TSP using Reinforcement Learning

Q- Learning method with Struct2vec Graph Embeddings

Structure

1. Description of the datasets
2. Description of a the Travelling Salesman Problem
3. Description of the specific algorithm
4. Q Learning Model
5. Graph Embedding
6. Architectures: visual graphs
7. Architectures Hyper-parameters: table
8. Hyper-parameter Tuning: Choices, rationale, observed impact on the model
9. Time vs Episode and loss

Description of Datasets

For training the data we use a synthetic dataset with the help of the numpy random function.

Using numpy random we build a Coordinate matrix of latitude and longitude values that represent the various nodes of the graph and based on this Coordinate Matrix we build a Distance Matrix that will hold the distance values between the various nodes of the Coordinate matrix.

We set the Episode values to 4000 which will train the model on 4000 different random generated graphs.

For testing, we have a test that can be run on test data set or a real dataset of cities. The real dataset has the following specifics: 1) 22 coordinates of Mediterranean cities 2) 52 coordinates of Berlin 3) 100 coordinates of real city(ref: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/)

Travelling Salesman Problem

We are trying to solve the travelling salesman problem using the Deep Reinforcement learning approach that will be using the Q-learning algorithm to solve the problem.

We will be using the unsupervised version on learning for this approach.

The problem description involves solving the TSP problem of visiting every node in the graph in the shortest possible distance with a constraint of fuel. When the mileage restriction occurs there will a search for the nearest fuel filling station which will be added to the tour to complete its path.

Description of the specific Algorithm

We consider that the agent that we are building will have the full information of the environment and the states corresponding to its environment.

This is achieved by having a state in the graph that will hold the values of the coordinates matrix in place, the distance matrix that will be produced out of this coordinate matrix and the set of nodes that are already been visited. This makes the environment fully observable and trivial.

We will be using this state representation in the build of our Q-learning network.

Q-learning model

We try to develop a function Q(s,a) that will help choose the specific action for a particular state and return the rewards that would be obtained on doing this at the end of the episode.

Performing this for several episode we would gather a good number of rewards and since it is a neural net we can differentiate based on the parameter of the function and use it to build a gradient of errors.

Using this gradient values we can improve our parameters of the entire function.

Q-learning model continued

The basic approach of the function is to enable running of this function for several episodes and improve our parameters, on each run the algorithm will be able to choose its next best action using a greedy approach to help improve the functions parameters.

Occasionally based on the epsilon value the greedy approach would be replaced by a exploration of the graph randomly to help get out of situation that will lead to a local optimal solution.

We also use the experience as a tuple to store values that help us in model learning. The experience will store the current state, the next action, the observer reward, and the next state that we can reach.

Graph Embedding

Used the struct2vec approach for the graph embedding process. In order to make our Q function be learning able over different size of graphs and avoid the need of constructing a new model every time we choose a different graph size we have.

We have used the approach suggested in the paper : "Khalil, Elias, et al. "Learning combinatorial optimization algorithms over graphs." *NeurIPS*. 2017."

The approach tries to figure out a vector representation of each node.

Graph Embedding continued

We try to build a graph using the belief propagation that will send messages recursively to the other nodes along the graph that will help build the structure.

$$\mu_v^{(t+1)} = relu(\theta_1 x_v + \theta_2 \sum_{u=N(v)} \mu_u^{(t)} + \theta_3 \sum_{u=N(v)} relu(\theta_4 w(v,u)))$$

μ is the node embedding created for each node v, iterated over t times .

1. Term 1 : x represents the state of each node. Is a vector that has variables for capturing each variable defined in state
2. Term 2 : returns node embeddings for each of the neighbors(N) of v.
3. Term 3: edge weights of the incoming edges connecting to the current node v.

