Are Neural Nets Modular?

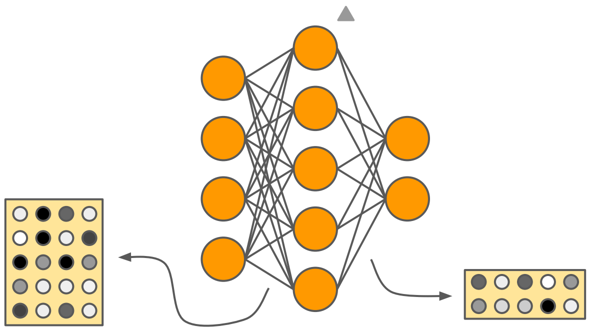
Inspecting Functional Modularity Through Differentiable Weight Masks

Róbert Csordás, Sjoerd van Steenkiste, Jürgen Schmidhuber

ICLR 2021

Inspection via Differentiable Weight Masking

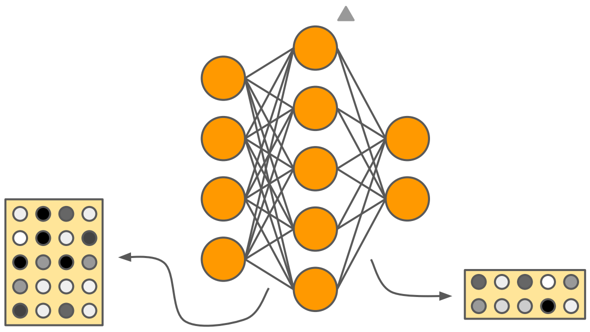
Start with pre-trained weights on the full task



▲ Illustration. We use multi-layer networks and do weight-level analysis.

Inspection via Differentiable Weight Masking

Start with pre-trained weights on the full task

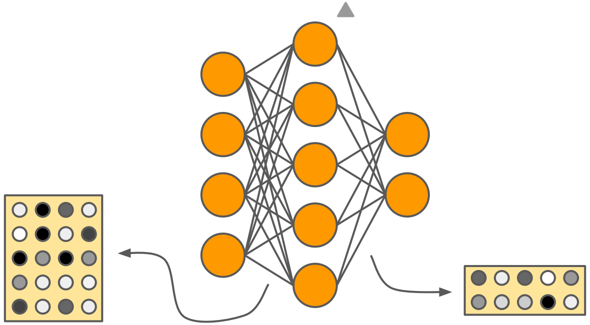


12+45 = 57
42*2 = 84
75+85 = 60
43*25 = 75
...

▲ Illustration. We use multi-layer networks and do weight-level analysis.

Inspection via Differentiable Weight Masking

Start with pre-trained weights on the full task



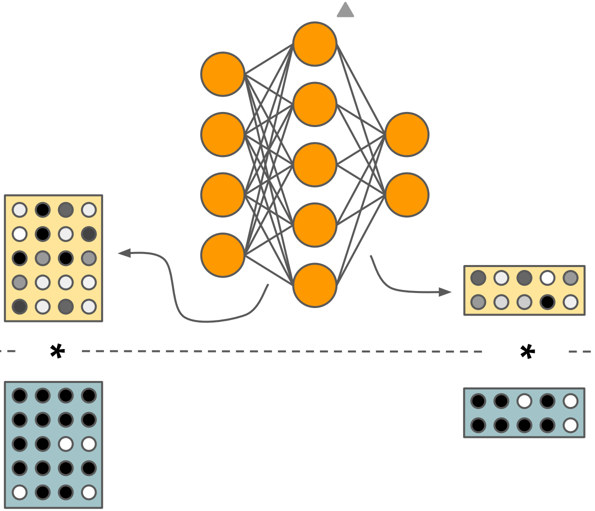
12+45 = 57
42*2 = 84
75+85 = 60
43*25 = 75
...

12+45 = 57
~~42*2 = 84~~
75+85 = 60
~~43*25 = 75~~
...

▲ Illustration. We use multi-layer networks and do weight-level analysis.

Inspection via Differentiable Weight Masking

Start with pre-trained weights on the full task



12+45 = 57
42*2 = 84
75+85 = 60
43*25 = 75
...

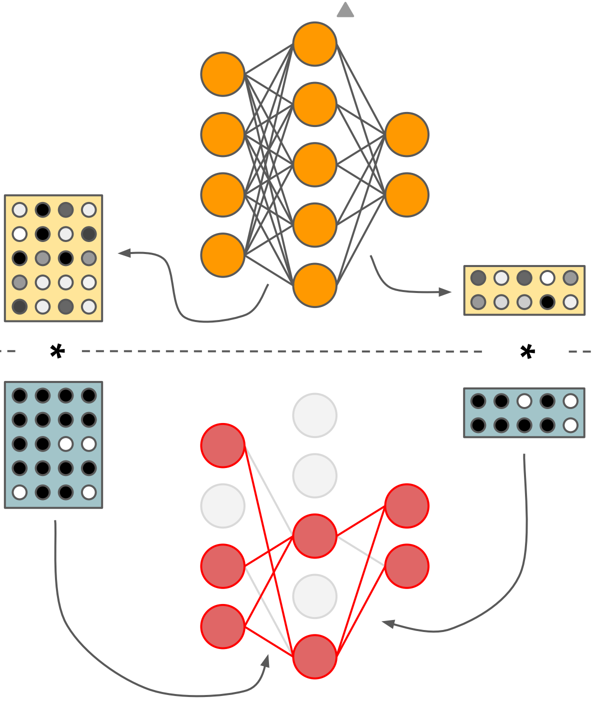
12+45 = 57
~~42*2 = 84~~
75+85 = 60
~~43*25 = 75~~
...

Train binary weight masks on a subtask

▲ Illustration. We use multi-layer networks and do weight-level analysis.

Inspection via Differentiable Weight Masking

Start with pre-trained weights on the full task



12+45 = 57
42*2 = 84
75+85 = 60
43*25 = 75
...

