TUMCREATE

Deep Q-Learning for Reducing Traffic Delay

Based on

Human-level control through deep reinforcement learning

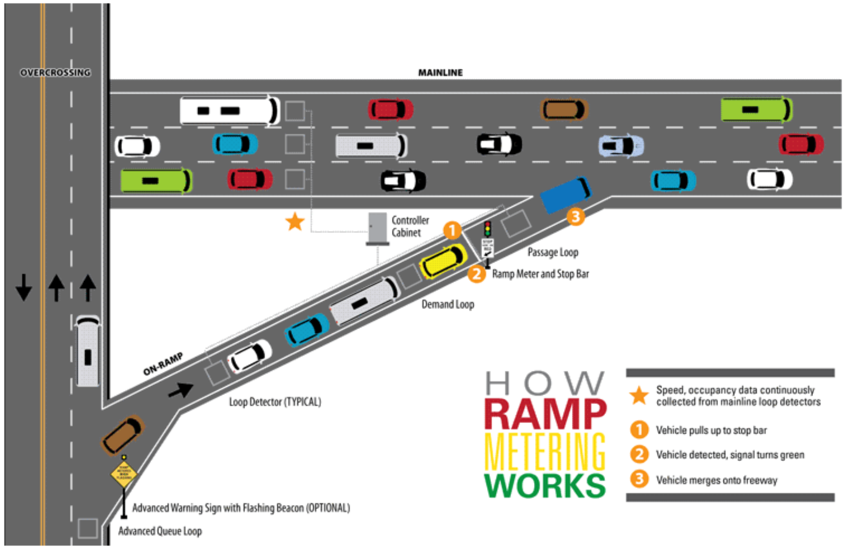
Volodymyr Mnih1*, Koray Kavukcuoglu1*, David Silver1*, Andrei A. Rusu1, Joel Veness1, Marc G. Bellemare1, Alex Graves1, Martin Riedmiller1, Andreas K. Fidjeland1, Georg Ostrovski1, Stig Petersen1, Charles Beattie1, Amir Sadik1, Ioannis Antonoglou1, Helen King1, Dharshan Kumaran1, Daan Wierstra1, Shane Legg1 & Demis Hassabis1

ABHINAV SUNDERRAJAN

Ramp metering

Control the flow of vehicles along an on-ramp of an expressway using traffic lights to:-

1) Minimize delay along the main line expressway and the ramps.
2) Avoid shock waves due merging of vehicles.
3) Minimize accidents and collisions to increase safety.



Source: Parsons Brinckerhoff.

