**libraries to use: CuDNN**

**research domain: Machine Learning/AI**

**Programming Interfaces: Anaconda**

**Equipment requested: Nvidia (Titan)**

**Hardware Grant Request: Proposal**

**Project title:** Transfer Learning for deepnets on racing games

**Team Leader:** Gordon Pipa, University of Osnabrueck

**Introduction**

Current techniques for controlling autonomous vehicles (AVs) are almost entirely focused on computer vision algorithms to better detect pedestrians or other dangers while driving. While this is certainly necessary for AVs, it is only one component of what such a system will have to do in the end. Another feature, which seems almost completely left out in this regard, is the movement efficacy of those vehicles. AVs almost never drive at their limit, and thus we think that once they are in a situation where they have to, they will have a high probability to behave suboptimally. It should be no problem to have a system that is both safe and efficient, by adding an intelligent way of handling curves and other obstacles, that also handles different conditions of streets, whether and the car itself.

In our approach, we try to make a simulated racing car learn to drive efficiently, to complete its track as fast as possible, under changing circumstances and outer conditions (like the weather changing the slip of the vehicle). Therefore, we developed a racing car simulation, in which the car learns to drive the given track via Deep-Reinforcement-Learning, mapping a vision-vector (corresponding to the ground truth of the cars position relative to the street) to the steering and pedal commands the car needs to perform.

After fulfilling our primary goal, which is to make the simulated car learn to drive on its own, there are many things we will implement afterwards. The next goal are semantic parameters in the network, giving the user external handles to control certain behaviour of the car, figuring out which parts of the network are specific to which part of the environment, the car, or other conditions. This will enable us to automatically adapt the network to different conditions. The long-term goal of this approach is for it to be applicable to actual, physical self-driving cars, to make them adapt better in unknown conditions.

**Relevance**

We hope that insights from this field will be applicable to real-world applications, especially whenever it is necessary to have an autopilot that needs to learn the physics of its outer world, to move with complex dynamics at the borderline of its capabilities. It is useful for AVs to have a general understanding of their surrounding, to make them able to cope with unknown situations in an approriate matter. For that, those vehicles need an intuitive understanding of physics in different situations. One application in that respect, which is currently far from being solved, is that of real-world racing cars. However the general approach is not only necessary for those, but also for ambulance coaches or the fire department. Further, it is a useful mechanism for all kinds of AVs, as it can be an additional component of a more general autonomous driving system.

**Approach**

Getting intuitive understandings about the surrounding physics of a system can only be learned by extensive trial-and-error learning. For this reason, we decided to program a simulated car using the Unity game engine. Unity incorporates realistic physics, as well as easy-to-manipulate parameters like the slipperiness of the street, the weight of the car or its motor torque, steering behavious, and many more. In contrast to known, "solved" racing games, our game is more complex, having realistic behaviour regarding accelleration and steering.

To make the network learn, certain data from the game (like a certain measure of the car's performance in comparison to previous rounds and a vision vector representing the street around the car) is streamed live to a python-script, in which an Artificial Neural Network (ANN) using TensorFlow calculates the correct input commands for the car. This is done with a deep recurrent network, which at first learns supervisedly from the inputs of a human teacher, and afterwards uses reinforcement learning to continually self-improve its behaviour. The network must be able to run in real-time (and optimally, for training, faster-than-real-time), to allow for good coorperation with the unity game engine, since the commands the ANN created are directly fed back to Unity, allowing for online-learning and live-feedback of the performance of the ANN.

As is known, ANNs rely on heavy computation and require millions of iterations to perform reasonably well, and our game will be no exception: As the game is supposed to be played in faster-than-realtime, it is necessary to perform the forward step as well as the learning of the ANN as fast as possible, a lot faster than a CPU manages to do. Tensorflow is a highly optimated library, and it is known for its good performance on NVIDIA GPUs. As Deep Learning is just coming up in the Machine Learning Department of our University, we don't have enough hardware to use for this project, but for it to be anyhow feasible, it is a necessity to work with decent hardware.

**Outlook**

After the first working system is completed, it is planned to tweak the resulting ANN to become more general and allow transfer-learning to different conditions a car could typically face. The idea is to generally work with handles as additional semantic parameters of the respective network, allowing the car to adapt to different situations.

This requires even more computation, as it requires many trials of a complete restructuring of the network parameters. As our current setup for the project cannot handle this, we are required to upgrade our workstation.

Sincerely, Gordon Pipa