Aufgabe 5: PyTorch-Einführung

Die Aufgabe führt in die Nutzung von PyTorch ein, ein Open-Source-Framework zum Lernen der Parameter und zur Nutzung neuronaler Netze. PyTorch ist insbesondere für seine Benutzerfreundlichkeit und Flexibilität bei der Entwicklung von Deep-Learning-Modellen bekannt.

Die Aufgabe besteht aus drei Teilen: Im ersten Teil geht es um den "Dataloader". Im zweiten Teil lernen Sie den Entwurf einfacher tiefer neuronaler Netze genauer kennen. Im dritten Teil nutzen Sie `PyTorch`, um Beispieldaten zu klassifizieren. Sie benötigen hierfür (und für spätere Aufgaben) die Bibliothek `torch`. Möglicherweise müssen Sie dazu noch weitere Pakete mitinstallieren.
Laden Sie die Datei Aufgabe5.zip aus dem ISIS-Kurs herunter. Entpacken Sie den Inhalt des Archivs und speichern Sie Ihre Dateien in folgender Struktur:

```
./my-repository/torch_intro/dataset/train.json
./my-repository/torch_intro/dataset/dev.json
./my-repository/torch_intro/dataset/test.json
./my-repository/torch_intro/local/train.py
./my-repository/torch_intro/local/utils.py
./my-repository/torch_intro/local/model.py
./my-repository/uebung5.py
```

Als Datensatz für diese Einführung verwenden wir den VoxCeleb-Gender-Korpus. Er enthält 5994 Sprachaufnahmen unterschiedlicher Ethnien, Akzente, Berufe und Altersgruppen. Die Aufgabe besteht darin, automatisch das Geschlecht anhand vorgegebener Sprachaufnahmen zu bestimmen. Sie können diesen Korpus über den folgenden Link herunterladen: `https://tubcloud.tu-berlin.de/s/Banx6DmqNMArGbq`. Bitte beachten Sie, dass der Korpus **NICHT** im `./my-repository`-Ordner gespeichert werden sollte, da dieses Dataset zu groß für eine sinnvolle Verwaltung mit Gitlab ist, und da git ohnehin nicht für das Management von Datenbanken gedacht ist. Legen Sie bitte die Datenbank in einen lokalen Ordner. Sie können den Pfad zum VoxCeleb-Gender-Korpus in der Funktion `get_args()` in der Datei `./my-repository/uebung5.py` mit der Variable `--sourcedatadir` konfigurieren.

In dieser Übung benutzen Sie Ihren eigenen Merkmalsextraktor. Kopieren Sie dazu die Dateien `feature_extraction.py` und `tools.py` aus Ihrem Repository in die Verzeichnisse `./my-repository/torch_intro/local/`.

Weitereführende Informationen zum Deep Learning mit PyTorch finden Sie z.B. unter folgendem Link: `https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`

Dataloader

Die Metainformationen der Trainings-, Development- und Testdatensätze sind bereits separat in den JSON-Dateien im `./my-repository/torch_intro/dataset/` Ordner gespeichert. Bitte vervollständigen Sie den Code im Skript `./my-repository/torch_intro/local/utils.py`, um den Dataloader für Trainings-, Development- und Testsets zu erstellen.

5.1 Ergänzen Sie die Funktion

```
def get_data(datadir):
    ...
```

in der Datei `utils.py`, die die JSON-Datei im Ordner `datadir` als Python-Dictionary zurückgeben soll. Dieses Python-Dictionary dient zur Speicherung von Metainformationen, nämliche den Pfad der akustischen Aufnahme des Sprechers sowie die Labels zum Geschlecht des Sprechers.
Hinweis: hierfür benötigen Sie die Bibliothek `json`.

