Dynamic Deep Neural Networks:
Optimizing Accuracy-Efficiency Trade-offs by Selective Execution
AAAI 2018

NAME : LEE. YEONSU

2019.10.14

Author

Lanlan Liu

1st Author of the paper

ph. D. Candidate

Education

Ph. D. candidate, Computer Science and Engineering

Computer Science and Engineering, University of Michigan, Ann Arbor

09. 2015 ~ Now

B. E., Computer Science and Technology

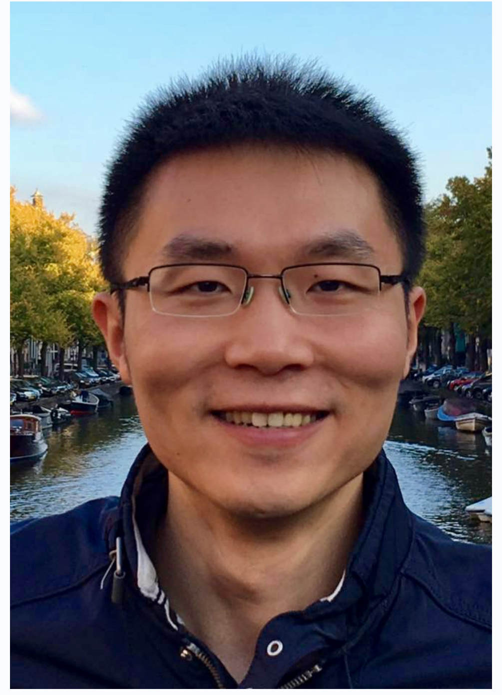
School for the Gifted Young, University of Science and Technology of China

08. 2011 ~ 06. 2015

Recent Published

Generative Modeling for Small-Data Object Detection

International Conference on Computer Vision(ICCV), 2019



Jia Deng

2nd Author of the paper

Assistant Professor, Department of Computer Science,Princeton University

Education

Ph. D., Computer Science

Computer Science, Princeton University

2012

M.A., Computer Science

Computer Science, Princeton University

2008

Recent Published

CornerNet: Detecting Objects as Paired Keypoints

European Conference on Computer Vision(ECCV), 2018

Abstract

- **Introduce Dynamic Deep Neural Networks** D^2NN
  - A new type of feed-forward deep neural network that allows selective execution
  - Given an input, only a subset of D^2NN neurons are executed, D^2NN provide a way to improve computational efficiency
- D^2NN augments a feed-forward deep neural-network with control node.
- Training is achieved by integrating backpropagation with reinforcement learning (Q-Learning)
- As a result,
  - They demonstrate that D^2NN are general and flexible, and can optimize accuracy - efficiency trade-offs.

Introduction

- **Motivation**
  - The need for computational efficiency, by need to deploy deep networks on mobile devices & data centers.
    - Mobile : constrained by energy and power, limiting the amount of computation that can be executed.
    - Data centers : need energy efficiency to scale to higher throughput and to save operating cost.

Introduction

- **Advantages**
  - Improve computational efficiency by selective execution
    - Pruning unnecessary computation depending on input.
  - It makes possible to use a bigger network under a computation budget by executing only a subset of the neurons each time.

Definition and Semantics of D^2NN

D^2NN definition

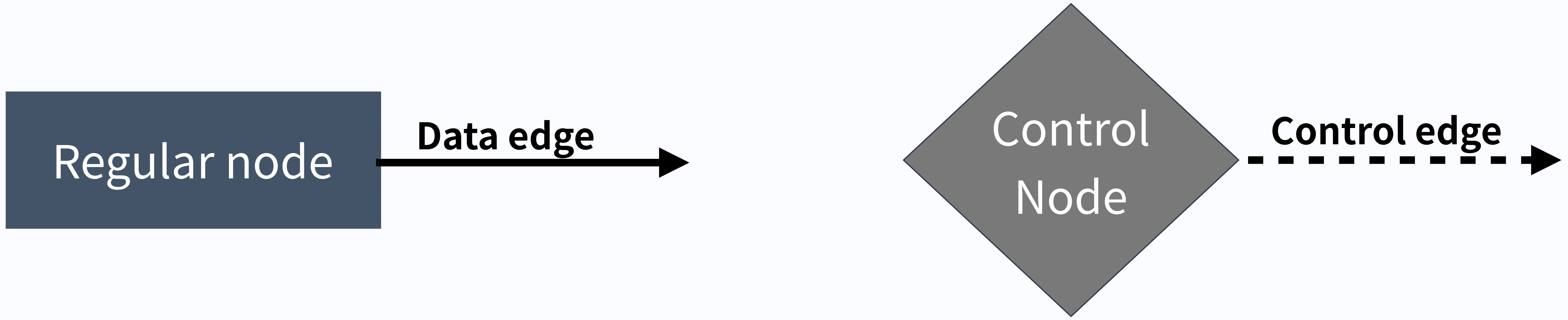
- **Node**
  - Input nodes & Output nodes
    - = input or output networks
  - Function nodes
    - = control node or data edge (depending on outgoing edge)
  - Dummy nodes
    - It is possible for a function node to take no data input and output a constant value.
- **Edge**
  - Data edge
    - A vector sent from one node to another, same as conventional DNN
    - optionally have a user-defined “default value”, representing the output will still be sent even if the function node does not execute.
  - Control edge
    - Control signal, a scalar, sent from one node to another

Definition and Semantics of D^2NN

Restrictions

- **The Outgoing edges from a node are either all data edges or all control edges.**
    - Cannot be a mix of data edges or all control edges

- **If a node has an incoming control edge, outgoing edge cannot be a control edge.**



Definition and Semantics of D^2NN

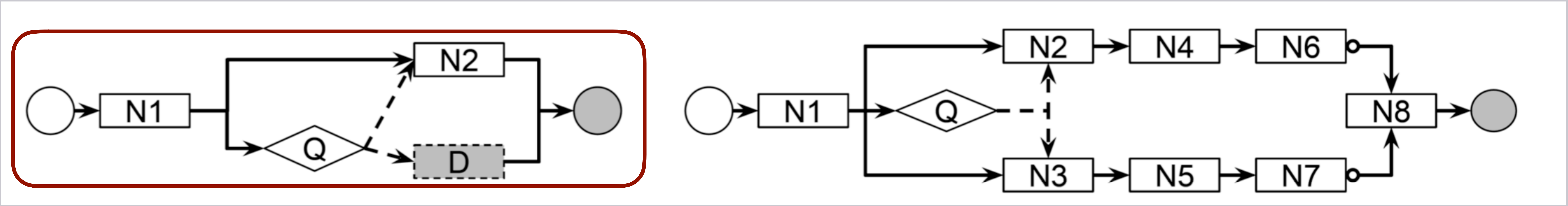
D^2NN Semantics

- Perform inference by traversing the graph starting from the input nodes
- Same as conventional DNNs except that the control nodes can cause the computation of One nodes to be skipped.
- After execute a Control node,
  - Output is a set of control scores, one for each of its outgoing control edges
  - **Highest score is "activated" -> allowed to execute**
- D^2NN can be though of as a program with conditional statements.
  - D^2NN introduces conditional statements with the conditions themselves generated by learnable functions.

Definition and Semantics of D^2NN

D^2NN Semantics

- **Control node**
  - Output is a decision that control whether other modules can execute.



- Q : control node
- N : regular module
- D : dummy node

——> : data edge
- - -> : control edge

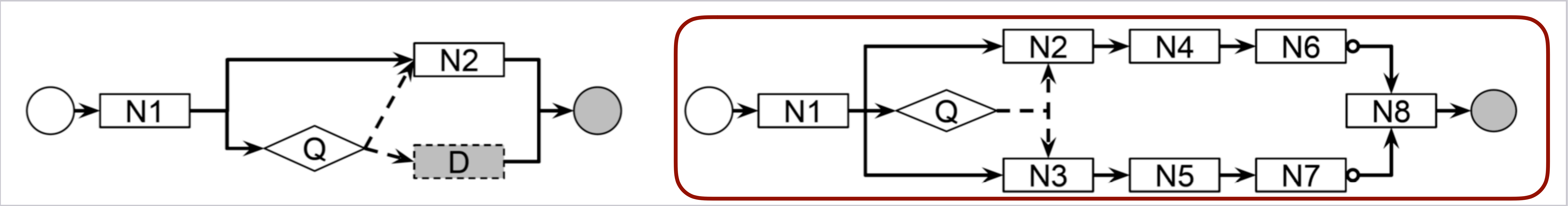
- Simple D^2NN with one control module (Q)
  - Q outputs a binary decision on whether module N2 executes
  - If Q decide that N2 is unnecessary , execute Dummy node(D) to save on computation.

- As an example, used for binary classification of images
  - Can be rapidly classified as negative after only a small amount of computation.

Definition and Semantics of D^2NN

D^2NN Semantics

- **Control node**
  - Output is a decision that control whether other modules can execute.



- Q : control ode
- N : regular module
- D : dummy node

——> : data edge
- - -> : control edge

- The node Q controls N2 and N3.
  - N2 or N3 - execute depending on which has the higher control score.
- If one of the node is skipped, its output will be default or null.
  - If output is default value, subsequent execution will continue as usual.
  - If output is null, any downstream nodes that depend on this output will be skipped.

D^2NN Learning

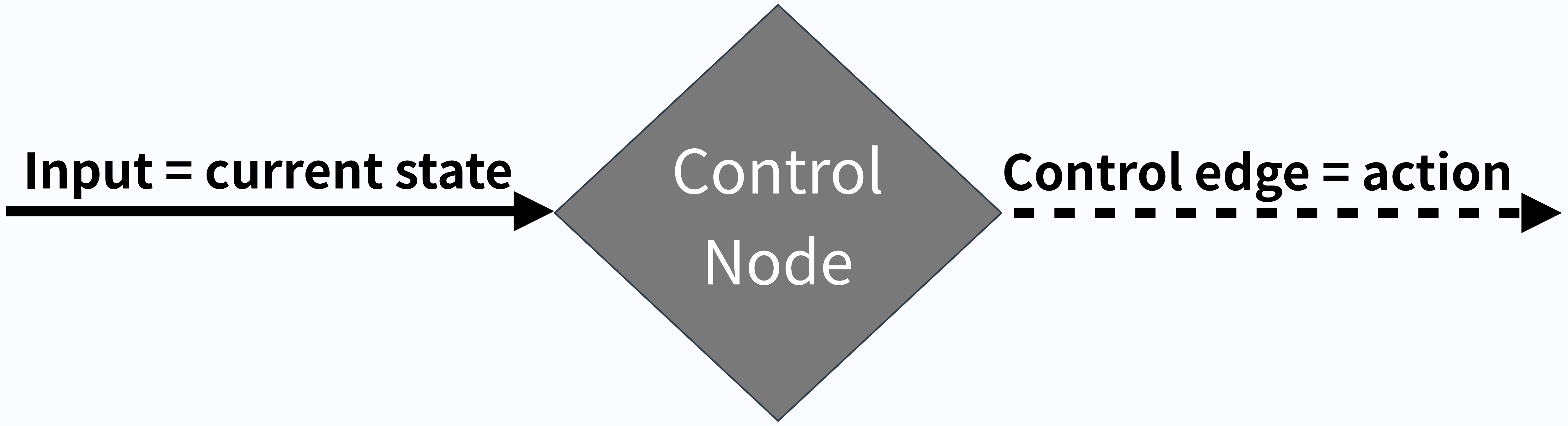
- D^2NN cannot be trained same way as a conventional DNN.
  - Back propagation cannot be directly applied.
  - Used Q-Learning(reinforcement learning) to discretized control node.

D^2NN Learning

Learning a Single Control Node

Learning a Single Control Node

- Start with one control node
  - The goal is to learn the parameters of the control node to maximize a user-defined reward.
    - User-defined reward == combination of accuracy and efficiency = $\lambda A + (1 - \lambda)E$
  - Learning a control policy to take actions so as to maximize reward
    - Method on Q-Learning (one of reinforcement learning)
- Outgoing control edge = action
- Control node approximate the action-value (Q) function
  - Each control node only executes once.