Graph Embedding continued

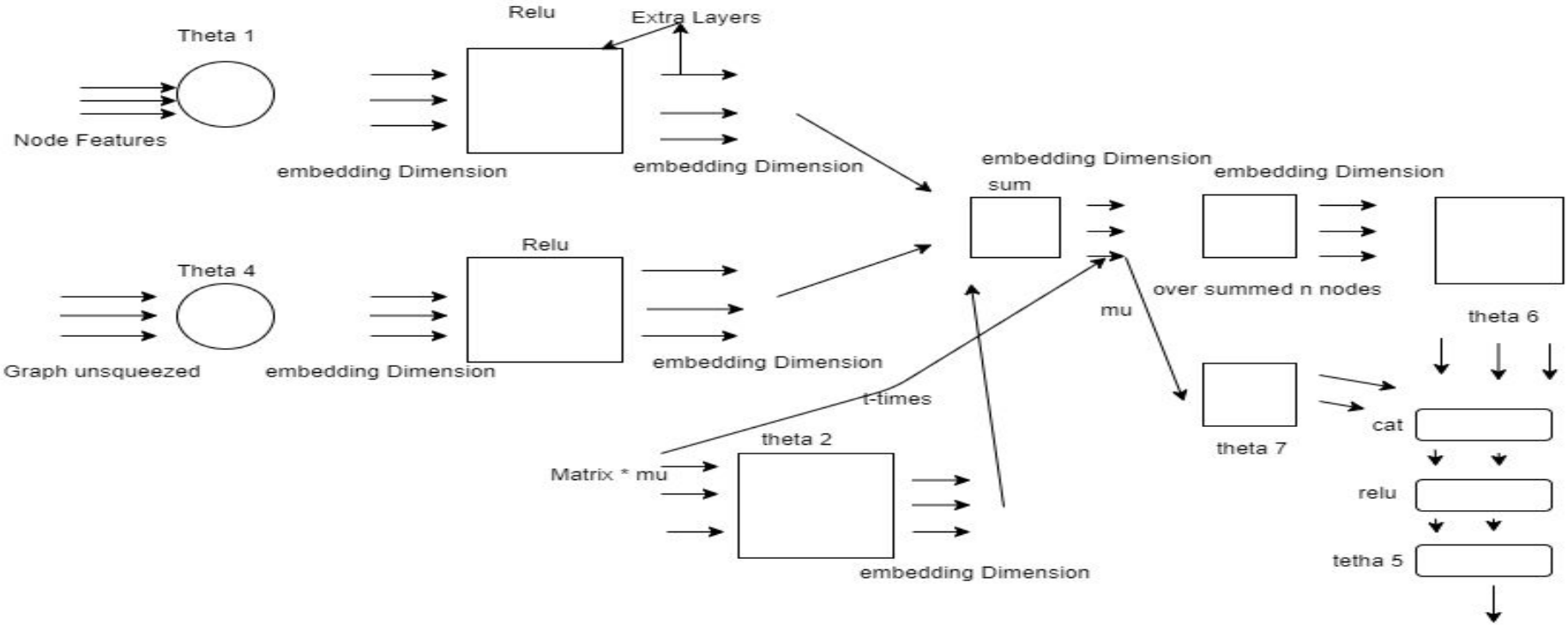
$$Q(s, v; \Theta) = \theta_5^T([\theta_6 \sum_{u=v} \mu_u^{(T)}, \theta_7 \mu_v^{(T)}])$$

