Now This is Podracing - Driving with Neural Networks

Alexander Dewing
Stanford University
adewing@stanford.edu

Xiaonan Tong
Stanford University
xiaonan@stanford.edu

Abstract

Games are a way to easily develop and test autonomous driving systems on virtual roads with changing conditions. In this paper, we present a convolutional neural network, as well as a reinforcement learning heuristic, that takes as input the raw video feed of the game and outputs turning decisions, while valuing speed first and safety second. Tasked with operating only one dimension of control (turning), the trained neural net can complete a test course in as fast as 33 seconds, slightly slower than the human performance of 28-30s.

1. Introduction

A growing success of Artificial Neural Networks in the research field of Autonomous Driving, such as the ALVINN (Autonomous Land Vehicle in a Neural Network) and recent commercial solutions from MobileEye, Google, and others prove that the flexibility of neural networks can outperform their traditional computer vision counterparts, especially in high noise and uncertain situations [7]. In addition, the capability for neural nets to learn on top of existing classification nets offer exciting abilities for research to stack and aggregate various network schemes to improve a Neural Network's decision making accuracy [12].

In addition to being a complex decision-making paradigm, Autonomous Driving is complicated by the practicalities in gathering real-world data. It takes many hours of driving to train any initial model, and substantially longer to test and reinforce-learn the model to reach acceptable accuracy. Moreover, testing autonomous vehicles forces researchers to take risks and are often a source of logistical red tape. We turn to a common source of high fidelity simulation, gaming, and train our Neural Network to race against other vehicles in the Star Wars Episode I Podracer racing game.

Using the game as our simulation environment, we develop a convolutional neural network to control the player's vehicle in real-time by generating a single turning tendency per frame, discussed in Section 4. We train weights for this CNN through three studies: offline, online and reinforcement learning, which we will also discuss in Section 4. Section 5 and 6 will offer our conclusions on the topic, as well as insights into future research directions.

2. Related Work

Simple Neural Networks have been used to great effect in the field of Autonomous Driving. The flexibility of the technique has seen it used in multiple ways. A surprisingly simple three layer feed-forward network, for instance, can be used to reliably detect pedestrians from a depth-map obtained via stereo camera matching; Zhao's 5-hidden-neuron three-layer-net achieved around 85% accuracy and 3% misidentification rate [3]. From CMU, the ALVINN [6] (autonomous land vehicle in a neural network) uses a separate three-layer Feed-Forward Neural Network structured at (1217 input, 29 hidden, 45 output) neurons to achieve very stable navigation at 1m/s. ALVINN's approach to driving as a classification approach was very similar to our own, as the 45 output units directly influenced the steering of the vehicle, but the speed was fixed to 1m/s due to processing speed and safety concerns. A purely simulated environment allowing for failure and accidents will have no such drawbacks.

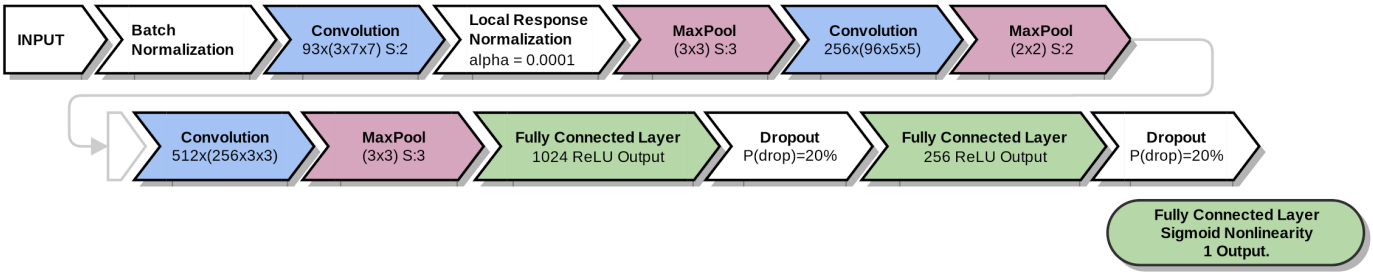

Figure 1. The 14-layer Neural Network derived from VGG_16 [1]. Note the lack of three consecutive 512-deep convolution layers. The output of the neural net denotes the net's decision to turn left (0.0), right (1.0) or stay straight ahead (0.5). The decision making mechanism alters the output by a random number of standard deviation 0.15, and issues left if $x < 0.36$, and right if $x > 0.63$, otherwise straight.

