cifar-10

April 12, 2025

Astar training 2025

This a notebook for the Astar training 2025. You are guided to train a simple classifier on the cifar-10 dataset.

Instruction

Please try to read through this notebook, and fill in the blank according to the instructions given after TODO .

You may download the notebook or excute directly on kaggle.

Submission

submit .pdf converted from this notebook after completion

code Requirements

This notebook requires the following packages

excute the following command to install the required packages

[1]:
```
!pip install torch torchvision
!pip install matplotlib
!pip install numpy
!pip install tqdm
```

```
Collecting torch
  Downloading torch-2.6.0-cp312-cp312-win_amd64.whl.metadata (28 kB)
Collecting torchvision
  Downloading torchvision-0.21.0-cp312-cp312-win_amd64.whl.metadata (6.3 kB)
Requirement already satisfied: filelock in
c:\users\fujing123\anaconda3\lib\site-packages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions>=4.10.0 in
c:\users\fujing123\anaconda3\lib\site-packages (from torch) (4.11.0)
Requirement already satisfied: networkx in
c:\users\fujing123\anaconda3\lib\site-packages (from torch) (3.3)
Requirement already satisfied: jinja2 in c:\users\fujing123\anaconda3\lib\site-
packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in c:\users\fujing123\anaconda3\lib\site-
packages (from torch) (2024.6.1)
Requirement already satisfied: setuptools in
c:\users\fujing123\anaconda3\lib\site-packages (from torch) (75.1.0)
```

```
Collecting sympy==1.13.1 (from torch)
  Downloading sympy-1.13.1-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
c:\users\fujing123\anaconda3\lib\site-packages (from sympy==1.13.1->torch)
(1.3.0)
Requirement already satisfied: numpy in c:\users\fujing123\anaconda3\lib\site-
packages (from torchvision) (1.26.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in
c:\users\fujing123\anaconda3\lib\site-packages (from torchvision) (10.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in
c:\users\fujing123\anaconda3\lib\site-packages (from jinja2->torch) (2.1.3)
Downloading torch-2.6.0-cp312-cp312-win_amd64.whl (204.1 MB)
   ---------------------------------------- 0.0/204.1 MB ? eta -:--:--
   - -------------------------------------- 5.5/204.1 MB 30.5 MB/s eta 0:00:07
   - -------------------------------------- 10.0/204.1 MB 24.8 MB/s eta 0:00:08
   --- ------------------------------------ 17.6/204.1 MB 28.4 MB/s eta 0:00:07
   ---- ----------------------------------- 25.4/204.1 MB 30.4 MB/s eta 0:00:06
   ------ --------------------------------- 33.0/204.1 MB 31.3 MB/s eta 0:00:06
   ------- -------------------------------- 40.4/204.1 MB 32.1 MB/s eta 0:00:06
   --------- ------------------------------ 47.2/204.1 MB 32.6 MB/s eta 0:00:05
   ---------- ----------------------------- 55.1/204.1 MB 33.1 MB/s eta 0:00:05
   ------------ --------------------------- 62.4/204.1 MB 33.4 MB/s eta 0:00:05
   ------------- -------------------------- 69.7/204.1 MB 33.7 MB/s eta 0:00:04
   --------------- ------------------------ 77.3/204.1 MB 33.8 MB/s eta 0:00:04
   ---------------- ----------------------- 84.7/204.1 MB 34.0 MB/s eta 0:00:04
   ------------------ --------------------- 92.3/204.1 MB 34.0 MB/s eta 0:00:04
   ------------------- -------------------- 99.4/204.1 MB 33.9 MB/s eta 0:00:04
   -------------------- ------------------ 107.0/204.1 MB 34.1 MB/s eta 0:00:03
