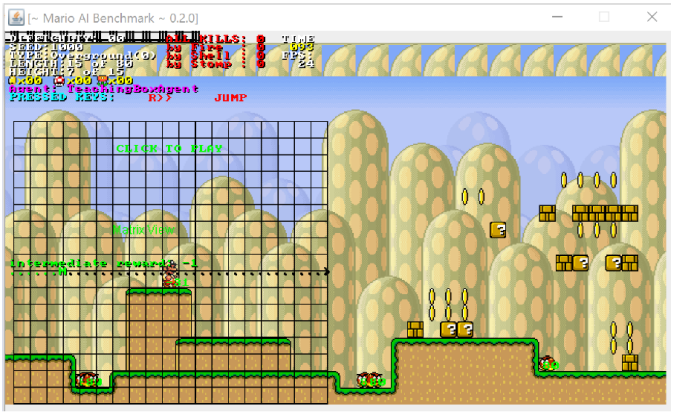
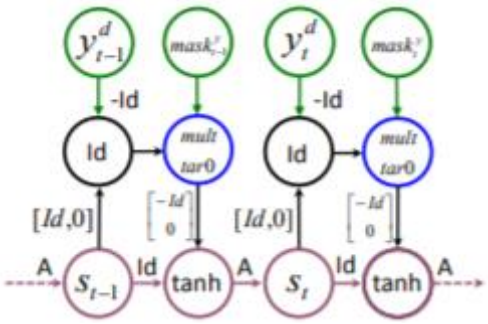
Themenvorschläge

1) **Offline Reinforcement Learning an „Super Mario AI”:**
   Quelle: https://github.com/micheltokic/marioai/tree/mario-python



   a. Generierung eines Datensatzes eines per Keyboard gesteuerten Marios in komplexen Levels (mit vielen Gegnern, etc…)
   b. Anreichern des Datensatzes durch Episoden mit zufälligen Bewegungen
   c. Training einer Policy durch Offline-RL, z.B. mit Neural-Fitted-Q-Iteration, + Literatur-Recherche was gibt es noch für aktuellere Ansätze?
   d. Evaluation der Ergebnisse
   e. iPynb für Übung in der Vorlesung

2) **TensorFlow Implementierung von „HCNN mit Missing Data“ als Markov-State-Estimator für RL (Python)**



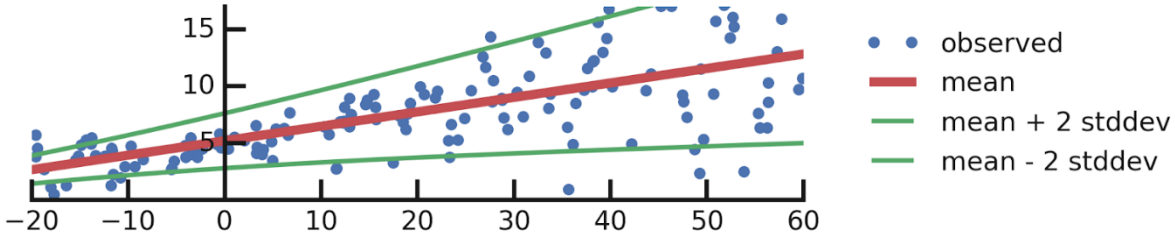
   RL-Algorithmen können von Haus aus nicht mit fehlenden Daten im Zustandsvektor umgehen, wie es z.B. in der realen Welt aufgrund von Sensorausfällen auftreten kann. Zum Schätzen des für RL benötigten Markov-Zustands soll ein HCNN (Historically-Consistent-Neural-Network) verwendet werden, welches die Systemdynamik aus einer Historie von z.B. der letzten 10 Transitionen bestimmt. Es soll die Performanz untersucht werden, im Falle von fehlenden Daten in den einzelnen Observationen. Wie genau ist die Ein-Schritt-Dynamik, wenn statt der realen Daten die interne Schätzung über diese Größen vom HCNN verwendet wird? Was ist die Auswirkung auf die Performanz einer RL-Policy?

   a. Literatur: https://www.esann.org/sites/default/files/proceedings/2020/ES2020-23.pdf
   b. Implementierung in TensorFlow als eigenständiger Keras-Layer
      i. Benötigt zwei Eingabe-Tensoren:
         1. Zustandsvektor pro Zeitscheibe (y^d)
         2. Bitmaske (mask^d) über den Zustand von y^d: (0: Datenelement fehlerhaft, 1: Datenelement ok)
   c. Untersuchung an einem geeigneten Beispiel, wo lange Horizonte von Relevanz sind. z.B. OpenAI Cart-Pole (ohne Geschwindigkeiten im Zustandsvektor?)
   d. Vorgehensweise:
      i. Lernen Sie die Systemdynamik (Ein-Schritt-Dynamik) durch ein HCNN

ii. Evaluation der Güte des HCNNs bei „Sensorausfällen“ von 1 bis n Schritte. Vergleich zur praktischen Methode „Konstant-halten des letzten Wertes“.
iii. Lernen Sie basierend auf dem HCNN (gekapselt als OpenAI-Gym Environment) eine RL-Policy, z.B. mit PPO2 aus stable-baselines. Wie unterscheidet sich die Performanz ?

