linear_regression

April 20, 2022

1 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.

[1]:
```python
# 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,)
```

[2]:
```
x_train = dataset["x_train"]
y_train = dataset["y_train"]
x_val = dataset["x_val"]
y_val = dataset["y_val"]
x_test = dataset["x_test"]
y_test = dataset["y_test"]
```

[3]:
```
# 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
)
```

2 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.

2.0.1 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.

[4]:
```
# 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 = 0.00005
```

[5]:
```
# 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)
```

[9]:
```

```

[11]:
```
# 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
```

2.0.2 Plot the Accuracies vs epoch graphs

[12]:
```
import matplotlib.pyplot as plt



def plot_accuracies(train_acc, val_acc, test_acc, titles):
    # Plot Accuracies vs Epochs graph for all the three
    epochs = np.arange(0, int(num_epochs_total / epochs_per_evaluation))
    plt.title(titles)
    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()
```

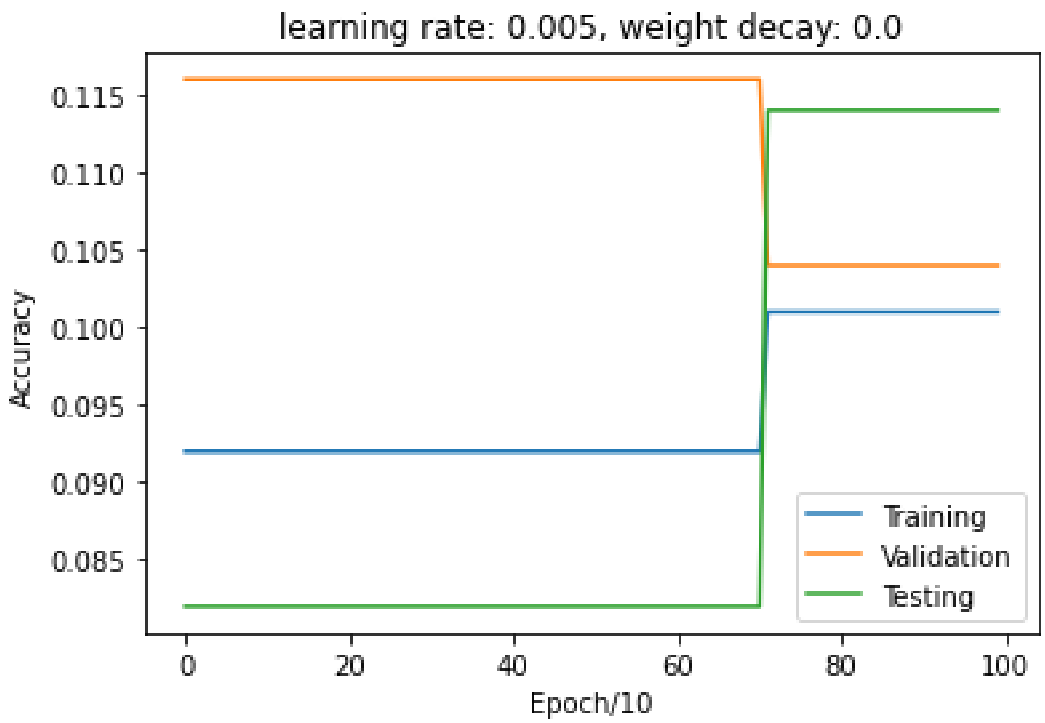
[13]:
```
# Run training and plotting for default parameter values as mentioned above
t_ac, v_ac, te_ac, weights = train(learning_rate, weight_decay)
```

[14]:
```
plot_accuracies(t_ac, v_ac, te_ac, "learning rate: 0.0001, weight decay: 0.
↪00005")
```

