ECE 285 Assignment 1: KNN

For this part of assignment, you are tasked to implement KNN algorithm and test it on the a subset of CIFAR10 dataset.

You sould run the whole notebook and answer the question in the notebook.

TO SUBMIT: PDF of this notebook with all the required outputs and answers.

```
# Import Packages
import numpy as np
import matplotlib.pyplot as plt
```

Prepare Dataset

Since CIFAR10 is a relative large dataset, and KNN is quite time-consuming method, we only a small sub-set of CIFAR10 for KNN part

```
from ece285.utils.data_processing import get_cifar10_data

# Use a subset of CIFAR10 for KNN assignments
dataset = get_cifar10_data(subset_train=5000, subset_val=250,
subset_test=500)

print(dataset.keys())
print("Training Set Data  Shape: ", dataset["x_train"].shape)
print("Training Set Label Shape: ", dataset["y_train"].shape)
```

```
dict_keys(['x_train', 'y_train', 'x_val', 'y_val', 'x_test',
'y_test'])
Training Set Data  Shape:  (5000, 3072)
Training Set Label Shape:  (5000,)
```

Implementation (60%)

You need to implement the KNN method in `algorithms/knn.py`. You need to fill in the prediction function(since the training of KNN is just remembering the training set).

For KNN implementation, you are tasked to implement two version of it.

- Two Loop Version: use one loop to iterate through training samples and one loop to iterate through test samples
- One Loop Version: use one loop to iterate through test samples and use broadcast feature of numpy to calculate all the distance at once

Note: It is possible to build a Fully Vectorized Version without explicit for loop to calculate the distance, but you do not have to do it in this assignment. You could use the fully vectorized version to replace the loop versions as well.

For distance function, in this assignment, we use Eucliean distance between samples.

```
from ece285.algorithms import KNN

knn = KNN(num_class=10)
knn.train(
    x_train=dataset["x_train"],
    y_train=dataset["y_train"],
    k=5,
)
```

Compare the time consumption of different method

In this section, you will test your different implementation of KNN method, and compare their speed.

```
from ece285.utils.evaluation import get_classification_accuracy
```

Two Loop Version:

```
import time

c_t = time.time()
prediction = knn.predict(dataset["x_test"], loop_count=2)
print("Two Loop Prediction Time:", time.time() - c_t)

test_acc = get_classification_accuracy(prediction, dataset["y_test"])
print("Test Accuracy:", test_acc)
```

```
Two Loop Prediction Time: 29.956589221954346
Test Accuracy: 0.278
```

One Loop Version

```
import time

c_t = time.time()
prediction = knn.predict(dataset["x_test"], loop_count=1)
print("One Loop Prediction Time:", time.time() - c_t)

test_acc = get_classification_accuracy(prediction, dataset["y_test"])
print("Test Accuracy:", test_acc)
```

```
One Loop Prediction Time: 113.83794379234314
Test Accuracy: 0.278
```

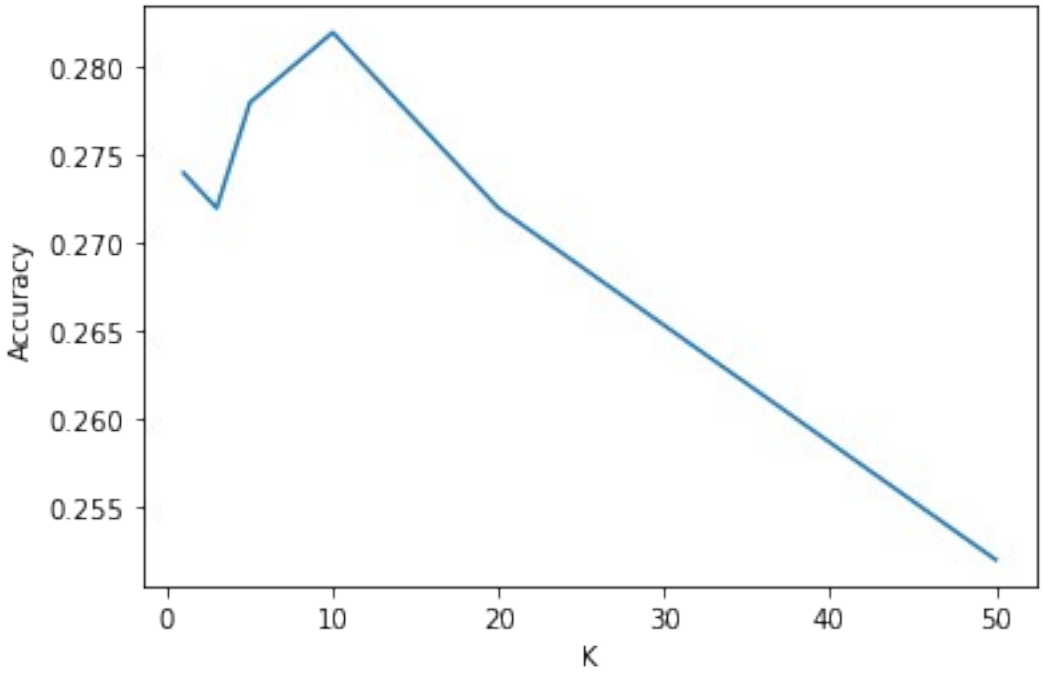
Your different implementation should output the exact same result

Test different Hyper-parameter(20%)

For KNN, there is only one hyper-parameter of the algorithm: How many nearest neighbour to use(**K**).

Here, you are provided the code to test different k for the same dataset.

```
accuracies = []

k_candidates = [1, 3, 5, 10, 20, 50]
for k_cand in k_candidates:
    prediction = knn.predict(x_test=dataset["x_test"], k=k_cand)
    acc = get_classification_accuracy(prediction, dataset["y_test"])
    accuracies.append(acc)
plt.ylabel("Accuracy")
plt.xlabel("K")
plt.plot(k_candidates, accuracies)
plt.show()
```




```
print(accuracies)
```

```
[0.274, 0.272, 0.278, 0.282, 0.272, 0.252]
```

Inline Question 1:

Please describe the output result you get, and provide some explanation as well.

Your Answer:

For k=10, we get maximum accuracy of 0.282 . We have discussed below what happens when k is too small or too big.

When k is too small, the model has high variance. This results in the model being overly sensitive to noise in the data, as it would rely on just a few neighbors for making predictions. The influence of outliers becomes more significant, which can lead to inaccurate predictions. As a result, we see low accuracies for k=1,3 and 5.

When k is too big, the model has low bias. This results in underfitting as a large value of k will result in the model being too smooth. The model would become less sensitive to local patterns in the data, and its decision boundaries may become too simplistic, reducing its ability to make accurate predictions. Also, when the model is too large, the majority class can dominate the voting, which can lead to the model becoming less sensitive to the minority class. As a result, accuracy decreases when k becomes greater than 10.

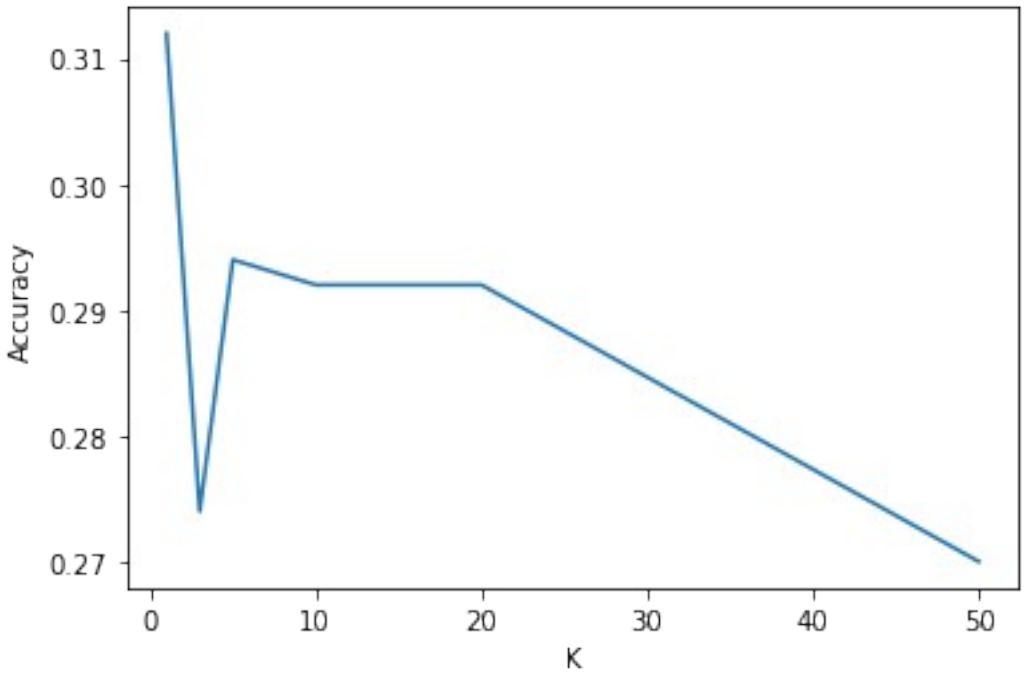
Try different feature representation(20%)

Since machine learning method rely heavily on the feature extraction, you will see how different feature representation affect the performance of the algorithm in this section.

You are provided the code about using **HOG** descriptor to represent samples in the notebook.

```
from ece285.utils.data_processing import get_cifar10_data
from ece285.utils.data_processing import HOG_preprocess
from functools import partial

# Delete previous dataset to save memory
del dataset
del knn

# Use a subset of CIFAR10 for KNN assignments
hog_p_func = partial(
    HOG_preprocess,
    orientations=9,
    pixels_per_cell=(4, 4),
    cells_per_block=(1, 1),
    visualize=False,
    multichannel=True,
)
dataset = get_cifar10_data(
    feature_process=hog_p_func, subset_train=5000, subset_val=250,
subset_test=500
)

Start Processing
Processing Time: 10.308724164962769
```

```
knn = KNN(num_class=10)
knn.train(
    x_train=dataset["x_train"],
    y_train=dataset["y_train"],
    k=5,
)
accuracies = []

k_candidates = [1, 3, 5, 10, 20, 50]
for k_cand in k_candidates:
    prediction = knn.predict(x_test=dataset["x_test"], k=k_cand)
    acc = get_classification_accuracy(prediction, dataset["y_test"])
    accuracies.append(acc)

plt.ylabel("Accuracy")
plt.xlabel("K")
plt.plot(k_candidates, accuracies)
plt.show()
```



Inline Question 2:

Please describe the output result you get, compare with the result you get in the previous section, and provide some explanation as well.

Your Answer:

Put Your Answer Here

In this case, we see a high accuracy when k=1. The accuracy remains high till k=20 and then starts decreasing.

When k is too small, the model is overfitting, where the algorithm is relying on just a few neighbors. The reason for high accuracy could be that the cifar data is very simple and does not have noise. When we use the subset of the data, it

ECE 285 Assignment 1: Linear Regression

For this part of assignment, you are tasked to implement a linear regression algorithm for multiclass classification and test it on the CIFAR10 dataset.

You sould run the whole notebook and answer the questions in the notebook.

CIFAR 10 dataset contains 32x32x3 RGB images of 10 distinct cateogaries, and our aim is to predict which class the image belongs to

TO SUBMIT: PDF of this notebook with all the required outputs and answers.

```
# Prepare Packages
import numpy as np
import matplotlib.pyplot as plt

from ece285.utils.data_processing import get_cifar10_data

# Use a subset of CIFAR10 for the assignment
dataset = get_cifar10_data(
    subset_train=5000,
    subset_val=250,
    subset_test=500,
)

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:  (5000, 3072)
Training Set Label Shape:  (5000,)
Validation Set Data  Shape:  (250, 3072)
Validation Set Label Shape:  (250,)
Test Set Data  Shape:  (500, 3072)
Test Set Label Shape:  (500,)
```

```
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"]
```

```
# Visualize some examples from the dataset.
# We show a few examples of training images from each class.
classes = [
    "plane",
    "car",
    "bird",
    "cat",
    "deer",
    "dog",
    "frog",
    "horse",
    "ship",
    "truck",
]
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(5000, 3, 32, 32).transpose(0, 2, 3, 1), classes,
samples_per_class
)
```

Linear Regression for multi-class classification

A Linear Regression Algorithm has 2 hyperparameters that you can experiment with:

- **Learning rate** - controls how much we change the current weights of the classifier during each update. We set it at a default value of 0.5, and later you are asked to experiment with different values. We recommend looking at the graphs and observing how the performance of the classifier changes with different learning rate.
- **Number of Epochs** - An epoch is a complete iterative pass over all of the data in the dataset. During an epoch we predict a label using the classifier and then update the weights of the classifier according the linear classifier update rule for each sample in the training set. We evaluate our models after every 10 epochs and save the accuracies, which are later used to plot the training, validation and test VS epoch curves.
- **Weight Decay** - Regularization can be used to constrain the weights of the classifier and prevent their values from blowing up. Regularization helps in combatting overfitting. You will be using the 'weight_decay' term to introduce regularization in the classifier.

Implementation (50%)

You first need to implement the Linear Regression method in `algorithms/linear_regression.py`. You need to fill in the training function as well as the prediction function.