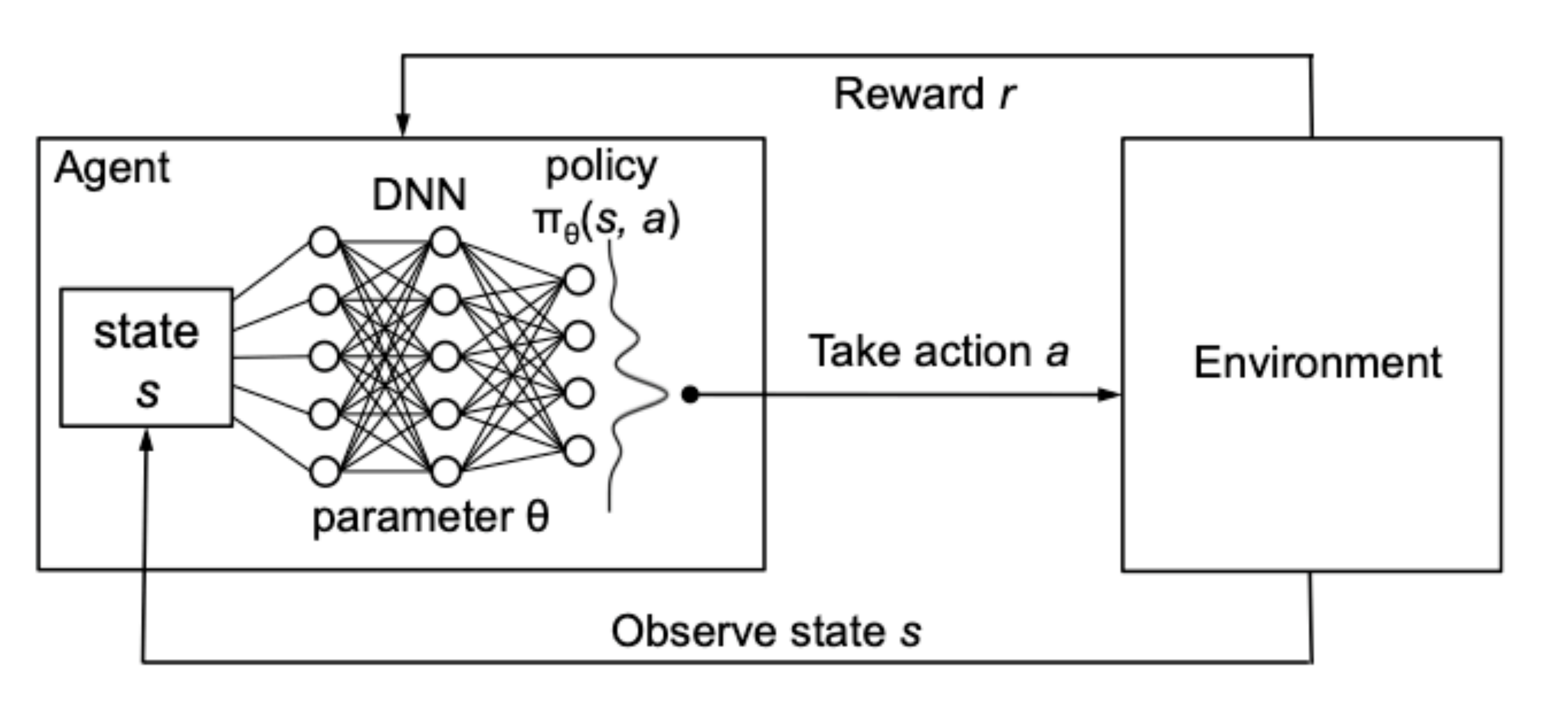


D^2NN Learning

Q-Learning

Q - Learning

- seeks to find the best action to take given the current state
- seeks to learn a policy that maximizes the total reward.

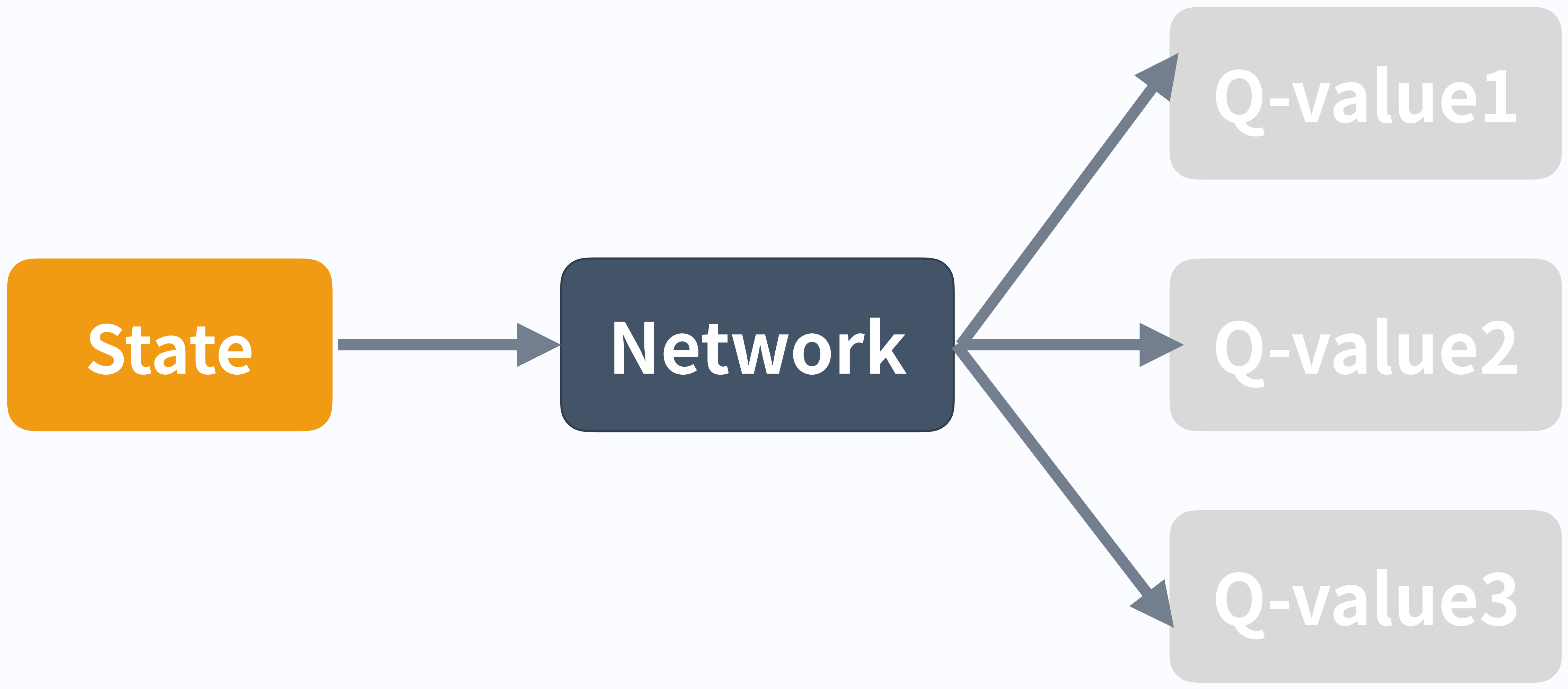


D^2NN Learning

Q-Learning

Q - Learning

- seeks to find the best action to take given the current state
- seeks to learn a policy that maximizes the total reward.



D^2NN Learning

Learning a Single Control Node

Learning a Single Control Node

- An action take on one input has no effect on another input.

$$\boxed{L = (Q(s,a) - r)^2}$$

- **r** = user-defined reward
- **a** = action
- **s** = input to control node
- **Q** = computed by control node
- Predict the reward for each action under an **L2** loss

D^2NN Learning

Learning a Single Control Node

Learning a Single Control Node

- During training we also perform $\epsilon - greedy\ exploration$ instead of always choosing with the best Q- value
- The hyperparameter ϵ is initialized to 1 and decreases over time.
  - A = accuracy (F-score)
  - E = efficiency (inverse of number of multiplications)
  - Reward = $\lambda A + (1 - \lambda)E$

D^2NN Learning

Mini-Bags for Set-Based Metrics

Mini-Bags for Set-Based Metrics

- Set of inputs = Mini-bag
  - With a mini-bag of images, any set-based metric can be computed and can be used to directly define a reward.
  - Mini-bag =! Mini-batch
- Calculate gradients using a mini-batch of mini-bags.
- Mini-bag $s = (s_1, \ldots, s_m)$
- Joint action $a = (a_1, \ldots, a_m)$

$$\boxed{Q = \sum_{i=1}^{m} Q(s_i, a_i)}$$

- $Q(s_i, a_i)$ is a score given by the control node when choosing the action a_i for example s_i

D^2NN Learning

Mini-Bags for Set-Based Metrics

Mini-Bags for Set-Based Metrics

- Then define new learning objective on a **mini-bag of size** m as where r is the **reward** observed by choosing the joint **action** a on **mini-bag** s.

$$\boxed{L = (r - Q(s, a))^2 = \sum_{i=1}^{m} (r - Q(s_i, a_i))^2}$$

- Control node predicts an action value for each example such that their sum approximates the reward defined on the whole mini-bag

- $Q(s, a)$ is simply the concatenation of the best actions for individual examples

$$\boxed{a_i^* = argmax_{a_i} Q(s_i, a_i) \qquad i = 1,2,\ldots,m\,.}$$

  - Because maximizing optimal a is equivalent to maximizing the individual summands.

D^2NN Learning

Mini-Bags for Set-Based Metrics

Mini-Bags for Set-Based Metrics

- Then define new learning objective on a mini-bag of size m as where r is the reward observed by choosing the joint action a on mini-bag s.

$$\boxed{\frac{\delta L}{\delta x_i} = 2\left(r - \sum_{j=1}^{m} Q(s_j, a_j)\right)\frac{\delta Q(s_i, a_i)}{\delta x_i}}$$

  - x_i is the output of any internal neuron for example i in the mini-bag.
  - Shows that there is no change to the implementation of back propagation except that we scale the gradient using the difference between the mini-bag Q-value Q and reward r.

DNN Learning

Joint Training of All Nodes

Joint Training of All Nodes

- When D^2NN has multiple control nodes, simply train them together.
- For each mini-bag, perform back propagation for multiple losses together.
  - observe a reward for the whole network, then use the same reward (which is a result of the actions of all control nodes) to back propagate for each control node.
- Important detail
  - The losses on regular nodes need to be properly weighted against the losses on the control nodes. ***
    - To eliminate this problem use Q-learning losses on regular nodes
      - For example treat the classification scores as action-values ——> an estimated reward for each classification decision.

Experiments

- **Experiment four D^2NN structures**
  - motivated by different demands of efficient network design to show its flexibility and effectiveness.

