DeepTraffic: Driving Fast through Dense Traffic with Deep Reinforcement Learning

Lex Fridman
Americans spend 8 billion hours stuck in traffic every year.
Goal:
Deep Learning for Everyone
accessible and fun: seconds to start, eternity* to master

http://cars.mit.edu
or search for:
“DeepTraffic”

* estimated time to discover globally optimal solution
Goal:
Deep Learning for Everyone

To Play:
- DeepTraffic
- DeepTesla
- ConvNetJS

To Win:
- OpenAI Gym
- TensorFlow
- PyTorch
Machine Learning from Human and Machine

Memorization

Human → Supervised Learning

Human → Augmented Supervised Learning

Understanding

Human → Semi-Supervised Learning

Human → Reinforcement Learning

Unsupervised Learning
DeepTraffic: Driving Fast through Dense Traffic with Deep Reinforcement Learning

http://cars.mit.edu/deeptesla
Naturalistic Driving Data

Teslas instrumented: 18

Hours of data: 6,000+ hours

Distance traveled: 140,000+ miles

Video frames: 2+ billion

Autopilot: ~12%
Naturalistic Driving Data
http://cars.mit.edu/deeptesla
• Localization and Mapping: Where am I?

• Scene Understanding: Where/who/what/why of everyone else?

• **Movement Planning:** How do I get from A to B?

• Driver State: What’s the driver up to?

• Communicate: How do I convey intent to the driver and to the world?
Autonomous Driving: A Hierarchical View

Applying Deep Reinforcement Learning to Micro-Traffic Simulation

Reference: [http://www.traffic-simulation.de](http://www.traffic-simulation.de)
Formulate Driving as Reinforcement Learning Problem

How to **formalize** and **learn** driving?
Philosophical Motivation for Reinforcement Learning

Takeaway from Supervised Learning:
Neural networks are great at memorization and not (yet) great at reasoning.

Hope for Reinforcement Learning:
Brute-force propagation of outcomes to knowledge about states and actions. This is a kind of brute-force “reasoning”.
(Deep) Reinforcement Learning

• Pros:
  • **Cheap**: Very little human annotation is needed.
  • **Robust**: Can learn to **act** under uncertainty.
  • **General**: Can (seemingly) deal with (huge) raw sensory input.
  • **Promising**: Our current best framework for achieving “intelligence”.

• Cons
  • **Constrained by Formalism**: Have to formally define the state space, the action space, the reward, and the simulated environment.
  • **Huge Data**: Have to be able to simulate (in software or hardware) or have a **lot** of real-world examples.
Agent and Environment

• At each step the agent:
  • Executes action
  • Receives observation (new state)
  • Receives reward

• The environment:
  • Receives action
  • Emits observation (new state)
  • Emits reward

References: [80]
Markov Decision Process

$S_0, a_0, r_1, S_1, a_1, r_2, \ldots, S_{n-1}, a_{n-1}, r_n, S_n$

References: [84]
Major Components of an RL Agent

An RL agent may include one or more of these components:

- **Policy**: agent’s behavior function
- **Value function**: how good is each state and/or action
- **Model**: agent’s representation of the environment

\[ S_0, a_0, r_1, S_1, a_1, r_2, \ldots, S_{n-1}, a_{n-1}, r_n, S_n \]

- state
- action
- reward
- Terminal state
Robot in a Room

actions: UP, DOWN, LEFT, RIGHT

- **UP**
  - 80% move UP
  - 10% move LEFT
  - 10% move RIGHT

- **START**

  - reward +1 at [4,3], -1 at [4,2]
  - reward -0.04 for each step

- what’s the strategy to achieve max reward?
- what if the actions were deterministic?
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- only if actions deterministic
  - not in this case (actions are stochastic)

- solution/policy
  - mapping from each state to an action
### Optimal policy

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+1</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>----</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>🅿️</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reward for each step -2
Reward for each step: -0.1

<table>
<thead>
<tr>
<th>←</th>
<th>→</th>
<th>→</th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>↑</td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td>↑</td>
<td>→</td>
<td>↑</td>
<td>←</td>
</tr>
</tbody>
</table>
Reward for each step: -0.04
Reward for each step: -0.01
Reward for each step: +0.01

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Value Function

• Future reward

\[ R = r_1 + r_2 + r_3 + \cdots + r_n \]

\[ R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_n \]

• Discounted future reward (environment is stochastic)

\[ R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n \]

\[ = r_t + \gamma (r_{t+1} + \gamma (r_{t+2} + \cdots)) \]

\[ = r_t + \gamma R_{t+1} \]

• A good strategy for an agent would be to always choose an action that maximizes the (discounted) future reward

References: [84]
Q-Learning

- State-action value function: $Q^\pi(s,a)$
  - Expected return when starting in $s$, performing $a$, and following $\pi$

