6.863 Final Project: Comparison of HMM and LSTMs to Kimmo

Prafull Sharma, Yada Pruksachatkun, Rui Li
Massachusetts Institute of Technology
{prafull, yadapruk, ruili}@mit.edu

Abstract

In this paper, we compare the performance and difference in learning abilities of the HMM and LSTM against the FST approach (Kimmo) of realizing the surface form from the lexical form. We focus specifically on pluralization rules for the scope of our project, and analyze what types of patterns the HMM and LSTM learn for this pluralization task, and discuss whether it is similar to the FST's branches of computation.

1. Introduction

The process of linguistic acquisition and how to model it has been at the forefront of computational linguistics for the past decade, first with context free grammars, finite state transducers and, with the onset of deep learning, neural networks. Finite state machines and context free grammars is an attempt to write rules which govern language as we understand it. Neural networks on the other hand try to learn the language using data. Even though neural networks have shown interesting results, it is unclear if they capture the concepts of language.

In this project, we will investigate how hidden markov models (HMMs), long short-term memory (LSTM) networks, and finite state transducers (FSTs) perform for pluralization of regular nouns. Since we understand the FST for pluralizations, we will use that knowledge to extract insight into how HMMs and LSTMs work with respect to pluralization. We will use Kimmo, which is a system of finite-state two-level models for computational phonology and morphology, as a baseline, and determine the rules that are being learned in the intermediate stages of the network. We will use a predetermined set of words and test our neural network on those.

Our goal is to investigate if the HMM and neural networks learn the rules in an FST by examining the activations of the cell state in the LSTM model.

In this paper, we first go over how we created our dataset, before diving into the Kimmo system as a baseline for the pluralization problem and the mathematics of HMMs and LSTMs in our Methodology section. We then explain our experiments in our Experiments section and results, where we go over specific analysis of the inner workings of the HMM and LSTM in its positive and negative results. We then wrap up in our conclusion with an analysis of parallelism between FSTs and HMMs and LSTMs.

2. Motivation

In this paper, we explore statistical methods for a basic linguistic system for pluralization. Specifically, we focus on using hidden markov models and neural networks and compare their behavior to the kimmo system which is based on finite state transducers. In the past decade, computational power and infrastructure have increased to the point where neural networks are feasible. This

resurgence of deep learning has affected multiple areas of computer science, including natural language processing. While deep learning has proven to be state-of-the-art for many problems such as question-and-answer, not much is known about the workings of these networks. Thus, by replicating and training statistical methods on basic datasets and comparing them to Kimmo system finite state transducers (FSTs), we hope to offer insight into the workings of recurrent neural networks (RNNs) and Hidden Markov Models (HMMs). We will also be the first to study HMMs for pluralization.

3. Previous Work

In terms of Hidden Markov models, there have not been any prior published papers that have attempted to model pluralization using HMMs.

The popularization of deep learning has resulted in an increase in popularity of the connectivist school of thought, which states that linguistics can be modeled in a non-symbolic way via the weights of a network as opposed to the classicist school of thought that believes that data can be represented symbolically in computer memory in a similar way it is stored in the brain [10]. Much prior work focuses on the cognitive integrity of grammars, deep learning, and bigram/tigram approaches to linguistic phenomena such as pluralization. Fodor and Lepore (1992, Ch. 6) [15] challenge connectivism by stating that human brains presumably vary significantly in the number of connections between their neurons, thus arguing that neural network implementations are not consistent with the way humans learn languages.

Berwick and Chomsky [5] investigate string substitution inference algorithm, Bayesian model selection algorithms, bigram/tigram statistical methods, and a neural network model. Reali and Christiansens [15] bigram model achieved an accuracy of 96%, while their simple recurrent neural network (SRN) model achieved an accuracy of 90%. To gain insight into the learning process of SRNs, they added a part of speech prefix of the test sentences up to the point at which grammatical versus ungrammatical divergence would occurfor example, for the pair, Is the boy who is hungry versus Is the boy who hungry is would follow the prefix sequence V DET N PRON. The bigram and trigram approach measures how frequently pairs or triples of adjacent words occur in a corpus. In Reali and Christiansens [25] bigram model, the probability of a sentence, $p(s)$, was expressed as the product of the probabilities of the words (w_i) that compose the sentence, with each word probability being conditional to the last $n1$ words. Berwick pointed out that the bigram overrelied on rules such as the who - is rule, and removal of these cues resulted in a dramatic drop in accuracy from 96% to 20% in their bigram model and 96% to 69% in their trigram model. They also found that the SRN was in fact simply acquiring knowledge about bigram lexical chunks, which would subject the network to the same downfalls as the bigram and trigram model.

Rumelhart and McClelland (1986) [30] developed a connectionist model of past tense acquisition in English on past-tense grammatical rules to model the three stages of language acquisition in children, creating an accuracy rate of 85%. However, Plunkett and Marchman [23] pointed out that the model did not rely on sound linguistic theory, and incorporated backpropagation and hidden units in order to find powerful generalizations to linguistic phenomena. They discovered that no-change verbs are more susceptible to overregularization than vowel change verbs. More recently, research has shifted from the comparison of neural networks and grammars to understanding how word embeddings and activation patterns impact linguistic accuracy. Dima and Hinrichs (2015) [8] applied deep learning and word embeddings to automatic classification of noun compounds, achieving a result of 79.48% F-1 score. This outperformed Tratz (2011), who used large, manually selected feature sets that were constructed for a specific task as the ini-

tial word representations on unseen compounds. Kadar (2017) [14] implemented the IMAGINET model introduced by Chrupaa, Kadar, and Alishahi (2015) to perform topic modelling on documents, finding that the model could utilize the sequential structure of language to interpret the same word type differently depending on the context. Rajana and Callison-Burch (2017) [24] similarly looked into improving neural embeddings of path representation in AntNET, a long short term memory (LSTM) based, morphology-aware neural network model for antonym detection. They were able to achieve higher precision compared to the path-based baseline for binary classification of antonyms and outperforms the SD model in precision, recall and F1 in the multiclass classification experiment.

For simplicity in our project, we will be focusing on pluralization linguistic phenomena and focusing on insights into long short-term memory (LSTM) networks and HMMs we can achieve from training RNNs on the two phenomena.

4. Dataset

We generated a corpus of singular nouns using NLTK [19], which has a textbook collection in the NLTK book module. The following books were used for collecting the dataset:

- text1: Moby Dick by Herman Melville
- text2: Sense and Sensibility by Jane Austen
- text3: The Book of Genesis
- text4: Inaugural Address Corpus
- text6: Monty Python and the Holy Grail
- text7: Wall Street Journal
- text8: Personals Corpus

We filtered these books by pos_tag and filtered all words which had pos as NN which is the tag for singular nouns. These words were then pluralized using the pattern [7] package in python. This gives us X, Y pairs where X is the singular version of the word and Y is the plural version of X. These samples were then bucketed into 4 categories:

- Words for which pluralization is by adding ‘s
- Words for which pluralization is by adding ‘es
- Words for which pluralization is by deleting y at the end of the singular word and adding ies
- Words that are not in any of the above categories.

Table 1: Word Distribution in the dataset

Word Categories	**Training Set**	**Test Set**
Words with plural -s ending	3150	350
Words with plural -es ending	3150	350
Words with plural -es ending	3150	350
Other words	450	50
Total Words	9900	1100

This process gave us a collection of 11,000 word pairs. We then split the dataset into training and test set, with 9900 samples in the train set and 1100 samples in the test set. The distribution of words in each categories in both training and test set are given in table 1.

