Deep Learning for Energy Efficient Cloud Computing

Dr. Xue Lin
Northeastern University
Nov 30, 2017 ES2

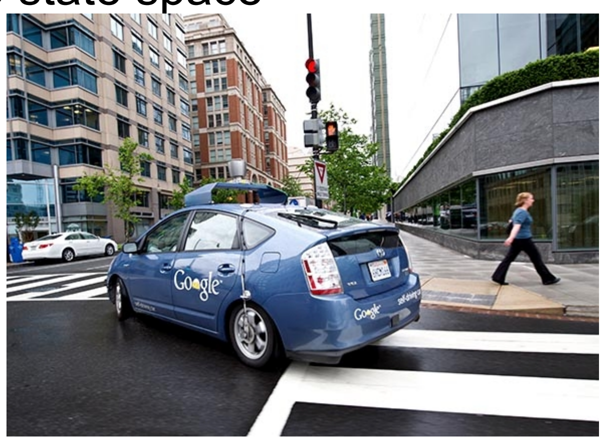
Outline

- Deep reinforcement learning (DRL) background
- Cloud computing resource allocation application
- Hybrid electric vehicle powertrain control application
- Deep learning acceleration with structured matrices

Deep Reinforcement Learning

DeepMind

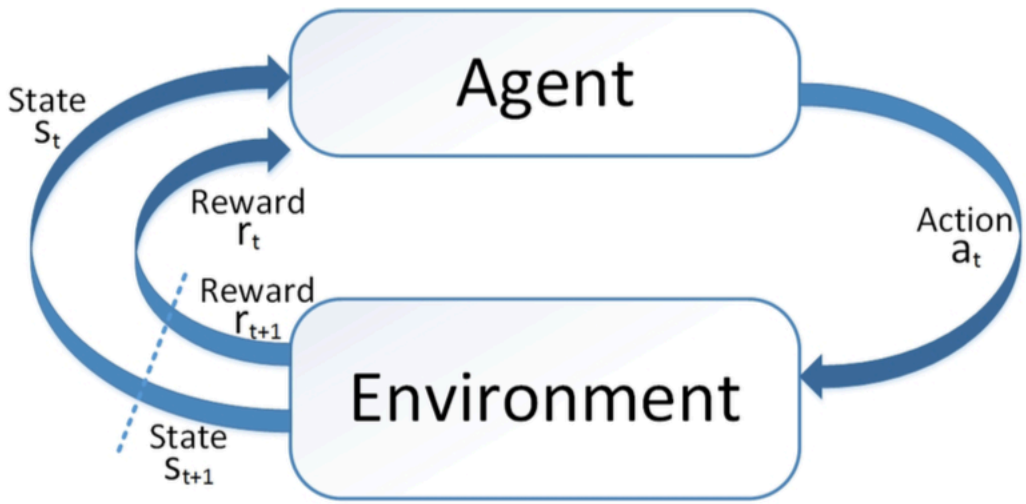
Deep Reinforcement Learning
From Google DeepMind
Control of complicated systems with large state space



Autonomous Driving

Reinforcement Learning Basics

- Agent-Environment Interaction System
  - Agent: decision-maker – take actions to maximize a discounted accumulative reward
  - Environment: everything outside the agent – present its state and award to the agent
  - A sequence of interactions at discrete time steps



Reinforcement Learning Algorithm and Limitation

- Algorithm
  - Exploration-exploitation when choosing the action

Algorithm 1 TD(λ)-Learning Algorithm

Initialize $Q(s,a)$ arbitrarily for all the state-action pairs.
for each time step t **do**
    Choose action a_t for state s_t using the exploration-exploitation policy.
    Take action a_t, observe reward r_{t+1} and next state s_{t+1}.
    $\delta \leftarrow r_{t+1} + \gamma \cdot \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)$.
    $e(s_t, a_t) \leftarrow e(s_t, a_t) + 1$.
    for all state-action pair (s,a) **do**
        $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot e(s,a) \cdot \delta$.
        $e(s,a) \leftarrow \gamma \cdot \lambda \cdot e(s,a)$.
    end for
end for

- Limitation
  - Convergence rate is proportional to the number of state-action pairs
  - Difficult to solve problems with high dimensional state and action spaces, such as AlphaGO and autonomous driving.

