Scalable Reinforcement Learning Methods for Demand Response

Demand Response
- ❖ Initiatives taken by the utility to modify consumer usage during stressed grid conditions
- ❖ Pricing plans offered are dynamic as opposed to fixed slabs – e.g. ToU, CPP, RT
- ❖ Dynamic pricing can reflect the real-time transactional behavior in Energy Markets
- ❖ Reducing peak load demand can result in: **B**etter operational efficiency, **L**ower cost of production, **D**ecrease in the commitment of high-polluting generators
- ❖ 2-way interaction in which responsive consumer benefits from lower tariffs
- ❖ Industrial plants, large-scale businesses, aggregators are major participants

Objective
- ❖ Design an agent that learns the pricing function of a given day
- ❖ Use approximate architecture to overcome the "Curse of Dimensionality"
- ❖ This allows a greater number of appliances to be scheduled as compared to episodic learning processes that use tabular methods
- ❖ Neural Networks (NN) are proposed for Value Function Approximation

Design Parameters
- ❖ Fig.1 shows the Fully Observable Markov Decision Process
- ❖ Day-ahead pricing data from MISO (perfect foresight assumption)
- ❖ Neural Networks used: **S**ingle / Multiple stack of **tanh / RELU** hidden layers, **R**ecursive Network with randomized hidden layer
- ❖ The input architecture is depicted in Fig. 2. State, action features are represented by the input vector $\boldsymbol{x_f(s,a)} \in R^{h+1}$, whereas the output $\tilde{Y}(s,a)$ is a scalar.
- ❖ Weight are updated after: **E**very step in an episode (online), **C**ompleting an episode

