Deep-Thinking Network for Number Comparison

Josue Casco-Rodriguez and Michael Sprintson

Abstract

Quite recently, the field of deep learning has seen breakthroughs in learning complexities in test datasets that are not represented in training. While it has previously been claimed that end-to-end pipelines can not perform algorithmic reasoning, Schwarzschild et al. (2021) seem to have developed a model that disproves this claim. We set out to evaluate the efficiency of one such model from Schwarzschild et al.,'s body of work for a new task that is different in nature from the other tasks they used to evaluate their model. By comparing the performance of this model to a feed-forward network, we hope to contribute to disproving the claim.

Introduction

While artificial neural networks (ANNs) have their roots in nonlinear regression methods, recent advances in computational power and efficiency have bolstered ANNs to the forefront of machine learning efforts and have spawned a large collection of novel neural network architectures and an entirely new body of literature. Common applications of ANNs include classification (e.g., image classification), inference (e.g., forecasting and language translation), and generation (e.g., music or image generation); specific architectures used to accomplish such tasks usually include variants of convolutional neural networks (CNNs), recurrent neural networks (RNNs), and feedforward neural networks.

Quite recently, the field of deep learning has seen breakthroughs in model generalization and algorithmic thinking. Unlike biological neural networks and advanced cognitive systems, traditional neural networks struggle heavily with generalization and symbolic reasoning. A pressing field of inquiry in deep learning efforts concerns whether neural networks can truly learn to reason instead of simply memorizing solutions to a given task. In order to answer such questions, Schwarzschild et al. (2021) developed a novel RNN architecture, which they refer to as a Deep Thinking (DT) architecture, that can solve algorithmic reasoning tasks such as number-sorting, maze navigation, and chess. Of key interest in the DT architecture is its ability to evaluate tasks outside of the distribution it was trained on – specifically, more complex versions of the tasks it was trained on (e.g., longer lists of numbers, bigger mazes, and more difficult chess board setups).

In this work, we aim to evaluate the DT architecture with respect to a new task – that of number comparison. Specifically, we aim to evaluate whether the DT architecture can learn and then generalize the task $\mathbf{x} < \mathbf{x}[0]$ for a real-valued random vector $\mathbf{x}$. In order to evaluate the architecture, we train and tune a DT model on the number comparison task using a set of

real-valued random vectors of a fixed length, and then evaluate the model's performance on vectors of significantly longer length than those it was trained on.

Related Work

The papers that precede our work focus on recurrent networks, specifically in formulating new algorithms and the resulting advantages over feed-forward networks. Previous work by Schwarzschild et al. in 2021 and other similar papers, such as Graves et al. in 2014 and Kaiser & Sutskever in 2015, all strive to create recurrent networks that can learn processes from sets of data, including both input/output pairs and code trace training. Such work has shown the ability to generalize across the length of the training input.

One important advantage to the newest generation of models is their ability to separate the amount of computation needed for a particular task from the length of the input itself. Work such as Cai et al. (2017) uses recursion to separate these concepts, allowing for the model to determine the amount of "thinking" needed for a particular input. However, these algorithms still restrict their training and testing data to come from the same distribution. Works such as those by Nuamah (2021) and Schwarzschild et al. (2021) both develop architectures outside of the network itself, respectively hybrid and weight-sharing architectures, to enable generalization to test sets containing samples that require additional levels of computational complexity to be implemented on the spot.

Methods

Our dataset consists of a thousand samples of length-32 vectors, all with elements valued between -0.5 and 0.5, with the labels of equal length generated by performing a boolean comparison on whether $\mathbf{x}[i] < \mathbf{x}[0]$ for each i in a particular sample x. At each index of each sample, a value is picked from a uniform random distribution with range [-0.5,0.5). This range was chosen to ensure the mean of the distribution remained 0. The intention of this data generation method was to create a randomized and synthetic dataset with a constant but sample-dependent rule that could be learned by an algorithm. Learning this rule would show that a model could learn a function more complex than a single constant data-independent function to apply. One flaw in our dataset is that this comparison rule requires that the first index in any label to always be 0. While resolving this by removing this index would be simple, we included this index as a clue that the first data point in any vector contained extra information relative to other points. As shown in Figure 1, one possible sample vector could contain any real value between [-0.5, 0.5), with the index i of the label containing a boolean indicating whether the value at that index in the sample is less than the value at the 0th index in the sample.

