JACOBS
UNIVERSITY

An Echo State Network approach to learning meaningful word vector representations

by

Valentine Chiwome

Bachelor Thesis in Computer Science

Supervisor : Prof. Dr. Herbert Jaeger

Date of Submission: March 26, 2014

Jacobs University — School of Engineering and Science

With my signature, I certify that this thesis has been written by me using only the indicated resources and materials. Where I have presented data and results, the data and results are complete, genuine, and have been obtained by me unless otherwise acknowledged; where my results derive from computer programs, these computer programs have been written by me unless otherwise acknowledged. I further confirm that this thesis has not been submitted, either in part or as a whole, for any other academic degree at this or another institution.

Signature Place, Date

Executive Summary

Word sequence modelling is a very complicated computational task. The primary source of the complexity is that language is driven by very loose and flexible rules, but computers require a compact unambiguous representation of information. This, together with the vast amount of text that can now be found on the web, has seen scientists try to model language(word sequences) using statistical and Machine Learning techniques that can then be trained from data. Statistical word sequence models range from n-grams, skip-grams, neural networks and many more.But most of these models suffer from sparse training data. This basically means that many training datasets only represent a very small part of the state space. Researchers have tried to get around this problem using various methods, some of which try to model language in a continuous space. For instance words can be represented as continuous vectors. Similar words then stay close together in this space and interpolation is generally easier in continuous spaces. In this work, I am going to present a way of using a reservoir computing method to get word vector representations.

Contents

1 Introduction

Language modelling is a big, active research area. Language models are created for Machine Translation, Data Mining, speech recognition among other applications. In this thesis, language modelling refers to word sequence modelling.

Language is inherently probabilistic and this has motivated Researchers to turn to Machine Learning models for language processing.

But to successfully perform any form of language modelling using Machine Learning algorithms, it is necessary to first solve the data sparseness problem that is associated with language. Basically, this means language is a set of rare events and many practical datasets are not representative enough of the real model that drives language. The vocabularies of many languages are huge so there are extremely huge possible combinations of words that can form a sentence.

Because of the problem discussed above, any Machine Learning model that hopes to achieve acceptable performance has to deal with this problem.

Researchers have come up with a number of effective Machine Learning models for natural language. Most of them are already tuned to deal with the problem of data sparseness. I will briefly describe some of them in the coming sections.

1.1 N-gram model

N-grams learn a probability distribution in a discrete space of size $N^{|V|}$ where N is the order of the model and V is the vocabulary size. For any reasonable value of N and V, the state space in question is enormous. For many finite training datasets, we are almost guaranteed to meet examples from only a tiny fraction of the state space. This makes it hard for n-grams to learn a function that can generalize well to unseen data.

This has seen many applications settle for a tri-gram model. Tri-grams only consider the last two words for context (ie they model $P(w_t|w_{t-1}, w_{t-2})$). However, it is clear that understanding most languages requires long range contexts.

1.2 Smoothed n-grams

N-grams are a probability distribution over length n word sequences, ie $P(w_i|w_{i-1}, ..., w_{i-(n+1)})$. One way of estimating this distribution is to use a maximum likelihood estimator, ie $P(w_i|w_{i-1}, w_{i-2}, .., w_{i-n}, w_{i-(n+1)}) = \frac{c(w_i, w_{i-1}, ..., w_{i-n}, w_{i-(n+1)})}{c(w_{i-1}, ..., w_{i-n}, w_{i-(n+1)})}$ where $c(seq_i)$ is the number of times sequence seq_i appeared in the training data.

But the maximum likelihood estimator has at least one major flaw: it assigns zero probability to all n-grams that are not part of the training data. But even if the training data did not contain the phrase, "the cow", it is still unreasonable to assign this bi-gram a probability of zero.

Hence the introduction of smoothing techniques. There are various smoothing techniques, the most basic one being to assign a small constant probability to all unseen sequences(n-grams). A different approach was presented by Katz [1, S. M. Katz, 1987].

He regards n-grams with low counts as unreliable evidence and then use Good-Turing estimates to distribute their probability mass among unseen data. Basically, he replaces maximum likelihood estimates with Good-Turing estimates.

