Dynamic Slicing for Deep Neural Networks

Ziqi Zhang, Yuanchun Li, Yao Guo,
Xiangqun Chen, Yunxin Liu

北京大學
PEKING UNIVERSITY

Microsoft

Overview

Program slicing is widely used in software engineering to help debugging, testing and verification

```
int i;
int sum = 0;
int product = 1;
for (i=0; i<N; i++){
    sum = sum + 1;
    product = product * i;
}
write(sum);
write(product);
```

Original program

```
int i;
int sum = 0;

for (i=0; i<N; i++){
    sum = sum + 1;

}
write(sum);
```

Slice on the criterion
<write(sum), sum>

But it is limited to traditional programs

Overview

Deep neural networks achieve remarkable success and is considered as ``software 2.0"

DNN slicing: *computing a subset of neurons and synapses that may significantly affect the values of certain interested neurons*

Applications

- Adversarial defense
- Network simplification and pruning
- Model protection

Background and Motivation

Deep neural networks

A deep neural network is composed of neurons and synapses

- Neurons: collect signals and perform mathematical operations
- Synapse: transmit signals

Program slicing

A program slice S consists of all statements in the program P that may affect the value of variable v in a statement v

Slicing criterion $C = (x, v)$

Category

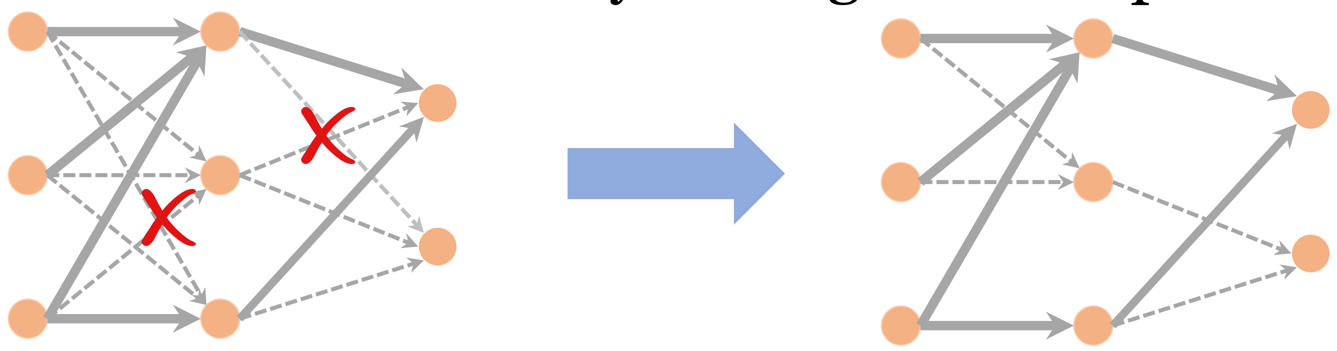
- Static slicing
- Dynamic slicing
- …

Background and Motivation

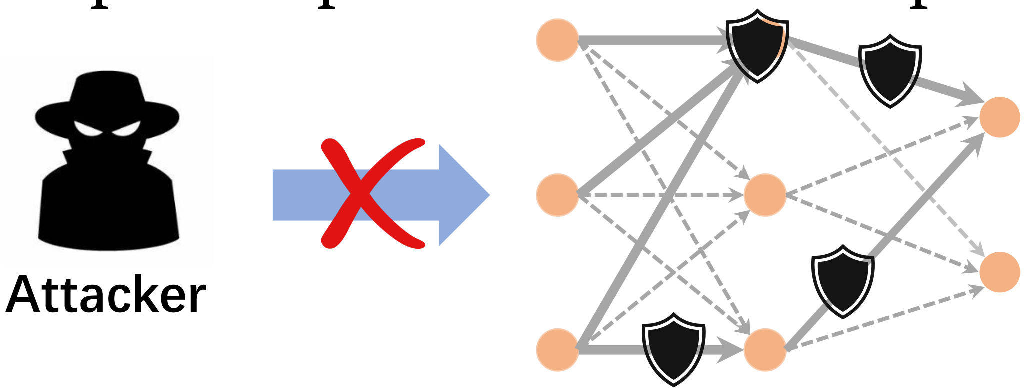
Motivation of slicing a DNN

- The data flow analysis of slicing technique helps analyze the DNN decision logic