Almost 50% of the words in the ”other” category have endings as ”man” for which the plural form is ending with ”men”. For example, plural of ”armyman” is ”armymen”. Rest of the words in the ”other” category are exceptions (for example, vacuum (plural is vacua), rice (plural is rice)).

5. Methodology

5.1. Kimmo System

As a starting point, to probe our RNN and set a point of comparison of which we already know the values, we use the Kimmo system to realize the surface form from the lexical form using a

simple example of pluralization. For a given set of words, we could realize the lexical forms using Kimmo and step through the process of realization and visualize the processes in action at each step of the computation. This will be useful for comparison later against our RNN probing.

Kimmo is a system using which one can realize the underlying lexical form of a word as the surface form, and generate the surface form the lexical form of a word. Lets take the word cat as an example for concreteness. Given the surface form of the word cat as input, Kimmo can realize the lexical form cat + s (Noun(cat)+PL). The reverse also works. Given the lexical form 'cat + s, it could realize he surface form cats.

Kimmo works via a system of finite state transducers working in parallel. Each finite state transducer represents a particular rule in the morphology of the language. In order to reach a success state, all the finite state transducers must be in an accepting state. If any finite state transducer rejects, then that branch of computation is rejected. In essence, the intersection of the computations accepted by all the finite state transducers is the result that Kimmo outputs. In the cases of ambiguity, Kimmo finds all the possibilities and outputs for the viewer. For example, dogs will be realized as both 'dog + s(Noun(dog)+ PL) or 'dog+s(Verb(dog) + 3sg.PRES.

There have been previous studies on using finite state transducers to parse words into their morphological forms. In chapter 3 of his book, *Speech and Language Processing*[13], Jurafsky explains how to use finite state transducers to split a word up into its many morphemes, including the stems and affixes (prefixes, suffixes, and infixes). Jurafsky splits the plural for regular nouns into two main cases illustrated by the two examples cat $\rightarrow$ cats and thrush $\rightarrow$ thrushes. There are other irregular nouns that Jurafsky notes. The first and the most commonly occurring rule for pluralization is to simply append an -s at the end of the noun. The second rule is if the noun ends in -s, -ss, -sh, -ch, -x, or -z we add -es at the end of the noun. Some more esoteric rules of pluralization exist as well . There are many more rarely occurring rules for pluralization beyond these. These fall under the category of irregular nouns as Jurafsky notes in his book. However, as our project is not about enumerating the various rules for English and creating a robust system for the language, but rather in drawing parallels between the Kimmo computations and a RNNs approach, we will focus solely on regular nouns where there are two pluralization rules :

1. Simply append -s or
2. Append -es if the noun ends in -sh, -ch

To achieve this focus and simplicity, we are hand selecting a set of words where we could correctly pluralize the singular base noun by appending an -s, or appending -es in the case that the ending is one of -s, -ss, -sh, -ch, -x, -z with no particular edge cases. The two sets of words will have no overlap and be mutually disjoint sets, again for simplicity. To handle our specific subset of carefully chosen words from the English language, we create a finite state transducer to be programmed into the Kimmo system. The schematic for this pluralization finite state transducer is given in Figure 1.

To elaborate, the symbol + indicates the appending of another morpheme, 0 indicates that nothing is realized in the surface form, # indicates the end of the word or string, @ indicates any other character transition. For instance, using this notation the pluralization of fox $\rightarrow$ foxes from lexical form to surface form would be fox+s foxes. As another example, the pluralization of cat $\rightarrow$ cats would be cat+s$\rightarrow$ cat0s. This is all captured in the finite state transducer for pluralization. The path $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4$ checks to see if the word ends in -sh or -ch. If so it ensures that -es is appended to the end. The other path $s_0 \rightarrow s_5 \rightarrow s_6$ is all other cases, in which case it simply appends -s. In the diagram s_{reject} is the reject state. All invalid transitions are blocked by having those transitions point to the rejecting

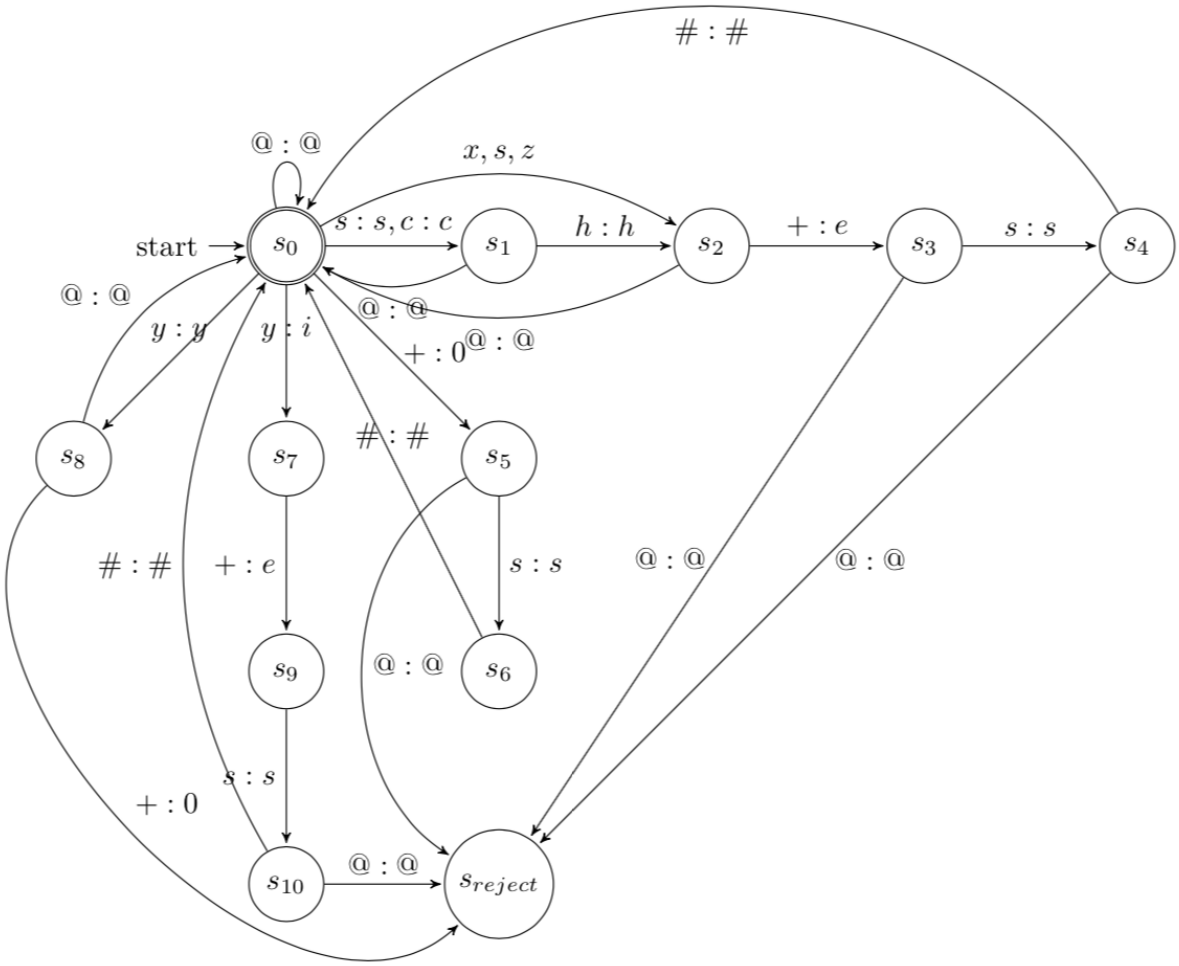

Figure 1: FST for pluralization

state. The start state s_0 is the only accepting state. So in essence, this FST only permits valid pluralization patterns via transitions back to the start state, and rejecting anything that lands in the rejecting state s_{reject}.

