

# Neural Architecture Search

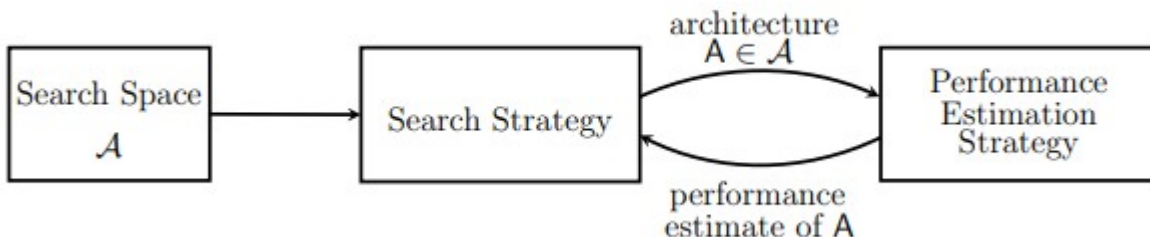
Ashish Kumar Jayant (M.Tech Research, CSA)  
Shikhar Bharadwaj (M.Tech Research, CSA)

# 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 all neural network architectures?
- Can we design a generalised approach that works across datasets?

# Neural Architecture Search (NAS) :

- Process of automating the architecture design part of Deep Learning neural networks.
- Performance should be comparable to state of the art results.
- Challenges : Search Space, Search strategy, Performance Estimation Strategy
- Usability across datasets?



# Elements of NAS

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

# 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 deterministic policy.
  - **Training mechanism** : Policy Gradient
    - Better convergence properties.
    - Effectiveness in high-dimensional or continuous action spaces
    - Ability to learn stochastic policies

# Neural Architecture Search with Reinforcement learning : Generation of network

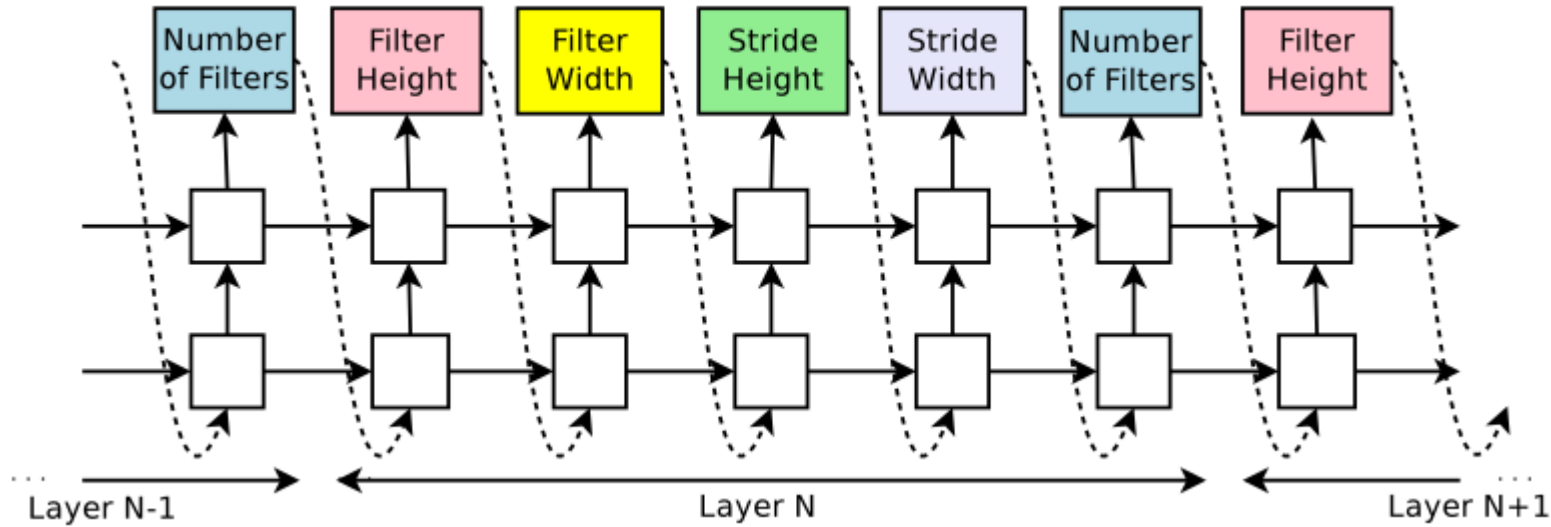
- Neural Networks can be represented as variable length string.
- For e.g (as given in paper) a simple feedforward CNN can be represented as -

```
layer {  
  name = layer_1  
  type = conv  
  connect_to = input  
  filter_width = 2  
  filter_height = 2  
  stride_width = 4  
  stride_height = 5  
  num_filters = 8  
}  
layer {  
  name = layer_2  
  type = conv  
  connect_to = layer_1  
  filter_width = 7  
  filter_height = 8  
  stride_width = 9  
  stride_height = 10  
  num_filters = 11  
}  
layer {  
  name = classifier  
  connect_to = layer_2  
  num_class = 10  
}
```