   --------------------- ----------------- 114.6/204.1 MB 34.2 MB/s eta 0:00:03
   ----------------------- --------------- 121.9/204.1 MB 34.3 MB/s eta 0:00:03
   ------------------------ -------------- 128.2/204.1 MB 34.0 MB/s eta 0:00:03
   ------------------------- ------------- 134.5/204.1 MB 33.8 MB/s eta 0:00:03
   --------------------------- ----------- 141.6/204.1 MB 33.7 MB/s eta 0:00:02
   ---------------------------- ---------- 148.4/204.1 MB 33.9 MB/s eta 0:00:02
   ----------------------------- --------- 154.7/204.1 MB 33.6 MB/s eta 0:00:02
   ------------------------------ -------- 162.0/204.1 MB 33.7 MB/s eta 0:00:02
   -------------------------------- ------ 169.1/204.1 MB 33.7 MB/s eta 0:00:02
   --------------------------------- ----- 176.7/204.1 MB 33.8 MB/s eta 0:00:01
   ----------------------------------- --- 184.3/204.1 MB 33.8 MB/s eta 0:00:01
   ------------------------------------ -- 191.4/204.1 MB 33.9 MB/s eta 0:00:01
   ------------------------------------- - 198.4/204.1 MB 33.8 MB/s eta 0:00:01
   --------------------------------------  203.9/204.1 MB 33.8 MB/s eta 0:00:01
   --------------------------------------- 204.1/204.1 MB 32.7 MB/s eta 0:00:00
Downloading sympy-1.13.1-py3-none-any.whl (6.2 MB)
   ---------------------------------------- 0.0/6.2 MB ? eta -:--:--
   ---------------------------------------- 6.2/6.2 MB 34.5 MB/s eta 0:00:00
Downloading torchvision-0.21.0-cp312-cp312-win_amd64.whl (1.6 MB)
   ---------------------------------------- 0.0/1.6 MB ? eta -:--:--
```

```
   ---------------------------------------- 1.6/1.6 MB 27.6 MB/s eta 0:00:00
Installing collected packages: sympy, torch, torchvision
  Attempting uninstall: sympy
    Found existing installation: sympy 1.13.2
    Uninstalling sympy-1.13.2:
      Successfully uninstalled sympy-1.13.2
Successfully installed sympy-1.13.1 torch-2.6.0 torchvision-0.21.0
Requirement already satisfied: matplotlib in
c:\users\fujing123\anaconda3\lib\site-packages (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\fujing123\anaconda3\lib\site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in
c:\users\fujing123\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\fujing123\anaconda3\lib\site-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\fujing123\anaconda3\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy>=1.23 in
c:\users\fujing123\anaconda3\lib\site-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in
c:\users\fujing123\appdata\roaming\python\python312\site-packages (from
matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in
c:\users\fujing123\anaconda3\lib\site-packages (from matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\fujing123\anaconda3\lib\site-packages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\fujing123\appdata\roaming\python\python312\site-packages (from
matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in
c:\users\fujing123\appdata\roaming\python\python312\site-packages (from python-
dateutil>=2.7->matplotlib) (1.16.0)
Requirement already satisfied: numpy in c:\users\fujing123\anaconda3\lib\site-
packages (1.26.4)
Requirement already satisfied: tqdm in c:\users\fujing123\anaconda3\lib\site-
packages (4.66.5)
Requirement already satisfied: colorama in
c:\users\fujing123\appdata\roaming\python\python312\site-packages (from tqdm)
(0.4.6)
```

