Literature survey

Azlaan Mustafa Samad

Contents

1 Introduction

Existing traffic light causes numerous issue such as delays, accidents, noise and air pollution and monetary losses. According to a CNN report, a 2012 study by Washington University in St. Louis noted that long commutes eat up exercise time. Thus, long commutes are associated with higher weight, lower fitness levels, and higher blood pressure—all strong predictors of heart disease, diabetes [20]. Non Smart traffic light leads to non-optimised and inefficient traffic flow.

In some recent research work [30], reinforcement learning (RL) is being used in order to control the traffic light by treating the problem as a sequential decision making problem. In RL, the agent learns from trial and error by interacting with the environment and the goal of the agent is to maximise the reward in the long term. In case of application of RL in traffic flow problems, Markov Decision Process is used to model the problem, where the states represent the different configurations of the traffic light at the intersection, actions are the changing of the traffic lights, rewards are formulated using factors like waiting time, delays, teleports(specific to SUMO [3] simulator), emergency stops etc. A Deep neural network or Deep Q-Learning is used to find an optimal policy which chooses action in order to maximise the total cumulative reward of the state. After learning to optimise the traffic flow for a single intersection, the same idea can be extrapolated to multi-agent system, where the number of intersection is more than one. This can be done using the idea of transfer planning [19] for training the agent for smaller sub-problems i.e global coordination is decomposed into local coordination using the theory of coordination graphs to find a joint global optimal action using max-plus algorithm [13].

2 Deep Reinforcement Learning

2.1 Reinforcement Learning.

Reinforcement Learning is a type of learning in which the states are mapped to actions in order to maximise a numerical reward signal. The action taker is known as the agent. It observes the environment either fully or partially and takes some action in order to land in a different state and receives a reward for taking that particular action. This action not only effects the current reward but also all subsequent rewards. The agent is not told which action to take instead it must learn which action yields most reward through a process of trial and error.

2.2 Markov Decision Process

Markov Decision process (MDP) is the mathematical framework of sequential decision making, where actions influence immediate reward, and subsequent future rewards. In this process, the agent observes the environment at each

time steps $t = 0, 1, 2, 3...$, in the form of state S_t selects some action At. One time step later, it receives a numerical reward R_{t+1}, and ends up in state S_{t+1}.

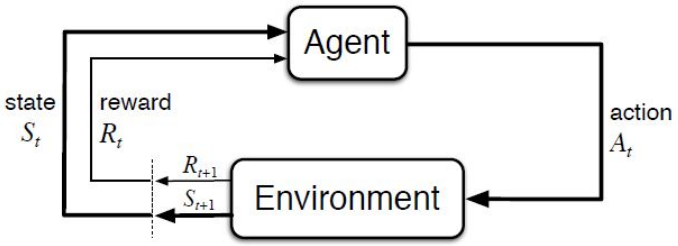

Figure 1: The agent–environment interaction in a Markov decision process [25]

MDP is formally represented as:

- $\mathcal{S}$ is the space of possible states;
- $\mathcal{A}$ is the space of possible actions;
- $p(s', r \,|s, a) \doteq Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$,
  the transition probability of ending up in state s' and obtaining reward r from previous state s by taking an action a

The agent's goal is to maximize total reward it receives over time, this means maximising not just immediate reward but cumulative reward in the long run. This can be represented as the following:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

where γ is the discounted rate such that $0 \leq \gamma \leq 1$.

γ determines the present value of future rewards. If $\gamma = 1$, then every reward is weighed equally, whereas when it equals 0, then the agent is myopic and is concerned only with maximising immediate rewards.

It can also be represented as:

$$G_t \doteq R_{t+1} + \gamma G_{t+1} \tag{2}$$

Policy is a mapping from state to probabilities. If an agent follows a policy π at time t, then $\pi(a|s)$ is the probability of taking an action $A_t = a$ when in state $S_t = s$ at time t. In Reinforcement Learning, this policy is updated from experience in order to attain maximum cumulative reward.

Value function is an estimate of how good it is for an agent to be in a particular state, which implies the expected future reward. It is the expected

cumulative reward when starting in state s and following a policy π.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right], \tag{3}$$

for all $s \in \mathcal{S}$.

Action-value function denoted as $q_\pi(s, a)$, as the expected return starting from state s, taking the action a, and thereafter following policy π:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{4}$$

where $\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \,|s, a) = 1$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$.

In RL, the values of $v_\pi(s)$ and $q_\pi(s, a)$ can be learned from experience. If one maintains the average of return that follow a particular state, for each time the state has been encountered then this value converges to the $V_\pi(s)$, since the number of times the state is encountered is infinity. If average is kept for each action taken when in a particular state, then this will converge to the action value function $q_\pi(s, a)$.

Bellman Equation: It is a representation of the value of the state, assuming one takes the best possible action now and at each subsequent step. Below, the recursive relationship between the value function of a state with respect to other state is shown. This is also known as the Bellman Equation.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma \mathbb{E}_\pi \left[\mathbb{G}_{\approx +\nVdash} \mid \mathbb{S}_{\approx +\nVdash} = \sim'\right]\right] \\ &= \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s')\right] \end{aligned} \tag{5}$$

for all $s \in \mathcal{S}$.

Optimal Policies and Optimal Value Function:

Optimal policies are the policies that secure maximum rewards in the long term (cumulative reward). A policy π is better or equal to another policy π' if its expected return is greater than or equal to that of π' for all states $s \in \mathcal{S}$.

Therefore, $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$. This policy π is often known as optimal policy, and often written as π^*.

Optimal State-Value Function:

$$v_*(s) \doteq \max_\pi v_\pi(s) \tag{6}$$

for all $s \in \mathcal{S}$.

Optimal Action-Value Function:

$$q_*(s,a) \doteq \max_{\pi} q_\pi(s,a) \tag{7}$$

for all $s \in \mathcal{S}$ and $A \in \mathcal{A}(s)$.

Its the expected reward when in state s and taking an action a and thereafter, following an optimal policy. Therefore, the optimal action-value function can be represented as a function of the optimal state-value function in the following way:

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]] \tag{8}$$

Bellman Optimality equation:

According to this, the value of a state under an optimal policy must equal the expected return for the best action from that state.

$$\begin{aligned} v_*(s) &\doteq \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_*(s') \right] \end{aligned} \tag{9}$$

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}), a') \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_*(s', a')] \end{aligned} \tag{10}$$

The Bellman Equation can be explained better with Figure 2. This diagram is also called the backup diagram. Here, the open circle represent the state, while the solid circle represent the state-action pair. The agent is in state s, representing the root node at the top and based on a policy, takes a certain action a, then the environment responds and lands the agent in any of the next probable state s' securing a reward r. The Figure 3 represents the backup diagram for the optimality equations.

If the state of the environment is well defined or in other words if the transition probability is known, then dynamic programming methods can be used to find the optimal solution using the recursive definition. One of these method is Value Iteration, in which the value function is updated for all states by updating each q-value and then using the maximum q-value to update the value function.

Explicitly solving the Bellman optimality equation provides one route to finding an optimal policy, and thus to solving the reinforcement learning problem. However, this solution is rarely directly useful. This is so because it is

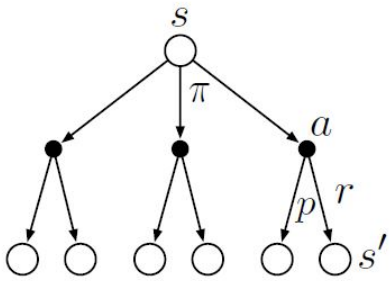

Figure 2: Backup Diagram for v_π [25]

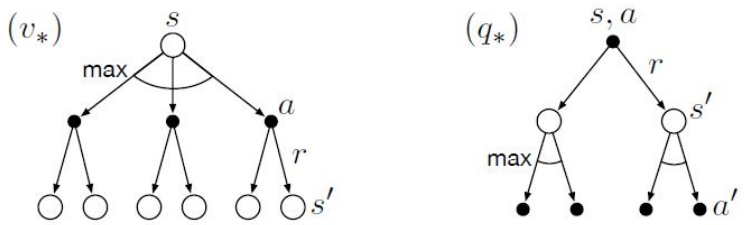

Figure 3: Backup Diagram for v_* and q_* [25]

based on the following three assumptions, which are rarely met: (1)the environment is accurately known, (2)Markov Property, (3)sufficient computational power.

