This is the 2-layer neural network notebook for ECE C147/C247 Homework #3

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

Please print out the notebook entirely when completed.

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

In [24]:
```
import random
import numpy as np
from utils.data_utils import load_CIFAR10
import matplotlib.pyplot as plt

%matplotlib inline
%load_ext autoreload
%autoreload 2

def 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. Make sure to read the description of TwoLayerNet class in neural_net.py file , understand the architecture and initializations

In [25]:
```
from nndl.neural_net import TwoLayerNet
```

In [26]:
```
# 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
num_inputs = 5

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

def 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

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

Compute forward pass scores

In [27]:
```
## Implement the forward pass of the neural network.
## See the loss() method in TwoLayerNet class for the same

# Note, there is a statement if y is None: return scores, which is why
# the following call will calculate the scores.
scores = net.loss(X)
print('Your scores:')
print(scores)
print()
print('correct scores:')
correct_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]])
print(correct_scores)
print()

# 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:
[[-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]]

correct 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]]

Difference between your scores and correct scores:
3.381231233889892e-08
```

Forward pass loss

In [28]:
```
loss, _ = net.loss(X, y, reg=0.05)
correct_loss = 1.071696123862817

# should be very small, we get < 1e-12
print("Loss:",loss)
print('Difference between your loss and correct loss:')
print(np.sum(np.abs(loss - correct_loss)))
```

```
Loss: 1.071696123862817
Difference between your loss and correct loss:
0.0
```

Backward pass

Implements the backwards pass of the neural network. Check your gradients with the gradient check utilities provided.

In [29]:

```
from utils.gradient_check import eval_numerical_gradient

# Use numeric gradient checking to check your implementation of the backward pass.
# 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.

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

# these should all be less than 1e-8 or so
for 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], verbose=False)
    print('{} max relative error: {}'.format(param_name, rel_error(param_grad_num, grads[param_name])))
```

```
W2 max relative error: 2.9632227682005116e-10
b2 max relative error: 1.248270530283678e-09
W1 max relative error: 1.2832823337649917e-09
b1 max relative error: 3.172680092703762e-09
```

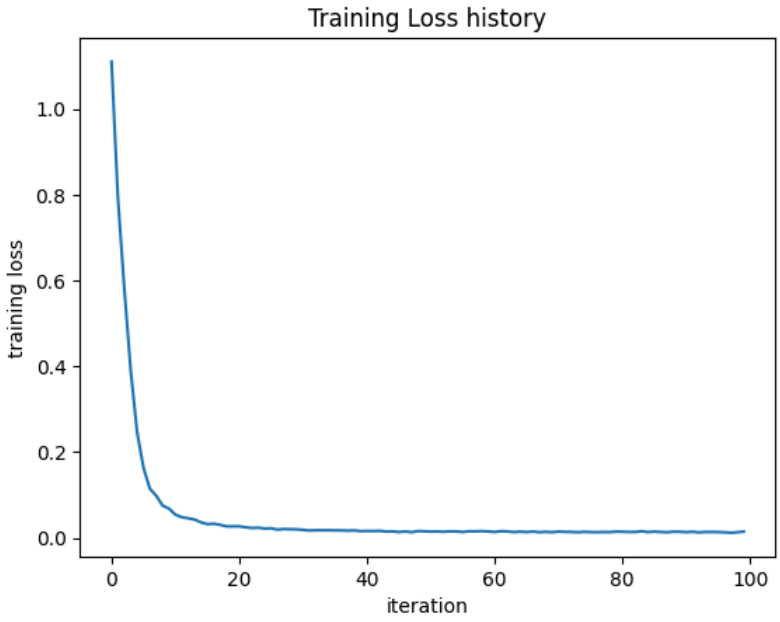
Training the network

Implement neural_net.train() to train the network via stochastic gradient descent, much like the softmax and SVM.

