harikrishna-dev-hw4

October 5, 2023

Solution HW4: MultiClass Classification with PyTorch Lightning

- Fill the missing code indicated by # CODE HERE
- Submit the following two files:
  1. FirstName_LastName_HW4.ipynb
  2. FirstName_LastName_HW4.pdf (pdf version of the above file)

1 Specify Project Folder

[1]:
```
# add lines for autoreload
%load_ext autoreload
%autoreload 2
```

[2]:
```
from pathlib import Path
import sys

if 'google.colab' in str(get_ipython()):
    from google.colab import drive
    drive.mount('/content/drive')

    base_folder = Path('/content/drive/MyDrive/Colab_Notebooks/
  ↪BUAN_6382_Applied_DeepLearning/Data') # Google Drive
    data_folder = Path('/content') # Keep data on Colab

    !pip install pytorch-lightning -U -qq
    !pip install torchinfo -U -qq
```

```
Mounted at /content/drive
                          727.7/727.7 kB
10.8 MB/s eta 0:00:00
                          805.2/805.2 kB
39.6 MB/s eta 0:00:00
```

[3]:
```
base_folder
```

[3]: PosixPath('/content/drive/MyDrive/Colab_Notebooks/BUAN_6382_Applied_DeepLearning
/Data')

[4]:
```
# Change the custom_function_folder to folder in your Google drive folder
# Make sure you keep the mlp_skip_two_layer.py and shared_utils.py files

custom_function_folder = Path('/content/drive/MyDrive/Colab_Notebooks/
 ↪BUAN_6382_Applied_DeepLearning/Custom_files') # Your Google Drive

sys.path.append(str(custom_function_folder))
model_folder = Path('/content/drive/MyDrive/Colab_Notebooks/
 ↪BUAN_6382_Applied_DeepLearning/Data') # Google drive folder where you want␣
 ↪to save model and logs
model_folder.mkdir(parents=True, exist_ok=True)
```

2 Import Libraries

[5]:
```
# this should not be Google Drive Folder but local folder on Colab
data_folder
```

[5]: PosixPath('/content')

[6]:
```
import random
import numpy as np
from collections import defaultdict
from sklearn.model_selection import train_test_split
from datetime import datetime

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader, Subset
from torchinfo import summary
from torchvision import transforms, datasets

import pytorch_lightning as pl
from pytorch_lightning.loggers import CSVLogger
from torchvision.datasets import ImageFolder
import torchmetrics
from fastdownload import FastDownload
from fastai.vision.all import get_image_files, PILImage

# The classea and functions we will import from .py files
from mlp_skip_two_layer import PytorchMLPSkip
from shared_utils import check_loader, check_transforms, check_label_dist,␣
 ↪show_confusion_matrix, compute_accuracy, plot_losses_acc
```

3 Imagenette Dataset

For this HW, you will use the Imagenette dataset created by fastai https://github.com/fastai/imagenette).

Imagenette is a subset of 10 easily classified classes from Imagenet (tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball, parachute).

The dataset can be downloaded from this link: https://s3.amazonaws.com/fast-ai-imageclas/imagenette2.tgz

Dictionary to map the 10 classes with intergers

```
label_dict = {
    'n01440764' : 0,
    'n02102040' : 1,
    'n02979186' : 2,
    'n03000684' : 3,
    'n03028079' : 4,
    'n03394916' : 5,
    'n03417042' : 6,
    'n03425413' : 7,
    'n03445777' : 8,
    'n03888257' : 9
}

# Array to map integral indices with the actual class names
actual_label_dict = [
    'tench',
    'English springer',
    'cassette player',
    'chain saw',
    'church',
    'French horn',
    'garbage truck',
    'gas pump',
    'golf ball',
    'parachute'
]
```

4 Data Module

Ceate a LightningDataModule for Imagenette2 Dataset - Use the images in `train` folder to create `training` dataset - Use the images in `val` folder to create both `validation` and `test` datasets. - **Use 50% of the images in the `val` folder for validation dataset and 50% for test dataset. - Use a batch size of 64.** - The functions(methods) in the class below are mandatory. Add any other methods (functions) as to the class if required. - Add the functionality to select a stratified random subset of validation and training sets for initial training (Hint see: modified data_loaders_dog_breed.py file on eLearning, please download the file again.)