One perceived drawback to learning in synthetic situations is the incapability to apply the model to real world. However, significant amount of research has paved the way for transitioning from simulated training to real-world results. Deep visual-motor representations can be translated from synthetic to reality through methods such as Generalized Domain Alignment to minimize the difference [13]. The fluidity of neural networks permits a portion of the neural network to be transplanted through Transfer Learning [12], and used as a feature extractor to pre-train and build more complex networks. In application, this will help a realistic self-driving car start with better awareness of the world, improve its learning speed, and obtain a better outcome by the end of its training. The graphics capability of modern games is outstripping that of typical simulators, and frequently the two are joined together seamlessly. For instance, the popular trucking game EuroTruck Simulator 2 was recently used as a testing bed to examine the effects of graded auditory warnings on human truck drivers while playing the game [2]. These effects all suggest the maturity of modern graphics and rendering.

3. Infrastructure

3.1. Game and Engine Modification

Our chosen target game is available for multiple platforms, and for reasons of compatibility, we chose to run the Nintendo64 edition inside an emulator. Requiring source code availability and high compatibility, we opted for the Mupen64Plus emulator [5]. In order to permit flexible loading and unloading of user or neural-net control modules, we modified the emulator to bind to a TCP socket, and pass video frames to any connected client. In return, the clients could issue keystrokes back into the emulator corresponding to that frame. We also implemented functionality to dump compressed video and keystroke logs to disk when recording functionality is enabled. As the emulator has over a decade of code history, capturing the framebuffer was nontrivial: the graphics emulation originally targeted 3dfx Voodoo graphics cards from the late 1990s, with wrappers on top of wrappers ultimately supporting OpenGL in a scheme that was unfamiliar to us, with recent OpenGL experience.

3.2. Neural Network Infrastructure

In an effort to implement a turnkey neural network and begin training quickly, we chose to base our architecture off of a VGG network and used a pretrained network acquired from the model zoo. The network was the VGG_16 model from BMVC-2014 [11]. We shrunk the final output layers to correspond to be a single neuron. Unfortunately, our training machine, containing a Nvidia GTX 780, only contained 3GB of VRAM, and was unable to reach desired performance with memory-constrained batch sizes. As a result, we ultimately shrunk the network to limit the parameter count, allowing larger batch sizes and faster training rates. The resulting neural net is presented in Figure 1.

In contrast to the VGG_16 network in BMVC-2014 [11], a standard Batch Normalization layer is inserted before the first convolution to account for the lack of mean image that VGG_16 provides. This change fundamentally changes how the input 3-channel 224x224 image is perceived by the convolution layers, and removes the possibility of transfer learning from pre-trained VGG weights. We also dropped the fourth and fifth convolution layers (originally duplicates of the third layer) for simplicity's sake. The fully connected layers are also shrunk from 4096 neurons, to 1024 and 256 neurons respectively.

Unfortunately, the modifications precluded the use of the pre-trained model, so training began with a He-initialized network [4]. Much of the processing logic to interface race footage and user testing was most easily implemented in Python, so we chose Theano as

the best framework for implementing our neural network, and built on top of Lasagne [9] for ease of use.

During our third study, which implemented reinforcement learning, we upgraded our visualizations to show live saliency maps, reinforcement learning stack visualization, and graphs of our objective metric. Screengrabs from this UI are presented in Section 4.3.

4. Methodology

4.1. Study 1: Offline Learning

4.1.1 Methods

Our first study attempted to perform offline learning against a large dataset of user-played recordings. We played the game manually for several hours, gathering approximately 300,000 video frames of training data and the associated keyboard inputs as labels. In an effort to prevent map-specific overfit, our dataset contained footage from 10-15 maps within the game. In Figure 2. Offline Training progression, training lasted over 20 hours. Brief analysis of this shows that we are not significantly overfitting the training set, and that the neural net responds well to unseen frames., we include a chart of how the learning progressed through 55 epochs. Every 8th batch of raw data was set aside to be the validation dataset. Races consist of multiple loops, so it's not a guarantee that the NN have never seen the scenery from the validation segments of the map, but it will suffice to estimate the Neural Net's response to frames it has never seen before.

