Neural Architecture Search in Deep Reinforcement Learning

Ashish Kumar Jayant (M.Tech Research, CSA)
Shikhar Bharadwaj (M.Tech Research, CSA)
Indian Institute of Science, Bangalore

Design of Deep Neural Network Architectures

- State of the art deep neural network architectures are handcrafted by team of researchers and engineers.
- These architecture are generally enormous and highly complex.
- New dataset requires effort from scratch.
- How can someone come up with this for any dataset they have, provided we can trade some compute and time?
- Can we design a efficient search strategy to search through space of neural network architectures?

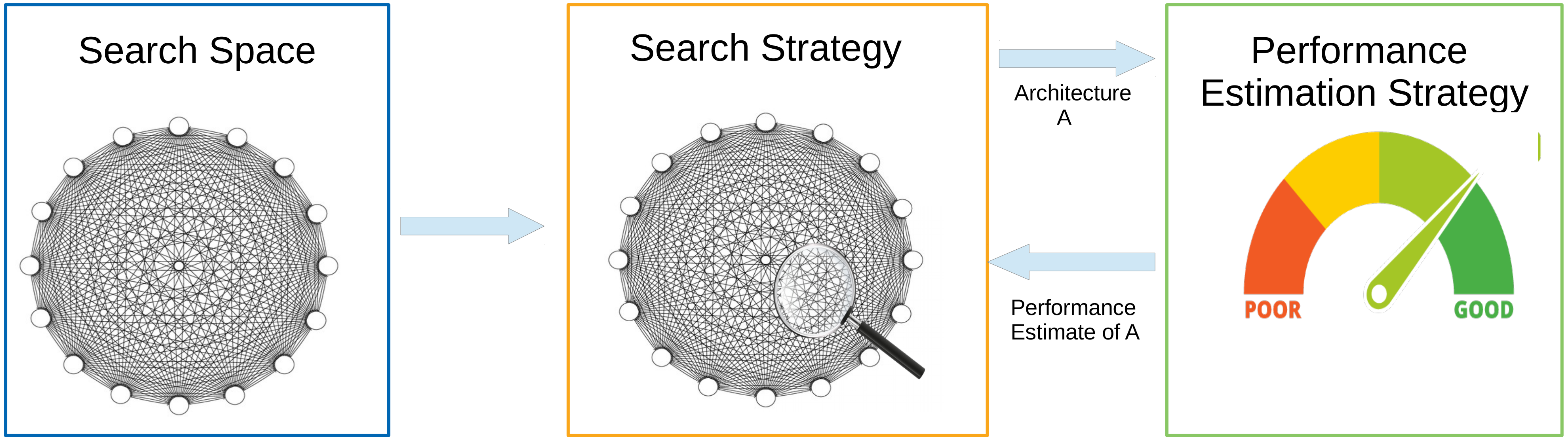
Neural Architecture Search (NAS) :

- Process of automating the architecture design part of Deep Learning neural networks.
- Research in this field is in early stages and the goal of NAS and methods to measure the effectiveness of NAS techniques is still evolving.
- **Ideal Goal*** – To come up with the **best possible architecture** for a problem or dataset which does not involve human bias in its design. Some people have made attempts into this. They are not caring about compute requirements for now in this. E.g – Google Auto ML Zero using genetic algorithms.
- **Modest Goal*** – To come up with **best architecture possible from a given bounded search space** for a problem & do that in most efficient way possible. There is considerable literature and good attempts with this goal.
  - Efficiency here includes time taken & cost of computation.
- We will show an attempt towards the modest goal of NAS in Deep Reinforcement Learning.

*not official terminology by community

Elements of NAS

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



Elements of NAS : Details

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

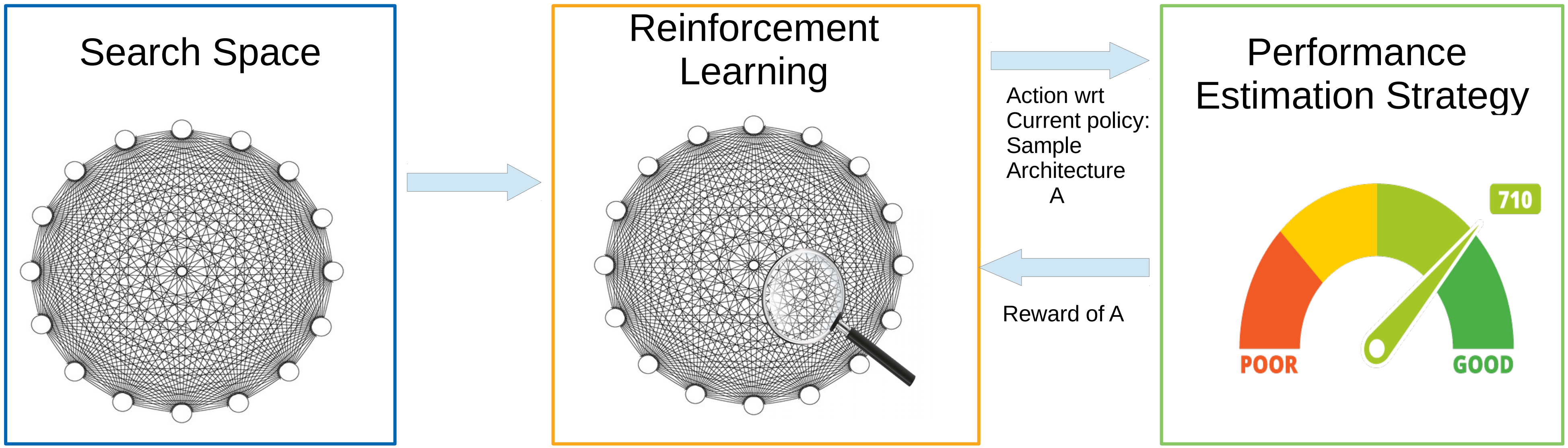
• **Search Space** : The search space consists of all architectures that can be represented in principle. However we use someprior knowledge about elements of architectures suited for a particular task for e.g we will include convolution and maxpool filters for a image classification problem in search space. Incorporating prior knowledge reduces search space but increases human bias.

