Pontifícia Universidade Católica do Rio Grande do Sul

Neural Architecture Search Using Automated Planning

Felipe Roque Tasoniero

Automated Planning 2019/02

Objectives

- Generating high-performance CNN automatically
    - Discover a CNN architecture that performs similar to the state-of-the-art model.

Objectives

- Generating high-performance CNN automatically
    - Discover a CNN architecture that performs similar to the state-of-the-art model.
- Reduce computational time consuming
    - Constraint memory and incentivize quick inference.

Objectives

- Generating high-performance CNN automatically
  - Discover a CNN architecture that performs similar to the state-of-the-art model.
- Reduce computational time consuming
  - Constraint memory and incentivize quick inference.
- Explore MetaQNN algorithm (Baker et al. 2016).
  - Ablation study on MNIST dataset.

Objectives

- Generating high-performance CNN automatically
  - Discover a CNN architecture that performs similar to the state-of-the-art model.
- Reduce computational time consuming
  - Constraint memory and incentivize quick inference.
- Explore MetaQNN algorithm (Baker et al. 2016).
  - Ablation study on MNIST dataset.
- Implementing the penalize configuration
  - Penalize the reward function (e.g., when the model searched reaches a determined size and/or when forward passes are slow).

Neural Architecture Search (NAS)

- Automatically search a neural network architecture that perform as well as ones designed manually by experts.

Neural Architecture Search (NAS)

- Automatically search a neural network architecture that perform as well as ones designed manually by experts.
- Reinforcement Learning and Evolutionary Algorithms (widely used, but there are other techniques that has been used for Neural Architecture Search).

Neural Architecture Search (NAS)

- Automatically search a neural network architecture that perform as well as ones designed manually by experts.
- Reinforcement Learning and Evolutionary Algorithms (widely used, but there are other techniques that has been used for Neural Architecture Search).
- Great amount of GPUs are necessary to find a high-performing architecture (In *Zoph et al. 2016*, they use 800 GPUs concurrently to train their model using a RL algorithm).

MetaQNN

- Algorithm based on Reinforcement Learning to automatically generate a high-performing CNN architecture for a given learning task.
- *Baker et al.* have shown that this algorithm is capable of searching a high-performing CNN architecture by using only 10 GPUs during 8~10 days.
- MetaQNN algorithm use Q-learning with ϵ-greedy exploration strategy and experience replay.

MetaQNN

- Q-learning
  - Used to design the CNN architecture.
  - Type of layers used (Search Space): Convolution, Fully Connected, Pooling, Global Average Pooling, and Softmax.
  - Q-value is updated using the validation accuracy value for each model searched.

MetaQNN

- Q-learning
    - Teaching an agent to find optimal paths as a Markov Decision Process (MDP).
    - The agent's goal is to maximize the reward over all possible paths.

$$Q(S_t, A_t) := (1 - \alpha_t)Q(S_t, A_t) + \alpha_t D_t(S_t, A_t) \quad (1)$$

$$D_t(S_t, A_t) = R_{t+1} + \gamma max_{A_t \in A(S_{t+1})} Q(S_{t+1}, A_t) \quad (2)$$

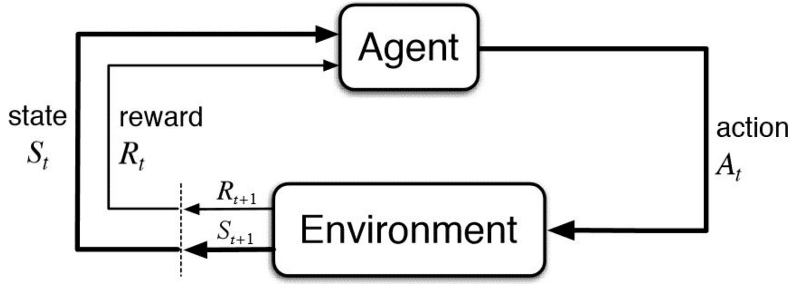

Figure 1: Q-learning update process.

MetaQNN

- ϵ-greedy exploration strategy and experience replay
  - We assume ϵ from 1 $\rightarrow$ 0 such as the agent begins in an exploration phase and slowly starts moving towards the exploitation phase.
  - We assume a schedule for the number of models generated for each ϵ-greedy step value.