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

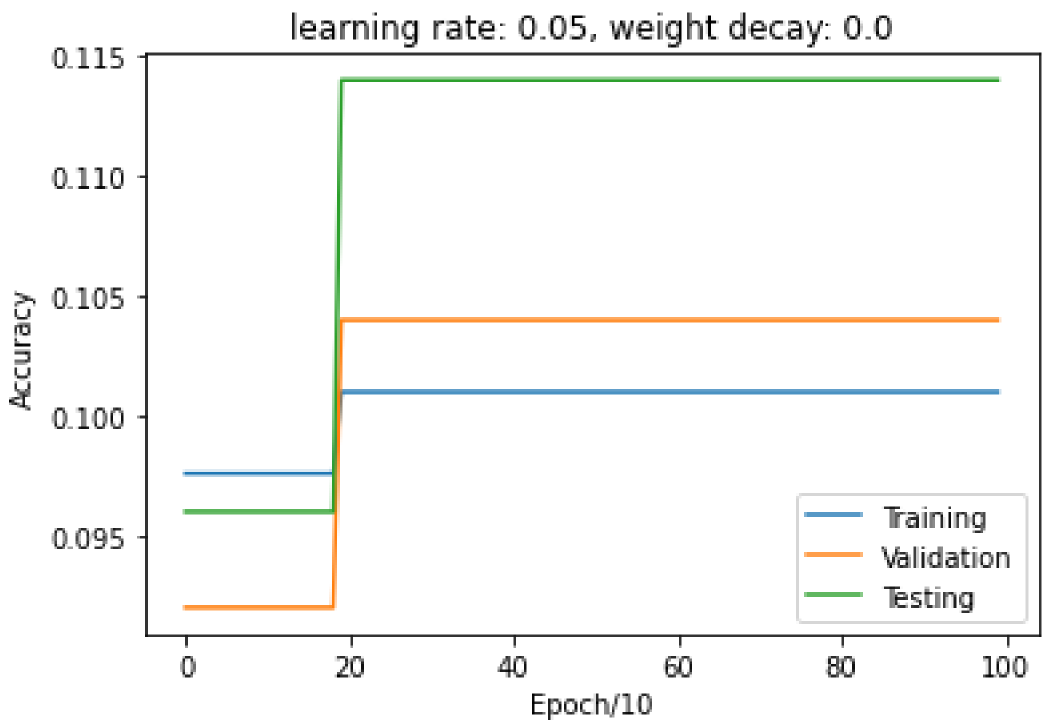
[15]:
```
# TODO
# Repeat the above training and evaluation steps for the following learning␣
↪rates and plot graphs
# You need to submit all 5 graphs along with this notebook pdf
learning_rates = [0.005, 0.05, 0.1, 0.5, 1.0]
weight_decay = 0.0  # No regularization for now
t_acs = []

# FEEL FREE TO EXPERIMENT WITH OTHER VALUES. REPORT OTHER VALUES IF THEY␣
↪ACHIEVE A BETTER PERFORMANCE
for i in range(len(learning_rates)):

    t_ac, v_ac, te_ac, weights = train(learning_rates[i], weight_decay)
    title = "learning rate: " + str(learning_rates[i]) + ", weight decay: " +␣
↪str(0.0)
    plot_accuracies(t_ac, v_ac, te_ac, title)
    t_acs.append(t_ac)

# 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)
```

```
C:\Users\hp\anaconda3\lib\site-packages\numpy\linalg\linalg.py:2556:
RuntimeWarning: overflow encountered in reduce
  return add.reduce(abs(x), axis=axis, keepdims=keepdims)
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:47: RuntimeWarning:
invalid value encountered in multiply
  self.weight_decay * w * np.linalg.norm(w, ord = 1)
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:46: RuntimeWarning:
overflow encountered in matmul
  dydw = 1/N * np.transpose(X_trains) @ (y_hat - X_trains @ w) +\
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:46: RuntimeWarning:
invalid value encountered in matmul
  dydw = 1/N * np.transpose(X_trains) @ (y_hat - X_trains @ w) +\
```