• **Search Strategy**: The search strategy is about how to explore the search space of architectures in an efficient way as well as we do not miss an optimal architecture. Exploration-exploitation tradeoff needs to balanced here.

• **Performance Estimation Strategy**: When we come up with an architecture, we evaluate its performance but doing that conventional fashion may result in computationally expensive NAS model, so we need strategy to do evaluation of our sampled models in efficient way as well.

NAS via RL

Search Strategy : Reinforcement Learning Idea

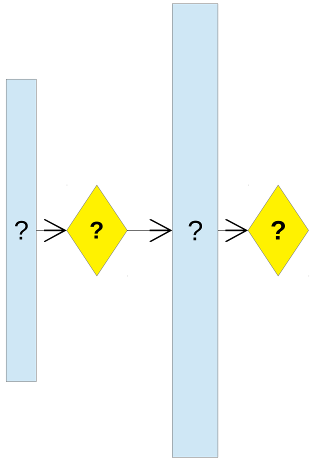

Search Space
Reinforcement Learning
Action wrt Current policy: Sample Architecture A
Reward of A
Performance Estimation Strategy
710
POOR
GOOD

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

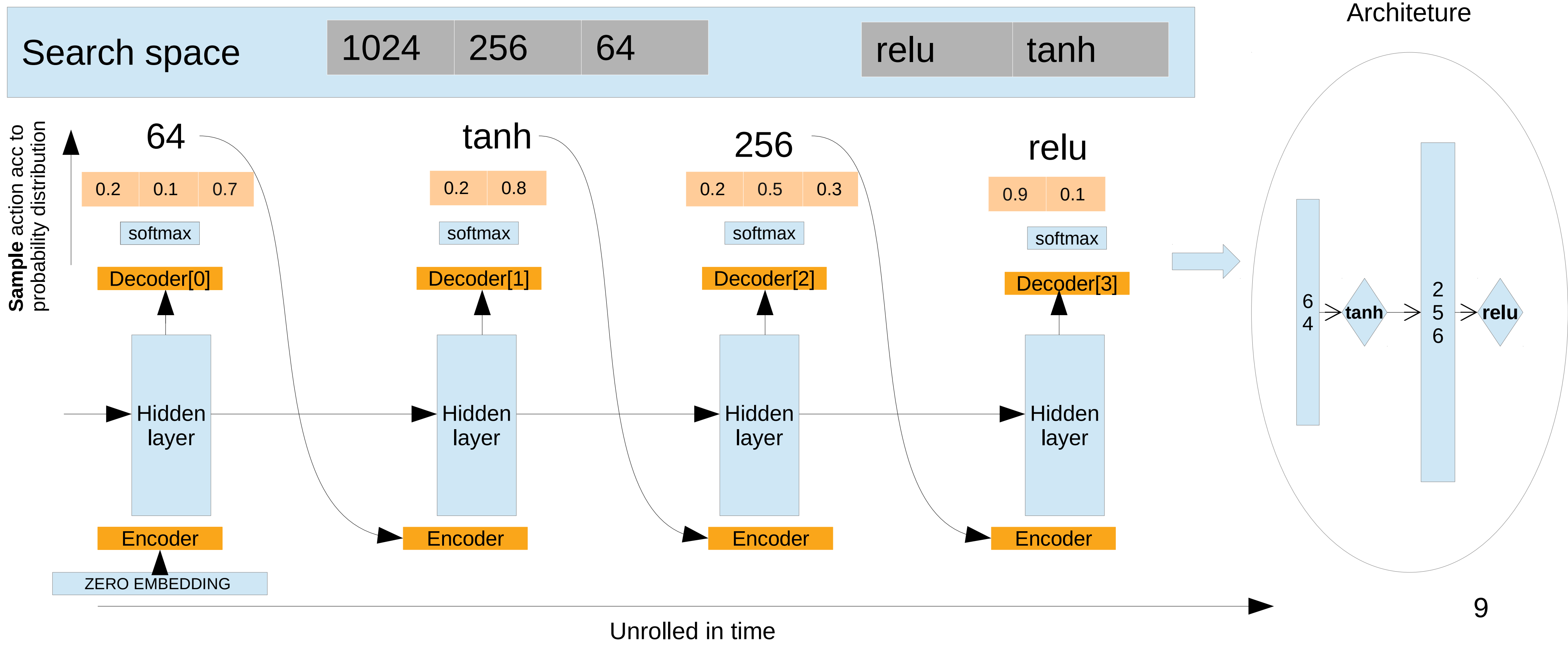
- Formulation of NAS as Reinforcement Learning problem –
  - **State Space** : set of possible architectures
    - So as to bound the search space we have some predefined elements for e.g Image Classification datasets we will have Conv kernel, Pooling kernel,activation functions etc with limited no of layers and language models will have recurrent cells and activation functions.
    - Note that search space do not include set of optimizers (sgd,adam,adagrad etc) here.
  - **Reward** : Corresponding Performance metric (for e.g Error rate for CiFAR,Perplexity for PennTreeBank dataset)
  - **Action**: Generating/Choosing an architecture
  - **Policy** : Stochastic Policy i.e, we will have probability of choosing an architecture not a determinsitic policy.
  - **Training mechanism** : 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 is actually parameters of our policy!**
- e.g. - Generate a 2 layered feedforward nn possible from 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 controller