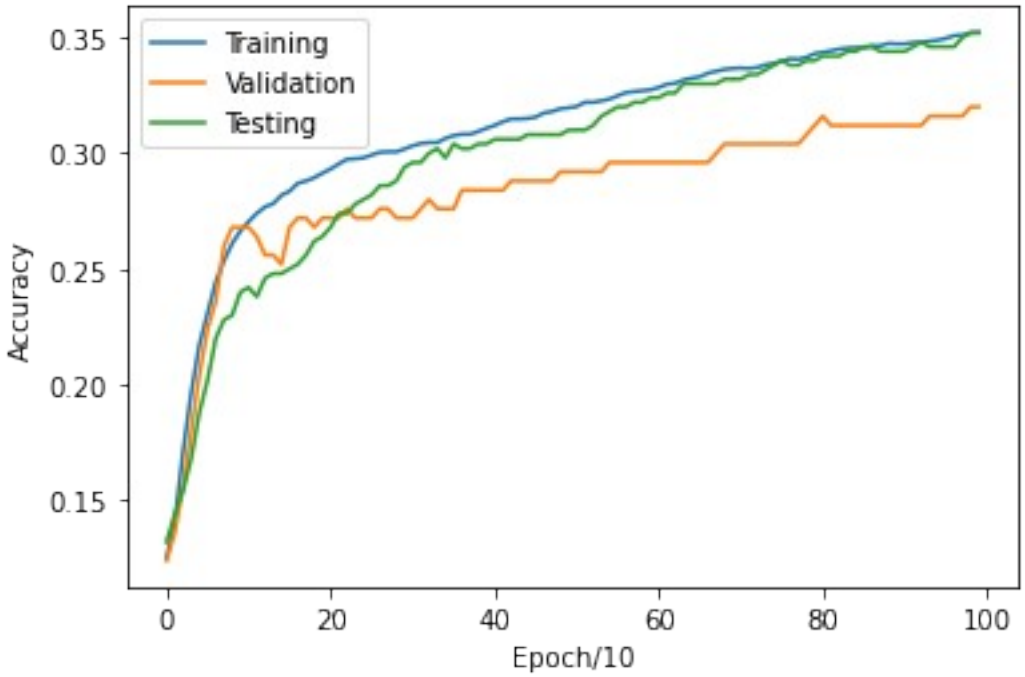
```
# Import the algorithm implementation (TODO: Complete the Linear
Regression in algorithms/linear_regression.py)
```

```
from ece285.algorithms import Linear
from ece285.utils.evaluation import get_classification_accuracy

num_classes = 10  # Cifar10 dataset has 10 different classes

# Initialize hyper-parameters
learning_rate = 0.0001  # You will be later asked to experiment with
different learning rates and report results
num_epochs_total = 1000  # Total number of epochs to train the
classifier
epochs_per_evaluation = 10  # Epochs per step of evaluation; We will
evaluate our model regularly during training
N, D = dataset[
    "x_train"
].shape  # Get training data shape, N: Number of examples,
D:Dimensionality of the data
weight_decay = 0.0

# Insert additional scalar term 1 in the samples to account for the
bias as discussed in class
x_train = np.insert(x_train, D, values=1, axis=1)
x_val = np.insert(x_val, D, values=1, axis=1)
x_test = np.insert(x_test, D, values=1, axis=1)

# Training and evaluation function -> Outputs accuracy data
def train(learning_rate_, weight_decay_):
    # Create a linear regression object
    linear_regression = Linear(
        num_classes, learning_rate_, epochs_per_evaluation,
weight_decay_
    )

    # Randomly initialize the weights and biases
    weights = np.random.randn(num_classes, D + 1) * 0.0001

    train_accuracies, val_accuracies, test_accuracies = [], [], []

    # Train the classifier
    for _ in range(int(num_epochs_total / epochs_per_evaluation)):
        # Train the classifier on the training data
        weights = linear_regression.train(x_train, y_train, weights)

        # Evaluate the trained classifier on the training dataset
        y_pred_train = linear_regression.predict(x_train)

train_accuracies.append(get_classification_accuracy(y_pred_train,
y_train))

        # Evaluate the trained classifier on the validation dataset
```

```
        y_pred_val = linear_regression.predict(x_val)
        val_accuracies.append(get_classification_accuracy(y_pred_val,
y_val))

        # Evaluate the trained classifier on the test dataset
        y_pred_test = linear_regression.predict(x_test)

test_accuracies.append(get_classification_accuracy(y_pred_test,
y_test))

    return train_accuracies, val_accuracies, test_accuracies, weights
```

Plot the Accuracies vs epoch graphs

```
import matplotlib.pyplot as plt


def plot_accuracies(train_acc, val_acc, test_acc):
    # Plot Accuracies vs Epochs graph for all the three
    epochs = np.arange(0, int(num_epochs_total /
epochs_per_evaluation))
    plt.ylabel("Accuracy")
    plt.xlabel("Epoch/10")
    plt.plot(epochs, train_acc, epochs, val_acc, epochs, test_acc)
    plt.legend(["Training", "Validation", "Testing"])
    plt.show()

# Run training and plotting for default parameter values as mentioned
above
t_ac, v_ac, te_ac, weights = train(learning_rate, weight_decay)

plot_accuracies(t_ac, v_ac, te_ac)
```




```
print(te_ac)
```

```
[0.132, 0.144, 0.154, 0.168, 0.188, 0.202, 0.22, 0.228, 0.23, 0.24,
0.242, 0.238, 0.246, 0.248, 0.248, 0.25, 0.252, 0.256, 0.262, 0.264,
0.268, 0.274, 0.274, 0.278, 0.28, 0.282, 0.286, 0.286, 0.288, 0.294,
0.296, 0.296, 0.3, 0.302, 0.298, 0.304, 0.302, 0.302, 0.304, 0.304,
0.306, 0.306, 0.306, 0.306, 0.308, 0.308, 0.308, 0.308, 0.308, 0.31,
0.31, 0.31, 0.312, 0.316, 0.318, 0.32, 0.32, 0.322, 0.322, 0.324,
0.324, 0.326, 0.326, 0.33, 0.33, 0.33, 0.33, 0.33, 0.332, 0.332,
0.332, 0.334, 0.334, 0.336, 0.338, 0.34, 0.338, 0.338, 0.34, 0.34,
0.342, 0.342, 0.342, 0.344, 0.344, 0.346, 0.346, 0.344, 0.344, 0.344,
0.344, 0.346, 0.348, 0.346, 0.346, 0.346, 0.346, 0.35, 0.352, 0.352]
```

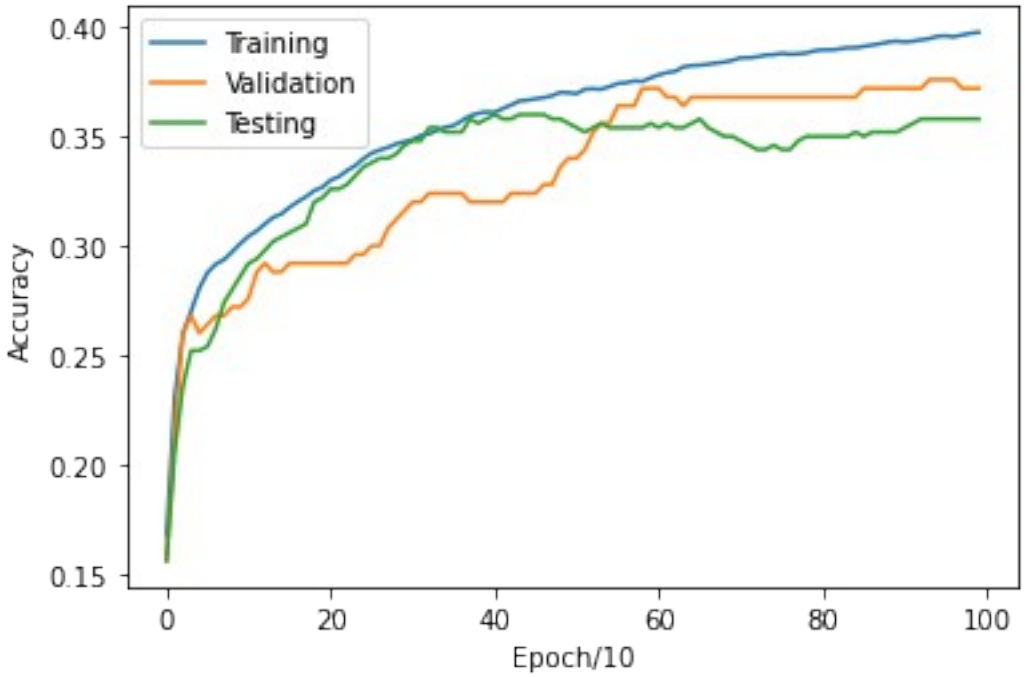
```
print(learning_rate)
```

```
0.0001
```

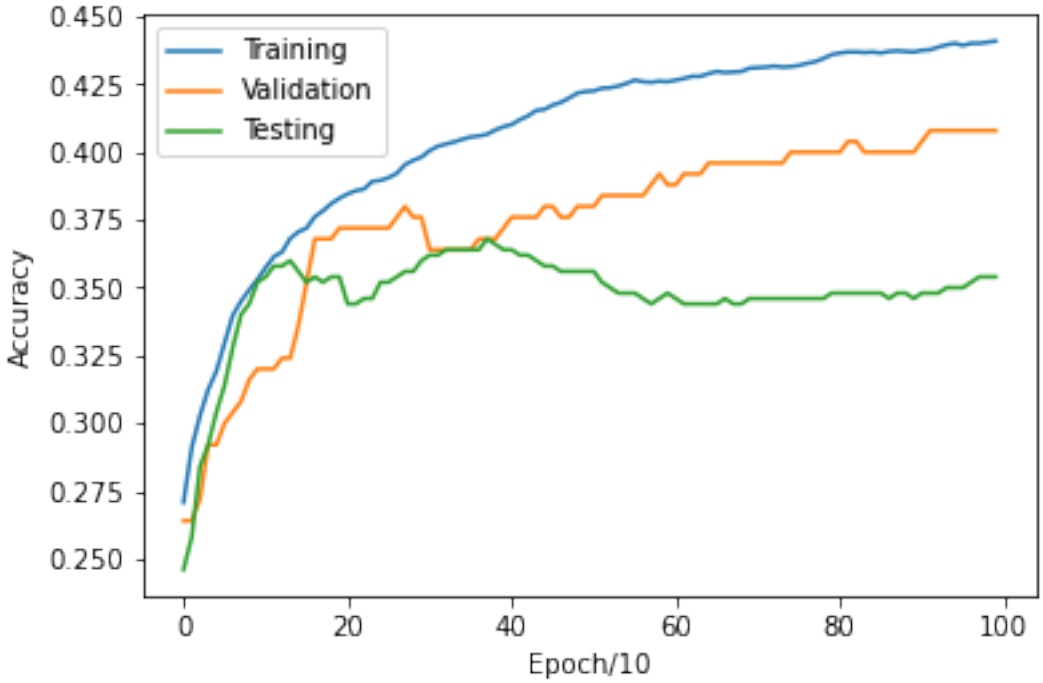
Try different learning rates and plot graphs for all (20%)

```
# Initialize the best values
best_weights = weights
best_learning_rate = learning_rate
best_weight_decay = weight_decay
# TODO
# Repeat the above training and evaluation steps for the following
learning rates and plot graphs
# You need to try 3 learning rates and submit all 3 graphs along with
this notebook pdf to show your learning rate experiments
learning_rates = [0.0003, 1e-3, 0.5e-3]
weight_decay = 0.0  # No regularization for now
```

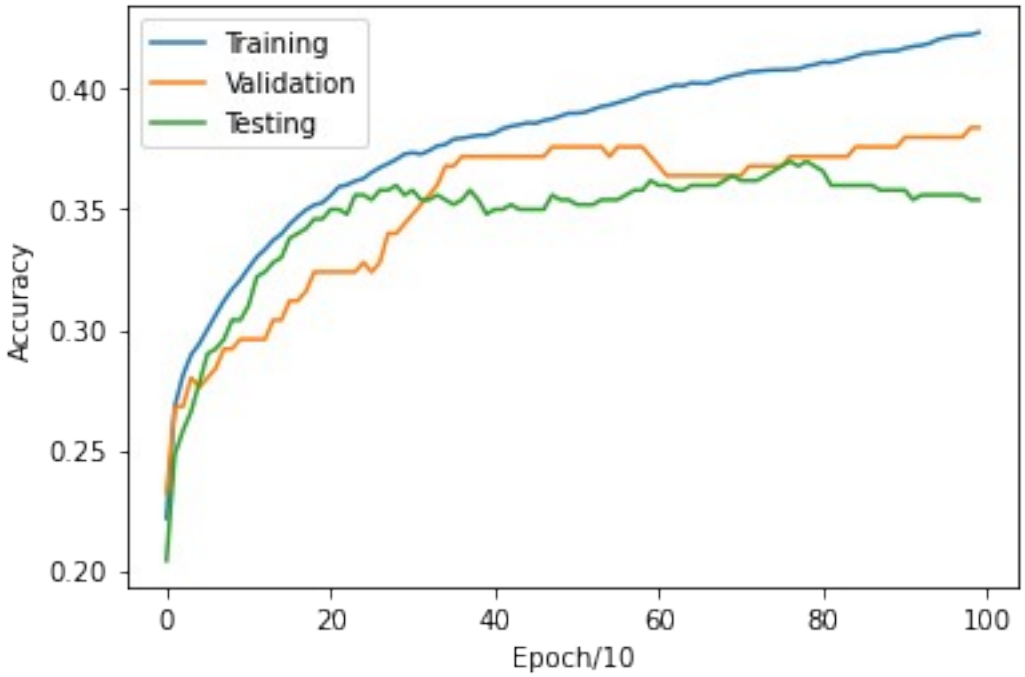
```
# FEEL FREE TO EXPERIMENT WITH OTHER VALUES. REPORT OTHER VALUES IF
THEY ACHIEVE A BETTER PERFORMANCE

# for lr in learning_rates: Train the classifier and plot data
# Step 1. train_accu, val_accu, test_accu = train(lr, weight_decay)
# Step 2. plot_accuracies(train_accu, val_accu, test_accu)

for learning_rate in learning_rates:
    # TODO: Train the classifier with different learning rates and
plot
    t_ac, v_ac, te_ac, weights = train(learning_rate, weight_decay)
    plot_accuracies(t_ac, v_ac, te_ac)
    print(te_ac)

print(te_ac)
```




```
[0.156, 0.206, 0.236, 0.252, 0.252, 0.254, 0.262, 0.274, 0.28, 0.286,
0.292, 0.294, 0.298, 0.302, 0.304, 0.306, 0.308, 0.31, 0.32, 0.322,
0.326, 0.326, 0.328, 0.332, 0.336, 0.338, 0.34, 0.34, 0.342, 0.346,
0.348, 0.348, 0.354, 0.354, 0.352, 0.352, 0.352, 0.358, 0.356, 0.358,
0.36, 0.358, 0.358, 0.36, 0.36, 0.36, 0.36, 0.358, 0.358, 0.356,
0.354, 0.352, 0.354, 0.356, 0.354, 0.354, 0.354, 0.354, 0.354, 0.356,
0.354, 0.356, 0.354, 0.354, 0.356, 0.358, 0.354, 0.352, 0.35, 0.35,
0.348, 0.346, 0.344, 0.344, 0.346, 0.344, 0.344, 0.348, 0.35, 0.35,
0.35, 0.35, 0.35, 0.35, 0.352, 0.35, 0.352, 0.352, 0.352, 0.352,
0.354, 0.356, 0.358, 0.358, 0.358, 0.358, 0.358, 0.358, 0.358, 0.358]
```




```
[0.246, 0.258, 0.284, 0.292, 0.304, 0.314, 0.328, 0.34, 0.344, 0.352,
0.354, 0.358, 0.358, 0.36, 0.356, 0.352, 0.354, 0.352, 0.354, 0.354,
0.344, 0.344, 0.346, 0.346, 0.352, 0.352, 0.354, 0.356, 0.356, 0.36,
0.362, 0.362, 0.364, 0.364, 0.364, 0.364, 0.364, 0.368, 0.366, 0.364,
0.364, 0.362, 0.362, 0.36, 0.358, 0.358, 0.356, 0.356, 0.356, 0.356,
0.356, 0.352, 0.35, 0.348, 0.348, 0.348, 0.346, 0.344, 0.346, 0.348,
0.346, 0.344, 0.344, 0.344, 0.344, 0.344, 0.346, 0.344, 0.344, 0.346,
0.346, 0.346, 0.346, 0.346, 0.346, 0.346, 0.346, 0.346, 0.346, 0.348,
0.348, 0.348, 0.348, 0.348, 0.348, 0.348, 0.346, 0.348, 0.348, 0.346,
0.348, 0.348, 0.348, 0.35, 0.35, 0.35, 0.352, 0.354, 0.354, 0.354]
```




```
[0.204, 0.248, 0.258, 0.266, 0.278, 0.29, 0.292, 0.296, 0.304, 0.304,
0.31, 0.322, 0.324, 0.328, 0.33, 0.338, 0.34, 0.342, 0.346, 0.346,
0.35, 0.35, 0.348, 0.356, 0.356, 0.354, 0.358, 0.358, 0.36, 0.356,
0.358, 0.354, 0.354, 0.356, 0.354, 0.352, 0.354, 0.358, 0.354, 0.348,
0.35, 0.35, 0.352, 0.35, 0.35, 0.35, 0.35, 0.356, 0.354, 0.354, 0.352,
0.352, 0.352, 0.354, 0.354, 0.354, 0.356, 0.358, 0.358, 0.362, 0.36,
0.36, 0.358, 0.358, 0.36, 0.36, 0.36, 0.36, 0.362, 0.364, 0.362,
0.362, 0.362, 0.364, 0.366, 0.368, 0.37, 0.368, 0.37, 0.368, 0.366,
0.36, 0.36, 0.36, 0.36, 0.36, 0.36, 0.358, 0.358, 0.358, 0.358, 0.354,
0.356, 0.356, 0.356, 0.356, 0.356, 0.356, 0.354, 0.354]
[0.204, 0.248, 0.258, 0.266, 0.278, 0.29, 0.292, 0.296, 0.304, 0.304,
0.31, 0.322, 0.324, 0.328, 0.33, 0.338, 0.34, 0.342, 0.346, 0.346,
0.35, 0.35, 0.348, 0.356, 0.356, 0.354, 0.358, 0.358, 0.36, 0.356,
0.358, 0.354, 0.354, 0.356, 0.354, 0.352, 0.354, 0.358, 0.354, 0.348,
0.35, 0.35, 0.352, 0.35, 0.35, 0.35, 0.35, 0.356, 0.354, 0.354, 0.352,
0.352, 0.352, 0.354, 0.354, 0.354, 0.356, 0.358, 0.358, 0.362, 0.36,
0.36, 0.358, 0.358, 0.36, 0.36, 0.36, 0.36, 0.362, 0.364, 0.362,
0.362, 0.362, 0.364, 0.366, 0.368, 0.37, 0.368, 0.37, 0.368, 0.366,
0.36, 0.36, 0.36, 0.36, 0.36, 0.36, 0.358, 0.358, 0.358, 0.358, 0.354,
0.356, 0.356, 0.356, 0.356, 0.356, 0.356, 0.354, 0.354]
```