Therefore, in many cases the transition probability is unknown. Under such scenario, the agent uses RL algorithms, the agent learns a mapping from state to actions from interacting with the environment and receiving feedback.

There are two types of Reinforcement Learning:
1. Model-based: Agent samples from the environment to estimate the transition probability, then uses planning algorithm to find an optimal policy,
2. Model-Free: Agent directly estimates the state-action value function from experience.

2.3 Tabular Q-Learning

Q-learning is a model-free reinforcement learning algorithm. That is, it does not build its own model of the environment's transition functions, but rather directly estimates the so-called Q-value of the s,a-pair, $q(s, a)$. Specifically, Q-learning is an off-policy algorithm, which is a class of algorithms that uses a different policy for estimating Q-values than for action-selection. That is, Q-learning updates the Q-values of the current s,a-pair using the greedy policy to estimate the Q-value of the optimal policy of the next s,a-pair.

In traditional Q-learning, the agent employs a lookup table of s,a-pairs and iteratively updates the Q-value estimates using:

$$q_{t+1}(s,a) = q_t(s,a) + \alpha[R_t + \gamma[\max_{a'} q_t(s_{t+1}, a'; \theta_t)] - q_t(s,a)] \tag{11}$$

In words, the difference between the current estimate of the s,a-pair, and the actual value of the s, a-pair. However, since the true value of the s, a-pair is not known upfront, the agent instead uses the current reward signal and the maximizing Q-value of the next state as a proxy for the true value.This is called tabular Q-learning, and it has the nice property that it converges given infinite samples.

2.4 Q learning with Function Approximation

Extending reinforcement learning to function approximation also makes it applicable to partially observable problems, in which the full state is not available to the agent.A solution to the problem of continuous S is function approximation, where supervised machine learning algorithms are used to approximate the Q-function. Q-value is a function parameterised by weight θ. These weights can be updated using gradient descent methods, minimizing the mean squared error between the current estimate of $q(s,a)$ and the target, which is defined as the true Q-value of the s, a-pair under policy $q_\pi(s,a)$.

The gradient descent update can be derived by taking the derivative of the mean squared error (MSE):

$$MSE(\theta) = \sum_{s \in \mathcal{S}} P(s) \left[q_\pi(s,a;\theta^*) - q_t(s,a;\theta_t)\right]^2 \tag{12}$$

where $P(s)$ is the sampling distribution, or the probability of visiting state s under policy π.

With the q-function approximation represented as a function with learnable parameters, a regular supervised learning method can be used to approximate the true Q-function.

2.5 Deep Learning

2.5.1 Neural Network

A neural network is a machine learning model parameterised by a set of parameters θ that maps an M-dimensional input vector, $\vec{x}$ through a series of hidden layers and activations, to a K-dimensional output vector, $\vec{y}$. It is used as a non-linear function approximator. Specifically, a neural network consists of interconnected layers, where each layer computes a linear mapping between the input x and its weights w, adding a bias term b and mapping the result through a non-linear activation function - needed to introduce non-linearity

into the model, e.g. a rectified linear unit. For example, mapping input vector $\vec{x}$ through one hidden layer with weights $W_0 \in \theta$, bias term $b_0 \in \theta$ and non-linearity h_0 results in the following equation:

$$\vec{x}' = h_0(W_0\vec{x} + b_0) \tag{13}$$

The output $\vec{x}''$ can be used as input to the next layer, with e.g. weights $W_1 \in \theta$, bias $b_1 \in \theta$ and non-linearity h_1:

$$\vec{x}'' = h_1(W_1 h_0(W_0\vec{x} + b_0) + b_1) \tag{14}$$

And so on. As the network grows deeper, the model can approximate more complex functions, but it also becomes harder to train. For that reason, much of the field of deep learning is dedicated to solving problems such as finding more reliable and faster methods of training neural networks and escaping local minima.

2.5.2 Convolution Network

Convolution Neural Networks(CNN) are analogous to Artificial Neural Networks (ANN) but with a difference. CNN is primarily used in the field of pattern recognition within images. This allows us to encode image-specific features into the architecture, making the network more suited for image-focused tasks-whilst further reducing the parameters required to set up the model [21]. In ANN the input layer is fully connected to a series of hidden layers which ultimately is connected to the output layer as shown in Figure 4. Whereas in CNN, only small regions of the input neurons are connected to the neurons in the hidden layer, these regions of the input layer are known as the local receptive fields. The local receptive field is translated across an image to create a feature map from input layer to the hidden layer.

Like a typical ANN, CNN also have neurons with weights and biases. The model learns these values during the training process and it continuously updates them with each new training example. However in CNN, the weights and biases are same for all neurons in a given hidden layer. This means all neurons are detecting the same feature such as an edge or blob in different regions of an image. Then the output of each neuron is transformed using an activation function. This can be further transformed by applying a pooling step for reducing the dimensionality of the feature map. Finally, in the last hidden layer each neuron is fully connected to the output layer, this produces the final output.

The architecture of the CNN can be divided into the following subdivisions as shown in Figure 5 [21]:

- As found in other forms of ANN, the input layer will hold the pixel values of the image.
- The convolutional layer will determine the output of neurons of which are connected to local regions of the input through the calculation of the scalar product between their weights and the region connected to the input

volume. The rectified linear unit (or ReLu) aims to apply an 'elementwise' activation function such as sigmoid function to the output of the activation produced by the previous layer.

- The pooling layer will then simply perform downsampling along the spatial dimensionality of the given input, further reducing the number of parameters within that activation.
- The fully connected layers will then perform the same duties found in standard ANNs and attempt to produce class scores from the activations, to be used for classification. It is also suggested that ReLu may be used between these layers, as to improve performance.

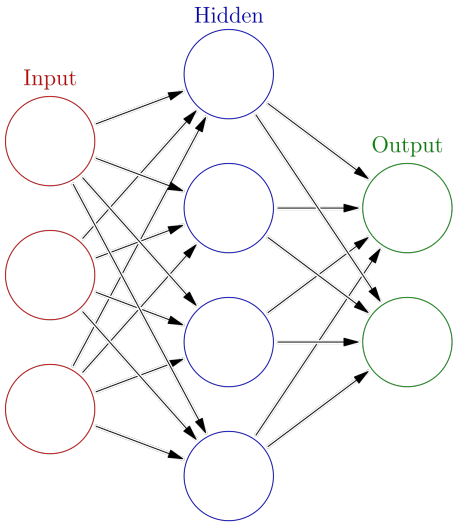

Figure 4: ANN [31].

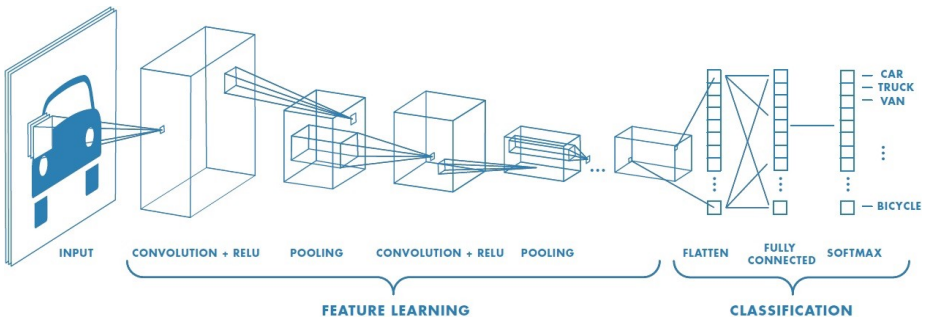

Figure 5: A typical CNN [1].