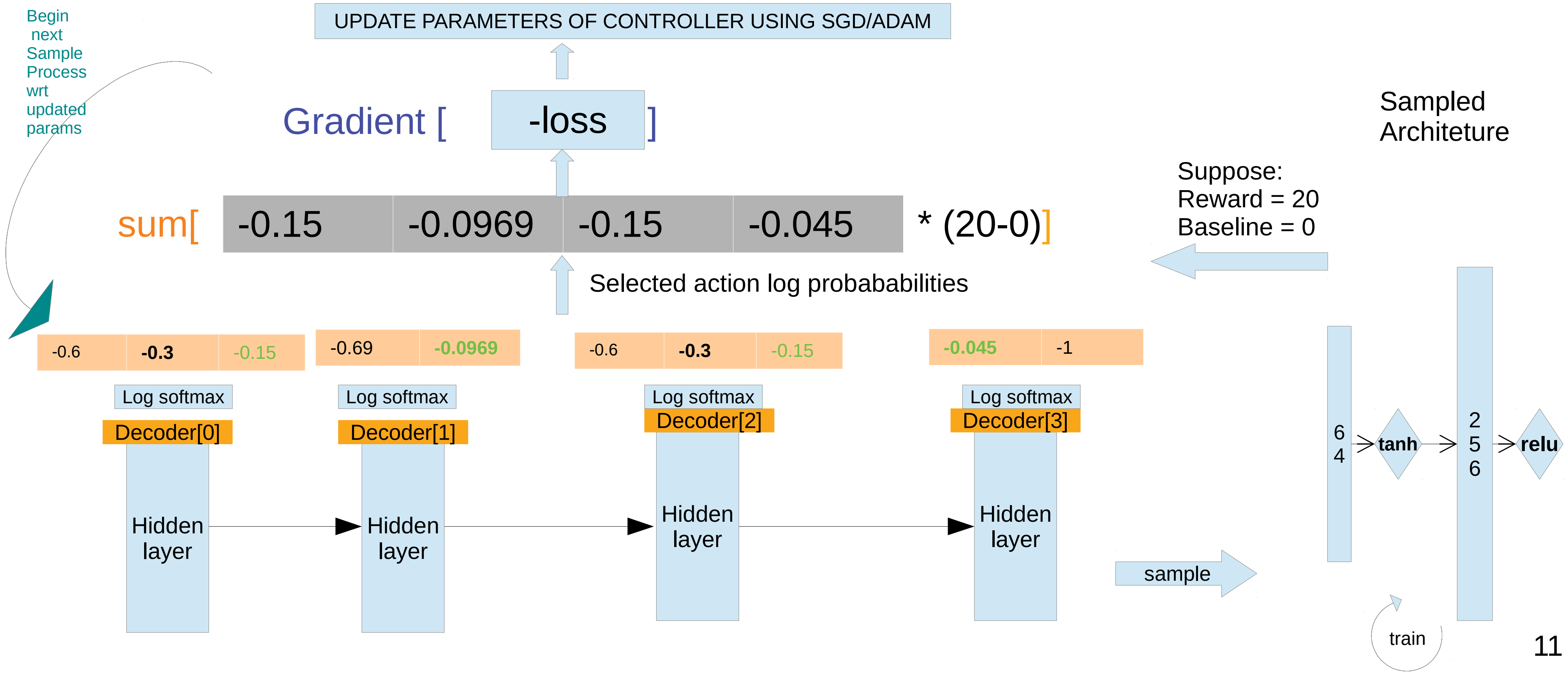
- Policy Gradient Training is done using empirical form of REINFORCE rule with baseline to maximise expected reward -

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

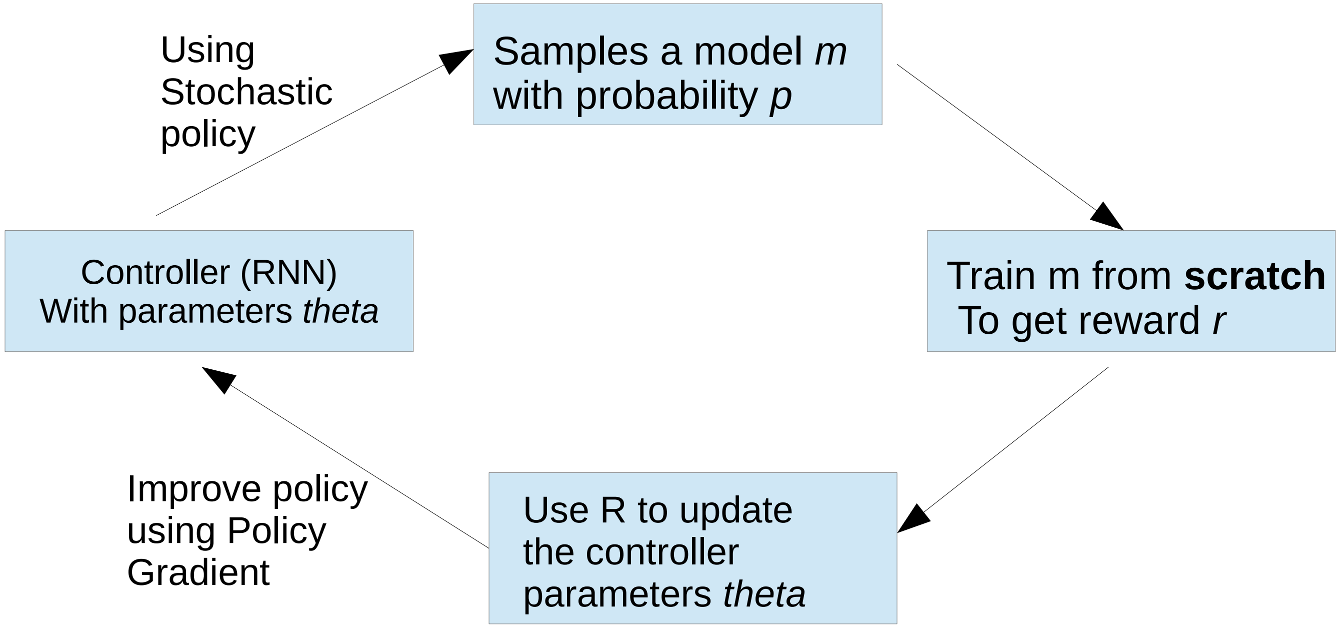
where $\boldsymbol{a_t}$ are action taken by controller at time step ***t***, ***T*** is no of time steps RNN takes to generate an architecture, $\boldsymbol{R_k}$ is reward of kth architecture, ***m*** is no of architectures generated by controller using same policy in a batch, ***b*** is baseline (in our case exponential moving average of previous architecture accuracies)

- Example run : IN NEXT SLIDE.