Inline Question 1.

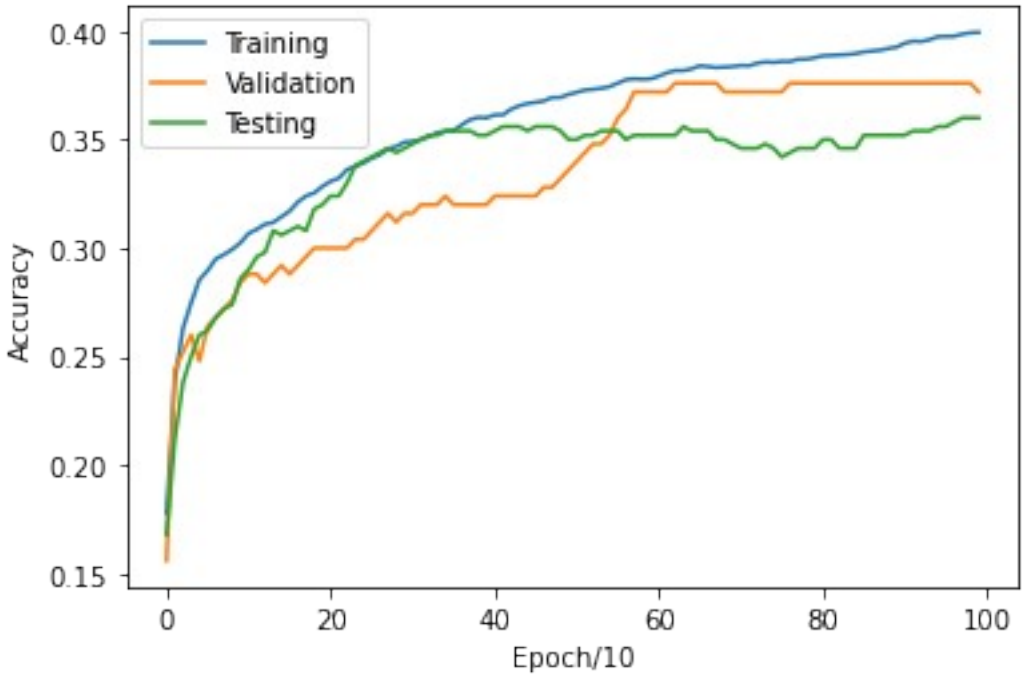
Which one of these learning rates (best_lr) would you pick to train your model? Please Explain why.

Your Answer:

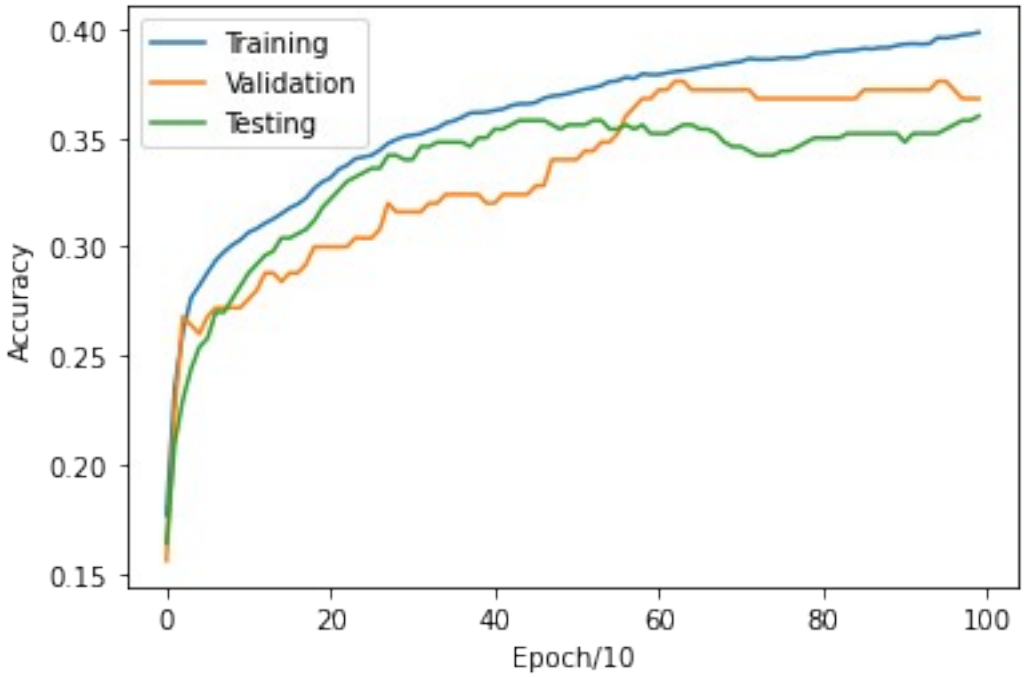
I tried with [0.0003, 1e-3, 0.5e-3] as the learning rates. I picked the best learning rate as 0.0003. This is because I am getting the maximum test accuracy of 0.358 on the test dataset. The learning rate of 1e-3 becomes slightly small which results in slow convergence thereby resulting in a drop in the test accuracy. The learning rate of 0.5e-3 is also small which results in the slower convergence, thereby decreasing the test accuracy.

Regularization: Try different weight decay and plot graphs for all (20%)

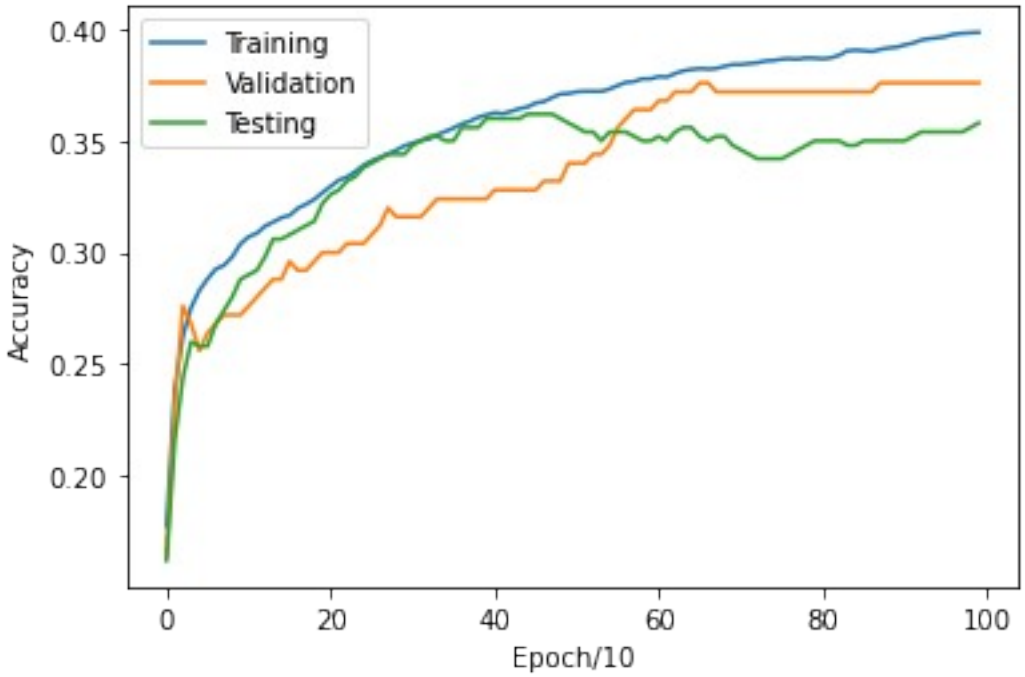
```
# Initialize a non-zero weight_decay (Regularization constant) term
and repeat the training and evaluation
# Use the best learning rate as obtained from the above exercise,
best_lr

# You need to try 3 learning rates and submit all 3 graphs along with
this notebook pdf to show your weight decay experiments
weight_decays = [0.001, 0.01, 0.1, 1, 5]
best_learning_rate = 0.0003
# FEEL FREE TO EXPERIMENT WITH OTHER VALUES. REPORT OTHER VALUES IF
THEY ACHIEVE A BETTER PERFORMANCE

# for weight_decay in weight_decays: Train the classifier and plot
data
# Step 1. train_accu, val_accu, test_accu = train(best_lr,
weight_decay)
# Step 2. plot_accuracies(train_accu, val_accu, test_accu)

for weight_decay in weight_decays:
    # TODO: Train the classifier with different weighty decay and plot
    t_ac, v_ac, te_ac, weights = train(best_learning_rate,
weight_decay)
    plot_accuracies(t_ac, v_ac, te_ac)
    print(te_ac)
```



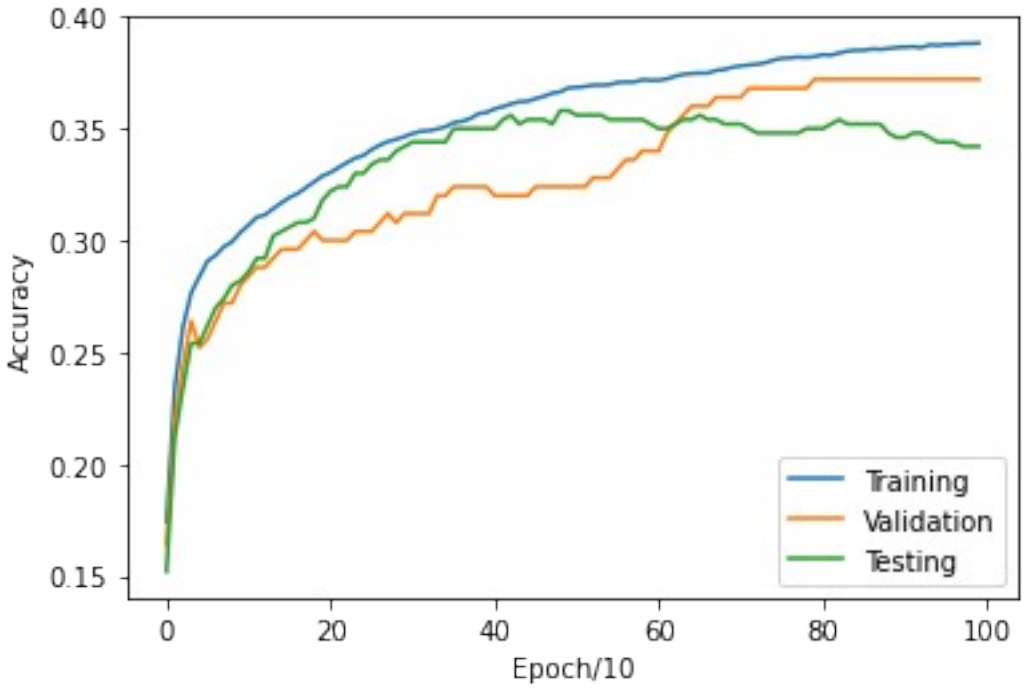
[0.168, 0.212, 0.238, 0.25, 0.26, 0.262, 0.268, 0.272, 0.274, 0.286,
0.29, 0.296, 0.298, 0.308, 0.306, 0.308, 0.31, 0.308, 0.318, 0.32,
0.324, 0.324, 0.33, 0.338, 0.34, 0.342, 0.344, 0.346, 0.344, 0.346,
0.348, 0.35, 0.352, 0.352, 0.354, 0.354, 0.354, 0.354, 0.352, 0.352,
0.354, 0.356, 0.356, 0.356, 0.354, 0.356, 0.356, 0.356, 0.354, 0.35,
0.35, 0.352, 0.352, 0.354, 0.354, 0.354, 0.35, 0.352, 0.352, 0.352,
0.352, 0.352, 0.352, 0.356, 0.354, 0.354, 0.354, 0.35, 0.35, 0.348,
0.346, 0.346, 0.346, 0.348, 0.346, 0.342, 0.344, 0.346, 0.346, 0.346,
0.35, 0.35, 0.346, 0.346, 0.346, 0.352, 0.352, 0.352, 0.352, 0.352,
0.352, 0.354, 0.354, 0.354, 0.356, 0.356, 0.358, 0.36, 0.36, 0.36]