Training our simplified network for 40 epochs on our entire dataset with slightly higher learning rate than shown in Figure 2 resulted in initially encouraging performance. We were able to achieve approximately 80% training accuracy, with 70% accuracy on the same validation set. Our hypothesis was that this might be a hard accuracy wall, due to the stochastic nature of on/off turning signals that even a human will be hard-pushed to reproduce perfectly. However, when this network was used to control the vehicle in-game, the actions were unintelligible: the vehicle would proceed in a straight line until it hits a wall, then it turned in circles forever in the opposite direction.

Analysis of saliency maps [10] revealed the issue at play. Rather than generalizing to map and environment features, the network learned the orientation of the vehicle. This is visible in Figure 3, where the region of strongest saliency is the vehicle, not the track. The vehicle tilts left and right in response to hard turns, and the network effectively learned to read these features. The offline learning nature of the network has inverted causality: the vehicle should tilt in respond to control inputs, rather than vise versa.

In response, we blacked out the pixels over the vehicle (and the top UI including the time) and continued training. As the network was initialized with the result from the previous experiment, the

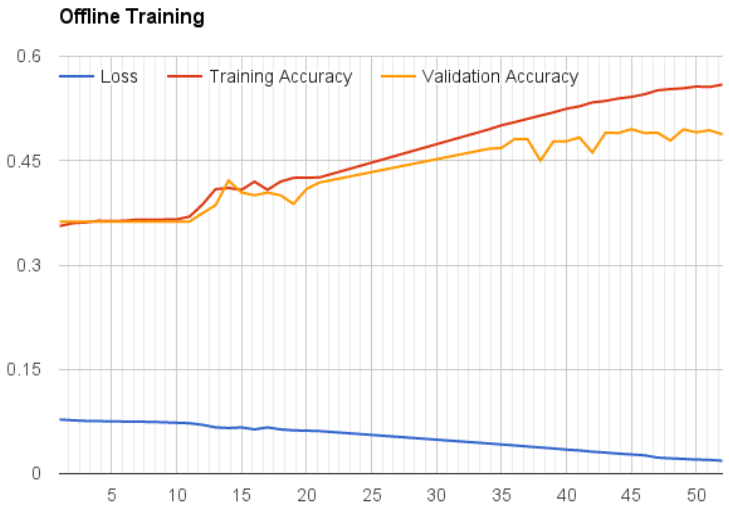

Figure 2. Offline Training progression, training lasted over 20 hours. Brief analysis of this shows that we are not significantly overfitting the training set, and that the neural net responds well to unseen frames. It is clear that we're not training at a high enough learning rate, which is rectified in a later session.

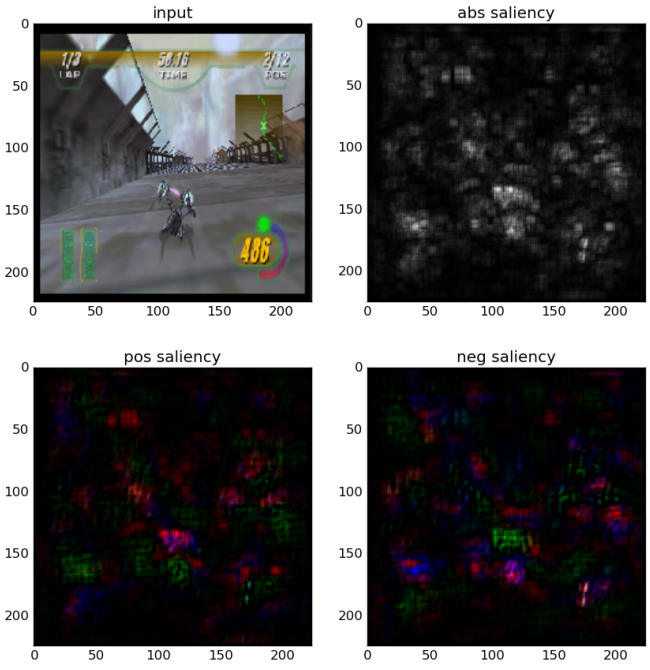

Figure 3. Saliency Map [10] for one training frame. Note the high intensity around the pod racer.

network converged fairly quickly, and within 5 epochs converged to 70% training accuracy and 67% validation. Again, the network chose to learn ineffective features, and the in-game test-drive performance was as bad as the previous method. This time, the network became a horizon detector. While the vehicle tilts substantially during turns, so does the camera to a smaller degree. In Figure 4, the visible horizon edge is strongly detected on the left.