< 4 typed of D^2NN model >

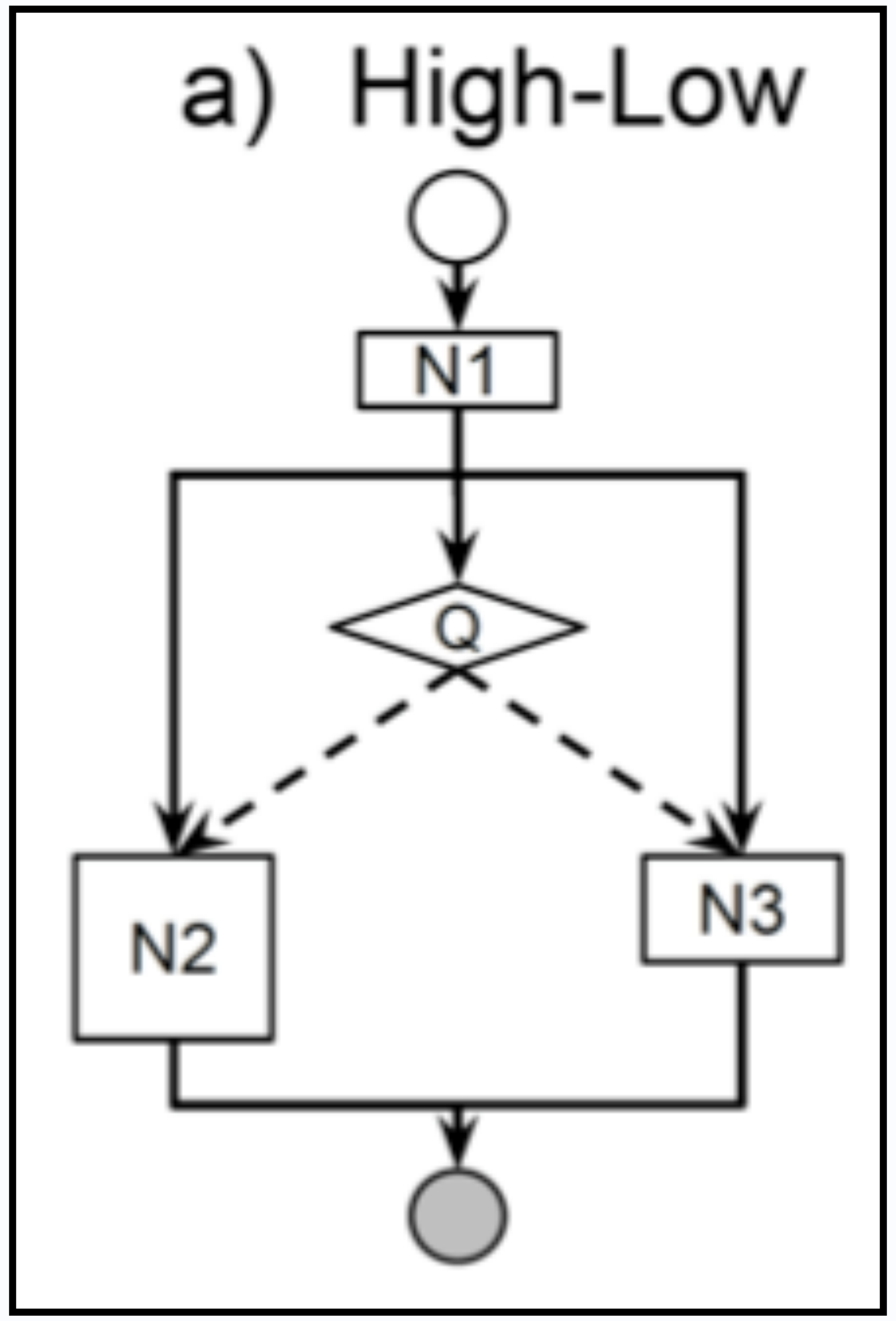
- High - Low
- Cascade
- Chain
- Hierarchy

Experiments

High-Low Capacity DNN

- Motivated by that can save computation by choosing a low-capacity subnetwork for easy examples.
  - High-capacity : N2
  - Low-capacity : N3

- Test with binary classification task
  - Input image : Labeled Faces in the Wild dataset
  - Accuracy : F1-score
  - Efficiency : Computational cost - number of multiplications
  - Reward : $\lambda A + (1 - \lambda)E$



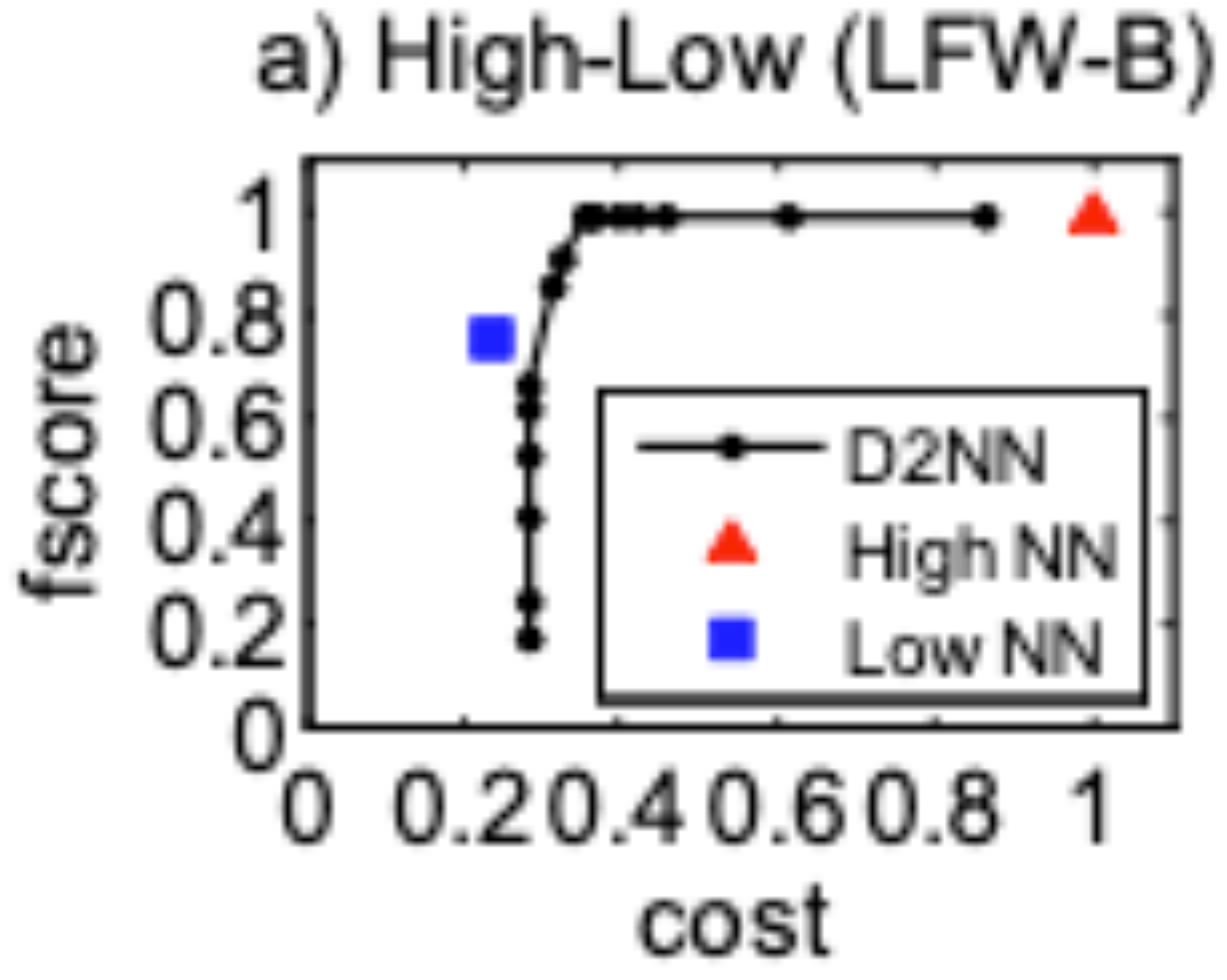
- N1 : **conv** (filter size = (3,3), filters = 8, stride = 2) + **max_pooling**(3x3, stride = 2)
- N2 : **conv** (filter size = (3,3), filters = 16) + **max_pooling**(3x3, stride = 2) + **reshape + fully_connected**(512) **+ fully_connected**(2-class output)
- N3 : **max_pooling**(3x3, stride = 2) + **fully_connected**(32) **+ fully_connected**(2-class output)
- Q1 : **conv** (filter size = (3,3), filters = 2) + **max_pooling**(3x3, stride = 2) + **reshape + fully_connected**(128) **+ fully_connected**(2-action output)

Experiments

High-Low Capacity DNN

- As λ increases, the learned D^2NN trades off efficiency for accuracy.
- This example suggest that this learning algorithm is effective for networks with a single control Node.
- With low NN, it achieves 0.2 cost and 0.8 accuracy.
- With high NN, it achieves 1 cost and 1.0 accuracy.

accuracy-cost graph



Experiments

High-Low Capacity DNN

- Fig 5 plots the distribution of examples going through different execution path.
  - It shows that as λ increases, accuracy becomes more important and more examples go through the high-capacity node.
- This example suggest that this learning algorithm is effective for networks with a single control Node.

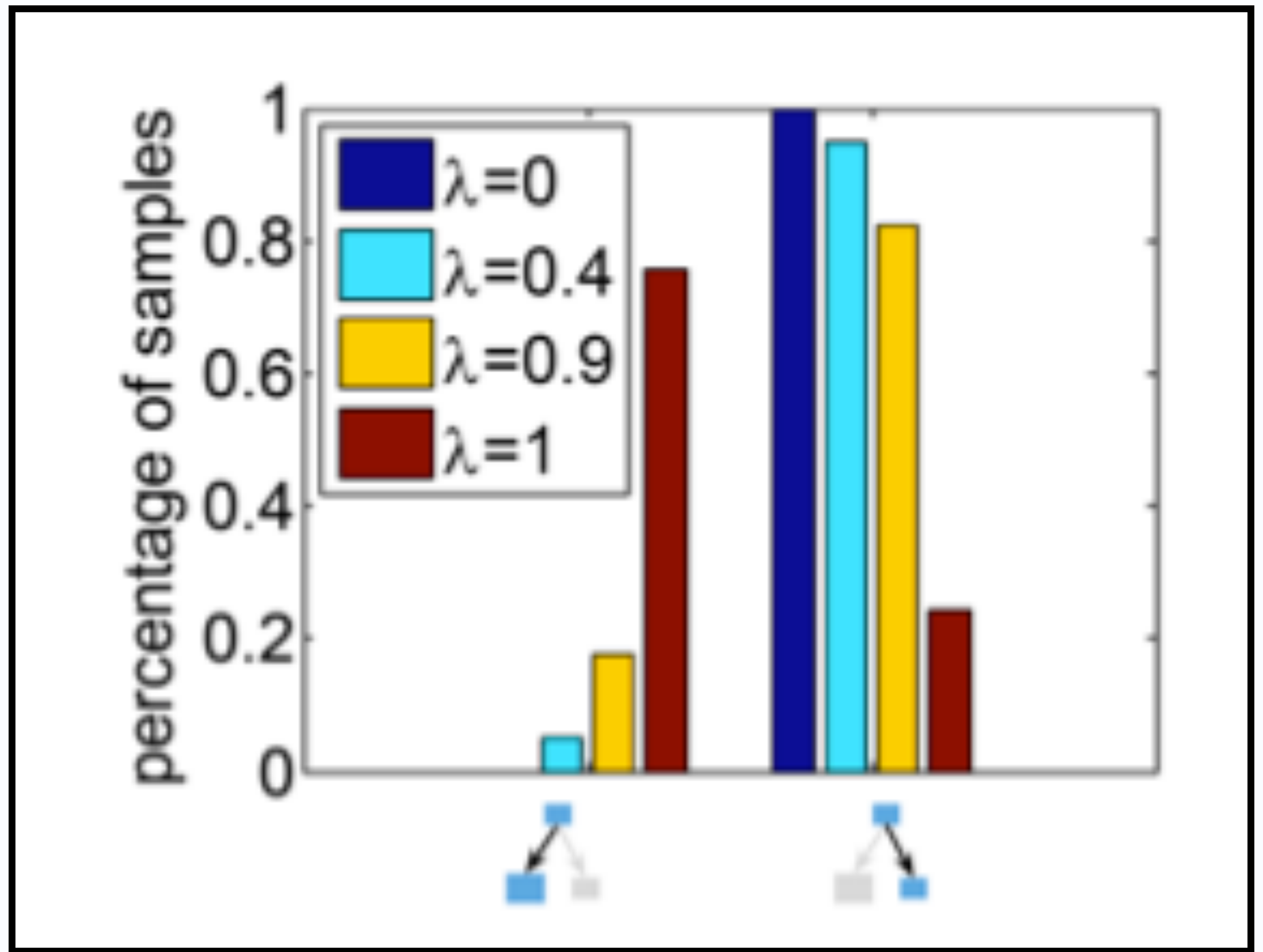
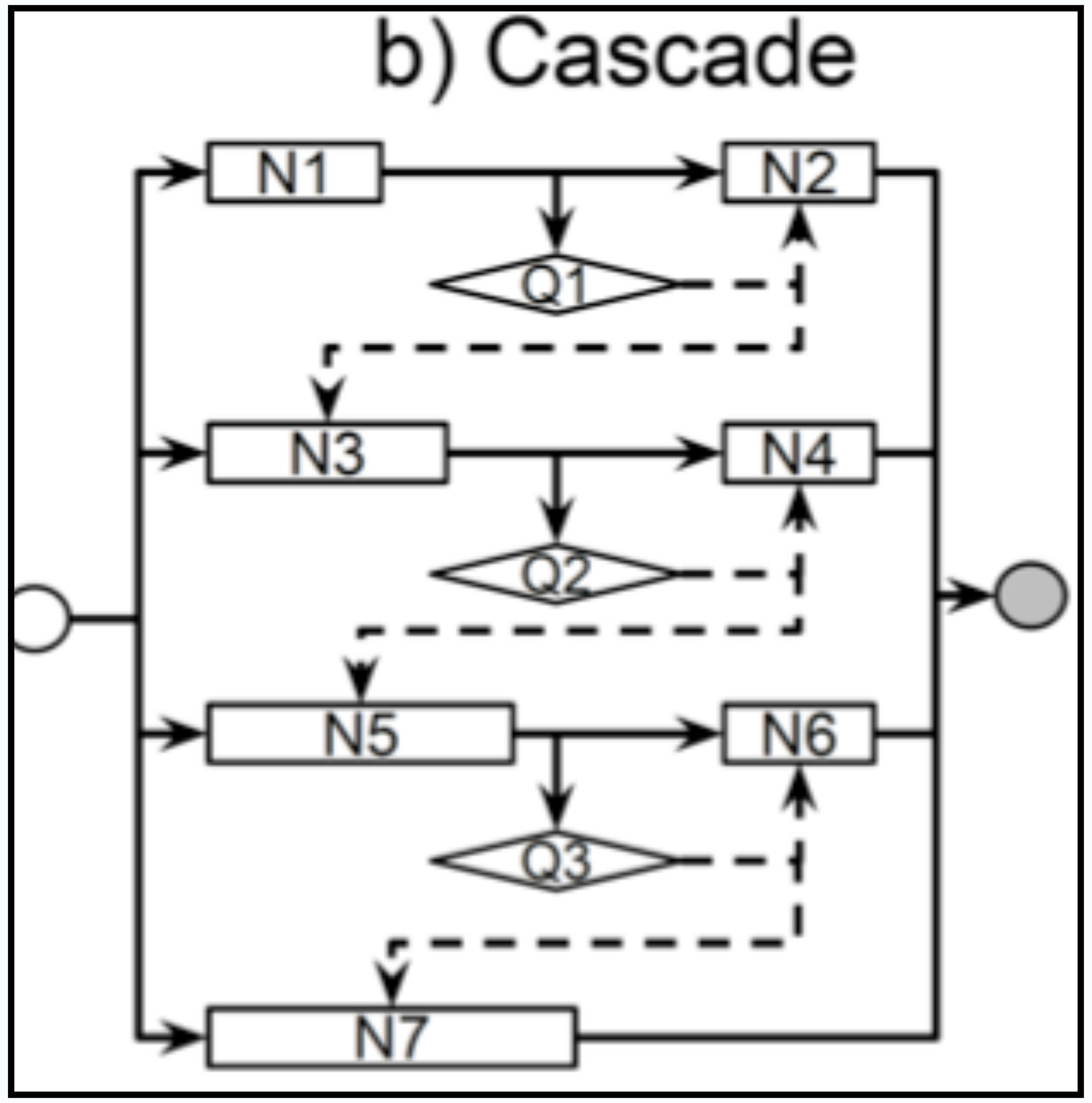