Environment

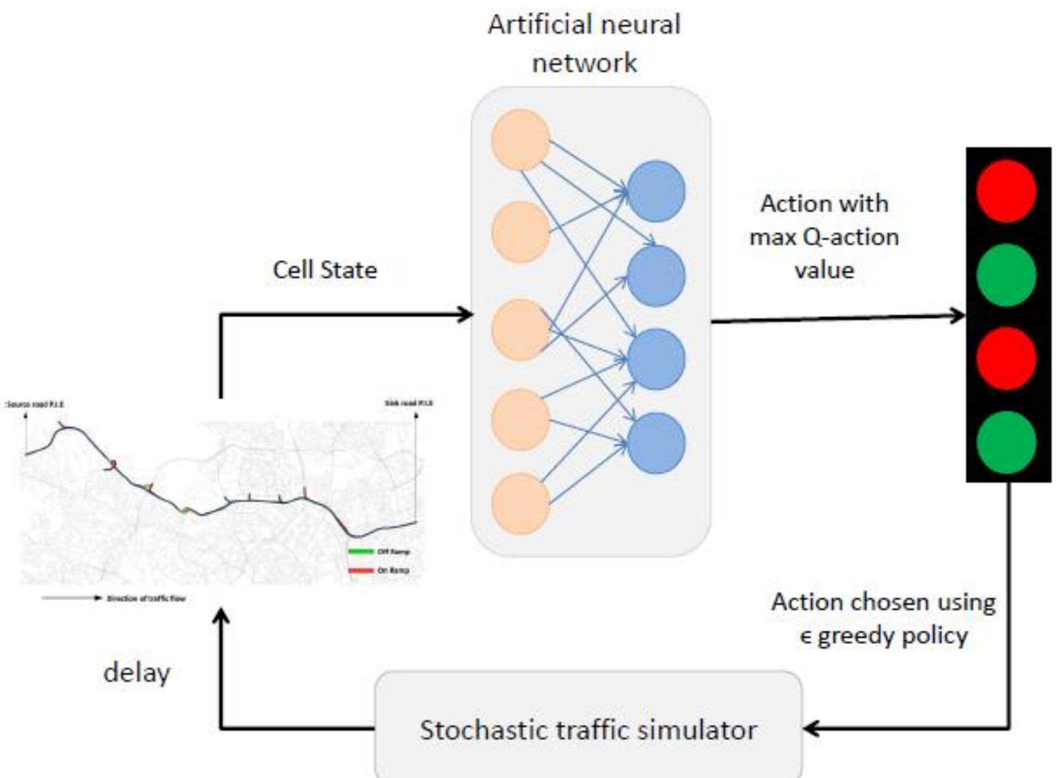
TUMCREATE

13 km stretch of P.I.E (Pan Island Expressway) in central Singapore with all on ramps and off ramps.

GOAL

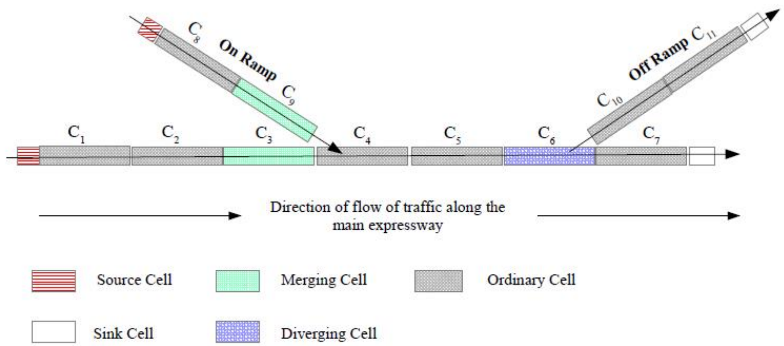
TUMCREATE

Find a ramp metering policy which minimizes the total delay experienced by all vehicles over a simulated time horizon of K time-steps.



Traffic Emulator Model

Stochastic Cell transmission Model macroscopic traffic simulation of P.I.E section with all on off ramps



Total number of cells in the simulated stretch of the expressway is 212

Algorithm 1 CTM based macroscopic predictive simulation.

while $t < K$ **do**
  for each $c_i \in \mathbb{C}$ **do**
    update sending and receiving potentials.
  for each $c_i \in \mathbb{C}$ **do**
    update outflow.
  for each $c_i \in \mathbb{C}$ **do**
    update number of vehicles.
    update density.
  for each $c_i \in \mathbb{C}$ **do**
    update average speed v_i
    update the maximum number of vehicles N_i^{max}
  $t = t + k.T_{ctm}$

https://www.researchgate.net/publication/303940227_SYMBIOTIC_TRAFFIC_SIMULATION_FRAMEWORK

Traffic Emulator State and Action

- State of the emulator at time t is defined in terms of state of each cell $c_i \in C$ is defined as $\frac{n_t}{nMax_t}$ which represents the ratio of the number of vehicles in c_i to the maximum number of vehicles that can be accommodated in c_i.
- The number of actions that can be taken at each state is $2^4 = 16$
  - Number of controllable on ramps =4
  - Each traffic light at these on ramps can be either R or G.
  - Hence total number of actions is 2^4
  - Generally speaking if number of ramps =ρ then number of actions is 2^ρ

Traffic Emulator Reward

TUMCREATE

- The traffic lights at all on ramps change phase (or continue to be in the same phase) every 12 seconds. Note that the simulation time step is 4 seconds.
- Delay for all cells $c_i \in C$ at time step k is defined as

$$r(k+1) = \begin{cases} 100.0, & \text{if } k = K \text{ and } D_{total}^{noRM} > D_{total} \\ -\omega_1 \sum_{c_i \in \mathbb{C}} d_i(k), & \text{otherwise} \end{cases} \tag{4}$$

Here D^{noRM} is the total delay over K time-steps when there is no ramp metering while D represents the delay with ramp metering as suggested by the neural network.

