UNDER REVIEW

RESCIENCE C

Replication / ML Reproducibility Challenge 2022

[Re] End-to-end Algorithm Synthesis with Recurrent Networks: Logical Extrapolation Without Overthinking

Sean McLeish[1] and Long Tran-Thanh[1]

[1] Department of Computer Science, University of Warwick, Coventry, England

Edited by
(Editor)

Reviewed by
Anonymous Reviewers

Received
–

Published
–

DOI
–

Reproducibility Summary

Scope of Reproducibility – In this report, we aim to validate the claims of Bansal et al [1]. These are that the recurrent architecture presented, with skip connections and a progressive loss function, prevent the original problem being forgotten or corrupted during processing, allowing for the recurrent module to be applied indefinitely and that this architecture avoids the overthinking trap. We use both code released by the authors and newly developed to recreate many results presented in [1]. Additionally, we present analysis of the newly introduced alpha hyperparameter and investigate interesting perturbation behaviour of prefix sums models. Further, we conduct a hyperparameter search and provide an analysis of the Asymptotic Alignment scores [2] of the models presented.

Methodology – We use the PyTorch code released by the authors to replicate accuracy experiments. We then, independently, develop our own code using PyTorchFI [3] to replicate perturbation experiments presented in [1]. Overall, providing a replication of all results shown in the main body of [1]. We then extend these results, providing an analysis of the alpha hyperparameter, analysis of perturbation recovery, Asymptotic Alignment scores [2] and a hyperparameter search. We used both a Nvidia RTX 2080Ti GPU and sets of three NVIDIA Quadro RTX6000 GPUs, taking a total of 982.2 GPU hours to create all results presented in the main body of this report.

Results – We verify the authors' claims by replicating the experiments presented in [1]. All of our experiments show identical results to the ones presented in [1], apart from perturbation testing for which we provide an additional in depth analysis. We also provide an analysis of the new alpha hyperparameter and a hyperparameter search.

What was easy – The code provided by Bansal et al gave clear instructions on how to use it, along with pretrained models being available for all problems.

What was difficult – Chess models required a considerable amount of time to train, putting a drain on resources. Also, code for reproducing perturbation results was not available so this had to developed from scratch.

Communication with original authors – We had good communication with the original authors, both emailing and meeting online.

Copyright © 2023 S. McLeish and L. Tran-Thanh, released under a Creative Commons Attribution 4.0 International license.
Correspondence should be addressed to Sean McLeish (sean.mcleish@warwick.ac.uk)
The authors have declared that no competing interests exist.
Code is available at https://github.com/mcleish7/MLRC-deep-thinking. – SWH swh:1:dir:be1a763d08c1af34a2721fad6b9bfe8c99527220.
Open peer review is available at https://openreview.net/forum?id=WaZB4pUVTi.

1 Introduction

In previous work by Schwarzchild et al. [4], it is shown that recurrent neural networks have the capability to logically extrapolate. This is where models trained on small/"easy" problems can extrapolate to solve larger/"harder" versions of the same problem, without the need for extra training. This is similar to the way humans learn, starting with toy problems, eventually being able to solve complex problems of the same nature.
Bansal et al. [1] extend this work introducing two new components to the neural networks used in [4]. These are 'recall', which is adding skip connections at the beginning of each application of the recurrent module, so the original problem is reintroduced to the network repeatedly; similar to rereading the question. Also added is a progressive loss function, this helps stop the model learning time related behaviour, crucial as we want to apply the recurrent module a different number of times, dependent on the complexity of the problem being solved. This progressive loss is analogous to truncated back propagation through time [5]. These models are then tested on three problems:

1. Prefix Sums: Given a bit string, compute its prefix sum.
2. Mazes: Given a maze with the start marked as green and end as red respectively, find a route between the two points using the white corridors.
3. Chess: Given a chess board, find the optimal next move.

The conclusion of this work is in the title of the paper, the neural networks introduced are able to synthesise algorithms to solve these problems. This is a large progression from previous work and combined with the claimed high out of distribution testing accuracy gained from the skip connections and progressive loss function, is a new and novel technique with possible far reaching impacts if the results are repeatable.
While Bansal et al. released code to reproduce the accuracy results they show, many require changes to input parameters and work to present them in the same format as originally shown in [1]. We then extend this, creating experiments to test all perturbation results shown along with an in depth analysis of these results. Also, we conduct a hyperparameter search by extensively searching the space of possible hyperparameters, especially focused on the alpha hyperparameter introduced in the progressive loss function, to verify hyperparameter choice. Moreover, we integrate this with work from Anil et al. [2] to generate Asymptotic Alignment scores for the models Bansal et al. propose. This new analysis shows on a deeper level the models unique behaviour under perturbation and the impact of the newly introduced alpha hyperparameter on both accuracy and Asymptotic Alignment score.

2 Scope of reproducibility

In this report we aim to validate the claims of Bansal et al., present analysis of the alpha hyperparameter and investigate interesting perturbation behaviour of prefix sums models.

1. First, we reproduce all results shown by Bansal et al. in [1]. These can be summarised as:
    - The provided recurrent architecture, with skip connections, prevents the original problem from being forgotten or corrupted.
    - The new progressive training routine prevents the network learning time specific behaviours, so the recurrent module can be applied indefinitely.

- The models with this recurrent architecture avoid the overthinking trap, scaling arbitrarily.

  These claims are validated in [1] by testing against the easy to hard data set [6], showing higher accuracy than previously achieved, extrapolation to much more complex examples and that the original problem is not forgotten, we will verify them in the same way.
  Bansal et al. often validate their three claims together for each of the three test cases, this can be seen in Figures 3 to 6 of their paper. However, they do also validate the robustness of the skip connections and convergence over time in Figure 7 of their work.

