Learning Iterative Reasoning through Energy Minimization

Yilun Du [1] **Shuang Li** [1] **Joshua Tenenbaum** [1] **Igor Mordatch** [2]

Abstract

Deep learning has excelled on complex pattern recognition tasks such as image classification and object recognition. However, it struggles with tasks requiring nontrivial reasoning, such as algorithmic computation. Humans are able to solve such tasks through iterative reasoning – spending more time thinking about harder tasks. Most existing neural networks, however, exhibit a fixed computational budget controlled by the neural network architecture, preventing additional computational processing on harder tasks. In this work, we present a new framework for iterative reasoning with neural networks. We train a neural network to parameterize an energy landscape over all outputs, and implement each step of the iterative reasoning as an energy minimization step to find a minimal energy solution. By formulating reasoning as an energy minimization problem, for harder problems that lead to more complex energy landscapes, we may then adjust our underlying computational budget by running a more complex optimization procedure. We empirically illustrate that our iterative reasoning approach can solve more accurate and generalizable algorithmic reasoning tasks in both graph and continuous domains. Finally, we illustrate that our approach can recursively solve algorithmic problems requiring nested reasoning. Code and additional information is available at https://energy-based-model.github.io/iterative-reasoning-as-energy-minimization/.

1. Introduction

Human thinking is often characterized in terms of mechanisms for two distinct modes of cognitive processing: SYSTEM 1 mechanisms for fast, habitual, and associative processing, and SYSTEM 2 mechanisms for slower, more deliberate and controlled, symbolic reasoning (Kahneman, 2011). Neural networks have excelled at habitual SYSTEM 1-style processing in familiar environments and task contexts, such as mapping images of familiar objects to their semantic classes, or familiar locations to the corresponding routes of movement. However, confronted with a novel environment or task demanding a more flexible response, humans can invoke controlled SYSTEM 2 processes, often in the form of iteratively reasoning about relationships between observed entities that builds on past experiences primarily through shared abstractions and algorithms rather than direct re-use of concrete, habitual responses. Such flexible processing of novel inputs is difficult for neural networks, with even large pretrained language models (Radford et al., 2019) that have proven effective in many zero-shot generalization contexts failing to extrapolate the simplest algorithmic operations, such as addition, to more complex inputs.

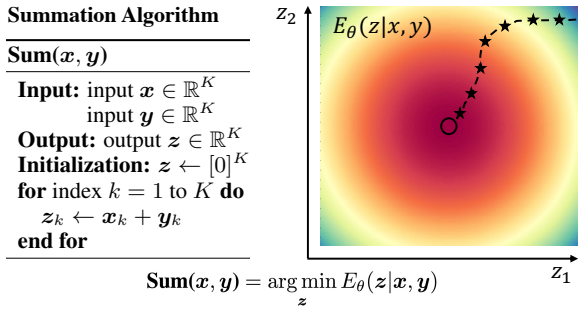

Figure 1. **Reasoning as Energy Minimization** – IREM formulates an algorithmic reasoning problem with inputs $\boldsymbol{x}, \boldsymbol{y}$ and output $\boldsymbol{z}$, such as summation (**left**), as an iterative energy minimization problem over a learned energy function $E_\theta(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})$ which parameterizes an energy landscape over all possible outputs $\boldsymbol{z}$ conditioned on inputs $\boldsymbol{x}, \boldsymbol{y}$ (**right**). We visualize the first two dimensions $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$. Each iteration of energy minimization is shown by a star.

Iterative reasoning, the ability to repeatedly apply an underlying computation to the outputs of previous reasoning steps, is a crucial component of scalable SYSTEM 2-style processing. We may take abstractions learned from simpler variants of a problem, and iteratively apply them to solve harder variants of potentially unbounded complexity. As an example, having learned how to compute shortest paths on small graphs, we might generalize and apply our algorithmic intuition sequentially to substantially larger graph

[1]MIT CSAIL [2]Google Brain. Correspondence to: Yilun Du <yilundu@mit.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

problems at test time through iterative application of learned computations. Such iterative processing enables humans to solve more challenging tasks, such as question answering, arithmetic calculation, or proof writing, even in novel or unfamiliar domains.

We present a new neural approach towards iterative reasoning, which we formulate as an energy minimization process on a learned energy landscape (Figure 1). By representing individual steps of reasoning as an optimization process, we may iteratively reason for longer on harder problems by running additional steps of optimization on the induced energy landscapes. Simultaneously, by monitoring the geometric energy landscape surrounding an optimized solution, we may automatically determine the completion of the algorithmic computation (by checking the presence of a local energy minimum). We refer to our underlying reasoning framework as Iterative Reasoning as Energy Minimization (IREM).

To evaluate and benchmark the effectiveness of iterative reasoning using IREM, we propose and construct a suite of different algorithmic reasoning tasks on both graph and continuous domains. Effective algorithmic reasoning requires repetitive application of underlying algorithmic computations, dependent on problem complexity, and thus serves as a natural benchmark for iterative reasoning. We compare IREM with past works on our algorithmic benchmark for iterative reasoning and find that IREM outperforms prior works in performance and generalization.

Our contributions in this paper are threefold. First, we present IREM, a new framework for iterative reasoning and analyze why it is beneficial to use such a framework for reasoning. Second, we present a benchmark for iterative algorithmic reasoning, both on graphs and continuous vector inputs, and show that IREM significantly outperforms prior approaches in both performance and generalization. Finally, we show how our approach can recursively solve algorithmic computation requiring nested reasoning. Our results point to IREM as a promising new approach towards iterative reasoning.

2. Related Work

Iterative Reasoning A variety of recent works have explored the integration of iterative reasoning into neural networks. One branch of works implements iterative reasoning by constructing neural programmatic operations (Graves et al., 2014; Reed & De Freitas, 2015; Banino et al., 2021) which are repeatedly executed till halting. Another branch of work implements iterative reasoning through recurrent computation (Graves, 2016; Bolukbasi et al., 2017; Chung et al., 2016; Schwarzschild et al., 2021). A key challenge with both types of approaches lies in the halting time of computation. Existing approaches learn halting policies through reinforcement learning (Chen et al., 2020; Chung et al., 2016), heuristic policies (Bolukbasi et al., 2017), or variational inference (Graves, 2016; Banino et al., 2021). Such approaches are unstable in nature (Banino et al., 2021), and many of them require manual hyper-parameter specification. We present an orthogonal approach towards implementing iterative reasoning as an energy minimization procedure on a learned energy landscape. By determining when a local energy minimum has been found, our approach provides a natural mechanism for terminating computation.

Algorithmic Reasoning with External Memory Several approaches towards iterative reasoning utilize an external memory scratchpad for algorithmic computation. Such a scratchpad enables models to store intermediate algorithmic computations, and thus boosts the underlying performance of the algorithm (Graves et al., 2014; Reed & De Freitas, 2015; Cai et al., 2017; Kaiser & Sutskever, 2015). In the Appendix B, we illustrate a manner through which we may utilize an external memory with IREM to improve underlying further improve reasoning performance.

Optimization Based Computational Blocks Prior works have explored optimization as a computation block to solve different tasks. In (Brockett, 1991), a dynamical system is constructed that can solve various algorithmic tasks. Optimization has since been used as an intermediate neural network computation block for quadratic programs (Amos & Kolter, 2017; Donti et al., 2017) and submodular programs (Djolonga & Krause, 2017; Wilder et al., 2019) for flexible neural networks. Most similar to our work, Bai et al. (2019) utilizes equilibrium energy minimization as an intermediate computation block for memory-efficient neural networks. In contrast, we explore how direct optimization over a learned energy landscape can enable generalizable iterative reasoning. Concurrent to our work, (Rubanova et al., 2021) utilizes energy minimization to simulate physical dynamics.

Energy-Based Models Our work is related to works in Energy-Based Models (EBMs) (LeCun et al., 2006). Most recent works using EBMs have focused on learning probabilistic models over data (Du & Mordatch, 2019; Nijkamp et al., 2019; Grathwohl et al., 2020; Du et al., 2021b; Arbel et al., 2020; Li et al., 2020; Xiao et al., 2020). Instead of using EBMs as a probabilistic model, we use EBMs to define an energy landscape for solving iterative reasoning problems.

Learning Optimizers Our work utilizes backpropagation through intermediate optimization steps to train our energy function. Prior work has explored a similar idea of backpropagation through optimization to learn meta optimizers (Andrychowicz et al., 2016; Ravi & Larochelle, 2016; Bengio et al., 1995; Schmidhuber, 1992; Hochreiter et al., 2001;

Li & Malik, 2016) which enable more sample efficient and faster training of neural networks. In this work, we utilize this approach to learn an energy function so that a number of optimization steps on the energy function results in a solution to the algorithmic reasoning problem.

3. Learning Iterative Reasoning through Energy Optimization

Let $\mathcal{D} = \{X, Y\}$ be a dataset of algorithmic reasoning problems consisting of inputs $\boldsymbol{x} \in \mathbb{R}^N$ and corresponding solutions $\boldsymbol{y} \in \mathbb{R}^M$. Our goal is to learn a neural network operator $\text{Alg}_\theta(\cdot)$ which can generalize $\text{Alg}_\theta(\boldsymbol{x}')$ to a test input $\boldsymbol{x}' \in \mathbb{R}^{N'}$, where $\boldsymbol{x}'$ can be significantly larger and more challenging than the training data $\boldsymbol{x} \in X$. We present our approach, IREM, to adaptive algorithmic computation in Section 3.1, and discuss how to learn such an energy landscape in Section 3.2. We further provide analysis on why such a framework is favorable in Section 3.3.

3.1. Iterative Reasoning as Energy Minimization

Iterative reasoning is typically formulated as the repeated application of a neural network operator $f_\theta(\boldsymbol{x}, \boldsymbol{y}) : \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}^M$, to generate partial solutions $\boldsymbol{y}^t$

$$\boldsymbol{y}^t = f_\theta(\boldsymbol{x}, \boldsymbol{y}^{t-1}), \tag{1}$$

where the final prediction $\boldsymbol{y} = \boldsymbol{y}^T$ is output after T successive applications of f_θ. The termination of iterative computation is often difficult to specify, with methods typically utilizing a halting policy that is difficult to train (Graves, 2016; Chung et al., 2016).

In this work, we propose an alternative approach towards iterative computation with a natural termination criteria, IREM, which represents iterative reasoning as an optimization process over an Energy-Based Model (EBM) $E_\theta(\boldsymbol{x}, \boldsymbol{y}) : \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}$:

$$\boldsymbol{y} = \arg\min_{\boldsymbol{y}} E_\theta(\boldsymbol{x}, \boldsymbol{y}). \tag{2}$$

An individual reasoning step is then represented as

$$\boldsymbol{y}^t = \boldsymbol{y}^{t-1} - \lambda \nabla_{\boldsymbol{y}} E_\theta(\boldsymbol{x}, \boldsymbol{y}^{t-1}), \tag{3}$$

where λ is the step size of each gradient descent step. We repeat the iterative computation in Equation 3 until $\boldsymbol{y}^t$ is a local minima of the energy landscape, $E_\theta(\boldsymbol{y}^t) = E_\theta(\boldsymbol{y}^{t-1})$, where additional steps of optimization do not change the energy value of $\boldsymbol{y}^t$. Finding a local minimum of the energy landscape thus provides us with a natural mechanism to halt iterative computation.

3.2. Learning Energy Landscapes for Reasoning

We next discuss how to learn IREM. Given an input problem $\boldsymbol{x}_i$ with a unique solution $\boldsymbol{y}_i$, the simplest method to learn $E_\theta(\boldsymbol{x}, \boldsymbol{y})$ is to directly supervise the minimal energy state $\arg\min_{\boldsymbol{y}} E_\theta(\boldsymbol{x}_i, \boldsymbol{y})$ with $\boldsymbol{y}_i$ through regression

$$\mathcal{L}_{\text{MSE}}(\theta) = \| \arg\min_{\boldsymbol{y}} E_\theta(\boldsymbol{x}_i, \boldsymbol{y}) - \boldsymbol{y}_i \|^2. \tag{4}$$

However, in practice, during training time, it is computationally expensive to compute $\arg\min_{\boldsymbol{y}} E_\theta(\boldsymbol{x}, \boldsymbol{y})$, as the underlying energy landscape may be complex.

As a fast approximation to $\arg\min_{\boldsymbol{y}} E_\theta(\boldsymbol{x}_i, \boldsymbol{y})$, we may approximate the $\arg\min$ operation with respect to $\boldsymbol{y}_i$ via N steps of gradient optimization. An approximate optimum $\boldsymbol{y}_i^N$ is obtained by:

$$\boldsymbol{y}_i^N = \boldsymbol{y}_i^{N-1} - \lambda \nabla_{\boldsymbol{y}} E_\theta(\boldsymbol{x}_i, \boldsymbol{y}_i^{N-1}). \tag{5}$$

In Equation 5, we initialize $\boldsymbol{y}_i^0$ using uniform noise, with λ denoting the step size for each gradient step, and sample $\boldsymbol{y}_i^N$ corresponding to the result after N steps of gradient descent. We then minimize the corresponding loss:

$$\mathcal{L}_{\text{MSE}}(\theta) = \|\boldsymbol{y}_i^N - \boldsymbol{y}_i\|^2, \tag{6}$$

where we may directly differentiate through the underlying optimization procedure (Finn et al., 2017). To alleviate the computational burden of computing second-order gradients across optimization steps, we empirically found that simply truncating back-propagation to the last optimization step maintained good performance at a significantly faster training speed as illustrated in Table 2.

An underlying difficulty of utilizing $\boldsymbol{y}_i^N$ as an approximation of $\arg\min_{\boldsymbol{y}} E_\theta(\boldsymbol{x}_i, \boldsymbol{y})$ is that since only a small finite number of steps of gradient descent is applied, $\boldsymbol{y}_i^N$ may be far from $\arg\min_{\boldsymbol{y}} E_\theta(\boldsymbol{x}_i, \boldsymbol{y})$. As a result, when running a larger number of iterative reasoning steps at test time to more precisely compute $\arg\min_{\boldsymbol{y}} E_\theta(\boldsymbol{x}_i, \boldsymbol{y})$, our underlying solution may degrade.

To remedy this issue, we maintain a replay buffer of previously optimized samples $\boldsymbol{y}_i^N$, and initialize $\boldsymbol{y}_i^0$ either from previously optimized values $\boldsymbol{y}_i^N$ or uniform noise. By initializing gradient descent optimization with previously optimized samples, we ensure that these samples are closer in approximation to $\arg\min_{\boldsymbol{y}} E_\theta(\boldsymbol{x}_i, \boldsymbol{y})$. A similar application of replay buffers has been utilized to train EBMs for consistent probability landscapes (Du & Mordatch, 2019).

We provide pseudocode for training IREM in Algorithm 1 and executing algorithmic reasoning with IREM our approach in Algorithm 2, where an energy minimum is determined at test time once the energy value of a solution does not change for several iterations. A fixed learning rate, $\lambda = 100$ is used for training IREM, with λ at test time empirically tuned so that energy values decrease between iterations of optimization (harder problems require smaller λ to optimize). In Appendix B, we discuss how we may utilize

Algorithm 1 IREM training algorithm

Input: Data Dist $p_D(\boldsymbol{x}, \boldsymbol{y})$, Replay Buffer $\mathcal{B}$, Step Size λ, Number of Steps N, EBM $E_\theta(\cdot)$, Uniform Distribution $U(-1, 1)$
$\mathcal{B} \leftarrow \varnothing$
while not converged **do**
  ▷ *Sample data and candidate solutions from p_d and replay buffer $\mathcal{B}$*
  $\boldsymbol{x}_i, \boldsymbol{y}_i \sim p_D, \tilde{\boldsymbol{y}}_i^0 \sim \mathcal{U}(-1, 1)$
  $\boldsymbol{x}_i^b, \boldsymbol{y}_i^b, \tilde{\boldsymbol{y}}_i^b \sim B$
  $\boldsymbol{x}_i, \boldsymbol{y}_i, \tilde{\boldsymbol{y}}_i^0 \leftarrow \boldsymbol{x}_i \cup \boldsymbol{x}_i^b, \boldsymbol{y}_i \cup \boldsymbol{y}_i^b, \tilde{\boldsymbol{y}}_i^0 \cup \tilde{\boldsymbol{y}}_i^b$
  ▷ *Generate low energy solutions through optimization:*
  for sample step $n = 1$ to N **do**
    $\tilde{\boldsymbol{y}}_i^n \leftarrow \tilde{\boldsymbol{y}}_i^{n-1} - \lambda \nabla_{\boldsymbol{y}} E_\theta(\boldsymbol{x}_i, \tilde{\boldsymbol{y}}_i^{n-1})$
  end for
  ▷ *Optimize objective $\mathcal{L}_{MSE}$ wrt θ:*
  $\Delta\theta \leftarrow \nabla_\theta \sum_{n=1}^{N} \|\tilde{\boldsymbol{y}}_i^N - \boldsymbol{y}_i\|^2$
  Update θ based on $\Delta\theta$ using Adam optimizer
  ▷ *Update replay buffer $\mathcal{B}$*
  $\mathcal{B} \leftarrow \mathcal{B} \cup (\boldsymbol{x}_i, \boldsymbol{y}_i, \tilde{\boldsymbol{y}}_i^N)$
end while

Algorithm 2 IREM prediction algorithm

Input: Data Dist $p_D(\boldsymbol{x})$, Step Size λ, Number of Steps K, EBM $E_\theta(\cdot)$, Uniform Distribution $U(-1, 1)$
▷ *Sample input from p_d and initialize candidate solution*
$\boldsymbol{x}_i \sim p_D$
$\tilde{\boldsymbol{y}}_i \sim \mathcal{U}(-1, 1)$
while Not at Energy Minima **do**
  ▷ *Optimize candidate solution $\tilde{\boldsymbol{y}}_i$ with gradient descent:*
  $\tilde{\boldsymbol{y}}_i \leftarrow \tilde{\boldsymbol{y}}_i - \lambda \nabla_{\boldsymbol{y}} E_\theta(\boldsymbol{x}_i, \tilde{\boldsymbol{y}}_i)$
end while
▷ *Final predicted solution:*
$\boldsymbol{y} = \tilde{\boldsymbol{y}}$

optimization to further incorporate an external scratchpad into iterative reasoning using IREM.

As IREM directly trains an energy landscape by optimizing samples to regress solutions, there is no guarantee that a smooth underlying energy landscape $E(\boldsymbol{x}, \boldsymbol{y})$ is learned. Empirically, as seen in Figure 1 and Figure 5, we find that our objective does lead to consistent energy landscapes similar to past work (Du et al., 2021a).

3.3. Analysis

In this section, we provide complexity-theoretic motivation for representing iterative reasoning as an energy minimization is advantageous. We consider two approaches to represent reasoning:

Feedforward Computation. We consider reasoning as a feedforward function f of the form $f(\boldsymbol{x})$ where $f(\boldsymbol{x}) : \mathbb{R}^M \rightarrow \mathbb{R}^N$ maps a input $\boldsymbol{x}$ to a predicted solution $\boldsymbol{y}$.

Energy Minimization. We next consider reasoning as a energy minimization problem, $\arg\min_{\boldsymbol{y}} E(\boldsymbol{x}, \boldsymbol{y})$, where $E(\boldsymbol{x}, \boldsymbol{y})$ is a function from $\mathbb{R}^M \times \mathbb{R}^N \rightarrow \mathbb{R}$, mapping a input $\boldsymbol{x}$ and solution $\boldsymbol{y}$ into an energy. This corresponds to the computation represented by IREM.

We construct a class of algorithmic reasoning tasks that are easier to learn using an energy function $E(\boldsymbol{x}, \boldsymbol{y})$ as opposed to the feedforward function $f(\boldsymbol{x})$. Our result is based on the intuition that learning the energy function $E(\boldsymbol{x}, \boldsymbol{y})$ corresponds to learning a solution verifier, which assigns minimal energy to the correct solution, and high energy to all other solutions. In contrast, learning a function $f(\boldsymbol{x})$ corresponds to explicitly generating a solution. We rely on the well-known theorem in complexity theory that constructing a solution verifier is easier than a solution generator to show that learning energy minimization is easier than feedforward computation.

To enable formal analysis of $E(\boldsymbol{x}, \boldsymbol{y})$, we consider solving the 3-SAT (Impagliazzo & Paturi, 1999) problem. We construct a energy function $E(\boldsymbol{x}, \boldsymbol{y})$ to verify the 3-SAT formula. Given a 3-SAT formula ϕ with D variables and K clauses, we construct an energy function to represent ϕ as $E(\boldsymbol{x}, \boldsymbol{y}) := \sum_{1 \leq k \leq K} e_k(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ encodes the clauses in a 3-SAT problem, $\boldsymbol{y}$ corresponds to boolean assignments to each variable, and e_k verifies whether the boolean assignments in $\boldsymbol{y}$ satisfies the k^{th} clause. To encode a set of K clauses using $\boldsymbol{x}$, we utilize an ordinal representation (e.g. $\boldsymbol{x}_1 = [1, 2, 3]$ to represent a particular clause $(\boldsymbol{y}_1 \wedge \boldsymbol{y}_2 \wedge \boldsymbol{y}_3)$). The energy function e_i is then constructed by taking in clause $\boldsymbol{x}_i$ and outputting 0 if the corresponding entries of $\boldsymbol{y}$ satisfy the encoded clause and 1 otherwise (polynomial time to compute). We assume the Exponential Time Hypothesis (ETH) (Impagliazzo & Paturi, 1999), which states that checking the satisfiability of a 3-SAT formula takes time exponential in the sum of the number of variables and the number of clauses.

Remark 1. *There exists a 3-SAT problem which may be encoded in an energy function $E(\boldsymbol{x}, \boldsymbol{y})$ which can be evaluated at any input in time polynomial in the number of dimensions of $\boldsymbol{x}$ but for which the computational complexity of encoding a feedforward solution $f(\boldsymbol{x})$ which may be evaluated at any input is (worse-case) exponential in the number of dimensions of $\boldsymbol{x}$.*

Proof. We encode the 3-SAT energy function $E(\boldsymbol{x}, \boldsymbol{y})$ as defined above, which is evaluated in time polynomial in the number of dimensions of $\boldsymbol{x}$. In contrast, ETH directly implies that constructing $f(\boldsymbol{x})$, which corresponds to solving the 3-SAT problem, is exponential in dimension of $\boldsymbol{x}$. □

Our remark shows that it is computationally advantageous to represent an energy function $E(\boldsymbol{x}, \boldsymbol{y})$ as opposed to a feedforward decoder $f(\boldsymbol{x})$. In particular, learning neural networks $E_\theta(\boldsymbol{x}, \boldsymbol{y})$ and $f_\theta(\boldsymbol{x})$ to approximate either $E(\boldsymbol{x}, \boldsymbol{y})$ or $f(\boldsymbol{x})$, our remark implies that a larger network is necessary to represent the exponential computations of $f(\boldsymbol{x})$.

Task	Method	Same Size	Larger Size	Task	Method	Same Diff.	Harder Diff.
Edge Copy	Feedforward	0.3016	0.3124	**Add**	Feedforward	0.0448	0.7029
	Recurrent	0.3015	0.3113		Recurrent	0.3610	2.6133
	Programmatic	0.3053	0.4409		Programmatic	0.0111	0.3446
	Iterative Feedforward	0.6163	0.6498		Iterative Feedforward	0.0144	0.1577
	IREM (Ours)	**0.0019**	**0.0019**		IREM (Ours)	**0.0003**	**0.0021**
Connected Components	Feedforward	0.1796	0.3460	**Matrix Completion**	Feedforward	0.0203	0.2720
	Recurrent	0.1794	0.2766		Recurrent	0.0266	0.3285
	Programmatic	0.2338	3.1381		Programmatic	0.0203	0.2637
	Iterative Feedforward	0.4908	1.2064		Iterative Feedforward	0.0253	0.2102
	IREM (Ours)	**0.1424**	**0.2171**		IREM (Ours)	**0.0183**	**0.2074**
Shortest Path	Feedforward	0.1233	1.4089	**Matrix Inverse**	Feedforward	0.0112	0.2150
	Recurrent	0.1259	0.1083		Recurrent	0.0109	0.2123
	Programmatic	0.1375	0.1290		Programmatic	0.0124	0.2209
	Iterative Feedforward	0.4588	0.7688		Iterative Feedforward	0.0270	0.5250
	IREM (Ours)	**0.0274**	**0.0464**		IREM (Ours)	**0.0108**	**0.2083**

Table 1. **Algorithmic Reasoning with IREM–** IREM is general framework for iterative reasoning which can learn algorithmic computation on both graph (**left**) and continuous (**right**) inputs. IREM generalizes at test time to both larger (left) and harder (right) algorithmic problems through iterative computation. Error reported on each task using elementwise mean square error. Approaches on the left are trained on graphs with ten nodes and evaluated on larger graphs with fifteen nodes. IREM significantly outperforms comparisons.

Such a network $f_\theta(\boldsymbol{s})$ would require either exponentially deeper or wider layers, if there is no iterative computation. With iterative computation, we may parameterize $f_\theta(\boldsymbol{x})$ and $E_\theta(\boldsymbol{x}, \boldsymbol{y})$ with a similar number of parameters, but then it would be necessary for $f_\theta(\boldsymbol{x})$ to be iteratively applied a exponential number of times. Direct recurrent backpropagation through such a number of iterative computations has been proven to be unstable to train. While in principle computing $\arg\min_{\boldsymbol{y}} E(\boldsymbol{x}, \boldsymbol{y})$, would require a similar number of computations, we may train $E(\boldsymbol{x}, \boldsymbol{y})$ with a simple inexact energy minimization procedure and run a more extensive minimization procedure at test time. We next analyze the computational complexity of representing $E(\boldsymbol{x}, \boldsymbol{y})$ and $f(\boldsymbol{x})$ as a function of problem size.

Remark 2. *As the underlying number of variables in 3-SAT problem increases, we may construct $E(\boldsymbol{x}, \boldsymbol{y})$ which can be evaluated at any input in time polynomial in the number of variables, but for which the computational complexity of encoding a feedforward solution $f(\boldsymbol{x})$ which may be evaluated at any input is (worse-case) exponential in the number of underlying variables.*

Proof. Our constructed energy function $E(\boldsymbol{x}, \boldsymbol{y})$ above may be evaluated in time polynomial in the number of input variables. In contrast, ETH implies that constructing $f(\boldsymbol{x})$, which corresponds to solving the 3-SAT problem, is exponential in the number of underlying variables. □

Similar to our previous remark, our result implies that representing $f(\boldsymbol{x})$ requires exponentially more computation as the underlying problem size of a 3-SAT problem increases. In particular, our remark has implications for **generalization**. If we learn $f_\theta(\boldsymbol{x})$ and $E_\theta(\boldsymbol{x}, \boldsymbol{y})$ on smaller problem instances of $\boldsymbol{x}$, for our neural networks to generalize correctly to larger problems instances, the underlying computation executed by the neural network must increase polynomially for $E_\theta(\boldsymbol{x}, \boldsymbol{y})$ and exponentially for $f_\theta(x)$. Existing architectures, such as transformers and graph networks, adaptively increase computation polynomially for larger inputs. In contrast, few architectures can adaptively increase computation exponentially for larger problem instances. To realize $f(\boldsymbol{x})$, we thus require iterative approaches which can execute exponentially longer on larger inputs. As seen in Figure 2, this is difficult for existing approaches.

4. Experiments

4.1. Experimental Setup

We compare IREM with feedforward and iterative baselines for reasoning. We discuss each approach in detail below.

Feedforward Computation. First, we compare with (one-step) feedforward computation, where we train a neural network that directly outputs the values of solutions.

Recurrent Network Computation. Next, we compare our approach with methods utilizing a recurrent neural network to execute iterative computation. Recurrent architectures have been shown to successfully execute reasoning recently (Schwarzschild et al., 2021). We use a LSTM network to represent iterative computation.

Learned Programmatic Computation. We compare our method with past works which construct iterative computation through building programmatic structures with neural networks. We compare our method with the recent architecture and training objective of PonderNet (Banino et al., 2021), which variationally learns both a halting probability and individual computation step networks.

Iterative Feedforward Computation. We further compare our approach with direct iterative application of a feed-

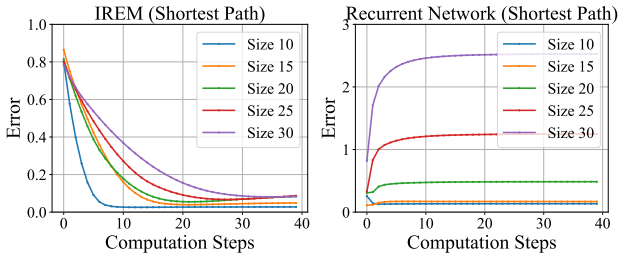

Figure 2. **Computation Steps vs Problem Size** – Error on shortest path computation as a factor of graph size and number of computation executed. Models are trained 5 steps of iterative computation on graph sizes of 10. IREM is able to generalize computation to larger graphs and a greater number of iterative computation steps, while a recurrent network fails to do so.

forward computation. We train the iterative feedforward computation using an iterative denoising objective (Sohl-Dickstein et al., 2015).

We scale network sizes to ensure that each individual baseline has roughly the same number of parameters as other baselines. We provide the architectural details of each model in Appendix D of the paper and utilize MLP neural networks for continuous algorithmic reasoning tasks and graph neural networks for graph algorithmic reasoning tasks. All iterative methods are trained with 5 steps of reasoning. We provide comparisons with each baseline on graphical algorithmic reasoning in Section 4.2 and on continuous algorithmic reasoning in Section 4.3. We provide additional benchmark comparisons of IREM on an existing iterative image denoising task in Appendix A.

4.2. Graphical Algorithmic Reasoning

Setup. We first evaluate our approach on graphical algorithmic reasoning. We train models on fully connected graphs of size 2 to 10, and evaluate performance on larger fully connected graphs of size 15. We report the underlying elementwise mean squared error between predictions from models and their associated ground truth outputs. We evaluate performance on the three different graphical algorithmic reasoning tasks, aiming to capture different aspects of reasoning, which we detail below, with additional details about each dataset in Appendix C.

1. *Edge Copy*: We first test the ability of models to copy and output the values of all edges in a dense graph that is given as input. This task serves as a simple test for iterative reasoning, and requires a method to sequentially copy over input edge values to the output.
2. *Connected Components*: Next, we evaluate the ability of models to infer the underlying connected components of a graph. We construct a sparse graph, where 5% of all fully connected edges exist, and ask models to predict a binary indicator on whether a node of a graph is connected to another for all pairs of nodes in a graph. This task tests structural discovery, an aspect of cognitive reasoning (Kemp & Tenenbaum, 2008).
3. *Shortest Path*: Finally, we evaluate the ability of models to compute the shortest path distances between all pairs of nodes in a graph. This task tests for planning, and the underlying calculation necessary to compute the shortest paths between nodes is analogous to that of planning.

Replay Buffer	Truncate Gradient	Test Performance	Training Speed	Memory Usage
No	No	0.0491	87%	295%
Yes	No	0.0287	83%	295%
Yes	Yes	0.0274	100%	100%

Table 2. **Ablations –** Ablations of proposed components of IREM on test performance on the shortest path algorithmic reasoning task. The use a replay buffer boosts the underlying performance of IREM and truncating gradient backpropogation to the last step of optimization reduces both training time and memory cost.

Quantitative Results. We present quantitative results on graphical algorithmic reasoning in the left column of Table 1. We evaluate on test problems with either similar or larger sizes than seen in training. Across all three tasks, we find that IREM outperforms all compared baselines. This difference in performance is magnified when evaluated on larger test graphs. We found approaches other than IREM struggled even on the relatively simple edge copy task. This is due to the fact that the small size of graph networks (hidden size 128, requires methods to iteratively copy different subsets of input edges to output predictions. While IREM successfully learns this iterative computation, we found that our compared baselines were unable to do so.

Adaptive Computation. Next, we analyze the ability of IREM and baselines to adapt its underlying computational budget to different larger problem instances. In Figure 2, we illustrate shortest path computation error on different input graphs sizes at test time. While all methods are only trained with 5 steps of iterative computation on size 10 graphs, we find that IREM can generalize iterative computation to size 30 graphs. In contrast, we found that learned iterative baselines, such as the recurrent network in Figure 2 failed to do so. We found similar behavior across other graphical algorithmic reasoning tasks.

Ablations. We run an ablation analysis on the impact of utilizing a replay buffer to train IREM, as well as the effect of truncating backpropagation to the last step of optimization. In Table 2, we find that utilizing a replay buffer significantly improves the performance of IREM. We further find that truncating backpropagation to the last step of optimization has a limited impact on IREM and greatly reduces the overall memory cost of training, as well as slightly improving the training speed of IREM.

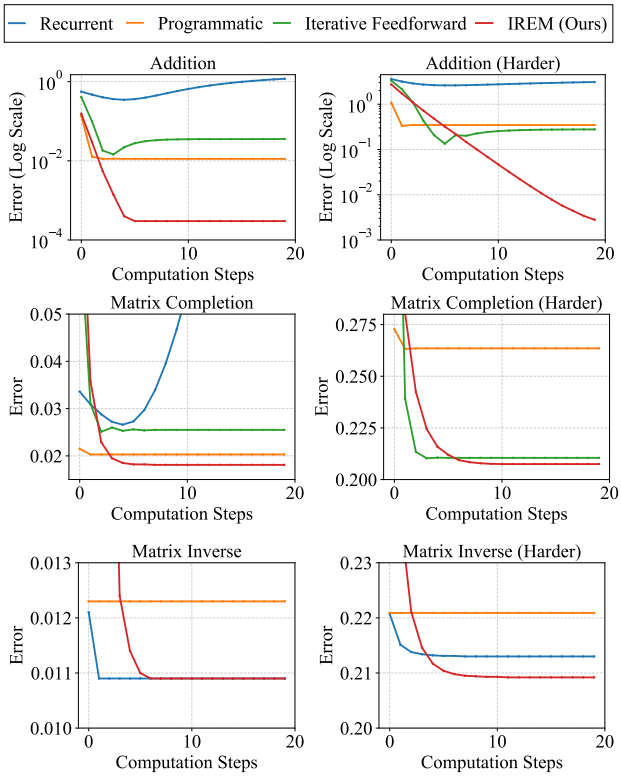

Figure 3. **Computation Steps vs Problem Difficulty** – Illustration of MSE error of prediction as a factor of the test time difficulty (harder difficulty right) of the task and computation steps applied. Each model is trained with 5 steps of iterative computation. Models missing in plots have errors greater than the range displayed in the plot. The error of IREM improves with the number of underlying algorithmic computation steps, with larger number of computation steps benefiting performance on harder algorithmic tasks.

4.3. Continuous Algorithmic Reasoning

Data Setup. We next evaluate IREM and baselines on continuous algorithmic reasoning tasks. We apply algorithmic operations on input vectors of size 400 (resized to 20×20 matrices for matrix operations). We report the underlying MSE error between the predictions and the associated ground truth outputs on test problem instances. We evaluate different methods on the following three tasks, aiming to capture different aspects of reasoning, with additional details in the Appendix C.

1. *Addition*: We first evaluate the algorithmic computation of addition. We train networks to add entries in two separate input vectors (element-wise). We construct harder variants of the addition problems at test time by feeding input vectors with larger magnitudes. This task aims to test simple arithmetic reasoning.
2. *Matrix Completion*: Next, we evaluate the algorithmic computation of matrix completion. We mask out 50% of the entries of a low-rank input matrix constructed from two separate low-rank matrices U and V, and train networks to reconstruct the original input matrix. We construct harder variants of the matrix completion problem at test time by increasing the complexity of U and V. This task aims to test both structural and analogical reasoning, with both being equivalent to matrix completion (Lampinen et al., 2017).
3. *Matrix Inverse*: Finally, we evaluate the algorithmic computation of matrix inverse. We train networks to compute the matrix inverse of an input matrix. We construct harder matrix inverse problems by considering less well-conditioned input matrices. This task aims to test the numerical reasoning, with matrix inversion a crucial operation across various numerical algorithms.

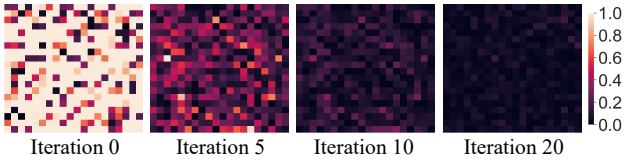

Figure 4. **Qualitative Illustration of Addition** – Illustration of per element error on the addition task as a function of underlying number iterative computation steps run with IREM. Individual inputs gradually approach ground truth values, with different elements approaching zero error at different rates.

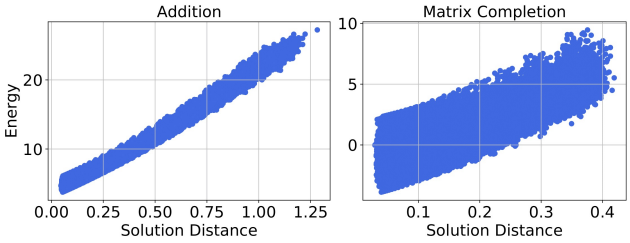

Figure 5. **Energy Landscape** – Plot of predicted energy values for $\boldsymbol{y}$ and the corresponding MSE distance of $\boldsymbol{y}$ from the problem solution. The predicted energy of $\boldsymbol{y}$ correlates well with the distance of $\boldsymbol{y}$ from the ground truth. Plot for matrix inverse is similar to matrix completion.

Quantitative Results. We present quantitative results on each of our three continuous algorithmic reasoning problems on the right side of Table 1 on test problems with either the same or harder difficulty than training problems. Across all three tasks, we find that IREM outperforms baselines, with the underlying difference magnified on harder, out-of-distribution test problems. In particular, on the task of addition, we find that IREM is able to nearly perfectly solve the underlying task even on harder, out-of-distribution test problems. In contrast, all other evaluated iterative and feedforward baselines perform poorly effectively solve the underlying problem. A primary difficulty is that underlying input vectors, both of size 400, are large compared to

$$\begin{bmatrix} -0.695 & -0.840 & 0.691 \\ 0.488 & -0.961 & -0.048 \\ -0.824 & -0.235 & 0.162 \end{bmatrix} + \begin{bmatrix} -0.590 & 0.972 & -0.544 \\ -0.318 & 0.748 & 0.638 \\ -0.216 & -0.118 & 0.864 \end{bmatrix} + \begin{bmatrix} -0.676 & -0.688 & 0.422 \\ 0.075 & 0.172 & -0.963 \\ 0.698 & 0.837 & -0.735 \end{bmatrix} + \begin{bmatrix} 0.945 & -0.432 & -0.940 \\ -0.567 & 0.989 & -0.302 \\ -0.526 & 0.492 & 0.574 \end{bmatrix} = \begin{bmatrix} -1.033 & -1.076 & -0.433 \\ -0.319 & 1.010 & -0.677 \\ -0.867 & 0.984 & 0.843 \end{bmatrix}$$

Figure 6. **Addition Composition** – Illustration of predictions from IREM (in blue) when three separate addition executions are nested together. IREM is applied on vectors with 400 entries – we visualize the first 9 elements of inputs and predictions.

Step Size	Same Difficulty	Harder Difficulty
10	0.0003	0.0021
30	0.0003	0.0020
100	0.0003	0.0021
300	0.0004	0.0023
1000	0.0004	0.0025

Table 3. **Ablation Analysis of Step Size in IREM–** Analysis of training step size of IREM performance on the continuous addition reasoning task.

the underlying hidden unit size of 512, thus requiring the networks to iteratively reason and execute the algorithmic computation on subsets of the input.

Qualitative Visualization. Next, we visualize the underlying iterative computation learned by IREM and baselines. In Figure 3, we illustrate the prediction error of different methods as a function of the number of computation steps applied. As we increase the number of iterative computation steps, the underlying performance of IREM continues to improve. In particular, iterative computation helps more substantially on harder variants of the algorithmic problem (right column), such as addition. In contrast, several iterative methods show significant degradation of performance with increased iterative computation.

We further qualitatively visualize the underlying iterative computation learned by IREM. In Figure 4, we visualize the element-wise mean square error of the predicted solution as a function of the number of iterative reasoning steps applied on the addition task. We find that energy minimization gradually refines a predicted solution to the ground truth additive answer, with different elements of the solution exhibiting different convergence rates.

Energy Landscape. IREM parameterizes an energy landscape across all possible solutions for a given problem. Such an energy landscape enables us to assess the relative quality of solutions dependent on their associated energies, and further gives a natural objective to terminate iterative computation when an underlying local energy minimum is reached. In Figure 5, we visualize the predicted energy of different candidate solutions and their corresponding MSE distances from the ground truth answer. We find that across different continuous algorithmic tasks, the underlying energy value assigned to a candidate solution is well correlated with its distance from the ground truth answer, with low energy solutions close to the ground truth.

Sensitivity to Step Size. We assess the performance of IREM under different values of step size λ using during energy optimization at training. We consider the continuous addition task in Table 3. We find that the underlying performance is not sensitive to hyperparameter choice for step size, and utilize a fixed step size of 100 across our experiments.

	Composed Operations		
Method	2	5	10
Feedforward	0.0445	0.2717	0.8898
Recurrent	2.1377	3.1861	4.8706
Programmatic	0.0203	0.1068	0.4587
Iterative Feedforward	0.0826	0.5930	3.6004
IREM (Ours)	**0.0014**	**0.0078**	**0.0422**

Table 4. **Algorithmic Composition –** Test performance when composing multiple instances of the addition operation. Error is reported using element-wise mean square error. IREM is able to generalize well when composing algorithmic computation.

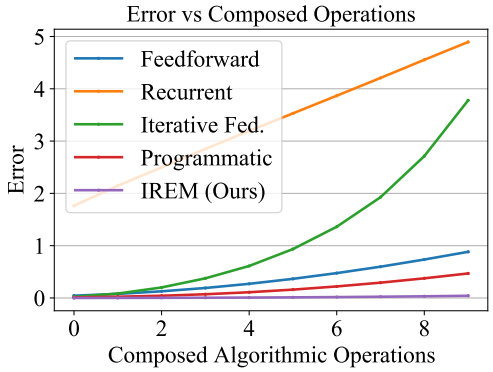

Figure 7. **Error Composing Algorithmic Computation** – Plot of error of predictions as factor of the number of composed algorithmic operations. IREM exhibits lower error when nesting a series of algorithmic operations together.

4.4. Recursive Algorithmic Computation

Setup. We further evaluate the ability of algorithms represented by IREM to be recursively applied on inputs. Recursively nesting algorithms enable complex computations, but require learned networks to be robust to out-of-distribution outputs from prior algorithmic execution.

We consider recursive applications of a learned algorithmic operator $\text{Alg}_\theta(\cdot)$ representing addition as introduced in Section 4.3. We evaluate the element-wise mean square error of the predicted output $\hat{y}$ of recursively applying the learned algorithmic operator k times

$$\hat{y}^k = \text{Alg}_\theta(\hat{y}^{k-1}, y_k), \tag{7}$$

with the corresponding ground truth solution being $\sum_k y_k$,

for different values k of recursive application.

Quantitative Results. We report the results of recursively applying each learned algorithmic operator between two to ten times in Table 4. We find that IREM supports the most stable recursive application of algorithmic operators. IREM exhibits significantly lower error than all compared baseline, due to its ability to utilize iterative computation to deal with out-of-distribution inputs and to accurately compute intermediate algorithmic outputs.

Qualitative Results. We illustrate error as a factor of the number of applied algorithmic operations in Figure 7, and find that the error of predictions from IREM rises slowly in comparison to other baselines. We further visualize the nested algorithmic predictions from IREM. We illustrate the inferred array sum predicted by IREM when four separate inputs are summed in Figure 6. As seen above, our approach enables us to closely approximate the addition of four input matrices (with the first nine entries shown).

5. Conclusion and Limitations

In this paper, we present IREM, a new approach towards iterative computation, by formulating it as an energy minimization process. We illustrate, on both continuous and graphical domains, how iterative computation utilizing IREM enables better algorithmic performance, as well as generalization to more complex instances of problems. We further illustrate how the underlying algorithmic computation learned by IREM may be nested to implement more complex algorithmic computations.

Iterative reasoning with IREM has several limitations. First, while IREM substantially outperforms existing iterative methods on tasks where output solutions have high dimensionality, limited gains are obtained when IREM is executed on problems with lower dimensionality solutions (such as parity prediction). Second, as training and inferring solutions with IREM relies on continuous gradient optimization, IREM struggles when output solutions have discrete values. An interesting line of future work would be to explore how discrete optimization could be integrated with training IREM to solve such discrete valued problems. Finally, since the training procedure of IREM requires backpropagation across gradient optimization steps, it is computationally expensive. An interesting line of future work could be exploring alternative ways to train an energy function for reasoning, such as utilizing gradient-free optimization.

6. Acknowledgements

We thank Ben Poole for feedback and helpful comments on a early version of the manuscript. Yilun Du is supported by a fellowship from the National Science Foundation.

References

Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pp. 136–145. PMLR, 2017.

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

Arbel, M., Zhou, L., and Gretton, A. Generalized energy based models. *arXiv preprint arXiv.org/abs/2003.05033*, 2020.

Bai, S., Kolter, J. Z., and Koltun, V. Deep equilibrium models. *arXiv preprint arXiv:1909.01377*, 2019.

Banino, A., Balaguer, J., and Blundell, C. Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.

Bengio, S., Bengio, Y., Cloutier, J., and Gecsei, J. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, volume 2, 1995.

Bolukbasi, T., Wang, J., Dekel, O., and Saligrama, V. Adaptive neural networks for efficient inference. In *International Conference on Machine Learning*, pp. 527–536. PMLR, 2017.

Brockett, R. W. Dynamical systems that sort lists, diagonalize matrices, and solve linear programming problems. *Linear Algebra and its applications*, 146:79–91, 1991.

Cai, J., Shin, R., and Song, D. Making neural programming architectures generalize via recursion. *arXiv preprint arXiv:1704.06611*, 2017.

Chen, X., Dai, H., Li, Y., Gao, X., and Song, L. Learning to stop while learning to predict. In *International Conference on Machine Learning*, pp. 1520–1530. PMLR, 2020.

Chung, J., Ahn, S., and Bengio, Y. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

Djolonga, J. and Krause, A. Differentiable learning of submodular models. *Advances in Neural Information Processing Systems*, 30:1013–1023, 2017.

Donti, P. L., Amos, B., and Kolter, J. Z. Task-based end-to-end model learning in stochastic optimization. *arXiv preprint arXiv:1703.04529*, 2017.

Du, Y. and Mordatch, I. Implicit generation and generalization in energy-based models. *arXiv preprint arXiv:1903.08689*, 2019.

Du, Y., Li, S., Sharma, Y., Tenenbaum, B. J., and Mordatch, I. Unsupervised learning of compositional energy concepts. In *Advances in Neural Information Processing Systems*, 2021a.

Du, Y., Li, S., Tenenbaum, B. J., and Mordatch, I. Improved contrastive divergence training of energy based models. In *Proceedings of the 38th International Conference on Machine Learning (ICML-21)*, 2021b.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pp. 1126–1135. PMLR, 2017.

Grathwohl, W., Wang, K.-C., Jacobsen, J.-H., Duvenaud, D., and Zemel, R. Cutting out the middle-man: Training and evaluating energy-based models without sampling. *arXiv preprint arXiv:2002.05616*, 2020.

Graves, A. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Hochreiter, S., Younger, A. S., and Conwell, P. R. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pp. 87–94. Springer, 2001.

Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

Impagliazzo, R. and Paturi, R. Complexity of k-sat. *Proceedings. Fourteenth Annual IEEE Conference on Computational Complexity (Formerly: Structure in Complexity Theory Conference) (Cat.No.99CB36317)*, Jun 1999. doi: 10.1109/ccc.1999.766282.

Kahneman, D. *Thinking, fast and slow*. Macmillan, 2011.

Kaiser, Ł. and Sutskever, I. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

Kemp, C. and Tenenbaum, J. B. The discovery of structural form. *Proceedings of the National Academy of Sciences*, 105(31):10687–10692, 2008.

Lampinen, A. K., Hsu, S., and McClelland, J. L. Analogies emerge from learning dyamics in neural networks. In *CogSci*, 2017.

LeCun, Y., Chopra, S., and Hadsell, R. A tutorial on energy-based learning. 2006.

Li, K. and Malik, J. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

Li, S., Du, Y., van de Ven, G. M., and Mordatch, I. Energy-based models for continual learning. *arXiv preprint arXiv:2011.12216*, 2020.

Nijkamp, E., Hill, M., Han, T., Zhu, S.-C., and Wu, Y. N. On the anatomy of mcmc-based maximum likelihood learning of energy-based models. *arXiv preprint arXiv:1903.12370*, 2019.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Ravi, S. and Larochelle, H. Optimization as a model for few-shot learning. 2016.

Reed, S. and De Freitas, N. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

Rubanova, Y., Sanchez-Gonzalez, A., Pfaff, T., and Battaglia, P. Constraint-based graph network simulator. *arXiv preprint arXiv:2112.09161*, 2021.

Schmidhuber, J. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Schwarzschild, A., Borgnia, E., Gupta, A., Huang, F., Vishkin, U., Goldblum, M., and Goldstein, T. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *arXiv preprint arXiv:2106.04537*, 2021.

Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. *arXiv preprint arXiv:1503.03585*, 2015.

Wilder, B., Dilkina, B., and Tambe, M. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1658–1665, 2019.

Xiao, Z., Kreis, K., Kautz, J., and Vahdat, A. Vaebm: A symbiosis between variational autoencoders and energy-based models. In *International Conference on Learning Representations*, 2020.

Zhang, M. and Chen, Y. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 31:5165–5175, 2018.

Appendix

In this appendix we provide additional evaluation and details on IREM. First, we illustrate how IREM may be applied to an existing image iterative reasoning task in Appendix A. Next we discuss how we may utilize optimization to execute IREM with an external scratchpad in Appendix B. We further discuss additional experimental details on our evaluated algorithmic tasks in Appendix C. Finally, we discuss individual model architectures used in Appendix D

A. Image Denoising

Setup We compare IREM with existing approaches on an existing image based iterative reasoning benchmark from (Chen et al., 2020). The benchmark task is to denoise images with various levels of Gaussian noise corruption added. Harder denoising tasks are constructed at test time by adding larger amounts of noise to input images. We directly compare with baselines and numbers for UNLNet, DnCNN and DNCNN-stop from (Chen et al., 2020) and utilize the authors provided training code to train IREM with five steps of iterative reasoning.

Quantitative Results We quantitatively evaluate the performance of IREM and baselines in terms of PSNR (numbers for baselines directly from (Chen et al., 2020)) in Table 5. While we found that IREM performed similarly to existing approaches when the test noise corruption is similar to that of training, IREM significantly outperformed the compared baselines when evaluated on harder images at test time which exhibited substantially larger amounts of noise corruption then seen during training. We found that such performance gain was due to iterative computation executed by IREM. While we trained IREM with 5 steps of reasoning, at test time we found that running up to 30 steps of reasoning at noise level $\sigma = 65$ improved performance, while running up to 100 steps of reasoning at noise level $\sigma = 75$ further improved performance.

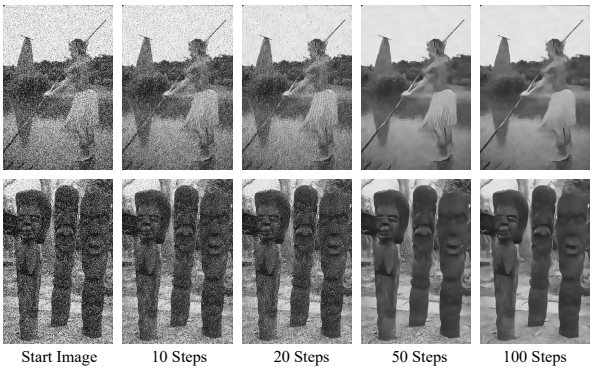

Figure 8. **Qualitative Illustration of Iterative Image Denoising** – Illustration of iterative denoising using IREM. Images continue to be cleaner after up to 100 iterative steps of computation.

σ	UNLNet	DnCNN	DnCNN-stop	IREM (Ours)
45	26.48	**26.56**	26.48	26.52
55	25.64	25.71	25.79	**25.85**
***65**	-	22.19	23.56	**23.87**
***75**	-	17.90	18.62	**23.43**

Table 5. **Image Denoising –** Reconstruction PSNR results for IREM and baseline comparisons. The $*$ indicates noise levels of 65 and 75 which are not in the training set. Numbers for baselines directly reported from (Chen et al., 2020). IREM generalizes substantially better to more complex denoising tasks.

Qualitative Results We qualitatively illustrate IREM being applied to an image corrupted with an unseen noise level $\sigma = 75$ (significantly larger than what is seen during training) in Figure 8. As seen in Figure 8, images continue to become clearer, even after 50 steps of iterative computation using IREM (with dress in the top row and floor in the bottom row becoming clearer).

B. Iterative Reasoning with a Scratchpad

We next discuss how we may incorporate an external computational scratchpad when executing an iterative computation with IREM. To enable the processing of an external scratchpad, $\boldsymbol{z} \in \mathbb{R}^D$, we construct a EBM, $E_\theta(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) : \mathbb{R}^N \times \mathbb{R}^M \times \mathbb{R}^D \to \mathbb{R}$, which takes as input an input problem $\boldsymbol{x}$, a candidate solution $\boldsymbol{y}$, and a scratchpad state $\boldsymbol{z}$.

We then define our predicted solution

$$\hat{\boldsymbol{y}} = \arg\min_{\boldsymbol{y}} \min_{\boldsymbol{z}} E_\theta(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}), \tag{8}$$

where optimize over both output solutions $\boldsymbol{y}$ and an external scratchpad $\boldsymbol{z}$ at prediction time. Analogous to the training procedure in the main paper, we may obtain a fast approximation of $\hat{\boldsymbol{y}}$ as $\boldsymbol{y}^N$ training time by jointly optimizing $\boldsymbol{y}$ and $\boldsymbol{z}$

$$\begin{aligned} \boldsymbol{y}_i^n &= \boldsymbol{y}_i^{n-1} - \lambda \nabla_{\boldsymbol{y}} E_\theta(\boldsymbol{x}_i, \boldsymbol{y}_i^{n-1}, \boldsymbol{z}_i^{n-1}) \\ \boldsymbol{z}_i^n &= \boldsymbol{z}_i^{n-1} - \lambda \nabla_{\boldsymbol{z}} E_\theta(\boldsymbol{x}_i, \boldsymbol{y}_i^{n-1}, \boldsymbol{z}_i^{n-1}). \end{aligned} \tag{9}$$

We then analogously minimize the corresponding loss:

$$\mathcal{L}_{\text{MSE}}(\theta) = \|\boldsymbol{y}_i^N - \boldsymbol{y}_i\|^2, \tag{10}$$

We provide the overall pseudocode for training IREM with external memory in Algorithm 3.

C. Experimental Details

Graphical Algorithmic Computation Models were trained in approximately 2 hours on a single Nvidia Titan X GPU using a training batch size of 64 and the Adam optimizer with learning rate 1e-4. Each model was trained for 10,000 iterations and evaluated on 1000 test problems.

Algorithm 3 IREM Training with External Memory

Input: Data Dist $p_D(\boldsymbol{x}, \boldsymbol{y})$, Replay Buffer $\mathcal{B}$, Step Size λ, Number of Steps N, EBM $E_\theta(\cdot)$, Uniform Distribution $U(-1, 1)$
$\mathcal{B} \leftarrow \varnothing$
while not converged **do**
  ▷ *Sample data and candidate solutions from p_d and replay buffer $\mathcal{B}$*
  $\boldsymbol{x}_i, \boldsymbol{y}_i \sim p_D, \tilde{\boldsymbol{y}}_i^0, \tilde{\boldsymbol{z}}_i^0 \sim \mathcal{U}(-1, 1)$
  $\boldsymbol{x}_i^b, \boldsymbol{y}_i^b, \tilde{\boldsymbol{y}}_i^b, \tilde{\boldsymbol{z}}_i^b \sim B$
  $\boldsymbol{x}_i, \boldsymbol{y}_i, \tilde{\boldsymbol{y}}_i^0, \tilde{\boldsymbol{z}}_i^0 \leftarrow \boldsymbol{x}_i \cup \boldsymbol{x}_i^b, \boldsymbol{y}_i \cup \boldsymbol{y}_i^b, \tilde{\boldsymbol{y}}_i^0 \cup \tilde{\boldsymbol{y}}_i^b, \tilde{\boldsymbol{z}}_i^0 \cup \tilde{\boldsymbol{z}}_i^b$
  ▷ *Generate low energy solutions through optimization:*
  for sample step $n = 1$ to N **do**
    $\tilde{\boldsymbol{y}}_i^n \leftarrow \tilde{\boldsymbol{y}}_i^{n-1} - \lambda \nabla_{\boldsymbol{y}} E_\theta(\boldsymbol{x}_i, \tilde{\boldsymbol{y}}_i^{n-1}, \tilde{\boldsymbol{z}}_i^{n-1})$
    $\tilde{\boldsymbol{z}}_i^n \leftarrow \tilde{\boldsymbol{z}}_i^{n-1} - \lambda \nabla_{\boldsymbol{z}} E_\theta(\boldsymbol{x}_i, \tilde{\boldsymbol{y}}_i^{n-1}, \tilde{\boldsymbol{z}}_i^{n-1})$
  end for
  ▷ *Optimize objective $\mathcal{L}_{MSE}$ wrt θ:*
  $\Delta\theta \leftarrow \nabla_\theta \sum_{n=1}^{N} \|\tilde{\boldsymbol{y}}_i^N - \boldsymbol{y}_i\|^2$
  Update θ based on $\Delta\theta$ using Adam optimizer
  ▷ *Update replay buffer $\mathcal{B}$*
  $\mathcal{B} \leftarrow \mathcal{B} \cup (\boldsymbol{x}_i, \boldsymbol{y}_i, \tilde{\boldsymbol{y}}_i^N, \tilde{\boldsymbol{z}}_i^N)$
end while

Each model was trained with five steps of iterative computation, with PonderNet trained with a halting geometric distribution of 0.8. Below, we provide additional numerical details about each of the evaluated algorithmic tasks.

1. *Edge Copy:* We randomly sample a value for each edge in a fully connected graph with a uniform value between -1 and 1. Models are then tasked with replicating the value of each individual edge in the graph in the final output prediction.

2. *Connected Components:* We randomly zero-out 95% of the edges in a fully connected graph. Models are then tasked with predicting the pairwise connectively of all possible pairs of nodes in the graph.

3. *Shortest Path:* We randomly sample an edge distance between 0 and 1 for each edge in a fully connected graph. Models are then tasked with predicting the pairwise shortest distance between all possible pairs of nodes in the graph.

Continuous Algorithmic Computation Models were trained in approximately 2 hours on a single Nvidia Titan X GPU using a training batch size of 128 and the Adam optimizer with learning rate 1e-4. Each model was trained for 10,000 iterations and evaluated on 1000 test problems. Each model was trained with five steps of iterative computation, with PonderNet trained with a halting geometric distribution of 0.8. We further provide individual dataset details below.

1. *Addition:* We randomly construct two separate vectors, each with 400 elements, with each element in the vector randomly sampled between -1 and 1. Models are then tasked with summing up the elements in each vector element-wise. When constructing more difficult addition problems at test time, each element in the vector is randomly sampled between -2.5 and 2.5.

2. *Matrix Completion:* We randomly construct a low-rank matrix M represented as the $M = U^T V + 0.1\mathcal{N}(0, 1)$, where U and V are 10×20 matrices, with each individual elements in U and V sampled from $\mathcal{N}(0, 0.22)$. Models are given 50% of the entries of M are tasked with recovering all entries of M. When constructing more difficult matrix completion problems at test time, each element in U and V are sampled from $\mathcal{N}(0, 0.47)$.

3. *Matrix Inverse:* We randomly construct a well conditioned invertible matrix $M = R + R^T + 0.5 * I$, where R is a random matrix, with individual elements sampled between -1 and 1. Models are tasked with computing the matrix inverse of M. When constructing more difficult matrix inversion problems at test time, we make M less well-conditioned by setting $M = R + R^T + 0.1 * I$.

D. Model Architectures

Graphical Algorithmic Computation For each iterative and feedforward method, we utilize the GINEConv layer from (Hu et al., 2019), where GINEConv$(128, 128)$ refers to a graph convolution operator with node features 128 and edge feature 128 . An input problem instance $\boldsymbol{x}$ consists of a set of nodes features $\boldsymbol{v}$ and edge features $\boldsymbol{e}$. To parameterize the EBM in IREM $E_\theta(\boldsymbol{x}, \boldsymbol{y})$ we concatenate the optimized prediction $\hat{\boldsymbol{y}}$, with $\boldsymbol{e}$, which then utilized in the GINEConv layer. To obtain per edge predictions for baselines, we pairwise concatenate node features for the given edges, and apply an FC layer to obtain the corresponding prediction following (Zhang & Chen, 2018). We specify the architecture for IREM in Table 6, the architecture for feedforward and iterative feedforward baselines in Table 7, the architecture for recurrent baselines in Table 8, and the architecture for programmatic execution baselines in Table 9. All models have roughly the same number of underlying parameters.

GINEConv(128, 128)
GINEConv(128, 128)
GINEConv(128, 128)
Linear $128 \rightarrow 1$

Table 6. The model architecture for IREM on graphical tasks.

GINEConv(128, 128)
GINEConv(128, 128)
GINEConv(128, 128)
Linear $256 \rightarrow$ Output Dim

Table 7. The model architecture for feedforward and iterative feedforward baselines on graphical tasks.

GINEConv(128, 128)
LSTM(128)
GINEConv(128, 128)
Linear 256 → Output Dim

Table 8. The model architecture for recurrent baseline on graphical tasks.

GINEConv(128, 128)
GINEConv(128)
GINEConv(128, 128)
Linear 256 → Output Dim
Linear → 1

Table 9. The model architecture for PonderNet baseline on graphical tasks.

Continuous Algorithmic Computation For each iterative and feedforward method, we utilize a MLP to implement continuous algorithmic computation. To parameterize the EBM in IREM $E_\theta(\boldsymbol{x}, \boldsymbol{y})$, we concatenate $\boldsymbol{x}$ and $\boldsymbol{y}$ together as input into the network. We utilize the ReLU activation in all networks except IREM, where we utilize the Swish activation. We specify the architecture for IREM in Table 10, the architecture for feedforward and iterative feedforward baselines in Table 11, the architecture for recurrent baselines in Table 12, and the architecture for programmatic execution baselines in Table 13. All models have roughly the same number of underlying parameters.

Linear 512
Linear 512
Linear 512
Linear → 1

Table 10. The model architecture for IREM on continuous tasks.

Linear 512
Linear 512
Linear 512
Linear → Output Dim

Table 11. The model architecture for feedforward and iterative feedforward baselines on continuous tasks.

Linear 196
LSTM 196
Linear → Output Dim

Table 12. The model architecture for recurrent baseline on continuous tasks.

Linear 512
Linear 512
Linear 512
Linear → Output Dim
Linear → 1

Table 13. The model architecture for PonderNet baseline on continuous tasks.