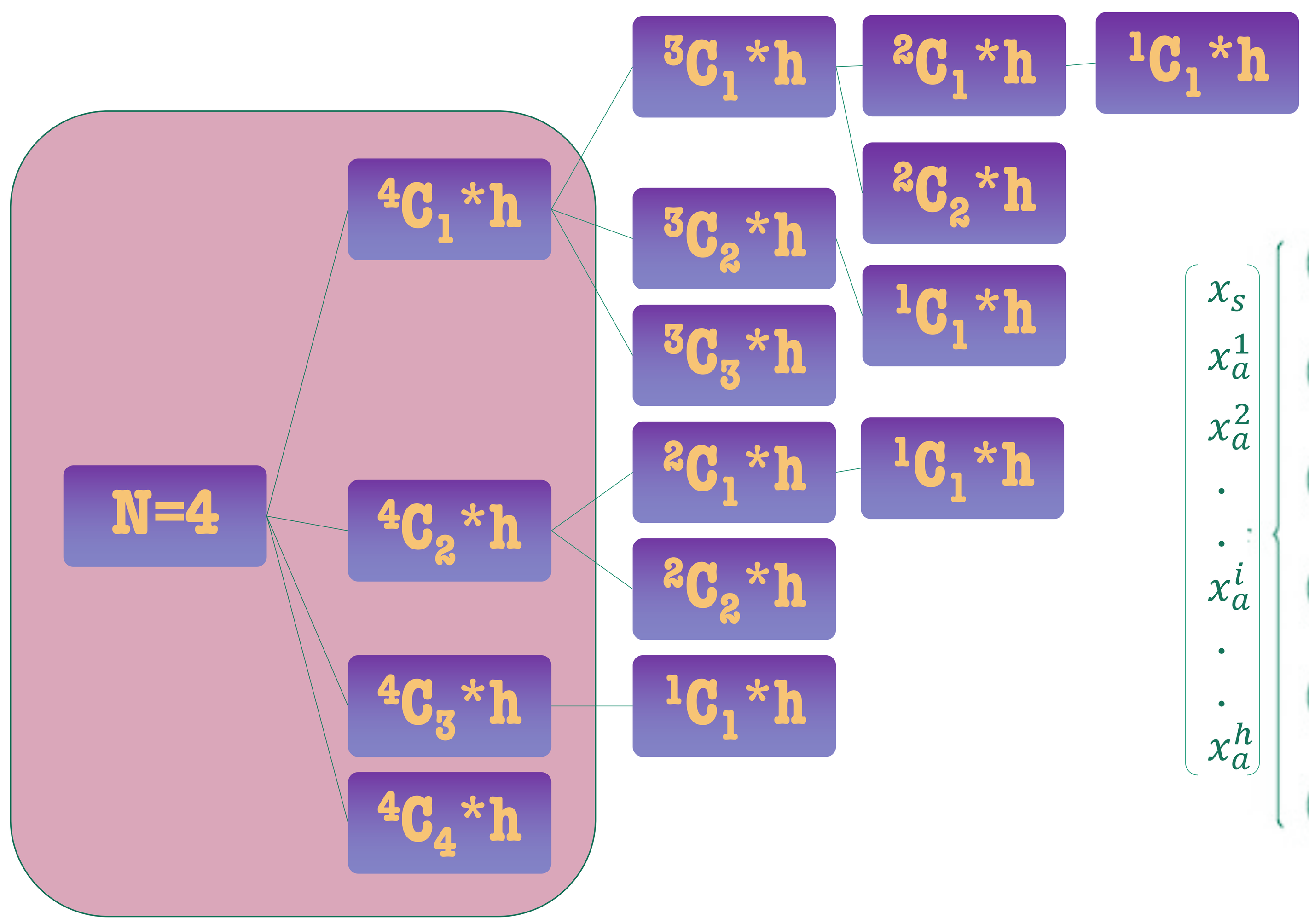

Fig.1 *Markov Decision Process of the appliance scheduling problem for N appliances over horizon h. $Q(s,a)$ values are stored in memory, resulting in an exponential increase in the time required for convergence to the optimal solution.* *The circle indicates the memory consumed after using a neural network*