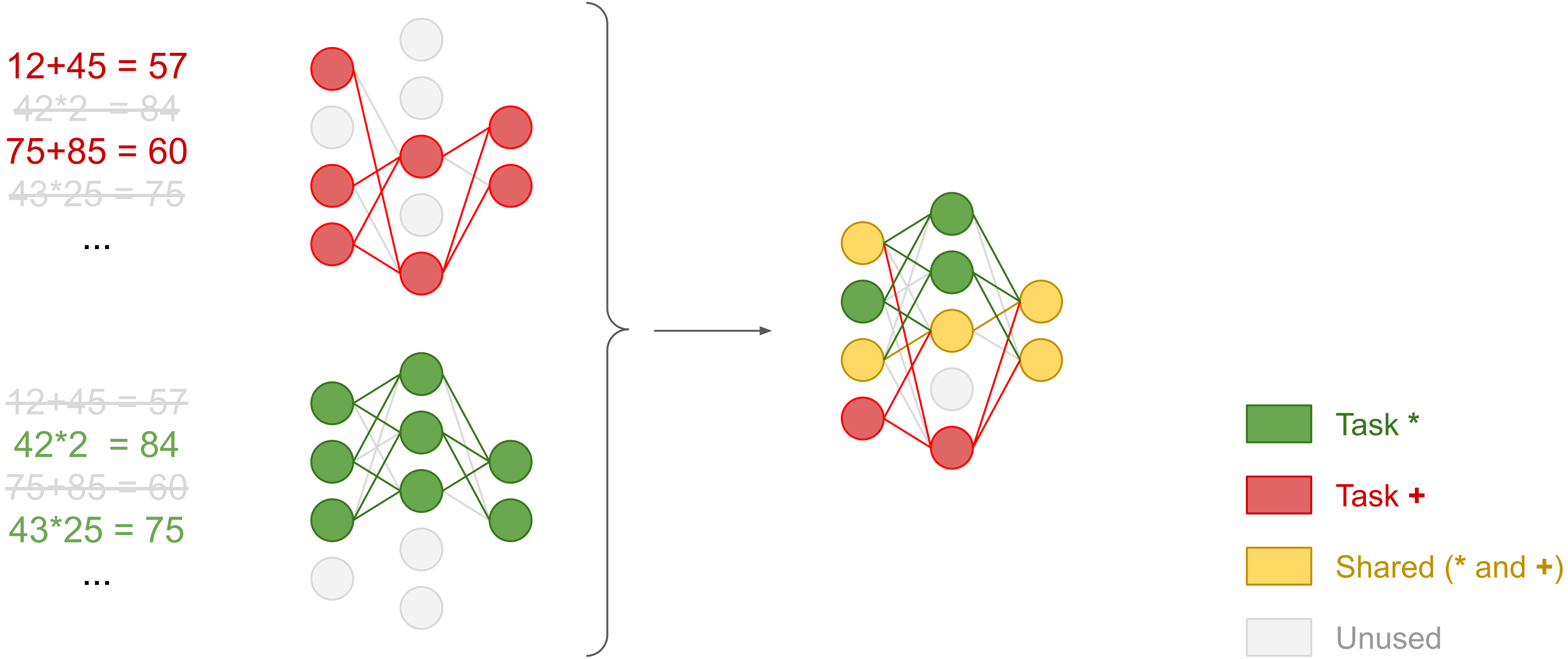
12+45 = 57
~~42*2 = 84~~
75+85 = 60
~~43*25 = 75~~
...

Train binary weight masks on a subtask

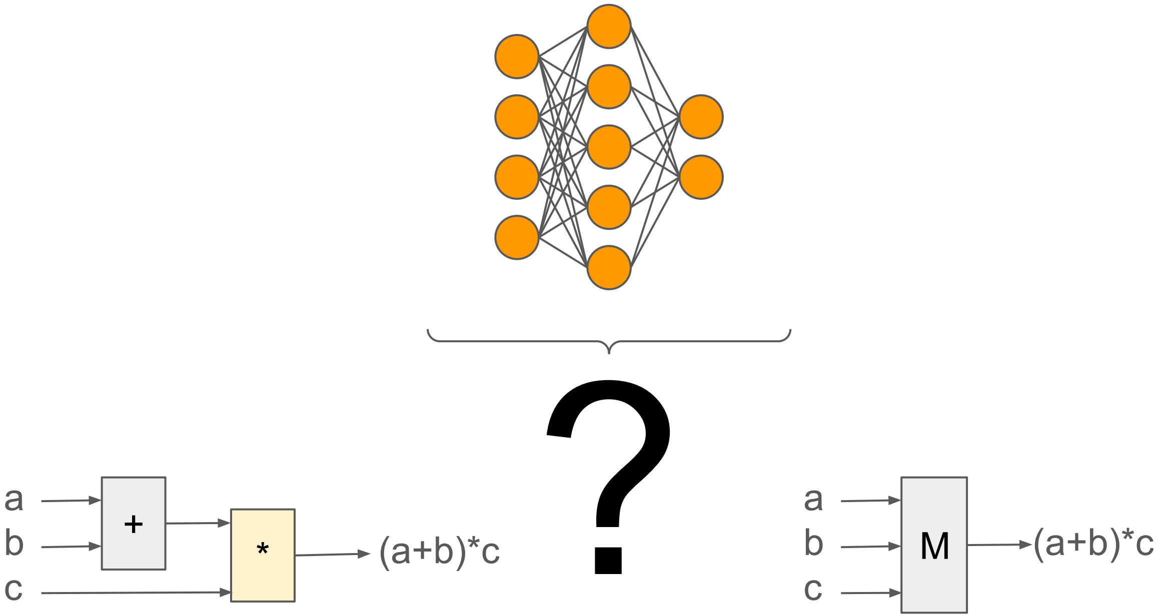
▲ Illustration. We use multi-layer networks and do weight-level analysis.

