

Heuristic Dynamic Programming Applied to a Transportation Resource Allocation Problem

Author: Ryan J. Meuth

Abstract: Adaptive Critic designs are particularly useful for complex problems, as they do not suffer from the ‘curse of dimensionality’ that reinforcement learning methods, such as Q-learning, suffer from. Here, heuristic dynamic programming is applied to a resource allocation problem modeled as a transportation system. Results are compared with a nearest resource policy, Q-Learning, and a hybrid method that removes the action network from the HDP architecture.

Introduction

As the complexity of optimization problems increase, the size of the state and action spaces increase exponentially, making many approximate dynamic programming methods intractable. Adaptive Critic designs seek to overcome this barrier by using neural networks as value and policy approximators. Since the parameters of neural networks increase linearly with the dimensionality of their inputs, this makes them highly desirable for use in high-dimensional problems. Also, neural networks are able to use continuous state variables, a feature that is not present in most approximate dynamic techniques, which enables the use of state variables that are rich with information.

Adaptive critic designs take advantage of the properties of two cross-training neural networks, an action network that approximates the best policy for action, and a critic network that approximates the value of an action in a given state. The two networks are coupled so that the rewards acquired by taking actions in a state can be used to update both the critic and action networks. In this project, the Heuristic Dynamic Programming method was used. The Heuristic Dynamic Programming method is a powerful tool for both online and offline control optimization. Approximate dynamic programming techniques store a value estimate for each state, and use this value approximation to construct a policy. This works well for low-dimensional problems where each state can be visited a number of times, but for complex problems the ADP methods suffer from memory and time requirements that increase exponentially. However, Neurodynamic programming techniques can take advantage of some limited generalization so that when a value estimate for a state is learned by the approximator, similar states will have similar estimates, so the approximator does not need to visit every state to make a value estimate.

A good example of a high-dimensional optimization task is that of a transportation resource allocation task. In this task a number of cities randomly generate loads with random destinations. For each load generated a truck must be allocated so that each load is serviced in the least amount of time. This is a challenging problem, as choosing the nearest available truck may not be the best policy, as the load's destination may be far away, and the truck may be needed to handle the loads of a nearby, frequently generating city.

Research Methods

For this project, 4 methods were implemented, a fixed policy method that was constructed by hand, a dynamic version of the Q-learning algorithm, a Heuristic Dynamic Programming method, and a Hybrid Neuro-dynamic programming method. The fixed policy method simply finds and allocates the nearest available truck to service each load that is generated. The performance of this policy is used as a baseline by which to evaluate the other methods.

The dynamic Q-learning algorithm uses a hash-table to hold the value estimates for each state and action. Each state encountered is used to generate a key value that becomes the index for the value estimate of actions under the state. If a state is encountered whose key does not yet appear in the hash table, the state is given a default set of values (usually high, to promote exploration) and is entered into the hash table. A disadvantage of this method is that the state representation must take on an integer form, which limits the amount of useful information. Also, the default state value causes the algorithm to spend much of its time exploring states that are not optimal, leading to slow convergence. During learning, the Q learning algorithm uses an 'e'-soft policy, meaning that the algorithm will choose a random action with a probability of 'e' for each state, otherwise the algorithm will follow the most valuable action. During evaluation, the method constructs a static policy from the most valuable action in each state.

The Heuristic Dynamic Programming method (HDP) uses two coupled neural networks to approximate both the value of states, and the policy of action for that state. The HDP architecture is illustrated in Figure 1.