2. We also do an analysis of the alpha hyperparameter introduced by Bansal et al. to control the progressive loss.
   The alpha hyperparameter is shown to be crucial by Bansal et al. as it varies from 1.0 for prefix sums models to 0.01 for maze models, we look at the behaviour of this parameter for maze models.

3. We do an analysis of what is causing prefix sums models to recover quicker from some perturbations than others.
   When reproducing the results we saw interesting behaviour in perturbation recovery, we look at this on a closer level for prefix sums models.

3 Methodology

We are thankful to Bansal et al. for releasing such a well written code base, [1] different to most research code bases, the authors use the Hydra package [7] to control hyperparameters making the code much more readable. There is ample documentation in the README.md file and launch directory both in the GitHub repository to begin training models. In this code base there is all of the material to reproduce any accuracy experiment shown in [1]. However, the code for perturbation experiments, which we believe to be as crucial as the accuracy experiments because they validate the methodology, was not released by the authors nor shared with us. We decided to use the PyTorchFI [3] library to implement these experiments as it allows run time access to the features of the networks with ease. We had good access to GPUs, thanks to department resources, sums and maze models were trained on a single Nvidia RTX 2080Ti GPU and chess models were trained on multiple sets of three NVIDIA Quadro RTX6000 GPUs.

3.1 Model descriptions

The models used are specific to this paper so there is a detailed description of them in [1] but we will highlight the key points here. The Deep-Thinking (DT) models are created by using a recurrent module which can be repeatedly applied and is based on a ResNet block [8] with ReLUs in between each layer in the block to create a recurrent neural network. The DT-Recall networks have the original input concatenated to the current features each time the recurrent module is applied. DT-Recall-Progressive (DT-Recall-Prog) models have an alpha hyperparameter strictly greater than 0 meaning that the progressive loss is used in training, which can be summarised as $L = (1 - \alpha) \cdot L_{max-iters} + \alpha \cdot L_{progressive}$ and is detailed fully in Algorithm 1 of [1] by Bansal et al.

[1] https://github.com/aks2203/deep-thinking

3.2 Datasets

All of the data used is taken from the easy to hard datasets [6], this is downloaded locally at run time automatically by installing the PIP package of the GitHub repository. [2] This is in the requirements.txt of the code base provided by Bansal et al. so should be installed automatically when setting up the whole code base. For prefix sums there are 10,000 examples of each size available and we used the default train to validation split of 80/20. Details of the examples available are in [6] or in the README.md of the GitHub repository for the datasets. As we are testing the model's ability to extrapolate, we test on a set of different size so we can use all the examples from this set. For mazes, we have 50,000 training examples again with a train to validation split of 80/20 and 10,000 testing examples. For maze models there is no defined difference between the sets for training and testing unlike for prefix sums where a set can be used for training or testing. For chess, we have 1.5 million examples sorted in ascending order of difficulty, for training we define a set from 0 to i and for testing we define a, not necessarily disjoint, set of [j-100,000, j] examples, the default i and j in [1] are 600,000 and 700,000 respectively.

3.3 Hyperparameters

We found the hyperparameters provided were stable and good choice in all cases, in the sense that they allowed us to reproduce all results with high accuracy but some can be varied without impact perhaps making them more suitable for particular use cases. We do however do a further analysis of the alpha hyperparameter in Section 4.2 and a larger hyperparameter search in Appendix Section A.2.

3.4 Experimental setup and code

The accuracy results presented are quickly recreated using the code base from Bansal et al. Specifically, the README.md, launch files and data analysis files are needed to do this. For the perturbation results, which we have implemented in Python independently, it is slightly more complicated; as we need to firstly train the models, then reference them separately to do the testing. Meaning that there is a small amount of python knowledge required here, specifically, how file paths work and creating a string, before the user can execute this code.

3.5 Computational requirements

As referenced above, we used both a Nvidia RTX 2080Ti GPU with 11GB of memory and sets of three NVIDIA Quadro RTX6000 GPUs with 22GB of memory. The time taken to reproduce accuracy results are shown in Table 1. The maze testing takes longer due to the number of iterations as we test for one thousand iterations. The chess training takes longer as our compute resources have a two day time limit so we use the implemented checkpoint system and restart from the best previous iteration meaning some iterations are ran twice, as they are not recalled in the second run when training the model. For perturbation results, the sums single bit flip result takes approximately 4.5 days on a single GPU as this goes through each of the 10,000 examples one at a time. The maze perturbation and average change in features results take 4.5 hours and 1 hour on average respectively.

4 Results

Our results almost flawlessly support all claims made by Bansal et al. Also, we provide extra detail on the alpha hyperparameter and its role in Section 4.2. Following this,

[2] https://github.com/aks2203/easy-to-hard-data

Problem	Average Time to Train (hours)	Average Time to Test (hours)
Prefix Sums	2	0.2
Mazes	9	2.25
Chess	72	1.3

Table 1. Average time taken to train and test the three types of models.

we do further analysis of the time taken to recover from perturbation for prefix sums models.

4.1 Results reproducing original paper

As stated in Section 2, the claims of the paper are often validated together for each of the three test cases in an ablation style, comparing each of the four possible types of models that can be created with the recall connections and progressive loss: Deep Thinking (DT), Deep Thinking Progressive (DT-Prog), Deep Thinking Recall (DT-Recall) and Deep Thinking Recall Progressive (DT-Recall-Prog). This further verifies the claims as we can be sure that both the recall and progressive loss are needed for the highest accuracy.

