Robsut Wrod Reocginiton via Semi-Character Recurrent Neural Network*

Keisuke Sakaguchi[†] **Kevin Duh**[‡] **Matt Post**[‡] **Benjamin Van Durme**[†‡]
[†]Center for Language and Speech Processing, Johns Hopkins University
[‡]Human Language Technology Center of Excellence, Johns Hopkins University
{keisuke,kevinduh,post,vandurme}@cs.jhu.edu

Abstract

Language processing mechanism by humans is generally more robust than computers. The *Cmabrigde Uinervtisy* (*Cambridge University*) effect from the psycholinguistics literature has demonstrated such a robust word processing mechanism, where jumbled words (e.g. Cmabrigde / Cambridge) are recognized with little cost. On the other hand, computational models for word recognition (e.g. spelling checkers) perform poorly on data with such noise.

Inspired by the findings from the *Cmabrigde Uinervtisy* effect, we propose a word recognition model based on a semi-character level recurrent neural network (scRNN). In our experiments, we demonstrate that scRNN has significantly more robust performance in word spelling correction (i.e. word recognition) compared to existing spelling checkers and character-based convolutional neural network. Furthermore, we demonstrate that the model is cognitively plausible by replicating a psycholinguistics experiment about human reading difficulty using our model.

Introduction

Despite the rapid improvement in natural language processing by computers, humans still have advantages in situations where the text contains noise. For example, the following sentences, introduced by a psycholinguist (Davis 2003), provide a great demonstration of the robust word recognition mechanism in humans.

> *Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer be at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe.*

This example shows the *Cmabrigde Uinervtisy* (*Cambridge University*) effect, which demonstrates that human reading is resilient to (particularly internal) letter transposition.

*Parts of this manuscript are intentionally jumbled to demonstrate the robust word processing ability of you, the reader.

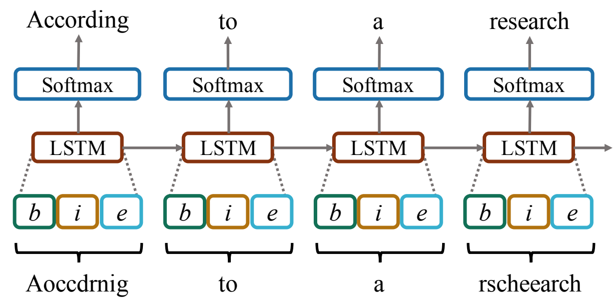

Figure 1: Schematic Illustration of semi-character recurrent neural network (scRNN).

Robustness is an important and useful property for various tasks in natural language processing, and we propose a computational model which replicates this robust word recognition mechanism. The model is based on a standard recurrent neural network (RNN) with a memory cell as in long short-term memory (Hochreiter and Schmidhuber 1997). We use an RNN because it has shown to be state-of-the-art language modeling (Mikolov et al. 2010) and it is also flexible to realize the findings from the *Cmabrigde Uinervtisy* effect. Technically, the input layer of our model consists of three subvectors: beginning (b), internal (i), and ending (e) character(s) of the input word (Figure 1). This semi-character level recurrent neural network is referred as scRNN.

First, we review previous work on the robust word recognition mechanism from psycholinguistics literature. Next, we describe technical details of scRNN which capture the robust human mechanism using recent developments in neural networks. As closely related work, we explain character-based convolutional neural network (CharCNN) proposed by Kim et al. (2015). Our experiments show that the scRNN significantly outperforms commonly used spelling checkers and CharCNN by (at least) 42% for jumbled word correction and 3% and 14% in other noise types (insertion and deletion). We also show that scRNN replicates recent findings from psycholinguistics experiments on reading difficulty depending on the position of jumbled letters, which indicates that scRNN successfully mimics (at least a part of) the robust word recognition mechanism by humans.

Cond.	Example	# of fixations	Regression(%)	Avg. Fixation (ms)
N	The boy could not solve the problem so he asked for help.	10.4	15.0	236
INT	The boy cuold not slove the probelm so he aksed for help.	11.4*	17.6*	244*
END	The boy coudl not solev the problme so he askde for help.	12.6†	17.5*	246*
BEG	The boy oculd not oslve the rpoblem so he saked for help.	13.0‡	21.5†	259†

Table 1: Example sentences and results for measures of fixation excerpt from Rayner et al. (2006). There are 4 conditions: N = normal text; INT = internally jumbled letters; END = letters at word endings are jumbled; BEG = letters at word beginnings are jumbled. Entries with * have statistically significant difference from the condition N ($p < 0.01$) and those with † and ‡ differ from * and † with $p < 0.01$ respectively.