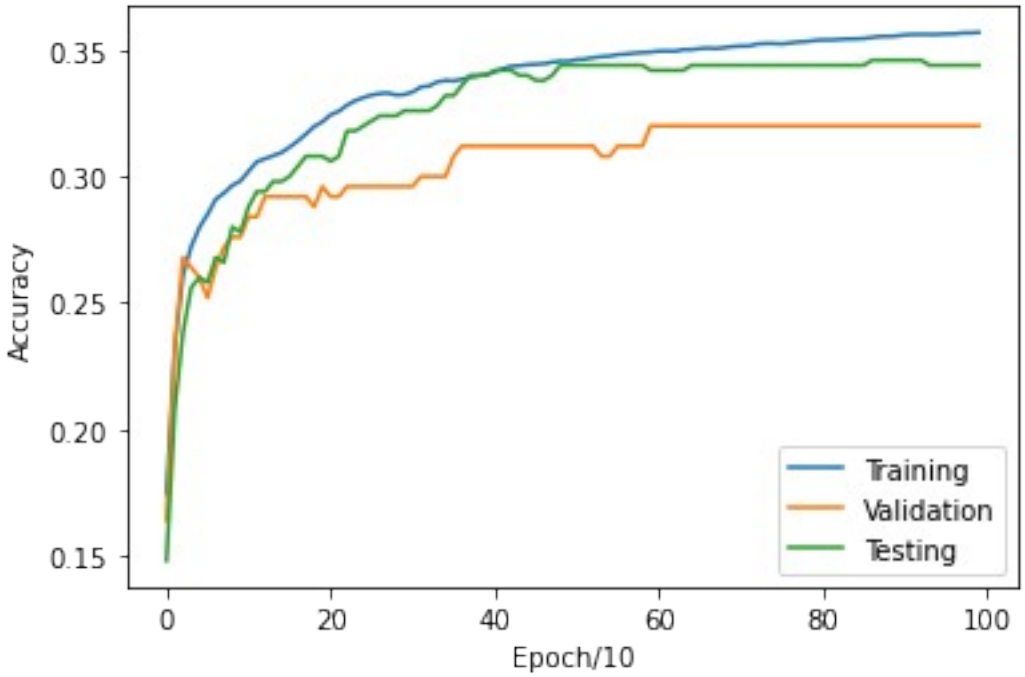

```
[0.164, 0.21, 0.23, 0.244, 0.254, 0.258, 0.27, 0.27, 0.276, 0.282,
0.288, 0.292, 0.296, 0.298, 0.304, 0.304, 0.306, 0.308, 0.312, 0.318,
0.322, 0.326, 0.33, 0.332, 0.334, 0.336, 0.336, 0.342, 0.342, 0.34,
0.34, 0.346, 0.346, 0.348, 0.348, 0.348, 0.348, 0.346, 0.35, 0.35,
0.354, 0.354, 0.356, 0.358, 0.358, 0.358, 0.358, 0.356, 0.354, 0.356,
0.356, 0.356, 0.358, 0.358, 0.354, 0.354, 0.356, 0.354, 0.356, 0.352,
0.352, 0.352, 0.354, 0.356, 0.356, 0.354, 0.354, 0.352, 0.348, 0.346,
0.346, 0.344, 0.342, 0.342, 0.342, 0.344, 0.344, 0.346, 0.348, 0.35,
0.35, 0.35, 0.35, 0.352, 0.352, 0.352, 0.352, 0.352, 0.352, 0.352,
0.348, 0.352, 0.352, 0.352, 0.352, 0.354, 0.356, 0.358, 0.358, 0.36]
```




```
[0.162, 0.216, 0.244, 0.26, 0.258, 0.258, 0.268, 0.274, 0.28, 0.288,
0.29, 0.292, 0.298, 0.306, 0.306, 0.308, 0.31, 0.312, 0.314, 0.322,
0.326, 0.328, 0.332, 0.334, 0.338, 0.34, 0.342, 0.344, 0.344, 0.344,
0.348, 0.35, 0.352, 0.352, 0.35, 0.35, 0.356, 0.356, 0.356, 0.36,
0.36, 0.36, 0.36, 0.36, 0.362, 0.362, 0.362, 0.362, 0.36, 0.358,
0.356, 0.354, 0.354, 0.35, 0.354, 0.354, 0.354, 0.352, 0.35, 0.35,
0.352, 0.35, 0.354, 0.356, 0.356, 0.352, 0.35, 0.352, 0.352, 0.348,
0.346, 0.344, 0.342, 0.342, 0.342, 0.342, 0.344, 0.346, 0.348, 0.35,
0.35, 0.35, 0.35, 0.348, 0.348, 0.35, 0.35, 0.35, 0.35, 0.35, 0.35,
0.352, 0.354, 0.354, 0.354, 0.354, 0.354, 0.354, 0.356, 0.358]
```



[0.152, 0.212, 0.234, 0.254, 0.254, 0.262, 0.27, 0.274, 0.28, 0.282,
0.286, 0.292, 0.292, 0.302, 0.304, 0.306, 0.308, 0.308, 0.31, 0.318,
0.322, 0.324, 0.324, 0.33, 0.33, 0.334, 0.336, 0.336, 0.34, 0.342,
0.344, 0.344, 0.344, 0.344, 0.344, 0.35, 0.35, 0.35, 0.35, 0.35, 0.35,
0.354, 0.356, 0.352, 0.354, 0.354, 0.354, 0.352, 0.358, 0.358, 0.356,
0.356, 0.356, 0.356, 0.354, 0.354, 0.354, 0.354, 0.354, 0.352, 0.35,
0.35, 0.352, 0.354, 0.354, 0.356, 0.354, 0.354, 0.352, 0.352, 0.352,
0.35, 0.348, 0.348, 0.348, 0.348, 0.348, 0.348, 0.35, 0.35, 0.35,
0.352, 0.354, 0.352, 0.352, 0.352, 0.352, 0.352, 0.348, 0.346, 0.346,
0.348, 0.348, 0.346, 0.344, 0.344, 0.344, 0.342, 0.342, 0.342]




```
[0.148, 0.208, 0.238, 0.256, 0.26, 0.258, 0.268, 0.266, 0.28, 0.278,
0.288, 0.294, 0.294, 0.298, 0.298, 0.3, 0.304, 0.308, 0.308, 0.308,
0.306, 0.308, 0.318, 0.318, 0.32, 0.322, 0.324, 0.324, 0.324, 0.326,
0.326, 0.326, 0.326, 0.328, 0.332, 0.332, 0.336, 0.34, 0.34, 0.34,
0.342, 0.342, 0.342, 0.34, 0.34, 0.338, 0.338, 0.34, 0.344, 0.344,
0.344, 0.344, 0.344, 0.344, 0.344, 0.344, 0.344, 0.344, 0.344, 0.342,
0.342, 0.342, 0.342, 0.342, 0.344, 0.344, 0.344, 0.344, 0.344, 0.344,
0.344, 0.344, 0.344, 0.344, 0.344, 0.344, 0.344, 0.344, 0.344, 0.344,
0.344, 0.344, 0.344, 0.344, 0.344, 0.344, 0.346, 0.346, 0.346, 0.346,
0.346, 0.346, 0.346, 0.344, 0.344, 0.344, 0.344, 0.344, 0.344, 0.344]
```

Inline Question 2.

Discuss underfitting and overfitting as observed in the 5 graphs obtained by changing the regularization. Which weight_decay term gave you the best classifier performance? HINT: Do not just think in terms of best training set performance, keep in mind that the real utility of a machine learning model is when it performs well on data it has never seen before

Your Answer:

- Overfitting happens when the weight_decay is too small as the regularization term becomes negligible.
- Underfitting happens when the weight_decay is too large as the regularization term dominates the loss function. I tried weight decays of [0.001, 0.01, 0.1, 1, 5].
- I picked 0.001 as the best weight_decay term because I am getting the best test classification accuracy of 0.36 in this case.
- The model suffers from overfitting in k = [0.001, 0.01, 0.1, 1] as the training accuracies are high but the test accuracies are low.

- The model with k=5 is optimal as the training and test accuracies are close to each other.
- Underfitting is not happening for any of the above values of k. If underfitting had happened, both the training and test accuracies would have been low.

Visualize the filters (10%)

```
# These visualizations will only somewhat make sense if your learning
rate and weight_decay parameters were
# properly chosen in the model. Do your best.

# TODO: Run this cell and Show filter visualizations for the best set
of weights you obtain.
# Report the 2 hyperparameters you used to obtain the best model.

# NOTE: You need to set `best_learning_rate` and `best_weight_decay`
to the values that gave the highest accuracy
best_learning_rate = 0.0003
best_weight_decay = 0.001
print("Best LR:", best_learning_rate)
print("Best Weight Decay:", best_weight_decay)

# NOTE: You need to set `best_weights` to the weights with the highest
accuracy
w = best_weights[:, :-1]
w = w.reshape(10, 3, 32, 32).transpose(0, 2, 3, 1)

w_min, w_max = np.min(w), np.max(w)

fig = plt.figure(figsize=(20, 20))
classes = [
    "plane",
    "car",
    "bird",
    "cat",
    "deer",
    "dog",
    "frog",
    "horse",
    "ship",
    "truck",
]
for i in range(10):
    fig.add_subplot(2, 5, i + 1)

    # Rescale the weights to be between 0 and 255
    wimg = 255.0 * (w[i, :, :, :].squeeze() - w_min) / (w_max - w_min)
    # plt.imshow(wimg.astype('uint8'))
    plt.imshow(wimg.astype(int))
    plt.axis("off")
```

```
    plt.title(classes[i])
plt.show()
```

```
Best LR: 0.0003
Best Weight Decay: 0.001
```

ECE 285 Assignment 1: Logistic Regression

For this part of assignment, you are tasked to implement a logistic regression algorithm for multiclass classification and test it on the CIFAR10 dataset.

You sould run the whole notebook and answer the questions in the notebook.

TO SUBMIT: PDF of this notebook with all the required outputs and answers.

```
# Prepare Packages
import numpy as np
import matplotlib.pyplot as plt

from ece285.utils.data_processing import get_cifar10_data

# Use a subset of CIFAR10 for KNN assignments
dataset = get_cifar10_data(
    subset_train=5000,
    subset_val=250,
    subset_test=500,
)

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:  (5000, 3072)
Training Set Label Shape:  (5000,)
Validation Set Data  Shape:  (250, 3072)
Validation Set Label Shape:  (250,)
Test Set Data  Shape:  (500, 3072)
Test Set Label Shape:  (500,)
```

Logistic Regression for multi-class classification

A Logistic Regression Algorithm has 3 hyperparameters that you can experiment with:

- **Learning rate** - controls how much we change the current weights of the classifier during each update. We set it at a default value of 0.5, and later you are asked to experiment with different values. We recommend looking at the graphs and

observing how the performance of the classifier changes with different learning rate.

- **Number of Epochs** - An epoch is a complete iterative pass over all of the data in the dataset. During an epoch we predict a label using the classifier and then update the weights of the classifier according the linear classifier update rule for each sample in the training set. We evaluate our models after every 10 epochs and save the accuracies, which are later used to plot the training, validation and test VS epoch curves.
- **Weight Decay** - Regularization can be used to constrain the weights of the classifier and prevent their values from blowing up. Regularization helps in combatting overfitting. You will be using the 'weight_decay' term to introduce regularization in the classifier.

The only way how a Logistic Regression based classification algorithm is different from a Linear Regression algorithm is that in the former we additionally pass the classifier outputs into a sigmoid function which squashes the output in the (0,1) range. Essentially these values then represent the probabilities of that sample belonging to class particular classes

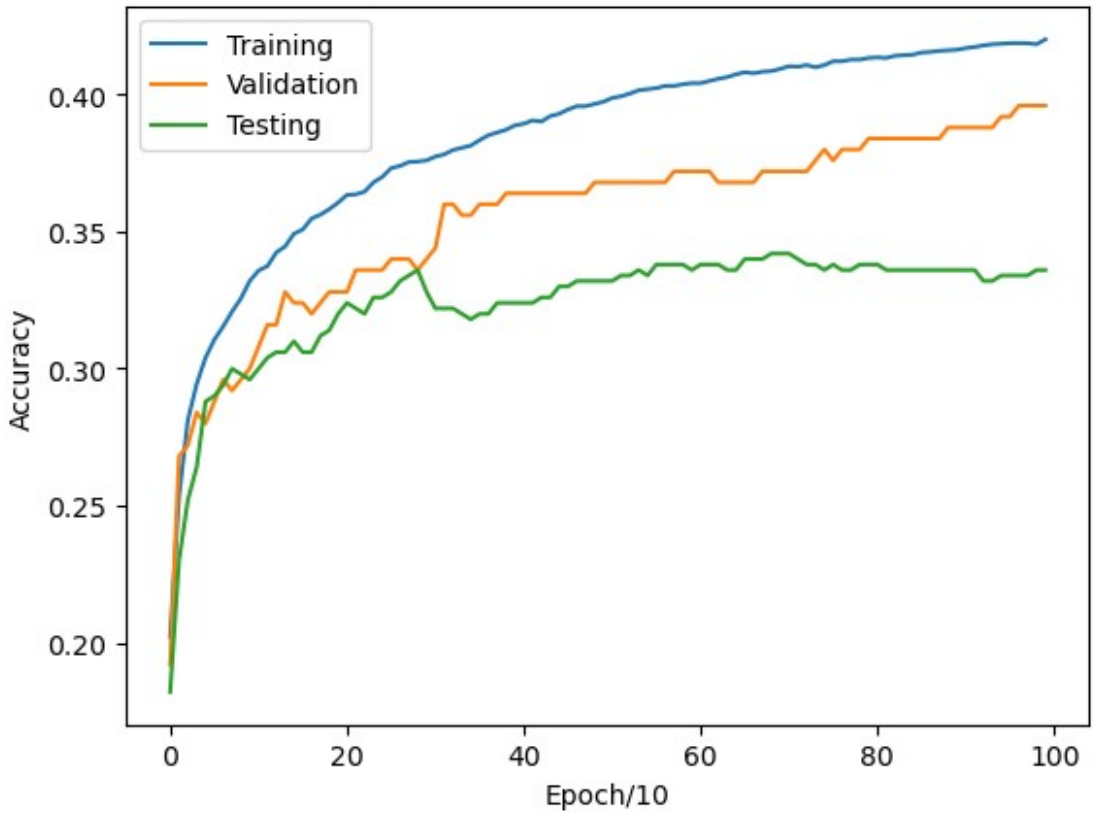
Implementation (40%)