[7]:
```
# from torch.utils.data import Dataset
# def get_stratified_subset(dataset, num_samples, seed=42):
#     if seed is not None:
#         random.seed(seed)

#     # Step 1: Identify label distribution
#     label_to_indices = defaultdict(list)
#     for idx, (_, label) in enumerate(dataset):
#         label_to_indices[label].append(idx)

#     # Step 2: Calculate proportions and initialize subset indices list
#     proportions = {label: len(indices) / len(dataset) for label, indices in␣
↪label_to_indices.items()}
#     subset_indices = []

#     # Step 3: Sample according to proportion
#     for label, indices in label_to_indices.items():
#         num_samples_for_label = round(proportions[label] * num_samples)
#         subset_indices += random.sample(indices, num_samples_for_label)

#     # Step 4: Combine samples
#     return torch.utils.data.Subset(dataset, subset_indices)
```

[8]:
```
def get_stratified_subset(dataset, num_samples_small=1000, labels=None,␣
↪seed=42):
    if labels is not None:
        _, subset_indices = train_test_split(
            range(len(labels)),  # Just indices, not the actual data
            test_size=num_samples_small,
            stratify=labels,
            random_state=seed
        )
    else:
        _, subset_indices = train_test_split(
            range(len(dataset)),  # Just indices, not the actual data
            test_size=num_samples_small,
            random_state=seed
        )
    return Subset(dataset, subset_indices)
```

[9]:
```
# from sklearn.model_selection import StratifiedShuffleSplit
# def get_stratified_subset(dataset, num_samples, seed=42):
#     if seed is not None:
#         random.seed(seed)

#     sss = StratifiedShuffleSplit(n_splits=1, test_size=num_samples,␣
↪random_state=seed)
```

```
#     for _, subset_indices in sss.split(np.zeros(len(dataset.classes)),dataset.
↪classes):
#         break
#     return Subset(dataset, subset_indices)
```

[10]:
```
def split_dataset(base_dataset, fraction = 0.5, seed=42):
    split_a_size = int(fraction * len(base_dataset))
    split_b_size = len(base_dataset) - split_a_size
    return torch.utils.data.random_split(
        base_dataset,
        [split_a_size, split_b_size],
        generator=torch.Generator().manual_seed(seed)
    )
```

[11]:
```
# d = FastDownload(base=data_folder, archive='archive', data='datasets')
# data_folder_imagenette2 = d.get('https://s3.amazonaws.com/fast-ai-imageclas/
↪imagenette2.tgz')
# image_files = get_image_files('datasets/imagenette2/')
```

[12]:
```
# train_dataset = ImageFolder('datasets/imagenette2/train',␣
↪transform=data_transforms['train'])
```

