Neural Architecture Search using Deep Reinforcement Learning

Ashish Kumar Jayant (M.Tech Research, CSA)
Shikhar Bharadwaj (M.Tech Research, CSA)

Instructors
Prof Shalabh Bhatnagar
Dr. Gugan Thoppe

Mentor
Mohd. Haroon Ansari

Indian Institute of Science, Bangalore

Motivation

- State of the art deep neural network architectures are handcrafted by a team of researchers and engineers.
- These architectures are generally enormous and highly complex.
- New dataset requires efforts from scratch.
- Can we automate this “architecture search”?

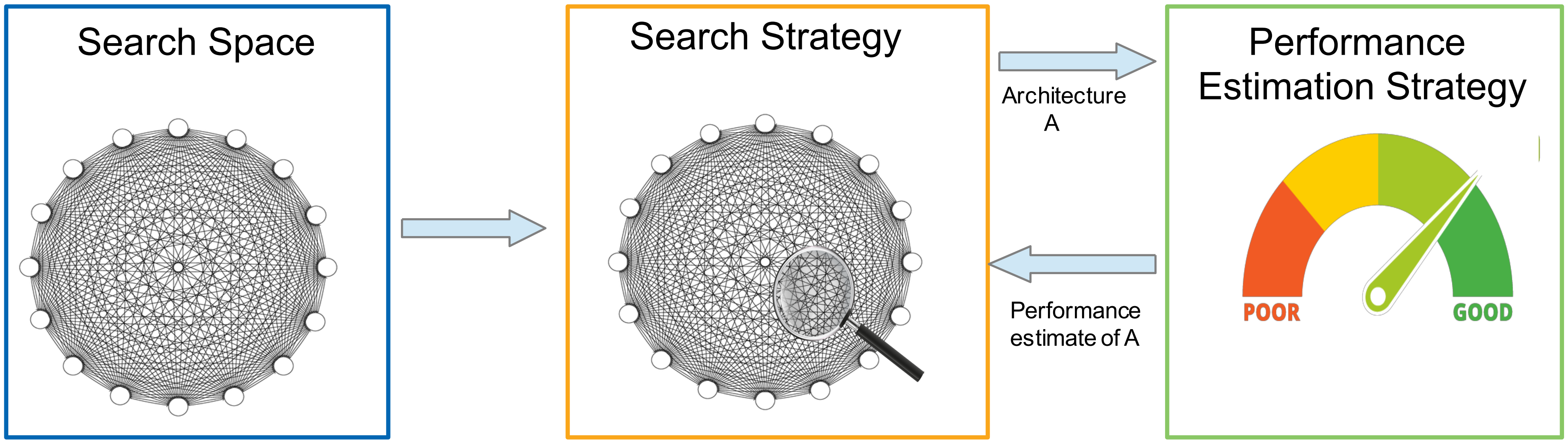
Neural Architecture Search (NAS) :

- Process of automating DNN architecture search.
- Research in this field is in early stages. The goals of NAS and methods to measure the effectiveness of NAS techniques are still evolving.
- **Ideal Goal*** –
  - To search in an **unconstrained search space**
  - with **no compute constraints.**
  - Few attempts into this. Eg – Google AutoML Zero (uses genetic algorithms).
- **Modest Goal*** –
  - To search in the most **efficient** way possible
  - in a **constrained search space**.
  - Considerable literature and good attempts with this goal. Eg – ENAS
- We present an attempt towards the modest goal of NAS in Deep Reinforcement Learning.

*not official terminology by community

Background: Elements of NAS

The problem of Neural Architecture Search has three components (Elsken et al., 2018):



Background : Elements of NAS - Details

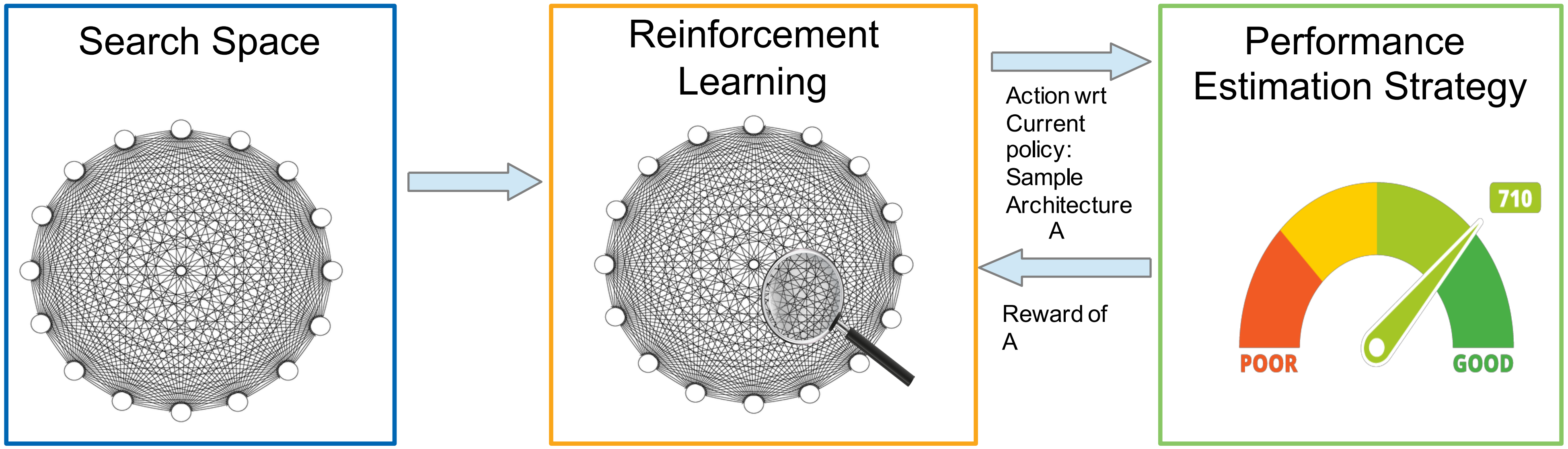
- **Search Space** : Where to search?
    - Search space formulations are usually constrained by prior knowledge about architectures suited for the task at hand. E.g. convolution and pooling filters are included for an image classification problem, and LSTM cell like architectures are included in language modelling task.
    - Incorporating prior knowledge reduces search space but increases human bias.

- **Search Strategy**: How to efficiently explore the search space of architectures for best possible results?
    - Exploration-exploitation trade-off comes into picture here.

