Fachhochschule Aachen

Fachbereich Medizintechnik und Technomathematik
Scientific Programming

Untersuchung von Recheneffizienz für Reinforcement Learning spezifische Softwaresetups

Autor:
Felix Szimtenings, 3215529

Betreuer:
Prof. Stephan Bialonski
Vladimir Samsonov

Inhalt

Kurzfassung

Die vorliegende Seminararbeit befasst sich mit der Optimierung von Reinfocement Learning Setups. Dazu werden zunächst die jeweiligen Versionen der Deep Learning Bibliotheken Pytorch, Tensorflow 1 und Tensorflow 2 miteinander verglichen und getestet. Außerdem wird ein Fokus daraufgelegt, inwiefern sich Installationsmöglichkeiten wie Pip, Cuda oder die Installation aus der Quelle auf die Performance auswirken. Weiterhin werden GPU und CPU Performances gegeneinander getestet. Hier werden für jede Bibliothek jeweils die drei Reinforcement Algorithmen: Soft Actor Critic, Deep Q-Network und Proximal Policy Optimization implementiert und gebenchmarkt. Die Tests werden alle in Docker Containern durchgeführt äußere Einflüsse zu limitieren. Außerdem wird Wert darauf gelegt Algorithmus Implementationen nur zu vergleichen, wenn sie dieselben Parameter verwenden. Bei den Versuchen werden Implementationen der jeweiligen Deep Learning Bibliotheken verwendet. Bei diesen handelt es sich um Stable Baselines 2, Stable Baselines 3 und tf2rl. Bei der Evaluation der Ergebnisse der Tests fällt auf, dass Stable Baselines 2 und tf2rl deutlich bessere Leistungen zeigen als Stable Baselines 3. Außerdem wird erkenntlich, dass eine Installation aus der Quelle die besten Ergebnisse erzeugt. Wenn diese jedoch nicht möglich ist bietet eine Conda Installation deutlich bessere Resultate als eine mit Pip.

1. Einleitung

Die jüngere Geschichte des Deep Learning ist eine vieler Erfolge. Neben dem Triumph der Maschine über den Menschen in Computerspielen und sich weltweit verbessernder Performance auf dem Gebiet der Bild- und Stimmerfassung werden auch Übersetzungen und viele weitere Aufgaben performanter. Jedoch wird dieser Fortschritt von einer wachsenden Anforderung an Rechenleistung begleitet. [20] Da aber neuere Hardware immer weniger in der Lage ist, diesen Anforderungen gerecht zu werden, ist es umso wichtiger, die performantesten Deep Learning Methoden zu verwenden. Zu diesem Zweck werden immer neue Algorithmen und Implementationen präsentiert, welche eine beschleunigte Performance versprechen. Diese Seminararbeit setzt ihren Fokus darauf, verschiedene Software Setups im Kontext von Reinforcement Learning zu vergleichen. Dazu werden verschiedene Implementationen von Algorithmen gegeneinander gebenchmarkt, sowie deren Versionen und die Auswirkung von verschiedenen Installationsmöglichkeiten auf die Performance. Um die unterschiedlichen Bibliotheken mit Implementationen, in denen Algorithmen umgesetzt werden, miteinander vergleichen zu können, werden verwendete Parameter so gut wie möglich abgeglichen und gleichgesetzt. Dabei sind verschiedene Batchsizes verwendet worden, um verschieden große Rechenanforderungen während des Trainings auszutesten. Die Ergebnisse werden anschließend in Plotly Graphen visualisiert und diskutiert. Da beim Reinforcement Learning der Reward als Metrik für positives Arbeiten des Algorithmus verwendet wird, wird bei dem Vergleich der Benchmarks untersucht, welche Setups am schnellsten hohe Rewards erreichen.

2. State-of-the-art

2.1. Algorithmen

„Reinforcement Learning“ (RL) ist einer von drei Bereichen des maschinellen Lernens, neben „Unsupervised learning“ und „Supervised learning“. RL Algorithmen studieren das Verhalten von Subjekten in Umgebungen und lernen dieses zu optimieren [2]. Es gibt verschiedene RL Algorithmen, welche unter anderem als Wert basiertes(Q-Learning) RL oder als Policy Based RL klassifiziert werden können. Diese Algorithmen werden alle als „Model-Free“ bezeichnet da sie, im Gegensatz zu den „Model-Based“ Algorithmen, keinen Gebrauch der Übergangswahrscheinlichkeitsverteilung machen, welche mit dem Markov-Entscheidungsprozess assoziiert wird [3]. State-of-the-art Q-Learning Algorithmen sind zum Beispiel DQN [4], DDPG [5] oder TD3 [6]. PPO [7], SAC [8], A2C [9] und TRPO [10] sind „Model-Free“ Algorithmen. Es existieren einige Terminologien wie: State: die Menge von Status, welche ein Agent in einer Umgebung haben kann, Action: die Menge von Aktionen, welche in einer Umgebung durchgeführt werden können, Reward: eine positive oder negative Zahl, abhängig davon wie sich die letzte Aktion des Agenten auswirkt. Return: die sich anhäufende Summe von Rewards, welche der Agent versucht zu maximieren [1]. Beim RL führt also ein Agent Actions in einer Umgebung aus. Nach jeder Action erhält der Agent einen Reward in Form von einer Zahl. Je höher diese Zahl ist, desto besser. Also versucht der Agent seine Actions so anzupassen, dass der Reward maximiert wird. [11]

2.2. Bibliotheken

Die derzeitig populärsten „Deep-Learning“ Bibliotheken, welche diese Algorithmen implementieren sind Tensorflow und Pytorch. Diese unterstützen sowohl CPU als auch GPU betriebenes Lernen. Generell sind GPUs bei Deep Learning Aufgaben Performanter als CPUs. Da der Marktanteil und die Nennungen von Tensorflow in wissenschaftlichen Papieren zugunsten von Pytorch abgenommen hat, hat Tensorflow im September 2019 eine neue Version seines Frameworks (Tensorflow 2) mit einigen Veränderungen herausgebracht [12]. Tensorflow selbst behauptet mit der neuen Version unter anderem signifikante

Leistungsverbesserungen für GPU betriebenes lernen realisiert zu haben. Diese soll bis zu dreimal schneller als die vorherige Tensorflow Version laufen [13]. Spätere Versionen von Tensorflow 1 und Pytorch Algorithmen sind in den Bibliotheken „Stable Baselines“ 2 und 3 (SB2 und SB3) implementiert, um Nutzern den Gebrauch zu vereinfachen.