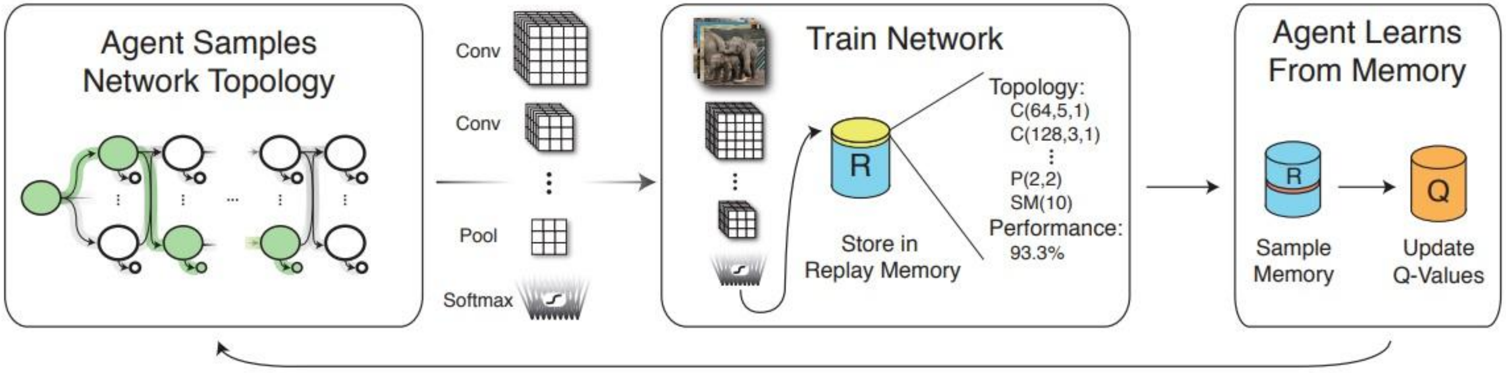

Figure 2: MetaQNN automated process for NAS using Q-learning (Baker et al. 2016).

MetaQNN

- Experience Replay
    - Experience replay provide a memory of its past explored paths and rewards
    - At a given interval, the agent samples from memory and update its Q-values via Eq. (1).

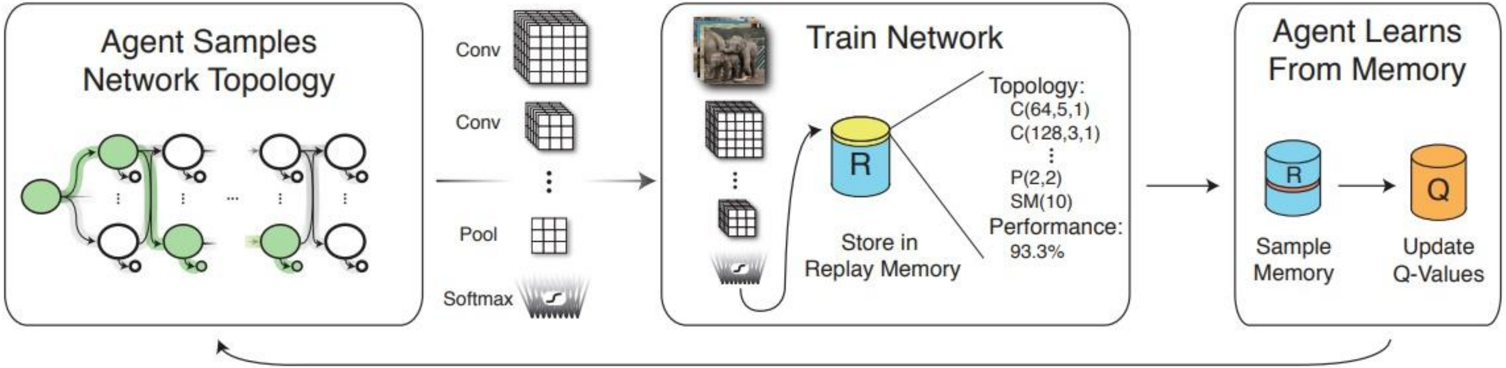

Figure 2: MetaQNN automated process for NAS using Q-learning (Baker et al. 2016).

MetaQNN

- State Space
  - Tuple of all relevant layers parameters.

Layer Type	Parameters	Values
Convolution (C)	*(i, f, l, d, n)*	<12, {1, 3, 5}, 1, {64,128,256,512}, {(∞, 8], (8, 4], (4, 1]}
Pooling (P)	*(i, (f, l), n)*	< 12, {(5,3), (3,2), (2,2)}, {(∞, 8], (8, 4], (4, 1]}
Fully Connected (FC)	*(i, n, d)*	<12, <3, {512, 256, 128}
Termination State	*(s, t)*	Global Avg. Pooling, Softmax

Figure 3: The state space for classification task. The parameters are: Layer depth *(i)*, Receptive field size *(f)*, Stride *(l)*, Receptive fields *(d)*, Representation size *(n)*, Previous state *(s)* and Type *(t)*

MetaQNN

- Action Space
  - The actions that the agent can perform on MetaQNN algorithm.
  - The actions are restricted, but the agent is allowed to terminate a path at any point.
  - The transitions are only allowed for a state with layer depth *i* to a state depth *i+1*
- Penalize configuration
  - Early stop (Baker et al. 2017).
  - Threshold for the validation accuracy value for each epoch.

Experiments and Results

- MetaQNN implementation using Pytorch framework.
- Experiments on MNIST dataset.
- Similar parameters to original MetaQNN algorithm.
- Using Adam as an optimizer for faster convergence.
- Reduce the number of training epochs from 40 to 10.
- Reduce the number of models generated for each ϵ-greedy schedule.
- Using a validation accuracy value threshold as a penalize criteria.
- Experiments done on a single GPU, taking 3 days to finish the experiment.
- The first results indicate that the model may be overfitting for some architectures, due to validation accuracy value being higher than training accuracy value.

Conclusion

- Our first analysis indicate reducing the number of models to search for each ϵ-greedy step, reducing the number of training epochs and using experience replay helps to reduce the total time.
- The using of a threshold as a penalize configuration was not significant in total time.
- As the model seems to overfitting, we think that search for weight regularization parameters could fix this problem.
- Future work:
  - More general and flexible search space.
  - Improvements on penalizing configuration.

THANK YOU!