You need to implement the Linear Regression method in `algorithms/logistic_regression.py`. You need to fill in the sigmoid function, training function as well as the prediction function.

```
# Import the algorithm implementation (TODO: Complete the Logistic
Regression in algorithms/logistic_regression.py)
from ece285.algorithms import Logistic
from ece285.utils.evaluation import get_classification_accuracy

num_classes = 10  # Cifar10 dataset has 10 different classes

# Initialize hyper-parameters
learning_rate = 0.01  # You will be later asked to experiment with
different learning rates and report results
num_epochs_total = 1000  # Total number of epochs to train the
classifier
epochs_per_evaluation = 10  # Epochs per step of evaluation; We will
evaluate our model regularly during training
N, D = dataset[
    "x_train"
].shape  # Get training data shape, N: Number of examples,
D:Dimensionality of the data
weight_decay = 0.00002

x_train = dataset["x_train"].copy()
y_train = dataset["y_train"].copy()
x_val = dataset["x_val"].copy()
y_val = dataset["y_val"].copy()
x_test = dataset["x_test"].copy()
```

```
y_test = dataset["y_test"].copy()

# Insert additional scalar term 1 in the samples to account for the
bias as discussed in class
x_train = np.insert(x_train, D, values=1, axis=1)
x_val = np.insert(x_val, D, values=1, axis=1)
x_test = np.insert(x_test, D, values=1, axis=1)

# Training and evaluation function -> Outputs accuracy data
def train(learning_rate_, weight_decay_):
    # Create a linear regression object
    logistic_regression = Logistic(
        num_classes, learning_rate_, epochs_per_evaluation,
weight_decay_
    )

    # Randomly initialize the weights and biases
    weights = np.random.randn(num_classes, D + 1) * 0.0001

    train_accuracies, val_accuracies, test_accuracies = [], [], []

    # Train the classifier
    for _ in range(int(num_epochs_total / epochs_per_evaluation)):
        # Train the classifier on the training data
        weights = logistic_regression.train(x_train, y_train, weights)

        # Evaluate the trained classifier on the training dataset
        y_pred_train = logistic_regression.predict(x_train)

train_accuracies.append(get_classification_accuracy(y_pred_train,
y_train))

        # Evaluate the trained classifier on the validation dataset
        y_pred_val = logistic_regression.predict(x_val)
        val_accuracies.append(get_classification_accuracy(y_pred_val,
y_val))

        # Evaluate the trained classifier on the test dataset
        y_pred_test = logistic_regression.predict(x_test)

test_accuracies.append(get_classification_accuracy(y_pred_test,
y_test))

    return train_accuracies, val_accuracies, test_accuracies, weights

import matplotlib.pyplot as plt


def plot_accuracies(train_acc, val_acc, test_acc):
```

```
    # Plot Accuracies vs Epochs graph for all the three
    epochs = np.arange(0, int(num_epochs_total /
epochs_per_evaluation))
    plt.ylabel("Accuracy")
    plt.xlabel("Epoch/10")
    plt.plot(epochs, train_acc, epochs, val_acc, epochs, test_acc)
    plt.legend(["Training", "Validation", "Testing"])
    plt.show()

# Run training and plotting for default parameter values as mentioned
above
%load_ext autoreload
%autoreload 2
t_ac, v_ac, te_ac, weights = train(learning_rate, weight_decay)

plot_accuracies(t_ac, v_ac, te_ac)
```



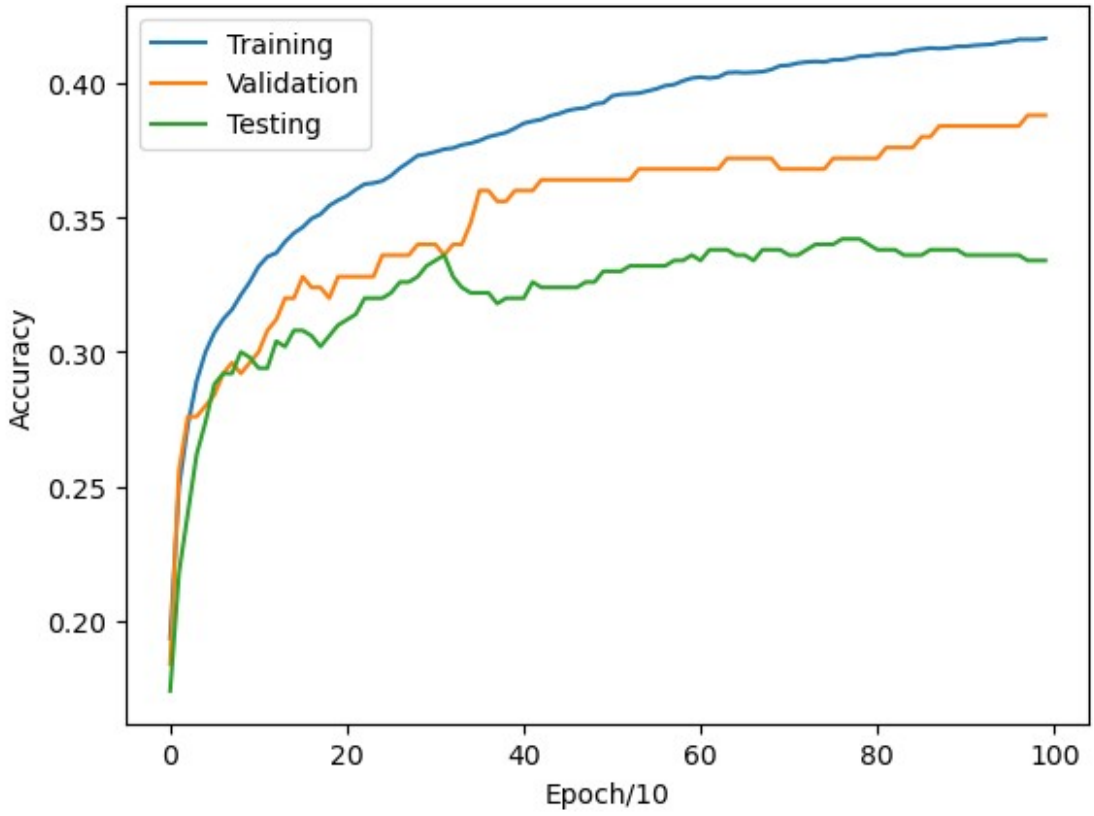

```
print(learning_rate)
```

```
0.01
```

Try different learning rates and plot graphs for all (20%)

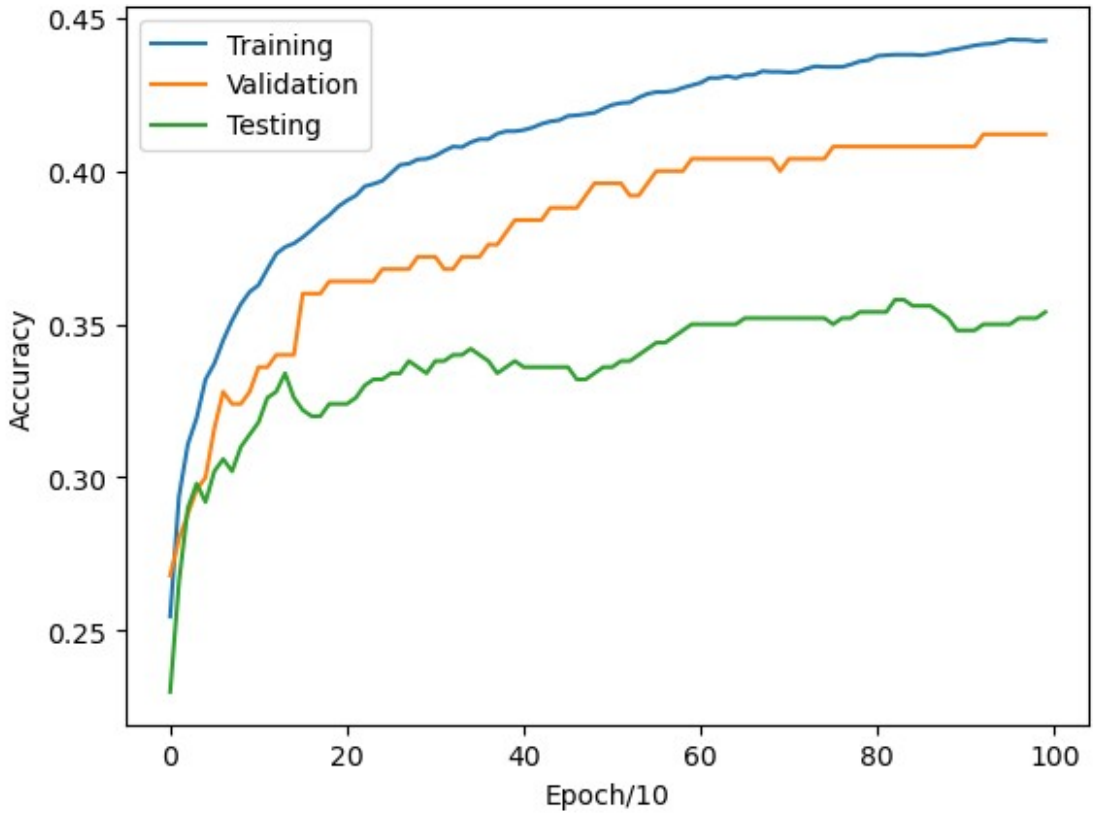
```
# Initialize the best values
best_weights = weights
best_learning_rate = learning_rate
best_weight_decay = weight_decay
```

```
# TODO
# Repeat the above training and evaluation steps for the following
learning rates and plot graphs
# You need to try 3 learning rates and submit all 3 graphs along with
this notebook pdf to show your learning rate experiments
learning_rates = [0.009, 0.02, 0.03]
weight_decay = 0.0  # No regularization for now

# FEEL FREE TO EXPERIMENT WITH OTHER VALUES. REPORT OTHER VALUES IF
THEY ACHIEVE A BETTER PERFORMANCE

# for lr in learning_rates: Train the classifier and plot data
# Step 1. train_accu, val_accu, test_accu = train(lr, weight_decay)
# Step 2. plot_accuracies(train_accu, val_accu, test_accu)

for learning_rate in learning_rates:
    # TODO: Train the classifier with different learning rates and
plot
    t_ac, v_ac, te_ac, weights = train(learning_rate, weight_decay)
    plot_accuracies(t_ac, v_ac, te_ac)
    print(te_ac)
```

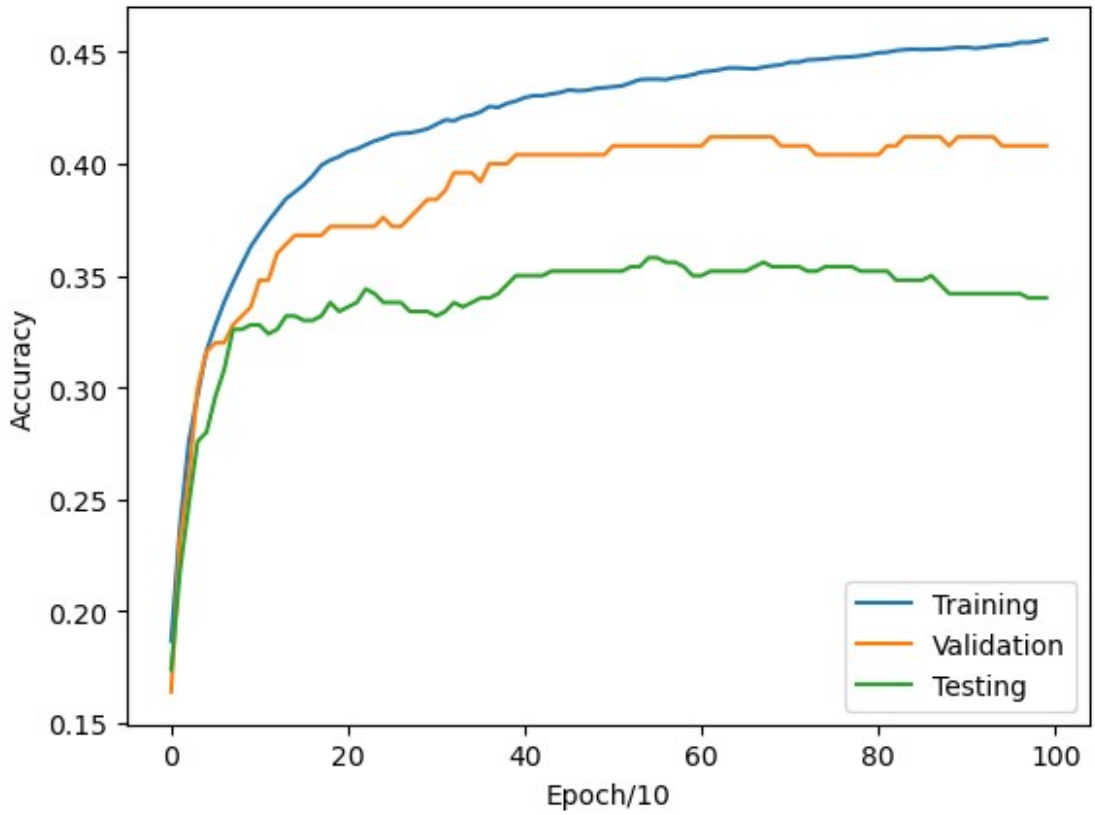



```
[0.174, 0.218, 0.24, 0.262, 0.274, 0.288, 0.292, 0.292, 0.3, 0.298,
0.294, 0.294, 0.304, 0.302, 0.308, 0.308, 0.306, 0.302, 0.306, 0.31,
```