Another smoothing technique is due to Jelinek [2, F Jelinek, 1983]. He proposed the interpolation of the maximum likelihood estimator with lower order n-grams, e.g by a recursive equation such as $P(w_i|w_{i-1},...w_{i-(n+1)}) = P_{ML}(w_i|w_{i-1},..,w_{i-(n+1))})+P(w_i|w_{i-1},..,w_{i-n})$. A more detailed analysis is presented in [3, J. Goodman, S. F. Chen, 1983]

1.3 Skip-grams

Besides the above mentioned smoothing techniques, one can also use skip-grams. The idea behind skip-grams is that, instead of only considering consecutive words when building n-grams, we can also consider words that appear with 1 or more words between them. For instance, a phrase such as "reasoning about facts in" will give the following bigrams : "reasoning about", "reasoning facts", "reasoning in", etc. With this method, we can extract more information from less data and this will hopefully help us get a model that generalizes well to unseen data. However, we have to take note that words that appear closer together are more dependent than those that appear with "gaps" between them. Formally, a k-skip-n-gram for the sentence $w_1, w_2...., w_m$ is the set $\{w_{i_1}, w_{i_2}, ..., w_{i_n} | \sum_{j=1}^{n} i_j - i_{j-1} < k\}$ [4, D. Guthrie].

1.4 Continuous Skip-grams

In general, generalization is hard for discrete models that have many variables. This is because a change in the value of one variable may lead to a swing in the function value. This is called the "off-domain problem". But in continuous spaces, interpolation is much easier and the "off-domain problem" is evaded.

This has motivated researchers to try to move towards continuous models for NLP.

A continuous skip-gram was introduced. In this case, for any word w_i, a continuous model is trained to predict the words that surround w_i (words that appear at a distance that is less than k to the left or to the right of the current word; the larger the value of k, the better the model but the more expensive training is) [5, T. Mikolov, 2013].

1.5 Neural Networks

Researchers have also tried to use Neural Networks for modelling text. [6, I. Sutskever, 2011] trained a Recurrent Neural Network to perform character prediction. Their network managed to write sentences that make "local sense". For instance, it generated the sentence "The wild pasteured with consistent street forests were incorporated by the 15th century BE".

Another approach is in [7, Y. Bengio, 2003]. They learnt a model for predicting the next word using some finite word history. This model consisted of two parts. The first part learnt a distributed vector representation of words in a space whose dimension is much

smaller than the vocabulary size. The next part used the learnt vector representations of words to predict the next word. Both parts of the system were learnt concurrently.

The important idea that was used in this work was to find a continuous vector representation of the words. Words that play similar roles had similar presentations. This is critical for the model's capacity to generalize to unseen data. For instance, if the training data contains "Today is Monday" then "Today is Tuesday", even if it was not part of the training data, will have a high probability if the vector representation of Tuesday and Monday are similar.

Similar work is also done in [5, T. Mikolov, 2013]. Their models also learnt vector representations of words. Further semantic word relationships were captured using linear vector operations. For instance, they reported that vec(Germany) + vec(capital) is close to vec(Berlin).

2 Statement and Motivation of Research

My work is in the same spirit as some of the methods that were mentioned above. I am going to train an ESN for predicting the next word given some finite context.

This will help me model syntactic and semantic word relationships using this compact model, a Recurrent Neural Network. The Neural Network approach will help us get a continuous model that needs a bounded number of parameters to represent an extremely huge domain.

As we will see in the next section, this model also allows us to get a lower dimensional, distributed, semantically rich vector representation of words. Such a word representation makes it easier for the models that will use it to learn a well-generalizing function on word sequences. It will also help us curb the "off-domain problem" that haunts discrete state spaces in large dimensions.

Another important aspect of this model is that it has a huge reservoir. The reservoir is composed of neurons that are in the recurrent, non-linear hidden layer of the Neural Network. This reservoir performs two critical functions.

First, it acts as a nonlinear expansion of the input [8, M. Lukosevicius, 2012]. Inputs that are not linearly separable in the original space can be separable in the expanded space.

Secondly, the reservoir acts as a short term memory for the network. Memory is very important in word sequence modelling and this makes the ESN model effective.

3 Theoretical Setting

3.1 Recurrent Neural Networks

A Recurrent Neural Network (RNN) consists of 3 main parts, the input layer, the hidden layer(s) and the output layer. The output and the input layers are usually linear but the hidden layer is non-linear.

