Towards Neural Abstract Logical Reasoning

Victor Kolev

under the direction of

Dr Svetlin Penkov
Sciro Research

Dimitar Vasilev
Microsoft Corporation

Research Science Institute
July 31, 2020

Abstract

The ability to learn abstract concepts and to quickly grasp logical rules are hallmarks of human intelligence, yet these traits are difficult for neural networks. In our work we take a backward design approach and we develop a neural framework based on abstract principles, in order to learn abstract rules. Our model uses a Differentiable Neural Computer in order to capture algorithmic logic and allow true computation. We apply spectral norm regularization, which adds preference to functions with low complexity, which we use as a proxy for abstraction. Model-agnostic Meta-learning is employed to adapt parameters to a given task at test time. We show that our approach outperforms classical baselines on the Abstraction and reasoning corpus, a few-shot pattern manipulation benchmark.

Summary

People have the ability to quickly learn abstract concepts and infer complex logical relationships with just a few examples. Ideally, programs based on Artificial Intelligence should also be capable of abstract reasoning, yet this is currently difficult to achieve with neural networks. The aim of our work is to design a neural framework for abstract reasoning. Instead of thinking about how to make neural networks learn abstract rules, we take a backward design approach – we implement abstractions from computer programming and human learning in neural networks. We show that this approach outperforms existing benchmarks and has potential to bring us closer to abstract logical reasoning in neural networks.

1 Introduction

Deep learning methods are powering the current wave of artificial intelligence research. They rely on *neural networks* – universal function approximators with parameters that are tuned with gradient-based optimization methods. Neural networks[1] are able to learn complex relationships in data, even surpassing humans in some tasks related to object classification [1] and speech recognition [2]. Deep learning methods have also beaten human champions in strategy-based games such as Go [3], Chess [4] and Starcraft [5], which require learning abstract features and strategies.

Many of these achievements, however, rely on extensive, incremental training on a large data set. This often leads to over-specialization (overfitting) and neural networks exhibiting poor performance on related tasks (*domain generalization* [6, 7, 8, 9, 10, 11]). For example, a network trained to play the ATARI game Breakout is not able to generalize to play Pong, despite the similarities between the two – both involve moving a platform to hit a ball.

However, applying acquired knowledge to other domains is seen as one of the hallmarks of human intelligence [12] and grasping concepts from very few examples is a desired trait of intelligence [13]. Let us consider the definition of intelligence for AI in [14], namely

Intelligence measures an agent's ability to achieve goals in a wide range of environments.

In order for neural networks to be considered intelligent by the above definition, it is important that they learn abstract rules, instead of exploiting coincidental relationships in data (i.e. identifying a picture of a cow by the grass background behind it [15]). We define an abstract rule (abstraction) as a principle that is agnostic to the object it is applied to.

A recently proposed benchmark, the Abstraction and Reasoning Corpus [16], addresses this dissonance and measures the performance of machine learning methods on a large set

[1]For the sake of clarity of the argument, we abuse language and refer to decision-making agents with policies, parametrized by a neural network, as neural network agents or simply neural networks.

of abstract logical problems. For instance, consider a task, in which a pattern needs to be singled out for being different from the rest (see example in Figure 2). The decision-making agent (in our case – the neural network) is presented with a small number of examples and has to infer the logical rule from them. The agent is then tested for the correctness rule. Naturally, we are under the assumption that a unique set of rules exist that explains the relationships between task inputs and task outputs.

Currently, the best performance on the Abstraction and Reasoning Corpus is 19% accuracy (it solves 19% of all tasks correctly). This is achieved with *decision trees*, which, unlike neural networks, are not universal function approximators and could not hope to scale to higher-dimensional objects or higher-order logic. In contrast, the best neural network-based solution achieves only 1% accuracy. [2]

The aim of our work is to develop neural network models, capable of abstract reasoning. We look at the challenge backwards – instead of thinking how to train neural networks to reason abstractly, we develop a framework for incorporating abstractions in neural networks. Abstraction performance is measured on the Abstraction and Reasoning Corpus.

Our approach relies on three components:

1. *Memory-augmented neural networks*, namely the Differentiable Neural Computer [17]. They allow us to execute programs with memory complexity, in addition to computational complexity;

2. *Optimization-based meta-learning*, which enables us to adapt rapidly to new environments and tasks using additional inner optimization steps;

3. *Complexity regularization*, which biases the network to learn simpler (as measured by a complexity measure) and smoother functions. The underlying intuition is that the function with smallest complexity would be the most abstract.

[2] Both results are presented in `kaggle.com`

2 Preliminaries

Meta-learning

The goal of *meta-learning* is to learn a mechanism for rapidly adapting to new tasks. A meta-learning problem consists of training and evaluation tasks. Each task has a train set and a test set. A meta-learning model will learn to solve the training tasks and generalize learnt rules to the evaluation set. It can also adapt to the evaluation using the train dataset of each of the tasks.

From a probabilistic perspective, successful meta-learning involves extracting prior information from a set of training tasks, which allows the model to efficiently derive posterior knowledge about a task given only a few examples.