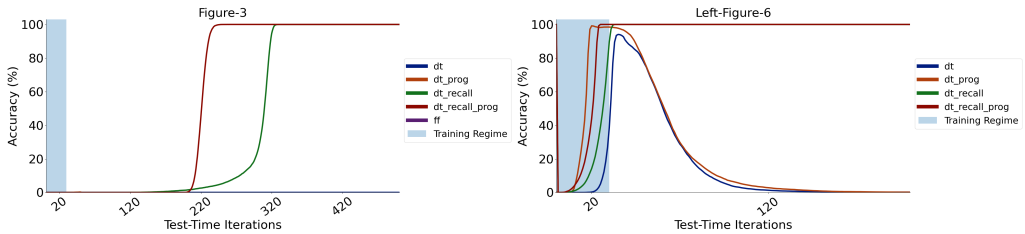

Figure 1. My reproductions of:
Left: 'Figure 3' of [1]. Prefix sums models trained on 32 bit data, tested on 512 bit data.
Right: 'Left of Figure 6' of [1]. Prefix sums models trained on 32 bit data, tested on 48 bit data.

Prefix Sums Experiments — These experiments support all claims in point one of Section 2. The prefix sums results from Bansal et al. can be summarised in Figures 3 and left of 6 of [1]. These results show DT-Recall models out performing DT and Feedforward networks in accuracy over time. This supports all of the claims as the original problem is not forgotten for 500 test iterations maintaining 100% accuracy to the point when testing is stopped. The recurrent module is applied for this many steps also, supporting part two of the claim. The overthinking trap is avoided as once the solution has been found it is maintained. Our reproductions can be seen in Figure 1, we do see a difference in the DT-Prog line not gaining some accuracy when testing on 512 bits and a difference in the speed of degradation in accuracy for the DT-Prog model when testing on 48 bit data. We questioned Bansal et al. on this and believe it is because they had a 99% training accuracy threshold for all models; we did not apply this, meaning some of our models may have performed worse. This reinforces the repeatability of the claim for prefix sums models as the DT-Recall models were both reproduced easily without this threshold.

Mazes Experiments — These experiments support all claims in point one of Section 2. The maze models maintain the original problem throughout, supporting part one of the claims as the models maintain high accuracy once it has been achieved; also supporting part three of the claims that the overthinking trap is avoided. The recurrent module is applied many more times than in the prefix sums experiments, further supporting that

it can be applied indefinitely and scale to arbitrarily many times. These results from Bansal et al. can be summarised in Figures 4 and Right of 6 of [1] and our reproductions can be seen in Figure 2. We see slight differences mainly attributed to our reproductions not using a 99% accuracy threshold; meaning we see some of our reproductions fall towards the lowest variance bounds in the plots in [1], such as the DT-Recall model in Left of Figure 2.

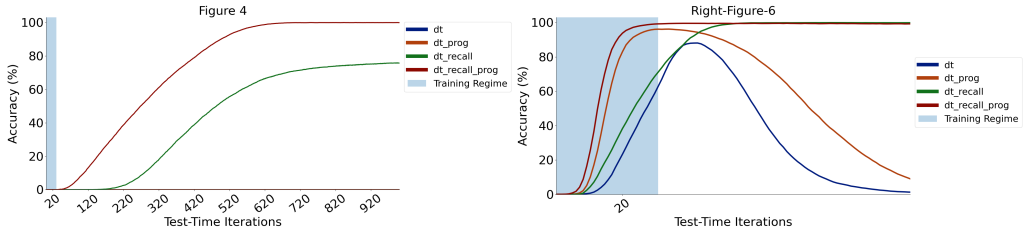

Figure 2. My reproductions of:
Left: 'Figure 4' of [1]. Maze models trained on 9x9 data, tested on 59x59 maze data.
Right: 'Right of Figure 6' of [1]. Maze models trained on 9x9 data, tested on 13x13 maze data.

Chess Experiments — These experiments support all claims in point one of Section 2. All claims are supported as the original problem is not forgotten and the overthinking tap is avoided as the DT-Recall-Prog model does not deteriorate after finding a high accuracy solution. Also, the models are tested for a different number of test iterations compared to prefix sums and mazes further supporting part two of the claims. This time we also reproduce an appendix result for completeness as for both prefix sums and mazes we showed testing for both a large and small extrapolation. Our reproductions of Figure 5 and 9 can be seen in Figure 3, of chess models trained on the first 600,000 examples and then tested on examples [600k,700k] and [1M, 1.1M] respectively. We do see the DT-Recall line peaking achieving multiple local maxima, throughout the testing in both cases; this could be the skip connections in practice but the lack of progressive loss function means that these peaks are not held and deteriorate. We see the DT-Prog models again achieving lower accuracy than expected, this is due to the lack of 99% training accuracy threshold, which appears to impact the progressive model in all experiments. Our DT models accuracy peaks higher than in [1] but without recall and progressive loss this accuracy deteriorates quickly, highlighting the impact of the additions Bansal et al. have made.

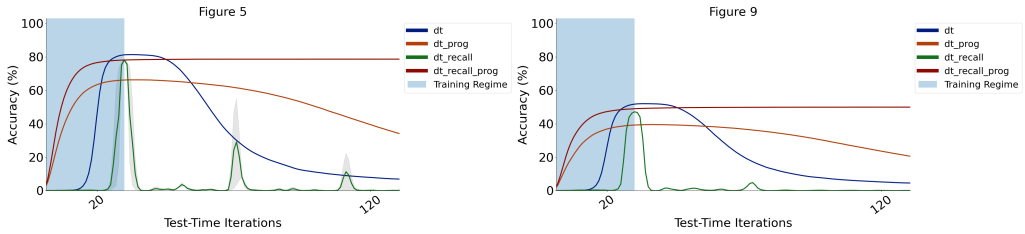

Figure 3. My reproductions of:
Left: 'Figure 5' of [1]. Chess models trained on the first 600k examples, tested on examples [600k,700k].
Right: 'Right of Figure 9' of [1]. Chess models trained on the first 600k examples, tested on examples [1M, 1.1M].

