Controlling Neural Network Smoothness for Algorithmic Neural Reasoning

Anonymous Author(s)
Affiliation
Address
email

Abstract

The modelling framework of *neural algorithmic reasoning* [1] postulates that a continuous neural network may learn to emulate the discrete reasoning steps of a symbolic algorithm. The purpose of this study is to investigate the underlying hypothesis in the most simple conceivable scenario – the addition of real numbers. We find that two layer neural networks fail to learn the structure of this task and that growing the network's width leads to a complex division of input space. This behaviour can be emulated with Gaussian processes using radial basis function kernels of decreasing length scale. Classical results establish an equivalence between Gaussian processes and infinitely wide neural networks. We demonstrate a tight link between the scaling of a network weights' standard deviation and its effective length scale on a sinusoidal regression problem, suggesting simple modifications to control the smoothness of the function learned by a neural network. This provides a partial remedy to the brittleness of neural network predictions. We validate this further in the setting of adversarial examples where we demonstrate the gains in robustness that our modification achieves on a standard classification problem of handwritten digit recognition. In conclusion, we show inherent problems of neural networks emulating even simple algorithmic tasks which, however, may be partially improved with smoothness priors inspired by Gaussian processes.

1 Introduction

The two most prominent paradigms in artificial intelligence research are discrete, symbolic algorithms on the one side, and continuous, neural information processing systems on the other [2]. While the latter have caused a revolutionary transformation of the field, they are often plagued by hard challenges, such as robustness to changes in the input distributions, for which algorithmic approaches can provide worst-case performance guarantees. Crucially, we know that both approaches are deployed by humans, akin to Kahneman's 1 and 2 reasoning systems [3]. Thus, algorithms must be implemented in biological neural networks in the human brain. Observing a scene in the world, we know that it is represented and processed in the distributed representation of neural activity in visual cortex. However, the same scene is also represented when we describe it with the use of symbols and the syntax of our language. Thus, one of the most mysterious questions in neuroscience as well as in artificial intelligence research is: *where and how do these two representation systems interact?*

The concept of neural algorithmic reasoning [1] is a recent proposal for a modeling framework at the intersection between symbol processing algorithms and continuous distributed information processing systems [see also 4, 5]. The obvious question is how such a hybrid architecture may be trained, since we usually require differentiability of the whole system for end-to-end training. To solve this, the authors [1] propose training a neural network to approximate the output of the algorithm in the middle of the model and allows end-to-end training.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

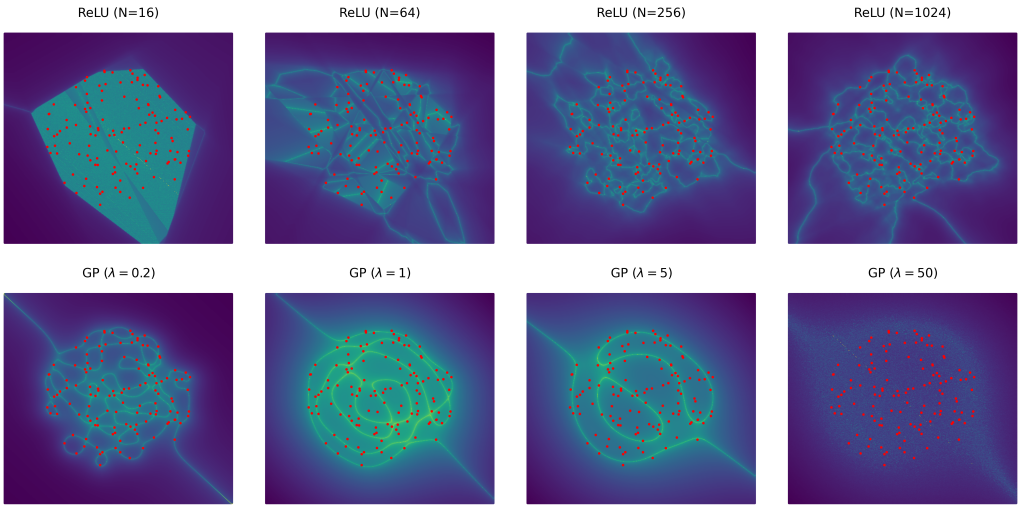

Figure 1: **Unit Disc Loss Surfaces**. These plots show the network error in input space (remember, we are trying to model $f : \mathbb{R}^2 \to \mathbb{R}, f(x) = x_1 + x_2, x \in \mathcal{D} \subseteq \mathbb{R}^2$), where brighter regions indicate lower error (red dots — training data points). Top row shows the learned function and its incurred loss for a ReLU neural network with increasing number of hidden units (N, left to right). Bottom row shows the loss surface for a GP with a RBF kernel of increasing length scale (λ).

The purpose of our work is to investigate the feasibility of neural algorithmic reasoning in one of the most simple conceivable settings: the addition of real numbers. Integer calculus and similarly floating-point arithmetic in binary (symbolic) representations have previously received more attention [6–9]. By contrast, we want to know if a simple (2-layer) neural network can learn to add real-valued numbers on a compact domain such as the unit disc. This is an interesting setting because we know the hypothesis class of neural networks with rectified linear unit (ReLU) activation functions σ trivially contains the correct solution, i.e., given inputs $x \in \mathbb{R}^2$, the following function is a perfect representation of the desired output $y = x_1 + x_2$

$$f(x) := W_2\sigma(W_1x + b_1) + b_2, \quad W_1 := \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}, \quad W_2 := \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}, \quad b_1, b_2 := \mathbf{0}.$$

Given input and output training pairs, a neural network can learn to approximate this function. However, it is an open question how accurate the approximation will be within the domain of the training data; possibly in the limit of infinite data and a model that is a universal function approximator [10]. Secondly, it is unclear how the model will generalise outside the domain of the training data.

In this paper, we find that artificial neural networks are unable to learn this simple function, even if abundant training data is available, leaving regions of high error within the training domain and struggling to extrapolate beyond. A comparison to Gaussian processes suggests a simple partial remedy, exploiting classic results about the equivalence between these two model classes, based on a correct adjustment of the smoothness of the learned function. We show that these modifications also translate to increased adversarial robustness on handwritten character image recognition.

2 Neural Networks and Gaussian Processes Learning Addition

We first investigate learning addition of real numbers in two dimensions. For this, we randomly draw 128 points on the unit disc and train a two layer neural network with ReLU nonlinearities after the first layer to solve the addition task using a simple squared loss function $\mathcal{L}(f, x, y) = (f(x, y) - (x + y))^2$. We train for $50,000$ steps with the Adam optimizer [11] (held fix after initial experiments indicating that this sufficed for convergence), an initial learning rate of 0.001 and a learning rate decay of 0.9. Across the top row of Fig. 1 we change the number of hidden units (i.e., the width) of the network and observe the effect on the learned solution. Few units exhibit the recently proposed polytope structure of neural network approximated functions [12]. While more units slightly improve the performance, we can see that the model uses the additional capacity to cut up the input space into

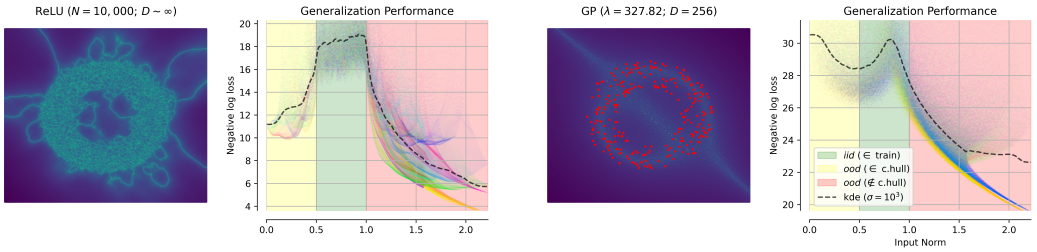

Figure 2: **Annulus Loss Surfaces and Generalisation Performance**. The first (third) column shows the loss surfaces for a NN (GP) with N hidden units, and an unlimited amount of training data (with optimal length scale and D training data points — red dots). The second (fourth) column shows the negative log loss as a function of eccentricity (dots coloured by angle, black dotted lines radially averaged) in the three regimes of OOD (within convex hull), IID and OOD (outside of convex hull) generalisation.

increasingly refined regions. Interestingly, the network learns intricate ridges of good performance on which the training data lies. These ridges appear to be connected on continuous paths — an intriguing observation for future foundational NN research.

Importantly, these patterns suggest that larger ReLU networks learn sieve-like solutions of increasing resolution, but they do not enter a qualitatively different regime that would resemble the simple algorithmic solution to the task. We can produce a similar sequence of model behaviours with Gaussian process (GP) regressors with varying radial basis function (RBF) kernel scales. We use the standard GP implementation in sklearn [13]. Usually, this would include maximum likelihood length scale selection (optimum near $\lambda \sim 150$), however, setting this by hand lets us visualise the different solutions for *suboptimal* length scales. Specifically, we see that setting the length scale too low (Fig. 1 bottom left) forces the model to learn ridges of good solutions through the training data — similar to the NN model. Note that the anti-diagonal line passing through the origin indicates the null space of the target function, i.e., where a 0 output is the correct answer.

The similarity between the solution patterns for neural networks with many hidden units and GPs with short length scales is not surprising given classical results [14]. Briefly, Neal established that for NNs with a hyperbolic tangent (TanH) nonlinearity and an infinite number of hidden units (with appropriate scaling of their initialisation variances) the distribution over learned functions (*ab initio*) becomes equivalent to that of a GP. This opens an interesting path forward in understanding and improving the neural network solution exposed in this section (see below).

3 Out of Domain Generalization

A crucial difference between algorithms and neural network solutions is the way they generalise to different inputs [2]. Addition is defined on all numbers in $\mathbb{R}$, but the approximation learned by a NN can only observe a subset of those inputs in its training data (i.e., *IID* — independent identically distributed). Recent discussions have investigated this from the point of view of interpolation versus extrapolation [15, 16]. Overparameterized NN exhibit a *double descent* phenomenon, which is thought to improve their generalization performance by interpolating between training points [17] — although see [18]. Going beyond the convex hull of the training data would, by contrast, require the ability to extrapolate to a new domain (*OOD* — out of distribution). We adjust our setting slightly to study both aspects of generalisation.

Specifically, the training data is now randomly sampled from the annulus $\mathcal{A} := \{x \in \mathbb{R}^2 \,|\, 0.5 \leq \|x\| \leq 1\}$. Moreover, to assess that the findings from Fig. 1 do not depend on limited training data or finite network size, we set the number of hidden units to $10,000$ and generate a new random batch for every gradient step (totalling $256 \times 50,000 = 12,800,000$ training examples). The learned solution by the NN is shown in Fig. 2 top left. Again, we see an intricate pattern emerging within the training data domain (IID) with ridges of good performance but valleys of bad predictions. By contrast, the predictions are bad both inside the convex hull of the training data $\|x\| \leq 0.5$ as well as outside $\|x\| \geq 1.0$. This is quantified in Fig. 2 top middle as a function of the input norm (we can also see the different ridges outside the training domain distinguished by their angles). For the GP, even with limited training data ($D = 256$), we see higher performance levels (Fig. 2 bottom middle) both on training data (green), in its convex hull (yellow) as well as outside (red).

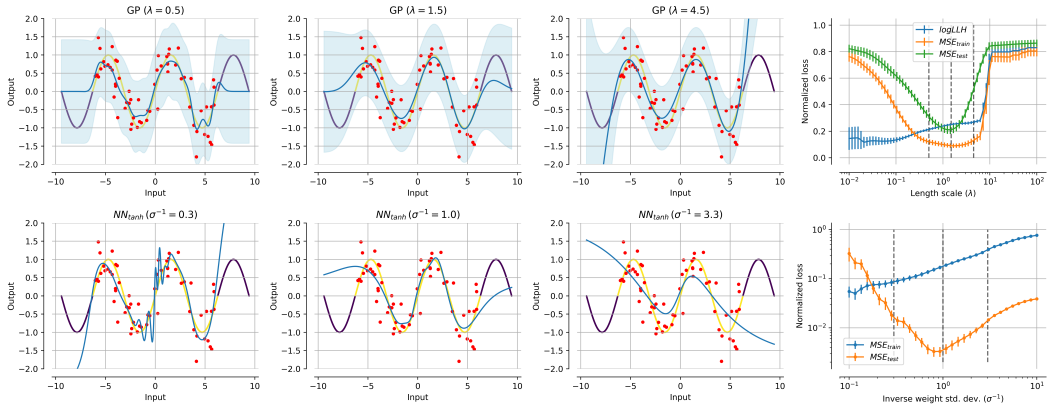

Figure 3: **Sinusoidal Regression**. Top row shows GP models with different length scales (λ) fitted to a sinusoidal regression problem with: true function (yellow IID, purple OOD), training points (red), model prediction (light blue), uncertainty (shaded, two standard deviations). The top right plot shows the different losses (mean squared error — MSE, and log likelihood) as a function of GP RBF length scale. The bottom row gives the same regression plots for the NN with TanH activation function and varying length scales (σ^{-1}) and (right) the training and test error as a function thereof.

4 Setting the Length Scale of Neural Networks

An important step in fitting a GP with a RBF kernel to data is finding the best length scale for the kernel. We explore this process, and its equivalent in NNs, in this section. To easily visualise the learned input-output mapping, the dataset is now just a noisy sine function on $x \in [-2\pi, 2\pi]$ (for OOD, we extrapolate to $[-3\pi, 3\pi]$). Precisely, we are trying to model the function $y = \sin(x) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, 0.25)$. In Fig. 3 we can see that there exists a sweet spot for the kernel's length scale (top middle), and that the model overfits (top left) for smaller length scales and underfits (top right) for larger length scales. This is confirmed quantitatively (top right) by looking at the likelihood (blue) as well as the training and test error as a function of the length scale (dotted lines indicate length scales in plots to the right).

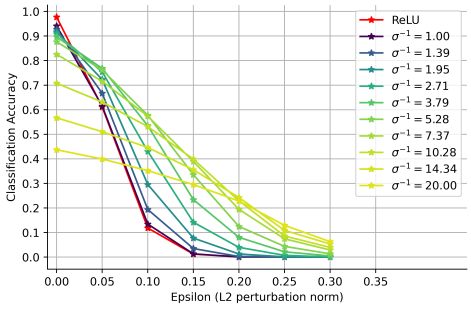

Figure 4: **Adversarial Robustness**. Accuracy on MNIST as a function of (L_2) adversarial perturbation size for different models (TanH NNs, length scale σ^-1).

To get a NN to behave like a GP, we closely follow the construction in [14]. That is, we use TanH and restrict the standard deviation of the weights to $\sqrt{N}$ where N is the number of inputs to a layer. Importantly, we enforce this standard deviation throughout training by using the scaled weights $\tilde{w} = w(s.d.(w)\sqrt{N})^{-1}\sigma$ with $sd(w)$ the standard deviation of w and σ a scaling factor. Intuitively, σ^{-1} behaves like the length scale in GPs (Fig. 3 bottom), i.e., a smaller σ^{-1} means larger weights and more overfitting to the training data whereas too large σ^{-1} means very small weights and underfitting of the training data. Again, we establish the existence of an optimum length scale σ^{-1} (Fig. 3, bottom right) that produces the smallest test error.

5 Controlling Neural Network Smoothness

Returning to the initial two examples of adding real numbers from a disc or an annulus, we can now observe the effect of varying the length scale (σ^{-1}) of a 2-layer NN with a large number of hidden units (approaching the GP regime) and TanH activation functions. On both datasets (Fig. 5) it is apparent that increasing the length scale makes the learned output function more smooth. Thus, the NN becomes qualitatively more similar to a GP with a well adjusted learning scale for its RBF kernel.

As a further verification for the claim that this approach effectively controls the smoothness of the learned NN input-output mapping, we turn towards a more complicated problem of performing image

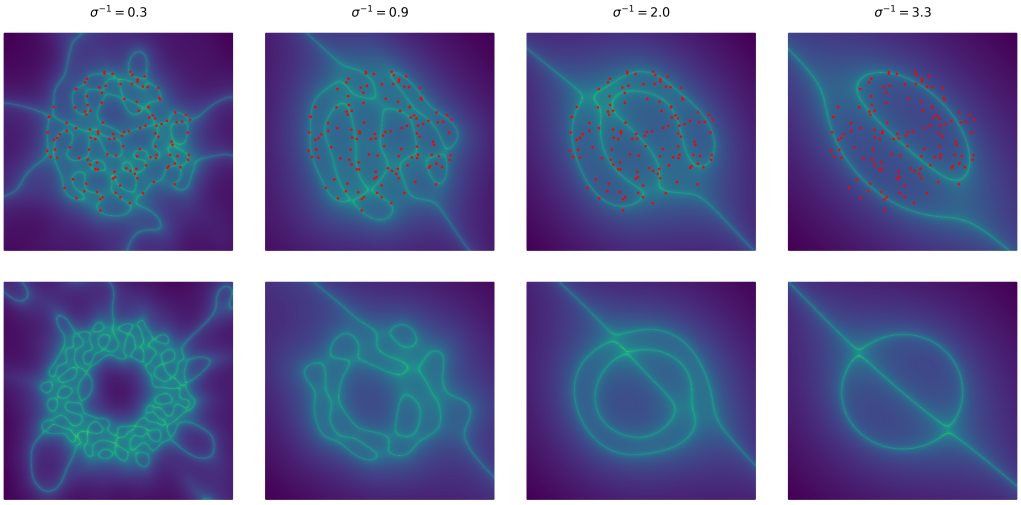

Figure 5: **Controlling the Smoothness of Neural Network Functions**. The four columns show learned TanH 2-layer NN loss surfaces for the two datasets (top disc, bottom annulus — same settings as above) for increasing length scales (left to right) producing an increasingly smooth mapping.

recognition on MNIST under worst case (adversarial) distribution shifts. Thus, we search for minimal perturbations that maximally change the output, which is a proxy for the model's smoothness [19].

Fig. 4 shows that controlling the length scale of a neural network on handwritten character recognition does indeed increase the robustness to L_2 adversarial perturbations above a simple ReLU baseline model (both 2-Layer MLPs as above). We can see a trade-off between the clean and the robust accuracy with large length scales increasing the robustness while decreasing the clean accuracy. These performance levels are far from the SOTA robust models on this task, however, they eschew the need for (expensive) adversarial training [20]. Thus, this is a proof-of-principle that the smoothness of a NN function can indeed be controlled with the construction proposed in this paper.

6 Discussion

The promise of neural algorithmic reasoning to combine distributed and discrete reasoning systems via differentiable NN approximations is intriguing. Here, we show that the functions learned by neural networks, even in one of the most simple conceivable examples of symbolic manipulation (real addition), is prone to learning a highly complex and varying output mapping that falls short of learning the proper algorithmic target. There are many different perspectives onto this problem. Many transformers are better at manipulating arithmetic expressions [9]. However, looking at the loss surface of the NN solution for a simple 2-layer MLP, revealed an intriguing structure and analogy to GPs that we decided to pursue in this work. Moreover, the intricate structure of the loss surface is an intriguing pointer for future directions in basic NN theory.

We have demonstrated how the NN mapping can be made more similar to that of a well-calibrated GP, specifically, by making it more smooth which also improved adversarial robustness. Note that designing Lipschitz neural networks with bounded smoothness is an open research area [21]. In our case, it is open for future work to investigate how the composite function of more than two layers contributes to global smoothness of a NN, potentially building on recent work extending the NN GP equivalence to multilayer networks [22].

Finally, while this study focused on a technical detail about NN approximations to simple algorithms, the larger question remains still open how NNs can make the inferential step (see Hume's problem of induction) from any finite amount of data to an *infinite look-up table* (see MLST episode 061). Surely, humans are a proof-of-concept that noisy and distributed processing systems (i.e., brains) can implement discrete symbolic algorithms [23]. We hope that future research in this area will benefit more from interdisciplinary approaches that take inspiration across fields.

References

[1] Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7):100273, 2021.

[2] Gary F Marcus. *The algebraic mind: Integrating connectionism and cognitive science*. MIT press, 2003.

[3] Daniel Kahneman. *Thinking, fast and slow*. Macmillan, 2011.

[4] Daniel Bear, Chaofei Fan, Damian Mrowca, Yunzhu Li, Seth Alter, Aran Nayebi, Jeremy Schwartz, Li F Fei-Fei, Jiajun Wu, Josh Tenenbaum, et al. Learning physical graph representations from visual scenes. *Advances in Neural Information Processing Systems*, 33:6027–6039, 2020.

[5] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *Advances in neural information processing systems*, 30, 2017.

[6] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks. *arXiv preprint arXiv:2102.13019*, 2021.

[7] Alon Talmor, Yanai Elazar, Yoav Goldberg, and Jonathan Berant. olmpics-on what language model pre-training captures. *Transactions of the Association for Computational Linguistics*, 8:743–758, 2020.

[8] Chengyue Jiang, Zhonglin Nian, Kaihao Guo, Shanbo Chu, Yinggong Zhao, Libin Shen, and Kewei Tu. Learning numeral embeddings. *arXiv preprint arXiv:2001.00003*, 2019.

[9] Avijit Thawani, Jay Pujara, Pedro A Szekely, and Filip Ilievski. Representing numbers in nlp: a survey and a vision. *arXiv preprint arXiv:2103.13136*, 2021.

[10] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Randall Balestriero et al. A spline theory of deep learning. In *International Conference on Machine Learning*, pages 374–383. PMLR, 2018.

[13] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[14] Radford M Neal. Priors for infinite networks. In *Bayesian Learning for Neural Networks*, pages 29–53. Springer, 1996.

[15] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

[16] Lukas Schott, Julius Von Kügelgen, Frederik Träuble, Peter Gehler, Chris Russell, Matthias Bethge, Bernhard Schölkopf, Francesco Locatello, and Wieland Brendel. Visual representation learning does not generalize strongly within the same domain. *arXiv preprint arXiv:2107.08221*, 2021.

[17] Niladri S Chatterji, Philip M Long, and Peter L Bartlett. When does gradient descent with logistic loss find interpolating two-layer networks? *J. Mach. Learn. Res.*, 22:159–1, 2021.

[18] Randall Balestriero, Jerome Pesenti, and Yann LeCun. Learning in high dimension always amounts to extrapolation. *arXiv preprint arXiv:2110.09485*, 2021.

[19] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[20] Eric Wong, Leslie Rice, and J Zico Kolter. Fast is better than free: Revisiting adversarial training. *arXiv preprint arXiv:2001.03994*, 2020.

[21] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[22] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

[23] Wim Fias, Muhammet Ikbal Sahan, Daniel Ansari, and Ian M Lyons. From counting to retrieving: Neural networks underlying alphabet arithmetic learning. *Journal of Cognitive Neuroscience*, 34(1):16–33, 2021.