[1]:
```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
from torchvision import datasets, transforms
```

```python
import matplotlib.pyplot as plt

import numpy as np
import pickle

import os

from tqdm import tqdm
```

Data preparation

The dataloader is already implemented for you. You can use it to load the cifar-10 dataset.

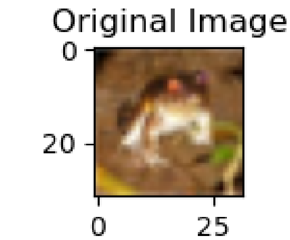
[2]:
```python
class CIFAR10(Dataset):
    #__init__ is a special method in python classes, it is called when an␣
 ↪object is created
    def __init__(self, root, train=True, transform=None):
        #root path to the dataset, you may ignore this
        self.root = root

        #this is a boolean value to indicate whether we are loading the␣
 ↪training or test set
        self.train = train

        #this is the transformation that will be applied to the images
        #it's none by default, you are required to pass a transform later
        self.transform = transform

        #checking if the dataset exists, if not download it
        if not self._check_exists():
            print("Downloading cifar10 dataset")
            self.download()

        #load the data
        #We are going to load the data in memory
        #Store the images and labels in the self.data and self.targets variables
        if self.train:
            self.data, self.targets = self._load_training_data()
        else:
            self.data, self.targets = self._load_test_data()

    #this method returns the length of the dataset
    def __len__(self):
        return len(self.data)

    #this method returns a sample from the dataset at the given index
    #this is very important because it allows us to iterate over the dataset
    def __getitem__(self, index):
```

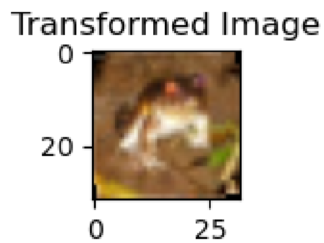
```
        img, target = self.data[index], int(self.targets[index])

        img = torch.from_numpy(img).float().permute(2,0,1) / 255.0

        if self.transform:
            img = self.transform(img)

        return img, target


    def _check_exists(self):
        return os.path.exists(os.path.join(self.root, "cifar-10-batches-py"))

    def download(self):
        if self._check_exists():
            print("Dataset already exists !!!")
            return
        return datasets.CIFAR10(self.root, train=self.train, download=True)

    #this function looks complicated but it's just reading the data from the␣
↪files
    def _load_batch(self, file_path):
        with open(file_path, 'rb') as f:
            data = pickle.load(f, encoding='bytes')
        return data[b'data'], data[b'labels']

    def _load_training_data(self):
        data = []
        targets = []
        for i in range(1, 6):
            file_path = os.path.join(self.root, "cifar-10-batches-py",␣
↪f"data_batch_{i}")
            batch_data, batch_labels = self._load_batch(file_path)
            data.append(batch_data)
            targets.extend(batch_labels)

        data = np.vstack(data).reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1)
        return data, np.array(targets)

    def _load_test_data(self):
        file_path = os.path.join(self.root, "cifar-10-batches-py", "test_batch")
        data, labels = self._load_batch(file_path)
        return data.reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1), np.
↪array(labels)
```

[3]:
```python
#We have define our datasetclass, now we are going to instantiate it
#The instantiation will download the dataset and load it in memory, it takes␣
↪about 1-2 mins
train_dataset = CIFAR10(root="data", train=True)
test_dataset = CIFAR10(root="data", train=False)

#simple demonstration __getitem__ method
img, target = train_dataset[0] #get the first image in the dataset
plt.figure(figsize=(1,1))
plt.imshow(img.permute(1,2,0))
#TODO: what does permute do?, and why do we need it here?
#answer:
#
#end of you answer
plt.title("Original Image")
print(img.shape, "label: ",target)
#the image is a torch tensor (3, 28, 28) and the target is the label of the␣
↪image


#TODO: define reasonable transformations for the images, you can use the␣
↪transforms module from torchvision
transform = transforms.Compose([
    transforms.ToPILImage(),

    #TODO: Implement the transformations here

    transforms.RandomRotation(15),
    transforms.RandomHorizontalFlip(),

    #end of your implementation;

    transforms.ToTensor()
])

train_dataset.transform = transform

img, target = train_dataset[0] #get the first image in the dataset
plt.figure(figsize=(1,1))
plt.imshow(img.permute(1,2,0))
plt.title("Transformed Image")
print(img.shape, "label: ",target)
```

```
Downloading cifar10 dataset
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
data\cifar-10-python.tar.gz

100%|     | 170M/170M [00:15<00:00, 11.1MB/s]
```

```
Extracting data\cifar-10-python.tar.gz to data
torch.Size([3, 32, 32]) label:  6
torch.Size([3, 32, 32]) label:  6
```