3. Experimentelles Design

3.1. Methode

Das Design der in dieser Arbeit beschriebenen Experimente ist an die Taguchi-Methode angelehnt. Dabei wird versucht die Prozesse möglichst resistent gegen Störeinflüsse zu gestalten. Diese Strategie wird in 3 Schritten umgesetzt. Diese lauten: Systemdesign, Parameterdesign und Toleranzdesign. Im Systemdesign wird entschieden, aus welchen Komponenten das Experiment bestehen soll was auf einem konzeptionellen Level stattfindet. Im Parameterdesign wird das im Systemdesign erstellte Konzept umgesetzt und die Parameter werden so festgelegt, dass das Experiment Robust gegenüber Einflüssen ist. Im Toleranzdesign werden die Parameter gemäß ihrem Einfluss festgelegt. Für Parameter mit geringem Einfluss wird eine breite Toleranz festgelegt. [14]

3.2. Komponenten

Die Versuche werden alle auf einem einheitlichen System mit gleicher Hardware durchgeführt. Die Hardware ist in Abbildung 1 gezeigt. Um Hintergrundeinflüsse möglichst gering zu halten und die Experimente replizierbar zu gestalten werden alle Versuche in Docker Containern ausgeführt. Da Pytorch und Tensorflow die derzeit am populärsten sind, fokussiert sich diese Arbeit darauf verschiedene Versionen und Installationen dieser Frameworks zu benchmarken.

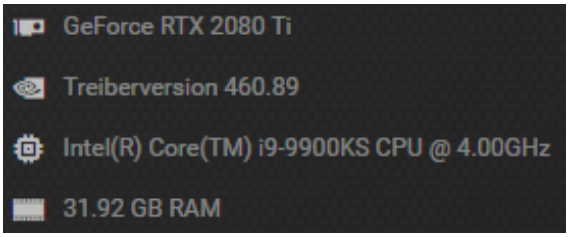

Abbildung 1

Zur Vereinfachung und um die Experimente reproduzierbarer zu gestalten werden verschiedene Implementationen der Frameworks verwendet.

Da Stable Baselines und tf2rl ausführliche Dokumentationen, einen aktiven Support und eine große Community besitzen, werden diese in diesem Experiment verwendet [15] [16] [17]. Da Stable Baselines 2 Tensorflow 1.8.0 – 1.15.0 unterstützt und Stable Baselines 3 Pytorch 1.4+ unterstützt, vereinfacht dies, beide - Pytorch und Tensorflow - mit selben Parametern zu vergleichen. In den Tests werden die Algorithmen SAC, PPO und DQN verwendet, um sowohl On- als auch Off-Policy, so wie Q-Learning Algorithmen zu testen. Um nach Leistungsunterschieden zwischen verschiedenen Versionen zu suchen werden unterschiedliche Tensorflow1 und 2 sowie Pytorch Versionen verwendet. Da Stable Baselines 2 Tensorflow 1.8.0-1.15.0 unterstützt werden Version 1.9.0, 1.12.0 und 1.15.0 getestet. Stable Baselines 3 unterstützt Pytorch ab 1.4. Da die neuste Version 1.8 noch im experimentellen Zustand ist, werden in dieser Arbeit die Versionen 1.4 und 1.7 getestet. Da Tensorflow 2 vergleichsweise neu ist, gibt es erst 5 Versionen. Tf2rl unterstützt 2.0 bis 2.3, jedoch nicht 2.4, weshalb ausschließlich 2.3 getestet wird. Darüber hinaus wird untersucht, welchen Einfluss der Installationsweg hat. Dazu werden die Frameworks mit Pip, Conda und aus der Quelle installiert. Außerdem wird Tensorflow-GPU mit Tensorflow-CPU verglichen. Beim Vergleichen der Bibliotheken und Versionen werden zunächst alle Versionen der jeweiligen Bibliotheken verglichen. Die jeweils bestabschneidenden Versionen der Bibliotheken werden anschließend miteinander verglichen. Um Zufällen entgegenzuwirken wird jeder Test 5-mal durchgeführt.

3.3. Parameterwahl und Auswertung

Um die Experimente möglichst reproduzierbar und vergleichbar zu gestalten werden bei den Versuchen für jedes Setup verschiedene festgelegte Sets von Parametern verwendet. Dazu werden alle Parameter wie „seed“, „batchsize“, „gamma“, „buffer_size“ etc. in Sets festgelegt und in den Einzelnen Experimenten verwendet. In Tabelle 1 und Tabelle 2 werden die Parameter, ihre variablen Namen in den jeweiligen Bibliotheken und deren gesetzte Werte für die Tests abgebildet.

Tabelle 1 Parameter für SAC und DQN Tests

	Learning Rate	**Gamma**	**Buffer Size**	**Batch Size**	**Tau**	**Seed**	**Time-steps**
SB2	learning_rate	gamma	buffer_size	batch_size	tau	seed	total_timesteps
SB3	learning_rate	gamma	buffer_size	batch_size	tau	seed	total_timesteps
tf2rl	lr	discount	discount	memory_capacity	tau	nicht vorhanden	max_steps
Wert	0.0003	0.99	50000	64, 128, 256	0.005	2	100000

Tabelle 2 Parameter für PPO/PPO2

	Learning Rate	**Gamma**	**Lambda**	**Total Timesteps**	**Layer**	**Epoch**	**Batch Size**
SB2	learning_rate	gamma	lam	total_timesteps	in MlpPolicy gesetzt	total_timesteps // n_batch	n_envs *n_steps
SB3	learning_rate	gamma	gae_lambda	total_timesteps	In MlpPolicy gesetzt	n_epochs	batch_size
tf2rl	lr	discount	lam	max_steps	actor_units	n_epoch	batch_size
Wert	0.0003	0.99	0.95	100000	(64, 64)	1562, 781, 390	64, 128, 256

Die Parameter in tf2rl unterscheiden sich jedoch stark von denen in Stable-Baselines 2 und 3, da zum Beispiel der Seed nicht festgelegt werden kann. Dennoch sind die Parameter so gut wie möglich abgeglichen worden. Um vergleichen zu können ob bestimmte Setups mit verschiedenen Batchsizes mehr oder weniger Performance haben werden die Sets jeweils mit einer Batchsize von 64, 128 und 256 durchgeführt. Für die verschiedenen Experimente

werden Passende Environments verwendet, welche mit dem Action Space des Algorithmus übereinstimmen. Dazu werden Probleme von OpenAi Gym verwendet, die passende Action Spaces bieten, um die Algorithmen zu testen. [18][19] Die gewählten Environments sind „MountainCarContinuous-v0“ für SAC und CartPole-v1 für DQN und PPO. Da in dieser Arbeit gemessen wird welche Versionen am Zeiteffizienztesten arbeiten, wird verglichen welche Setups mit identischen Parametern den besten Reward erreichen. Zeit spielt eine kritische Rolle bei der Leistungsmessung, wobei eine Kürzere Zeit für einen besseren Reward mit besserer Leistung assoziiert wird. [20]

4. Resultate der Experimente

4.1. Tensorflow 1 Versionen

In den folgenden Abbildungen wird auf der X-Achse die vergangene Zeit seit Testanfang und auf der Y-Achse der Reward zu diesem Zeitpunkt dargestellt. Tensorflow 1.15.0 schneidet, wie in den Abbildung 2 und Abbildung 12 zu sehen, im Vergleich zu 1.12.0 und 1.9.0 in SAC besser ab. Dies ist auch bei größeren Batchsizes zu beobachten. Zur Verdeutlichung ist in Abbildung 12 Version 1.15.0 ausgeblendet. In SAC schaffen es Version 1.9.0 und 1.12.0 nicht einen positiven Reward zu erreichen, wobei Version 1.15.0 in 4 von 5 Versuchen den maximalen Reward erreicht.