- **Performance Estimation Strategy**: How to evaluate a sampled architecture?
    - Naïve estimation using performance metric of the associated task becomes a bottleneck for efficient architecture search.

Background: NAS via RL

Search Strategy : Reinforcement Learning Idea

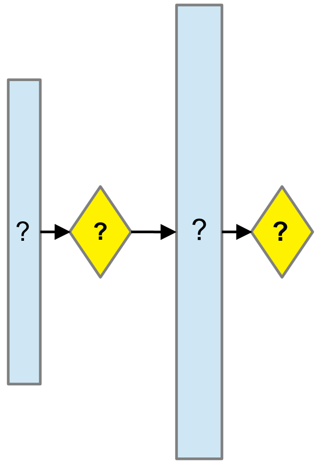

Search Space
Reinforcement Learning
Action wrt Current policy: Sample Architecture A
Reward of A
Performance Estimation Strategy
710
POOR
GOOD

Neural Architecture Search with Reinforcement learning (Zoph, B. and Le, Q. V. ,Google Brain,2016)

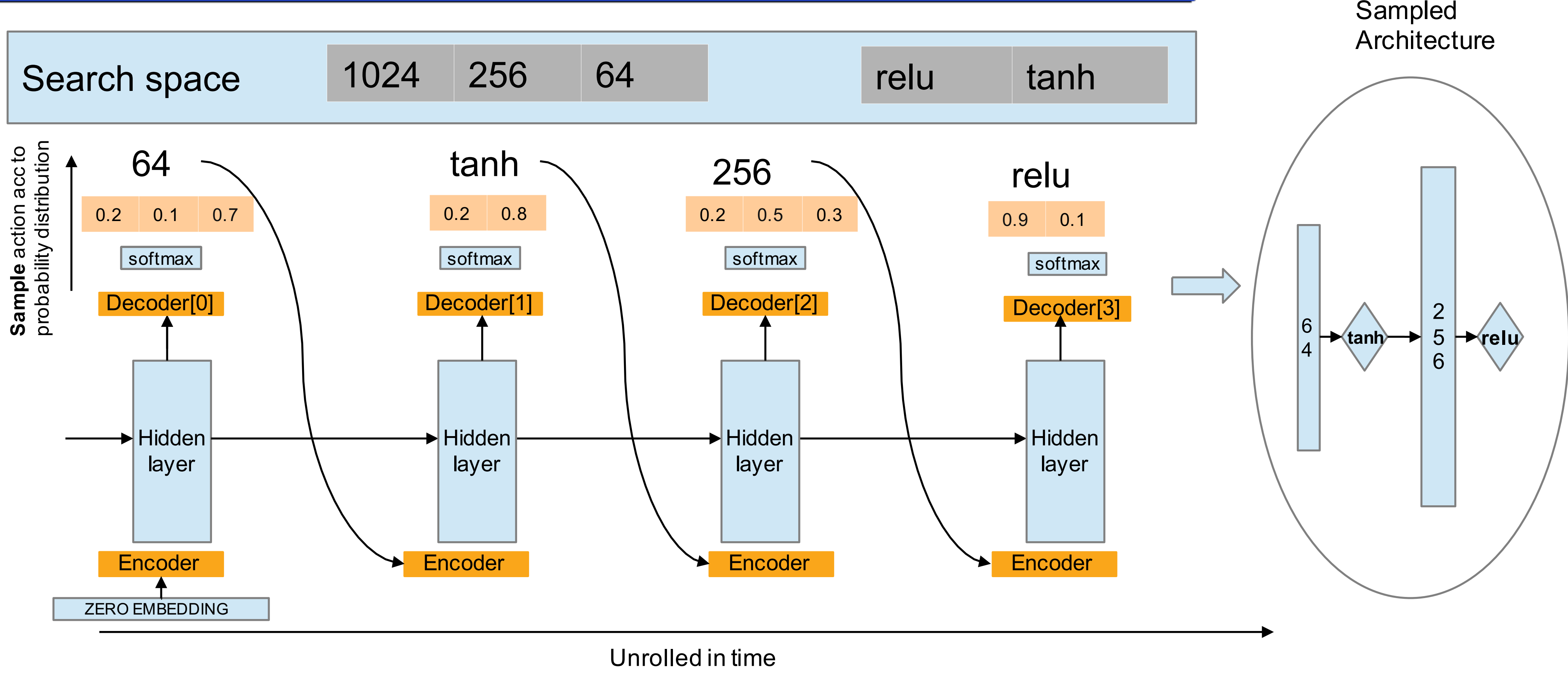
- Formulation of NAS as Reinforcement Learning problem –
  - **State Space** :
    - Constrained search space according to task. E.g. For Image Classification - Conv kernel, Pooling kernel, activation functions etc with limited no of layers, and for language models - recurrent cells and activation functions.
    - Note that search space does not include set of optimizers (sgd,adam,adagrad etc) here.
  - **Reward** : Corresponding Performance metric
    - Image classification- Error rate for CiFAR on held out data.
    - Language modelling- Perplexity for PennTreeBank dataset.
    - Differs from the normal setting in that the reward is not obtained immediately after executing an action but after a series of actions have been executed.
  - **Action**: Generating/Choosing an element of the architecture.
  - **Policy** : Stochastic Policy.
  - **Training algorithm**: Policy Gradient
    - Better convergence properties.
    - Effectiveness in high-dimensional or continuous action spaces
    - Ability to learn stochastic policies

Let’s generate a network

- Generation is done using an RNN Controller- **Parameters of this controller are actually parameters of our policy!**
- E.g. - Generate a 2 layered feedforward NN from a search space of dense layer (1024,256,64) and activation functions (relu,tanh)