5.2. Hidden Markov Models

Hidden Markov models (HMMs) have been used in Natural Language Processing with applications to parts-of-speech (POS) tagging. HMMs are essentially Markov models which follow the Markov property of $P(X_n = x_n \mid X_{n-1} = x_{n-1}, \ldots, X_0 = x_0) = \mathbb{P}(X_n = x_n \mid X_{n-1} = x_{n-1})$. Let us define a few preliminary objects. Hidden markov models consist of a hidden state set $S_1, S_2, .., S_n$. Let there be M distinct observation states, and denote the set of all possible output symbols as V = $v_1, v_2, .., v_M$. Denote the output symbol at time t as O_t and the hidden state at time t as Q_t. There exists a transition probability matrix A such that $a_{i,j}$ is the probability of

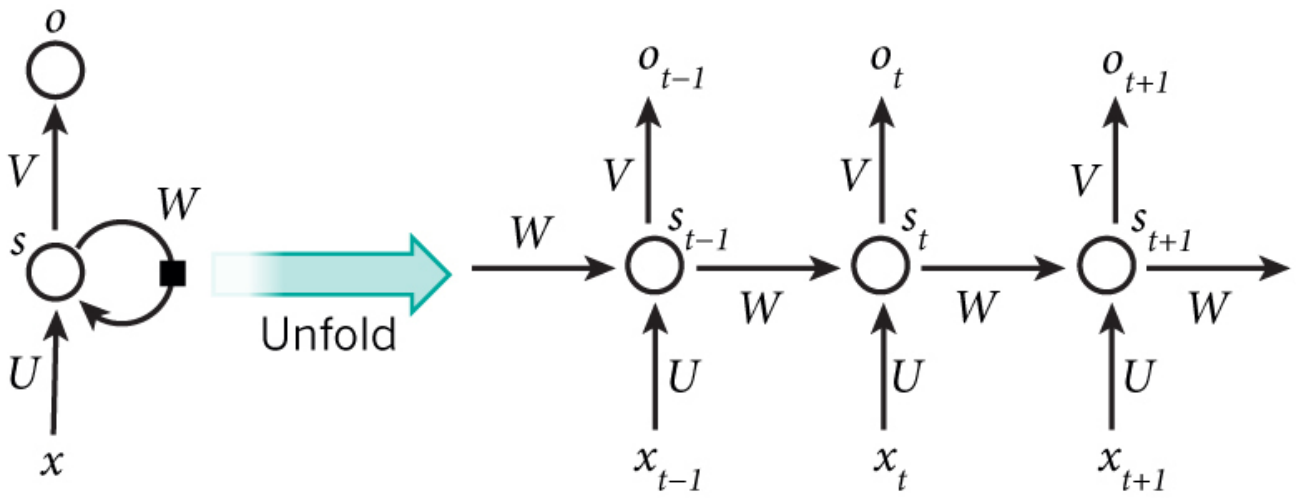

Figure 2: Visualization of RNN [4]

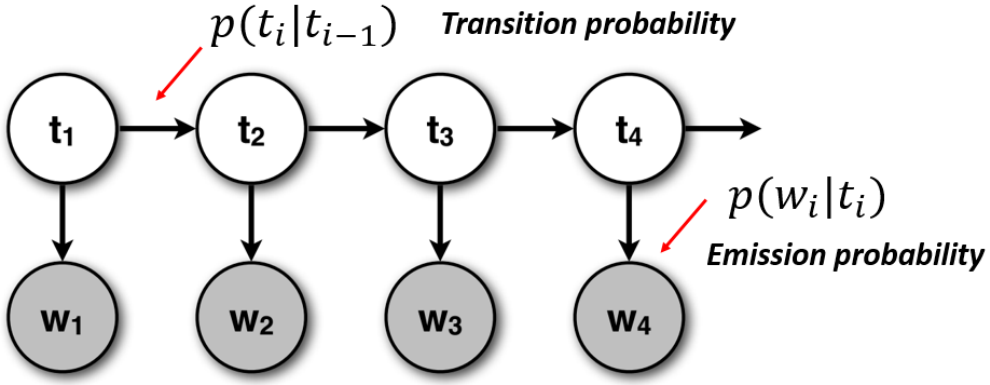

Figure 3: Diagram of HMM [18]

moving from state S_i to state S_j and

$$\sum_{n=1}^{\infty} a_{i,j} = 1$$

Let $p_i(O_t)$ represent the observation probability (or emission probability) of observing O_t given S_i. Finally, define S_0 and S_l as the start and end states such that there are only outward transitions from S_0 ($a_{0,1}$, $a_{0,2}$,..., $a_{0,n}$) and inward transitions into S_f ($a_{f,1}$, $a_{f,2}$,...,$a_{f,n}$). Note that the probability of an output observation is independent from any other hidden states or observations.

HMMs can be used with sequential data inputs, as well as problems that necessitate outputting sequences. Given the observation sequences and an initialized emission and transition probabilities, HMMs finetune its probabilities to maximize the probability of producing the observation sequence given the hidden states using the Viterbi algorithm. This task of determining which sequence of variables is the most probable sequence of hidden states given an observation is called the decoding task.

For the purposes of the Viterbi algorithm, let us define $q(v|w, u)$ as the probability that the hidden state S_v is visited after S_w and S_u.

The Viterbi algorithm works as follows: First, find the distribution $\pi(k, u, v)$ where $1 \leq$ k $\leq$ n,

and u and v represents all the values that each of the hidden state sequences can take (namely v_1, v_2,..., v_M) such that

$$\pi(k, u, v) = \max_{w \in V}(\pi(k-1, w, u) * q(v|w, u) * p_v(O_k))$$

The output X_k of most likely sequence of hidden variables given an observation thus is

$$X_1, X_2, ...X_k = \max_{1 \leq k \leq n}(\pi(k, u, v))$$

HMM parameters are trained using the Expectation Maximizing (EM) method, which maximizes log-likelihood. The EM method works as follows.

Given observed sequence O and hidden states S, we want to find parameters θ_{MLE} such that it maximizes the probability of predicting the observed states with the hidden states. We do this in two steps.

Let t = 1,2,3,...

$$E - Step : Q(\theta, \theta_t) = E_{S|O,\theta_t}(\log(p(X, Z|\theta)))$$

$$M - Step : \theta_{t+1} = arg\max_{\theta} Q(\theta, \theta_t)$$

and iterate on t until convergence. In the E-step, we calculate the expectation with respect to the conditional distribution of Z given X with the current best parameters θ.

Thus, using EM, Hidden Markov Models can be fine-tuned to learn the parameters that will fit data [27].

5.3. Neural Networks

In this paper, we use a sequence to sequence model which takes input character-by-character [28] and outputs the pluralized version of the word character-by-character. A seqeuence to sequence model has an encoder and a decoder. We use LSTMs for both encoder and decoder. The encoder is given the singular word as the input and is expected to encode the important information for performing pluralization in the output vectors from the encoder. We first explain RNNs which are fundamental towards understanding LSTMs, and then go through the working of LSTMs and its implementation details.

5.3.1 Recurrent Neural Networks

One of the initial developments in recurrent networks based on human cognition is Simple Recurrent Network (SRN) [9]. In a non-recurrent neural network, we assume no dependency on the sequence of the inputs, but this is not a good assumptions for language processing. In language processing, we know that a future character or word depends on what we have processed so far. The idea behind SRNs are that to take decision at time t, we need the decisions taken until t-1. Recurrent networks use sequential information to decide at time t by feeding the output of the previous step and the current input. They use the same recurrent operation which takes in the current input and the output from previous operation to predict at time t. Figure 2 shows a basic interpretation of the recurrent network architecture.

