11_exercise_image_classification_robertson

April 29, 2025

Exercise 11: Image classification with Neural Networks

CPSC 381/581: Machine Learning

Yale University

Instructor: Alex Wong

Student: Hailey Robertson

Prerequisites:

1. Enable Google Colaboratory as an app on your Google Drive account
2. Create a new Google Colab notebook, this will also create a "Colab Notebooks" directory under "MyDrive" i.e.

`/content/drive/MyDrive/Colab Notebooks`

3. Create the following directory structure in your Google Drive

`/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine Learning/Exercises`

4. Move the 11_exercise_exercise_image_classification.ipynb into

`/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine Learning/Exercises`

so that its absolute path is

`/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine Learning/Exercises/11_exercise_exe`

5. Prior to starting this exercise, please create a directory called 'data' within your 'Exercises' directory and within 'data' create a directory called 'exercise_11', i.e.

`/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine Learning/Exercises/data/exercise_`

6. Set up GPU runtime by selecting `Runtime` on the top tool bar, then selecting `Change runtime type` in the drop-down menu, selecting `GPU` under Hardware accelerator and clicking `Save`.

In this exercise, we will create a simple neural network model and a logistic regression model for classifying images. We will experiment with learning rate, batch size, and different configurations of layers within the network. We will demonstrate this on the CIFAR-10 dataset. Note: Accuracy of Neural Network should exceed 52%.

Submission:

1. Implement all TODOs in the code blocks below.
2. Report your training and testing scores.

```
Training neural_network model
Epoch=1/50  Loss: 2.241
Epoch=2/50  Loss: 1.969
Epoch=3/50  Loss: 1.809
Epoch=4/50  Loss: 1.708
Epoch=5/50  Loss: 1.640
Epoch=6/50  Loss: 1.589
Epoch=7/50  Loss: 1.537
Epoch=8/50  Loss: 1.496
Epoch=9/50  Loss: 1.465
Epoch=10/50  Loss: 1.435
Epoch=11/50  Loss: 1.399
Epoch=12/50  Loss: 1.383
Epoch=13/50  Loss: 1.337
Epoch=14/50  Loss: 1.317
Epoch=15/50  Loss: 1.286
Epoch=16/50  Loss: 1.264
Epoch=17/50  Loss: 1.228
Epoch=18/50  Loss: 1.219
Epoch=19/50  Loss: 1.185
Epoch=20/50  Loss: 1.169
Epoch=21/50  Loss: 1.137
Epoch=22/50  Loss: 1.120
Epoch=23/50  Loss: 1.092
Epoch=24/50  Loss: 1.070
Epoch=25/50  Loss: 1.039
Epoch=26/50  Loss: 1.022
Epoch=27/50  Loss: 0.994
Epoch=28/50  Loss: 0.975
Epoch=29/50  Loss: 0.945
Epoch=30/50  Loss: 0.928
Epoch=31/50  Loss: 0.895
Epoch=32/50  Loss: 0.882
Epoch=33/50  Loss: 0.849
Epoch=34/50  Loss: 0.824
Epoch=35/50  Loss: 0.797
Epoch=36/50  Loss: 0.782
Epoch=37/50  Loss: 0.747
Epoch=38/50  Loss: 0.728
Epoch=39/50  Loss: 0.700
Epoch=40/50  Loss: 0.682
Epoch=41/50  Loss: 0.651
Epoch=42/50  Loss: 0.629
Epoch=43/50  Loss: 0.597
Epoch=44/50  Loss: 0.588
Epoch=45/50  Loss: 0.553
Epoch=46/50  Loss: 0.541
Epoch=47/50  Loss: 0.507
```

```
Epoch=48/50  Loss: 0.491
Epoch=49/50  Loss: 0.463
Epoch=50/50  Loss: 0.449

Evaluating neural_network model from ./checkpoint-neural_network.pth
Mean accuracy over 10000 images: 55.310%

---

Training logistic_regression model
Epoch=1/50  Loss: 1.973
Epoch=2/50  Loss: 1.880
Epoch=3/50  Loss: 1.835
Epoch=4/50  Loss: 1.824
Epoch=5/50  Loss: 1.796
Epoch=6/50  Loss: 1.794
Epoch=7/50  Loss: 1.773
Epoch=8/50  Loss: 1.768
Epoch=9/50  Loss: 1.740
Epoch=10/50  Loss: 1.740
Epoch=11/50  Loss: 1.719
Epoch=12/50  Loss: 1.729
Epoch=13/50  Loss: 1.718
Epoch=14/50  Loss: 1.710
Epoch=15/50  Loss: 1.705
Epoch=16/50  Loss: 1.702
Epoch=17/50  Loss: 1.690
Epoch=18/50  Loss: 1.694
Epoch=19/50  Loss: 1.687
Epoch=20/50  Loss: 1.685
Epoch=21/50  Loss: 1.676
Epoch=22/50  Loss: 1.672
Epoch=23/50  Loss: 1.681
Epoch=24/50  Loss: 1.669
Epoch=25/50  Loss: 1.662
Epoch=26/50  Loss: 1.660
Epoch=27/50  Loss: 1.663
Epoch=28/50  Loss: 1.658
Epoch=29/50  Loss: 1.658
Epoch=30/50  Loss: 1.655
Epoch=31/50  Loss: 1.652
Epoch=32/50  Loss: 1.652
Epoch=33/50  Loss: 1.650
Epoch=34/50  Loss: 1.646
Epoch=35/50  Loss: 1.643
Epoch=36/50  Loss: 1.643
Epoch=37/50  Loss: 1.643
Epoch=38/50  Loss: 1.641
```

```
Epoch=39/50  Loss: 1.640
Epoch=40/50  Loss: 1.639
Epoch=41/50  Loss: 1.636
Epoch=42/50  Loss: 1.637
Epoch=43/50  Loss: 1.635
Epoch=44/50  Loss: 1.635
Epoch=45/50  Loss: 1.632
Epoch=46/50  Loss: 1.632
Epoch=47/50  Loss: 1.632
Epoch=48/50  Loss: 1.631
Epoch=49/50  Loss: 1.631
Epoch=50/50  Loss: 1.630

Evaluating logistic_regression model from ./checkpoint-logistic_regression.pth
Mean accuracy over 10000 images: 40.560%
```

