Augmenting Classic Algorithms with Neural Components for Strong Generalisation on Ambiguous and High-Dimensional Data

Imanol Schlag, Jürgen Schmidhuber
The Swiss AI Lab IDSIA, USI & SUPSI
Lugano, Switzerland
{imanol, juergen}@idsia.ch

Abstract

We augment classic algorithms with learned components to adapt them to domains currently dominated by deep learning models. Two traditional sorting algorithms with learnable neural building blocks are applied to visual data with apriori unknown symbols and rules. The models are quickly and reliably trained end-to-end in a supervised setting. Our models learn *symbol representations* and generalise better than generic neural network models to longer input sequences.

1 Introduction

A physical symbol system, or formal system, takes rigid patterns called symbols, organises them into data structures, and transforms them through processes [Newell and Rosenbloom, 1981]. Algorithms are typically descriptions of useful symbol manipulation sequences. They can be implemented as computer programs and applied to any input satisfying certain structural conditions (e.g., a sequence of integer numbers). A *good* algorithm can be applied successfully to inputs that would be considered out of distribution from a statistical perspective (e.g., much longer input sequences) which is the main advantage of algorithms.

However, on real world problems such as speech recognition, image classification, and natural language processing, classic symbol-processing algorithms have had limited success. Furthermore, the search for *good* algorithms given a set of examples, is notoriously hard to automatize, and usually left to highly trained humans.

Another way of representing programs are connectionist models. They are "non-symbolic" and rarely programmed by humans (though some are, e.g., Smolensky [1990]). Instead, a good parametrisation is typically found through first-order optimisation, i.e., scalable greedy search based on a large number of input-output examples. In recent years, connectionist models based on gradient descent have had remarkable success in many domains where classic programs struggle. Nevertheless, in stark contrast to classic algorithms, many deep learning methods tend to underperform when applied to novel settings, i.e., problem instances which would be considered out of distribution [Lake and Baroni, 2018]. Already in 1988, connectionist models were criticised for their lack of systematicity and productivity [Fodor and Pylyshyn, 1988], arguments that are still relevant to this day [Hupkes et al., 2020].

Here, we propose a hybrid approach that augments a problem-specific symbolic program with learned neural components that are trained by gradient descent in a supervised learning setting. This may facilitate program search because certain program parts will be learned directly from examples. However, in this preliminary work, we focus on learning to sort sequences of MNIST images without any apriori knowledge of the digits in the images nor their relation to each other. The neural components are part of a differentiable symbolic algorithm which allows end-to-end training in a supervised setting. Our experiments demonstrate that our hybrid programs can be trained efficiently in just a few thousand steps on short examples (6 digits) and that they generalise better to longer input sequences than generic neural networks.

2 Method

We augment two classic sorting algorithms with neural components: the odd-even transposition sort (also known as parallel bubble sort) and insertion sort. Both algorithms take in a sequence of L MNIST images $\boldsymbol{x} = [\boldsymbol{x}_0, ..., \boldsymbol{x}_{L-1}]$ and predict the ordered sequence of L MNIST classes $\boldsymbol{y} = [\boldsymbol{y}_0, ..., \boldsymbol{y}_{L-1}]$. In the odd-even transposition sort, the model repeatedly processes the sequences as a whole analogous to Transformer models. In insertion sort, the model processes the images sequentially – similar to how recurrent neural networks process a sequence. Our models use two neural components which are always trained from a random initialisation.

The first component is the *symbol extractor* f which is a learned non-linear map of the input image to a *symbol representation* $\boldsymbol{z}_i = f(\boldsymbol{x}_i), \boldsymbol{z}_i \in \mathbb{R}^d$. A symbol representation is a distributed neural representation that captures the information of the input necessary for the successful execution of the succeeding program. In both models, the symbol extractor is a randomly initialised neural network with two convolutional layers followed by two fully connected layers.