Q() function, for Θ ,which is the set of parameters θ

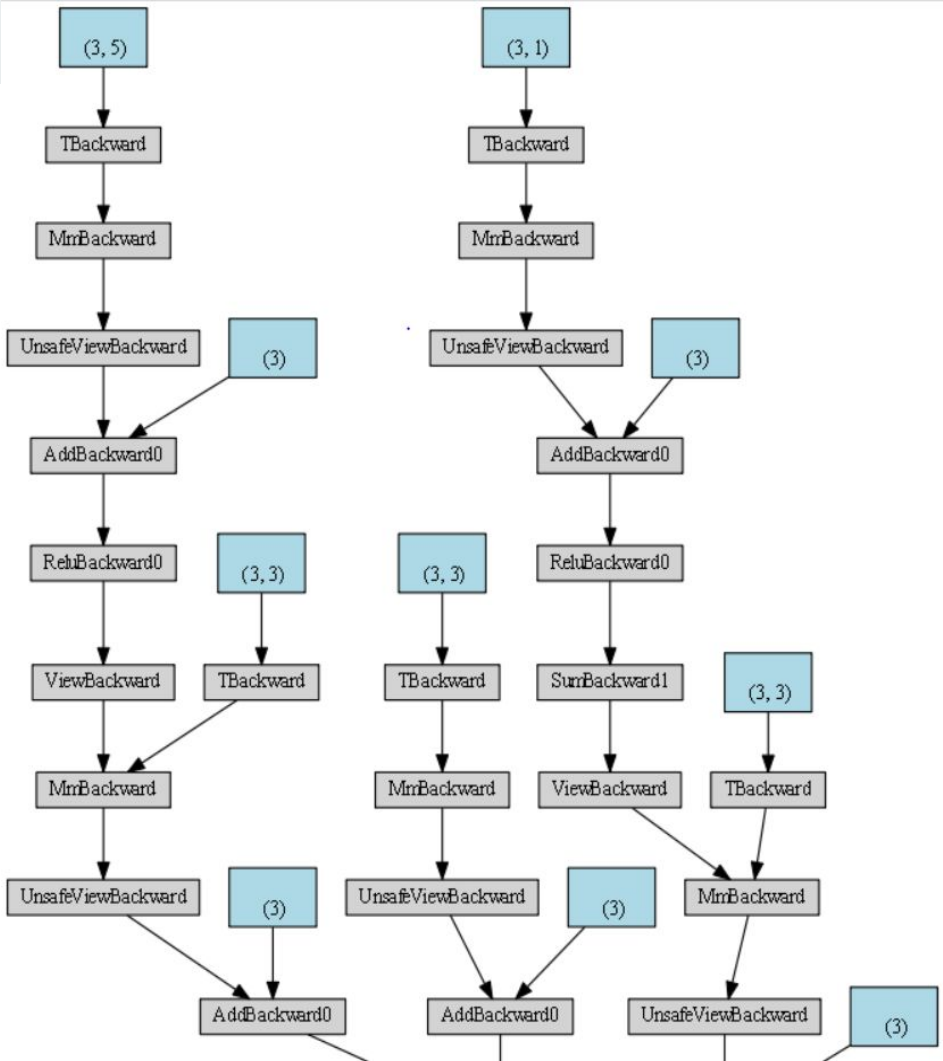
v which is the action represents the next node to be visited

Q() value is calculated by merging local embedding(u) and aggregation of embeddings of all nodes.

