State Denoised Recurrent Neural Networks

by

Denis Kazakov

B.S., University of Colorado, Boulder, 2017

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Master of Science

Department of Applied Mathematics

2018

This thesis entitled:
State Denoised Recurrent Neural Networks
written by Denis Kazakov
has been approved for the Department of Applied Mathematics

_____

Prof. Michael Mozer

_____

Prof. Jem Corcoran

_____

Prof. Stephen Becker

Date _____

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Kazakov, Denis (M.S., Department of Applied Mathematics)

State Denoised Recurrent Neural Networks

Thesis directed by Prof. Michael Mozer

We investigate the use of attractor neural networks for denoising the internal states of another neural network, thereby boosting its generalization performance. Denoising is most promising for recurrent sequence-processing networks (i.e. recurrent neural networks), in which noise can accumulate in the hidden states over the elements of a sequence. We call our architecture *state-denoised recurrent neural network* (SD-RNN). We conduct a series of experiments to demonstrate the benefit of internal denoising, from small experiments like detecting parity of a binary sequence to larger natural language processing data sets. We characterize the behavior of the network using an information theoretic analysis, and we show that internal denoising causes the network to learn better on less data.

Dedication

To my wonderful family - Liudmila, Pavel, and Max Kazakov - for supporting and encouraging all of my pursuits.

Acknowledgements

I am infinitely grateful to my advisors: Dr. Michael Mozer for picking me up under his wing since Spring 2016, Dr. Jem Corcoran for shining an optimistic beam of light on everything that we ever discussed. Attending Dr. Mozer's lab meetings taught me a great deal about research and that there are always more questions to pursue. I thank professor Stephen Becker for agreeing to be on my committee and for always being so supportive of what students are working on.

I want to thank the departments of Applied Mathematics and Computer Science at CU Boulder. If it was not for the myriad of amazing students and professors that I met during my 5 years of school, my education would have been substantially dimmer. I thank professor Elizabeth Bradley for believing in my ability my freshman year and letting me work on her research project. I want to thank professor Chris Ketelsen for being the first professor in my college career to so thoroughly shine light onto how math is used in the world and for sparking my curiosity for math that has lasted me ever since. I thank professor Scott Douglass for organizing Engineering Honors Program, for his infinite investment into students' growth, and for letting me be part of such an amazing community. I thank professor Richard Han and Zack Nies for opening my eyes on the world of entrepreneurship. I am especially grateful that CU Boulder had so many theorists, philosophers, and probabilists seeing the world beyond the veil of normality, noting the beauty and elegance of thought and of the world that we live in.

To all the people that have played any role in my personal and professional development in the past 5 years, you have been my support, my lesson, my cornerstone. Thank you for coming into my life and I wish all of you the best of everything.

Contents

Chapter

Figures

Figure

Chapter 1

Introduction

Noise robustness is critical in statistical models. Machine learning models, a subset of statistical models, are expected to generalize their learned problem solution to unseen and oftentimes corrupted samples from the problem distribution. However, it is an infamously difficult challenge for machine learning models to apply their features to corrupted input. For example, even industry standard face recognition models often struggle with recognizing faces as the same in an unusual light.

Figure 1.1: An example of how the same face can look in varying light conditions.

Source: http:///nofilmschool.com/2017/05/watch-10-ways-create-different-moods-lighting

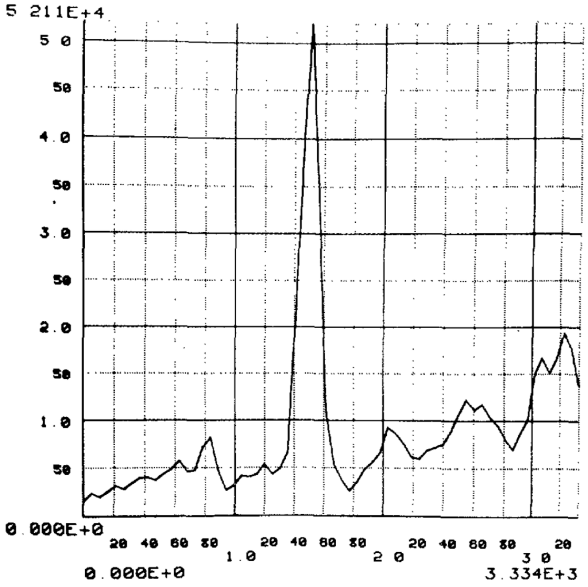
Noise in the input can cause an abnormal mapping of the input to the model's feature space, detecting irrelevant/abnormal features, thus undermining the model's decision making algorithm.

A partial remedy to this problem has been to reduce the input noise via hardware by installing additional sensors into phones such as infrared lighting for providing a consistent face illumination

and a "dot projector" to measure the 3D profile of a face. [1]

An alternative to a hardware-based input denoising would be an algorithm-based input denoising. For example, we could represent the image in terms of its Fast Fourier Transform (FFT) decomposition and remove the high frequency components that are often associated with noise. In the sound domain, a similar idea was used by Boll 1979 [2] to suppress acoustic noise in a speech processing system. By identifying an average FFT profile of a noise to remove (helicopter sound), it became possible to subtract it from the signal's FFT profile; thus, getting a noise-suppressed sound signal.

Figure 1.2: Averaged FFT decomposition of a helicopter sound (From Boll, 1979 [2])



More recently in machine learning, loss functions have been explored to achieve invariance to task-irrelevant perturbations in the input [3, 4]. Input noise removal methods described above are suitable for handling external input noise, but we argue in this thesis that **internal** noise can be an even greater challenge.

To explain what we mean by internal noise, consider a deep-net architecture in which representations are successively transformed from one hidden layer to the next. To the degree that the network weights are not precisely tuned to extract only critical features of the domain, irrelevant

features may be selected and have the potential to interfere with subsequent processing. Recurrent architectures are particularly fragile because internal-state dynamics can amplify noise over the course of sequence processing [24]. In this article, we propose a suppression method that improves the generalization performance of deep nets and test its effectiveness on recurrent networks, where we expect the effect to be most pronounced.

Suppressing variability facilitates **communication** in information-processing systems. Whether we are talking about groups of humans, regions of a single brain, components of an articulated neural net architecture, or layers of a feedforward network, in each case information must be communicated from a **sender** to a **receiver**. To ensure that the receiver interprets a message as intended, the sender should limit its messages to a canonical form that the receiver has interpreted successfully in the past. Language serves this role in inter-human communication. We explore noise suppression methods to serve this role in inter-state communication in deep-learning models.

We recall a constraint-satisfaction method traditionally used for cleaning up noisy representations - an **attractor neural network**. We propose integrating attractor networks into deep networks, specifically recurrent networks, for denoising their internal states. We present a series of experiments on small-scale problems to illustrate the methodology and analyze its benefits, followed by experiments on more naturalistic problems which also show reliable benefits of denoising, particularly for small training sets.

Chapter 2

Background

In this chapter, we go over the prerequisite knowledge we need to know in order to proceed with our work. We discuss deep learning methods for sequence processing, as well as provide an introduction to information theory.

2.1 Deep Learning

Deep neural networks have revolutionized many supervised and unsupervised tasks including sequence processing.

The main advantage of deep learning is its ability to develop useful-for-the-task features with little to no guidance. This can be, for example, image filters in image processing networks (e.g. Convolutional Neural Networks [5]), or sequential patterns in sequence prediction networks (e.g. Recurrent Neural Networks [6]). By tracking those patterns and knowledgeably combining them, deep learning models can interpret the content and solve tasks of remarkable complexity.

2.1.1 Recurrent Neural Network

In particular, we are interested in the Recurrent Neural Network (RNN) architectures since their recurrence presents an opportunity for internal-state noise to amplify over the course of sequence processing. RNN is best thought of as a layer of "cells" (i.e. neurons) such that each cell (i) receives some input, (ii) computes an internal state that the cell will "remember", and (iii) produces some output. In particular, we have the following.

(1) The **input** to each cell, denoted as x, is defined as a nonlinear activation of a weighted combination of all the previous states, and an event input that is also distributed among cells.

(2) The internal **hidden state**, denoted as s, is computed according to the rules of the chosen RNN architecture. This is the most important part of the model where we define how the model is going to treat and interact with the information it is presented with.

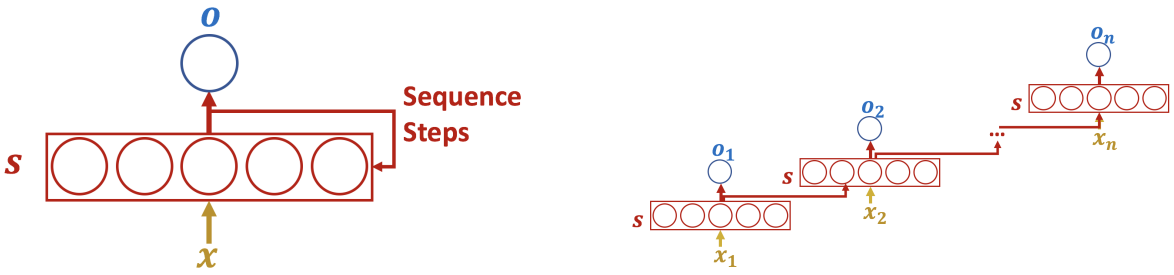
(3) The newly computed hidden state is processed in two ways.

   (a) The hidden state is used to compute an **output** value, denoted by o.

   (b) The hidden state is used as a recurrent input for next sequence step.

These steps are depicted in Figure 2.1.

Figure 2.1: Recurrent Neural Network and Recurrent Neural Network unrolled over the sequence computation.



RNNs use an assumption that the history of events (i.e. cells inputs) in the sequence contains relevant information how to interpret current event. Relevant parts of that history are represented as numerical values in an internal state of each cell. In RNNs, every cell is recurrently connected to every other cell. Combining that trait with its potential to propagate information about the past for long segments of the sequence, RNNs becomes a model that can find complex dependencies and interactions in sequences of interest.

It is easiest to understand the intuition behind RNN through an example. Suppose, we are interested in learning the syntax and spelling of the English language. In this case, our sequence consists of alphabet letters, punctuation, and spaces. Then, using a RNN model, we can try to predict the next letter based on the observed history of letters so far. If we set up the problem this way, we will learn the spelling of words, tenses (e.g. suffixes to change the verb tense), punctuation rules etc. [17]. All of those grammar rules are tracked, enforced, and represented through some "features" (i.e. individual neurons or combinations of neurons) of the RNN model.