In the above figure, x_t is the input at time step t. One of the ways to represent this is using a one-hot vector. There is a hidden state at time t represented as s_t. One can interpret this as a vector representing the information processed until time t. It is computed as $s_t = f(Ux_t + Ws_{t-1})$ where s_{t-1} is the hidden state at time t-1. The function f is a nonlinear function such as tanh or ReLU [22]. For technical completeness, we initialize s_{-1} as a vector of zeroes. The output of the layer at time t is o_t, which is vector of probabilities over possible words or characters, calculated as $softmax(Vs_t)$. A RNN shares the parameters U, V, W across all recurrent calls of the hidden layer, which tells us that we are performing the same operation at each step.

RNNs are trained using an extension of the regular backpropagation algorithm [26], called backpropagation through time [21]. The general idea is to backpropagate the error using gradient of the predicted output with respect to the true output. This gradient is then used to update the network parameters, weights and biases of the different layers. Since the parameters U, V, W are shared by the layers at all time steps of the RNN, the

gradient at each time step depends on the output of current time step but also on the output of the previous calculations. Backpropagation through time enables us to compute gradient till time t and sum the gradients to compute the gradient used to update the weights. Since RNNs have many recurrent multiplications, the training suffers from vanishing/exploding gradients. One of the ways of dealing with this problem is proper initialization of the W matrix. Another way is to use ReLU [22] instead of tanh or sigmoid functions. ReLUs derivative is a constant, 0 or 1, which gets rid of the vanishing gradients.

Recurrent networks have proven to be effective in many different tasks, such as machine translation [29, 2], speech recognition [11], and generating image description [17]. One of the prominent RNN architectures is char-rnn. It is a character level model, which reads one character at a time as the input and also predicts one character at a time. It models the probability distribution of the next character based on the sequence of previous characters. This allows us to generate new text one character at a time. Some of the experiment present substantial evidence that RNNs learn the concept of periods and quotes by 300 iterations [16].

5.3.2 Long Short-Term Memory

Long short-term memory networks (LSTMs) are a special architecture of RNN which enable learning long-term dependencies [12]. They were designed especially to avoid the long-term dependency problem. These networks retain important information for the task for long periods of time. They also have structure which have the ability to add or remove information in that layer. LSTMs are composed of a cell, an input gate, output gate, and a forget gate. The cell is the main container of information in a LSTM regulated by different gates.

The first step is the forget gate which controls what information should be removed from the cell state. The forget gate is made of a sigmoid layer, which looks at h_{t-1} and x_t and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. If the forget gate calculates 0 for a cell value, then we completely forget that information, and if the forget gate calculates 1 then we retain that information.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

The next gate is the input gate which decides what new information should be stored in the cell state. This gate consists of two parts, a sigmoid layer and a tanh layer. The sigmoid layer controls which values of the cell should be updated. Then the tanh layer generates a vector of new values C_t which are then added using the sigmoid layers result to the cell.

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$$
$$C_t = tanh(W_C[h_{t-1}, x_t] + b_C)$$

Once we have calculated C_t, we calculate the cell state C_t. The new state is calculated as C_{t-1} multiplied by f_t and adding $i_t * C_t$. The first part gives us the information that need to be retained from C_{t-1} and the second part of the equation provides the new information that should be included in C_t.

$$C_t = f_t * C_{t-1} + i_t * C_t$$

For the output of the LSTM layer, we output two sets of values, the hidden states and the output states. For the output state we apply a sigmoid layer to decide what information we need to output. This is computed as:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

Then we can multiply this o_t with tanh of the cell state C_t.

$$h_t = o_t * tanh(C_t)$$

For our experiments, we use a sequence to sequence model which consists of an encoder and

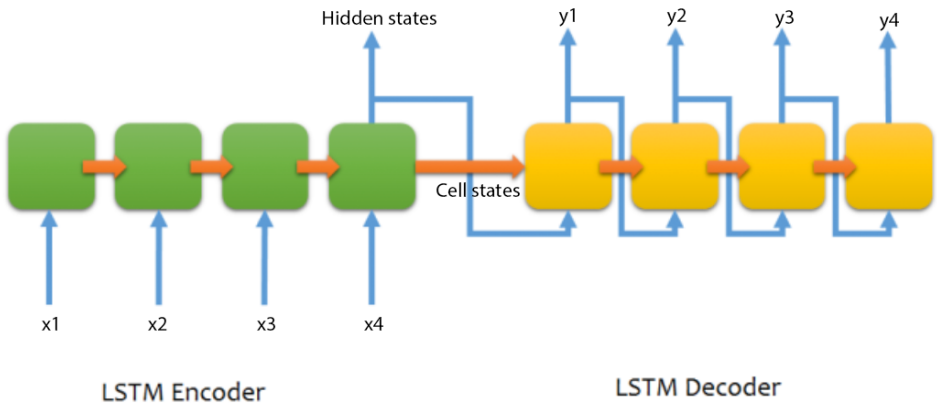

Figure 4: Diagram of LSTM Encoder Decoder model

decoder, which are both LSTMs. We run two kinds of experiments, the first experiment is based on retention of the word and also pluralization. The encoder is given the input word character-by-character and is expected to encode the information required by the decoder to output the plural form of the word character-by-character. We chose a character based model so that we can investigate the behavior of the models on a character level. This is also helpful for probing the network with different outputs. We use the back propagation in time algorithm to train the model on 9900 samples from the training set and evaluate it on the 1100 samples of the test set. We only count the output to be correct if the decoder outputs the correctly retained word along with the correct pluralized form. For example, on input "apple" we only count the output as correct if the output is "apples". If the model misses any of the characters or appends the wrong plural suffix, then we mark the output as incorrect. We used keras with a tensorflow backend for the implementation of the LSTM layers [6, 1]. We trained 10 different models, where we used different number of latent representation (same as number of neurons for this case) from the encoder. We sample the number of latent representation uniformly from 25 to 250 (i.e. 25, 50, 75, and so on). We trained these networks for 300 epochs using RMSProp as the optimizer [31].

This is a very difficult task as we are combining two tasks of retaining the word and also pluralization in the same network. To understand the behavior of RNNs, we also train similar sequence to sequence models where the encoder is given the word character-by-character and the decoder just needs to output the plural ending for the input. So, the decoder can output "s", "es", "ies", and "" (empty string). An empty string is the expected output for the words which don't belong to the normal categories of pluralized form (with endings as 's', 'es', and 'ies'). With this experiment, we explore the patterns learned by the models and also the failures of the models. Even for the ending predicting network, we train 10 different models, where we used different number of latent representation from the encoder. We sample the number of latent representation uniformly from 5 to 50 (i.e. 5, 10, 15, and so on). All these networks were trained for 75 epochs using RMSprop as the optimizer.

6. Experiments

6.1. HMM

Our supervised HMM model takes as input the lexical form of the words in the same way Kimmo does. During training the HMM takes as target

outputs the surface form of the words. For example, if the sample word is apple the lexical input to the HMM model would be the array "apple+s". The output of the HMM would be the surface form of the word. So using apple as our example, the output would be "apple s". A summary of the various types of inputs are given below in the following table. Each row is one noun of each category, where we have three categories as shown in the FST diagram in Figure 6. To reiterate, the first category is the case where a simple s is appended, the second category is where -es is appended after ch, z, sh, x. Then theres the third case, where a y is realized as a surface i with -es appended at the end. Examples of input for each of the three categories is illustrated in Figure 5.

