Bachelor thesis

Artificial Intelligence

Radboud University

Using Deep Probabilistic Reinforcement Learning to Improve Traffic Flow with Partial Vehicle Detection for Intelligent Traffic Signal Control

Author:
Noah van der Vleuten
s1018323

First supervisor/assessor:
dr. L. Ambrogioni
Donders Centre for Cognition
luca.ambrogioni@donders.ru.nl

Second assessor:
dr. M. Hinne
Donders Centre for Cognition
m.hinne@donders.ru.nl

June 18, 2021

Abstract

With the constant rise of vehicles on the streets, traffic flow needs to be as optimal as possible. Improving traffic flow requires intelligent traffic signal control (the field of controlling traffic lights). Mismanagement of traffic lights can cause traffic congestion. Traffic congestion can lead to accidents, wasted productivity, and, most importantly, is terrible for the environment (with fossil fuel cars). Several deep reinforcement learning methods have been suggested to improve TSC, but they all assume perfectly observed data. However, the assumption of always being able to observe how many cars are on each lane is not realistic. Real-world observations are inherently noisy, sensors can fail, and some sensors might be left out as cost-saving measures. This work, therefore, focuses on partial vehicle detection in traffic signal control. Specifically trying to improve average travel time given only partial observations of cars at intersections. A variational autoencoder (VAE) has been developed using graph neural networks (GNN's) for the encoder and decoder network to solve this problem. This VAE tries to reconstruct the real-world situation and the uncertainty of its prediction given lane counts at a mix of observed and unobserved intersections. A state-of-the-art reinforcement learning method called CoLight is then used to test this VAE in the Cityflow simulator to see how well it performs. Results show that the CoLight model performs better when using the reconstructed modes from the VAE (515) compared to no information at all (1146) at the unobserved intersections. The CoLight model with the additional uncertainty measure performs marginally better (490) than the model that only gets the reconstruction. However, the MaxPressure + FixedTime baseline still outperforms all models using the average travel time metric (402).

Contents

Chapter 1

Introduction

1.1 Problem description

Traffic Signal Control (TSC) is the field that focuses on how to intelligently coordinate the signals of traffic lights in such a way that it optimizes a particular metric. Usually, this metric is traffic flow, where it is desired for all cars to get to their destination as fast as possible by keeping the traffic flowing. Hence trying to maximize the traffic flow. Another popular method is average travel time, which is the average time it takes for all cars to get from their origin to their destination.

Current state-of-the-art (SOTA) methods for improving traffic flow use meta-learning (MetaLight [1]/GeneraLight [2]) and Graph Attention Networks (CoLight) [3]. However, these state-of-the-art models assume they have a perfect observation of the number of cars waiting in front of the traffic lights. Of course, this is not a very realistic scenario. Some countries do use cameras or loop detectors, but even those will never give 100% accuracy of the observations and are not always economically viable [4].

Due to this economic viability, a model that can be trained on fully observed data and then transferred to a place with less money for sensors could be hugely beneficial. This model will have learned from traffic at a different place. Still, the way traffic behaves might be similar enough across different places, given a comparable (but possibly smaller) intersection topology.

It is also possible that sensors are perturbed in real-world use. Meaning they could stop giving information to the traffic signal controller due to a for example faulty sensors. This could have devastating effects for the current state of the art models, as they were not trained on the possibility of deterministic measurements disappearing. This could be very detrimental to the performance of such networks, as they are not robust enough. To back

up this claim the robustness of one of these state-of-the-art models will be shown in the results section.

1.2 Research goal

To improve these models in the real world, it could be beneficial to take into account the fact that the real world is not always fully observed. Therefore the following research question will attempt to be answered in this thesis:

> How can traffic flow, in terms of average travel time, be improved given only partial observations of cars at intersections?

It is hypothesized that to solve this problem and to leverage the power of reinforcement learning methods, a reconstruction of the real-world situation will have to be made. The model is not likely to reconstruct the real-world data perfectly if data of entire intersections are missing. To combat this, the reconstruction model will have to output some measure of uncertainty about its predictions such that a reinforcement learner can reason about this uncertainty. The belief is that due to this additional layer of information, the reinforcement learner will perform better than with a pure reconstruction. The expectation is that a good reinforcement learner will learn to mistrust the input of the reconstruction model and find some clever way to coordinate traffic lights thanks to the mistrust.

1.3 Societal Relevance

A deep probabilistic model that could reason about the number of cars waiting at a certain intersection, can explain its uncertainty, and use these predictions to improve traffic flow would be of great benefit to society. It could help bring these deterministic TSC models (those that rely on full observations) to the real world and possibly alleviate costs by not having to place cameras or long induction-loop traffic sensors everywhere. Using the correct neural network architectures, it can be possible to transfer learn to new scenarios without needing new measurements of the place of deployment.

Additionally, it could decrease real-world travel times and have a smaller impact on the environment due to fossil-fuel cars not wasting fuel while waiting. The model could perhaps explain travel dynamics that could be useful for Traffic Signal Control research. This could, for example, be achieved by using a model-based approach in which the model could be interpretable. However, in this research, a model-free approach will be described due to time restrictions.

1.4 Solution to the problem

Together with the thesis group, a solution was developed using a variational autoencoder with graph neural networks and using CoLight, one of the SOTA models mentioned before, for the reinforcement learning/traffic signal controlling part. For more information on these topics, please refer to chapter 2.

The variational autoencoder takes the features of the intersections (in this case, the lane counts, the number of cars at each lane) as input. Then due to the graph neural networks, message passing occurs, which means the intersections communicate with their neighbors through the encoder part of the VAE. Afterward, these features get mapped into the latent space of the VAE by parameterizing a (multivariate) normal distribution, which means each feature gets assigned a mean μ and a standard deviation σ. Following that, the decoder part is again a graph neural network where message passing between this and the adjacent intersections can again take place. Finally, the decoded features get mapped to a categorical distribution. This allows for the reconstruction of the input, with uncertainty about the observations. This will be further explained in chapter 4.

Once the VAE has reconstructed the input and made a distribution of it, the mode and entropy of the categorical distribution of each feature of masked/missing intersection are given to the reinforcement learner, in this case, CoLight. The intersections that are fully observed are given directly to the reinforcement learner, with the observation being the mode and the entropy being 0. Additionally, for all feature observations, a one-hot encoded label is given that indicates if the observation is fully observed (was given from the data directly) or was masked (given in reconstructed form using the VAE). The reinforcement learner should then learn to deal with the uncertainty and potentially perform better than the baseline (inserting 0's and letting the network figure out how to deal with it).

The intuition behind this solution is that these road observations (the would-be real-world data) are inherently uncertain. Hence it makes sense to use probabilistic deep learning. The intuition behind the VAE reconstruction is that a probabilistic latent space is used, which can be sampled from, allowing for a probabilistic approach. In turn, the decoder infers the posterior distribution of an observation given these latent variables [5]. Finally, these posteriors can be used to model one categorical distribution as a final layer for each of the features. A distribution at the end of the VAE was chosen to give uncertainty information to the reinforcement learner. The choice for a categorical distribution was taken because it worked better than a Gaussian and log-normal distribution. Graph neural networks are

used because this problem is a graph problem. Intersections have neighbors and are connected in a grid-like manner; hence, exploiting graph neural networks' message passing abilities can aid with the reconstruction. This is because observed intersections are likely to carry information about their unobserved neighbor(s).

1.5 Comparison to related work

Since the state-of-the-art models assume full observation of the number of cars (which is not always realistic), research into applying reinforcement learning methods to partially observable cars can also prove a lot of value to the scientific community. From my literature review, applying deep probabilistic learning methods to partial vehicle detection is novel. More on this can be found in chapter 3.

1.6 Overview

Firstly, in this thesis, the preliminaries (required background knowledge to read this thesis) will be explained. Then, in the related work section, a literature review of the required components and the TSC field will be executed. In the research section, the entire research will be explained, from start to finish, including all experimental results. Next, in the discussion section, the results will be interpreted and discussed, further research will be proposed as well. Finally, in the conclusions section, the thesis will get its concluding remarks.

Chapter 2

Preliminaries

2.1 Cityflow

Cityflow is a multi-agent reinforcement learning environment for large-scale city traffic scenarios [6]. This simulator is widely used in the research of this project and is therefore important to explain. The reason Cityflow was chosen over a contemporary like SUMO is that Cityflow allows for multi-threading [6]. The state-of-the-art models also train using Cityflow (most likely because of its speed benefit), so this was the most logical choice. Like SUMO, Cityflow uses a Python interface in which data can be observed using API calls, and traffic lights can be controlled.

A map that is widely used in this research is the `Manhattan 16x3` road network. Using the frontend renderer provided by Cityflow the road net, the simulation can be replayed using a replay file. This is very beneficial as errors in the network can become easily visualized thanks to this tool.

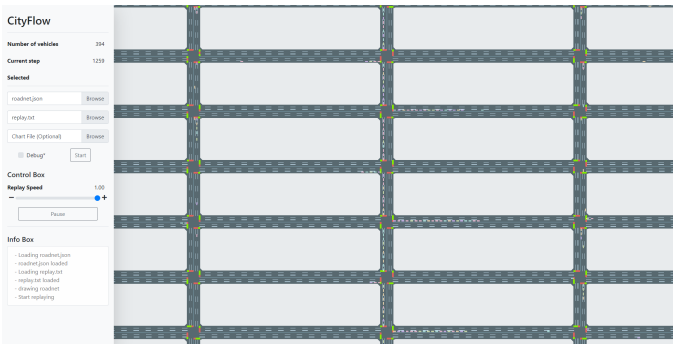

Figure 2.1: The map `Manhattan 16x3` rendered using the Cityflow frontend HTML page. The traffic is entirely congested thanks to poor management of the phases.

The Cityflow files used in this thesis are mostly taken from https://github.com/traffic-signal-control/sample-code/tree/master/data.
This repo has a combination of real-world and synthetic road networks and data which are heavily used in the field of TSC.

Cityflow relies on several files to get the simulation started. Firstly a config file that is used to initialize the engine in Python is required. This `configFile` lists several things about the configuration of the simulation such as the `roadnetFile`, `flowFile`, whether to use reinforcement learning (controlled traffic lights by the python script) and where to store the log and replay files.

For example one of the config files used in this project:

```
{
  "interval": 1.0,
  "seed": 0,
  "dir": "./",
  "roadnetFile": "sample-code/data/manhattan_16x3/
     roadnet_16_3.json",
  "flowFile": "sample-code/data/manhattan_16x3/
     anon_16_3_newyork_real.json",
  "rlTrafficLight": true,
  "saveReplay": true,
  "roadnetLogFile": "logs/roadnet.json",
  "replayLogFile": "logs/replay.txt"
}
```

The `roadnetFile` contains the network description of the road network. This file describes the connections of the intersections, the lanes, and to which intersection they belong. It also describes the `lightPhases` of the intersections. A phase is a combination of traffic lights that are allowed to be green at a single timestep. The `lightPhases` are expressed in the time they will be turned on (green) if `rlTrafficLight` is turned off. The index in the `lightphases` represents the index of the light that will be turned green.

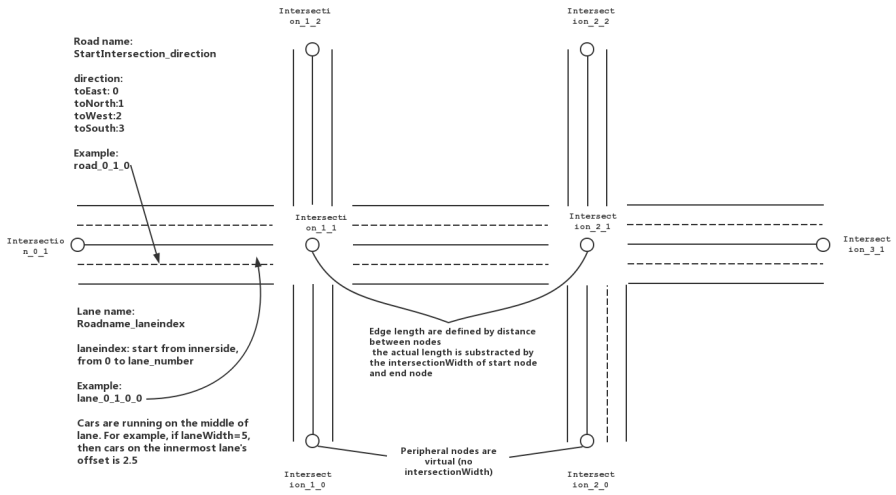

Figure 2.2: Information of how the road layout is represented internally. Taken from the Cityflow documentation listed below.

The `flowFile` contains information about the cars, their length, width, acceleration information, and when they should be spawned, amongst other factors.

For more information, please refer to the Cityflow documentation at https://cityflow.readthedocs.io/en/latest/introduction.html.

A way to generate random flow files was also developed for this thesis. This was done such that there would be enough training and testing data. However, the use of this is limited as random data will most likely not have the same patterns as the real-world data provided by the aforementioned repo. Therefore in the training and testing phase, a good balance has to be struck.

2.2 Average travel time

The average travel time is used as the evaluation metric in this thesis. It is the most frequently used measure to judge the performance of methods in the traffic signal control field [2]. Conveniently it is included in the Cityflow API, meaning the user can ask for the average travel time at any point during the simulation.

The average travel time is calculated by averaging the time it takes for all vehicles to get from their origin to the their destination (in seconds) [2].

2.3 Variational Autoencoders

This section is based on Diederik Kingma and Max Welling's paper: "An Introduction to Variational Autoencoders" [7].

2.3.1 Overview

Variational autoencoders provide a principled framework for learning deep latent-variable models and corresponding inference models. Just like a normal autoencoder, the VAE usually has two models. The models are an encoder model (recognition model) and a decoder model (generative model). The models support each other; however, they are independently parameterized.