Generation of network through RNN Controller



Neural Architecture Search with Reinforcement learning : Training the controller

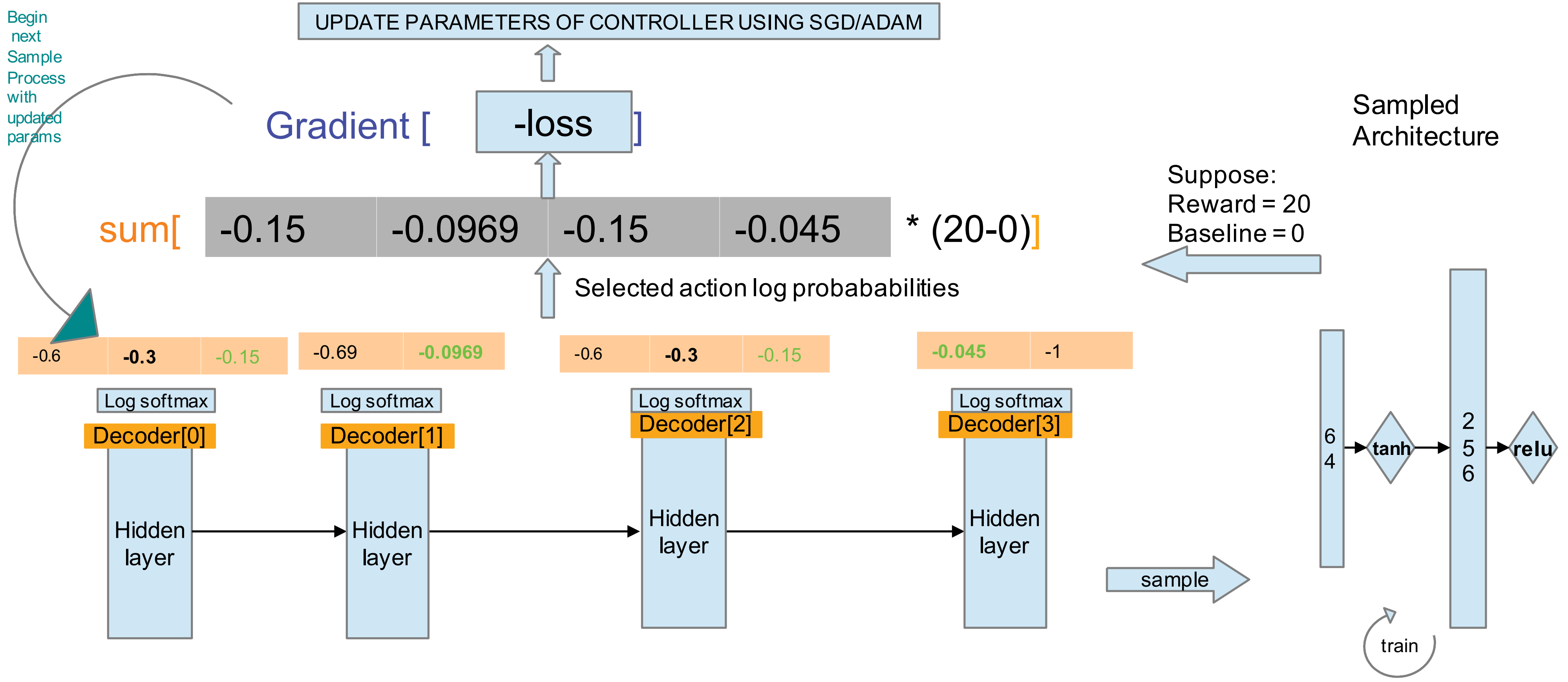
- Policy Gradient Training is done using empirical form of REINFORCE rule with baseline to maximise expected reward -

$$\nabla J(\Theta) = \frac{1}{m} \sum_{k=1}^{m} \sum_{t}^{T} \nabla \, log P(a_t | a_{1:t-1}; \Theta)(R_k - b)$$

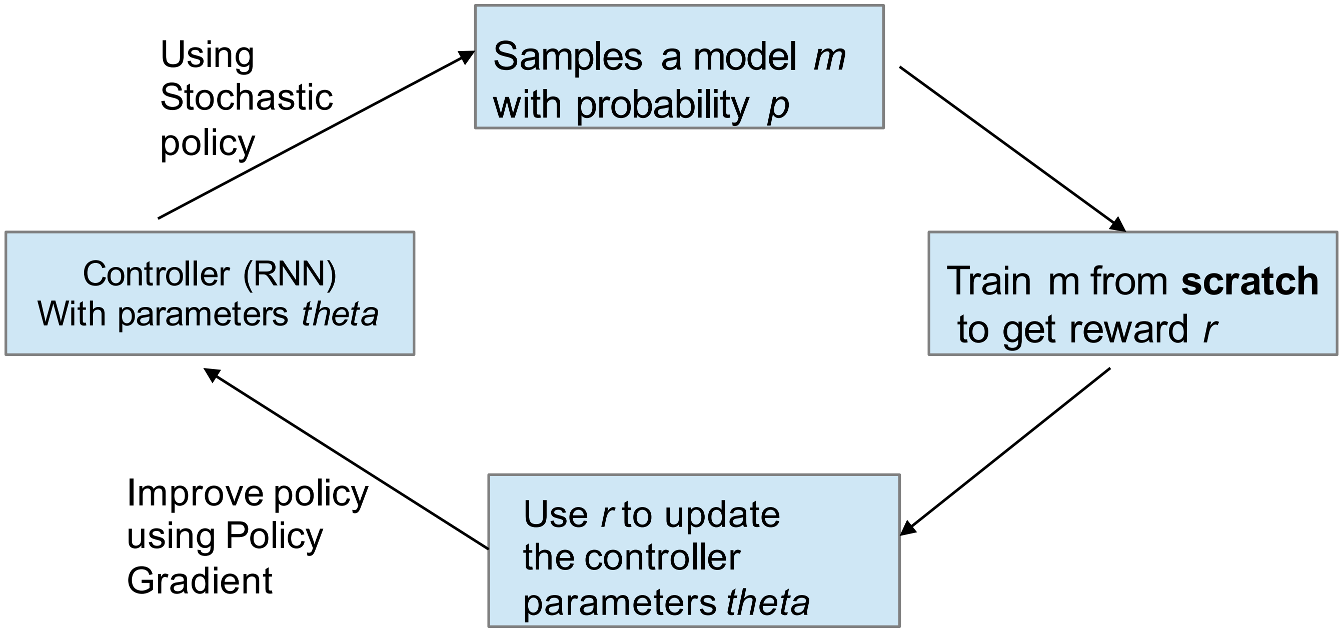
- where
  - ***a**$_t$* are action taken by controller at time step ***t***,
  - ***T*** is no of time steps RNN takes to generate an architecture,
  - ***R**$_k$* is reward of kth architecture,
  - ***m*** is no of architectures generated by controller using same policy in a batch,
  - ***b*** is baseline (in this case exponential moving average of previous architecture accuracies)

- Example run : IN NEXT SLIDE.

