Traffic Control and Infrastructure Organization Using Reinforcement Learning

Daniel Kuknyo

2022 Fall Semester

Abstract

The purpose of this research was to develop a reinforcement learning model that can optimize traffic flow in a city by building roads, lanes and intersections between predefined nodes that are defined on a user-friendly interface. The model will take into account how fast and dynamically does traffic flow and how friendly a city is to its residents: the number of lanes and infrastructure cost have to be minimal while also minimizing the time taken to travel between junctions. To conduct the experiment, the machine learning model uses a simulated environment that incorporates an intelligent driver model in order to provide the most accurate mapping between real life and the modeled environment. Future uses cases include optimizing current road structure and generating new paths between already existing cities or settlements that use the least amount of taxpayer money while also providing them with the fastest travel times. The developed framework can also be generalized for uses in public transport: using smart forecasting to optimize bus or train lanes to cover the most area, for as many people as possible.

Part I
Introduction

1 Historic overview

In recent years the traffic of cities became a rising topic with more and more city governments realizing that a motor-focused city design is unsustainable. Cities all around western Europe have started designing their cities around humans and public transit, and not around cars. E.g. Paris is planning to be a 15-minute city, Barcelona is incorporating the superblock design, Finland is creating non-intersecting paths between the common intersections and the Dutch are using intelligent traffic light systems to manipulate traffic flow in order to optimize it for both cars and pedestrians.

If one takes a look at how the Dutch infrastructure is designed, they will see that despite having less car lanes and overall less space for cars, the traffic flows more smoothly. This is thanks to the intelligent design of intersections, traffic lights and infrastructure. The methodology of this has been known ever since the 1970s, but the auto industry has been fighting against it ever since in order to gain more market. In Northern America one can observe what happens if a city is designed with cars in mind, requiring everyone to own a vehicle in order to participate in society. This results in worse accessibility to the city for the disabled, incapable, elderly and young people as well. The methodology of how to create walkable, human-centered and intelligent infrastructure that is optimal for both pedestrians and cars is laid out in detail by books from authors such as Strong Towns, an urban planning organization.

The aim for this research is to able to model a system of roads or city, and being able to pinpoint mistakes made by development engineers, with the goal in mind to make the city more humanly livable, liquidate urban highways and make traffic infrastructure more efficient.

2 Goals and Outline

The project will focus on building an interface that models traffic flow in a graph-based structure, then train a reinforcement learning algorithm to find the optimal configuration of the roads in order to transport the most cars in the most effective way possible. Here's when the urban design principles come in: one can easily observe that the most effective way to transport as many cars as possible is if all roads are 8-lane highways. However it's also easy to see that it's miserable to live in a city where there are no quiet, auto-low streets and only 8-lane highways. This might be the best configuration for cars, but it would make the life of people living in the city absolutely horrible. The rewarding system of the reinforcement learning environment will be designed in order to reflect these principles: building cost, traffic light/roundabout tradeoffs, how humans would feel living next to the road. The agent will have to decide where to build, destruct, or make roads 1-way to make the city's transportation flow dynamic but also make it livable for humans. The rewarding scheme will reflect the principles laid down by Strong Towns and other urban planning organizations significant in the field like Happy Cities: Transforming Our Lives through urban design.

The main metrics that are taken into account during the research are:

1. Time taken to arrive to the goal junction
2. Cost of infrastructure
3. Livability by humans (number of lanes, connection density)

Part II

Methodology

3 Constructing a city

The developed software will provide an interface where the user can make a graph, describing the intersections (nodes) and roads (edges) of the city in question. This for example can be done in GeoGebra and exported into a construction protocol in order to work as an input for the model. Below is an example simple city constructed:

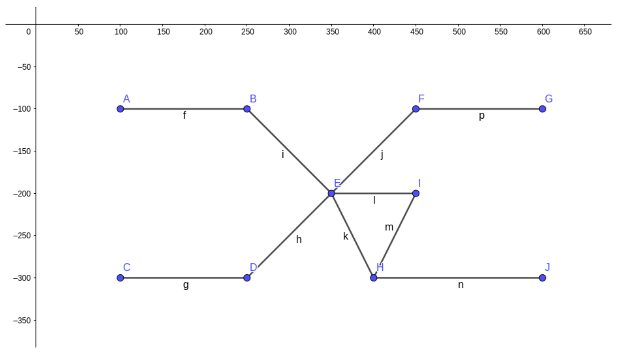


Figure 1:

The interface will read a construction protocol, construct a graph and all the possible pathways from it. As a starting configuration every road is 2-way on the edges that have been defined in the graph. The vehicle rate and distribution can be controlled before starting the simulation. A constructed "starting" city according to the previously shown graph will look like as follows:

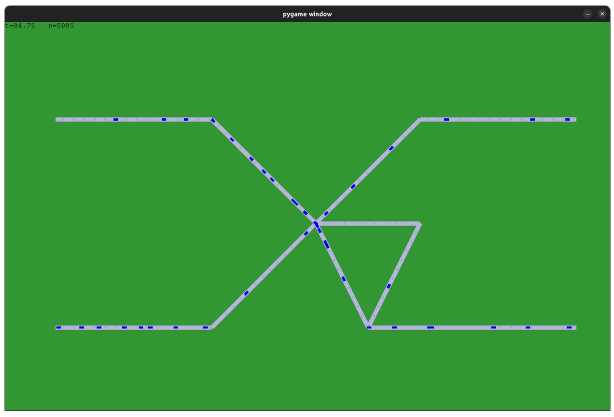


Figure 2:

So far this is a simple setup for demonstration. The vehicles are passing from one entry point to another, without necessarily choosing the shortest path, or being evenly distributed among all the roads, just as one would find in real life. The driver model will incorporate an intelligent behavior, like slowing down after the car in front is slowing down or gradually speeding up after a light has turned green with a comfortable acceleration parameter.

The road configuration will be examined with multiple metrics like how many steps does it take for the roads to transport 100 cars or how much the road infrastructure would cost. If the agent is handed a road configuration it will be able to find the optimal one, with the least cost, least unnecessary roads and fastest transportation for a given amount of cars.

As a distant goal it would also be reasonable for the application to accept Osmosis data and construct a graph based on that. This would require a more sophisticated preprocessor as the Osmosis data would have to be stripped of metadata and guarantees of format conversions between world-coordinates and graph-node coordinates would have to be implemented. However it is possible to convert into the inner representation format of the road simulation module. Osmosis construction has the following steps:

1. Highlight a map segment
2. Load it into a graph
3. Load the graph into a simulation

The steps are visualized on the following graphics from left to right:

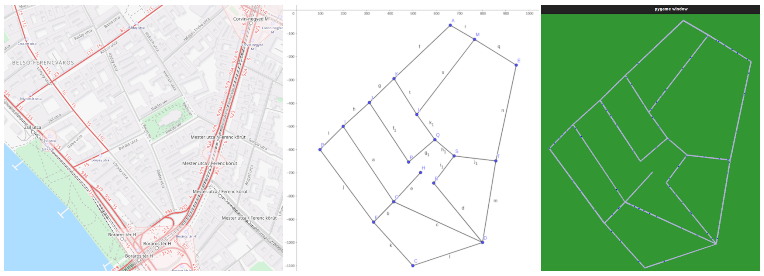


Figure 3:

4 Intelligent driver model

The intelligent driver model is a time-continuous car following model for the simulation of urban traffic. It describes the behavior of the drivers. and the positions of vehicles. The model defines 6 parameters that influence how a driver behaves:

1. v_0: The desired velocity of the vehicle.
2. T: Safe following time.
3. a: Maximum acceleration.
4. b: Comfortable deceleration.
5. δ: Acceleration exponent.
6. s_0: Minimum distance.

Using these variables and the positions of other vehicles it is possible at every time step to completely determine the position of every vehicle. Furthermore, vehicles can influence eachothers' positions, as there can be car pileups in an intersection with cars waiting for eachother. This model will consider only cars of uniform size, without taking into account other types of vehicles like vans and semitrucks. Every driver is assumed to have the same skillset e.g. safe following time, comfortable deceleration and so on. This can be a fairly good approximation of a real world driver's parameters.

5 Deep learning architecture

5.1 Action space

A starting configuration will provide the agent with a 2-way road between each graph node. From here on the goal is to add / destroy roads, lanes and intersections in order to optimize the throughput and cost of the road. Each action of the agent will take two graph nodes as a parameter and the roads will be configured accordingly. Any action for node A and B will assume a single directed edge $A \to B$ starting for A and ending in B. There are cases where semantically it would make more sense to have more or less than 2 parameters but these cases can be generalized to an action with two parameters and hence be channeled into a neural network output of the same shape and size as all other cases.

