KIP 23/24 - Praktikum 4: Neuronale Netze

Rahmenbedingungen

Programmiersprache: Python.
Persönliche Empfehlung: PyCharm als IDE verwenden, andere Optionen wie z.B. Jupyter Notebooks aber auch legitim. Lösung, wenn z.B. kein Laptop vorhanden ist, welcher zum Praktikumstermin mitgebracht werden kann: Jupyter Notebooks auf Laborrechnern im 11. Stock oder Google Collab → Jupyter Notebooks in Google Cloud.
Implementierung: In den Aufgaben werden einige "Grundpfeiler" für die Implementierungen vorgegeben, davon ab werden euch jedoch sehr viele Freiheiten gelassen. Grundprinzipien aus Programmieren 1 und 2 und Software Engineering sollten natürlich auch in KIP Anwendung finden, so dass die Implementierung gut "lesbar", wartbar, erweiterbar etc. sein sollte.
LLM Policy: Die Verwendung von z.B. ChatGPT ist grundsätzlich erlaubt. Falls ihr dieses verwenden solltet, dokumentiert grob eure Prompts, wie gut das geklappt hat, wo ihr Verbesserungen vorgenommen habt und ob ihr meint, dadurch einen Zeitgewinn erhalten zu haben.
Persönliche Einschätzung: Der Zeitgewinn wird eher gering ausfallen, da der generierte Code auch verstanden und häufig verbessert werden muss. Dadurch, dass eben diese Konzepte auch verstanden und vermittelt werden müssen für die Abgabe (s.u.), bietet es sich für den Lernprozess ggf. eher an, den Code von Grund auf selbst zu implementieren.
Abnahme:
Grundsätzlich gilt: Beide Teammitglieder müssen sowohl den Code als auch die relevanten Konzepte vollständig erklären können. **Die Aufgaben sollten bis zu dem Praktikumstermin fertig sein und nicht erst im Praktikumstermin bearbeitet werden**.
Der generelle Ablauf ist wie folgt: Idealerweise redet ihr den größten Teil der Zeit über eure Aufgabenlösungen, führt grob durch den Code und stellt die dahinterstehenden Konzepte vor und beantwortet die Teilfragen der einzelnen Aufgaben, sodass nur noch wenige Rückfragen bleiben. Zwischendurch und danach werde ich ggf. einige Fragen zur Implementierung und den dazugehörigen Konzepten der Vorlesung stellen. Idealerweise dauert dieser Prozess etwa 20 Minuten - bereitet euch also vor!

Aufgabe

Für diese Aufgabe giltt:
Die Verwendung von fertigen Implementierungen ist explizit erlaubt :). Insbesondere die Verwendung von PyTorch für die Implementierung des Neuronalen Netzes.
Es soll für diese Aufgabe der Datensatz in "**Praktikum4_Datensatz.csv**" verwendet werden, um ein neuronales Netz zu trainieren. Hierbei handelt es sich um die gleiche Aufgabenstellung wie in Aufgabe 3, nur dass die Datenverteilung eine leicht andere ist. Dementsprechend ist der erste Aufgabenteil, die Vorverarbeitung, sehr ähnlich, nur dass hier anstelle eines Train-/Testsplits ein Train-/Val-/Testsplit verwendet wird.

1. Einlesen
Lest die Zeilen des Datensatzes ein, z.B. unter Verwendung des csv-Packages von Python. Am Ende sollte eine Liste von Listen mit den Datensamples stehen.

2. Train/Val/Test Split
Teilt den Datensatz in einen Trainings- und einen Testdatensatz auf. Der Trainingsdatensatz soll aus den ersten 80% der Samples bestehen, der Valdatensatz aus den nächsten 10% und der Testdatensatz aus den letzten 10%.

3. Preprocessing
Der Datensatz besteht aus 6 Spalten, die ersten 5 ("grundstuecksgroesse", "stadt", "hausgroesse", "kriminalitaetsindex", "baujahr") sollen den Input des Modells darstellen und “klasse” ist die Klasse, welche vorhergesagt werden soll. Von den Inputvariablen sind einige kontinuierlich und einige diskret. Wendet auf die kontinuierlichen eine Standardisierung an (“zero mean, unit variance”/”Z-norm”/”Z-score”) und auf die diskreten ein One-hot-encoding (vgl. Vorlesung).
Am Ende sollte X_train, eine Liste von Listen mit den Eingabewerten und y_train, eine Liste mit den zugehörigen Ausgabewerten, stehen. Analog für X_test und y_test.