3. List any collaborators.

```
Collaborators: N/A
```

Import packages

[132]:
```
from google.colab import drive
from google.colab import auth
from google.auth import default
import os

drive.mount('/content/drive/', force_remount=True)
os.chdir('/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine Learning/
 ↪Exercises')
```

```
Mounted at /content/drive/
```

[133]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch, torchvision
```

Hyper-parameters for training neural network

[134]:
```
# TODO: Choose hyper-parameters

# Model - either neural network or logistic regression
MODEL_NAME = 'logistic_regression'

# Batch size - number of images within a training batch of one training␣
 ↪iteration
N_BATCH = 128

# Training epoch - number of passes through the full training dataset
N_EPOCH = 50
```

```
# Learning rate - step size to update parameters
LEARNING_RATE = 0.01

# Learning rate decay - scaling factor to decrease learning rate at the end of␣
↪each decay period
LEARNING_RATE_DECAY = 0.9

# Learning rate decay period - number of epochs before reducing/decaying␣
↪learning rate
LEARNING_RATE_DECAY_PERIOD = 2
```

Define Neural Network

```
[135]: class NeuralNetwork(torch.nn.Module):
    '''
    Neural network class of fully connected layers

    Arg(s):
        n_input_feature : int
            number of input features
        n_output : int
            number of output classes
    '''

    def __init__(self, n_input_feature, n_output):
        super(NeuralNetwork, self).__init__()

        # Create your 6-layer neural network using fully connected layers with␣
↪ReLU activations
        # https://pytorch.org/docs/stable/generated/torch.nn.Linear.html
        # https://pytorch.org/docs/stable/generated/torch.nn.functional.relu.
↪html
        # https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html

        # DONE: Instantiate 5 fully connected layers
        self.fully_connected_layer_1 = torch.nn.Linear(n_input_feature, 1024)
        self.fully_connected_layer_2 = torch.nn.Linear(1024, 512)
        self.fully_connected_layer_3 = torch.nn.Linear(512, 256)
        self.fully_connected_layer_4 = torch.nn.Linear(256, 128)
        self.fully_connected_layer_5 = torch.nn.Linear(128, 64)

        # DONE: Define output layer
        self.output = torch.nn.Linear(64, n_output)

    def forward(self, x):
        '''
```

```
        Forward pass through the neural network

        Arg(s):
            x : torch.Tensor[float32]
                tensor of N x d
        Returns:
            torch.Tensor[float32]
                tensor of n_output predicted class
        '''