Inspection via Differentiable Weight Masking

Same is done for all subtasks of interest

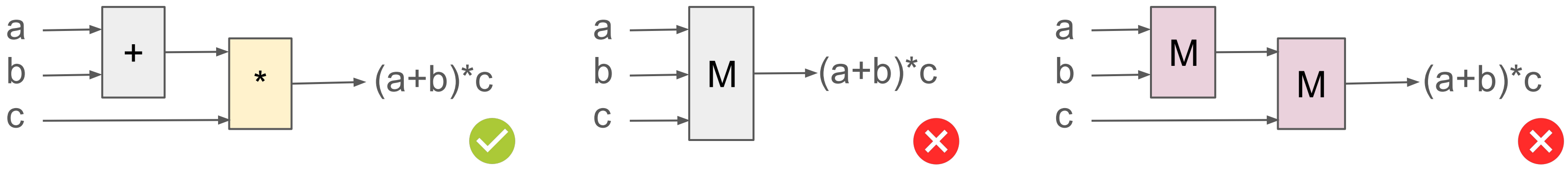


Are neural networks modular?



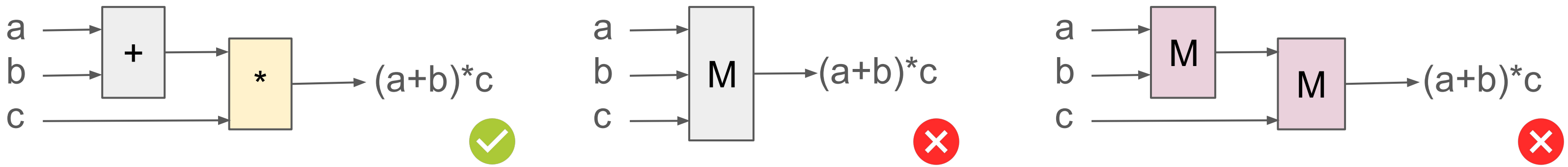
Compositionality and modularity

- $P_{specialize}$: Different modules for separate functions



Compositionality and modularity

- $P_{specialize}$: Different modules for separate functions



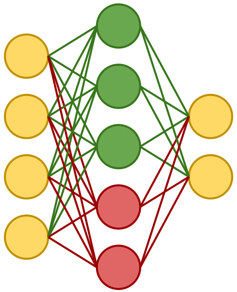
- P_{reuse}: Use the same module for identical functions

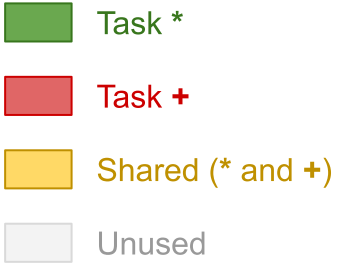
Analysis

$P_{specialize}$: Different modules for separate functions

Input vector:	n_1 n_2 op=*	n_1 n_2 op=+
Output vector:	$n_1 * n_2$	$n_1 + n_2$

Expectation



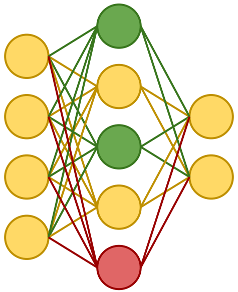


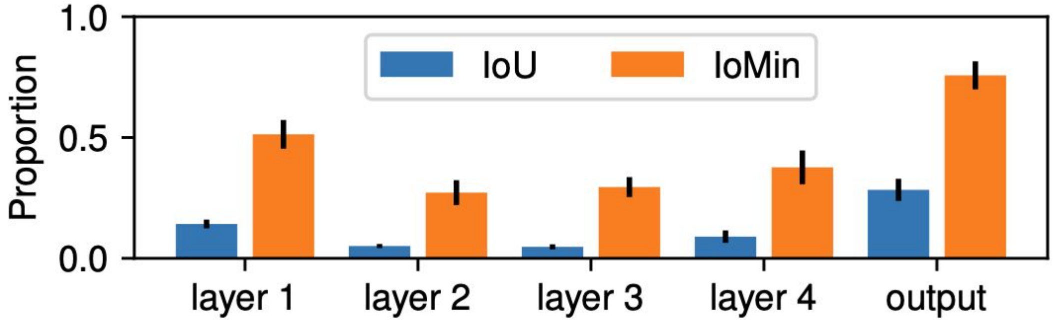
Analysis

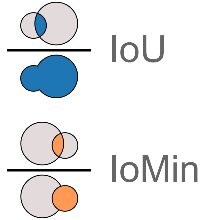
$P_{specialize}$: Different modules for separate functions

Input vector:	n_1 n_2 op=*	n_1 n_2 op=+
Output vector:	$n_1 * n_2$	$n_1 + n_2$

What we found:





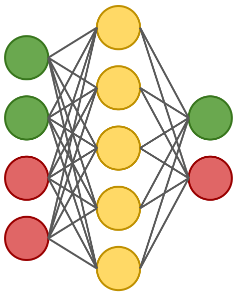


Analysis

P_{reuse}: Use the same module for identical functions

Input vector:	n_1	n_2	0	0	0	0	n_3	n_4
Output vector:	$n_1 + n_2$		--		--		$n_3 + n_4$	

Expectation



Task 1 (n_1+n_2)

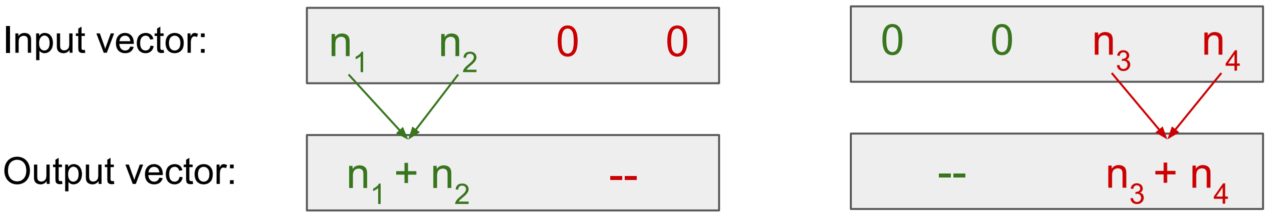
Task 2 (n_3+n_4)

