Allocating optimal GPU resources to multiple DNN inference models using Reinforcement Learning

Seong Sihoon, Phou Mithona

27/06/2022

About Motivation

⇨ Recently, main applications make use of DNN require GPU resource to handle their tasks. e.g. inference of image classification, natural language processing, etc.

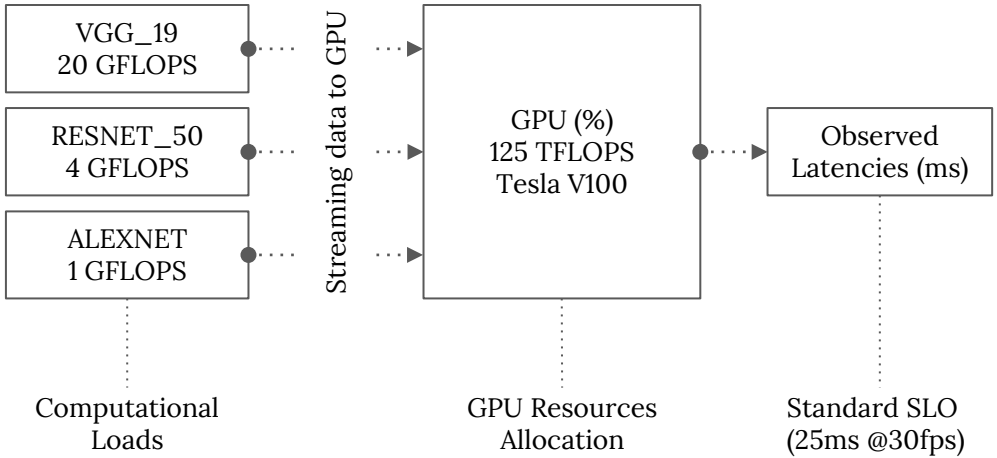
⇨ Execution of different DNN models concurrently often leads to wastage of GPU resources because of time-sharing.

⇨ Is it essential to spatially distribute the right amount of GPU resources to meet model's inference performance.

⇨ So we have to deploy appropriate GPU resources to multiple DNNs in multi-tenant environment to raise their performance.

⇨ The need of dynamic GPU resource-allocation mechanisms.

⇨ Thus, we used RL algorithm to get optimal GPU resources percentage of multiple DNNs.

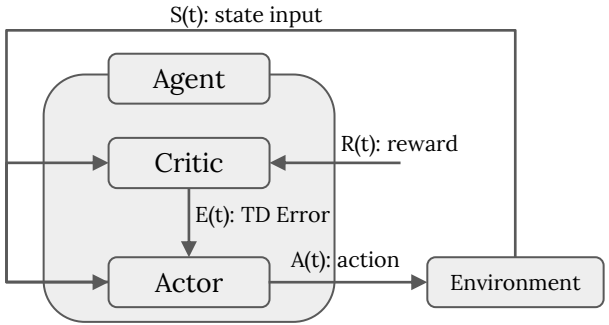


⇨ In this project, we will perform a simulation of allocating optimal GPU resources to multiple DNNs using DDPG algorithm targeting Nvidia MPS technique.

Background

DDPG Algorithm (Actor-Critic model)

⇨ DDPG provides continuous state and action spaces. It uses TD and experience replay to learn and update the policy. It is possible to calculate a future reward estimate for all values. DDPG computes TD target and update TD error.



⇨ Actor-network: acts as the agent and predicts the expected reward. It proposes action and computes action predictions for the current state. It computes and update the policy.

⇨ Critic-network: is trained to predict the accuracy of the actor-network given a state and action. It computes Q- function, minimizes the loss, and generates TD error signal each time step.

⇨ TD error, E(t): reports the difference between the estimated reward and the actual reward at any given state or time step.

⇨ Experience Replay: stores list of tuples (state, action, reward, next state) and learn from sampling experience.

The state-of-the-art CUDA MPS (multi-process-service)

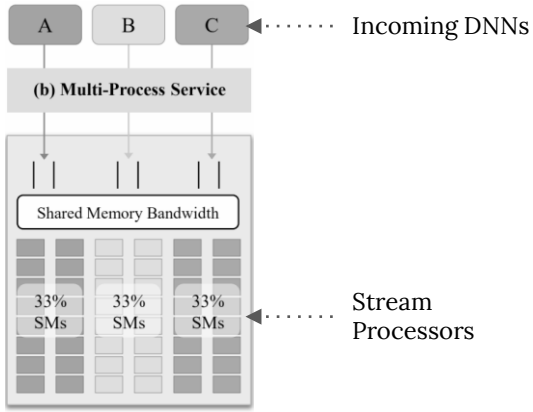
⇨ MPS is useful for GPU utilization (multi-process concurrently).

⇨ Spatial sharing different Deep Learning (DL) inference tasks

⇨ Batch processing

⇨ Static provisioning (set fixed limit for resources GPU%),

but would not be adaptive to workload variations.



