e-05,  1.73257611e-04, -2.72508860e-04],
       [ 2.05484895e-05,  4.96161144e-05, -7.01646039e-05],
       [ 8.20539510e-05,  9.37063861e-05, -1.75760337e-04],
       [ 2.45831715e-05,  8.74369112e-05, -1.12020083e-04],
       [ 1.34073379e-04,  1.86253064e-04, -3.20326443e-04],
       [ 8.86473128e-05,  2.35554414e-04, -3.24201726e-04],
       [ 3.57433149e-05,  1.91164061e-04, -2.26907376e-04]])

grad_b2 = np.array([-0.1666649 ,  0.13333828,  0.03332662])

difference = np.sum(np.abs(gradients[0]-grad_w1)) + np.sum(np.abs(gradients[1
np.sum(np.abs(gradients[3]-grad_b2))
print("Difference in Gradient values", difference)
```

```
Difference in Gradient values 5.665200855318841e-12
```

Train the complete network on the toy data.

To train the network we will use stochastic gradient descent (SGD), we have implemented the optimizer for you. You do not implement any more functions in the python files. Below we implement the training procedure, you should get yourself familiar with the training process. Specifically looking at which functions to call and when.

Once you have implemented the method and tested various parts in the above blocks, run the code below to train a two-layer network on toy data. You should see your the training loss decrease to

In [71]:
```
# Training Procedure
# Initialize the optimizer. DO NOT change any of the hyper-parameters here or
# We have implemented the SGD optimizer class for you here, which visits each
# get the gradients and optimize the respective parameters.
# You should work with the given parameters and only edit your implementation

epochs = 1000
optim = SGD(net, lr=0.1, weight_decay=0.00001)

epoch_loss = []
for epoch in range(epochs):
            # Get output scores from the network
            output_x = net(X)
            # Calculate the loss for these output scores, given the true labe
            loss = Loss.forward(output_x, y)
            # Initialize your gradients to None in each epoch
            optim.zero_grad()
            # Make a backward pass to update the internal gradients in the la
            net.backward(Loss.backward())
            # call the step function in the optimizer to update the values of
            optim.step()
            # Append the loss at each iteration
            epoch_loss.append(loss)

            print("Epoch Loss: {:3f}".format(epoch_loss[-1]))
```

```
Epoch Loss: 0.008825
Epoch Loss: 0.008819
Epoch Loss: 0.008808
Epoch Loss: 0.008797
Epoch Loss: 0.008786
Epoch Loss: 0.008775
Epoch Loss: 0.008765
Epoch Loss: 0.008758
Epoch Loss: 0.008747
Epoch Loss: 0.008736
Epoch Loss: 0.008725
Epoch Loss: 0.008715
Epoch Loss: 0.008708
Epoch Loss: 0.008698
Epoch Loss: 0.008688
Epoch Loss: 0.008678
Epoch Loss: 0.008667
Epoch Loss: 0.008657
Epoch Loss: 0.008647
Epoch Loss: 0.008636
```

In [72]:
```
# Test your predictions. The predictions must match the labels
print(net.predict(X))
print(y)
```

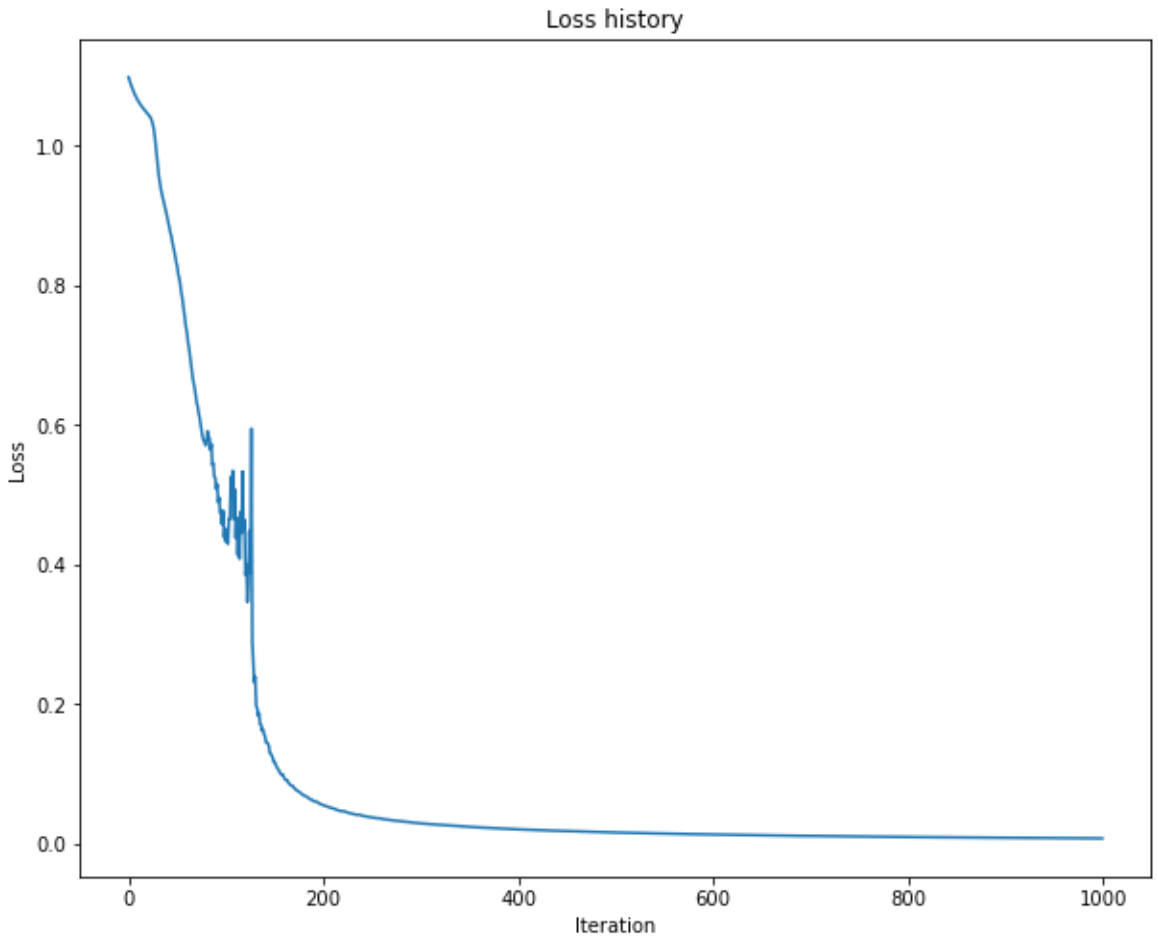
```
[2 1 0 1 2 0 0 2 0 0]
[2 1 0 1 2 0 0 2 0 0]
```

In [73]:
```
# You should be able to achieve a training loss of less than 0.02 (10%)
print("Final training loss", epoch_loss[-1])
```

```
Final training loss 0.007845036460819067
```

In [74]:
```
# Plot the training loss curve. The loss in the curve should be decreasing (2
plt.plot(epoch_loss)
plt.title('Loss history')
plt.xlabel('Iteration')
plt.ylabel('Loss')
```

Out[74]:
```
Text(0, 0.5, 'Loss')
```



In [ ]:
```

```

Assignment 2: Train and test your model on CIFAR10 dataset

Now that you have developed and tested your model on the toy dataset set. It's time to get down and get dirty with a standard dataset such as cifar10. At this point, you will be using the provided training data to tune the hyper-parameters of your network such that it works with cifar10 for the task of multi-class classification.

Important: Recall that now we have non-linear decision boundaries, thus we do not need to do one vs all classification. We learn a single non-linear decision boundary instead. Our non-linear boundaries (thanks to relu non-linearity) will take care of differentiating between all the classes

In [1]:
```
# Prepare Packages
import numpy as np
import matplotlib.pyplot as plt

from utils.data_processing import get_cifar10_data
from utils.evaluation import get_classification_accuracy