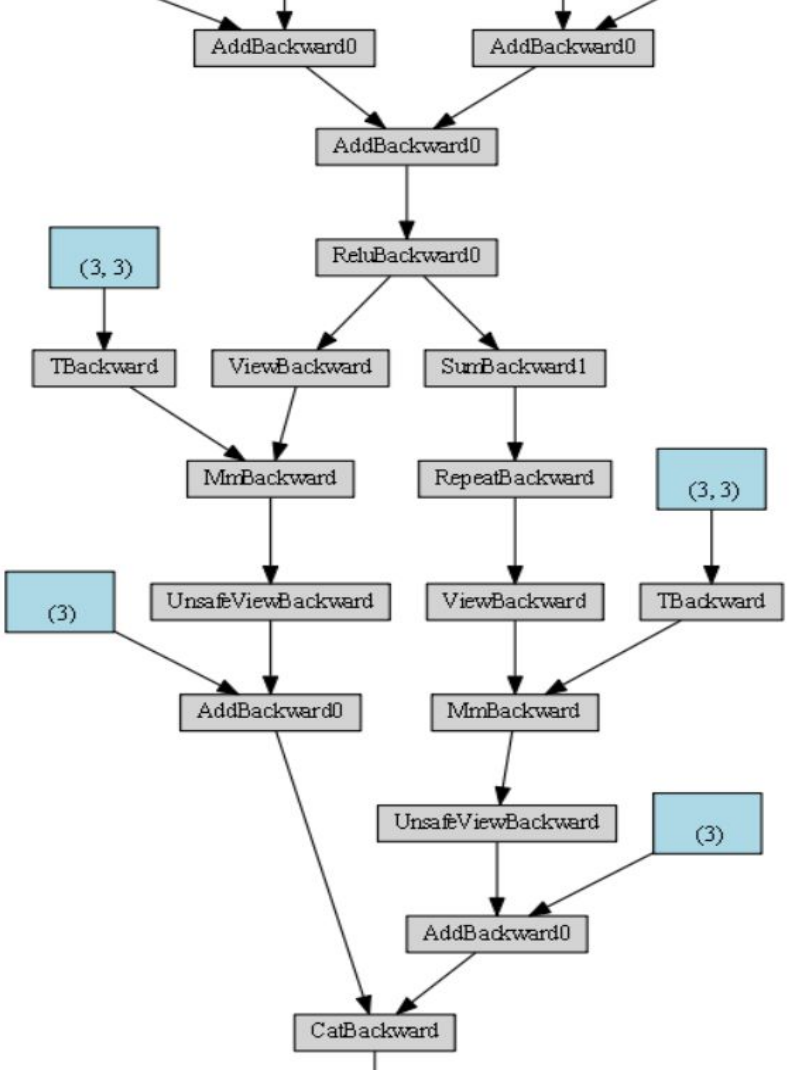
Architectural Visual Graphs



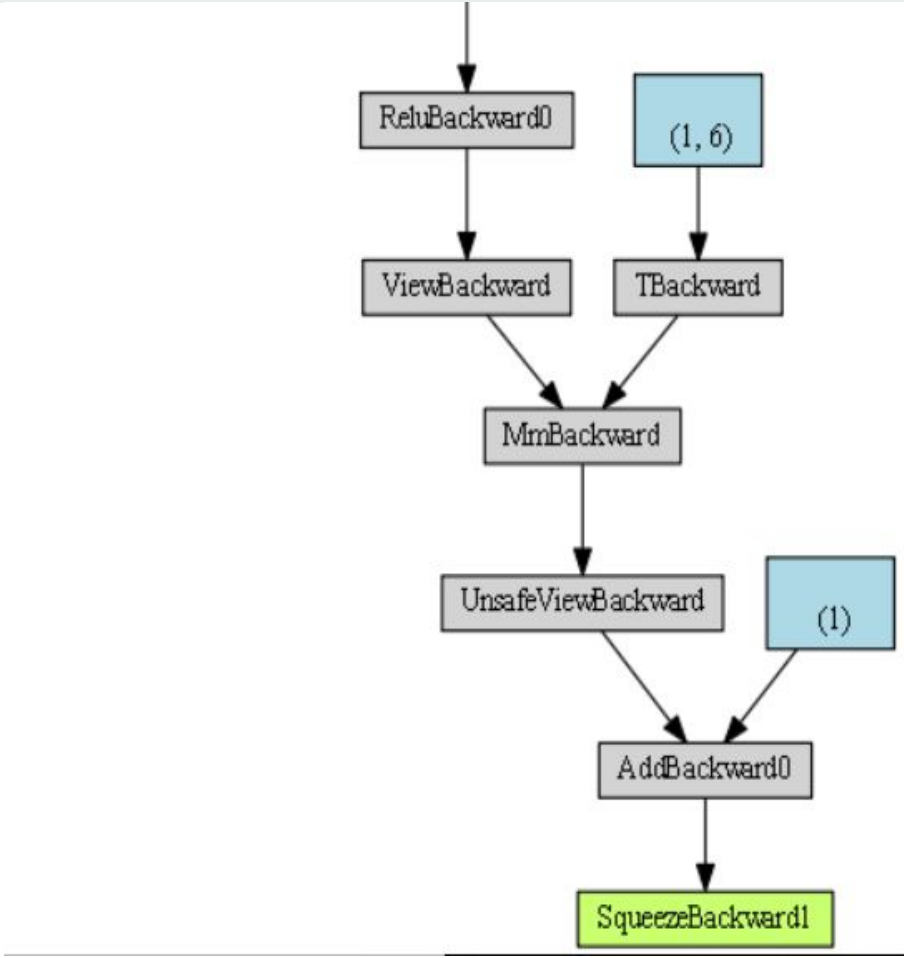
Architecture Visual Graphs



Architecture Visual Graphs



Architecture Visual Graphs



Hyper Parameters

The Various hyper parameters used in the algorithm are :