Design of RL Environment and Implementation

Considerations

⇨ Observation Space: continuous (ms)

-latency_vgg_19 (initial=56ms)
-latency_resnet_50 (initial=22ms)
-latency_alexnet (initial=6ms)
-SLO_vgg_19 (standard=25ms)
-SLO_resnet_50 (standard=25ms)
-SLO_alexnet (standard=25ms)

⇨ Action Space: continuous (GPU%)

-ACTION_VGG_19
-ACTION_RESNET_50
-ACTION_ALEXNET

⇨ Reward Function: continuous (scalar)

-reward_vgg_19
-reward_resnet_50
-reward_alexnet

⇨ Termination Conditions: logic

$$done = \frac{100 \times Reward}{SLO} < 5$$

-This expression normalizes state's latencies to %
-True: when Latencies meets SLO standard and with less than GPU 5%

⇨ Algorithm: DDPG (based on the above considerations)

Defined Functions

⇨ Rewards Calculation

$$Reward = SLO - Latency$$

-Latency < SLO: (+) reward
-Latency > SLO: (-) reward
-reward_vgg_19 [-375, 5]
-reward_resnet_50 [-25, 5]
-reward_alexnet [10, 23]

⇨ Actions Calculation

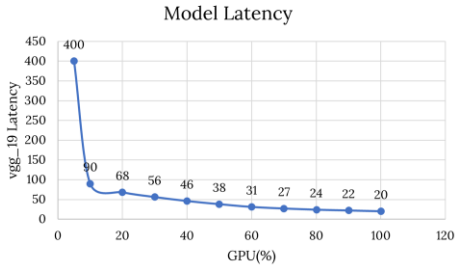
$$GPU\% = 100 \times \frac{act_{vgg19,\, resnet50,\, alexnet}}{act_{vgg19} + act_{resnet50} + act_{alexnet}}$$

-Vgg_19 GPU% [0, 100]
-resnet_50 GPU%[0, 100]
-alexnet GPU% [0, 100]

⇨ Model-Latency States

$$Latency = ae^{(-bx)+C}$$

-latency_vgg_19 [20, 400]ms
-latency_resnet_50 [15, 50]ms
-latency_alexnet [2, 15]ms



Simulation Tools

⇨ Gym toolkit:
-Environment (state, action, reward)

⇨ TF-Agents library:
-Actor-network
-Critic-network
-GPUallocator agent

⇨ Google Colab:
-Notebook
-Remote engine

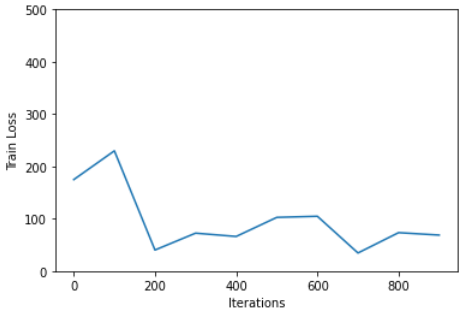
⇨ Evaluation on GPU machine:
-NVIDIA RTX 3090

Simulation Results

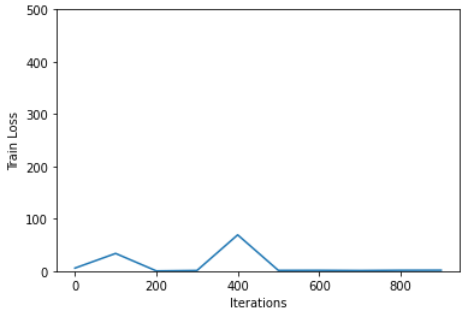
Training Loss and Average Reward

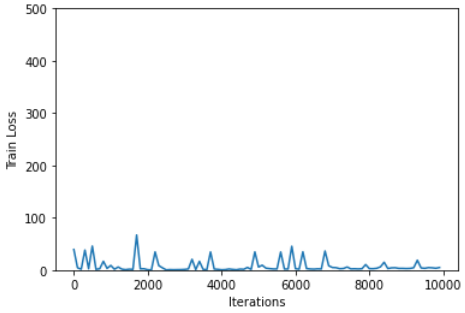
⇨ If training proceeds correctly, the average reward will increase with time.

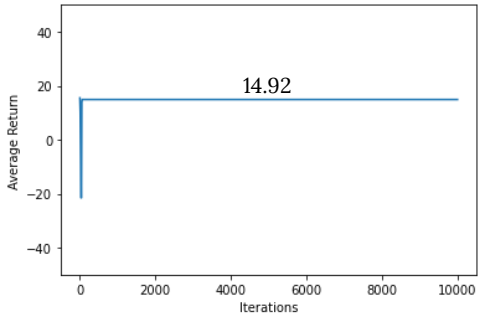
Train on Remote GPU



Train on Local GPU







States: Latencies and Action: GPU Percentages

```
vgg-19 latency: 56
resnet-50 latency: 22
alexnet latency: 6
SLO vgg-19: 25
SLO resnet-50: 25
SLO alexnet: 25
vgg-19 GPU%: 0
resnet-50 GPU%: 0
alexnet GPU%: 0
```

```
vgg-19 latency: 56
resnet-50 latency: 22
alexnet latency: 6
SLO vgg-19: 25
SLO resnet-50: 25
SLO alexnet: 25
vgg-19 GPU%: 0
resnet-50 GPU%: 0
alexnet GPU%: 0


vgg-19 latency: 36.53
resnet-50 latency: 18.16
alexnet latency: 5.39
SLO vgg-19: 25
SLO resnet-50: 25
SLO alexnet: 25
vgg-19 GPU%: 33
resnet-50 GPU%: 33
alexnet GPU%: 33
```

Conclusion

⇨ This DDPG algorithm works for our custom environment in this project. By applying this RL method, we can solve our target problem to dynamically allocate GPU actions for states of multiple DNN inferences.

⇨ Our model converges very fast with less than 1,000 training iterations.

⇨ We did not code the DDPG algorithm from scratch, instead we used the basic dense models of the Actor and Critic networks.

⇨ In this project, we learnt how to design our custom RL environment and implemented this environment in simulation.

⇨ At the end of this project, we have learnt practical knowledge which is good contribute to our future researches.

Source code and paper:

⇨ Our Github Repo: https://github.com/phoumithona/reinforcement-learning-course-2022-1
⇨ Prof. Jeff Heaton: T81-558: Application of Deep Neural Networks
⇨ GSLICE: Controlled Spatial Sharing of GPUs for a Scalable Inference Platform