- Slicing can reduce the size of DNN by finding the unimportant neurons and synapses



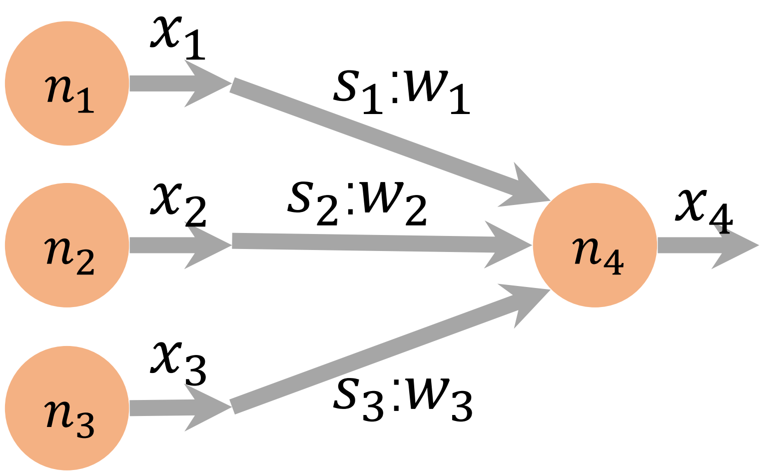
- Slicing can select the important part of the DNN and protect them with low cost



Problem Formulation

Neuron and synapse

- A neuron n takes several numerical inputs and yields one numerical output
- n has synapses $s_1, s_2, \ldots, s_k$ weighted with $w_1, w_2, \ldots, w_k$, respectively.
- Each synapse s_i scales the activation value of a preceding neuron x_i with w_i and pass the scaled value to neuron n as input.



Problem Formulation

Neural network slicing

$M_C = (N_C, S_C)$ - significantly contributes to the value of any output $o \in O$ for any input sample $\xi \in I$.

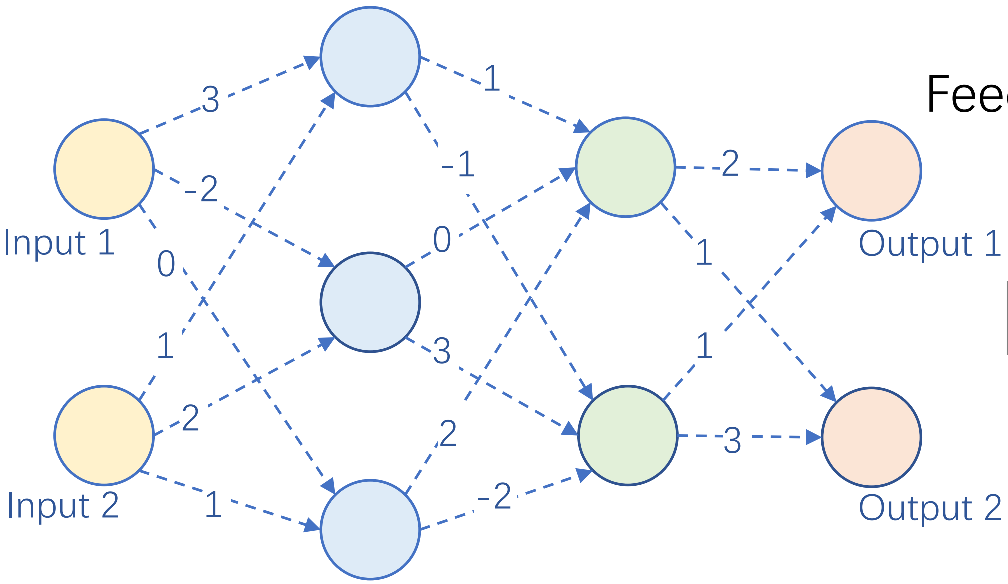
Notation	Meaning
$M = (N, S)$	A neural network
$C = (I, O)$	A slicing criterion
$I = \xi_1, \xi_2, \dots, \xi_n$	a set of model input samples of interest
$O = o_1, o_2, \dots, o_k$	a set of M's output neurons of interest