- Q-Learning: Use any policy to estimate $Q$ that maximizes future reward:
  - $Q$ directly approximates $Q^*$ (Bellman optimality equation)
  - Independent of the policy being followed
  - Only requirement: keep updating each $(s,a)$ pair

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left( R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right)$$
Exploration vs Exploitation

- Key ingredient of Reinforcement Learning

- Deterministic/greedy policy won’t explore all actions
  - Don’t know anything about the environment at the beginning
  - Need to try all actions to find the optimal one

- Maintain exploration
  - Use soft policies instead: $\pi(s, a) > 0$ (for all $s, a$)

- $\varepsilon$-greedy policy
  - With probability $1 - \varepsilon$ perform the optimal/greedy action
  - With probability $\varepsilon$ perform a random action

- Will keep exploring the environment
  - Slowly move it towards greedy policy: $\varepsilon \to 0$
Q-Learning: Value Iteration

\[ Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left( R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right) \]

<table>
<thead>
<tr>
<th>S1</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>+2</td>
<td>-1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>+2</td>
<td>0</td>
<td>+1</td>
<td>-2</td>
</tr>
<tr>
<td>S3</td>
<td>-1</td>
<td>+1</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>S4</td>
<td>-2</td>
<td>0</td>
<td>+1</td>
<td>+1</td>
</tr>
</tbody>
</table>

initialize \( Q[\text{num\_states, num\_actions}] \) arbitrarily
observe initial state \( s \)

repeat

select and carry out an action \( a \)
observe reward \( r \) and new state \( s' \)

\[ Q[s, a] = Q[s, a] + \alpha (r + \gamma \max_a Q[s', a'] - Q[s, a]) \]

\( s = s' \)

until terminated

References: [84]
Q-Learning: Representation Matters

- In practice, Value Iteration is impractical
  - Very limited states/actions
  - Cannot generalize to unobserved states

- Think about the **Breakout** game
  - State: screen pixels
    - Image size: $84 \times 84$ (resized)
    - Consecutive 4 images
    - Grayscale with 256 gray levels

\[ 256^{84 \times 84 \times 4} \] rows in the Q-table!

References: [83, 84]
Philosophical Motivation for Deep Reinforcement Learning

Takeaway from Supervised Learning:
Neural networks are great at memorization and not (yet) great at reasoning.

Hope for Reinforcement Learning:
Brute-force propagation of outcomes to knowledge about states and actions. This is a kind of brute-force “reasoning”.

Hope for Deep Learning + Reinforcement Learning:
General purpose artificial intelligence through efficient generalizable learning of the optimal thing to do given a formalized set of actions and states (possibly huge).
Deep Q-Learning

Use a function (with parameters) to approximate the Q-function

- Linear
- Non-linear: **Q-Network**

$$Q(s, a; \theta) \approx Q^*(s, a)$$

References: [83]
Deep Q-Network: Atari


References: [83]
Atari Breakout

After 10 Minutes of Training

After 120 Minutes of Training

After 240 Minutes of Training

References: [85]
DQN Results in Atari

References: [83]

DeepTraffic: Driving Fast through Dense Traffic with Deep Reinforcement Learning

Lex Fridman
fridman@mit.edu

GTC 2017
May 11
Deep Q-Network: DeepTraffic

Value Function Approximating Neural Network:

- Speed: 80 mph
- Cars Passed: 2445

DeepTraffic

cars.mit.edu/deeptraffic
Deep Q-Network Training

Given a transition \(<s, a, r, s'>\), the Q-table update rule in the previous algorithm must be replaced with the following:

- Do a feedforward pass for the current state \(s\) to get **predicted Q-values for all actions**
- Do a feedforward pass for the next state \(s'\) and calculate maximum overall network outputs \(\max_a Q(s', a')\)
- Set Q-value target for action to \(r + \gamma \max_a Q(s', a')\) (use the max calculated in step 2).
  - For all other actions, set the Q-value target to the same as originally returned from step 1, making the error 0 for those outputs.
- Update the weights using backpropagation.

References: [83]
Philosophical Motivation for Deep Reinforcement Learning

**Takeaway from Supervised Learning:**
Neural networks are great at memorization and not (yet) great at reasoning.

**Hope for Reinforcement Learning:**
Brute-force propagation of outcomes to knowledge about states and actions. This is a kind of brute-force “reasoning”.