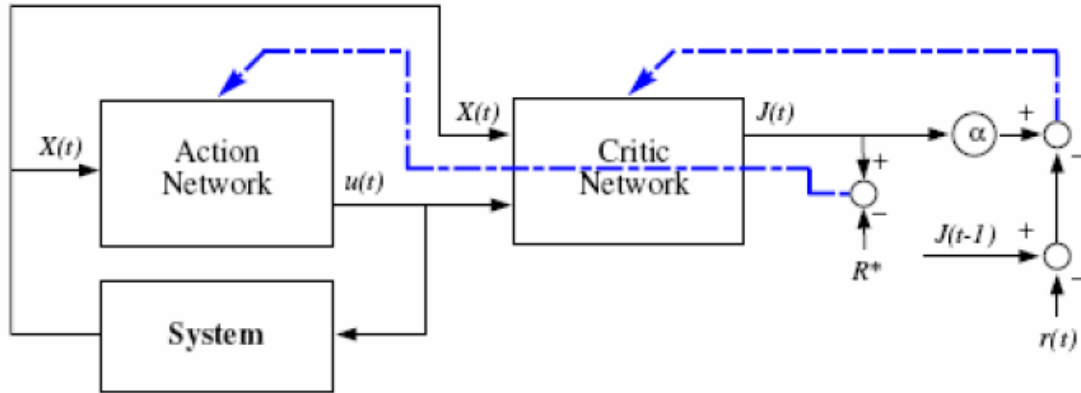


Figure 1. HDP Architecture

In Figure 1 we can see HDP as an online learning technique, but it can also be trained off-line by replacing the system with a model. We can see that the system produces a state, given as $X(t)$, which is fed-forward through the action network to produce the trial action given as $u(t)$, which is taken in the system, and elicits a reward, $r(t)$ and a new state. The action is then combined with the current state and fed forward through the critic network, which results in an estimate of the state-action value, also known as the cost to go function. To train the critic, the cost to go estimate is discounted by 'alpha' and the critic error is calculated from the previous state value and the current reward according to the Bellman equation. This critic error is used to adjust the weights of the critic network using error back propagation. To train the action network, the state and action are fed-forward to produce a fresh state-action value using the latest critic weights. The difference between this $J(t)$ and the ideal reward, R^* , is back-propagated through the critic to find the error due to the action network's output. This error is then used to update the action network weights using error back-propagation. In this way, the action and critic networks update each other to approximate both the optimal policy and the value function.

Since the available actions in this problem are discrete, a Hybrid neuro-dynamic programming (NDP) method was implemented. The Hybrid NDP method uses the same critic network as the HDP method, but it does not use an action network. Instead, the policy is determined by feeding forward all of a state's available actions through a critic network, and taking the action that results in the lowest cost-to-go. The critic network is updated by the same methods described in HDP.

Method Evaluation

To evaluate the performance of these different optimization methods, a simple transportation simulation was constructed consisting of a number of cities, C , uniformly distributed across the simulation space. Each unit in the simulation space represents 1 mile. Each city generates loads with an independent distribution, as time evolves. A number of trucks, L , are capable of providing transportation services for loads. Once allocated, trucks simply follow a straight-line path to the city where the load was generated, acquire the load, and then follow a straight-line path to the destination city. Allocations are able to be made only once during a time-step. Each time-step represents 1 minute of simulation time. Trucks have a fixed velocity of 1 simulation space unit per time-step, which is effectively 60 miles per hour. Though simple, this model is able to capture many of the nuances of the transportation problem.

The state vector, shown in Figure 2, was implemented as the status of each city, where a 1 indicates that a load is ready to be acquired at that city. The state vector also includes the source and destination of the current load, as well as the distances of available trucks to the source city. The vector was normalized to the range of $[0,1)$ to prevent network saturation. In the HDP method, the Actor network was constructed with a linear input, linear hidden layer, logarithmic sigmoid output layer architecture. The hidden layer of the Actor network had $2*n+1$ nodes where n is the width of the state vector. The output layer of the actor network was set at L , producing a choice vector. The action was determined by taking the arg max of the Actor output vector. The Actor network's learning rate was set at 0.01. The ideal reward for the HDP method was set to 0, so that the actor network would tend toward actions that minimized service time.

For both the HDP method and the Hybrid method the critic network was constructed with a linear input layer, logarithmic sigmoid hidden layer, and linear output layer. The Critic network's learning rate was set at 0.002, and the update discount factor was set at 0.01. Each method, except for the fixed policy, was evaluated for 2000 epochs, each epoch being a run through 10,000 minutes of simulation time, or roughly 1 week, after which the simulation was reset. For both the HDP and Hybrid methods, the critic was trained for 500 epochs under a random action policy. Under HDP, during the remaining 1500 epochs the critic continued to train, while the policy was generated by the action network, which was also training.