Challenges

- Understanding the behavior of each neuron
- Quantifying the contribution of each neuron
- Dealing with large models

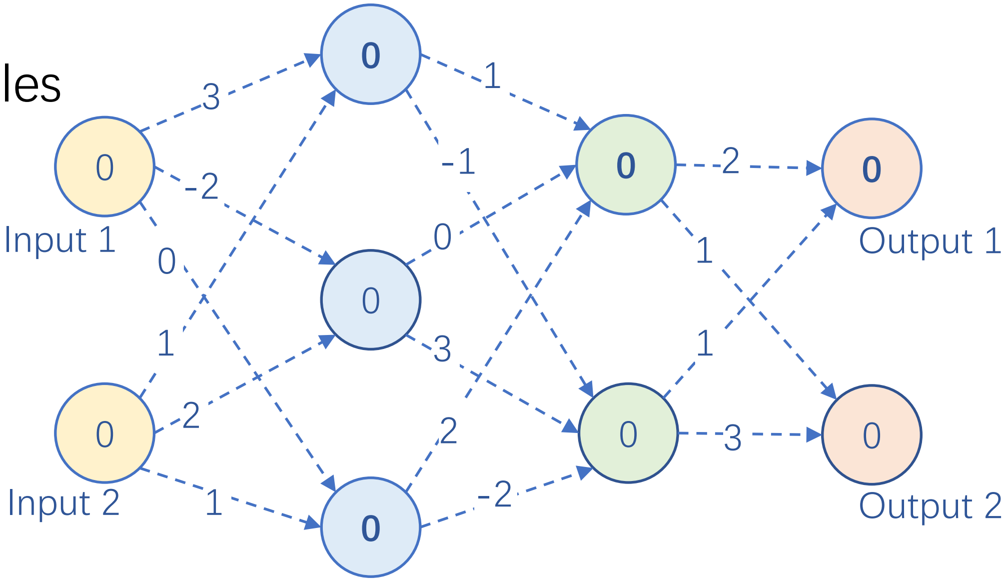
Approach: NNSlicer

Overview



Pretrained model

① **Profiling**
Feed all training samples to the model

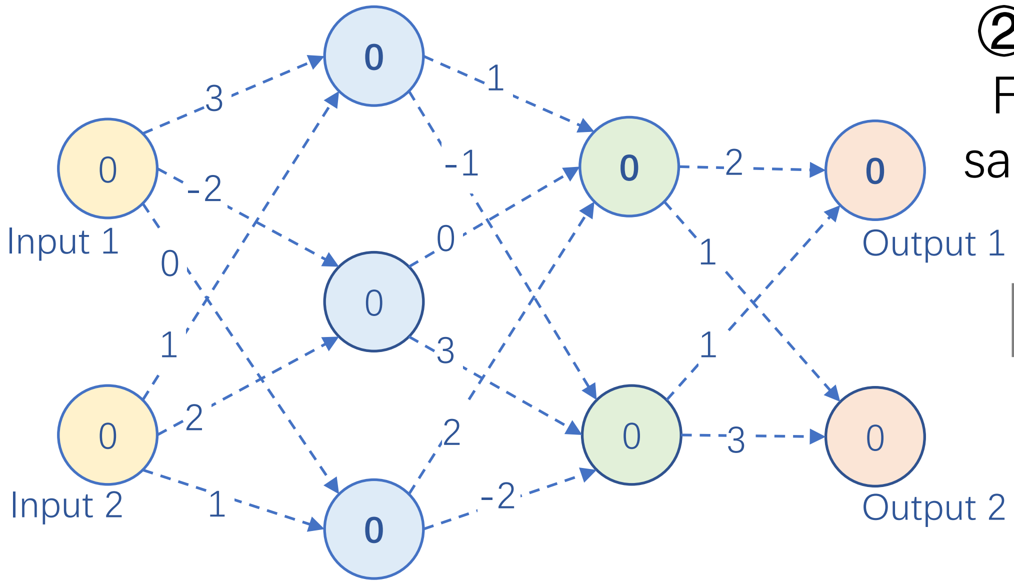


Each neuron's average activation value

Profiling: record the activation value of each neuron for all input samples and compute the mean value

