Deep Reinforcement Learning Algorithm for Self Driving Cars using Open Source Simulator

Sushant Burde, Prof. Arun Aryal

Department of Information System, California State University, Los Angeles - College of Business and Economics

Abstract

Researchers and automakers have long been fascinated by the possibility of self-driving cars roaming in the streets. What seemed like a science-fiction, a few years ago, and now looks more like an imminent future. Benefits of self-driving cars include:
Safety: A National Highway Traffic Safety Administration (NHTSA) study showed that 94% of serious crashes are due to human error. Automated vehicles have the potential to remove human error.
Economic benefits: A NHTSA study showed motor vehicle crashes in 2010 cost $242 billion in economic activity.
Mobility: A study reveals that Self-Driving Cars Could Open two Million Employment Opportunities for People with Disabilities.

Despite the benefits, two main roadblocks towards self-driving cars are
Public perceptions: Public perceptions towards injury or death by autonomous vehicles are more harmful than damage or death caused by traditional automobiles.
Lack of real-life trials: There are not enough tryouts in public roadways.

To address the lack of testing, in this exploratory research, we modify open-source simulation software CARLA that provides simple city environments and plugins to run the Python programming code for deep reinforcement learning directly. We specifically used deep Q-Learning and Convolutional Neural Network (CNN) that allowed the model to train itself per the surroundings. We also measured the accuracy of the model for 100 simulation steps.

Methodology

Deep Q-Learning Agent

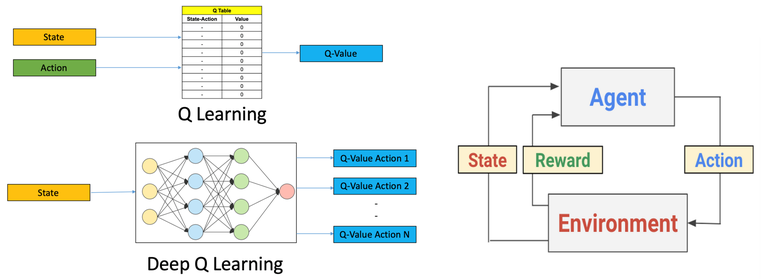

fig.1.a Q learning and Deep Q- Learning Workflow

fig.1.b A conceptual process diagram of the Reinforcement Learning problem

- Q learning is an action-state function or action-value function is defined as the expected return the AI agent would get by starting in state S, taking action A gives the q-value.
- Deep Q-Learning uses Neural Networks to learn the patterns between state and q-value, using the reward as the expected output.
- DQN agent requires some constant parameter such as replay memory size, batch size (number of samples that will be propagated through the network), simulation steps, discount, rewards and actions in order to bring the highest Q values.

Source: https://en.wikipedia.org/wiki/Q-learning

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

```
New_Q= (1 - LEARNING_RATE) * current_q + LEARNING_RATE *
(reward + DISCOUNT * max_future_q)
```

- The **DISCOUNT** is a measure of how much we want to care about FUTURE reward rather than immediate reward. Typically, this value will be fairly high, and is between 0 and 1. We want it high because the purpose of Q Learning is indeed to learn a chain of events that ends with a positive outcome.
- The **max_future_q** is grabbed after we've performed our action already, and then we update our previous values based partially on the next-step's best Q value. Over time, once we've reached the objective once, this "reward" value gets slowly back-propagated, one step at a time, per episode.

Convolution and Pool Layers

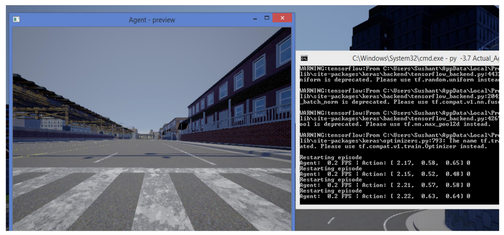

fig.2.a Actual representation of Image captured by the Agent front camera during model training

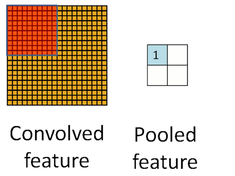

fig.2.b Convolution and Pooled feature representation

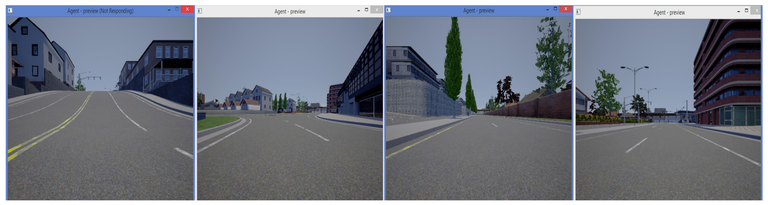

fig.2.c Sample Images captured by Agent front camera

- The layer after convolutional layer is mostly pooling layer in CNN architecture. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs a value.
- The intuition is that the exact location of a feature is less important than its rough location relative to other features.

Convolutional Neural Network (CNN)

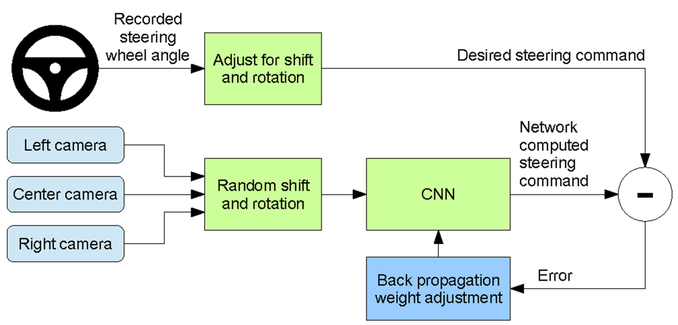

fig.3.a Training the neural network to generate steering commands from a single front-facing center camera.