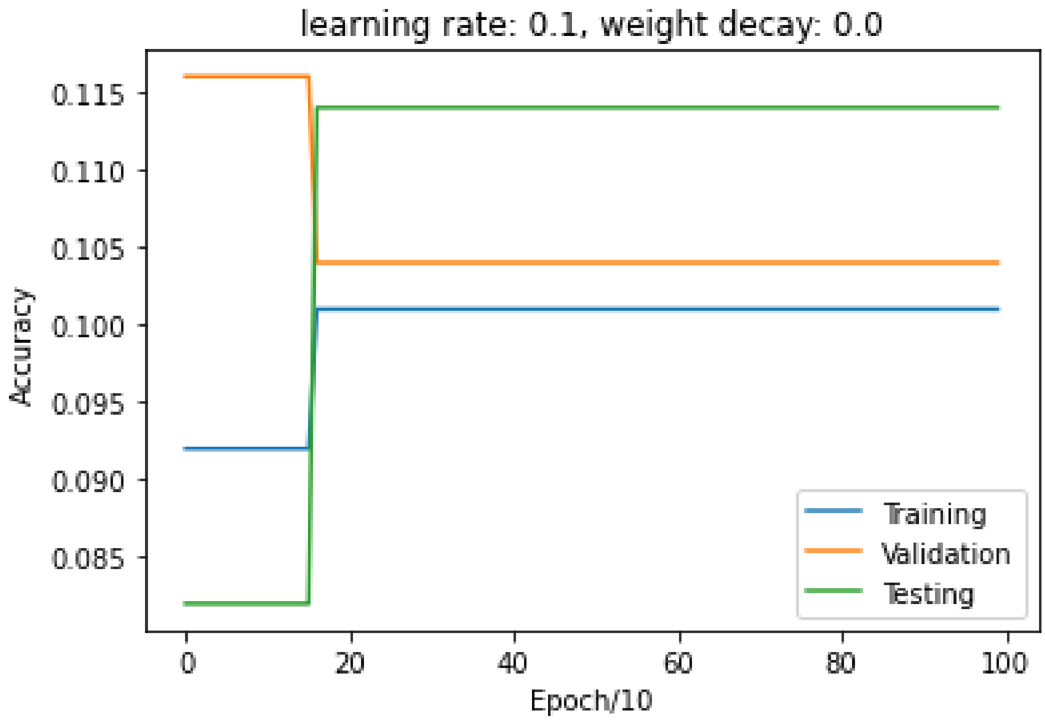



```
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:46: RuntimeWarning:
overflow encountered in matmul
  dydw = 1/N * np.transpose(X_trains) @ (y_hat - X_trains @ w) +\
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:46: RuntimeWarning:
invalid value encountered in matmul
  dydw = 1/N * np.transpose(X_trains) @ (y_hat - X_trains @ w) +\
C:\Users\hp\anaconda3\lib\site-packages\numpy\linalg\linalg.py:2556:
RuntimeWarning: overflow encountered in reduce
  return add.reduce(abs(x), axis=axis, keepdims=keepdims)
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:47: RuntimeWarning:
invalid value encountered in multiply
```

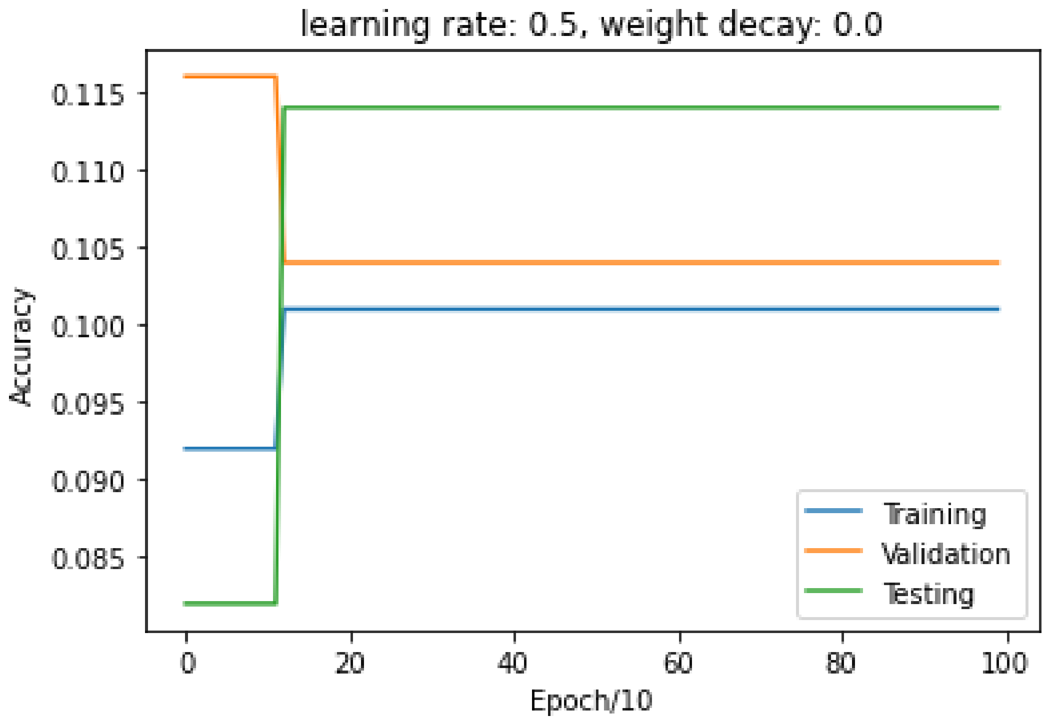
```
  self.weight_decay * w * np.linalg.norm(w, ord = 1)
```