In the Convolutional layer of the network, features are extracted form the images. It performs basic matrix multiplication between the matrix representing the pixels in the image and the kernel matrix. The Kernel matrix is a random matrix which slides over the image matrix and performs an element-wise matrix multiplication and adds the result in order to produce a feature map. This can be better explained by the Figure 6 below.

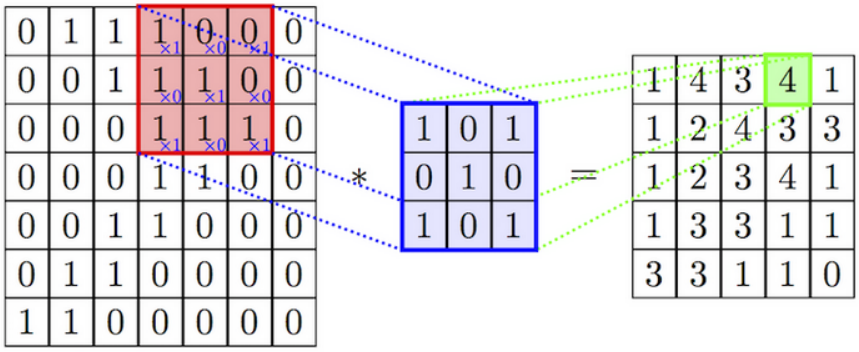

Figure 6: Convolutional Operation: The matrix in blue represents the Kernel matrix and it slides over the input matrix(bigger matrix) to produce an output.

Usually, the real world data has non-linearity in them whereas the convolutional operation is linear in nature and therefore in order to take into account the non-linearity, we usually add another operation after the convolutional operation, this is Rectified Linear Unit(ReLU). It can be mathematically represented as follows:

$$Output = Max(0, x) \tag{15}$$

Then comes the pooling step, in pooling the dimensionality of the feature map in reduced but the important information is not disregarded. One of the ways to do this, is by using the Max Pooling function. Max Pooling function defines a window over the feature map and replaces the window by a single value which is the maximum in that particular window and it slides over the entire feature map in order to perform this action as shown by the Figure 7.

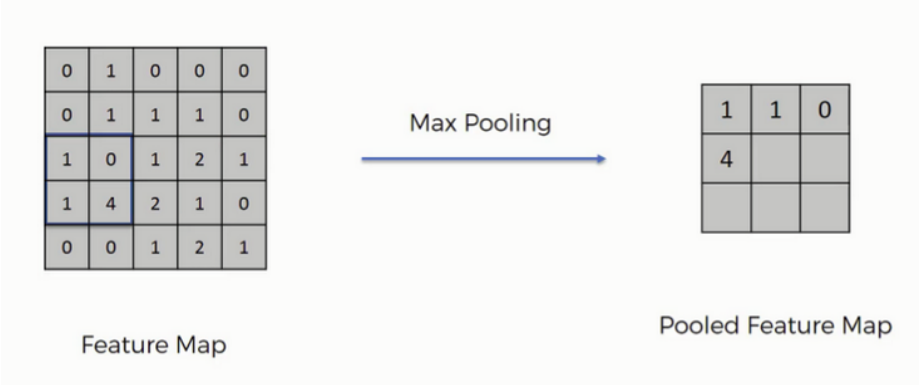

Figure 7: Max Pooling Operation.

2.5.3 Training the Neural Network

At the beginning of training of the neural network, we randomise the weights. But in order to produce optimum results we have to updates the weights so that our final result is as close as possible to the target weights. In order to do this, we define a Loss function or an error function and minimise it. There are various methods used in order to update the weights to reach an optimum result. These are known as the Optimisation Algorithms, which help us minimise the Loss or Objective function and are based on certain learnable parameters of the model.

2.5.4 Gradient Descent

Gradient is a multi variate generalisation of a derivative. It has a direction and points to the direction of steepest increase of the function. Our job is to minimise the loss function, therefore we take a step in the negative direction of the gradient. If we compute the gradient of the loss function with respect to our weights and take a step in the negative gradient and update the new weights, eventually our loss function will decrease and converge to some local minima [4]. The idea is to choose a optimum step in the negative direction. If the step is too big, then the algorithm diverges and we jump over the minima. And if it is too small, we might converge to the local minima. Mathematically, it can be represented as the following:

$$w^{(i+1)} = w^{(i)} - \eta \nabla E(w^{(i)}), \tag{16}$$

where,
E : loss function
w : weight
η : learning rate.

In machine learning, the error function is generalised in the form of sums as

follows:

$$E(w) = \frac{1}{n}\sum_{i=1}^{n} E_i(w) \tag{17}$$

There are various types of Gradient Descent method. The important ones are Mini-Batch Gradient Descent(MBGD) and Stochastic Gradient Descent(SGD). In Mini-Batch, we approximate the derivative on some small batch of the dataset and use it to update the weights as shown in equation (18), whereas in Stochastic approach we update the weights for each training example as shown in equation (19).

Mini-Batch GD:

$$w^{(i+1)} = w^{(i)} - \eta \sum_{i=1}^{n} \nabla E_i(w)/n, \tag{18}$$

Stochastic GD:

$$w^{(i+1)} = w^{(i)} - \eta \nabla E_i(w), \tag{19}$$

In Mini-Batch updates can be really slow and memory can be an issue for large datasets. While in SGD, there can be fluctuations.

2.5.5 Momentum

Momentum [24] was invented to accelerate the SGD and soften the fluctuations. It does so by adding a fraction of the past update vector to the current one. It can be mathematically represented as follows:

$$v(t) = \gamma v(t-1) + \eta \sum_{i=1}^{n} \nabla E_i(w)/n, \tag{20}$$

$$w^{(t+1)} = w^{(t)} - v(t) \tag{21}$$

where γ is the momentum term.

2.5.6 AdaGrad

AdaGrad [7] is a method used to adapt the hyper-parameters accordingly. It makes big updates for infrequent parameters and small updates for frequent parameters. It uses different learning rates for every parameter w at every time step. If we consider $g(t,i)$ to be the gradient of the loss function w.r.t to parameter w_i at the time step t, then the update is given by:

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i} \tag{22}$$

where G_t is a diagonal matrix where each of the diagonal element i,i is the sum of the squares of the gradient with respect to w_i up to time step t, while ϵ is a smoothing term that avoids division by zero. If past gradients for parameter i

were large, the learning rate for i is small. On the other hand, if past gradients for i have been small/sparse, the learning rate for i is large.
Thus, it eliminates the need to upgrade the learning rate manually. AdaGrad's main weakness is the accumulation squared gradient term in the denominator and this leads to a continuous decrease in the learning rate, thereby making the learning process very slow.

2.5.7 RMSProp

RMSProp [27] is used to modify the AdaGrad and solve the problem of decaying learning rate. It does so by changing the gradient accumulation with a exponentially weighted moving average.

$$G_{t+1,ii} = \gamma G_{t,ii} + (1-\gamma)(g_{t,i})^2 \tag{23}$$

In contrast to the AdaGrad, here we weigh recent past more heavily when compared to the distant past.

2.5.8 ADAM

ADAM (Adaptive Moment Estimation) [12] is a variant of combination of AdaGrad and RMSProp [27].
It makes use of both the average first moment(mean)and the average second moment(variance) of the gradient.

$$m^{t+1} = \beta_1 \cdot m^t + (1-\beta_1).\nabla E \tag{24}$$

$$v^{t+1} = \beta_2 \cdot v^t + (1-\beta_2) \cdot \nabla E^2 \tag{25}$$

where, β_1 and β_2 are hyperparameters.