At each time step, the state of the hidden layer is computed by $x_i = tanh(W * x_{i-1} + W_{in} * u_i)$ where u_i is the current input, x_i is the state of the hidden layer at time step *i* and W and W_{in} are the hidden to hidden weight matrix and input to hidden layer weight matrix respectively. The output is computed by $y_i = W_{out} * x_i$.

Given this architecture, an RNN is a very powerful computational model that implements a dynamical system. One commonly used algorithm for training an RNN is the Backpropagation Through Time algorithm. But it suffers from the vanishing gradient problem [9, Sepp Hochreiter, 1998], making the training of RNNs very difficult.

Researchers have tried to avenge this problem using various techniques and one of the most successful is using an Echo State Network (ESN).

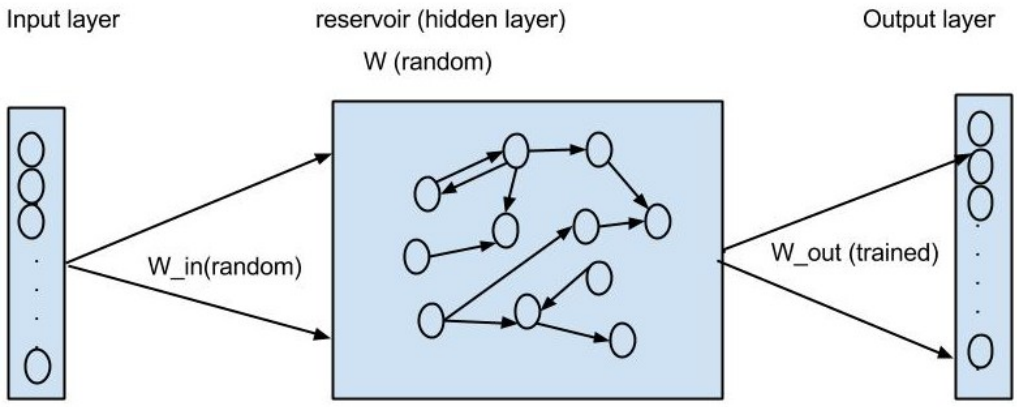

Figure 1: Basic architecture of an Echo State Network.

An ESN is an instantiation of an RNN where W_{in} and *W* are initialized randomly but carefully, only W_{out}, the output weight matrix is trained [10, H. Jaeger, 2007]. W_{out} is trained using any linear regression method of choice. In this work, we use ridge regression that I am going to describe in more detail in the next section. Note that, we do not have vanishing gradients problem anymore.

The hidden layer, also known as the reservoir, of an ESN is central to the success of an ESN. First, it acts as a nonlinear expansion of the input [8, M. Lukosevicius, 2012]. Inputs that are not linearly separable in the original space can be separable in the expanded space. Secondly, the reservoir acts as a short term memory for the network.

Supervised training of an ESN works as follows, you need an input sequence $u_0, u_1, ..., u_n$ and the corresponding correct output sequence $y_0, y_1, ..., y_n$. At each time instant i, we compute $x_i = W_{in}u_i + Wx_i$. After k steps, the network will have gained "momentum" and we can start storing $s_{i-k} = (1, u_i, x_i)$ in a matrix X.

After that we can then train W_{out} using ridge regression with $s_0, s_1, ..., s_{n-k}$ as input and $y_k, y_{k+1}, ..., y_n$ as output.

3.2 Ridge Regression

In this work, I am going to use Ridge Regression to train the output weight matrix of the ESN.

Given two vectors X and Y, we may want to find a function $f(X)$ that estimates Y. This function has to minimize an error function $E(Y, f(X))$.

One way of solving this problem is to use linear regression. With linear regression, $f(X) = \beta_0 + \sum_{i=1}^{n} \beta_i x_i$. We can then try to find a function that minimizes the error function on the given data. If we define our loss function to be the mean square error, i.e $E(f) = \sum_{i=1}^{m}(y_i - \beta_0 + \sum_{j=1}^{n} \beta_j x_j)^2$, we may estimate β as follows: $\beta = (\beta_0, \beta_1, .., \beta_n)^T = (X^T X)^{-1} X^T y$.