Figure 5. Examples with different paths in a high-low D^2NN (left) and a hierarchical D^2NN (right).

Experiments

Cascade DNN

- Cascade design
  - Inspired by the standard cascade design commonly used in computer vision.
  - The intuition is that many negative examples may be rejected early using simple functions.
  - Regular node N1-N7 form 4 cascade stages
    - N1 + N2
    - N3 + N4
    - N5 + N6
    - N7

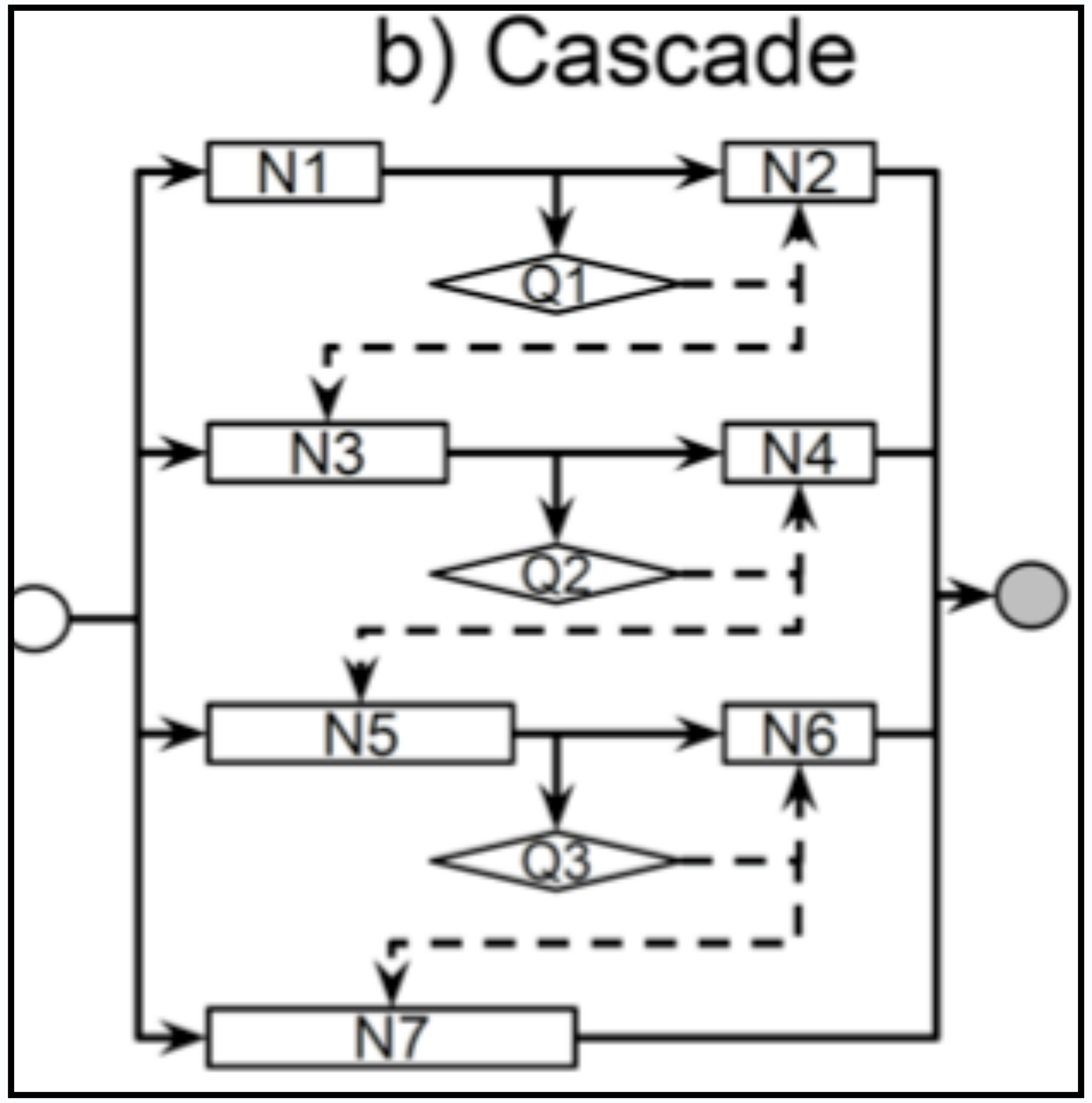


- N1 : **conv** (filter size = (3,3), filters = 2, stride = 2) + **max_pooling**(3x3, stride = 2)
- N2 : **conv** (filter size = (3,3), filters = 16) + **max_pooling**(3x3, stride = 2) + **fully_connected**(2-class output)
- N3 : **conv** (filter size = (3,3), filters = 2, stride = 2)+ **max_pooling**(3x3, stride = 2) + **conv** (filter size = (3,3), filters = 8, stride = 2) + **max_pooling**(3x3, stride = 2)
- N4, N6 : **max_pooling**(3x3, stride = 2) + **max_pooling**(3x3, stride = 2) + **fully_connected**(2-class output)
- N5 : **conv** (filter size = (3,3), filters = 4, stride = 2)+ **max_pooling**(3x3, stride = 2) + **conv** (filter size = (3,3), filters = 16, stride = 2) + **max_pooling**(3x3, stride = 2)
- N7 : **conv** (filter size = (3,3), filters = 2, stride = 2)+ **max_pooling**(3x3, stride = 2) + **conv** (filter size = (3,3), filters = 8, stride = 2) + **max_pooling**(3x3, stride = 2) + **conv** (filter size = (3,32), filters = 2, stride = 2) + **conv** (filter size = (3,3), filters = 32, stride = 2)+ **max_pooling**(3x3, stride = 2) + **conv** (filter size = (3,3), filters = 64, stride = 2) + **fully_connected**(512) + **fully_connected**(2-class output)
- Q1, Q2, Q3 : **fully_connected**(2-action output)

Experiments

Cascade DNN

- Cascade design
  - Regular node N1-N7 form 4 cascade stages
    - N1 + N2
    - N3 + N4
    - N5 + N6
    - N7
  - Each control node decide whether to execute the next cascade stage or not.

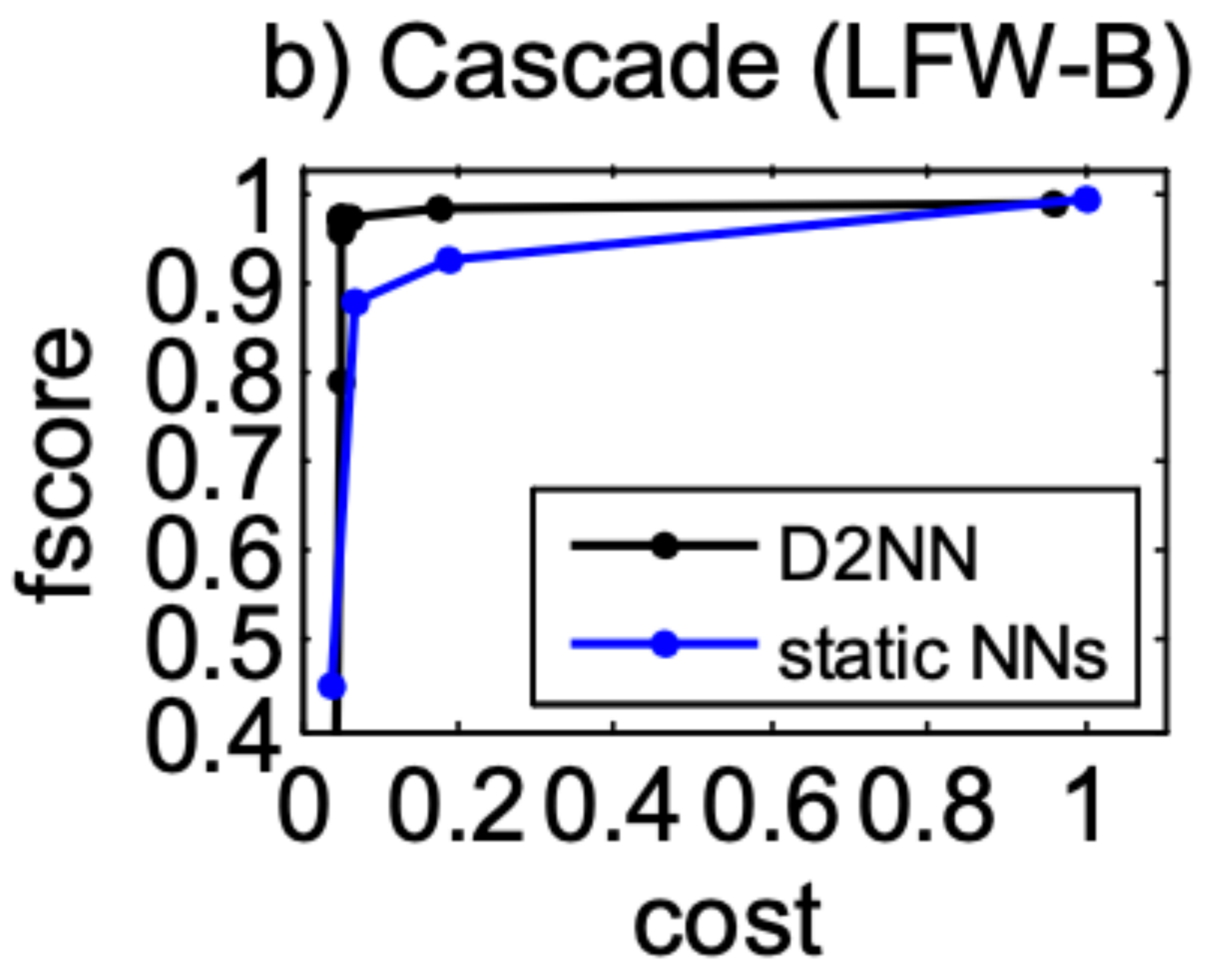


Experiments

Cascade DNN

- Cascade design
  - achieve a close to optimal trade-off, reducing computation significantly with negligible loss of accuracy.
  - This result demonstrates that our algorithm is successful for jointly training multiple control nodes.