Similar to the Kimmo system, the input (lexical) length and the output (surface) length need to be the same for the HMM implementation as well. In the Kimmo system, as illustrated in our diagram, a "+" could be realized as a 0 to be realized as an empty string. Likewise, the "+" could be realized as an "e" if "es" appendage is necessary. We realize the lexical form as either a blank space if only a simple addition of an "s" at the end or as an "e" character if an "es" needs to be added at the end (Figure 5) We then later post process the blanks spaces out from all the output words for displaying purposes.

Our Hidden Markov Model implementation consist of 27 hidden states, each representing the set of observation value space (a-z and the lexical marker " ") in the surface form. We will draw connections between linguistic phenomena in Kimmo with HMMs by probing the emission probabilities. For example, if the probability of the state at time i+1 being "b" given the state at time i is "c" is significantly higher than the probability of the state at time i+1 taking any other value means that the HMM has learned a rule "b $\rightarrow$ c", where it translates the lexical "b" into surface-form "c". Thus, by probing the emission probabilities, we will gain insight into the inner workings of HMMs.

7. Testing & Results

7.1. Hidden Markov Models

Looking at the heat map of the probabilities of transitions from certain lexical characters to surface characters, it is evident that the HMM is learning the common patterns of lexical to surface realization. The reason that there is a diagonal of very high probabilities is that in the general case it is highly likely that the surface form character will just be the lexical form character. There is also very high probability of the "+" symbol being realized as "e" or the blank symbol, because these are the predominant samples in the dataset, where a -s is appended or -es is appended. The reason there is a high probability of "y" to "i" probability is because the ending "y" all have to be realized as an "i" for proper pluralization according to English rules. There are also some non-negligant moderate probability of "f" to "v" transitions. The most likely reason for this is that there are some words in our dataset that require a "f" to "v" realization during the pluralization process. One such example is "half" to "halves".

We were able to achieve 78.82% accuracy with Hidden markov models.

Below are some examples of the three categories of pluralization and the mistakes the HMM made.

When it came to "y $\rightarrow$ ies" rules, the HMM learned to weight the "y $\rightarrow$ i" transition much higher than "y $\rightarrow$ consonent" transitions, which caused it to erroneously correct words such as the below.

```
singular:  hyphema
actual:  hyphemas
prediction:  hiphemas
```

Here, the HMM replaced all instances of "y" in the lexical form to "i" in the surface form.

However, the HMM was able to correctly pluralize words that ended with "y" and had only one instance of "y" in the singular form.

```
singular:  spry
```

Endings	Lexical Form (Input in HMM)	Surface Form (Target in HMM)
ch, sh, z, x, s	["w", "a", "t", "c", "h", "+", "s"]	["w", "a", "t", "c", "h", "e", "s"]
y	["t", "a", "]", "]", "y", "+", "s"]	["t", "a", "]", "]", "i", "e", "s"]
regular ending	["c", "a", "t", "+", "s"]	["c", "a", "t", " ", "s"]

Figure 5: Input and Output for Supervised HMM model

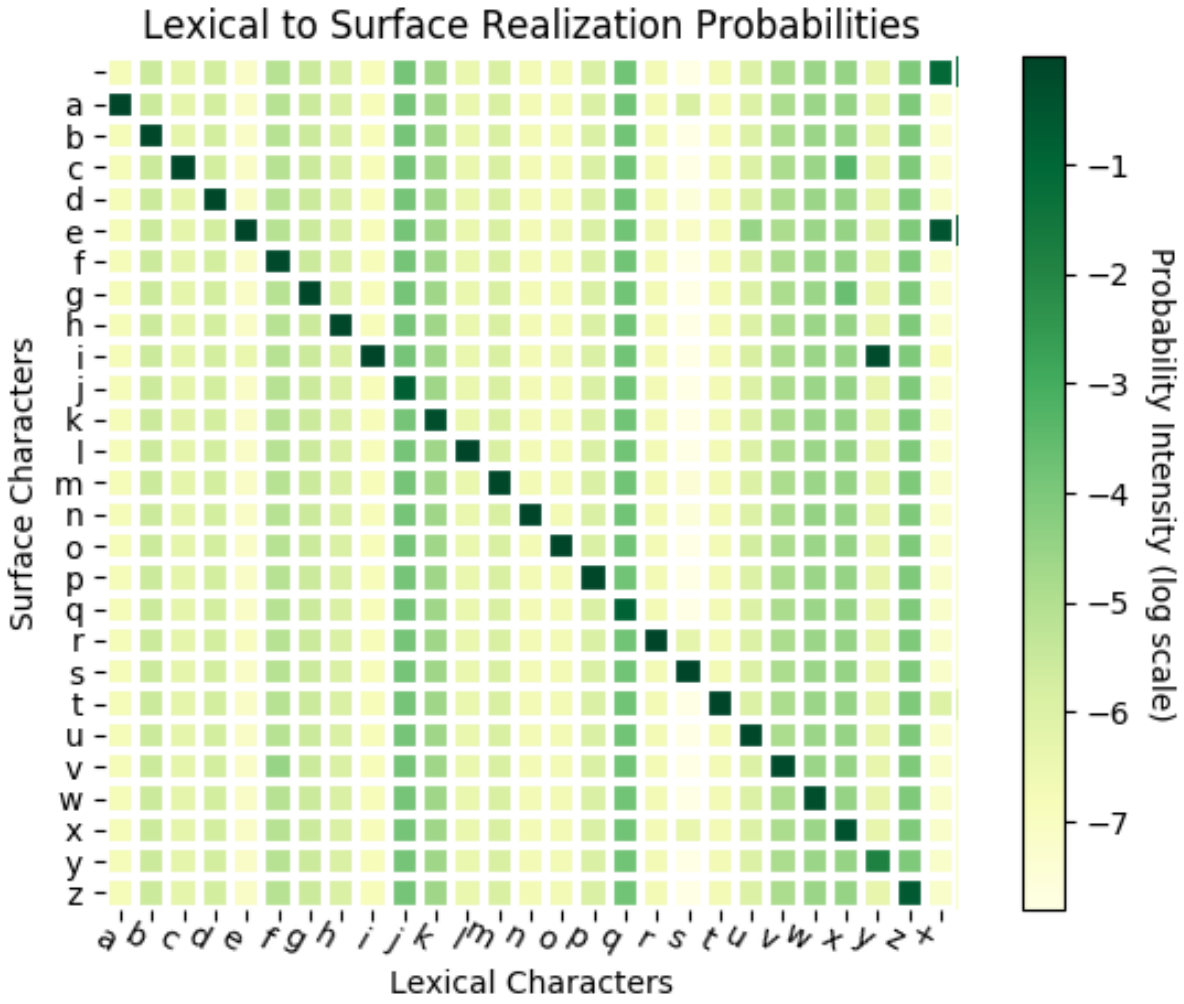

Figure 6: HeatMap of Lexical to Surface Probabilities

```
plural:  spries
```

The HMM also was biased to pluralize words by appending an ”-es” such as below for words that should simply have an ”s” appending for pluralization.

```
singular:  crawl
actual:  crawls
prediction:  crawles
```

This example shows that the plus sign is more likely to be realized as a ”e” rather than a blank is that there are twice as many samples that need

the plus sign to be changed to an ”e” than a blank since nouns that have ”y $\rightarrow$ ies” pluralizations can also be interpreted as changing the y to ”e” and then appending ”es” to the resulting form. The HMM, however, has not learned what ”es” can be appended to. Thus, the HMM predictions will have bias towards ”es” rather than a ”_s” pluralizations. We confirmed this bias by training our HMM on a dataset with a lower distribution of ”ies” and ”es” plural nouns, which resulted in a higher prediction accuracy.