But there is a problem in Machine Learning that is called overfitting. It means that we can get a model that has very low training error but will have a high test error. This is because the model might fit noise or other irrelevant patterns in the data. A model that does not fit noise will obviously be better at generalizing that the one which does. To overfit data, a model usually has to learn a complicated function.

To that end, Researchers try to avoid overfitting by keeping a model simple. Although "simple" is difficult to define, it is reasonable to say that large parameters are more complicated than smaller ones.

This has motivated researchers put a constraint on β; they try to fit the data, while keeping its magnitude small.

We can achieve this by changing $E(f)$ to $E(f) = \sum_{i=1}^{m}(y_i - \beta_0 + \sum_{j=1}^{n} \beta_j x_j)^2 + \lambda \sum_{j=1}^{n} {\beta_j}^2$. This gives us $\beta^{ridge} = (X^T X + \lambda I)^{-1} X^T y$. This is Ridge regression, and the parameter λ can be used to control overfitting.

3.3 Principle Component Analysis

Another algorithm that was put to work in this thesis is Principle Component Analysis (PCA). PCA can be used (and is used in this work) as a dimensionality reduction algorithm.

To be concrete, say we have an $n \times m$ dimensional matrix X. This matrix consists of m vectors each of which is n dimensional and represents a data point.

If we want to reduce the dimensionality of each vector x_i in X, we could do 2 things: remove noise and remove redundancy from the data.

$K = XX^T$ is called the covariance matrix of X. $K_{ij} = x_i {x_j}^T$ is the covariance between x_i and x_j where x_i and x_j are the i-th and the j-th rows respectively. Note that in our case, the i-th row represents the value of the i-th attribute across all data points while the i-th column represents the i-th data point.

This implies that K_{ij} is the correlation between the i-th and the j-th attributes of the data points. If there was no redundancy in the data at all, all non-diagonal entries of K would be 0; K would be a diagonal matrix. So to remove redundancy, we would like to replace X with $Y = PX$ such that $K_y = YY^T$ is diagonal.

To achieve this, PCA settles for the easy solution. PCA assumes that all columns p_i of P are orthonormal; it assumes that P is an orthonormal matrix. This assumption makes our problem solvable with linear algebra tools. We can take p_i to be the i-th eigenvector of K. This means we can now replace X with $X^{'} = PX$ ie $x^{'}{}_i = (p_1 x_i, p_1 x_2, ..., p_m x_i)^T$.

The second assumption that PCA makes is that the p_i that are most important are those that are associated with the highest variance among the data points in X. The $p_i s$ that are associated with the highest variance are those that have the largest eigenvalues.

With this in mind, the new representation only uses the first k eigenvectors of the matrix k as the new basis. $X^{'}{}_i = (p_1 x_i, p_1 x_2, ..., p_k x_i)^T$.

We now have a k dimensional representation of an n-dimensional vector that preserves most of the variance between the data points while removing redundancy. This is basically how PCA works.

3.4 Context free grammars

We are also going to use a context free grammar for sentence generation.

Formally, a context free grammar is a tuple $G = (S, \sum, V, P)$.

P is a set of productions of the form $v \Rightarrow w$ where $v \in V$ and w is a sequence of symbols from $V \cup \sum$. For compactness, if we have two production, one of the form $v \Rightarrow w$ and another one of the form $v \Rightarrow w^{'}$, we can simply write $v \Rightarrow w|w^{'}$.

A sequence of zero or more symbols is called a **word** The length of a word is the number of symbols it contains. There is a special word of length zero that is called ϵ.

$\sum$ is the set of n symbols, $\sigma_1, .., \sigma_n$. called terminal symbols. None of the terminal symbols σ_i appears on the left hand side of any production, hence their name. Elements of V, the non-terminal symbols are the only ones that appear on the left hand side of productions.

If we have a production $p = v \Rightarrow w$, and a word $w^{'}$ and v is a substring of $w^{'}$, we can replace the substring v of $w^{'}$ by w. This is called applying a production and is denoted as $w \Rightarrow_p w^{'}$. In general, we **derive** $w^{'}$ from w if we apply a sequence of productions to w and we end up with $w^{'}$.

S is called the start symbol. A set of all words that can be derived from the word S and contain only terminal symbols are the elements of the context free language that is defined by the context free grammar. We call the language $L(G)$.