Deep Reinforcement Learning Overview

- Agent-Environment Interaction System
- Offline DNN construction phase
  - Train a DNN that computes Q(s,a) value for a given state-action pair
  - Training data (Q(s,a) samples) can be accumulated from model-based procedure or actual measurement data
- Online deep Q-learning phase
  - At a decision epoch, the agent performs inference using the DNN to obtain the $Q(s_k,a)$ value estimate for each action a
  - Action with the maximum $Q(s_k,a)$ value is selected with probability $1-\epsilon$
  - Q values are updated with the observed new state and received reward
- DNN is updated by new Q values at the end of execution sequence

Deep Reinforcement Learning Algorithm

Algorithm 1 The General DRL Framework

Offline:

Extract real data profiles using certain control policies and obtain the corresponding state transition profiles and $Q(s, a)$ value estimates;
Store the state transition profiles and $Q(s, a)$ value estimates in experience memory $\mathcal{D}$ with capacity $N_{\mathcal{D}}$;
Iterations may be needed in the above procedure;
Pre-train a DNN with features (s, a) and outcome $Q(s, a)$;

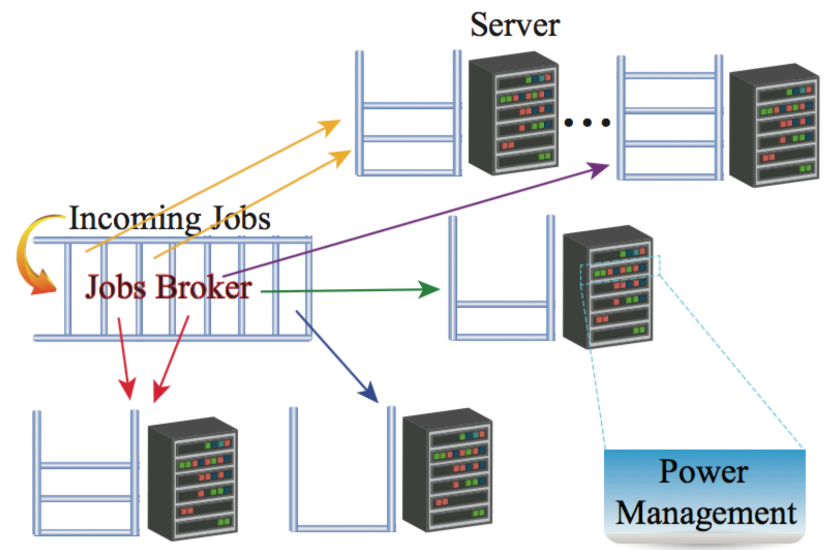
Online:

for each execution sequence **do**
  for at each decision epoch t_k **do**
    With probability ϵ select a random action, otherwise $a_k = \operatorname{argmax}_a Q(s_k, a)$, in which $Q(s_k, a)$ is derived (estimated) from DNN;
    Perform system control using the chosen action;
    Observe state transition at next decision epoch t_{k+1} with new state s_{k+1}, receive reward $r_k(s_k, a_k)$ during time period $[t_k, t_{k+1})$;
    Store transition (s_k, a_k, r_k, s_{k+1}) in $\mathcal{D}$;
    Updating $Q(s_k, a_k)$ based on $r_k(s_k, a_k)$ and $\max_{a'} Q(s_{k+1}, a')$ based on Q-learning updating rule;
  end for
  Update DNN parameters θ using new Q-value estimates;
end for