%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

# Use a subset of CIFAR10 for the assignment
dataset = get_cifar10_data(
    subset_train = 45000,
    subset_val = 1000,
    subset_test = 10000,
)

print(dataset.keys())
print("Training Set Data  Shape: ", dataset['x_train'].shape)
print("Training Set Label Shape: ", dataset['y_train'].shape)
print("Validation Set Data  Shape: ", dataset['x_val'].shape)
print("Validation Set Label Shape: ", dataset['y_val'].shape)
print("Test Set Data  Shape: ", dataset['x_test'].shape)
print("Test Set Label Shape: ", dataset['y_test'].shape)
```

```
dict_keys(['x_train', 'y_train', 'x_val', 'y_val', 'x_test', 'y_test'])
Training Set Data  Shape:  (45000, 3072)
Training Set Label Shape:  (45000,)
Validation Set Data  Shape:  (1000, 3072)
Validation Set Label Shape:  (1000,)
Test Set Data  Shape:  (10000, 3072)
Test Set Label Shape:  (10000,)
```

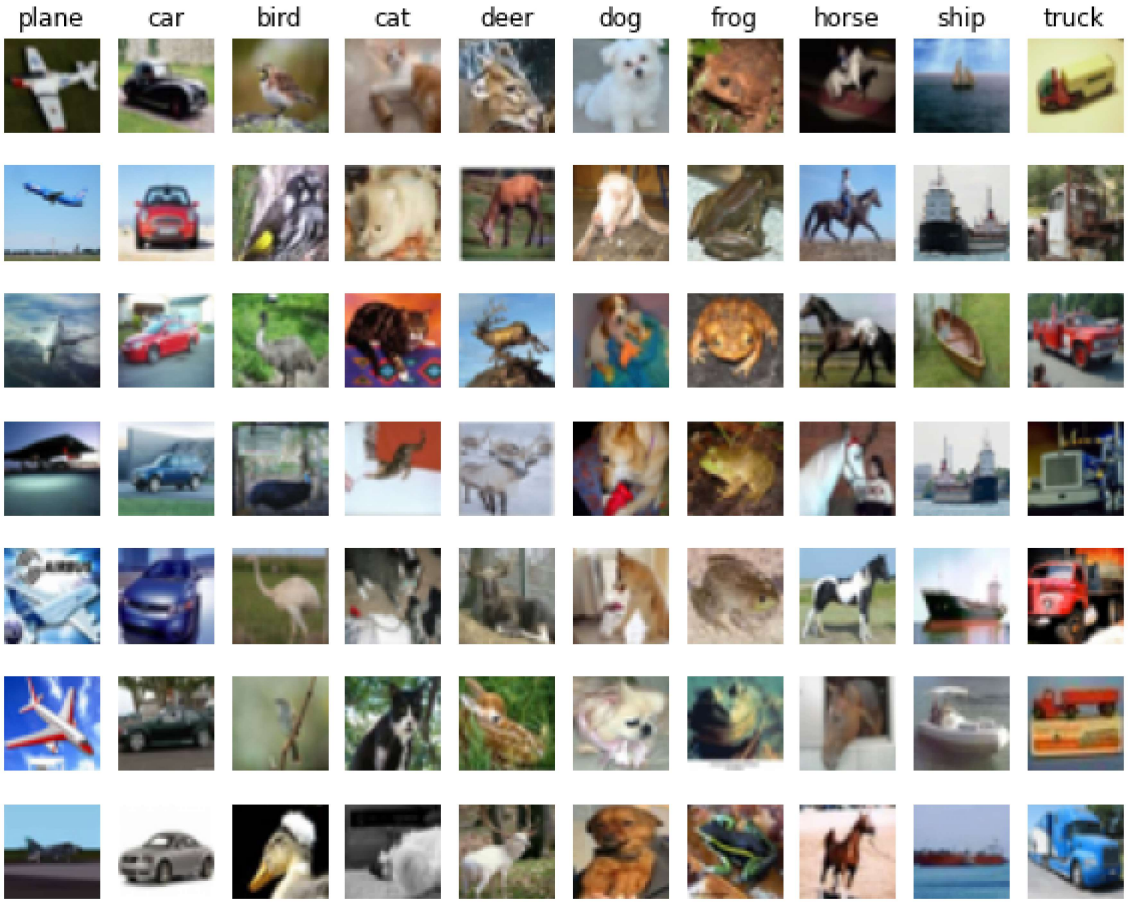
In [2]:
```
x_train = dataset['x_train']
y_train = dataset['y_train']
x_val = dataset['x_val']
y_val = dataset['y_val']
```

```
x_test = dataset['x_test']
y_test = dataset['y_test']
```

In [3]:
```
# Import more utilies and the layers you have implemented
from layers.sequential import Sequential
from layers.linear import Linear
from layers.relu import ReLU
from layers.softmax import Softmax
from layers.loss_func import CrossEntropyLoss
from utils.optimizer import SGD
from utils.dataset import DataLoader
from utils.trainer import Trainer
```

Visualize some examples from the dataset.

In [4]:
```
# We show a few examples of training images from each class.
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truc
samples_per_class = 7

def visualize_data(dataset, classes, samples_per_class):
    num_classes = len(classes)
    for y, cls in enumerate(classes):
      idxs = np.flatnonzero(y_train == y)
      idxs = np.random.choice(idxs, samples_per_class, replace=False)
      for i, idx in enumerate(idxs):
        plt_idx = i * num_classes + y + 1
        plt.subplot(samples_per_class, num_classes, plt_idx)
        plt.imshow(dataset[idx])
        plt.axis('off')
        if i == 0:
          plt.title(cls)
    plt.show()

visualize_data(x_train.reshape(45000, 3, 32, 32).transpose(0, 2, 3, 1), classes, sample
```



Initialize the model

In [5]:
```
input_size = 3072
hidden_size = 50 # Hidden layer size (Hyper-parameter)
num_classes = 10 # Output

# For a default setting we use the same model we used for the toy dataset.
# This tells you the power of a 2 layered Neural Network. Recall the Universal Approxim
# A 2 layer neural network with non-linearities can approximate any function, given lar
def init_model():
    #np.random.seed(0) # No need to fix the seed here
    l1 = Linear(input_size, hidden_size)
    l2 = Linear(hidden_size, num_classes)

    r1 = ReLU()
    softmax = Softmax()
    return Sequential([l1, r1, l2, softmax])
```

In [6]:
```
# Initialize the dataset with the dataloader class
dataset = DataLoader(x_train, y_train, x_val, y_val, x_test, y_test)
net = init_model()
optim = SGD(net, lr=0.002, weight_decay=0.00025)
loss_func = CrossEntropyLoss()
epoch = 300 # (Hyper-parameter)
batch_size = 100 # (Reduce the batch size if your computer is unable to handle it)
```

In [7]:
```
#Initialize the trainer class by passing the above modules
trainer = Trainer(dataset, optim, net, loss_func, epoch, batch_size, validate_interval=
```

In [8]:
```
# Call the trainer function we have already implemented for you. This trains the model
# hyper-parameters. It follows the same procedure as in the last ipython notebook you u
train_error, validation_accuracy = trainer.train()
```