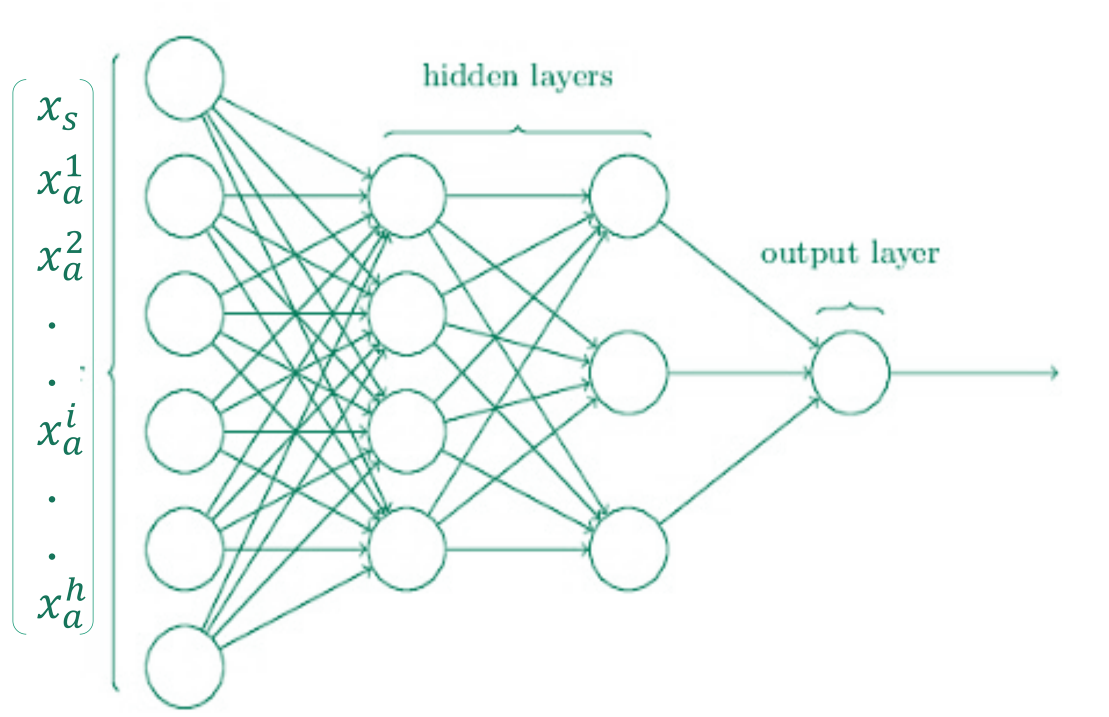

Fig.2 *Input to the NN consists of a state feature x_s, where $x_s \in R$ is the total power consumption of appliances that have already been scheduled, and the action feature vector $\boldsymbol{x_a} = x_a.\boldsymbol{e_t}$, where $x_a \in R$ is the total power consumption of appliances selected to be scheduled and $\boldsymbol{e_t}$ is the standard basis vector of dimension **h** corresponding to the activation time **t** of the selected appliances*

Reinforcement Learning
- ❖ Reinforcement Learning is an optimization method in which an agent continuously interacts with the environment to discover actions that minimize long-term costs
- ❖ *Bellman's optimality equation for $\boldsymbol{Q_*}$:* For optimal state, action pairs i.e. s^*, a^* the following equation holds: $Q_*(s,a) = \sum_{s',r} p(s',r|s,a).[r + \gamma.maxQ(s',A')]$, A' is the set of all actions given s'
- ❖ Semi-gradient SARSA(0) is a 1-step, Temporal Difference (TD) algorithm that uses NNs to predict $\tilde{Y}(s,a) \approx Q(s,a)$ for all s,a. The optimal policy $\boldsymbol{\pi_*}$ is achieved based on the Generalized Policy Iteration (GPI) framework shown in Fig.3
- ❖ *Supervisor?* The error for the i^{th} step is: $\delta_i = \tilde{Y}(s_i,a_i) - [R_{i+1} + \gamma\tilde{Y}(s_{i+1},a_{i+1})]$. The rightmost term is the target and is assumed to be constant during stochastic gradient descent. However, for a fixed input the target continuously varies with every iteration and introduces learning noise
- ❖ In Deep Q-networks a separate NN is used for predicting the target output. This, along with Experience Replay, has enabled the discovery of optimal strategies while playing Atari using AI