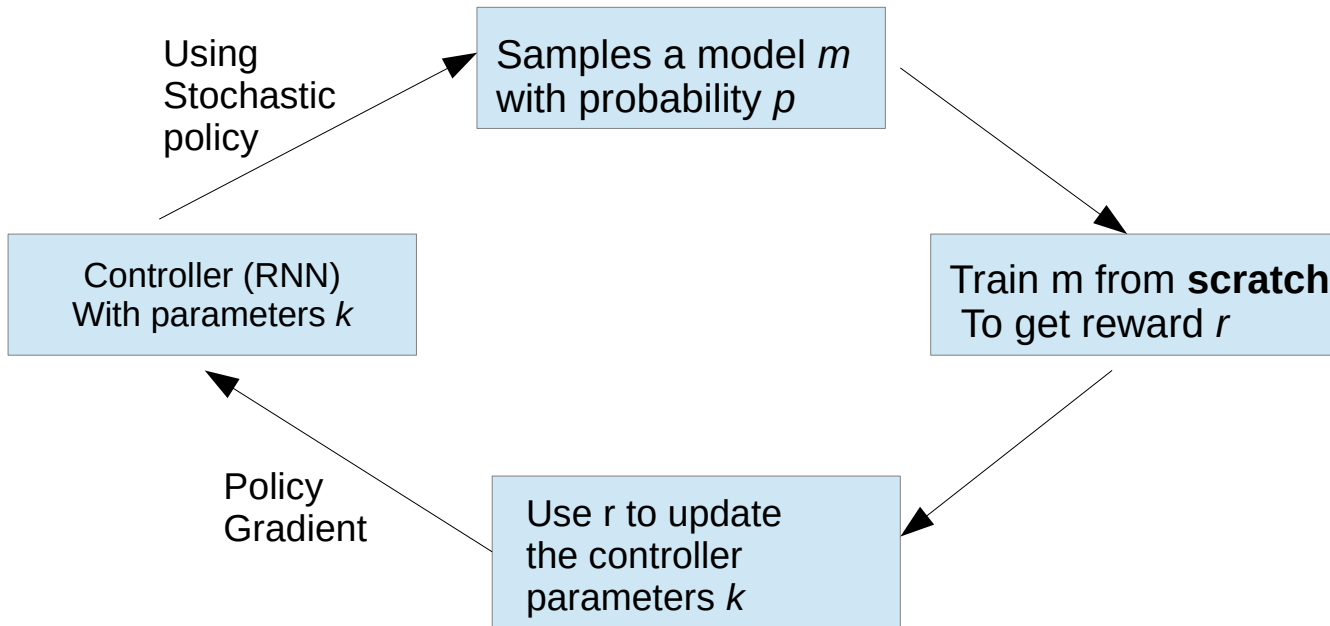
Layer 1					Layer 2				
“2, 2, 4, 5, 8,”					7, 8,9,10,11”				
Filter width	Filter height	Stride width	Stride height	Num filters	Filter width	Filter height	Stride width	Stride height	Num filters

# Neural Architecture Search with Reinforcement learning : Generation of network contd



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

# Neural Architecture Search with Reinforcement learning : Flow





# Neural Architecture Search with Reinforcement learning : Training controller

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return  $v_t$  as an unbiased sample of  $Q^{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