Approach: NNSlicer

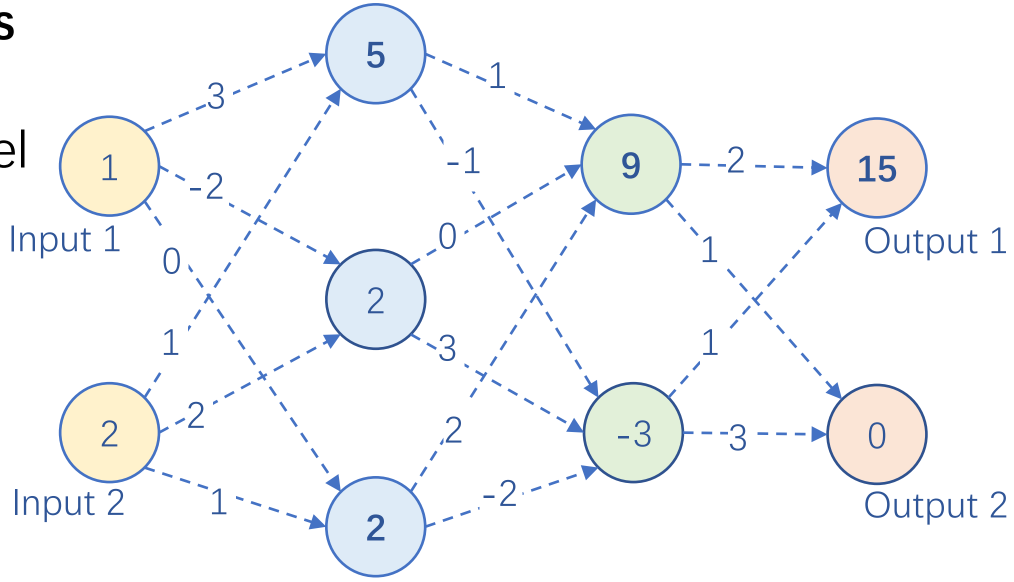
Overview



Each neuron’ s average activation value

② Forward analysis

Feed the interested samples to the model

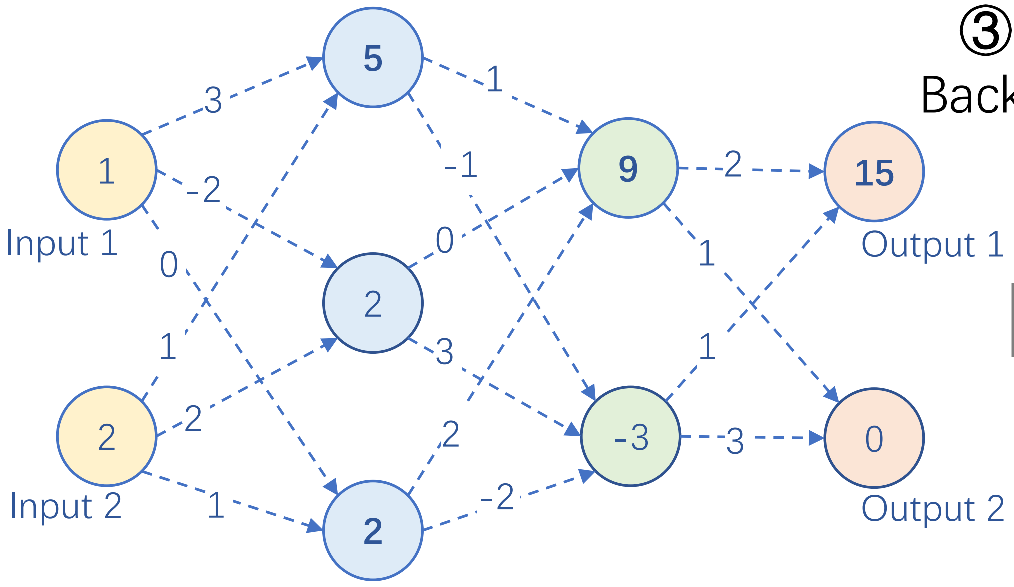


Each neuron’ s reaction to input (1,2) i.e. difference between the activation value and the average