The HMM has learned that ”s” should be appended to vowels that are not ”o”, as well as certain consonants. For example,

```
singular:  apple
plural:  apples
```

The HMM has also learned to append ”es” to ”o”

```
singular:  tho
plural:  thoes
```

Furthermore, the HMM also incorrectly predicts plurals of irregular nouns such as the below.

```
singular:  doberman
plural: dobermen
prediction: dobermans
```

This is because of the relative lack of irregular nouns in our dataset.

7.2. Neural Networks

In figure 7, we present the accuracy of the 10 LSTM models with different number of neurons for predicting the pluralized word. As we can infer from the graph, any model with more than 75 neurons in the LSTM layer is overfitting over the training set. The accuracy of all the models with more than 75 neurons is above 99%. This is a surprising result as there are so many sequences of characters that the network has to learn, which it seems to be doing with just a few 100 neurons. But, as we can observe the trend of the test accuracy, we do not see generalizable results as the accuracy of any of these models is not above 86.9%.

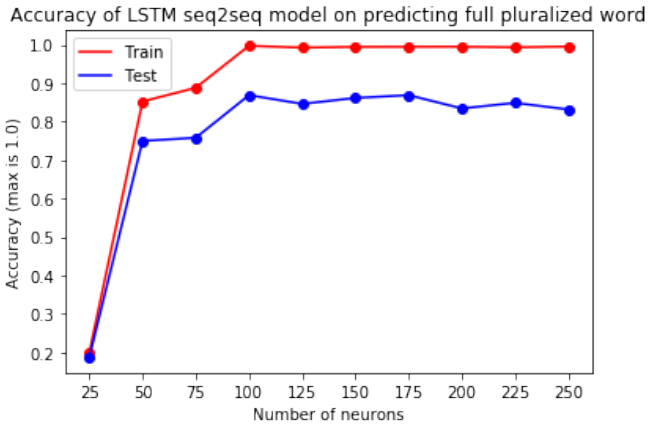

Figure 7: Accuracy of LSTM model for predicting the plural word

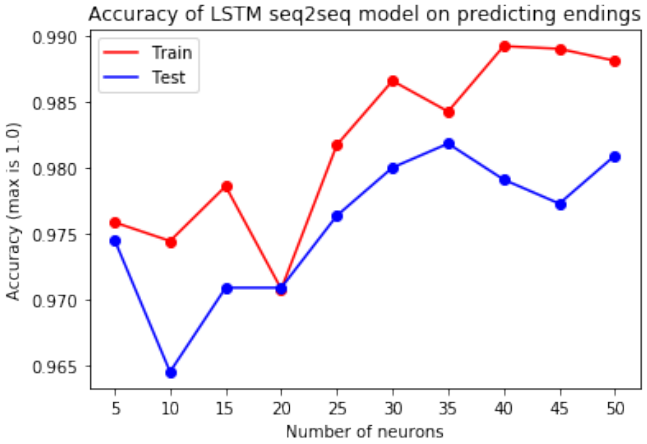

Figure 8: Accuracy of LSTM model for predicting ending of the the plural words

Following are a few positive samples that are predicted by a model using 75 neurons which had an accuracy of 85.27% on the training set and 75% on the test set.

```
singular: apple
plural: apples

singular: torch
plural: torches

singular: convexity
plural: convexities
```

```
singular: man
plural: men
```

We can observe in the above samples that the LSTM with 75 neurons is able to learn correct pluralization rules like appending 's' to "apple" and 'es' to "torch". It also learns to remove 'y' when it the word ends with 'y' and appending 'ies' to the end. It also seems to have learned some pattern to process "man" to "men".

Although the model has an accuracy of 75% on the test set, it had failure cases as listed below.

```
singular: volvox
actual: volvoxes
predicted: volviches

singular:  autobiography
actual: autobiographies
predicted:  autobiogharies

singular: armyman
actual: armymen
predicted: armynams
```

From the examples listed above, we can see that the model is unable to retain the information for reconstructing the word as in the case of "volvox", "autobiographies", and "armyman". But the model correctly appends the pluralization for "volvox" and "autobiographies". To investigate the capabilities of LSTMs to pluralize words, we limited our second experiment to just prediction of the pluralization suffix.

In figure 8, we present the accuracies of the 10 LSTM models with different number of neurons for predicting the plural ending of the input word. We can observe that all models are performing reasonably well above 96%. For the purpose of simplicity and clarity, we analyzed the model with 5 neurons. Following are some of the positive examples to show the performance of the model with 5 neurons.

```
singular: apple
plural: 's'

singular: torch
plural: 'es'

singular: convexity
plural: 'ies'
```

Given the examples above and test accuracy of 96.45%, we can observe that the model with 5 neurons learns to predict the correct plural ending for the words. But, we discovered some adversarial examples for which this model fails. To remind the reader from the methodology section, this model outputs '' (empty string) for words belonging to the irregular nouns (other category in the dataset).

```
singular: chewy
actual: 'ies'
predicted: 's'

singular: cheesy
actual: 'ies'
predicted: 's'

singular: pronoun
actual: 's'
predicted: ''
```

To understand the behavior of the cell state activations, we plotted the mean subtracted activations of each of the four ending categories. It is important to analyze the cell state activations as they hold the information required by the decoder to predict the plural ending for the word. To elaborate, we compute the cell state activations for each substring prefix of the words in the test set. For example, given the word 'puzzle', we compute the cell state activations outputted by the encoder for the input "p", "pu", "puz", "puzz", "puzzl", and "puzzle". We then retain the cell state activations for the last 5 substrings prefixes. So, for the word "puzzle", we retain the cell state

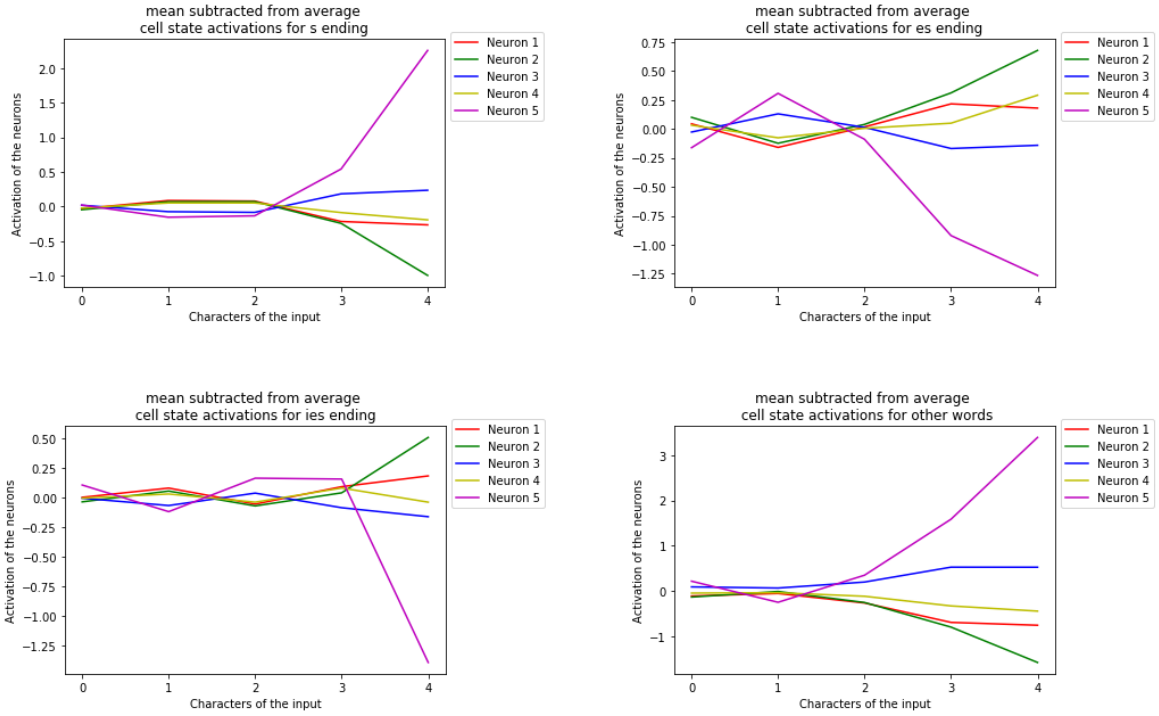

Figure 9: Cell state activations for each ending category with mean subtracted of all cell state activations

activations for “pu”, “puz”, “puzz”, “puzzl”, and “puzzle”. These last 5 substring activations are retained for all the words in the test set that are of length greater than 5 (1049 words out of 1100 words). We then compute the mean of all these activations, which we will refer as `mean_c`.