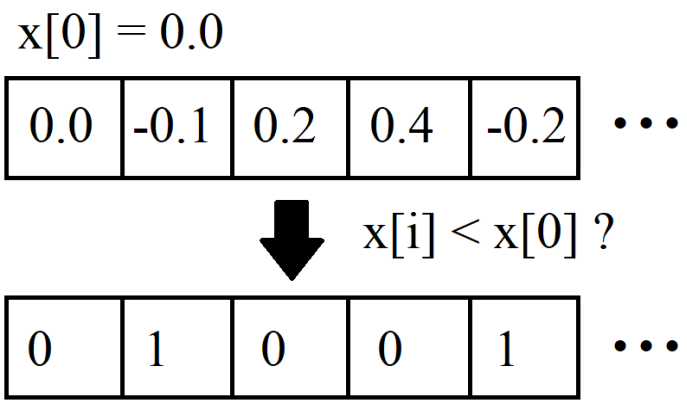

Figure 1: An illustration of the number-comparison task.

In order to evaluate the task $\mathbf{x} < \mathbf{x}[0]$, we employ a DT architecture based on the work done by Schwarzchild et al. (2021) in evaluating prefix sums, since the prefix sums task resembles our number-comparison task more closely than the maze or chess tasks. The architecture consists of three parts: a projection, a series of recall layers that are evaluated in a recurrent fashion, and a head, as depicted in Figure 2.

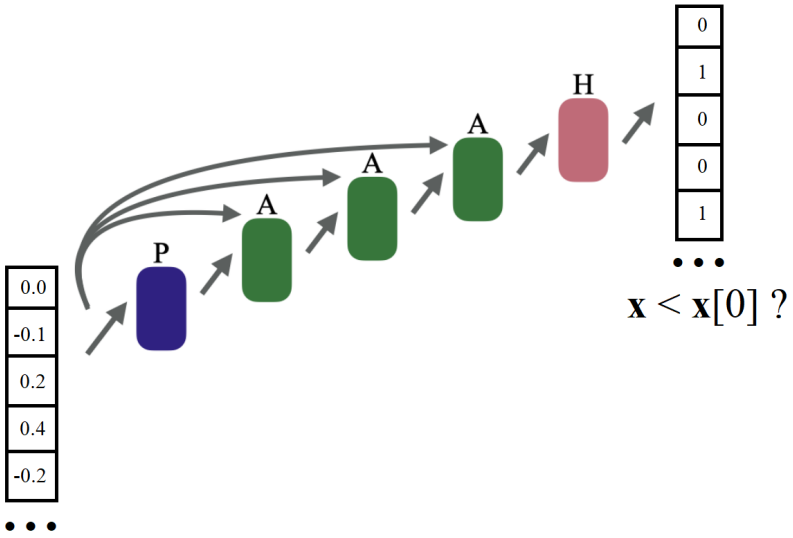

Figure 2: An illustration of the DT architecture in the context of number-comparison; P is the projection, A is the repeated series of recall layers, and H is the final head.

The projection consists of a convolutional layer of width w that projects the input vector **x** to a vector of length w. It pads by 1, has a stride of 1, and has a kernel size of 3. The output of the layer is then passed through a rectified linear unit (ReLU). Another convolutional layer, similar to the first, connects the output of the ReLU with the intermediate series of blocks.