Shared

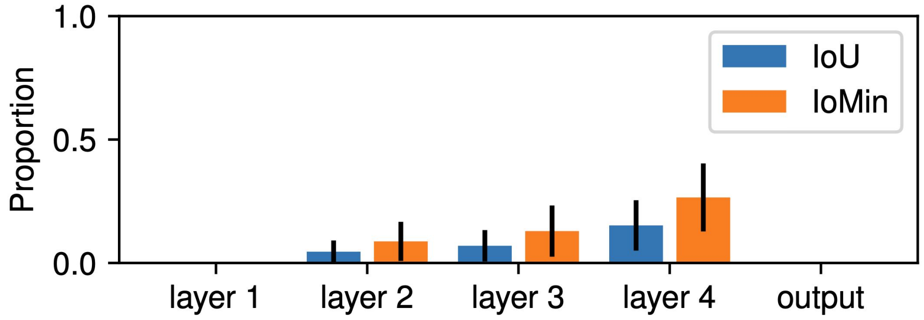
Unused

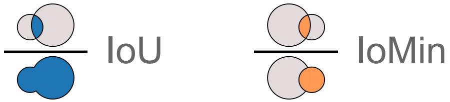
Analysis

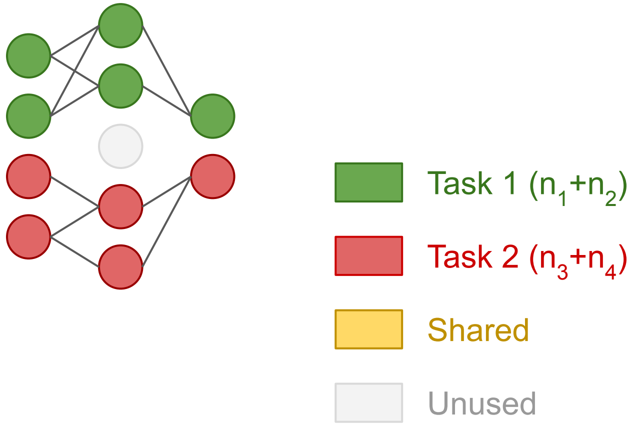
P_{reuse}: Use the same module for identical functions



What we found



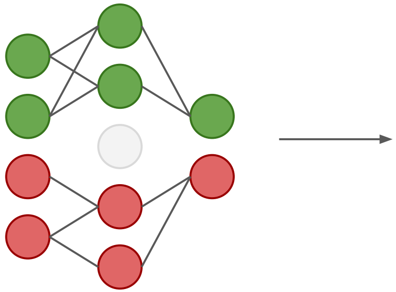




Analysis

P_{reuse}: Use the same module for identical functions

Confirmation: invert masks



Task 1 (n_1+n_2)

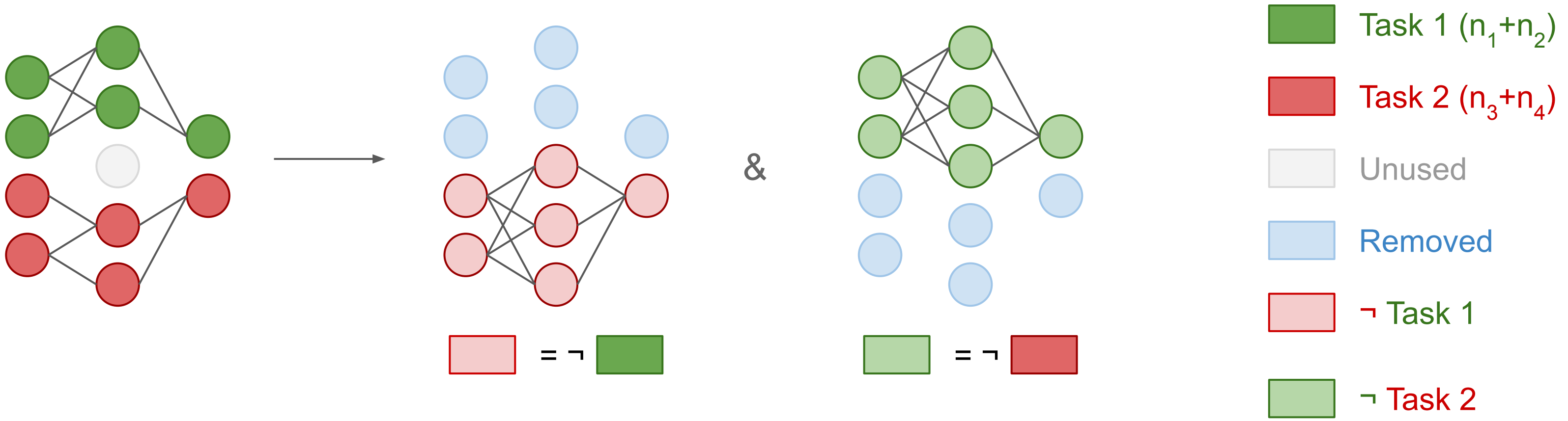
Task 2 (n_3+n_4)