The recognition model delivers to the generative model an approximation to its posterior over latent random variables. In contrast, the generative model tries to learn meaningful representations of the data given the samples from the latent space. This differs from a normal autoencoder, as these do not sample from the latent space.

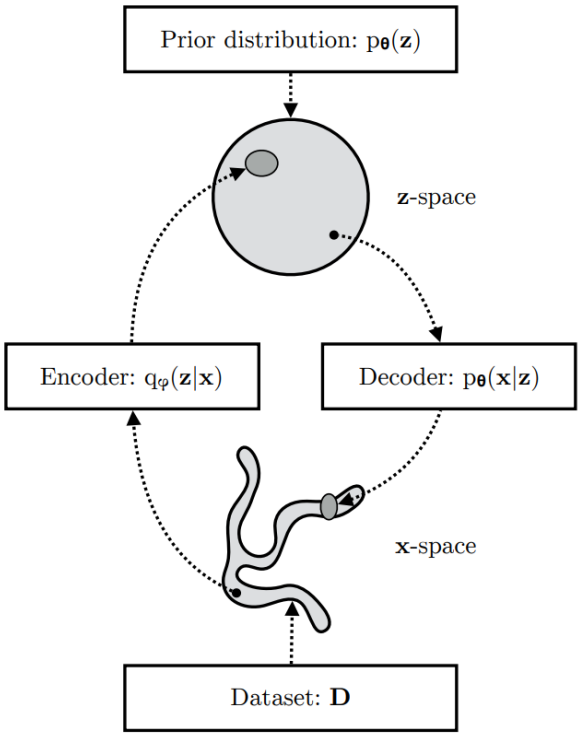

Figure 2.3: An overview of the VAE architecture with relevant mathematical notation, which will be explained in the next sections.

2.3.2 Encoder

Traditionally deep latent-variable models' (DLVMs) posterior inference is intractable. To turn this into a tractable problem a parametric inference model $q_\phi(\mathbf{z} \mid \mathbf{x})$ is introduced. This model is also called the (stochastic) encoder or recognition model.

Symbol ϕ notates the parameters of the encoder model, which are also called the variational parameters. These variational parameters ϕ are optimized in such a way that:

$$q_\phi(\mathbf{z} \mid \mathbf{x}) \approx p_\theta(\mathbf{z} \mid \mathbf{x}) \tag{2.1}$$

This means that the stochastic encoder $q_\phi(\mathbf{z} \mid \mathbf{x})$ approximates the true but intractable posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$ of the generative model. This approximation to the posterior will help us optimize the marginal likelihood, as will be seen in section 2.3.4.

In deep learning research the distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$ is usually parameterized using deep neural networks. This means that ϕ would include the weights and biases of the neural network. This means that the recognition model distribution could be generated as such:

$$(\boldsymbol{\mu}, \boldsymbol{\sigma}) = \text{EncoderNeuralNetwork}_\phi(\mathbf{x}) \tag{2.2}$$

$$q_\phi(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma})) \tag{2.3}$$

It is important to note that the encoder neural network provides parameters for a distribution (in this case, a multivariate normal distribution), which is equal to the stochastic encoder $q_\phi(\mathbf{z} \mid \mathbf{x})$).

Usually, this encoder neural network is re-used to perform posterior inference over every data point in the dataset. Conversely, the variational parameters are not shared in traditional variational inference methods, meaning they are separately and iteratively optimized per data point. Hence VAE's usually need fewer parameters. This sharing of variational parameters across datapoints is also called amortized variational inference.

2.3.3 Decoder

The decoder then takes the sampled $\mathbf{z}$ variables and generates the following distribution: $p_\theta(\mathbf{x} \mid \mathbf{z})$, the decoder is also called the stochastic decoder. The generative model learns a joint distribution, which is usually written as:

$$p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{z}) p_\theta(\mathbf{x} \mid \mathbf{z}) \tag{2.4}$$

With a prior distribution over the latent space $p_\theta(\mathbf{z})$.

2.3.4 Evidence Lower Bound (ELBO)

The optimization objective of a variational autoencoder, just like in other variational methods, is the evidence lower bound (ELBO).

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x})\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right] \\
&= \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right]\right]}_{\substack{=\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) \\ \text{(ELBO)}}} + \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right]}_{=D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))}
\end{aligned}$$

Figure 2.4: The derivation of the ELBO and KL divergence from $\log p_\theta(\mathbf{x})$.

The second term in figure 2.4 is the Kullback-Leibler (KL) divergence. The KL divergence is always non-negative, and zero iff $q_\phi(\mathbf{z} \mid \mathbf{x})$ equals the true posterior distribution. Hence this term describes the distance between the approximated and true posterior distribution. It also determines the gap between the ELBO and the marginal likelihood $\log p_\theta(\mathbf{x})$. The closer the posterior approximation is to the true posterior distribution, the smaller the gap.

The ELBO can be rewritten as:

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z} \mid \mathbf{x})] \tag{2.5}$$

Because the KL divergence is always non-negative, the ELBO is a lower bound on the log-likelihood of the data.

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \log p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z} \mid \mathbf{x})||p_\theta(\mathbf{z} \mid \mathbf{x})) \leq \log p(\mathbf{x}) \tag{2.6}$$

From equation 2.6 it can be seen that maximizing the ELBO w.r.t. the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ will optimize the two most important things that are important.

1. Maximizing the ELBO will try to maximize the marginal likelihood $p_\theta(\mathbf{x})$. This means the generative model will improve because its parameters will model the probability distribution of the data more accurately.

2. It will minimize the KL divergence (distance) of the approximate posterior and the true posterior, so the approximate posterior improves.

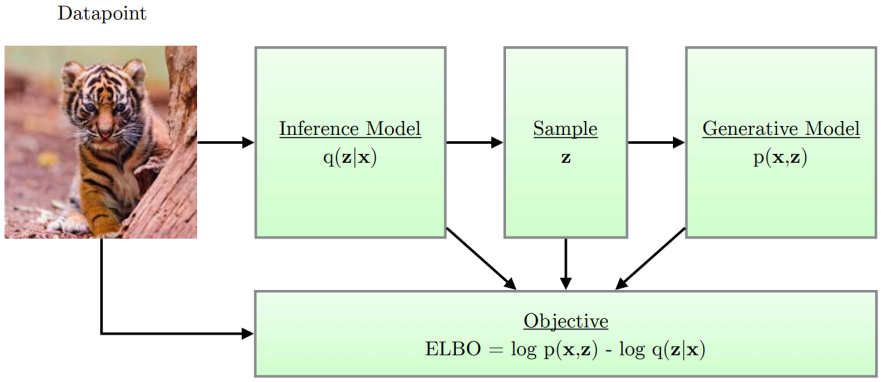

Figure 2.5: Schematic of the computational flow in a VAE to optimize the objective.

An advantageous property of the ELBO is that it can be optimized w.r.t. the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ using stochastic gradient descent (SGD). This is important because it allows for training the networks until convergence using stochastic gradient-based optimization.

2.3.5 Reparameterization Trick

To allow for backpropagation in a VAE the sampling of the latent space variable $\mathbf{z} \sim q_\theta(\mathbf{z} \mid \mathbf{x})$ will have to be represented differently. If z is sampled directly, it is impossible to backpropagate, as backpropagating through a random node is impossible.

$\mathbf{z}$ will be defined as a differentiable (and invertible) transformation of another random variable $\boldsymbol{\epsilon}$, given $\mathbf{x}$ and $\boldsymbol{\phi}$. This function will be called g.

$$\mathbf{z} = g(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x}) \tag{2.7}$$

This function g allows $\mathbf{z}$ to be exactly defined, and therefore makes $\mathbf{z}$ a deterministic node. This allows for backpropagation through the node. This is called the reparametrization trick.

For example, suppose a multivariate normal distribution is used for $\mathbf{z}$ and for sampling a normal distribution $\boldsymbol{\epsilon}$, then the reparameterization trick can be applied. After reparameterization we can write the model as follows:

$$\boldsymbol{\epsilon} \sim \mathcal{N}(0, I) \tag{2.8}$$

$$(\boldsymbol{\mu}, \boldsymbol{\sigma}) = \text{EncoderNeuralNetwork}_{\boldsymbol{\phi}}(\mathbf{x}) \tag{2.9}$$

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \tag{2.10}$$

With $\odot$ being element-wise multiplication. Now $\mathbf{z}$ is defined instead of sampled, which means backpropagation is now possible, as can be seen in figure 2.6.

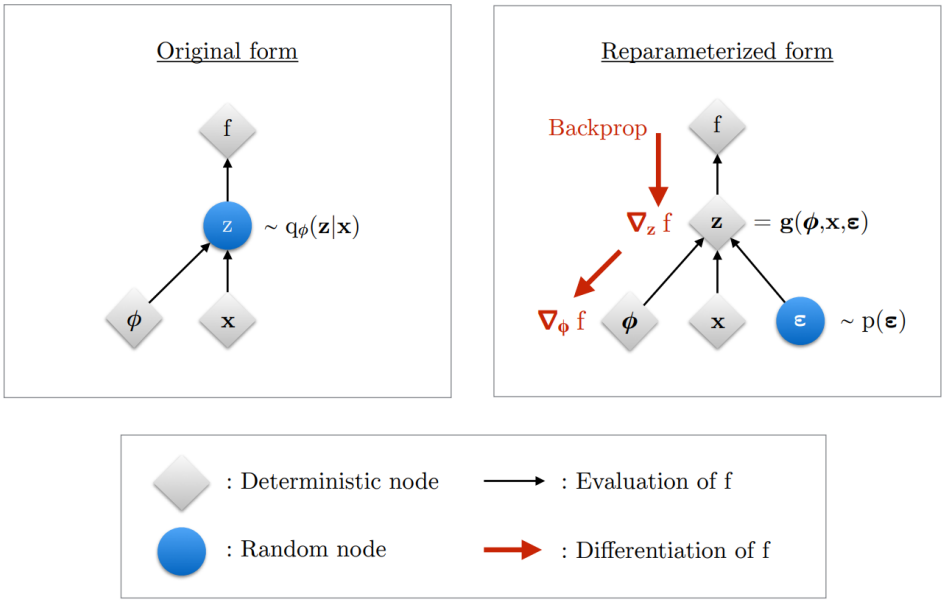

Figure 2.6: Illustration of the reparameterization trick. It is needed to compute the gradients $\nabla_\phi f$ to optimize the objective. Due to the random node on the left, this is impossible, as it is not possible to backpropagate through sampling. Hence on the right, the "randomness" variable $\boldsymbol{\epsilon}$ is sampled outside of the backpropagation signal route. This allows for backpropagation "through $\mathbf{z}$".

2.3.6 Decoder probability distribution layer

For the final layer of the decoder, another probability distribution can be used. This means that the decoder has to predict the parameters for this specified distribution. Taking a distribution as a final layer helps preserve the uncertainty of the reconstruction, which is required for this thesis.

2.4 Graph Neural Networks

Graph Neural Networks (GNN's) come in many shapes and sizes. This section will give a basic overview of GNN's as they are used in the variational autoencoder model of this thesis.

2.4.1 Overview

This section is largely based on the works of Zhou. et al., please check out the paper named "Graph neural networks: A review of methods and applications" for more information [8].

Many problems have a graph structure underlying them. These graph structures contain rich relation information among their elements (nodes). These problems include: modeling physics systems, learning molecular fingerprints, predicting protein interface, and classifying diseases. If the data can be represented as a graph or is a graph (potentially with edge weights), Graph Neural Networks can be used to learn from them. Graph Neural Networks are neural models that capture the dependence of graphs via message passing between the nodes of graphs. This allows them to learn many complex structures in the data by cleverly exploiting the graph information.

2.4.2 Explanation

The graph neural network used in this thesis is defined in the paper of Morris et al. [9], and slightly rephrased in the PyTorch Geometric documentation, named `GraphConv` [10]. The PyTorch Geometric notation will be used for consistency's sake.

For each node in the graph i the new value of the feature $\mathbf{x}_i$ will be defined as:

$$\mathbf{x}'_i = \mathbf{\Theta}_1 \mathbf{x}_i + \mathbf{\Theta}_2 \sum_{j \in \mathcal{N}(i)} e_{j,i} \cdot \mathbf{x}_j \tag{2.11}$$

where $e_{j,i}$ denotes the edge weight from source node j to target node i (the default is 1). $\mathbf{\Theta}_1$ and $\mathbf{\Theta}_2$ are weight matrices. $\mathcal{N}(i)$ is the set containing every neighbor of node i.

Hence, as can be seen from the formula, the previous state of the feature of i is taken and weighed with $\mathbf{\Theta}_1$. Then for the "aggregating" part, also called message passing, all neighbors are multiplied with the edge weights of j to i and with the feature of j, defined as $\mathbf{x}_j$. Then after the aggregation over all neighbors has taken place, this value is weighed by the matrix $\mathbf{\Theta}_2$.

As can be seen, the aggregation scheme in this example is a simple sum, this is the default in the `GraphConv` layer. However, it is also possible to specify a different aggregation scheme, the choices are: `"add", "mean", "max"`. Scheme `mean` averages the weighted neighbor features and `max` takes the maximum weighted neighbor feature, the weighting in this case is done by the edge weights. These aggregation functions (Agg) can be used to generalize the formula as follows:

$$\mathbf{x}'_i = \mathbf{\Theta}_1 \mathbf{x}_i + \mathbf{\Theta}_2 \ \text{Agg}(\{\forall j \in \mathcal{N}(i), e_{j,i} \cdot \mathbf{x}_j\}) \tag{2.12}$$

Since these operations are differentiable, it is possible to backpropagate through the GNN's and learn the weight matrices $\mathbf{\Theta}_1$ and $\mathbf{\Theta}_2$.