**Hope for Deep Learning + Reinforcement Learning:**
General purpose artificial intelligence through efficient generalizable learning of the optimal thing to do given a formalized set of actions and states (possibly huge in size).
Driving may need more than SLAM, Perception, and Control

References: (Karaman RRT*)

DeepTraffic

cars.mit.edu/deeptraffic

Value Function Approximating Neural Network:

- input(140)
- fc(50)
- relu(50)
- fc(5)

Speed:
80 mph

Cars Passed:
2445
Moravec’s Paradox: The “Easy” Problems are Hard

Soccer is harder than Chess

References: [8, 9]
Formulate Driving as a Reinforcement Learning Problem

DeepTraffic

Americans spend 8 billion hours stuck in traffic every year. Deep neural networks can help!

```javascript
1
2 //<![CDATA[
3 // a few things don't have var in front of them - they update already
4 existing variables the game needs
5 lanesSide = 1; //1;
6 patchesAhead = 10; //13;
7 patchesBehind = 0; //7;
8 trainIterations = 100000;
9
10 // begin from convnetjs example
11 var num_inputs = (lanesSide + 2 + 1) * (patchesAhead + patchesBehind);
12 var num_actions = 5;
13 var temporal_window = 3; //1 // amount of temporal memory. 0 = agent lives
14 in-the-moment ;
15 var network_size = num_inputs * temporal_window * num_actions
```

http://cars.mit.edu/deeptrafficjs
The Road, The Car, The Speed

<table>
<thead>
<tr>
<th>Speed:</th>
<th>80 mph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cars Passed:</td>
<td>2142</td>
</tr>
</tbody>
</table>
The Road, The Car, The Speed

- Solo motion planning subtasks
  - Longitudinal: speed
  - Lateral: lane choice

- Vehicular interaction subtasks
  - Longitudinal: car-following
  - Lateral: lane changing

Speed:
80 mph
Cars Passed:
2142
The Road, The Car, The Speed

Speed: 80 mph
Cars Passed: 2142
“Safety System”: Motion and Control are Given

Speed: 68 mph
Cars Passed: 2838
Learning the “Behavioral Layer” Task

Q-value 1 \rightarrow \text{Network} \rightarrow \text{State}

Q-value 2 \rightarrow \text{Network} \rightarrow \text{State}

Q-value n \rightarrow \text{Network} \rightarrow \text{State}
Learning the “Behavioral Layer” Task
Action Space

```
var noAction = 0;
var accelerateAction = 1;
var decelerateAction = 2;
var goLeftAction = 3;
var goRightAction = 4;
```
Driving / Learning

```
learn = function (state, lastReward) {
    brain.backward(lastReward);
    var action = brain.forward(state);
    return action;
}
```
Learning Input

```
lanesSide = 1;
patchesAhead = 10;
patchesBehind = 0;

lanesSide = 2;
patchesAhead = 10;
patchesBehind = 0;

lanesSide = 1;
patchesAhead = 10;
patchesBehind = 10;
```
Deep RL: Q-Function Learning Parameters

```plaintext
var num_inputs = (lanesSide * 2 + 1) * (patchesAhead + patchesBehind);
var num_actions = 5;
var temporal_window = 3;
var network_size = num_inputs * temporal_window + num_actions *
temporal_window + num_inputs;
```
Deep RL: Layers

```javascript
layer_defs.push(
    {
        type: 'fc',
        num_neurons: 10,
        activation: 'relu'
    }
);
```
Deep RL: Output (Actions)

```javascript
layer_defs.push({
  type: 'regression',
  num_neurons: num_actions
});
```
ConvNetJS: Options

```javascript
var opt = {};
opt.temporal_window = temporal_window;
opt.experience_size = 3000;
opt.start_learn_threshold = 500;
opt.gamma = 0.7;
opt.learning_steps_total = 10000;
opt.learning_steps_burnin = 1000;
opt.epsilon_min = 0.0;
opt.epsilon_test_time = 0.0;
opt.layer_defs = layer_defs;
opt.tdtrainer_options = {
    learning_rate: 0.001, momentum: 0.0, batch_size: 64, l2_decay: 0.01
};

brain = new deepqlearn.Brain(num_inputs, num_actions, opt);
```
Formulate Driving as a Reinforcement Learning Problem

DeepTraffic

Americans spend 8 billion hours stuck in traffic every year.
Deep neural networks can help!

```
1 //<![CDATA[
2 // a few thing's don't have var in front of them - they update already
3 existing variables the game needs
4 lanesSide = 1; ///1;
5 patchesAhead = 10; ///13;
6 patchesBehind = 0; ///7;
7 trainIterations = 100000;
8
9 // begin from convnetjs example
10 var num_inputs = (lanesSide * 2 + 1) * (patchesAhead + patchesBehind);
11 var num_actions = 5;
12 var temporal_window = 3; ///1 // amount of temporal memory. 0 = agent lives in-the-moment ;)
13 var network_size = num_inputs * temporal_window * num_actions +
```

http://cars.mit.edu/deeptrafficjs
OpenAI Gym

Slides available at http://cars.mit.edu/gtc

```python
import gym
env = gym.make("Taxi-v1")
observation = env.reset()
for _ in range(1000):
    env.render()

# your agent here (this takes random actions)
action = env.action_space.sample()

observation, reward, done, info = env.step(action)
```
OpenAI Gym: From JS to TensorFlow