0.312, 0.314, 0.32, 0.32, 0.32, 0.322, 0.326, 0.326, 0.328, 0.332,
0.334, 0.336, 0.328, 0.324, 0.322, 0.322, 0.322, 0.318, 0.32, 0.32,
0.32, 0.326, 0.324, 0.324, 0.324, 0.324, 0.324, 0.326, 0.326, 0.33,
0.33, 0.33, 0.332, 0.332, 0.332, 0.332, 0.332, 0.334, 0.334, 0.336,
0.334, 0.338, 0.338, 0.338, 0.336, 0.336, 0.334, 0.338, 0.338, 0.338,
0.336, 0.336, 0.338, 0.34, 0.34, 0.34, 0.342, 0.342, 0.342, 0.34,
0.338, 0.338, 0.338, 0.336, 0.336, 0.336, 0.338, 0.338, 0.338, 0.338,
0.336, 0.336, 0.336, 0.336, 0.336, 0.336, 0.336, 0.334, 0.334, 0.334]



[0.23, 0.266, 0.29, 0.298, 0.292, 0.302, 0.306, 0.302, 0.31, 0.314,
0.318, 0.326, 0.328, 0.334, 0.326, 0.322, 0.32, 0.32, 0.324, 0.324,
0.324, 0.326, 0.33, 0.332, 0.332, 0.334, 0.334, 0.338, 0.336, 0.334,
0.338, 0.338, 0.34, 0.34, 0.342, 0.34, 0.338, 0.334, 0.336, 0.338,
0.336, 0.336, 0.336, 0.336, 0.336, 0.336, 0.332, 0.332, 0.334, 0.336,
0.336, 0.338, 0.338, 0.34, 0.342, 0.344, 0.344, 0.346, 0.348, 0.35,
0.35, 0.35, 0.35, 0.35, 0.35, 0.352, 0.352, 0.352, 0.352, 0.352,
0.352, 0.352, 0.352, 0.352, 0.352, 0.35, 0.352, 0.352, 0.354, 0.354,
0.354, 0.354, 0.358, 0.358, 0.356, 0.356, 0.356, 0.354, 0.352, 0.348,
0.348, 0.348, 0.35, 0.35, 0.35, 0.35, 0.352, 0.352, 0.352, 0.354]




```
[0.174, 0.218, 0.248, 0.276, 0.28, 0.296, 0.308, 0.326, 0.326, 0.328,
0.328, 0.324, 0.326, 0.332, 0.332, 0.33, 0.33, 0.332, 0.338, 0.334,
0.336, 0.338, 0.344, 0.342, 0.338, 0.338, 0.338, 0.334, 0.334, 0.334,
0.332, 0.334, 0.338, 0.336, 0.338, 0.34, 0.34, 0.342, 0.346, 0.35,
0.35, 0.35, 0.35, 0.352, 0.352, 0.352, 0.352, 0.352, 0.352, 0.352,
0.352, 0.352, 0.354, 0.354, 0.358, 0.358, 0.356, 0.356, 0.354, 0.35,
0.35, 0.352, 0.352, 0.352, 0.352, 0.352, 0.354, 0.356, 0.354, 0.354,
0.354, 0.354, 0.352, 0.352, 0.354, 0.354, 0.354, 0.354, 0.352, 0.352,
0.352, 0.352, 0.348, 0.348, 0.348, 0.348, 0.35, 0.346, 0.342, 0.342,
0.342, 0.342, 0.342, 0.342, 0.342, 0.342, 0.342, 0.34, 0.34, 0.34]
```

Inline Question 1.

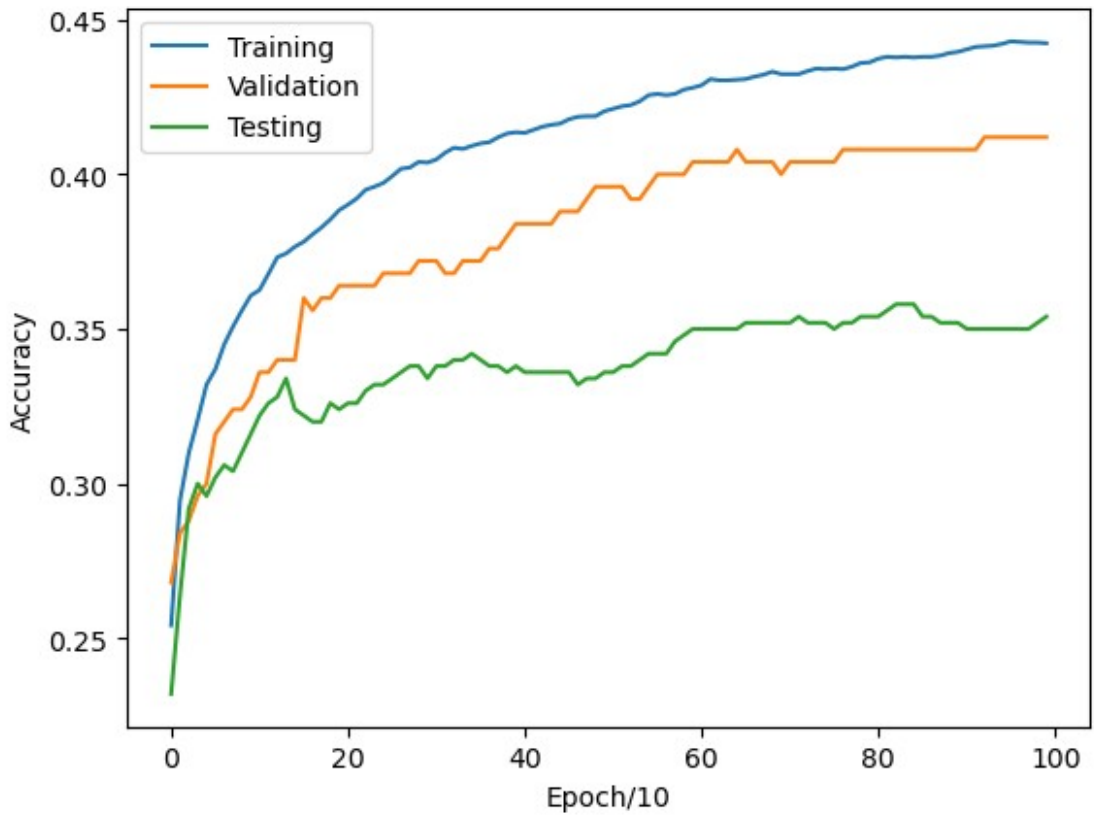
Which one of these learning rates (best_lr) would you pick to train your model? Please Explain why.

Your Answer:

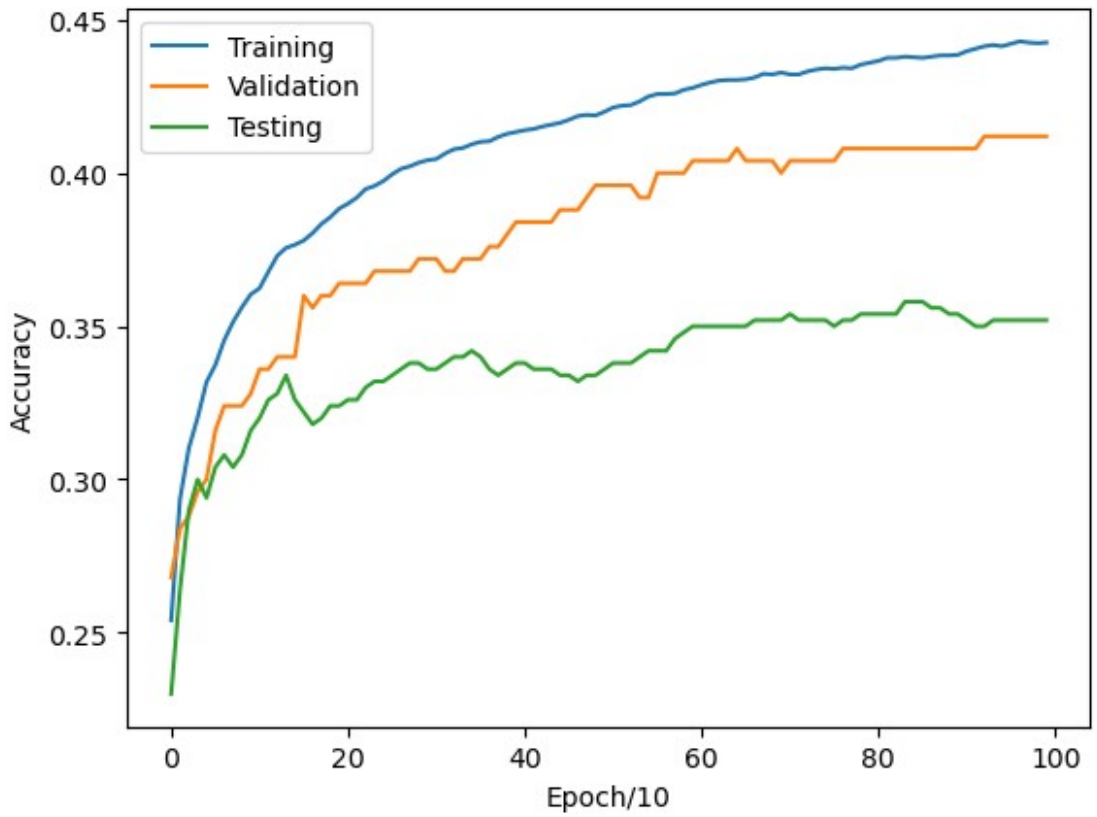
I tried with [0.009, 0.02, 0.03] as the learning rates. I picked the best learning rate as 0.02. This is because I am getting the maximum test accuracy of 0.354 on the test dataset. The learning rate of 0.009 becomes small which results in slow convergence thereby resulting in a drop in the test accuracy. One the other hand, The learning rate of 0.03 becomes too large, thereby decreasing the test accuracy.

Regularization: Try different weight decay and plots graphs for all (20%)

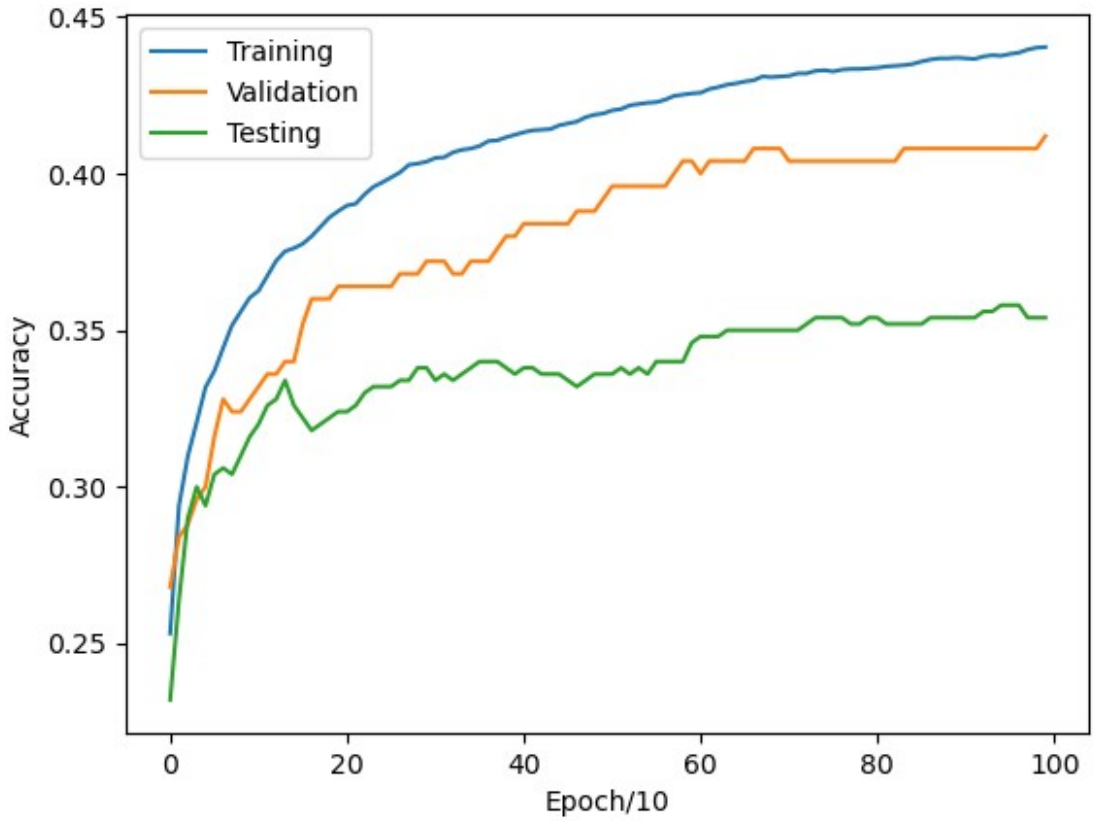
```
# Initialize a non-zero weight_decay (Regulzarization constant) term
and repeat the training and evaluation
# Use the best learning rate as obtained from the above excercise,
best_lr

# You need to try 3 learning rates and submit all 3 graphs along with
this notebook pdf to show your weight decay experiments
weight_decays = [0.0001, 0.001, 0.01, 0.1, 1, 10, 50]
best_learning_rate = 0.02
# FEEL FREE TO EXPERIMENT WITH OTHER VALUES. REPORT OTHER VALUES IF
THEY ACHIEVE A BETTER PERFORMANCE

# for weight_decay in weight_decays: Train the classifier and plot
data
# Step 1. train_accu, val_accu, test_accu = train(best_lr,
weight_decay)
# Step 2. plot_accuracies(train_accu, val_accu, test_accu)

for weight_decay in weight_decays:
    # TODO: Train the classifier with different weight decay and plot
    t_ac, v_ac, te_ac, weights = train(best_learning_rate,
weight_decay)
    plot_accuracies(t_ac, v_ac, te_ac)
    print(te_ac)
```