Online computational complexity is low

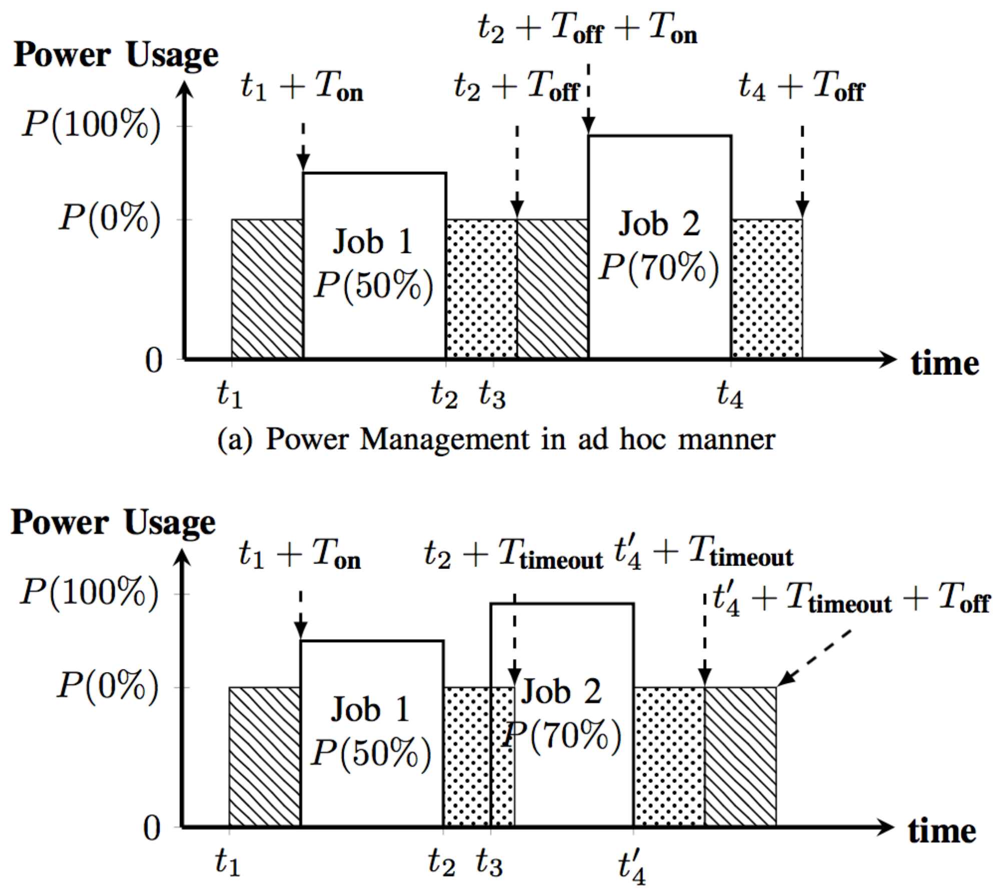
Cloud Computing System Modeling (Global)

- A server cluster with M physical servers that offer D types of resources
- When a job comes, the (global) job broker dispatches it to one of the servers for processing
- Each server queues all assigned jobs and allocates resources for them in a first-come-first-serve manner



Cloud Computing System Modeling (Local)

- Each server performs power management by turning on/off
- The global job assignment and local server power management effect the overall power consumption and system performance



(a) Power Management in ad hoc manner

(b) Power Management with DPM technique

DRL-Based Global Control

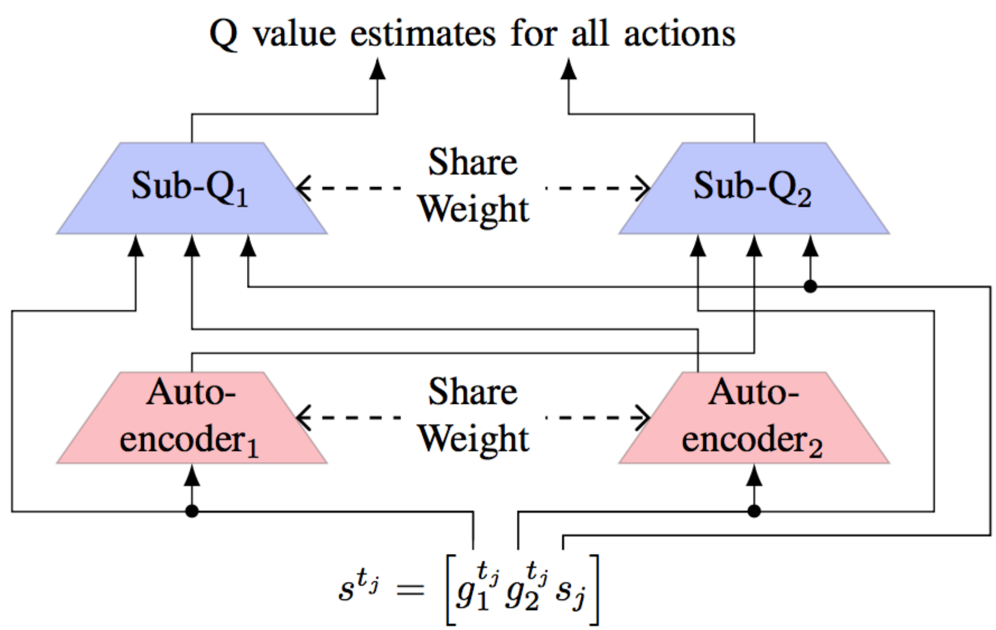
- Event-driven continuous-time decision-making model to ensure enumerable action space
  - Decision epoch coincides with arrival time of a new job
  - Action is then the target server for processing the new job
- State representation: server cluster state + job state

