Are Neural Nets Modular? Inspecting Functional Modularity Through Differentiable Weight Masks

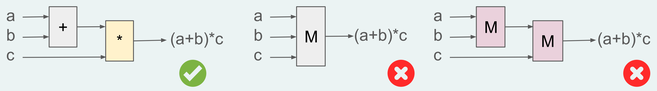
Róbert Csordás robert@idsia.ch
Sjoerd van Steenkiste sjoerd@idsia.ch
Jürgen Schmidhuber juergen@idsia.ch

IDSIA — Scuola universitaria professionale della Svizzera italiana — SUPSI — Università della Svizzera italiana

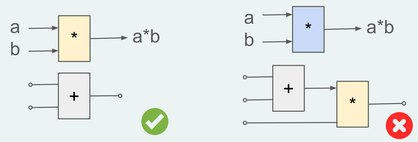
Motivation

- Modularity provides a natural way of achieving compositionality, which appears essential for systematic generalization.
- We investigate if FNNs, LSTMs, Transformers and CNNs are modular, focusing on two desirable properties:

1 Use different modules for separate functions ($P_{specialize}$)



2 Use the same module for identical functions (P_{reuse})



Re-learning the module multiple times makes the model more data-hungry. It also loses the ability to generalize from different data distribution.

Method

We identify weights responsible for specific functions as follows:

1 Pre-train the network on the full task and freeze its weights.

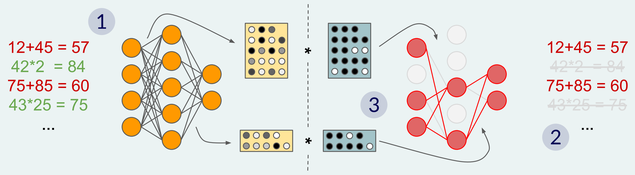
2 Create a new dataset focusing on the target functionality. It is usually a subset of the original task.

3 Train regularized binary mask for each weight. The weights not needed for the task will be masked out.

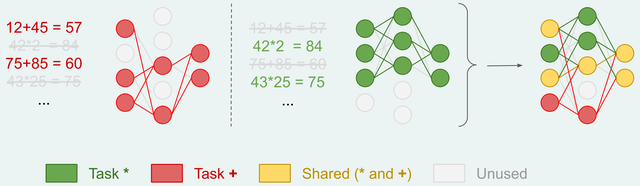
The differentiable binary masks are learned using Gumbel-Sigmoid:

$$s_i = \sigma\left(\left(l_i - \log\left(\log U_1 / \log U_2\right)\right)/\tau\right) \quad \text{with} \quad U_1, U_2 \sim \mathrm{U}(0,1)$$

$$w_i' = w_i * b_i,\ b_i = \left[\mathbb{1}_{s_i > 0.5} - s_i\right]_{\text{stop}} + s_i$$



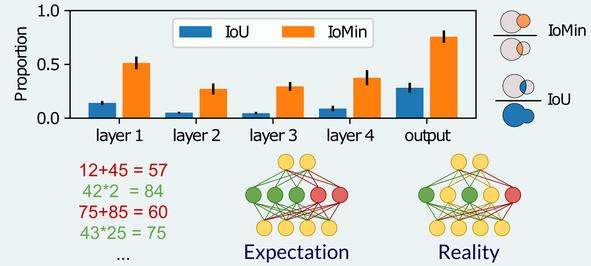
4 Repeat the process for all functionalities of interest. This assigns each weight to some functionality.



NNs specialize, but don't reuse

- **$P_{specialize}$: Use different modules for separate functions**

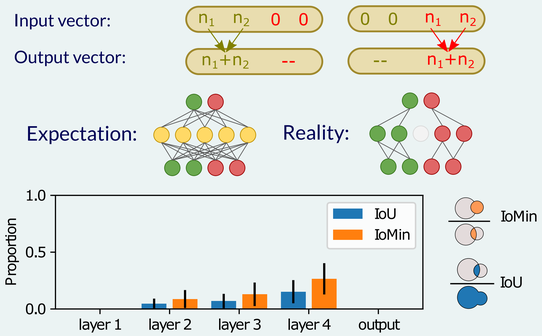
Dataset consisting of two very different functions: addition and multiplication. The inputs are shared between the functions.



There is some separation, but it is not perfect

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

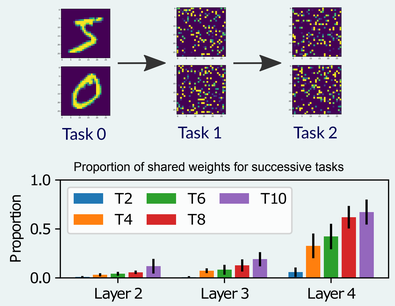
Dataset consisting of the same operation twice. The inputs are not shared.



There is no reuse

Case studies

- **Continual learning on Permuted MNIST**



- FNN learns the permutations sequentially.
- Weights frozen before learning a new permutation.
- First layer always reinitialized.
- Expectation: re-learn the first layer only, reuse the rest.

No weight sharing up until there are enough free weights

- **NNs on Algorithmic tasks learn composition specific weights and do not generalize systematically**

Which of the following hypothesis explains the generalization issues on SCAN?

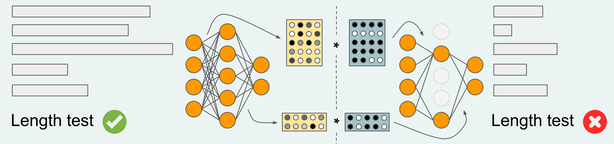
a The training set is not enough to find analogies between working and non-working primitives, but the learned algorithm is otherwise general.

b The NN has not learned a general algorithm.

How to find it out?

1 Pre-train the network on all data. The resulting netwok performs well on all splits. This rules out a as the only issue.

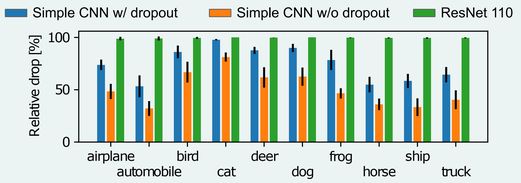
2 Train the masks on the specific data splits. We found split-specific weights, meaning the learned algorithm is not general.



This occurs on SCAN & DeepMind Mathematics with both LSTM and Transformers

The learned solution is inherently non-compositionaly

- **CNNs on CIFAR 10 rely heavily on class-exclusive weights**



Removing such weights hurts performance

Analyzing the changes in the confusion matrix reveals class similarities

Conclusion

- NNs exhibit some level of modularity: different weights become responsible for different functions.
- NNs resists weight sharing even when the underlying functionality is shared.
- NNs learn composition-specific weights: thus the learned solution is not general.