## **function REINFORCE**

Initialise  $\theta$  arbitrarily

**for** each episode  $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$  **do**

**for**  $t = 1$  to  $T - 1$  **do**

$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$

**end for**

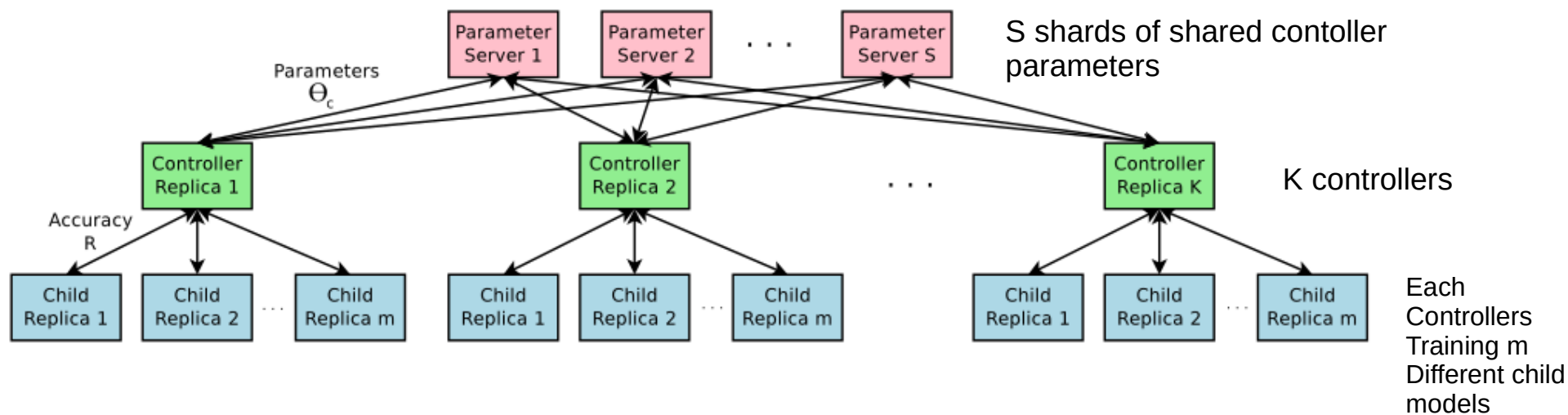
**end for**

**return**  $\theta$

**end function**

\*Source –  
David Silver's RL  
lecture series

# Neural Architecture Search with Reinforcement learning : Training setup



\*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 disadvantages of policy gradient method.

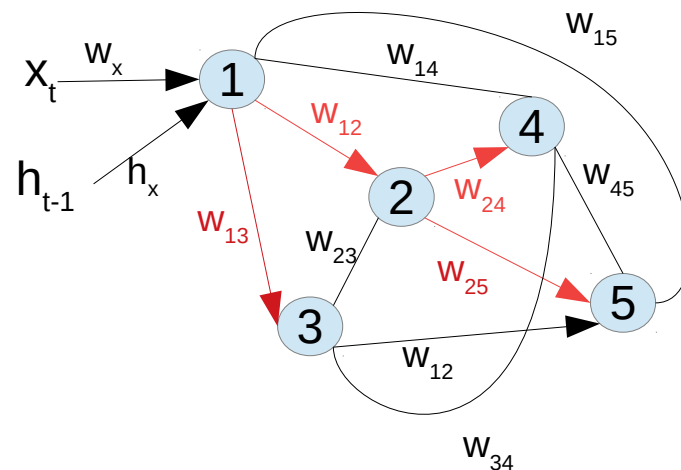
# Efficient Neural Architecture Search via Parameter Sharing

(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.

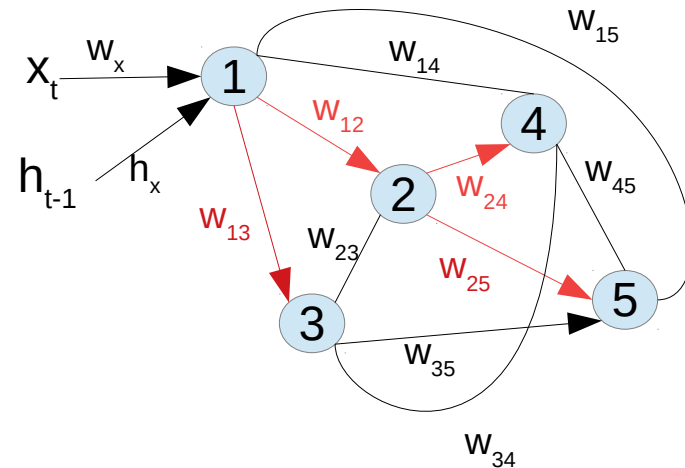
# Efficient Neural Architecture Search via Parameter Sharing : Generation

- 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.