[4]:
```python
#apart from test set, we are going to use the training set to create a validation set
#we are going to split the training set into two parts
train_size = int(0.9 * len(train_dataset))
val_size = len(train_dataset) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(train_dataset, [train_size, val_size])

#we are going to use the DataLoader class to create an iterator for our dataset
#this iterator will be used to iterate over the dataset in batches
#tentatively we are going to use a batch size of 32
#TODO: change different batch sizes and see how it affects the training process
train_loader = DataLoader(train_dataset, batch_size=256, shuffle=True, num_workers= 0, pin_memory=True)
val_loader = DataLoader(val_dataset, batch_size=256, shuffle=False,num_workers= 0, pin_memory = True)
test_loader = DataLoader(test_dataset, batch_size=256, shuffle=False,num_workers= 0, pin_memory = True)
```

Model

This defines the model architecture. You can use the model as it is or modify it as per your requirements.

[6]:

```
class LinearModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.name = "Linear"
        self.num_inputs = 3*32*32
        hidden_size = 512
        num_classes = 10
        super().__init__()
        self.linear = nn.Sequential(
            nn.Linear(self.num_inputs, hidden_size),    # batch_size x 784 ->␣
↪batch_size x 512
            nn.ReLU(), #activation function             # batch_size x 512 ->␣
↪batch_size x 512
            nn.Linear(hidden_size, num_classes)         # batch_size x 512 ->␣
↪batch_size x 10
        ) #nn.Sequential is a container for other layers, it applies the layers␣
↪in sequence

    #forward is the method that defines the forward pass of the network
    #not rigurously: model.forward(x) = model(x)x
    def forward(self, x):
        x = x.view(-1, self.num_inputs) # flatten the image from 3x32x32 to 3072
        x = self.linear(x)
        return x

class CNNModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.name = "CNN"
        self.conv = nn.Sequential(
            #TODO: wirte the size of the input and output of each layer e.g
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1), #input:␣
↪3x32x32, output: 32x32x32
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.fc = nn.Sequential(
            nn.Linear(128*4*4, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )
```

```
    def forward(self, x):
        x = self.conv(x)
        x = x.view(-1, 128*4*4)
        x = self.fc(x)
        return x

    def getFeature(self, x):
        x = self.conv(x)
        feat = x.view(-1, 128*4*4) #this is the 128*4*4 feature
        return feat
```

[10]:
```
print(torch.__version__)
print(torch.version.cuda)
print(torch.cuda.is_available())
print(torch.cuda.get_device_name(0))
```

```
2.5.1+cu121
12.1
True
NVIDIA GeForce RTX 4060 Laptop GPU
```

Training

This is the training loop. You can modify the training loop as per your requirements.

The training takes some time. You can do other tasks while the training is in progress.

you may use the gpu on kaggle or colab to speed up the training process. https://www.kaggle.com/discussions/general/97939

[9]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("using device:", device)
#instantiating the model
#TODO: change the model to the CNNModel and Your model, and evaluate the␣
 ↪performance.
#model = LinearModel()
model = CNNModel()
model = model.to(device)
print(f"training {model.name} model")

#defining the loss function
criterion = nn.CrossEntropyLoss()