2.1.2 GRU cell

In this work, we use a popular variant of RNN architecture which is called a Gated Recurrent Unit (GRU). There exist alternative cell architectures such as Long-Short Term Memory (LSTM) [18] and variations of it. However, our work lies outside of specific cell architecture choice. Both GRU and LSTM cell architectures use a variant of a gating mechanism that addresses a well-known vanishing gradient problem of RNNs [19], [20].

GRU incorporates the same recurrent principle of RNN, but in a gated manner. In particular, at every sequence processing step, each cell has an opportunity to either (i) shunt its previous hidden state using a *reset gate* r if it wants to selectively consider information from the past when computing (i.e. detecting) relevant signal q, or (ii) use an *update gate* s to choose how much of the previous hidden state to copy directly and how much to update with newly detected signal. From this formulation, it is notable that there is a way for the GRU cell to propagate previous hidden state without modifying it at all if that is helpful to the task, which directly addresses the vanishing gradient problem. Formally, given an external input x_k at step k and the previous hidden state h_{k-1}, the GRU layer is updated as follows.

(1) **Determine reset gate settings:**

$$r_k \leftarrow \text{logistic}(W_r x_k + U_r h_{k-1} + b_r)$$

(2) **Detect relevant event signals:**

$$q_k \leftarrow \tanh(W_q x_k + U_q(r_k \circ h_{k-1}) + b_q)$$

(3) **Determine update gate settings**

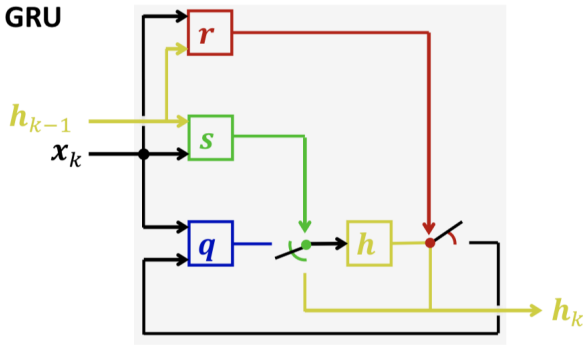
$$s_k \leftarrow \text{logistic}(W_o x_k + U_o h_{k-1} + b_o)$$

(4) **Update hidden state**

$$h_k \leftarrow s_k \circ h_{k-1} + (1 - s_k) \circ q_k.$$

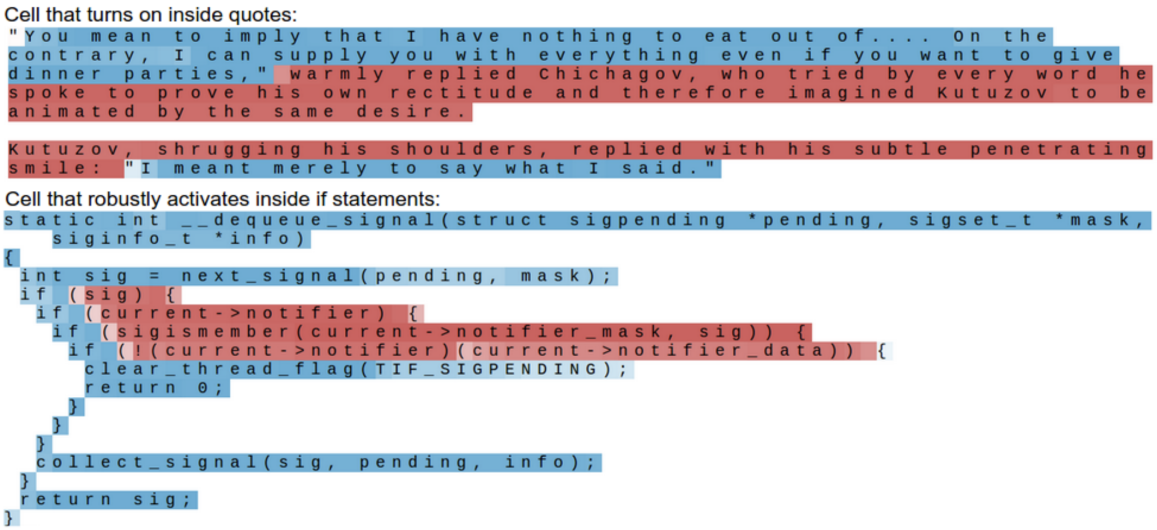
Here, W_r, U_r, b_r are the *reset gate* weights, W_q, U_q, b_q are the *relevant signal detection* weights, and W_o, U_o, b_o are the *update gate* weights. A depiction of how these GRU components interact with each other is given in Figure 2.2.

Figure 2.2: A diagram of how all GRU components interact between each other (From Mozer et al, 2017 [22])



As a result of overcoming the vanishing gradient problem, Gated RNN models can capture sequential patterns of impressive complexity and span. For example, in Figure 2.3, we see activation strength (represented via color) of a few hidden units from trained RNNs. The first hidden units is taken from a text generating RNN. The second hidden unit is from a code generating RNN. It is clear that the range at which gated RNNs are detecting features can be quite large, spanning across about 100 sequence elements in the first example.

Figure 2.3: Activation strength (represented via color) of some hidden units from trained RNNs. (Source: Karpathy, 2015 [21])



2.1.3 Tanh cell

A much simpler cell architecture that we use for small synthetic experiments is a cell without any gating mechanism that relies exclusively on recurrence.

(1) **Detect relevant event signals:**

$$h_k \leftarrow \tanh(W_q x_k + U_q h_{k-1} + b_q),$$

where W_q, U_q, b_q - relevant signal detection weights

2.1.4 Optimization

Any neural network is designed with an intention to *learn* a set of features that are useful for solving a particular task. The network learns those features via a gradient based optimization process called back-propagation. [10] Back-propagation solves an important problem of credit assignment for distributing the error gradient of the model's current performance to the weights that contributed to that error.

In our work, we use a popular and effective ADAM optimizer [11]. ADAM stands for ADAptive Moment estimation. Besides using back-propagation to assign error gradients to each parameter we are optimizing with respect to, ADAM also uses exponential moving averages to estimate moments for each parameter and assigns individual learning rates to parameters. Intuitively, if the parameter's error gradient consistently points in one direction, its learning rate will increase; whereas, if the error gradient is noisy during training, the learning rate will decrease.

2.1.5 Dropout

Dropout has been proposed as a regularization method for deep neural networks by Srivistava et al. [12]. it is a randomized method of regularization that may be applied to models trained via SGD. With probability p, this method will "keep" the output of hidden units of the network. Accordingly, the hidden units that were not "kept" are "dropped".

More precisely, during each iteration of the network training, the optimizer will set the output value of each hidden unit participating in the dropout to zero with probability $(1-p)$ independently at random. The output of units that were not set to zero is scaled by a factor $\frac{1}{p}$ in order to maintain the expected mean of that layer's output as if no dropout was used.

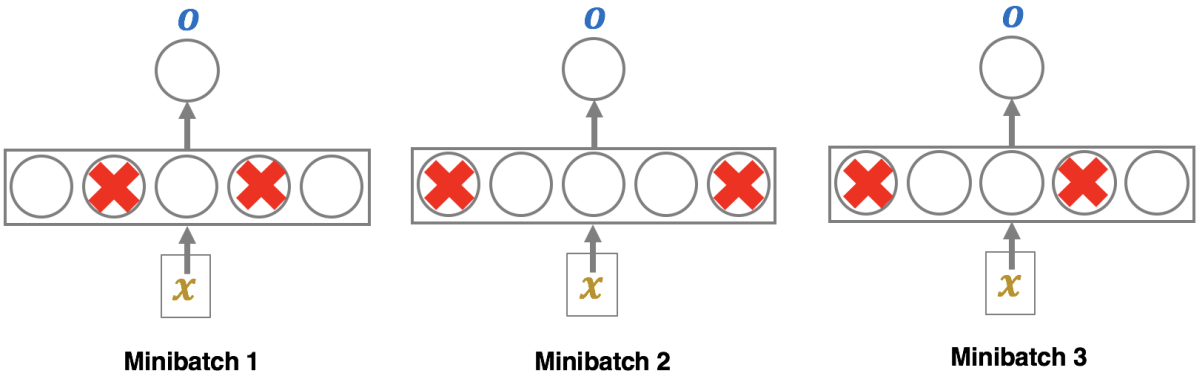
Dropout is only applied during the training procedure of the network. When we evaluate the network on the test set or use it in practice, we use the output of all hidden units.

This seemingly modest modification results in a fundamental shift of network's training objective. In particular, dropout is reducing the expected cooperation between hidden units of the network. Since all hidden units within the layer where dropout is applied may be "dropped", hidden units may not form complex relationships between themselves and are instead incentivized to learn features that work independently of each other.

A particularly illuminating way to look at dropout is to see it as ensemble training as proposed by Baldi and Sadowski [13]. Dropout randomly chooses a collection of hidden units from a given layer, and then a gradient based optimizer updates the weights of the network that work for that hidden unit collection. Therefore, we are essentially training a collection of networks that all share

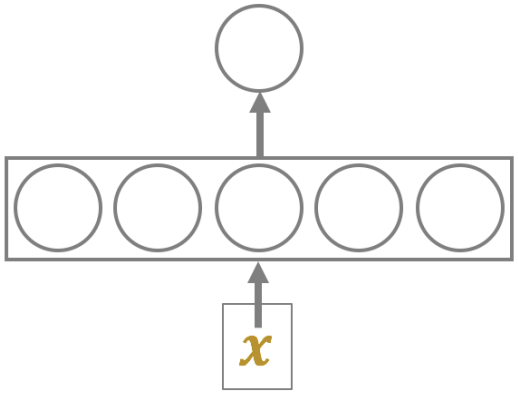
weights between each other, but are meant to work well on their own.

Figure 2.4: An ensemble of feed-forward networks during network training with Dropout of probability $p = \frac{3}{5}$. Red cross indicates a "dropped" hidden unit.



When we no longer apply dropout at test data set evaluation and when we use the network later in practice, we are combining the predictions of all those partial networks into one, thus creating an ensemble prediction as depicted in Figure 2.5

Figure 2.5: A combination of all partial networks' prediction is formed by not applying dropout.



2.1.6 Word Embeddings

In this work, we consider Natural Language Processing (NLP) as our real world problem domain for RNNs. It became standard to use pre-trained dense word vector embeddings in NLP

deep learning community. Therefore, we explain in detail how word embeddings are produced and their significance.

Unlike continuous feature settings such as computer vision, NLP works in a mostly discrete feature setting (words, part of speech tags etc.). One way to address this is to represent each word as a sparse representation using *one-hot vectors*. In that case, all vector elements are set to zero, except a single value, which is set to 1 to indicate that word's unique ID. This way, each one-hot vector could be considered as an independent feature that the model will learn how to work with.

However, there exists is a lot of codependency between NLP features that one-hot vector representation is not capturing.

Consider a task of speech synthesis. We want to train a model that would learn how to generate grammatically correct sentences. Words like "cat", "dog", and "hamster" are all used to represent animals and are expected to be used in similar language context. Therefore, if we use a word representation that captures contextual similarities, we can use one piece of information (e.g. observed individual sentence "I like *cats*") to learn how to generate sentences with animals in general (e.g. "I like *dogs*", "I like *hamsters*"). Whereas, independent word representation (i.e. one-hot vector) would need to observe all 3 sentences (i.e. "I like *cats*", "I like *dogs*", "I like *hamsters*") in order to learn to generate them correctly.

This assumption of feature non-independence is exactly the premise of word embeddings. *Word embeddings* introduced by Mikolov et al. 2013 [14] is a way to represent words as dense vectors, where each vector's value represents individual feature (e.g. type of animal, object's usual color, word's sentiment if any etc.). This way, each word's n-dimensional embedding is simply a point in the n-dimensional feature space. Mikolov et al. 2013/1/16 [15] introduced effective methods to train an embedding feature space and words' position in it. In particular, they make use of **Skip-gram** and **CBOW** models which we now review.

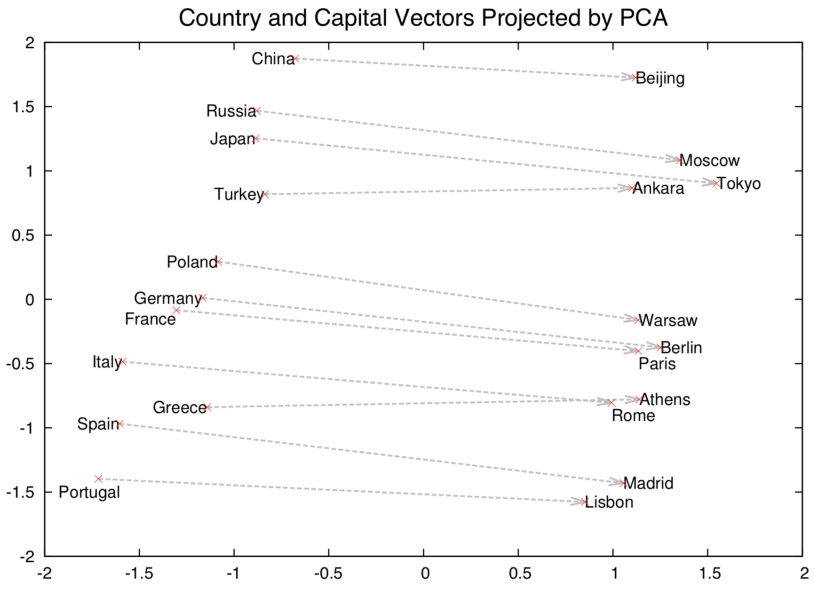
A **Skip-gram** model learns word embeddings that can predict the word's context only from that word's embedding. Formally, given a sequence of length T of words as they appear in text (i.e. each word is its natural context), define the words' respective embeddings to be $w_1, w_2, ..., w_T$.

Then, the Skip-gram model will maximize the average log probability

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t),$$

where c is the window of context we consider as relevant for each word, w.r.t words' embeddings. This is depicted in Figure 2.6 where we show PCA projection of word embeddings for countries and their capitals. Notably, the lines drawn between each pair are close to parallel, implying we could offset the embedding for each country by the same amount in the same direction within the word embedding feature space and get country's capital. This implies consistency and independence in the embedding feature space.

Figure 2.6: PCA projection of word embeddings for countries and their capitals. (Source: Mikolov et al, 2013 [14])



The **CBOW**(continuous bag of words) model learns word embeddings that can predict a word only from its context. Formally, given a sequence of length T of words as they appear in text, define the words' respective embeddings to be $w_1, w_2, ..., w_T$.Then, CBOW model will maximize

average log probability

$$\frac{1}{T}\sum_{t=1}^{T}\log p(w_t|w_{t-c},...,w_{t-1},w_{t+1},...,w_{t-c}),$$

where c is the window of context we consider as relevant for each word, w.r.t words' embeddings.

2.2 Information Theory

Information theory came into existence from the work by Shannon [7] as a means to quantitatively and qualitatively analyze the transmission of messages (i.e. information) from "source" to "destination". Measure of information (entropy) from Shannon's work is particularly relevant to us.

2.2.1 Entropy

To define how uncertain we are about the outcome of some event, Shannon defined entropy. Entropy is a mathematical measure of uncertainty that quantifies the amount of information in a stochastic source of data.

Definition: For a given set of n possible events that occur with probabilities $p_1, p_2, ..., p_n$, entropy H is defined by:

$$H = -\sum_{i}^{n} p_i \log_2 p_i \tag{2.1}$$

Note, that the base of log is chosen as 2 for convenience. It allows us to set units of measurement of entropy as *bits* (i.e. either 1 or 0).

H has the properties of a measure such as:

(1) continuity in the p_i space

(2) if all $p_i = \frac{1}{n}$, H is a monotonically increasing function of n.

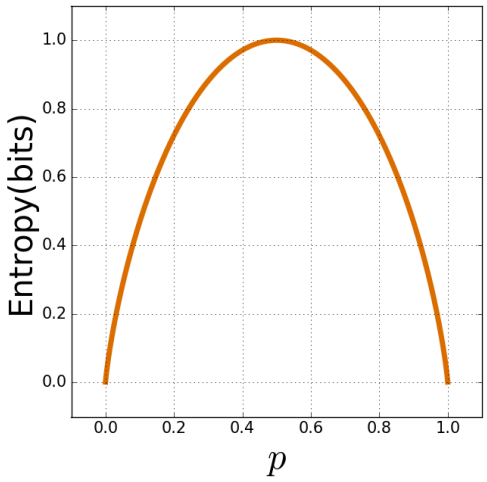
To demonstrate how entropy works on an example, suppose that we have a biased coin (i.e.

random variable X) that flips heads with probability p, and tails with probability $(1-p)$.

$$z = 0; \qquad V_z = \begin{cases} heads, & p \\ \\ tails, & (1-p) \end{cases} \tag{2.2}$$

If $p = 0$ or $p = 1$, we are guaranteed to see accordingly tails or heads all the time. Intuitively, in such scenarios, we have no uncertainty (i.e. entropy is zero). However, if $p = 0.5$, we are guaranteed to see tails or heads with equal probability as number of flips goes to infinity. Intuitively, the uncertainty of what our coin will flip to is maximized in this scenario.

If we now plot the value of entropy for our biased coin as a function of p, we see that our intuition still holds up:



2.2.2 Information Theory in Deep Learning

The framework of information theory could be used in the context of deep learning. We can approximate the bounded continuous output range of hidden neurons by simply binning them into a fixed number of evenly spaced buckets as done in [8].

By treating the layers of neural network as multidimensional random variables that produce different values depending on the input, we can use standard definitions of entropy and mutual information to gain insight about or control the model's behavior. For example, we could measure

how much information about the input is preserved at each layer [8], or add a regularization formula that aims to maximize mutual information between the input generation control variable and the produced output of a generative neural network model.[16]

Chapter 3

Related Work

There has been a few notable directions in exploring the theme of modifying recurrent network sequence step update and viewing noise within neural network models in general. We will briefly describe them in this section.

3.1 Adaptive Computation Time

Adaptive computation time (ACT) introduced the idea of giving each sequence update a "budget" of computation. Once the budget is exhausted, next sequence input is presented. The budget is modified by the network through a differentiable computation. This way, RNN could decide to use more than 1 recurrent update before proceeding onto the next sequence element. The motivation for this was that if the sequence element is "difficult", we want to give our RNN model more time (i.e. updates) to decide how to process that element. [25]

3.1.1 Information Bottleneck

The Information Bottleneck (IB) method was introduced by Tishby et al [9], and later was applied in neural networks context for interpreting the process of neural net training [8]. IB is formulated as an optimization problem where we are minimizing mutual information of the model's representation T with its input X - $I(X;T)$), while maximizing mutual information of representation T with output variable of interest (i.e. model prediction) Y - $I(T;Y)$. This way, we are searching for the simplest model that makes the best output prediction.

IB formulation and motivation is best understood through the definition of a *minimal sufficient statistic*. A *statistic* is a measure of the distribution sample that contains information about some parameter for that sample. A statistic is *sufficient* if no other statistic could provide additional information about the sample's parameter of interest. A statistic is *minimally* sufficient if it can be represented as a function of any other sufficient statistic. (That is, it most efficiently captures all possible information about the parameter of interest.) Therefore, IB is trying to approximate the minimal sufficient statistic. It maintains *sufficiency* by maximizing $I(T;Y)$, while searching for the simplest (i.e. "most minimal") statistic with minimizing $I(X;T)$.

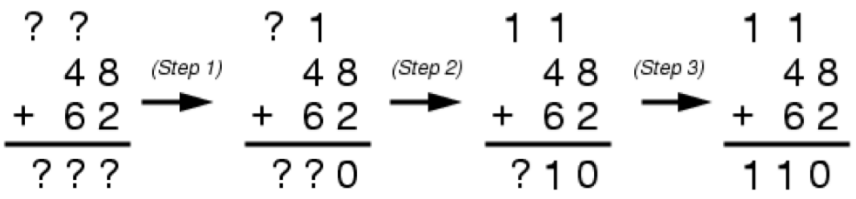
3.2 Cleanup of States

The idea of state cleanup is the most related conceptually to our current work. This work demonstrates a clear benefit from "cleaning up" the representation passed forward in recurrent networks. [24]

Seeing the complexity and span that some features of RNNs capture in Chapter 2, it is easy to realize that noise can easily accumulate over such long sequence patterns. Mathis, Mozer 1995 [24] demonsrate in their work exactly the effect of what happens if such noise is not removed on a simple RNN task.

The task recurrent network is solving in this work is a step-wise addition computation. An example is shown in Figure 3.1.

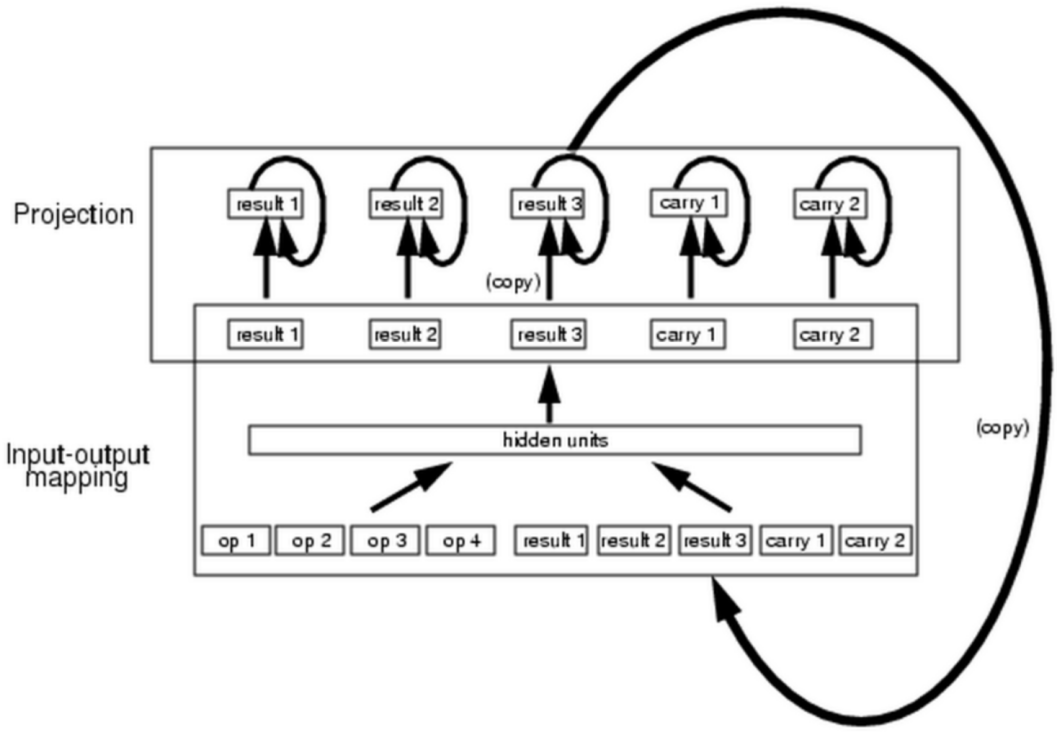
Figure 3.1: Sequence of steps done to add two numbers together. (Source: Mathis and Mozer, 1995)



We are representing model's partial solution as 5 one-hot embedding cells (3 for the "result"

numbers that the model will produce, 2 for "carry" flags at each position). This is depicted in Figure 3.2.

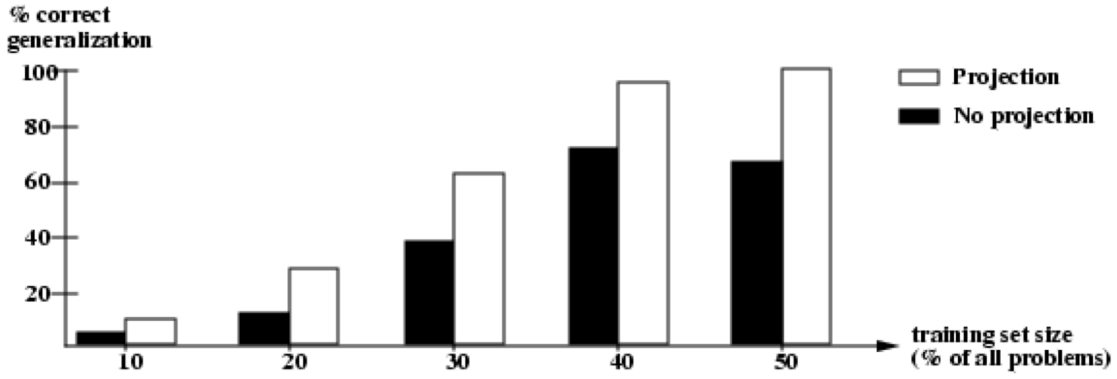
Figure 3.2: from (Mathis and Mozer (1995)); A diagram of the recurrent network model.



The "cleanup" this model can do then is simple - instead of passing a partial solution that is a distribution of confidence in each "result" and "carry" cell, assign all of the activation to the most confident choice in those cells. This way, we are using a "winner take all" heuristic to denoise the intermediate state.

We then see in Figure 3.3 that the denoised recurrent network formulation generalizes better.

Figure 3.3: **Projection** - the denoised model, **No projection**- original model. (Source: Mathis and Mozer, 1995)



Chapter 4

Attractor Networks

In 1982, J.J. Hopfield introduced an idea of *attractor neural networks*. Motivated by the idea of memory recall, attractor network is designed to recover stored vectors (i.e. *attractors*) from some associated stimulus such as noise corrupted or only partially observed attractor vector.

4.1 Intuition

An attractor recurrent network can be seen as computing a sequential iteration $a_1, ..., a_n$, where our initial state a_1 acts as the "memory stimulus", and the final converged state a_n is the recovered attractor state. The iteration is guaranteed to converge to a fixed point as shown by Koiran [23].

Using an energy function to map every state a_i to an energy value defined by a_i's orientation within the attractor transformation weights W and b, it can be shown that the energy function E is a Lyapunov function. Therefore, the recurrent iteration of an attractor recurrent network is guaranteed to converge to a cycle of limit 2, which in turn is a fixed point.[23]

4.2 Attractor Network Formulation

If our goal is to reconstruct the attractor point only to a certain degree of precision, it is not necessary to wait until the attractor iteration converges to a fixed point. Instead, we can simply run the attractor iteration for a fixed number of steps.

Having such an objective saves us some computation time at the cost of introducing an

uncertainty about whether the attractor iteration has converged to an acceptable error radius within the fixed point objective by the end. However, we later demonstrate empirically that such uncertainty is not hindering the performance in practice.

Here, we can present a complete formulation of an attractor network.

(1) **Initialize attractors:**

$a_{1...N} \sim Uniform(-1, 1)$,

(2) **Corrupt attractors**

$x_i := a_i + \alpha_{noise}\mathcal{N}(\mu = 0, \sigma = 1)$, for $i \in 1, ..., N$

(3) **Train attractor model on noise removal objective**

x_{clean_i}, x_i , a_i
$n\times1$ $n\times1$ $n\times1$

x_{bias}, h_t
$m\times1$ $m\times1$

b_I , b_O U_I , W , U_O $\sim \mathcal{N}(0, 0.1)$
$m\times1$ $n\times1$ $n\times m$ $m\times m$ $m\times n$

for $i \in 1...N$:

(a) Initialize: $h_0 = 0$

$x_{bias} = b_I + U_I\sigma^{-1}(x_i)$

(b) Update for $t \in 1, ..., T$:

$h_t = W\sigma(h_{t-1}) + x_{bias}$

(c) Output:

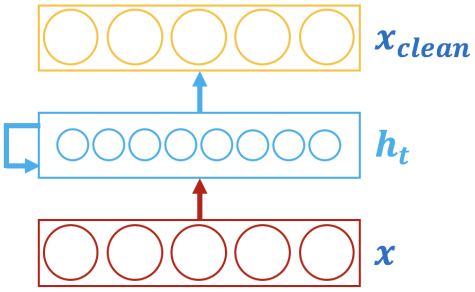
$x_{clean_i} = \sigma(b_O + U_O h_T)$

Optimize using back-propagation ($\frac{\text{noise removed from corrupted attractors}}{\text{noise added to corrupt attractors}}$)

$$\begin{aligned} \min_{b_I, b_O, W, U_I, U_O} \quad & \frac{\sum_{i=1}^{N}(x_{clean_i} - a_i)^2}{\sum_{i=1}^{N}(x_i - a_i)^2} \\ \text{subject to} \quad & W_{ij} = W_{ji} \quad W_{ii} > 0 \end{aligned} \tag{4.1}$$

These steps are illustrated in Figure 4.1.

Figure 4.1: Attractor Recurrent Network



4.3 Using an Attractor Network for State Denoising

To verify that our intuition about how an attractor network will recover stored attractors is correct, we conduct a simple experiment. Following our outlined algorithm from the previous section and optimization objective (4.1), we initialize N attractors in $n = 20$ dimensional space.

In this experiment, we vary 2 variables. One is m which is the dimension of the attractor dynamics space. The second is N which is the number of attractors we are trying to recover. We vary these variables while measuring optimization objective (4.1), which is a direct normalized measure of how closely we were able to reconstruct our original attractors.

In Figure 4.2, we show the performance of the attractor network at removing noise and in Figure 4.3 we show the cumulative distance traveled by each attractor iteration trajectory through time.

Figure 4.2: Noise removal objective (4.1) vs.N(horizontal), m (color)

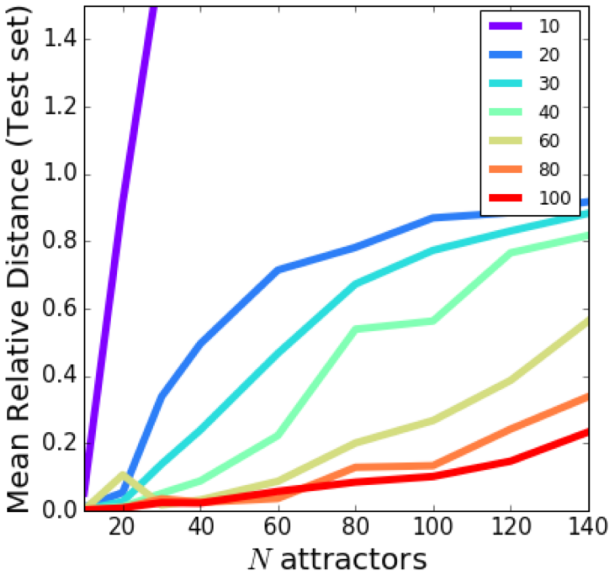
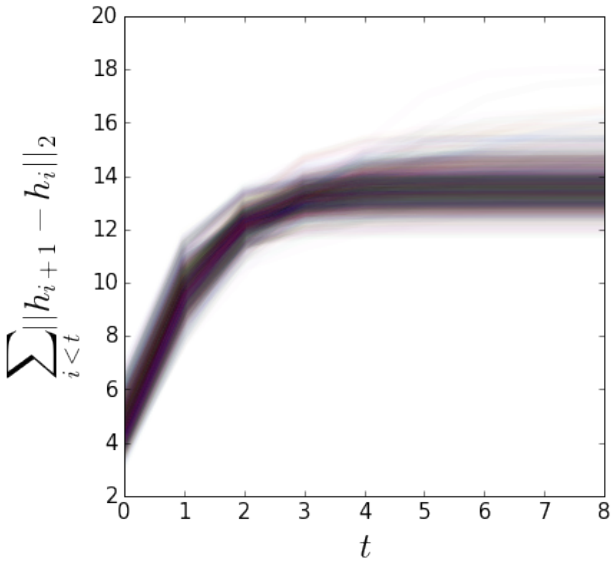


Figure 4.3: Cumulative distance traveled by each attractor iteration trajectory up to time t



From this experiment, we conclude that the attractor network can indeed reconstruct the attractors stored in it within a limited number of attractor steps t taken, and that the performance of reconstruction degrades gracefully with the number of attractors N rising.

Another interesting observation to make is that the higher is the dimensionality m of the attractor matrix $\underset{m\times m}{W}$, the better is the performance for large number of attractors N. That is explained by the fact that each attractor point has a *basin of attraction* around it with a radius of α_{noise}. The larger N is, the more chances there are for those basins of attraction to overlap between multiple attractors, forming a new energy minimum at the interpolation between the attractor points whose basins of attraction overlap. This will degrade the performance of the network.

Chapter 5

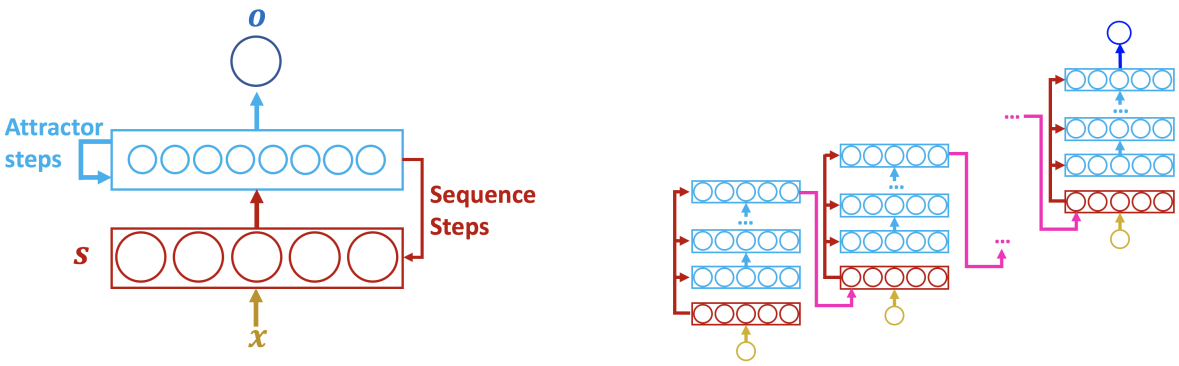
State Denoised Recurrent Neural Network

Now that we have demonstrated the effectiveness of attractor networks in recovering noise corrupted attractor states, we will integrate the attractor network into a sequence processing recurrent neural network (RNN).

Since our motivation is to remove the potential noise acquired during sequence processing and to standardize the internal states of the RNN, we will continue to formulate the objective of the attractor network as to remove some noise α_{noise}. However, we will now do so from the observed internal states of the RNN.

The idea is to run a standard RNN, but before recurrently feeding in the internal state to the next sequence processing step, we run an attractor network to denoise the internal state. We call this formulation a State Denoised Recurrent Neural Network (SD-RNN). Figure 5.1 illustrates this type of network.

Figure 5.1: SD-RNN and SD-RNN unrolled over the sequence.

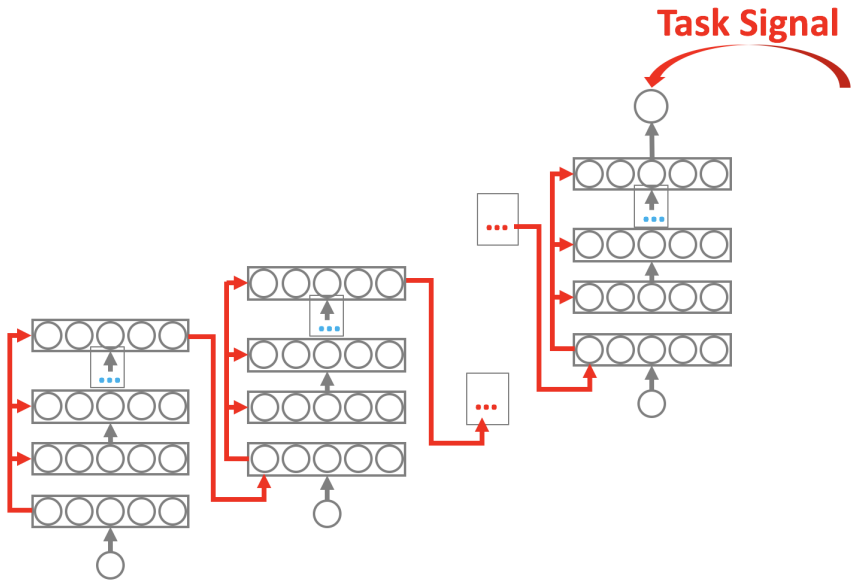


5.1 Training Procedure

It is important to realize that in the formulation of SD-RNN, we have two separate objectives that we need to optimize.

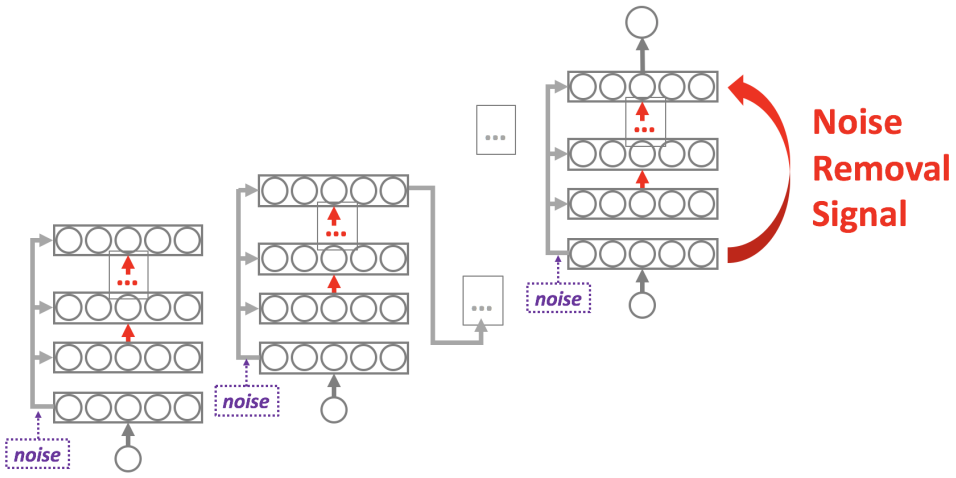
(1) Like any machine learning model, SD-RNN has to optimize the **task objective** That is, one must classify the input and label each element within the sequences.

Figure 5.2: An illustration of a task objective.



(2) Uniquely to SD-RNN, there is also a **state denoising objective** to optimize. Attractor state denoising objective is defined by formulation (4.1), where attractors used are the most recent internal states obtained during task objective training. To keep only the most recent internal states as attractors, we add standard L_2 regularization to the attractor weights W.

Figure 5.3: An illustration of the state denoising objective.



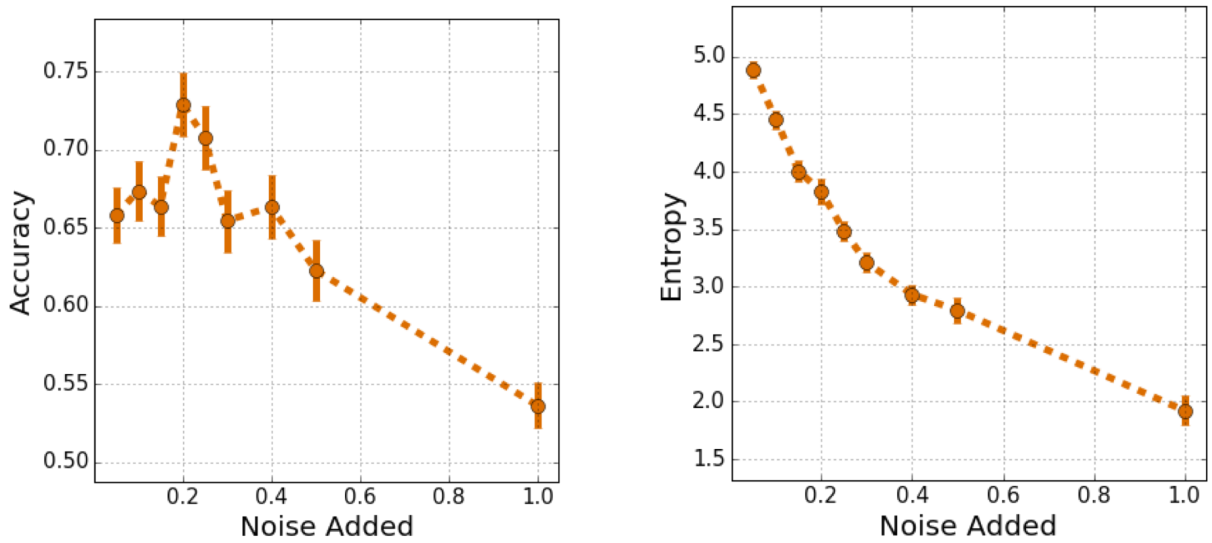
Therefore, we introduce a method of training for SD-RNNs, where we alternate between optimizing *task weights* on the **task objective** and *attractor weights* on the **state denoising objective**. Currently, the SD-RNN training procedure simply alternates between taking one update step for each training objective.

5.2 Basin of Attraction

To verify our intuition of how denoising works in SD-RNN, we vary the amount of α_{noise} added to change the radius of the basin of attractor convergence in Figure 5.4. Intuitively, there is some noise around the internal states of the network, the removal of which helps the network achieve and use a clearer internal state representation. However, by increasing the basin of attraction beyond the radius of noise, we also start collapsing clusters of internal states that were meant to be distinct.

Therefore, we expect there to be an optimal α_{noise} to be added such that the attractor network will remove the noise, but not collapse the internal states. Additionally, we expect the entropy of internal states to be decreasing the more collapsing our attractor network is doing. Indeed, we observe both of those results on a simple synthetic task of *parity classification* discussed in detail later.

Figure 5.4: Performance is increasing until we start collapsing internal states. Entropy is decreasing monotonically.



5.3 Synthetic Experiments

To verify that our formulation of SD-RNN is actually helpful in practice, we compare the performances of 3 models.

(1) Standard RNN: In plots, we will refer this model as **No Attractor**.

(2) SD-RNN: In plots, we will refer to this model as **Attractor**

(3) SD-RNN but training both *attractor weights* and *task weights* on the task objective alone: In plots, we will refer to this model as **Attractor (on task)**.

We make these comparisons on 100 replications of 4 synthetic tasks and observe consistent results that SD-RNN performs best, standard RNN comes second, while SD-RNN (on task) fails to make use of the additional architecture we added on top of RNN. In our synthetic experiments, we use Tanh RNN cells to reduce the effect from using a complicated cell architecture.

By comparing all 3 model variations, we are making sure that the increase in performance is due to state denoising, instead of simply having extra model parameters.

5.3.1 Experiment Practices

To ensure that our comparison is as fair as possible, we employ several practices.

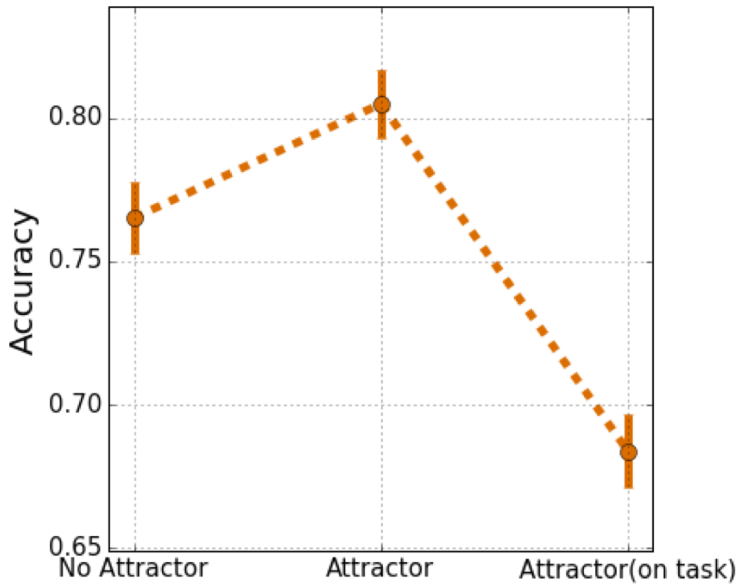
- We are resetting random seeds for our experiments. When you set a random seed, you are guaranteed that the same random numbers will be generated. Therefore, we are guaranteed the same datasets to be generated across all 3 models we are trying to compare.

- We evaluate the performance of each model on 100 replications. The datasets are the same across models to guarantee a fair comparison, but different across replications to introduce some diversity in each replication. For error estimates, we use standard error (SE) $\sigma_{SE} = \frac{\sigma}{\sqrt{n}}$ where n is the number of replications and σ is the standard deviation of the population.

5.3.2 Majority Task

The Majority Task involves processing a sequence of binary digits and outputting 1 if the majority of digits were 1's and 0 if the majority were 0's. In this task, we chose to work with sequences of length 12. This way, there are a total of 2^{12} possible sequences we could possibly generate for each replication. Therefore, for each task replication, we generate a training set of size 32, and let the test set consist of the 4064 remaining possible sequences. In Figure 5.5 we compare the performance of the three models. We see a trend that will continue throughout our synthetic experiments: SD-RNN (**Attractor**) model outperforms the baseline RNN model (**No Attractor**), and SD-RNN trained without noise removal objective (**Attractor(on task)**) fails to make use of the additional cell architecture it has and is losing to both other models.

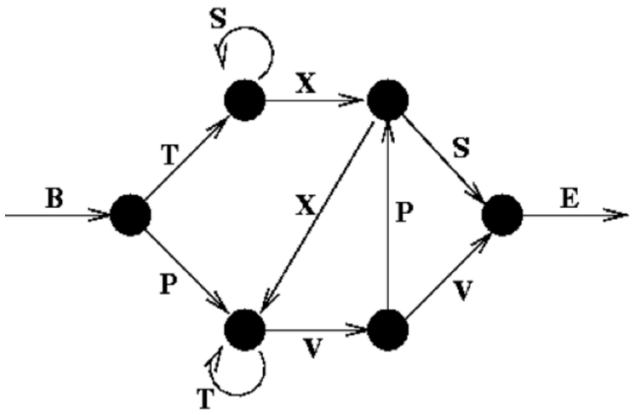
Figure 5.5: Performance Comparison



5.3.3 In/Out of Grammar Classification

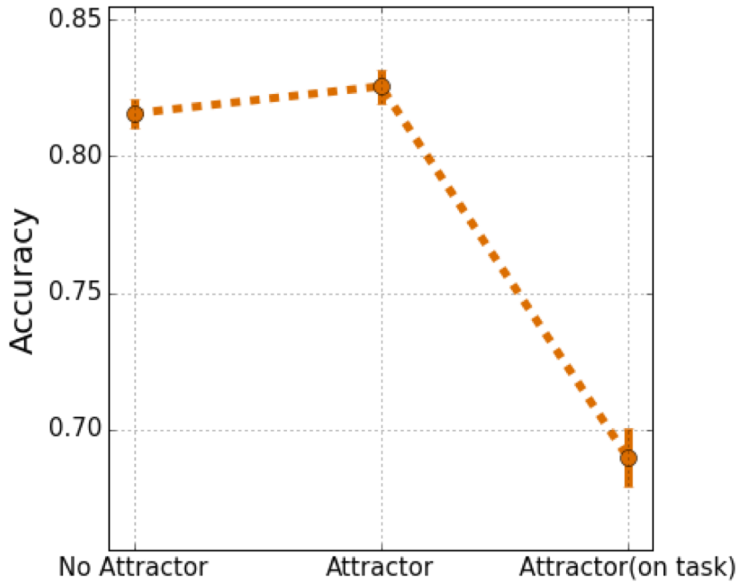
Reber grammar is an artificial grammar which generates a sequence of letters. In this Section, we attempt to determine whether the observed sequence could have been generated by the Reber grammar or not. [26] Positive (i.e. in-grammar) dataset is generated using random path choice in the Reber grammar. Negative (i.e. not in-grammar) dataset is generated by randomly perturbing one of the elements of an in-grammar sequence and verifying that such sequence could not have been produced by Reber grammar. By phrasing the task this way, RNN implicitly has to learn a **Reber grammar graph** and be able to track what path an observed sequence could or could not have come from. Such a graph is depicted in Figure 5.6.

Figure 5.6: A Reber Grammar Graph (Source: Reber, 1967)



For each task replication, we generated a training set of size 200, and a test set of size 400. Both were sampled randomly from the Reber grammar and have an equal number of positive and negative examples in them. Figure 5.7 compares the performance for our three models. We can see that SD-RNN model (**Attractor**) still performs best.

Figure 5.7: Performance Comparison



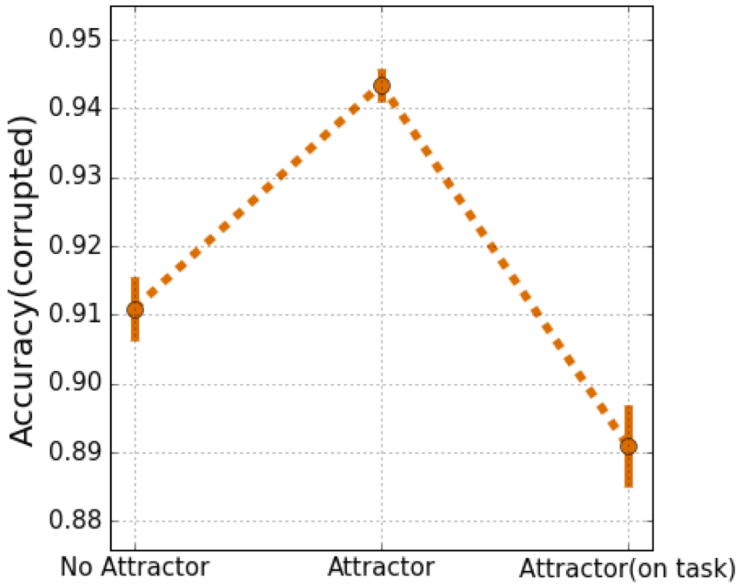
5.3.4 Parity Classification (corrupted)

In this Section, we want to determine whether the number of '1's in a sequence was even or odd based on *observed*, but *noise corrupted*, sequences. The variance of added noise corruption $\alpha_{corrupt}$ is twice variance of noise used for attractor training ($\alpha_{noise} = 0.2$). This way, none of the models was explicitly trained to process the corrupted sequences.

By phrasing the task as working with noise corrupted sequences, we are mimicking the framework of input denoising discussed in the introduction. By corrupting the input, we expect some noise to be created in the internal states of RNN due to an imperfect input to feature mapping that will propagate recurrently.

For this task, we are using sequences of length 12. For each task replication, we generate a train set of size 1024. Then, the test set is generated by adding $\alpha_{corrupt}\mathcal{N}(0, 1)$ to each train sequence element. This process is repeated twice to get 2048 test set size. The performance for our three models are compared in Figure 5.8. Here, we again see that SD-RNN model (**Attractor**) outperforms the other models that do not do internal state denoising.

Figure 5.8: Performance Comparison

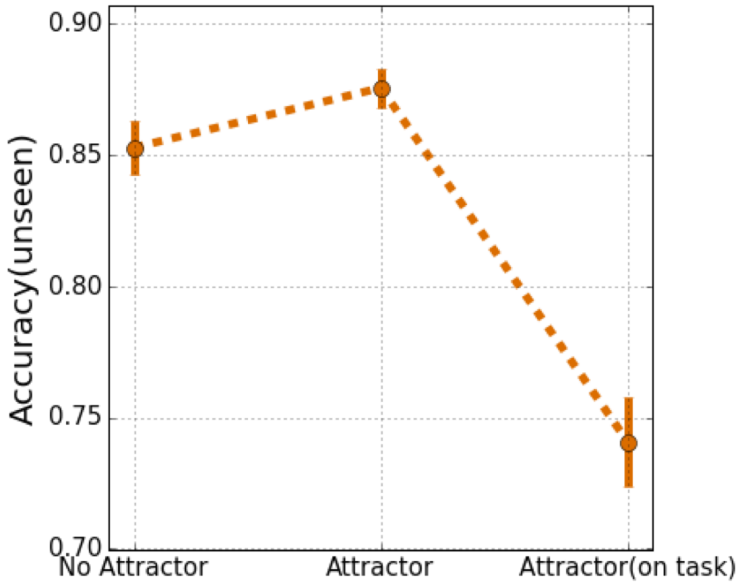


5.3.5 Parity Classification (unseen)

In this Section, we wish to determine whether the number of '1's in the sequence was even or odd based on *unobserved* sequences. This parity task is difficult for RNNs because only one sequence element change will result in a change of output label.

For this task, we are using sequences of length 12. For each task replication, we generated a training set of size 1024, while letting the test set be the remaining possible 3072 sequences. In Figure 5.9, we see that again SD-RNN model (**Attractor**) shows a reliable improvement over other models.

Figure 5.9: Performance Comparison



5.4 Real World Dataset Experiments

Now that we have verified that SD-RNN consistently outperforms standard RNN on synthetic problems, we proceed to larger and more complex problems. In particular, we consider the domain of Natural Language Processing (NLP).

Natural language always contains some ambiguity to it. For example:

- "I **run** every day": "**run** " is used as a verb

- "I went for a **run**": "**run**" is used as a noun.

Therefore RNN models trained on NLP tasks should be sensitive to noise. We demonstrate that this sensitivity is in particular noticeable during a data shortage.

5.4.1 Experiment Practices

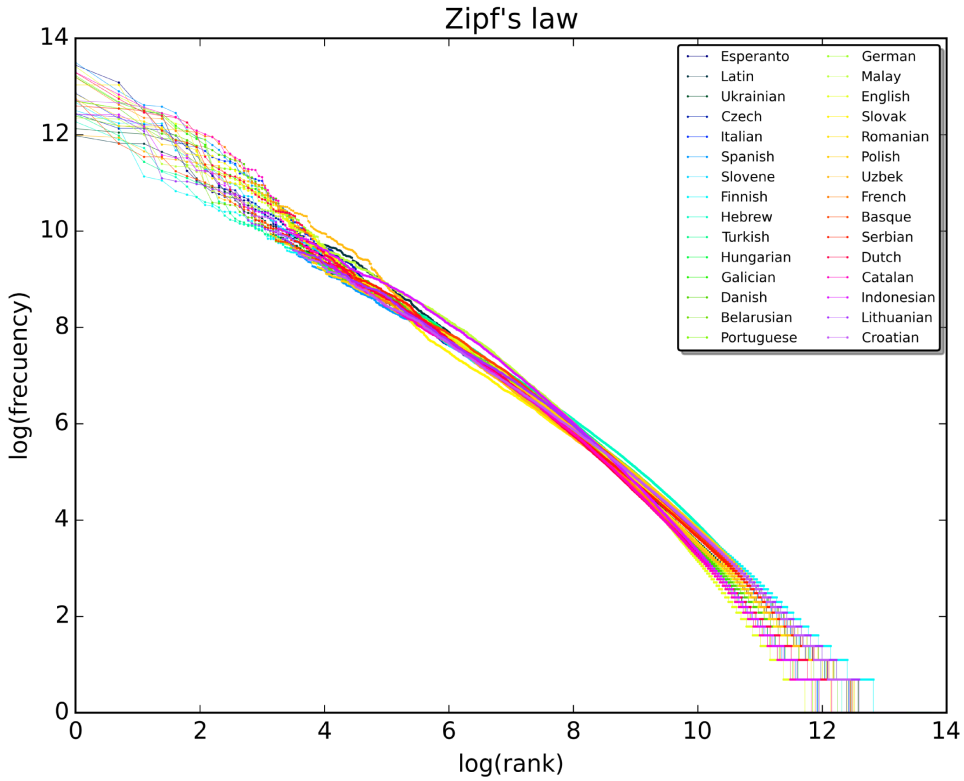
In addition to already described experimental procedures for synthetic tasks, we use the following practices when working with real world datasets:

- We use a **dropout** of 0.2 on the output of RNN.
- We use a **bidirectional GRU** RNN architecture. One of the simplest ways to improve the amount of information found in the sequences is to observe them from both directions. [27]
- We use **GloVe** (Global Vectors for Word Representation) [28] pre-trained word embeddings and hold them stationary during task learning. Using pre-trained word embeddings is a standard way to focus on task learning, since the embeddings already capture a substantial amount of information about words and how they relate to one another in the context of language.
- We use a **validation set** (or **dev** set) to ensure our comparison between SD-RNN and RNN is valid. We split the problem dataset into 3 partitions: **train** (what the model learn from), **dev** (we record the best observed performance for this dataset), **test** (we compare models' accuracies on this set). This isolation of **test** set is essential in keeping our best performance selection unbiased by never exposing the model to the **test** set until model training is complete.
- We use an **early stopping criterion** to automatically stop model training if the model is clearly overfitting. That means if the performance on the **dev** set has not seen any improvement for some number of steps (i.e. *patience* of the early stopping criterion), and

the loss value of **dev** and **train** set are not within ϵ (found experimentally) value of each other, then we conclude that the model started overfitting on **train** set and stop its training.

- We set a **vocabulary limit** on how many unique words we process from the original text. It is a known fact that words within natural language follow Zipf's law which says that the data can be approximated with a Zipfian distribution. In particular, that means words' frequency vs. their rank follow a power law relationship.

Figure 5.10: Power law relationship demonstrated for 30 languages. Text dumps used are from Wikipedia articles themselves. (Source: Wikipedia)



Therefore, it is natural to expect all word distributions in real world datasets to be extremely heavy tailed. For the sake of memory efficiency (to keep a substantially smaller word embedding lookup table in computer memory), we choose to keep only the top N

most frequent words within our datasets, while converting the rest to a specially designated symbol.

All of these practices are used for simply making RNNs applicable and useful for NLP tasks. However, ultimately, we care about comparing SD-RNN ("Attractor") with regular RNN ("No Attractor") performance under equivalent conditions, so we are not using a myriad of other methods used in the NLP community (e.g stacked RNN layers, recurrent dropout, etc.) to get state of the art performance.

5.4.2 Part of Speech (POS) tagging

This problem requires RNNs to tag every word in a given sentence with a corresponding part of speech (POS tag).

We are using Brown University Standard Corpus of Present-Day American English (or *Brown Corpus)*, which was first published in 1964 by H. Kucera and W. N. Francis at Brown University, Providence. It contains diverse samples from 500 articles of English that were published in 1961 (2000+ words in each sample). In total, it contains 1,014,312 tokens (total words in the combined text). [29]

We set test set to be 10,000 of those sequences. In total, the dataset contains 4825000 tokens. An average length of a movie review is 193, with standard deviation of 80.

We are using the version of Brown Corpus from Natural Language Toolkit (NLTK) package. We split article fragments by sentences, since POS information does not carry over to different sentences. Test set is 10,000 sentences of the total 50,000. An average length of a sentence is 20, with standard deviation of 10.

There are a total of 144 POS tags we are distinguishing between. Those include [29]:

(1) mayor form-classes (noun, verb, adjective, adverb, etc.)

(2) function words (determiners, prepositions, conjunctions, pronouns, etc.)

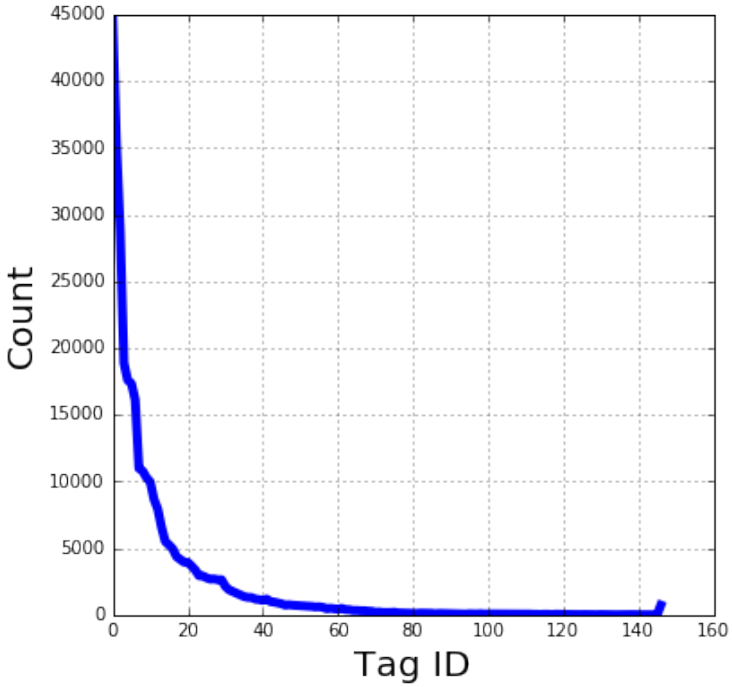
(3) important words (not, there, to, etc.)

(4) punctuation marks of syntactic significance

(5) others.

In Figure 5.11, we show the distribution of POS tags. Note the final tag count is inflected upwards. That is our "unknown" tag to indicate words not in our chosen vocabulary.

Figure 5.11: Distribution of POS tags.

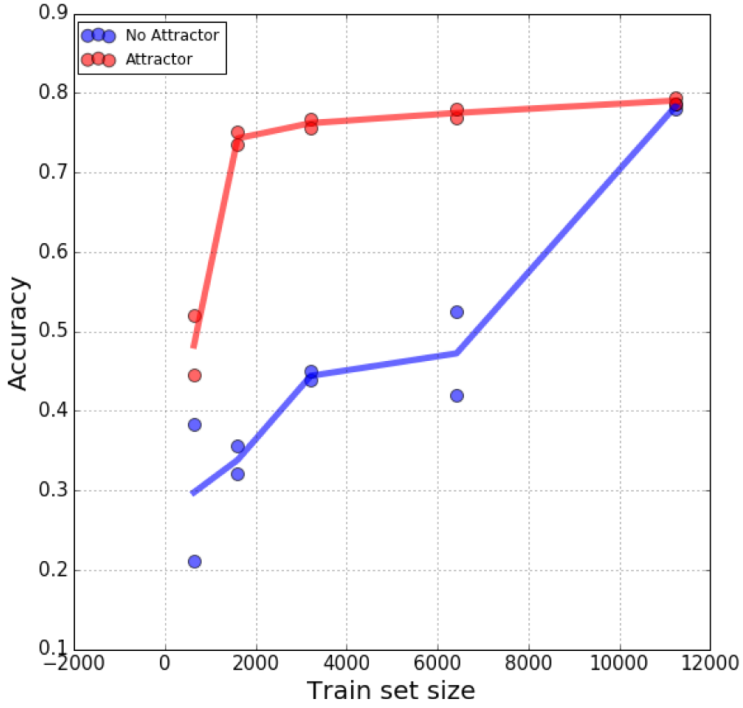


Outputting the most popular POS tag for all elements would give us only 13.07% accuracy on the test set.

This task requires RNN models to keep track of what POS given word usually is, as well as how the word changes its POS tag depending on the context. For example, "run" could be a verb or a noun.

We vary the size of training set, while keeping the test set constant to measure the generalization performances of both SD-RNN and RNN models. The results are shown in Figure 5.12.

Figure 5.12: **Line** - average of 2 replications, **point** - value of each replication.



The generalization performance of our state denoising RNN model clearly outperforms the baseline RNN with no state denoising on smaller train set samples, until their performance converges to nearly identical for the largest test sample tried.

5.4.3 Movie Review Classification

This problem requires RNNs to classify given movie review as either positive or negative in sentiment. The dataset has 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative). We set test set to be 10,000 of those sequences. In total, the dataset contains 4825000 tokens. An average length of a movie review is 193, with standard deviation of 80.

To give the reader a flavor of how a movie review looks like, we present one pre-processed positive sentiment example below:

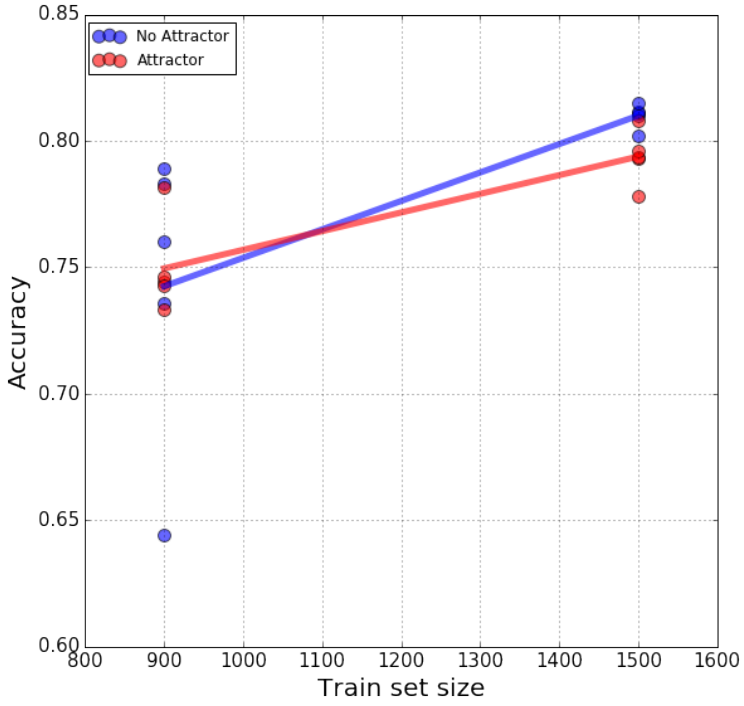
‘‘this film was just brilliant casting location scenery story direction everyone’s really suited the part they played and you could just imagine being there robert <UNK>

```
is an amazing actor and now the same being director <UNK> father came from the same
scottish island as myself so i loved the fact there was a real connection with this
film the witty remarks throughout the film were great it was just brilliant so much
that i bought the film as soon as it was released for retail and would recommend it
to everyone to watch and the fly fishing was amazing really cried at the end it was
so sad and you know what they say if you cry at a film it must have been good and this
definitely was also congratulations to the two little boy's that played the <UNK> of
norman and paul they were just brilliant children are often left out of the praising
list i think because the stars that play them all grown up are such a big profile for
the whole film but these children are amazing and should be praised for what they have
done don't you think the whole story was so lovely because it was true and was someone's
life after all that was shared with us all''
```

Note, how the tone of the speech alone sounds positive because of the word choice such as "brilliant", "amazing". The sentiment is not always this obvious and can be more contextual. For example, the writer could say "this movie was far from the brilliance it could have achieved" turning the overall positive word "brilliance" into a negative connotation.

In the same manner as we did with POS task, we vary the size of training set, while keeping the test set constant to measure the generalization performances of both SD-RNN and RNN models. Performance of both models is shown in Figure 5.13. The points represent the values of each replication and the lines represent the average values of two replications.

Figure 5.13: Performance comparison (sentiment classification task) between SD-RNN and regular RNN.



The generalization performance of both models seems to be around the same level. It might be the case that having only 2 classes does not generate enough noise for the network to be confused by. This result prompts further exploration via alternative datasets.

Chapter 6

Conclusion and Future Work

In this work, we have introduced a practical way to denoise internal states of RNNs. We motivated the idea of using an attractor network as a denoising mechanism and demonstrated its effectiveness and robustness in reconstructing noise-corrupted internal states. We then integrated the attractor network within the framework of Recurrent Neural Networks(RNNs) used for sequence processing, making a modified architecture—State-Denoised RNN (SD-RNN). We verified that SD-RNN reliably outperforms the baseline RNN on 4 synthetic tasks, as well as 1 out of 2 tested so far NLP tasks. This gives us a firm motivation to test SD-RNN on more real world datasets and investigate the effects of internal state denoising in more detail.

Bibliography

[1] J.V. Chamary *How Face ID Works On iPhone X*, Forbes.

[2] S. Boll. *Suppressionof Acoustic Noise in Speech Using Spectral Subtraction.* IEEE Transactions on Acoustics, Speech, and Signal Processing, VOL. ASSP-27, NO.2.

[3] P. Simard, B. Victorri, Y. LeCun, J. Denker. *Tangent Prop - A formalism for specifying selected invariances in an adaptive network.* Advances in Neural Information Processing Systems 4 (NIPS 1991).

[4] S. Zheng, Y. Song, T. Leung, I. Goodfellow. *Improving the Robustness of Deep Neural Networks via Stability Training.* Conference: 2016 IEEE Conference on Computer Vision and Pattern Recognition.

[5] A. Krizhevsky, I. Sutskever, G. E. Hinton. *Imagenet classification with deep convolutional neural networks.* Advances in neural information processing systems, 2012.

[6] Mozer, Michael C. *"Neural net architectures for temporal sequence processing."* Santa Fe Institue Studies in the Scienes of Complexity-Processdings volume-. Vol. 15., 1993.

[7] C. E. Shannon *A Mathematical Theory of Communication.* Bell System Technical Journal 1948.

[8] R. Schwartz-Ziv, N. Tishby *Opening the black box of Deep Neural Networks via Information.* ICLR 2018.

[9] N. Tishby, F. C. Pereira, W. Bialek. *The information bottleneck method.* arXiv preprint physics/0004057 (2000).

[10] D. E. Rumelhart, G. E. Hinton, R. J. Williams *Learning representations by back-propagating errors.* Nature volume 323, pages 533536 (09 October 1986).

[11] D. Kingma, J. Ba. *Adam: A method for stochastic optimization.* In ICLR, 2015.

[12] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. *Dropout: a simple way to prevent neural networks from overfitting.* Journal of Machine Learning Research, 15(1):19291958, 2014.

[13] P. Baldi, P. Sadowski. *Understanding Dropout.* A. E. P. VIlla et al. (Eds.): ICANN 2016 (Part II, LNCS 9887, pp. 1-8, 2016).

[14] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, J. Dean. *Distributed representations of words and phrases and their compositionality.* Advances in neural information processing systems, 3111-3119

[15] T. Mikolov, K. Chen, G. Corrado, J. Dean. *Efficient estimation of word representations in vector space.* arXiv preprint arXiv:1301.3781

[16] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, P. Abbeel *InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets.* Neural Information Processing Systems 2017.

[17] A. Karpathy, *The Unreasonable Effectiveness of Recurrent Neural Networks* `karpathy.github.io`.

[18] S. Hochreiter and J. Schmidhuber. *Long short-term memory.* Neural computation, 9(8):17351780, 1997.

[19] Y. Bengio, P. Simard, and P. Frasconi. *Learning long-term dependencies with gradient descent is difficult.* IEEE transactions on neural networks, 5(2):157166, 1994.

[20] R. Pascanu, T. Mikolov, and Y. Bengio. *On the difficulty of training recurrent neural networks.* International Conference on Machine Learning. 2013.

[21] A. Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks.* Personal blog(karpathy.github.io).

[22] M. C. Mozer, D. Kazakov, R. V. Lindsey *Discrete-Event Continuous-Time Recurrent Nets.* arXiv:1710.04110v1.

[23] P. Koiran. *Dynamics of Discrete Time, Continuous State Hopfield Networks.* Neural Computation, Volume: 6, Issue: 3, May 1994.

[24] D. A. Mathis, M. C. Mozer. (1995), *On the computational utility of consciousness.* In G. Tesauro, D. S. Touretzky, T. K. Leen (Eds.), Advances in Neural Information Processing Systems 7 (pp. 10-18). Cambridge, MA: MIT Press.

[25] A. Graves. *Adaptive Computation Time for Recurrent Neural Networks.* CoRR, Vol abs/1603.08983. 2016.

[26] A. S. Reber *Implicit learning of artificial grammars.* Verbal Learn. Verbal Behav. 6, 855863.

[27] M. Schuster and K.K. Paliwal, *Bidirectional Recurrent Neural Networks.* IEEE Transations on Signal Processing, VOL. 45, NO. 11.

[28] J. Pennington, R. Socher, C.D. Manning, *GloVe: Global Vectors for Word Representation.*

[29] W. N. Francis, H. Kucera. *Brown Corpus Manual.* Brown University July 1979.