Under the Hybrid method, the critic also trained during the remaining epochs, but the policy was generated by the greedy actor.

$$\begin{bmatrix} C_L \\ L_{src} \\ L_{dest} \\ T_{dist(1)} \\ T_{dist(2)} \\ T_{dist(3)} \\ \vdots \\ T_{dist(n)} \end{bmatrix}$$

Figure 2. State Vector Construction.

Results

Under the Fixed Policy a 62.06 minute average service time was achieved. The Q-Learning method performed much poorer, with an average service time of 88.43minutes. Figure 3 shows the HDP critic training error, while Figure 4 shows the HDP performance. The HDP method achieved a 56.33 minute average service time. Figures 5 and 6 show the critic error and performance of the Hybrid NDP method. The Hybrid NDP method achieved a 55.28 minute average service time, making it the best performing method.

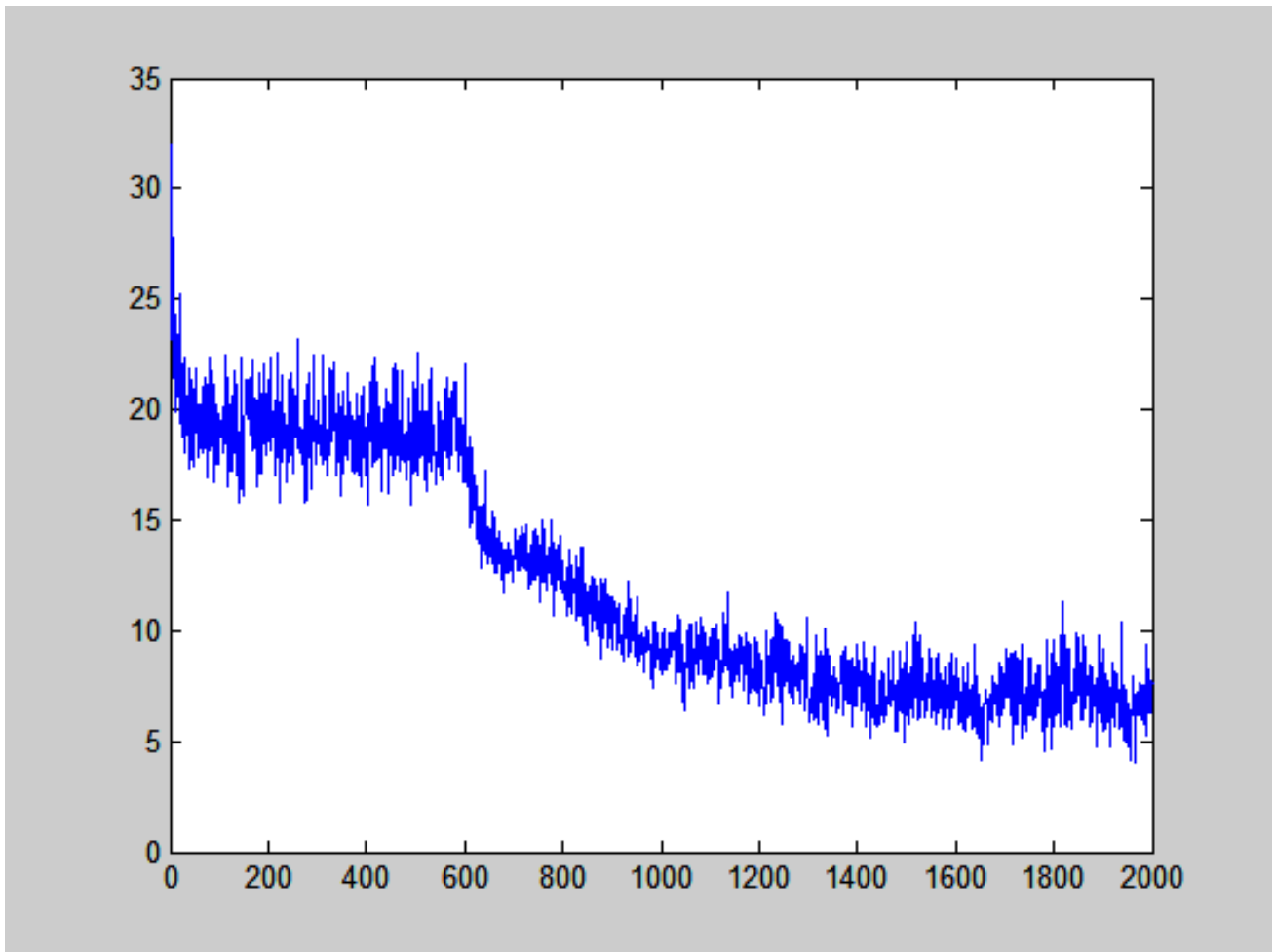


Figure 3. HDP Critic Training, Average Error per Epoch.

