Verteiltes Deep Reinforcement Learning System zum Trainieren von Game AI - Praktikum 16.04.2021 -

Ganz Wichtig: Arbeitet euch durch die Dokumentation!

ML-Agents Toolkit Release 15 Dokumentation

Aufgabe 1: ML-Agents Toolkit klonen und mit Unity öffnen

- ml-agents repository https://github.com/Unity-Technologies/ml-agents klonen
- Wechselt zum commit, welcher mit dem Tag “release_15” versehen ist
- Öffnet mit Unity 2019.4.* den Ordner “Project” innerhalb dem geklonten Repository

- Oder per Kommandozeile
    - git clone --branch release_15 https://github.com/Unity-Technologies/ml-agents.git

Aufgabe 2: Python Umgebung installieren

- Anaconda Prompt ausführen
- Navigiert zum Repository
- Python Umwelt erstellen
    - conda create -n ml-agents python=3.6
    - activate ml-agents
- ML-Agents Python Pakete installieren
    - cd ml-agents-envs
    - pip install -e .

    - cd ..
    - cd ml-agents
    - pip install -e .
- PyTorch installieren
    - https://pytorch.org/get-started/locally/
    - conda install pytorch torchvision torchaudio cpuonly -c pytorch

Aufgabe 3: Beispielumwelt ausprobieren und trainieren

- Öffnet eine Beispielszene z.B.
    - \Project\Assets\ML-Agents\Examples\3DBall\Scenes\3DBall.unity
- Um ein bereits trainiertes Verhalten anzugucken (Inference Mode), wechselt einfach in den Play Mode, die meisten Beispiele verfügen über ein trainiertes Modell
- Agenten trainieren
    - mlagents-learn config/ppo/3DBall.yaml --run-id=first_run
- Ergebnisstatistiken mittels Tensorboard auslesen
    - tensorboard --logdir results

Aufgabe 4: Eigene Umwelt bauen, ausprobieren und trainieren

- Making a new learning environment
- Skizziert eine Idee für ein Minispiel als Umwelt
    - Welches Ziel soll der Agent in der Umwelt erreichen
    - Ggf. vereinfacht die Umwelt
    - Ziel: Proof of Concept
    - mögliche Inspirationsquelle: Pummel Party
    - Baut keine Multi-Agent Umwelt!
- Plant was für Informationen in den Observationsraum enthalten sein könnten
    - Verzichtet zunächst auf visuelle Observationen wegen dem Rechenaufwand
- Plant welche Aktionen der Agent ausführen kann
- Plant eine Belohnungsfunktion
- Implementiert eure Idee mithilfe von Unity und ML-Agents
    - Schreibt eine neue Klasse die von Agent erbt
    - Überschreibt die relevanten Methoden, u.a.:
        - OnEpisodeBegin()
        - CollectObservations(VectorSensor sensor)
        - OnActionReceived(ActionBuffers actionBuffers)
    - Fügt die Agenten-Komponente eurem GameObject hinzu, welcher der Agent sein soll und stellt die Eigenschaften der Komponente ein

    - Bzgl. Observationen könnt ihr auch von den verfügbaren Sensor-Komponenten Gebrauch machen, z.B.
        - RayPerceptionSensor
        - CameraSensor
        - GridSensor
- Erstellt ein Heuristikverhalten, bei dem der Agent über Tastatureingabe gesteuert werden kann um die Funktionalitäten der Umwelt und des Agenten zu testen
- Legt eine neue Datei für eine Trainingskonfiguration an
    - \config\ppo\env.yaml oder \config\sac\env.yaml
    - Kopiert zunächst die Hyperparameter einer vergleichbar komplexen Umwelt aus den Beispielen. Ihr könnt u.a. folgende Eigenschaften versuchen zu vergleichen
        - Größe und Varianz in Informationen des Observationsraums
        - Größe und Typ des Aktionsraums
        - Varianz/Vielfältigkeit der Umwelt
        - Episodenlänge
- Trainiert die Umwelt mittels mlagents-learn
- Ggf. vervielfältigt eure Umwelt um so mehrere Agenten zum Sammeln der Trainingsdaten einzusetzen
- Ggf. erhöht in den Physikeinstellungen die time scale um die Simulation zu beschleunigen