2.5 (Deep) Q-Learning

This section is largely based on the monumental work of DeepMind on Deep Reinforcement Learning by Mnih et al. [11]. Q-Learning is covered because a traffic signal control reinforcement learning method called CoLight, used in this thesis, uses (deep) Q-Learning.

2.5.1 Problem description

Reinforcement learning is the field that focuses on controlling agents that interact with a certain environment in such a way that they solve a particular task. Usually, this is achieved by having the agent behave "intelligently". To do this, several reinforcement learning algorithms have been proposed, one of which is Q-Learning.

Reinforcement learning problems usually have high-dimensional sensory input. The inputs can span multiple modalities such as vision and speech. In the case of DeepMind, they applied deep learning to the raw pixel values of Atari 2600 games to train their agent.

Before the advent of neural networks, most successful RL applications applied to these domains have relied on hand-crafted features combined with linear value functions or policy representations. These methods are heavily reliant on the quality of these hand-crafted features. To solve this, (deep) neural networks can be applied to Q-Learning to let the agent extract the features it deems most relevant independently.

In this section, classical Q-Learning will be explained. After that, deep Q-Learning will be explained, which uses deep neural networks to solve the problems described above.

2.5.2 Classical Q-Learning

In reinforcement learning, there usually is an environment $\mathcal{E}$, which in the case of DeepMind is an Atari emulator. At each timestep t the agent interacts with this environment by selecting a certain action a_t from a set of legal actions: $\mathcal{A} = \{1, \ldots, K\}$. From these interactions, the agent gets a new observation x_t and the reward r_t that this action produced.

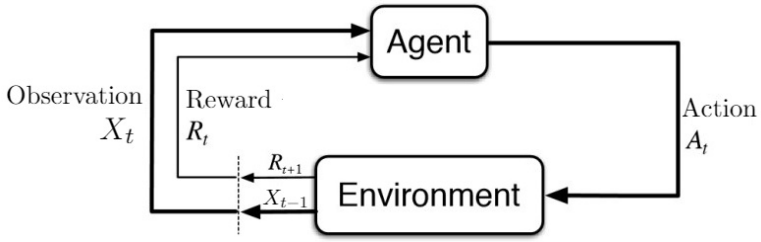

Figure 2.7: Classical visual representation of reinforcement learning problems. Adapted from (Bhatt, 2018) [12].

If the agent is only shown x_t, it will be impossible to understand the current situation from the current screen fully. It will not be able to take into account the context of how it found itself in this state. Therefore sequences of actions and observations are considered, $s_t = x_1, a_1, x_2, \ldots, a_{t-1}, x_t$ and game strategies will be learned that depend on these sequences.

The agent's goal then is to interact with the emulator by selecting the actions in such a way that it maximizes future reward. The future rewards are discounted by a specified factor γ every time-step. The future discounted return at time t is then defined as follows:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{2.13}$$

where T is the time-step at which the game terminates.

Then the optimal action-value function $Q^*(s, a)$ is defined as the maximum expected return achieved by following any of the strategies, after observing some sequence s and taking some action a. The optimal action-value this function produces is also called the Q-value.

$$Q^*(s, a) = \max_{\pi} \mathbb{E}[R_t \mid s_t = s, a_t = a, \pi] \tag{2.14}$$

where π is a policy mapping sequences to actions.

This optimal action-value functions obeys an important identity known as the Bellman equation. The intuition behind this can be described as follows: if the optimal value $Q^*(s', a')$ of the sequence s' at the next time step is known for all actions possible, then the optimal strategy is to select the action a' which maximizes the expected value of $r + \gamma Q^*(s', a')$, which gives the following formula:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a'} Q^*(s', a') \mid s, a] \tag{2.15}$$

For most reinforcement learning algorithms the goal is to estimate the action-value function, using the Bellman equation as an iterative update, $Q_{i+1}(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a')]$. These value iteration algorithms converge to the optimal action-value function as i tends to infinity. In practice, this basic approach is intractable, as the action-value function is estimated separately for each sequence, without any generalization. Instead, a function approximator to estimate the action-value function is oftentimes used, $Q(s, a; \theta) \approx Q^*(s, a)$.

These approximators used to be linear functions. Nowadays non-linear approximators are used such as neural networks. A neural network that is parameterized by the weights θ is referred to as a Q-network. A Q-network can be trained by minimizing a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration i:

$$L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)}[(\mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \mid s, a] - Q(s, a; \theta_i))^2] \tag{2.16}$$

where $\rho(s, a)$ is a probability distribution over sequences s and actions a that is named the behavior distribution. The parameters of the previous iteration θ_{i-1} are held fixed when optimizing the loss function of the current iteration $L_i(\theta_i)$.

This loss function can be differentiated with respect to the weights, and in this way, stochastic gradient descent can be used to optimize the Q network iteratively. Computing the full expectations of the gradients is usually computationally intractable.

The algorithm is model-free as it solves the reinforcement learning problems using samples from the environment $\mathcal{E}$ without constructing an estimate of the environment itself. The algorithm is also off-policy, as it learns about the greedy strategy $a = \max_a Q(s, a; \theta)$ without using a separate policy π. In practice, a parameter ϵ is usually used during the training phase, also called the exploration parameter, which decides if a random action should be taken or not with a probability of $1 - \epsilon$.

2.5.3 Deep Q-Learning

As mentioned in the problem description, section 2.5.1, many reinforcement learning problems have a very high dimensional input of data.

Deep Q-Learning sets itself apart from classical Q-learning by using a deeper non-linear approximator. This approximator, for example, a deep convolutional neural network, helps with processing this high-dimensional data without relying on hand-crafted features.

Deep learning also uses some tricks that stabilize learning the neural network. Usually, a technique called experience replay is used. The agent's experiences are stored at each time step in a data-set, pooled over many episodes they get stored in what's called a replay memory. While updating the Q-learning network, samples of the (previously gained) experiences are drawn at random (experience replay). By using experience replay, the behavior distribution is averaged over many of its previous states. This smoothes out learning and mitigates catastrophic forgetting that can result from large divergences from parameters.

Deep Q-Learning managed to get impressive results compared to other reinforcement learning methods at the time.

	B. Rider	**Breakout**	**Enduro**	**Pong**	**Q*bert**	**Seaquest**	**S. Invaders**
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Figure 2.8: Results of average total rewards for various learning methods. In the top table an ϵ of 0.05 is used. The lower table shows single best performing episodes for both HNeat versions and DQN.

Chapter 3

Related Work

As mentioned in the introduction, all surveyed state-of-the-art deep learning methods rely on full observations. Meaning that the amount of cars waiting at each lane is given as-is to the network. In this section, a few of these traffic signal control models will be described. Additionally, the few papers that concern partial vehicle detection will be covered as well.

3.1 (Reinforcement learning) approaches for TSC

3.1.1 FixedTime

A simple baseline for most TSC papers is the FixedTime "algorithm". This method switches phases by a preset time regardless of the changes in traffic conditions [13]. A similar method is used in the Cityflow simulator when `rlTrafficLight` is turned off. Hence it serves as an inherent baseline when Cityflow is used without reinforcement learning control turned on.

It often seems to perform relatively well compared to more sophisticated methods, which will become apparent in the results of the CoLight section in chapter 3.1.3.

3.1.2 MaxPressure

Another simple (yet very effective) baseline that is often used is the MaxPressure algorithm. This method works by calculating the pressure of a phase. Formally, the pressure of a phase is defined as the number of incoming cars, minus the number of outgoing cars, assuming the phase in question would be active [14]. MaxPressure is proven to maximize the throughput of the whole road network [15]. Since this method is effective yet easy to implement, it is also taken as one of the baselines for the result section of this thesis.

For a visual explanation, please refer to figure 3.1.

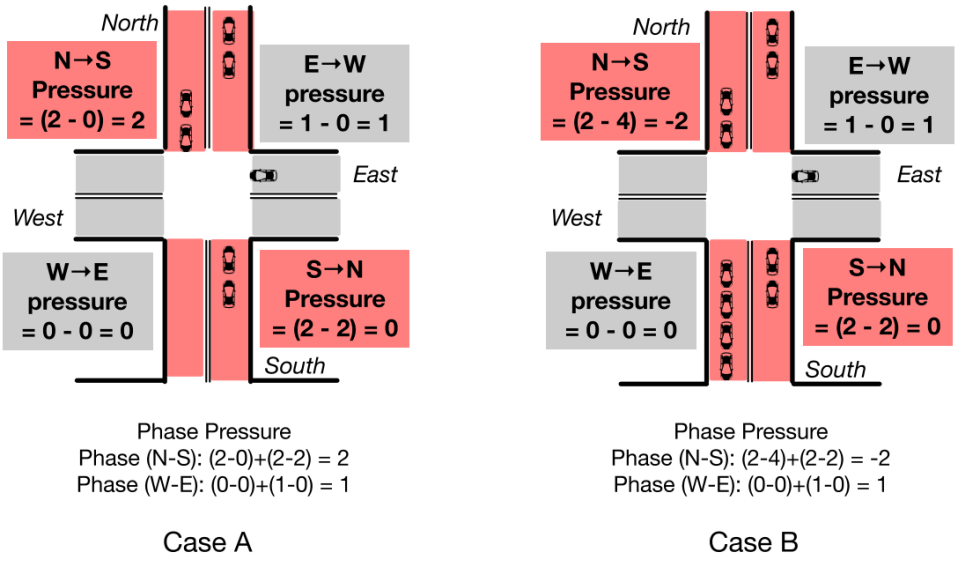

Figure 3.1: Illustration of the MaxPressure algorithm for two cases. In both cases, there are four movement signals: North→South, South→North, East→West and West→East and there are two phases: Phase(N - S) which enables green signals in the North→South and South→North direction, and Phase(N - S) which enables green signals in the East→West and West→East direction. In Case A, MaxPressure selects Phase(N - S) since the pressure of Phase(N - S) is higher than Phase(W - E); in Case B, Max-pressure selects Phase(W - E). Taken from [14], but fixed the (N - S) error in the top left.

The MaxPressure control algorithm is formalized in the following way [14]:

Algorithm 1: MaxPressure Control

Input: Time phase has been active t, minimum phase duration time t_{min}.

forall *timesteps* **do**
  $t = t + 1$;
  if $t \geq t_{min}$ **then**
    Calculate the pressure P_i for each phase i;
    Set the next phase as $\text{argmax}_i\{P_i\}$;
    $t = 0$;
  end
end

3.1.3 CoLight

For more information about CoLight, please refer to the CoLight paper this section was based on [3].

Overview

CoLight is a deep reinforcement learning method that uses Graph Attention Networks (GAT) to facilitate intersection communication. This allows them to capture the temporal and spatial influences of neighboring intersections to the target intersection. Thanks to the attention mechanism, the adjacent intersections are attended to more or less depending on the perceived importance. These spatial and temporal influences are needed for dynamic environments because conventional transportation approaches implement cooperation by pre-calculating the offsets between two intersections. Such pre-calculated offsets are not suitable for dynamic traffic environments.

Colight is used to compare many of the other TSC papers that followed due to its state-of-the-art performance.

Model architecture

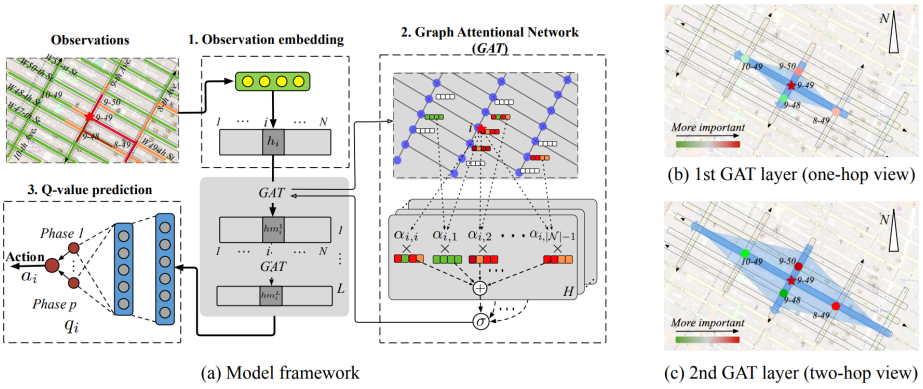

Figure 3.2: Left: Framework of the proposed CoLight model. Right: variation of cooperation scope (light blue shadow, from one-hop to two-hop) and attention distribution (colored points, the redder, the more important) of the target intersection. Image is taken from the Colight paper [3].

The model first constructs a 1D embedding of the fully observed lane counts per intersection and the phase the intersection is currently in (the configuration of green and red lights, represented as a one-hot vector). Then a GAT is applied to this embedding, which is then piped through another GAT (producing the two-hop-view on the right of the picture (c)). The amount

of GATs is arbitrary and can be changed using the L parameter. This result is then piped through a deep Q-learning network for a Q-value prediction of the phases. These Q-values are evaluated, and finally, an action (which phase should be chosen next) is given to the environment to execute. For more information on Q-learning, please refer to section 2.5.

From figure 3.1.3 (b) on the right-hand side, an example is depicted of the GAT attention layers. In the first GAT layer, the "communication"-depth is 1 neighbor. Also, thanks to the attention mechanism, the perceived importance of the neighbor's features can be visualized (the attention distribution). This can be seen in the legend from green (less important) to red (very important). The blue lines depict how deep the communication goes.

Below that (c), the second GAT layer is visualized. Here the neighbor communication depth is 2. This is because the hidden states of adjacent neighbors from the first GAT layer carry their respective neighborhood messages. This means that in the second GAT layer, the cooperation scope of Inter 9-49 expands significantly to 8 intersections.