We increasingly realized that the offline learning scheme was flawed, but attempted one last experiment by randomly rotating our blacked out training data ±15° before passing it into the training phase. Surprisingly, this experiment resulted in 90% training accuracy, which suggested very strong overfitting, even on our very large dataset. Equally surprising was the 55% validation accuracy, where 50% approximates random. Increasing regularization up to 100 times our previous value 0.0001, could not raise validation accuracy above 57%. Naturally, there was no semblance of intelligent control during testing.

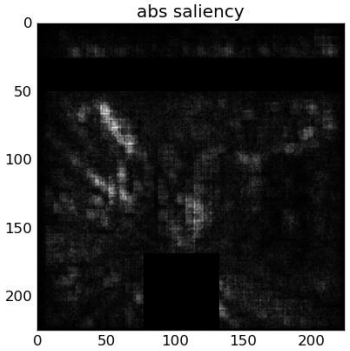

Figure 4. Blocking out pod racer creates tilt detection tendencies.

Throughout our first study, we concluded that the open-loop control model of offline learning was not going to evolve effective control algorithms for our vehicle. As the human test drivers achieved performance far better than the network initially, the footage covers a very small subset of the states that could be experienced by the neural net. The stochastic nature of control inputs paired with the large state space informed us of the need for feedback in our training procedures.

4.2. Study 2: Online Learning

Our online learning scheme was our first attempt to close the control loop of the neural net in training. In the new setup, the neural network (running a model trained in Study 1) would directly control the vehicle, but a human would define the ground-truth labels used for training. During each frame of the game, the frame would be randomly rotated and trained on using the human-supplied label in real time. We would only train when the user defined a turn (or defined straight with a third key), and ignore frames where the user found it unnecessary to override the network. Initially, training rates were exceptionally high to develop semblance of behavior quickly, but as the network improved, we reduced the rate from 0.01 to 0.0004, which combined the ability to still meaningfully affect the network with a resistance to overwhelm existing behavior in the network. We also employed a low-pass filter on the user training labels. The filter was implemented as a sinc convolution with length of 7. The presence of the sinc noticeably improved learning performance by reducing the variance in control outputs, but the frequency parameters were not very sensitive. We settled on about 10% of the control decision being decided by ± 3 frames from the initial frame.

Very quickly, we were able to see the live evolution of behavior, such as basic wall avoidance. Within about 10 minutes of live training (about 14000 frames of play, of which training occurred on about 10%), the neural network was able to finish its first unassisted lap on the Mon Gazza map, which represents our timing benchmark henceforth. The network still had a tendency to make 180° turns and start racing backwards, which would only be reset by sufficiently violent crashes where the game resets the vehicle. It is difficult for the network to realize that it is driving the wrong way because it was never trained against the eventuality. As such, the first lap completion took 2 minutes and 12 seconds, a far cry from a typical human race time of 28-30 seconds.

Another 10 minutes of training resulted in the network becoming sufficiently competent to complete a full 5-lap race unassisted. The first successful race time was 3:36 (average of 43 seconds / lap), with a best lap time of 37 seconds, and the peak performance of all our online-trained networks was 3:25 (41 seconds / lap). For context, our best network from Study 3 was able to complete the race in 3:10 (38 seconds / lap).

In an attempt to evaluate overfit, we took models trained on Mon Gazza Speedway (a relatively narrow and simple map), and tested them on a map with different color scheme, track style, and a far larger variety of terrain (Ando Prime). While the directly implanted models were either unsuccessful or very slow, only a small amount of training was needed to re-fit the model for the new map. Usually, 3-5 minutes of training was sufficient to learn map-specific behavior. As these retrained models were then useless on the original map, we chose not to further experiment with additional maps due to our obviously inadequate regularization scheme, and instead focus on further research on learning.