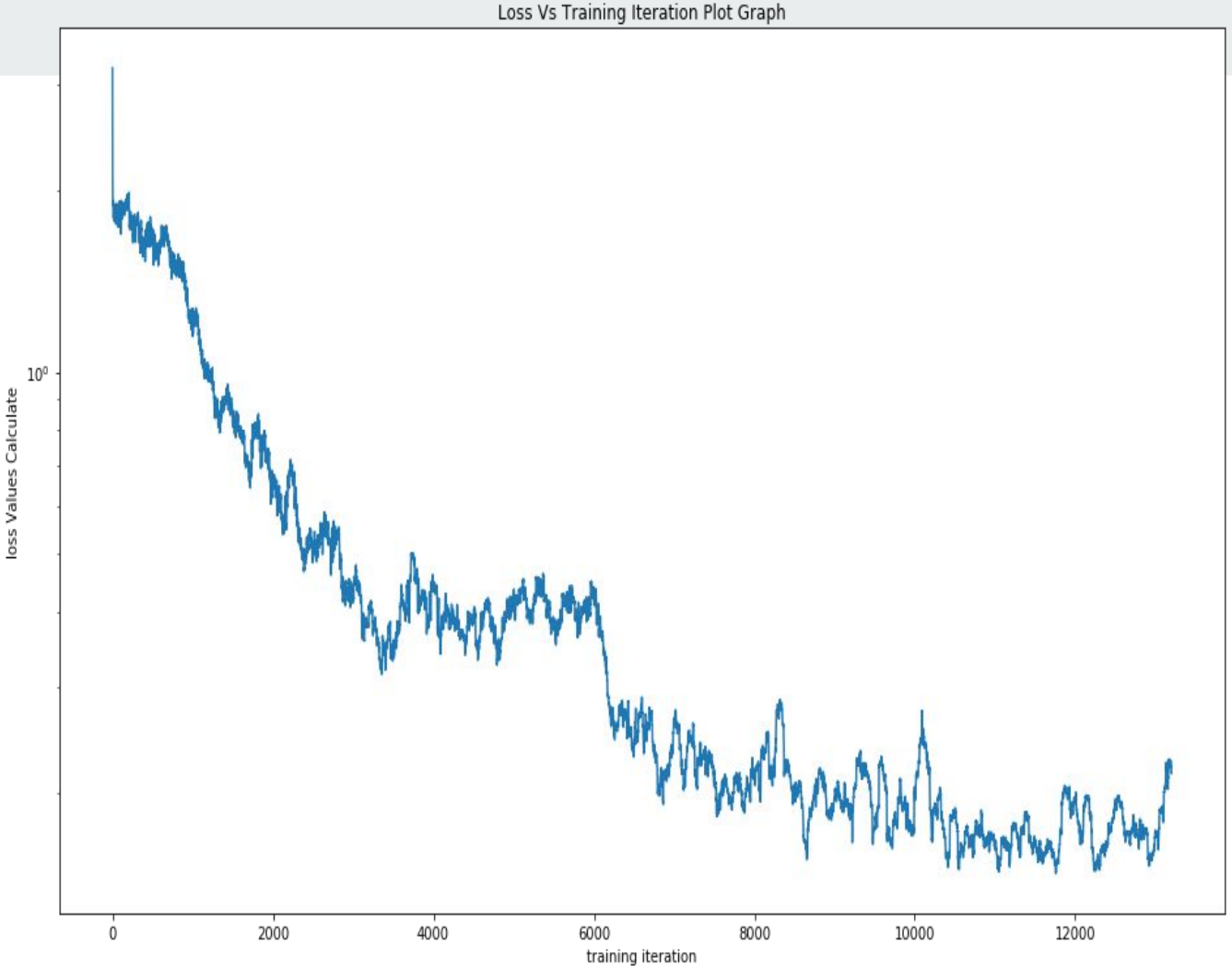
1) SEED = 1 -> set s the seed values for the random function
2) GRAPH_NODE = 10 -> number of nodes in the graph
3) EMBEDDIM = 5 -> dimension of graph embedding
4) EPISODES = 4000 -> iteration of the function run
5) MEMORY = 10000 -> memory function to store the experiences
6) QL_STEPS = 2 -> number of steps in our Q-learning
7) GAMMAVAL = 0.9 #gammavals
8) INIT_LR = 5e-3 -> The initial Learning rate
9) LR_DECAY_RATE = 1. - 2e-5 -> the decay rate
10) MIN_EPSILON = 0.1 -> min value epsilon take
11) EPSILON_DECAY_RATE = 6e-4 -> epsilon decay rate
12) NAME = './models' -> folder to store checkpoint models.