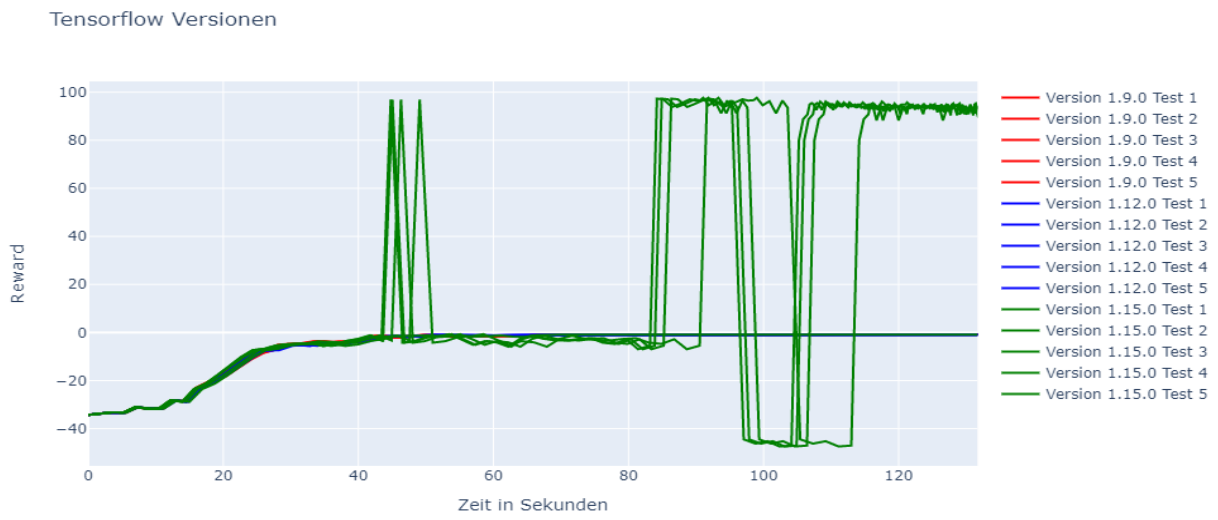

Abbildung 2 Tensorflow 1 Versionen SAC Benchmarks

In Abbildung 3 wird ersichtlich, dass Version 1.15.0 schneller höhere Rewards erreicht als die anderen Versionen. Ähnliche Ergebnisse sind bei PPO Tests erkennbar.

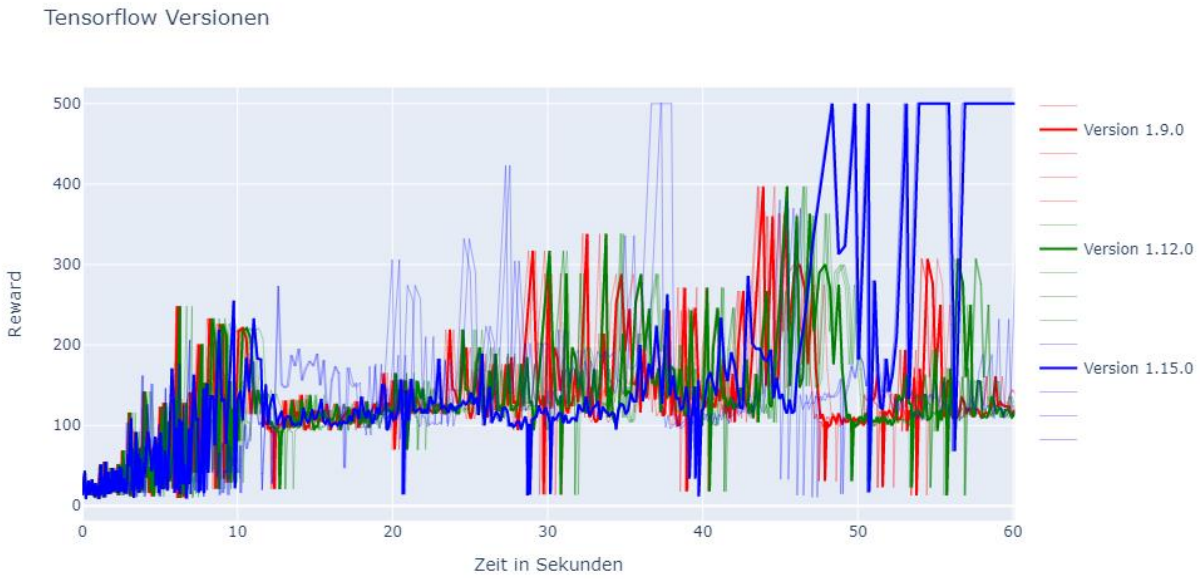

Abbildung 3 Tensorflow 1 Versionen DQN Benchmarks

Bei den Versuchen wird deutlich, dass Tensorflow 1.15.0 am schnellsten und performantesten arbeitet.

4.2 Pytorch Versionen

Pytorch Version 1.4 und 1.7 SAC Implementationen zeigen bei einer niedrigen Batchsize wie z.B. 64 eine etwas bessere Performance für Version 1.4 wie in Abbildung 4 zu sehen ist. Erhöht man jedoch die Batchsize auf 128 oder 256 zeigt Version 1.7 eine etwas bessere Performance. Allerdings schneidet wie in Abbildung 5 ersichtlich Version 1.7 auf größere Batchsizes besser ab. Beide schaffen es jedoch nicht einen positiven Reward zu erreichen.

(vgl. Abbildung 13)