# 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',  

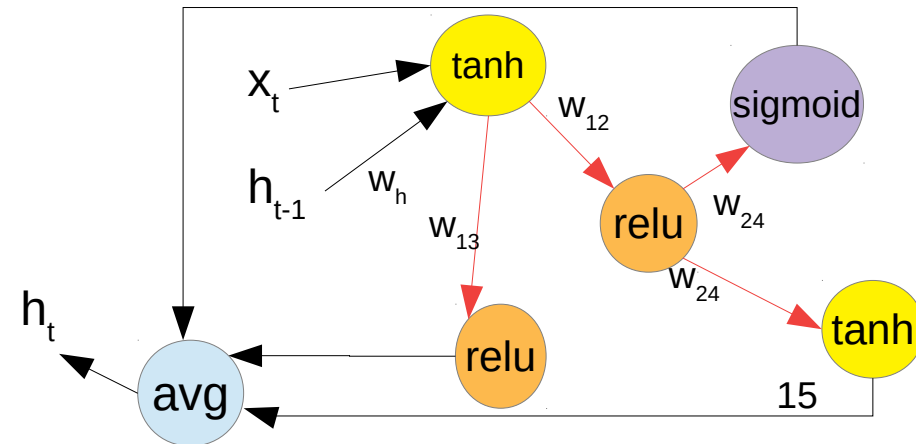
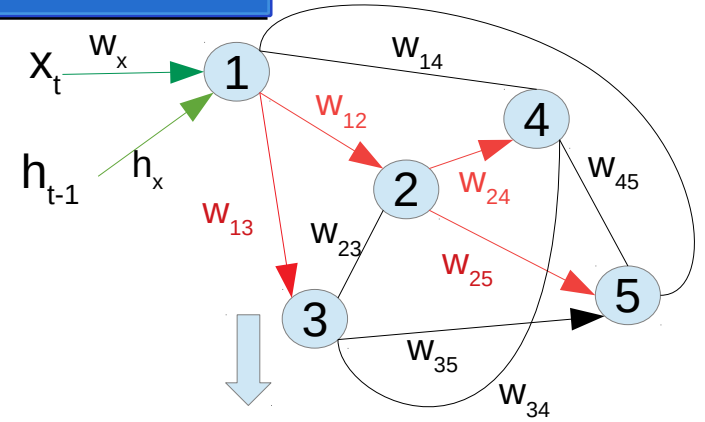
$$h_1 = \tanh(x_t \cdot W_x + h_{t-1} \cdot W_h)$$
- At node 2, controller sample 'relu' and prev node index '1' which activates globally shared  $w_{12}$   

$$h_2 = \text{relu}(w_{12} \cdot h_1)$$
- At node 3, controller samples 'relu' and prev node index '1'.  

$$h_3 = \text{relu}(w_{13} \cdot h_1)$$
- At node 4, controller samples 'sigmoid' and prev node index '2'.  

$$h_4 = \text{sigmoid}(w_{24} \cdot h_2)$$
- At node 5, controller samples 'tanh' and prev node index '2'.  

$$h_5 = \tanh(w_{25} \cdot h_2)$$
- Final output :  $h_t = \text{mean}(h_4, h_5, h_3)$