- Thus the goal of the reinforcement learning game is to take an action which minimizes the delay over K time steps and perform better than the baseline of no ramp metering.
- Hence each episode consist of $\frac{K}{12} < s, a, r', s', a' >$ state action pairs.

Training

Algorithm 1 Deep Q learning for adaptive ramp metering control.

Initialize replay memory $\mathbb{M}$ with size N.
Initialize the neural network (action-value function) Q with weights Θ.
Initialize the target action-value function (another neural network) $\bar{Q}$ with weights $\bar{\Theta} \leftarrow \Theta$.
repeat
  Set random flow rates at all source cells.
  Warm up traffic emulator.
  for $k = 0, K$ **do**
    Get the state s_t from the cell network in emulator.
    if $random_{double} < \epsilon$ **then**
      Select random action a_t
    else
      Select action $a_t = argmax_a Q(s_t, a; \theta)$
    Observe the reward $r_{t+1} = d(k)$ for the action a_t and the new cell state s_{t+1}
    Create transition tuple $\mathbf{T} = (s_t, a_t, r_{t+1}, s_{t+1})$
    if $size(\mathbb{M}) < N$ **then**
      Add $\mathbf{T}$ to memory $\mathbb{M}$
    else
      replace first element in $\mathbb{M}$ with the tuple $\mathbf{T}$
      Sample random mini-batch $\mathbb{B}$ from $\mathbb{M}$
      For all $\mathbf{T} \in \mathbb{B}$
      $$y_j = \begin{cases} r_j + \gamma max_a(\bar{Q}(s_{j+1}, a'; \bar{\Theta})) & \text{if } (j+1) < k \\ r_j, & \text{otherwise} \end{cases}$$
      Use $(y_j - Q(s_j, a_j; \Theta))^2$ for gradient descent with respect to Θ.
      After C steps, reset $\bar{Q} \leftarrow Q$
until $epoch < epoch_{max}$

Hyper-parameters

Hyper Parameter	Value	Description
number of hidden layers	2	The number of hidden layers in the neural network used for Deep Q-learning. The first hidden layer consists of 200 neurons while the second hidden layer is composed of 150 neurons.
learning rate	0.00025	Learning rate used by the RMSProp algorithm.
discount factor	0.98	Discount factor used in Q-learning update.
Replay memory size	150000	Stochastic gradient descent updates are sampled from this number of recent $(s_t, a_t, r_{t+}, s_{t+1})$ tuples stored in memory.
mini batch size	32	The batch size used by the stochastic gradient descent (SGD) update of neural network weights.
rms decay	0.95	Gradient momentum used by the RMSProp algorithm.
initial exploration	1.0	The initial ϵ in the ϵ greedy evaluation.
final exploration	0.1	The finalϵ in the ϵ greedy evaluation.
target network update frequency	50	The frequency with which target network (in number of steps) is evaluated.