The list of actions for the discrete action space:

1. *add_lane(A, B):* Adds a single one way lane going from $A \to B$.
2. *remove_lane(A, B):* Removes a single lane going from $A \to B$.
3. *add_road(A, B):* Adds two lanes between the nodes A and B going $A \to B$ and $B \to A$. Only valid in case of nodes that don't have edges connecting them.

4. *remove_road(A, B):* Removes two lanes between nodes A and B going from $A \rightarrow B$ and $B \rightarrow A$. Only valid between nodes that have edges connecting them.

5. *add_trafficlight(A, B):* Creates a traffic light system to all roads entering the intersection of graph node B.

6. *add_roundabout(A, B):* Adds a roundabout to all roads entering the intersection of graph node B.

7. *add_righthand(A, B):* Removes current traffic light and roundabout infrastructure to create a right-hand priority intersection in graph node B.

5.2 State space

Representation

First, the plan will focus on how to represent a certain type of road between two nodes. The state between graph node A and B will have to be represented by a single number in every case. The bases that this scalar value will have to cover the number of lanes between $A \rightarrow B$ and the type of intersection in the node B.

For a state-vector element corresponding to the one-way connection between nodes A and B the possible values are as follows:

1. One-lane road between $A \rightarrow B \Rightarrow 1$.

2. Two-lane road betwen $A \rightarrow B \Rightarrow 2$.

3. Two-way road, one lane in each direction: $A \rightarrow B \Rightarrow 1$; $B \rightarrow A \Rightarrow 1$.

4. Two way road, two lanes in each direction: $A \rightarrow B \Rightarrow 2$; $B \rightarrow A \Rightarrow 2$.

5. One-lane road ending in roundabout between $A \rightarrow B \Rightarrow 11$.

6. Two-lane road ending in a traffic light junction between $A \rightarrow B \Rightarrow 22$.

The state vector keeps track of the connections in a directed fashion, storing the number of lanes from $A \rightarrow B$ and $B \rightarrow A$ in separate values. A single scalar of k means that it's a k-laned road ending in a right-hand priority intersection without any dedicated infrastructure. A scalar of value $10 < k < 20$ means that the road ends in a roundabout. A scalar of value $20 < k < 30$ means that the road ends in a traffic light based junction. The last digit always translates to the number of lanes going from $A \rightarrow B$.

Naive implementation

The state space requires precise designing and execution as there's a possibility of exponential explosion if it's represented using an all-to-all fashion. If the state space keeps track of all the incoming connections from all the nodes to all other nodes, for a graph G of nodes $N_1, N_2, \ldots, N_k$the state space will require a vector of length $k^2 - k$. There's no need to keep record of connections to and from the same node (self-loops): $N_i \to N_i = \emptyset; \forall i \in [1, k]$.

N_1	N_1	N_1	N_1	N_2	N_2	N_2	N_2	...	N_k	N_k	N_k	N_k
N_2	N_3	...	N_k	N_1	N_3	...	N_k	...	N_1	N_2	...	N_{k-1}

This gives reason to explore the possibilities of being able to shorten the state space vector by eliminating connectioins that are for sure not going to participate in the learning. The rationale behind this is that it's an errenous design decision to connect the furthest junctions with direct roads. This would mean destroying currently existing residential zoning and thereby cut the city in half.

Radius-based approach

It's easy to notice that it isn't needed to have roads connecting all intersections with all other intersections. This would make the city a convoluted mess. A valid approach to reduce the size of the state vector is to define a radius around the nodes and only store the number of lanes between them. For this a radius parameter will have to be input in order to define the possible connections before running the simulation. On the image below an example of such a circle is shown for graph node E with the radius of 150.