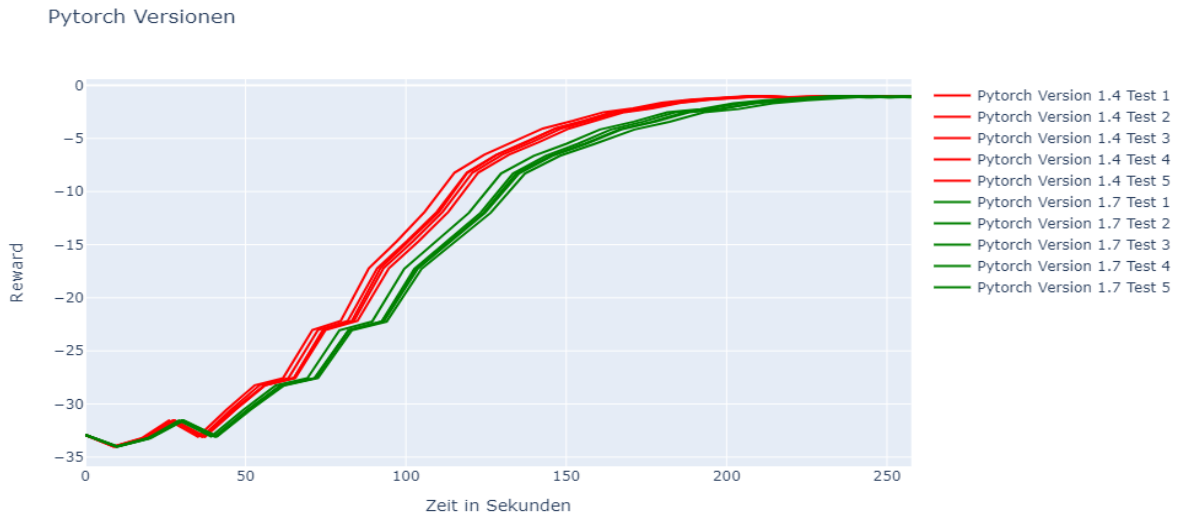

Abbildung 4 Pytorch Versionen SAC Benchmarks auf 64 Batchsize

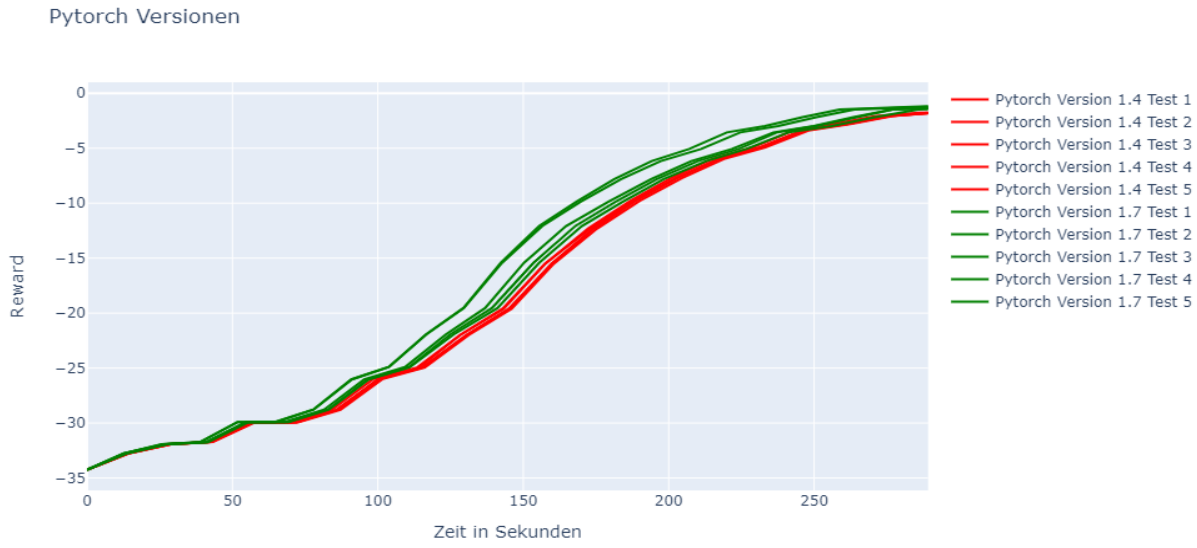

Abbildung 5 Pytorch Versionen SAC Benchmarks auf 256 Batchsize

In Abbildung 6 ist zu ersehen, dass es keine signifikanten Performance Unterschiede zwischen Pytorch Version 1.4 und 1.7 gibt, was auch für größere Batchsizes gilt (vgl. Abbildung 7). Dieselben Resultate haben sich die DQN Tests ergeben.

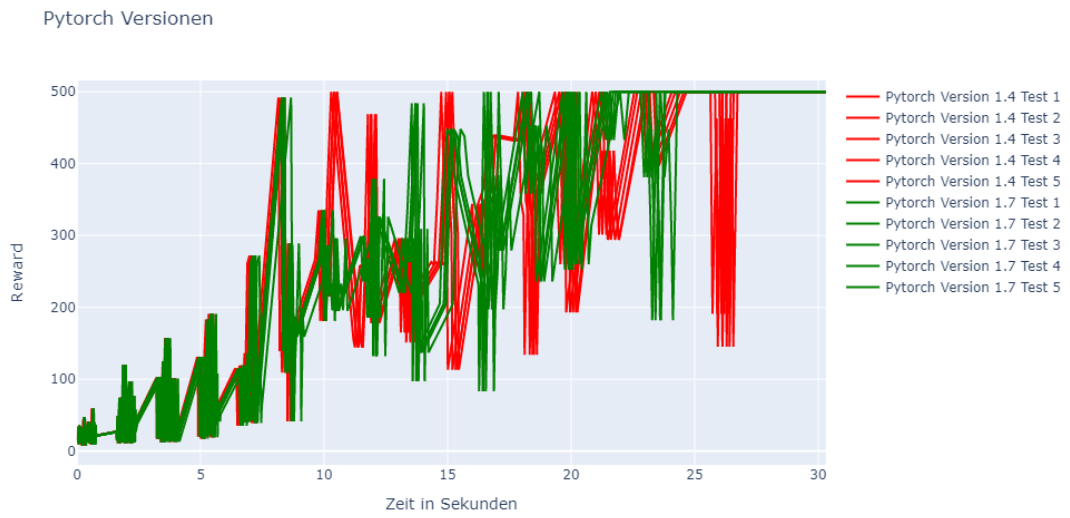

Abbildung 6 Pytorch Versionen PPO Benchmarks auf 64 Batchsize

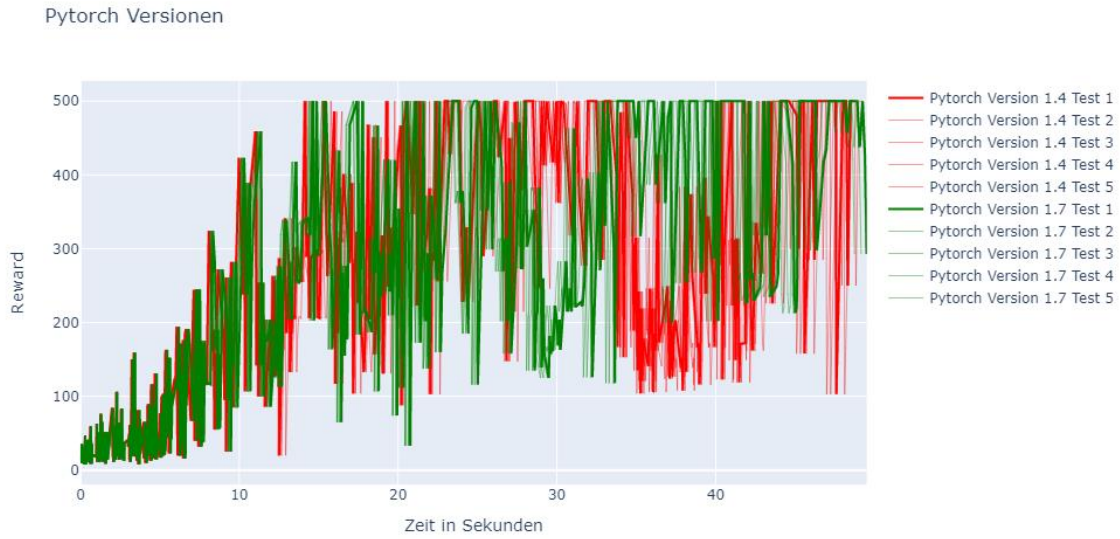

Abbildung 7 Pytorch Versionen PPO Benchmarks auf 256 Batchsize

4.3 Frameworks im Vergleich

Vergleicht man die verschiedenen Bibliotheken direkt miteinander fällt auf, dass, wie in Abbildung 8 zu sehen ist, Tensorflow 1.15.0 trotz derselben Parameter die einzige Bibliothek ist, die einen positiven Reward erreicht. Darüber hinaus ist Pytorch noch langsamer als beide Tensorflow Versionen.