In [30]:
```
net = init_toy_model()
stats = net.train(X, y, X, y,
            learning_rate=1e-1, reg=5e-6,
            num_iters=100, verbose=False)

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

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

```
Final training loss:  0.014497864587765886
```

```
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "orientation" which is no longer supported as of 3.3 and will becom
e an error two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "dpi" which is no longer supported as of 3.3 and will become an err
or two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "facecolor" which is no longer supported as of 3.3 and will become
an error two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "edgecolor" which is no longer supported as of 3.3 and will become
an error two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "bbox_inches_restore" which is no longer supported as of 3.3 and wi
ll become an error two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
```



Classify CIFAR-10

Do classification on the CIFAR-10 dataset.

In [32]:

```python
from utils.data_utils import load_CIFAR10

def 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.
    """
    # Load the raw CIFAR-10 data
    cifar10_dir = '/Users/sujitsilas/Desktop/UCLA/Winter 2025/EE ENGR 247/Homeworks/HW2/student_copy/cif
    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.
X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
print('Train data shape: ', X_train.shape)
print('Train labels shape: ', y_train.shape)
print('Validation data shape: ', X_val.shape)
print('Validation labels shape: ', y_val.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

```
Train data shape:  (49000, 3072)
Train labels shape:  (49000,)
Validation data shape:  (1000, 3072)
Validation labels shape:  (1000,)
Test data shape:  (1000, 3072)
Test labels shape:  (1000,)
```

Running SGD

If your implementation is correct, you should see a validation accuracy of around 28-29%.

In [ ]:
```
input_size = 32 * 32 * 3
hidden_size = 50
num_classes = 10
net = TwoLayerNet(input_size, hidden_size, num_classes)

# Train the network
stats = net.train(X_train, y_train, X_val, y_val,
            num_iters=1000, batch_size=200,
            learning_rate=1e-4, learning_rate_decay=0.95,
            reg=0.25, verbose=True)

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

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

```
iteration 0 / 1000: loss 2.302757518613176
iteration 100 / 1000: loss 2.302120159207236
iteration 200 / 1000: loss 2.2956136007408703
iteration 300 / 1000: loss 2.2518259043164135
iteration 400 / 1000: loss 2.188995235046776
iteration 500 / 1000: loss 2.1162527791897747
iteration 600 / 1000: loss 2.064670827698217
iteration 700 / 1000: loss 1.990168862308394
iteration 800 / 1000: loss 2.002827640124685
iteration 900 / 1000: loss 1.94651768178565
Validation accuracy:  0.283
```

Questions:

The training accuracy isn't great.

(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)?

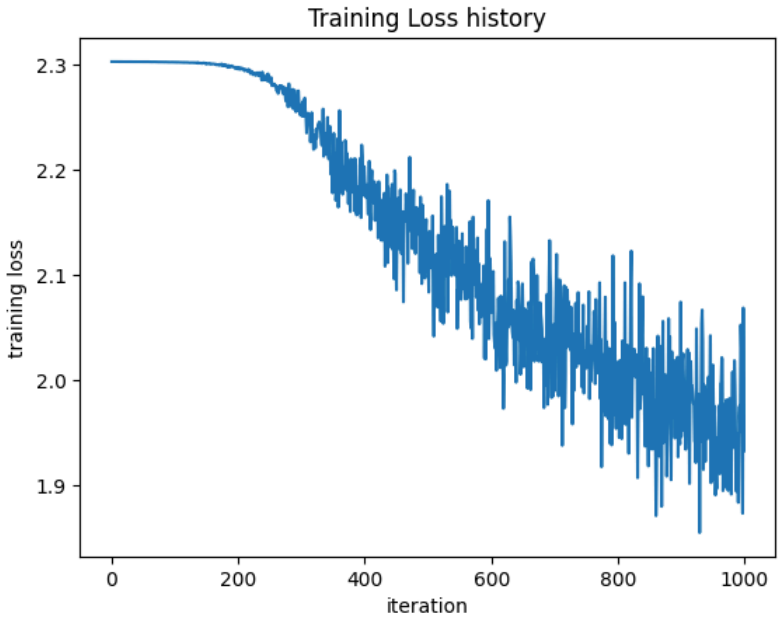
In [34]:
```
stats['train_acc_history']
```

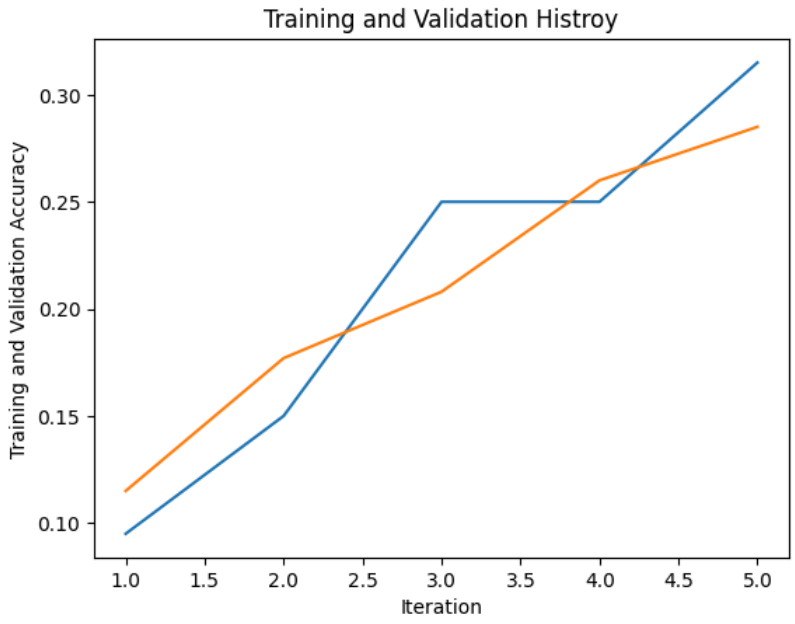
Out[34]:
```
[0.095, 0.15, 0.25, 0.25, 0.315]
```

In [ ]:

```
# ================================================================ #
# YOUR CODE HERE:
#   Do some debugging to gain some insight into why the optimization
#   isn't great.
# ================================================================ #

# Plot the loss function and train / validation accuracies

plt.plot(stats['loss_history'], label='loss')
plt.xlabel('iteration')
plt.ylabel('training loss')
plt.title('Training Loss history')
plt.show()

plt.plot([1,2,3,4,5], stats['train_acc_history'], label='Train Accuracy')
plt.plot([1,2,3,4,5], stats['val_acc_history'], label='Validation Accuracy')
plt.xlabel('Iteration')
plt.ylabel('Training and Validation Accuracy')
plt.title('Training and Validation Histroy')
plt.show()

# ================================================================ #
# END YOUR CODE HERE
# ================================================================ #
```

```
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "orientation" which is no longer supported as of 3.3 and will becom
e an error two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "dpi" which is no longer supported as of 3.3 and will become an err
or two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "facecolor" which is no longer supported as of 3.3 and will become
an error two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "edgecolor" which is no longer supported as of 3.3 and will become
an error two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "bbox_inches_restore" which is no longer supported as of 3.3 and wi
ll become an error two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
```





Answers:

(1) The training performance isn't great because the loss is highly volatile (zigzagging) instead of smoothly converging. This instability can be caused by improper weight initialization with large values, insufficient training time, overly strong regularization, or learning rate issues—either too high, causing overshooting, or too low, leading to slow convergence.

(2) The problem can be fixed by extending traning time, improving weight initialization with methods such as Xavier initiliation to prevent gradients from exploding. We can also experiment with different regularization strenghts and learning rates.

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.

In [48]:

```
# ================================================================ #
# YOUR CODE HERE:
#   Optimize over your hyperparameters to arrive at the best neural
#   network.  You should be able to get over 50% 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 - 28%)) / %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)!
# ================================================================ #
input_size = 32 * 32 * 3
hidden_size = 50
num_classes = 10
net = TwoLayerNet(input_size, hidden_size, num_classes)