$$
\begin{aligned}
s^{t_j} &= \left[s_c^{t_j}, s_j\right] = \left[g_1^{t_j}, \cdots, g_K^{t_j}, s_j\right] \\
&= [u_{11}^{t_j}, \cdots, u_{1|D|}^{t_j}, \cdots, u_{|M||D|}^{t_j}, u_{j1}, \cdots, u_{j|D|}, d_j]
\end{aligned}
$$

- Reward:
  - Hot spot avoidance
  - Power consumption
  - Job latency

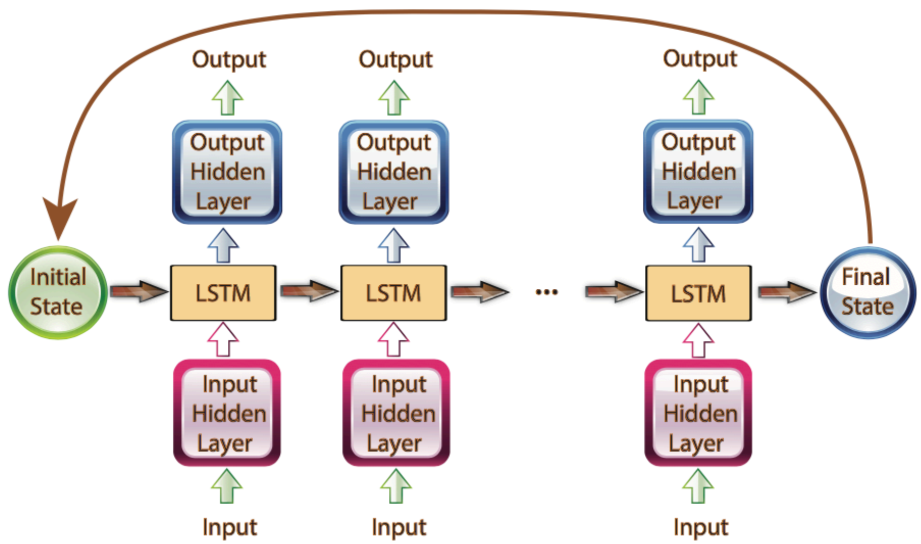
Global Control - Offline DNN Construction Phase

- To reduce training time for this high dimensional state space, use representation learning and weight sharing
  - Autoencoder to extract low-dimensional high-level representation of the state of server group k, denoted by g_k
  - Train sub-network Sub-Q_k with g_k, s_j and all other $g_{k'}$ as input features
  - Introduce weight sharing for all K autoencoders and sub-networks for reduced training time and shared samples



Local Server Power Management

- For controlling the turning on/off of each local server
- Consist of LSTM-based workload predictor and model-free, continuous-time Q-learning-based adaptive power management
  - Workload predictor provides partial observation of the actual future workload characteristics i.e., inter-arrival times of jobs