One method for carrying out this adaptation is with additional "inner" optimization steps, such as in *Model-Agnostic Meta-:earning* [18] (Figure 1). Consider a neural network f_θ with parameters θ. It has to minimize a loss objective $\mathcal{L}(f_\theta)$ on a set of tasks $\mathcal{T}$

$$\theta^* = \arg\min_\theta \mathop{\mathbb{E}}_{T\sim\mathcal{T}} \left[\mathcal{L}_T(f_{\theta'})\right]$$

where θ' are parameters derived with k steps of gradient descent from the initial parameters $\theta_0 = \theta$ and step size β

$$\theta_{i+1} = \theta_i - \beta\nabla_{\theta_i}\mathcal{L}_T(f_{\theta_i})$$

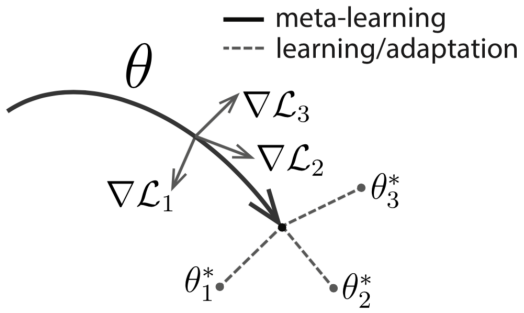

Figure 1: Intuitive illustration of the MAML algorithm. [3]

Another method for meta-learning is using *Memory-augmented neural networks* (MANNs) [19]. These networks implement differentiable external memory, equipped with addressing mechanisms that allow it to be indexable. Additionally, memory capacity is easily scalable, permitting large amounts of information to be encoded instantly. This allows them to function as a structured cache, which can be used for adaptation by retrieving relevant stored information.

While memory-based meta-learning has shown a lot of promise, it has a fundamental limitation – an upper bound of adaptation exists, since memory is finite. In contrast, optimization-based meta-learning has no such bound, as it adapts parameters and it can therefore approximate the task arbitrarily well, as per the universal approximation theorem.

The Abstraction and Reasoning Corpus

The Abstraction and Reasoning Corpus (ARC) [16] is a dataset of grid-based pattern recognition and pattern manipulation tasks (Figure 2). A decision-making agent sees a small number of examples of input and output grids that illustrate the underlying logical relationship between them. It then has to infer this logical rule and apply it correctly on a test query.

In many ways, the benchmark is similar to the Bongard problems (view [20]) – relations are highly abstract and geometric. Moreover, only 3-5 examples are presented for each task, therefore, the benchmark tests the ability of an decision-making agent to (i) grasp abstract logic and (ii) adapt quickly to new tasks.

There are 400 training and 400 evaluation task examples, structured as follows:

- each task consists of a train and a test set;
- the train set includes 3-5 input/output pairs;
- the test set includes 1 input/output pair;

[3]Source: Finn et. al [18]

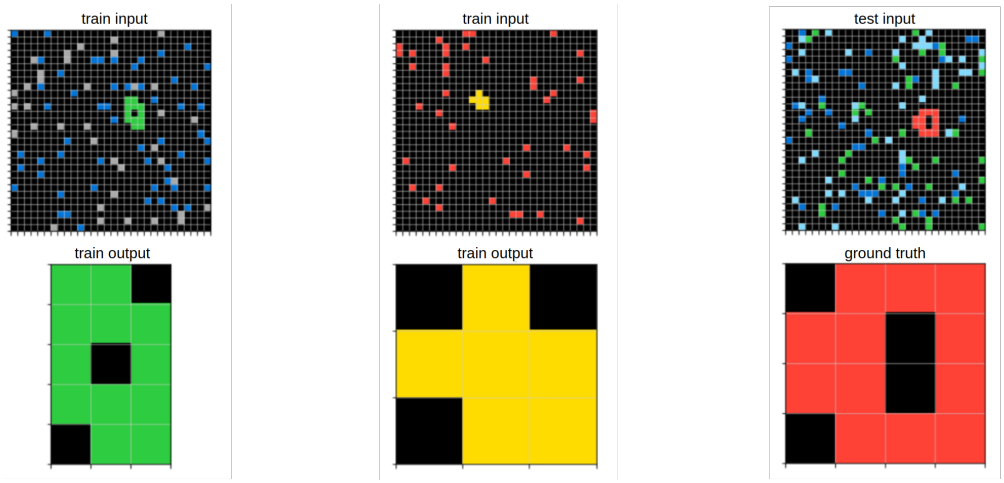

Figure 2: A visualization of a task from the ARC ("crop" task – one of the patterns is different from the others and we want to identify it). Two training examples and one test example are provided. The decision-making agent receives an input and has to predict the output.

- an input/output pair is comprised of an input grid and an output grid, the relation between which follows a consistent logic throughout the task;
- grids are rectangular and are divided into 1×1 squares;
- grid patterns are drawn with 10 colors;
- grid sizes vary between 1 and 30 in length and width; input and output grid sizes are not necessarily equal.

No set of rules exists that can solve all tasks, and while some skills are useful for multiple of them, each task has its own unique principle. This makes trivial approaches like brute-force computation impractical.