# Hyperparameter tuning with Xavier initialization
best_val_acc = 0
best_net = None

# Grid search over hyperparameters
learning_rates=[0.001, 0.005, 0.0001, 0.0005]
regularization_strengths = [0.1, 0.25]
learning_rate_decays = [0.55, 0.60, 0.80]

for lr in learning_rates:
    for reg in regularization_strengths:
        for lr_decay in learning_rate_decays:
            print(f"Training with learning_rate={lr}, reg={reg}, lr_decay={lr_decay}")

            # Initialize the network with Xavier initialization
            net = TwoLayerNet(input_size, hidden_size, num_classes)

            # Train the network
            stats = net.train(X_train, y_train, X_val, y_val,
                              num_iters=1000,
                              batch_size=200,
                              learning_rate=lr,
                              learning_rate_decay=lr_decay,
                              reg=reg,
                              verbose=False)

            # Predict on the validation set
            val_acc = (net.predict(X_val) == y_val).mean()
            print(f"Validation accuracy: {val_acc}")

            # Save the best model
            if val_acc > best_val_acc:
                best_val_acc = val_acc
                best_net = net

print('Best validation accuracy: ', best_val_acc)

# ================================================================ #
# END YOUR CODE HERE
# ================================================================ #


```

```
Training with learning_rate=0.001, reg=0.1, lr_decay=0.55
Validation accuracy: 0.422
Training with learning_rate=0.001, reg=0.1, lr_decay=0.6
Validation accuracy: 0.424
Training with learning_rate=0.001, reg=0.1, lr_decay=0.8
Validation accuracy: 0.467
Training with learning_rate=0.001, reg=0.25, lr_decay=0.55
Validation accuracy: 0.413
Training with learning_rate=0.001, reg=0.25, lr_decay=0.6
Validation accuracy: 0.435
Training with learning_rate=0.001, reg=0.25, lr_decay=0.8
Validation accuracy: 0.472
Training with learning_rate=0.005, reg=0.1, lr_decay=0.55
Validation accuracy: 0.489
Training with learning_rate=0.005, reg=0.1, lr_decay=0.6
Validation accuracy: 0.477
Training with learning_rate=0.005, reg=0.1, lr_decay=0.8
```

```
/Users/sujitsilas/Desktop/UCLA/Winter 2025/EE ENGR 247/Homeworks/HW3/code_student_version/nndl/neural_net.p
y:113: RuntimeWarning: divide by zero encountered in log
  loss += -np.sum(np.log(prob[np.arange(N), y])) / N # softmax loss