2.5.9 Batch Normalisation

Batch Normalisation [11] is a pre-processsing method, which helps in speeding up the neural network by incorporating higher learning rates. It is used in order to fix the problem of internal co-variance shift. As described in the paper [11], the internal layers of a neural network have to adapt during training because the distribution of the activations are constantly changing in every layer which slows down the training process, therefore each layer has to learn to adapt to the new distribution. This is known as internal covariance shift [11]. Batch Normalisation solves this problem by normalising the input to the next layer as following:

$$\overline{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{26}$$

where ϵ is a smoothing term that avoids division by zero, μ_B is the batch mean and σ_B^2 is the batch variance and B is the batch as shown below,

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{27}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{28}$$

Finally, the output is represented as:

$$y_i = \gamma\overline{x_i} + \beta \tag{29}$$

As mentioned in the original paper by Ioffe and Szegedy [11], simply normalizing each input of a layer may change what the layer can represent. For instance, normalizing the inputs of a sigmoid would constrain them to the linear regime of the nonlinearity. To address this, we make sure that the transformation inserted in the network can represent the identity transform. In order to do this, two parameters γ and β are introduced, these parameters are learned along with the original model parameters, and restore the representation power of the network. In a nutshell, Batch Normalisation helps in speeding the learning procedure, by allowing for higher learning rates and also makes more activation function viable.

3 Multi-Agent Coordination

When the number of agents increases, the common goal of these multiple agents together is to find a joint optimal action such that the common reward is maximised. The joint optimal action is represented as the following:

$$\vec{a^*} = (a_1, a_2, ..., a_N) \tag{30}$$

where a_i is the local action taken by the agent i. Using a centralised approach to finding the joint optimal action is not feasible since the joint action space increases exponentially as the number of agents increase.

In case of Multi-agent traffic light control problem, Van der Pol [28] describes each intersection as a single agent. A two agent traffic light problem can be shown as in the Figure 8.

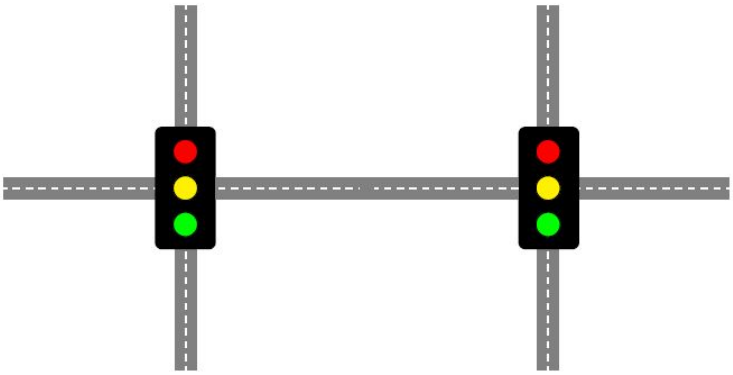

Figure 8: Two Agent Traffic Light Scenario.

3.1 Coordination Graph

Coordination graph is a graphical representation of the decomposition of the global payoff function(reward function) g into set of smaller local factors $f_{i,j}$, where each factors is a smaller function(function with less variables), depending on the subset of the agents in the Figure 9 and the graph as shown below:

$$CG(a_1, a_2, a_3, a_4) = f_{1,2}(a_1, a_2) + f_{2,3}(a_2, a_3) + f_{3,4}(a_3, a_4) \tag{31}$$

where $f_{i,j}$ are the factored components of the graph and depend on the actions of their respective agents.

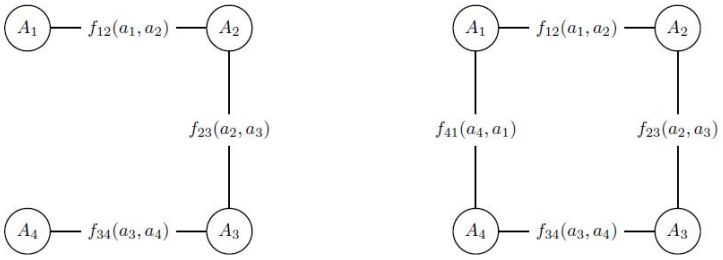

Figure 9: Examples of Multi Agent systems represented as a Cyclic(right) and Acyclic(left) Coordination Graphs.

The joint optimal action that maximises the global payoff function g:

$$\vec{a^*} = \underset{\vec{a}}{arg\,max}\; g(\vec{a}) \tag{32}$$

Equation 31 above can be decomposed into the following form:

$$\max_{\vec{a}} g(\vec{a}) = \max_{a_1} ... \max_{a_N} g(\vec{a}) \tag{33}$$

Using the distributive law for the max operator, sums and maximizations can be switched to find a more computationally efficient order and replacing the global payoff by its factored parts:

$$\max_{\vec{a}} g(\vec{a}) = \max_{a_1} [\max_{a_2} [f_{12}(\cdot) + [... \max_{a_N} f_{N-1,N}(\cdot)]]] \tag{34}$$

The above problem representation can be solved using coordination algorithms like Variable Elimination and Max-Plus algorithm.

3.1.1 Variable Elimination

Variable Elimination [10] is an algorithm used to solve the coordination graphs by eliminating agents one at a time and maximising over that agent, that is finding the best action of the eliminated agent for each action of the non-eliminated agents. This is an exact inference algorithm, but for large scale agents the problems scales exponentially and since it is not a anytime algorithm(an algorithm; which can be stopped anytime during running and get approximate results), it cannot be run for fewer iterations to get an approximate answer.

3.1.2 Max-Plus Algorithm

Max-Plus [13] is an inference algorithm based on message passing, used to find the maximum a posteriori (MAP) state in graphical models. It converges to an exact solution for acyclic graphs but cannot guarantee convergence for cyclic graphs. It is based on a message passing parameter as shown:

$$\mu_{ij}(j) = \max_{a_i} \left[f_{ij}(s, a_i, a_j) + \sum_{k \in ne(i) \backslash j} \mu_{ki}(i) \right] \tag{35}$$

Message containing information over locally optimal actions are sent between agents to iteratively find the optimal joint action. Thus, a message from i to j is as shown above, where $ne(i)\backslash j$ is the set of $i's$ neighbours, excluding j. In short, i sends a message to j that consists of a maximization over i and $j's$ factor and the messages i has received from its neighbours that are not j. By iteratively sending messages, max-plus converges to the maximal joint action in acyclical graphs. Moreover, the algorithm has an anytime solution, meaning that it can be run using fewer iterations, and still get an approximate solution.

3.2 Transfer Planning

Based on the previous work by Oliehoek et al. [28] [29] [19], transfer planning can be a good approach for solving Multi-agent systems. In Transfer Planning [19], Q-function is learnt for a subproblem of a larger multi-agent problem. Training

on the source problem results in an approximation of its Q-function. Provided that the source problem and other subproblems are similar, we can then re-use the source problem's Q-function for each subproblem in the larger multi-agent problem, rather than training a Q-function for each separate subproblem. in the environment introduced by multiple agents learning and acting simultaneously.

This transfer planning approach circumvents two problems present in multi-agent reinforcement learning. The first is the non stationarity in the environment introduced by multiple agents learning and acting simultaneously. By training on a source problem. the environment dynamics do not change during learning. The second is the cost of training many agents simultaneously. Because the source problems are independent, they can be solved independently (e.g. sequentially). Moreover, exploiting symmetries of our source further reduces the computational cost.

Transfer Planning can be formalised in the following way:

- $\sum$ the set of source problems $\sigma \in \sum$;
- $\mathcal{D}^{\sigma} = \{1^{\sigma}, ..., \mathrm{n}^{\sigma}\}$, the agent set for source problem σ;
- E maps each Q-value component e to a source problem $E(e) = \sigma \in \sum$;
- $\mathrm{A}^{\sigma} : \mathbb{A}(\mathrm{e}) \rightarrow \mathcal{D}^{\mathbf{E(e)}}$ maps agent indices $i \in \mathbb{A}(\mathrm{e})$ in the target task to indices $A^{e}(i) = j^{\sigma} \in \mathcal{D}^{\sigma}$in the source task σ for that particular component $\sigma = E(e)$.