- Figure shows a block diagram of our training system. Images are fed into a CNN that then computes a proposed steering command. The proposed command is compared to the desired command for that image, and the weights of the CNN are adjusted to bring the CNN output closer to the desired output.
- We trained a convolutional neural network (CNN) to map raw pixels from a single front-facing camera directly to steering commands.
- The system automatically learns internal representations of the necessary processing steps such as detecting useful road features with only the human steering angle as the training signal.

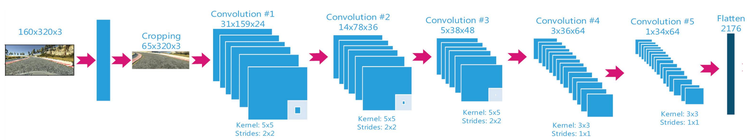

fig.3.b Deep Convolutional Neural Network image representation captured by front Camera

- The goal of a convolutional layer is filtering. As we move over an image we effective check for patterns in that section of the image.
- This works because of filters, stacks of weights represented as a vector, which are multiplied by the values outputted by the convolution.
- When training an image, these weights change, and so when it is time to evaluate an image, these weights return high values if it thinks it is seeing a pattern it has seen before.
- The combinations of high weights from various filters let the network predict the content of an image.

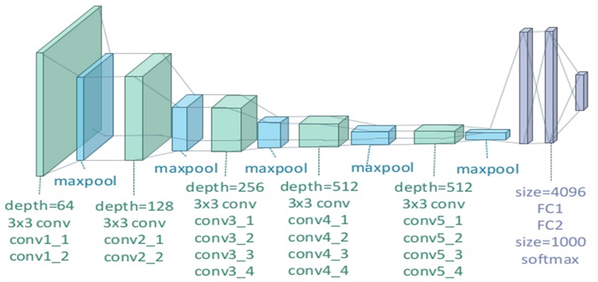

fig.3.c Deep Convolutional Neural Network Architecture

Model Training

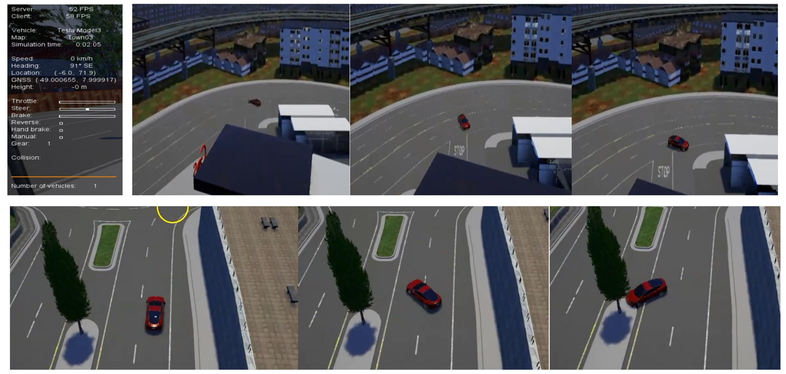

fig.4. Actual Representation of Agent adapting the Environment using DQN and CNN

- Figure illustrates the behavior of agent being trained by adapting the surrounding. The collision will be penalized which impact the reward parameter.
- The model will run in minibatches for current state. Agent keeps on changing the positions and accordingly captured the significant rewards based on surrounding.
- Speed(In Kmph) is used for each frame driving to update reward parameter.

Model Results

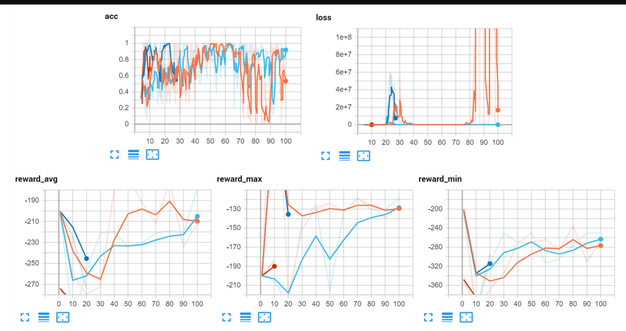

fig.5. Tensor Board graphs for Model Results

For rewards:
+0.005 (1/200) for each frame driving > 50KMH
-0.005 (-1/200) for each frame driving < 50KMH
-1 (-200/200) for a collision and episode is over

For 64x3 CNN layers:
Total params: 3,024,643
Trainable params: 3,024,643
Non-trainable params: 0
Accuracy: 90%

Conclusion

Bringing the significant result from 100 simulation steps, our deep learning model is training with great accuracy and minimum loss. To achieve more accurate results, we will train our model with 200000 simulation steps.

References

[1] NHTSA and the U.S. Department of Transportation, September 2016. "Automated Vehicles for Safety".
[2] Huang, B.-Q., G.-Y. Cao and M. Guo (2005). "Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance". 2005 International Conference on Machine Learning and Cybernetics, IEEE.
[3] Urmson, C. (2008). "Self-driving cars and the urban challenge." IEEE Intelligent Systems 23(2): 66-68.
[4] Ojarski, M., D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller and J. Zhang (2016). "End to end learning for self-driving cars." arXiv preprint arXiv:1604.07316.