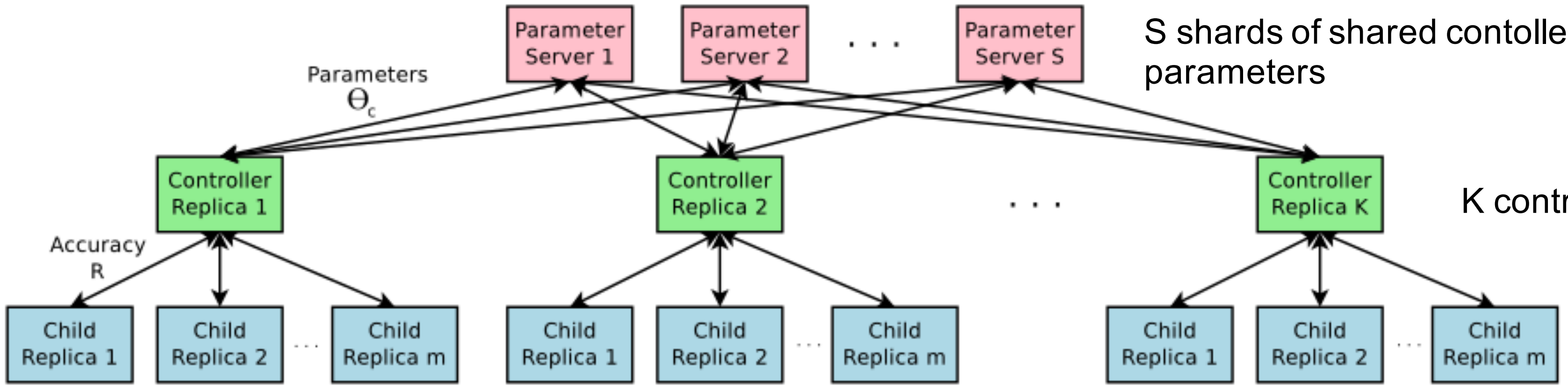
Neural Architecture Search with Reinforcement learning : Training the controller



Neural Architecture Search with Reinforcement learning : Flow


Samples a model *m* with probability *p*
Train m from **scratch** to get reward *r*
Use *r* to update the controller parameters *theta*
Improve policy using Policy Gradient
Controller (RNN) With parameters *theta*
Using Stochastic policy

Neural Architecture Search with Reinforcement learning : Training setup



S shards of shared contoller parameters

K controllers

Each controller training m different child models

*Image Source - Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning, 2016

Neural Architecture Search with Reinforcement learning : Pros & Cons

- ✓ Achieves State of the Art results on CiFAR and PenTreebank datasets
- ✓ Can be used as baseline for other NAS models

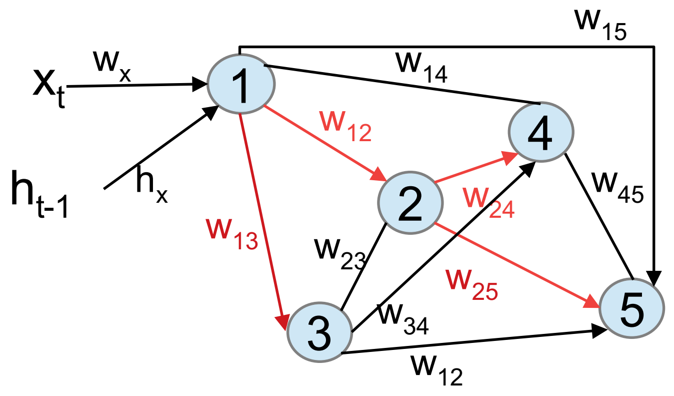
- ✗ Computationally expensive
  - ✗ 800 GPUs used for CiFAR,
  - ✗ 450 GPUs used for PenTreeBank
- ✗ Complexity of implementation due to distributed training setup.

Efficient Neural Architecture Search via Parameter Sharing (Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean. J,Google Brain-Stanford-CMU collab, 2018)

- Search Strategy is still Reinforcement learning based, hence our framework will remain same.
- Two important **changes** -
  - Generation of Architecture : a novel DAG representation.
  - Sharing parameters of child models : to speed up evaluation.

Efficient Neural Architecture Search via Parameter Sharing : Generation

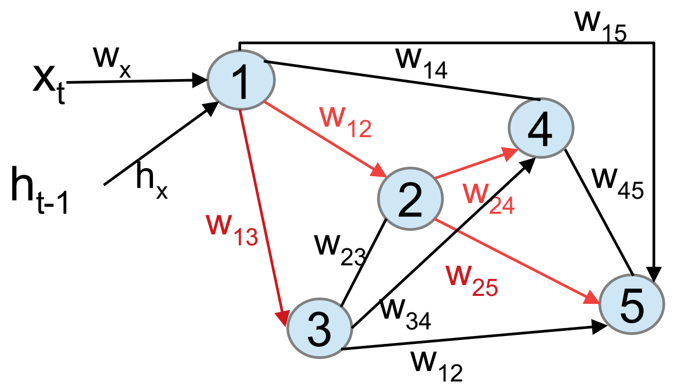
- Main idea - search space of NAS can be represented as a single directed acyclic graph (DAG).
- Sampling an architecture maps to choosing a sub-graph of this large DAG.
- This design allows parameters to be shared between child models.