Given a set of source problems and some (heuristic) Q-value functions for them, the transfer planning Q-value function Q_{TP} is defined as:

$$\mathcal{Q}^{\mathrm{e}}_{\mathrm{TP}}(\vec{\theta}^{\mathrm{t}}_{\mathrm{e}}, \mathrm{a}_{\mathrm{e}}) = \mathcal{Q}^{\sigma}(\vec{\theta}^{\mathrm{t}}_{\mathrm{A^e(e)}}, \mathrm{a}_{\mathrm{A^e(e)}}), \quad \sigma = E(e)$$

Thus, we only have to define the source problems, define the corresponding mapping A^e for each component e, and find a heuristic Q-value function $Q^{\sigma}(\vec{\theta^t}, a)$ for each source problem. Since the source problems are typically chosen to be small, we can treat them as non-factored and use the heuristic Q_{MDP}, Q_{POMDP}, Q_{BG} even the true Dec-POMDP value functions.

4 Deep Reinforcement Learning for Traffic Light Control

Traffic flow problem can be visualised as a Single or Multi Agent problem, which can be trained in order to optimise the traffic flow. In terms of traffic light control, the agent can be modeled as the traffic signal itself, which takes certain actions like changing the traffic light and receives rewards based on these actions and tries to maximise the cumulative long term reward effectively leading to smoothh flow without any delays. The simulation of the traffic flow can be performed in one of the simulation softwares known as SUMO(Simulation of Urban MObility) [14]. It is discussed further in the section 3.0.1

4.1 SUMO(Simulation of Urban MObility)

SUMO [14] is a free, open source software that allows for a realistic simulation of different traffic networks. It comes with plethora of tools which can be used for visualisation, emission calculations, network importing and finding the route. It allows to import maps from OpenStreetMap, VISUM, VISSIM etc. It can be used to model multimodal traffic like vehicles, pedestrians, public transport as well as cyclists. SUMO is implemented in C++ and only uses portable libraries.

4.2 Single Agent

One of the earliest work done in the application of Deep Reinforcement Learning in the field of traffic flow problem was done by Li et al [15] which used Deep Q-Learning to control a single intersection. Several other researches started exploring this area and came up with different ways to define the states, rewards fucntions and action for modelling the traffic flow problem.

4.2.1 States

The states in terms of traffic flow problem is usually defined in terms of vehicle's position and velocity. Similar approach is taken by Liang et al. [16]. The idea proposed is to divide the intersection in network of grids such that each element of the grid can contain maximum of one vehicle. The smallest element of the grid are small-sized square shaped. Depending on the presence of vehicle the grid is assigned binary values, either 1 or 0. Value is 1 when the vehicle is present or 0 when there is no vehicle in the grid. This can be quantified in terms of matrices, where corresponding to the cell of the grid, there is an element in the matrix, with binary values. This is more clear in the Figure 10 & 11 below.

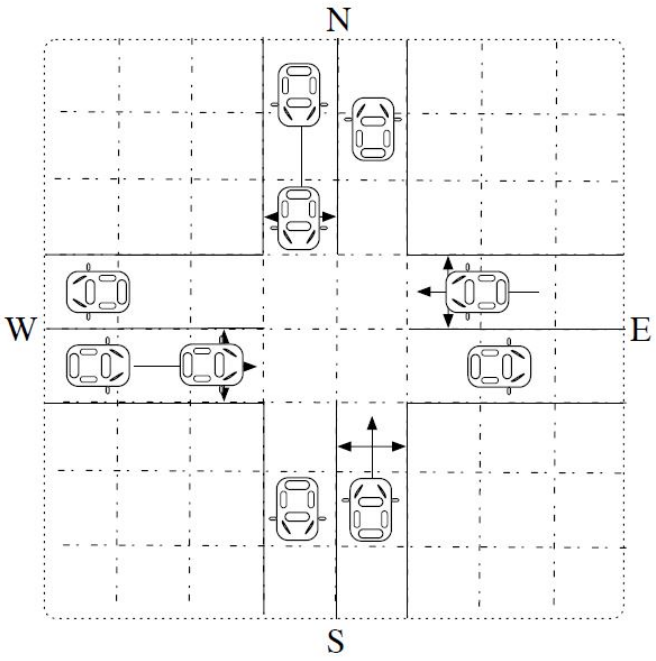

Figure 10: Snapshot of traffic intersection at a particular time [16].

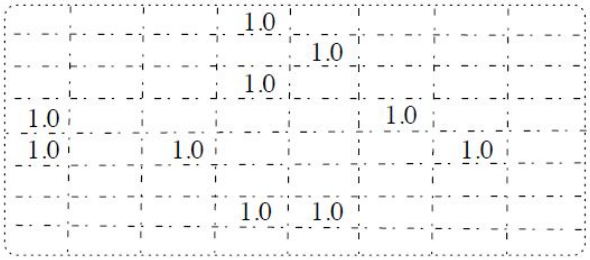

Figure 11: Corresponding position matrix on the road [16].

The value of vehicle's speed can be represented in the similar fashion, the cells contains the velocity of the vehicle when it resides inside the cell or else it is 0. Similar approach was taken by Van der Pol in her master thesis [28] and others researchers [8] [9].
However Rijken [23] in his PhD. thesis, proposes this very same method but argues that it can be computationally inefficient. He also suggests that when the size of the bianry matrix is chosen too large, one might encounter a situation in which a car seems to 'disappear'. This phenomenon occurs when the centres

of two different cars fall within the same cell. In that case, the cell will be denoted by a 1, but we will be unable to tell how many cars there are in that cell. Therefore he suggests an alternative approach for state representation, he represents the traffic situation as a vector of length n, where n is the number of lanes approaching the junction as shown below. This vector represents the number of vehicles in the different incoming lanes in the traffic network, i.e. it is vector that keeps a count of the number of vehicles. Only vehicles in "upstream" lanes are counted. That is, the vehicles that have passed the traffic light already are not counted.

$$Number of Lanes \begin{cases} \begin{bmatrix} 6 \\ 5 \\ 0 \\ 1 \\ 2 \end{bmatrix} \end{cases} \tag{36}$$

In other works, one of them being by Lin et al. [17], the data regarding the state of the intersection is collected by the sensors installed at traffic lights, this data is formatted into a triple $< C, H, W >$, where C is the number of channels, H is the height of input tensor, and W the width of input tensor. A series of intersections is divided into 3x3 grid containing 9 intersections. Each intersection has 4 arms whose length is 500 meters. Eight sensors are placed on each traffic light to monitor the halting vehicle number and the mean speed. Since two types of information: the halting vehicle number and the mean speed are collected in each intersection, the substate can then be formatted into a 2x4x4 tensor as shown in Figure 12. Therefore, the complete state s_t agent received is in a shape of (2, 12, 12).

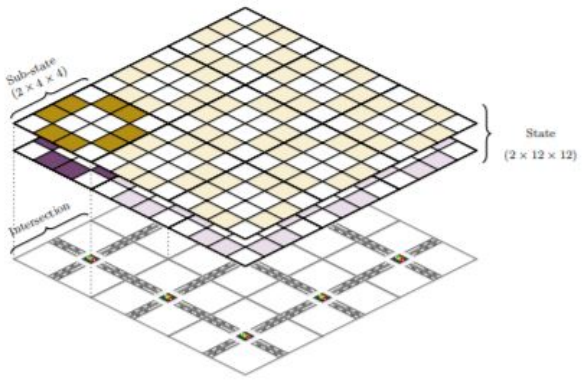

Figure 12: Traffic grid and corresponding formatted tensor [17].