Number of epochs trained=1000

Neural Network

TUMCREATE

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
.seed(random.nextLong())
.optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
.iterations(1)
.activation("leakyrelu")
.weightInit(WeightInit.RELU)
.learningRate(learningRate)
.updater(Updater.RMSPROP)
.rmsDecay(0.9)
.regularization(true)
.l2(2.0e-4)
.list()
.layer(0,
new DenseLayer.Builder().nIn(numOfCells).nOut(204).activation("leakyrelu")
.weightInit(WeightInit.RELU).build())
.layer(1,
new DenseLayer.Builder().nIn(204).nOut(150).activation("leakyrelu")
.weightInit(WeightInit.RELU).build())
.layer(2,
new OutputLayer.Builder(LossFunction.MSE).activation("identity").nIn(150)
.nOut(numOfActions).build()).pretrain(false).backprop(true).build();
```

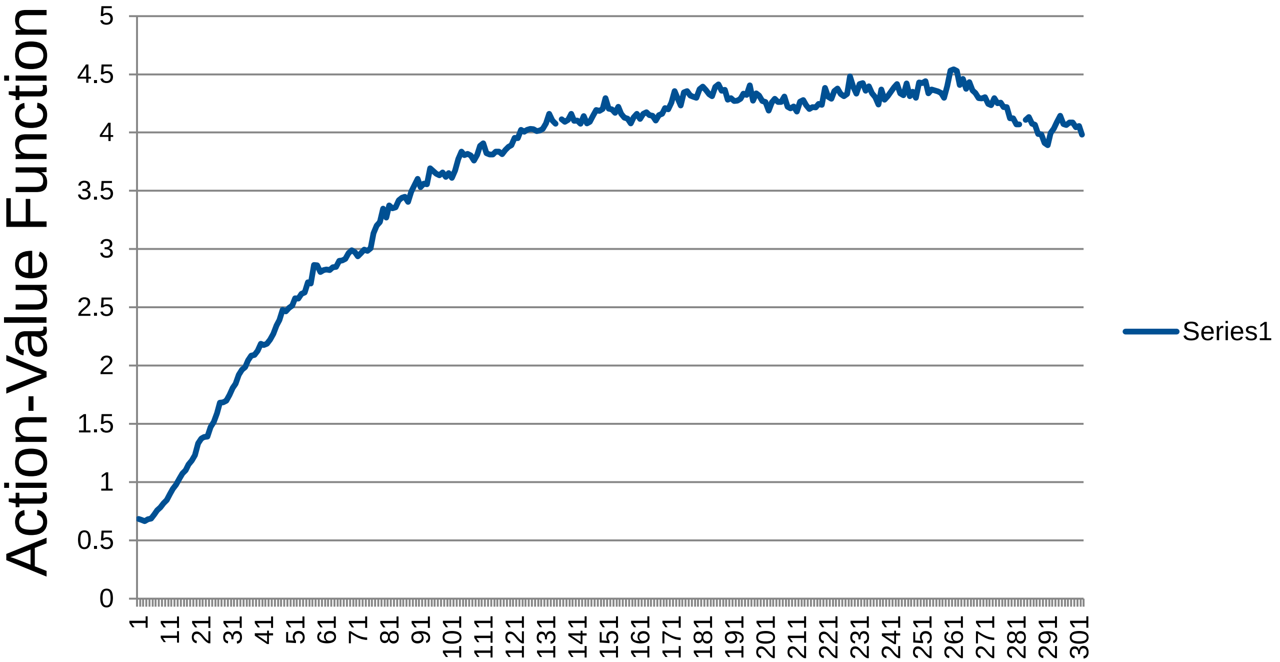
TESTING

1. Generate random flow at the source links with constant mean inter-arrival times.
2. Determine the net delay experienced by all vehicles with no ramp metering over K time steps.
3. Determine the net delay experienced by all vehicles with the ramp metering strategy determined by the trained neural network i.e. choose the action with the highest assigned probability by feeding forward the current state.
4. Compare and plot the results from steps 2 and 3.

RESULTS

TUMCREATE

PROOF OF Q-LEARNING



Without Ramp Metering	With Deep-RL based ramp metering
1026160	947915

Source Link ID	Mean IAT
30633	3800
82	1400
29310	1300
28946	1000
30790	1400
37980	1400