Chosen subgraph : in red

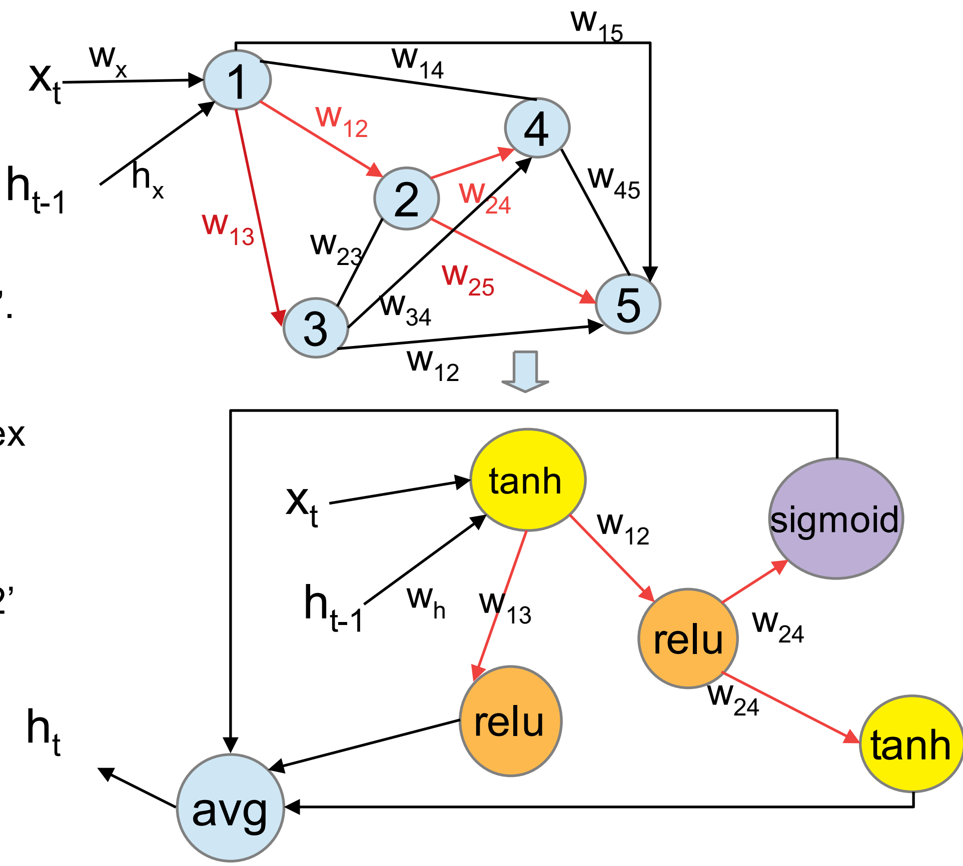
Efficient Neural Architecture Search via Parameter Sharing : Generation of Recurrent cell (for NLP problems) - An example

- Previous state h_{t-1} and input x_t are inputs to node 1.
- w_x and h_x are weight matrices associated with input, initialised randomly.
- W_{ij} is global and is shared among all sampled sub-graphs (this enables parameter sharing).
- let list of activation functions we want to sample from be [tanh, relu, sigmoid].
- We iterate on the nodes and at each node(from 2 onwards) the controller has two decisions to make
    - Which activation function?
    - Which previous node to connect from? Equivalently, which weight matrix w_{ij} must be activated?



Efficient Neural Architecture Search via Parameter Sharing : Generation of Recurrent cell (for NLP problems) - Example contd...

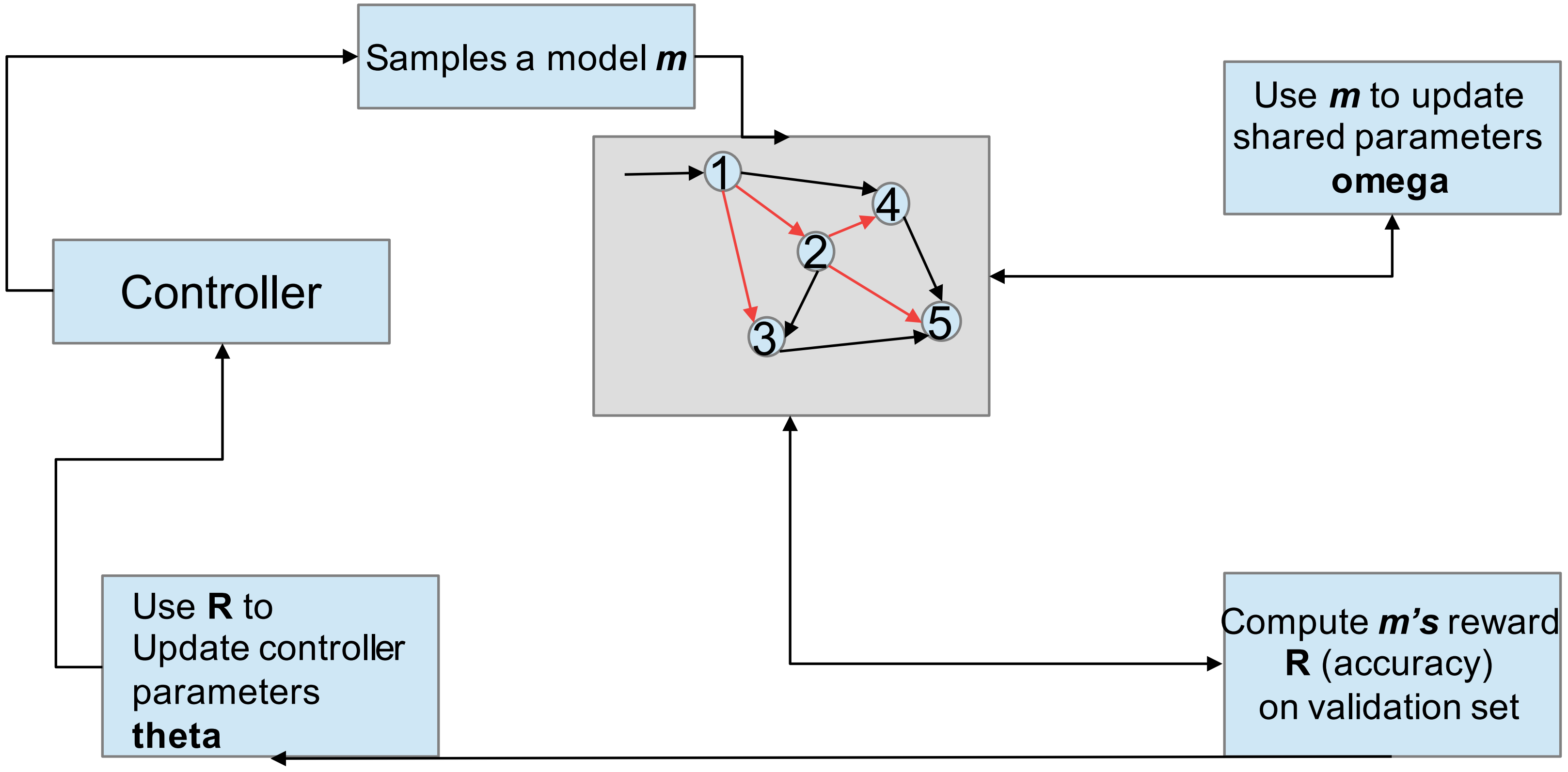
- At node 1 controller samples activation function ‘tanh’,
- $h_1 = \tanh(x_t.W_x + h_{t-1}.W_h)$
- At node 2, controller sample ‘relu’ and prev node index ‘1’ which activates globally shared w_{12}
- $h_2 = \text{relu}(w_{12}.h_1)$
- At node 3, controller samples ‘relu’ and prev node index ‘1’.
- $h_3 = \text{relu}(w_{13}.h_1)$
- At node 4, controller samples ‘sigmoid’ and prev node index ‘2’.
- $h_4 = \text{sigmoid}(w_{24}.h_2)$
- At node 5, controller samples ‘tanh’ and prev node index ‘2’
- $h_5 = \tanh(w_{25}.h_2)$
- Final output : h_t = $\text{mean}(h_4,h_5,h_3)$