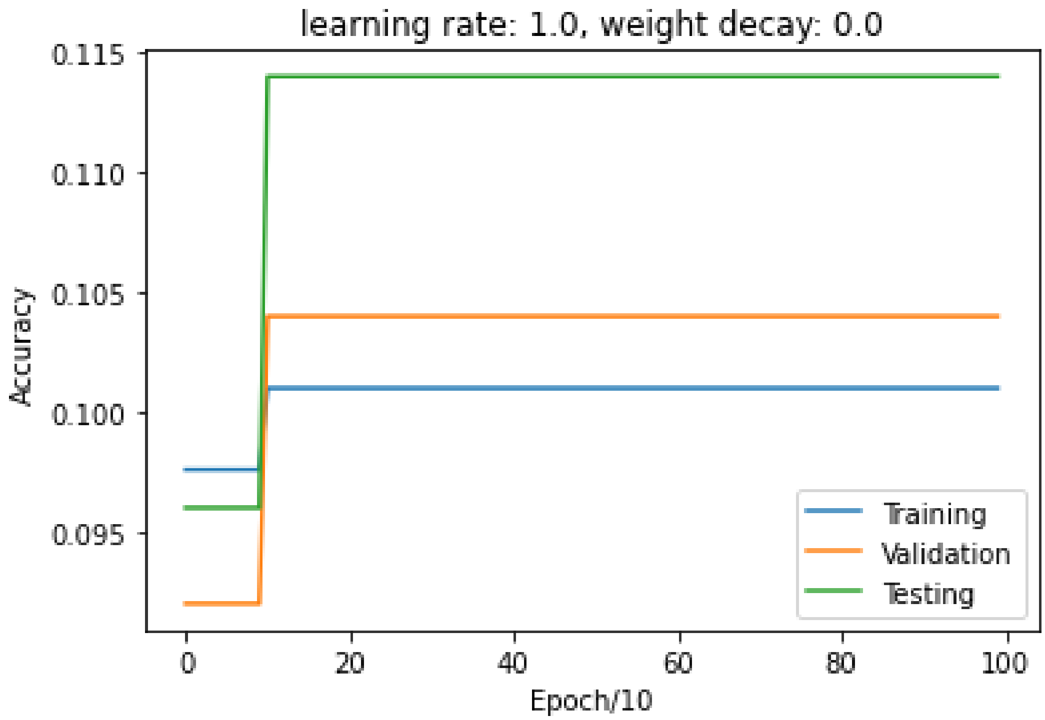

```
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:46: RuntimeWarning:
overflow encountered in matmul
  dydw = 1/N * np.transpose(X_trains) @ (y_hat - X_trains @ w) +\
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:46: RuntimeWarning:
invalid value encountered in matmul
  dydw = 1/N * np.transpose(X_trains) @ (y_hat - X_trains @ w) +\
C:\Users\hp\anaconda3\lib\site-packages\numpy\linalg\linalg.py:2556:
RuntimeWarning: overflow encountered in reduce
  return add.reduce(abs(x), axis=axis, keepdims=keepdims)
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:47: RuntimeWarning:
invalid value encountered in multiply
  self.weight_decay * w * np.linalg.norm(w, ord = 1)
```




```
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:46: RuntimeWarning:
overflow encountered in matmul
  dydw = 1/N * np.transpose(X_trains) @ (y_hat - X_trains @ w) +\
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:46: RuntimeWarning:
invalid value encountered in matmul
  dydw = 1/N * np.transpose(X_trains) @ (y_hat - X_trains @ w) +\
C:\Users\hp\anaconda3\lib\site-packages\numpy\linalg\linalg.py:2556:
RuntimeWarning: overflow encountered in reduce
  return add.reduce(abs(x), axis=axis, keepdims=keepdims)
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:47: RuntimeWarning:
invalid value encountered in multiply
  self.weight_decay * w * np.linalg.norm(w, ord = 1)
```




```
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:46: RuntimeWarning:
overflow encountered in matmul
  dydw = 1/N * np.transpose(X_trains) @ (y_hat - X_trains @ w) +\
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:46: RuntimeWarning:
invalid value encountered in matmul
  dydw = 1/N * np.transpose(X_trains) @ (y_hat - X_trains @ w) +\
C:\Users\hp\anaconda3\lib\site-packages\numpy\linalg\linalg.py:2556:
RuntimeWarning: overflow encountered in reduce
  return add.reduce(abs(x), axis=axis, keepdims=keepdims)
C:\Users\hp\AppData\Local\Temp/ipykernel_36284/3759777417.py:47: RuntimeWarning:
invalid value encountered in multiply
  self.weight_decay * w * np.linalg.norm(w, ord = 1)
```