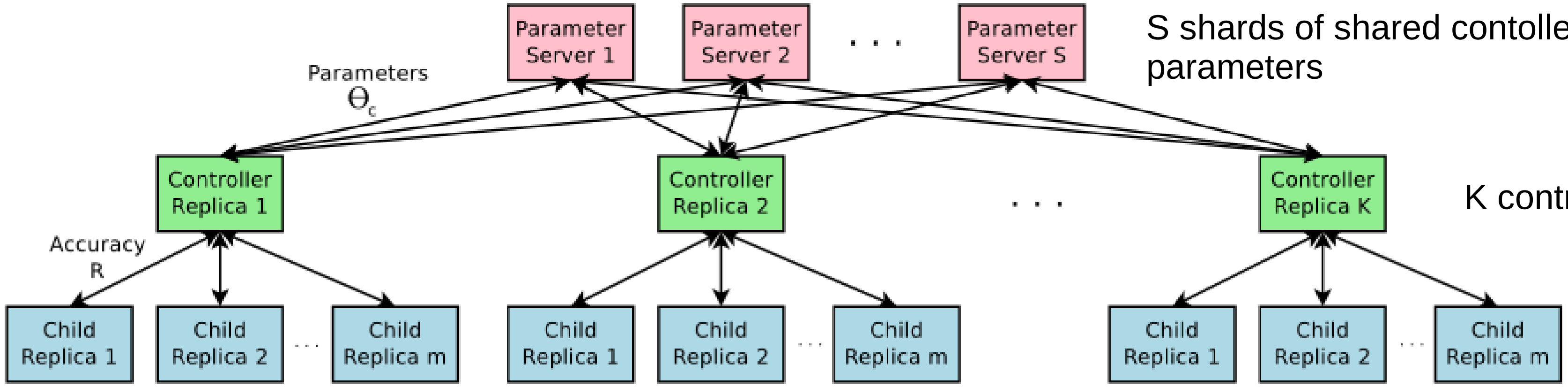
Neural Architecture Search with Reinforcement learning : Training controller



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 Controllers 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

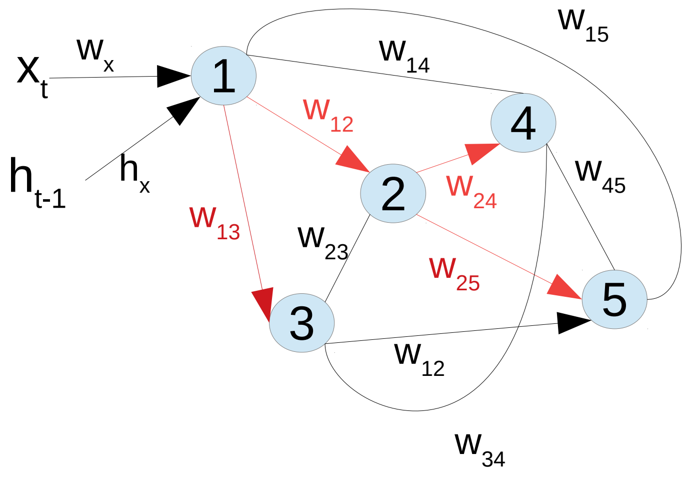
- ✗ Computationally expensive
  - ✗ 800 GPUs used for CiFAR,
  - ✗ 450 GPUs used for PenTreeBank
- ✗ Complexity of implementation due to distributed training setup.
- ✗ Suffers with disdavantages of policy gradient method.

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
  - Sharing of parameters of child models instead of re-training it again.

Efficient Neural Architecture Search via Parameter Sharing : Generation

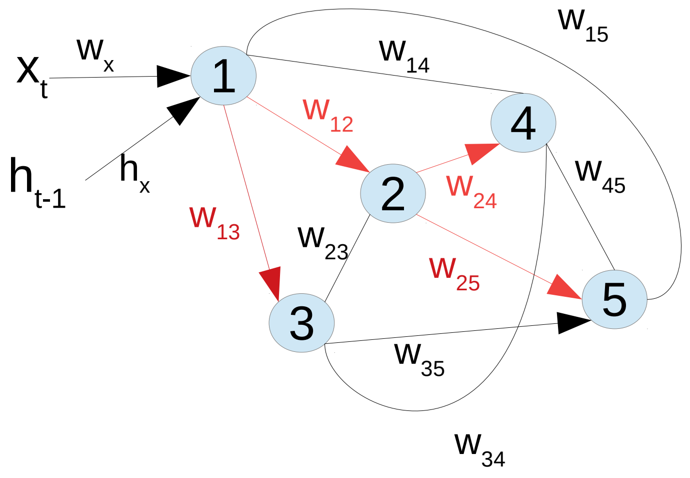
- Main idea is that we can represent search space of NAS as a single directed acyclic graph (DAG).
- Choosing/sampling an architecture will mean choosing a sub-graph of this large DAG.
- This design also allows parameters to be shared between child models.



Chosen subgraph : in red

Efficient Neural Architecture Search via Parameter Sharing : Generation of Recurrent cell (for text based problems) example

- Assume node 1 to be input node, h_{t-1} and w_x are initialized by us and let list of activation functions we want to sample from be [tanh,relu,sigmoid].
- W_{ij} are global and is shared among all sampled sub-graphs which enables parameter sharing.
- Controller have two decision to make
  - Which activation function?
  - Which previous node to connect from or which weight matrix w_{ij} will be activated?