Forward analysis: record the activation value and compute the difference between the profiled mean value and the recorded value

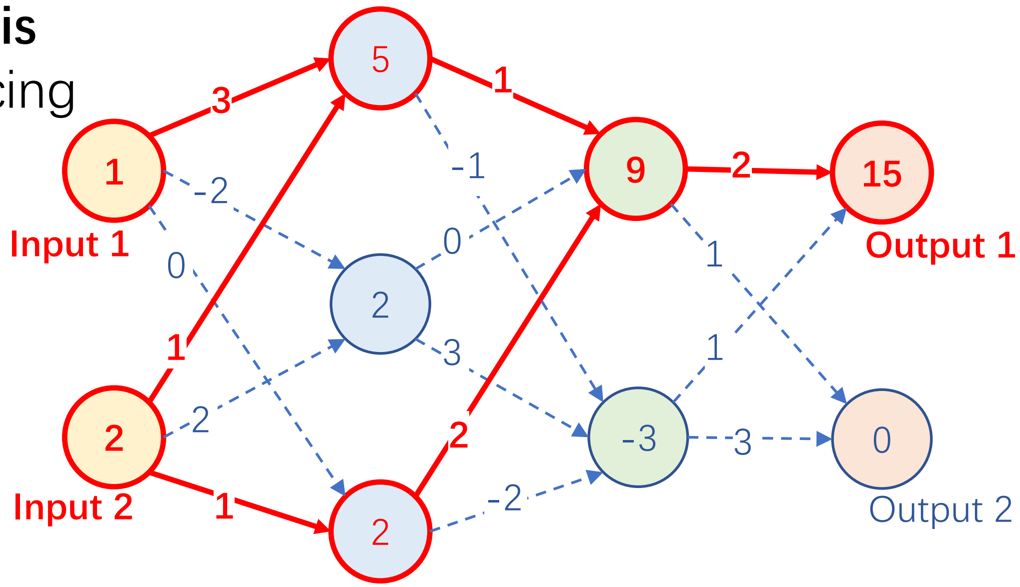
Approach: NNSlicer

Overview



Each neuron' s reaction to input (1,2)
i.e. difference between the activation value and the average

③ **Backward analysis**
Backtrack from the slicing criterion neuron



The slice for output 1 and input (1,2)

Backward analysis: iteratively compute the contribution of preceding synapses and neurons

Approach: NNSlicer

Profiling

For each sample, observe an activation value $y^n(\xi) = mean_{i=1}^{m} y_i^n(\xi)$

Average neuron activation over the training dataset $\overline{y^n(D)} = \frac{\sum_{\xi \in D} y^n(\xi)}{|D|}$

Forward analysis

Quantify the reaction of a neuron n for a data sample ξ as a relative value

$$\Delta y^n(\xi) = y^n(\xi) - \overline{y^n(D)}$$

Symbol	Meaning
D	Dataset
y_i^n	The i th activation value of neuron n
ξ	An input sample

Approach: NNSlicer

Backward analysis

A neural network can be viewed as a densely connected data flow graph

Recursively compute the contributions from back to front

- Consider the neurons that are directly connected to the interested neurons
- Set the neurons with non-zero contributions as the target neurons

Operation	Usage	Formula	Local contribution $contrib_i$
Weighted sum	Convolutional layers FC layers	$y = \sum_{i=1}^{k} w_i x_i$	$CONTRIB_n \times \Delta y \times w_i \Delta x_i$
Average	Average-pooling layers	$y = \frac{1}{k}\sum_{i=1}^{k} x_i$	$CONTRIB_n \times \Delta y \times \Delta x_i$
Maximum	Max-pooling layers	$y = \max_{i+1}^{k} x_i$	$CONTRIB_n \times \Delta y \times \Delta x_i$ if $x_i = y$ else 0
Rectify	ReLU activation	$y = x$ if $x > 0$ else 0	$CONTRIB_n \times \Delta y \times \Delta x$ if $x > 0$ else 0
Scale	Batch normalization	$y = \frac{x - \mu}{\sigma}$	$CONTRIB_n \times \Delta y \times \Delta x$