Results

Model	$Grid_{6\times6}$-Uni	$Grid_{6\times6}$-Bi	$D_{NewYork}$	$D_{Hangzhou}$	D_{Jinan}
Fixedtime [15]	209.68	209.68	1950.27	728.79	869.85
MaxPressure [24]	186.07	194.96	1633.41	422.15	361.33
CGRL [23]	1532.75	2884.23	2187.12	1582.26	1210.70
Individual RL [30]	314.82	261.60	-*	345.00	325.56
OneModel [5]	181.81	242.63	1973.11	394.56	728.63
Neighbor RL [1]	240.68	248.11	2280.92	1053.45	1168.32
GCN [18]	205.40	272.14	1876.37	768.43	625.66
CoLight-node	178.42	176.71	1493.37	331.50	340.70
CoLight	**173.79**	**170.11**	**1459.28**	**297.26**	**291.14**

*No result as *Individual RL* can not scale up to 196 intersections in New York's road network.

Figure 3.3: Performance on synthetic data and real-world data w.r.t average travel time. CoLight performs best. Image taken from the Colight paper [3].

From figure 3.1.3 one can conclude that the CoLight model was truly state of the art, at least when it came out. One important thing to note is the difference between CoLight-node and CoLight. CoLight-node uses the node distance for the neighborhood scope, while CoLight uses the geographical distance for the neighborhood scope. This means that for CoLight-node, the

neighborhood scope is between the nodes directly connected with only one edge in the `roadNet` file. For more information about the other methods, please refer to the paper.

Since the code of CoLight is all open-source, unlike some of the other methods, it was a good candidate for the RL component of the solution discussed in the introduction. More of this will be elaborated on in the research section of this thesis.

3.2 Partial vehicle detection

Studies assuming only partial vehicle detection have been done before. However, they have a strong focus on Vehicle to Infrastructure (V2I) communication.

One paper states that traditional Intelligent Traffic Signal Control (ITSC) algorithms, in most cases, assume that every vehicle is detected. These observations are done by, for example, a camera or a loop detector. However, V2I implementation would detect only those vehicles equipped with wireless communications capability [4]. The solution to V2I induced partial observations described in this paper helps with improving traffic flow, even with low detection rates of cars, compared to their fixed-time baseline [4]. This performance was achieved using a deep Q-learning algorithm.

Another paper used the research of Zhang et al. to integrate a priority system for public transport, which could encourage more people to take the bus [16]. It was also shown that the deep Q-learning method with meager detection rates outperforms the fixed-time baseline.

Therefore, these papers are focused on partial vehicle detection due to the nature of limited adoption of V2I systems but do not mention the possibility of traditional sensors failing or their noisy input. The financial cost of the conventional sensors is described, but dealing with them is different from what will be covered in this thesis.

Instead of replacing the need for traditional sensors by using V2I, a model will be proposed in this thesis that will support the inherent uncertainty and failure risk of traditional sensors as there are already so many in use. While still being able to use state-of-the-art RL methods.

Chapter 4

Research

In this section, the complete research will be explained. The variational autoencoder (VAE) used for reconstruction will be described, and learning parameters will be listed. Even though it is not the main focus of this research, the reinforcement tests will also be described. This is done because the VAE on its own is not that interesting if it can not actually aid state-of-the-art methods in partially observable scenarios.

4.1 Methods

All code used in this thesis is made available in the Github repo. In this section, references will be made to the code files for easy understanding and reproducibility.

4.1.1 Processing the data

As mentioned in section 2.1, in this thesis, public datasets are used, which need to be processed before they can be used to train the VAE. Furthermore, information about the `roadNet` is also needed to, for example, spawn agents that only need to be spawned at a particular intersection given the fact if they are observed or not.

To generate the data, a script has been made called `gen_data.py`. This script spawns MaxPressure agents such that the data will be reasonably realistic. However, it is also possible to use many different RL agents during more varied traffic data training. For example, to train the VAE, data that a learning CoLight agent produced was also used. This causes diversity in the `laneCounts` of the data. This can also be done using MaxPressure with a decaying epsilon (random choices). The critical insight is that data is required where cars are stuck (achieved by a high epsilon or by a learning CoLight agent) and situations with only a few cars.

The data can be generated by running the simulation using these Max-Pressure agents to control the traffic lights. The reason the simulation has to be run is that the dataset only provides the topology of the roads (`roadNet`) and the places where which cars should be spawned (`flowFile`).

This means that to train the VAE on the number of cars at every lane, the simulation must be run, and this data has to be saved. In the generate data script the `laneCounts` (amount of vehicles at each lane), `intersectionPhases` (the phase of the intersections) and `laneVehicleInfos` (the id and closest intersection of every car) get appended to a list every step, and finally written to a `.json` file, for the VAE training script to import.

4.1.2 Reconstructing variational autoencoder

As mentioned in the introduction, to solve the problem of reconstructing the perturbed data to its original form, a variational autoencoder was used. This gives the ability to get not only the reconstruction but also the entropy (uncertainty) of the reconstruction.

Model input

As input, the VAE gets the `laneCounts` of all lanes in an intersection plus the current phase of the intersection (one-hot encoded), plus a one-hot encoded label if the intersection was observed or not. Whether an intersection is observed or not is randomly sampled from a Beta distribution such that $p \sim \text{Beta}(1.575, 3.675)$ where p is the probability of this intersection being unobserved. This means that during the training of the VAE, the unobserved intersections constantly switch places. This is to make sure the VAE can generalize to any type of sensor failure. The data loader then sets the `laneCounts` of the unobserved intersection to all 0's accordingly. Finally, the original observed values are used as targets for the model's output when calculating the loss. Technically, the graph data information is also give to the VAE for the graph neural networks in the encoder and decoder models.

Thanks to message passing, the VAE can communicate with the other intersections without needing all the lanes of all intersections as input neurons.

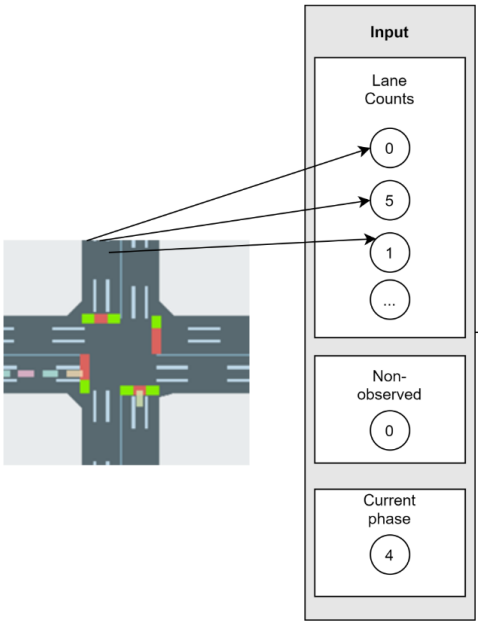

Figure 4.1: Input to the model visualized.

Model architecture

The VAE consists of multiple parts. All these parts will be described in this subsection.

Encoder

First, there is the encoder, which is a GNN implemented using PyTorch Geometric's `GraphConv` layer. In the end, 2 `GraphConv` layers were chosen as they seemed to perform well. Thanks to message passing, the lanes communicate with neighboring lanes at different intersections. With each pass through the `GraphConv` layer, a "Gaussian Error Linear Units" activation function is applied.

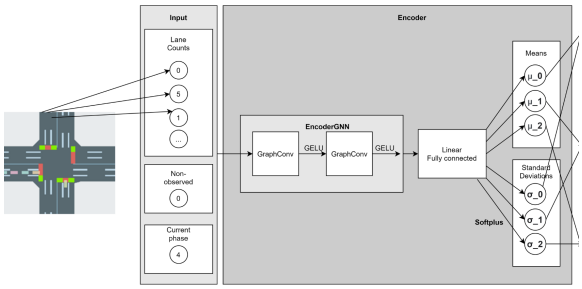

Figure 4.2: Encoder visualized.

Variational encoder

Then the encoder output gets put into the variational encoder, which first has a linear layer that maps the output of the GNN to multiple pairs of μ and σ parameters that are used to initialize a Normal distribution for every feature. These Normal distributions are then sampled using the reparameterization trick. The output of which ($\mathbf{z}$), which is also called the latent space, is then given to the decoder. The prior of the latent space variables is $\mathcal{N}(0, 1)$. The VAE has (almost) no bottleneck as there are just as many latent space features as the amount of output features. However, the latent space does need to stay close to the prior.

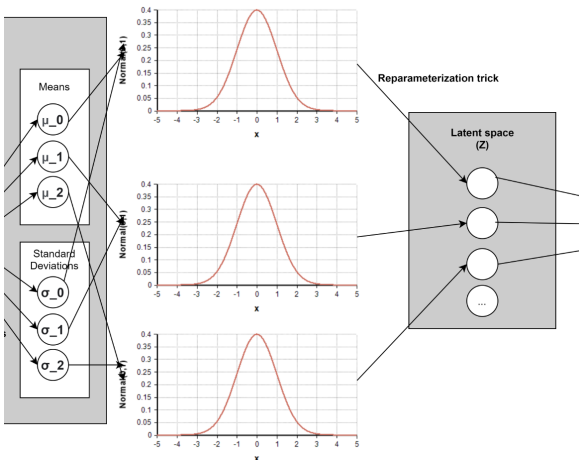

Figure 4.3: Variational encoder visualized.

Decoder

The decoder receives the sampled values from the latent space and then has the exact same setup as the encoder. 2 `GraphConv` layers and GELU activation functions for every step.

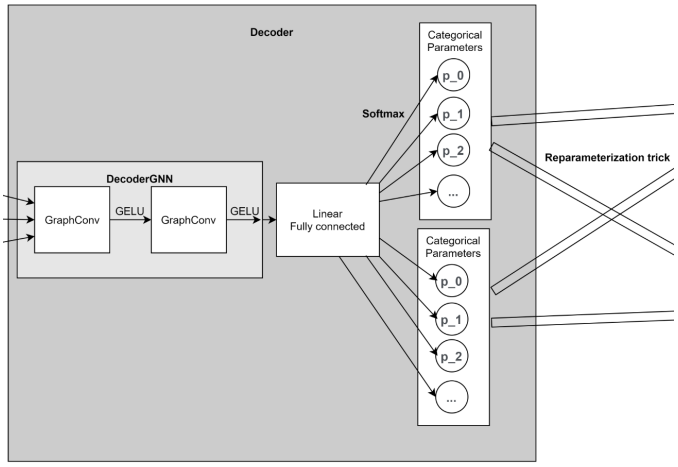

Figure 4.4: Decoder visualized.

Variational decoder

Finally, the output of the decoder gets propagated to the variational decoder, which will result in the final distribution of the model. Here again, the linear layer is mapped, through a softmax activation function, to the number of parameters necessary for the final distribution. For this model, that is a Categorical distribution for each node feature with 30 categories. The last category represents 29 or more cars, but this rarely, if ever, happens. Here too, the reparameterization trick is used to sample the output and parameters of the Categorical distribution. These parameters, amongst others, are then used to calculate the loss.

Calculating the loss

To train the network, a loss has to be specified. Then, using this loss and backpropagation, the network's weights can be trained to minimize this loss.

$$D_{KL}(\mathrm{q}_\phi(\mathbf{z} \mid \mathbf{x}) \,||\, \mathrm{p}_\theta(\mathbf{z})) - \log \mathrm{p}_\theta(\mathbf{x} \mid \mathbf{z}) \tag{4.1}$$

The loss consists of two parts. First the Kullback-Leibler (KL) divergence on the left makes sure that the approximated posterior $\mathrm{q}_\phi(\mathbf{z} \mid \mathbf{x})$ does not stray too far from the prior $\mathrm{p}_\theta(\mathbf{z})$.

The second part on the right is the reconstruction term. It is generated by calculating the log probabilities of the final distribution layer, in this case, the categorical distribution. First, these categorical distributions are initialized with the parameters generated by the decoder; then, the log prob is calculated using this distribution and the actual reconstruction targets that are desired.

4.1.3 Experimental setup reinforcement learning methods/-models

It is important to note that the focus of this thesis was not on the reinforcement learning part but rather the reconstruction of the data given partially observed data. However, since time was available and it would be interesting to see whether the VAE was working as intended, it was decided to include this at the end of the research.

For all algorithms described here, they have a minimum time ($t_{\min}$) of 10 seconds (except Cityflow FixedTime). This is done because the CoLight implementation from the Generalight repository, which is used in this thesis, is parameterized with this option. To make the comparison fair, this limit was incorporated in the other algorithms as well.

All tests have been run on the `Manhattan 16x3` map (48 agents) for 3600 steps. The plan was to use the `flowFile`: `anon_16_3_newyork_real.json` based on the real-world data provided by the traffic-signal-control/sample-code repository, however due to a mixup while training CoLight, `testflow.json` was used. This `flowFile` was generated using the `gen_flow_file.py` created for this thesis, meaning training has been done on random vehicle data. Retraining all CoLight models would take too long. Hence the `testflow.json` results will be used, which will likely paint an accurate picture of the methods' differences as they all have to deal with the same data. The downside is that real-world dynamics are lost, as in roads frequently occupied by real-world drivers in Manhattan, New York. However, the real-world data is quite sparse, while the `testflow.json` file is a lot busier, which might help CoLight learn more quickly. CoLight was trained for 200 epochs using the default parameters of the GeneraLight implementation. This copied version can also be found in the Github repo of this thesis which includes all the additions made as well. Please refer to the appendix A, for the unobserved intersections used in this test.

FixedTime

Please refer to section for more information on FixedTime.

As a simple baseline, the FixedTime "algorithm" will be used. FixedTime cycles are described in the `roadNet` file. It also seems that, at least in the `Manhattan 16x3` dataset that is used, the phase configuration is the same at every intersection at a timepoint. Also, there is no easy way to specify the *t_min*; hence it can not easily be confirmed that they switch every 10 seconds or quicker. This does not seem very realistic or optimal; therefore, an own implementation was made as well.

The own implementation randomly assigns a phase to every intersection at the start. Then they are forced to cycle every 10 seconds. Keep in mind that these two baselines do not take observations into account, which means they are one of the simplest baselines possible.