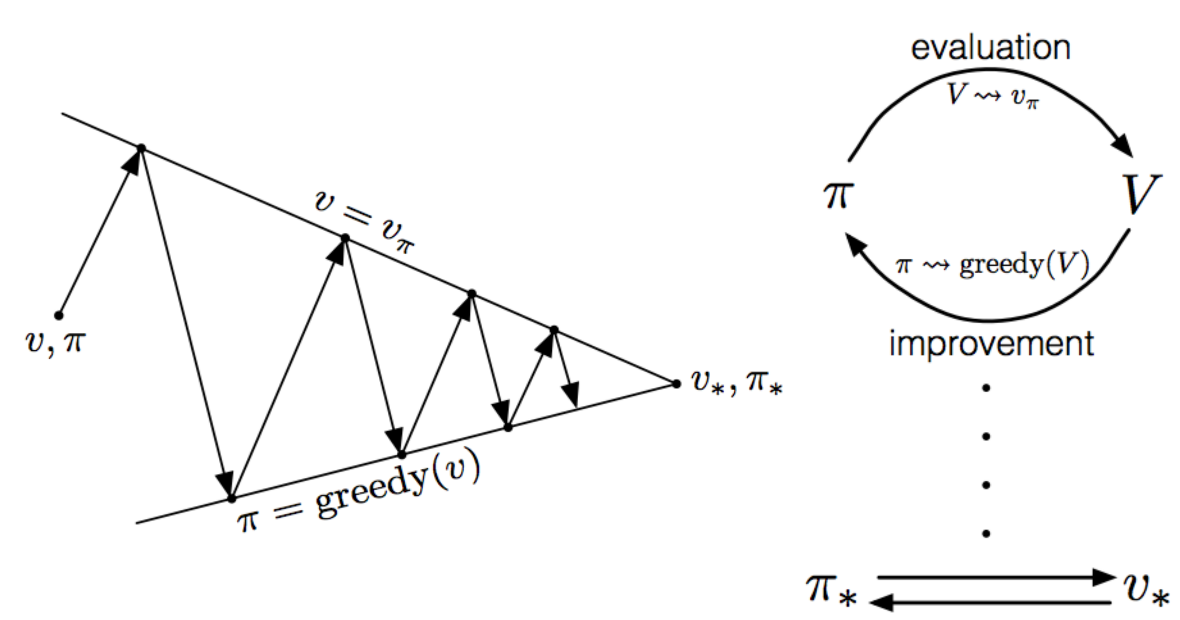

Fig.3 At the end of an episode, the NN weights are updated only ONCE to improve the prediction of $Q(s,a)$ from $\tilde{Y}_k(s_i,a_i)$ to $\tilde{Y}_{k+1}(s_i,a_i)$ By acting greedily with respect to the new update the policy changes from $\boldsymbol{\pi_k}$ to $\boldsymbol{\pi_{k+1}}$, which is then evaluated by the NN in the next iteration. By alternating between policy evaluation and policy improvement, the agent eventually converges to the optimal policy $\boldsymbol{\pi_*}$*

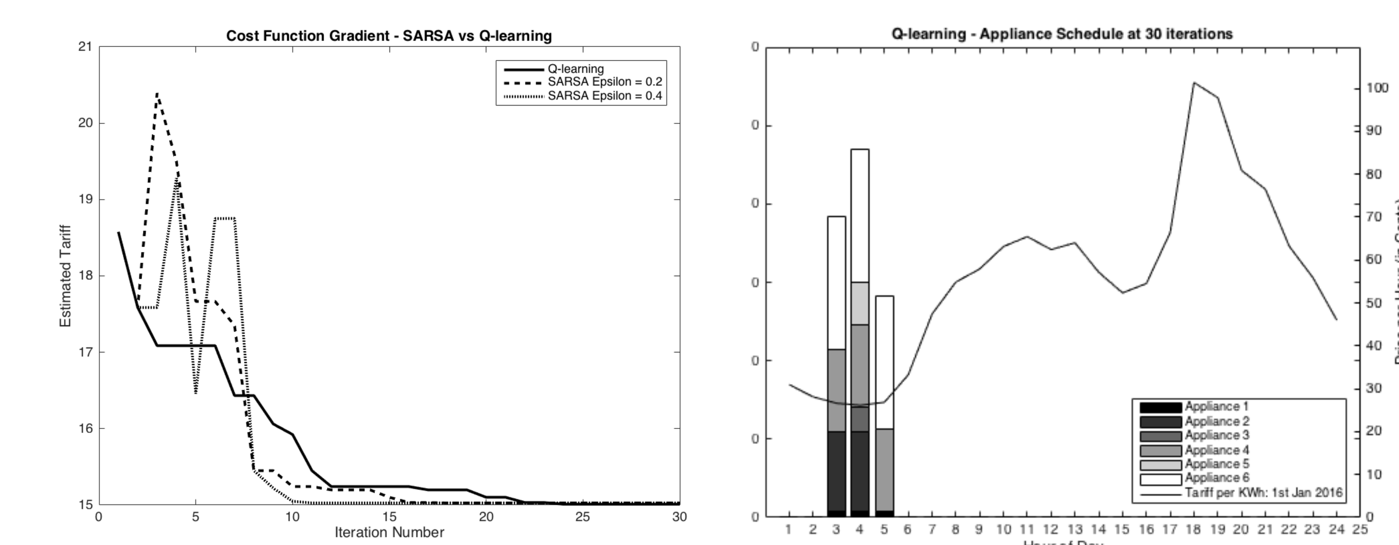

Fig.4 *Quick convergence to optimality using modified SARSA, Q-learning. In this, 1-steps are generated for all state, action pairs, after which the action vector for a given state is updated as follows:* $\boldsymbol{Q_{k+1}(s,a)} = (1-\alpha).\boldsymbol{Q_k(s,a)} + \alpha.\{\boldsymbol{R(s,a)} + \boldsymbol{Q'_k(s',a')}\}$, *where* $\boldsymbol{Q'_k(s',a')} = [max(\boldsymbol{Q_k(s'_1, A'_1)})..max(\boldsymbol{Q_k(s'_L, A'_L)})]$