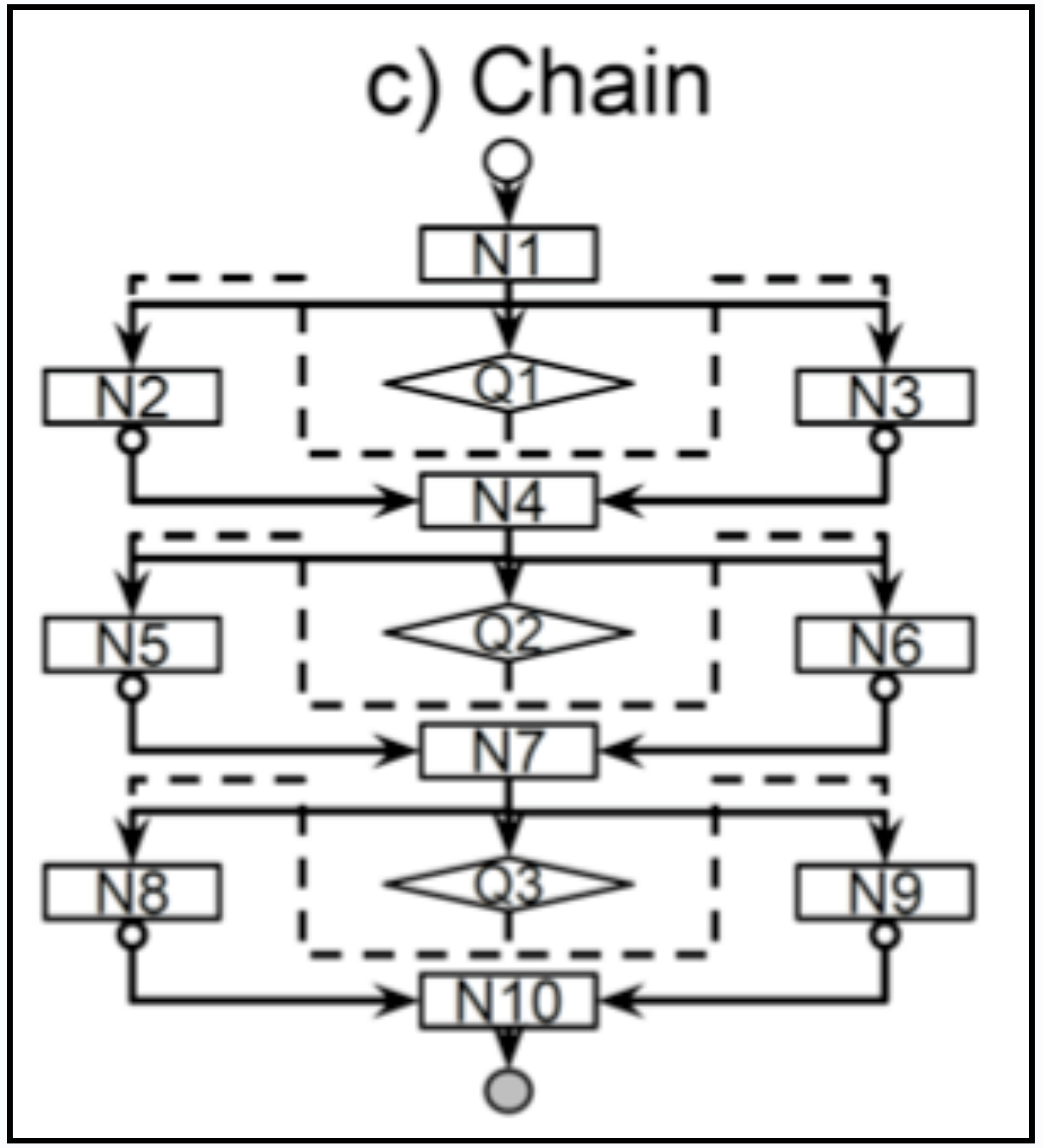
accuracy-cost graph



Experiments

Chain DNN

- Chain design ,
  - Tree-shaped data graph and it allows two divergent data paths to merge again.
  - Number of possible execution paths can be exponential to the number of nodes.



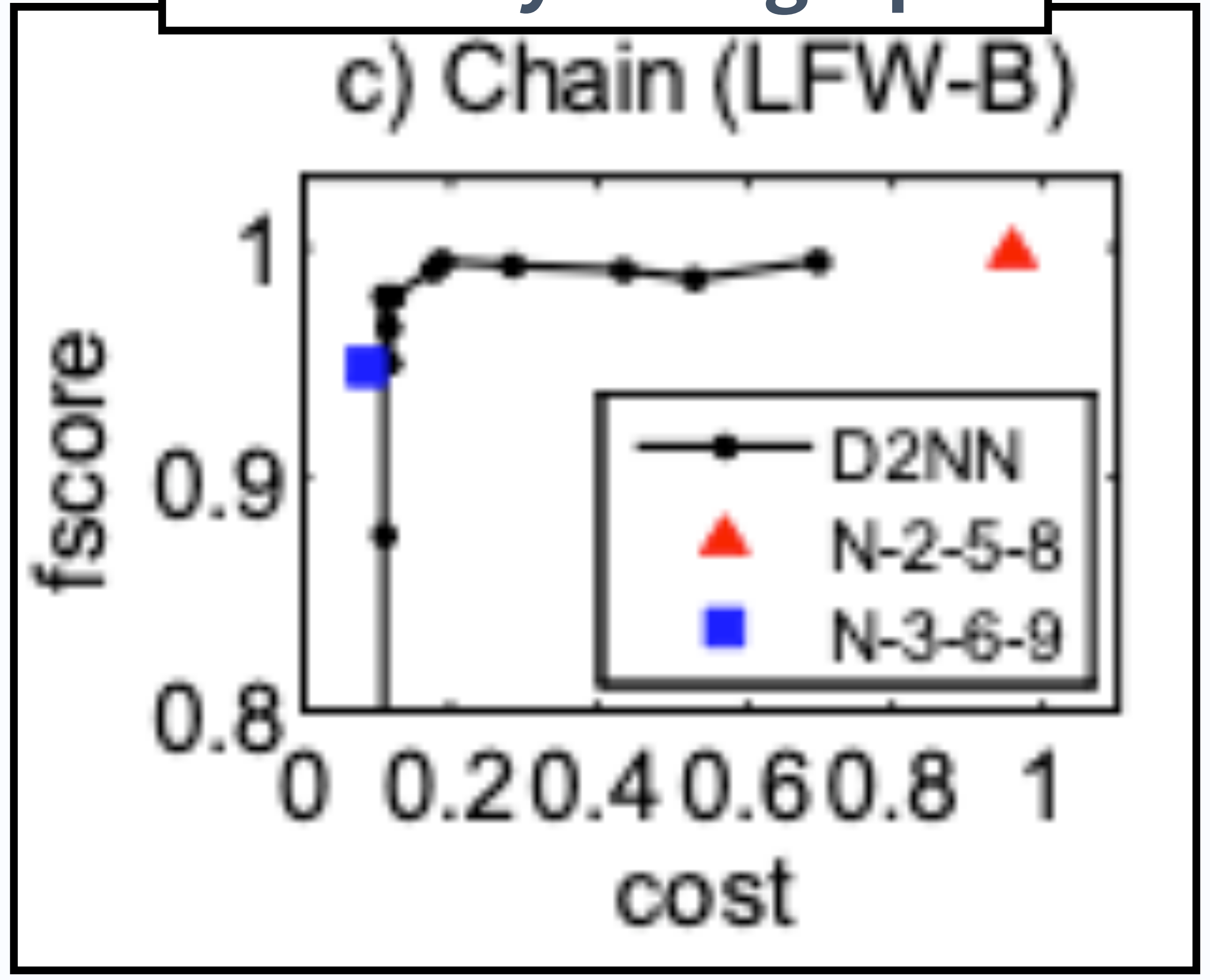
- N1 : **conv** (filter size = (3,3), filters = 2, stride = 2) + **max_pooling**(3x3, stride = 2)
- N2 : **conv** (filter size = (1,1), filters = 16)
- N3 : **conv** (filter size = (3,3), filters = 16)
- N4 ,N7: **max_pooling**(3x3, stride = 2)
- N5 : **conv** (filter size = (1,1), filters = 32)
- N6 : **conv** (filter size = (3,3), filters = 32) + **conv** (filter size = (3,3), filters = 32)
- N8 : **conv** (filter size = (1,1), filters = 32)+ **max_pooling**(3x3, stride = 2) + **fully_connected**(256)
- N9 : **conv** (filter size = (3,3), filters = 64) + **fully_connected**(256)
- N10 : **fully_connected**(2-class output)
- Q1 : **max_pooling**(3x3, stride = 2) + **conv** (filter size = (3,3), filters = 8) + **max_pooling**(3x3, stride = 2) + **fully_connected**(64) + **fully_connected**(2-action output)
- Q2 : **max_pooling**(3x3, stride = 2) + **conv** (filter size = (3,3), filters = 4) + **fully_connected**(64) + **fully_connected**(2-action output)
- Q3 : **conv** (filter size = (3,3), filters = 2) + **fully_connected**(64) + **fully_connected**(2-action output)