Efficient Neural Architecture Search via Parameter Sharing : Training

- There are two sets of learnable parameters:
  - the parameters of the controller LSTM, denoted by θ, and
  - the shared parameters of the child models, denoted by ω.
- The training procedure of ENAS consists of two interleaving phases.
  - Updating **ω**
    1. Fix the controller policy and sample model **m**
    2. Train child models using SGD on **ω** to minimize the task loss function. E.g, Loss function is **cross-entropy loss L(m;ω)** computed on mini-batch of training data,
  - Updating **θ :**
    1. Fix **ω** and obtain *reward* from task
    2. Update the policy paramters **θ** aiming to maximise expected reward [using Monte Carlo Policy Gradient (REINFORCE) described earlier.]

Efficient Neural Architecture Search via Parameter Sharing : Flow


Samples a model *m*
Use ***m*** to update shared parameters **omega**
Controller
1
2
3
4
5
Use **R** to Update controller parameters **theta**
Compute ***m’s*** reward **R** (accuracy) on validation set

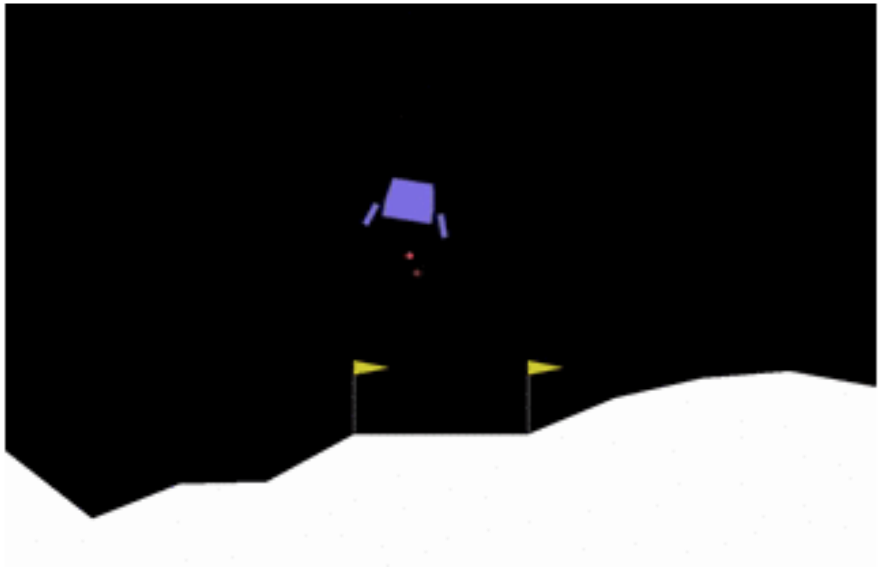
Efficient Neural Architecture Search via Parameter Sharing : Pros & Cons

- ✓ Achieves State of the Art results on PenTreebank datasets and almost beats NAS on CiFAR.
- ✓ Required Nvidia GTX 1080i for 16 hours for training PenTreeBank dataset.

- ✗ Cannot search through optimizers in theory as it will require freezing of shared parameters.
- ✗ Human bias in designing the search space.

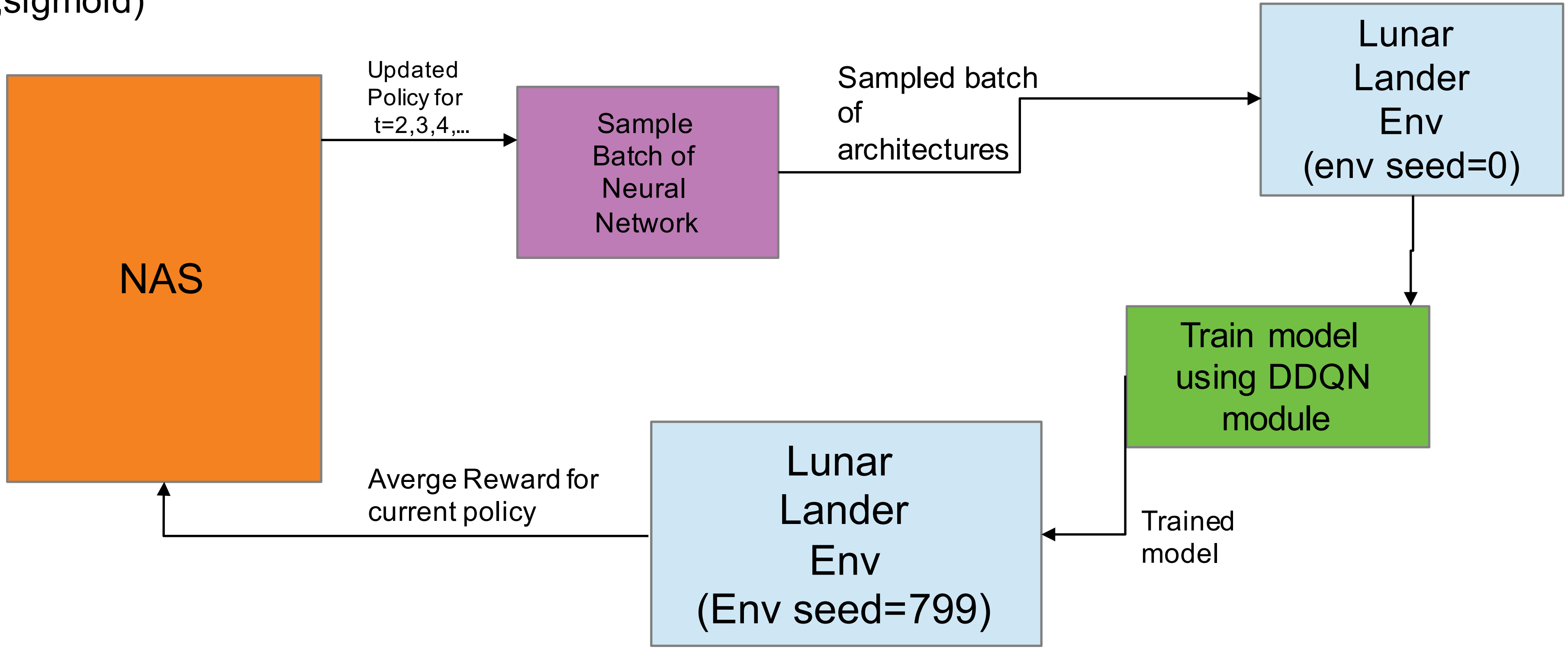
Open AI Lunar Lander Environment

- Goal is to land lunar lander on surface between the flags.
- State space size = 8
- Action size = 4
- Imperfect landing and crashes attract penalties.