Perturbation Experiments — As stated earlier, this code has been independently reproduced by ourselves, so this further supports the repeatability of the results shown.
There are three perturbation results shown in Figure 7 of [1]. Left of Figure 7 shows the time taken to recover from a one bit flip at the ith index after solving for 50 iterations in a prefix sums test. A reproduction of this is shown in Figure 4, we see hooked ends on the left side, a further analysis of this is provided in Section 4.2 - Hooked End Perturbation Analysis. Middle of Figure 7 of [1] shows the time taken to recover after swapping the features, after solving for 50 iterations for a maze model. We instead reproduce Figure 12 where all features are set to zero instead of another maze. These two results reinforce part one of the claim that the skip connections prevent the original problem being forgotten as even when perturbed the models still solve for the original problem solution. Also supporting part two of the claims of [1], that the learning is independent of time as the models recover from these perturbations after the 50th iteration when the original problem had already been solved. Right of Figure 7 of [1] shows the change in features at each step of solving, measured by the L2 norm. This supports part three of the claim that the overthinking trap is being avoided as both the DT-Recall models continue to converge over time. This result does depend heavily on the models being used to produce it which explains the difference in values but the same overall pattern is seen.

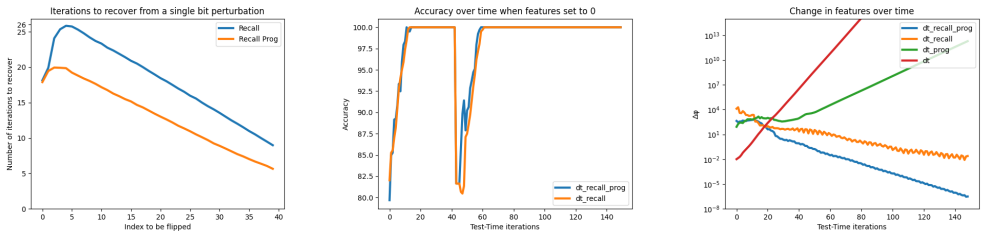

Figure 4. My reproductions of:
Left: 'Left of Figure 7' of [1]. Prefix sums models perturbed by flipping the ith bit after 50 iterations.
Middle: 'Figure 12' of [1]. Maze models perturbed by setting all features to 0 after the 50th iteration.
Right: 'Right of Figure 7' of [1]. Change in the features of maze models over time when solving, measured by the L2 norm.

4.2 Results beyond original paper

The new progressive loss measure is well reasoned in [1] but there is no evidence to relate this to the choice of the alpha value for each model. In this section we further investigate this and also look into what is causing the hooked ends in Left of Figure 4.

Alpha Value Analysis — As seen in Figure 1 both the DT-Recall and DT-Recall-Prog lines reach 100 % accuracy for prefix sums models but for mazes only the DT-Recall-Prog model reaches 100% accuracy, so we do our investigation on maze models. This will increase the reliability of results as in prefix sums models there is a lot of variability from the optimiser found, also noted by Bansal et al. in their paper. There is research into stabilising the optimiser that is found, such as by Bai et al. in [9] but this is not implemented in the current code base. Here we found that there does appear to be a point of degradation in accuracy as we change the alpha value, shown in Figure 5. We do note that the networks may find a good optimiser from other alpha values due to the random nature of the learning process but in Figure 5 we average multiple models to gain a repeatable picture of the models behaviour. It is important to note the default alpha value of 0.01 for mazes is above the threshold of 0.1 we have found, so this supports the authors choice.

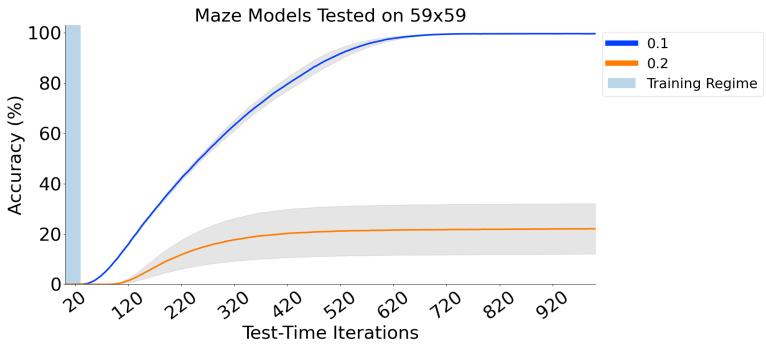

Figure 5. Multiple maze models trained on 9x9 mazes, tested on 59x59 mazes with alpha values of 0.1 or 0.2.

Hooked End Perturbation Analysis — We decided we would look at the amount of work done by the models when recovering from perturbation which we measured by the amount of times the value of a bit is changed. We then averaged these results, which can be seen in Left of Figure 6. We then multiplied the average number of bits changed per iteration by the average time to recover to get the average total amount of work done over all iterations, which can be seen in Right of Figure 6. These both suggest that the amount of work the model is doing is causing the hooks, possibly implying the models are learning the importance of these bits at the left end of the string, as these as the most important. This highly tailored behaviour is extraordinary and from conversations with the authors we agree this difference to the origional paper is because we have averaged over all testing examples, where as they only tested a subset for Figure 7 of [1]. Bansal et al. noted similar extraordinary behaviour was seen in maze models when they are perturbed but we could not reproduce this in a manner which can be presented in a paper. Linking this point back to the previous two, when creating Middle of Figure 4 we repeated the result over multiple alpha values and found that testing on a subset we could produce the result with a large range of alpha values; where as when we worked with all examples available, the set of alpha values which could recover was only those smaller than 0.1, i.e. above the threshold of 0.1 found in Section 4.2 - Alpha Value Analysis.