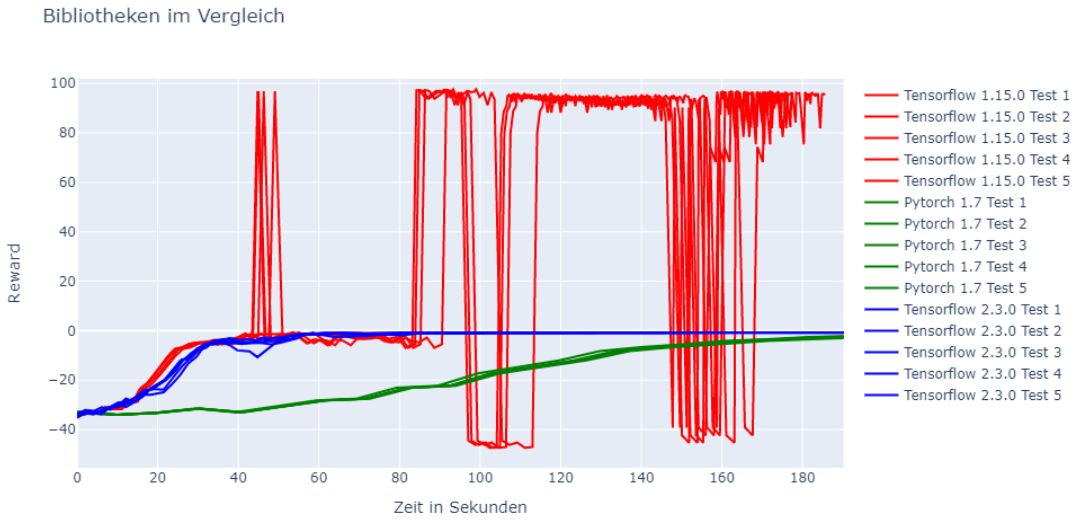

Abbildung 8 Bibliotheken SAC Benchmarks auf 64 Batchsize

Beim Vergleich der DQN Implementationen zeigt Pytorch ähnlich wie bei SAC stark von den anderen Implementationen abweichende Ergebnisse. Trotz gleicher Parameter ist die Laufzeit bei allen Batchsizes etwa halb so lang wie die der anderen Bibliotheken, jedoch wird auch nur ein halb so hoher Reward erreicht (Abbildung 14). Das in Abbildung 9 zu sehende Muster ist für alle Batchsizes dasselbe: Tensorflow 2 performt etwas besser als Tensorflow 1 wobei Pytorch nicht den halben Reward erreicht.

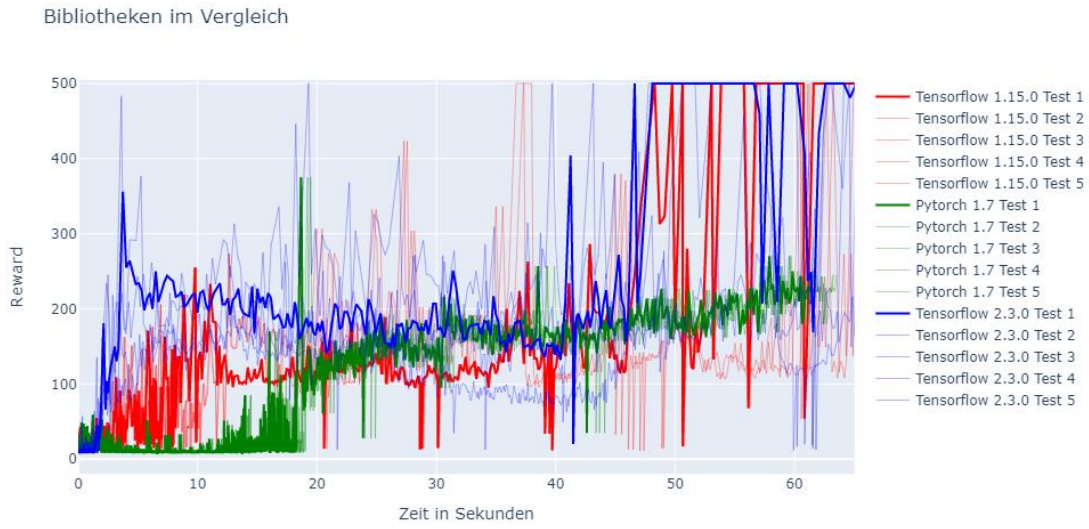

Abbildung 9 Bibliotheken DQN Benchmarks auf 64 Batchsize

4.4 Installationsmöglichkeiten

Vergleicht man verschiedene Installationsmöglichkeiten miteinander fällt wie in Abbildung 10 zu sehen ist auf, dass die Installation mit Pip schlechtere Ergebnisse erzeugt als die aus der Quelle oder mit Conda. Weiterhin ist die Installation aus der Quelle etwas schneller als die Conda Installation. Diese Ergebnisse sehen in PPO und anderen Batchsizes gleich aus.

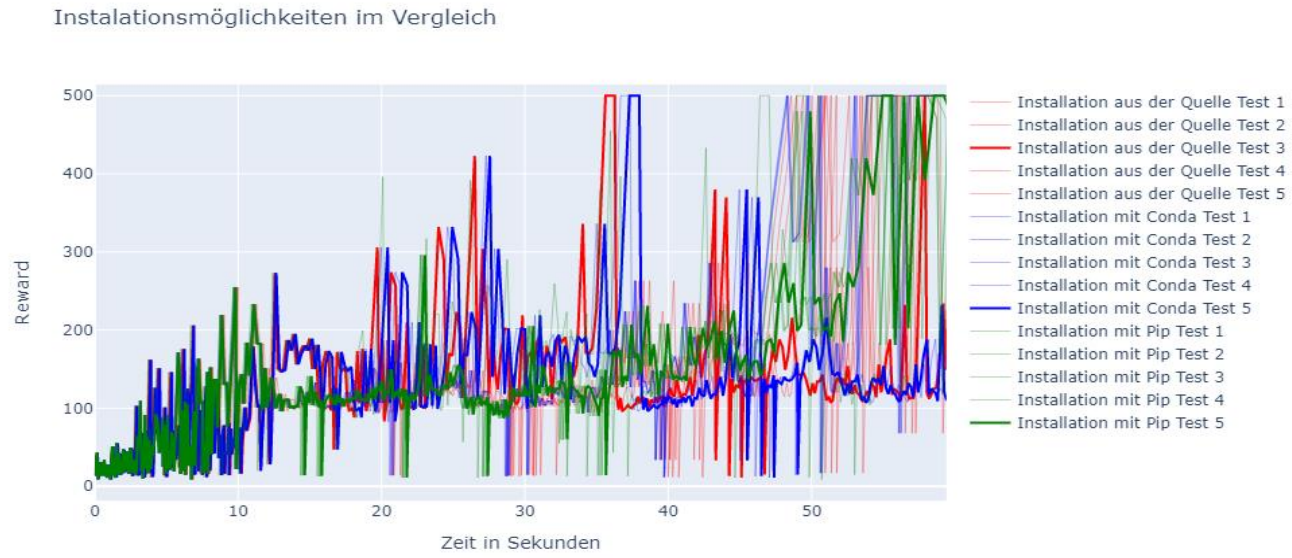

Abbildung 10 Installationsmöglichkeiten im Vergleich mit DQN

4.5 GPU und CPU Tests

Betrachtet man Abbildung 11 ist zu sehen, dass GPU durchschnittlich doppelt so lange braucht, um dasselbe Skript auszuführen. Diese Ergebnisse sind bei allen Tests die selben (vgl. Abbildung 15)