4 Experiments

4.1 The Neural Network

For the following experiments, I used an ESN approach that is described in the first section of this paper. The input sequence was a sequence of words in text and the input for output $word_i$ was $word_{i-1}$. I used ridge regression for training W_{out} using a constant value, 10^{-7} as the parameter λ. This was done after performing various experiments with different values of λ and not seeing major dependency for lambda in range

$10^{-30} \leq \lambda \leq 10^{-6}$. I encoded the words as vectors using one-of-n encoding. Here n was the number of unique words that are in the input text.

4.2 Experiment 1

4.2.1 Motivation

Any model that hopes to compete with state-of-the-art natural language models has to deal with the data sparseness problem. One way of mitigating this problem is to have a huge training dataset with respect to the vocabulary size. This motivated me to create a synthetic text dataset. I restricted the vocabulary size to make the required training data lie within the capacity of my computational resources.

But the synthetic dataset had to be generated by a model that simulates some characteristics of real natural language. The most important requirement was to maintain the probabilistic nature of language.

I also wanted to have a clear definition of similarity among the words. Below, I explain how I designed a language that tries to meet the above mentioned requirements.

4.2.2 Execution

I first created a synthetic text dataset using 65 words. The words were categorized as nouns, verbs, adverbs, adjectives, determiners and prepositions.

I generated sentences from the grammar that is described below. To expand a non-terminal symbol, any production that has that symbol on the left hand side was chosen with equal probability.

S $\Rightarrow$ *NP VP* | *S COORD S* | *S SUBD S*
NP $\Rightarrow$ *D ADJ N*
VP $\Rightarrow$ *ADV V NP PP*
PP $\Rightarrow$ *PRE NP*
D $\Rightarrow$ *the* | *a* | *an*
N $\Rightarrow$ *leopard* | *antelope* | *elephant* | *kudu* | *tiger* | *monkey* | *baboon* | *lion* | *hyena* | *cheetah*
V $\Rightarrow$ *ate* | *ran* | *fought* | *lived* | *killed* | *hunted* | *loved* | *drank* | *protected* | *survived*
PREP $\Rightarrow$ *in* | *on* | *under* | *with* | *above* | *among* | *after* | *before* | *around* | *across* | *over*
ADJ $\Rightarrow \epsilon|$ *blue* | *red* | *green* | *yellow* | *big* | *giant* | *fat* | *swift* | *small*
ADV $\Rightarrow \epsilon|$ *swiftly* | *amazingly* | *angrily* | *jokingly* | *furiously* | *hungrily*
COORD $\Rightarrow$ *for* | *and* | *nor* | *but* | *or* | *yet* | *so*
SUBD $\Rightarrow$ *because* | *since* | *after* | *although* | *when*
ADVCON $\Rightarrow$ *that* | *which* | *where* | *who*

I then generated 10000 sentences from such a grammar. In our context, generate sentence means expand the symbol S until the derivation tree consists entirely of terminals.

From the 10000 sentences, I trained a network with 65 input units, 100 hidden units and 65 output units. W_{out} was a 65×165 dimensional matrix. Using PCA, I reduced the dimensionality of W_{out} to 65×20. I then interpreted the 20 dimensional i-th entry of W_{out} as the vector representation of $word_i$. Reducing W_{out} to a 65×10 dimensional matrix also worked equally well as with a 65×20 matrix.

I computed the closest neighbors for different words and below are some of the results. I say v_i and v_k are closest neighbors if $|v_i - v_j| \geq |v_i - v_k|$, for all j. The table below lists the closest neighbors for the head of the column.

cheetah	angrily	small	ate	for
baboon	furiously	red	lived	so
kudu	swiftly	green	loved	nor
leopard	jokingly	yellow	protected	yet
hyena	protected	swift	ran	because
elephant	lived	above	fought	or

Table 1: Neighbor information of word vectors learnt by the ESN after one cycle of training.

4.2.3 Explanation

In our case, similar words are the terminal symbols that appear on the right hand side of the same production.

Interestingly, the network learnt some meaningful word vector representations. The closest neighbors to nouns are nouns while the closest neighbors to verbs are verbs etc.

Since I was feeding the network using text from left to right, the network learnt word vector representations using past context. Words that share the same history context are similar.