```
Epoch Average Loss: 2.302516
Validate Acc: 0.090
Epoch Average Loss: 2.302202
Epoch Average Loss: 2.301113
Epoch Average Loss: 2.297915
Validate Acc: 0.090
Epoch Average Loss: 2.290991
Epoch Average Loss: 2.277926
Epoch Average Loss: 2.259583
Validate Acc: 0.127
Epoch Average Loss: 2.240886
Epoch Average Loss: 2.225274
Epoch Average Loss: 2.213205
Validate Acc: 0.149
Epoch Average Loss: 2.203971
Epoch Average Loss: 2.196885
Epoch Average Loss: 2.191192
Validate Acc: 0.158
Epoch Average Loss: 2.186730
Epoch Average Loss: 2.183068
Epoch Average Loss: 2.180109
Validate Acc: 0.165
Epoch Average Loss: 2.177490
Epoch Average Loss: 2.175474
Epoch Average Loss: 2.173616
Validate Acc: 0.166
Epoch Average Loss: 2.171998
Epoch Average Loss: 2.170910
Epoch Average Loss: 2.169779
Validate Acc: 0.169
Epoch Average Loss: 2.168803
Epoch Average Loss: 2.168057
Epoch Average Loss: 2.167329
Validate Acc: 0.167
Epoch Average Loss: 2.166557
Epoch Average Loss: 2.166109
Epoch Average Loss: 2.165598
Validate Acc: 0.163
Epoch Average Loss: 2.165213
Epoch Average Loss: 2.164571
Epoch Average Loss: 2.164091
Validate Acc: 0.163
Epoch Average Loss: 2.163411
Epoch Average Loss: 2.162704
Epoch Average Loss: 2.161879
Validate Acc: 0.165
Epoch Average Loss: 2.160889
Epoch Average Loss: 2.159469
Epoch Average Loss: 2.157904
Validate Acc: 0.177
Epoch Average Loss: 2.156050
Epoch Average Loss: 2.153642
Epoch Average Loss: 2.150960
Validate Acc: 0.194
Epoch Average Loss: 2.147927
Epoch Average Loss: 2.144576
Epoch Average Loss: 2.141130
Validate Acc: 0.220
Epoch Average Loss: 2.137517
Epoch Average Loss: 2.134063
Epoch Average Loss: 2.130644
```

```
Validate Acc: 0.226
Epoch Average Loss: 2.127413
Epoch Average Loss: 2.124509
Epoch Average Loss: 2.121739
Validate Acc: 0.232
Epoch Average Loss: 2.119247
Epoch Average Loss: 2.116981
Epoch Average Loss: 2.114949
Validate Acc: 0.225
Epoch Average Loss: 2.112995
Epoch Average Loss: 2.110915
Epoch Average Loss: 2.108763
Validate Acc: 0.230
Epoch Average Loss: 2.105745
Epoch Average Loss: 2.099892
Epoch Average Loss: 2.088747
Validate Acc: 0.208
Epoch Average Loss: 2.073367
Epoch Average Loss: 2.059620
Epoch Average Loss: 2.049451
Validate Acc: 0.248
Epoch Average Loss: 2.042102
Epoch Average Loss: 2.036246
Epoch Average Loss: 2.032114
Validate Acc: 0.241
Epoch Average Loss: 2.028379
Epoch Average Loss: 2.025600
Epoch Average Loss: 2.023217
Validate Acc: 0.249
Epoch Average Loss: 2.020925
Epoch Average Loss: 2.019147
Epoch Average Loss: 2.017291
Validate Acc: 0.250
Epoch Average Loss: 2.015686
Epoch Average Loss: 2.013929
Epoch Average Loss: 2.012463
Validate Acc: 0.267
Epoch Average Loss: 2.010893
Epoch Average Loss: 2.009251
Epoch Average Loss: 2.007897
Validate Acc: 0.273
Epoch Average Loss: 2.006437
Epoch Average Loss: 2.004980
Epoch Average Loss: 2.003933
Validate Acc: 0.274
Epoch Average Loss: 2.002378
Epoch Average Loss: 2.001054
Epoch Average Loss: 2.000011
Validate Acc: 0.284
Epoch Average Loss: 1.999199
Epoch Average Loss: 1.997546
Epoch Average Loss: 1.997077
Validate Acc: 0.281
Epoch Average Loss: 1.996216
Epoch Average Loss: 1.995368
Epoch Average Loss: 1.994716
Validate Acc: 0.281
Epoch Average Loss: 1.994102
Epoch Average Loss: 1.993045
Epoch Average Loss: 1.992587
Validate Acc: 0.283
Epoch Average Loss: 1.992092
Epoch Average Loss: 1.991264
Epoch Average Loss: 1.990941
Validate Acc: 0.277
```

```
Epoch Average Loss: 1.990277
Epoch Average Loss: 1.990155
Epoch Average Loss: 1.989498
Validate Acc: 0.276
Epoch Average Loss: 1.989053
Epoch Average Loss: 1.988706
Epoch Average Loss: 1.988257
Validate Acc: 0.287
Epoch Average Loss: 1.988022
Epoch Average Loss: 1.987579
Epoch Average Loss: 1.986981
Validate Acc: 0.278
Epoch Average Loss: 1.986992
Epoch Average Loss: 1.986734
Epoch Average Loss: 1.986265
Validate Acc: 0.283
Epoch Average Loss: 1.985763
Epoch Average Loss: 1.985765
Epoch Average Loss: 1.985496
Validate Acc: 0.289
Epoch Average Loss: 1.985440
Epoch Average Loss: 1.984914
Epoch Average Loss: 1.984737
Validate Acc: 0.284
Epoch Average Loss: 1.984393
Epoch Average Loss: 1.984273
Epoch Average Loss: 1.984093
Validate Acc: 0.283
Epoch Average Loss: 1.983952
Epoch Average Loss: 1.983741
Epoch Average Loss: 1.983317
Validate Acc: 0.285
Epoch Average Loss: 1.983105
Epoch Average Loss: 1.982887
Epoch Average Loss: 1.982779
Validate Acc: 0.292
Epoch Average Loss: 1.982501
Epoch Average Loss: 1.982100
Epoch Average Loss: 1.982103
Validate Acc: 0.290
Epoch Average Loss: 1.981793
Epoch Average Loss: 1.981858
Epoch Average Loss: 1.981502
Validate Acc: 0.281
Epoch Average Loss: 1.981714
Epoch Average Loss: 1.981302
Epoch Average Loss: 1.981052
Validate Acc: 0.292
Epoch Average Loss: 1.981034
Epoch Average Loss: 1.980459
Epoch Average Loss: 1.980729
Validate Acc: 0.290
Epoch Average Loss: 1.980478
Epoch Average Loss: 1.980213
Epoch Average Loss: 1.980048
Validate Acc: 0.295
Epoch Average Loss: 1.979646
Epoch Average Loss: 1.979818
Epoch Average Loss: 1.979648
Validate Acc: 0.289
Epoch Average Loss: 1.979490
Epoch Average Loss: 1.979336
Epoch Average Loss: 1.978885
Validate Acc: 0.285
Epoch Average Loss: 1.978991
```

```
Epoch Average Loss: 1.978719
Epoch Average Loss: 1.978809
Validate Acc: 0.289
Epoch Average Loss: 1.978311
Epoch Average Loss: 1.978464
Epoch Average Loss: 1.978114
Validate Acc: 0.291
Epoch Average Loss: 1.978040
Epoch Average Loss: 1.978101
Epoch Average Loss: 1.977876
Validate Acc: 0.292
Epoch Average Loss: 1.977934
Epoch Average Loss: 1.977653
Epoch Average Loss: 1.977775
Validate Acc: 0.292
Epoch Average Loss: 1.977493
Epoch Average Loss: 1.977373
Epoch Average Loss: 1.977231
Validate Acc: 0.293
Epoch Average Loss: 1.977331
Epoch Average Loss: 1.977168
Epoch Average Loss: 1.976825
Validate Acc: 0.290
Epoch Average Loss: 1.976985
Epoch Average Loss: 1.976800
Epoch Average Loss: 1.976757
Validate Acc: 0.294
Epoch Average Loss: 1.976588
Epoch Average Loss: 1.976476
Epoch Average Loss: 1.976556
Validate Acc: 0.299
Epoch Average Loss: 1.976479
Epoch Average Loss: 1.976542
Epoch Average Loss: 1.976241
Validate Acc: 0.294
Epoch Average Loss: 1.976137
Epoch Average Loss: 1.976267
Epoch Average Loss: 1.976021
Validate Acc: 0.300
Epoch Average Loss: 1.975883
Epoch Average Loss: 1.975941
Epoch Average Loss: 1.975911
Validate Acc: 0.292
Epoch Average Loss: 1.975781
Epoch Average Loss: 1.975710
Epoch Average Loss: 1.975707
Validate Acc: 0.297
Epoch Average Loss: 1.975618
Epoch Average Loss: 1.975506
Epoch Average Loss: 1.975346
Validate Acc: 0.293
Epoch Average Loss: 1.975455
Epoch Average Loss: 1.975448
Epoch Average Loss: 1.975436
Validate Acc: 0.291
Epoch Average Loss: 1.975223
Epoch Average Loss: 1.975109
Epoch Average Loss: 1.975091
Validate Acc: 0.298
Epoch Average Loss: 1.975144
Epoch Average Loss: 1.974871
Epoch Average Loss: 1.974973
Validate Acc: 0.297
Epoch Average Loss: 1.974881
Epoch Average Loss: 1.974986
```