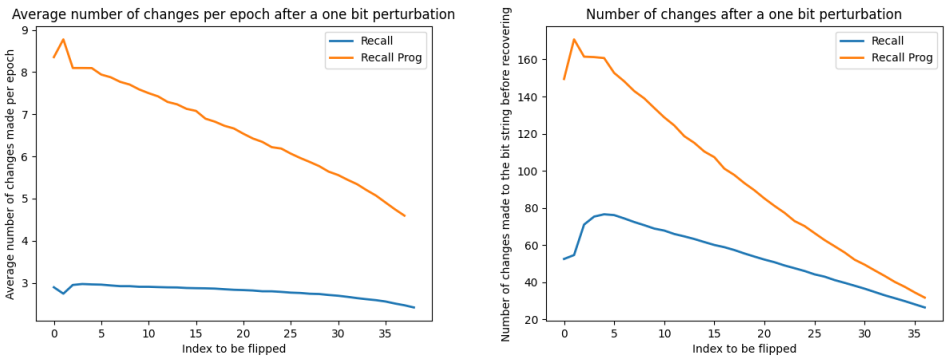

Figure 6. Left: The average number of changes to a bits value per iteration after a one bit perturbation.
Right: The total number of changes to a bits value after a one bit perturbation.

5 Discussion

Overall, all of our results support those of Bansal et al. We shed more light on the new alpha hyperparameter presented and tested aspects more rigorously revealing the truly remarkable nature of the models created in [1]. The largest noticeable weakness in our work is that the code base released by the authors is used and we have not recreated fully the code from the description given in the paper. However, over the course of this project we have become very well accustomed with both the paper and code and believe there is no discrepancy between the two. Also, we did not reproduce perturbations in the exact same way that the authors did but the benefit of this is that we have seen the same results from two perspectives. There are many strengths to this reproduction, the main ones being the rigour; models were trained and tested in batches of all models needed for one experiment, so there was no cherry picking of results. Moreover, for the perturbations section we reproduced the results fully independently, we believe this to be the gold standard of validation for the claims made in the paper by Bansal et al.

5.1 What was easy

The code base released by Bansal et al. is very well written and documented, the use of the Hydra package [7] made all hyperparameters easily available to both change and understand their function. The paper is written in a very intuitive way with the wording lending itself to the popular definitions of the words, such as 'overthinking' meaning deteriorating accuracy due to processing for too long. The authors were easily accessible and more than happy to help with any questions we had, which made the experience of reproducing the results much better.

5.2 What was difficult

The chess models requiring so much time to train was a drain both on time and resources but overall added to the rigour of the report. We had to request access to extra compute resources to do this as they require a lot of memory to train and test. From the wording in [1] we originally believed all code was available in the GitHub repository, so having to create all of the code for perturbation analysis was unexpected but did give us a deeper understanding of the code base and models, as we were dealing with them on a layer by layer basis.

5.3 Communication with original authors

We had very good communication from the authors, with Avi Schwarzschild providing an email chain for questions and setting up multiple meetings with Eitan Borgnia and Arpit Bansal, for which we are extremely grateful. Through out the report we have mentioned where the authors added to our discussions.

5.4 Acknowledgements

We thank Avi Schwarzschild for feedback and comments on the report. We would also like to thank the Richard Cunningham and the Scientific Computing Research Technology Platform from the University of Warwick for maintaining the compute facilities used for this project. We also thank the reviewers for their detailed feedback.

References

1. A. Bansal, A. Schwarzschild, E. Borgnia, Z. Emam, F. Huang, M. Goldblum, and T. Goldstein. "End-to-end Algorithm Synthesis with Recurrent Networks: Logical Extrapolation Without Overthinking." In: **Advances in Neural Information Processing Systems** 35 (2022).
2. C. Anil, A. Pokle, K. Liang, J. Treutlein, Y. Wu, S. Bai, J. Z. Kolter, and R. B. Grosse. "Path Independent Equilibrium Models Can Better Exploit Test-Time Computation." In: **Advances in Neural Information Processing Systems**. Ed. by A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho. 2022. URL: https://openreview.net/forum?id=kgT6D7Z4Xv9.
3. A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari. "PyTorchFI: A Runtime Perturbation Tool for DNNs." In: **2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)**. 2020, pp. 25–31.
4. A. Schwarzschild, E. Borgnia, A. Gupta, F. Huang, U. Vishkin, M. Goldblum, and T. Goldstein. "Can You Learn an Algorithm? Generalizing from Easy to Hard Problems with Recurrent Networks." In: **Advances in Neural Information Processing Systems** 34 (2021).
5. H. Jaeger. "Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach." In: (2002).
6. A. Schwarzschild, E. Borgnia, A. Gupta, A. Bansal, Z. Emam, F. Huang, M. Goldblum, and T. Goldstein. **Datasets for Studying Generalization from Easy to Hard Examples**. 2021. arXiv:2108.06011.
7. O. Yadan. **Hydra - A framework for elegantly configuring complex applications**. Github. 2019. URL: https://github.com/facebookresearch/hydra.
8. 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**. 2016, pp. 770–778.
9. S. Bai, V. Koltun, and J. Z. Kolter. "Stabilizing Equilibrium Models by Jacobian Regularization." In: **International Conference on Machine Learning (ICML)**. 2021.
10. L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han. "On the Variance of the Adaptive Learning Rate and Beyond." In: **International Conference on Learning Representations**. 2020. URL: https://openreview.net/forum?id=rkgz2aEKDr.
11. I. Loshchilov and F. Hutter. "Fixing weight decay regularization in adam." In: (2018).