Approach: NNSlicer

Backward analysis

Only update the neurons with large local contribution

- Sort the local contributions
- Find the maximum index j so that

  $\sum_{l=j}^{k} w_l x_l > \theta \sum_{i=1}^{k} w_i x_i$

Algorithm 1 ComputeContrib: Computing the contributions of neurons and synapses to a list of target neurons for an input sample

Require: A neural network model $\mathcal{M} = (\mathcal{N}, \mathcal{S})$, an input sample ξ and a list of target neurons O. A global table $CONTRIB$ that stores the cumulative contribution of each neuron and synapse during the inference pass of ξ, initialized to 0.

```
Terminate if O is empty
Initialize O' = ∅
for each neuron o ∈ O do
    Find o's preceding neurons and synapses (N', S')
    Compute local contributions of N' and S' as contrib
    Update CONTRIB with contrib
end for
for each neuron n ∈ N do
    Add n to O' if n is a predecessor of O and CONTRIB_n ≠ 0
end for
Obtain N' by removing neurons in O from N
Call ComputeContrib by setting O = O' and N = N'
return The global cumulative contribution table CONTRIB
```

Approach: NNSlicer

GPU and multi-thread acceleration

GPU: Profiling and forward analysis of large batch

CPU: Backward analysis of small batch in multiple threads

Implementation

NNSlicer is implemented in Python with TensorFlow

Multi-thread computation is implemented by Ray (https://ray.io)

Overhead

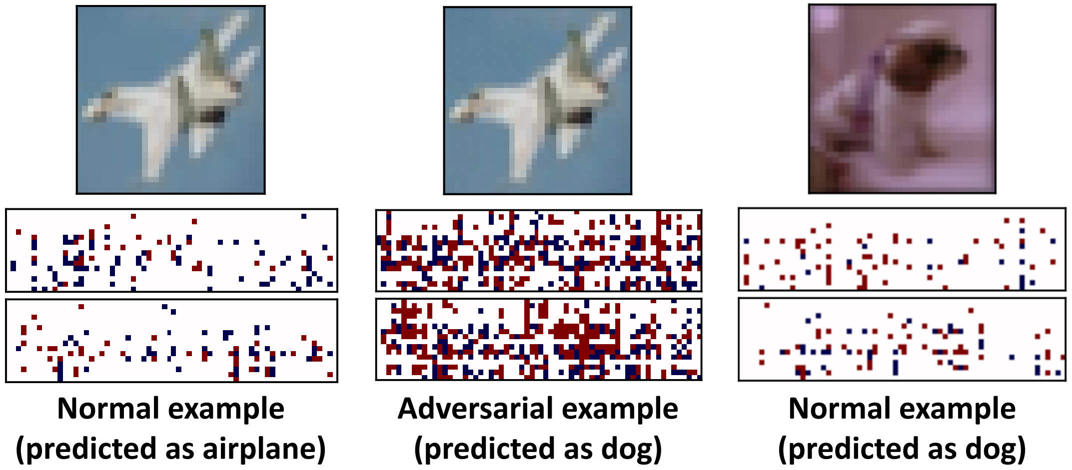
Model	#Params	Profiling / Forward		Backward	
		Single	Batch	Single	Batch
LeNet	42784	3.0s	0.3s	0.5s	0.3s
ResNet10	300K	8.9s	0.4s	30.1s	3.0s
ResNet18	11M	9.6s	0.8s	543.0s	40.4s