5.2 Ergänzen Sie die Unterfunktion

```
def __getitem__(self, index):
    ...
```

in der Klasse

```
class Dataloader(Dataset):
    ...
```

in der Datei `utils.py`. Die `Dataloader`-Klasse ermöglicht die Iteration über den Datensatz, um später einen einfachen Zugriff auf die Samples zu ermöglichen. Die Unterfunktion `__getitem__(self, index)` lädt die Metainformationen einer Sprachaufnahmen und extrahiert die Merkmale. Dafür sollten Sie Ihre implementierte Funktion `compute_features` aus dem Skript `feature_extraction.py` aufrufen.

Schließlich werden die extrahierten Merkmale und die Ground-Truth-Labels mithilfe von `torch.FloatTensor()` in Tensoren umgewandelt. Die akustischen Merkmale sollen die Dimensionen `[f_len, idim]` haben, wobei `f_len` die Länge der Rahmensequenz und `idim` die Dimension der Merkmale repräsentieren. Gemäß der Konfiguration in der Datei `uebung5.py` soll `idim` 60 betragen. Als Beispiel finden Sie einen fertigen Dataloader unter folgendem Link: `https://pytorch.org/tutorials/beginner/data_loading_tutorial.html`.

5.3 Die extrahierten Merkmale haben zunächst unterschiedliche Längen. Ergänzen Sie die Funktion

```
def padding(sequences):
    ...
```

in der Datei `utils.py`, um Zero-Padding anzuwenden und die akustischen Merkmale eines Batches auf die gleiche Länge (maximale Batchgröße) zu bringen.

Hilfreich ist u.U. die Funktion `torch.nn.utils.rnn.pad_sequence()`. Setzen Sie beim Funtionsaufruf den optionalen Parameter `batch_first` auf `True`.

Einfaches Feedforward-Netzwerk mit PyTorch

In diesem Aufgabenteil implementieren Sie ein einfaches neuronales Netzwerk ohne Rückkopplungen, wie es in der Vorlesung besprochen wurde.

5.4 Ergänzen Sie die Klasse

```
class Classification(torch.nn.Module):
        def __init__(self, idim, odim, hidden_dim):
            ...
        def forward(self, audio_feat):
            ...
```

Die einzelnen neuronalen Schichten sollen in der Unterfunktion `__init__()` definiert werden, dabei bezeichnet `idim` die Eingabedimension, `odim` die Ausgabedimension und `hidden_dim` die Anzahl der Neuronen in der verborgenen Schicht. Implementieren Sie bitte ein Feedforward-Netzwerk mit drei vollständig verbundenen Schichten unter Verwendung der Funktion `torch.nn.Linear()`. Auf jede vollständig verbundene Schicht sollte eine ReLU-Aktivierungsfunktion folgen. Implementieren Sie zusätzlich eine vollständig verbundene Klassifizierungsschicht, gefolgt von einer Sigmoid-Aktivierungsfunktion, für eine binäre Klassifizierungsaufgabe.

In der `forward()` Funktion sollen die Merkmale die Dimension `[BS, f_len, idim]` haben, wobei `BS` die Batchgröße ist. Nutzen Sie das erstellte neuronale Netzwerk, um die Merkmale zu mappen. Berechnen Sie dann den Mittelwert über die gemappten Merkmale entlang der Sequenzdimension mit der Funktion `torch.mean()`, um die Merkmalsdarstellung auf Sprecherebene zu erhalten. Verwenden Sie abschließend die Klassifizierungsschicht, um die Merkmale in die gewünschte Ausgabedimension umzuwandeln, und nutzen Sie die Sigmoidfunktion, um die posteriori-Wahrscheinlichkeiten zu ermitteln. Ein Beispiel für den Entwurf eines ähnlichen neuronalen Netzes finden Sie hier: `https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html`.

In diesem Beispiel haben wir nur ein einfaches Feedforward-Netzwerk mit vollständig verbundenen Schichten implementiert, es gibt jedoch noch weitere Schichttypen. Weiterführende Informationen hierzu finden Sie u.a. hier: `https://pytorch.org/docs/stable/nn.html`.

