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

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

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 export 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 [6]:

```
# TODO: Fill your name and UID here
my_name = "Yuyang Gong"
my_student_id = "406405460"
```

In [7]:

```
# Run this cell without modification

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

```
Oh, I finished this assignment! I am Yuyang Gong (406405460)
```

Kindergarten Pytorch

1. Please install pytorch in your virtual environment following the instruction: 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 until you can understand the quick start tutorial.
3. The following code is copied from the quick start tutorial, please solve all `TODO` s and print the result in the cells before generating the PDF file.

In [8]:

```
# You can also run this cell in notebook:
# !pip install torch torchvision
```

Prepare data

In [9]:

```
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 [10]:

```
# 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()

        # TODO: Define the self.linear_relu_stack by uncommenting next few lines
        # and understand what they mean
        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 cuda 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 [11]:

```
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

        # TODO: Uncomment next three lines and understand what they mean
        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()

            # TODO: Uncomment next line and understand what it means
            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_loss:>
```

Run the training and test pipelines

In [12]:

```
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: 2.306682  [   64/60000]
loss: 2.294751  [ 6464/60000]
loss: 2.275012  [12864/60000]
loss: 2.265236  [19264/60000]
loss: 2.253177  [25664/60000]
loss: 2.213197  [32064/60000]
loss: 2.218814  [38464/60000]
loss: 2.178283  [44864/60000]
loss: 2.185725  [51264/60000]
loss: 2.159421  [57664/60000]
Test Error: 
 Accuracy: 43.7%, Avg loss: 2.145947 

Epoch 2
-------------------------------
loss: 2.159055  [   64/60000]
loss: 2.146083  [ 6464/60000]
loss: 2.090998  [12864/60000]
loss: 2.107200  [19264/60000]
loss: 2.044127  [25664/60000]
loss: 1.981666  [32064/60000]
loss: 2.017932  [38464/60000]
loss: 1.927647  [44864/60000]
loss: 1.950258  [51264/60000]
loss: 1.872963  [57664/60000]
Test Error: 
 Accuracy: 54.9%, Avg loss: 1.865887 

Epoch 3
-------------------------------
loss: 1.902189  [   64/60000]
loss: 1.866654  [ 6464/60000]
loss: 1.757522  [12864/60000]
loss: 1.801751  [19264/60000]
loss: 1.669057  [25664/60000]
loss: 1.633178  [32064/60000]
loss: 1.668319  [38464/60000]
loss: 1.561932  [44864/60000]
loss: 1.600659  [51264/60000]
loss: 1.493386  [57664/60000]
Test Error: 
 Accuracy: 61.0%, Avg loss: 1.504016 

Epoch 4
-------------------------------
loss: 1.573032  [   64/60000]
loss: 1.534886  [ 6464/60000]
loss: 1.396047  [12864/60000]
loss: 1.466018  [19264/60000]
loss: 1.330662  [25664/60000]
loss: 1.340911  [32064/60000]
loss: 1.360982  [38464/60000]
loss: 1.281906  [44864/60000]
loss: 1.320264  [51264/60000]
```

```
loss: 1.222195  [57664/60000]
Test Error: 
 Accuracy: 63.7%, Avg loss: 1.241347 

Epoch 5
-------------------------------
loss: 1.318838  [   64/60000]
loss: 1.299932  [ 6464/60000]
loss: 1.143537  [12864/60000]
loss: 1.243943  [19264/60000]
loss: 1.111459  [25664/60000]
loss: 1.144135  [32064/60000]
loss: 1.168695  [38464/60000]
loss: 1.104421  [44864/60000]
loss: 1.144678  [51264/60000]
loss: 1.061474  [57664/60000]
Test Error: 
 Accuracy: 65.0%, Avg loss: 1.077101 

Done!
```

Save model

In [13]:

```python
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 [14]:

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

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_23732\2085806346.py:2: FutureWarning: Yo
u are using `torch.load` with `weights_only=False` (the current default value), whic
h uses the default pickle module implicitly. It is possible to construct malicious p
ickle data which will execute arbitrary code during unpickling (See https://github.c
om/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a fu
ture release, the default value for `weights_only` will be flipped to `True`. This l
imits the functions that could be executed during unpickling. Arbitrary objects will
no longer be allowed to be loaded via this mode unless they are explicitly allowlist
ed by the user via `torch.serialization.add_safe_globals`. We recommend you start se
tting `weights_only=True` for any use case where you don't have full control of the 
loaded file. Please open an issue on GitHub for any issues related to this experimen
tal feature.
  model.load_state_dict(torch.load("model.pth"))
```

Out[14]:

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

In [15]:

```python
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"
```