Applications: adversarial defense

Slice can be viewed as an abstraction of the decision process

The decision processes of normal examples and adversarial examples are different

Capture the mapping pattern between the slice and the output category



Advantages

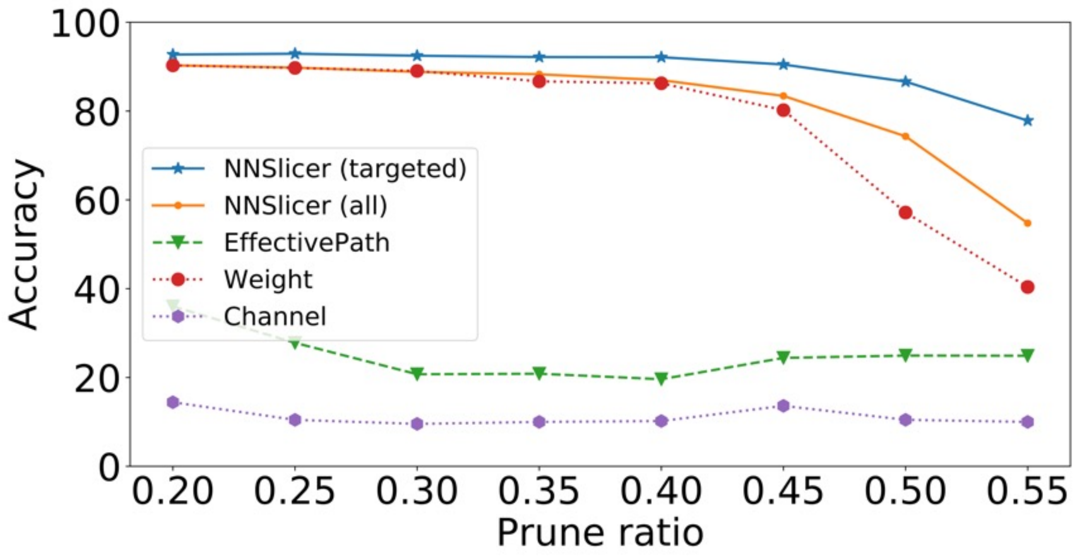
- Do not need to modify the original model
- Scale up to large state-of-the-art DNN models
- Requires only the normal samples

Applications: network simplification and pruning

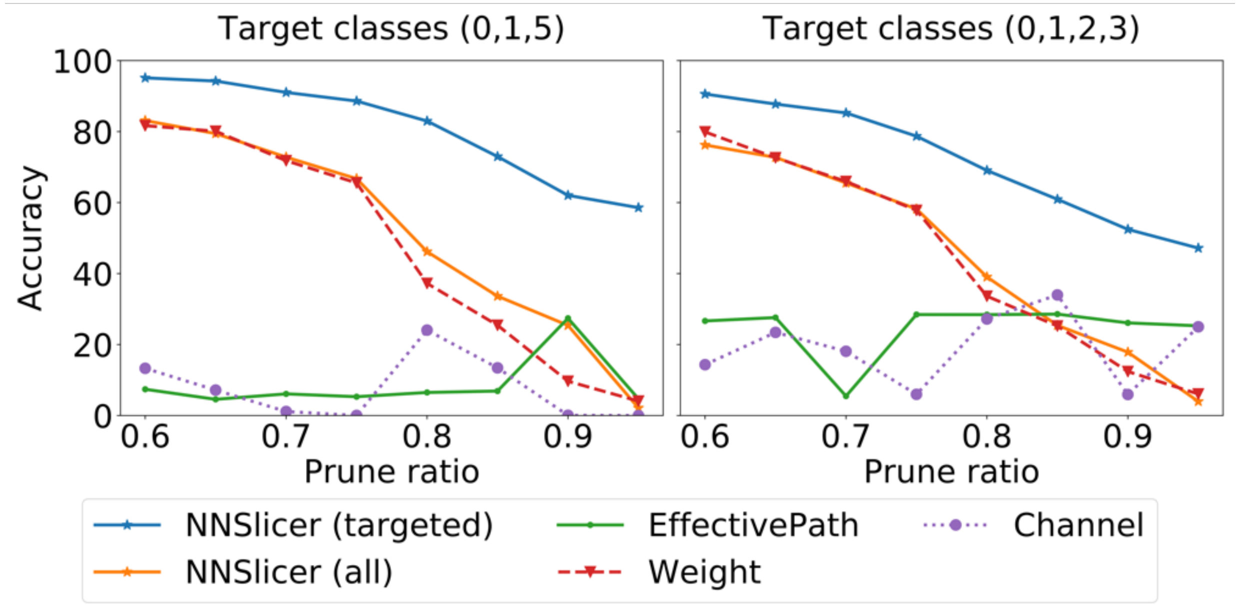
NNSlicer enables flexible pruning by focusing on a subset of output classes