A Appendix

A.1 Asymptotic Alignment Score

Bansal et al. looked to find models which are able to extrapolate from small to larger problems. In [2], Anil et al. define the notion of path independence for equilibrium models. The models developed by Bansal et al. in [1] are a subset of equilibrium models. This allows us to view the problem from another perspective knowing how our models behave in practice, we can relate this to the theory. Calling the process of training on an "easy" set of problems and testing on a "harder" set of problems upwards generalisation. Anil et al. define path independence of a model to be "converg[ing] to the same limiting behaviour regardless of the current state." We have already seen examples of the path independence property being displayed by the Deep Thinking models, with the models recovering from perturbation, converging to the same solution from the initial start point and after perturbation. Anil et al. propose the Asymptotic Alignment score (AA score) to quantitatively measure path independence, which is detailed in Algorithm 1, where f_w is a network which in the case of the Deep Thinking models is run for many iterations on each input. Overall, an AA score of 1 indicates high path independence and lower values less so, as a score of 1 means that the two solutions found from two different start points converged to the same point.

Linking back to Section 4.2 - Alpha Value Analysis, we looked at how the AA score is impacted as the alpha value changes; this can be seen in Figure 7. Here we see the AA score inferring to an extent the accuracy of the models for the larger extrapolations; a low AA score also indicates low accuracy as if a model does not converge, it will also be inaccurate. The maze models have higher AA scores for lower alpha values for which we expect high accuracy and the chess models have higher AA scores for higher alpha val-

Algorithm 1 Asymptotic Alignment score

Input: $X = [x_1, x_2, \ldots, x_n]$, f_w ▷ Each x_i is an input to f_w
Z = $[0, 0, \ldots, 0]$ ▷ The same shape as x
$Z' \leftarrow f_w(X, Z)$ ▷ Run the input network on input X(i) starting from Z(i)

$Z_p \leftarrow$permute Z' so $Z'(i) \neq Z_p(i)$
$Z'' \leftarrow f_w(X, Z_p)$
return average(cosDistance($Z'(i), Z''(i)$))

procedure COSDISTANCE(x_1, x_2)
return $\frac{x_1}{\|x_1\|_2} \cdot \frac{x_2}{\|x_2\|_2}$
end procedure

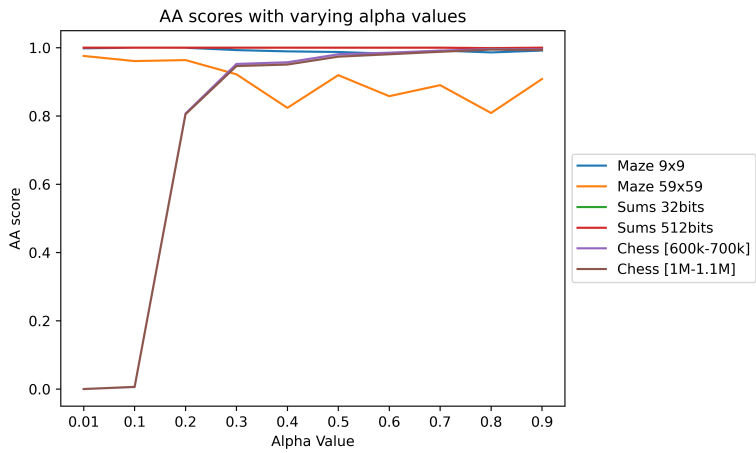

Figure 7. The AA scores of DT-Recall-Prog models for all three problems with varying alphas

ues, for which we also see high accuracy in testing. Interestingly, the AA scores are high for all values of alpha for prefix sums models. This may infer as the problem becomes harder, the choice of alpha parameter is more important to achieving high accuracy; obviously this cannot be verified without extending the networks to solve more problems. Overall, we have empirically analysed at the link between the AA score and the accuracy of the models by comparing to the link between alpha value and the models accuracy.

A.2 Hyperparameter Search

As we were using the default hyperparameters given by Bansal et al., we felt it was important to try some variations of these hyperparameters. The authors did note that this is the third iteration of these models to be produced, so many of the hyperparameters have been well tuned over time. For clarity, we include Table 2 detailing the original hyperparameters used by Bansal et al.
Due to the cost of training chess and maze models compared to prefix sums models the majority of our search is done over prefix sums models. For each of the tests only the stated hyperparameters are varied and all others remain the same, meaning we can compare to the plots shown in Section 4.1 to see the impact. We do not vary the number of epochs as the DT-Recall-Prog models maintain high accuracy once it is achieved, due to their architecture as shown in Section 4. All prefix sums experiments shown are trained on 32 bit data and tested on 512 bit data. All mazes experiments shown are trained on 9x9 data and tested on 59x59 data. We show a subset of alpha values for each experiment, for prefix sums higher alpha values are used and for mazes lower alpha

values are used, as evidence shown suggests these are the most likely alpha values to consistently achieve high accuracy. Each experiment has been repeated multiple times to verify the results shown.

	Prefix Sums	Mazes	Chess
Alpha	1	0.01	0.5
Epochs	150	50	130
Learning Rate	0.001	0.001	0.01
LR Decay	step	step	step
LR Factor	0.1	0.1	0.1
LR Schedule	60,100	100	100,115
LR Throttle	False	True	False
Optimiser	Adam	Adam	SGD
Clip	1	N/A	N/A
Train Batch Size	100	50	300
Test Batch Size	500	25	300

Table 2. The default hyperparameters from [1].