```
Epoch Average Loss: 1.974834
Validate Acc: 0.291
Epoch Average Loss: 1.975150
Epoch Average Loss: 1.974923
Epoch Average Loss: 1.974797
Validate Acc: 0.292
Epoch Average Loss: 1.974650
Epoch Average Loss: 1.974616
Epoch Average Loss: 1.974848
Validate Acc: 0.295
Epoch Average Loss: 1.974556
Epoch Average Loss: 1.974466
Epoch Average Loss: 1.974275
Validate Acc: 0.290
Epoch Average Loss: 1.974333
Epoch Average Loss: 1.974299
Epoch Average Loss: 1.974518
Validate Acc: 0.297
Epoch Average Loss: 1.974331
Epoch Average Loss: 1.974413
Epoch Average Loss: 1.974176
Validate Acc: 0.295
Epoch Average Loss: 1.974108
Epoch Average Loss: 1.974335
Epoch Average Loss: 1.973932
Validate Acc: 0.289
Epoch Average Loss: 1.974008
Epoch Average Loss: 1.974124
Epoch Average Loss: 1.973978
Validate Acc: 0.294
Epoch Average Loss: 1.974023
Epoch Average Loss: 1.973891
Epoch Average Loss: 1.973815
Validate Acc: 0.292
Epoch Average Loss: 1.974074
Epoch Average Loss: 1.973775
Epoch Average Loss: 1.973617
Validate Acc: 0.294
Epoch Average Loss: 1.973553
Epoch Average Loss: 1.973463
Epoch Average Loss: 1.973590
Validate Acc: 0.292
Epoch Average Loss: 1.973279
Epoch Average Loss: 1.973719
Epoch Average Loss: 1.973645
Validate Acc: 0.294
Epoch Average Loss: 1.973282
Epoch Average Loss: 1.973418
Epoch Average Loss: 1.973407
Validate Acc: 0.292
Epoch Average Loss: 1.973107
Epoch Average Loss: 1.973291
Epoch Average Loss: 1.973255
Validate Acc: 0.295
Epoch Average Loss: 1.973200
Epoch Average Loss: 1.973248
Epoch Average Loss: 1.973392
Validate Acc: 0.287
Epoch Average Loss: 1.973056
Epoch Average Loss: 1.973167
Epoch Average Loss: 1.972623
Validate Acc: 0.289
Epoch Average Loss: 1.973039
Epoch Average Loss: 1.973138
Epoch Average Loss: 1.972953
```

```
Validate Acc: 0.293
Epoch Average Loss: 1.972811
Epoch Average Loss: 1.972845
Epoch Average Loss: 1.972635
Validate Acc: 0.298
Epoch Average Loss: 1.972745
Epoch Average Loss: 1.972780
Epoch Average Loss: 1.972543
Validate Acc: 0.291
Epoch Average Loss: 1.972371
Epoch Average Loss: 1.972634
Epoch Average Loss: 1.972544
Validate Acc: 0.293
Epoch Average Loss: 1.972533
Epoch Average Loss: 1.972418
Epoch Average Loss: 1.972067
Validate Acc: 0.288
Epoch Average Loss: 1.972675
Epoch Average Loss: 1.972419
Epoch Average Loss: 1.972212
Validate Acc: 0.295
Epoch Average Loss: 1.972405
Epoch Average Loss: 1.972135
Epoch Average Loss: 1.972309
Validate Acc: 0.292
Epoch Average Loss: 1.972150
Epoch Average Loss: 1.972098
Epoch Average Loss: 1.972139
Validate Acc: 0.286
Epoch Average Loss: 1.972155
Epoch Average Loss: 1.972090
Epoch Average Loss: 1.972018
Validate Acc: 0.292
Epoch Average Loss: 1.971951
Epoch Average Loss: 1.972149
Epoch Average Loss: 1.971988
Validate Acc: 0.296
Epoch Average Loss: 1.971929
Epoch Average Loss: 1.971999
Epoch Average Loss: 1.971938
Validate Acc: 0.293
Epoch Average Loss: 1.971797
Epoch Average Loss: 1.971836
Epoch Average Loss: 1.971799
Validate Acc: 0.292
Epoch Average Loss: 1.971718
Epoch Average Loss: 1.971439
Epoch Average Loss: 1.971948
Validate Acc: 0.289
Epoch Average Loss: 1.971414
Epoch Average Loss: 1.971632
Epoch Average Loss: 1.971705
Validate Acc: 0.301
Epoch Average Loss: 1.971485
Epoch Average Loss: 1.971376
Epoch Average Loss: 1.971467
Validate Acc: 0.289
Epoch Average Loss: 1.971522
Epoch Average Loss: 1.971554
Epoch Average Loss: 1.971376
Validate Acc: 0.295
Epoch Average Loss: 1.971323
Epoch Average Loss: 1.971473
Epoch Average Loss: 1.971461
Validate Acc: 0.293
```

```
Epoch Average Loss: 1.971204
Epoch Average Loss: 1.971091
Epoch Average Loss: 1.971100
Validate Acc: 0.297
Epoch Average Loss: 1.971254
Epoch Average Loss: 1.971071
Epoch Average Loss: 1.971188
Validate Acc: 0.290
Epoch Average Loss: 1.970850
Epoch Average Loss: 1.971307
Epoch Average Loss: 1.971074
Validate Acc: 0.296
Epoch Average Loss: 1.970825
Epoch Average Loss: 1.970910
```

Print the training and validation accuracies for the default hyper-parameters provided

In [9]:
```
from utils.evaluation import get_classification_accuracy
out_train = net.predict(x_train)
acc = get_classification_accuracy(out_train, y_train)
print("Training acc: ",acc)
out_val = net.predict(x_val)
acc = get_classification_accuracy(out_val, y_val)
print("Validation acc: ",acc)
```