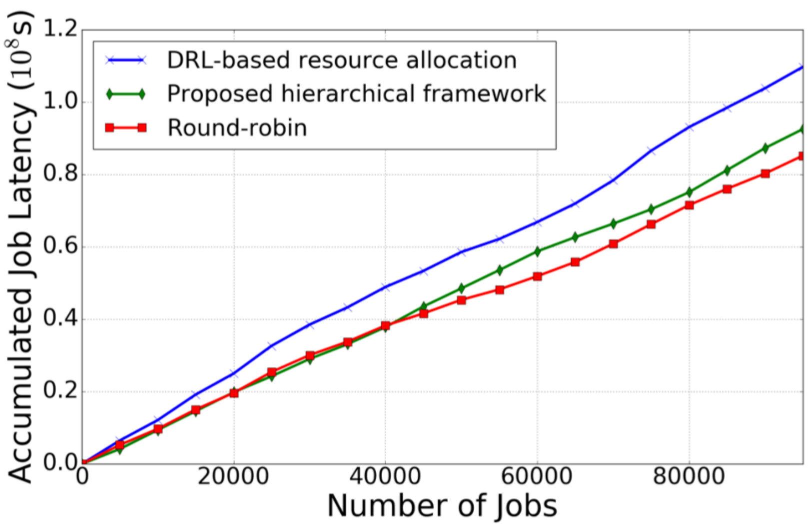
Local Server Power Management

- For controlling the turning on/off of each local server
- Consist of LSTM-based workload predictor and model-free, continuous-time Q-learning-based adaptive power management
  - Workload predictor provides partial observation of the actual future workload characteristics i.e., inter-arrival times of jobs
  - Q-learning determines whether to turn on/off the server [action] based on the server mode (active, idle, or sleep) and the predicted job inter-arrival time [state]

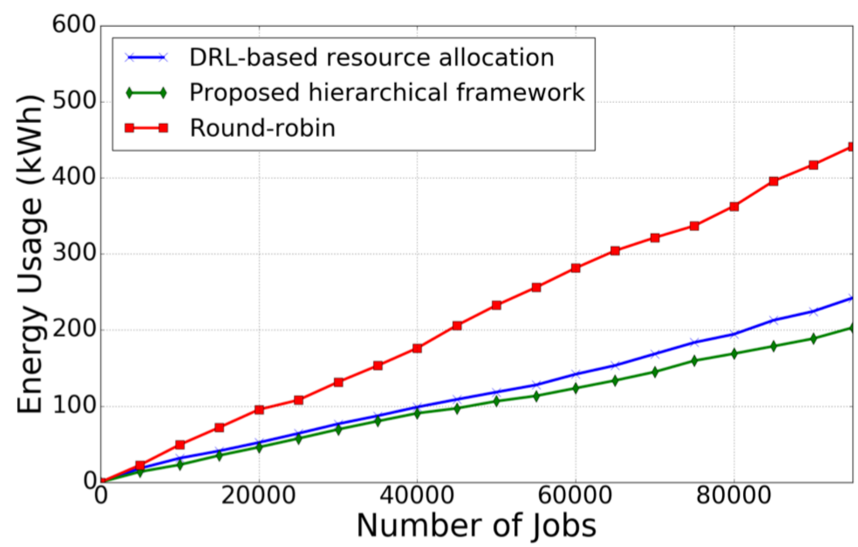
Simulation Setup

- Compare among
  - The proposed DRL method including both the global control and the local server power management
  - DRL for the global control WITHOUT local server power management
  - Baseline round-robin virtual machine allocation method
- Google cluster data trace
  - One-month job trace of the Google server cluster
  - Include job arrival times, durations, and resource requirements

Simulation Results



(a) Accumulated job latency versus the number of jobs



(b) Energy usage versus the number of jobs

Hybrid Electric Vehicle - Introduction

- Hybrid electric vehicle (HEV) Propulsion
  - Internal combustion engine (ICE)
  - Electric motor (EM)
- Compared to ICE vehicles, HEVs can achieve
  - Higher fuel economy
  - Lower pollutant emission
- HEVs on the market
  - Toyota Prius, Honda Insight, Ford Fusion Hybrid
  - Plug-in HEV: Chevrolet Volt, BYD F3DM

DRL-Based HEV Power Management

- Model-free, Discrete-time system
- State Representation
  - Power demand
  - Vehicle speed
  - Battery pack charge level
- Action
  - Battery discharging current
  - Gear ratio
- Reward
  - Fuel consumption in each time step