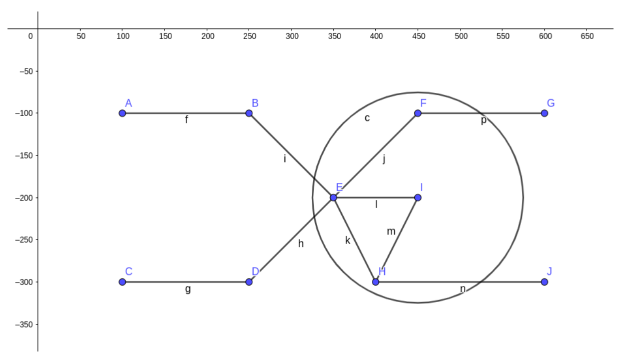


Figure 4:

It is fairly straightforward to see that this would result in a configuration where no road's length exceeds the radius of the neighborhood. This can be considered as a good design approach because it would eliminate the need to build long and straight roads through the city which are shown to be more dangerous regarding traffic accidents. The reason for this is that motorists can go with higher speeds as the road is wide and forgiving like a highway. It is empirically shown that the best way to reduce speeds is to force the drivers to obey speed limits by creating the geometry of the road in a way that enforces it. One won't go with 70-90 km/h on a street that is say 150 meters long.

The disadvantage of this method is that the state vector would be unevenly distributed between the nodes of the graph. Some of the nodes would have more elements in the vector than others resulting an unnecessary high representation. It's also possible that an outlier node gets cut off from the rest, and becomes an 'island' that is unreachable from anywhere. This happening is considered a design fail.

NN-based method

The number of tracked connections in the state vector can also be reduced if only the m nearest neighbors to a node is allowed. This would allow for a more evenly distributed state representation as for each node there's only a need to register m connections instead of $k-1$. This approach would however also limit the number of incoming connections into a junction. E.g. if the parameter is set to $m=4$ the previously shown graph would not be possible to construct and some node may be separated from all other ones, isolating them from the rest of the infrastructure. This is considered a bad resolution of the problem, but the approach can be used in combination with other approaches.

Practical implementation

A combination of the previously mentioned methods can also be taken into consideration in order to reduce the state vector size, e.g. allowing connections from a specific radius but in cases where there's not a sufficient number of possible neighboring nodes the system will have to find the m closest nodes and keep track of roads coming and going from them. It's also worth noting that combining the advantages of the two methods intelligently can yield a representation where each node has a specific, even number of connections that is significantly less than the total number of nodes in the system.

5.3 Rewarding mechanism

At each time step in the reinforcement learning agent-environment framework, the agent takes an action and as a result, the environment changes its state and returns a reward to the environment. The agent can also observe the environment through the state variable described in the previous section. This part will deal with the principles that will be reflected during sending the reward signal to the agent. The main directives like exact costs and traffic engineering perspectives will be further explored in the literature review. For now it's enough to deal with the main.

The main goal for the agent is to build a city that is livable for humans by keeping the number of lanes and roads to the minimum at all times. If this constraint was not present, the agent could simply build 8-lane highways in streets that are a few hundred meters long. This would render being a pedestrian miserable. It's also necessary to not build roads between nodes that is not necessary from a transportation perspective. If there's a reasonable detour between two endpoints, the cars should take that road.

Inter-node infrastructure

The cost of building a lane has to be taken into consideration. It is more expensive to add new lanes to existing infrastructure than to build the first lanes at the beginning. This is because the sidewalk has to be destructed and then rebuilt. Also traffic flow is slower in the time of the construction because of closed roads, leading to congestions. Out of the 7 possible actions of the agent 4 are in relation to building roads and lanes. The relationship between them is the following:

1. The cost of building and destroying is linearly dependend on the length of the segment. The length cost of a unit of lane is l_{lane}.
2. The cost of building a unit (e.g. meter) of one-way lane is: c_{lane}.
3. The cost of building a lane is taken as $b_{lane} = c_{lane} * l_{lane}$.
4. The cost of building a road is equivalent to building two lanes: $b_{road} = 2b_{lane} = 2 * c_{lane} * l_{lane}$.
5. The cost of destroying a lane is less than building a lane. The coefficient parameter is denoted δ: $0 < \delta < 1$ Then the cost of destroying is: $d_{lane} = \delta b_{lane} = \delta * c_{lane} * l_{lane}$.
6. Then the cost of destroying a road: $d_{road} = 2\delta b_{lane}$.

The necessary inputs are: c_{lane}, δ.

Intra-node infrastructure

The next section will describe the cost of building different types of junctions. The types of junctions in increasing order of cost are:

1. Right-hand intersection: it is the cheapest, however it is also the most unsafe. Cheap to build and to maintain. Also inexpensive to convert to from all other types of junctions.