Efficient Neural Architecture Search via Parameter Sharing : Generation of Recurrent cell (for text based 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}

  $h_2 = \mathrm{relu}(w_{12}.h_1)$

- At node 3, controller samples ‘relu’ and prev node index ‘1’.

  $h_3 = \mathrm{relu}(w_{13}.h_1)$

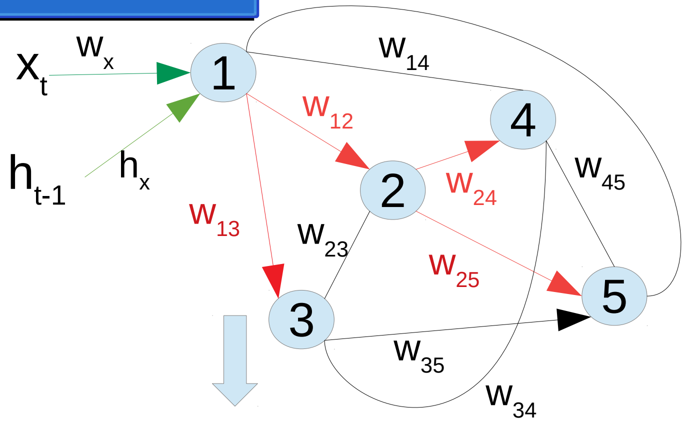
- At node 4, controller samples ‘sigmoid’ and prev node index ‘2’.

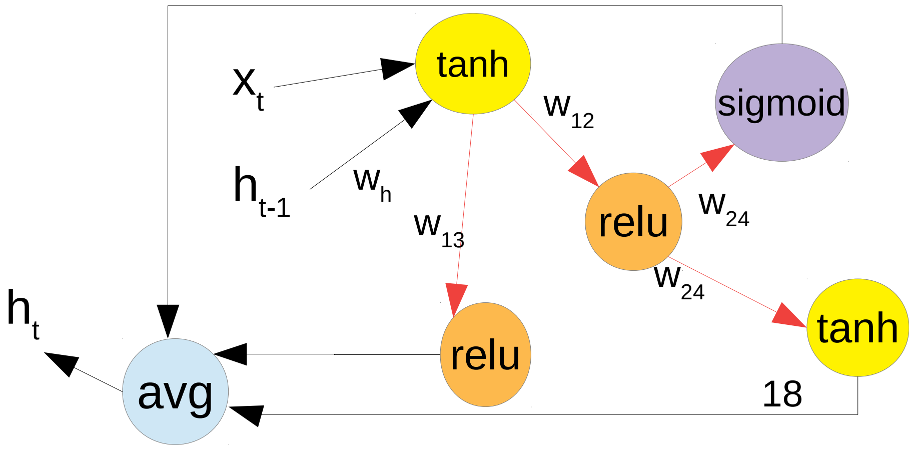
  $h_4 = \mathrm{sigmoid}(w_{24}.h_2)$

- At node 5, controller samples ‘tanh’ and prev node index ‘2’