If a human was approaching those tasks, they would easily be able to spot logical relations – we have developed the necessary priors to find similarities and infer logic. Therefore, ideally the neural network would derive this prior during training, and that would allow it to generalize well to the evaluation dataset.

For the current scope of our research we use all tasks with grids of size not larger than 10×10.

3 Methods

Grid Embedding

Before we can utilize meta-learning, data should first be pre-processed. We embed the grids in a 256-dimensional vector space for performing the learning using an *autoencoder* – a neural network compresses the input grid and then needs to decompress it. It consists of an encoder $\mathcal{E}_{\theta_e}$ with parameters θ_e, and a decoder $\mathcal{D}_{\theta_d}$ with parameters θ_d. Let $G \in \{0,1\}^{C\times H\times W}$ be a grid, in which colors are represented by one-hot vectors. The autoencoder learns a lossless embedding, therefore it requires no labelled data (unsupervised learning). It outputs a a grid prediction $\widehat{G}$ that is a probability distribution over possible colors. If we consider that pixel $p \in G$ is color c and $p_{c'}$ represents the score of color c', then we want to find optimal parameters θ^* that minimize the negative log-likelihood of color c

$$\theta^* = \arg\min \mathcal{L}(\theta) = \sum_{p\in\widehat{G}} -\log\left(\frac{e^{p_c}}{\sum_{c'} e^{p_{c'}}}\right)$$

No labelled data is required, which this is referred to as unsupervised training. The autoencoder outputs a probability distribution over all of the colors and the goal is to minimize the negative log-likelihood of the correct color.

We utilize an architecture, inspired by InceptionNet [21] – we utilize multiple filter sizes for the convolutional layers, thus capturing features and patterns of different scales. We linearly interpolate the convolutional outputs with weights, obtained from an independent neural network that determines the relevance of each convolution in the context of the input grid and its patterns. This approach bears similarities to a *self-attention mechanism* [22], where a score of relative importance is computed for each attribute in an object.

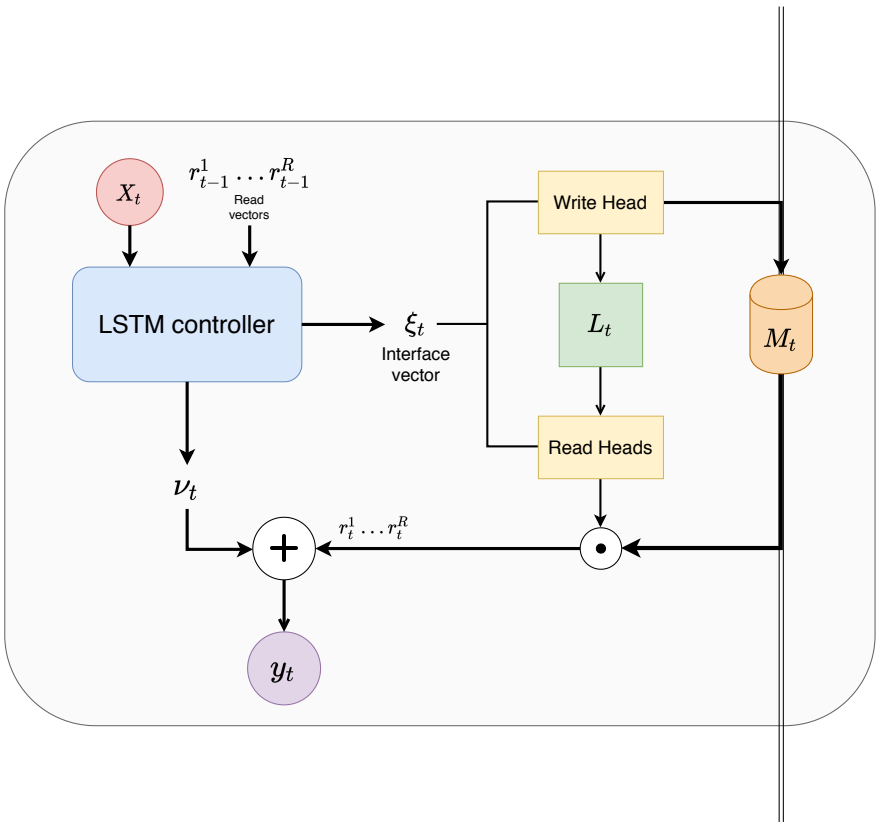

Figure 3: The DNC architecture. For more details, see Appendix B

The Differentiable Neural Computer

With the grids embedded, we need to capture the logical rules that relate an input grid embedding to an output grid. In computer programming, abstraction is achieved through utilizing variables, however that is not feaseable in traditional neural networks due to computation and memory being co-dependent.

This problem can be solved with external differentiable memory. We utilize a well-established memory-augmented neural architecture, the Differentiable Neural Computer [17] (Figure 3). It implements an external memory matrix with structure and functionality similar to a dictionary. This allows us to separate memory operations and computation, and use variables.

