State-Reification Networks: Improving Generalization by Modeling the Distribution of Hidden Representations

Alex Lamb [1] **Jonathan Binas** [1 2] **Anirudh Goyal** [1] **Sandeep Subramanian** [2] **Ioannis Mitliagkas** [1]
Denis Kazakov [3] **Yoshua Bengio** [2] **Michael C. Mozer** [3 4]

Abstract

Machine learning promises methods that generalize well from finite labeled data. However, the brittleness of existing neural net approaches is revealed by notable failures, such as the existence of adversarial examples that are misclassified despite being nearly identical to a training example, or the inability of recurrent sequence-processing nets to stay on track without teacher forcing. We introduce a method, which we refer to as *state reification*, that involves modeling the distribution of hidden states over the training data and then projecting hidden states observed during testing toward this distribution. Our intuition is that if the network can remain in a familiar manifold of hidden space, subsequent layers of the net should be well trained to respond appropriately. We show that this state-reification method helps neural nets to generalize better, especially when labeled data are sparse, and also helps overcome the challenge of achieving robust generalization with adversarial training.

1. Introduction

The fundamental objective of machine learning is to build models of complex data. By abstracting from the data, models are typically more useful for domain understanding and prediction than are the raw data. This substitution of a model in place of the data is a form of *reification*. In this article, we argue that reification of data has similar value even when the data originate from within the model, i.e., its latent states. We propose a recursive model-within-a-model that reifies internal states in a neural network, leading to robustness and improved generalization.

[1]Université de Montréal, Montréal, Quebec [2]Montréal Institute for Learning Algorithms, Montréal, Quebec [3]University of Colorado, Boulder, CO [4]Presently at Google Brain, Mountain View, CA. Correspondence to: Alex Lamb <alex6200@gmail.com>, Michael C. Mozer <mcmozer@google.com>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

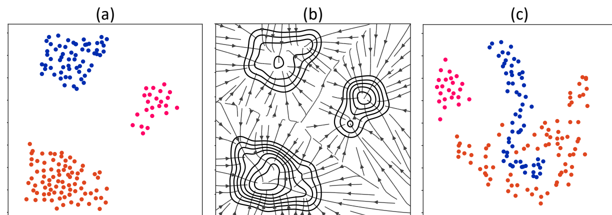

Figure 1. (a) A distribution of hidden states, with class label indicated by color. (b) State reification dynamics to map hidden states toward regions of higher density. (c) A distribution of input states, showing poorly separated classes, making it not suitable for reification.

Our proposed method, which we call *state reification*, is based on the idea that it is possible to model the distribution of the hidden states over the training data, and then map less likely states toward more likely states. Because the network has experienced these latter states frequently during training, we would expect to obtain better generalization from them. To offer an intuition, consider a simple task: training a recurrent net to output the parity of a stream of binary digits. The ideal internal state for solving this task is discrete, yet deep neural networks have continuous activations. Consequently, when evaluated on long test sequences, the continuous dynamics may cause the net to wander from the ideal states. State reification will map these rogue states back to the values observed during training, leading to dramatically better generalization.

Our approach stems from the observation that latent states, such as hidden representations, tend to lie on one or more manifolds. Figure 1a depicts a hidden representation of a training set in a classification task, with class label indicated by color. States within the manifold are 'familiar' in the sense that subsequent layers of the net have been tuned to process them. However, states lying outside the manifold are potentially problematic; they are not reached given the distribution of training inputs, and therefore the model's extrapolatory response may be unreliable.

In this work, we explore an approach in which we construct a model-within-a-model that implicitly encodes the distribution of states in the latent space and then projects states from off-manifold regions back to the manifold, where the network is likely to perform robustly (Figure 1b). We argue that this projection operation serves as a useful inductive bias during training that restricts the state space and induces a clustering of states. Not only does it boost generalization performance, but it also makes networks less sensitive to adversarial input perturbations, which tend to throw the state off the training manifold. Explicit detection of off-manifold states has proven useful for adversarial robustness and detecting out-of-distribution samples (Carrara et al., 2018; Lee et al., 2017; 2018), as has incorporating losses to shape the manifolds (Pang et al., 2018); we take this work further by presenting a general method that can be applied to any architecture, any layer of a network, and goes beyond identifying off-manifold states to projecting these states back to the manifold.

One could in-principle reify off-manifold inputs rather than off-manifold hidden states. A large body of literature exists on this topic, from early work achieving noise robustness via preprocessing stages that estimate and filter noise from an input signal (Boll, 1979) to more recent work in machine learning involving loss functions to achieve invariance to task-irrelevant perturbations in the input (Simard et al., 1992; Zheng et al., 2016). However, there are two reasons to prefer reification of hidden states. First, distinct semantic classes are typically more intertwined in the input space than in the hidden space (Figure 1c), and the manifolds are therefore simpler and smoother in an abstract space with simpler statistical structure. Second, state reification should have particular value in recurrent nets in which steps off manifold may compound as the hidden state evolves over a sequence.

2. State Reification

To show the robustness of our underlying insight, we describe two distinct but related mechanisms for state reification: denoising autoencoders and attractor networks.

2.1. Denoising Autoencoders

Denoising autoencoders (DAEs) are neural networks that map a noise-corrupted version of vector x to a clean version of x. This approach has been widely used for feature learning and generative modeling in deep learning (Bengio et al., 2013). More formally, denoising autoencoders are trained to minimize a reconstruction error or negative log-likelihood of generating the clean x. For example, with Gaussian log-likelihood of the clean vector given the corrupted vector, the reconstruction loss for data set $\mathbf{x} = \{x^{(1)}, \ \ldots, x^{(N)}\}$ is

$$\mathcal{L}_{\text{rec}}(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N}\left(\left\|r_\theta\left(x^{(n)}+a^{(n)}\right)-x^{(n)}\right\|_2^2\right), \quad (1)$$

where r_θ is the learned denoising function and $a^{(n)} \sim \mathbb{N}(\mathbf{0}, \sigma^2\mathbf{I})$ is a Gaussian noise vector.

Given loss $\mathcal{L}_{\text{rec}}$ and Gaussian corruption, a well-trained denoising autoencoder's reconstruction vector is proportional to the gradient of the log-density (Alain et al., 2012):

$$\frac{r_\sigma(x)-x}{\sigma^2} \rightarrow \frac{\partial \log p(x)}{\partial x} \quad \text{as} \quad \sigma \rightarrow 0. \quad (2)$$

The theory of Alain et al. (2012) establishes that the reconstruction vectors from a well-trained denoising autoencoder form a vector field which points in the direction of the data manifold. However, this result is not guaranteed for points distant from the manifold, as these points are rarely sampled during training. In practice, denoising autoencoders are trained with not just tiny noise levels but also with large noise levels, which blurs the data distribution as seen by the learner but makes the network learn a useful vector field even far from the data.

2.2. Attractor Networks

DAEs can be applied iteratively by cycling the output back to the input. A related but more principled approach is an *attractor network* (AN), which is essentially a DAE with recurrent connections within the hidden layer that results in a discrete-time nonlinear dynamical system with attractor manifolds, achieving trajectories like those shown in Figure 1b. Attractor nets have a long history starting with the seminal work of (Hopfield, 1982) that was partly responsible for the 1980s wave of excitement in neural networks. We adopt Koiran's (1994) framework, which dovetails with the standard deep learning assumption of synchronous updates on continuous-valued neurons. Koiran shows that a hidden layer with symmetric weights, nonnegative self-connections, and a bounded nonlinearity that is continuous and strictly increasing except at the extrema (e.g., tanh), the network asymptotically converges over iterations to a fixed point or limit cycle of length 2. Although the AN is recurrent, its training barely suffers from vanishing gradients (Bengio et al., 1994; Hochreiter, 1998) because the input projects to the hidden layer at each iteration, acting as a type of skip connection. Technical details are presented in the supplementary materials.

2.3. Incorporating State Reification Into a Model

We incorporate state reification within a neural net's internal layers to transform the representation toward the training-data manifold. For example, Figure 2a shows a feedforward

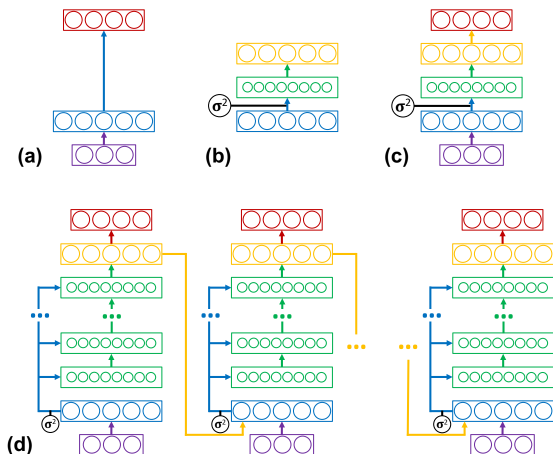

Figure 2. (a) A network that performs some input-output mapping task with one intervening hidden layer. (b) A DAE that produces a reified output. (c) Integrating the two architectures to perform state reification on the hidden state. (d) A recurrent sequence processing architecture, unrolled in time horizontally, with an attractor net—unrolled vertically—reifying the hidden state.

net that maps inputs to outputs through a hidden layer. Figure 2b shows a DAE with one internal layer, producing a reified output. Figure 2c integrates the feedforward net and DAE to reify the hidden state of the feedforward net. Figure 2d shows a more elaborate architecture, with a recurrent sequence-processing net integrated with an attractor net. Each column denotes a single time step with a corresponding input and output. The hidden state is denoised by an attractor net, unrolled vertically, yielding a reified state which is combined with the next input to determine the next hidden state. Intuitively, the method aims to regularize the hidden representations by projecting activations to the training-data manifold through the application of a DAE or attractor net.

To train the simple integrated model (Figure 2c), training data are processed in mini-batches, and the loss per (x, y) example is:

$$\mathcal{L} = \mathcal{L}_{\text{task}}(x, y) + \lambda_{\text{rec}}\, \mathcal{L}_{\text{rec}}(h) \tag{3}$$

is minimized, where $\mathcal{L}_{\text{rec}}$ (Equation 1) is applied to the hidden state, h, $\mathcal{L}_{\text{task}}$ is a primary-task loss, and the coefficient $\lambda_{\text{rec}} > 0$ controls the contribution of reification. This approach allows us in principle to reify multiple hidden layers at once, each with its own $\mathcal{L}_{\text{rec}}$ loss. In the next sections, we present results for two related applications: obtaining robust generalization to out-of-sample cases in sequential tasks, and obtaining robustness to standard adversarial attacks in feedforward nets. We use slightly different training procedures for each application, due to the different goals. For improving test-set generalization, we train only the reifier (DAE or AN) weights on $\mathcal{L}_{\text{rec}}$, all weights on $\mathcal{L}_{\text{task}}$, and we set the noise level, $\sigma^2 = 0$, for evaluation. For adversarial robustness, we train all weights on the joint loss, and perform simulations with and without the noise during evaluation; we also incorporate additional adversarial loss terms that are duals to $\mathcal{L}_{\text{rec}}$ and $\mathcal{L}_{\text{task}}$, to be described shortly.

3. Experiments

We demonstrate the effectiveness of state reification on three classes of problems: sequence classification in a data-limited training environment, generation of long sequences, and adversarial perturbations in image processing.

3.1. Recurrent Networks for Sequence Classification

Our first experiments involve symbolic sequence-classification tasks using recurrent networks like that in Figure 2d, where state reification is performed with attractor dynamics. We chose symbolic tasks—tasks with discrete inputs, and input-output mappings that can be characterized in terms of rules—because symbolic tasks have always been a challenge for continuous neural networks (Craven and Shavlik, 1993).

3.1.1. Parity

We studied a streamed *parity* task in which 10 binary inputs are presented in sequence and the target output following the last sequence element is 1 if an odd number of 1s is present in the input or 0 otherwise. The architecture has 10 hidden units, 20 attractor units, and a single input and a single output. We experimented with both tanh and GRU hidden units. We trained the attractor net with $\sigma = .5$ and ran it for exactly 15 iterations (more than sufficient to converge). Models were trained on 256 randomly selected binary sequences. Two distinct test sets were used to evaluate models: one consisted of the held-out 768 binary sequences, and a second test set consisted of three copies of each of the 256 training sequences with additive uniform $[-0.1, +0.1]$ input noise. We performed one hundred replications of a baseline architecture (RNN), an architecture with the additional layers to implement attractor dynamics but trained solely on $\mathcal{L}_{\text{task}}$ (RNN+, the '+' indicating the additional hardware), and an architecture with state reification (RNN+SR). Other details of this and subsequent simulations are presented in the Supplementary Materials.

Figure 3a shows relative performance on the held-out sequences by the RNN, RNN+, and RNN+SR with a tanh hidden layer. Figure 3b shows the same pattern of results for the noisy test sequences. RNN+SR significantly outperforms both the RNN and the RNN+: it generalizes better to novel sequences and is better at ignoring additive noise in test cases, although such noise was absent from training. Figures 3c,d show similar results for models with a GRU hidden

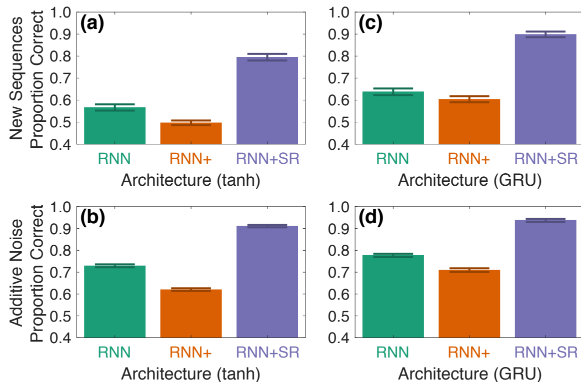

Figure 3. Parity simulations. Top row shows generalization performance on novel binary sequences; bottom row shows performance on trained sequences with additive noise. Unless otherwise noted, the simulations of with state reification use $\sigma = 0.5$, 15 attractor iterations, and L_2 regularization (a.k.a. weight decay) 0.0. Error bars indicate ± 1 SEM, based on a correction for confidence intervals with matched comparisons (Masson and Loftus, 2003).

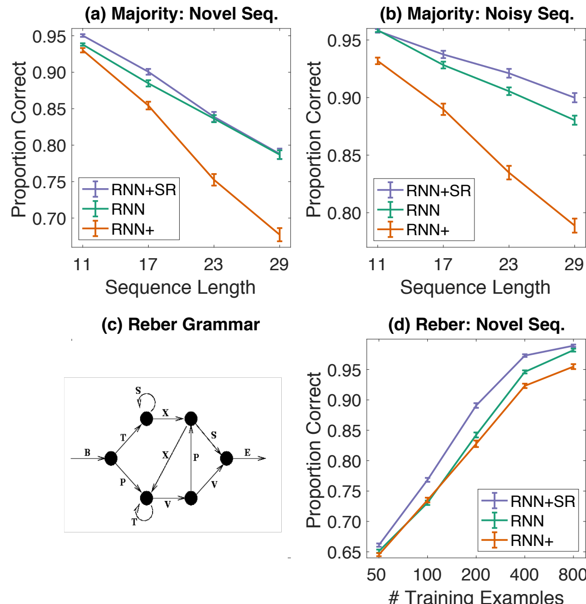

Figure 4. Simulation results on majority task with (a) novel and (b) noisy sequences. (c) Reber grammar. (d) Simulation results on Reber grammar. Error bars indicate ± 1 SEM, based on a correction for confidence intervals with matched comparisons (Masson and Loftus, 2003).

layer. Absolute performance improves for all three recurrent net variants with GRUs versus tanh hidden units, but the relative pattern of performance is unchanged. Note that the improvement due to denoising the hidden state (i.e., RNN+SR versus RNN for both tanh and GRU architectures) is much larger than the improvement due to switching hidden unit type (i.e., RNN with GRU vs. tanh hidden), and that the use of GRUs—and the equivalent LSTM—is viewed as a critical innovation in deep learning.

In principle, parity should be performed more robustly if a system has a highly restricted state space. Ideally, the state space would itself be binary, indicating whether the number of inputs thus far is even or odd. Such a restricted representation should force better generalization. Indeed, quantizing the hidden activation space for all sequence steps of the test set, we obtain a lower entropy for the tanh RNN+SR (3.70, standard error .06) than for the tanh RNN (4.03, standard error .05). However, what is surprising about this simulation is that gradient-based procedures could learn such a restricted representation, especially when two orthogonal losses compete with each other during training. The competing goals are clearly beneficial, as RNN+ and RNN+SR share the same architecture and differ only in the addition of the denoising loss.

The noise being suppressed during training is neither input noise nor label noise; it is noise in the internal state due to weights that have not yet been adapted to the task. Nonetheless, denoising internal state during training appears to help the model overcome input noise and generalize better.

3.1.2. Majority Task

We next studied a *majority* task in which the input is a binary sequence and the target output is 1 if a majority of inputs are 1, or 0 otherwise. We trained networks on 100 distinct randomly drawn fixed-length sequences, for length $l \in \{11, 17, 23, 29, 35\}$. We performed 100 replications for each l and each model. We ensured that runs of the various models were matched using the same weight initialization and the same training and test sets. All models had 10 tanh hidden units, 20 attractor units, $\sigma = .25$.

We chose the majority task because, in contrast to the parity task, we were uncertain if a restricted state representation would facilitate task performance. For the majority task of a given length l, the network needs to distinguish roughly $2l$ states. Collapsing them together is potentially dangerous: if the net does not keep exact count of the input sequence imbalance between 0's and 1's, it may fail.

As in the parity task, we tested both on novel binary sequences and training sequences with additive uniform noise. Figures 4a,b show that neither the RNN nor RNN+ beats RNN+SR for any sequence length on either test set. RNN+SR seems superior to the baseline RNN for short novel sequences and long noisy sequences. For short noisy sequences, both architectures reach a ceiling. The only disappointment in this simulation is the lack of a difference for novel long sequences.

3.1.3. Reber Grammar

The Reber grammar (Reber, 1967), shown in Figure 4c, has long been a test case for artificial grammar learning (e.g., Hochreiter and Schmidhuber, 1997). The task involves discriminating between strings that can and cannot be generated by the finite-state grammar. We generated positive strings by sampling from the grammar with uniform transition probabilities. Negative strings were generated from positive strings by substituting a single symbol for another symbol such that the resulting string is out-of-grammar. Examples of positive and negative strings are BTSSXXTTVPSE and BPTVPXTSPSE, respectively. Our networks used a one-hot encoding of the seven input symbols, $m = 20$ tanh hidden units, $n = 40$ attractor units, and $\sigma = 0.25$. The number of training examples was varied from 50 to 800, always with 2000 test examples. Both the training and test sets were balanced to have an equal number of positive to negative strings. One hundred replications of each simulation was run.

Figure 4d presents mean test set accuracy on the Reber grammar as a function of the number of examples used for training. As with previous data sets, RNN+SR outperforms the baseline RNN, which in turn outperforms RNN+.

3.1.4. Symmetry Task

The symmetry task involves detecting symmetry in fixed-length symbol strings such as ACAFBBFACA. This task is effectively a memory task for an RNN because the first half of the sequence must be retained to compare against the second half. We generated strings of length $2s + f$, where s is the number of symbols in the left and right sides and f is the number of intermediate fillers. For $i \in \{1, ..., s\}$, we generated symbols $S_i \in \{\text{A}, \text{B}, ..., \text{H}\}$. We then formed a string X whose elements are determined by S: $X_i = S_i$ for $i \in \{1, ..., s\}$, $X_i = \emptyset$ for $i \in \{s + 1, ..., s + f\}$, and $X_i = S_{2s+f+1-i}$ for $i \in \{s + f + 1, ..., 2s + f\}$. The filler $\emptyset$ was simply a unique symbol. Negative cases were generated from a randomly drawn positive case by either exchanging two adjacent distinct non-null symbols, e.g., ACAFBBAFCA, or substituting a single symbol with another, e.g., AHAFBBFACA. Our training and test sets had an equal number of positive and negative examples, and the negative examples were divided equally between the sequences with exchanges and substitutions.

We trained on 5000 examples and tested on an additional 2000, with the half sequence having length $s = 5$ and with an $f = 1$ or $f = 10$ slot filler. The longer filler makes temporal credit assignment more challenging. As shown in Figures 5, RNN+SR obtains as much as a 70% reduction in test error over either RNN or RNN+.

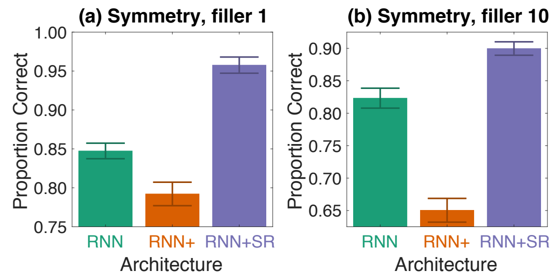

Figure 5. (a) Symmetry task with $f = 1$ filler; (b) Symmetry task with $f = 10$ filler. Error bars indicate ± 1 SEM.

3.2. Language Modeling

Turning to a second use of state reification, we explored whether the technique could be used to *detect* when the hidden state has wandered from the training manifold. Our experiment was performed with an RNN language model that generates word predictions as output. The model can be run in generative mode by sampling from the output distribution and feeding it back to the input. This free-running mode often produces wide divergences from training (Bengio et al., 2015), because during training, teacher forcing ensures that the model's input sequence is a valid (observed) word sequence. The divergence increases as the sequence progresses.

Our experiment studies if state reification can detect when it has been given outputs from its own model (sampling mode) when trained using ground truth input sequences. We trained a language model on the standard Text8 dataset, which is derived from Wikipedia articles. We trained a single-layer LSTM with 1000 units at the character-level, and included DAE state-reification between the hidden states and the output on each time step. In a given sequence, following 50 sampling steps, the state reification layers had a reconstruction error on average 103% of the teacher forcing reconstruction error. Following 180 sampling steps, this value increased to 112%. Following 300 sampling steps this value increased even further to 134%. These results provide clear evidence that the outputs move off of the manifold with more sampling steps, and that this is effectively measured by state reification.

3.3. Adversarial Training

In this section, we turn to a third application of state reification: obtaining networks robust to adversarial attacks, which consist of making small changes to input patterns that alter the predicted class. For image processing, the modulations of the input images can be small enough that they are unnoticeable to the human eye; the modulations can be so robust that even when captured through a camera, they change the predicted class with high probability (Brown et al., 2017).

Such *adversarial examples* (Goodfellow et al., 2014) can be found via gradient-based methods (Szegedy et al., 2013; Goodfellow et al., 2014).

Defenses proposed against adversarial examples include feature squeezing (Xu et al., 2017), adapted encoding of the input (Jacob Buckman, 2018), and distillation-related approaches (Papernot et al., 2015). Many have been shown to be providing the illusion of defense by lowering the quality of the gradient signal, without actually providing improved robustness (Athalye et al., 2018). One defense that is resilient to this *obfuscated-gradient problem* is adversarial training (Madry et al., 2017). Adversarial training consists of augmenting the dataset with adversarial examples and training the model's predictions to be unchanged by the adversarial noise.

However, a major challenge of incorporating adversarial training is that adversarial robustness is often dramatically worse on test data as compared to train data, suggesting difficulty in generalization (Schmidt et al., 2018). For this reason we explored the possibility of improving the performance of adversarial training by using state reification.

Adversarial training is a flexible procedure and can be used with any adversarial attack. For our investigation, we looked at the multi-step *projected gradient descent* (*PGD*) attack (Madry et al., 2017). We used an l^∞ attack with ε ranging from 0.03 to 0.3 and number of iterations ranging from 7 to 200. The PGD attack (Madry et al., 2017), also referred to as FGSM^k, is a multi-step extension of the Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2014) attack. The PGD attack is characterized as follows:

$$x^{t+1} = \Pi_{x+\mathcal{S}} \left(x^t + \alpha \, \mathrm{sgn}(\nabla_x \mathcal{L}_{\mathrm{task}}(x, y)) \right) \quad (4)$$

initialized with x^0 as the clean input x and with the corrupted input $\widetilde{x}$ as the last step in the sequence. Π refers to the projection operator, which in this context means projecting the adversarial example back onto the region within an ε radius of the original data point after each step in the adversarial attack.

To apply state reification to adversarial training, we modified our original state-reification training loss (Equation 3) with the standard adversarial training loss to encourage the network to not misclassify the adversarial example, yielding a combined loss for a given example (x, y) with an adversarial counterpart $\widetilde{x}$:

$$\mathcal{L} = \mathcal{L}_{\mathrm{task}}(x, y) + \mathcal{L}_{\mathrm{task}}(\widetilde{x}, y) + \lambda_{\mathrm{rec}} \sum_{i \in S} \mathcal{L}^i_{\mathrm{rec}}(h_i)$$

where S is the set of one or more hidden layers to which reification is applied, and the coefficient $\lambda_{\mathrm{rec}} \geq 0$ can be tuned to control the degree of reification. Because we potentially apply reification to multiple layers, we replaced the AN of our earlier simulations with the simpler DAE.

We have discussed advantages to performing reification in the hidden space instead of the input space, but the question of where exactly reification should be performed in a deep net remains unanswered: just the final hidden layer? Every hidden layer? We outline two important considerations regarding this issue. On the one hand, identifying states that are off-manifold or close to the margin is easier in the deeper hidden layers (see Figure 6, which we explain shortly). On the other hand, the states in the deeper hidden layers may already look non-adversarial due to the effect of the adversarial perturbations in the shallower layers. While we are not aware of any formal study of this phenomenon, it is clearly possible. (Imagine, for example, state reification performed on the output from the classifier softmax, which could only identify unnatural combinations of class probabilities.) Given these opposing concerns, we argue for the inclusion of reification at multiple stages of the network, very much analogous to the inclusion of reification at each time step of the recurrent net in our previous simulations.

We collected experimental evidence that more directly supports our decision to perform state reification at many levels of representation. We constructed FGSM adversarial examples ($\varepsilon = 0.3$) on small MNIST fully-connected networks trained normally. As Figure 6 shows, we found that detecting adversarial examples by reconstruction error is possible both in input and hidden layers, but could be performed by much smaller autoencoders via the hidden layers.

Tables 1 and 2 present results applying state reification on CIFAR10 using non-ResNet and ResNet convolutional nets (CNNs), respectively. Substantially better test-set adversarial robustness is attained via adversarial training when done in conjunction with state reification, evaluated on a wide range of ε values (0.03 to 0.3) and number of attack steps (7 to 200).

Athalye et al. (2018) suggest that models which introduce components with noisy or unreliable gradients can reduce the quality of gradient-based attacks. To test this hypothesis, they introduced *backward-pass differentiable approximation*, where the attack treats the "reconstructor" (in our case, the DAE) as the identity function when computing gradients for the attack. Our results showed that bypassing the DAE substantially reduced the strength of the attack, resulting in an increase in PGD accuracy to 67.1% from 40.1% (higher accuracy implies a weaker attack). Additionally, we ran a noiseless attack in which the forward and backward passes were performed without noise. This change strengthened the attack: PGD accuracy rises to 40.1% from 38.2% (lower accuracy implies a stronger attack), but we note that this is much less than the overall gap between state reification and the same-capacity baseline, suggesting that adding noise did partially obfuscate gradients, but not to such a degree as to nullify the improvements from state reification.

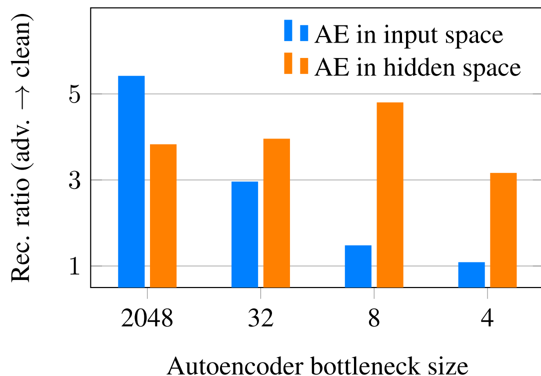

Figure 6. Direct experimental evidence that reification is easier in hidden layers than the input: we added denoising autoencoders with different capacities to MLPs trained on MNIST, and display the value of the total reconstruction errors for adversarial examples divided by the total reconstruction errors for clean examples. A high value indicates success at detecting adversarial examples. Our results support the central motivation for state reification: that off-manifold points can much more easily be detected in the hidden space (as seen by the relatively constant ratio for the autoencoder in hidden space) and are much harder to detect in the input space (as seen by this ratio rapidly falling to zero as the input-space autoencoder's capacity is reduced

Table 1. CIFAR-10 PGD Results with (non-ResNet) CNNs. In these experiment we apply state reification (with single hidden layer convolutional autoencoders) following each convolutional layer. Both experiments were run for 200 epochs and with all hyperparameters and architecture kept the same with the exception of state reification being added. We considered different types of baselines: CNN means we simply remove state reification. CNN+ means that we added extra layers with the same number of units to match the capacity added by state reification. The three blocks of results use slightly different architectures for the CNNs and are thus not directly comparable. All models reported were trained with adversarial training with a PGD attack. Note that higher PGD accuracy indicates a stronger defense.

Attack Type	PGD Steps	Attack Epsilon	PGD Accuracy		
			CNN	CNN+	CNN+SR
Normal	7	0.03	33.0	34.2	45.0
Normal	50	0.03	31.6	32.5	42.1
Normal	200	0.03	31.4	32.2	41.5
Normal	100	0.03		35.3	39.2
Normal	100	0.04		24.8	28.0
Normal	100	0.06		14.3	15.6
Normal	100	0.08		12.0	13.0
Normal	100	0.10		11.7	12.9
Normal	100	0.20		10.2	11.3
Normal	100	0.30		8.4	9.6
Normal	100	0.03		33.4	40.1
Noiseless Attack	100	0.03			38.2
BPDA, Skip-DAE	100	0.03			67.1

4. Related Work

State reification seems related to several recent papers with a cognitive science focus. Andreas et al. (2017) proposed a model that efficiently learns new concepts and control policies by operating in a linguistically constrained representational space. The space is obtained by pretraining on a language task, and this pretraining imposes structure on subsequent learning. One can view reification as imposing similar structure, although the bias comes not from a separate task or data set, but from representations already learned for the primary task. Related to language, the *consciousness prior* of Bengio (2017) suggests a potential role of operating in a reduced or simplified representational space. Bengio conjectures that the high dimensional state space of the brain is unwieldy, and a restricted representation that selects some information at the expense of other may facilitate rapid learning and efficient inference. For related ideas, also see Hinton (1990).

On the subject of our experimental results on adversarial robustness, the observation that adversarial examples often consist of points off the data manifold and that deep networks may not generalize well to these points motivated several authors to consider the use of the generative models as a defense against adversarial attacks (Gu and Rigazio, 2014; Ilyas et al., 2017; Samangouei et al., 2018; Liao et al., 2017). Ilyas et al. (2017); Gilmer et al. (2018) also showed the existence of adversarial examples which lie on the data manifold, and Ilyas et al. (2017) showed that training against adversarial examples forced to lie on the manifold is an effective defense. Our method shares a closely related motivation to these prior works, with a key difference being that we propose to consider the manifold in the space of learned representations, not the manifold directly in the visible space. One motivation for this is that the learned representations have a simpler statistical structure (Bengio et al., 2012), which makes the task of modeling this manifold and detecting unnatural points much simpler. Learning the distribution directly in the visible space is still very difficult (even state of the art models fall short of real data on metrics like Inception Score) and requires a high capacity model. Additionally, working in the space of learned representations allows for the use of a relatively simple generative model, in our case a small denoising autoencoder. Finally, another important difference is that we always use state reification together with adversarial training.

Denoising Feature Matching (Warde-Farley and Bengio, 2017) proposed to train a denoising autoencoder in the hidden states of the discriminator in a generative adversarial

Table 2. CIFAR-10 PGD Results with two powerful ResNet architectures: PreActResNet18 (He et al., 2016) and WideResNet28-10 (Zagoruyko and Komodakis, 2016). In this experiment we used a single state reification layer following the 2nd resblock, and the baseline consists of the same network but with the state reification removed. Both experiments were run for 200 epochs and with all hyperparameters and architecture kept the same with the exception of the state reification layer being added. ResNet-SR refers to the ResNet with state reification and ResNet refers to the baseline model. Note that higher PGD accuracy indicates a stronger defense.

Model	PGD Accuracy (20 steps) baseline	SR
PreActResNet18	37.87	39.20
WideResNet28-10	43.28	44.06

network. The generator's parameters are then trained to make the reconstruction error of this autoencoder small. This has the effect of encouraging the generator to produce points which are easy for the model to reconstruct, which will include true data points. Both this and state reification use a learned denoising autoencoder in the hidden states of a network. A major difference is that the denoising feature matching work focused on generative adversarial networks and tried to minimize reconstruction error through a learned generator network, whereas our approach targets the adversarial examples problem. Additionally, our objective encourages the output of the DAE to denoise adversarial examples so as to point back to the hidden state of the original example, which is different from the objective in the denoising feature matching work, which encouraged reconstruction error to be low on states from samples from the generator network.

MagNet (Meng and Chen, 2017) also proposed a method using autoencoders in the input space of a deep network to detect adversarial examples and "reform" them back to the input space. Their work differs from our approach in two critical ways. First, our method uses denoising autoencoders at several levels of representation, whereas MagNet (Meng and Chen, 2017) only operated in the input space. Second, our method is used together with adversarial training and is motivated primarily from the perspective of improving generalization in adversarial training. Many methods that have used autoencoders by themselves as a defense against adversarial examples are successful only when the autoencoder is ignored during the attack (Athalye et al., 2018); however, with state reification, we are able to improve robustness even when the autoencoder is used for the attack. In Table 1, we also present various alternative attacks that skip the autoencoder or don't inject noise, and found that robustness was preserved in all cases.

Gilmer et al. (2018) studied the existence of adversarial examples in the task of classifying between two hollow concentric shells. Intriguingly, they prove and construct adversarial examples which lie on the data manifold (although Ilyas et al., 2017, also looked for such examples experimentally using GANs). The existence of such on-manifold adversarial examples demonstrates that a simplified version of our model trained with only $\mathcal{L}_{\text{rec}}$ and not adversarial training could not protect against all adversarial examples. However, combined with adversarial training, state reification may still help with on-manifold adversarial examples as well by mapping the hidden state back to regions where the model performs well.

5. Discussion

Noise robustness is a highly desirable property in neural networks. When a neural net performs well, it naturally exhibits a sort of noise suppression: activation in a layer is relatively invariant to noise injected at lower layers (Arora et al., 2018). We described a method, state reification, which has the explicit objective of attaining robustness to unfamiliar variation, and we demonstrated that state reification helps neural nets to generalize better, especially when labeled data are sparse, and also helps overcome the challenge of achieving robust generalization with adversarial training. We also described two different implementation substrates for state reification, one using attractor nets and the other denoising autoencoders. We suspect that other kinds of unsupervised learning mechanisms that perform representation compression and density estimation will work as well if not better, especially those with explicit probabilistic underpinnings.

Our aim has been to show that state reification is an idea with breadth—over the quite disparate domains of symbolic sequence recognition and generation tasks and adversarial robustness. Although state reification appears to have some practical uses, more basic research is needed to understand how neural nets perform in regions of hidden state space outside the training manifold. More broadly, state reification addresses an issue that is often neglected in deep learning: how to build robust models given that internal state spaces are continuous, high dimensional, and often unbounded. The human brain has solved this problem, and artificial intelligence needs to do so as well.

References

Alain, G., Bengio, Y., and Rifai, S. (2012). Regularized autoencoders estimate local statistics. *CoRR*, abs/1211.4246.

Andreas, J., Klein, D., and Levine, S. (2017). Learning with latent language. *CoRR*, abs/1711.00482.

Arora, S., Ge, R., Neyshabur, B., and Zhang, Y. (2018). Stronger generalization bounds for deep nets via a compression approach. *CoRR*, abs/1802.05296.

Athalye, A., Carlini, N., and Wagner, D. (2018). Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. *ArXiv e-prints*.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 1171–1179. Curran Associates, Inc.

Bengio, Y. (2017). The consciousness prior. *CoRR*, abs/1709.08568.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*, 35(8):1798–1828.

Bengio, Y., Mesnil, G., Dauphin, Y., and Rifai, S. (2012). Better mixing via deep representations. *CoRR*, abs/1207.4404.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166.

Boll, S. (1979). Suppression of acoustic noise in speech using spectral subtraction. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(2):113–120.

Brown, T. B., Mané, D., Roy, A., Abadi, M., and Gilmer, J. (2017). Adversarial Patch. *ArXiv e-prints*.

Carrara, F., Becarelli, R., Caldelli, R., Falchi, F., and Amato, G. (2018). Adversarial examples detection in features distance spaces. http://www.nmis.isti.cnr.it/falchi/Draft/2018-ECCV-WOCM-Draft.pdf.

Craven, M. W. and Shavlik, J. W. (1993). Learning symbolic rules using artificial neural networks. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 73–80. Morgan Kaufmann.

Gilmer, J., Metz, L., Faghri, F., Schoenholz, S. S., Raghu, M., Wattenberg, M., and Goodfellow, I. (2018). Adversarial Spheres. *ArXiv e-prints*.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and Harnessing Adversarial Examples. *ArXiv e-prints*.

Gu, S. and Rigazio, L. (2014). Towards deep neural network architectures robust to adversarial examples. *CoRR*, abs/1412.5068.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Identity mappings in deep residual networks. *CoRR*, abs/1603.05027.

Hinton, G. E. (1990). Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence*, 46:47–75.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116.

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

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558.

Ilyas, A., Jalal, A., Asteri, E., Daskalakis, C., and Dimakis, A. G. (2017). The Robust Manifold Defense: Adversarial Training using Generative Models. *ArXiv e-prints*.

Jacob Buckman, Aurko Roy, C. R. I. G. (2018). Thermometer encoding: One hot way to resist adversarial examples. *International Conference on Learning Representations*.

Koiran, P. (1994). Dynamics of discrete time, continuous state hopfield networks. *Neural Computation*, 6(3):459–468.

Lee, K., Lee, K., Lee, H., and Shin, J. (2017). Training confidence-calibrated classifiers for detecting out-of-distribution samples. *CoRR*, abs/1711.09325.

Lee, K., Lee, K., Lee, H., and Shin, J. (2018). A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 7167–7177. Curran Associates, Inc.

Liao, F., Liang, M., Dong, Y., Pang, T., Zhu, J., and Hu, X. (2017). Defense against Adversarial Attacks Using High-Level Representation Guided Denoiser. *ArXiv e-prints*.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017). Towards Deep Learning Models Resistant to Adversarial Attacks. *ArXiv e-prints*.

Masson, M. and Loftus, G. (2003). Using confidence intervals for graphically based data interpretation. *Canadian Journal of Experimental Psychology*, 57:203–220.

Meng, D. and Chen, H. (2017). Magnet: A two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 135–147, New York, NY, USA. ACM.

Pang, T., Du, C., Dong, Y., and Zhu, J. (2018). Towards robust detection of adversarial examples. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 4584–4594. Curran Associates, Inc.

Papernot, N., McDaniel, P. D., Wu, X., Jha, S., and Swami, A. (2015). Distillation as a defense to adversarial perturbations against deep neural networks. *CoRR*, abs/1511.04508.

Reber, A. S. (1967). Implicit learning of artificial grammars. *Verbal learning and verbal behavior*, 5:855–863.

Samangouei, P., Kabkab, M., and Chellappa, R. (2018). Defense-gan: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations*, volume 9.

Schmidt, L., Santurkar, S., Tsipras, D., Talwar, K., and Madry, A. (2018). Adversarially robust generalization requires more data. *CoRR*, abs/1804.11285.

Simard, P., Victorri, B., LeCun, Y., and Denker, J. (1992). Tangent prop - a formalism for specifying selected invariances in an adaptive network. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 895–903. Morgan-Kaufmann.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *ArXiv e-prints*.

Warde-Farley, D. and Bengio, Y. (2017). Improving generative adversarial networks with denoising feature matching. In *International Conference on Learning Representations 2017 (Conference Track)*.

Xu, W., Evans, D., and Qi, Y. (2017). Feature squeezing: Detecting adversarial examples in deep neural networks. *CoRR*, abs/1704.01155.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *CoRR*, abs/1605.07146.

Zheng, S., Song, Y., Leung, T., and Goodfellow, I. (2016). Improving the robustness of deep neural networks via stability training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4480–4488.

Supplementary Materials for State-Reification Networks: Improving Generalization by Modeling the Distribution of Hidden Representations

Alex Lamb [1] **Jonathan Binas** [1 2] **Anirudh Goyal** [1] **Sandeep Subramanian** [2] **Ioannis Mitliagkas** [1]
Denis Kazakov [3] **Yoshua Bengio** [2] **Michael C. Mozer** [3 4]

1. Attractor net details

The attractor nets we explore are discrete-time nonlinear dynamical systems. Given a static n-dimensional input $\boldsymbol{c}$, the network state at iteration k, $\boldsymbol{a}_k$, is updated according to:

$$\boldsymbol{a}_k = f\left(\boldsymbol{W}\boldsymbol{a}_{k-1} + \boldsymbol{c}\right), \qquad (1)$$

where f is a nonlinearity and $\boldsymbol{W}$ is a weight matrix. Under certain conditions on f and $\boldsymbol{W}$, the state is guaranteed to converge to a *limit cycle* or *fixed point*. A limit cycle of length λ occurs if $\lim_{k\to\infty} \boldsymbol{a}_k = \boldsymbol{a}_{k+\lambda}$. A fixed point is the special case of $\lambda = 1$.

Attractor nets have a long history starting with the work of (Hopfield, 1982) that was partly responsible for the 1980s wave of excitement in neural networks. Hopfield defined a mapping from network state, $\boldsymbol{a}$, to scalar *energy* values via an energy (or Lyapunov) function, and showed that the dynamics of the net perform local energy minimization. The shape of the energy landscape is determined by weights $\boldsymbol{W}$ and input $\boldsymbol{c}$ (Figure 1a). Hopfield's original work is based on binary-valued neurons with asynchronous update; since then similar results have been obtained for certain nets with continuous-valued nonlinear neurons acting in continuous (Hopfield, 1984) or discrete time (Koiran, 1994). We adopt Koiran's 1994 framework, which dovetails with the standard deep learning assumption of synchronous updates on continuous-valued neurons. Koiran shows that with symmetric weights ($w_{ji} = w_{ij}$), nonnegative self-connections ($w_{ii} \geq 0$), and a bounded nonlinearity f that is continuous and strictly increasing except at the extrema (e.g., tanh, logistic, or their piece-wise linear approximations), the network asymptotically converges over iterations to a fixed point or limit cycle of length 2. As a shorthand, we refer to convergence in this manner as *stability*.

Attractor nets have been used for multiple purposes, including content-addressable memory, information retrieval, and constraint satisfaction (Mozer, 2009; Siegelmann, 2008). In each application, the network is given an input containing partial or corrupted information, which we will refer to as the *cue*, denoted $\boldsymbol{c}$; and the cue is mapped to a canonical or *well-formed* output in $\boldsymbol{a}_\infty$. For example, to implement a content-addressable memory, a set of vectors, $\{\boldsymbol{\xi}^{(1)}, \boldsymbol{\xi}^{(2)}, \ldots\}$, must first be stored. The energy landscape is sculpted to make each $\boldsymbol{\xi}^{(i)}$ an attractor via a supervised training procedure in which the target output is the vector $\boldsymbol{\xi}^{(i)}$ for some i and the input $\boldsymbol{c}$ is a noise-corrupted version of the target. Following training, noise-corrupted inputs should be 'cleaned up' to reconstruct the state.

In the model described by Equation 1, the attractor dynamics operate in the same representational space as the input (cue) and output. Historically, this is the common architecture. By projecting the input to a higher dimensional latent space, we can design attractor nets with greater representational capacity. Figure 1b shows our architecture, with an m-dimensional input, $\boldsymbol{x} \in [-1,+1]^m$, an m-dimensional output, $\boldsymbol{y} \in [-1,+1]^m$, and an n-dimensional attractor state, $\boldsymbol{a}$, where typically $n > m$ for representational flexibility. The input $\boldsymbol{x}$ is projected to the attractor space via an affine transform:

$$\boldsymbol{c} = \boldsymbol{W}^{\text{IN}}\boldsymbol{x} + \boldsymbol{v}^{\text{IN}}. \qquad (2)$$

The attractor dynamics operate as in Equation 1, with initialization $\boldsymbol{a}_0 = \boldsymbol{0}$ and $f \equiv \tanh$. Finally, the asymptotic attractor state is mapped to the output:

$$\boldsymbol{y} = f^{\text{OUT}}\left(\boldsymbol{W}^{\text{OUT}}\boldsymbol{a}_\infty + \boldsymbol{v}^{\text{OUT}}\right), \qquad (3)$$

where f^{OUT} is an output activation function and the $\boldsymbol{W}^*$ and $\boldsymbol{v}^*$ are free parameters of the model.

To conceptualize a manner in which this network might operate, $\boldsymbol{W}^{\text{IN}}$ might copy the m input features forward and

*Equal contribution [1]Université de Montréal, Montréal, Quebec [2]Montréal Institute for Learning Algorithms, Montréal, Quebec [3]University of Colorado, Boulder, CO [4]Presently at Google Brain, Mountain View, CA. Correspondence to: Alex Lamb <alex6200@gmail.com>, Michael C. Mozer <mcmozer@google.com>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

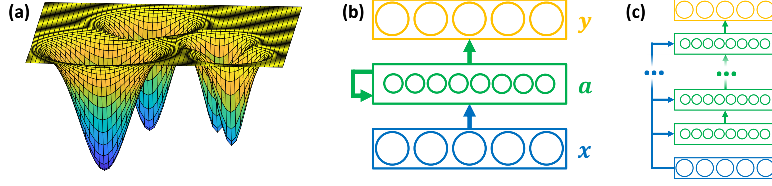

Figure 1. (a) energy landscape, (b) attractor net, (c) attractor net unfolded in time.

the attractor net might use m degrees of freedom in its state representation to maintain these *visible* features. The other $n - m$ degrees of freedom could be used as *latent* features that impose higher-order constraints among the visible features (in much the same manner as the hidden units in a restricted Boltzmann machine). When the attractor state is mapped to the output, $\boldsymbol{W}^{\text{OUT}}$ would then transfer only the visible features.

As Equations 1 and 2 indicate, the input $\boldsymbol{x}$ biases the attractor state $\boldsymbol{a}_k$ at every iteration k, rather than—as in the standard recurrent net architecture—being treated as an initial value, e.g., $\boldsymbol{a}_0 = \boldsymbol{x}$. Effectively, there are short-circuit connections between input and output to avoid vanishing gradients that arise in deep networks (Figure 1c). As a result of the connectivity, it is trivial for the network to copy $\boldsymbol{x}$ to $\boldsymbol{y}$—and thus to propagate gradients back from $\boldsymbol{y}$ to $\boldsymbol{x}$. For example, the network will implement the mapping $\boldsymbol{y} = \boldsymbol{x}$ if: $m = n$, $\boldsymbol{W}^{\text{IN}} = \boldsymbol{W}^{\text{OUT}} = I$, $\boldsymbol{v}^{\text{IN}} = \boldsymbol{v}^{\text{OUT}} = \boldsymbol{0}$, $\boldsymbol{W} = \boldsymbol{0}$, and $f^{\text{OUT}}(\boldsymbol{z}) = \boldsymbol{z}$ or $f^{\text{OUT}}(\boldsymbol{z}) = \max(-1, \min(+1, \boldsymbol{z}))$.

In our simulations, we use an alternative formulation of the architecture that also enables the copying of $\boldsymbol{x}$ to $\boldsymbol{y}$ by treating the input as unbounded and imposing a bounding nonlinearity on the output. This variant consists of: the input $\boldsymbol{x}$ being replaced with $\hat{\boldsymbol{x}} \equiv \tanh^{-1}(\boldsymbol{x})$ in Equation 2, $f^{\text{OUT}} \equiv \tanh$ in Equation 3, and the nonlinearity in the attractor dynamics being shifted back one iteration, i.e., Equation 1 becomes $\boldsymbol{a}_k = \boldsymbol{W} f(\boldsymbol{a}_{k-1}) + \boldsymbol{c}$. This formulation is elegant if $\boldsymbol{x}$ is the activation pattern from a layer of tanh neurons, in which case the tanh and $\tanh^{-1}$ nonlinearities cancel. Otherwise, to ensure numerical stability, one can define the input $\hat{\boldsymbol{x}} \equiv \tanh^{-1}[(1 - \varepsilon)\boldsymbol{x}]$ for some small ε.

1.1. Training a denoising attractor network

We demonstrate the supervised training of a set of attractor states, $\{\boldsymbol{\xi}^{(1)}, \boldsymbol{\xi}^{(2)}, \ldots \boldsymbol{\xi}^{(A)}\}$, with $\boldsymbol{\xi}^{(i)} \sim \text{Uniform}(-1, +1)^m$. The input to the network is a noisy version of some state i, $\hat{\boldsymbol{x}}^{(i)} = \tanh^{-1}(\boldsymbol{\xi}^{(i)}) + \boldsymbol{\eta}$, with $\boldsymbol{\eta} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 I)$, and the corresponding target output is simply $\boldsymbol{\xi}^{(i)}$. With κ noisy instances of each attractor for training, we define a normalized denoising loss

$$\mathcal{L}_{\text{denoise}} = \frac{1}{\kappa A} \sum_{i=1}^{\kappa A} \frac{||\boldsymbol{y}^{(i)} - \boldsymbol{\xi}^{(i)}||^2}{||\tanh(\hat{\boldsymbol{x}}^{(i)}) - \boldsymbol{\xi}^{(i)}||^2}, \tag{4}$$

to be minimized by stochastic gradient descent. The aim is to sculpt attractor basins whose diameters are related to σ^2. The normalization in Equation 4 serves to scale the loss such that $\mathcal{L}_{\text{denoise}} \geq 1$ indicates failure of denoising and $\mathcal{L}_{\text{denoise}} = 0$ indicates complete denoising. The $[0, 1]$ range of this loss helps with calibration when combined with other losses.

We trained attractor networks varying the number of target attractor states, $A \in [25, 250]$, the noise corruption, $\sigma \in \{0.125, 0.250, 0.500\}$, and the number of units in the attractor net, $n \in [25, 250]$. In all simulations, the input and output dimensionality is fixed at $m = 50$, $\kappa = 50$ training inputs and $\kappa = 50$ testing inputs were generated for each of the A attractors. The network is run to convergence. Due to the possibility of a limit cycle of 2 steps, we used the convergence criterion $||\boldsymbol{y}_{k+2} - \boldsymbol{y}_k||_\infty < \delta$, where $\boldsymbol{y}_k$ is the output at iteration k of the attractor dynamics. This criterion ensures that no element of the state is changing by more than some small δ. For $\delta = .01$, we found convergence typically in under 5 steps, nearly always under 10.

Figure 2 shows the percentage of noise variance removed by the attractor dynamics on the test set, defined as $100(1 - \mathcal{L}_{\text{denoise}})$. The three panels correspond to different levels of noise in the training set; in all cases, the noise in the test set was fixed at $\sigma = 0.250$. The 4 curves each correspond to a different size of the attractor net, n. Larger n should have greater capacity for storing attractors, but also should afford more opportunity to overfit the training set. For all noise levels, the smallest ($n = 50$) and largest ($n = 200$) nets have lower storage capacity than the intermediate ($n = 100, 150$) nets, as reflected by a more precipitous drop in noise suppression as A, the number of attractors to be stored, increases. One take-away from this result is that roughly we should choose $n \approx 2m$, that is, the hidden attractor space should be about twice as large as the input/output space. This result does not appear to depend on the number of attractors stored, but it may well depend on the volume of training data. Another take-away from this simulation is that the noise level in training should match

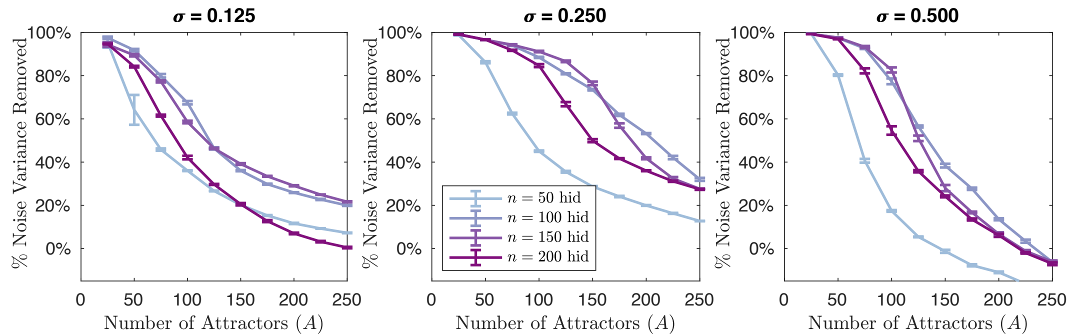

Figure 2. Percentage noise variance suppression by an attractor net trained on 3 noise levels (σ). In each graph, the number of hidden units in the attractor net (n) and number of attractors to be learned (A) is varied. In all cases, inputs are 50-dimensional and evaluation is based on a fixed $\sigma = 0.250$. Ten replications are run of each condition; error bars indicate ± 1 SEM.

that in testing: training with less ($\sigma = 0.125$) and more ($\sigma = 0.500$) noise than in testing ($\sigma = 0.250$) resulted in poorer noise suppression. Based on this simulation, the optimal parameters, $\sigma = 0.250$ and $n = 100$, should yield a capacity for the network that is roughly $2n$, as indicated by the performance drop that accelerates for $A > 100$.

1.2. Attractor net initialization

We initialize the attractor net with all attractor weights being drawn from a normal distribution with mean zero and standard deviation .01. Additionally, we add 1.0 to the on-diagonal weights of $\boldsymbol{W}^{\text{IN}}$ and $\boldsymbol{W}^{\text{OUT}}$, i.e., $W_{ii}^{\text{IN}} = 1 + \varepsilon$ and $W_{ii}^{\text{OUT}} = 1 + \varepsilon$ for $i \leq \min(m, n)$, where $\varepsilon \sim \mathcal{N}(0, .01)$ and m is the number of units in the input to the attractor net and n is the number of internal (hidden) units in the attractor net.

2. Experimental details

In all experiments, we chose a fixed initial learning rate with the ADAM optimizer. We used the same learning rate for $\mathcal{L}_{\text{denoise}}$ and $\mathcal{L}_{\text{task}}$. For all tasks, $\mathcal{L}_{\text{task}}$ is mean squared error. For the synthetic simulations (parity, majority, Reber, and symmetry), we trained for a fixed upper bound on the number of epochs but stopped training if the training set performance reached asymptote. (There was no noise in any of these data sets, and thus performance below 100% is indicative that the network had not fully learned the task. Continuing to train after 100% accuracy had been attained on the training set tended to lead to overfitting, so we stopped training at that point. In all simulations, for testing we use the weights that achieve the highest accuracy on the training set (not the lowest loss).

2.1. Parity

We use mean squared error for $\mathcal{L}_{task}$ in Parity, Majority, Reber, and Symmetry. In Parity, the learning rate is .008 for $\mathcal{L}_{\text{task}}$ and $\mathcal{L}_{\text{denoise}}$. Because the data set is noise free, training continues until classification accuracy on the training set is 100% or until 5000 epochs are reached. Training is in complete batches. Hidden units are tanh, and the data set has a balance on expectation of positive and negative examples. Our entropy calculation is based on discretizing each hidden unit to 8 equal-sized intervals in $[-1, +1]$ and casting the 10-dimensional hidden state into one of 10^8 bins. For this and only this simulation, the attractor weights were trained only on the attractor loss. In all following simulations, the attractor weights are trained on both losses (as described in the main text). This change was due to an evolution in our approach, not because we tried both and attractor-loss-only training was necessary.

Figure 3a,b show the effect of varying σ in training attractors. If σ is too small, the attractor net will do little to alter the state, and the model will behave as an ordinary RNN. If σ is too large, many states will be collapsed together and performance will suffer. An intermediate σ thus must be best, although what is 'just right' should turn out to be domain dependent. We explored one additional manipulation that had no systematic effect on performance. We hypothesized that because the attractor-net targets change over the course of learning, it might facilitate training to introduce weight decay in order to forget the attractors learned early in training. We introduced an L_2 regularizer using the ridge loss, $\mathcal{L}_{\text{ridge}} = \lambda ||\boldsymbol{W}||_2^2$, where $\boldsymbol{W}$ is the symmetric attractor weight matrix in Equation 1 and λ is a weight decay strength. As Figures 3c,d indicate, weight decay has no systematic effect, and thus, all subsequent experiments use a ridge loss $\lambda = 0$.

2.2. Majority

The networks are trained for 2500 epochs or until perfect classification accuracy is achieved on the training set. The attractor net is run for 5 steps. Ten hidden units are used and $\sigma = 0.25$. On expectation there is a balance between the number of positive and negative examples in both the training and test sets.

2.3. Reber

The networks are trained for 2500 or until perfect classification accuracy is achieved on the training set. We filtered out strings of length greater than 20, and we left padded strings with shorter lengths with the begin symbol, B. The training and test sets had an equal number of positive and negative examples. The architecture included $m = 20$ hidden units and $n = 40$ attractor units, and the attractor net is run for 5 steps. Without exploring alternatives, we decided to postpone the introduction of $\mathcal{L}_{\text{denoise}}$ until 100 epochs had passed.

2.4. Symmetry

The networks are trained for 2500 or until perfect classification accuracy is achieved on the training set. A learning rate of .002 was used for both losses for $f = 10$ and .003 for both losses for $f = 1$. The attractor net was run for 5 iterations.

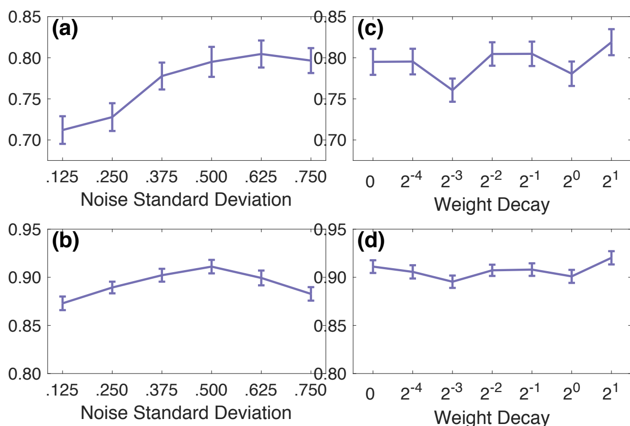

Figure 3. Parity simulations. Top row shows generalization performance on novel binary sequences; bottom row shows performance on trained sequences with additive noise. Unless otherwise noted, the simulations of the SDRNN use $\sigma = 0.5$, 15 attractor iterations, and L_2 regularization (a.k.a. weight decay) 0.0. Error bars indicate ± 1 SEM, based on a correction for confidence intervals with matched comparisons (Masson and Loftus, 2003).

References

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, 81(10):3088–3092.

Koiran, P. (1994). Dynamics of discrete time, continuous state hopfield networks. *Neural Computation*, 6(3):459–468.

Masson, M. and Loftus, G. (2003). Using confidence intervals for graphically based data interpretation. *Canadian Journal of Experimental Psychology*, 57:203–220.

Mozer, M. C. (2009). Attractor networks. In Bayne, T., Cleeremans, A., and Wilken, P., editors, *Oxford Companion to Consciousness*, pages 88–89. Oxford University Press, Oxford UK.

Siegelmann, H. T. (2008). Analog-symbolic memory that tracks via reconsolidation. *Physica D: Nonlinear Phenomena*, 237(9):1207 – 1214.