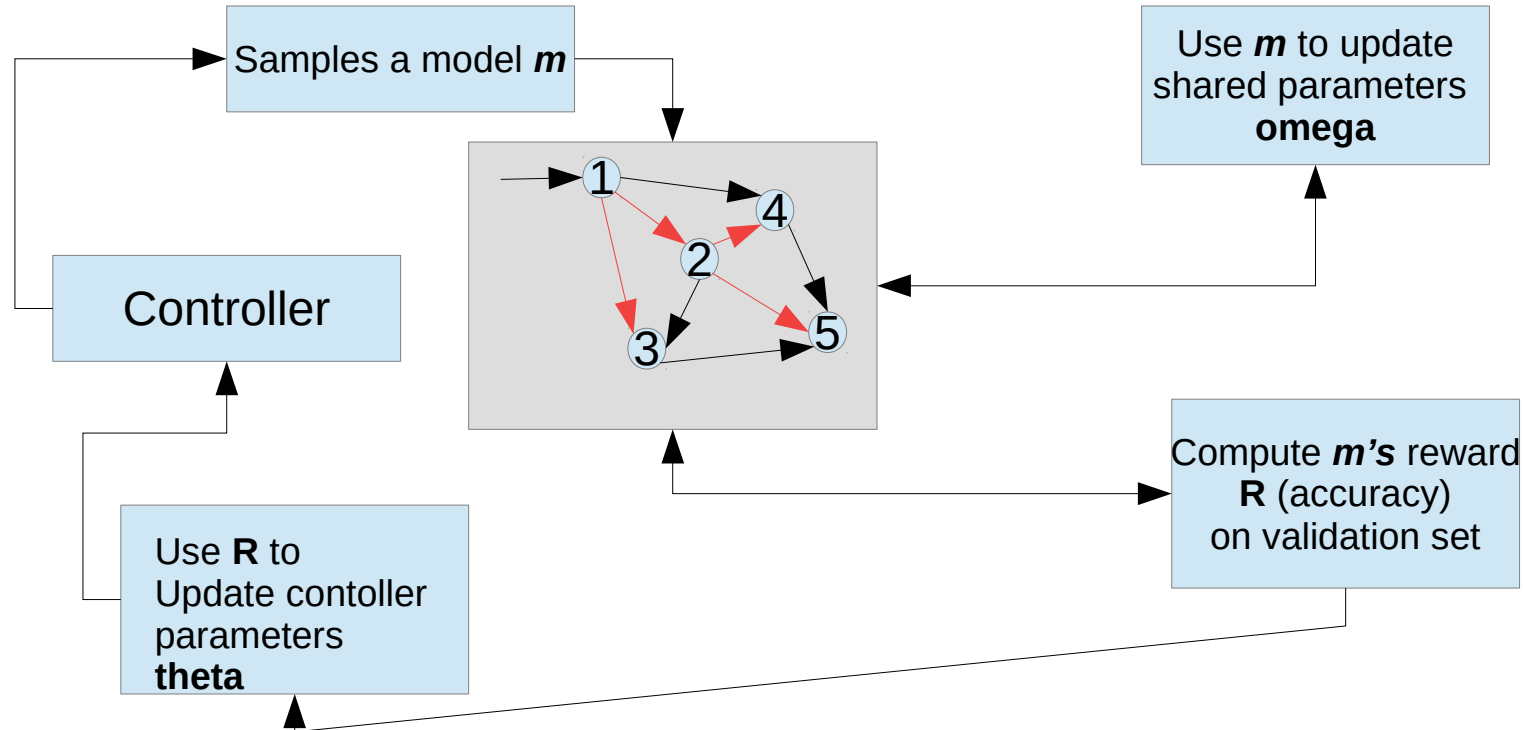


# Efficient Neural Architecture Search via Parameter Sharing : Training

- There are two sets of learnable parameters: the parameters of the controller LSTM, denoted by  $\theta$ , and the shared parameters of the child models, denoted by  $\omega$ .
- The training procedure of ENAS consists of two interleaving phases.
  - Fix the controller policy and train child models using SGD on  $\omega$  to minimize the expected loss function.
    - Loss function is **cross-entropy loss**  $L(\mathbf{m}; \omega)$  computed on mini-batch of training data, model  $\mathbf{m}$  sampled using parametrized stochastic policy  $\pi(\mathbf{m}; \theta)$  (\*turns out any sampled model can be used to update the gradient) then perform SGD(**batch\_size**,  $\mathbf{m}$ )
  - Fix the  $\omega$  and update the policy parameters  $\theta$  aiming to maximise expected reward using Monte Carlo Policy Gradient (REINFORCE) described in slide 7.



# Efficient Neural Architecture Search via Parameter Sharing : Flow



# 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 disadvantages of policy gradient method.