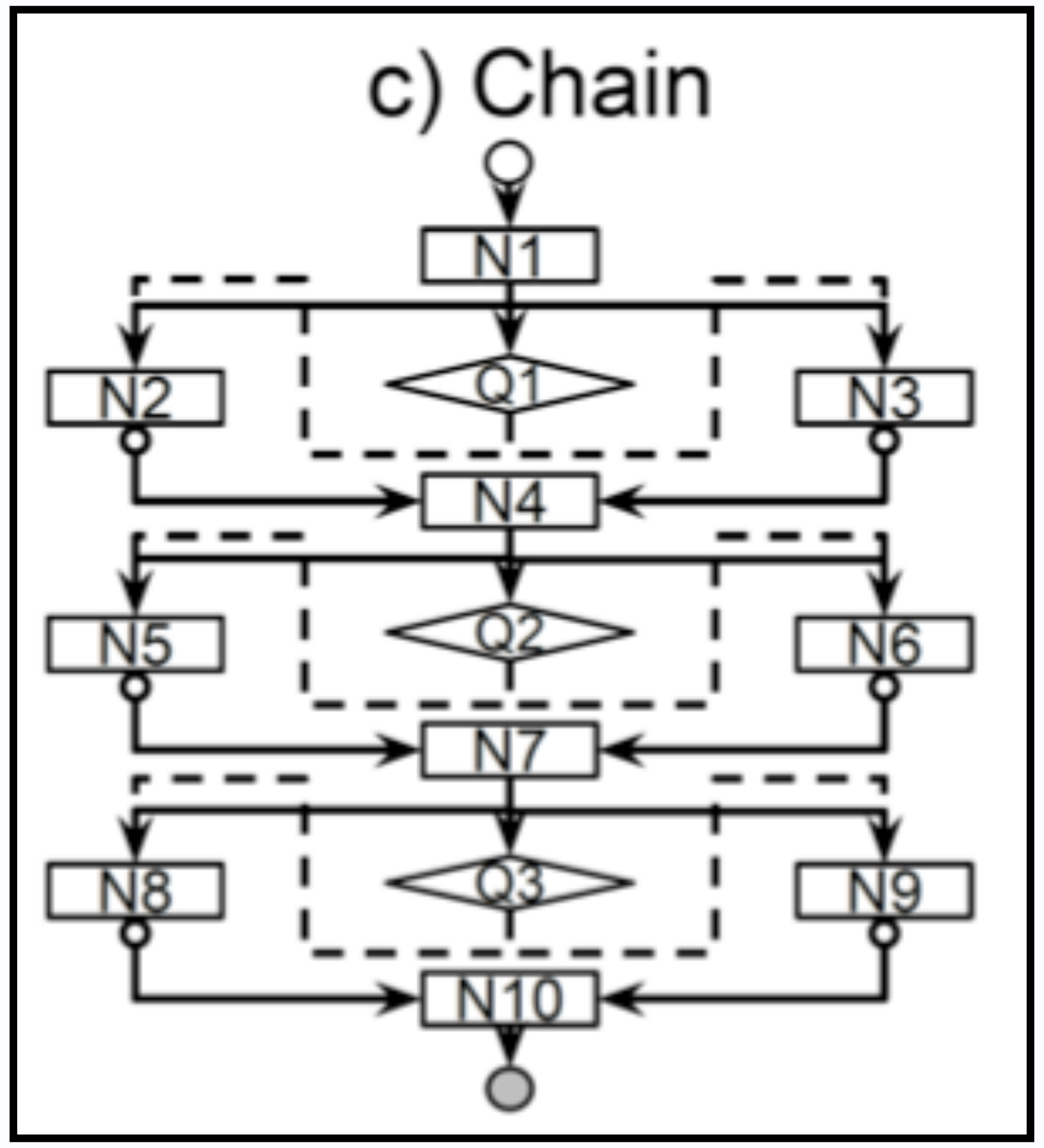
Experiments

Chain DNN

- Q1 chooses low-capacity N2 or high-capacity N3
  - One of them chosen and the other will output a default value zero.
- Path
  - Lowest capacity : N1-N2-N5-N8-N10
  - Highest capacity : N1-N3-N6-N9-N10
- The chain DNN achieves trade-off curve close to optimal and can speed up computation significantly with little accuracy loss.

accuracy-cost graph





Experiments

Hierarchical DNN

- Hierarchical multi class classification
- The idea is to first classify images to coarse categories and then to fine categories.
    - Data : ILSVRC-10, a subset of the ILSVRC-65
        - D^2NN mirrors the semantic hierarchy in ILSVRC-10.
        - 10classes are organized into a 3-layer hierarchy
            - 2 superclasses, 5 coarse classes and 10 leaf classes.

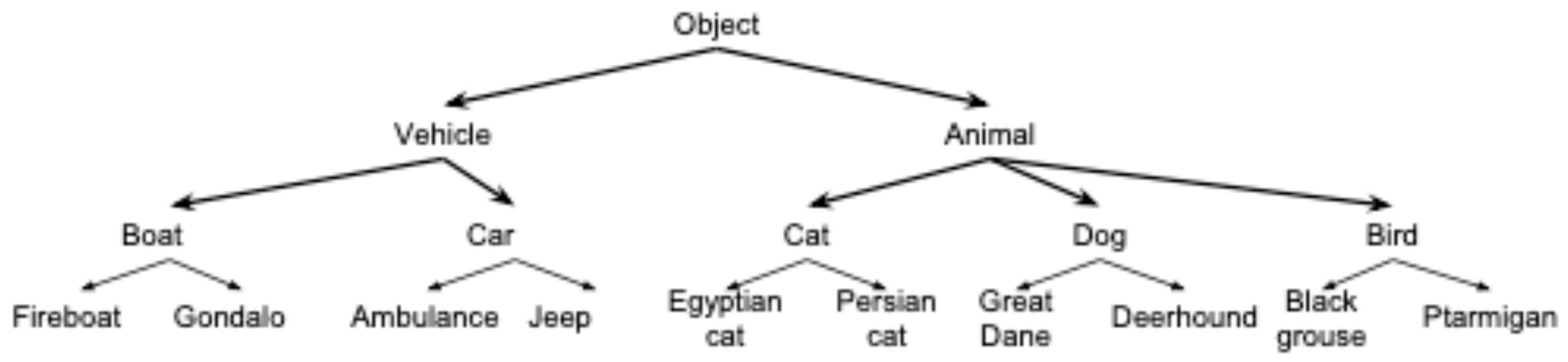
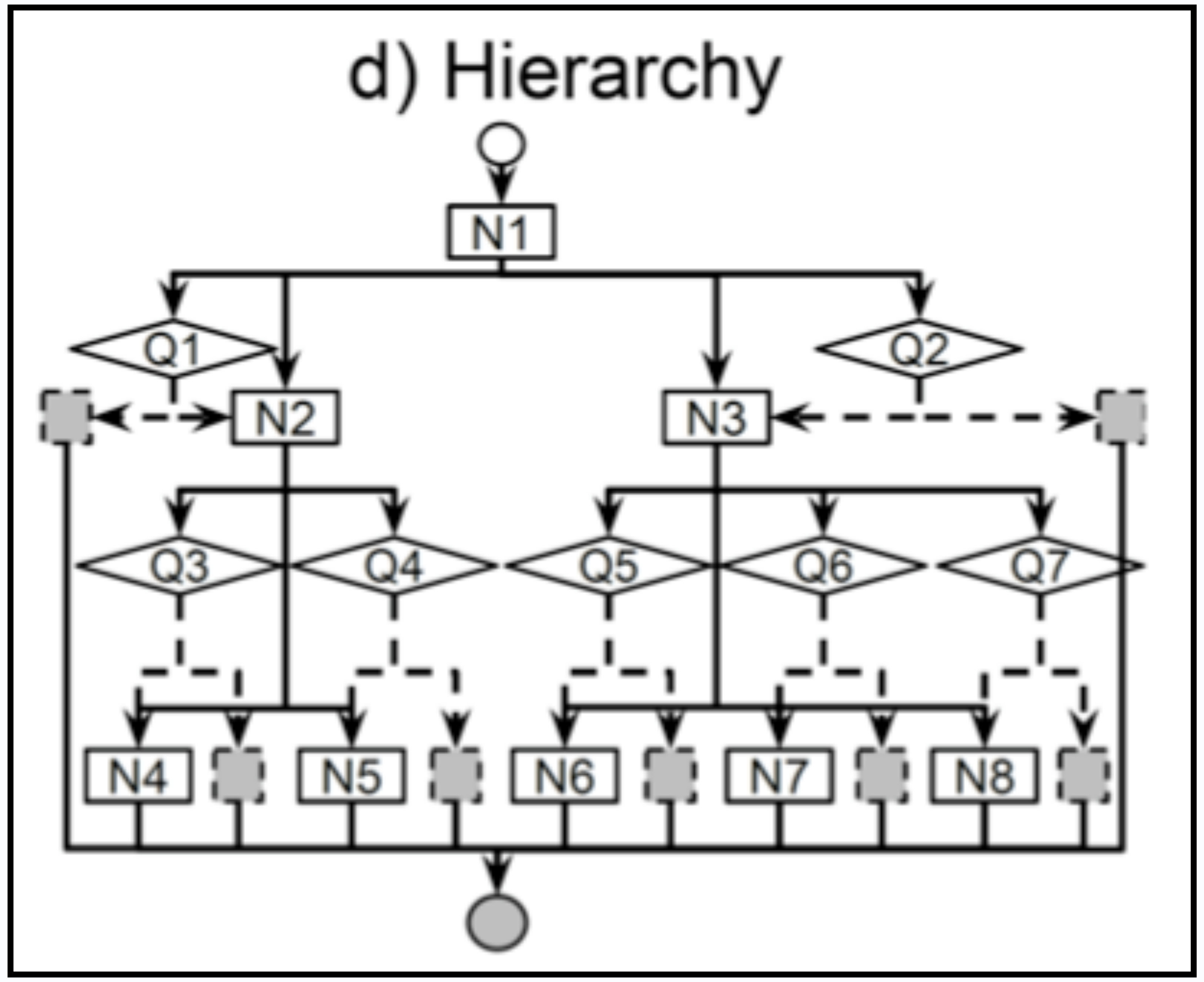


Figure 7. The semantic class hierarchy of the ILSVRC-10 dataset.

Experiments

Hierarchical DNN

- Start with root N1
- Q1 decides where to descend the N2 or children
- Q2 decides where to descend the N3 or children
- Leaf nodes N4-N8 are each responsible for classifying two fine-grained leaf classes.
- (***)Input image can go down parallel paths in the hierarchy descending left & right together.
    - Because Q1 and Q2 make separate decisions.
- “Multi-threading” allows the network to avoid committing to a single path prematurely if an input is ambiguous.



- N1 : **conv** (filter size = (11,11), filters = 64, stride = 24, 2x2padding) + **max_pooling**(3x3, stride = 2)
- N2, N3 : **conv** (filter size = (5,5), filters = 96, 2x2padding)
- N4 ~ N8 : **max_pooling**(3x3, stride = 2) + **conv** (filter size = (3,3), filters =160) + **conv** (filter size = (3,3), filters =128) + **conv** (filter size = (3,3), filters =128) + **max_pooling**(3x3, stride = 2) + **fully_connected**(2048) + **fully_connected**(2048) + **fully_connected**(2 fine-class output)
- Q1, Q2 : **conv** (filter size = (5,5), filters = 16, 2x2padding) + **max_pooling**(3x3, stride = 2) + **conv** (filter size = (5,5), filters = 32) + **max_pooling**(3x3, stride = 2) + **fully_connected**(1024) + **fully_connected**(1024) + **fully_connected**(2-action output)
- Q3 ~ Q7 : **conv** (filter size = (5,5), filters = 16, 2x2padding) + **max_pooling**(3x3, stride = 2) + **conv** (filter size = (3,3), filters = 32) + **max_pooling**(3x3, stride = 2) + **fully_connected**(1024) + **fully_connected**(1024) + **fully_connected**(2-action output)