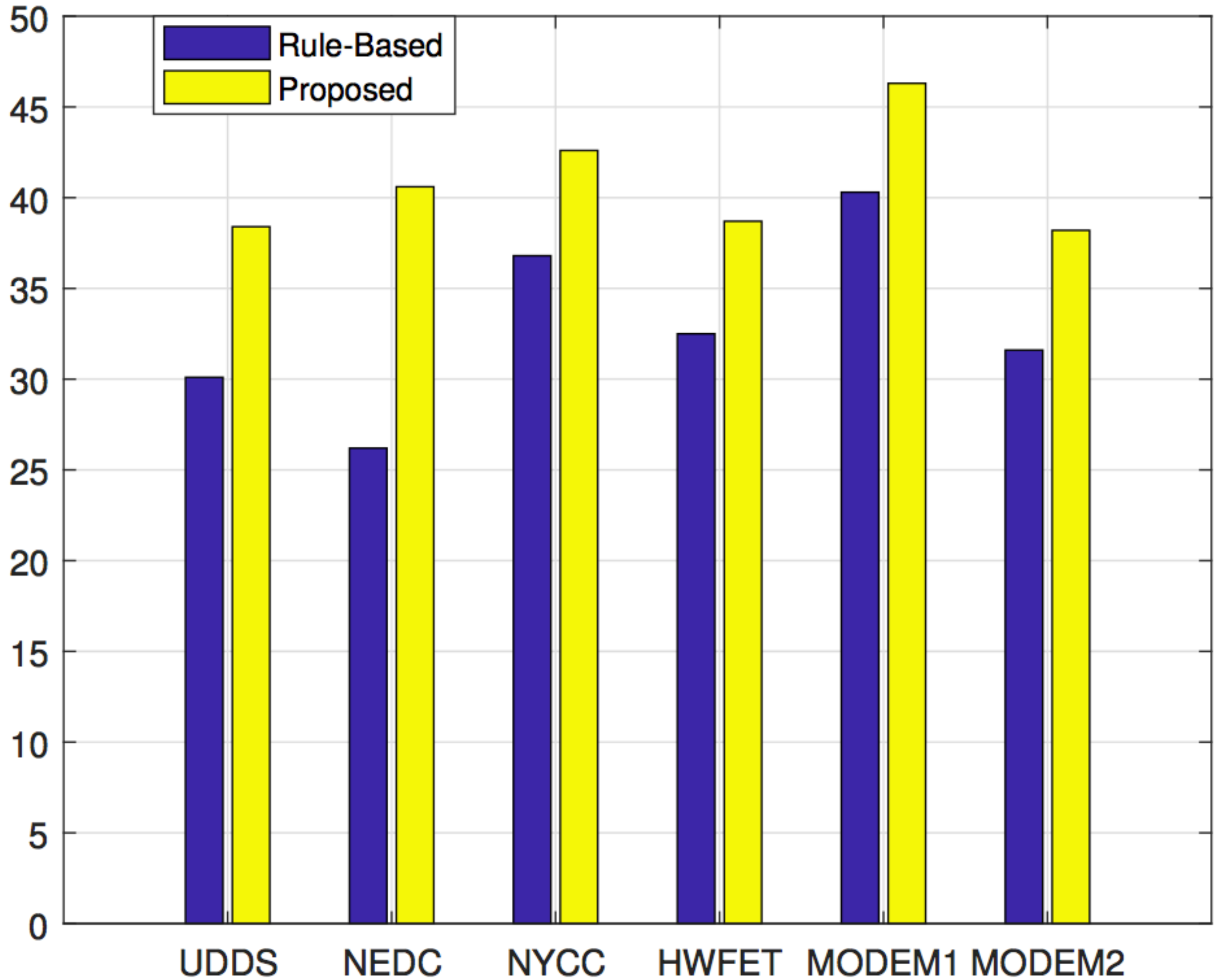
Simulation Setup

- HEV parameters

Vehicle	**Transmission**	**ICE**
$m = 1254kg$	$\rho_{reg} = 1.75$	Peak power 41kW
$C_R = 0.009$	$\eta_{reg} = 0.98$	Peak eff. 34%
$C_D = 0.335$	$\eta_{gb} = 0.98$	**EM**
$A_F = 2m^2$	R(k) = [13.5;7.6;	Peak power 56kW
$r_{wh} = 0.282m$	5.0;3.8;2,8]	Peak eff. 92%
Battery		
Capacity 25 A.h Voltage 240V		

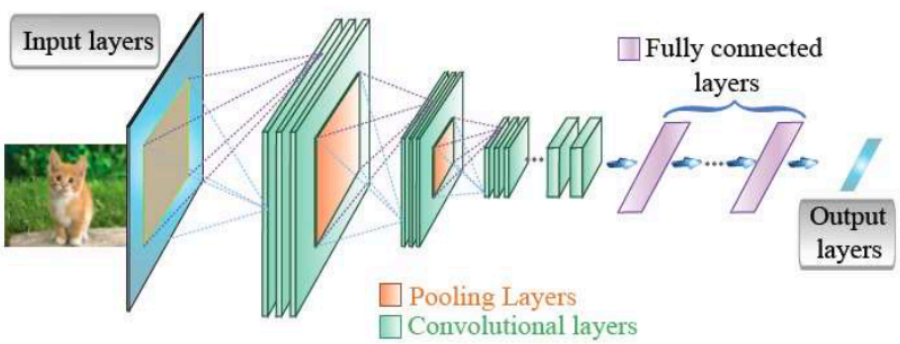
- Real-world and synthetic driving profiles (vehicle speed and acceleration traces)