Closing the feedback by using online learning to take ‘encouragements’ from the user was a breakthrough that began to distill intelligent features into our neural net. While able to complete tracks and able to win first place against ‘medium’ ingame

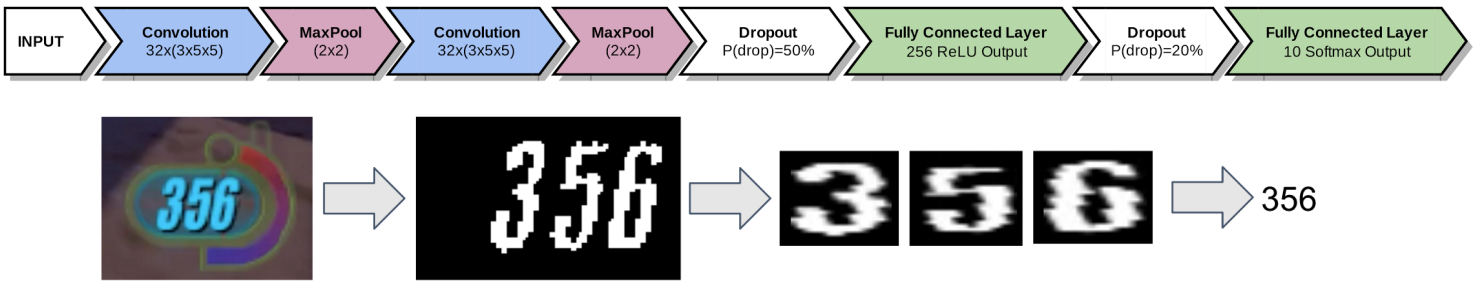

Figure 5. Digit-Parsing CNN Architecture. Each digit is processed through affine transformation and sliced apart, then classified via CNN and recombined.

computer opponents, this learning scheme was still not able to replicate human capabilities.

4.3. Study 3: Reinforcement Learning

Reinforcement learning requires an objective function, and we opted to use vehicle velocity as the input feature to a local objective heuristic. To that end, we preprocess the game UI and perform OCR on the velocity indicator. The preprocessing steps are color segmentation (select blue digits), affine transformation (straighten digits), digit separation, and resampling. The open-source Tesseract OCR engine[8] resulted in unacceptable performance on the output from all of these steps, so instead, the final OCR is achieved through a small secondary neural net presented in Figure 5. Initially, this model was initialized from a MNIST-trained network, though surprisingly, the unrefined results were worse than random (6% accuracy on 100 manually labeled characters). Hence, we needed to train the model on our specific digit dataset which apparently differed substantially from handwritten digits. We built a framework for extracting digits from the game recordings and labeling them quickly. Using a dataset of 1000 hand-labeled digits, we were able to achieve 100% accuracy using the above network. This is not surprising, as there is only mild noise in the UI coloration and therefore little noise in the input to the network.

The Reinforcement Learning method is built on top of a history Queue. As each frame is processed, it is added to a variable length queue, the length of which is determined by the current vehicle speed (between 3 and 24 frames at 0~500 speed). The history variability allows for vehicle to have a semblance of reaction speed – a faster vehicle should react earlier than a slow vehicle, so a history frame should stay longer in the queue. After training each frame, we pop off every frame that exceed this length, and train it based on the following performance heuristic.

Our performance heuristic was tuned extensively as we watched the behaviors that each iteration encouraged. Initially, we convolved acceleration with a sawtooth to form our heuristic. This resulted in heavy penalties for immediate deceleration and more modest penalties for decisions that led to deceleration several frames into the future. On the converse, positive acceleration-inducing decisions were selected positively.

We implemented reinforcement learning by setting a frame-by-frame learning rate to our goodness heuristic. Good behavior (goodness>0) would be trained positively, and bad behavior trained negatively to diffuse the decision. Our peak learning rates were substantially lower than in Study 2. We settled on the learning rate of 0.00001 as a balance between meaningful learning and the quality limitations of the heuristic.

At this point, the race times of the reinforcement-learned network were meaningfully worse than the model from Study 2 from which it inherited its initialization. Race times could not break 3:45 (9 second degradation).

We quickly realized that since most decisions are to continue straight ahead (and most of these positively reinforce due to lack of obstacles), the network was encouraged to strengthen straight behavior to the point of never turning. To counter this behavior, we reinforce straight decisions with a weight 100x lower than turns. We also capped the negative learning rate for turns during rapid deceleration (crashes), to prevent the unlearning of all turning behavior. These tunings were able to reduce the performance gap to about parity (3:38), but still improvements eluded us.