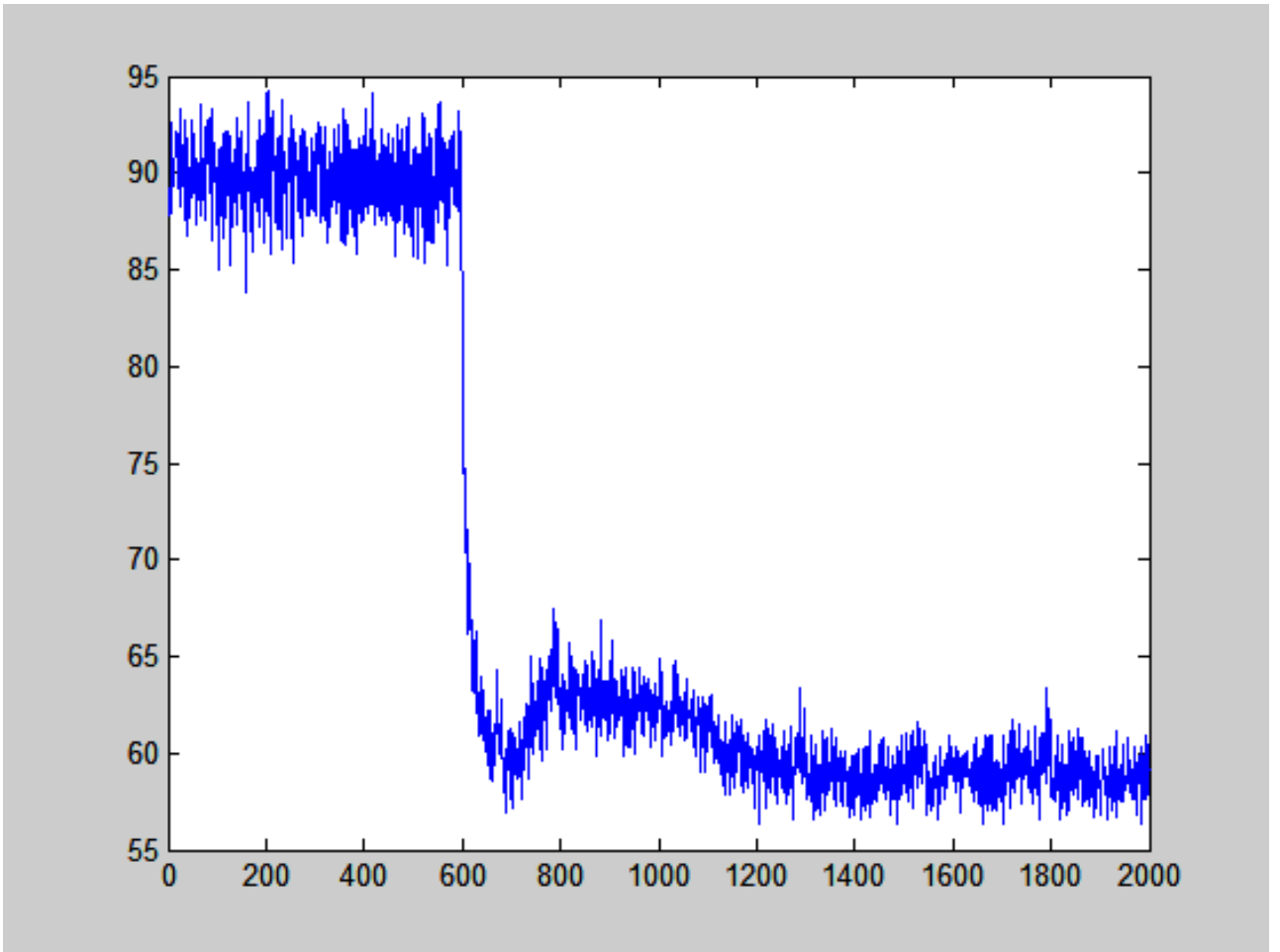


Figure 4. HDP Performance