Basically, the network managed to learn similar vectors for words that are statistically indistinguishable with respect to the words that come before them.

4.3 Experiment 2

4.3.1 Motivation

Once I discovered that the ESN could actually learn vector representations of words depending on words that appear just before them, the next challenge was to investigate whether it could also learn some form of ”transitivity” in the relationships among words.

Our ESN knows that two words are similar because the words that appear before them are the same.

But what if the words never share any predecessors, but rather they have similar predecessors? Assume we know that *colorful* and *huge* are similar (it helps to think of them as adjectives). Say our network learns that these words are actually similar. But *elephant* always comes after *huge* while *leopard* always comes after *colorful*. To know that *elephant*

and *antelope* are similar, the model first has to learn that *colorful* and *huge* are similar. I then modified my language to capture this idea.

4.3.2 Execution

To capture the concept that is described above, I generated data from a new grammar. I replaced the production:

$N \Rightarrow$ *leopard* | *antelope* | *elephant* | *kudu* | *tiger* | *monkey* | *baboon* | *lion* | *hyena* | *cheetah*
with a new production:
$N \Rightarrow$ *colorful leopard* | *big antelope* | *huge elephant* | *kudu* | *tiger* | *clever monkey* | *dump baboon* | *brave lion* | *greedy hyena* | *fast cheetah*

I also replaced:
$ADV \Rightarrow \epsilon|$ *swiftly* | *amazingly* | *angrily* | *jokingly* | *furiously* | *hungrily*
with: $ADV \Rightarrow \epsilon$

Using the model described earlier, I trained the network again and here are the results.

colorful	lion	cheetah	hyena	that	in	ate	for
big	huge	,	nor	who	among	hunted	or
fast	dump	leopard	that	where	before	loved	and
greedy	fierce	the	but	which	over	ran	yet
huge	colorful	an	and	when	above	protected	but
fierce	clever	under	across	and	across	killed	so

Table 2: ESN fails to learn transitive word relations.

4.3.3 Explanation

The table above tells us at least two things, the first is that word relationships for nouns were completely lost. See the neighbor information for *lion*, *cheetah* and *hyena*. The second thing to note is that, all other relationships are captured, including the modifiers for the nouns, i.e the column that starts with *colorful*.

This prompted me to conclude that to make a prediction, the network might be putting too much emphasis on the immediately preceding word.

4.4 Experiment 3

4.4.1 Motivation

The experiment described above prompted me to use what the information that the network had already learnt. Since the network had already learnt that the adjectives are similar, I wanted to check whether we can transfer this knowledge and also learn that the nouns are also similar, even if they are preceded by different words.

I also wanted to reduce the network's dependency on the immediately preceding word when it makes a prediction.

4.4.2 Execution

To transfer knowledge, I decided to train the network for the second time, this time using the learnt 20 dimensional vector representations instead of one-of-n encoding. In our example, if the network has learnt that *huge* and *colorful* are similar, it should be able deduce that *elephant* and *leopard* are also similar, not because they are preceded by the same words but because they are preceded by similar words.

To reduce the dependency of the network on the immediately preceding finger, I decided to use another variant of the algorithm for training an ESN. In section 3.1, we said that we store the vectors $(x_i, u_i, 1)$ so that we can use the for training the model. But this time I used the variant that only stores $(x_i, 1)$ thus reducing the dependency of the network on the immediately preceding word, u_i.

I will only report on the results for nouns since nothing changed for all other word classes.

lion	cheetah	hyena
around	hyena	monkey
across	fierce	fierce
under	huge	but
in	an	ran
among	clever	dump

Table 3: ESN fails to learn transitive relation after second training epoch.

4.4.3 Explanation

It is clear that word relationships among nouns are still in the wild.

But then I noted that on the intermediate steps, we need not reduce the dimensionality of the word vector representations. They only need to be reduced when we compute the final vectors. So I first trained the network and learnt 65 dimensional vector representations of words. I then trained the network using the same data but instead of one-of-n encoding, we used the learnt 65-dimensional vector representation of words. Keeping all the 65 dimensions would help us not lose information through dimensionality reduction. But we later realized that keeping all the dimensions could potentially introduce noise into the vectors.

I finally used PCA as before to get 20-dimensional vector representations of words. This did not help, the results were not stable. At this moment, I did not have a clear explanation for this, so I kept making more experiments.

4.5 Experiment 4

4.5.1 Motivation

With the background that the strength of an ESN lies in its reservoir and W_{out}, and that the vector representations only come from W_{out}, I thought it would be necessary to force most of the work on W_{out}. This would ensure that W_{out} stores as much information about the dynamics of the language as possible and will potentially produce richer vectors. I thought this could be achieved by making the reservoir weak.

4.5.2 Execution

I then decided to reduce the size of the reservoir to be just 30, almost half the vocabulary size. I learnt 5-dimensional vector representations of words.

Below are the results for this case.

lion	cheetah	hyena
cheetah	monkey	baboon
monkey	lion	elephant
tiger	tiger	leopard
zebra	zebra	antelope
antelope	antelope	tiger

Table 4: ESN learns the transitive relations after reducing the reservoir size.

4.5.3 Explanation

Finally the network clearly managed to recover the similarities among the nouns. Feeding back the learnt vectors helped the model realize that even though *lion* and *cheetah* always have different words immediately before them, they are similar because they have similar words before. I claim this a form of ”transitivity” of the similarities among words that we described earlier.

There are various ways of explaining the results above. One could be the one that we already stated; making the reservoir weaker forces the network to use W_{out} to do most of the work. W_{out} becomes more discriminative and hence represents more information about the words. This makes the word vectors more informative.

A different way of describing the above improvement is that a smaller reservoir has less memory. So the history context for nouns only included few words. This means the history did not include other nouns, whose vectors are not accurately learnt in the first learning phase.

5 Conclusions and Discussion

This work has shown that an ESN is a powerful model for natural language processing. The model managed to learn vector representations of words that encode statistical relationships among the same.

It has also shown that by using Principle Component Analysis, we can reduce the dimensionality of the learnt vector representations to a dimension that is much lower than the vectors inside W_{out}.

It also showed that feeding back the vectors several times helps to learn similarities that are not ”hard-coded” in the data, but are described in terms of similarity of other words.

However, the synthetic dataset that I created is still relatively far from real natural language. The learnt vector representations of words were similar for words that are statistically indistinguishable. But some words that we call similar in natural language are usually still statistically distinguishable.

6 Future Research

This work has shown that the method we described can generate for us, meaningful word vector representations for statistically similar words. This is a good thing but it is no surprise.

It would be more interesting to know if the same procedure can also learn word vector representations for statistically distinguishable words. In real language, similar words are sometimes statistically distinguishable. The model has to prove that it can extract similarity even among statistical distinguishable but similar words. I therefore encourage research in this direction.

One more interesting thing to do would be to create a language that encodes word similarities on different levels. More similar words should be closer to each other than less similar words. Similarity should have different levels and the level of similarity between two words has to be encoded in the distance between them.

7 Acknowledgements

I would like to thank my supervisor Professor. Dr Herbert Jaeger for his continuous support and constant patience during my first real-life contact with Neural Networks and Machine Learning in general. I would also like particularly thank him for agreeing to every urgent meeting that I requested which he used to give me many helpful hints and point me in the right direction.

References

[1] Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, pages 400–401, 1987.

[2] Lalit R. Bahl, Frederick Jelinek, and Robert L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 5(2):179–190, 1983.

[3] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

[4] David Guthrie, Ben Allison, Wei Liu, Louise Guthrie, and Yorick Wilks. A closer look at skip-gram modelling. In *In Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC 2006*, pages 1222–1225, 2006.

[5] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. 2013.

[6] I. Sutskever, J. Martens, and G. Hinton. Generating text with recurrent neural networks. In *Proc. ICML 2011 (online)*, 2011.

[7] Y. Bengio, R. Ducharme, P. Vincent, and Ch. Jauvin. A neural probabilistic language model. *J. of Machine Learning Research*, 3:1137–1155, 2003.

[8] Mantas Lukosevicius. A practical guide to applying echo state networks. In Gregoire Montavon, GenevieveB. Orr, and Klaus-Robert Muller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 659–686. Springer Berlin Heidelberg, 2012.

[9] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06(02):107–116, 1998.

[10] Herbert Jaeger. Echo state network. *Scholarpedia*, 2(9):2330, 2007.