The second component models the binary branching *rules* $g : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^2$. It is another learned non-linear map from two symbol representations, here denoted with subscripts p and q where $p, q \in [0, L-1]$, to the weight $\boldsymbol{a} \in \mathbb{R}^2_{\geq 0}$ of branching one way or the other. Thus,

$$\boldsymbol{a} = g(\boldsymbol{z}_p, \boldsymbol{z}_q) \tag{1}$$

$$\boldsymbol{a}_i = \frac{\exp(\boldsymbol{l}_i/\tau)}{\exp(\boldsymbol{l}_0/\tau) + \exp(\boldsymbol{l}_1/\tau)} \tag{2}$$

$$\boldsymbol{l} = \text{MLP}([\boldsymbol{z}_p; \boldsymbol{z}_q]), \tag{3}$$

where $[\boldsymbol{z}_p; \boldsymbol{z}_q]$ is the concatenation of two vectors, MLP is a non-linear map such as the multi-linear perceptron or a multi-linear map, and τ is a softmax temperature which we always set to 1 during training. g is a learned binary relation between two symbol representations which can be used to create weighted combinations of two possible program continuation p_α and p_β. To do so, both programs p_α and p_β are computed in parallel and their results combined weighted by g before being added.

In the sorting algorithms that we consider, $p_{i,j} \in \{0, 1\}^{L \times L}$ is a permutation matrix which either swaps $\boldsymbol{z}_j$ and $\boldsymbol{z}_i$ or inserts $\boldsymbol{z}_j$ in front of $\boldsymbol{z}_i$. They are generated from the symbolic context of the program (e.g. such as running indices i and j). When a branch occurs (e.g. to swap or not to swap) g is evaluated both programs (swapping and not swapping) are executed in parallel followed by a convex combination of the results weighted by $\boldsymbol{a}$. Finally, norm$(\boldsymbol{Z})$ scales each $\boldsymbol{z}$ in $\boldsymbol{Z}$ such that $||\boldsymbol{z}||_2 = \sqrt{d}$ (also known as the RMS normalisation [Zhang and Sennrich, 2019]) to prevent vanishing or exploding activations.

```
function BRANCH($\boldsymbol{z}_\alpha, \boldsymbol{z}_\beta, p_\alpha, p_\beta$)
    $\boldsymbol{Z} = [\boldsymbol{z}_0, ..., \boldsymbol{z}_{L-1}]$                ▷ $\boldsymbol{Z} \in \mathbb{R}^{d \times L}$
    $\boldsymbol{a} = g(\boldsymbol{z}_\alpha, \boldsymbol{z}_\beta)$                   ▷ $\boldsymbol{a} \in \mathbb{R}^2_{\geq 0}$
    return $\boldsymbol{a}_0 p_\alpha(\boldsymbol{Z}) + \boldsymbol{a}_1 p_\beta(\boldsymbol{Z})$
end function
```

2.1 Neural Odd-Even Transposition Sort (NOETS)

The odd-even transposition sort is a parallel sorting algorithm related to bubble sort. It compares adjacent numbers in the input sequence and swaps them if the first is greater than the second. It has two phases: in the odd phase, every odd-indexed element is compared with the next element; in the even phase, every even-indexed element is compared with the next element. Because of the parallel processing of all pairs in a sequence, the time complexity is $O(L)$. Our neural odd-even transposition sort is analogous but with neural components, see Algorithm 1.

Algorithm 1 Neural Odd-Even Transposition Sort

```
1: $l = 0$
2: while $l < L$ do
3:     $k = 0$ if $l \bmod 2 == 0$ else $k = 1$
4:     for $i$ in $\{j \in [0, L-1] | j \bmod 2 == k\}$ do in parallel
5:         $\boldsymbol{Z} = \text{norm}(\text{BRANCH}(\boldsymbol{z}_i, \boldsymbol{z}_{i+1}, p_{i,(i+1)}, p_{\text{identity}}))$
6:     end for
7:     $l = l + 1$
8: end while
```

2.2 Neural Insertion Sort (NIS)

Insertion sort divides the sequence into a sorted and an unsorted part. Initially, the sorted part contains just one element. One by one, elements are sequentially picked from the unsorted part and inserted at the correct position in the sorted part. The list is sorted once the unsorted part is empty. It has a time complexity of $O(L^2)$. Our neural insertion sort is Algorithm 2.