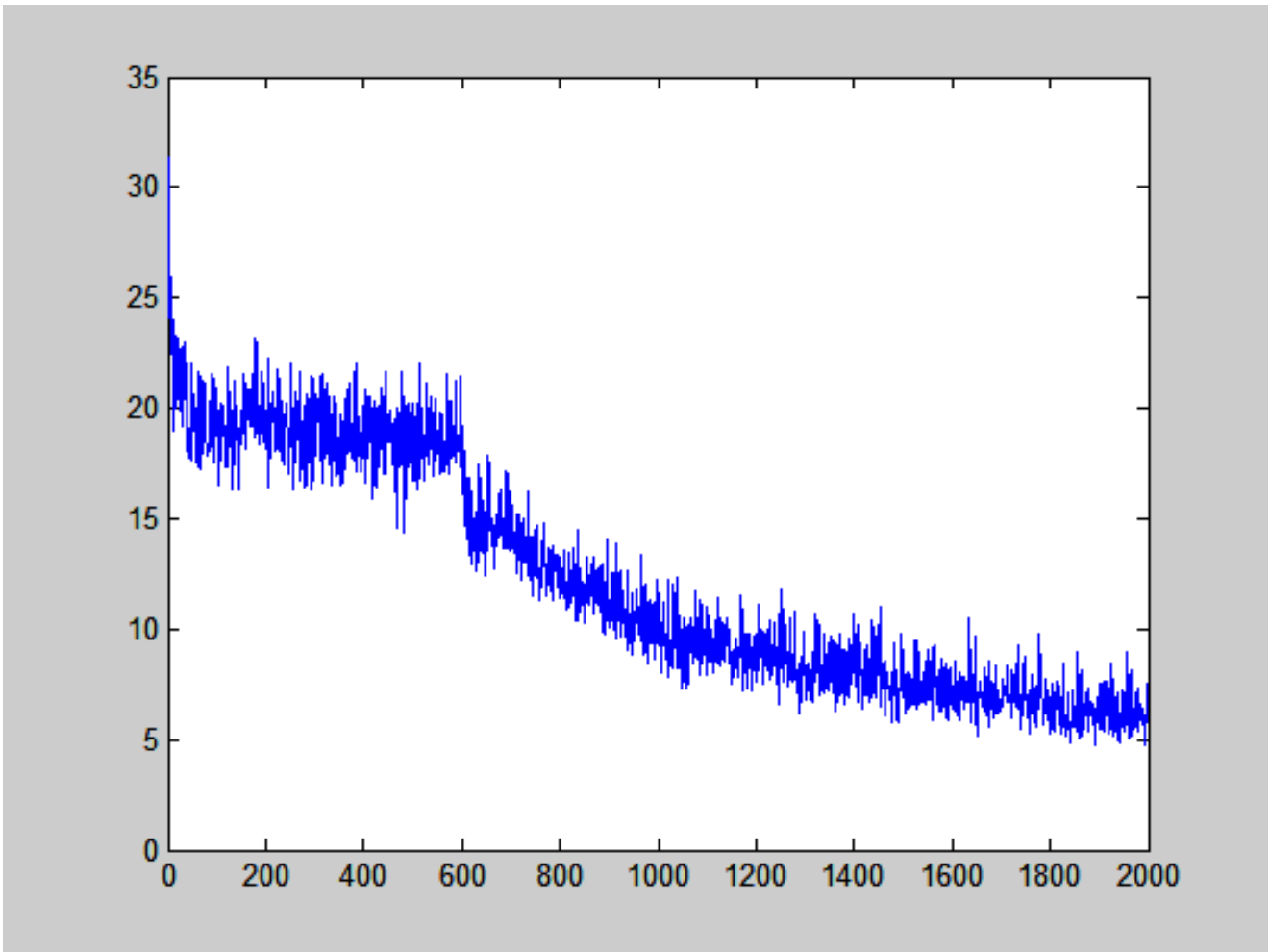


Figure 5. Hybrid NDP Critic Training, Error vs. Epoch.