Training und Evaluierung eines Klassifikators

Sie können das Modell mithilfe des Skripts `./my-repository/torch_intro/local/train.py` trainieren, evaluieren und speichern. Das Modelltraining übernimmt die `train()` Funktion im Skript `train.py`.

5.5 Ergänzen Sie dazu die Funktion

```
def train(dataset, model, device, optimizer=None, criterion=None):
    ...
```

in der Datei `train.py`. Bitte versetzen Sie das Modell in dieser Funktion zunächst mit `model.train()` in den Trainingsmodus. Verwenden Sie das Modell, um die Posteriori-Wahrscheinlichkeiten für alle Batches zu berechnen, ermitteln Sie den jeweiligen Wert der Loss-Funktion `criterion()`, berechnen Sie die Gradienten und aktualisieren Sie die Modellparameter mit `optimizer`. Nach einem kompletten Durchlauf berechnen Sie bitte auch die Genauigkeit über den gesamten Trainingsdatensatz. Die Variable `device` hat zwei mögliche Werte: `'cpu'` oder `'cuda'`. Damit können Sie entscheiden, ob Sie Ihren Code mit der CPU oder der GPU ausführen möchten. Ein Beispiel für einen vollständigen Trainingsprozess finden Sie z.B. unter: `https://pytorch.org/tutorials/beginner/introyt/trainingyt.html`.

5.6 Ergänzen Sie die Funktion

```
def evaluation(dataset, device, model):
    ...
```

in der Datei `train.py`. Bitte versetzen Sie das Modell in dieser Funktion zunächst mit `model.eval()` in den Evaluierungsmodus. Durchlaufen Sie wieder alle Batches, um mit dem Modell die a-posteriori Wahrscheinlichkeiten $p(k|\mathbf{x})$ zu berechnen und speichern Sie die Vorhersagen, die Ground-Truth-Labels und die a-posteriori Wahrscheinlichkeiten aller Samples in einem Python-Dictionary. Berechnen Sie nach einem vollständigen Durchlauf bitte auch die Genauigkeit über den gegebenen Evaluierungsdatensatz.

Der gesamte Optimierungsprozess wird von der nachfolgend gezeigten `run()` Funktion realisiert:

```
criterion = torch.nn.BCELoss()
model = Classification(idim=40, odim=1, hidden_dim=512)
optimizer = torch.optim.Adam(model.parameters(),
                             lr=config["lr"]
                             )
for epoch in range(config["epochs"]):
    trainscore, model = train(data_loader_train,
                              model,
                              optimizer=optimizer,
                              criterion=criterion)
    evalscore, outpre, model = evaluation(data_loader_dev, model)
    OUTPUT_DIR = os.path.join(modeldir,
                        '_'.join([str(epoch),
                        str(trainscore)[:6],
                        str(evalscore)[:6]] + '.pkl')
    torch.save(model, OUTPUT_DIR) # save trained model
```

In jeder Epoche wird das Modell auf dem Trainingsset trainiert und anschließend auf dem Development-Set evaluiert, um die aktuelle Leistung des Modells zu beurteilen. Nach dem Ende aller Epochen bietet es sich an, das Modell mit der jeweils besten Leistung auf dem Development-Set zu benutzen, um damit das Testset zu evaluieren.

```
besttrainmodel, besttrainacc = find_best_model(os.listdir(modeldir))
model = torch.load(os.path.join(modeldir, besttrainmodel),
                   map_location=config["device"])
testacc, outpre, _ = evaluation(data_loader_test, model)

with open(os.path.join(resultsdir,
                   'testacc_' +
                   str(round(testacc, 6)) +
                   ".json"), 'w',
                   encoding='utf-8') as f:
    json.dump(outpre, f, ensure_ascii=False, indent=4)
```

Alle Klassifizierungsergebnisse werden unter `./my-repository/torch_intro/trained/results` abgelegt.