Learning rate and decay rate

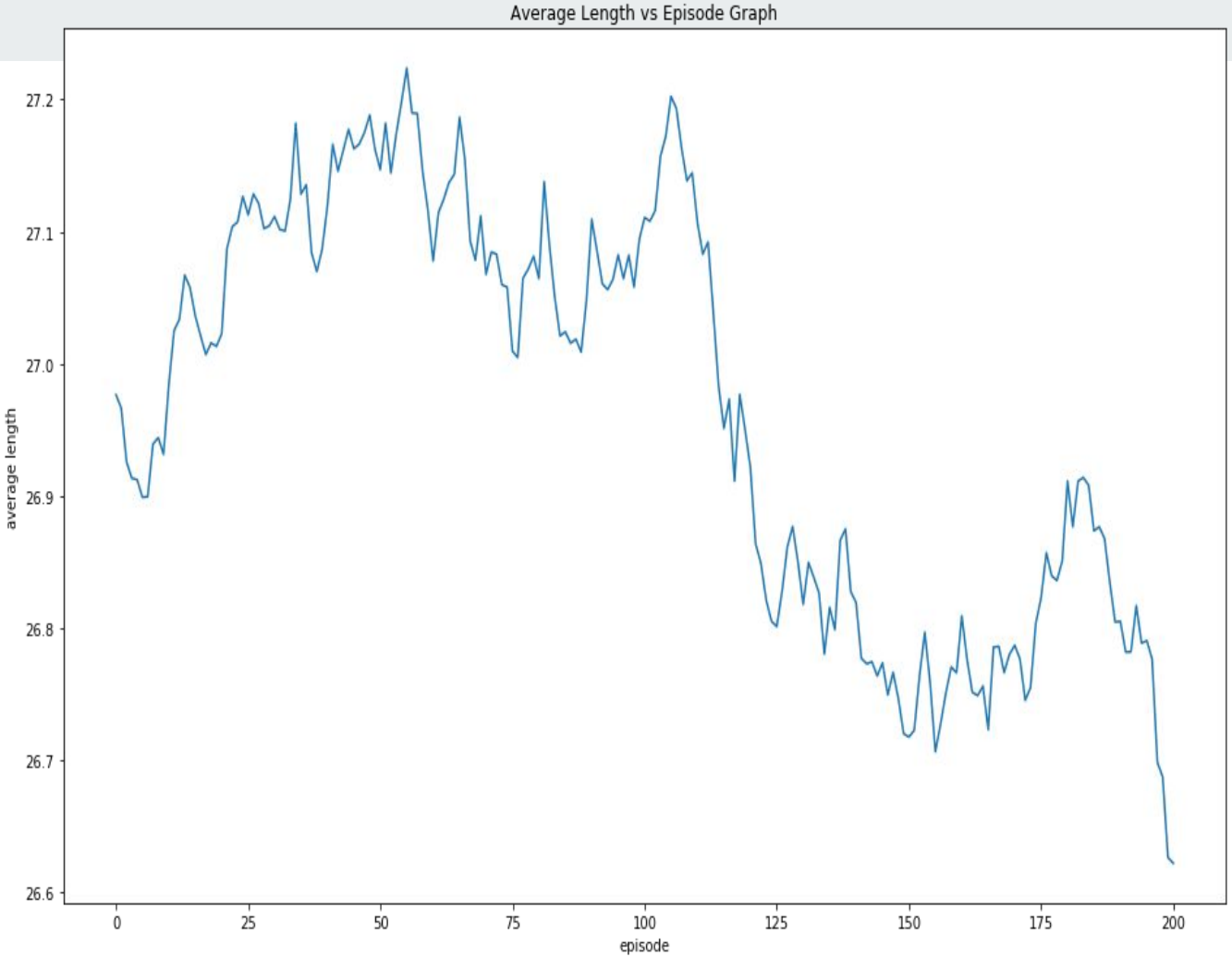
The learning rate was set at 5e-3. This was chosen to make sure the number of episodes required was not being too large and also the convergence did not happen quickly.

The decay rate was set at 1. - 2e-5, this was chosen as to let the model take large initial step and once the convergence starts to let it take smaller steps in the process. Impacted in reducing the loss in the function.

Results


Loss Vs Training Iteration Plot Graph
loss Values Calculate
10^0
0
2000
4000
6000
8000
10000
12000
training iteration

Results


Average Length vs Episode Graph
average length
episode
27.2
27.1
27.0
26.9
26.8
26.7
26.6
0
25
50
75
100
125
150
175
200

References

[1]https://medium.com/unit8-machine-learning-publication/routing-traveling-salesmen-on-random-graphs-using-reinforcement-learning-in-pytorch-7378e4814980

[2]https://www.sharpsightlabs.com/blog/numpy-random-seed

[3]https://stackoverflow.com/questions/52288635/how-to-use-torch-stack-function

[4]https://jamesmccaffrey.wordpress.com/2019/07/02/the-pytorch-view-reshape-squeeze-and-flatten-functions