[13]:
```
# train_dataset.targets[1]
```

[14]:
```
from fastai.vision.all import *
from sklearn.model_selection import StratifiedShuffleSplit
from torch.utils.data import Dataset
class ImagenetteDataModule(pl.LightningDataModule):
    def __init__(self,batch_size, url = 'https://s3.amazonaws.com/
↪fast-ai-imageclas/imagenette2.tgz', img_folder='/content', transform=None,␣
↪has_labels=False, base_folder = Path('/content/drive/MyDrive/Colab_Notebooks/
↪BUAN_6382_Applied_DeepLearning/Data'),
                 small_subset = False, num_samples=1000,seed=42,split_frac = 0.
↪5):
        super().__init__()
        self.url = url
        self.img_folder = img_folder  # Convert to Path object for filesystem␣
↪safety
        self.transform = transform  # Store the transform function
        self.has_labels = has_labels
        self.base_folder = base_folder
        self.batch_size = batch_size
        self.image_files = img_folder
        self.small_subset = small_subset
        self.num_samples = num_samples
        self.seed = seed
```

```
        self.n_workers = os.cpu_count() -1
        self.split_frac = split_frac

    def prepare_data(self):
      d = FastDownload(base=self.img_folder, archive='archive', data='datasets')
      data_folder_imagenette2 = d.get(self.url)
      self.image_files = get_image_files('datasets/imagenette2/')
      # tgz_url = self.img_folder/'archive/imagenette2.tgz'
      # data_folder_imagenette2 = d.extract(tgz_url)
      # image_files = get_image_files(data_folder_imagenette2)
      # path = untar_data(self.url)
      # self.data_folder_imagenette2 = path

    def setup(self, stage: str):
        self.train_dataset = ImageFolder('datasets/imagenette2/train',␣
↪transform=data_transforms['train'])
        self.val_dataset = ImageFolder('datasets/imagenette2/val',␣
↪transform=data_transforms['test'])
        self.val_dataset, self.test_dataset = split_dataset(self.val_dataset,␣
↪fraction = self.split_frac)
        if  self.small_subset:
            self.train_labels = self.train_dataset.targets
            # val_labels = self.val_dataset.targets
            # test_labels = self.test_dataset.targets
            # train_labels = train_labels[self.val_dataset.indices]
            # valid_labels = val_labels[self.train_dataset.indices]
            # test_labels = test_labels[self.test_dataset.indices]
            # print(self.test_dataset.targets)
            self.trainset_transformed = get_stratified_subset(dataset = self.
↪train_dataset,num_samples_small= self.num_samples,labels = self.train_labels)
            self.validset_transformed = get_stratified_subset(dataset = self.
↪val_dataset,num_samples_small= self.num_samples)
            self.testset_transformed = get_stratified_subset(dataset = self.
↪test_dataset,num_samples_small= self.num_samples)

    def train_dataloader(self):
        if self.small_subset:
            return DataLoader(self.trainset_transformed, batch_size=self.
↪batch_size, shuffle=True,drop_last=True, num_workers=self.n_workers )
        else:
            return DataLoader(self.train_dataset, batch_size=self.batch_size,␣
↪shuffle=True,drop_last=True, num_workers=self.n_workers )

    def val_dataloader(self):
        if self.small_subset:
```

```
            return DataLoader(self.validset_transformed, batch_size=self.
↪batch_size,num_workers=self.n_workers)
        else:
            return DataLoader(self.val_dataset, batch_size=self.
↪batch_size,num_workers=self.n_workers)

    def test_dataloader(self):
        if self.small_subset:
            return DataLoader(self.testset_transformed, batch_size=self.
↪batch_size,num_workers=self.n_workers)
        else:
            return DataLoader(self.test_dataset, batch_size=self.
↪batch_size,num_workers=self.n_workers)


    # def _get_stratified_subset(dataset, num_samples, seed=42):
    #     if seed is not None:
    #         random.seed(seed)

    #     sss = StratifiedShuffleSplit(n_splits=1, test_size=num_samples,␣
↪random_state=seed)
    #     for _, subset_indices in sss.split(np.zeros(len(dataset.
↪classes)),dataset.classes):
    #         break
    #     return Subset(dataset, subset_indices)


    # def _get_stratified_subset(dataset, num_samples, seed=42):
    #     if seed is not None:
    #         random.seed(seed)

    #     # Step 1: Identify label distribution
    #     label_to_indices = defaultdict(list)
    #     for idx, (_, label) in enumerate(dataset):
    #         label_to_indices[label].append(idx)

    #     # Step 2: Calculate proportions and initialize subset indices list
    #     proportions = {label: len(indices) / len(dataset) for label, indices␣
↪in label_to_indices.items()}
    #     subset_indices = []

    #     # Step 3: Sample according to proportion
    #     for label, indices in label_to_indices.items():
    #         num_samples_for_label = round(proportions[label] * num_samples)
    #         subset_indices += random.sample(indices, num_samples_for_label)
```

```
    #     # Step 4: Combine samples
    #     return torch.utils.data.Subset(dataset, subset_indices)

    # def _get_stratified_subset(self, dataset,num_samples,seed=42):
    #   _, subset_indices = train_test_split(
    #     range(len(dataset.class_to_idx)),  # Just indices, not the actual data
    #     test_size=self.num_samples,
    #     stratify=dataset.class_to_idx,
    #     random_state=self.seed
    #     )
    #   return Subset(dataset, subset_indices)
```