        # DONE?: Implement forward function
        output_fc1 = torch.relu(self.fully_connected_layer_1(x))
        output_fc2 = torch.relu(self.fully_connected_layer_2(output_fc1))
        output_fc3 = torch.relu(self.fully_connected_layer_3(output_fc2))
        output_fc4 = torch.relu(self.fully_connected_layer_4(output_fc3))
        output_fc5 = torch.relu(self.fully_connected_layer_5(output_fc4))

        output_logits = self.output(output_fc5)

        return output_logits
```

[136]:

```
class LogisticRegression(torch.nn.Module):
    '''
    Logistic regression class

    Arg(s):
        n_input_feature : int
            number of input features
        n_output : int
            number of output classes
    '''

    def __init__(self, n_input_feature, n_output):
        super(LogisticRegression, self).__init__()

        # Create your logistic regression model using a fully connected layer
        # https://pytorch.org/docs/stable/generated/torch.nn.Linear.html

        # DONE: Define linear layer
        self.linear = torch.nn.Linear(n_input_feature, n_output)

    def forward(self, x):
        '''
        Forward pass through the neural network

        Arg(s):
            x : torch.Tensor[float32]
```

```
                tensor of N x d
        Returns:
            torch.Tensor[float32]
                tensor of n_output predicted class
        '''

        # DONE: Implement forward function
        output_logits = self.linear(x)

        return output_logits
```

Define training loop

[137]:

```
def train(model,
          dataloader,
          n_epoch,
          optimizer,
          learning_rate_decay,
          learning_rate_decay_period,
          device):
    '''
    Trains the model using optimizer and specified learning rate schedule

    Arg(s):
        model : torch.nn.Module
            neural network or logistic regression
        dataloader : torch.utils.data.DataLoader
            # https://pytorch.org/docs/stable/data.html
            dataloader for training data
        n_epoch : int
            number of epochs to train
        optimizer : torch.optim
            https://pytorch.org/docs/stable/optim.html
            optimizer to use for updating weights
        learning_rate_decay : float
            rate of learning rate decay
        learning_rate_decay_period : int
            period to reduce learning rate based on decay e.g. every 2 epoch
        device : str
            device to run on
    Returns:
        torch.nn.Module : trained network
    '''

    device = 'cuda' if device == 'gpu' or device == 'cuda' else 'cpu'
    device = torch.device(device)
```

```
    # DONE: Move model to device using .to()
    model = model.to(device)

    # DONE: Define cross entropy loss (note: torch.nn.CrossEntropyLoss takes␣
↪logits as inputs)
    # https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html
    loss_func = torch.nn.CrossEntropyLoss()

    for epoch in range(n_epoch):

        # Accumulate total loss for each epoch
        total_loss = 0.0

        # DONE: Decrease learning rate when learning rate decay period is met
        # Directly modify param_groups in optimizer to set new learning rate
        # e.g. decrease learning rate by a factor of decay rate every 2 epoch
        if epoch % learning_rate_decay_period == 0 and epoch != 0:
            for param_group in optimizer.param_groups:
                param_group['lr'] *= learning_rate_decay

        # DONE?: Enumerate through batches of (images, labels) from dataloader
        for batch_idx, (images, labels) in enumerate(dataloader):

            # DONE: Move images and labels to device using .to()
            images = images.to(device)
            labels = labels.to(device)

            # DONE?: Vectorize images from (N, H, W, C) to (N, d)
            n_dim = images.shape[1] * images.shape[2] * images.shape[3]
            images = images.view(-1, n_dim)

            # DONE: Forward through the model
            outputs = model(images)

            # DONE: Clear gradients so we don't accumlate them from previous␣
↪batches
            optimizer.zero_grad()

            # DONE: Compute loss function
            loss = loss_func(outputs, labels)

            # DONE: Update parameters by backpropagation
            loss.backward()
            optimizer.step()

            # DONE: Accumulate total loss for the epoch
            total_loss = total_loss + loss.item()
```

```
        # DONE: Compute average loss for the epoch
        mean_loss = total_loss / len(dataloader)

        # Log average loss over the epoch
        print('Epoch={}/{}  Loss: {:.3f}'.format(epoch + 1, n_epoch, mean_loss))

    return model
```

Define evaluation loop

[138]:
```
def evaluate(model, dataloader, class_names, device):
    '''
    Evaluates the network on a dataset