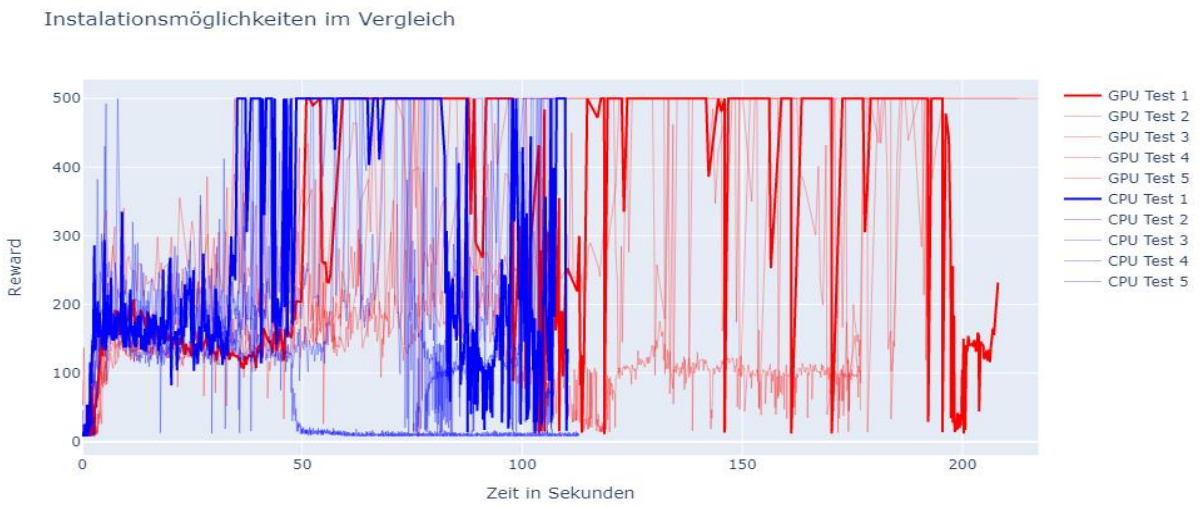

Abbildung 11 GPU gegen CPU DQN

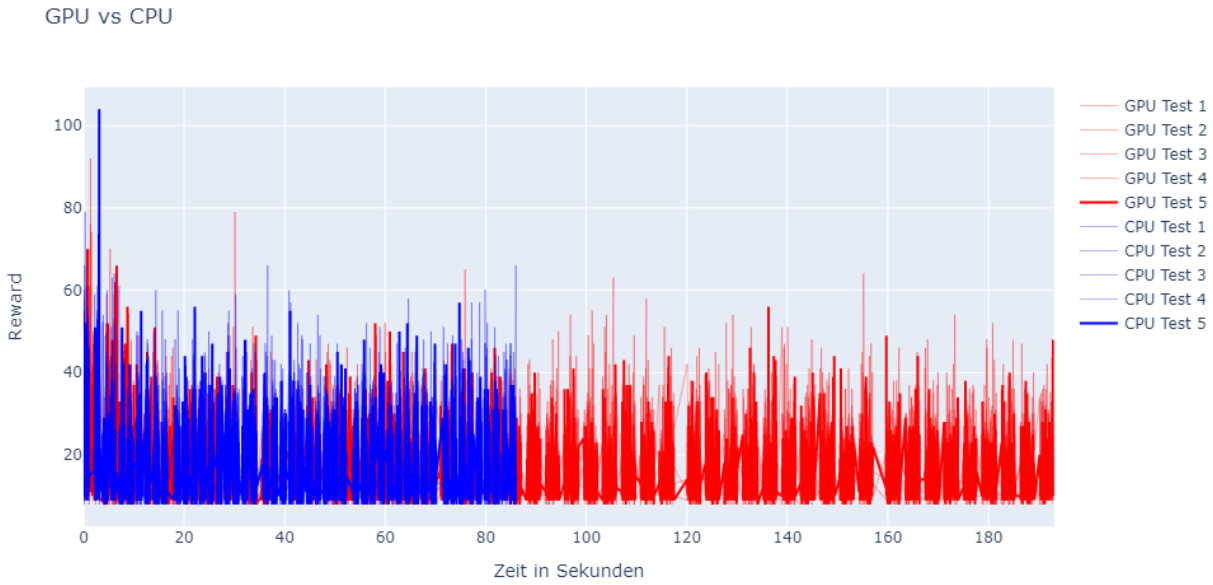

Abbildung 15: GPU gegen CPU PPO

5. Diskussion

Betrachtet man die Ergebnisse der Tests, fällt zunächst auf, dass mit wenigen Ausnahmen wie in Abbildung 4 zu sehen, die neuste Version der Bibliothek die beste Performance zeigt. Dennoch zeigt die in Abbildung 4 schlechter als ihr Vorgänger performende Pytorch Version 1.7 bei größeren Batchsizes bessere Performance als 1.4.
In Hinblick auf die Installationsmöglichkeiten wird sehr deutlich, dass Installationen mit und aus der Quelle besser performen als ein mit Pip installiertes Tensorflow. Dies kann man damit erklären, dass das Conda Paket ab Version 1.9.0 die Intel Math Kernel Library enthält welche einen starken Einfluss auf die Performance von Deep Learning hat [21]. Des Weiteren zeigt die Installation aus der Quelle die besten Ergebnisse. Dies ist darauf zurückzuführen, dass Tensorflow Pakete mit Einstellungen installiert werden, die mit so vielen GPU/CPUs wie möglich kompatibel sein sollen. Dies erleichtert es zwar dem Nutzer zwar Tensorflow zu installieren, hat jedoch seinen Einfluss auf die Leistung. Installiert man Tensorflow aus der Quelle kann man bei der Installation die Einstellungen so modifizieren, dass sie genau auf das eigene Setup ausgerichtet sind. Dazu gehört zum Beispiel die “Advanced Vector Extension” AVX2 welche die Leistung der CPU beschleunigen kann [22].
Vergleicht man die einzelnen Bibliotheken miteinander, wird deutlich, dass Pytorch im Vergleich zu den anderen Bibliotheken deutlich langsamer arbeitet.
(Abbildung 8, Abbildung 9). Tensorflow dagegen zeigt bessere Ergebnisse, wobei Tensorflow 1 in SAC deutlich bessere Ergebnisse als Tensorflow 2 zeigt. Dies kann auf die Implementation von Tensorflow zurückgeführt werden, da tf2rl eine deutlich weniger ausgearbeitete Bibliothek ist als Stable Baselines 2.
In den Versuchen wurde auch über Tensorflow 2.3 die GPU mit der CPU-Leistung verglichen und somit tf2rl verwendet. Das könnte die Resultate, in denen die CPU besser abschnitt als die GPU auch erklären. Zu erwarten wäre eigentlich, dass die GPU besser abschneidet, da die Bandweitenoptimierung der GPU zu verbesserter Thread Parallelisierung führt. Ein anderer Grund könnte eine ungünstige Initialisierung bei der Durchführung der Experimente sein. Wieso die Ergebnisse die CPU also besser abgeschnitten haben ist also unklar. Darauf einzugehen würde den Rahmen dieser Seminararbeit sprengen.