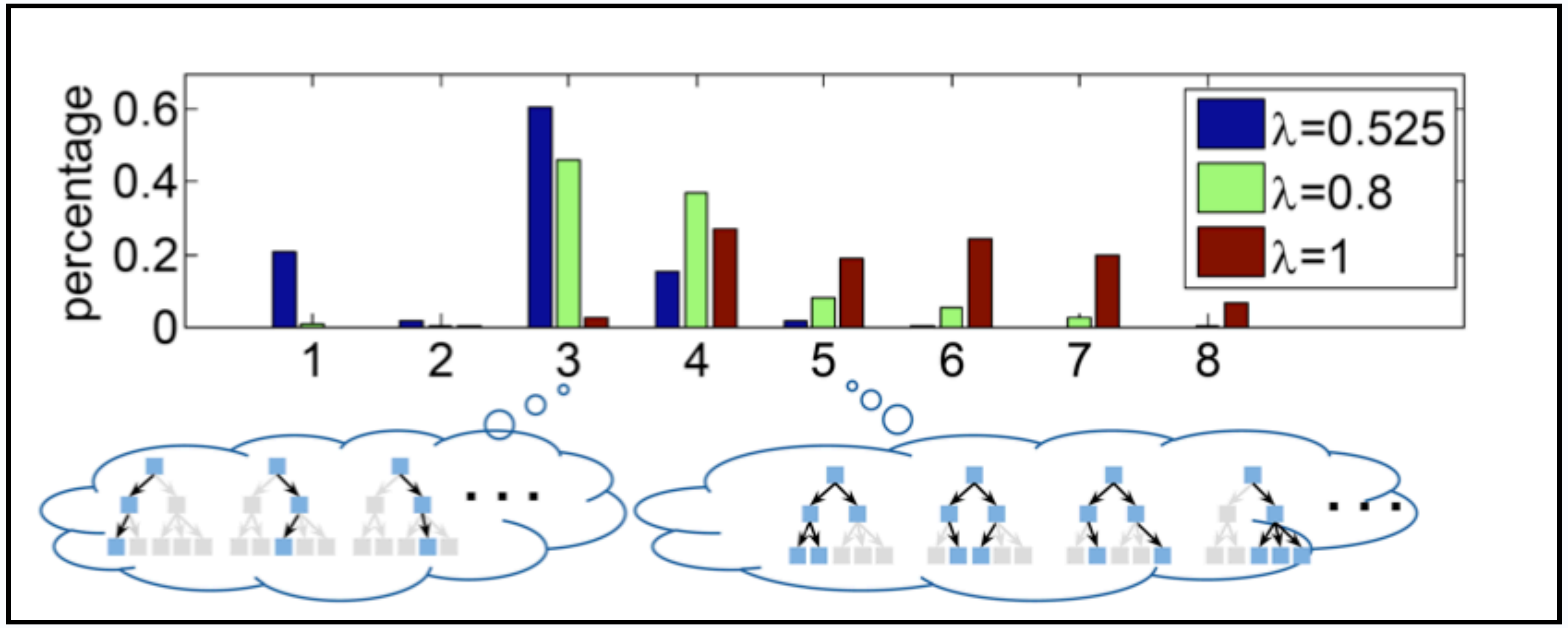
Experiments

Hierarchical DNN

- In **accuracy-cost graph**,
  - We can see the hierarchy D^2NN can match the accuracy of the full network with about half of the computational cost.

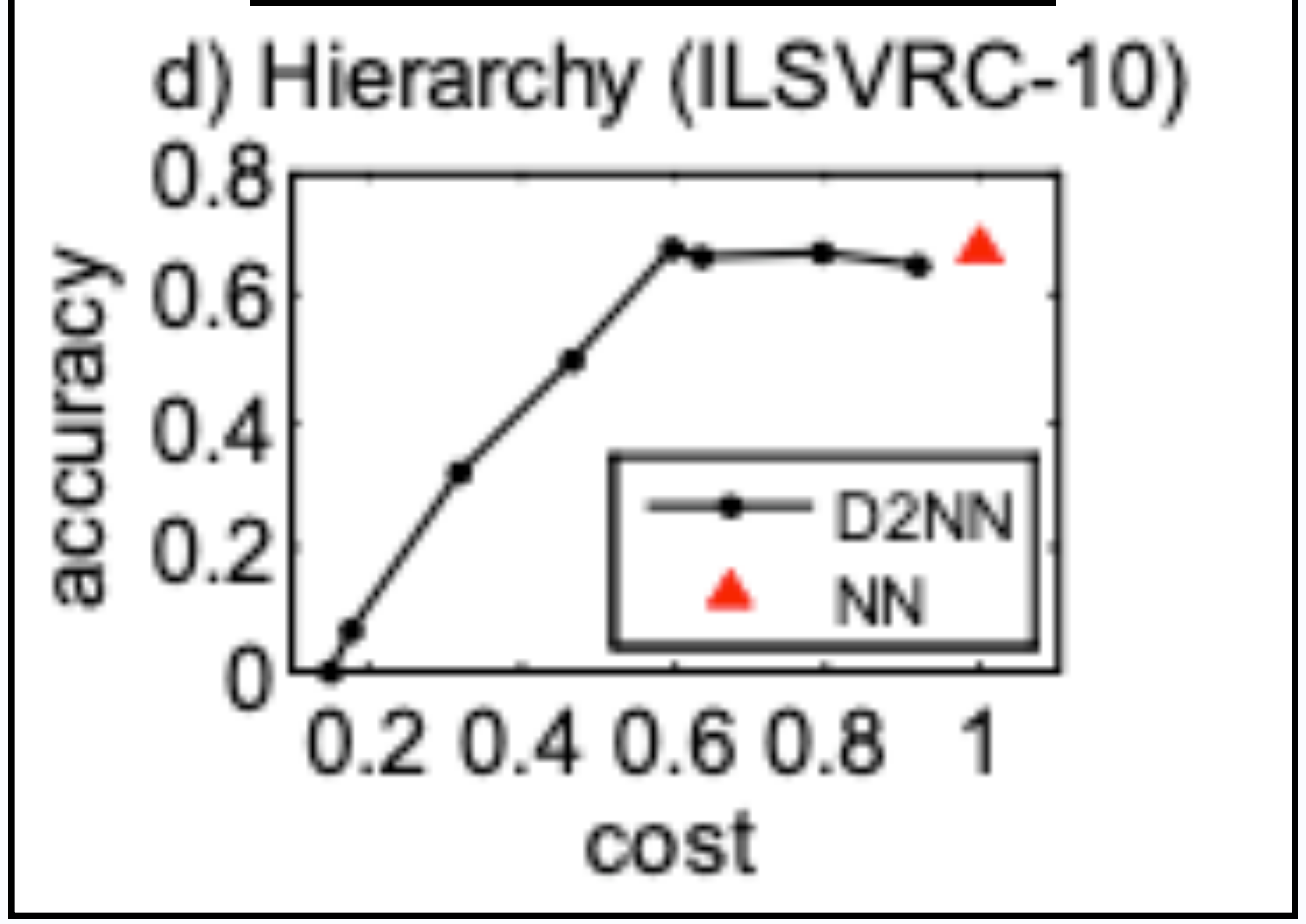
- **Fig 4.** plots for the distribution of examples going through execution sequences with different numbers of nodes activated.
- Due to the parallelism, there can be many execution sequences.
- λ increases, accuracy is given more weight and more nodes are activated.

Fig 4. (right)



- x-axis = number of nodes activated

accuracy-cost graph



- Accuracy : proportion of correctly classified test examples.
- Cost : number of multiplication

Experiments

Comparison with Dynamic Capacity Networks

- Compare D^2NN (Chain design D^2NN)with Dynamic Capacity Networks(DCN)
  - Efficiency measurement = absolute number of multiplications
  - Dataset : Cluttered MNIST
- DCN applies additional high capacity subnetwork for certain image
  - Idea is that more intensive processing is only necessary for certain image regions.

- Achievement :
  - DCN
    - Accuracy : 0.9861
    - Efficiency : 2.77 **x** 10^7
  - D^2NN
    - Accuracy : 0.9698
    - Efficiency : 2.66 **x** 10^7

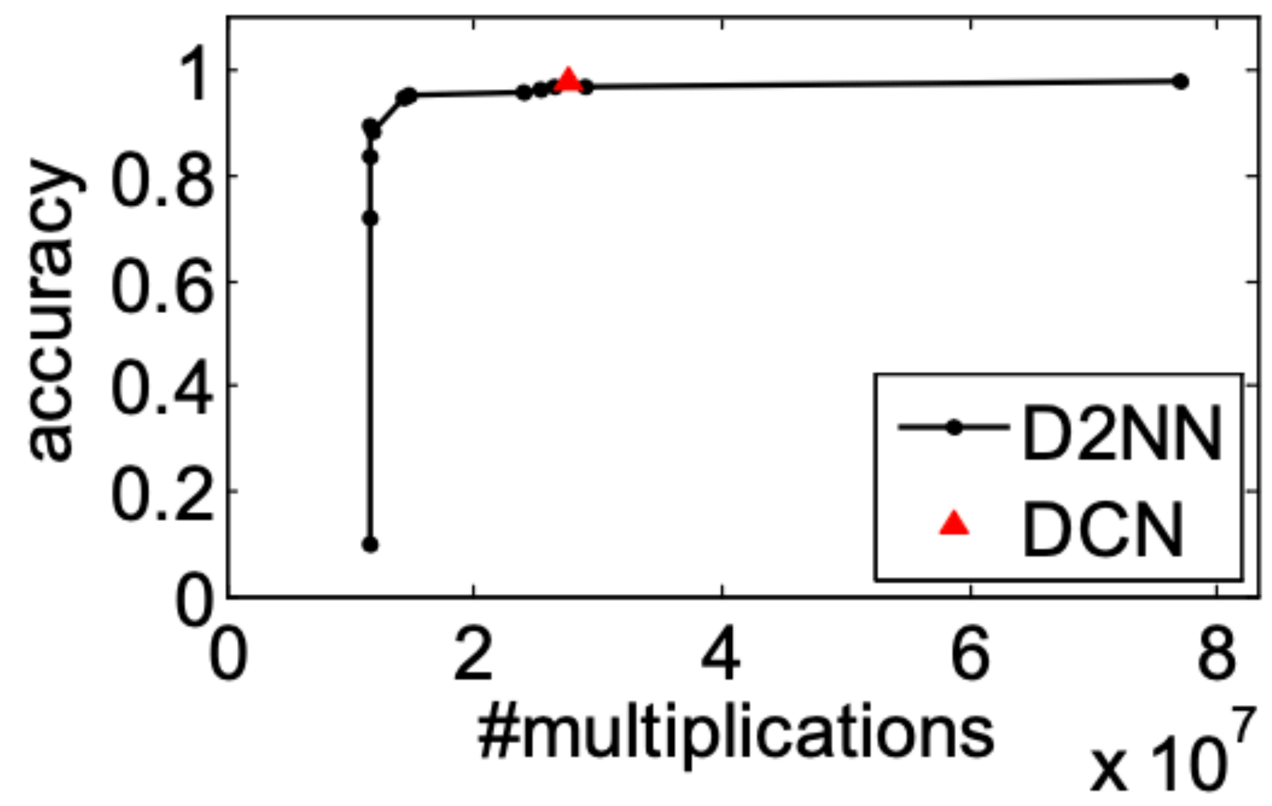


Figure 6. Accuracy-cost curve for a chain D^2NN on the CMNIST task compared to DCN [2].

Experiments

Visualization of Examples in Different Paths

- **LEFT** image = face examples in the high-low D^2NN for
  - Examples in low-capacity path are more frontal than high-capacity path
- **RIGHT** image = car examples in the hierarchical D^2NN with
  1. Single path executed
  2. Full graph executed ($\lambda = 1$)

→ Show single path executed should be easier to classify than full graph executed.

Figure 5. Examples with different paths in a high-low D^{2}NN (left) and a hierarchical D^{2}NN (right).

Experiments

Cifar-10 Results

- Train a cascade D^2NN on CIFAR-10
- Initialize this D^2NN with pre-trained ResNet-110weights, apply cross-entropy losses on regular nodes, and tune the mixed-loss weight as explained in Sec 4.
  - Result
    - **30% reduction** of cost with **2% loss** on accuracy
    - **62% reduction** of cost with **7% loss** on accuracy
- In CIFAR-10, all images are low resolution(32x32) few images are significantly easier to classify.
- As a result,
  - The efficiency improvement is **modest** compared to other datasets.