The concept of variables is what provides abstraction in algorithms, as they make it

agnostic to the datum. Therefore, a neural network capable of abstract reasoning would need to manipulate variable-like structures. Other architectures, such as *Recurrent Neural Networks* (neural networks that retain their hidden state and use it for the next input), do not natively permit this. Since they lack mechanisms for indexing, they use the whole hidden state, which includes information that is not directly relevant to a given task. Additionally, an RNN's "working memory" (its hidden state) is tied to the number of parameters, hence there is a practical computational bound to the information it can store at once.

Spectral Norm Regularization

While the internal structure of the Differentiable Neural Computer enables it to theoretically learn more abstract computation, it also makes it more unstable. Hence, it needs to be regularized to learn abstract relations. We employ *spectral norm regularization* [23, 24], which places a penalty on the spectral norms of the weight matrices of the networks.

The spectral norm of a matrix is defined as

$$\sigma(W) = \max_{\varepsilon} \frac{||W\varepsilon||_2}{||\varepsilon||_2}$$

which corresponds to the largest singular value of the matrix W. Spectral norm regularization can have two interpretations – on one hand, it is reducing sensitivity towards *random perturbations* (noise). On the other hand, it penalizes large singular values, thus constraining the Lipschitz constant of the function, thereby adding bias towards "more linear" functions. Intuitively, this limits how fast the value of a function can change.

Referring to computer programming again, let us analyze the shortest possible program (in an arbitrary language) that solves a given task. If more lines were added, they would either decrease efficiency (undesired) or impede abstraction (by over-specialization to a subset of tasks). The neural networks should therefore approximate the algorithm with lowest *Kolmogorov complexity* (program length). Spectral norm regularization adds bias in favor of functions with lower Lipschitz constants and lower singular values, which is closely re-

lated to Kolmogorov complexity. Hence, spectral regularization is theoretically grounded in aiding networks to achieve greater abstraction, while the same could not be said for other regularization techniques such as $L_2(W) = ||W||_2 = \sqrt{\sum_i \sum_j w_{i,j}^2}$.

Computational meta-learning algorithm

As we already utilize a DNC, the MANN for meta-learning approach from [19] (memory-based meta-learning) is natural. However, the DNC is already using memory for storing variables, hence we face an information throughput bottleneck if task adaptation is completely memory-reliant.

To alleviate this, we perform additional optimization steps with the Model-agnostic meta-learning (MAML) algorithm [18]. In this manner, the memory operations and the computational process are tuned to the task. More details can be found in Algorithm 1.

Algorithm 1: Computational Meta-Learning

$p(T)$ distribution over tasks
Hyperparameters: α, β step sizes; k number of steps
Randomly initialize θ
while *not done* **do**
  Sample task $T_i \sim p(\mathcal{T})$
  $T_i = \{\mathcal{D}^i, \mathcal{D}^i_{test}\}$
  Form input data for DNC $X^t = (X^D_t, y^D_{t-1})$
  $\theta_0 \leftarrow \theta$
  for *j in 1:k* **do**
    for *t in 1:*$|\mathcal{D}^i|$ **do**
      $\hat{y}_t = \mathcal{N}_{\theta_{j-1}}\Big(X_t | X_{t-1}, \ldots, X_1\Big)$
    end
    Evaluate $\nabla_{\theta_{j-1}} \mathcal{L}_{\mathcal{D}^i}(\hat{y}_t, y_t)$
    Adjust parameters $\theta_j \leftarrow \theta_{j-1} - \alpha \nabla_{\theta_{j-1}} \mathcal{L}_{\mathcal{D}^i}$
  end
  Compute $\hat{y}_{test} = \mathcal{N}_{\theta_k}\Big((X_{test}, y_t) | X_t, \ldots, X_1\Big)$
  Update $\theta \leftarrow \theta - \beta \nabla_\theta \mathcal{L}(\hat{y}_{test}, y_{test})$
end

Net	Training Dataset		Evaluation Dataset	
	Pixel	Grid	Pixel	Grid
RNN-WD	81.12% (±0.13)	0.86% (±0.02)	78.28% (±12.45)	0.88% (±2.19)
RNN-Sp	92.64% (±0.05)	6.53% (±0.31)	90.51% (±8.24)	5.96% (±8.38)
Att-WD	86.25% (±0.08)	1.31% (±0.05)	83.03% (±10.42)	1.31% (±5.95)
Att-Sp	99.93% (±0.02)	96.29% (±0.82)	99.12% (±1.76)	79.14 % (±23.78)

Table 1: Performance of grid embedding networks. It is evaluated based on pixel accuracy (what part of the pixels are predicted correctly) and grid accuracy (the percentage of all grids that are reconstructed perfectly; one wrong pixel labels the grid incorrect). Numbers in parenthesis indicate one standard deviation, as measured on 10 runs.

4 Results

We train embedding separately of meta-learning, and so we present results independently.

Autoencoding performance

Two architectures – *Att* and *RNN* – are tested to embed the grids, both utilizing a basket of convolutional layers with different filter sizes. *Att* uses an "attention-like" weighted sum to combine the convolution outputs (as presented in Methods), while *RNN* processes them sequentially with a Long-Short Term Memory network [25]. Dropout [26] is used in both variants and they are tested with standard weight decay regularization (L_2 matrix norm) [27] or spectral norm regularization [23]. Results are presented in Table 1.