A few subtler tunings were able to substantially improve our performance. Acceleration after a collision would be very positive due to rapid acceleration, overwhelming actual good driving. Accordingly, we weight accelerating with a quadratic of speed, to encourage acceleration at high velocity, without encouraging acceleration at low velocity (recovery, not necessarily good). Finally, we discard all training frames below a low velocity threshold, as the control regime in the game appears markedly different (turning is vastly less sensitive), to prevent training to a different turning model in an atypical regime. These modifications, combined with hours of reinforcement training, were able to reduce race

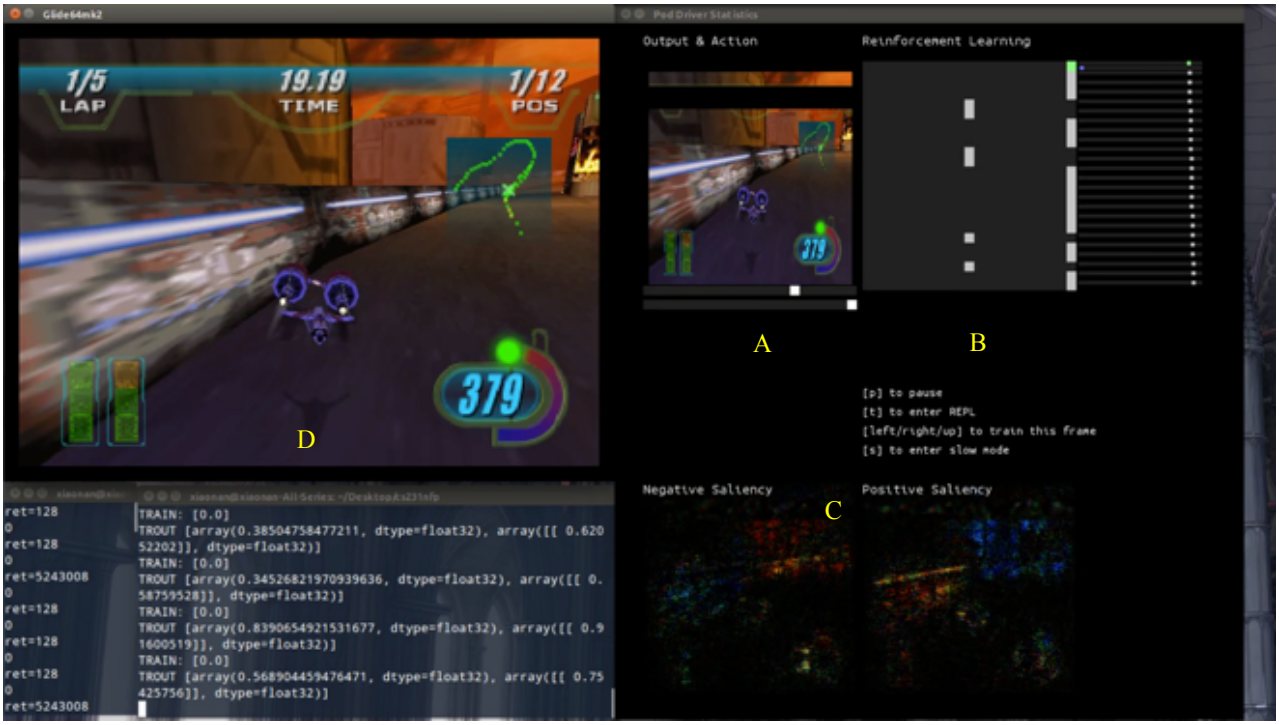

Figure 7. Screenshot of final monitoring UI implemented in PyGame [6]. In (A), we show the Neural network input, the output by the Neural network, and the decision made by the stochastic process described in Figure 1. At (B), we visualize the reinforcement learning history, with the recent frames on the bottom. Each frame bubbles up to the top, where a green shade denotes positive training, and red denotes negative training. The small white fader denotes the speed of each frame, whereas the blue fader shows how hard we are learning (or unlearning) a frame. At (C), we visualize the real time saliency of the neural network input, which gives a huge insight into how the Neural Network is parsing the scene and what data it found important. Other features are shortkeys to force an online-training on the current frame, pausing the neural net, or entering into REPL mode to do surgery on python objects (such as layer activation of the neural net, or manually request an offline-training loop). (D) is the game window. Video available at https://www.youtube.com/watch?v=58XjxOqHlws

times to our best race time of 3:10 (38 seconds / lap), with a best lap time of 33 seconds.