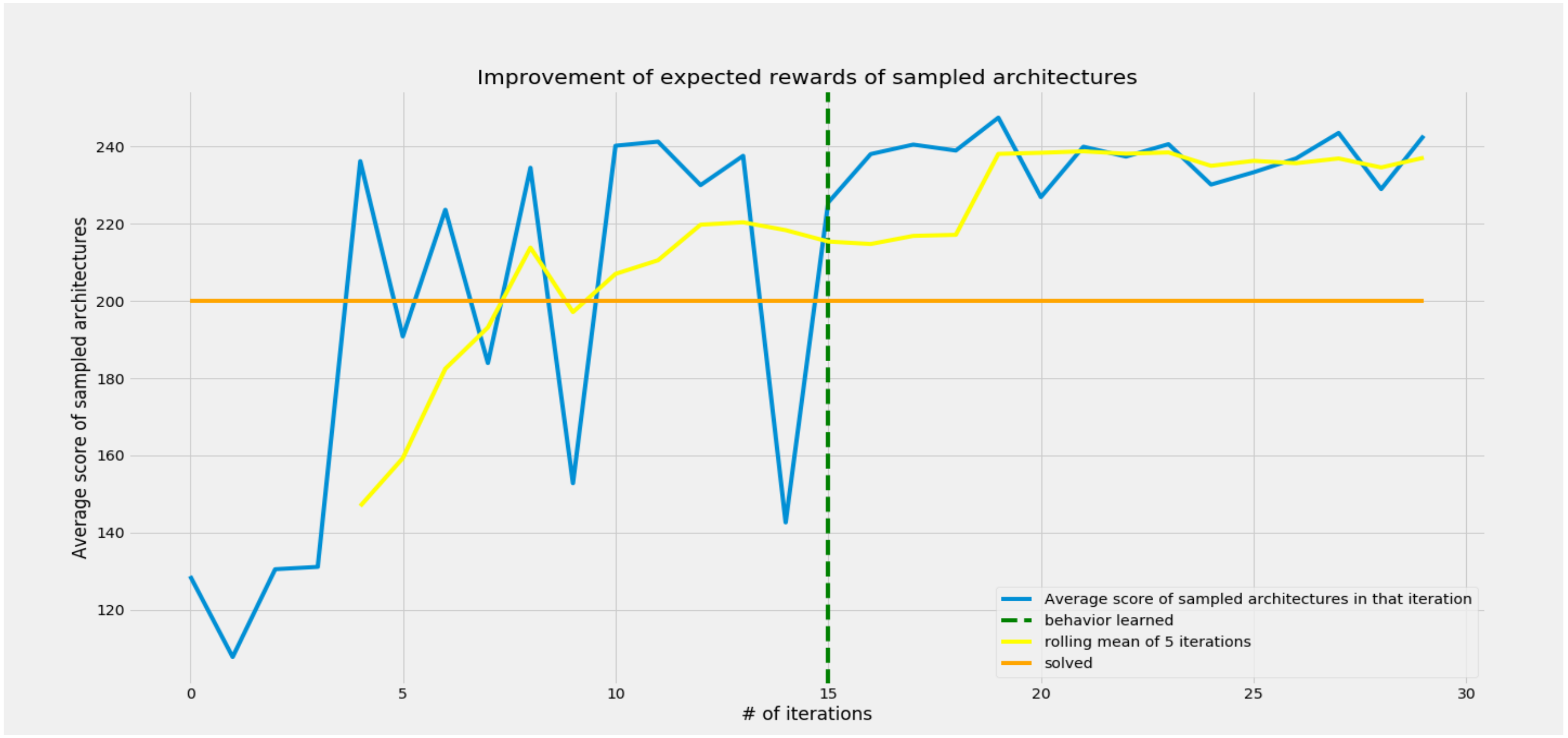


Our Attempt : ENAS on DDQN Neural Network Approximation

- For a start we tried ENAS for searching a best 2 layered neural network architecture for Lunar Lander v2 environment used in DDQN approximation possible from dense (64,128,256,1024,2048) and (relu,sigmoid)


NAS
Updated Policy for t=2,3,4,...
Sample Batch of Neural Network
Sampled batch of architectures
Lunar Lander Env (env seed=0)
Train model using DDQN module
Trained model
Lunar Lander Env (Env seed=799)
Averge Reward for current policy

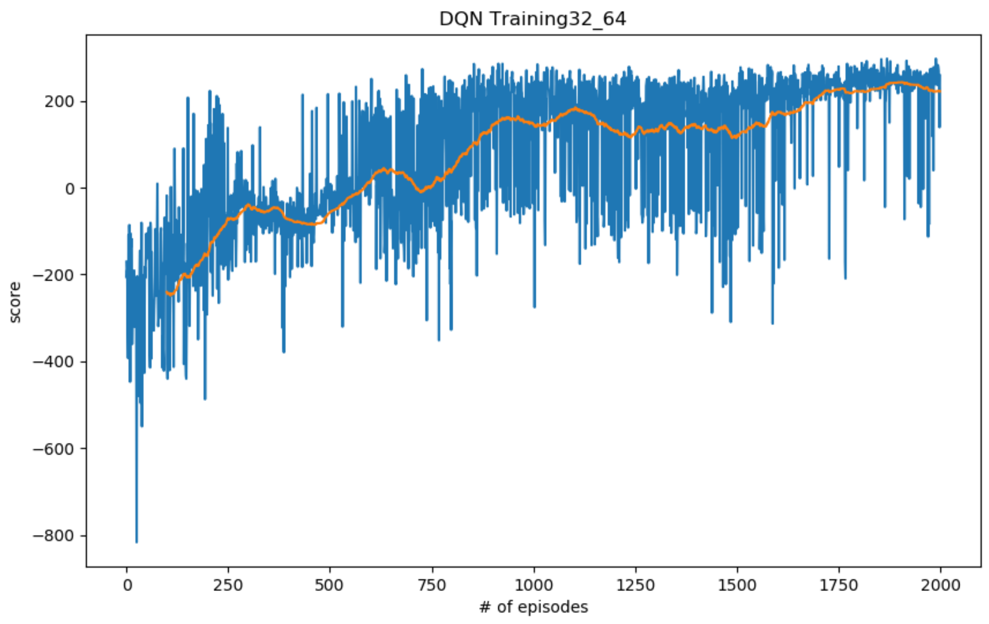
Results :