MaxPressure FixedTime

Please refer to section 3.1.2 for more information on MaxPressure.

As MaxPressure does rely on actual (preferably real world, integer) observations when intersections are not observed, it is guaranteed to fail. This means a replacement algorithm has to be used to control intersections when they are not observed. For one of the experiments, it was decided to have the FixedTime algorithm control the unobserved intersections.

The MaxPressure implementation has an epsilon setting, which will pick a random phase with a probability of ϵ. This was done such that the data generated for the VAE would be less predictable and more varied. However, during the execution of the experiments, this epsilon parameter has been turned to 0. Meaning no random actions are taken.

MaxPressure VAE

Another MaxPressure agent has been constructed, which takes the mode output of the VAE for the hidden intersections. For the observed intersections, it just gets the plain observed data. It would not make sense to use the reconstruction of the VAE for the observed data as this can, in the worst case, only introduce errors in the observations that are already available.

In this version, the random choice parameter ϵ has also been set to 0 to avoid randomness in the results.

CoLight without VAE

Three tests with CoLight will also be run. This is an actual deep reinforcement learning algorithm. For more information on CoLight, please check out section 3.1.3.

To see how CoLight would perform in situations where no data would be provided about certain intersections, CoLight is trained without the VAE. The input to CoLight, in this case, are all 0's at the unobserved intersections, with a non-observed label being 1. CoLight does get the phase of the model because the reinforcement learning model controls the phase, so even though the intersection is unobserved, the agent should easily be able to remember

the phase it chose previously. The input for observed intersections is the original data again.

CoLight with VAE (mode + uncertainty)

Then, to see how much the VAE model would help in unobserved situations, its reconstruction is linked to CoLight. It might be possible for CoLight to then learn how to interpret the reconstruction coupled with the uncertainty the VAE gives, hence reason about this uncertainty to make better predictions.

In this case, for unobserved intersections, CoLight will get the mode of the output of the categorical distribution of the VAE and the entropy per intersection. In addition, of course, it will also get the phase encoded as one-hot labels and the non-observed (1) label.

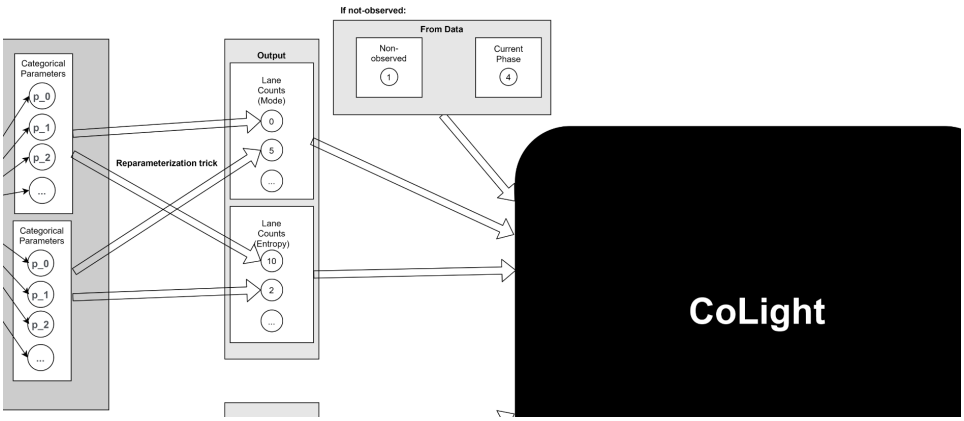

Figure 4.5: CoLight VAE input for unobserved intersections from VAE.

For the observed intersections, the original data is again used for the mode of the observations, and the entropy is put to 0 (because there is no uncertainty). Then again also the phase and non-observed (0) label is given.

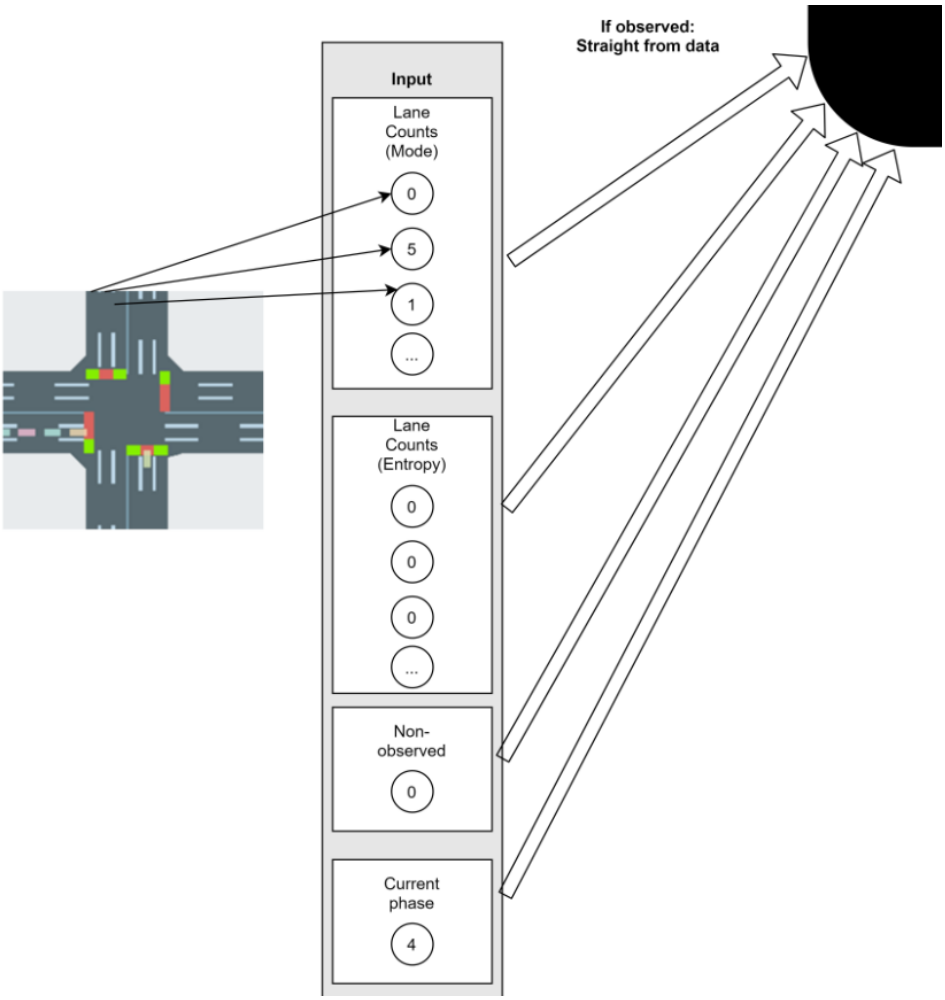

Figure 4.6: CoLight VAE input for observed intersections straight from the data.

CoLight with VAE (mode only)

To observe if providing the CoLight model with an uncertainty measure of predictions makes a difference, a CoLight model with only the mode, current phase, and non-observed label as input has also been trained. This is another ablation that will show if the current solution with uncertainty contributes something or if just reconstruction is enough.

4.2 Results

Please refer to the discussion section of the respective results in the next chapter to interpret these results.

4.2.1 VAE reconstruction results

First, the reconstruction results will be shown, as these are needed to provide the reinforcement learners with the relevant output.

Training results

Several distributions have been experimented with for the last (probabilistic) layer; the training results will be shown here.

Normal distribution

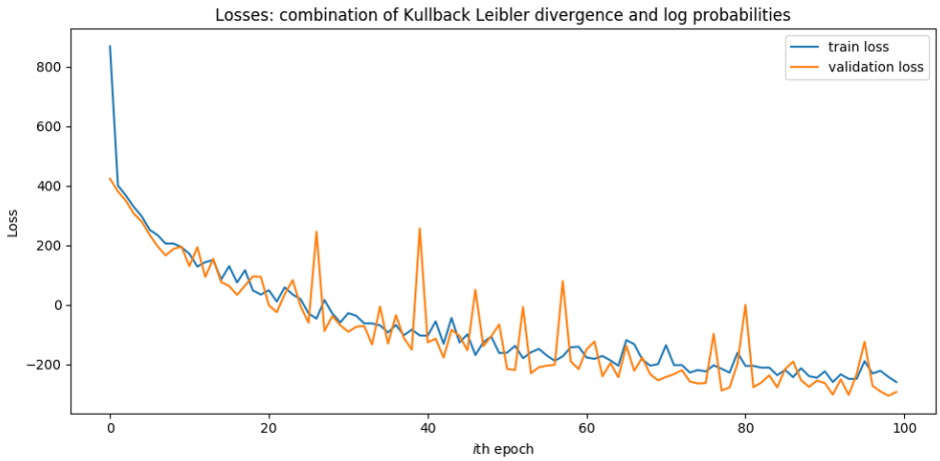

Figure 4.7: Training loss of the normal distribution.

Figure 4.5 shows that when using the Normal distribution as the last layer, from the figure, it can be seen that the combination loss decreases.

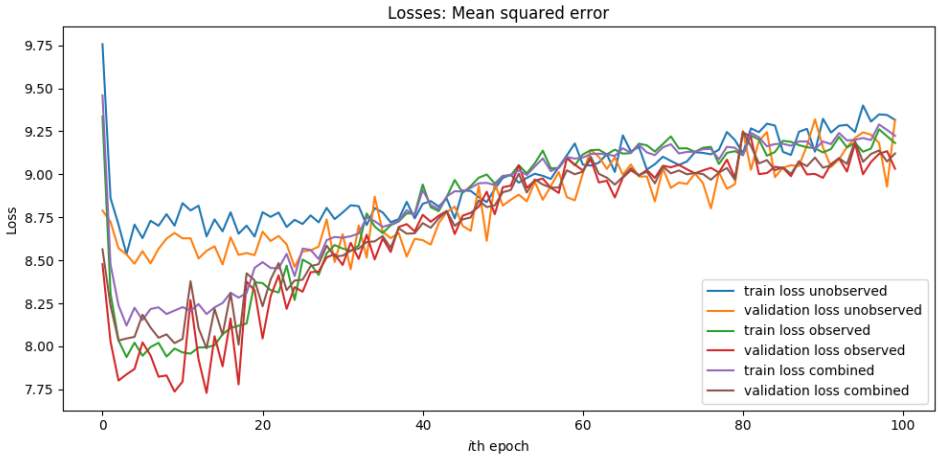

Figure 4.8: MSE loss of the normal distribution.

Figure 4.6 shows the train and validation loss of the observed and unobserved intersections. As can be observed, after convergence, there is almost no difference between the loss in unobserved and observed intersections. The MSE loss also seems to be getting worse the longer the training goes on.

Log-normal distribution

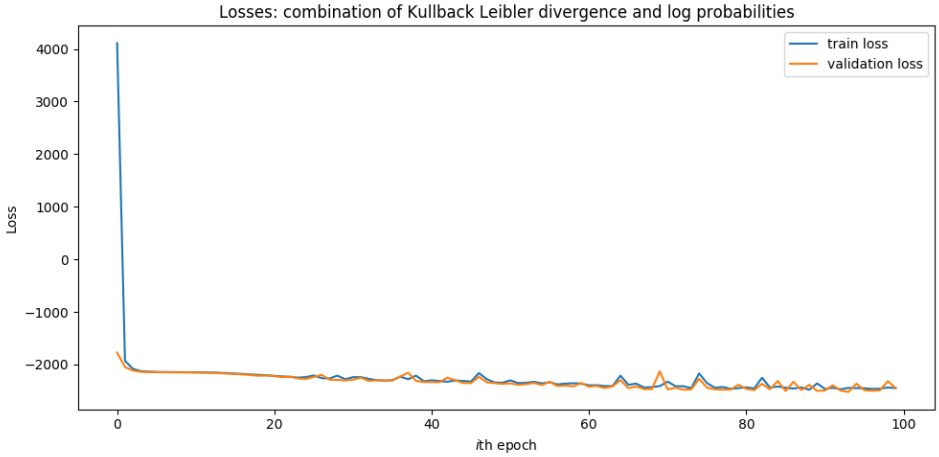

Figure 4.9: Training loss of the log-normal distribution.

As can be seen from this figure, the combination loss rapidly decreases, much more than that of the normal distribution.

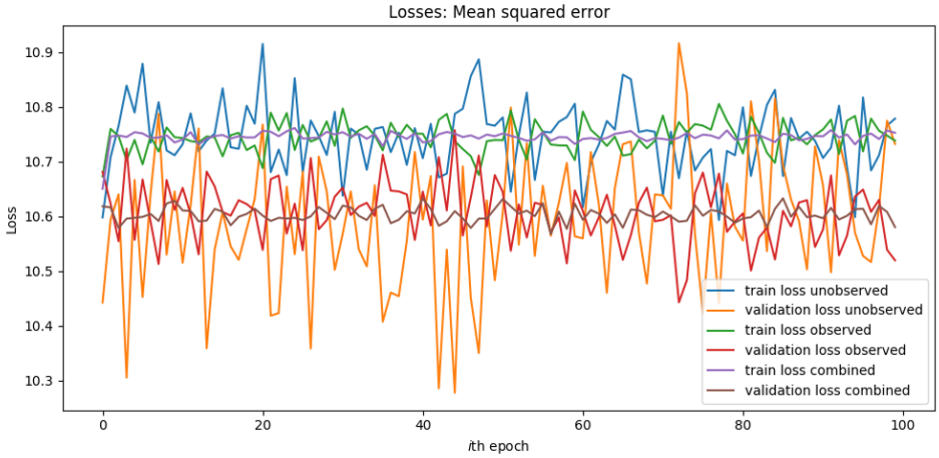

Figure 4.10: MSE loss of the log-normal distribution.

Instead of the MSE losses getting worse during training, they almost stay constant throughout training; there also seems to be minimal difference between observed and unobserved intersections.

