CS260R Reinforcement Learning Assignment 0: Jupyter Notebook usage and assignment submission workflow

CS260R 2023Fall: Reinforcement Learning. Department of Computer Science at University of California, Los Angeles. Course Instructor: Professor Bolei ZHOU. Assignment author: Zhenghao PENG, Yiran WANG.

You are asked to finish four tasks:

1. Fill in your name and University ID in the next cell.
2. Install pytorch and finish the `Kindergarten Pytorch` section.
3. Run all cells and save this notebook **as a PDF file**.
4. Compress this folder `assignment0` **as a ZIP file** and submit **the PDF file and the ZIP file separately as two files** in BruinLearn.

In [1]:

```
# TODO: Fill your name and UID here
my_name = "Yuchen Wang"
my_student_id = "805846047"
```

In [2]:

```
# Run this cell without modification

text = "Oh, I finished this assignment! I am {} ({})".format(my_name, my_student
print(text)
with open("{}.txt".format(text), "w") as f:
    f.write(text)
```

```
Oh, I finished this assignment! I am Yuchen Wang (805846047)
```

Kindergarten Pytorch

1. Please install pytorch in your virtual environment following the instruction: https://pytorch.org/get-started/locally/ (https://pytorch.org/get-started/locally/).

   ```
   pip install torch torchvision
   ```

2. If you are not familiar with Pytorch, please go through the tutorial in official website (https://pytorch.org/tutorials/beginner/basics/intro.html) until you can understand the quick start tutorial (https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html).
3. The following code is copied from the quick start tutorial (https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html), please solve all `TODO` s and print the result in the cells before generating the PDF file.

Prepare data

In [1]:

```
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor

# Download training data from open datasets.
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)

# Download test data from open datasets.
test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)

batch_size = 64

# Create data loaders.
train_dataloader = DataLoader(training_data, batch_size=batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size)

```

Define model

In [2]:

```
# Get cpu, gpu or mps device for training.
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")

# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)
print(model)
```

```
Using mps device
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

Define training and test pipelines

In [3]:

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)

def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_los
```

Run the training and test pipelines

In [4]:

```python
epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)
print("Done!")
```

```
Epoch 1
-------------------------------
loss: 2.313366  [   64/60000]
loss: 2.296317  [ 6464/60000]
loss: 2.269812  [12864/60000]
loss: 2.258034  [19264/60000]
loss: 2.247507  [25664/60000]
loss: 2.220511  [32064/60000]
loss: 2.230436  [38464/60000]
loss: 2.193467  [44864/60000]
loss: 2.185473  [51264/60000]
loss: 2.153778  [57664/60000]
Test Error:
 Accuracy: 35.0%, Avg loss: 2.142713

Epoch 2
-------------------------------
loss: 2.158500  [   64/60000]
loss: 2.147049  [ 6464/60000]
loss: 2.077717  [12864/60000]
loss: 2.093690  [19264/60000]
loss: 2.047469  [25664/60000]
loss: 1.983252  [32064/60000]
loss: 2.018713  [38464/60000]
loss: 1.926184  [44864/60000]
loss: 1.937092  [51264/60000]
loss: 1.865396  [57664/60000]
Test Error:
 Accuracy: 55.9%, Avg loss: 1.853672

Epoch 3
-------------------------------
loss: 1.890282  [   64/60000]
loss: 1.857141  [ 6464/60000]
loss: 1.727536  [12864/60000]
loss: 1.780272  [19264/60000]
loss: 1.671083  [25664/60000]
loss: 1.627853  [32064/60000]
loss: 1.662363  [38464/60000]
loss: 1.547891  [44864/60000]
loss: 1.584457  [51264/60000]
loss: 1.484833  [57664/60000]
Test Error:
 Accuracy: 61.4%, Avg loss: 1.490892

Epoch 4
-------------------------------
loss: 1.555492  [   64/60000]
loss: 1.525028  [ 6464/60000]
loss: 1.365754  [12864/60000]
loss: 1.449020  [19264/60000]
loss: 1.334438  [25664/60000]
loss: 1.336161  [32064/60000]
loss: 1.360559  [38464/60000]
loss: 1.269854  [44864/60000]
loss: 1.314767  [51264/60000]
loss: 1.218156  [57664/60000]
Test Error:
 Accuracy: 64.1%, Avg loss: 1.235937

Epoch 5
```

```
-------------------------------
loss: 1.307462  [   64/60000]
loss: 1.297187  [ 6464/60000]
loss: 1.124118  [12864/60000]
loss: 1.233752  [19264/60000]
loss: 1.116695  [25664/60000]
loss: 1.145562  [32064/60000]
loss: 1.174787  [38464/60000]
loss: 1.096280  [44864/60000]
loss: 1.145186  [51264/60000]
loss: 1.061263  [57664/60000]
Test Error: 
 Accuracy: 65.4%, Avg loss: 1.076073 

Done!
```

Save model

In [5]:

```
torch.save(model.state_dict(), "model.pth")
print("Saved PyTorch Model State to model.pth")
```

```
Saved PyTorch Model State to model.pth
```

Load model and run the inference

In [6]:

```
model = NeuralNetwork().to(device)
model.load_state_dict(torch.load("model.pth"))
```

Out[6]:

```
<All keys matched successfully>
```

In [7]:

```
classes = [
    "T-shirt/top",
    "Trouser",
    "Pullover",
    "Dress",
    "Coat",
    "Sandal",
    "Shirt",
    "Sneaker",
    "Bag",
    "Ankle boot",
]

model.eval()
x, y = test_data[0][0], test_data[0][1]
with torch.no_grad():
    x = x.to(device)
    pred = model(x)
    predicted, actual = classes[pred[0].argmax(0)], classes[y]
    print(f'Predicted: "{predicted}", Actual: "{actual}"')
```

```
Predicted: "Ankle boot", Actual: "Ankle boot"
```