Experiments

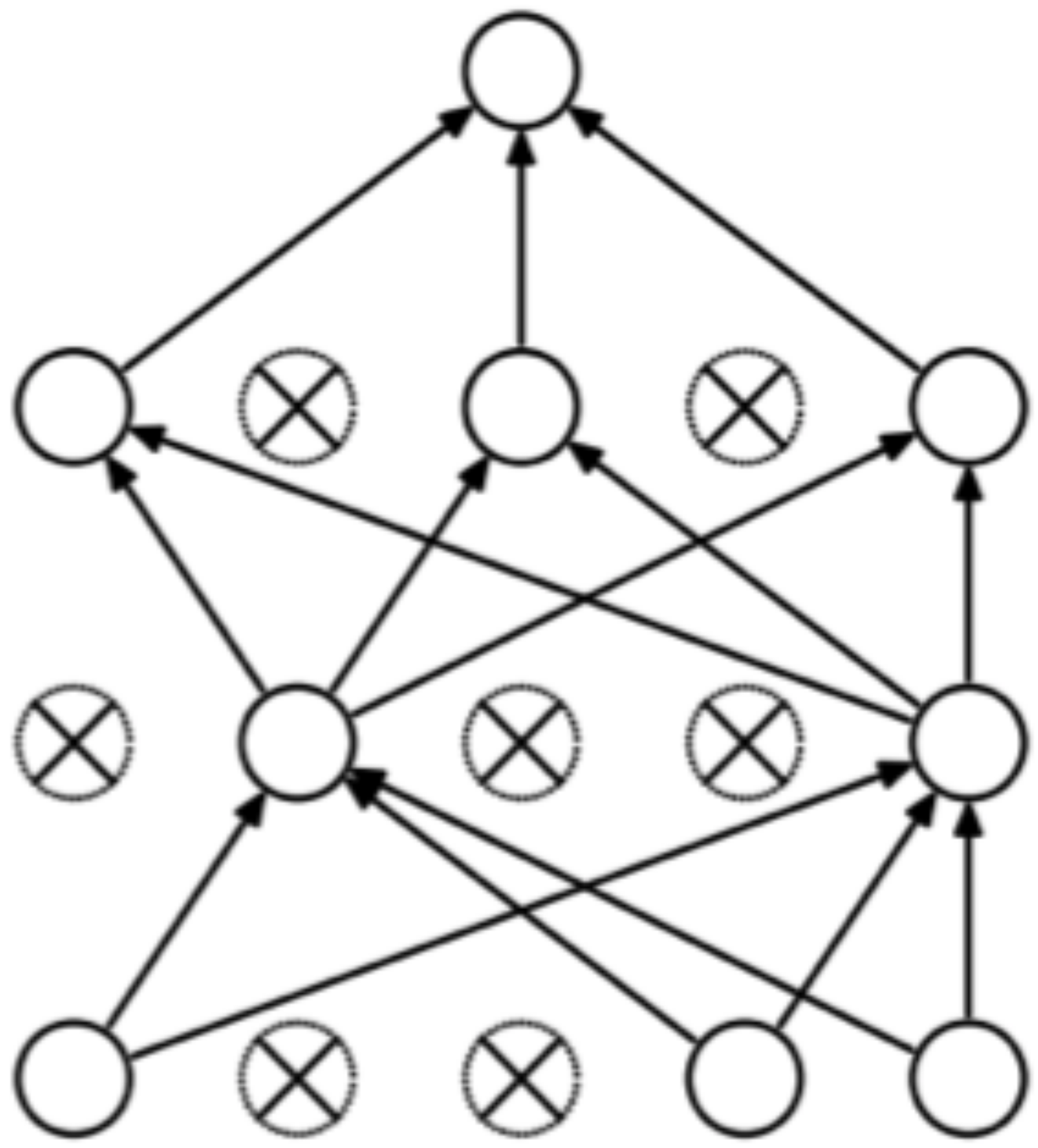
Conclusion

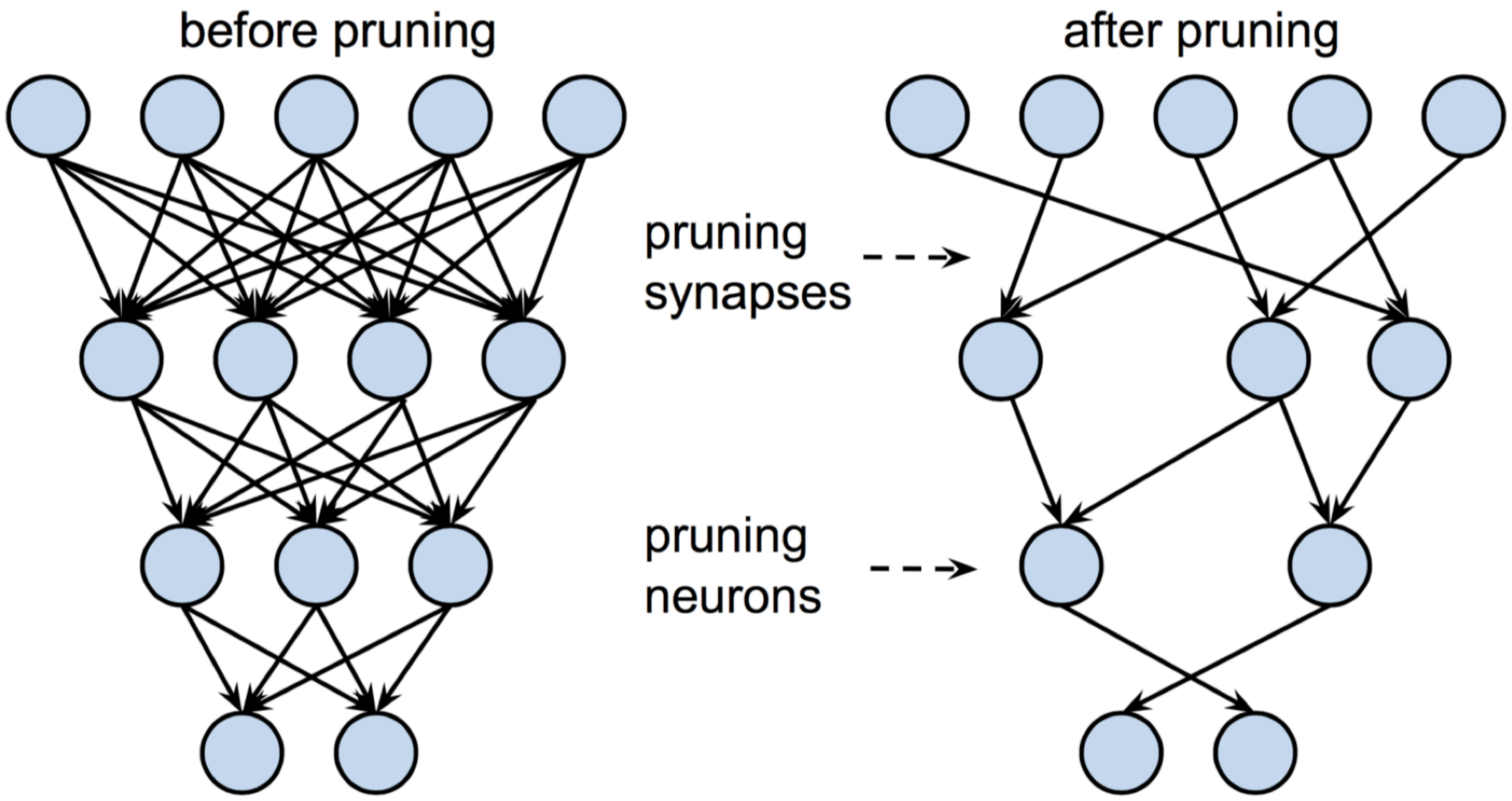
- Introduced Dynamic Deep Neural Networks with **selective execution.**
- Extensive experiments have shown that D^2NN are flexible and effective for optimizing accuracy-efficiency trade-offs.

Thank you.

Backup Slides

Dropout VS Pruning





End-to-end learning

- D^2NN is trained end to end.
  - Optimize the weights by considering the inputs and outputs directly
  - 반대의 의미로 divide-and-train도 있음.
  - Regular models and control modules are jointly trained to optimize both accuracy and efficiency.

- Achieve such training by integrating back propagation with reinforcement learning, necessitated by the non-differentiability of control modules.

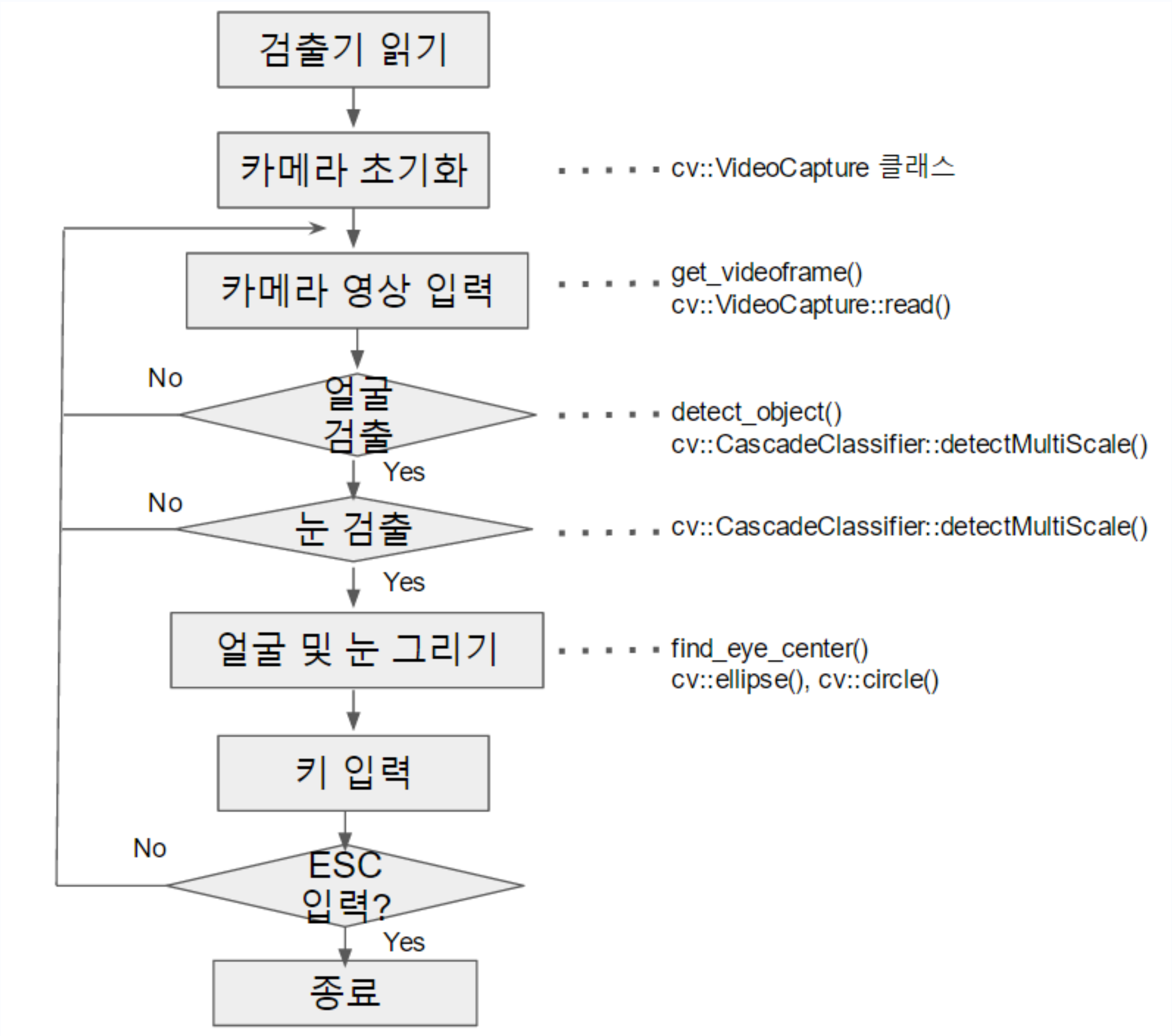
Introduction

- **Main Contribution**
  - Allows user to augment a static feed-forward network with control modules to achieve dynamic selective execution.
  - Provides a new tool for designing and training computationally efficient neural network models.

- **Advantages**
  - Improve computational efficiency by selective execution
    - Pruning unnecessary computation depending on input.
  - It makes possible to use a bigger network under a computation budget by executing only a subset of the neurons each time.

Experiments

Cascade



Experiments

Comparison with Dynamic Capacity Networks

- Compare D^2NN (Chain design D^2NN)with Dynamic Capacity Networks(DCN)
  - E

- Achievement :
  - D