5 Data Transforms

[15]:
```
# DO NOT CHANGE THIS CELL
# YOU HAVE TO USE THESE TRANSFORMATIONS
# USE data_transforms['TRAIN] FOR TRAINING SET AND data_transforms['test] FOR
 ↪BOTH VALIDATION AND TEST SET
data_transforms = {
    "train": transforms.Compose(
        [
            # Resize the shorter side of the image to (500, 375) pixels
            transforms.Resize((500, 375)),
            # Convert image to PyTorch tensor
            transforms.ToTensor(),
            # Normalize tensor values to range [-1, 1]
            transforms.Normalize((0.5,), (0.5,)),
        ]
    ),
    "test": transforms.Compose(
        [
            # Resize the shorter side of the image to 640 pixels
            transforms.Resize((500, 375)),
            # Convert image to PyTorch tensor
            transforms.ToTensor(),
            # Normalize tensor values to range [-1, 1]
            transforms.Normalize((0.5,), (0.5,)),
        ]
    ),
}
```

6 Instantiate Data Module

- **Use 50% of the images in the `val` folder for validation dataset and 50% for test dataset.**
- **Use a samll subset (1000 samples) for both validation and training dataset**

- **Use a batch size of 64.**

[16]:
```
torch.manual_seed(123)
# Use batch size of 64
# Select only 1000
dm = ImagenetteDataModule(batch_size=64,small_subset = True,␣
↪num_samples=1000,split_frac=0.5)
```

[17]:
```
dm.prepare_data()
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

[18]:
```
dm.setup(stage='fit')
check_loader(dm.train_dataloader())
```

```
 Batch Number: 1 | Batch size: 64 | x shape: torch.Size([64, 3, 500, 375]) | y
shape: torch.Size([64])
 Batch Number: 2 | Batch size: 64 | x shape: torch.Size([64, 3, 500, 375]) | y
shape: torch.Size([64])
 Batch Number: 3 | Batch size: 64 | x shape: torch.Size([64, 3, 500, 375]) | y
shape: torch.Size([64])

Labels from current batch
 tensor([6, 5, 2, 1, 8, 3, 9, 1, 0, 2, 6, 8, 3, 0, 4, 1, 9, 9, 1, 4, 3, 2, 5, 9,
        1, 5, 4, 2, 3, 2, 1, 1, 4, 3, 8, 2, 7, 6, 6, 4, 2, 8, 6, 1, 4, 1, 5, 0,
        4, 3, 0, 3, 0, 9, 5, 4, 6, 2, 5, 2, 7, 1, 5, 0])
```

[19]:
```
len(dm.train_dataloader())
```

[19]: 15

7 LightningModule

- **Use SGD optimizer.**
- **Print accuracy and loss at the end of every epoch.**
- **Log loss and accuracy at the end of each epoch or both training and validation.**
- **Also log training loss at every 10 steps.**

[20]:
```
class LightningModel(pl.LightningModule):
    def __init__(self, model, learning_rate):
        super().__init__()
        self.learning_rate = learning_rate
        self.model = model
        self.train_acc = torchmetrics.Accuracy(task="multiclass",␣
↪num_classes=10)
        self.val_acc = torchmetrics.Accuracy(task="multiclass", num_classes=10)
```