6. Fazit

Nach Durchführung und Diskussion der Experimente wird deutlich, dass Tensorflow 1 und 2 deutlich besser abschneiden als Pytorch. Da die Tensorflow 1 Implementation Stable Baselines 2 älter und besser dokumentiert ist als die Tensorflow 2 Implementation tf2rl ist sie nutzerfreundlicher zeigt jedoch außer in SAC kaum Leistungsunterschiede zu tf2rl. Bei der Installation ist es empfehlenswert diese aus der Quelle durchzuführen da dies, auch wenn arbeitsaufwändiger, zu besserer Leistung führt. Dadurch, dass man die Installationseinstellungen dabei selbst modifizieren kann, kann man sie passend auf die Flags der eigenen CPU einstellen. Ist dies jedoch keine Möglichkeit, ist das mit Conda installierte Paket um einiges recheneffizienter als das Pip Paket.

Literaturverzeichnis

[1] Thompson, N. C., Greenewald, K., Lee, K., & Manso, G. F. (2020). The Computational Limits of Deep Learning. *ArXiv*.

[2] Wiering, M., & Otterlo, M. (2012). *Reinforcement Learning: State-Of-The-Art* (Vol. 12). https://doi.org/10.1007/978-3-642-27645-3

[3] Sutton, R. S., & Barto, A. G. (2015). *Reinforcement Learning: An Introduction at http://incompleteideas.net*. http://incompleteideas.net/book/

[4] Hester, T., Schaul, T., Sendonaris, A., Vecerik, M., Piot, B., Osband, I., Pietquin, O., Horgan, D., Dulac-Arnold, G., Lanctot, M., Quan, J., Agapiou, J., Leibo, J. Z., & Gruslys, A. (2018). Deep q-learning from demonstrations. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 3223–3230.

[5] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2016). Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*.

[6] Franke, J. K. H., Köhler, G., Biedenkapp, A., & Hutter, F. (2020). *Sample-Efficient Automated Deep Reinforcement Learning*. 1–19. http://arxiv.org/abs/2009.01555

[7] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *ArXiv*, 1–12.

[8] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *35th International Conference on Machine Learning, ICML 2018*, *5*, 2976–2989.

[9] Li, X., Gao, J., Liu, J., Chen, Y., & Wong, K. (2018). ADVERSARIAL ADVANTAGE ACTOR-CRITIC MODEL FOR TASK-COMPLETION DIALOGUE POLICY LEARNING Baolin Peng The Chinese University of Hong Kong , Hong Kong. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6149–6153.

[10] Aubert, A. (2017). Multitudes : Aux origines d'une revue radicale. *Raisons Politiques*, *67*(3), 31–47. https://doi.org/10.3917/rai.067.0031

[11] Yasin, M. A., Al-Ashwal, W. A. M., Shire, A. M., Hamzah, S. A., & Ramli, K. N. (2015). Tri-band planar inverted F-antenna (PIFA) for GSM bands and bluetooth applications. *ARPN Journal of Engineering and Applied Sciences*, *10*(19), 8740–8744.

[12] He, H. (2019). The State of Machine Learning Frameworks in 2019. In *The Gradient*. https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/ (Zugriff am 31.12.2020)

[13] Hull, I., & Hull, I. (2021). TensorFlow 2. In *Machine Learning for Economics and Finance in TensorFlow 2* (pp. 1–59). https://doi.org/10.1007/978-1-4842-6373-0_1 (Zugriff am 31.12.2020)

[14] Semioshkina, N., & Voigt, G. (2006). An overview on Taguchi Method. *Journal Of Radiation Research, 47 Suppl A*(2), A95–A100. http://www.ncbi.nlm.nih.gov/pubmed/19879888

[15] *GitHub - hill-a/stable-baselines: A fork of OpenAI Baselines, implementations of reinforcement learning algorithms*. (n.d.). https://github.com/hill-a/stable-baselines (Zugriff am 31.12.2020)

[16] *DLR-RM/stable-baselines3: PyTorch version of Stable Baselines, improved implementations of reinforcement learning algorithms.* (n.d.). https://github.com/DLR-RM/stable-baselines3 (Zugriff am 31.12.2020)

[17] *GitHub - keiohta_tf2rl_ TensorFlow2 Reinforcement Learning*. (n.d.). https://github.com/kei-ohta/tf2rl (Zugriff am 31.12.2020)

[18] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). *Scaling Laws for Neural Language Models*. http://arxiv.org/abs/2001.08361

[19] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym*. 1–4. http://arxiv.org/abs/1606.01540

[20] Dai, W., Berleant, D., Dai, W., Berleant, D., Contemporary, B., Learning, D., & Frameworks, A. (2020). *and Frameworks : A Survey of Qualitative Metrics To cite this version : HAL Id : hal-02501736 Benchmarking Contemporary Deep Learning Hardware and Frameworks : A Survey of Qualitative Metrics*.

[21] *Why conda install instead of pip install_ _ by Siddhesh Gunjal _ Analytics Vidhya _ Medium*. (n.d.). https://medium.com/analytics-vidhya/why-conda-install-instead-of-pip-install-ba4c6826a0ae (Zugriff am 31.12.2020)

[22] Seiler, G. (2018). Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography. *Eprint*. https://eprint.iacr.org/2018/039.pdf