Simulation Results

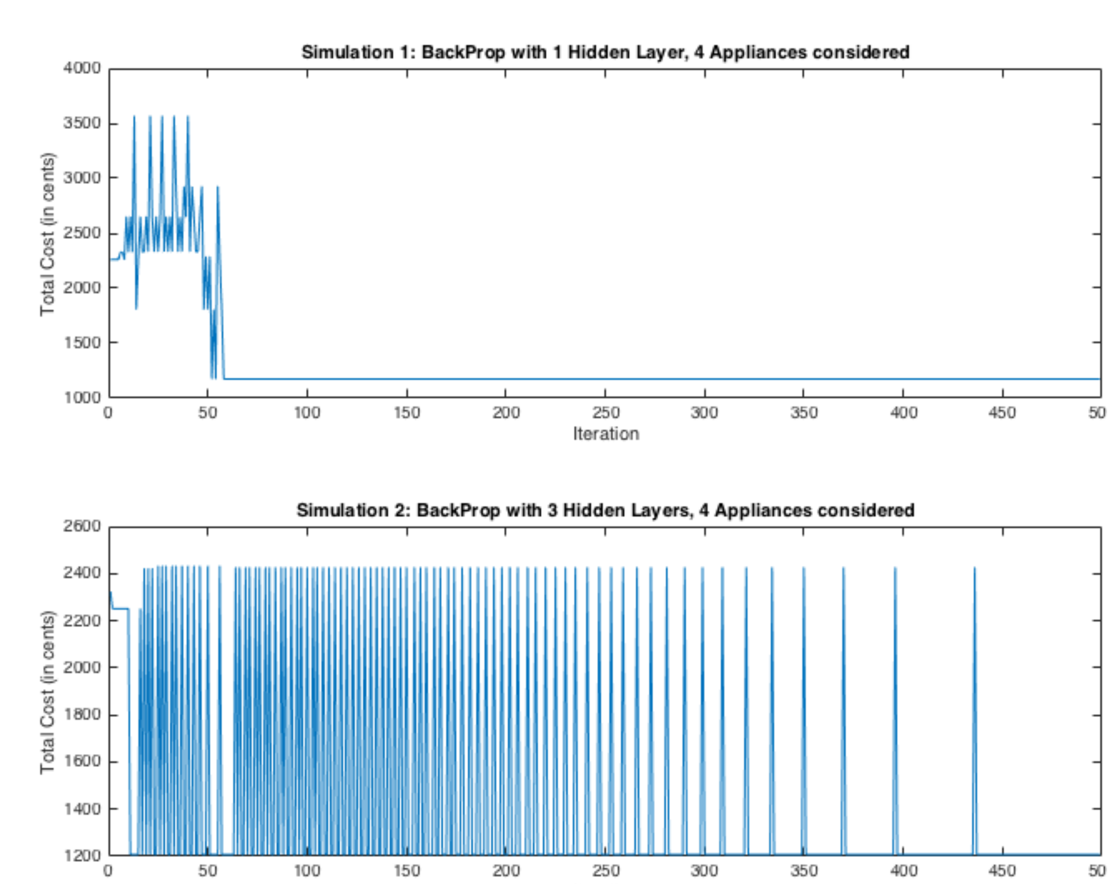

Fig.5 *Near-optimal results by using single/multiple tanh stacks on 14 appliances*

