

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 Bo Lei 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 [1]: # TODO: Fill your name and UID here
my_name = "Mubai Hua"
my_student_id = "905571231"
```

```
In [2]: # Run this cell without modification

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

Oh, I finished this assignment! I am Mubai Hua (905571231)

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 [3]: # You can also run this cell in notebook:

!pip install torch torchvision
```

Requirement already satisfied: torch in c:\users\r9000x-2021\anaconda3\envs\cs260r\lib\site-packages (2.5.1+cu118)

Requirement already satisfied: torchvision in c:\users\r9000x-2021\anaconda3\envs\cs260r\lib\site-packages (0.20.1)

Requirement already satisfied: filelock in c:\users\r9000x-2021\anaconda3\envs\cs260r\lib\site-packages (from torch) (3.16.1)

Requirement already satisfied: typing-extensions>=4.8.0 in c:\users\r9000x-2021\anaconda3\envs\cs260r\lib\site-packages (from torch) (4.12.2)

Requirement already satisfied: networkx in c:\users\r9000x-2021\anaconda3\envs\cs260r\lib\site-packages (from torch) (3.4.2)

Requirement already satisfied: Jinja2 in c:\users\r9000x-2021\anaconda3\envs\cs260r\lib\site-packages (from torch) (3.1.5)

Requirement already satisfied: fsspec in c:\users\r9000x-2021\anaconda3\envs\cs260r\lib\site-packages (from torch) (2024.12.0)

Requirement already satisfied: sympy==1.13.1 in c:\users\r9000x-2021\anaconda3\envs\cs260r\lib\site-packages (from torch) (1.13.1)

Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\r9000x-2021\anaconda3\envs\cs260r\lib\site-packages (from sympy==1.13.1->torch) (1.3.0)

Requirement already satisfied: numpy in c:\users\r9000x-2021\anaconda3\envs\cs260r\lib\site-packages (from torchvision) (2.2.1)

Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in c:\users\r9000x-2021\anaconda3\envs\cs260r\lib\site-packages (from torchvision) (11.1.0)

Requirement already satisfied: MarkupSafe>=2.0 in c:\users\r9000x-2021\anaconda3\envs\cs260r\lib\site-packages (from Jinja2->torch) (3.0.2)

Prepare data

```
In [4]: 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 [5]: # 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 [6]: 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:>0.1f}")

```

Run the training and test pipelines

```

In [7]: 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: 2.300956 [64/60000]
loss: 2.290339 [6464/60000]
loss: 2.262995 [12864/60000]
loss: 2.261135 [19264/60000]
loss: 2.251179 [25664/60000]
loss: 2.212415 [32064/60000]
loss: 2.228665 [38464/60000]
loss: 2.184602 [44864/60000]
loss: 2.182182 [51264/60000]
loss: 2.172390 [57664/60000]

Test Error:

Accuracy: 45.9%, Avg loss: 2.150298

Epoch 2

loss: 2.156924 [64/60000]
loss: 2.151670 [6464/60000]
loss: 2.089000 [12864/60000]
loss: 2.108562 [19264/60000]
loss: 2.069667 [25664/60000]
loss: 2.005252 [32064/60000]
loss: 2.033814 [38464/60000]
loss: 1.953695 [44864/60000]
loss: 1.953263 [51264/60000]
loss: 1.902689 [57664/60000]

Test Error:

Accuracy: 60.2%, Avg loss: 1.887714

Epoch 3

loss: 1.919424 [64/60000]
loss: 1.892962 [6464/60000]
loss: 1.771802 [12864/60000]
loss: 1.812960 [19264/60000]
loss: 1.714339 [25664/60000]
loss: 1.663552 [32064/60000]
loss: 1.681090 [38464/60000]
loss: 1.579933 [44864/60000]
loss: 1.600514 [51264/60000]
loss: 1.507567 [57664/60000]

Test Error:

Accuracy: 60.8%, Avg loss: 1.516864

Epoch 4

loss: 1.584322 [64/60000]
loss: 1.549750 [6464/60000]
loss: 1.394725 [12864/60000]
loss: 1.469310 [19264/60000]
loss: 1.353618 [25664/60000]
loss: 1.345411 [32064/60000]
loss: 1.359520 [38464/60000]
loss: 1.280375 [44864/60000]
loss: 1.316883 [51264/60000]

```
loss: 1.224042 [57664/60000]
Test Error:
  Accuracy: 63.9%, Avg loss: 1.246329
```

Epoch 5

```
-----
loss: 1.323764 [ 64/60000]
loss: 1.304758 [ 6464/60000]
loss: 1.136304 [12864/60000]
loss: 1.245611 [19264/60000]
loss: 1.122009 [25664/60000]
loss: 1.142868 [32064/60000]
loss: 1.165937 [38464/60000]
loss: 1.099485 [44864/60000]
loss: 1.141156 [51264/60000]
loss: 1.063916 [57664/60000]
Test Error:
  Accuracy: 65.0%, Avg loss: 1.081099
```

Done!

Save model

```
In [8]: 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 [9]: model = NeuralNetwork().to(device)
        model.load_state_dict(torch.load("model.pth"))
```

C:\Users\R9000X-2021\AppData\Local\Temp\ipykernel_19224\2085806346.py:2: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See <https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models> for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits 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 allowed by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `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 experimental feature.

```
model.load_state_dict(torch.load("model.pth"))
```

Out[9]: <All keys matched successfully>

```
In [11]: 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"