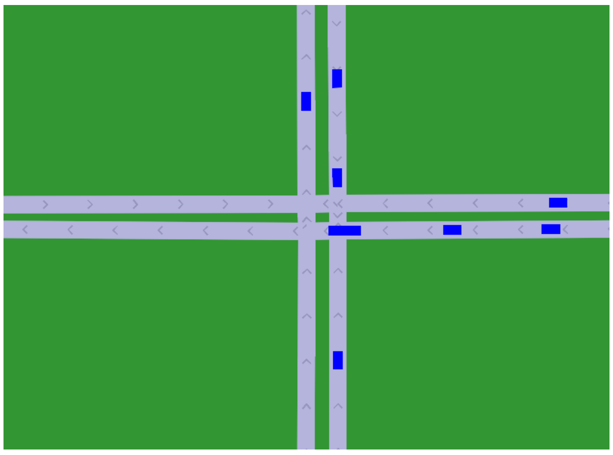


Figure 5:

2. Roundabout: a reasonable tradeoff between cost and safety, however it is difficult to destroy an intersection and build a roundabout in place of it as it requires widening the intersection.

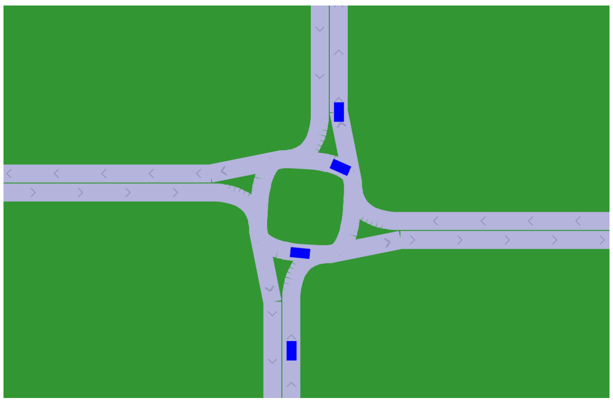
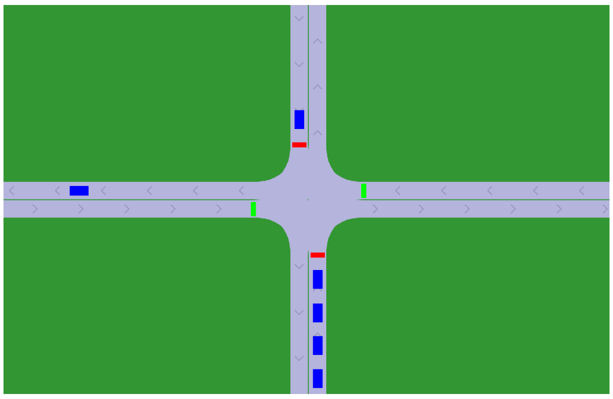
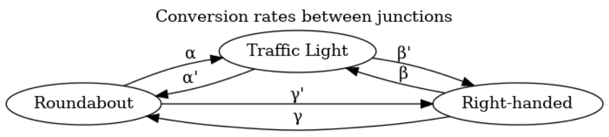


Figure 6:

3. Traffic light: the most expensive and the safest type of junction is the traffic light one as it requires dedicated infrastructure and the maintenance costs are higher than in case of all the other junctions.

Figure 7: 

As a graph node must be at all times assigned to a type of junction, and each type can be converted into any other type, the relationship between them can be defined by a triangle. The cost of converting from one to another can be parametrized and can be subject to change depending on how expensive building infrastructure is on a given terrain or economical environment. The conversion table is shown below:

Figure 8: 

The conversion will require the inputs to and from each type of intersection, as these are not necessarily functions on eachother. Some dependency can be added later but that is independent of the current high-level model. The required inputs are: $\alpha, \alpha\prime, \beta, \beta\prime, \gamma, \gamma\prime$. Each variable represents a conversion cost between two types of infrastructure. The prime version of the variable is the reverse of the same building procedure. The prime ($'$) will always reflect a *'destruction'* e.g. demolishing an expensive piece of infrastructure and building a cheaper one in place. The non-prime variable denotes the building of a more expensive piece of infrastructure from a cheaper one.

