KTH ROYAL INSTITUTE
OF TECHNOLOGY

Designing Neural Network Architectures Using Reinforcement Learning

Authors : Bowen Baker, Otkrist Gupta, Nikhil Naik and Rameh Raskar

Presenter: Ezgi Korkmaz

CNN and RL

- A typical CNN consists of several convolution, pooling and fully connected layers.
- Numeraous design choices for CNN architecture
  - The number of layers each type
  - The ordering of layers
  - The hyperparameters of the each type of layer.
- The design space of CNN architecture is extremely large
- In this paper automate the process of CNN architecture selection based on reinforcement learning
- Construct a Q-learning agent whose goal to discover CNN architecture which performs well
  - The agent learns through random exploration finds the higher performing models using the $\epsilon-$greedy strategy.
  - Agents gets a validation accuracy score instead of a reward

CNN and RL

- Construct a Q-learning agent whose goal to discover CNN architecture which performs well
  - The agent learns through random exploration finds the higher performing models using the $\epsilon-$greedy strategy.
  - Agents gets a validation accuracy score instead of a reward
  - Expedited learning process through repeated memory sampling using experience replay.
  - Q-learning based meta-modelling method as MetaQNN

Desingning Neural Network Architectures

- Related work
  - 1980s genetic algorithm based approaches [Schaffer et al. 1992]
  - Bayesian optimization for network architecture and [Bergstra et al. 2013]
  - Bayesian optimization for hyperparameters [Shahriari et al. 2016]
  - Meta modelling approach based on Tree of Parzen Estimators (TPE) [Bergstra et al. 2011]
- All failed to match the performance of handcrafted networks.

Q-Learning Background

- The task of teaching an agent to find an optimal paths as a Markov Decision Process (MDP) in a finite horizon environment.
- Environment will have a discrete finite state and action space S and U
- For any state $s_i \in S$ there is a finite set of actions $U(s_i) \subseteq U$
- Transitions are stochastic transition from s_i to s_j by taking $u \in U(s_i)$ is $p_{s'|s,u}(s_j|s_i, u)$
- At each time step t the reward r_t can be stochastic
- Agents goal to maximize the total expected reward over all possible trajectories $\max_{\tau_i \in \tau} R_{\tau_i}$

Q-Learning Background

- Total expected reward for trajectory τ_i

$$R_{\tau_i} = \sum_{(s,u,s')\in\tau_i} \mathbb{E}_{r|s,u,s'}[r|s,u,s'] \qquad (1)$$

- Maximum total expected reward to be $Q^*(s_i,u)$
- $Q(.)$ is the action value funtion
- The recursive maximization equation, Bellman's equation,

$$Q^*(s_i,u) = \mathbb{E}_{s_j|s_i,u}[\mathbb{E}_{r|s_i,u,s_j}[r|s_i,u,s_j] + \gamma \max_{u'\in U(s_j)} Q_t(s_j,u')] \qquad (2)$$

Q-Learning Background

- In many cases it is impossible to solve analytically, but can be formulated as iterative update

$$Q_{t+1}(s_i, u) = (1 - \alpha)Q_t(s_i, u) + \alpha[r_t + \gamma \max_{u' \in U(s_j)} Q_t(s_j, u')] \tag{4}$$

$$\lim_{t \to \infty} Q_t(s, u) = Q^*(s, u) \tag{5}$$

- The update equation has two parameters,
  - α is the Q-learning rate
  - γ is the discount factor

Q-Learning Background

- Q learning is $model free$
  - Agent can solve the task without explicitly constructing an estimate of the environmental dynamics
- Q learning is $off\ policy$
  - Agent can learn about optimal policies while exploring via non-optimal behavioral distribution
- Choose the behaviour distribution using ϵ-greedy strategy
  - Take a random action with probability ϵ
  - And greedy action $\max_{u \in U(s_t)} Q_t(s_i, u)$ with $1 - \epsilon$
- Experience replay
  - A memory of its explored paths and rewards
  - At a given interval agent samples from the memory and updates the Q-values

Designning Neural Network Architectures with Q-Learning

- The task of training a learning agent to sequentially choose neural network layers
- The agent will get a reward based on the validation accuracy
- Three main design choices
  - Reducing CNN layer definitions to simple state tuples
  - The set of layers the agent may pick
  - Balancing the size of the state-action space

The State Space

- Each state is a tuple of relevant layer parameters
  - Convolution (C)
  - Pooling (P)
  - Fully connected (FC)
  - Global average pooling (GAP)
  - Softmax (SM)
- Each layer has a parameter called layer depth and representation size (R-size)
- Convolutional nets progressively compress the representation of the original signal through pooling and convolution
- These layers in the state space may lead the agent on a trajectory where the intermediate signal representation gets reduced to a size that is too small for further processing.

The Action Space

- Restrict agent from taking certain actions.
  - Allow agent to terminate a path at any point
  - Allow transitions from state i to state with layer depth $i + 1$
  - Limit the number of fully connected (FC) layers max 2
  - A state s of type FC with number of neurons d may only transition to either a termination state or a state s' of type FC with number of neurons $d' \leq d$
  - An agent with state type of convolution (C) may transition to a state with any other layer type.
  - An agent with state type of pooling (P) may transition to a state with any other layer type other than P.

Q-Learning Training Procedure

- Learning rate $\alpha = 0.01$
- Discount factor $\gamma = 1$ not to over prioritize the short-term rewards
- Decrease the ϵ from 1 to 0.1 in steps
- Stop the agent $\epsilon = 0.1$ to obtain a stochastic final policy
- Main goal is to find well-performing model topologies can be ensembled to improve prediction performance
- During the training process maintain a replay dictionary storing
  - Network topology
  - Prediction performance on a validation set
- 100 models in the replay dictionary

Results

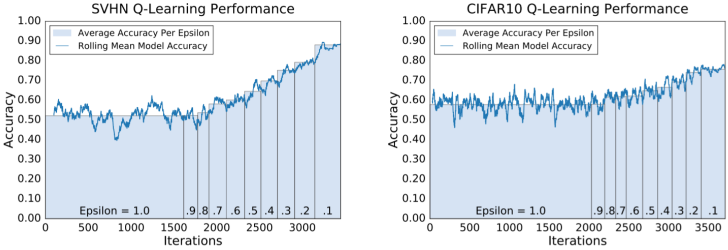

Figure: Q-Learning Performance on CNN accuracy for SVHN and CIFAR-10