Learning Rate — For prefix sums the default learning rate is 0.001, we tested both 0.01 and 0.0001 over a range of alpha values. The learning rate of 0.01 led to vanishing gradients for all alpha values and the learning rate of 0.0001 made no discernible difference when compared to the learning rate of 0.001. This is shown in Left of Figure 8. We have not included plots for models which experienced vanishing gradients as they fail to achieve any noticeable accuracy.

Optimiser — We tested stochastic gradient descent (SGD), RAdam [10] and AdamW [11] as alternatives to the default Adam optimiser for prefix sums models. We varied alpha values and used learning rates of 0.01, 0.001 and 0.0001 for SGD and a learning rate of only 0.001 for both variations of the Adam optimiser. Interestingly, the SGD optimiser led to vanishing gradients for all learning rates and all alpha values, we spoke to Bansal et al. and they said they had experienced the same problem with SGD leading them to use Adam. For both RAdam and AdamW we saw no discernible difference to the normal Adam optimiser in the accuracy of the models produced, still seeing high variability in the time taken by the models to reach high accuracy. These results can be seen in Figure 9.

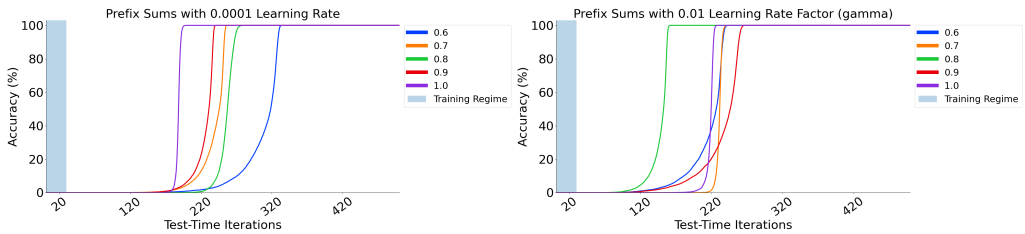

Figure 8. Left: Accuracy of prefix sums models with varying alpha values and learning rate of 0.0001.
Right: Accuracy of prefix sums models with varying alpha values and learning rate factor of 0.01.

Learning Rate Factor — Next in our hyperparameter search on prefix sums models, we looked at learning rate factor, this is the amount the learning rate is multiplied by at each step, sometimes referred to as the gamma hyperparameter. We tested a factor of

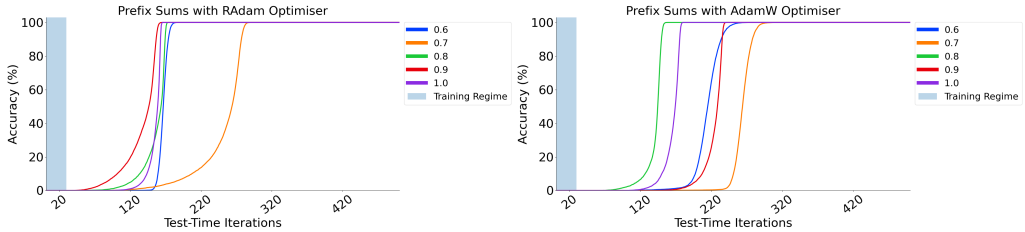

Figure 9. Left: Accuracy of prefix sums models with varying alpha values and RAdam optimiser.
Right: Accuracy of prefix sums models with varying alpha values and AdamW optimiser.

0.01, ten times smaller than the default value; we did not test values smaller as we felt the difference each iteration would then make to the overall optimisation would be too small. This did not lead to any noticeable change in the overall accuracy or time taken to reach peak accuracy of the models. This result is shown in Right of Figure 8.

Learning Rate Scheduler — We also experimented by changing the scheduler from a mutli-step scheduler to a cosine annealing scheduler on prefix sums. We used the PyTorch default values for torch.optim.lr_scheduler.CosineAnnealingLR scheduler. The result of this can be seen in Figure 10. The evidence suggests changing the scheduler does not impact the ability of the prefix sums models to extrapolate, therefore either scheduler can be used for these Deep Thinking models.

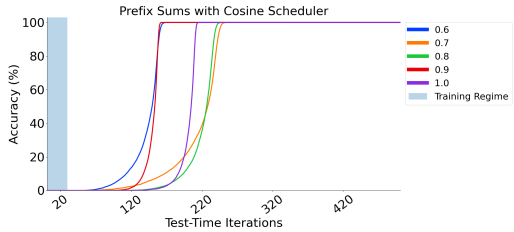

Figure 10. Accuracy of prefix sums models with varying alpha values and cosine annealing scheduler

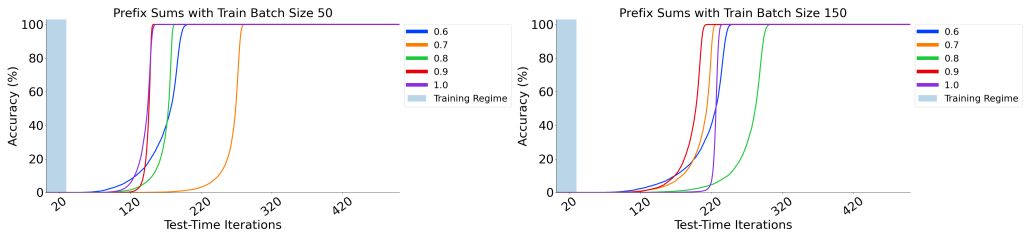

Figure 11. Left: Accuracy of prefix sums models with varying alpha values and training batch size of 50.
Right: Accuracy of prefix sums models with varying alpha values and training batch size of 150.

Training Batch Size — To complete our hyperparameter search for prefix sums models we look at the batch size when training models. This is often limited by memory when training on GPUs but is non the less a crucial hyperparameter and often impacts accuracy. We tested training batch sizes of 50 and 150, i.e. ±50 from the default train batch size suggested by Bansal et al. The results of this experiment can be seen in Figure 11; this

suggests that the batch size can be increased or decreased during training whilst maintaining high accuracy when extrapolating. This is a positive result as if the GPU being used has enough memory we can increase the speed of training by increasing the batch size and if the GPU has limited memory we can decrease the batch size; increasing the accessibility of the models.