```
Training acc:  0.31024444444444443
Validation acc:  0.295
```

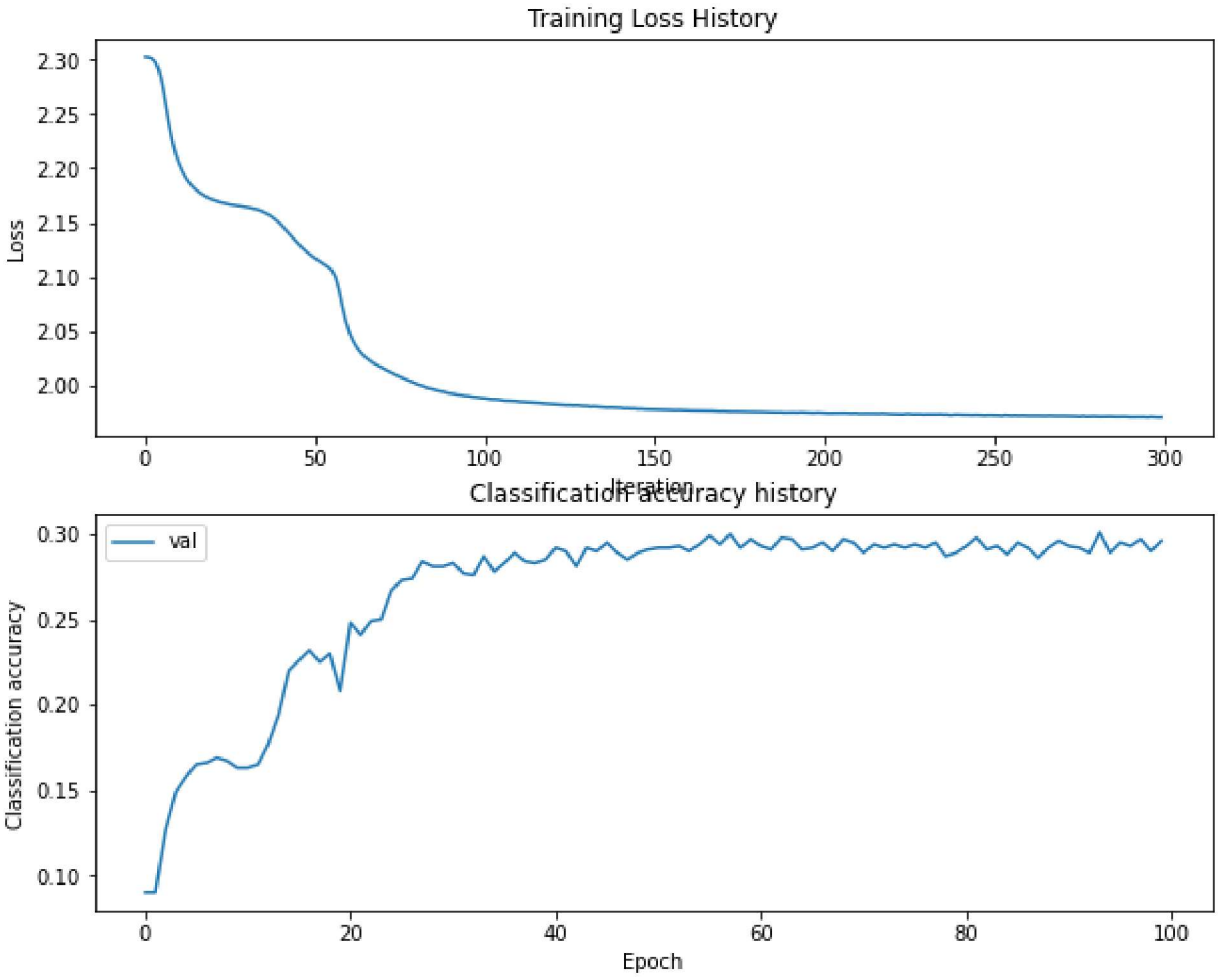
Debug the training

With the default parameters we provided above, you should get a validation accuracy of about 0.29 on the validation set. This isn't very good.

One strategy for getting insight into what's wrong is to plot the training loss function and the validation accuracies during optimization.

Another strategy is to visualize the weights that were learned in the first layer of the network. In most neural networks trained on visual data, the first layer weights typically show some visible structure when visualized.

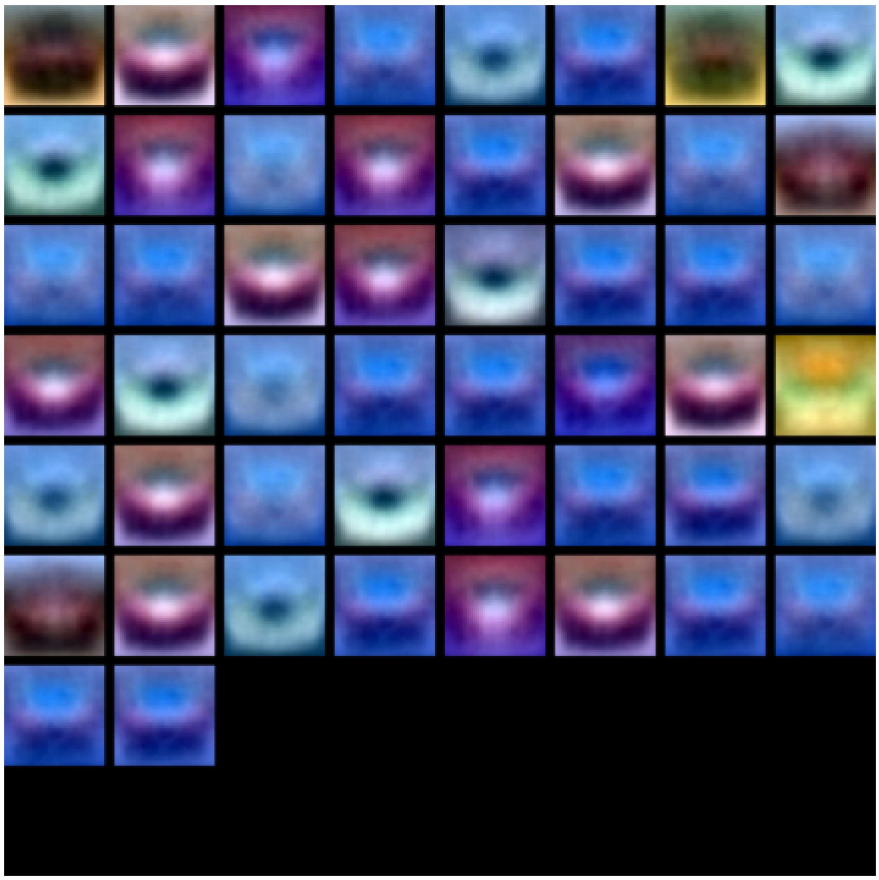
In [10]:
```
# Plot the training loss function and validation accuracies
plt.subplot(2, 1, 1)
plt.plot(train_error)
plt.title('Training Loss History')
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
#plt.plot(stats['train_acc_history'], label='train')
plt.plot(validation_accuracy, label='val')
plt.title('Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()
plt.show()
```



In [11]:
```
pip install past
```

```
Note: you may need to restart the kernel to use updated packages.
ERROR: Could not find a version that satisfies the requirement past (from versions: non
e)
ERROR: No matching distribution found for past
```

In [12]:
```
from utils.vis_utils import visualize_grid
# Credits: http://cs231n.stanford.edu/

# Visualize the weights of the network
def show_net_weights(net):
    W1 = net._modules[0].parameters[0]
    W1 = W1.reshape(3, 32, 32, -1).transpose(3, 1, 2, 0)
    plt.imshow(visualize_grid(W1, padding=3).astype('uint8'))
    plt.gca().axis('off')
    plt.show()

show_net_weights(net)
```



Tune your hyperparameters (50%)

What's wrong?. Looking at the visualizations above, we see that the loss is decreasing more or less linearly, which seems to suggest that the learning rate may be too low. Moreover, there is no gap between the training and validation accuracy, suggesting that the model we used has low capacity, and that we should increase its size. On the other hand, with a very large model we would expect to see more overfitting, which would manifest itself as a very large gap between the training and validation accuracy.

Tuning. Tuning the hyperparameters and developing intuition for how they affect the final performance is a large part of using Neural Networks, so we want you to get a lot of practice. Below, you should experiment with different values of the various hyperparameters, including hidden layer size, learning rate, numer of training epochs, and regularization strength.

Approximate results. You should be aim to achieve a classification accuracy of greater than 48% on the validation set. Our best network gets over 52% on the validation set.

Experiment: You goal in this exercise is to get as good of a result on CIFAR-10 as you can (52% could serve as a reference), with a fully-connected Neural Network.

Explain your hyperparameter tuning process below.

$YourAnswer:$

I upped the learning rate to 0.01 and got an accuracy below 52% so I upped to 0.025 and upped the weight decay to 0.001 but the accuracy was still low so I put it to 0.0001 and with the 0.025 learning rate and got the accuracy I wanted

In [13]:
```
best_net_hyperparams = [0.025, 0.0001, 300, 100] # store the best model into this

################################################################################
# TODO: Tune hyperparameters using the validation set. Store your best trained  #
# model hyperparams in best_net.                                                #
#                                                                               #
# To help debug your network, it may help to use visualizations similar to the  #
# ones we used above; these visualizations will have significant qualitative    #
# differences from the ones we saw above for the poorly tuned network.          #
#                                                                               #
# You are now free to test different combinations of hyperparameters to build   #
# various models and test them according to the above plots and visualization   #