  $h_5 = \tanh(w_{25}.h_2)$

- Final output : $h_t = \mathrm{mean}(h_4,h_5,h_3)$

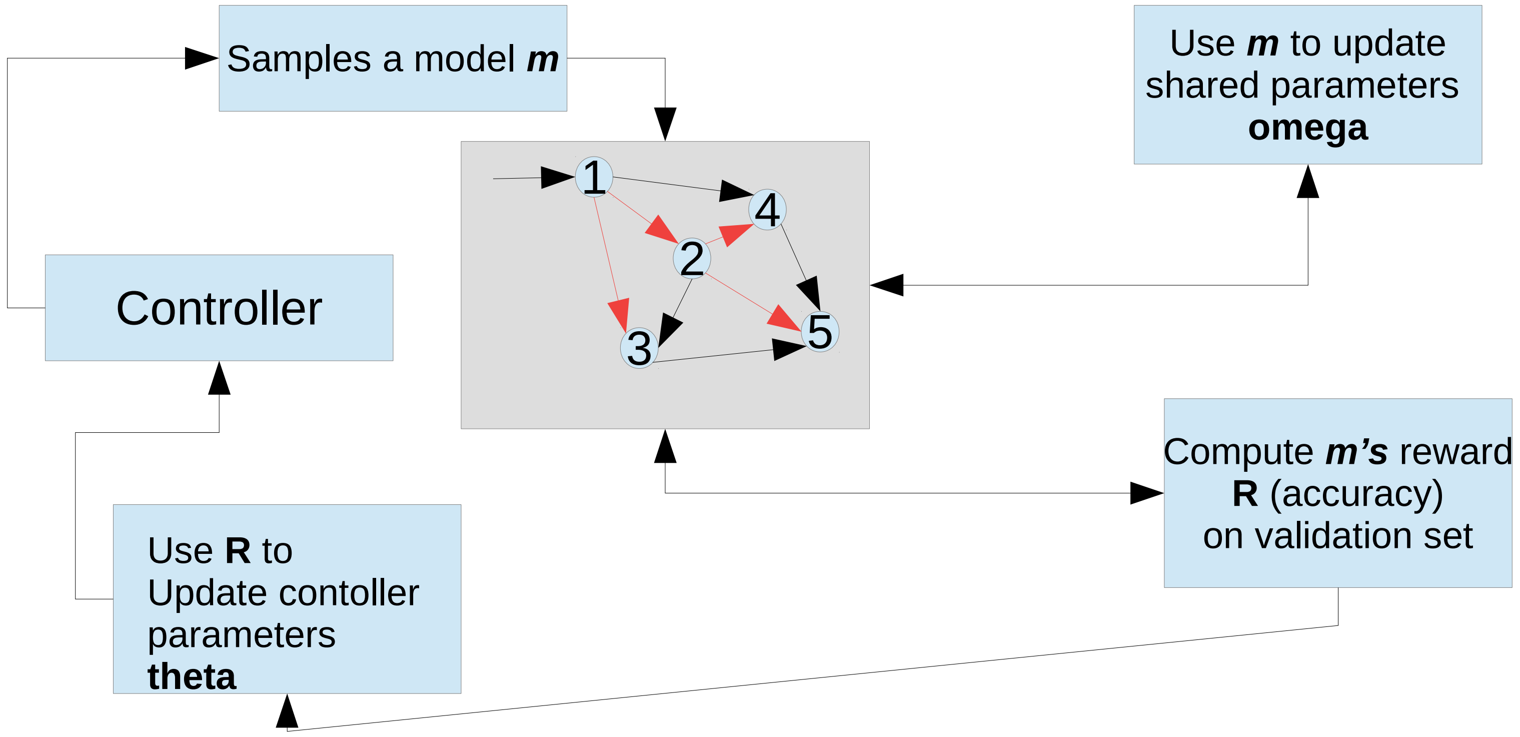




Efficient Neural Architecture Search via Parameter Sharing : Training

- 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.
  - Fix the controller policy and train child models using SGD on **ω** to minimize the expected loss function.
    - Loss function is **cross-entropy loss L(m;ω)** computed on mini-batch of training data, model **m** sampled using parametrized stochastic policy **π(m;θ)** then perform SGD(**batch_size**, **m**)
  - Fix the **ω** and update the policy paramters **θ** aiming to maximise expected reward using Monte Carlo Policy Gradient (REINFORCE) described in slide 7.

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
Compute *m’s* reward **R** (accuracy) on validation set
Use **R** to Update contoller parameters **theta**

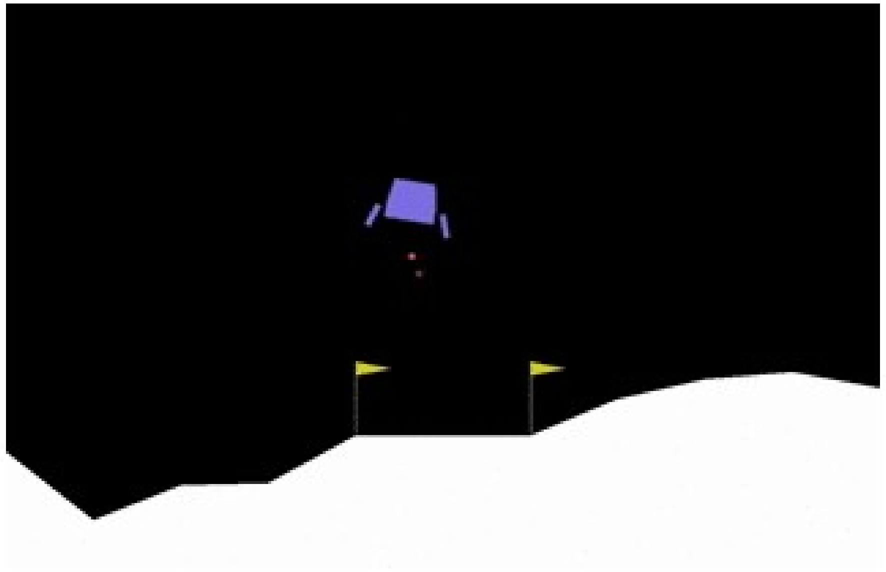
Efficient Neural Architecture Search via Parameter Sharing : Pros & Cons

- ✔ 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.
- ✗ Suffers with disdavantages of policy gradient method.
- ✗ Human bias in designing the search space.