Unused

Analysis

P_{reuse}: Use the same module for identical functions

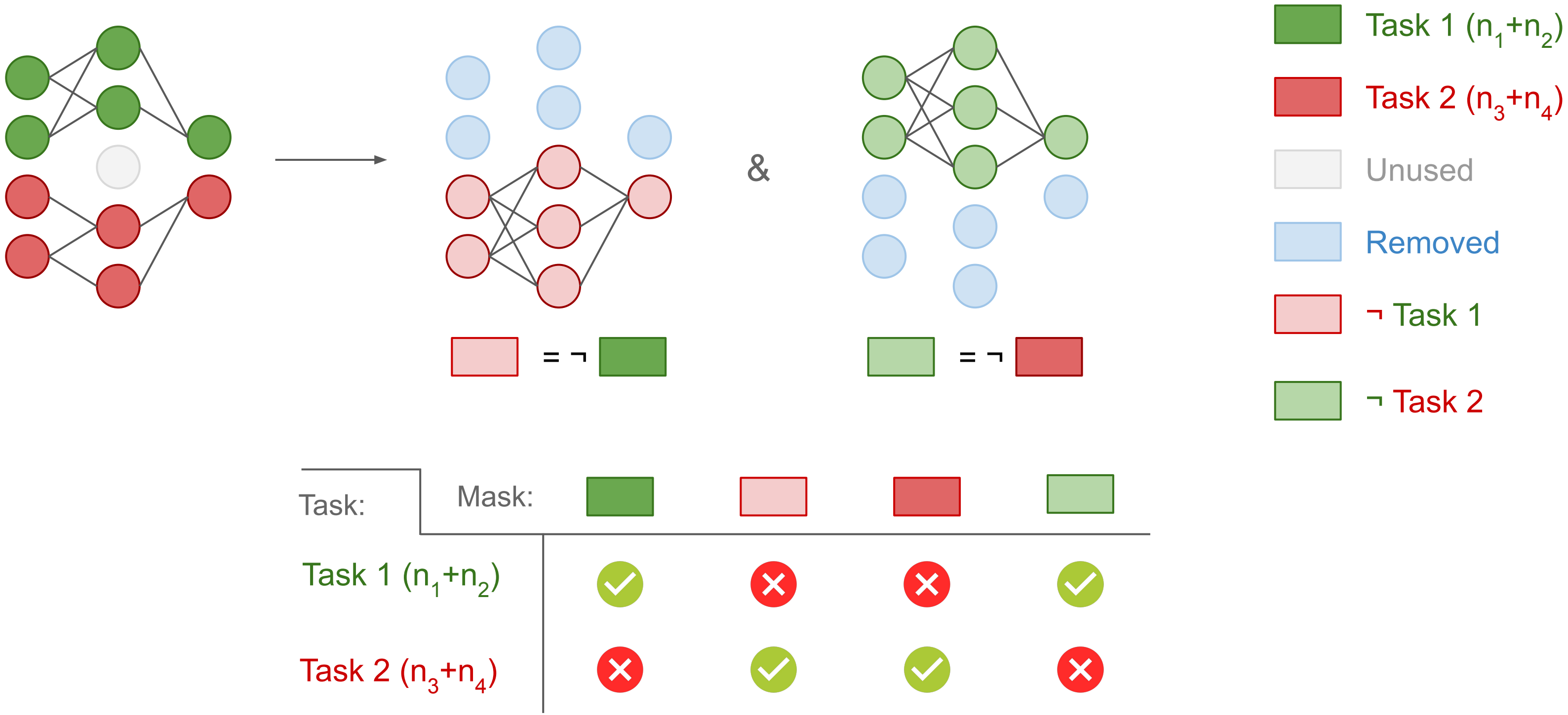
Confirmation: invert masks



Analysis

P_{reuse}: Use the same module for identical functions

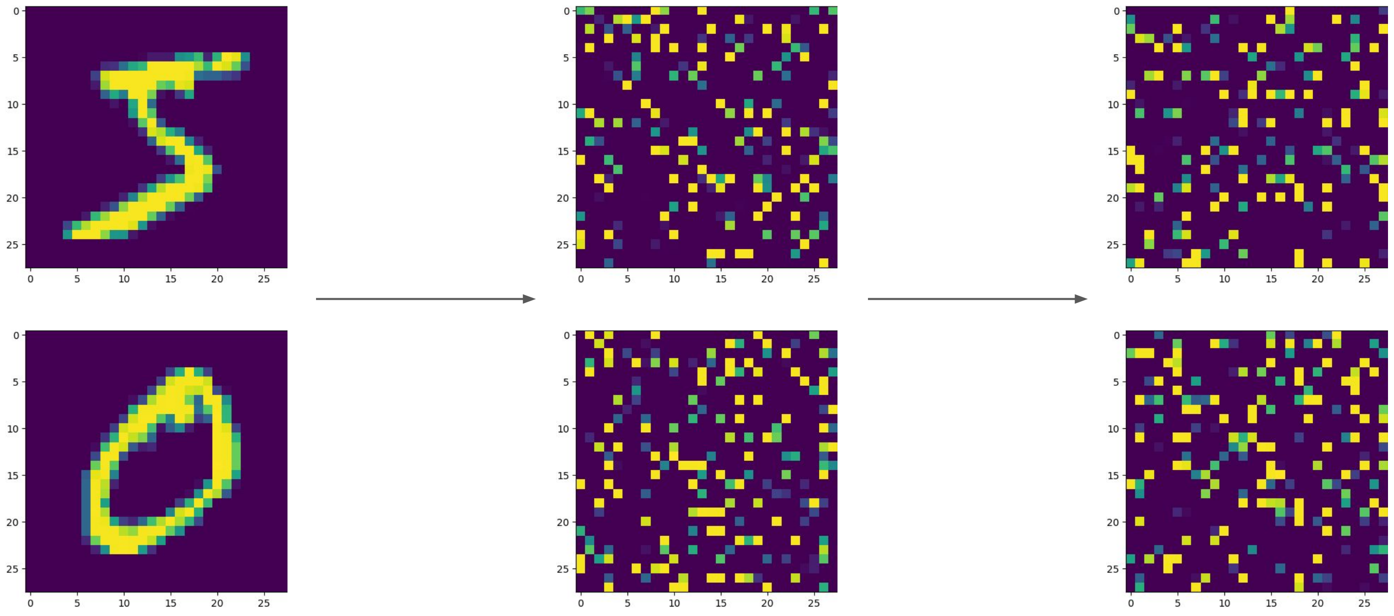
Confirmation: invert masks



Analysis

P_{reuse}: Use the same module for identical functions

- We also test the effect on sequentially training to classify new permutations of MNIST.