There are limiting factors to different types of junctions e.g. how many roads can it intersect. The roundabout is famous for being able to intersect five or more roads without any problem, because of the structure of it. Traffic lights can also have more roads than average coming in and out of it. For the most part right-hand intersections can handle at most four bidirectional roads. This is because human attention cannot be focused to so many places at the same time. More roads will translate to a more dangerous environment and will require highly dedicated infrastructure. It is possible, but it's rarely seen in the real world and the cost will have to be tuned accordingly.

5.4 Network architecture

The modeling structure is vastly dependent on the task that is given to it, therefore this section will only give a general overview of the inner workings of the deep learning model. The main steps of a single iteration of simulation are given below

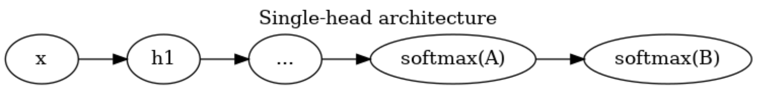
```
Algorithm 1 The modeling process
Input: a state vector $x$ that describes connetions $A \rightarrow B$ in a graph $G$.
Output: the road configuration which is optimal for travel time, cost and livability.
Repeat until stopping criterion isn't met:

  1. Agent observes state vector $x$.
  2. Agent chooses a 'from' node $A \in N_1, N_2, \dots, N_k$.
  3. Agent chooses a 'to' node $B \in N_1, N_2, \dots, N_k; B \neq A$.
  4. Agent chooses an action $a \in a_1, a_2, \dots, a_7$.
  5. The environment executes action $a$ on the connection: $A \xrightarrow{a} B$.
  6. The simulation runs to a given stopping criterion (e.g. time elapsed, number
     of cars finished).
  7. The environment passes the agent the reward signal $r$ and the modified state
     vector $x$.
  8. The agent executes a learning step based on the reward $r$.
  9. Back to step 1.
```

It's easy to observe that the machine learning model has a particularly difficult job in this case. The action it has to take in each iteration consists of multiple variables, not just a single one: $A \xrightarrow{a} B$. For this reason a highly specialized deep learning configuration is needed. The input vector is the size of the state space as described by the node connections in section 5.2 of this chapter and the number and size of the hidden layers is highly dependant on the size of the input graph, so the rest of this section will deal with the problem of predicting A, B and a. First the *'to'* and *'from'* node prediction will be taken into account. For this there are two proposed architectures, the single and the multi-headed ones.

Single-head architecture

This is a simple configuration that uses a feedforward network to predict nodes A and B. The input is the state vector, and there are an arbitrary number of hidden layers. This has been denoted as ... in the figure. In the prediction phase of the modeling the model first predicts A, then based on that prediction produces a prediction for graph node B.

Figure 9: 

Both of the predictions use softmax nodes that estimates the probability $\hat{p_k}$ that the action belongs to action class k thereby choosing an action from $a_1, a_2, \ldots, a_7$, where $k \in \{1, 2, \ldots, 7\}$. For k as the number of classes the softmax function is defined as:

$$\hat{p_k} = \sigma(s(x))_k = \frac{exp(s_k(x))}{\sum_{j=i}^{k} exp(s_j(x))}$$

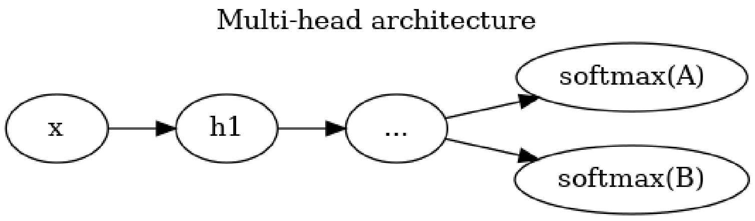
Where $s(x)$ is a vector containing the scores for each action of the current state x and $\sigma(s(x))_k$ is the estimated probability that the state vector x belongs to class k, given the scores for each class of that instance. The final prediction is the class where the probability is the highest.

$$\hat{y} = \underset{k}{argmax} \sigma(s(x))_k$$

In this case the prediction is a node from the set of all nodes. Another important criterion is that $A \neq B$. In the case where the two predictions match the agent should either run the prediction again or receive a negative reward to penalize the weight within the neural network. This research denotes a softmax unit that predicts class C as $softmax(C)$.

Multi-head architecture

Another way of predicting the starting and ending nodes is by using two heads on the same neural network. In this case the last hidden node will be in connection with both the softmax neurons:

Figure 10: 

In the same way as before, the two predictions must be different.