# TODO: Show the above plots and visualizations for the default params (already #
# done) and the best hyper-params you obtain. You only need to show this for 2  #
# sets of hyper-params.                                                         #
# You just need to store values for the hyperparameters in best_net_hyperparams #
# as a list in the order
# best_net_hyperparams = [lr, weight_decay, epoch, hidden_size]
################################################################################

Lr = best_net_hyperparams[0]
Weightdecay = best_net_hyperparams[1]
epochs = best_net_hyperparams[2]
batch_size = best_net_hyperparams[3]

nu_net = init_model()

optim = SGD(nu_net, lr=Lr, weight_decay=Weightdecay)
loss_func = CrossEntropyLoss()

trainer = Trainer(dataset, optim, nu_net, loss_func, epochs, batch_size, validate_inter
train_error, validation_accuracy = trainer.train()
```

```
Epoch Average Loss: 2.241096
Validate Acc: 0.169
Epoch Average Loss: 2.125412
Epoch Average Loss: 2.036844
Epoch Average Loss: 1.955349
Validate Acc: 0.287
Epoch Average Loss: 1.895586
Epoch Average Loss: 1.848127
Epoch Average Loss: 1.806558
Validate Acc: 0.329
Epoch Average Loss: 1.766173
Epoch Average Loss: 1.731965
Epoch Average Loss: 1.705697
Validate Acc: 0.396
Epoch Average Loss: 1.680954
Epoch Average Loss: 1.666421
Epoch Average Loss: 1.647634
Validate Acc: 0.416
Epoch Average Loss: 1.631943
Epoch Average Loss: 1.618325
Epoch Average Loss: 1.604315
Validate Acc: 0.408
```

```
Epoch Average Loss: 1.595931
Epoch Average Loss: 1.584065
Epoch Average Loss: 1.576126
Validate Acc: 0.443
Epoch Average Loss: 1.562295
Epoch Average Loss: 1.551990
Epoch Average Loss: 1.544662
Validate Acc: 0.426
Epoch Average Loss: 1.539534
Epoch Average Loss: 1.530122
Epoch Average Loss: 1.526384
Validate Acc: 0.454
Epoch Average Loss: 1.511151
Epoch Average Loss: 1.513174
Epoch Average Loss: 1.504139
Validate Acc: 0.436
Epoch Average Loss: 1.497620
Epoch Average Loss: 1.494806
Epoch Average Loss: 1.493054
Validate Acc: 0.472
Epoch Average Loss: 1.484712
Epoch Average Loss: 1.482217
Epoch Average Loss: 1.476422
Validate Acc: 0.454
Epoch Average Loss: 1.473267
Epoch Average Loss: 1.470186
Epoch Average Loss: 1.468078
Validate Acc: 0.453
Epoch Average Loss: 1.467035
Epoch Average Loss: 1.463855
Epoch Average Loss: 1.459286
Validate Acc: 0.462
Epoch Average Loss: 1.457692
Epoch Average Loss: 1.454057
Epoch Average Loss: 1.452056
Validate Acc: 0.484
Epoch Average Loss: 1.450131
Epoch Average Loss: 1.447299
Epoch Average Loss: 1.444539
Validate Acc: 0.463
Epoch Average Loss: 1.445107
Epoch Average Loss: 1.438764
Epoch Average Loss: 1.438547
Validate Acc: 0.477
Epoch Average Loss: 1.437946
Epoch Average Loss: 1.437197
Epoch Average Loss: 1.433660
Validate Acc: 0.482
Epoch Average Loss: 1.429819
Epoch Average Loss: 1.433175
Epoch Average Loss: 1.428987
Validate Acc: 0.452
Epoch Average Loss: 1.429116
Epoch Average Loss: 1.424877
Epoch Average Loss: 1.423645
Validate Acc: 0.487
Epoch Average Loss: 1.424995
Epoch Average Loss: 1.419581
Epoch Average Loss: 1.420101
Validate Acc: 0.470
Epoch Average Loss: 1.419877
Epoch Average Loss: 1.415901
Epoch Average Loss: 1.415233
Validate Acc: 0.487
Epoch Average Loss: 1.409624
```

```
Epoch Average Loss: 1.415624
Epoch Average Loss: 1.409490
Validate Acc: 0.486
Epoch Average Loss: 1.407335
Epoch Average Loss: 1.411262
Epoch Average Loss: 1.407392
Validate Acc: 0.475
Epoch Average Loss: 1.407123
Epoch Average Loss: 1.402580
Epoch Average Loss: 1.405636
Validate Acc: 0.474
Epoch Average Loss: 1.399450
Epoch Average Loss: 1.398835
Epoch Average Loss: 1.399392
Validate Acc: 0.467
Epoch Average Loss: 1.398352
Epoch Average Loss: 1.402007
Epoch Average Loss: 1.397436
Validate Acc: 0.475
Epoch Average Loss: 1.397347
Epoch Average Loss: 1.392460
Epoch Average Loss: 1.399742
Validate Acc: 0.475
Epoch Average Loss: 1.389381
Epoch Average Loss: 1.391759
Epoch Average Loss: 1.395754
Validate Acc: 0.499
Epoch Average Loss: 1.390833
Epoch Average Loss: 1.394161
Epoch Average Loss: 1.390992
Validate Acc: 0.514
Epoch Average Loss: 1.381151
Epoch Average Loss: 1.390230
Epoch Average Loss: 1.384808
Validate Acc: 0.470
Epoch Average Loss: 1.390014
Epoch Average Loss: 1.387721
Epoch Average Loss: 1.383815
Validate Acc: 0.473
Epoch Average Loss: 1.379634
Epoch Average Loss: 1.382199
Epoch Average Loss: 1.379247
Validate Acc: 0.489
Epoch Average Loss: 1.382114
Epoch Average Loss: 1.378393
Epoch Average Loss: 1.378804
Validate Acc: 0.500
Epoch Average Loss: 1.385585
Epoch Average Loss: 1.376533
Epoch Average Loss: 1.377670
Validate Acc: 0.507
Epoch Average Loss: 1.376171
Epoch Average Loss: 1.377519
Epoch Average Loss: 1.375801
Validate Acc: 0.508
Epoch Average Loss: 1.371977
Epoch Average Loss: 1.374274
Epoch Average Loss: 1.377641
Validate Acc: 0.498
Epoch Average Loss: 1.376767
Epoch Average Loss: 1.370655
Epoch Average Loss: 1.373555
Validate Acc: 0.503
Epoch Average Loss: 1.368691
Epoch Average Loss: 1.370300
```

```
Epoch Average Loss: 1.374292
Validate Acc: 0.499
Epoch Average Loss: 1.372288
Epoch Average Loss: 1.371779
Epoch Average Loss: 1.374580
Validate Acc: 0.456
Epoch Average Loss: 1.370122
Epoch Average Loss: 1.369882
Epoch Average Loss: 1.374022
Validate Acc: 0.475
Epoch Average Loss: 1.370188
Epoch Average Loss: 1.368587
Epoch Average Loss: 1.372111
Validate Acc: 0.512
Epoch Average Loss: 1.369422
Epoch Average Loss: 1.366030
Epoch Average Loss: 1.368338
Validate Acc: 0.505
Epoch Average Loss: 1.365068
Epoch Average Loss: 1.367221
Epoch Average Loss: 1.368972
Validate Acc: 0.462
Epoch Average Loss: 1.363119
Epoch Average Loss: 1.363624
Epoch Average Loss: 1.367097
Validate Acc: 0.521
Epoch Average Loss: 1.369830
Epoch Average Loss: 1.366397
Epoch Average Loss: 1.363651
Validate Acc: 0.497
Epoch Average Loss: 1.364659
Epoch Average Loss: 1.365223
Epoch Average Loss: 1.362226
Validate Acc: 0.503
Epoch Average Loss: 1.362969
Epoch Average Loss: 1.369488
Epoch Average Loss: 1.365423
Validate Acc: 0.491
Epoch Average Loss: 1.360710
Epoch Average Loss: 1.358946
Epoch Average Loss: 1.364710
Validate Acc: 0.483
Epoch Average Loss: 1.361687
Epoch Average Loss: 1.362754
Epoch Average Loss: 1.363598
Validate Acc: 0.499
Epoch Average Loss: 1.362949
Epoch Average Loss: 1.358705
Epoch Average Loss: 1.360099
Validate Acc: 0.516
Epoch Average Loss: 1.360193
Epoch Average Loss: 1.361180
Epoch Average Loss: 1.363828
Validate Acc: 0.491
Epoch Average Loss: 1.360838
Epoch Average Loss: 1.363654
Epoch Average Loss: 1.356647
Validate Acc: 0.506
Epoch Average Loss: 1.361900
Epoch Average Loss: 1.359774
Epoch Average Loss: 1.362647
Validate Acc: 0.496
Epoch Average Loss: 1.358857
Epoch Average Loss: 1.356474
Epoch Average Loss: 1.357523
```