Simulation Result



Deep Learning Acceleration with Structured Matrices

- Deep neural networks (DNNs) consist of multiple cascaded layers with millions to hundreds of millions of parameters (weights)
- By using structured weight matrices and the corresponding FFT-based matrix-vector multiplication algorithms, the storage and computation complexities of each layer can be reduced from $O(n^2)$ to $O(n)$, and from $O(n^2)$ to $O(n\log n)$.



Circulant $(c_{n-j+i \bmod n})_{i,j=0}^{n-1}$

$$\begin{pmatrix} c_0 & c_{n-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & \cdots & c_2 \\ \vdots & c_1 & c_0 & \cdots & \vdots \\ c_{n-2} & \vdots & \vdots & \vdots & c_{n-1} \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{pmatrix}$$

Cauchy $(1/(u_i - y_j))_{i,j=0}^{n-1}$

$$\begin{pmatrix} 1/(u_0 - y_0) & \cdots & 1/(u_0 - y_{n-1}) \\ 1/(u_1 - y_0) & \cdots & 1/(u_1 - y_{n-1}) \\ \vdots & \vdots & \vdots \\ 1/(u_{n-1} - y_0) & \cdots & 1/(u_{n-1} - y_{n-1}) \end{pmatrix}$$

Toeplitz $(t_{i-j})_{i,j=0}^{n-1}$

$$\begin{pmatrix} t_0 & t_{-1} & \cdots & t_{1-n} \\ t_1 & t_0 & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & t_{-1} \\ t_{n-1} & \cdots & t_1 & t_0 \end{pmatrix}$$

Hankel $(h_{i+j})_{i,j=0}^{n-1}$

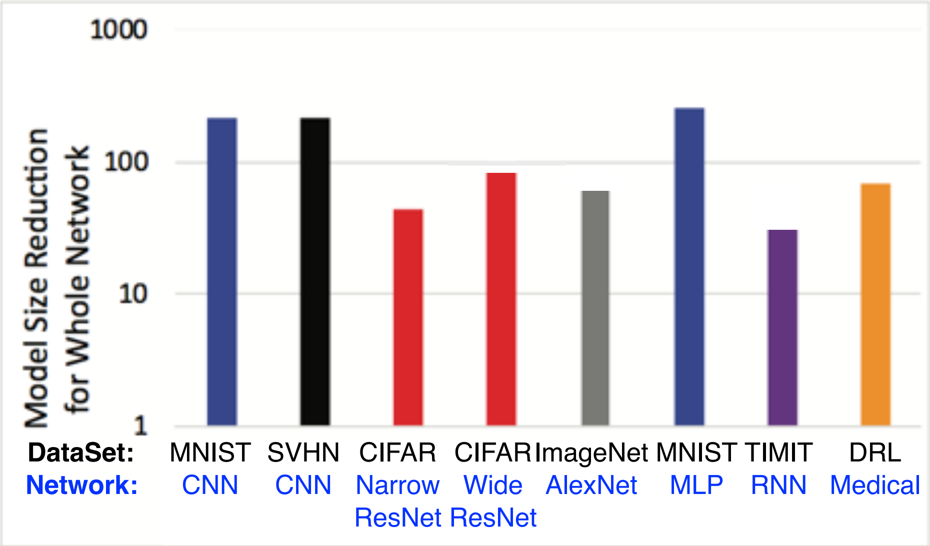
$$\begin{pmatrix} h_0 & h_1 & \cdots & h_{n-1} \\ h_1 & h_2 & \cdots & h_n \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ h_{n-1} & h_n & \cdots & h_{2n-2} \end{pmatrix}$$

Vandermonde $(v_i^j)_{i,j=0}^{n-1}$

$$\begin{pmatrix} 1 & v_0 & \cdots & v_0^{n-1} \\ 1 & v_1 & \cdots & v_1^{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & v_{n-1} & \cdots & v_{n-1}^{n-1} \end{pmatrix}$$

Compression Ratio and Test Accuracy

- We can achieve significant model compression ratios for different applications and network structures with only 1~2% accuracy loss



- Thank you!