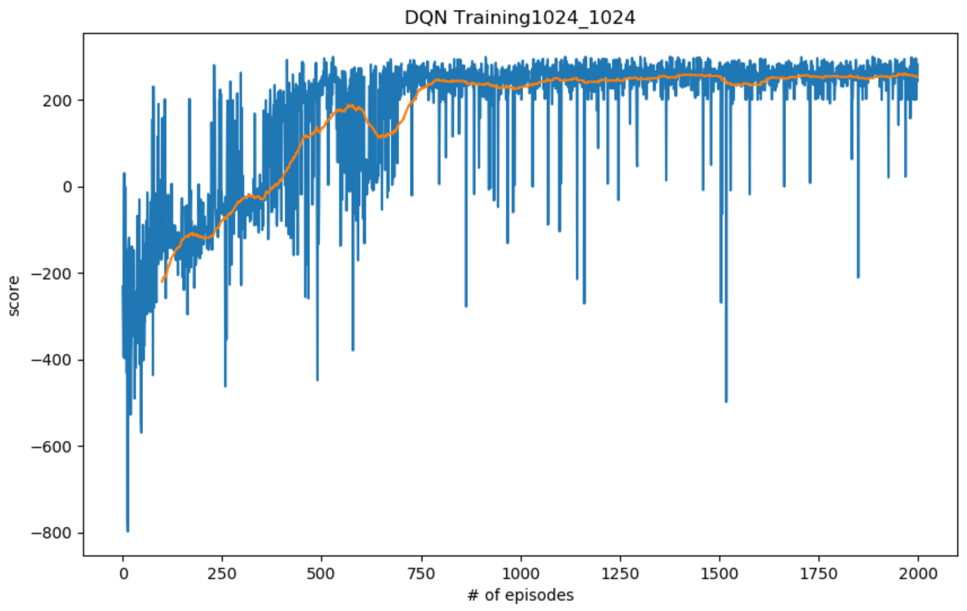

Improvement of expected rewards of sampled architectures
Average score of sampled architectures
# of iterations
120
140
160
180
200
220
240
0
5
10
15
20
25
30
Average score of sampled architectures in that iteration
behavior learned
rolling mean of 5 iterations
solved

Results :

- Time taken : ~12 hrs on single NVIDIA GTX 1050 Ti
- Brute force exhaustive search : ~20 hrs with same setup

Effects of Network Architecture on DDQN training

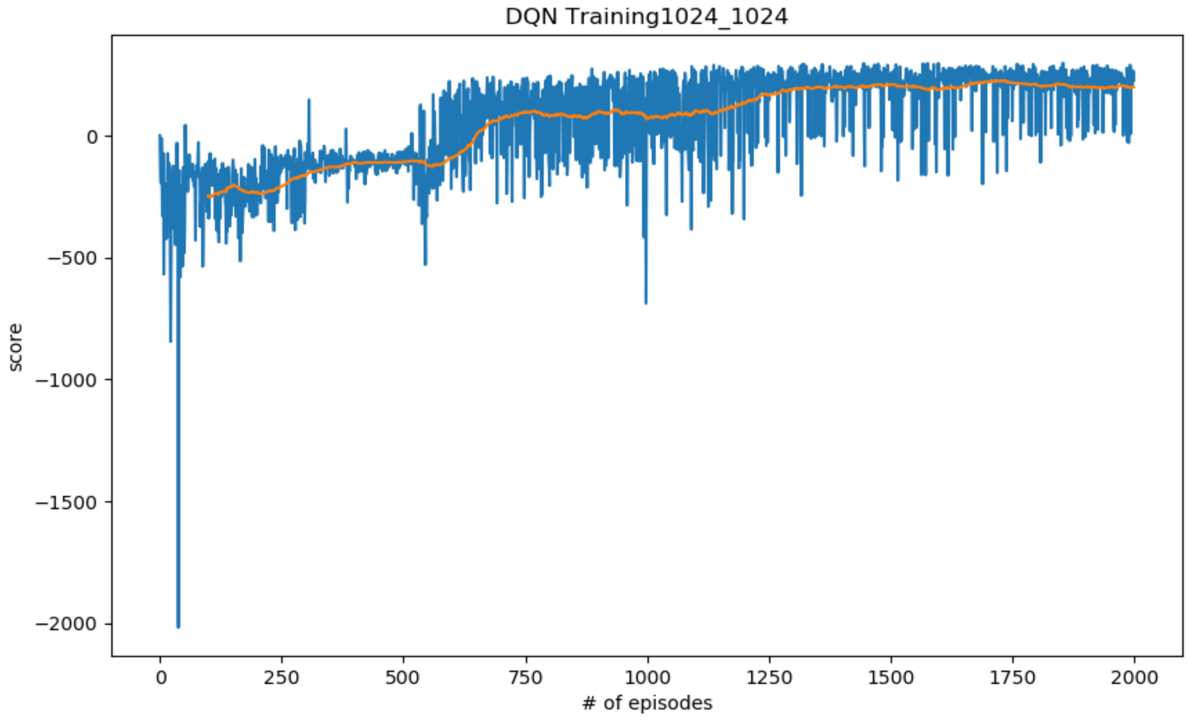




32,relu,64,rel
u

1024,relu,1024,rel
u

Effects of Network Architecture on DDQN training



1024,sigmoid,1024,sigmoi
d

Limitations & Other works

- Limitation with our experiment of ENAS on Deep RL is randomness in training data because every child model sees different data unlike normal supervised setup due to nature of training algorithm even though our env seed is same, however architectures found are good irrespective of this.
- Talking about other methods, there are other search techniques as well like gradient based (Liu et al., 2018) which uses continuous architecture space, Bayesian Optimization based(Kandasamy et al., 2018) and evolutionary based(Real et al., 2018).
- Techniques till now suffer from one big disadvantage, human bias in search space and the paper (Yang et al., 2019) critically investigated it and found that success of this techniques are attributed to carefully crafted architecture spaces and hence random search from same search space give competitive models as well. Google AutoML Zero is an attempt to reduce this human bias in NAS but still in proceedings.