28595	1400
29152	1000
29005	1000
29553	1500
28613	1500
31991	800

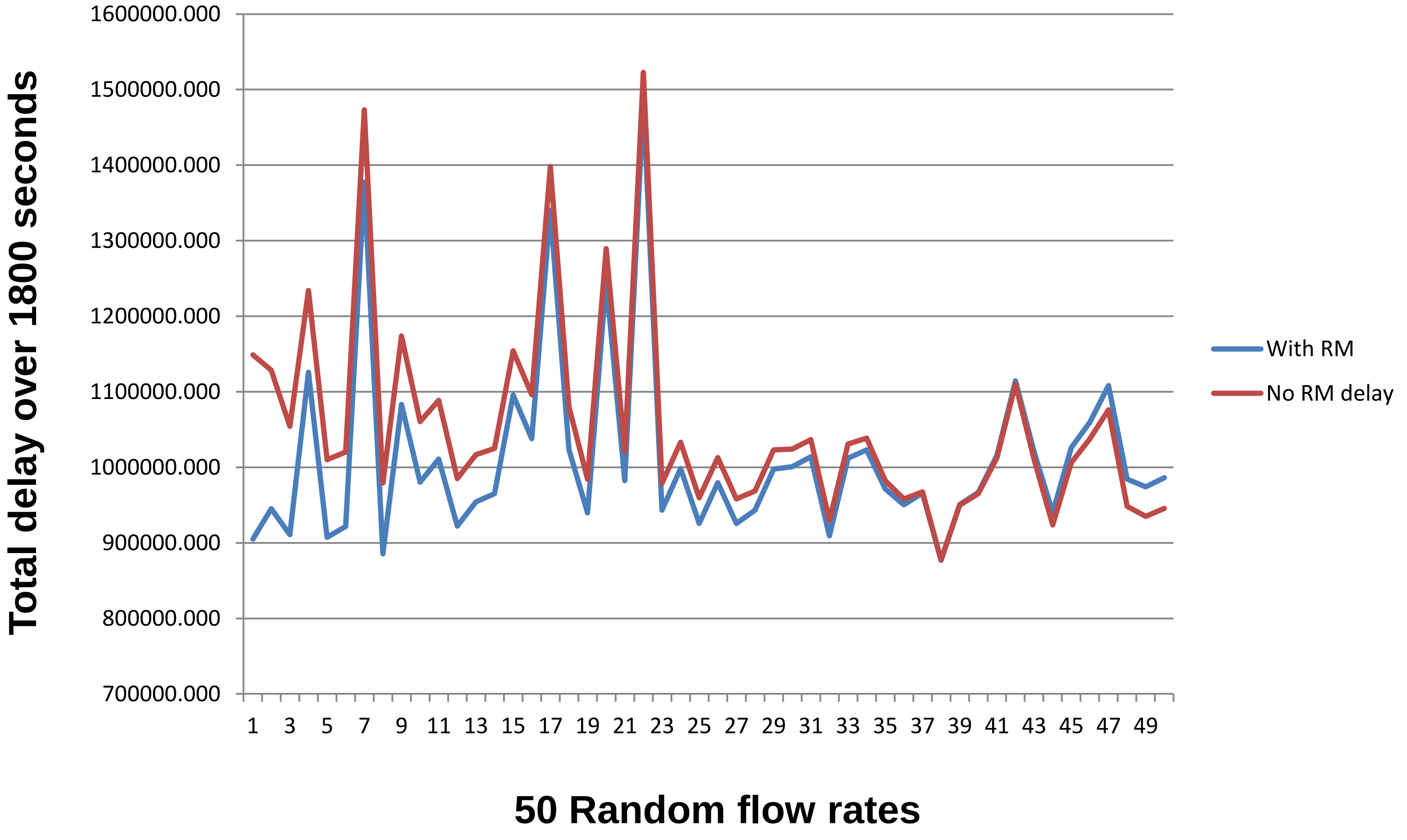
RESULTS

TUMCREATE

12	4254	RGGG		492	5225	GGGR		972	6492	RGGG		1452	8217	GGGR
24	3980	GGGR		504	5349	RGGG		984	6551	RGGG		1464	8215	GGRG
36	3998	RGRR		516	5350	GGGR		996	6611	GGGR		1476	8202	RGGG
48	4049	RGGG		528	5389	GGGR		1008	6620	GGGR		1488	8272	RGGG
60	4086	GRRR		540	5463	GGGR		1020	6665	RGGG		1500	8374	RGGG
72	4086	GRRR		552	5466	RGGG		1032	6717	RGRR		1512	8399	RGRR
84	4129	GRRR		564	5539	RGRG		1044	6731	GGGR		1524	8408	RGRR
96	4120	RGGG		576	5532	RGGG		1056	6765	GGRR		1536	8442	RGRR
108	4143	RGGG		588	5477	GGGR		1068	6802	GGGG		1548	8509	RGGG
120	4164	RGGG		600	5528	GGGG		1080	6847	RGRR		1560	8583	RRGR
132	4214	RGRG		612	5605	GRGG		1092	6938	RGGG		1572	8609	RGGG
144	4203	GGRR		624	5597	RGRG		1104	6957	RGGG		1584	8627	GGGR
156	4219	GGGR		636	5653	RGGG		1116	7037	GGGG		1596	8675	GGGR
168	4254	GGGR		648	5663	GGGR		1128	7106	RGGG		1608	8699	RGGG
180	4318	RGGG		660	5677	GGGR		1140	7117	GGGG		1620	8711	GGGR
192	4377	RGGG		672	5706	RGGG		1152	7122	RGGG		1632	8751	RGGG
204	4359	RGGG		684	5727	GGGR		1164	7268	GGGR		1644	8794	RGRR
216	4391	RGGG		696	5754	GGGR		1176	7300	RGGG		1656	8946	RGGG
228	4448	GGGR		708	5774	RGGG		1188	7312	RGGG		1668	9005	RGGG
240	4464	GRRR		720	5839	RGGG		1200	7326	RGGG		1680	9096	RGGG
252	4529	RGRR		732	5900	GGGR		1212	7384	RGRG		1692	9108	GGGR
264	4587	RGGR		744	5920	GGRR		1224	7439	GGGR		1704	9193	GGGR