The best performance is achieved with *Att-Sp*, which we attribute to spectral norm regularization biasing the network towards an optimum with low Kolmogorov complexity and high abstraction, as well as the commutativity of the attention mechanism. Additionally, we test *Att-Sp* with tanh and ReLU activation functions. The latter is less robust to Gaussian noise, presumably because ReLU is unbounded, while tanh is asymptotically bounded, hence small perturbations in tanh would not have any effect when the function argument approaches $\pm\infty$.

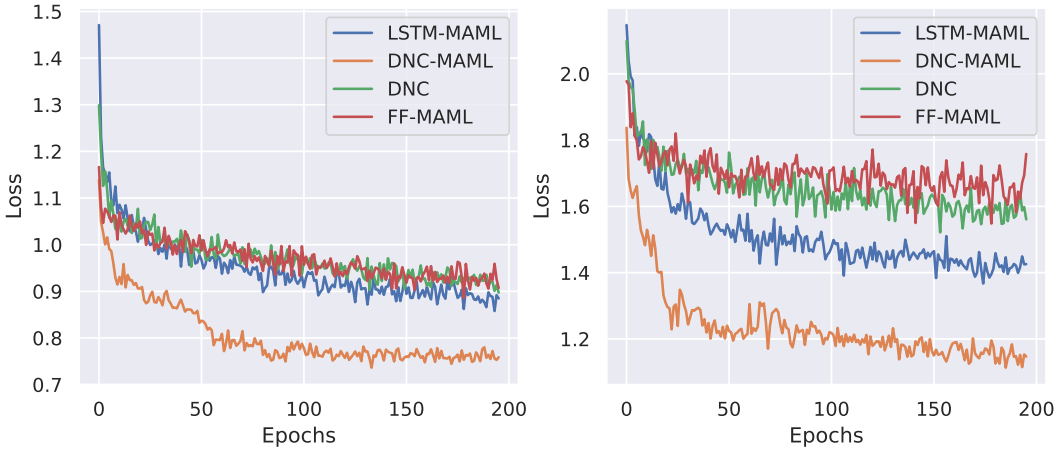

Figure 4: Training curves for each of the networks. On the left they are trained only on the "crop" task (single skill), and on the right – on the whole ARC dataset.

It is noteworthy that performance of *Att-Sp* degrades on the test set, meaning that it has overfitted to the train set. The classic interpretation of this is "bad" training, however in this case it is possible that the autoencoder has learned to recognize the patterns in the training dataset and compresses them in a structure similar to a hash table.

Computational meta-learning experiments

We compare our approach (*DNC-MAML*) with the following baselines:

- MAML with a feed-forward network (*FF-MAML*)
- memory-based meta-learning with an LSTM and MAML steps (*LSTM-MAML*)
- memory-based meta-learning with a DNC, no MAML optimization steps (*MANN*)

All models are evaluated on a single skill ("crop" task, Figure 2) and on the whole dataset. Results are presented in Figure 4. Performance is measured by the achieved mean squared error loss.

The worst performer is FF-MAML, which is in line with our hypothesis that memory is required to infer the task solution. Without MAML, the DNC also achieves worse performance, confirming our postulations that memory throughput is a bottleneck. Whether

this limitation can be alleviated with multiple write heads or a substantially larger memory matrix is a subject of further study.

While the DNC-MAML model is the best-performing, it still achieves less than 1% grid accuracy, which is comparable to other parametric approaches to the ARC dataset, but more work is needed to stabilize the networks.

5 Related Works

Regarding abstract problems, there has been work on training networks on multiple choice questions for pattern expansion. In [28] they embed patterns with an RNN to determine which of 4 possible answers has the highest probability of correctness. We experimented with such an approach but found that an attention-like mechanism yields better results. Another approach in literature is Neural Module Networks [29]: different modules are trained independently and to solve a task, a composition of them is derived with meta-learning. This argument is expanded in [30] with Graph Neural Networks [31, 32] to provide a further level of abstraction by deriving the graph that corresponds to the optimal function composition. While such a modular approach is promising, the large number of skills required for the ARC makes it impractical.

Domain generalization is also related to our work, since each task is effectively a new domain. In [33] meta-learning is used to model a regularization function, which is key for generalization. In contrast, we use a fixed regularizer, as it is theoretically grounded and related to program complexity (Kolmogorov complexity).

The DNC has been exploited for meta-learning in [19], using memory to adapt to the current task, without modifying parameters. We found that this proved insufficient for the task, so we instead rely on the DNC for implementing abstract computational logic and update parameters accordingly.

6 Conclusion and Future Development

We developed a model with built-in abstractions for computation and showed that it outperforms existing models on the Abstraction and Reasoning Corpus dataset. We combined two meta-learning approaches, memory-augmented neural networks, and theoretically grounded complexity regularization, and we showcased that our framework has potential for algorithmic and abstract learning. Possible practical implications include general-purpose robots and general intelligence.