    Arg(s):
        model : torch.nn.Module
            neural network or logistic regression
        dataloader : torch.utils.data.DataLoader
            # https://pytorch.org/docs/stable/data.html
            dataloader for training data
        class_names : list[str]
            list of class names to be used in plot
        device : str
            device to run on
    '''

    device = 'cuda' if device == 'gpu' or device == 'cuda' else 'cpu'
    device = torch.device(device)

    # DONE: Move model to device using .to()
    model = model.to(device)

    n_correct = 0
    n_sample = 0

    # Make sure we do not backpropagate
    with torch.no_grad():

        # DONE: Iterate through samples (images, labels) from dataloader
        for images, labels in dataloader:

            # DONE: Move images and labels to device using .to()
            images = images.to(device)
            labels = labels.to(device)

            # DONE: Vectorize images from (N, H, W, C) to (N, d)
```

```
            n_dim = images.shape[1] * images.shape[2] * images.shape[3]
            images = images.view(-1, n_dim)

            # DONE: Forward through the model
            outputs = model(images)

            # DONE?: Take the argmax over the outputs
            outputs = torch.argmax(outputs, dim=1)

            # DONE?: Accumulate number of samples
            # maybe labels.size(0)
            n_sample = n_sample + outputs.size(0)

            # DONE?: Check if our prediction is correct
            n_correct = n_correct + (outputs == labels).sum()

    # DONE: Compute mean accuracy
    mean_accuracy = (n_correct / n_sample) * 100

    print('Mean accuracy over {} images: {:.3f}%'.format(n_sample,␣
 ↪mean_accuracy))
```

Training a neural network and logistic regression for image classification

[139]:
```
# Create transformations convert data to torch tensor
# https://pytorch.org/docs/stable/torchvision/transforms.html
transforms = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
])

# Set path to save checkpoint
checkpoint_path = './checkpoint-{}.pth'.format(MODEL_NAME)
```

[140]:
```
'''
Set up dataloading
'''
# Download and setup CIFAR10 training set using preconfigured torchvision.
 ↪datasets.CIFAR10
cifar10_train = torchvision.datasets.CIFAR10(
    root=os.path.join('data', 'exercise_11'),
    train=True,
    download=True,
    transform=transforms)

# DONE?: Setup a dataloader (iterator) to fetch from the training set using
# torch.utils.data.DataLoader and set shuffle=True, drop_last=True,␣
 ↪num_workers=2
```

```python
# Set your batch size to the hyperparameter N_BATCH
dataloader_train = torch.utils.data.DataLoader(
    cifar10_train,
    batch_size=N_BATCH,
    shuffle=True,
    drop_last=True,
    num_workers=2)

# Define the possible classes in CIFAR10
class_names = [
    'plane',
    'car',
    'bird',
    'cat',
    'deer',
    'dog',
    'frog',
    'horse',
    'ship',
    'truck'
]

# CIFAR10 has 10 classes
n_class = len(class_names)

'''
Set up model and optimizer
'''
# DONE: Compute number of input features. Hint: They are RGB images of size 32 x 32
n_input_feature = 3 * 32 * 32

if MODEL_NAME == 'neural_network':
    # DONE: Instantiate neural network
    model = NeuralNetwork(n_input_feature, n_class)
elif MODEL_NAME == 'logistic_regression':
    # DONE: Instantiate logistic regression
    model = LogisticRegression(n_input_feature, n_class)
else:
    raise('Unsupported model name: {}'.format(MODEL_NAME))

print('Training {} model'.format(MODEL_NAME))

# DONE?: Setup learning rate SGD optimizer and step function scheduler
# https://pytorch.org/docs/stable/optim.html?#torch.optim.SGD
optimizer = torch.optim.SGD(
    model.parameters(),
```

```python
    lr=LEARNING_RATE,
    momentum=0.9,
    weight_decay=5e-4
    )


'''
Train model and store weights
'''
# DONE: Set model to training mode
model.train()

# DONE: Train model with device='cuda'
model = train(
    model=model,
    dataloader=dataloader_train,
    n_epoch=N_EPOCH,
    optimizer=optimizer,
    learning_rate_decay=LEARNING_RATE_DECAY,
    learning_rate_decay_period=LEARNING_RATE_DECAY_PERIOD,
    device='cuda')