```
      self.test_acc = torchmetrics.Accuracy(task="multiclass", num_classes=10)

  def forward(self, x):
      return self.model(x)

  def _shared_step(self, batch):
    inputs, labels = batch
    output = self(inputs)
    loss = F.cross_entropy(output, labels)
    predicted_labels = torch.argmax(output, dim=1)
    return loss, labels, predicted_labels


  def training_step(self, batch, batch_idx):
    loss, labels, predicted_labels = self._shared_step(batch)

    if batch_idx % 10 == 0:
          self.log("train_loss_step", loss, on_step=True, on_epoch=False)

    self.train_acc(predicted_labels, labels)

      # Log for the epoch average
    self.log("train_loss_epoch", loss, on_step=False, on_epoch=True)
    self.log("train_acc", self.train_acc, prog_bar=True, on_step=False, on_epoch=True)

    return loss


  def validation_step(self, batch, batch_idx):
    loss, labels, predicted_labels = self._shared_step(batch)
    self.log("val_loss", loss, on_epoch=True, on_step=False)
    self.val_acc(predicted_labels, labels)
    self.log("val_acc", self.val_acc, prog_bar=True, on_epoch=True, on_step=False)


  def on_train_epoch_end(self):
    metrics = self.trainer.callback_metrics

      # Using 'train_loss_epoch' to get the average loss for the epoch
    print(f"Train_Loss: {metrics['train_loss_epoch']:.2f}, Train_Acc: {metrics['train_acc']:.2f}")



  def on_validation_epoch_end(self):
```

```
        metrics = self.trainer.callback_metrics
        epoch_num = self.current_epoch
        print(f"Epoch {epoch_num + 1}: Val_Loss: {metrics['val_loss']:.2f},␣
 ↪Val_Acc: {metrics['val_acc']:.2f}" ,end=" | ", flush=True)



    def configure_optimizers(self):
        optimizer = torch.optim.SGD(self.parameters(), lr=self.learning_rate)
        return optimizer


    def test_step(self, batch, batch_idx):
        loss, labels, predicted_labels = self._shared_step(batch)
        self.test_acc(predicted_labels, labels)
        self.log("test_acc", self.test_acc)
```

[21]:
```
# DO NOT CHANGE THIS CELL
# Define the model architecture and training parameters
num_features = 3*500*375
hidden_dim1 = 300
hidden_dim2 = 200
hidden_dim3 = 100
num_classes = 10
epochs = 10
learning_rate = 0.03


# Set a random seed for reproducibility
torch.manual_seed(42)

# Create the neural network model
model = PytorchMLPSkip(num_features, hidden_dim1, hidden_dim2, hidden_dim3,␣
 ↪num_classes)

lightning_model = LightningModel(model=model, learning_rate=learning_rate)


# Determine the computing device (CPU or GPU) to use
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

summary(model, (1, 3, 500, 375))
```

[21]:
```
================================================================================
==========
Layer (type:depth-idx)                   Output Shape              Param #
================================================================================
```

```
==========
PytorchMLPSkip                                       [1, 10]                   --
 Flatten: 1-1                                       [1, 562500]               --
 Linear: 1-2                                        [1, 300]                  168,750,300
 ReLU: 1-3                                          [1, 300]                  --
 Linear: 1-4                                        [1, 200]                  60,200
 ReLU: 1-5                                          [1, 200]                  --
 Linear: 1-6                                        [1, 100]                  50,100
 ReLU: 1-7                                          [1, 100]                  --
 Linear: 1-8                                        [1, 10]                   3,010
================================================================================
==========
Total params: 168,863,610
Trainable params: 168,863,610
Non-trainable params: 0
Total mult-adds (M): 168.86
================================================================================
==========
Input size (MB): 2.25
Forward/backward pass size (MB): 0.00
Params size (MB): 675.45
Estimated Total Size (MB): 677.71
================================================================================
==========
```

[22]:

```python
# DO NOT CHANGE THIS CELL
from pytorch_lightning.loggers import CSVLogger
trainer = pl.Trainer(
        max_epochs=10,
        accelerator="auto",  # set to "auto" or "gpu" to use GPUs if available
        devices="auto",  # Uses all available GPUs if applicable
        deterministic=True,
        log_every_n_steps = 10,
        logger=CSVLogger(save_dir=model_folder/'logs', name="skip_two_layer")
    )
```

```
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used:
True
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU
cores
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
```

[23]:

```python
# DO NOT CHANGE THIS CELL
trainer.fit(
        model=lightning_model,
        datamodule=dm,
```

```
)
```

```
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]
INFO:pytorch_lightning.callbacks.model_summary:
  | Name      | Type               | Params
-------------------------------------------------
0 | model     | PytorchMLPSkip     | 168 M
1 | train_acc | MulticlassAccuracy | 0
2 | val_acc   | MulticlassAccuracy | 0
3 | test_acc  | MulticlassAccuracy | 0
-------------------------------------------------
168 M     Trainable params
0         Non-trainable params
168 M     Total params
675.454   Total estimated model params size (MB)
```

```
Sanity Checking: 0it [00:00, ?it/s]
```

```
Epoch 1: Val_Loss: 2.30, Val_Acc: 0.11 |
```

```
Training: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]
```

```
Epoch 1: Val_Loss: 2.30, Val_Acc: 0.22 | Train_Loss: 2.39, Train_Acc: 0.20
```

```
Validation: 0it [00:00, ?it/s]
```

```
Epoch 2: Val_Loss: 2.16, Val_Acc: 0.26 | Train_Loss: 1.90, Train_Acc: 0.35
```

```
Validation: 0it [00:00, ?it/s]
```

```
Epoch 3: Val_Loss: 2.01, Val_Acc: 0.32 | Train_Loss: 1.59, Train_Acc: 0.49
```

```
Validation: 0it [00:00, ?it/s]
```

```
Epoch 4: Val_Loss: 2.00, Val_Acc: 0.34 | Train_Loss: 1.56, Train_Acc: 0.52
```

```
Validation: 0it [00:00, ?it/s]
```

```
Epoch 5: Val_Loss: 2.04, Val_Acc: 0.32 | Train_Loss: 1.13, Train_Acc: 0.67
```

```
Validation: 0it [00:00, ?it/s]
```

```
Epoch 6: Val_Loss: 2.17, Val_Acc: 0.31 | Train_Loss: 0.98, Train_Acc: 0.72
```

```
Validation: 0it [00:00, ?it/s]
```

```
Epoch 7: Val_Loss: 2.73, Val_Acc: 0.30 | Train_Loss: 0.77, Train_Acc: 0.78
```

```
Validation: 0it [00:00, ?it/s]
```

```
Epoch 8: Val_Loss: 2.43, Val_Acc: 0.31 | Train_Loss: 0.59, Train_Acc: 0.82
```

```
Validation: 0it [00:00, ?it/s]
```

```
Epoch 9: Val_Loss: 2.47, Val_Acc: 0.31 | Train_Loss: 0.42, Train_Acc: 0.89
```

```
Validation: 0it [00:00, ?it/s]

Epoch 10: Val_Loss: 2.46, Val_Acc: 0.31 | Train_Loss: 0.28, Train_Acc: 0.93

INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped:
`max_epochs=10` reached.
```

7.1 Plot Losses

[24]:
```python
# DO NOT CHANGE THIS CELL
file = f"{trainer.logger.log_dir}/metrics.csv"
file
```

[24]:
```
'/content/drive/MyDrive/Colab_Notebooks/BUAN_6382_Applied_DeepLearning/Data/logs
/skip_two_layer/version_6/metrics.csv'
```

[25]:
```python
# DO NOT CHANGE THIS CELL
plot_losses_acc(file)
```