4. Erstellung eines neuronalen Netzes
Im Rahmen des Praktikums soll das neuronale Netz mit dem Framework PyTorch erstellt werden. I.d.R. funktioniert die Installation hiervon mit pip am besten. Die Installationsbefehle lauten:
Linux:
```
pip3 install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cpu
```
Mac:
```
pip3 install torch torchvision torchaudio
```
Windows:
```
pip3 install torch torchvision torchaudio
```

Weitere Optionen sind zu finden unter https://pytorch.org/get-started/locally/ , eigentlich sollte es mit obigen Befehlen jedoch funktionieren. Es ist keine GPU-Unterstützung für dieses Aufgabenblatt notwendig, alles funktioniert im vertretbaren Zeitrahmen auf handelsüblichen Rechnern!

Erstellt und trainiert ein neuronales Netz, welches die obige Klassifikationsaufgabe lösen soll! Orientiert euch dabei z.B. an

- den PyTorch Minimalbeispielen in der Vorlesung
- Beispielen im Internet, z.B. https://machinelearningmastery.com/building-multilayer-perceptron-models-in-pytorch
- dem offiziellen Quick Start Tutorial https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html
- ChatGPT :).

Kern dieser Aufgabe soll es sein, zum einen das Erstellen von Neuronalen Netzen mit PyTorch zu üben, zum anderen verschiedene Hyperparametereinstellungen zu erproben und

so die Performanz des Modells (hoffentlich) zu verbessern. Dabei sollen die Konzepte der Vorlesung praktisch angewendet werden. Der Ablauf soll wie folgt sein:

1. Beginne mit einem relativ einfachen Modell (Achtet insbesondere auf ***Outputlayer, Outputaktivierung und Fehlerfunktion***!) und trainiere es auf den Trainingsdaten. Logge und visualisiere hierbei den Verlauf des Trainings- und Validierungslosses (D.h. die Losses pro Gesamtepoche, nicht Einzelschritt). Für welche, eigentlich immer genutzte und recht simple, Regularisierungsmethode sind diese Werte notwendig? ***Implementiere und nutze diese Methode bei jedem Modell!*** (Leider ist sie nicht standardmäßig in PyTorch vorhanden)
2. Nach dem Training: Ermittle Accuracy, Precision und Recall auf den Trainings- und Validierungsdaten.
3. Versuche das Modell durch Hyperparameteranpassungen zu verbessern, d.h. die Accuracy zu erhöhen. Liegt ein Overfitting vor? Dann wende Regularisierungsmethoden an (***Was steckt jeweils hinter diesen Methoden?***). Könnte das Modell besser sein? Dann füge z.B. ein Layer oder Neuronen hinzu! Mit dem neuen Modell: Wiederhole Schritt 1 und 2.
4. Wiederhole Schritte 1 bis 3, bis ein Modell enstanden ist, bei welchem du dir sicher bist, dass es nah am Optimum ist, was für diesen Fall bedeutet, dass sich die Accuracy auf dem Validierungsdatensatz nicht mehr/kaum verbessert bei weiteren Modellanpassungen. Probiere dabei mindestens 5 verschiedene Modellvarianten aus und protokolliere dabei alle Zwischenergebnisse (Lernkurven, Accuracy,...). Ermittle dann auf diesem besten Modell final die Accuracy, Recall und Precision auf dem Testdatensatz (***Warum? Wie nennt sich das in den obigen vier Schritten beschriebene Vorgehen?***).

Wichtiger Hinweis zu Trainings- und Evaluation-/Testmodus:

PyTorch unterscheidet zwischen einem Train und Eval-Modus. Standardmäßig ist der Trainingsmodus aktiviert. Soll das Modell evaluiert werden, muss der Evaluierungsmodus aktiviert werden über model.eval(). Der Evaluierungsmodus ist insbesondere bei Dropout und Batch Normalization wichtig (***Warum?***). Zusätzlich wird torch.no_grad() verwendet, um keine unnötigen Gradientenberechnungen durchzuführen:

```
def some_evaluation_things(model, val_dataloader):
    model.eval() # Evaluationsmodus
    with torch.no_grad():
        for step, (X, y) in enumerate(val_dataloader):
            #Metriken berechnen z.B.
            return None
```

Wenn das Modell wieder trainiert werden soll, muss der Trainingsmodus vorher wieder aktiviert werden mit model.train(), einfacherweise stellt man dieses also generell vor die Trainingsschleife:

```
#Stochastic Gradient Descent
model.train()
for n in range(num_epochs):
    epoch_loss = 0
    for step, (X, y) in enumerate(train_dataloader): # für jeden Mini-batch (Stochastic Gradient Descent)
        y_pred = model(X) # Vorhersagen erzeugen (Ist-Werte)
        # etc.
```