Algorithm 2 Neural Insertion Sort

```
1: $p_{\text{carry}} = 1$
2: $\boldsymbol{Z}_{new} = [0]^{d,L}$
3: for $i$ in $[0, \dots, L-1]$ do
4:     $j = i - 1$
5:     while True do
6:         if $j < 0$ then
7:             $\boldsymbol{Z}_{new} = \boldsymbol{Z}_{\text{new}} + p_{\text{carry}}\text{norm}(p_{i,j}(\boldsymbol{Z}))$
8:             break
9:         end if
10:        $\boldsymbol{a} = g(\boldsymbol{z}_i, \boldsymbol{z}_j)$
11:        $\boldsymbol{Z}_{\text{new}} = \boldsymbol{Z}_{\text{new}} + p_{\text{carry}}\boldsymbol{a}_0\text{norm}(p_{i,j}(\boldsymbol{Z}))$        ▷ Permute with weight $\boldsymbol{a}_0$
12:        $p_{\text{carry}} = p_{\text{carry}}\boldsymbol{a}_1$        ▷ Don't Permute and go to the next element.
13:        $j = j - 1$
14:    end while
15:    $\boldsymbol{Z} = \boldsymbol{Z}_{\text{new}}$
16: end for
```

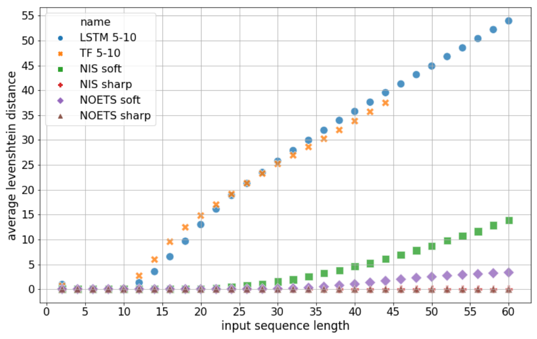
3 Experiments

We train our models in two settings. In the first setting, we train the models on the training data of the MNIST dataset and evaluate on longer sequences: once with images from the training data and once with images from the test data.

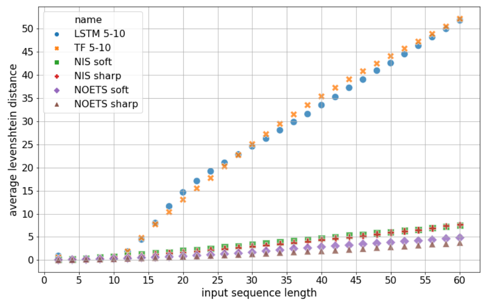
The second setting is an ablation which we refer to as the *symbol-embedding* setting. Here we simplify the problem by providing symbolic inputs to the model instead of MNIST images. This allows us to evaluate the models independently of the noise that is introduced by the MNIST image representation of a digit. For this purpose, the symbol extractor f is replaced with a learnable symbol embedding.

In both settings, we train with a sequence length of 6. To improve performance we also present results where we set the temperature $\tau = 0.01$ during testing. We refer to those results with the suffix *sharp*.

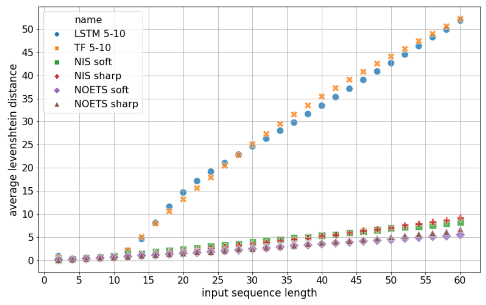
For comparison, we provide a parallel and sequential baseline using general neural networks. Recall that L is the length of the input sequence. The parallel baseline is a Transformer encoder layer where we add positional encodings of size L to the inputs and repeat the same layer L times with shared weights. The sequential baseline is a single layer LSTM model where we first encode the input sequence, followed by L thinking steps, and L decoding steps. Both baselines have a hidden state size of 512. To improve generalisation of our baselines we train both with sequence lengths sampled uniformly from 5 to 10.