```
Validate Acc: 0.500
Epoch Average Loss: 1.361581
Epoch Average Loss: 1.358316
Epoch Average Loss: 1.360654
Validate Acc: 0.476
Epoch Average Loss: 1.356717
Epoch Average Loss: 1.358212
Epoch Average Loss: 1.355439
Validate Acc: 0.502
Epoch Average Loss: 1.359548
Epoch Average Loss: 1.359769
Epoch Average Loss: 1.357440
Validate Acc: 0.507
Epoch Average Loss: 1.358119
Epoch Average Loss: 1.355149
Epoch Average Loss: 1.358511
Validate Acc: 0.495
Epoch Average Loss: 1.357007
Epoch Average Loss: 1.357059
Epoch Average Loss: 1.355231
Validate Acc: 0.512
Epoch Average Loss: 1.357311
Epoch Average Loss: 1.358459
Epoch Average Loss: 1.357125
Validate Acc: 0.494
Epoch Average Loss: 1.354064
Epoch Average Loss: 1.357340
Epoch Average Loss: 1.353602
Validate Acc: 0.495
Epoch Average Loss: 1.356047
Epoch Average Loss: 1.352962
Epoch Average Loss: 1.353475
Validate Acc: 0.460
Epoch Average Loss: 1.354139
Epoch Average Loss: 1.352252
Epoch Average Loss: 1.353474
Validate Acc: 0.498
Epoch Average Loss: 1.356667
Epoch Average Loss: 1.355219
Epoch Average Loss: 1.354171
Validate Acc: 0.475
Epoch Average Loss: 1.354713
Epoch Average Loss: 1.353511
Epoch Average Loss: 1.352644
Validate Acc: 0.491
Epoch Average Loss: 1.355139
Epoch Average Loss: 1.354011
Epoch Average Loss: 1.355600
Validate Acc: 0.486
Epoch Average Loss: 1.353374
Epoch Average Loss: 1.352596
Epoch Average Loss: 1.351523
Validate Acc: 0.503
Epoch Average Loss: 1.354047
Epoch Average Loss: 1.351482
Epoch Average Loss: 1.352086
Validate Acc: 0.504
Epoch Average Loss: 1.351442
Epoch Average Loss: 1.349260
Epoch Average Loss: 1.351557
Validate Acc: 0.488
Epoch Average Loss: 1.351664
Epoch Average Loss: 1.347298
Epoch Average Loss: 1.352837
Validate Acc: 0.476
```