276	4641	GGRR		756	5887	GGGR		1236	7504	GGRR		1716	9233	RGGG
288	4730	RGRR		768	5924	GGGR		1248	7582	RGGG		1728	9317	GRRR
300	4761	RGRR		780	5935	GGGR		1260	7588	RGGR		1740	9378	RGGG
312	4766	RGGG		792	5988	GGGG		1272	7608	GRRR		1752	9388	RGGG
324	4721	GGGG		804	6026	GGGR		1284	7643	GGRR		1764	9455	RGGG
336	4780	GGGR		816	6111	GGGR		1296	7601	RGGG		1776	9559	GGGR
348	4841	GGGR		828	6113	GGGR		1308	7687	RGGG		1788	9605	RGGG
360	4856	RGGG		840	6168	GGGR		1320	7760	GGRG		1800	9716	GGGR
372	4884	GGRR		852	6259	GGGG		1332	7806	RGGG				
384	4972	GGGG		864	6261	RGGG		1344	7891	RGGG				
396	5018	GGGR		876	6301	GGGR		1356	7897	GGGG				
408	5056	GGGR		888	6333	GGGR		1368	7945	GGRR				
420	5107	GGGG		900	6413	GGGR		1380	7984	GGGR				
432	5090	GGGG		912	6461	RGGG		1392	7970	GGGR				
444	5156	GGRG		924	6426	RGGG		1404	8012	RGGR				
456	5175	RGGG		936	6436	RGGG		1416	8082	RGGG				
468	5188	RRRG		948	6453	RGRR		1428	8132	GGRR				
480	5240	RGGG		960	6493	RGRG		1440	8155	RGGG				

RESULTS

TUMCREATE



CONCLUSION

1. The results how that deep reinforcement learning has promise for optimizing traffic flow through control mechanisms such a ramp metering.
2. A more suitable emulator (Cellular Automata based) is required to analyze and optimize the effect of other control mechanisms such as variable speed limits, adaptive cruise.
3. I have not found a single paper which uses Deep Q learning for traffic flow optimization and more specifically with ramp metering. This is the biggest contribution of this paper.