Categorical distribution

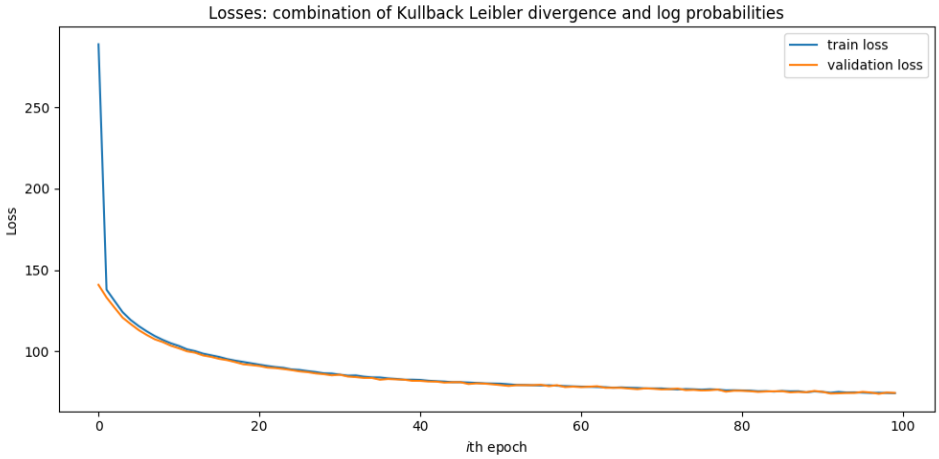

Figure 4.11: Training loss of the categorical distribution.

From the figure above, it can be seen that the categorical distribution decreases the combination loss throughout the training period.

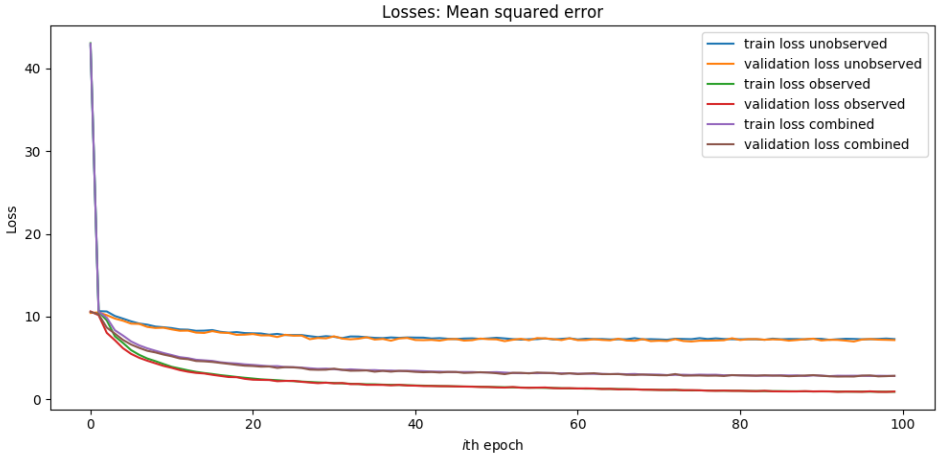

Figure 4.12: MSE loss of the categorical distribution.

Figure 4.12 shows that the MSE loss decreases throughout training, and there is a distinct difference between the observed and unobserved intersection scores. The unobserved loss seems to have dropped slightly, but not as much as the observed loss. This is quite different from the other distributions showcased because their reconstruction was much worse, and there was almost no difference between observed and unobserved intersections.

Output of the model

A plotter was made to visualize the data and the VAE output. In this section, an example will be given to show how well the VAE has learned. In the plots, black intersections are observed, and white intersections are masked.

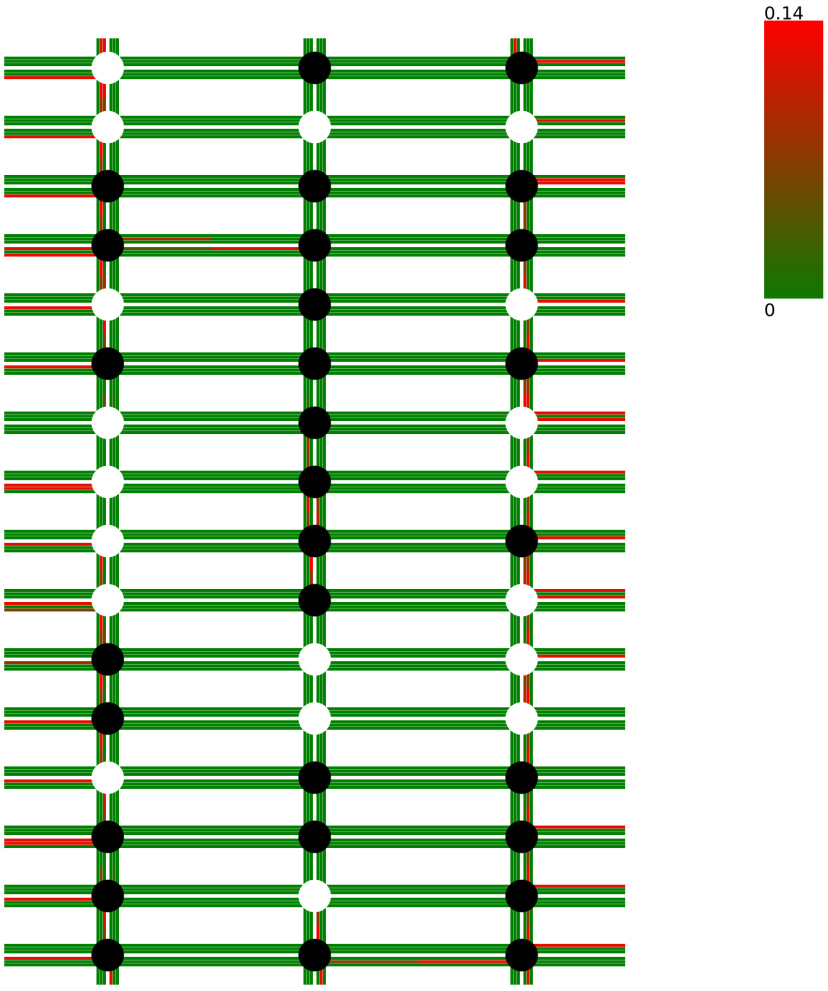

Figure 4.13: Input to the VAE.

As can be seen from the input, there are several unobserved intersections, many of which are on the sides. Hence, the network can not get information from the intersections that sent these cars to the sides, as this is where the engine spawns them.

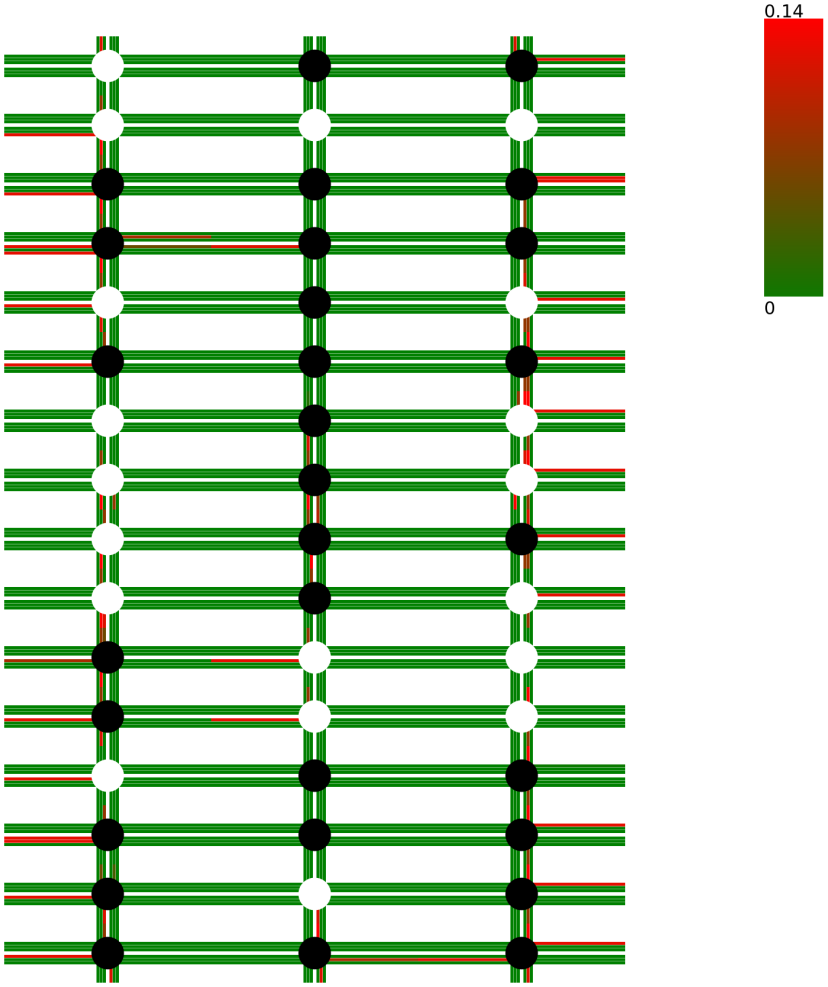

Figure 4.14: Output to the VAE.

The output of the network can be seen in the figure above. Note that most cars at the unobserved intersections on the left and right sides are not accurately reconstructed. The interpretation of the output will be given in the discussion section.

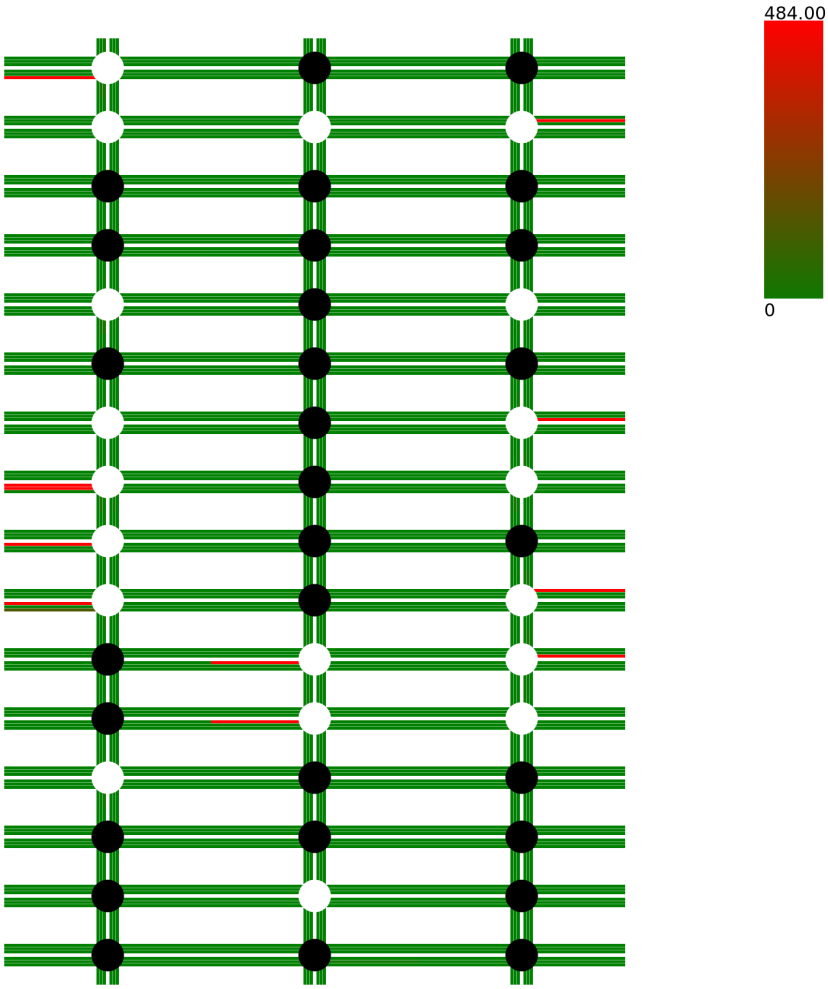

Figure 4.15: Errors (MSE loss) of predictions of VAE.

As can be seen from the legend on the top right of the figure above, the predictions are far off on some of the unobserved intersections. This likely overshadows the smaller errors that were made. Because of this, to get an accurate picture of the predictions, one should compare the input and output directly. Referring back to the categorical MSE losses during training will also give a good idea how well the model typically performs.

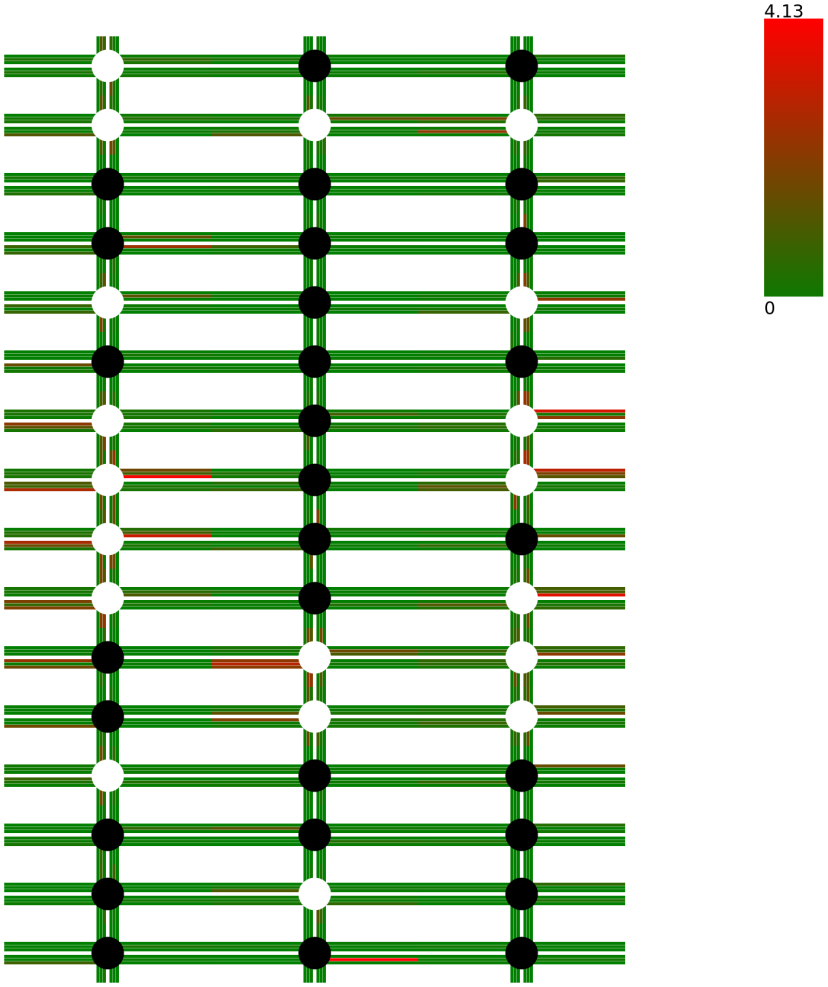

Figure 4.16: Entropy of the prediction.