(a) MNIST class input (symbol-embedding setting).



(b) MNIST train images.



(c) MNIST test images.

Figure 1: Lenvenshtein distance for longer input sequences. NIS and NOETS are trained with input sequence of length 6. LSTM and Transformer (TF) are trained with sequences ranging from 5-10. Notice that NIS and NOETS generalise perfectly in the symbol setting which has unambiguous inputs and generalise very well to longer sequences with MNIST images as inputs.

All models are trained for 5,000 steps using the Adam optimiser with default parameters (except the Transformer baseline which is trained with a learning rate of 1e-4). All models achieve 100% train accuracy in the symbol-embedding setting and converge in the regular setting.

The results are presented in Figure 1. In our evaluation, we measure the performance of our models using the Levenshtein distance, which measures the minimum number of single digit edits necessary to change the predicted sequence into the target sequence. This is a more accurate than a per-token accuracy measure because missclassifying one ambiguous image of a "9" for a "1" could result in a reordering of large parts of the sequence such that many more tokens are missclassified.

4 Discussion

Numerous models have been proposed to learn “neuro-symbolic programs" from examples. Often such models are neural network architectures trained to mimic symbolic programs by predicting execution traces (e.g. Reed and de Freitas [2016]) or by explicitely generating executable programs (e.g. Mao et al. [2019]). Our work differs from such approaches. Instead, we propose to augment existing algorithms with learnable neural components to improve the generalisation in settings where the algorithm naturally applies (such as ordering sequences in the case of insertion sort). Our results demonstrate perfect generalisation in the absence of input ambiguity despite the apriori unknown set of symbols and rules which parallels the utility of such classic algorithms.

Another approach revolves around the idea of using deep learning methods as an interface between noisy/high-dimensional data and a symbolic program (e.g., Manhaeve et al. [2018] or Pogančić et al.

[2019]). Our method draws inspiration from such approaches but doesn't limit learnable subprograms to only act as interfaces (see, e.g., the *rules* component in Section 2) and it doesn't assume the set of symbols to be known.

5 Conclusion

We presented neural versions of two classic sorting algorithms and applied them to ambiguous and high-dimensional inputs. The neural parts are quickly learned from examples using gradient descent. The neural algorithms strongly generalize way beyond the training distribution of input images and sequence lengths.

Acknowledgments and Disclosure of Funding

We thank the AIPLANS reviewers for valuable comments on the first version of this paper and Aleksandar Stanić for helpful discussions. This research was partially funded by ERC Advanced grant no: 742870, project AlgoRNN. We thank NVIDIA Corporation for donating several DGX machines, and IBM for donating a Minsky machine.

References

A. Newell and P. S. Rosenbloom. Mechanisms of skill acquisition and the law of practice. In J. R. Anderson, editor, *Cognitive Skills and Their Acquisition*, chapter 1, pages 1–51. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1981.

Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2):159–216, 1990.

Brenden Lake and Marco Baroni. Still not systematic after all these years: On the compositional skills of sequence-to-sequence recurrent networks, 2018. URL `https://openreview.net/forum?id=H18WqugAb`.

Jerry A Fodor and Zenon W Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71, 1988.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, pages 757–795, 2020.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/1e8a19426224ca89e83cef47f1e7f53b-Paper.pdf`.

Scott Reed and Nando de Freitas. Neural programmer-interpreters. In *International Conference on Learning Representations (ICLR)*, 2016. URL `http://arxiv.org/pdf/1511.06279v3`.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJgMlhRctm`.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/dc5d637ed5e62c36ecb73b654b05ba2a-Paper.pdf`.

Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2019.