In a different approach by Liu et al. [18], they make use of the length of

the waiting queues to be states for both vehicles and pedestrians in each traffic direction at an intersection, as expressed by:

$$S_{i,d}^t = \{q_{1i,d}^t, q_{2i,d}^t, ..., q_{ji,d}^t, m_{1i,d,L}^t, m_{1i,d,R}^t, ..., m_{ji,d,L}^t, m_{ji,d,R}^t\} \tag{37}$$

where i, j are IDs of intersections and $j \in N_i$; N_i is neighborhood intersections of i; $S_{i,d}^t$ is the state at intersection i, at day d and time t; $q_{ji,d}^t$ is the queue length from intersection j to intersection i, at day d and time t; $m_{ji,d,L}^t$ is the queue length for pedestrians at the left side from intersection j to i, at day d and time t; and $m_{ji,d,R}^t$ is the queue length for pedestrians at the right side from intersection j to i, at day d and time t.

4.2.2 Rewards

Several factors should be taken into account while formulating the reward function. One way to do is by penalising the agent every time the car stops which is commonly known as the waiting time. Other ways of penalising the agent is when the average speed of the vehicles in a lane is below the maximum allowed speed, well established as delay.

In a paper by Lin et al. [17], it is defined as the absolute negative difference between queue length in north-south/south-north direction and those in east-west/west-east direction, i.e.

$$r_t^{TLS_i} = - \mid maxq_t^{WE} - maxq_t^{NS} \mid \tag{38}$$

For each intersection TLS_i, q_t^{WE} is the number of halting vehicles in lanes from west to east or vice-versa. Similarly, q_t^{NS} is that from north to south or vice-versa.

In other works, Liang et al. [16] defines the rewards as the change of the cumulative waiting time between two neighboring cycles. Let i_t denote the i^{th} observed vehicle from the starting time to the starting time point of the t^{th} cycle and N_t denote the corresponding total number of vehicles till the t^{th} cycle. The waiting time of vehicle i till the t^{th} cycle is denoted by $w_{i_t,t}, (1 \leq i_t \leq N_t)$. The reward in the t^{th} cycle is defined by the following equation,

$$r_t = W_t - W_{t+1} \tag{39}$$

$$W_t = \sum_{i_t=1}^{N_t} w_{i_t,t} \tag{40}$$

Similar approach is used by Gao et al. [8] and Calvo et al. [5]. Whereas, Liu et al. [18] represents the reward function in the following way by taking into account the pedestrian waiting time as well:

$$R^t_{i,d}(a^t_{i,d}, a^t_{j,d}, S^t_{i,d}, S^t_{j,d}, W^t_{i,d}) = -\Bigg(\frac{w^t_{1,d}}{\mid N_i \mid}\sum_{j\in N_i} q^t_{ji,d} + \frac{w^t_{2,d}}{\mid N_i N_j \mid}\sum_{j\in N_i}\sum_{k\in N_j} q^t_{kj,d} + \\ + \frac{w^t_{3,d}}{2 \mid N_i \mid}\sum_{j\in N_i}(m^t_{ji,d,L} + m^t_{ji,d,R})\Bigg) \quad (41)$$

where $R^t_{i,d}$ is the reward at intersection i, at day d and time t; $a^t_{i,d}$ is the action at intersection i, at day d and time t; $w^t_{1,d}$ is the weight to present the local vehicular queues at intersection i, $w^t_{2,d}$ is the weight to present the neighbourhood vehicular queues at the neighbours of intersection i; $w^t_{3,d}$ is the weight to present the total pedestrian queues at intersection i and $W^t_d = \{w^t_{1,d}, w^t_{2,d}, w^t_{3,d}\}$. The sum of these weights equals 1.

$\sum_{j\in N_i} q^t_{ji,d}$ is the incoming vehicular queues from intersection j to intersection i; $\frac{1}{|N_j|}\sum_{j\in N_i}\sum_{k\in N_j} q^t_{kj,d}$ is the total vehicular queues at all neighbouring intersection js, including the outgoing vehicular traffic from intersection i to intersection j.

Noe [6] introduces another term called speed score, that is defined for detector i as:

$$speed\_score_i = min(\frac{V_{avg,i}}{V_{max,i}}, 1.0) \quad (42)$$

where $V_{avg,i}$ refers to the average of the speeds measured by traffic detector i and $V_{max,i}$ refers to the maximum speed in the road where detector i is located. Finally, the reward function is defined as:

$$reward_i = \alpha \cdot count_i \cdot (speed\_score_i - baseline_i) \quad (43)$$

4.2.3 Actions

Actions in case of traffic control problem are defined as the different combinations of traffic light signal at the intersection. It is usually expressed as different phases and the agent selects one of these phases or actions to maximise the long term cumulative reward. Liang et al. [16] defines the action to be taken by the traffic lights at the intersection in the following four phases: north-south green, east-north & west-south green, east-west green, and east-south & west-north green, while the other unmentioned directions are red by default. Let a four-tuple $< t_1, t_2, t_3, t_4 >$ denote the duration of the four phases in current cycle. It is best described by the Figure 13 where each circle means the duration of the four phases in one cycle. Time change from current cycle to succeeding cycle is discretised to 5 seconds. The duration of one and only one phase in the next cycle is the current duration added or subtracted by 5 seconds. This idea can be extended to more complex intersections with more than four streets meeting at a junction. The more complex the intersection, more are the number of phases.

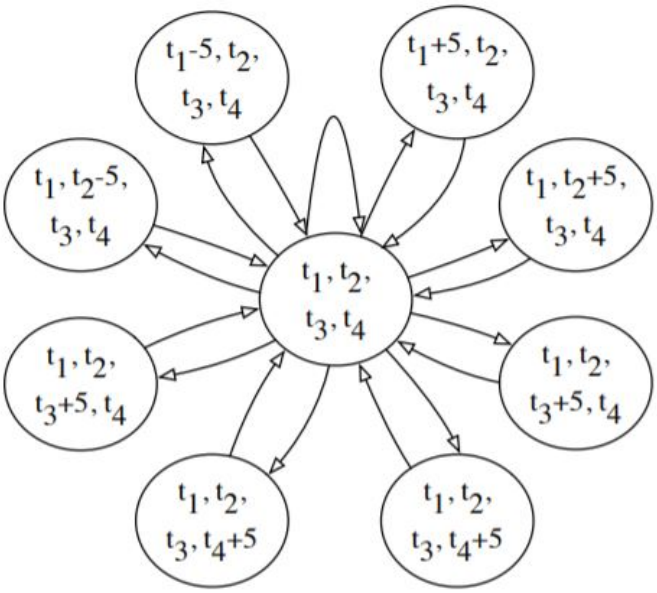

Figure 13: The action space [16].

Yellow signal is important for switching between two phases as it guarantees safety by allowing speeding vehicles to stop by providing them with enough time when the switch is being made between green and red signals. It can be formulated either as a fixed time for yellow light when switching between two actions or it can be selected as an action itself. The time duration of the yellow light is defined by Liang [16] as the ratio of the maximum speed and the most commonly seen deceleration as follows:

$$T_{yellow} = \frac{v_{max}}{a_{dec}} \tag{44}$$

This provides enough time for the traffic to come to a halt safely.

In the paper by Lin et al. [17], yellow light phase lasts for 3 seconds, while they use the similar approach of phases for defining the action space as shown in the Figure 14.

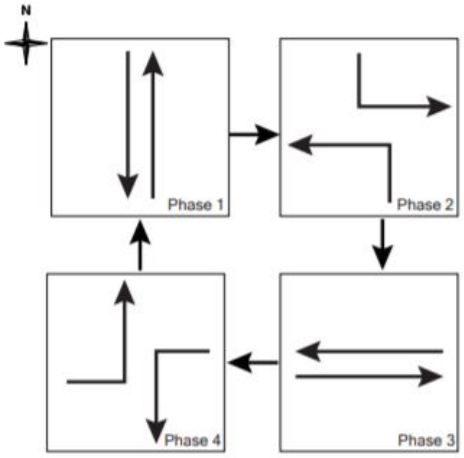

Figure 14: The four phases of traffic lights [17].

Similar phase approach is being used by Gendera et al. [9], Van der Pol [28] and other researchers [18] [5] [6].