Validation accuracy: 0.284
Training with learning_rate=0.005, reg=0.25, lr_decay=0.55
Validation accuracy: 0.496
Training with learning_rate=0.005, reg=0.25, lr_decay=0.6
Validation accuracy: 0.513
Training with learning_rate=0.005, reg=0.25, lr_decay=0.8
Validation accuracy: 0.227
Training with learning_rate=0.0001, reg=0.1, lr_decay=0.55
Validation accuracy: 0.182
Training with learning_rate=0.0001, reg=0.1, lr_decay=0.6
Validation accuracy: 0.183
Training with learning_rate=0.0001, reg=0.1, lr_decay=0.8
Validation accuracy: 0.246
Training with learning_rate=0.0001, reg=0.25, lr_decay=0.55
Validation accuracy: 0.161
Training with learning_rate=0.0001, reg=0.25, lr_decay=0.6
Validation accuracy: 0.194
Training with learning_rate=0.0001, reg=0.25, lr_decay=0.8
Validation accuracy: 0.244
Training with learning_rate=0.0005, reg=0.1, lr_decay=0.55
Validation accuracy: 0.337
Training with learning_rate=0.0005, reg=0.1, lr_decay=0.6
Validation accuracy: 0.365
Training with learning_rate=0.0005, reg=0.1, lr_decay=0.8
Validation accuracy: 0.427
Training with learning_rate=0.0005, reg=0.25, lr_decay=0.55
Validation accuracy: 0.334
Training with learning_rate=0.0005, reg=0.25, lr_decay=0.6
Validation accuracy: 0.363
Training with learning_rate=0.0005, reg=0.25, lr_decay=0.8
Validation accuracy: 0.421
Best validation accuracy:  0.513
```

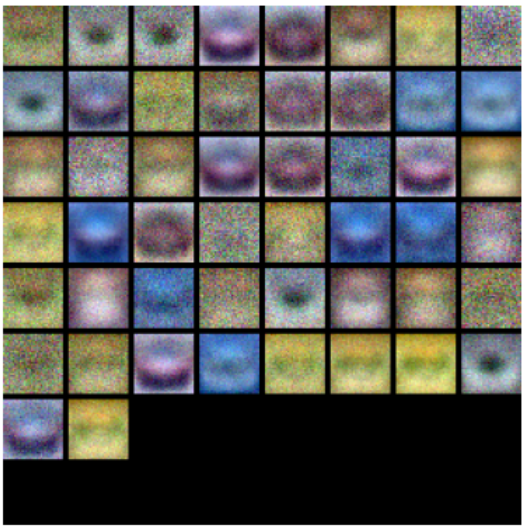
Best Validation Accuracy: 0.513

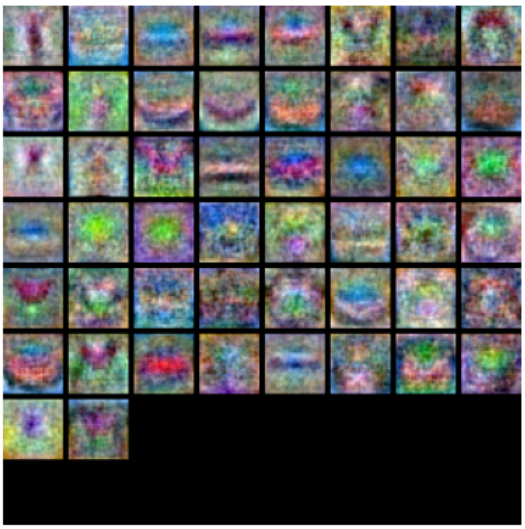
- Training with learning_rate=0.005, reg=0.25, lr_decay=0.6

In [49]:

```
from utils.vis_utils import visualize_grid

# Visualize the weights of the network

def 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()

show_net_weights(subopt_net)
show_net_weights(best_net)
```

```
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "orientation" which is no longer supported as of 3.3 and will becom
e an error two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "dpi" which is no longer supported as of 3.3 and will become an err
or two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "facecolor" which is no longer supported as of 3.3 and will become
an error two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "edgecolor" which is no longer supported as of 3.3 and will become
an error two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
/opt/homebrew/lib/python3.11/site-packages/IPython/core/pylabtools.py:152: MatplotlibDeprecationWarning: sa
vefig() got unexpected keyword argument "bbox_inches_restore" which is no longer supported as of 3.3 and wi
ll become an error two minor releases later
  fig.canvas.print_figure(bytes_io, **kw)
```





Question:

(1) What differences do you see in the weights between the suboptimal net and the best net you arrived at?

Answer:

(1) The best net has more evenly distributed and well-scaled weights compared to the suboptimal net, which may have erratic or extreme values due to small learning rates. The features are more distinctly visible on the best net leading to higher validation accuracy. The weights of the best net look more like a template that the model can usee to assign images to certain classes. The suboptimal net's weights might exhibit a wider spread, leading to vanishing or exploding gradients, while the best net maintains controlled weight distributions for stable optimization.

Evaluate on test set

In [50]:
```
1 test_acc = (best_net.predict(X_test) == y_test).mean()
2 print('Test accuracy: ', test_acc)
```

```
Test accuracy:  0.49
```