MNIST

Scenarie beskrivelse

Efter et jordskælv skabte kaos i ”BigStyles” tøjlager har de hyret dig til at hjælpe med at sortere deres kasser. De vil helst ikke til at åbne kasserne og forsegle dem igen, så de har sat et X-Ray system på der kan tage billeder af tøjet. Din opgave er nu at lave en machine learning algoritme der kan tage X-Ray billeder og fortælle hvilken type tøj artikel det er.
Det data du har fået tilgængeligt, har fire dele:
to træningssæt
train-images-idx3-ubyte og
train-labels-idx1-ubyte
samt to data sæt til at teste præcisionen:
t10k-images-idx3-ubyte og
t10k-labels-idx1-ubyte.

Opgavevejledning

Opsætning

- Start med at lave den mappe struktur du ønsker, f.eks. som nedestående. Dog skal data mappen have præcis denne struktur for at pytorch kan læse data ind

```
your_project_folder
    ├── .gitignore
    ├── requirements.txt
    ├── README.md
    ├── .venv
    ├── data
    │   └── FashionMNIST
    │       └── raw
    │           ├── train-images-idx3-ubyte
    │           ├── train-labels-idx1-ubyte
    │           ├── t10k-images-idx3-ubyte
    │           └── t10k-labels-idx1-ubyte
    └── src
        └── main.py
```

- Opret en config fil. Du vælger selv om det er json, .env eller lignende. Hvis man foretrækker det, kan man lave commandline interfaces. I config filen skal du specificere tre variabler:
  - Epochs:
    - Antal gange systemet træner.
    - Sæt den til **5**.
  - Learning_rate:
    - Den værdi systemet kan ændre sine vægtninge med
    - Sæt den til **1e-3**.
  - Batch_size:
    - Den mængde data systemet læser af gangen.
    - Sæt den til **64**.
- Opret en main.py fil, der loader config filerne.

Data loading og præparation

- Pytorch har en specifik klasse til at loade data fra dette datasæt ind: torchvision.datasets.FashionMNIST.
  - Brug klassen som beskrevet for at definerer to datasets. Et til træning og et andet til testning.
- Klassen tager imod to forskellige transform argumenter, *transform* og *target_transform*. Der findes mange forskellige former for transformationer, der kan gives som disse argumenter. Begge argumenter er optional og vi bruger kun det første, *transform*.
  - Specificer i begge datasæt, at de skal bruge ***torchvision.transforms.ToTensor()***. På den måde bliver billederne transformeret til tensorer.
- Datasættende er nu læst ind. Dog skal der præciseres en loader, der definerer hvor meget data netværket skal håndtere af gangen. Lav derfor en dataloader til begge datasæt. Som batch_size, skal den bruge den værdi du definerede i opsætningen.

Det neurale netværk

- I din src mappe skal du oprette en fil der hedder network.py.
  - Inde i filen skal du lave en klasse der repræsenterer dit neurale netværk.
  - Opret en klasse der hedder neural_network eller lignende. Det er vigtigt at den nedarver fra torch.nn.Module. Dette kan gøres ved

```
class neural_network(nn.Module):
```

- Klassen skal have mindst to funktioner. En ***init*** og en ***forward*** funktion.
  - Init funktionen sørger for at sætte klassen op.
  - Forward funktionen fortæller hvordan systemet skal behandle data.
- ***Init*** funktionen skal se ud som følgende

```
def __init__(self) -> none:
    super().__init__()
    self.flatten: Flatten = nn.Flatten()
    self.network_stack: Sequential = nn.Sequential(
        nn.Linear(in_features = 28*28, out_features = 512),
        nn.ReLU(),
        nn.Linear(in_features = 512, out_features = 10)
    )
```

- ***super.init()*** sørger for at klassen ***nn.Module*** bliver initialiseret korrekt. ***Flatten*** sørger for at tensoren bliver 1 dimensionel. ***Network_stack*** er selve strukturen af dit neurale netværk. Den består af følgende:
  - ***nn.Linear*** er et lineært lag, der tager billedet og reducerer det ned til 512 værdier
  - ***nn.ReLU*** er activation funktionen Rectified Liniear Unit
  - Endnu et ***nn.Linear*** lag, der yderligere reducerer data ned til 10 grupper
- ***Forward*** funktionen skal se ud som følgende

```
def forward(self,x):
    x = self.flatten()
    output = self.network_stack(x)
    return output
```

- Det vil sige at hver gang din model bliver kaldt med et batch af data, vil den først flatten, og derefter køre den igennem netværket.

Gør systemet klar til test

I din main.py skal vi definere 4 objekter.

- Systemet skal have en enhed den kan træne på, ofte en GPU. Det er dog ikke altid der er en GPU til rådighed (specielt ikke på disse bærbare). Man kan derfor definere en fall-back mulighed til CPU'en, som følgende.

```
device = torch.accelerator.current_accelerator().type if
torch.accelerator.is_available() else "cpu" # skal stå på en linie
```

- Systemet skal have din model. Den kan importeres og instantieres via:

```
model = neural_network.to(device)
```

- Systemet skal have en optimizer. For nu bruger vi en stochastic gradient descent (SGD) algoritme. Lav den ved:

```
optimizer = torch.optim.SGD(params = model.parameters(), lr = learning_rate)
```

- Til sidst skal systemet kende til en loss function. For nu bruger vi cross entropy loss. Lav den ved:

```
loss_fn = nn.CrossEntropyLoss()
```

Lav trænings og test loops

- Kig her for opsætning

Træn og test

- Vi kan nu træne og teste modellen. Lav derfor et loop der kører epochs antal gange. Det kan eventuelt se således ud:

```
for t in range(epochs):
    print(f"Epoch {t+1}\n-----------------")
    train_loop(
        train_dataloader,
        model=model,
        loss_fn=loss_fn,
        optimizer=optimizer
    )
    test_loop(
        test_dataloader,
        model=model,
        loss_fn=loss_fn
    )
```

Gem trænet model

- Når modelen er blevet trænet og testet vil man typisk gemme sin model, således at man kan bruge den senere. Dette kan gøres på to måder:
  - Gem kun vægtene
  - Gem hele modelen
- Hvilken man bruger variere. Dog vil det mest fordelagtige værer at kun gemme vægtene
- Gemning og loading af vægtene kan gøres på denne måde

```
model_file = "model.pt"
model = neural_network()
if os.path.exist(model_file):
    model.load_state_dict(torch.load(model_file, weights_only=True))
    model.eval
```

```
.
.
torch.save(model.state_dict,model_file)
```

- Dog vil vi bemærke at model filen ikke har en bestemt type extension. Her bruger vi .pt for pytorch
- Gemning og loading af hele modelen kan gøres på denne måde

```
model_file = "model.pt"
if os.path.exist(model_file):
    model = torch.load(model_file,weights_only=False)
else:
    model = neural_network()
.
.
torch.save(model,model_file)
```