Gao [8] divides each of the four lanes meeting at the intersection into four parts: L_0, L_1, L_1, L_3. This is an assumption made by Gao where each lane consists of four sublanes. The innermost lane (referred to as L_0) is for vehicles turning left, the middle two lanes (L_1 and L_2) are for vehicles going straight and the outermost lane (L_3) is for vehicles going straight or turning right. Vehicles at this intersection run under control of traffic signals: green lights mean vehicles can go through the intersection, however vehicles at left-turn waiting area should let vehicles going straight pass first; yellow lights mean lights are about to turn red and vehicles should stop if it is safe to do so; red lights mean vehicles must stop. The action defined for each lane can be understood by the Figure 15 and Figure 16.

Rijken [23] defines an admissible traffic light configuration as a D-tuple:

$$l =< s_1, s_2, ..., s_d > \tag{45}$$

where D is the number of signals (or the number of links) in the junction and s_d is the state of signal d. A signal can have one of three states: 'G' for priority green, 'g' for non priority green, and 'r' for red. Each junction has a set of admissible traffic light configurations $S = \{l_1, l_2, ..., l_c\}$. It is best visualised by the Figure 17, and it has $< G, G, g, r, r, r, r, G, G, G >$ is an admissible traffic light configuration that allows incoming traffic from the East and West to cross the junction. The link from East to South has a non-priority green light. As cars on that link want to turn, they have to give priority to the cars approaching from the West.

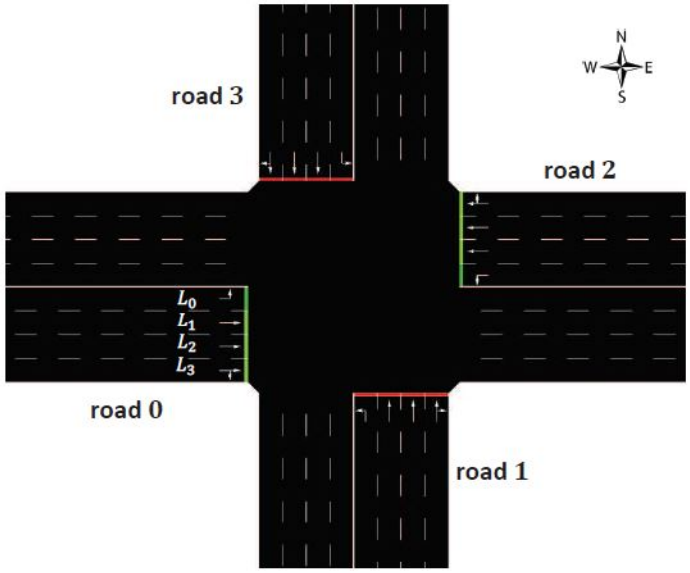

Figure 15: A four way intersection [8].

timeline			road 0				road 1				road 2				road 3			
			L_0	L_1	L_2	L_3	L_0	L_1	L_2	L_3	L_0	L_1	L_2	L_3	L_0	L_1	L_2	L_3
step $t-1$ green		τ_g	g	G	G	G	r	r	r	r	g	G	G	G	r	r	r	r
step t green		τ_g	g	G	G	G	r	r	r	r	g	G	G	G	r	r	r	r
step $t+1$	transition	τ_y	g	y	y	y	r	r	r	r	g	y	y	y	r	r	r	r
		τ_g	G	r	r	r	r	r	r	r	G	r	r	r	r	r	r	r
		τ_y	y	r	r	r	r	r	r	r	y	r	r	r	r	r	r	r
	green	τ_g	r	r	r	r	g	G	G	G	r	r	r	r	g	G	G	G

r: red light
G: green light
g: green light for vehicles turning left, letting vehicles going straight pass first
y: yellow light
L_i : lane i

Figure 16: Example of traffic signal timing for actions in Fig. 15 [8].

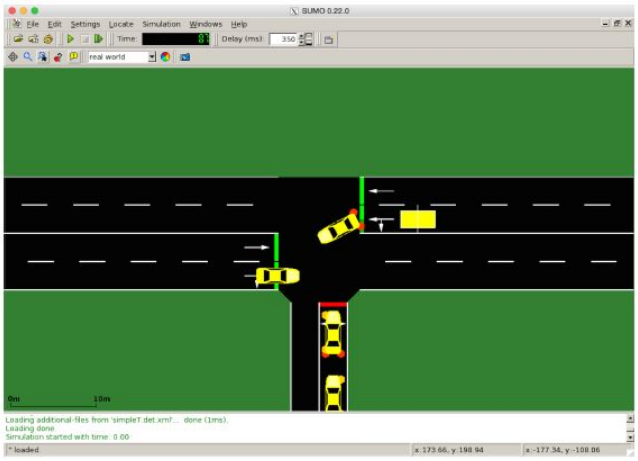

Figure 17: The T-junction considered here has ten links and three admissible traffic light configurations. The incoming lane from the South has four links: two going left and two going right. The other roads have three links each. The lane that crosses the junction has one link. The lane from which one can turn South has two links: one to cross the junction and one to turn South.

4.3 Multi Agent Traffic Light

As mentioned in earlier Section 3, when the number of agents increases, the common goal of these multiple agents together is to find a joint optimal action such that the common reward is maximised. A good approach will be using the method of Transfer Planning(TP) [19] (refer to Section 3.2).

In TP, once the joint Q-value function for the two-agent scenario is learned, this is used as the source problem in transfer planning. To solve the multi-agent scenarios, this source problem is used for each pair of neighbouring agents. After finding the local joint Q-value function for the factors in the multi-agent problem, the Q-function is re-used for all similar factors in the larger multi-agent problem. For an example regarding a four-agent scenario, see Figure 18, where the Q-values for two two-agent source problems - $\mathcal{Q}^{\sigma}$ and $\mathcal{Q}^{\zeta}$, for the rotated factor are learned separately, and then applied to a coordination algorithm in the four-agent scenario. For the three-agent scenario, $\mathcal{Q}^{\sigma}$ is re-used for both factors in the graph. Thus, the two-agent Q-function is re-used to compute a globally optimal joint action for the three-agent scenario in Figure 19.

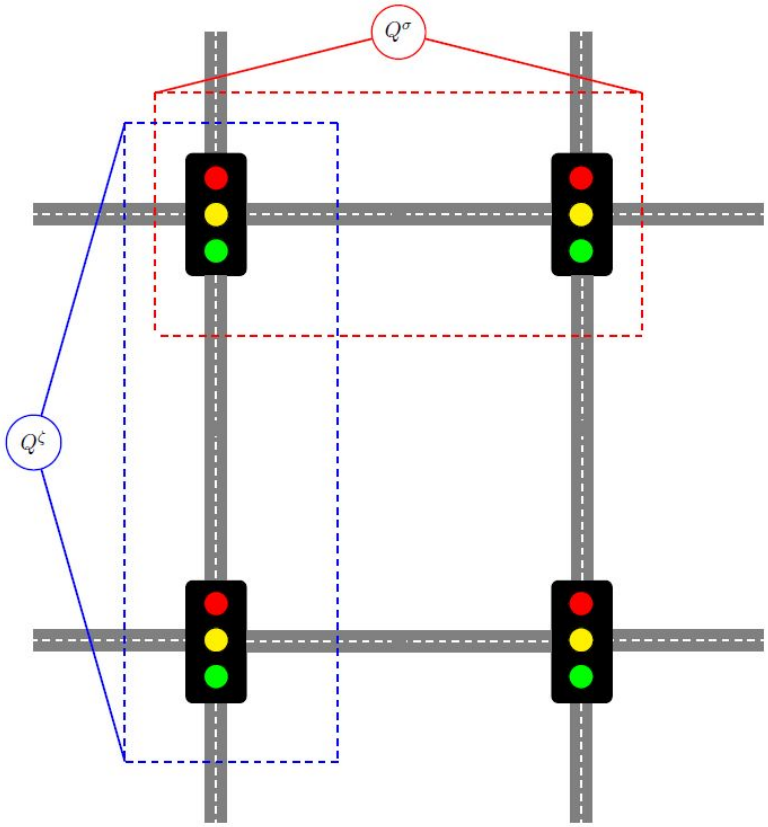

Figure 18: Transfer planning for traffic light control.