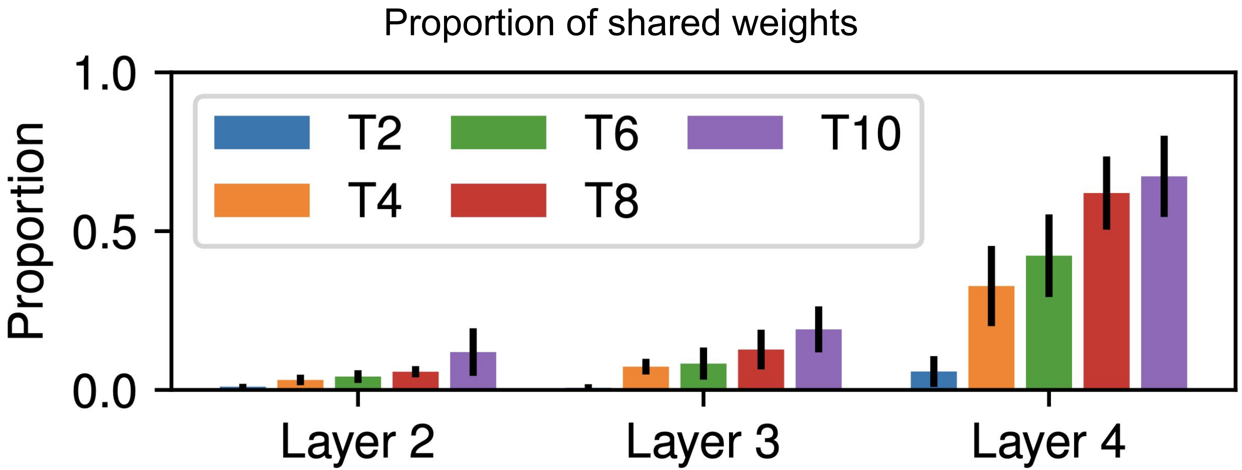


T0 T1 T2

Analysis

P_{reuse}: Use the same module for identical functions

- We also test the effect on sequentially training to classify new permutations of MNIST.
- Layers are not reused, even though re-learning the first layer is enough.



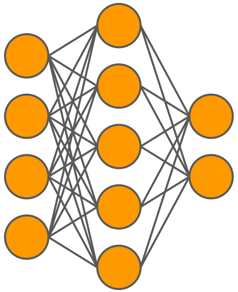
Analysis - summary

$P_{specialize}$: Different modules for separate functions ✓

P_{reuse}: Use the same module for identical functions ✗

Analyzing generalization: SCAN

All networks have the **same**, frozen weights

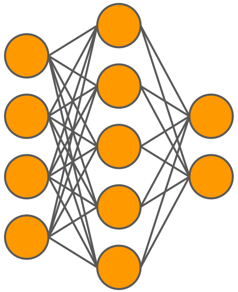


Trained: i.i.d

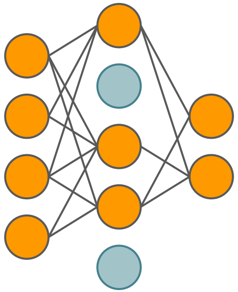
- Train set
- Jump test
- Length test

Analyzing generalization: SCAN

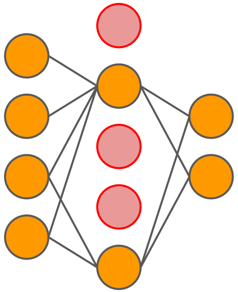
All networks have the **same**, frozen weights