The recall layers begin with a convolutional layer similar to the projection's convolutional layer. The output of the convolutional layer undergoes a ReLU activation function,

and then goes through 2 blocks. Each block consists of a convolutional layer of width w, a group normalization (using 4 groups and no learnable affine parameters), another convolutional layer, another group normalization (Wu and He, 2018), and a shortcut. The input to the recall layers consists of the prior output (either from the projection or from the previous iteration of recall layers) concatenated with the original input vector **x**, so as to prevent the model from distorting information as the number of iterations increases (Bansal et al., 2022). The number of iterations in the model corresponds to how many times the recall layers are called recurrently.

The final head consists of three convolutional layers, each with ReLU activation functions. The first is of width w, the second projects from width w to width $^{w}/_{2}$, and the last projects from width $^{w}/_{2}$ to width 2. The final output has 2 channels because the values of each channel are passed through a softmax function at inference time in order to estimate the probability of whether the value is less than **x**[0].

Experiments

Using the DT architecture, we proceeded to tune the hyperparameters of the network by examining the accuracy and loss of our network on the number-comparison task with length-32 samples. Our training set consists of 8,000 samples, while our validation set consists of 2,000 samples. Once we had sufficient accuracy on our training and validation data, we proceeded to evaluate our model's performance on 10,000 samples of length-128 vectors. We also evaluated the performance of a feedforward net on the same distribution of training, validation, and testing data, using the same hyperparameters as that of the DT model.

During training, we used an Adam optimizer along with various modifications to the training process as specified by Bansal et al. (2022): gradient clipping, progressive training (i.e., training the model to work towards an accurate solution given an intermediate solution), and a 10-epoch exponential warmup. The final hyperparameters we used were:

- Batch size = 100 samples
- Number of training epochs = 150
- Initial learning rate = 0.001
- Step learning rate decay by a factor of 0.1 at epochs 60 and 100
- Width w = 400
- Number of iterations during training = 15

Results

As specified in the original DT net publication, the loss of our process is the average cross-entropy loss, and the accuracy of our process is the number of vectors for which the solution is guessed entirely correctly; every imperfect solution is considered wrong for the purposes of accuracy. Using our trained DT and feedforward models on our training, validation, and test sets, the accuracy of our models is shown in Figure 3. Our DT model achieved 94%

accuracy on its training set, 90% accuracy on its validation set, and a maximum accuracy of 17% on the test set once its number of recurrent iterations was increased from 15 to 65. The feedforward model had an accuracy of 90% on its training set, 85% on its validation set, and 1% on its testing set.