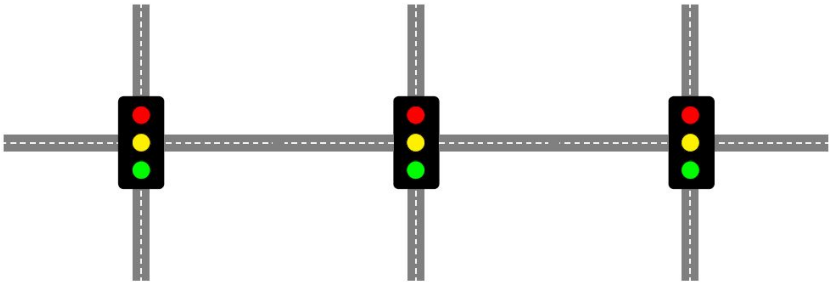

Figure 19: Three-agent scenario

As suggested by Abdulhai [2] multiagent situation permits additional state

information, since communication between agents extends the effective field of view of individual agents. In contrast to Abdulhai's work, Prasanth et. al [22] uses a feature based representation with function approximation in order to reduce the curse of dimensionality in multi agent systems. The Q-function is approximated as:

$$\mathcal{Q}(\mathrm{s},\mathrm{a}) \approx \theta^{\mathrm{T}} \sigma_{\mathrm{s,a}}$$

where, $\sigma_{s,a}$ is a d-dimensional feature (column) vector that corresponds to the state–action tuple (s, a), with $s \in S$, and $a \in A(s)$. The dimension d is significantly less compared to the cardinality of the set of feasible state–action tuples (s, a). Here, θ is a tunable parameter whose dimension is the same as in $\sigma_{s,a}$. The features are chosen based on the queue lengths(similar to the idea used by Thorpe [26]) and elapsed times of each signalled lane of the road network as shown by the equation below:

$$\sigma_{\mathrm{s_n,a_n}} = (\sigma_{\mathrm{q_1}}(\mathrm{n}), ..., \sigma_{\mathrm{q_N}}(\mathrm{n}), \sigma_{\mathrm{t_1}}(\mathrm{n}), ..., \sigma_{\mathrm{t_N}}(\mathrm{n}), \sigma_{\mathrm{a_1}}(\mathrm{n}), ..., \sigma_{\mathrm{a_m}}(\mathrm{n}))^{\mathrm{T}} \tag{46}$$

where, σ_{s_n,a_n} is feature vector for the state-action tuple (s, a) at time instance n, $q_i(n)$ is the queue length on lane i at time n, and $t_i(n)$ is the elapsed time for the red signal on lane i at time n. $\sigma_{a_1}(n), ..., \sigma_{a_m}(n)$ corresponds to the actions or sign configurations chosen at each of the m junctions. And finally, N is the total number of lanes (inclusive of all junctions). This approach as mentioned in the paper is computationally faster since the number of features is less compared to the total number of state action tuple. As an example, in case of a 3x3 intersection grid the number of state-action tuple is 10^{101} while the number of features was just 200.

5 Research Question

- How to decide the reward function in order to effectively co-ordinate traffic and thereby reduce traffic jams?
- How to scale from few agents involving few intersections to a suburb of a city and finally the entire city?
- Training the agent for different types of intersections which we come across in the real life scenario and extending it to bigger maps.
- How to exploit the sparsity of the state matrix? Since most of the entries are null, and therefore computation time is high.

Each of these research questions can be a thesis project in itself, therefore during the course of thesis work the main problem will be to tackle the extension of traffic light control to a bigger maps(comprising of multiple intersections) using the idea of Transfer Planning.

References

[1] USA A Medium Corporation. Understanding of convolutional neural network (cnn): Deep learning. `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-\eli5-way-3bd2b1164a53`.

[2] Baher Abdulhai, Rob Pringle, and Grigoris J Karakoulas. Reinforcement learning for true adaptive traffic signal control. *Journal of Transportation Engineering*, 129(3):278–285, 2003.

[3] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. Sumo–simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[4] Christopher M Bishop. *Pattern Recognition and Machine Learning, 2006*. Springer, 2006.

[5] Jeancarlo Josué Argüello Calvo and Ivana Dusparic. Heterogeneous multi-agent deep reinforcement learning for traffic lights control. In *The 26th Irish Conference on Artificial Intelligence and Cognitive Science*, pages 1–12, 2018.

[6] Noe Casas. Deep deterministic policy gradient for urban traffic light control. *arXiv preprint arXiv:1703.09035*, 2017.

[7] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[8] Juntao Gao, Yulong Shen, Jia Liu, Minoru Ito, and Norio Shiratori. Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network. *arXiv preprint arXiv:1705.02755*, 2017.

[9] Wade Genders and Saiedeh Razavi. Using a deep reinforcement learning agent for traffic signal control. *arXiv preprint arXiv:1611.01142*, 2016.

[10] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *ICML*, volume 2, pages 227–234. Citeseer, 2002.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Jelle R Kok and Nikos Vlassis. Using the max-plus algorithm for multi-agent decision making in coordination graphs. In *Robot Soccer World Cup*, pages 1–12. Springer, 2005.

[14] Daniel Krajzewicz, Georg Hertkorn, Christian Feld, and Peter Wagner. Sumo (simulation of urban mobility); an open-source traffic simulation. pages 183–187, 01 2002.

[15] Li Li, Yisheng Lv, and Fei-Yue Wang. Traffic signal timing via deep reinforcement learning. *IEEE/CAA Journal of Automatica Sinica*, 3(3):247–254, 2016.

[16] Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han. Deep reinforcement learning for traffic light control in vehicular networks. *arXiv preprint arXiv:1803.11115*, 2018.

[17] Yilun Lin, Xingyuan Dai, Li Li, and Fei-Yue Wang. An efficient deep reinforcement learning model for urban traffic control. *arXiv preprint arXiv:1808.01876*, 2018.

[18] Ying Liu, Lei Liu, and Wei-Peng Chen. Intelligent traffic light control using distributed multi-agent q learning. *arXiv preprint arXiv:1711.10941*, 2017.

[19] Frans A Oliehoek, Shimon Whiteson, and Matthijs TJ Spaan. Approximate solutions for factored dec-pomdps with many agents. In *Proceedings of the 2013 International Conference on Autonomous agents and Multi-Agent Systems*, pages 563–570. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[20] GMA News Online. Stress, pollution, fatigue: How traffic jams affect your health. `https://www.gmanetwork.com/news/lifestyle/healthandwellness/536203/\stress-pollution-fatigue-how-traffic-jams-affect-your-health/story/`.

[21] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[22] LA Prashanth and Shalabh Bhatnagar. Reinforcement learning with function approximation for traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):412–421, 2011.

[23] Tobias Rijken. *DeepLight: Deep reinforcement learning for signalised traffic control*. PhD thesis, Master's Thesis. University College London, 2015.

[24] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[25] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[26] Thomas L Thorpe and Charles W Anderson. Tra c light control using sarsa with three state representations. Technical report, Citeseer, 1996.

[27] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[28] Elise van der Pol. Deep reinforcement learning for coordination in traffic light control. 2016.

[29] Elise Van der Pol and Frans A Oliehoek. Coordinated deep reinforcement learners for traffic light control. *Proceedings of Learning, Inference and Control of Multi-Agent Systems (at NIPS 2016)*, 2016.

[30] MA Wiering. Multi-agent reinforcement learning for traffic light control. In *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*, pages 1151–1158, 2000.

[31] Wikipedia.org. Ann. `https://commons.wikimedia.org/wiki/File:Colored_neural_network.svg`.