#defining the optimizer
#TODO: try to modify the optimizer and see how it affects the training process
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

```python
best_accuracy = 0
#early stopping
early_stopping = 5
early_stopping_counter = 0



#TODO: adjust the number of epochs and see how it affects the training process
epochs = 20
for epoch in range(epochs):
    #training
    for images, labels in tqdm(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        #forward pass
        outputs = model(images)
        #calculate the loss
        loss = criterion(outputs, labels)
        #zero the gradients
        optimizer.zero_grad() #ensure that the gradients are zero
        #backward pass
        loss.backward()
        #optimize
        optimizer.step()
    #validation
    total = 0
    correct = 0
    for images, labels in val_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
    accuracy = correct / total

    #TODO: implement early stopping
    #what is early stopping? https://en.wikipedia.org/wiki/Early_stopping

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        early_stopping_counter=0
        #save the model
        torch.save(model.state_dict(), f"best_model_{model.name}.pth")
        print(f"Epoch: {epoch}, Loss: {loss.item()}, Accuracy: {accuracy} ***")
    else:
        early_stopping_counter += 1
```

```python
        print(f"Epoch: {epoch}, Loss: {loss.item()}, Accuracy: {accuracy}")

    if(early_stopping_counter == early_stopping):
        break

    #end of early stopping
```

```
using device: cuda
training CNN model

100%|          | 176/176 [00:08<00:00, 19.75it/s]

Epoch: 0, Loss: 1.580361008644104, Accuracy: 0.3976 ***

100%|          | 176/176 [00:08<00:00, 20.85it/s]

Epoch: 1, Loss: 1.5204532146453857, Accuracy: 0.4254 ***

100%|          | 176/176 [00:08<00:00, 20.88it/s]

Epoch: 2, Loss: 1.4409593343734741, Accuracy: 0.4712 ***

100%|          | 176/176 [00:08<00:00, 20.75it/s]

Epoch: 3, Loss: 1.3205548524856567, Accuracy: 0.483 ***

100%|          | 176/176 [00:08<00:00, 20.72it/s]

Epoch: 4, Loss: 1.3125721216201782, Accuracy: 0.5206 ***

100%|          | 176/176 [00:08<00:00, 20.22it/s]

Epoch: 5, Loss: 1.3749263286590576, Accuracy: 0.5222 ***

100%|          | 176/176 [00:09<00:00, 19.54it/s]

Epoch: 6, Loss: 1.3880326747894287, Accuracy: 0.5304 ***

100%|          | 176/176 [00:08<00:00, 19.79it/s]

Epoch: 7, Loss: 1.1898235082626343, Accuracy: 0.5376 ***

100%|          | 176/176 [00:08<00:00, 19.64it/s]

Epoch: 8, Loss: 1.196560025215149, Accuracy: 0.5396 ***

100%|          | 176/176 [00:08<00:00, 19.97it/s]

Epoch: 9, Loss: 1.2712793350219727, Accuracy: 0.5552 ***

100%|          | 176/176 [00:08<00:00, 19.80it/s]

Epoch: 10, Loss: 1.1496901512145996, Accuracy: 0.5484

100%|          | 176/176 [00:08<00:00, 19.91it/s]

Epoch: 11, Loss: 1.0585979223251343, Accuracy: 0.5644 ***

100%|          | 176/176 [00:08<00:00, 19.70it/s]

Epoch: 12, Loss: 1.2388828992843628, Accuracy: 0.5618
```

```
100%|      | 176/176 [00:09<00:00, 18.82it/s]

Epoch: 13, Loss: 1.2217577695846558, Accuracy: 0.556

100%|      | 176/176 [00:10<00:00, 17.26it/s]

Epoch: 14, Loss: 1.238285779953003, Accuracy: 0.552

100%|      | 176/176 [00:12<00:00, 14.29it/s]

Epoch: 15, Loss: 1.3147448301315308, Accuracy: 0.5558

100%|      | 176/176 [00:15<00:00, 11.61it/s]

Epoch: 16, Loss: 1.4128390550613403, Accuracy: 0.5678 ***

100%|      | 176/176 [00:14<00:00, 11.78it/s]

Epoch: 17, Loss: 1.2173353433609009, Accuracy: 0.5726 ***

100%|      | 176/176 [00:08<00:00, 20.39it/s]

Epoch: 18, Loss: 1.1345105171203613, Accuracy: 0.5786 ***

100%|      | 176/176 [00:08<00:00, 19.88it/s]

Epoch: 19, Loss: 1.0586787462234497, Accuracy: 0.5848 ***
```

Evaluation

[11]:
```
#load the best model
model.load_state_dict(torch.load(f"best_model_{model.name}.pth",␣
↪weights_only=False))

#testing
total = 0
correct = 0
for images, labels in test_loader:
    images = images.to(device)
    labels = labels.to(device)
    outputs = model(images)
    _, predicted = torch.max(outputs, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()
accuracy = correct / total
print(f"Test Accuracy: {accuracy}")
```

```
Test Accuracy: 0.6049
```