1. Formulate DeepTraffic as a reinforcement learning task.
2. Use TensorFlow/Keras/PyTorch to train an agent

```
1 import gym
2 env = DeepTrafficEnv()
3 observation = env.reset()
4 for _ in range(1000):
   env.render()
7 # your agent here (this takes random actions)
8 action = env.action_space.sample()
9 observation, reward, done, info = env.step(action)
```
Formulate DeepTraffic as a Reinforcement Learning Task

```javascript
var learnServer = new WebSocket("ws://localhost:8080");

learnServer.onmessage = function (message) {
    var messageContent = JSON.parse(message.data);
    var response = {};

    if ("action" in messageContent) {
        var nextAction = messageContent.action;
        var data = step(nextAction);
        response.observation = data; // { state: learningInput, reward: speed }
    }

    if ("render" in messageContent) {
        draw();
        response.image = document.getElementById('canvas').toDataURL("image/jpeg");
    }

    learnServer.send(JSON.stringify(response));
}
```
Adding a Deep Q-Network (with Keras)

Example: [https://github.com/matthiasplappert/keras-rl](https://github.com/matthiasplappert/keras-rl)

```python
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten

model = Sequential()
model.add(Flatten(input_shape=(1,) + env.observation_space.shape))
model.add(Dense(16))
model.add(Activation('relu'))
model.add(Dense(env.action_space.n))
model.add(Activation('linear'))
```
Adding a Deep Q-Network (with Keras)

Example: [https://github.com/matthiasplappert/keras-rl](https://github.com/matthiasplappert/keras-rl)

```python
from keras.optimizers import Adam
from rl.agents.dqn import DQNAgent
from rl.policy import LinearAnnealedPolicy, BoltzmannQPolicy, EpsGreedyQPolicy
from rl.memory import SequentialMemory

# configure and compile our agent
memory = SequentialMemory(limit=10000, window_length=1)
policy = LinearAnnealedPolicy(EpsGreedyQPolicy(),
   attr='eps', value_max=1., value_min=.1,
   value_test=.05, nb_steps=10000)
dqn = DQNAgent(model=model, nb_actions=env.action_space.n,
   memory=memory, nb_steps_warmup=1000,
   target_model_update=1e-2, policy=policy)
dqn.compile(Adam(lr=1e-3), metrics=['mae'])

# run the training
dqn.fit(env, nb_steps=10000, visualize=False)

# evaluate our agent
dqn.test(env, nb_episodes=1, nb_max_episode_steps=300, visualize=True)
```
DeepTraffic
http://cars.mit.edu

v1.0: In MIT

Purnawirman (74.48 mph)
Winning: Deep Learning book (Goodfellow)
Comment: "I used a single hidden layer, window as 0). Spent some time on hyperparameters, because the test scores have a big

Michael Gump (74.04 mph)
Winning: Udacity Self-Driving Car Engin
Comment: "I mainly played around with deeper layers to see if they would get stuck in suboptimal strategies"

Jeffrey Hu (73.59 mph)
Winning: Udacity Self-Driving Car Engin
Comment: "I preprocessed to reduce the number of input features, then I tried to get the network to converge."

v1.1: Outside MIT

<table>
<thead>
<tr>
<th>User</th>
<th>MPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hoang Nguyen</td>
<td>76.29</td>
</tr>
<tr>
<td>Jorces96</td>
<td>76.16</td>
</tr>
<tr>
<td>leiming yu</td>
<td>75.97</td>
</tr>
<tr>
<td>Jordan Ott</td>
<td>75.91</td>
</tr>
<tr>
<td>Abheet Vaidya</td>
<td>75.88</td>
</tr>
<tr>
<td>Nándor Kedves</td>
<td>75.83</td>
</tr>
<tr>
<td>3upperm2n</td>
<td>75.80</td>
</tr>
<tr>
<td>Tamás Csaba Kádár</td>
<td>75.77</td>
</tr>
<tr>
<td>Mark S.</td>
<td>75.73</td>
</tr>
</tbody>
</table>
DeepTraffic 2.0

1st place: Titan XP
2nd place: GeForce GTX 1080 Ti
3rd place: Jetson TX2
Challenge to GTC Attendees:

- Create account on the site and put “GTC” as how you heard about us.
- Make a neural network that travels 70+ mph.
Have fun with Deep RL and DeepTraffic!
Have fun with Deep RL and DeepTraffic!

But not too much fun...

Slides available at http://cars.mit.edu/gtc