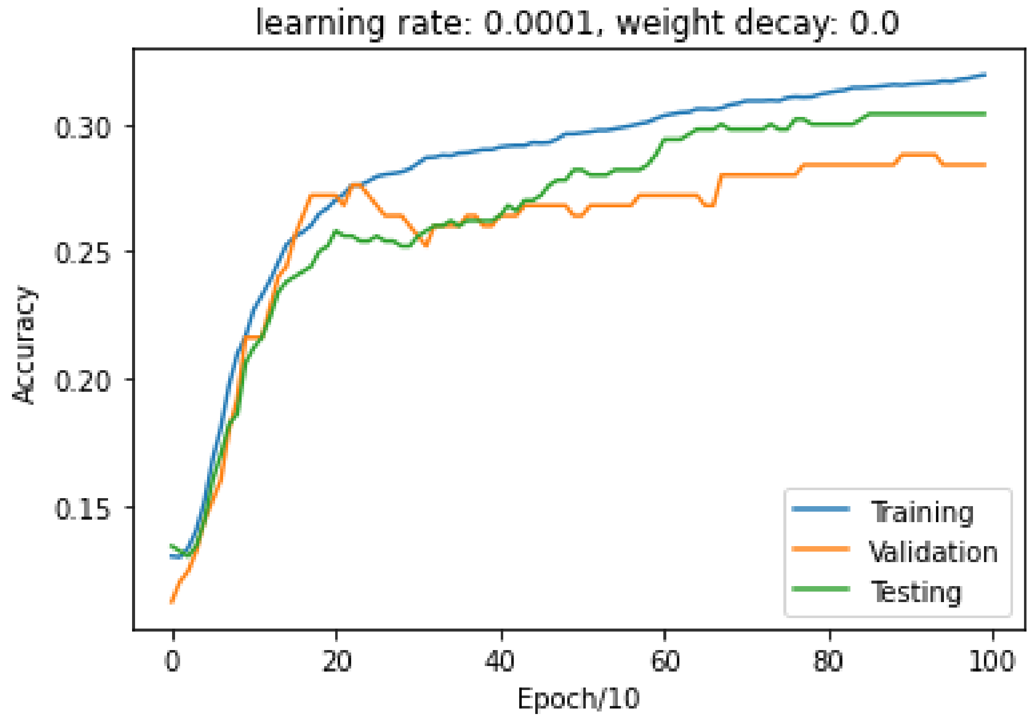
Inline Question 1. Which one of these learning rates (best_lr) would you pick to train your model? Please Explain why.