e. Untersuchung der Policy-Performanz bei fehlenden Werten im Falle von:
   i. Konstant halten des letzten Wertes
   ii. Modellierung durch die vom HCNN geschätzten Größen

3) TensorFlow-Implementierung von Model-based RL mit Unsicherheit: (Python)



Es soll ein probabilistische Neuronales Netzwerk (z.B. BNN) zur Modellierung der Systemdynamik eines geeigneten Environments gelernt werden, auf dessen Basis eine RL-Policy trainiert werden kann. Aufgaben:

i. Data sampling am OpenAI Gym Environment.
   1. Die Umgebung soll in bestimmten Teilen des Zustandsraumes (z.B. bei Cartpole Winkelposition 10°-30° oder ein Teil des Tracks) gezielt verrauschte Daten liefern.
   2. Kapseln des originalen OpenAI-Gym-Environments, von welchem Daten gesampelt werden.

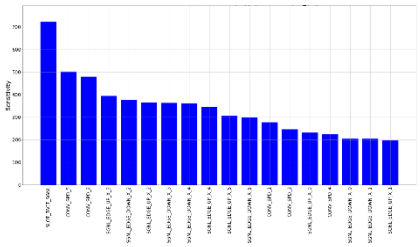
ii. Lernen eines State-Transition-Modells auf Basis der gesampelten Daten:
   1. BNN oder deterministische Gewichte mit Normalverteilung als Ausgabeneuronen → NLL-Loss Funktion (→ Tensorflow Probability)
   2. Gegeben einer Historie von Zuständen (z.B. 5-10 Lags) sollen die Erwartungs- und Streuwerte der einzelnen Zustandskomponenten zur aktuellen Zeitscheibe mittels einem Bayesianischen Neuronalen Netzwerkes geschätzt werden.
   3. Vergangenheitshorizont zur Schätzung des Markov-Zustands: ca. die letzten 4-6 Zeitschritte.
   4. Loss-Function: Negative Log-Likelyhood

iii. RL-Policy trainieren gegen das neuronale Dynamik-Modell:
   1. Kapseln des Dynamik-Modells als OpenAI-Gym-Environment
   2. Lernverfahren: z.B. PPO2 aus stable-baselines
   3. Unsicherheit beim Lernen gezielt verwenden, z.B. als Kosten in der Reward-Funktion
   4. Policy-Auswertung auf der richtigen Simulation (originales OpenAI-Gym-Environment mit Ausgabe der Unsicherheit zur Laufzeit.

b. Quellen: https://blog.tensorflow.org/2019/03/regression-with-probabilistic-layers-in.html

4) RL-Policy Training mit Neuronalen Netzwerken und Sensitivitätsanalyse

1) RL-Policy Training für ein Lernproblem mit ca. 10-20 Zustandsdimensionen
   - Sensitivitätsanalyse zum Herausfinden wichtige Features / Time-Lags (Tensorflow → GradientTape)
2) RL-Policy Training mit auf dem Ergebnis der Sensitivitätsanalyse wichtigsten n Features
3) Experimente:
   - Performanz-Vergleich vor/nach Features-Selektion
   - Evaluation auf ca. 2-3 Lernprobleme



5) RL for Stock Trading

Lernen einer RL-Policy für den Handel von Aktien / Währungspaare / etc...
Anforderungen:

a. Literaturrecherche, welche Ansätze gibt es bereits? Was sind übliche Heuristiken, gegen die man sich vergleichen kann?
b. Plattform: Metatrader 5
c. RL-Bibliothek: Stable-Baselines3 (https://stable-baselines3.readthedocs.io/en/master/)
   i. Kapselung der Datenquelle als OpenAI-Gym Environment
   ii. Bildung geeigneter Zustands-Features (z.B. Bollinger-Bänder, Rolling mean / std, und weiterer einschlägiger Kennziffern), Time-Lags, …
d. Training der Policy auf Basis historischer Daten
   i. Identifikation von Abhängigkeiten, z.B. für Lufthansa ist auch der Ölpreis als Input-Features wichtig, etc…
   ii. Wählen eines geeigneten Zeitrasters (1 Lag entspricht: 1 Minute / 10 Sekunden / 1 Stunde / 1 Tag ???)
e. Experimente:
   i. Anwendung der Policy auf „Live-Daten“
   ii. Evaluation RL-Policy vs. Standard-Heuristiken
   iii. Festhalten, was war wichtig, was eher nicht?
f. Empfohlene Unterstützung:
   https://www.coursera.org/specializations/machine-learning-trading →
   https://www.coursera.org/learn/trading-strategies-reinforcement-learning

6) Eigenes Thema ?