Subject of further work could be to combine our approach with Neural Module Networks [29], in order to embed multiple priors relevant to logical reasoning, in a sense developing basis skills, from which to derive all other tasks. Expanding the DNC with multiple write heads and larger memory are also worthy of additional analysis. Challenges in front of that include that addressing mechanisms become exponentially more unstable as memory size grows. Possible solutions are to use sparse memory, pre-training read and write mechanisms, and utilizing hard indexing (non-differentiable, therefore gradient-free mechanisms have to be used). Lastly, a more mathematically-rigorous embedding module, such as with Symmetry-Based Disentangled Representation Learning [34], could provide a better latent representations, which are key to successful meta-learning.

7 Acknowledgments

I would like to thank Svetlin Penkov for all of the conversations we have had – they have been insightful, inspiring, and incredibly interesting. It still amazes me how there is no meeting, after which my imagination is not replete with ideas, which I cannot wait to experiment with. Thank you for all the intellectual challenges, the advice, and the constant support.

I thank Dimitar Vasilev for his support in facilitating the experiments.

Many thanks to MIT, CEE, all of their sponsors and Dr. Amy Sillman for facilitating this wonderful experience. A big thank you to the High School Student Institute of Mathematics and Informatics and its founders for giving me the opportunity to be part of RSI.

I would like to express utmost gratitude to Rafael Rafailov, who was the one to introduce me to research and has continuously supported me ever since. I certainly would not be where I am now if it was not for him, and I thank him for opening my eyes to a whole new world.

I also give special thanks Jenny The Greatest Sendova, whose energy levels at time t are $f(t)$ and $\min_t f(t) = +\infty$, and are incredibly contagious. You are an inspiration and I am honoured to know you.

A big, big thank you to the whole Rickoid family, with special attention to the Bash2DFuture counselor group. You guys are the best.

A special mention to all the people who have rigorously gone through this paper and provided their feedback, including Peter Gaydarov, Dimitar Chakarov, Stanislav Atanasov, Veselin Kulev, Gary Lee, Stella Pantela, Jenny, Svetlin, and Rafael.

There are too many people to name individually that have supported me on my journey, so here is one big thank you to you. You know who you are.

Last but by no means least, would also like to thank my parents and my family for the love and support. I owe it all to you.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[2] D. Yu and L. Deng. Automatic speech recognition a deep learning approach.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[5] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[6] J. Farebrother, M. C. Machado, and M. Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

[7] A. Zhang, N. Ballas, and J. Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018.

[8] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.

[9] S. Whiteson, B. Tanner, M. E. Taylor, and P. Stone. Protecting against evaluation overfitting in empirical reinforcement learning. In *2011 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*, pages 120–127. IEEE, 2011.

[10] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[11] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[12] N. Jankowski, W. Duch, and K. Grabczewski. *Meta-learning in computational intelligence*, volume 358. Springer, 2011.

[13] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[14] S. Legg, M. Hutter, et al. A collection of definitions of intelligence. *Frontiers in Artificial Intelligence and applications*, 157:17, 2007.

[15] R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann. Shortcut learning in deep neural networks. *arXiv preprint arXiv:2004.07780*, 2020.

[16] F. Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

[17] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[18] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[19] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850, 2016.

[20] A. Linhares. A glimpse at the metaphysics of bongard problems. *Artificial Intelligence*, 121(1-2):251–270, 2000.

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[23] Y. Yoshida and T. Miyato. Spectral norm regularization for improving the generalizability of deep learning. *arXiv preprint arXiv:1705.10941*, 2017.

[24] A. Bietti, G. Mialon, D. Chen, and J. Mairal. A kernel perspective for regularizing deep neural networks. In *International Conference on Machine Learning*, pages 664–674, 2019.

[25] J. Schmidhuber and S. Hochreiter. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[27] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[28] F. Hill, A. Santoro, D. G. Barrett, A. S. Morcos, and T. Lillicrap. Learning to make analogies by contrasting abstract relational structure. *arXiv preprint arXiv:1902.00120*, 2019.

[29] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48, 2016.

[30] F. Alet, M. Bauza, A. Rodriguez, T. Lozano-Perez, and L. P. Kaelbling. Modular meta-learning in abstract graph networks for combinatorial generalization. *arXiv preprint arXiv:1812.07768*, 2018.

[31] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[32] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[33] Y. Balaji, S. Sankaranarayanan, and R. Chellappa. Metareg: Towards domain generalization using meta-regularization. In *Advances in Neural Information Processing Systems*, pages 998–1008, 2018.

[34] I. Higgins, D. Amos, D. Pfau, S. Racaniere, L. Matthey, D. Rezende, and A. Lerchner. Towards a definition of disentangled representations. *arXiv preprint arXiv:1812.02230*, 2018.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[36] T. Weyand, I. Kostrikov, and J. Philbin. Planet-photo geolocation with convolutional neural networks. In *European Conference on Computer Vision*, pages 37–55. Springer, 2016.

[37] H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449, 1992.

[38] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[39] C. Packer, K. Gao, J. Kos, P. Krähenbühl, V. Koltun, and D. Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

[40] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales. Learning to generalize: Meta-learning for domain generalization. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[41] S. Chen and Z. Niu. The research about recurrent model-agnostic meta learning. *Optical Memory and Neural Networks*, 29:56–67, 2020.