As can be seen, the entropies around the unobserved intersections are much higher than the entropies around the observed intersections. They also seem to correlate with the errors in the previous figure.

4.2.2 Reinforcement learning results

In this section, both the training results of the reinforcement learning methods that need training (which in this case is only the three CoLight models) will be displayed and their results in the testing phase.

Training results

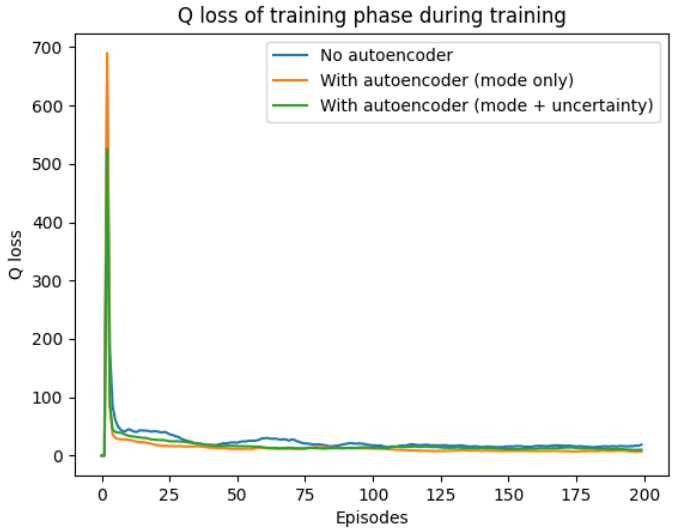

Figure 4.17: Q-loss of CoLight trained with and without VAE, including some hidden intersections. For one CoLight only the mode is given and for the other one both the mode and uncertainty are given.

The CoLight models with autoencoder seem to converge in fewer episodes than the no autoencoder model. However, it should be noted that even though it might converge in fewer episodes, running the VAE model takes more time, and that will add up during all those timesteps.

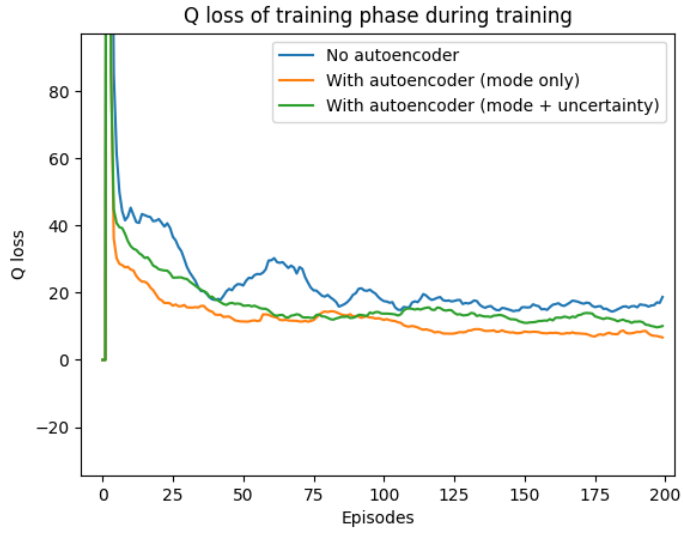

Figure 4.18: Zoomed in version Q-loss of CoLight trained with and without VAE, including hidden intersections.

From the zoomed-in plot, it can be seen that the mode-only version converges quicker than the other models based on Q-loss.

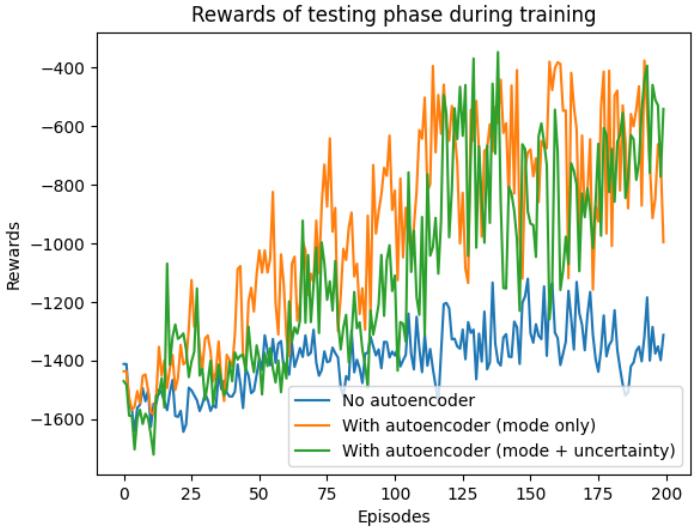

Figure 4.19: Rewards of CoLight models with some hidden intersections (reward should be maximized).

This figure shows that the CoLight models with autoencoder get a better

reward, although it is more variable than the version without autoencoder. Here too, there are significant spikes early on with the mode-only version, suggesting it reaches a higher level of competence faster. However, it is essential to note that the final difference between the two CoLight models with VAE is not very big.

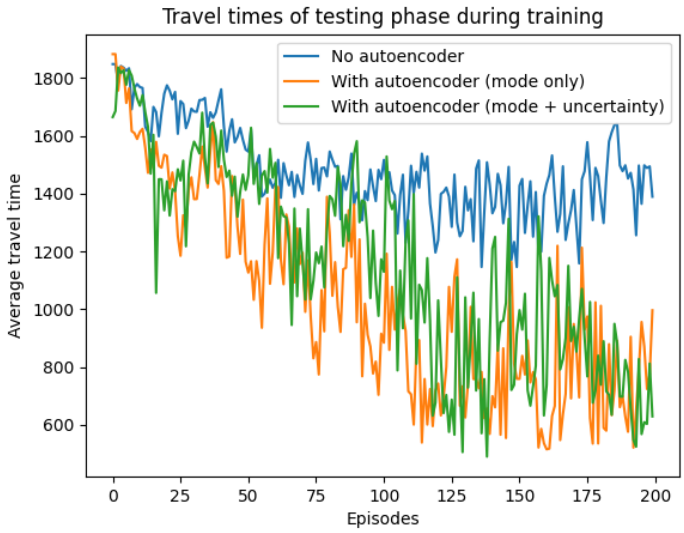

Figure 4.20: Average travel times of CoLight models with some hidden intersections.

Since reward (GeneraLight's wrapper defines the negated sum of waiting vehicle counts at an intersection as the reward) and average travel time are related, the autoencoder models perform visibly better.

Testing results

The following results have been generated on `Manhattan 16x3 roadNet` file with the `testflow.json flowFile`.

Model	Last Average Travel Time
MaxPressure with VAE	1436
CoLight without VAE	1389
FixedTime (CityFlow)	1164
FixedTime (own implementation)	911
CoLight with VAE (mode + uncertainty)	772
CoLight with VAE (mode)	629
MaxPressure with FixedTime	402

Table 4.1: Results of the tests run with all agents. Average travel time should be minimized for optimal traffic flow, hence lower is better. For the CoLight models the last testing phase results are taken after 200 episodes of training.

From this table, it appears that actually, the model without the uncertainty performs better. However, from the training results graph earlier, it was clear that there are many fluctuations between each episode during the training process. To correct this, the minimum (best) average travel time was also extracted from the training phases shown in the table below.

Model	Best Average Travel Time
MaxPressure with VAE	1436
FixedTime (CityFlow)	1164
CoLight without VAE	1146
FixedTime (own implementation)	911
CoLight with VAE (mode)	515
CoLight with VAE (mode + uncertainty)	490
MaxPressure with FixedTime	402

Table 4.2: Results of the tests run with all agents. Average travel time should be minimized for optimal traffic flow, hence lower is better. For the CoLight models the minimum testing phase results are taken after 200 episodes of training.

As can be seen from Table 4.2, the CoLight model with uncertainty performs slightly better than the CoLight model that only gets the mode. However, this difference might not be significant. Another interesting finding is that CoLight without any reconstruction performs better than the FixedTime described in the `roadNet` file. However, the MaxPressure + FixedTime algorithm still outperforms all CoLight models.

For the interpretations of these results, please read the discussion in the next chapter, section 5.2.

Chapter 5

Discussion

In this chapter, the results given in the result section will be interpreted, the implications will be expressed, the limitations and recommendations will be discussed.

5.1 Variational autoencoder

5.1.1 Distribution comparison

Normal distribution

From the training results, it can be inferred that even though the normal distribution does show that it lowers the combination loss while training, the network does not learn to reconstruct the data. This can be caused by the fact that the normal distribution as the last layer to a VAE can introduce instabilities [5]. Furthermore, the normal distribution also has the problem of producing negative values, which would not make sense for this particular scenario because there can be no negative number of cars on the road.

However, for the reinforcement learning methods (and MaxPressure), this might not be a real problem if the VAE could reconstruct the values well. But from the MSE loss, it can be seen that this not the case. Hence the normal distribution is ruled out as a distribution for the last layer.

Log-normal distribution

To combat the problem of the normal distribution potentially not predicting realistic results, the log-normal distribution has also been tried. But as explained previously, the results (MSE loss) are even worse than the normal distribution. With barely any difference between observed and unobserved intersections from the start.

A possible explanation for this phenomenon is that the log-normal distribution can not predict a zero (0) number of cars. A tiny number (0.000001) was constantly added to the original observations when calculating the log probabilities of the parameterized distribution to circumvent this. This ensures that it could train on close to zero values without crashing, but this, combined with the odd shape of the log-normal distribution, might have caused some numerical instabilities and/or inability to learn correctly. Hence also the log-normal distribution has not been used as the final layer of the VAE.

Categorical distribution

The results of the categorical distribution look promising. First of all, as previously described, the combination loss decreases sufficiently. But, most importantly, the MSE loss decreases in a way that is desired. Additionally, the MSE losses between the observed and unobserved intersections are significant this time.

If you go back to figure 4.10, it can be seen that the maximal reconstruction loss of the observed intersections approaches zero. This is expected of a well-functioning final distribution as all the VAE has to do is pass through the values (all be it in an encoded manner due to the prior on the latent space values). The unobserved intersection error also slightly decreases after a few epochs showing that the network does pick up some cues as to how to reconstruct the missing data.

The biggest downside to the categorical distribution is the fact that the loss does not care about the distance of reconstructions, e.g., the network is punished as severely for a reconstruction of 2 cars at a lane while there is only 1 as when in the case of 1 car it predicts 30. Clearly, the 30 case should be punished harder, but due to the nature of the categorical distribution, this is not possible with the currently defined loss. In this case, then, the loss acts more like a cross-entropy loss. However, as seen in figure 4.10, this does work better than the other distributions; thus, the categorical distribution was chosen as the final layer for the VAE.

5.1.2 VAE reconstruction results

The results have already been described in section 4.2.1. In this section, a brief discussion will be done based on this reconstruction example.

Firstly, please observe that there are a large number of unobserved intersections on the side. This could be realistic in a scenario with a massive failure of sensors along the outside but not very realistic when trying to downscale a trained model. Therefore, in the downscaled model, it is more reasonable to take a part of the larger model with functioning sensors on the outside. This could help with communication more as the unobserved intersection would have more observed neighbors. Granted, there need to be intersections around those unobserved intersections that are observed.

From the output, one can observe that the network does a reasonable job at reconstructing the input, even for some of the unobserved nodes. However, from the error plot in figure 4.13, one can see that the legend goes up to a tremendous value. The biggest mistakes happen around the unobserved intersections, which is expected and is mainly a problem with the network having too little information about the number of cars at that spot, preventing it from giving accurate reconstructions.

Though, from the RL results that will be discussed in the next section, it is clear that this reconstruction works much better for the learning reinforcement learning method (CoLight) than no reconstruction at all.

As described before, the entropies seem to be mostly centered around the unobserved intersections. This seems to imply that the network has indeed learned to be uncertain about its predictions. This information could help the RL models reason about the uncertainty of the prediction. This might give it the ability to trust the output less and try to combine the information of other intersections as well.

Preferably these predictions would be even better. A large expected improvement that can be made is adding recurrency to the data or VAE. More on this will be elaborated in the future work section: 5.4.

5.2 Reinforcement learners/methods

MaxPressure with VAE

This method is the worst of them all. After inspecting the `replay.txt` (file that shows the paths of the cars) in the Cityflow frontend, it is easily observable why this is the case. The reconstruction from the VAE always contains minor errors, even if fully observed but mainly when not observed. Due to the fact that MaxPressure expects complete and fully observed values, presenting it with the noisy VAE reconstruction, this goes horribly wrong.