Trained: i.i.d
- Train set ✓
- Jump test ✓
- Length test ✓



Trained: jump train
- Train set ✓
- Jump test ✗

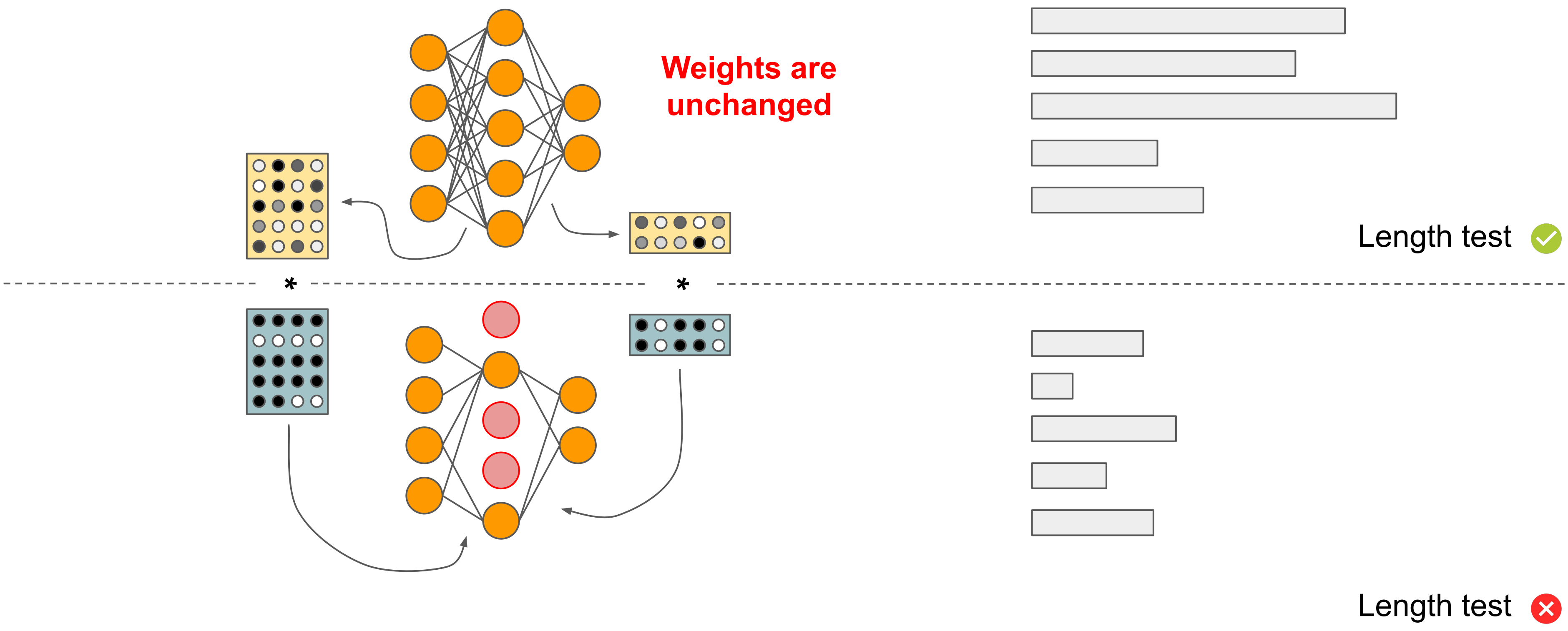


Trained: length train
- Train set ✓
- Length test ✗

Responsible for complex JUMP

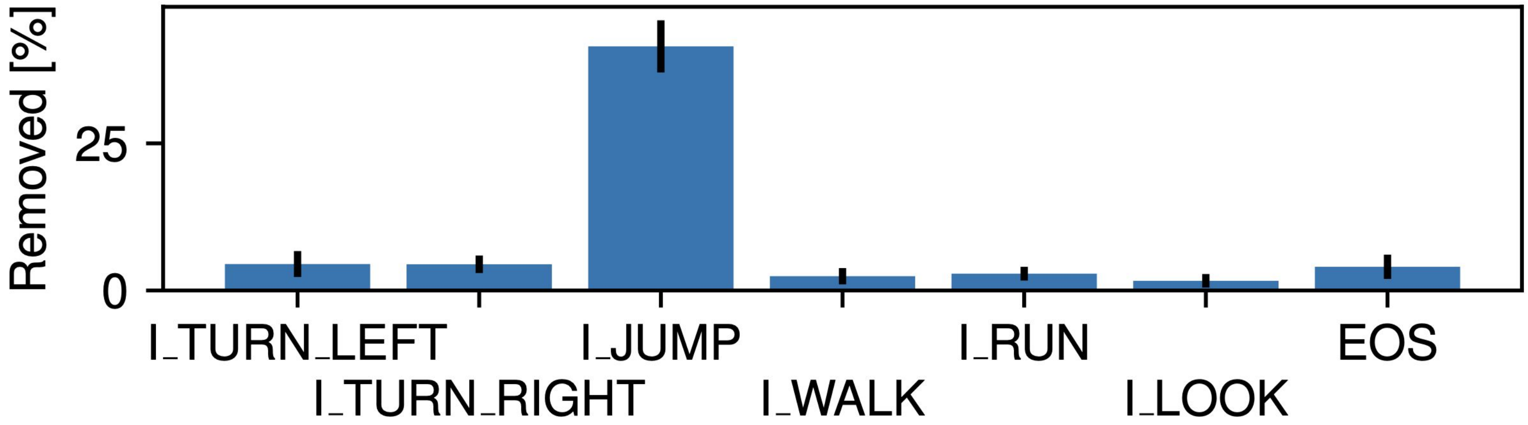
Responsible for longer samples

Analyzing generalization: SCAN



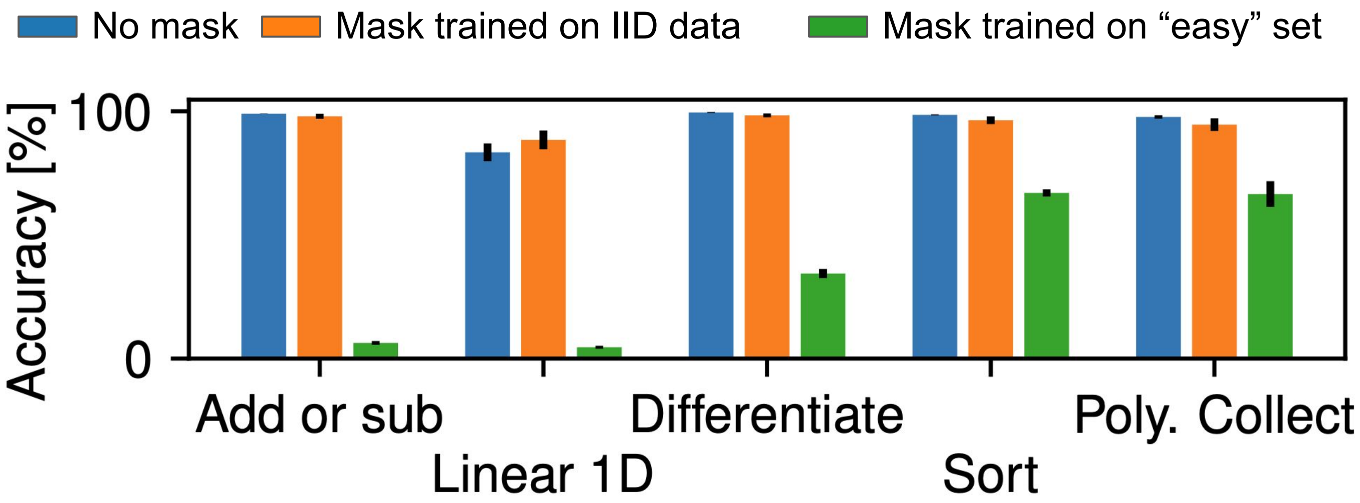
Analyzing generalization: SCAN

Percentage of weights per output token removed from last layer when trained on Add Jump split



Task-specific weights are responsible for solving different splits,
even after successfully trained on the i.i.d data