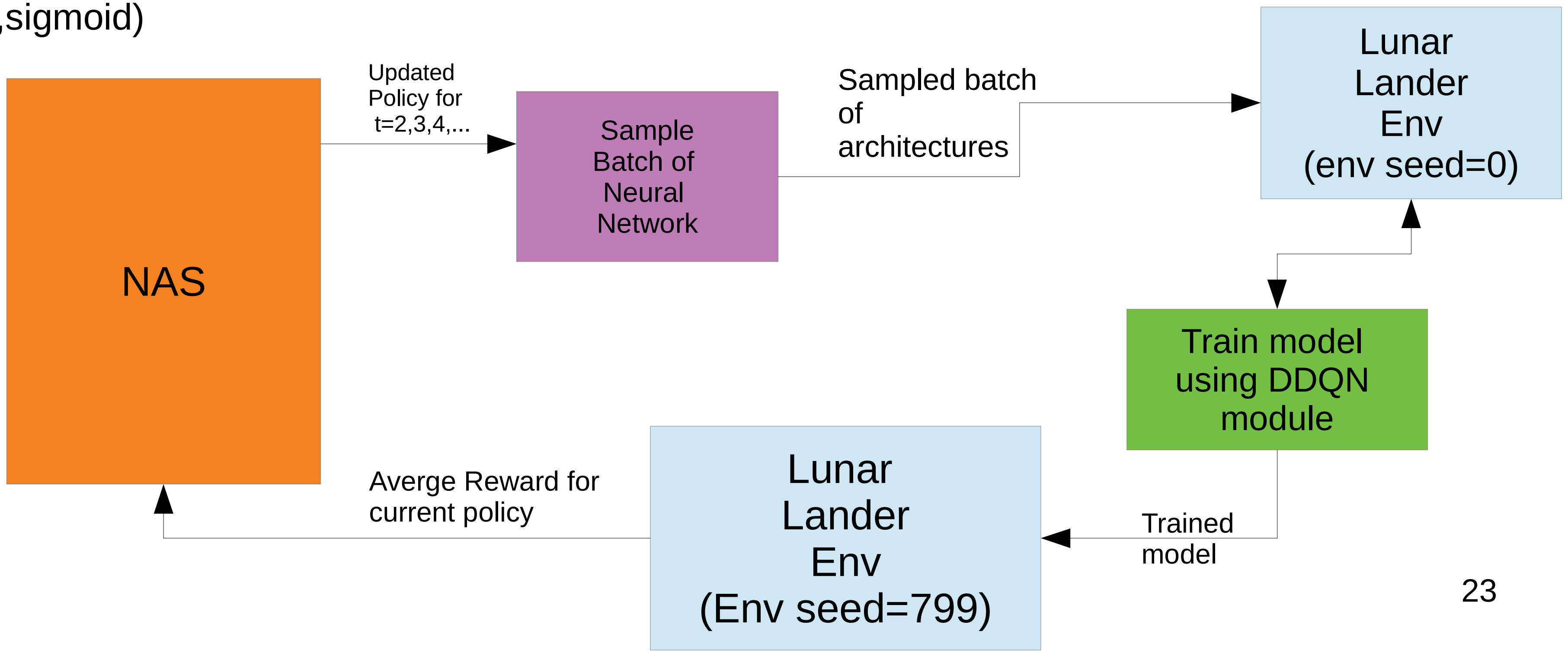
Open AI Lunar Lander Environment

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

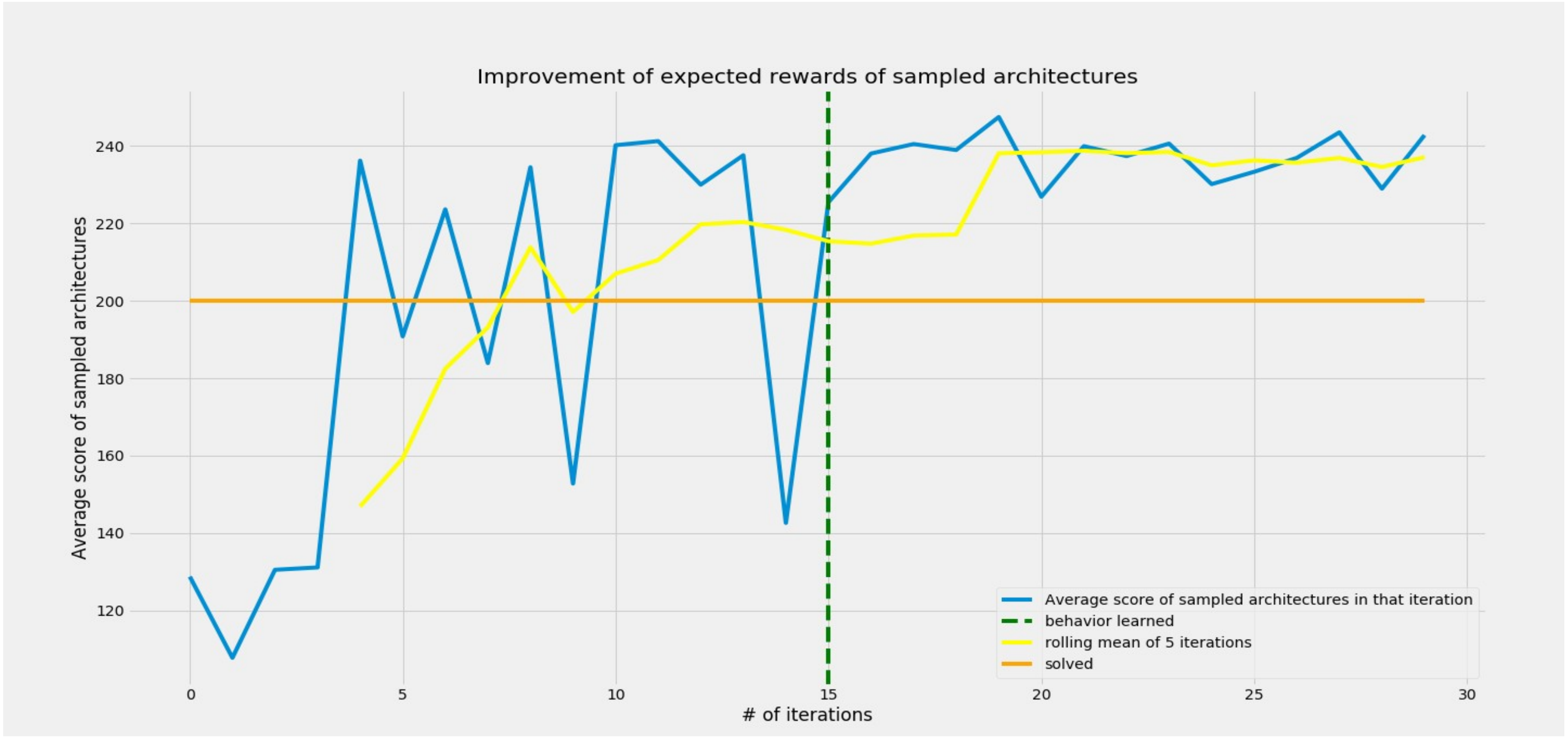


Our Attempt : ENAS on DDQN Neural Network Approximation

- 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

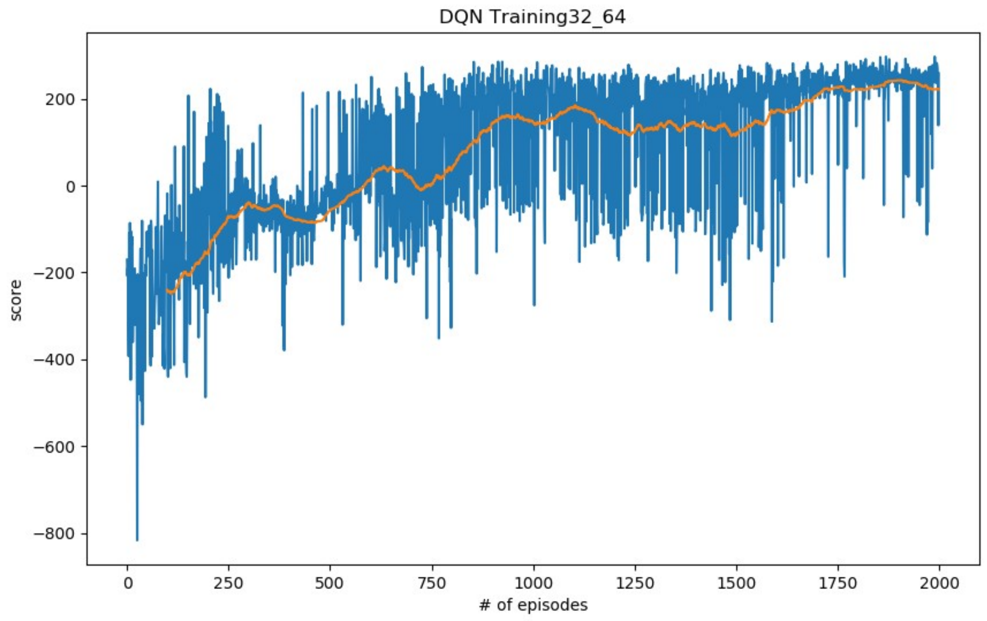
Results :