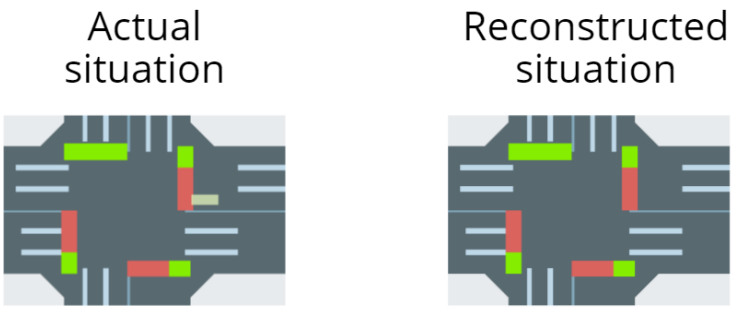

Figure 5.1: VAE reconstruction where the VAE's mode is 0, leading the car to be stuck.

When there is only one car waiting at an intersection, but the reconstruction says no cars are waiting, then this car will be stuck indefinitely, or until everything is so stuck the pressure might change anyway. By then, it will have been too late.

Since this version does not reason about the uncertainty of the prediction, it makes sense it does not perform well. This could potentially be fixed by sampling from the VAE's last layer in future research instead of taking the mode.

CoLight without VAE

CoLight without VAE also seems to perform poorly. This seems to imply that CoLight cannot reconstruct the required data independently by communicating with the neighboring intersections. Of course, this is not good for CoLight, but it does mean that the research in this thesis is essential. If CoLight cannot deal with these kinds of perturbations independently, a reconstruction network of some sorts is required.

From the minimum average travel times, it does seem that CoLight without VAE at least performs better than the FixedTime of CoLight. However,

this might be because CoLight can switch phases every 10 seconds, and with the `roadNet` file, this not as easily controllable. This means that the FixedTime implementation of CoLight might take more than 10 seconds per phase, which leads to unfair comparison.

FixedTime (Cityflow)

As expected, the FixedTime of Cityflow does not perform very well. However, as mentioned before, there was no control over the time the lights can switch phases ($t_{\min}$; hence these results should be taken with a grain of salt).

The results show, that at least on this `roadNet` with this specific `flowFile` it is not optimal to have all intersections on the same fixed timer.

FixedTime (own implementation)

The FixedTime implementation performs slightly better than the FixedTime model provided by the `roadNet` file. This could have two reasons, either starting all the intersections on random phases is beneficial, or the $t_{\min}$ of this script is lower, and that is what is contributing most. However, both reasons could be valid. Regardless, this is the fairest baseline without observations as it has the same $t_{\min}$ as the other algorithms.

The own implementation of FixedTime also performs better than that of the CoLight without VAE. This clearly shows that CoLight is not robust and was not designed for these partial vehicle situations, at least on its own.

Colight with VAE (mode + uncertainty)

The results in table 4.2.2 seem surprising at first. Typically CoLight outperforms MaxPressure even when it is mainly with a small margin, see section 3.1.3. There it can also be observed that FixedTime is almost the worst method. Thus, it would not be weird to expect MaxPressure + FixedTime to perform worse than CoLight with the reconstruction and entropy for hidden intersections. However, when unobserved intersections are introduced, it seems to perform worse.

This can have several reasons; the first reason could be that reinforcement methods, at least CoLight, are just not that good when intersections aren't observed. Even when the data is reconstructed, and the uncertainty about this reconstruction is given. However, this may seem odd as, usually, CoLight outperforms both MaxPressure and FixedTime, so a reconstruction should be more informative than a FixedTime schedule. It could be that, because the temporal information is not taken into account, CoLight might be stuck in the same phase too long, like with the MaxPressure + VAE

algorithm.

The second reason can be because, in the original CoLight paper, the authors state: "Following the tradition, each green signal is followed by a three-second yellow signal and two-second all red time" [3]. However, in this research, the Generalight implementation of CoLight has been used. This has been done because the wrapper of the public CoLight repo was not as neat as the one of Generalight. However, from the code, it seems that Generalight violated the tradition described by Colight. In their world wrapper, the agents can switch phases immediately, without yellow and red time. It could mean that CoLight normally performs much better than MaxPressure when having this red-light limitation. Perhaps communication with neighbors is essential when trying to coordinate when to switch the phase, as phase switching would be penalized with 5 seconds of red lights. Obviously, having a time between phases where cars have to stop is more realistic than switching phases instantly.

The third reason can be that CoLight just does not perform that well in `Manhattan 16x3` and does better in other maps. For this, CoLight will have to be trained on more maps. Ideally, CoLight is transferrable, and it would be possible to transfer learn. However, the Generalight code does not allow for re-loading the model currently due to a bug. Hence it would be needed to train CoLight from scratch for all the maps, however, training CoLight from scratch usually takes around 12 hours on a Ryzen 5600x. Hence this will not be experimented with in this thesis.

Other than that, an improvement of an average travel time of 656 (minimum overall training phases) compared to the previous CoLight model (without VAE) is substantial. Hence, the VAE makes a tremendous difference.

Additionally, a difference with MaxPressure + FixedTime of 88 minimum average travel time is not poor. However, the amount of engineering that went into the CoLight + VAE (mode + uncertainty) solution is much greater than the simple MaxPressure + FixedTime solution. Therefore, the expectation would be that a government would instead apply the understandable and easy to run MaxPressure + FixedTime algorithm to their intersections rather than the CoLight + VAE model that has to be trained for hours upon hours and relies on communication between intersections.

CoLight with VAE (mode only)

From the results table 4.2.2 the CoLight version that only gets the mode seems to perform reasonably marginally worse than that of the CoLight ver-

sion with VAE (mode + uncertainty), however the difference is not that large. From this one `roadNet` and `flowFile` it is therefore too early to claim if the uncertainty measure helps. More tests will have to be run on real flowFiles and bigger/more diverse maps to conclusively say something about this.

The VAE could also be improved by choosing better distributions or parameters, to improve the reconstruction of the unobserved nodes. This will most likely greatly benefit both CoLight VAE models and likely change the difference between the two models. If the VAE will be better at reconstructing the entropy will most likely go down and this can have large effects on the comparison.

MaxPressure + FixedTime

As said before, MaxPressure + FixedTime performs best while the expectation would be that CoLight works better. Baseline MaxPressure achieves an average travel time of 200 (using the same flowFile, but fully observed), so MaxPressure does lose some of its edge by having FixedTime components, but it means there is still room for improvement by an RL method.

The reason why MaxPressure + FixedTime might perform so well is, as explained previously, most likely because it forces the unobserved intersections to change phases periodically instead of relying on the reconstructions being accurate. It could be that the VAE outputs all 0's on a specific intersection while actually, cars are waiting somewhere.

5.3 Research question

At the start of the thesis, the following question was posed: "How can traffic flow, in terms of average travel time, be improved given only partial observations of cars at intersections?"

Given the results, it seems that it can be confidently stated that the VAE model that was proposed can effectively help reduce average travel time, coupled with a state-of-the-art traffic signal control reinforcement learning method (CoLight). Thus, even though it seems like MaxPressure + FixedTime seems best with the current setup, the effectiveness of the reconstruction and the fact that uncertainty can be modeled holds much promise. Especially because many improvements can be made in future work, while MaxPressure will not be able to improve as it cannot learn.

5.4 Future work

For future work, the first thing that should be done is to look into how these models perform when the TSC research tradition of 5-second red-light after phase change is applied. For example, it could be that RL methods like CoLight are better equipped to deal with this than the baselines of Max-Pressure and FixedTime.

Another thing to look into is giving the VAE noisy data instead of masked/omitted data. As described in the introduction, some of the sensors, such as camera's carry some uncertainty about the predictions they make. Especially when faulty, the observations these sensors make can be noisy and should be corrected for optimal travel flow.

Another improvement that can be made which would most likely help the performance of the VAE significantly is by adding recurrency to the model. Right now, a lot of temporal information is lost. It likely helps the model knowing it previously sent 5 cars to a (masked) intersection. The model can then infer that these 5 cars would arrive sometime later at that (masked) intersection. Just improving the VAE in general, with more layers or a smarter choice of distributions might help the reconstruction tremendously which will likely benefit all models that depend on the VAE.

There is also the possibility of adding the recurrency at the RL agent or giving the RL agent a measure of how long they have been in the current state. This is outside of the scope of this thesis, but especially with unobserved intersections. This measure is crucial as it can prevent cars at intersections from being stuck. As was seen in MaxPressure+VAE, this can easily happen when the reconstruction is not perfect. If an RL agent knew it was in the current phase for a long time, it might be time to switch just to make sure.

Temporal information can also be given by making the graph a 3d structure, with on the additional axis the time such that each intersection (node) connects with itself of the past. This allows for message passing through time. An additional weight parameter could also be learned for these messages through time, which will help decide how much the network will value these messages.

Chapter 6

Conclusions

In conclusion, this work has focused on the current challenges in traffic signal control. It has proposed a method to alleviate the problem of partial vehicle detection due to sensor failure or cost-saving measures.

Using a variational autoencoder and graph neural networks, it is possible to reconstruct the traffic data so that an uncertainty measure about the uncertainty of the prediction can be generated. Reinforcement learning methods could exploit this uncertainty by learning to "mistrust" the reconstruction and therefore make more educated decisions than with pure reconstructions. This thesis has only shown a marginal improvement between pure reconstruction and a model with the uncertainty information. However, the reinforcement learning methods with reconstruction perform better than the ones without VAE. This thesis has shown that the VAE described can help make state-of-the-art TSC learners more robust.

Because these learning models can be improved as opposed to defined algorithms like MaxPressure this way of reconstructing and learning to deal with the uncertainty still holds a lot of promise. In the related works section, many changes have been discussed that can help improve the pipeline. Recurrency can be added to one of the networks to make it aware of temporal dependencies. Different and deeper layers could be used to improve message passing communication and perhaps gather more information that way. Most importantly a larger variety of maps should be used to see if the proposed pipeline of models (VAE + CoLight) does not already outperform MaxPressure + FixedTime in some maps.

The reconstruction VAE model described in this thesis helps make learners more robust and can therefore aid with bringing these state-of-the-art RL models to the real world. If these models outperform the current traffic systems it could have a large positive impact on society.

Bibliography

[1] X. Zang, H. Yao, G. Zheng, N. Xu, K. Xu, and Z. Li, "MetaLight: Value-Based Meta-Reinforcement Learning for Traffic Signal Control," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, pp. 1153–1160, 2020. 3

[2] H. Zhang, C. Liu, W. Zhang, G. Zheng, and Y. Yu, "GeneraLight: Improving Environment Generalization of Traffic Signal Control via Meta Reinforcement Learning," *International Conference on Information and Knowledge Management, Proceedings*, pp. 1783–1792, 2020. 3, 10

[3] H. Wei, N. Xu, H. Zhang, G. Zheng, X. Zang, C. Chen, W. Zhang, Y. Zhu, K. Xu, and Z. Li, "Colight: Learning network-level cooperation for traffic signal control," *International Conference on Information and Knowledge Management, Proceedings*, pp. 1913–1922, 2019. 3, 24, 25, 55

[4] R. Zhang, A. Ishikawa, W. Wang, B. Striner, and O. K. Tonguz, "Using Reinforcement Learning with Partial Vehicle Detection for Intelligent Traffic Signal Control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 1, pp. 404–415, 2021. 3, 27

[5] H. Takahashi, T. Iwata, Y. Yamanaka, M. Yamada, and S. Yagi, "Student-t Variational Autoencoder for Robust Density Estimation," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, vol. 2018-July. California: International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 2696–2702. [Online]. Available: https://www.ijcai.org/proceedings/2018/3745, 50

[6] H. Zhang, Y. Ding, W. Zhang, S. Feng, Y. Zhu, Y. Yu, Z. Li, C. Liu, Z. Zhou, and H. Jin, "CityFlow: A multi-agent reinforcement learning environment for large scale city traffic scenario," *The Web Conference 2019 - Proceedings of the World Wide Web Conference, WWW 2019*, pp. 3620–3624, 2019. 7

[7] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Foundations and Trends in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019. 11

[8] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, no. January, pp. 57–81, 2020. [Online]. Available: https://doi.org/10.1016/j.aiopen.2021.01.001 17

[9] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*, pp. 4602–4609, 2019. 17

[10] M. Fey, "torch_geometric.nn pytorch_geometric 1.7.0 documentation graphconv layer." [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html?highlight=layers#torch_geometric.nn.conv.GraphConv 17

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," pp. 1–9, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602 18

[12] S. Bhatt, "5 Things You Need to Know about Reinforcement Learning - KDnuggets," 2018. [Online]. Available: https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html 19

[13] T. Urbanik, A. Tanaka, B. Lozner, E. Lindstrom, K. Lee, S. Quayle, S. Beaird, S. Tsoi, P. Ryus, D. Gettman, S. Sunkari, K. Balke, and D. Bullock, "Signal Timing Manual," *NCHRP Report*, no. 812, p. 317p, 2015. [Online]. Available: http://www.trb.org/Main/Blurbs/173121.aspx%0Ahttps://trid.trb.org/view/1367911 22

[14] H. WEI, G. ZHENG, V. GAYAH, and Z. LI, "A survey on traffic signal control methods," *arXiv*, vol. 1, no. 1, 2019. 22, 23

[15] P. Varaiya, "Max pressure control of a network of signalized intersections," *Transportation Research Part C: Emerging Technologies*, vol. 36, no. April, pp. 177–195, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.trc.2013.08.014 22

[16] C. Geortay, "Partially Detected Intelligent Traffic Signal Control using Connectionist Reinforcement Learning," 2020. 27

Appendix A

Reproducibility details

Please check out the github repo to reproduce the results with the code designed for this thesis.

The unobserved intersections that were used in the test:

```
unobserved_intersections = {'intersection_1_2', '
    intersection_2_1', 'intersection_2_8', 'intersection_1_4', '
    intersection_2_4', 'intersection_1_6', 'intersection_1_14', '
    intersection_3_11', 'intersection_2_12', 'intersection_3_16',
     'intersection_3_3', 'intersection_3_14', 'intersection_1_13
    ', 'intersection_3_9', 'intersection_3_12', 'intersection_3_4
    '}
```