[42] F. Alet, T. Lozano-Pérez, and L. P. Kaelbling. Modular meta-learning. *arXiv preprint arXiv:1806.10166*, 2018.

[43] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[44] J. Franke, J. Niehues, and A. Waibel. Robust and scalable differentiable neural computer for question answering. *arXiv preprint arXiv:1807.02658*, 2018.

[45] J. Jacobs, C. T. Weidemann, J. F. Miller, A. Solway, J. F. Burke, X.-X. Wei, N. Suthana, M. R. Sperling, A. D. Sharan, I. Fried, et al. Direct recordings of grid-like neuronal activity in human spatial navigation. *Nature neuroscience*, 16(9):1188–1190, 2013.

[46] A. Banino, C. Barry, B. Uria, C. Blundell, T. Lillicrap, P. Mirowski, A. Pritzel, M. J. Chadwick, T. Degris, J. Modayil, et al. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705):429–433, 2018.

A Grid Embedding

Prior to embedding, we zero-pad all grids to be 10×10, with the original grid in the center of the image. Additionally, colors in the grids are represented as one-hot vectors, making the final dimensionality of the grids $10 \times 10 \times 10$ (10 colors).

The embedding is done with a convolutional neural network, comprised of an encoder and a decoder. The encoder consists of a basket of 10 convolutions of filter sizes equal to $1, \ldots 10$ (the $\mathcal{C}$ module). The convolution outputs are flattened and passed to linear layers with hyperbolic tangent activation functions (used to normalize the embedded grid space by constraining it to $[-1, 1]$), which transform them into the desired dimensionality ($n = 256$). A second neural network $\mathcal{R}$ computes weights for summing the 10 resulting vectors. The decoder shares the same architecture with the encoder, but in reverse order – first linear layers, after that convolutions and then a weighted sum; finally a softmax over the color dimension.

Summing the convolutional outputs enables the embedding network to be agnostic to the order in which it receives them (as would be with an RNN for instance). The weighs reflect the fact that grid sizes vary and therefore not all filter sizes would be equally applicable or useful. In a sense, the weighted summing can be regarded as a self-attention mechanism [22].

B Differentiable Neural Computer

The Differentiable Neural Computer (DNC, Figure 3) implements an external memory matrix in a traditional RNN model, whilst preserving full end-to-end differentiablility. The memory has the following features:

- content-based lookup - the DNC writes and reads information based on a search key from the controller

- dynamic memory allocation - information is written based on the free space in each memory location (row of the matrix)
- memory deallocation - information can be deleted from memory after it is read
- multiple read heads - information is read by multiple read heads, each with a different search key

The content-based lookup mechanism makes the DNC resemble a dictionary data structure, since it uses soft attention to choose the memory location that is closest to the search key (measured by the cosine similarity). Hence, the attention vector is effectively an index.

Here we elaborate in greater detail about each of the mechanisms of the DNC.

At each timestep, the LSTM controller receives an input X_t and read vectors. The controller then emits an output vector and an interface vector. The latter determines what new information is written in the memory matrix and what is read for the next timestep. After the new data is read, it is processed with the output of the controller to from the final DNC output y_t.

Interfacing with the memory

The aforementioned interface vector ξ_t carries the necessary instructions for the read and write heads. It is split as follows:

$$\xi_t = [\mathbf{k}_t^{r,1}, \ldots, \mathbf{k}_t^{r,R}, \hat{\beta}_t^{r,1}, \ldots, \hat{\beta}_t^{r,R}, \hat{\mathbf{m}}_t^{r,1}, \ldots, \hat{\mathbf{m}}_t^{r,R}, \hat{\mu}_t^1, \ldots, \hat{\mu}_t^R, s_t^{\phi,1}, \ldots, s_t^{\phi,R}, s_t^{b,1}, \ldots, s_t^{b,R},$$
$$\mathbf{k}_t^w, \hat{\beta}_t^w, \hat{\mathbf{m}}_t^w, \hat{e}_t, v_t, \hat{f}_t^1, \ldots, \hat{f}_t^R, \hat{g}_t^a, \hat{g}_t^w]$$

where

- $\mathbf{k}_t^{r,i} \in \mathbb{R}^W$ is the lookup key for read head i; lookup keys act as search terms and are used in the content-based lookup mechanism

- $\beta_t^{r,i} = oneplus(\hat{\beta}_t^{r,i}) \in \mathbb{R}^4$ is the key strength of read head i; keys are assumed to have different impact and this is reflected in their key strength
- $\mathbf{m}_t^{r,R} = \sigma(\hat{\mathbf{m}}_t^{r,R}) \in \mathbb{R}^W$ is the search mask for read head i; the lookup key might only include a partial information to seek in the rows of the memory matrix, similar to keywords. Subsequently, it is imperative that the rest of the key is masked, as to prevent noise in the lookup operation.
- $\mathbf{k}_t^w \in \mathbb{R}^W$ is the lookup key for the write head
- $\beta_t^w = oneplus(\hat{\beta}_t^w) \in \mathbb{R}$ is the key strength of the write head
- $\mathbf{m}_t^w = \sigma(\hat{\mathbf{m}}_t^w) \in \mathbb{R}^W$ is the search mask for the write head
- $e_t = \sigma(\hat{e}_t) \in \mathbb{R}^W$ is the erase vector, which governs how information is erased
- v_t is the write vector, containing information that is to be written
- $f_t^i = \sigma(\hat{f}_t^i) \in \mathbb{R}$ is the free gate of read head i, which determines whether the most recently read-from location by the read head can be freed or not
- $g_t^a = \sigma(\hat{g}_t^a) \in \mathbb{R}$ is the allocation gate, which controls how much will be written based on the allocation manager's output and how much based on the content lookup mechanism
- $g_t^w = \sigma(\hat{g}_t^w) \in \mathbb{R}$ is the write gate, which regulates how strong the write weighting is.