# DONE: Save weights into checkpoint path
torch.save(model.state_dict(), checkpoint_path)
```

```
Training logistic_regression model
Epoch=1/50  Loss: 1.973
Epoch=2/50  Loss: 1.880
Epoch=3/50  Loss: 1.835
Epoch=4/50  Loss: 1.824
Epoch=5/50  Loss: 1.796
Epoch=6/50  Loss: 1.794
Epoch=7/50  Loss: 1.773
Epoch=8/50  Loss: 1.768
Epoch=9/50  Loss: 1.740
Epoch=10/50  Loss: 1.740
Epoch=11/50  Loss: 1.719
Epoch=12/50  Loss: 1.729
Epoch=13/50  Loss: 1.718
Epoch=14/50  Loss: 1.710
Epoch=15/50  Loss: 1.705
Epoch=16/50  Loss: 1.702
Epoch=17/50  Loss: 1.690
Epoch=18/50  Loss: 1.694
Epoch=19/50  Loss: 1.687
Epoch=20/50  Loss: 1.685
Epoch=21/50  Loss: 1.676
Epoch=22/50  Loss: 1.672
```

```
Epoch=23/50  Loss: 1.681
Epoch=24/50  Loss: 1.669
Epoch=25/50  Loss: 1.662
Epoch=26/50  Loss: 1.660
Epoch=27/50  Loss: 1.663
Epoch=28/50  Loss: 1.658
Epoch=29/50  Loss: 1.658
Epoch=30/50  Loss: 1.655
Epoch=31/50  Loss: 1.652
Epoch=32/50  Loss: 1.652
Epoch=33/50  Loss: 1.650
Epoch=34/50  Loss: 1.646
Epoch=35/50  Loss: 1.643
Epoch=36/50  Loss: 1.643
Epoch=37/50  Loss: 1.643
Epoch=38/50  Loss: 1.641
Epoch=39/50  Loss: 1.640
Epoch=40/50  Loss: 1.639
Epoch=41/50  Loss: 1.636
Epoch=42/50  Loss: 1.637
Epoch=43/50  Loss: 1.635
Epoch=44/50  Loss: 1.635
Epoch=45/50  Loss: 1.632
Epoch=46/50  Loss: 1.632
Epoch=47/50  Loss: 1.632
Epoch=48/50  Loss: 1.631
Epoch=49/50  Loss: 1.631
Epoch=50/50  Loss: 1.630
```

Testing the trained neural network on image classification

[141]:

```
'''
Set up dataloading
'''
# DONE: Download and setup CIFAR10 testing set using
# preconfigured torchvision.datasets.CIFAR10
cifar10_test = torchvision.datasets.CIFAR10(
    root=os.path.join('data', 'exercise_11'),
    train=False,
    download=True,
    transform=transforms)

# DONE: Setup a dataloader (iterator) to fetch from the testing set using
# torch.utils.data.DataLoader and set shuffle=False, drop_last=False,␣
 ↪num_workers=2
# Set batch_size to 25
dataloader_test = torch.utils.data.DataLoader(
    cifar10_test,
```

```
    batch_size=25,
    shuffle=False,
    drop_last=False,
    num_workers=2)

'''
Set up model
'''
# DONE: Compute number of input features. Hint: They are RGB images of size 32␣
 ↪x 32
n_input_feature = 3 * 32 * 32

if MODEL_NAME == 'neural_network':
    # DONE: Instantiate neural network
    model = NeuralNetwork(n_input_feature, n_class)
elif MODEL_NAME == 'logistic_regression':
    # DONE: Instantiate logistic regression
    model = LogisticRegression(n_input_feature, n_class)
else:
    raise('Unsupported model name: {}'.format(MODEL_NAME))

print('Evaluating {} model from {}'.format(MODEL_NAME, checkpoint_path))

'''
Restore weights and evaluate model
'''
# DONE: Load model from checkpoint
checkpoint = model.load_state_dict(torch.load(checkpoint_path))


# DONE: Set model to evaluation mode
model.eval()

# DONE: Evaluate model on testing set with device='cuda'
evaluate(
    model=model,
    dataloader=dataloader_test,
    class_names=class_names,
    device='cuda')
```

```
Evaluating logistic_regression model from ./checkpoint-logistic_regression.pth
Mean accuracy over 10000 images: 40.560%
```