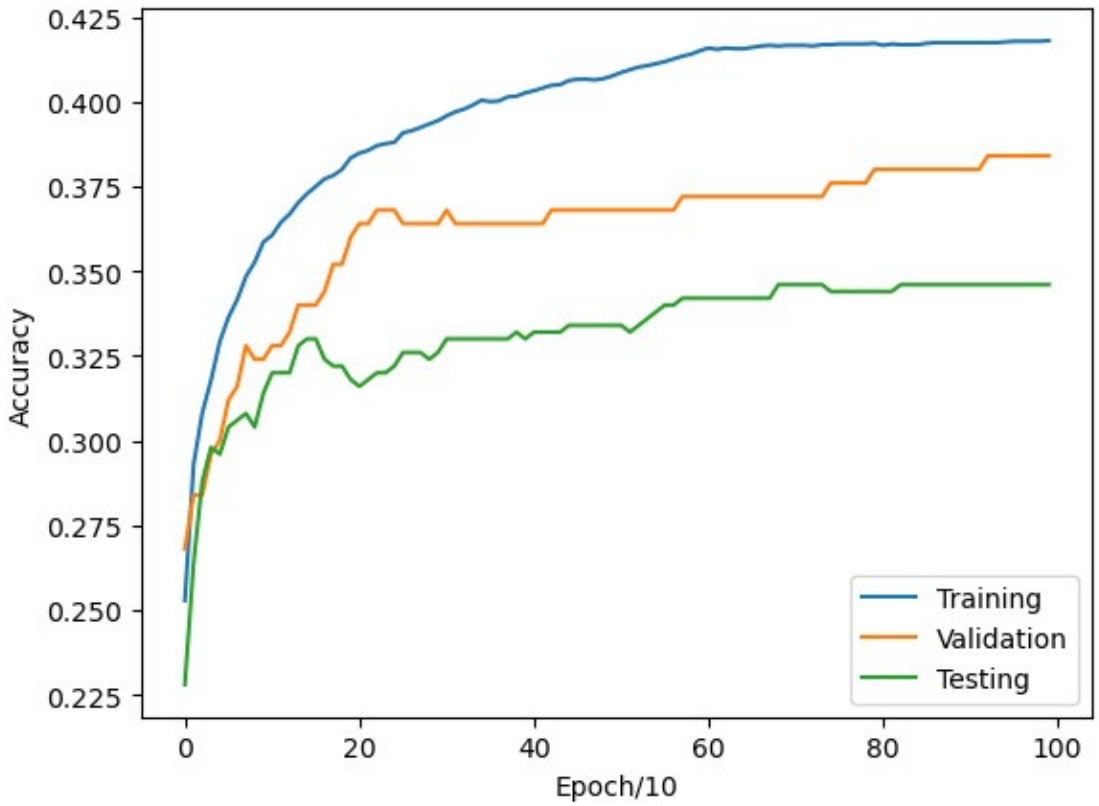

```
[0.232, 0.264, 0.292, 0.3, 0.296, 0.302, 0.306, 0.304, 0.31, 0.316,
0.322, 0.326, 0.328, 0.334, 0.324, 0.322, 0.32, 0.32, 0.326, 0.324,
0.326, 0.326, 0.33, 0.332, 0.332, 0.334, 0.336, 0.338, 0.338, 0.334,
0.338, 0.338, 0.34, 0.34, 0.342, 0.34, 0.338, 0.338, 0.336, 0.338,
0.336, 0.336, 0.336, 0.336, 0.336, 0.336, 0.332, 0.334, 0.334, 0.336,
0.336, 0.338, 0.338, 0.34, 0.342, 0.342, 0.342, 0.346, 0.348, 0.35,
0.35, 0.35, 0.35, 0.35, 0.35, 0.352, 0.352, 0.352, 0.352, 0.352,
0.352, 0.354, 0.352, 0.352, 0.352, 0.35, 0.352, 0.352, 0.354, 0.354,
0.354, 0.356, 0.358, 0.358, 0.358, 0.354, 0.354, 0.352, 0.352, 0.352,
0.35, 0.35, 0.35, 0.35, 0.35, 0.35, 0.35, 0.35, 0.352, 0.354]
```



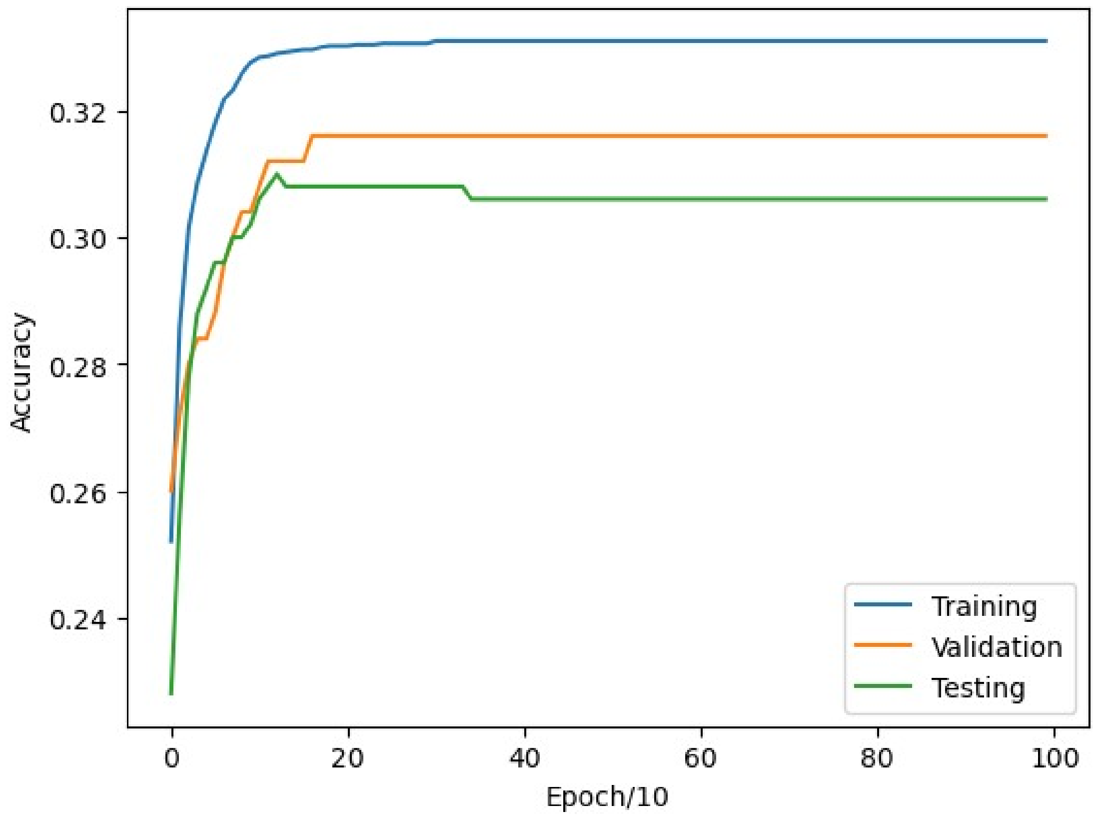

```
[0.23, 0.264, 0.29, 0.3, 0.294, 0.304, 0.308, 0.304, 0.308, 0.316,
0.32, 0.326, 0.328, 0.334, 0.326, 0.322, 0.318, 0.32, 0.324, 0.324,
0.326, 0.326, 0.33, 0.332, 0.332, 0.334, 0.336, 0.338, 0.338, 0.336,
0.336, 0.338, 0.34, 0.34, 0.342, 0.34, 0.336, 0.334, 0.336, 0.338,
0.338, 0.336, 0.336, 0.336, 0.334, 0.334, 0.332, 0.334, 0.334, 0.336,
0.338, 0.338, 0.338, 0.34, 0.342, 0.342, 0.342, 0.346, 0.348, 0.35,
0.35, 0.35, 0.35, 0.35, 0.35, 0.35, 0.352, 0.352, 0.352, 0.352, 0.354,
0.352, 0.352, 0.352, 0.352, 0.35, 0.352, 0.352, 0.354, 0.354, 0.354,
0.354, 0.354, 0.358, 0.358, 0.358, 0.356, 0.356, 0.354, 0.354, 0.352,
0.35, 0.35, 0.352, 0.352, 0.352, 0.352, 0.352, 0.352, 0.352]
```



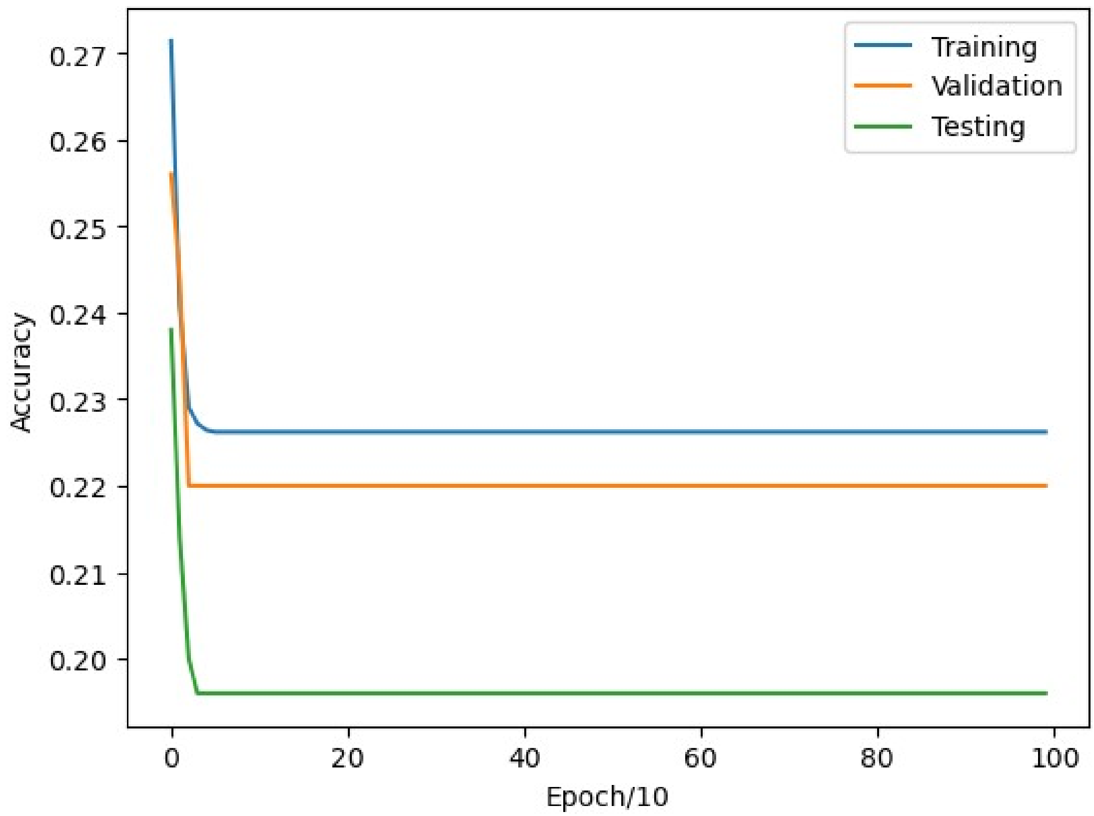

```
[0.232, 0.264, 0.29, 0.3, 0.294, 0.304, 0.306, 0.304, 0.31, 0.316,
0.32, 0.326, 0.328, 0.334, 0.326, 0.322, 0.318, 0.32, 0.322, 0.324,
0.324, 0.326, 0.33, 0.332, 0.332, 0.332, 0.334, 0.334, 0.338, 0.338,
0.334, 0.336, 0.334, 0.336, 0.338, 0.34, 0.34, 0.34, 0.338, 0.336,
0.338, 0.338, 0.336, 0.336, 0.336, 0.334, 0.332, 0.334, 0.336, 0.336,
0.336, 0.338, 0.336, 0.338, 0.336, 0.34, 0.34, 0.34, 0.34, 0.346,
0.348, 0.348, 0.348, 0.35, 0.35, 0.35, 0.35, 0.35, 0.35, 0.35, 0.35,
0.35, 0.352, 0.354, 0.354, 0.354, 0.354, 0.352, 0.352, 0.354, 0.354,
0.352, 0.352, 0.352, 0.352, 0.352, 0.354, 0.354, 0.354, 0.354, 0.354,
0.354, 0.356, 0.356, 0.358, 0.358, 0.358, 0.354, 0.354, 0.354]
```



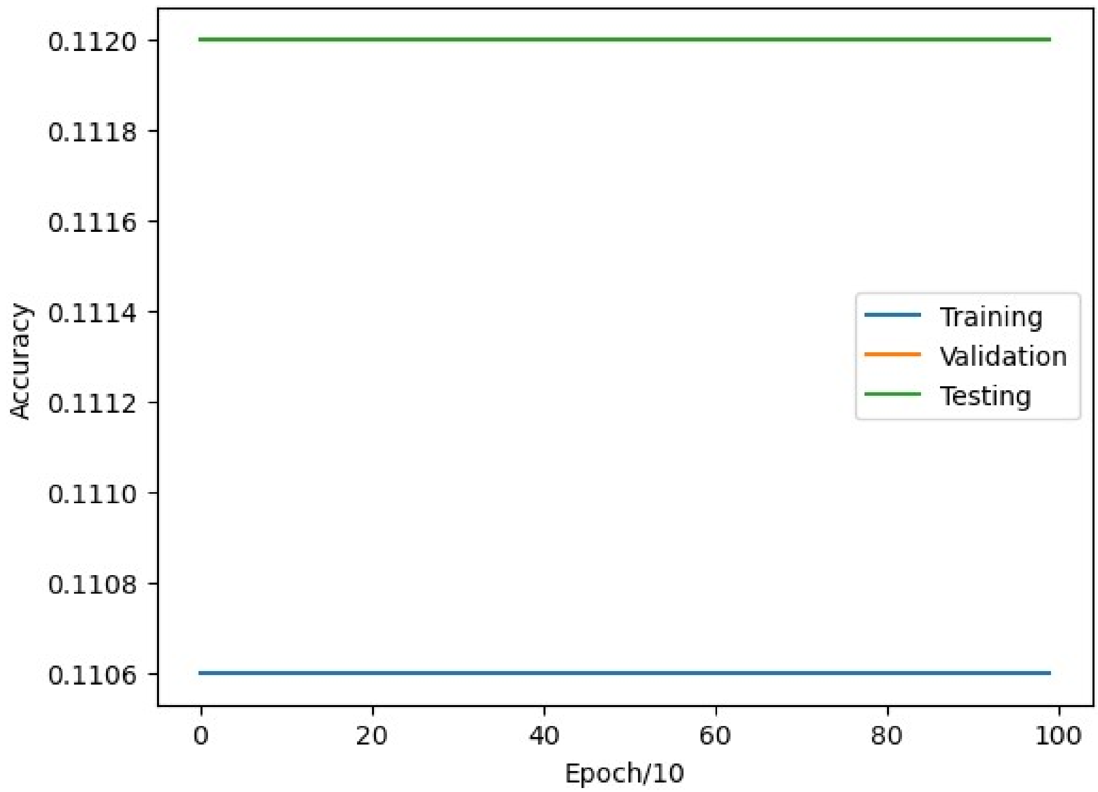

```
[0.228, 0.264, 0.288, 0.298, 0.296, 0.304, 0.306, 0.308, 0.304, 0.314,
0.32, 0.32, 0.32, 0.328, 0.33, 0.33, 0.324, 0.322, 0.322, 0.318,
0.316, 0.318, 0.32, 0.32, 0.322, 0.326, 0.326, 0.326, 0.324, 0.326,
0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.332, 0.33, 0.332,
0.332, 0.332, 0.332, 0.334, 0.334, 0.334, 0.334, 0.334, 0.334, 0.334,
0.332, 0.334, 0.336, 0.338, 0.34, 0.34, 0.342, 0.342, 0.342, 0.342,
0.342, 0.342, 0.342, 0.342, 0.342, 0.342, 0.342, 0.346, 0.346, 0.346,
0.346, 0.346, 0.346, 0.344, 0.344, 0.344, 0.344, 0.344, 0.344, 0.344,
0.344, 0.346, 0.346, 0.346, 0.346, 0.346, 0.346, 0.346, 0.346, 0.346,
0.346, 0.346, 0.346, 0.346, 0.346, 0.346, 0.346, 0.346, 0.346]
```



[0.228, 0.256, 0.278, 0.288, 0.292, 0.296, 0.296, 0.3, 0.3, 0.302,
0.306, 0.308, 0.31, 0.308, 0.308, 0.308, 0.308, 0.308, 0.308, 0.308,
0.308, 0.308, 0.308, 0.308, 0.308, 0.308, 0.308, 0.308, 0.308, 0.308,
0.308, 0.308, 0.308, 0.308, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306,
0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306,
0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306,
0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306,
0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306,
0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306,
0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306, 0.306]




```
[0.238, 0.214, 0.2, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196,
0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196,
0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196,
0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196,
0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196,
0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196,
0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196,
0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196,
0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196,
0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196, 0.196]
```




```
[0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112,
0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112,
0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112,
0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112,
0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112,
0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112,
0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112,
0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112,
0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112,
0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112, 0.112]
```

Inline Question 2.