5. Feature Analysis

Reviewing saliency maps offers further insight into the feature detection pipeline of the neural net. Consider Figure 6, where the obvious decision is to turn right away from the wall (as the network did choose). The positive saliency map (right turn selection) has detected wall-like structures on the left strongly, as well as characteristics of the plain track on the right. It makes sense that walls to your left and open track to your right suggests a right turn. Also interesting is the negative map. Its strong indicators are the walls in a distance, towards the right of the screen, which would typically mean they're on the right of the track.

Also of note are the other obstacles the network fails to avoid. When approaching the object in Figure 7 from the side, the vehicle easily avoids it, but when, by bad luck, it approaches it head on, the network

Figure 6. Neither negative nor positive saliency took significant notice of the huge pillar in front of the pod racer, leading to this head-on crash.

fails to see a giant obstacle right in front of it: the saliency of the center region is not strong.

A final interesting characteristic of the reinforcement learning scheme is a performance oscillation. As the average velocity of the vehicle increases due to an improved model, wall scrapes become crashes with more negative learning rates. This neural net, being first-order in nature, will not realize that it was velocity and not turning decision that led to the crash. In effect, a better model lends to higher average speed, which lends to out-of-control turns that unlearns correct turning behavior. There appears to be a critical density of crashes (due to their associated strong learning parameters), which once passed, results in a cascade effect of poor learning decisions, reducing performance by up to 5 seconds / lap. Eventually, however, the system recovers and improves until the process begins anew.

6. Conclusion

We have demonstrated a very efficient learning system that utilized the advantage of offline, online and reinforcement learning, and achieved an autonomous driving neural net capable of issuing first order turning commands in response to visual cues at smooth framerates (20fps). Though the very nature of first order control and the lack of acceleration/deceleration puts a limit on how well the neural network can adapt to changing conditions, it is inevitably adapting to maximize the behavior that leads to a faster lap.

7. Further Work

For future work, a degree of acceleration control can be ceded over to the neural net as well, provided that relevant information (such as speed) is also fed in as input. Secondly, the use of RNN/LSTM layers would allow for learning second-order control parameters in an environment that has a performance cap on first-order controllers. A deeper neural net could also contribute positively by allowing for encoding of more complicated situations and map structures. Special attention could also be paid to the minimap in the top right corner, which the neural net has thus far ignored in favor of map specific features. It may be beneficial to ensemble our initial network with a network trained exclusively on the map display, which parallels a GPS in practice.

References

[1] Chatfield, K., Simonyan, K., Vedaldi, A., & Zisserman, A. Return of the devil in the details: Delving deep into convolutional nets (2014). *arXiv preprint arXiv:1405.3531*.

[2] Johan Fagerlönn, Stefan Lindberg, and Anna Sirkka. 2012. Graded auditory warnings during in-vehicle use: using sound to guide drivers without additional noise. In Proceedings of the 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI '12). ACM, New York, NY, USA, 85-91.

[3] L. Zhao and C. E. Thorpe. 2000. Stereo- and neural network-based pedestrian detection. *Trans. Intell. Transport. Sys.* 1, 3 (September 2000), 148-154.

[4] Kaiming He et al. (2015): Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*

[5] MuPen64Plus. Retrieved from https://github.com/mupen64plus 2015 (Linux)

[6] P. Shinners. 2011. PyGame - Python Game Development.

[7] Pomerleau, D. A. (1989). *Alvinn: An autonomous land vehicle in a neural network* (No. AIP-77). CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE AND PSYCHOLOGY PROJECT.

[8] R. Smith. 2007. An Overview of the Tesseract OCR Engine. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02* (ICDAR '07), Vol. 2. IEEE Computer Society, Washington, DC, USA, 629-633.

[9] S. Dielman et. al. 2015. Lasagne v2.0.1. doi: 10.5281/zenodo.27878

[10] Simonyan, K., Vedaldi, A., & Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

[11] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[12] Torrey, L., & Shavlik, J. (2009). Transfer learning. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques, 1,* 242.

[13] Tzeng, E., Devin, C., Hoffman, J., Finn, C., Peng, X., Levine, S., ... & Darrell, T. (2015). Towards Adapting Deep Visuomotor Representations from Simulated to Real Environments. *arXiv preprint arXiv:*1511.07111.