```
Epoch Average Loss: 1.352573
Epoch Average Loss: 1.348850
Epoch Average Loss: 1.350201
Validate Acc: 0.507
Epoch Average Loss: 1.349109
Epoch Average Loss: 1.355248
Epoch Average Loss: 1.353443
Validate Acc: 0.477
Epoch Average Loss: 1.351224
Epoch Average Loss: 1.349944
Epoch Average Loss: 1.345762
Validate Acc: 0.510
Epoch Average Loss: 1.352956
Epoch Average Loss: 1.356812
Epoch Average Loss: 1.349443
Validate Acc: 0.531
Epoch Average Loss: 1.349989
Epoch Average Loss: 1.353487
Epoch Average Loss: 1.348763
Validate Acc: 0.496
Epoch Average Loss: 1.349357
Epoch Average Loss: 1.352284
Epoch Average Loss: 1.349153
Validate Acc: 0.488
Epoch Average Loss: 1.346898
Epoch Average Loss: 1.351128
Epoch Average Loss: 1.349654
Validate Acc: 0.490
Epoch Average Loss: 1.351600
Epoch Average Loss: 1.348804
Epoch Average Loss: 1.350393
Validate Acc: 0.471
Epoch Average Loss: 1.347717
Epoch Average Loss: 1.350751
Epoch Average Loss: 1.349475
Validate Acc: 0.505
Epoch Average Loss: 1.353384
Epoch Average Loss: 1.351796
Epoch Average Loss: 1.350365
Validate Acc: 0.490
Epoch Average Loss: 1.349336
Epoch Average Loss: 1.351403
Epoch Average Loss: 1.348818
Validate Acc: 0.516
Epoch Average Loss: 1.348049
Epoch Average Loss: 1.352009
Epoch Average Loss: 1.347701
Validate Acc: 0.504
Epoch Average Loss: 1.350013
Epoch Average Loss: 1.344317
Epoch Average Loss: 1.348380
Validate Acc: 0.511
Epoch Average Loss: 1.349290
Epoch Average Loss: 1.345754
Epoch Average Loss: 1.348424
Validate Acc: 0.504
Epoch Average Loss: 1.345815
Epoch Average Loss: 1.352811
Epoch Average Loss: 1.352443
Validate Acc: 0.488
Epoch Average Loss: 1.349481
Epoch Average Loss: 1.345980
Epoch Average Loss: 1.343192
Validate Acc: 0.485
Epoch Average Loss: 1.346689
```

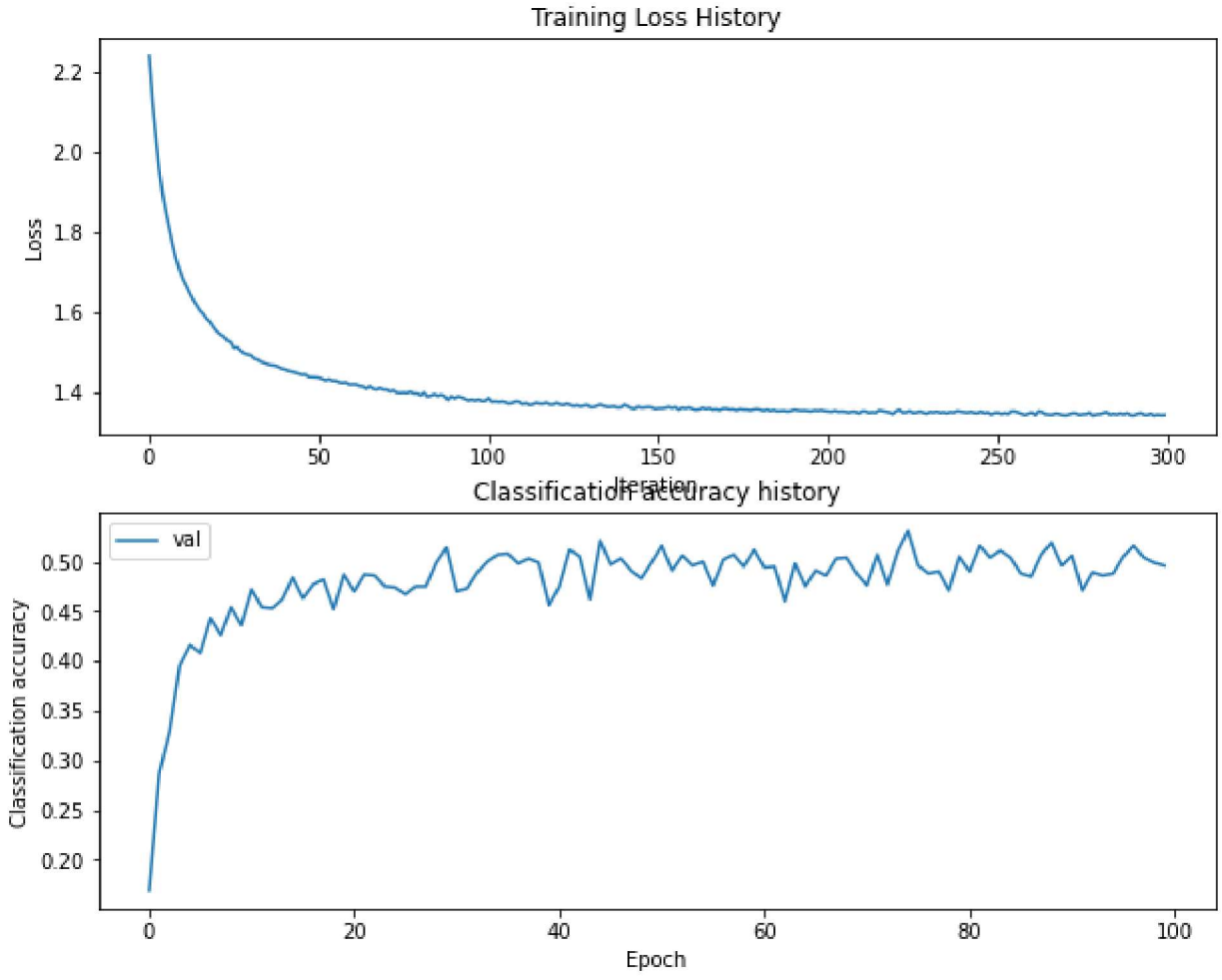
```
Epoch Average Loss: 1.349507
Epoch Average Loss: 1.343067
Validate Acc: 0.507
Epoch Average Loss: 1.352967
Epoch Average Loss: 1.350717
Epoch Average Loss: 1.345463
Validate Acc: 0.519
Epoch Average Loss: 1.344334
Epoch Average Loss: 1.344041
Epoch Average Loss: 1.347282
Validate Acc: 0.496
Epoch Average Loss: 1.345986
Epoch Average Loss: 1.343411
Epoch Average Loss: 1.342492
Validate Acc: 0.506
Epoch Average Loss: 1.345104
Epoch Average Loss: 1.347813
Epoch Average Loss: 1.347679
Validate Acc: 0.471
Epoch Average Loss: 1.343445
Epoch Average Loss: 1.345334
Epoch Average Loss: 1.345379
Validate Acc: 0.489
Epoch Average Loss: 1.349347
Epoch Average Loss: 1.341943
Epoch Average Loss: 1.345120
Validate Acc: 0.486
Epoch Average Loss: 1.345382
Epoch Average Loss: 1.347938
Epoch Average Loss: 1.350910
Validate Acc: 0.488
Epoch Average Loss: 1.344121
Epoch Average Loss: 1.346478
Epoch Average Loss: 1.344316
Validate Acc: 0.505
Epoch Average Loss: 1.347673
Epoch Average Loss: 1.344596
Epoch Average Loss: 1.348383
Validate Acc: 0.516
Epoch Average Loss: 1.343126
Epoch Average Loss: 1.342402
Epoch Average Loss: 1.345841
Validate Acc: 0.504
Epoch Average Loss: 1.348359
Epoch Average Loss: 1.343954
Epoch Average Loss: 1.343444
Validate Acc: 0.499
Epoch Average Loss: 1.346912
Epoch Average Loss: 1.342198
Epoch Average Loss: 1.344352
Validate Acc: 0.496
Epoch Average Loss: 1.343037
Epoch Average Loss: 1.343527
```

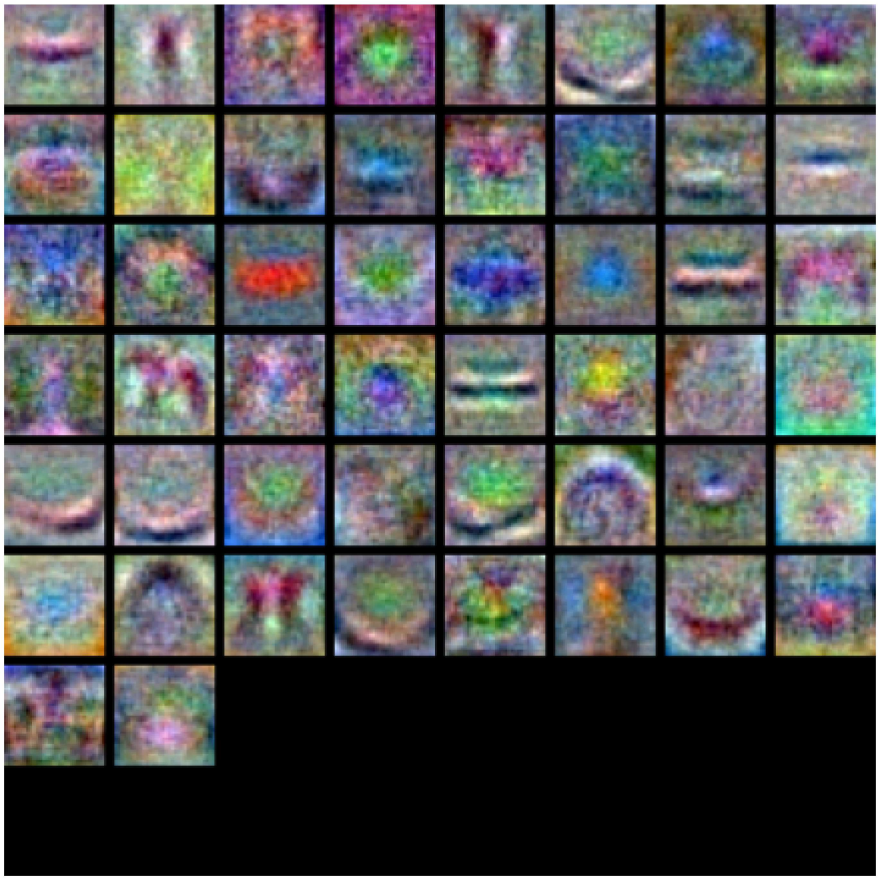
In [14]:

```python
# TODO: Plot the training_error and validation_accuracy of the best network (5%)
out_train = nu_net.predict(x_train)
acc = get_classification_accuracy(out_train, y_train)
print("Training acc: ",acc)
out_val = nu_net.predict(x_val)
acc = get_classification_accuracy(out_val, y_val)
print("Validation acc: ",acc)

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

```
plt.plot(train_error)
plt.title('Training Loss History')
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
#plt.plot(stats['train_acc_history'], label='train')
plt.plot(validation_accuracy, label='val')
plt.title('Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Classification accuracy')
plt.legend()
plt.show()

# TODO: visualize the weights of the best network (5%)

show_net_weights(nu_net)
```

```
Training acc:  0.48575555555555555
Validation acc:  0.453
```





Run on the test set (30%)

When you are done experimenting, you should evaluate your final trained network on the test set; you should get above 48%.

In [17]:
```
test_acc = (nu_net.predict(x_test) == y_test).mean()
print('Test accuracy: ', test_acc)
```

```
Test accuracy:  0.4483
```

Inline Question (10%)

Now that you have trained a Neural Network classifier, you may find that your testing accuracy is much lower than the training accuracy. In what ways can we decrease this gap? Select all that apply.

1. Train on a larger dataset.
2. Add more hidden units.
3. Increase the regularization strength.
4. None of the above.

$Your Answer$: My answer is all 3 of the above would decrease the gap.

$Your Explanation$: Training on a larger dataset, adding more hidden units, and increasing the regularization strength will decrease the gap. Training on a larger dataset will decrease the gap because the model will become more accustomed to a larger variety of data, adding more hidden

units as the model will be able to train to more general data. Increasing regularization strength will reduce overfitting from the model training on training data.

In [ ]: