RADBOUD UNIVERSITY NIJMEGEN

FACULTY OF SCIENCE

Noise-induced replay in shallow autoencoders

LEARNING BY DREAMING

THESIS BSC PHYSICS & ASTRONOMY

July 2021

First supervisor:
dr. Francesco BATTAGLIA

Daily supervisor & First reader:
dr. Federico STELLA

Second reader:
prof. dr. Bert KAPPEN

Author:
Jelmar GERRITSEN

Abstract

It is well-known that the mammalian brain is able to replay past experiences, an ability which has been linked to the hippocampus and one which appears to be important for memory consolidation. However, the reactivated neural patterns are not always exactly the same as when they originally occurred, and they can even represent novel activations that were never experienced in the first place.

We aim to get a better understanding of replay and its potential mechanisms through the use of artificial neural networks. Specifically, a denoising autoencoder is trained in an unsupervised manner to reconstruct a hierarchical toy dataset, whose latent space is then analyzed using methods from attractor dynamics. When a trained model is initially fed random noise and its outputs are continually looped-back to serve as input for the next iteration, fixed-point attractor states can be observed. We show that the patterns produced by this iterative process can be reliably used to train a second model to reproduce the original dataset, which it is never trained on directly. We evaluate the behavior and feasibility of models trained in this manner, and discuss its potential implications for the nature of replay.

Keywords: machine learning, autoencoder, replay, memorization, attractor, neural networks

Contents

1 Introduction

The formation of memories is one of the most important capabilities of the mammalian brain. By storing experiences for later use, future decisions can be adapted to provide better outcomes for the organism, thus improving its ability to survive. But while the brain is able to recall past episodic events where relevant, the observed benefit to inference can mostly be attributed to the brains' remarkable ability to *generalize* across multiple memories. This general acquisition of knowledge is also termed *learning*, a concept which has recently been expanded beyond the realm of biology into the world of the computer in the paradigm of machine learning.

Artificial neural networks (ANNs) are the general class of computer models which aim to mimic the biological systems that are capable of learning. They do so by partially replicating the behaviors that neurons and synapses perform in their biological counterparts, but in a vastly simplified manner. They are traditionally trained on samples from a training set for a certain amount of time, and then tested on an unseen "validation" dataset to check if the model generalizes well. However, this process is not without its problems - it has been shown that increasing the amount of data the network sees at any given moment (the *batch size*) contributes to the existence of a so-called "generalization gap" [8], where the model gets stuck in a sharp local minimum preventing it from generalizing further. In addition, neural networks are prone to *overfitting* [4], which further hampers generalization, and *catastrophic forgetting*, where previously learned information is abruptly "forgotten" in order to accomodate new information [11].

The capability for *replay* has been theorized to play an important role in the consolidation of memories and the ability to generalize in biological brains, and it might be essential for preventing the interference that artificial neural networks are known to suffer from [2]. The complementary learning systems (CLS) theory posits that the brain facilitates learning by having two specialized systems, which can communicate through the replaying of patterns. This effectively creates an *interleaving* of experiences, which has been shown to alleviate interference [9].

In order to investigate the apparent discrepancy between the learning capabilities of biological and artificial neural networks, we adopt a similar connectionist approach. We build upon recent work done by [18, 9] in analyzing the learning of hierarchical data, with the important distinctions that we instead train the model in an *unsupervised* manner and include nonlinear activation functions. Specifically, we modeled simplified versions of both components of the CLS framework by using denoising autoencoders. The first "hippocampal" model was trained to reproduce a highly orthogonal, hierarchical binary dataset adapted from [15], while the second "neocortical" model was trained only on overlapping, distributed patterns "replayed" by the first model produced by iterating a Markov chain starting from random noise. This was done for two different scenarios, where the size of the hidden layer in the first model was varied between over- and underparameterized. The results were interpreted using methods from the field of dynamical systems.

2 Cognitive neuroscience

While some of the following concepts from neuroscience have managed to find their way into the terminology of machine learning, their background and motivation will now briefly be discussed.

2.1 Catastrophic interference

Early neural networks were able to perfectly store sets of binary patterns, but also completely unable to interpolate this data when faced with unseen inputs. This represented one extreme in the tradeoff between stability and plasticity; while the memories stored were immutable and could always be perfectly recalled, no new knowledge could be accommodated. The invention of the backpropagation algorithm allowed models to generalize, but at the cost of accuracy in accessing older memories.

The challenges involved in sequentially training a network that is both sensitive to new information while also retaining older knowledge was first raised in the late 90s [11, 14]. In these works, the problem of *catastrophic interference* (or *catastrophic forgetting*) was established as having two main consequences: the rapid forgetting of established information and reduced ability to discriminate between new and unseen items as new information is learned. Both of these findings were in marked contrast to the perceived capabilities of the human brain, which is perfectly able to learn without forgetting while even *improving* its discriminative performance.

The problem was sourced to the nature of overlapping distributed representations inside the hidden layer(s), where weights are shared between many different input items. Many (partial) solutions to the problem were proposed, including increasing the share of the data the network sees at any given time (the so-called batch size), which came with its own problems [8]. It was also shown that reducing the overlap by having a (more) orthogonal dataset, and the *interleaving* of items during learning could heavily alleviate the resulting interference [9]. However, a solution was also presented that was informed by the neuroscientific side of things, and it is one which we will now explore.

2.2 Complementary learning systems

At the start of the 1980s, the parallel distributed processing (PDP) paradigm was the dominant connectionist approach. As its name implies, it stressed the importance of distributed representations within the brain, which allows multiple calculations to occur at the same time [16]. The categorization of nodes into *input*, *output* and *hidden* units allowed networks to attain a much greater degree of flexibility compared to perceptrons, and the invention of the backpropagation algorithm [17] provided a general way to train them.

These models were not without issues, however. At the time, connectionism, the idea that artificial neural networks could be made to model mental phenomena, was still suffering from the problem of *catastrophic interference* (as detailed in the previous section), the consequences of which fundamentally challenged the ability of neural networks to serve as models for neural correlates [11]. If animal brains are able to learn sequentially, a connectionist framework must also be able to do so, and its apparent inability to do so was taken by some as a sign that the PDP framework was fundamentally flawed.

The theory of CLS was first proposed in [10] to address these concerns, and it proved an attractive prospect for the future of connectionist models. Its central proposition is that the brain is able to overcome the problem of catastrophic interference by employing two separate 'complementary' systems, which are both specialized in their respective niches. The general process of learning and memorization can then be said to originate from the interplay between these two systems: as new information is learned by the first, fast system, the dependence is gradually shifted towards the second, slower system in a mechanism called *consolidation*.

The first of these systems is termed the **hippocampal system**, and is associated with the fast learning of sparse, orthogonal *pattern separated* representations. It has high neuroplasticity, and is able to act on a rapid timescale. The non-overlapping nature of the patterns enables this system to perform sequential learning, while the same property also makes it bad at generalizing. The importance of the hippocampus in the formation of episodic memories is supported by lesion studies, which impair the learning of novel experiences while leaving intact other cognitive functions.

In contrast, the second **neocortical system** gradually integrates experiences across multiple episodes, a process which lends itself well to forming dense, overlapping *distributed* representations. This system (which includes most, but not all parts comprising the neocortex; the actual structures included are beyond the scope of this paper) is robust to sudden changes but receptive to the repetition of similar content.

This characterization would leave the neocortical system prone to catastrophic interference, were it not for the important observation that the hippocampus is able to *replay* experiences to the neocortex, thereby achieving a form of *interleaved* training which opposes the degradation of older memories. The concept of replay extends well beyond the confines of CLS and takes a central role in this work, so it merits a discussion of its own.

2.3 Replay

It has been shown that the mammalian brain continues to process memories 'offline' after they have initially formed, in wakeful states as well as during sleep [7, 22]. While the exact effects on memory consolidation remain unclear, it has been hypothesized that offline processing can strengthen or weaken synaptic plasticity, or reorganize memory traces altogether [2]. The general phenomenon of the offline (partial) reactivation of memory traces is called **replay**.

While it is attractive to speak of a specific memory being activated, the actual content of reactivated patterns has not been intensively studied. The nature of these patterns is generally time-compressed (meaning they span a shorter duration than during memory formation), and they may only be a subset of the actual "memory" in question [2]. In fact, in research concerning 299 dream reports, only 1-2% reported experiencing an episodic memory that played out in the same manner as during its relevant waking event [1].

It is this apparent **generative** nature of replay that is most interesting for this study. While the concept of replay has recently found its way into the toolkit of machine learning engineers [19] with great success, experience replay merely samples patterns from a replay buffer containing episodic memories; it is therefore a form of interleaved learning (with the advantage that not all of the data has to be known before training), which is also unrealistic from a biological standpoint since it requires large amounts of storage. Instead, the use of generative replay averts the storage problem, and it has been shown to be a viable strategy in the paradigm of continual learning [23].

3 Machine learning

We now provide an explanation of some fundamental concepts related to the study of machine learning and neural networks used in this thesis.

3.1 Artificial neural networks

When we speak of a **neural network** in this work, we are referring to a feedforward neural network (FNN). Information only flows forward in this type of network, and as such it can be represented by a directed acyclic graph (DAG). This is in contrast to recurrent neural networks (RNNs), where connections can be bidirectional. In a FNN, we can speak of **input** and **output** layers, which are the sets of "visible" units where information enters or leaves the model respectively. In between these layers, **hidden** layers can optionally be included to increase the predictive power of the model. These layers are often followed by a nonlinear **activation function**, without which the network would only be able to produce a linear decision boundary, severely restricting the datasets it is capable of learning.

Neural networks are useful because they can be trained to perform a variety of different tasks. The main paradigm through which they are trained is by using **supervised learning algorithms**, where we have a dataset $\mathcal{X}$ containing *features* (which are often crafted by hand), and a corresponding set of *labels* $\mathcal{Y}$. The model is presented with samples from $\mathcal{X}$, after which the predicted label $\hat{\boldsymbol{y}}$ is used together with the actual "target" label $\boldsymbol{y} \in \mathcal{Y}$ to compute the loss using a specified **loss function**. Backpropagation is then performed with respect to this loss, and the weights of the model are adjusted accordingly. In doing so, the model thus learns to estimate the distribution $p(\boldsymbol{y} \mid \boldsymbol{x})$. This process is called "supervised" because it requires a teacher to construct the set of labels $\mathcal{Y}$ which the model uses to learn [3].

3.2 Unsupervised learning

In contrast to supervised learning, **unsupervised learning algorithms** do *not* require an additional set of ground-truth labels. Usually, they are used to train a network to approximate the probability distribution $p(\boldsymbol{x})$ from which the training examples $\boldsymbol{x}^{(i)}$ are sampled. They do so by extracting useful features in order to minimize the reconstruction loss between a sample $\boldsymbol{x}$ and its prediction by the model $\hat{\boldsymbol{x}}$.

This last part is often the motivating factor in choosing an unsupervised model: sometimes, we are only interested in learning representations that successfully capture the nature of a complex dataset, without knowing what they are beforehand. This might be done to perform *unsupervised pretraining* on a model which is later fine-tuned in a supervised manner to achieve better performance, or as a way to learn representations which are useful across multiple different tasks in the paradigm of *transfer learning* [12]. The general goal of learning useful features and the techniques involved in achieving it are commonly grouped under the term **representation learning**.

The seminal example of an unsupervised model is the **autoencoder**. It is composed of an **encoder** part which projects the input to a (often lower-dimensional) hidden representation, and a **decoder** part which is responsible for turning that representation back into the input vector. A basic autoencoder architecture with a single hidden layer can be seen in figure 1. By limiting the size of the hidden layer or by imposing restrictions on the gradient through *regularization*, the learned representation can be chosen to have specific desirable properties [3].

3.3 Denoising autoencoders

One such autoencoder model with special properties is called the **denoising autoencoder** (DAE). Instead of being presented with a sample $\boldsymbol{x} \in \mathcal{X}$ as input, it receives a *corrupted version* $\tilde{\boldsymbol{x}}$. The loss is still computed with respect to the original, uncorrupted training vector, and so the model effectively learns to remove the noise from the noisy samples, hence the name "denoising". Given

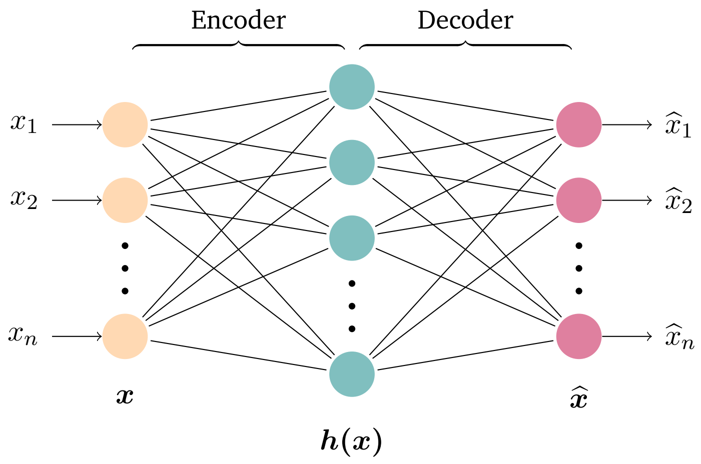


Figure 1: Basic autoencoder network architecture with one hidden layer.

enough time, it learns the reconstruction distribution $p(\boldsymbol{x} \mid \tilde{\boldsymbol{x}})$ which is robust to small input perturbations. Mathematically speaking, a traditional autoencoder attempts to minimize the function

$$\mathcal{L}(\boldsymbol{x},\ g(f(\boldsymbol{x}))) \tag{1}$$

where $\mathcal{L}$ is a specific loss function, and f and g are the functions learned by the encoder and decoder parts of the network respectively. Instead, a DAE minimizes the function

$$\mathcal{L}(\boldsymbol{x},\ g(f(\tilde{\boldsymbol{x}}))) \tag{2}$$

where $\tilde{\boldsymbol{x}}$ is a copy of $\boldsymbol{x}$ corrupted by some form of noise. The addition of noise reduces the risk that the model simply learns the *identity function*, which is almost always undesirable. This risk can be further lowered by adding a *penalty term* to the loss function:

$$\mathcal{L}(\boldsymbol{x},\ g(f(\tilde{\boldsymbol{x}}))) + \Omega(\boldsymbol{\theta}) \tag{3}$$

where $\boldsymbol{\theta}$ denotes the parameters of the model. This term can take a variety of forms, but in this work it was used to perform $L2$ regularization, also known as **ridge regression**. The penalty term then takes the following form:

$$\Omega(\boldsymbol{\theta}) = \lambda \sum_i \theta_i^2 \tag{4}$$

where λ is a coefficient controlling the strength of the regularization. This penalty was added to reduce the chances of overfitting.

The denoising nature of the network makes it robust to small perturbations in the input space. A consequence of this is that any fixed point x of the model will have a surrounding neighborhood of points that can map to x, and not just x itself. What this means and how it influences our chosen methodology will now be addressed from the perspective of dynamical systems.

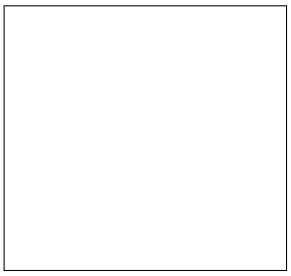
4 Dynamical systems

We now introduce a few concepts from the study of dynamical systems that are used in this work to analyze artificial neural networks. For an exhaustive overview of the subject, one can refer to the work 'Nonlinear dynamics and Chaos' [21].

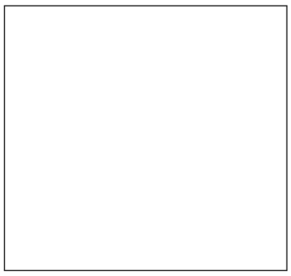
4.1 Iterative fixed points

A *fixed point* of a function f is defined as a point x for which $f(x) = x$. While an autoencoder learns to minimize the reconstruction loss between the input data and their predictions, in reality the training examples are rarely perfectly retrieved after training; instead being close to but almost never exactly equal to the original input.

For this reason, we introduce the concept of *iterative fixed points*. This concept is partially inspired by the recirculation algorithm first proposed in [5], although importantly the models



(a) Iterating the *Canary* item results in itself



(b) …while *Pine* instead transitions to *Oak* after iteration

Figure 2: In this particular instance of an underparameterized model, the *Canary* item is stored as an iterative fixed point, while *Pine* is not.

used in this work are still trained through backpropagation. Because an autoencoder has an input dimension d equal to its output dimension d_o, model predictions can be reused as input in a sequential fashion. The successive application of a model to an input vector is a process we will refer to as **iteration**.

Definition. *Let $f_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}^d$ be the function learned by a $d-$dimensional autoencoder trained on a dataset $\mathcal{X}$. We say that a point x is an* ***iterative fixed point*** *if successive applications of $f_{\boldsymbol{\theta}}(\cdot)$ given an initial input of x eventually converge to x. This trivially holds for $\forall x \in \mathcal{X}$ given that the model is overparameterized and trained until the loss is sufficiently small.*

An example of the latter condition not being met can be seen in figure 2, where not all training examples are stored as iterative fixed points in a model where the hidden dimension is constrained.

Using this definition, we can talk about fixed points even though the model doesn't always learn to perfectly memorize the input dataset. The nature of the denoising autoencoder allows us to further expand this concept to the more general notion of *attractors*.

4.2 Attractors

Fixed points are merely points that map to themselves, without revealing much about the neighborhood they find themselves in. For example, a saddle point of a function can be an iterative fixed point if approached from the right direction, while any small perturbation in another direction would cause the iteration to diverge. This is an example of an *unstable equilibrium*. The notion of an *attractor* imposes an additional criterium on a fixed point concerning their stability. In this paper, we use following definition of an attractor:

Definition. *A fixed point x is said to be an* ***attractor*** *if there exists a neighborhood of x, N, such that for every point $x_N \in N$ the sequence $\{f^k_{\boldsymbol{\theta}}(x_N)\}_{k\in\mathbb{N}}$ converges to x as the number of iterations $k \to \infty$.*

Remark. This definition applies to both "regular" as well as iterative fixed points.

For the sake of identifying attractors, we use the equivalent condition presented in [21] that a fixed point x can be said to be an attractor if the largest eigenvalue of the Jacobian $\mathbf{J}$ of $f_{\boldsymbol{\theta}}$ at point x is strictly less than 1.

From these definitions, it follows that it is possible to iteratively recirculate the previous output of a model starting from some perturbed training input $\tilde{\boldsymbol{x}}$, lying in the so-called *basin of attraction* of $\boldsymbol{x}$, to retrieve the original training pattern $\boldsymbol{x}$ given that the number of iterations is large enough. By traversing the phase space of the model from an initial vector constrained to the input space of the training set $\mathcal{X}$, the training examples that are stored as attractors can ultimately be retrieved.

5 Methods

To study the efficacy of using replay to train a student model, a total of four autoencoder models were built in and trained using Python and the PyTorch framework [13], distributed over two different scenarios. In the first scenario, both models are overparameterized, while in the second scenario, the first model was underparameterized and the second one is overparameterized.

(a) Hierarchical relations **(b)** Correlation matrix **(c)** Items and attributes

Figure 3: Category structure in the used hierarchical dataset adapted from [15]. The dataset has a shape of $(8,\ 9)$, and contains multiple levels of hierarchy as is visible in the correlation matrix.

Figure 4: Singular value decomposition of a subset of the training data. Technically speaking, this is the *thin SVD*, where the matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ contain only the first $k = min(m,\ n)$ columns (the input-output correlation matrix $\boldsymbol{\Sigma}^{yx}$ is $m \times n$). The columns of $\boldsymbol{U}$ and rows of $\boldsymbol{V}$ each form an orthonormal basis. *Note:* in future references, the singular value matrix here shown as $\boldsymbol{S}$ is instead called $\boldsymbol{\Sigma}$.

5.1 Dataset

The first model was trained to reproduce a hierarchical binary dataset adapted from [15], the statistics of which can be seen in figure 3. The used dataset contained 8 different items: *Oak*, *Pine*, *Daisy*, *Rose*, *Salmon*, *Sunfish*, *Canary* and *Robin*. Each item had a dimension of 9, where entries of 0 or 1 coded for the attributes *Move*, *Fly*, *Swim*, *Bark*, *Petals*, *Roots*, *Oak*, *Pine*, *Daisy*, *Rose*, *Salmon*, *Sunfish*, *Canary* and *Robin*. The last 8 dimensions individually code for only a single item, and are essentially just the 8×8 identity matrix.

This is an example of how some attributes carry a larger amount of information about the dataset than others, and the expectation is that a model constrained in size will gradually drop these dimensions in favor of those that allow the discrimination of more items. This measure of information content can be more intuitively explained by the **singular value decomposition** of the dataset $\mathcal{X}$. It is a matrix factorization which decomposes an $m \times m$ matrix into three matrices of size $m \times m$, $m \times n$ and $n \times n$ respectively (examples of which are shown in figure 4) and is described by the following equation:

$$\boldsymbol{\mathcal{X}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\boldsymbol{T}} = \sum_{i=1}^{N} \sigma_i \boldsymbol{u}^i \boldsymbol{v}^{iT} \tag{5}$$

Each of the three matrices has a distinct semantic interpretation. In the terminology we borrow from [18], the first matrix $\boldsymbol{U}$ is also called the *feature synthesizer* vector. It links the attributes of $\mathcal{X}$ to dimensions called *modes*, which encode semantic distinctions. For example, in figure 4 mode 3 codes for the difference between *Bark* and *Petals*; allowing *Oak* to be discriminated from *Rose*. The second mode captures more information, essentially splitting the dataset between the *Animals* and the *Plants*.

Similarly, the second matrix $\boldsymbol{\Sigma}$ is called the *singular value matrix*, and it is a diagonal matrix with ordered values in decreasing intensity. It can be interpreted as a ranking of the information content contained in the specific modes of $\boldsymbol{U}$: the first mode is the only one which codes for *Grow*, an attribute which every item in $\mathcal{X}$ possesses, as such it carries the highest information content and also has the largest singular value.

Finally, the third matrix $\boldsymbol{V^T}$ of the SVD can be thought of as the *object analyzer vector*, linking individual items (or objects) $\boldsymbol{v}^i$ to their respective modes $\boldsymbol{u}^i$ coupled by their singular value σ_i. For example, since mode 3 distinguishes between *Bark* and *Petals*, attributes which only the *Oak* and *Rose* possess respectively, it is in this matrix that their dissimilarity is captured along the *tree-flower* dimension.

As the network gradually learns to reconstruct the dataset $\mathcal{X}$, broader modes (which describe a larger part of $\mathcal{X}$) are expected to arise first, with finer categories following later, if at all. It is our hypothesis that an underparameterized model will not learn to represent the modes which explain the lowest share of the total variance in $\mathcal{X}$.

5.2 Experimental setup

Every training epoch, the first model was presented with a specific amount of *items* that possess certain *attributes*, which are represented by binary vectors. During the training a batch size of 4 was used for both models, meaning that every epoch gradient descent was performed twice, once on each minibatch. Model statistics (such as the loss) were averaged over 25 model runs to reduce the effect of random initialization. The parameters were initialized by using Xavier Initialization.

The rectified linear unit (ReLU) was used as the activation function, except for after the last hidden layer in the decoder where a Sigmoid nonlinearity was instead used to constrain the output to the interval $[0,\ 1]$, which is the range the dataset spans. In all models a dropout layer with a probability of 0.15 was used after the input and hidden layers to reduce the probability of over-fitting [20]. Initially both models also contained batch normalization layers [6], but these were discarded as they did not seem to make a meaningful difference in performance. Optimization was performed using the AdamW optimizer, with a learning rate of 10^{-2} in all cases. The loss included an $L2$ penalty term with a λ of 0.005. The loss function used was the mean-squared error (MSE), with the reduction set to sum instead of $mean$:

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{6}$$

The overparameterized model had a hidden size of 32 and was trained for a maximum of 3500 epochs, while the underparameterized one had a hidden size of only 2 units and instead was trained for a maximum of 1500. These values were chosen after experimentation with early stopping, where models were trained on the dataset until the average loss of all items reached 10^{-3}, or when the difference between the average loss in one epoch and the next was less than 10^{-4}, whichever came first. In the overparameterized case, the former criterion tended to be the cause for early stopping, whereas the underparameterized model never satisfied the former condition and so relied on the latter one for training to stop.

After the first model was trained (whether under- or overparameterized), a second model was trained whose size depended on the specific scenario under consideration. It was trained in the same manner as discussed above, except for the input data now being comprised of patterns generated by iterating random noise. From this "replay" dataset, 8 patterns (equal to the number of items in the original dataset) were uniformly picked from the list of patterns every epoch, and presented in batch sizes of 4 just like before. The process of pattern generation will now be further discussed.

5.3 Pattern generation

Once the first model had been trained to a sufficient level, a new dataset was produced through iterating the model from an initial state of random Gaussian noise. At every step, a certain amount of random noise was optionally added to the generated pattern. The expectation behind this is that perturbing the input will increase the probability of a less stable attractor state being reached, biasing the sampled states away from the strongest attractor. The full algorithm is detailed in algorithm 1 below, and the motivation behind it will now be explained.

The used algorithm is in many ways similar to a **Markov chain Monte Carlo** method. As mentioned above, the goal of the sampling with noise is to traverse the phase space of the model in a more "equal" manner, biasing it away from the strongest attractors. But the question remains what values for μ and σ constitute this "equality". The used values for the mean and standard deviation of the noise during the pattern generation were determined by the following process.

If a second model is to reliably learn the initial dataset $\mathcal{X}$, two characteristics are desirable in the generated dataset used for training the second model. First, the patterns should have a high

Algorithm 1 The algorithm used for pattern generation.

```
Input: a model $f_{\boldsymbol{\theta}}$, the number of patterns $n$, the number of steps $s$, the noise mean $\mu$,
    the noise standard deviation $\sigma$
Output: $p$, a list containing $n$ patterns
Require: $f_{\boldsymbol{\theta}}$ is a trained model
  $p \leftarrow [\,]$
  for $i \leftarrow 0, n$ do
      $\boldsymbol{x} \leftarrow \boldsymbol{U}(a,\ b)$        ▷ Initialize with uniform noise; $[a,\ b]$ spans the latent space of $f_{\boldsymbol{\theta}}$
      for $j \leftarrow 0, s$ do
          $\boldsymbol{y} \leftarrow f_{\boldsymbol{\theta}}(\boldsymbol{x})$                                  ▷ Model prediction for $\boldsymbol{x}$
          $\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{\mathcal{N}}(\mu,\ \sigma^2)$                     ▷ Optionally inject Gaussian noise
          $\boldsymbol{y} \leftarrow max(0,\ min(1,\ \boldsymbol{y}))$         ▷ Ensure $y_i \in [0,\ 1]\ \forall i$, since the dataset is binary
          $\boldsymbol{x} \leftarrow \boldsymbol{y}$
      end for
      $p \leftarrow p \cup [\boldsymbol{x}]$
  end for
  return $p$
```

similarity to the items in $\mathcal{X}$. Second, the relative frequency of the items should be well-balanced; this to ensure a sufficient degree of interleaving such that catastrophic forgetting does not occur. While the first condition can be satisfied by removing the patterns that are sampled during the "burn-in time" of the chain, the latter one poses more of a problem. When every training example $x \in \mathcal{X}$ is an attractor, this is not an issue since patterns would occur relative to the strength of their respective attractor, but when items are not stored as an attractor (e.g. in the underparameterized model), they are instead not retrieved through the process of iteration and end up being rare to non-existent in the generated dataset.

In order to include these non-attractor states which are nonetheless part of the dataset we would like the second model to be able to reproduce, it follows that during iteration we should like transitions to occur between items that are categorically similar. If sampling without noise would in large amounts produce the final state of *Oak*, noise-assisted perturbations during iteration should in some cases cause the state to instead jump to the basin of attraction of an item that lies nearby in the latent space, e.g. the *Pine*, even if *Pine* is itself a weak attractor or not an attractor at all. In the latter case, the state representing the item would not be favored over neighboring states that are not in $\mathcal{X}$ at all, and so our expectation is that sampling with noise comes paired with a general degradation in the quality of the produced patterns.

The notion of transitions during the process of iteration is encapsulated by the **transition matrix**, which captures what outputs certain inputs map (or "transition") to after model prediction. If every item is an iterative fixed point and iteration is performed without injecting intermediary noise, this matrix would be equal to the identity matrix with every item always transitioning to itself. The desire that the transition matrix should reflect the category structure of the dataset leads us to posit that the optimal values of μ and σ for the added noise during iteration are those that minimize the difference between the transition matrix and the rescaled correlation matrix seen in figure 3b. A transition matrix produced by generating patterns with these "optimal" values for μ and σ can be seen in figure 5c.

Mathematically, this means the following: let $\mathbf{A}(\mu,\ \sigma)$ be the transition matrix generated by iterating for n runs where n is a large number (e.g. 10000), and let $\mathbf{S}$ be the Pearson correlation matrix of $\mathcal{X}$ rescaled such that all of its entries lie in the closed interval $[0,\ 1]$. Then the sum of the Hadamard product of the difference between these matrices $\sum(\mathbf{A}(\mu,\ \sigma) - \mathbf{S}) \circ (\mathbf{A}(\mu,\ \sigma) - \mathbf{S})$ has a minimum value, as shown in figure 5a. These values were found to be very close to $\mu = 0$ and $\sigma = 0.5$, which were the values ultimately chosen for the noisy generation of patterns.

5.4 Reliability and limitations

In order to improve reproducibility, the

While we have made the assumption here that pattern generation is optimal for model performance when the transition matrix produced by iteration most accurately reflects the rescaled correlation matrix, it remains to be seen whether this is actually true. Though we did not attempt

(a) Error surface produced by measuring the dissimilarity between the rescaled correlation matrix of the dataset and the transition matrix at different values of μ and σ

(b) The rescaled correlation matrix $\mathbf{S}$ used in the calculation of categorical similarity

(c) An example transition matrix $\mathbf{A}$ after iterating with the "optimal" values of $\mu = 0$ and $\sigma = 0.5$

Figure 5: The existence of a global minimum for μ and σ in the categorical dissimilarity between $\mathbf{A}$ and $\mathbf{S}$, and the corresponding transition matrix. $\boldsymbol{A}$ was produced by iterating the trained model according to algorithm 1 for $10,000$ runs, and $\boldsymbol{S}$ was made by using min-max scaling to rescale the correlation matrix to the interval $[0,\ 1]$.

to provide a quantitative measure for model learning as a function of noise μ and σ, it would nevertheless have been interesting to investigate the effects of different noise parameters on the generated set of patterns.

Even though the generated patterns were produced by running n independent chains where $n \sim 10000$, this number might not actually have been large enough to sample the whole state space. Although the results were averaged over 25 different model runs to increase reliability, the ergodicity of the Markov chain could have been explored to ensure that no path-dependence was influencing the learning of the second model.

6 Results

As mentioned before, two scenarios were considered. In the first one, the first model was *underparameterized* with a hidden size of 2, and in the second one it was *overparameterized* having a hidden size of 32. In both cases, the second model was overparameterized, also sporting a hidden layer with a code size of 32.

6.1 Underparameterized scenario

The training statistics for the first model being underparameterized can be seen in figure 6. When the first model is constrained in size, the network is unable to accommodate every dimension in order to fully replicate the dataset, and so it is forced to generalize. This is especially apparent in figure 6c, where the diagonal entries of the singular value matrix $\mathbf{\Sigma}$ are plotted (produced by decomposing the matrix of model predictions every epoch): only 5 of the total of 8 singular values are nonzero when the model finishes training. The average loss per item hovers around 0.5 in epoch 1500, with a large deviation across different model runs. The injection of noise during the sampling process caused the *Rose* and *Pine* items to be included in the replay dataset to a much higher degree, which improved their loss at the expense of the other items.

Since the underparameterized has a code size of only 2 dimensions, we can directly visualize its latent space without having to resort to dimensionality reduction. This visualization highlighting the existence of attractors can be seen in figure 7.

Training metrics for the second (overparameterized) model when trained on patterns sampled from the first, underparameterized one can be seen in figure 8. These metrics were also averaged over 25 model runs, although the observed variation is much smaller.

6.2 Overparameterized scenario

In this scenario, both models were overparameterized. The relevant training curves are visible in figure 9. While the model was trained for a total of 1500 epochs, the plots are constrained to

(a) Loss curves

(b) Individual item losses

(c) Singular value trajectories

Figure 6: Training metrics for an *underparameterized* first model with a hidden size of 2, averaged over 25 model runs. Because of limited capacity, the network is forced to generalize, which is visible in the large deviations in individual item losses between model runs, and also in the small amount of nonzero singular values.

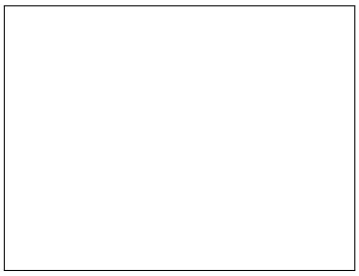
Figure 7: Visualization of the latent space learned by an underparameterized first model. Not every item is stored as an attractor, and some representations heavily overlap. This plot was produced by generating a grid of $10,000$ $2d$ points and feeding it only through the decoder part of the network, after which they were categorized as the item that was closest (measured by Euclidean distance) to the model prediction and colored appropriately. The arrows stem from iterating each point for 25 steps and computing the difference between start and end states in the hidden layer; their magnitudes are proportional to the distance covered. The stars represent the hidden encodings of the items $\boldsymbol{x} \in \mathcal{X}$.

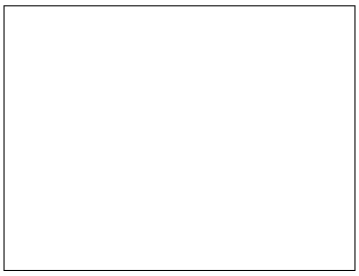
(a) Loss curves for the second model when training patterns are sampled without noise

(b) Loss curves for the second model when trained on patterns generated with a noise level of $\mu = 0$ and $\sigma = 0.5$

(c) Relative increase of sampled patterns when noise is used

Figure 8: The addition of noise during the sampling process increases the prevalence of certain items in the replay dataset, causing them to be learned better at the expense of other items.


(a) Loss curves


(b) Individual item losses

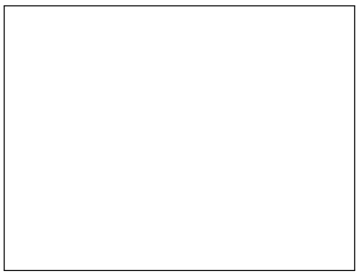

(c) Singular value trajectories

Figure 9: Training metrics for an *overparameterized* first model with a hidden size of 32. In a short amount of time, all items are able to approach a loss of 0.

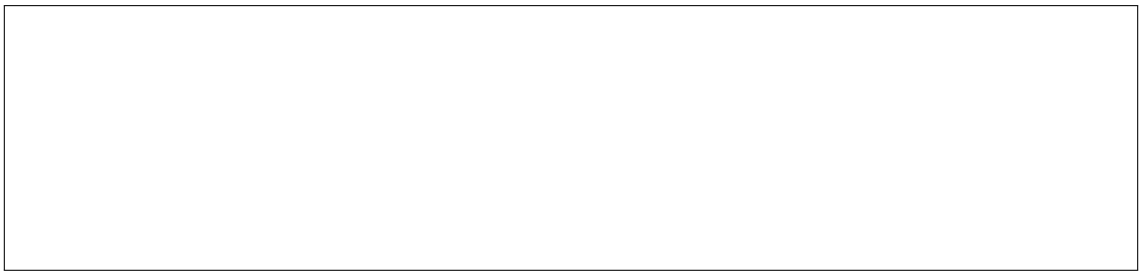


Figure 10: The existence of iterative fixed points and their basins of attraction in the latent space of an overparameterized model with a hidden size of 32. The explained variance by the PCA dimensions were 0.58, 0.21 and 0.12 respectively, adding up to 0.91. Note that some stars (representing the items) lie in areas of a different color; this is due to the grid points being passed through the *decoder* and colored based on the prediction, whereas the items were only passed through the *encoder* to get their position in the latent space. Passing the point corresponding to the item through the decoder too would classify it according to the background color of the points around it.

$t \in [0,\ 150]$ to increase visibility; all loss curves eventually approach 0 as t goes beyond 150. Only 6 out of 8 singular values are nonzero after training.

In parallel to figure 7 produced for the underparameterized model, figure 10 attempts to do the same for the overparameterized one, with the important distinction that PCA is now performed on the hidden dimension of size 32 instead of it directly corresponding to the latent space. The explained variance of the three plotted PCA dimensions was 0.58, 0.21 and 0.12 for each dimension respectively (in ascending order), for a total of 0.91.

Similarly, the three plots in figure 11 convey the same information as figure 8 did for the previous scenario. The addition of noise degrades the quality of the patterns to a lesser extent compared to the sampling performed on the underparameterized model.

7 Discussion

Injecting noise during the sampling process improved the prevalence and learning of under-represented items. In general, it seems that the injection of noise during the pattern generation process allows some items to be learned to a much higher degree, at the cost of degrading the general quality of the predictions. Remarkably, this phenomenon occurred for *both* the under- and overparameterized networks, although the pattern quality was much worse in the underparameterized scenario. This is surprising since the underparameterized model is only able to store very few attractors, as visible in figure 7, and so sampling was not predicted to produce the same effect as for the overparameterized model. Our hypothesis is that the closeness of the items in the latent space and their position within the basin of attraction of an attractor, even if they are not one themselves, accounts for their increased prevalence in the replay dataset. This may be due to the step size of the Markov chain being too small for the state to converge to the global minimum.

Overlap in the 2d latent space. As can be seen in figure 7, the underparameterized model only (in general) had 3 attractor states in its latent space, none of which corresponded exactly to an item from the dataset. For example, *Oak*, *Pine* and *Daisy* are tightly grouped in the leftmost attractor,

(a) Loss curves for the second model when training patterns are sampled without noise

(b) Loss curves for the second model when trained on patterns generated with a noise level of $\mu = 0$ and $\sigma = 0.5$

(c) Relative increase of sampled patterns when noise is used

Figure 11: Even when the first model is overcomplete, a second model can occasionally fail to learn specific items when trained on sampled patterns. The addition of noise can improve the general performance of the second model.

while *Rose* occupies an area further away in the basin. *Salmon* and *Robin* share an attractor, as do *Canary* and *Sunfish*, suggesting that the *red-yellow* mode capturing the color has come to dominate the learning over the *animal-plant* dimension.

Memorization in the overparameterized latent space. While the multidimensional nature of the hidden space makes it difficult to see in figure 10, all items *except* for *Pine* are memorized as attractors in the overparameterized first model. Remarkably, *Pine* and *Oak* were never stored as individual attractors in any of our experiments and instead always shared a common one. This is probably due to them being the most related items in the dataset, as can be glanced from the correlation matrix in figure 3b.

Low individual loss does not entail attractor status. From figure 6b, it appears that the *Rose* item is able to consistently achieve a much lower than average loss. It is therefore quite surprising to see that it actually lies the *furthest* away from an attractor state in the latent space in figure 7 compared the rest. From this, we can conclude that an item having low loss does not justify the prediction that it also encodes an attractor state in the latent space, something which seems like a reasonable expectation.

Every fixed point is also an attractor. All of the iterative fixed points were found to be attractors after evaluation of their Jacobians. In an exploratory analysis, fixed points which were *not* attractors only arose for a large hidden dimension ($\geq$ 64). Interestingly, *spurious attractors* (attractor states $\notin \mathcal{X}$ in an overparameterized model) also started to appear at this level of network capacity.

Differences between the singular values. While the underparameterized model is seen to only have 5 nonzero singular values after training (figure 6c), the overparameterized model has 6 nonzero entries, an increase of only 1 which still leaves 2 dimensions as being irrelevant for model prediction. Upon closer inspection, these two ignored modes were found to code for the *Grow* attribute and the set of attributes which coded only for individual items. While the mode which encodes *Grow* is associated with the *largest* singular value in figure 4, it explains 0 variance when discriminating between items, since every item possesses this attribute. This matches our expectations for the learning dynamics.

8 Conclusions

In this paper, we have shown that a second student model is able to learn an initial hierarchical dataset by training on replayed patterns generated by iterating a first teacher model, which is trained on the dataset, starting from random, uniform noise. Through the intermediate injection of Gaussian noise, this process can be biased away from strong attractors, producing a dataset more fitting for the equal learning of items. We have detailed a general process for navigating the state space of a model in order to sample attractor states, and posited that there exists an optimal noise level for the injections, achieved when the transition matrix produced by iteration most resembles the category structure of the data, which benefits the learning of the second model the most.

While the implications of this may not be significant in terms of model performance given that the second model did suffer a reasonable penalty in accuracy when noise was injected, a way to retrieve knowledge stored in the weights of a model in an unsupervised manner might be interesting for the further study of generative replay from a connectionist standpoint.

9 Recommendations

We now make some recommendations for further work concerning this subject.

Different model architectures. Experimentation with different model architectures is an obvious avenue to pursue. Model depth and layer width can be varied (this work only considered shallow autoencoders with 1 hidden layer), or completely different architectures could be explored. While DAEs incentivize the reconstruction function to resist small but finite-sized perturbations of the input, **contractive autoencoders** instead bestow this property on to the function responsible for feature extraction. This makes the latent space *locally contractive*, a characteristic which might influence attractor formation. Another interesting direction is that of **variational autoencoders**, which learn a multivariate normal distribution that best captures the training set. They are inherently generative, and their probabilistic nature makes them much easier to sample patterns from.

Optimal noise level. As mentioned in subsection 5.4, it remains to be seen what the actual influence of the noise mean μ and standard deviation σ on the learning actually means quantitatively. Evaluating the patterns and learning produced by varying these values might reveal a more concrete relation between the two.

Quantitative metric for hierarchical learning. Since the amount of models we used was so small, we used no quantitative metric to capture the quality of learning, however this would be useful for large-scale experimentation and comparison. This could be accomplished by generating large hierarchical datasets through a process like branching diffusion [18], and training many different models in the fashion presented here. The existence of a suitable metric could allow those results to be interpreted in a meaningful way.

Attractor strength and prevalence in sampled patterns. Lastly, an investigation could be warranted about the potential relationship between attractor strength and the effect the noise injection during the sampling has on the pattern prevalence in the replay dataset. By coupling this with more tests, our hypothesis that noise allows lower-strength attractors to be reached could be confirmed or rejected, since the amount of models used here does not allow such a general relationship to be extrapolated.

References

[1] Magdalena J. Fosse, Roar Fosse, J. Allan Hobson, and Robert J. Stickgold. Dreaming and episodic memory: A functional dissociation? *Journal of Cognitive Neuroscience*, 15(1):1–9, 2003.

[2] Lisa Genzel, Marijn C.W. Kroes, Martin Dresler, and Francesco P. Battaglia. Light sleep versus slow wave sleep in memory consolidation: a question of global versus local processes? *Trends in Neurosciences*, 37(1):10–19, 2014.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] Douglas M. Hawkins. The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12, 2004.

[5] Geoffrey E. Hinton and James L. McClelland. Learning representations by recirculation. In *Neural Information Processing Systems*, pages 358–366, 1987.

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[7] Daoyun Ji and Matthew A Wilson. Coordinated memory replay in the visual cortex and hippocampus during sleep. *Nature Neuroscience*, 10(1):100–107, 2007.

[8] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations, ICLR 2017*, 2016.

[9] James L McClelland, Bruce L McNaughton, and Andrew K Lampinen. Integration of new information in memory: new insights from a complementary learning systems perspective. *Philosophical Transactions of the Royal Society B*, 375(1799):20190637–20190637, 2020.

[10] James L. McClelland, Bruce L. McNaughton, and Randall C. O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102(3):419–457, 1995.

[11] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24, pages 109–165. 1989.

[12] Grégoire Mesnil, Yann N. Dauphin, Xavier Glorot, Salah Rifai, Yoshua Bengio, Ian J. Goodfellow, Erick Lavoie, Xavier Muller, Guillaume Desjardins, David Warde-Farley, Pascal Vincent, Aaron C. Courville, and James Bergstra. Unsupervised and transfer learning challenge: a deep learning approach. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 97–110, 2012.

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8026–8037, 2019.

[14] Roger Ratcliff. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97(2):285–308, 1990.

[15] Timothy T. Rogers and James L. McClelland. *Semantic Cognition: A Parallel Distributed Processing Approach*. The MIT Press, 06 2004.

[16] D. E. Rumelhart, G. E. Hinton, and J. L. McClelland. *A general framework for parallel distributed processing*. 1986.

[17] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning representations by back-propagating errors*, volume 323. 1988.

[18] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. A mathematical theory of semantic development in deep neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 116(23):11537–11546, 2019.

[19] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *ICLR 2016 : International Conference on Learning Representations 2016*, 2016.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[21] Steven H. Strogatz. *Nonlinear dynamics and Chaos*. 1994.

[22] Wenbo Tang, Justin D. Shin, Loren M. Frank, and Shantanu P. Jadhav. Hippocampal-prefrontal reactivation during learning is stronger in awake compared with sleep states. *The Journal of Neuroscience*, 37(49):11789–11805, 2017.

[23] Gido M. van de Ven, Hava T. Siegelmann, and Andreas S. Tolias. Brain-inspired replay for continual learning with artificial neural networks. *Nature Communications*, 11(1):4069–4069, 2020.