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

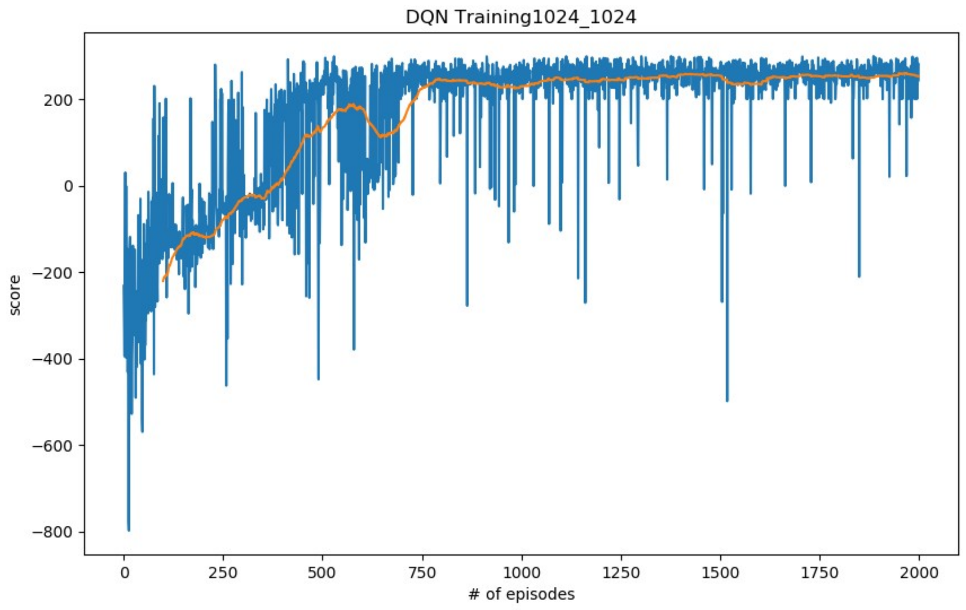
Results :

- 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

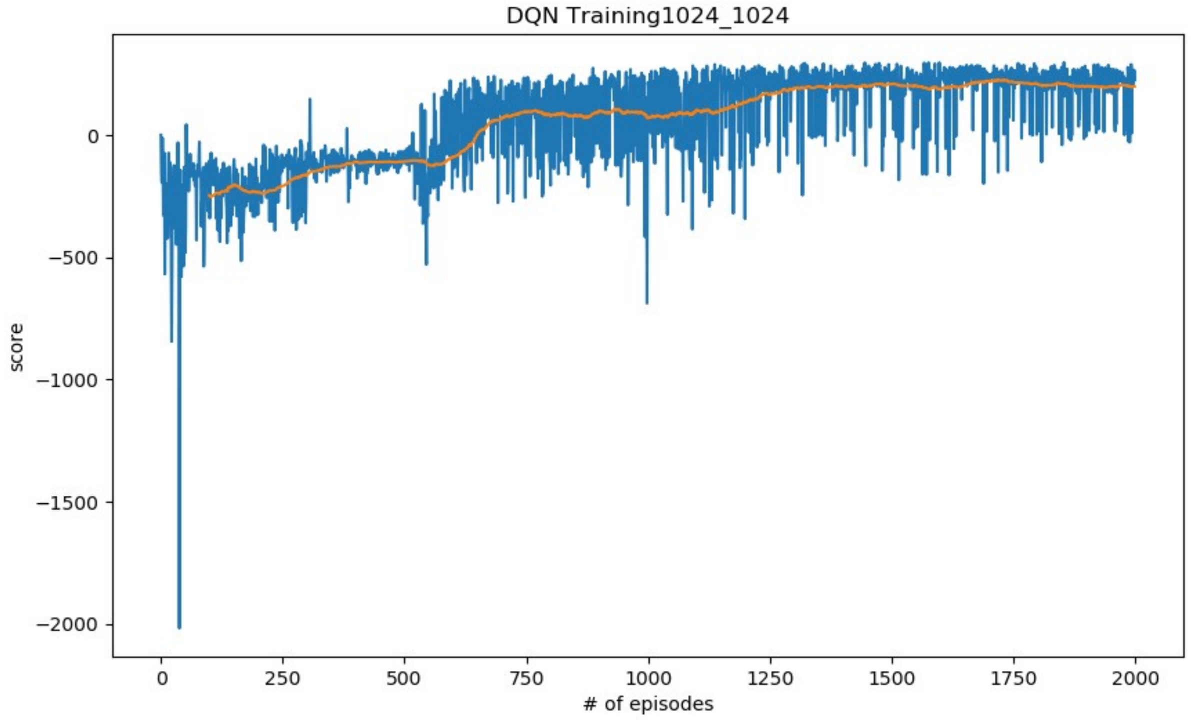


32,relu,64,relu



1024,relu,1024,relu

Effects of Network Architecture on DDQN training



1024,sigmoid,1024,sigmoid