Anhang

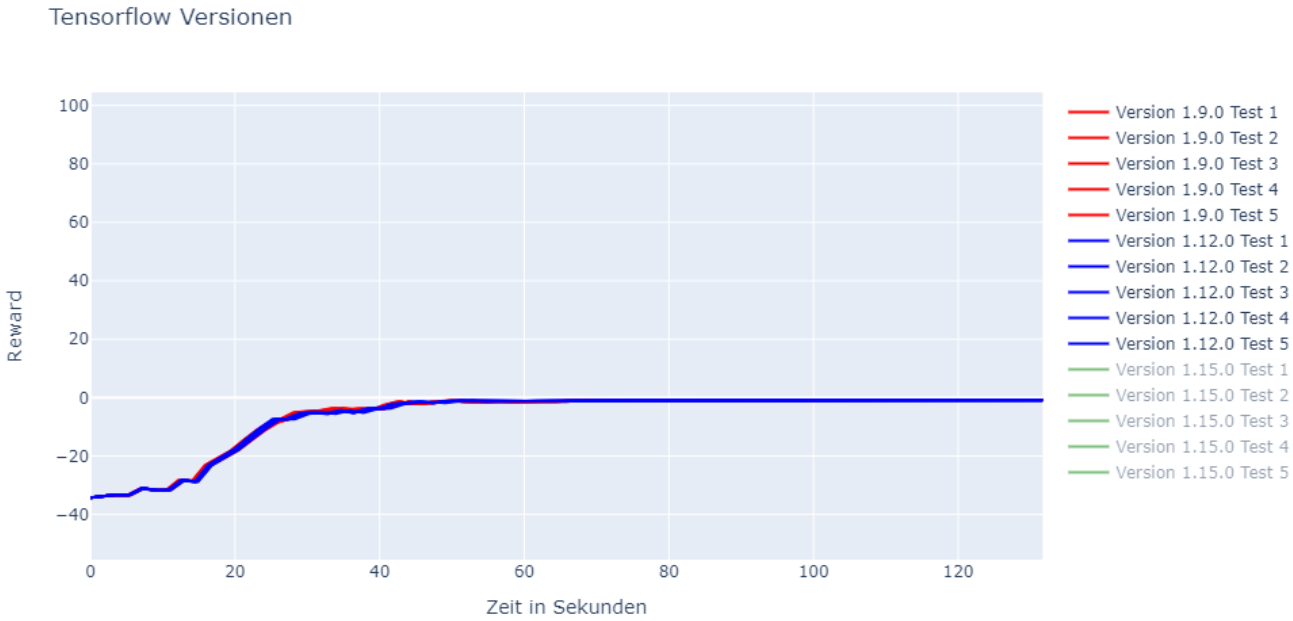

Abbildung 12 Tensorflow 1 Version Benchmarks SAC ohne Version 1.15.0

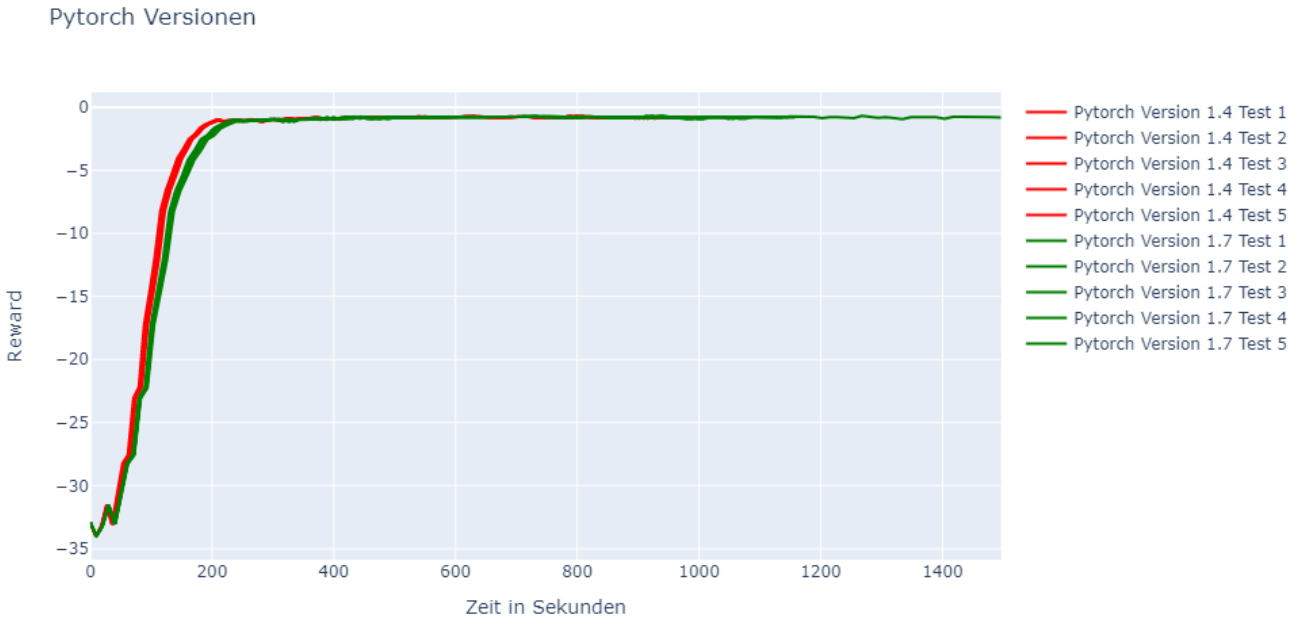

Abbildung 13 Pytorch Version Benchmarks SAC auf 1400 Sekunden

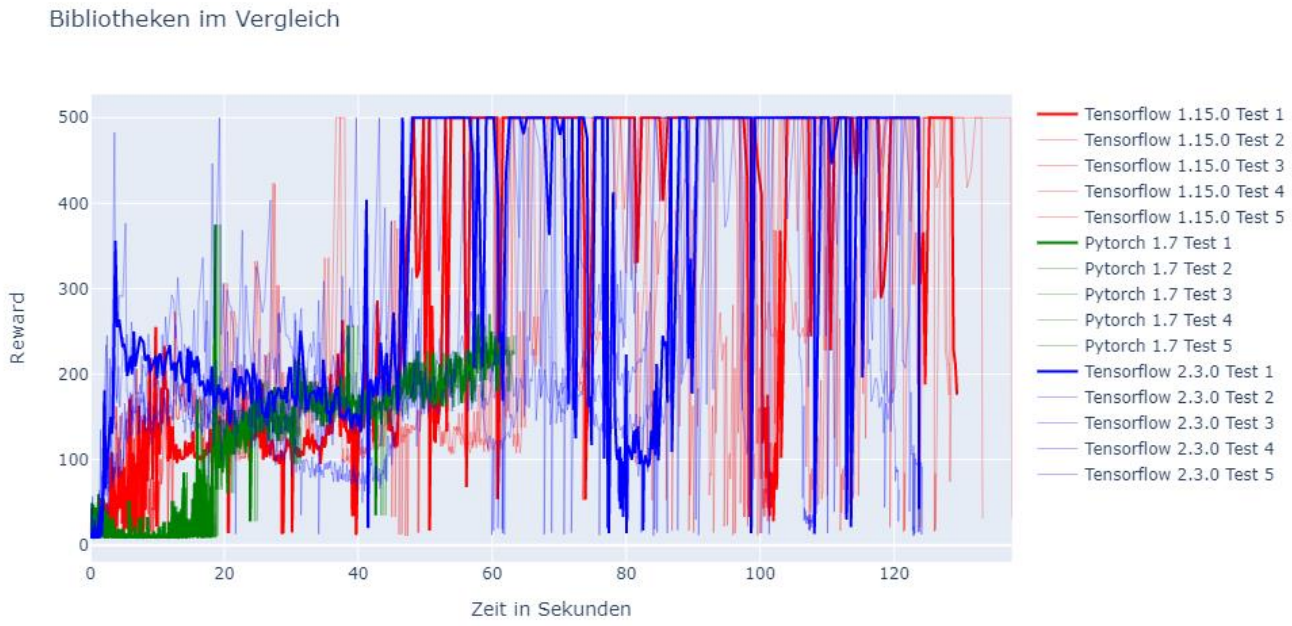

Abbildung 14 Bibliotheken DQN Benchmarks auf 64 Batchsize volle Laufzeit

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema

Untersuchung von Recheneffizienz für Reinforcement Learning

spezifische Softwaresetups

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Name: Felix Szimtenings

Aachen, den 31.12.2020

Unterschrift der Studentin / des Studenten