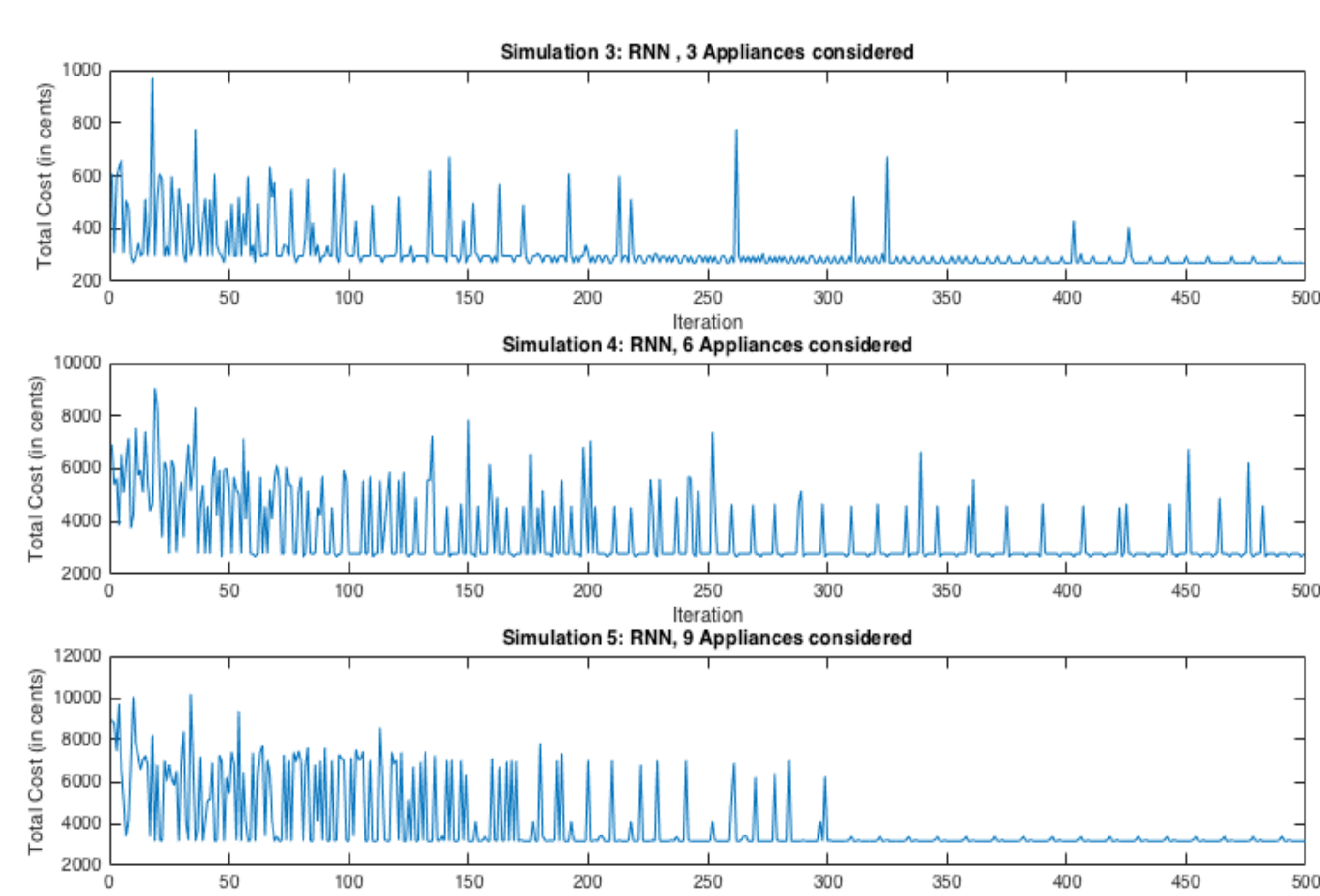

Fig.6 *Near-optimal results by using RNN on 3, 6, & 9 appliances*

**Obtained from Reinforcement Learning: An Introduction by Andrew Barto and Richard S. Sutton*

Namit Chauhan

Electrical Engineering, University of Washington