We also changed the batch size during training for maze models. We tested training batch sizes of 25 and 75, i.e. ±25 from the default train batch size suggested by Bansal et al. As you can see in Figure 12, the maze models, which we already know to be less stable than prefix sums, are even less stable when varying the training batch size. The bound of 0.1, previously found Section 4.2 - Alpha Value Analysis, is broken by the models with alpha values of 0.1 failing to gain high accuracy in both cases. However, the default hyperparameter value of alpha being 0.01 suggested by Bansal et al. still gains high accuracy in both cases.

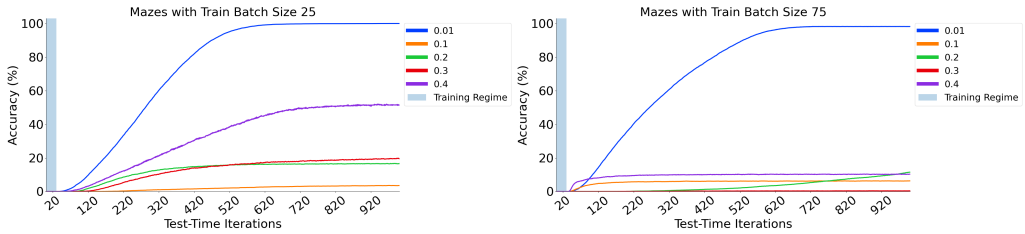

Figure 12. Left: Accuracy of maze models with varying alpha values and training batch size of 25. Right: Accuracy of maze models with varying alpha values and training batch size of 75.

Learning Rate Schedule — For maze models the default learning rate schedule is higher than the number of epochs meaning it is never used when training. We trialled learning rate schedules of [35] and [14,30,45] over the default 50 epochs and varying alpha values. The results, shown in Figure 13, show that these trials lead to worse accuracy than the default. Implying any schedule for maze models only decreases the accuracy, as even models with alpha values above the threshold of 0.1 found in Section 4.2 - Alpha Value Analysis fail to reach noticeable accuracy in both cases. We also turned the learning rate throttle to false, when turned on the error is only backpropagated once for the recurrent module instead of once for each time the recurrent module is used. Mazes were the only one of the three problems to have this hyperparameter set to true by default. This led to the training process failing to gain any noticeable accuracy, so this result is not shown graphically.

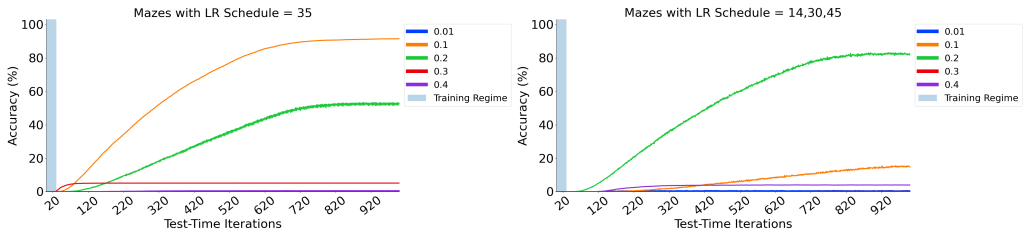

Figure 13. Left: Accuracy of maze models with varying alpha values and [35] learning rate schedule. Right: Accuracy of chess models with varying alpha values and [14,30,45] learning rate scheduler.

Chess Training Range — When discussing the direction of research with Bansal et al. in a meeting, they mentioned that they did limited testing around which data is used to train chess models. The default set is [0,600k] of the 1.5 million available examples. We decided to test values of 400k, 500k, 700k and 800k with the default alpha value of 0.5. In

Left of Figure 14, we see that in an extrapolation of 100,000 from the training range, the lower training range values achieve higher accuracy but still only similar to that seen by the default value. In Right of Figure 14, when testing on [1M, 1.1M] we see the higher training range values achieve higher accuracy, slightly higher than that of the default value but this also a smaller extrapolation for those higher training range values, so this is expected. Overall, we see no evidence to suggest the training range should be changed from these experiments but are encouraged to see that similar observations hold with different training ranges.

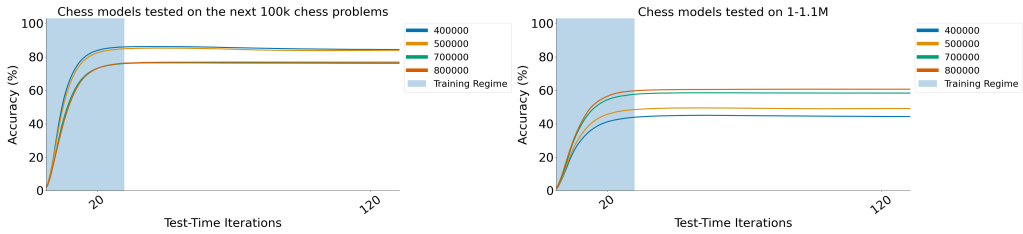

Figure 14. Left: Accuracy of chess models with train ranges 0 to 400k, 500k, 700k and 800k tested on the next 100k examples after their train ranges respectively.
Right: Accuracy of chess models with train ranges 0 to 400k, 500k, 700k and 800k tested on [1M,1.1M].

To conclude, after testing five of the hyperparameters there was no evidence to suggest that any of the default hyperparameters provided by Bansal et al. should be changed. However, evidence does suggest that some of these hyperparameters can be varied without impacting accuracy.