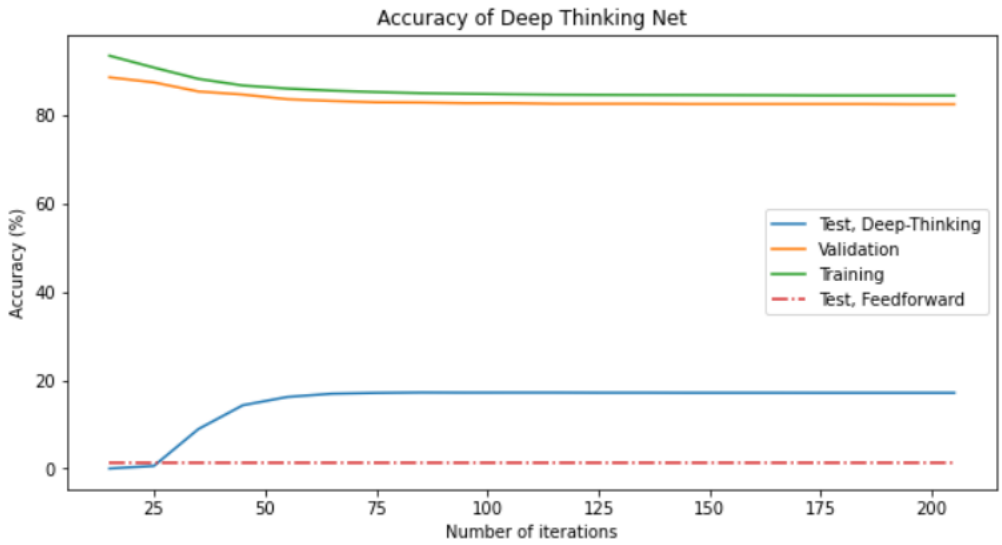

Figure 3: Results of Deep-Thinking Architecture on Number Comparison Task

While our DT model did not achieve the same level of accuracy on more difficult tasks as the original DT model did under Bansal et al. (2022), there is a clear improvement in test accuracy between the DT net and the feedforward net; the latter has no mechanism by which to accommodate more complex inputs via increasing model depth. Interestingly, the DT net suffered from some overthinking at inference time when increasing the number of iterations—however, the recall structure and aforementioned additional training modifications (Bansal et al., 2022) successfully keep the DT model from collapsing when using significantly more iterations than it was trained with.

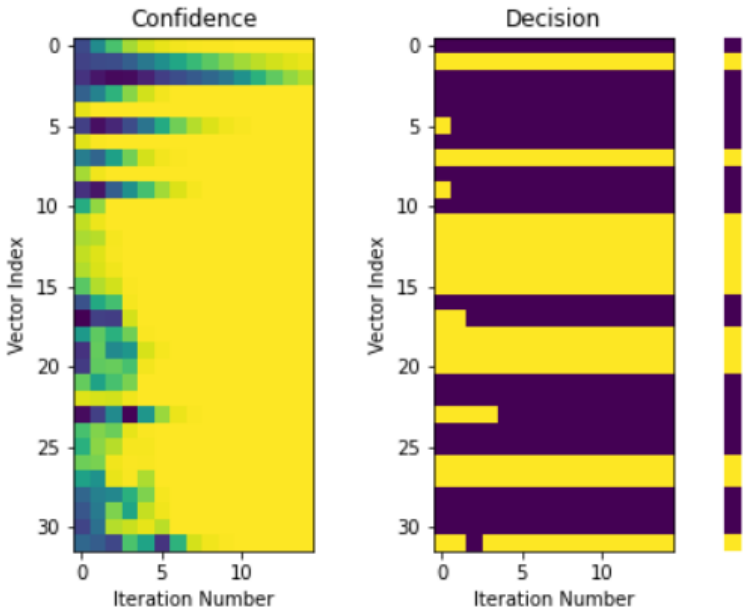

Figure 4: Deep-Thinking Architecture's Evaluation of a 32-Length Vector with **x**[0] = -0.0955

In addition to measuring how the DT model performs, we also seek to visualize how it works. Figure 4 contains the decisions and the decision confidences of our DT net, as functions of the net's 15 iterations. The vector indices span all integers in [0, 31], since the vector **x** in this case is of length 32. In this vector **x**, the first element **x**[0] equals -0.0955. Thus, since all elements of **x** are distributed uniformly between -0.5 and 0.5, approximately 40% of the elements of **x** should be less than **x**[0]. Accordingly, 13 of the 32 elements were less than **x**, since $^{13}/_{32}$ approximates to 41%, which is close to the expected value of 40%.

Furthermore, the confidence values in Figure 4 are each real-valued floats within [0, 1], with 1 being depicted as yellow, and darker colors corresponding to values close to 0. As would be expected, the DT net's inference confidences begin low, but then increase over time, eventually converging to strong confidences close to 1 for each element in the output.

Each decision value in Figure 4 is either 0 (dark-colored) or 1 (yellow), representing the DT net's inference concerning whether the value at the associated vector index is less than **x**[0]. The correct decision pattern is plotted by itself next to the decision plot for reference. In this case, the model correctly predicted the function **x** < **x**[0] for all 32 values of **x**.