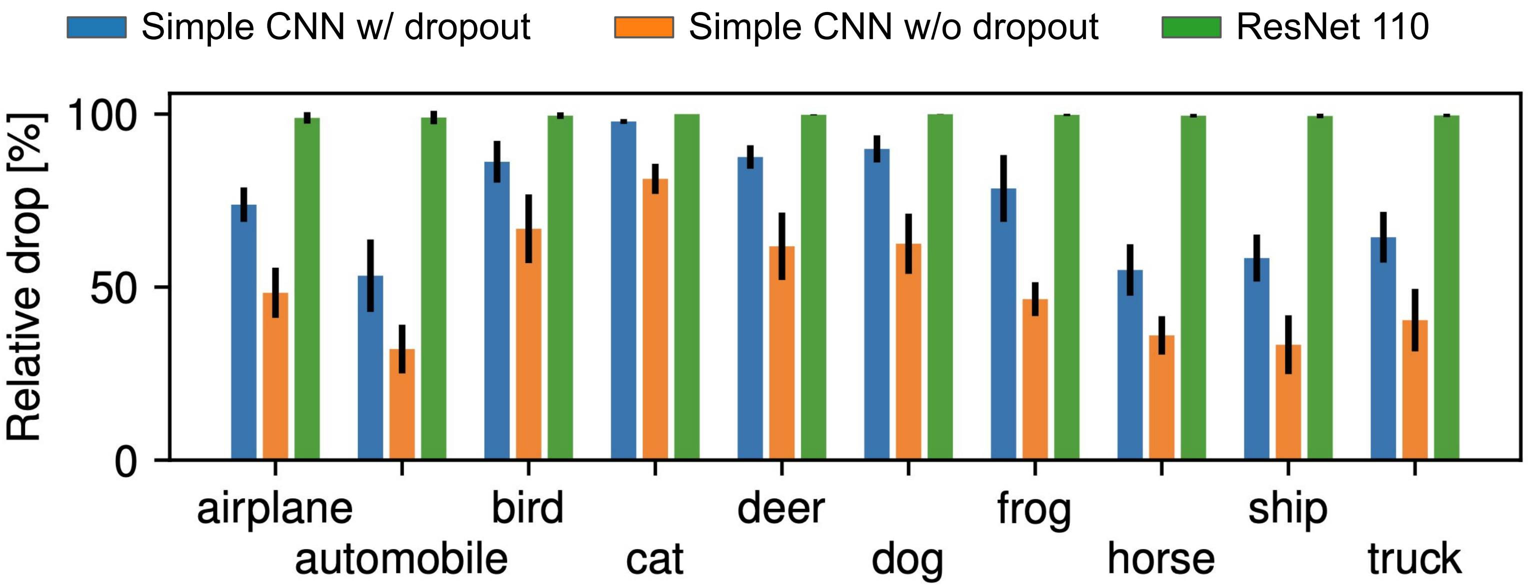
Mathematics Dataset: Similar results



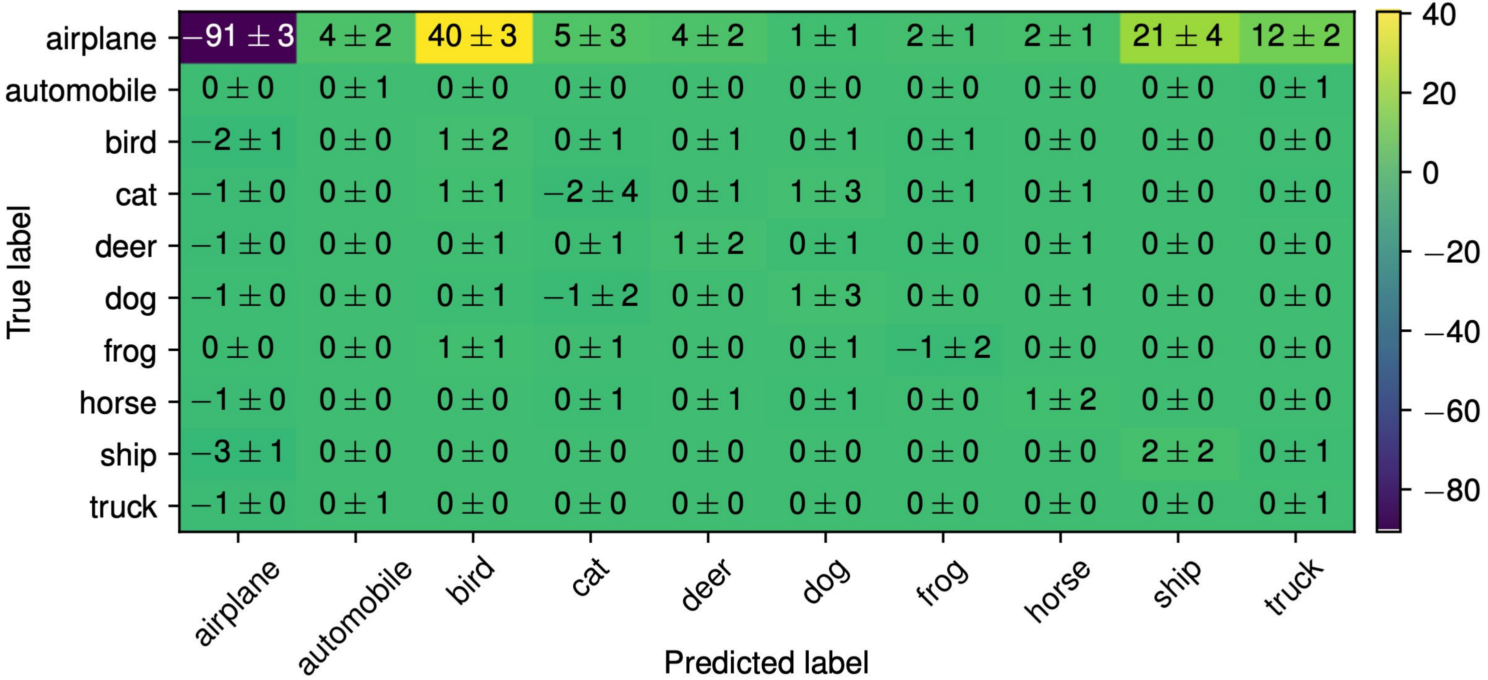
Task-specific weights are responsible for solving different splits,
even after successfully trained on the i.i.d data

Sharing in CNNs (CIFAR 10)

Classification depends heavily on unshared features, which, when removed, cause a huge drop in performace.

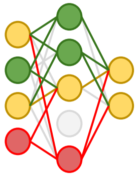


Sharing in CNNs (CIFAR 10)

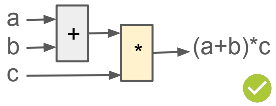


Concluding remarks

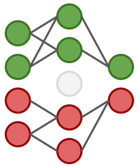
- We proposed a masking based method for discovering subnetworks responsible for specific functions



- Analyze properties of discovered modules



- We found that modules tend to resist sharing



- Generalization issues on SCAN and Mathematics Dataset is a result of learning a non-universal, pattern recognition-like solution.