Similarly, we compute the mean of the cell state activations for words belonging to each plural ending category. We will refer to these as `mean_c_s_ending`, `mean_c_es_ending`, `mean_c_ies_ending`, and `mean_c_other_ending`. In figure 9, we plot the subtraction of `mean_c` from the category wise mean (i.e. top left is `mean_c_s_ending` - `mean_c`).

Interestingly, we find some of the patterns of the neuron activations based on visual analysis of the plots presented in figure 9.

- In the plot for s ending category, we just observe an increase in neuron 5 activations along with divergence of the activations of neuron 2 and neuron 5 as we approach the end of sequence. Neuron 5 activations has a positive slope and neuron 2 activations have a negative slope in the plot. We cannot find a reason for this behavior.

- In the plot for es ending category, we observe a drop in the activation of neuron 5 in the last two characters (marked as 3 and 4 on the x-axis). This can be attributed to the detection of ‘c’ and ‘s’ in the second last character (index 3) followed by ’h’ in the last character, and also detection of ‘s’, ‘x’, and ‘z’.

- In the plot for the ies ending category, we observe a sudden decrease in the activation of neuron 5 on the last character (marked as 4th index on the x-axis). Also, we observe

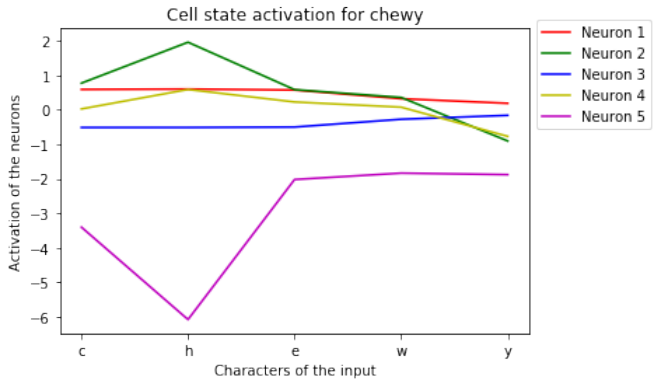

Figure 10: Cell state activations for the word "chewy"

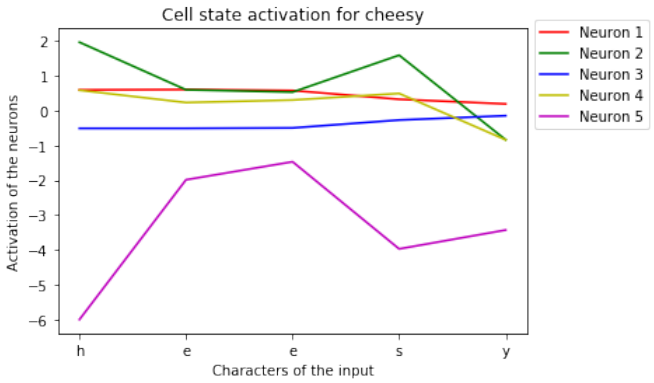

Figure 11: Cell state activations for the word "cheesy"

a divergence in the activations of neuron 2 and neuron 5. This can be attributed to the detection of 'y' as the last character.

- In the plot for the other words category, very similar to the plot of s ending category, we find increase in the activation of neuron 5 and divergence of the activations of neuron 2 and neuron 5 as we approach the end of sequence. Even in this plot, neuron 5 activations has a positive slope and neuron 2 activations have a negative slope in the plot. The only distinction between the two categories is that the activation of neuron 5 for the other word category is observed to be higher on an average at the end of the sequence.

The observations above are not proofs but just observations based on the average behavior of the neurons.

To understand why the model doesn't predict the correct plural endings for "chewy", "cheesy", and "pronoun", we investigated the neuron activations for these words. As we did while computing the mean, we plot the 5 neuron activation in the cell state for the last 5 substring prefixes. We present the plot of the cell states activations for the word chewy in figure 10. Below are the predicted plural endings for each of last 5 substrings of the words "chewy", "cheesy", and "pronoun".

```
Input: c Predicted ending: s
Input: ch Predicted ending: es
Input: che Predicted ending: s
Input: chew Predicted ending: s
Input: chewy Predicted ending: s

Input: ch Predicted ending: es
Input: che Predicted ending: s
Input: chee Predicted ending: s
Input: chees Predicted ending: es
Input: cheesy Predicted ending: s

Input: pro Predicted ending: s
Input: pron Predicted ending: s
Input: prono Predicted ending: s
Input: pronou Predicted ending: ''
Input: pronoun Predicted ending: ''
```

In the figure 10 for the word "chewy", we can see that neuron 2 and neuron 5 are not diverging. Neuron 5 activations have a positive slope (moving to slope = 0) and neuron 2 have a negative slope towards the end of the word. Assuming the statistical trends from figure 9, this behavior is the closest to that for s ending category. Similarly, we can see a similar pattern in the activations for the word "cheesy" in figure 11.

This behavior can be attributed to the 'ch' at the beginning of the two inputs. As according to the plot for es endings in figure 9, neuron 5 activation decreases on observing 'ch' in succession.

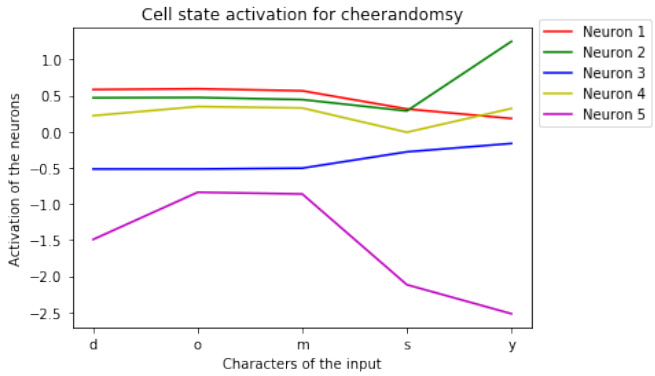

Figure 12: Cell state activations for the word "cheerandomsy"

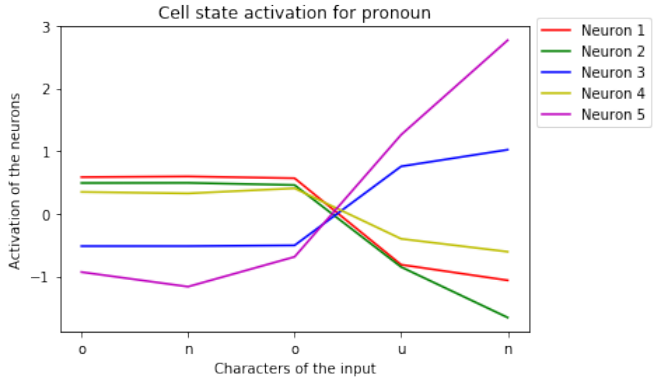

Figure 13: Cell state activations for the word "pronoun"

So we experiment by adding a string "random" before 'sy' in "cheesy" to create the test word "cheerandomsy". For this input, the model outputs the correct plural ending 'ies'. We can see in figure 12 that the pattern of the activations is similar to that of the average pattern of 'ies' endings. This points out that neurons require characters to remover from any increase and decrease in activation. But an adversarial example is adding more 'e' in the middle of "cheesy" to form inputs like "cheeeeeeeesy". For such input, the output of the model still remains as 's'. This behavior cannot be explained considering the high dimensionality of possible strings.