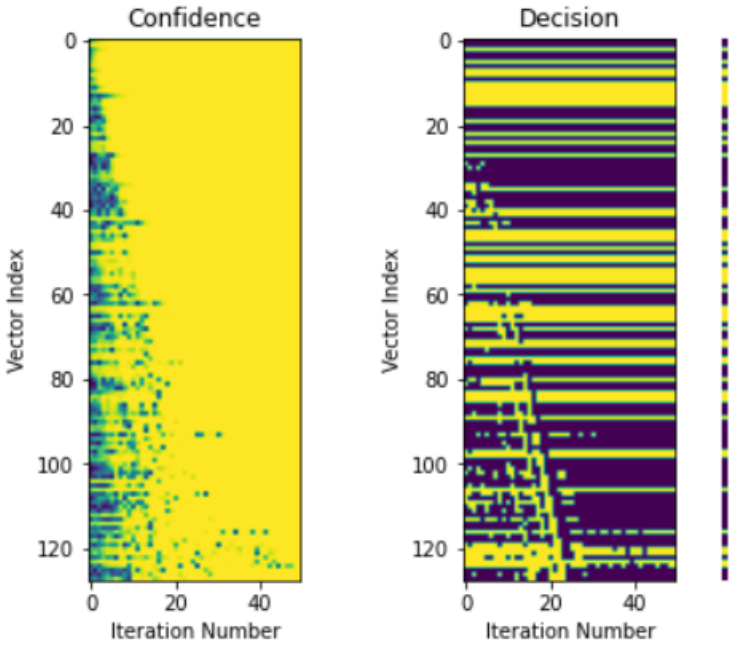

Figure 5: Deep-Thinking Architecture's Evaluation of a 128-Length Vector with **x**[0] = -0.1019

As in Figure 4, Figure 5 contains the inferences of the DT net on a test vector of 128 elements, along with the confidences of its inferences, as a function of its 50 iterations. In the case shown in Figure 5, **x**[0] = -0.1019, meaning that approximately 40% of the elements of **x** are less than **x**[0]. In this case, the model correctly predicted the function **x** < **x**[0] for all 128 values of **x**. Very interestingly, the model has a form of propagation, first predicting the early elements of **x** < **x**[0] and then proceeding in an iterative fashion to predict the other elements of **x** < **x**[0]. Other decision plots for test cases where the predicted function **x** < **x**[0] look similar, albeit with a few errors that result in the proposed solution being deemed incorrect.

Conclusion

Our DT model performed well on its training and validation data, and was able to learn a variation of the algorithm **x** < **x**[0], albeit a slightly spurious one. The DT model performed significantly better than a standard feedforward model due to its ability to engage in algorithmic processing at a level of complexity adjustable for the complexity of the presented task. While the task **x** < **x**[0] seems to ultimately be mismatched with the deep-thinking method we explored (perhaps due to the more iterative nature of the task compared to maze navigation and number-sorting), our experiments were helpful in understanding the limitations of novel architectures, such as the DT architecture. Our findings contribute to the growing body of literature showing that neural networks have the potential to think algorithmically about the tasks presented to them, similar to how humans engage in algorithmic reasoning.

References

Bansal, A. et al. End-to-end Algorithm Synthesis with Recurrent Networks: Logical Extrapolation Without Overthinking. ArXiv, 2022. doi=10.48550/ARXIV.2202.05826.

Cai, Jonathon, Richard Shin, and Dawn Song. "Making neural programming architectures generalize via recursion." arXiv preprint arXiv:1704.06611 (2017).

Graves, et al. Neural turing machines, 2014.

Kaiser, Sutskever. Neural gpus learn algorithms. arXiv preprint arXiv:1511.08228, 2015.

Nuamah, Kwwabena. Deep algorithmic question answering: Towards a compositionally hybrid AI for algorithmic reasoning. arXiv preprint arXiv:2109.08006, 2021.

Schwarzchild, A. et al. Datasets for studying generalization from easy to hard examples, 2021

Schwarzchild, A. et al. Can You Learn an Algorithm? Generalizing from Easy to Hard Problems with Recurrent Networks. *ArXiv*, 2021. doi=10.48550/ARXIV.2106.04537.

Wu, Y., and He, K. Group Normalization. *ArXiv*, 2018. doi=10.48550/ARXIV.1803.08494.