Read and Write Heads

Reading from Memory

The purpose of a read head is to efficiently obtain the demanded from the controller information from the memory matrix. To do so, each read head outputs a read weighting

[4] $oneplus(x) = 1 + ln(1 + e^x)$

$w_t^{r,i} \in \Delta_N$, where $\Delta_N := \{\alpha \in \mathbb{R}^N | \sum_{i=1}^{N} \alpha_i \leq 1\}$ is the N-dimensional unit simplex. This weighting determines the locations from which the head will read data from. Therefore, the read vector of the head i is:

$$r_t^i = M_t^T w_t^{r,i}$$

Writing in Memory

Similarly, the write head computes a write weighting $w_t^w \in \Delta_N$, which encodes the locations to be overwritten. The write head receives from the controller a write vector v_t and an erase vector e_t to update the matrix. Based on the formerly read from location and free gates, dictating whether that data is still necessary, the write head constructs the vector $\psi_t \in \mathbb{R}^N$, which removes unneeded information from memory.

$$M_t = M_{t-1} * \psi_t \mathbf{1}^T * (\mathbf{E} - w_t^w e_t^T) + w_t^w v_t^T$$

where "$*$" is element-wise multiplication and $\mathbf{E} \in \mathbb{R}^{N \times W}$ is a matrix of ones.

Content-based Lookup

Content-based lookup implements a mechanism for searching in memory. It uses a key $\mathbf{k} \in \mathbb{R}^W$, a key mask $\mathbf{m}$ that prevents noise from impacting the search process and, finally, a key strength β (the importance of the keyword).

We define the content-based lookup operation as:

$$\mathcal{C}(M, \mathbf{k}, \mathbf{m}, \beta) = softmax\Big[\mathcal{D}(\mathbf{k} * \mathbf{m}, M * \mathbf{1m})\beta\Big]$$

where $\mathcal{D}(u, v)$ is the cosine similarity

$$\mathcal{D}(u, v) = \frac{u^T v}{|u||v| + \epsilon} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2}\sqrt{\sum_{i=1}^{n} v_i^2} + \epsilon}$$

$u, v \in \mathbb{R}^n$ are vectors and ϵ is added for numerical stability in computation.

Dynamic Memory Allocation

Just like computers with Random-access memory, the DNC utilizes a dynamic memory allocation mechanism, which dictates where new information is written.

The allocation manager receives the read weightings $w_{t-1}^{r,i}$, $i \in 1:R$ from the previous timestep and R free gates $f_t^i \in [0,1]$ from the interface vector. It then computes the vector ψ_t, which shows how much of the most recently read-from location will remain after the memory update.

$$\psi_t = \prod_{i=1}^{R} 1 - w_{t-1}^{r,i} f_t^i \qquad \psi_t \in [0,1]^N$$

The next step for the allocation manager is to update its usage vector u_{t-1} (per-row usage). Since the last update, additional memory has been written in w_{t-1}^w and some data that has been read can be discarded. Therefore, the update takes the following form:

$$u_t = (u_{t-1} + w_{t-1}^w - u_{t-1} * w_{t-1}^w) * \psi_t$$

After that, an index vector ς_t is constructed that contains the memory locations in ascending order of usage ($\varsigma_t[1] = \arg\min_i u_t[i]$), so that the rows that are least used get more data written. Finally, the allocation vector a_t, signifying where new data should be written, is calculated as:

$$a_t\big[\varsigma_t[j]\big] = \Big(1 - u_t\big[\varsigma_t[j]\big]\Big) \prod_{i=1}^{j-1} u_t\big[\varsigma_t[i]\big]$$

Write Weighting

Along with the allocation vector, the write head can also choose to write based on content and so, we also have a content write weighting

$$c_t^w = \mathcal{C}(M_{t-1}, \mathbf{k}_t^w, \mathbf{m}_t^w, \beta_t^w)$$

The allocation gate g_t^a linearly interpolates the content weighting and the allocation vector. The write head could also choose not to write at all, therefore the gate g_t^w is used.

$$w_t^w = g_t^w \Big(g_t^a a_t + (1 - g_t^a) c_t^w\Big)$$

Read Weighting

The read head utilizes content-based lookup. The i-th read head computes its read weighting as

$$w_t^{r,i} = c_t^{r,i} = \mathcal{C}(M_t, \mathbf{k}_t^{r,i}, \mathbf{m}_t^{r,i}, \beta_t^{r,i})$$