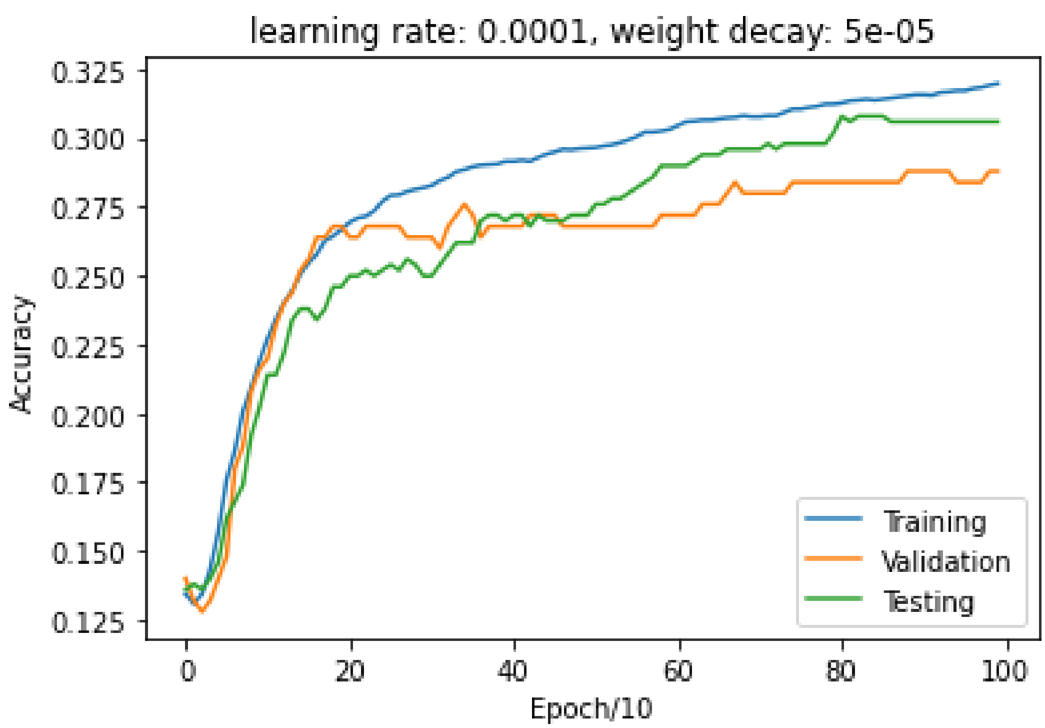
Your Answer: the optimal learning rate is 0.0001. When selected a large learning rate, Linear regression will crash due to overflow thus return a model that is more like random guess.

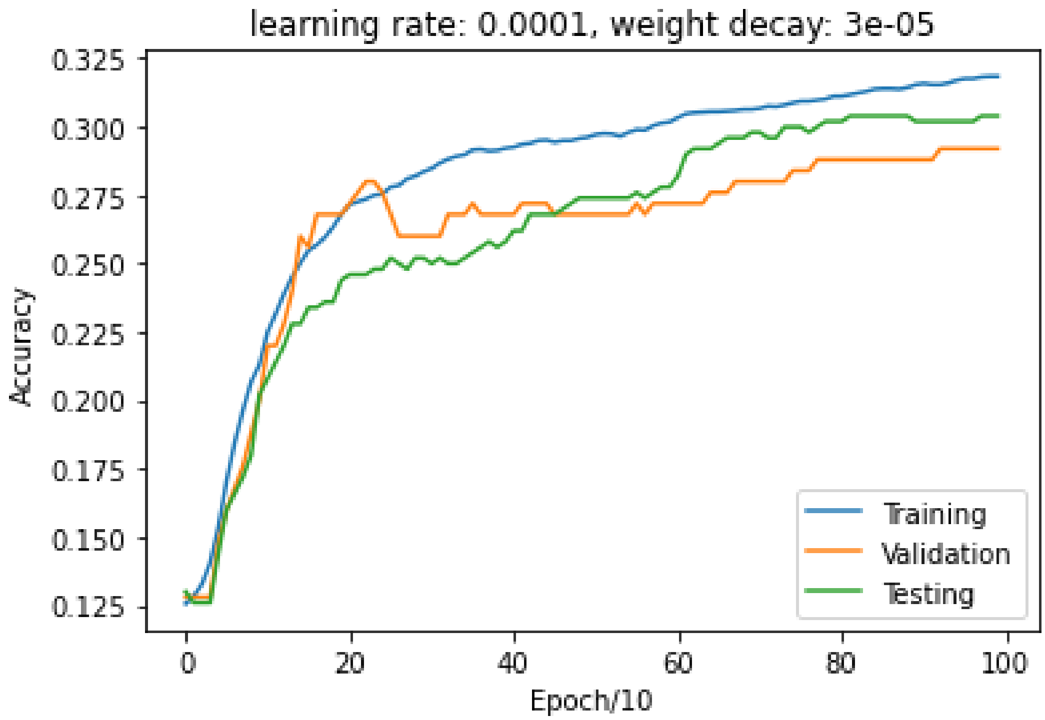
2.0.4 Regularization: Try different weight decay and plot graphs for all (20%)