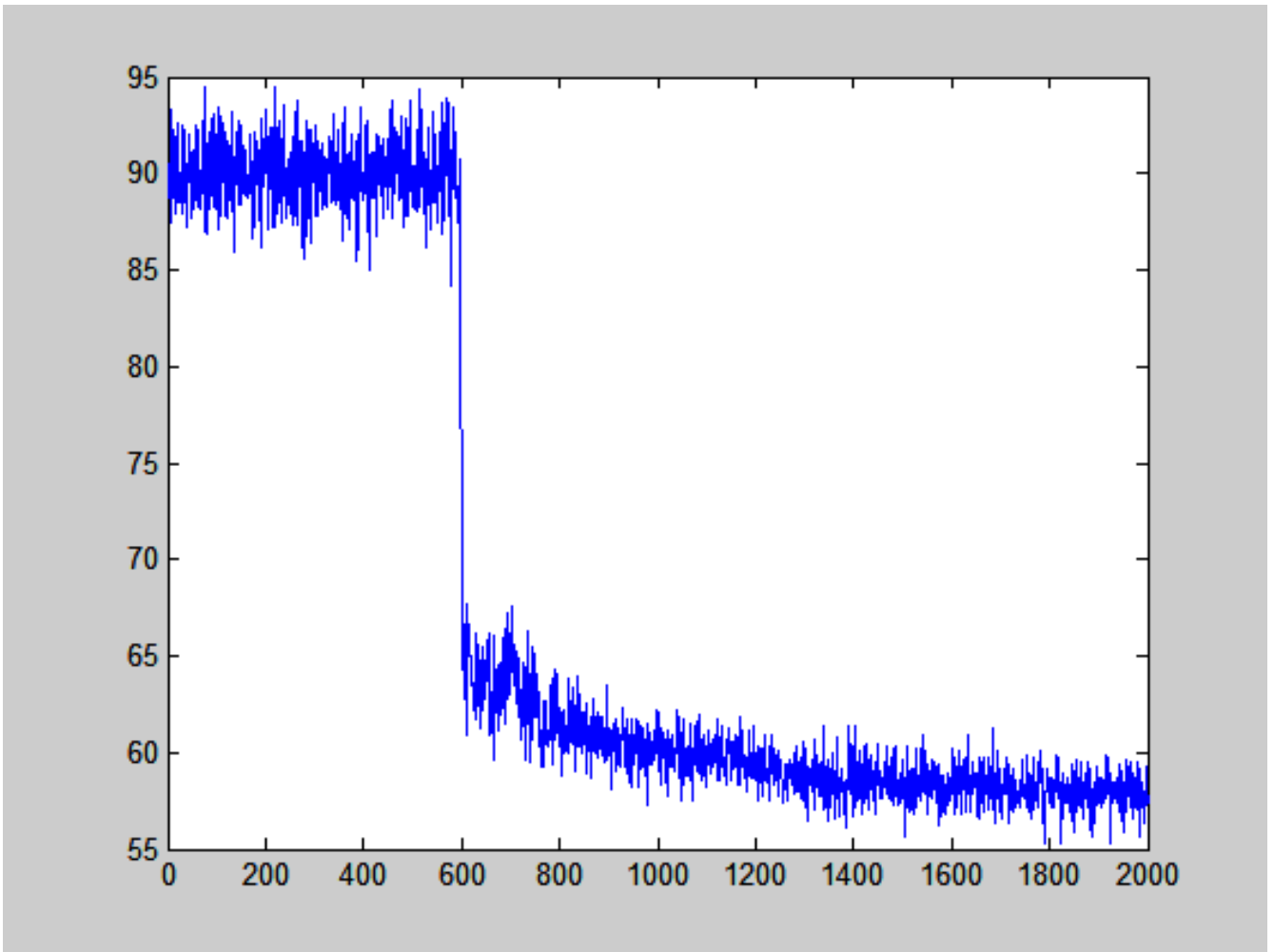


Figure 6. Hybrid NDP Performance, Average service time per epoch.

Discussion of Results

We can see in both figures 3 and 5 that during the critic training phase, epochs 0 through 600, that the critic error decreases sharply and converges, suggesting to training has stagnated. Then, at epoch 600, when both methods switched to non-random policies, the training error drops again as fewer random states are introduced, and the policy begins to converge. We can see in the second half of the critic training that the HDP method seems to converge, while the Hybrid NDP method appears to be able handle additional training. Figures 4 and 6 show the performance of the two methods per epoch. We can see that in both methods there is a sharp improvement over the random policy at epoch 600, and both methods continue to improve over the length of the remaining evaluation period.

Conclusion

We can see from the results that while Q-learning is nearly useless on this problem, and the fixed policy performed well, the HDP and NDP methods both performed very well. However these latter two methods did not perform significantly differently to warrant a conclusion about which is better than the other.

Future Work

While most objectives were achieved, namely the implementation of Neuro-dynamic and adaptive critic methods in a high dimension resource allocation problem, much remains to be explored. Future work can focus on performance of these methods on different scales of problems, as well as alternate methods of critic and actor training. However, these avenues were not able to be explored before reporting time due to time constraints.

References

- [1] - W. B. Powell, B. Van Roy, Approximate Dynamic Programming for High – Dimensional Resource Allocation Problems. Handbook of Learning and Approximate Dynamic Programming, Ch. 10, pp. 261-280, 2004.
- [2] - G. Godfrey and W. B. Powell, An adaptive, dynamic programming algorithm for stochastic resource allocation problems. Transportation Science, vol. 36, no. 1, pp. 21-39, 2002.
- [3] – R. Cheung and W. B. Powell, An algorithm for Multistage dynamic networks with random arc capacities, with an application to dynamic fleet management, Operations Research, vol. 44, no. 6, pp. 951-963, 1996.
- [4] - S. Ferrari, R. F. Stengel, Model-Based Adaptive Critic Designs, Handbook of Learning and Approximate Dynamic Programming, Ch. 3, pp. 65 - 95, 2004.
- [5] – Sutton, Richard S., Barto, Andrew, Reinforcement Learning: An Introduction, MIT Press, Cambridge Mass, pp. 151, 184. 1996.

Appendix A – Code Index

TranSim.m – Main Simulation Code, generates city and load sets, simulates interactions, and handles method operation.

makeQController.m – Q-Learning Initialization function

MakeKey.m – Generates Key from State for Q-Learning.

UpdateQ.m – Updates Q-Learning structure

SetQValue.m – Sets Value for a State in the Q-Learning Method

GetQAction.m – Gets e-soft action for a given state from the Q-Learning structure.

GetQValue.m – Gets value of a given state and action from the Q-Learning Structure.

makeHDPController.m – Initializes HDP Controller structure. Used for both HDP and Hybrid NDP methods.

HDPActorFFD.m – Feeds forward action network, given a state in the HDP structure.

HDPCriticFFD.m – Feeds forward critic network, given a state and action in the HDP structure.

HDPTrainActor.m – Action network training function.

HDPTrainCritic.m - Critic Network training function.

HDPTrainHybridCritic.m – Hybrid Network critic training function. Operates similarly to HDPTrainCritic, but leaves out action network.