Slicing criterion $C = (I_T, O), I_T \subset I$

Order synapses by the contribution and prune the less contributed synapses



Accuracy without fine-tuning



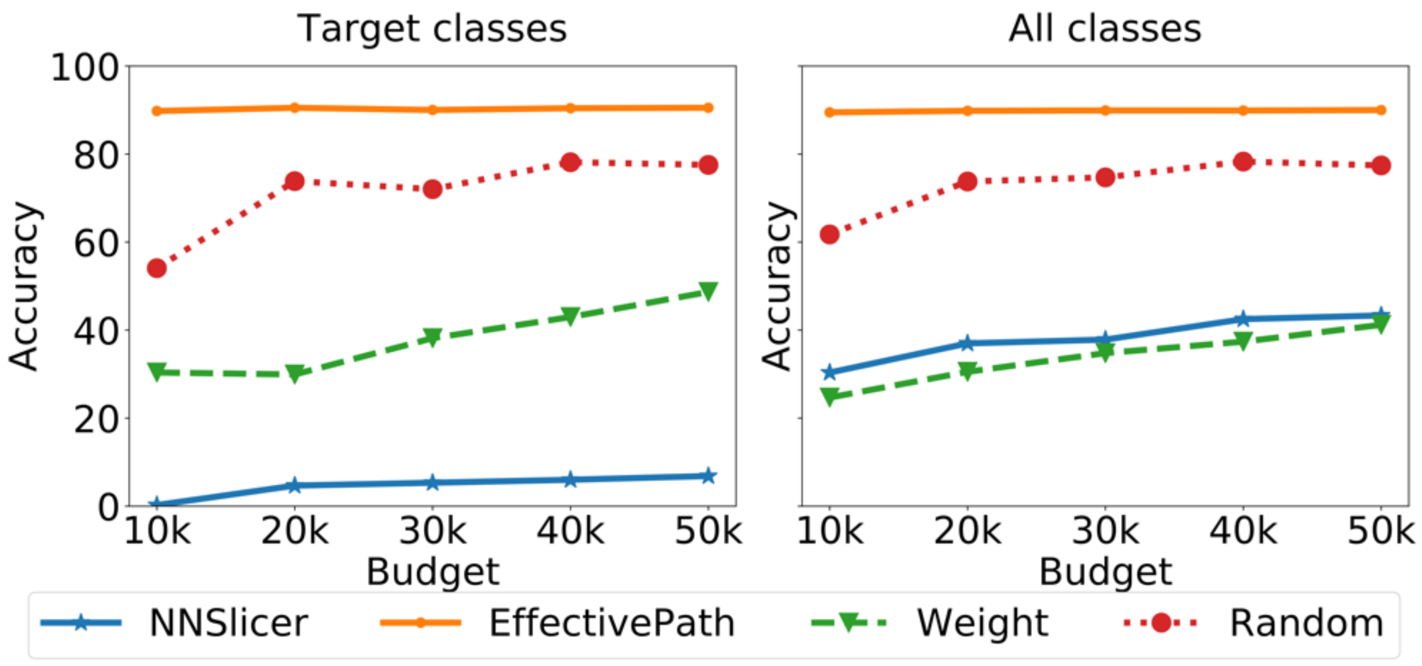
Accuracy after fine-tuning

Applications: model protection

DNN models are valuable assets and model stealing threats the confidentiality

An adversary can reproduce the model with low cost

Select synapses in a similar way as pruning but to select the most crucial ones



Model stealing accuracy after protection

Limitations and Discussion

DNN architecture

NNSlicer can be extended to other architectures such as RNN and GCN

Other slicing techniques

Static slicing, conditioned slicing, amorphous slicing …

Slicing criterion

Slicing for an intermediate neuron is interesting

More applications

Is it possible to compose different slices to a new model?

Is it possible to slice certain attributes from a trained model?

Can NNSlicer be used to debug model or diagnose fragile weights?

Conclusion

Summary of our work

Apply slicing to DNNs We propose an idea of dynamic slicing on deep networks and and implement a tool called NNSlicer

A data-flow analysis process NNSlicer consists of a profiling phase, a forward analysis phase and a backward analysis phase

Usefulness and effectiveness We develop three interesting applications with NNSlicer: detect adversarial inputs, prune and simplify neural networks and selectively protect the model from stealing. Empirical results demonstrate the usefulness and effectiveness