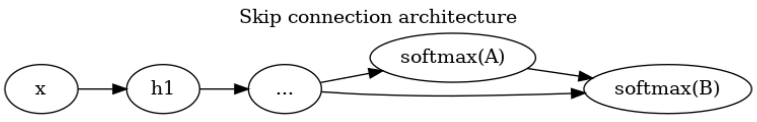
Comparison

This part is dedicated to describing the advantages and disadvantages of the two proposed network architectures. The main idea behind the single-head (SH) architecture is that the neural network has connections between the two prediction nodes, so the B node will be dependant on the A node. This is very useful in learning what junctions are close to eachother and are relevant in building the infrastructure and also has a built-in way of learning that the two nodes should be different. However in the SH configuration the B node has no prior information about the state vector as it has already been accumulated in the softmax node. The output of the softmax node is a single class, therefore the B node should be only predicted based on that integer number.

The multi-head architecture (MH) can counter this problem: both prediction nodes are in connection with the last hidden state vector, therefore they both have knowledge of the information extracted. This leads to a problem where the two nodes have no connection to eachother, hence there is no way to actively penalize predictions where the two outputs match.

A practical realization of this problem is to introduce a skip connection between the last hidden unit and the B node. Using this method the *'to'* node would acquire knowledge about the *'from'* node and the information extracted by the neural network. A high-level architectural diagram of this approach is shown below:

Figure 11:



Action-prediction architecture

The last task of the network is to predict what action to take between the nodes A and B. The action denotes if the agent will build or destroy lanes or change the type of the intersection. For this purpose it's necessary that the network has knowledge about the predictions made on the two softmax nodes defined in the previous section. It's also essential that the network can access the state vector transformed by the hidden units and the current state of the graph connection, e.g. how many lanes it currently has and what type of intersection is currently built there. The proposed architecture is as follows:

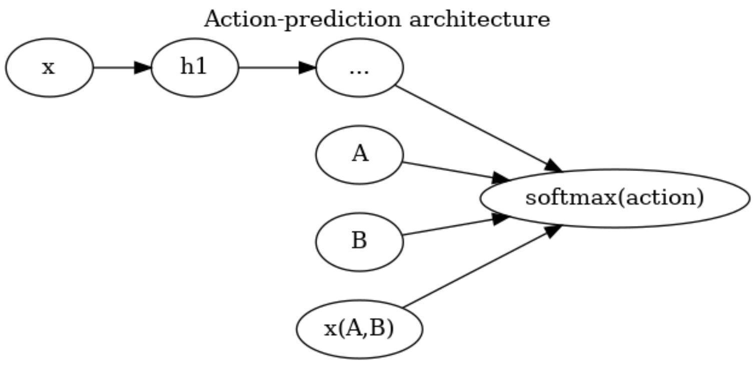


Figure 12:

Where $x(A, B)$ denotes the current state of the graph connection between A, B.

Practical implementation

If the architecture was made by the method defined before, it would require two separate neural networks, one for predicting A, B and one for a. This would be errenous in practice as two complete neural networks would have to be trained separately. This approach is redundant and can lead to information being lost and increased computation times. In practice it's beneficial to have a single neural network that trains, and can be able to predict 3 target variables in a single pass. This is also a good approach in the sense that there are dependencies between the variables: $A \rightarrowtriangle B$; $A \rightarrowtriangle a$; $B \rightarrowtriangle a$; $x(A, B) \rightarrowtriangle a$. These dependencies can be brought into the same neural network using the architectural design below:

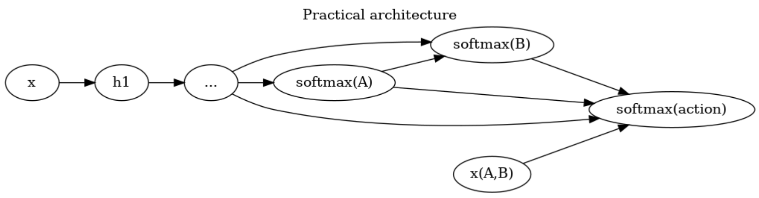


Figure 13:

Where each softmax node serves as a prediction thereby defining $A \xrightarrow{a} B$. This architecture encompasses every dependency between the softmax units under the same scope while being able to provide every prediction. Deciding if the input unit $x(A, B)$ can be left out of the network is up to experimentation in the following sections.