Discuss underfitting and overfitting as observed in the 5 graphs obtained by changing the regularization. Which weight_decay term gave you the best classifier performance? HINT: Do not just think in terms of best training set performance, keep in mind that the real utility of a machine learning model is when it performs well on data it has never seen before

Your Answer:

- Overfitting happens when the weight_decay is too small as the regularization term becomes negligible.
- Underfitting happens when the weight_decay is too large as the regularization term dominates the loss function. I tried weight decays of [0.0001, 0.001, 0.01, 0.1, 1, 10, 50].

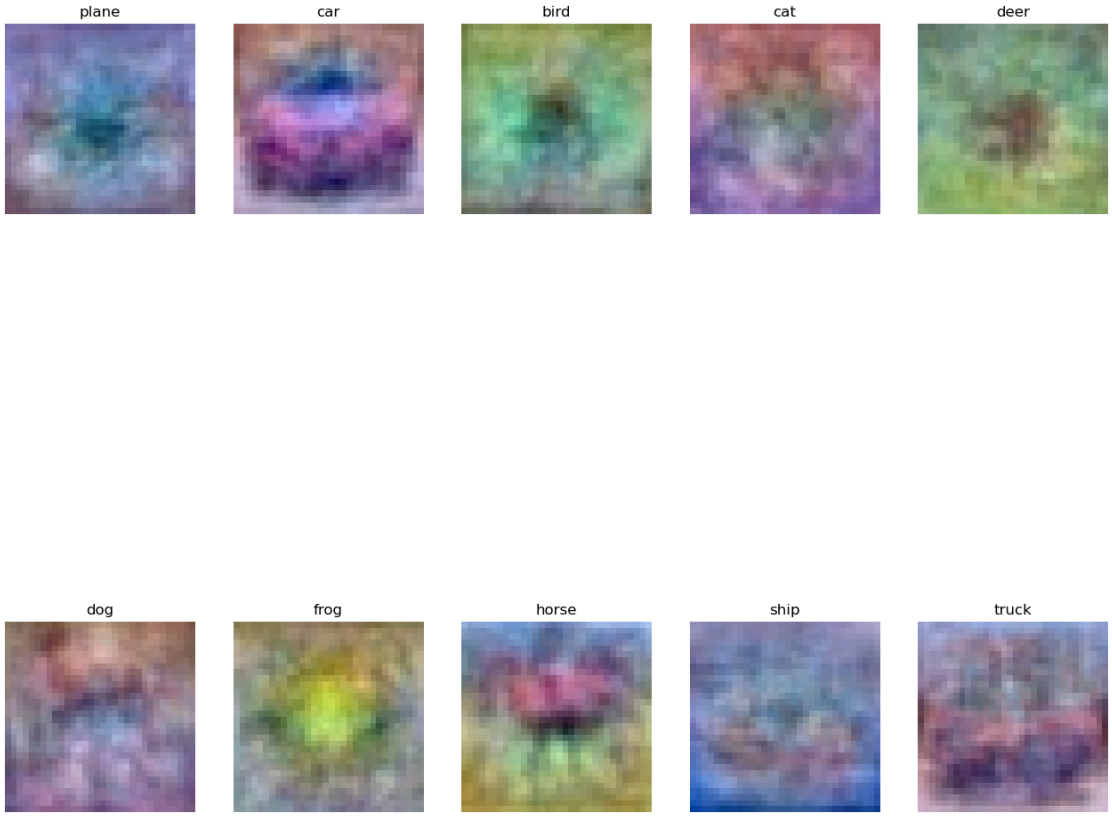
- I picked 0.01 as the best weight_decay term because I am getting the best test classification accuracy of 0.354 in this case.
- The model suffers from overfitting in k = [0.0001, 0.001, 0.01, 0.1, 1] as the training accuracies are high but the test accuracies are low.
- The model with k=1 is optimal as the training and test accuracies are close to each other.
- The model suffers from underfitting in k = [10, 50] as both the training and test accuracies are low.

Visualize the filters (10%)

```
# These visualizations will only somewhat make sense if your learning
rate and weight_decay parameters were
# properly chosen in the model. Do your best.

# TODO: Run this cell and Show filter visualizations for the best set
of weights you obtain.
# Report the 2 hyperparameters you used to obtain the best model.

# NOTE: You need to set `best_learning_rate` and `best_weight_decay`
to the values that gave the highest accuracy
best_learning_rate = 0.02
best_weight_decay = 0.01
print("Best LR:", best_learning_rate)
print("Best Weight Decay:", best_weight_decay)

# NOTE: You need to set `best_weights` to the weights with the highest
accuracy
w = best_weights[:, :-1]
w = w.reshape(10, 3, 32, 32).transpose(0, 2, 3, 1)

w_min, w_max = np.min(w), np.max(w)

fig = plt.figure(figsize=(16, 16))
classes = [
    "plane",
    "car",
    "bird",
    "cat",
    "deer",
    "dog",
    "frog",
    "horse",
    "ship",
    "truck",
]
for i in range(10):
    fig.add_subplot(2, 5, i + 1)
```

```
    # Rescale the weights to be between 0 and 255
    wimg = 255.0 * (w[i, :, :, :].squeeze() - w_min) / (w_max - w_min)
    plt.imshow(wimg.astype(int))
    plt.axis("off")
    plt.title(classes[i])
plt.show()
```

```
Best LR: 0.02
Best Weight Decay: 0.01
```



Inline Question 3. (10%)

a. Compare and contrast the performance of the 2 classifiers i.e. Linear Regression and Logistic Regression.

b. Which classifier would you deploy for your multiclass classification project and why?

Your Answer:

1. Linear Regression achieves better test accuracy than Logistic Regression. Also, we see that Linear Regression converges faster.
2. I would deploy linear regression for multiclass classification as it achieves better test accuracy compared to the linear regression. Therefore, I expect logistic regression to perform better on real world unseen data.

neural_network

April 28, 2023

1 ECE285 Assignment 1: Neural Network in NumPy

Use this notebook to build your neural network by implementing the following functions in the python files under `ece285/algorithms` directory:

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

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

TO SUBMIT: PDF of this notebook with all the required outputs and answers.

[1]:
```
# Setup
import matplotlib.pyplot as plt
import numpy as np

from ece285.layers.sequential import Sequential
from ece285.layers.linear import Linear
from ece285.layers.relu import ReLU
from ece285.layers.softmax import Softmax
from ece285.layers.loss_func import CrossEntropyLoss
from ece285.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-ipython
%load_ext autoreload
%autoreload 2
```

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.

[2]:
```
# 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()
    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()
```

1.0.1 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

[7]:
```
%autoreload 2
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:
8.799388540037256e-08
```

1.0.2 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.

[8]:
```
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)))
```

```
input_x shape (10, 3)
target_y shape (10,)
1.098612723362578
Difference between your loss and correct loss:
0.0
```

1.0.3 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.

We check the values for these gradients by calculating the difference, it is expected to get difference $<$ 1e-8.

[14]:
```
%autoreload 2
# 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 parameters
# (4, 10) -> Layer 1 W
# (10,)   -> Layer 1 b
# (10, 3) -> Layer 2 W
# (3,)    -> Layer 2 b
```

```
(4, 10)
```

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

[15]:
```
# No need to edit anything in this block ( 20% of the above 40% )
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,
        6.08849787e-07,
        -1.92177796e-06,
        3.92085824e-06,
        -5.40772636e-06,
        2.93354593e-06,
        -3.14568138e-06,
        5.27501592e-11,
    ]
)

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] - grad_b1))
    + np.sum(np.abs(gradients[2] - grad_w2))
    + np.sum(np.abs(gradients[3] - grad_b2))
```

```
)
print("Difference in Gradient values", difference)
```

```
Difference in Gradient values 7.70191643436727e-09
```

1.1 Train the complete network on the toy data. (30%)

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 training loss decrease below 0.01.

[16]:
```
# Training Procedure
# Initialize the optimizer. DO NOT change any of the hyper-parameters here or␣
 ↪above.
# We have implemented the SGD optimizer class for you here, which visits each␣
 ↪layer sequentially to
# get the gradients and optimize the respective parameters.
# You should work with the given parameters and only edit your implementation␣
 ↪in the .py files

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 labels
    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 layers
    net.backward(Loss.backward())
    # call the step function in the optimizer to update the values of the␣
 ↪params with the gradients
    optim.step()
    # Append the loss at each iteration
    epoch_loss.append(loss)

    if (epoch + 1) % 50 == 0:
        print("Epoch {}, loss={:3f}".format(epoch + 1, epoch_loss[-1]))
```

```
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 50, loss=0.832706
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 100, loss=0.454687
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 150, loss=0.118350
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 200, loss=0.055911
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 250, loss=0.038039
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 300, loss=0.029528
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 350, loss=0.024400
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 400, loss=0.020819
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 450, loss=0.017947
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
Epoch 500, loss=0.015866
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 550, loss=0.014198
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 600, loss=0.012916
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 650, loss=0.011859
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 700, loss=0.010943
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 750, loss=0.010198
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 800, loss=0.009540
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 850, loss=0.008970
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 900, loss=0.008454
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
```

```
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
Epoch 950, loss=0.008003
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
input_x shape (10, 3)
target_y shape (10,)
```

```
input_x shape (10, 3)
target_y shape (10,)
Epoch 1000, loss=0.007593
```

[17]:
```
# Test your predictions. The predictions must match the labels
print(net.predict(X))
print(y)
```

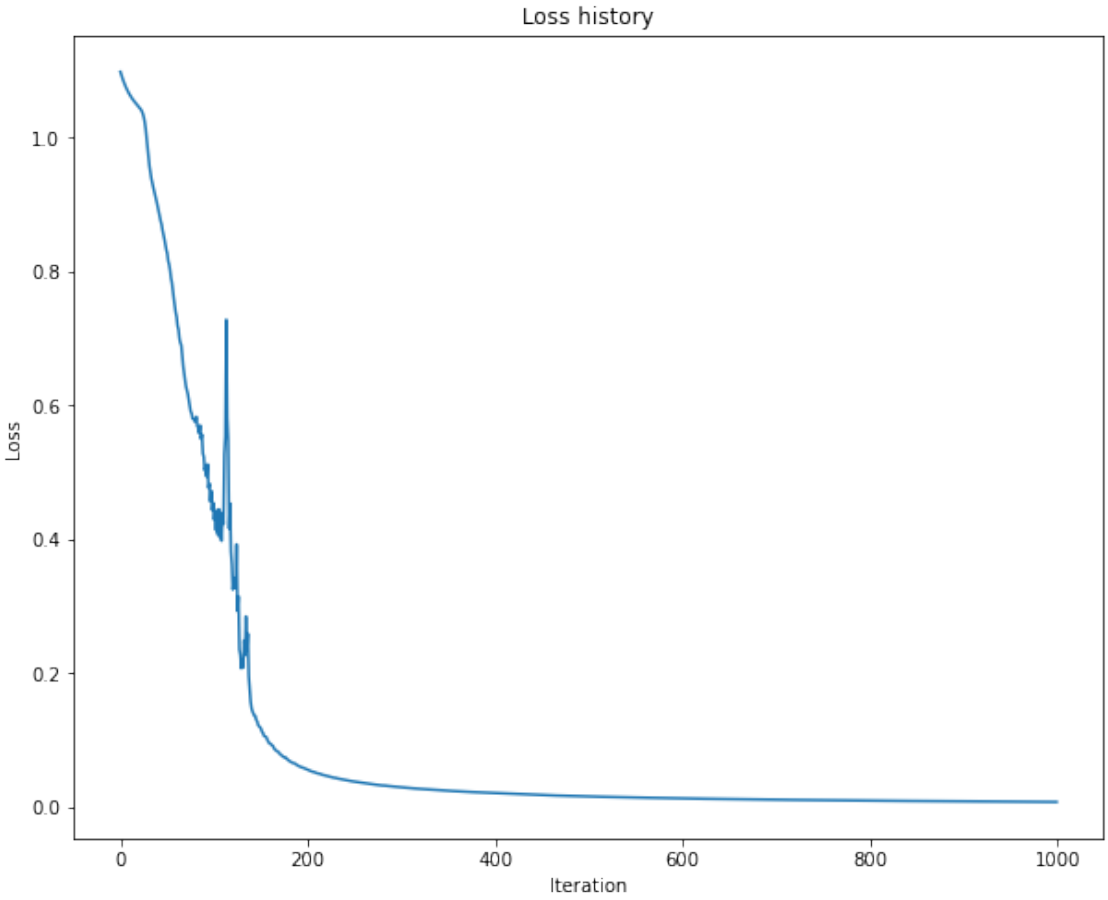
```
[2 1 0 1 2 0 0 2 0 0]
[2 1 0 1 2 0 0 2 0 0]
```

[18]:
```
# 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 0.007593419801731 28
```

[19]:
```
# Plot the training loss curve. The loss in the curve should be decreasing (20%)
plt.plot(epoch_loss)
plt.title("Loss history")
plt.xlabel("Iteration")
plt.ylabel("Loss")
```

[19]:
```
Text(0, 0.5, 'Loss')
```


Loss history
Loss
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
Iteration