We show the activations for the input "pronoun" in figure 13. The activation of neuron 5 are close to that of other category which seems the reason why the model predicts empty string which is the output for the words in the other category. We cannot find the cause for this increase in the activation of neuron 5 as we don't know any statistical patterns in the other category which are similar to the input "pattern".

8. Evaluation

8.1. HMM

Our HMM model was able to perform at 78% accuracy. From our analysis, we can see that FSTs work better than HMMs because of statistical bias towards our dataset, which caused it to give an over-bias of appending "es" as a method of pluralization. On the other hand, since FSTs are deterministic and have very definite rules for pluralization, it is able to get 100% accuracy for the pluralization problem.

8.2. Neural Network

On the test set evaluation, the best accuracy of neural network for the predicting full word with pluralization was 86.9% (model with 100 neurons) and best accuracy of network for predicting the plural endings was 98% (model with 50 neurons). Even though these networks have good statistical accuracy, we found them very influenced by the previous sequences of characters. We found that by tweaking the inputs in random ways changes the behavior of the model. As compared to FSTs, neural networks took longer sequences to forget the effect of character sequences such as 'ch'. We know that FSTs have a deterministic way of processing sequences which generalizes well for well studied rules such as pluralization. Such deterministic behavior was not observed in the neural networks. Overall, we observed that neural networks learned some patterns from the finite dataset, but they did not generalize well on the test data and were also easily fooled by making small perturbations in the input sequence.

9. Conclusion

In this paper, we explored two other techniques besides FST's (HMM and LSTM) for recognizing the pluralized surface form from the lexical form. We found that the HMM and LSTM learned via a statistical approach and predicted the most common occurrences and patterns. The FST on the other hand, was more robust and determines the surface form by hard-coded spelling rules.

One area of future work is to build the FST from the HMM. It may be possible to write an algorithm to cluster the HMM states into different clusters. For example, low probability transitions could be grouped together to form one new state in the FST. Right now different characters are each a unique state in the HMM, so clustering the low probability transitions means that the insignificant lexical to surface transitions in the root of the word are clustered as state s_0 as indicated in the FST diagram given in Figure 6. Higher probability transitions could be its own separate state, such states as s_2, s_5, s_7, s_8. This process would be similar to grammar induction for probabilistic context free grammars. In the process, we would iteratively cluster the transitions and states similar to how grammar rule induction works. However all this will not be exact and it would be a challenge to recover the exact FST from the probabilities matrix of the HMM. Future work is needed to discover the optimal algorithm for forming a FST from HMM internal information.

Similar work on generating FSTs could be done using LSTMs by analyzing activations of the networks for different character sequences. This is an exponential process as there are exponential number of strings that have to be analyzed using the network. However, it will be interesting to see if we can find small branches of the FST using a finite set of sequences. Moreover, we can experiment with the network type and training methods. One possible direction is to use attention models which also pass activations of each character (attention vectors) to the decoder [3]. This enables to perform the same experiment of predicting the full pluralized word with much better accuracy as the cell state from the encoder will just contain the information about pluralization. Retention of the word in the decoding process will be possible using the attention vectors which are also passed from the encoder.

Another area of future work is that our project can be extended by conducting similar experiments using more complex linguistic principles such as subject object consistency and handling different languages to see the effectiveness of statistical methods in comparison to non-statistical methods in those domains.

Our investigations into pluralization using HMMs and deep learning also parallelize to how children acquire language. Researchers at University of Massachusetts Amherst [20] have discovered that over-generalization of pluralization is relatively rare, occurring less than 4% of the time. The two competing psycho-linguistic schools of thoughts are those that argue children learn algebra-like rules, while the second propose that children learn specific instances of pairings between singular and plural forms and produce novel plural forms as a consequence of similarity-based generalizations over these specific instances [32].

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] M. Auli, M. Galley, C. Quirk, and G. Zweig. Joint language and translation modeling with recurrent neural networks. 2013.

[3] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[4] D. Batista. Hidden markov model and naive bayes relationship. `http://www.davidsbatista.net/blog/2017/11/11/HHM_and_Naive_Bayes/`.

[5] R. C. Berwick, P. Pietroski, B. Yankama, and N. Chomsky. Poverty of the stimulus revisited. *Cognitive Science*, 35(7):1207–1242, 2011.

[6] F. Chollet et al. Keras. `https://keras.io`, 2015.

[7] T. De Smedt and W. Daelemans. Pattern for python. *J. Mach. Learn. Res.*, 13:2063–2067, June 2012.

[8] C. Dima and E. Hinrichs. Automatic noun compound interpretation using deep neural networks and word embeddings. In *Proceedings of the 11th International Conference on Computational Semantics*, pages 173–183, 2015.

[9] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211.

[10] J. Garson. Connectionism, 1997.

[11] A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *International Conference on Machine Learning*, pages 1764–1772, 2014.

[12] S. Hochreiter and J. Schmidhuber. Long short-term memory. 9:1735–80, 12 1997.

[13] D. Jurafsky and J. H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

[14] A. Kádár, G. Chrupała, and A. Alishahi. Representation of linguistic form and function in recurrent neural networks. *Computational Linguistics*, 43(4):761–780, 2017.

[15] X.-N. C. Kam, I. Stoyneshka, L. Tornyova, J. D. Fodor, and W. G. Sakas. Bigrams and the richness of the stimulus. *Cognitive Science*, 32(4):771–787, 2008.

[16] A. Karpathy. The unreasonable effectiveness of recurrent neural networks. *Andrej Karpathy blog*, 2015.

[17] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[18] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[19] E. Loper and S. Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[20] G. Marcus. Children's overregularization and its implications for cognition. In *Models of language acquisition: Inductive and deductive approaches*, 2000.

[21] M. Mozer. A focused backpropagation algorithm for temporal pattern recognition. 3, 01 1995.

[22] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[23] K. Plunkett and V. A. Marchman. Learning from a connectionist model of the acquisition of the english past tense. *Cognition*, 61(3):299–308, 1996.

[24] S. Rajana, C. Callison-Burch, M. Apidianaki, and V. Shwartz. Learning antonyms

with paraphrases and a morphology-aware neural network. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (* SEM 2017)*, pages 12–21, 2017.

[25] F. Reali and M. H. Christiansen. Uncovering the richness of the stimulus: Structure dependence and indirect statistical evidence. *Cognitive Science*, 29(6):1007–1028, 2005.

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[27] B. Schuster-Böckler and A. Bateman. An introduction to hidden markov models. *Current protocols in bioinformatics*, 18(1):A–3A, 2007.

[28] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[29] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[30] M. S. Thomas and J. L. McClelland. Connectionist models of cognition. *Cambridge handbook of computational cognitive modelling*, pages 23–58, 2008.

[31] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[32] J. A. Zapf and L. B. Smith. When do children generalize the plural to novel nouns? *First Language*, 27(1):53–73, 2007.