[10]:
```
# 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
weight_decays = [0.0, 0.00005, 0.00003, 0.00002, 0.00001, 0.000005]

# FEEL FREE TO EXPERIMENT WITH OTHER VALUES. REPORT OTHER VALUES IF THEY␣
↪ACHIEVE A BETTER PERFORMANCE
for i in range(len(weight_decays)):
    t_ac, v_ac, te_ac, weights = train(0.0001, weight_decays[i])
    title = "learning rate: " + str(0.0001) + ", weight decay: " +␣
↪str(weight_decays[i])
    plot_accuracies(t_ac, v_ac, te_ac, title)
# 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)
```

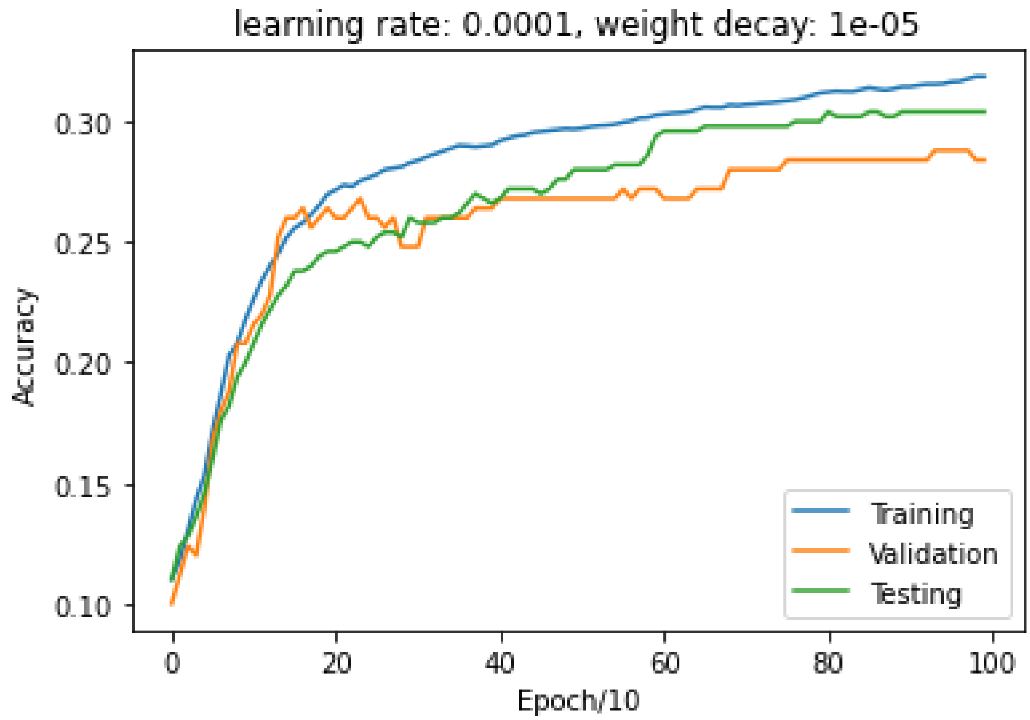






learning rate: 0.0001, weight decay: 3e-05
0.325
0.300
0.275
0.250
0.225
0.200
0.175
0.150
0.125
Accuracy
0
20
40
60
80
100
Epoch/10
Training
Validation
Testing

learning rate: 0.0001, weight decay: 2e-05
0.30
0.25
0.20
0.15
0.10
Accuracy
0
20
40
60
80
100
Epoch/10
Training
Validation
Testing


learning rate: 0.0001, weight decay: 1e-05
Accuracy
Epoch/10
Training
Validation
Testing
0.30
0.25
0.20
0.15
0.10
0
20
40
60
80
100


learning rate: 0.0001, weight decay: 5e-06
Accuracy
Epoch/10
Training
Validation
Testing
0.30
0.25
0.20
0.15
0.10
0
20
40
60
80
100

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: The optimal weight decay is 5e-6. From the graph above, it is easily to see that weight decay rate should be really small. And from the graph above, the training accuracy is larger than testing accuracy that means overfitting occur. However, overfitting is not a big problem since the gap is small. when weight decay value equal to 5e-6 (largest value in testing), the gap will decrease most and provide optimal result

2.0.5 Visualize the filters (10%)

[141]:
```
# These visualizations will only somewhat make sense if your learning rate and␣
↪weight_decay parameters were
# properly chosen in the model. Do your best.

w = 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()
```

```python
# TODO: Run this cell and Show filter visualizations for the best set of weights you obtain.
# Report the 3 hyperparameters you used to obtain the best model.

# Be careful about choosing the 'weights' obtained from the correct trained classifier
```

[ ]:

```python

```