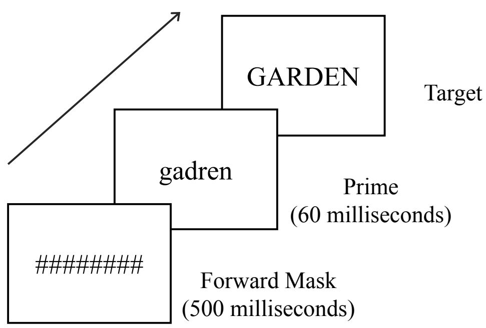

Figure 2: Example of the masked priming procedure.

Raeding Wrods with Jumbled Lettres

Sentence processing with jumbled words has been a major research topic in psycholinguistics literature. One popular experimental paradigm is *masked priming*, in which a (lower-cased) stimulus, called *prime*, is presented for a short duration (e.g. 60 milliseconds) followed by the (upper-cased) target word, and participants are asked to judge whether the target word exists in English as quickly as possible (Figure 2).[1] The prime is consciously imperceptible due to the instantaneous presentation but it proceeds to visual word recognition by participants. The masked priming paradigm allows us to investigate the machinery of lexical processing and the effect of prime in a pure manner.

Forster et al. (1987) show that a jumbled word (e.g. gadren-GARDEN) facilitates primes as large as identity primes (garden-GARDEN) and these results have been confirmed in cases where the transposed letters are not adjacent (caniso-CASINO) (Perea and Lupker 2004) and even more extreme cases (sdiwelak-SIDEWALK) (Guerrera and Forster 2008).

These findings about robust word processing mechanism by humans have been further investigated by looking at other types of noise in addition to simple letter transpositions. Humphreys, Evett, and Quinlan (1990) show that deleting a letter in a word still produces significant priming effect (e.g. blck-BLACK), and similar results have been shown in other research (Peressotti and Grainger 1999; Grainger et al. 2006). Van Assche and Grainger (2006) demonstrate that a priming effect remains when inserting a character into a word (e.g. juastice-JUSTICE).

Another popular experimental paradigm in psycholinguistics is *eye-movement* tracking. In comparison to the masked priming technique, eye-movement paradigm provides data from normal reading process by participants. Regarding word recognition, the eye-tracking method has shown the relationship between a word difficulty and the eye fixation time on the word: when a word is difficult to process, average time to fixation becomes long. In addition, words that are difficult to process often induce regressions to words previously read.

With the eye-movement paradigm, Rayner et al. (2006) and Johnson, Perea, and Rayner (2007) conduct detailed experiments on the robust word recognition mechanism with jumbled letters. They show that letter transposition affects fixation time measures during reading depending on which part of the word is jumbled. Table 1 presents the result from Rayner et al. (2006). It is obvious that people can read smoothly (i.e. smaller number of fixations, regression, and average of fixation duration) when a given sentence has no noise (referring to this condition as N). When the characters at the beginning of words are jumbled (referring to this condition as BEG), participants have more difficulty (e.g. longer fixation time). The other two conditions, where words are internally jumbled (INT) or letters at word endings are jumbled (END), have similar amount of effect, although the number of fixations between them showed a statistically significant difference ($p < 0.01$). In short, the reading difficulty with different jumble conditions is summarized as follows: N < INT ≤ END < BEG.

It may be surprising that there is statistically significant difference between END and BEG conditions despite the difference being very subtle (i.e. fixing either the first or the last character). This result demonstrates the importance of beginning letters for human word recognition.[2]

Semi-Character Recurrent Neural Network

In order to achieve the human-like robust word processing mechanism, we propose a semi-character based recurrent neural network (scRNN). The model takes a semi-character vector (x) for a given jumbled word, and predicts a (correctly

[1]There is another variant for masked priming technique, where backward mask is inserted between the prime and target in addition to the forward mask.

[2]While there is still ongoing debate in the psycholinguistics community as to exactly how (little) the order of internal letters matter, here we follow the formulation of Rayner et al. (2006), considering only the letter order distinctions of BEG, INT, and END.

spelled) word (y) at each time step. The structure of scRNN is based on a standard recurrent neural network, where current input (x) and previous information is connected through hidden states (h) by applying a certain (e.g. sigmoid) function (g) with linear transformation parameters (W) and the bias (b) at each time step (t).

One critical issue of vanilla recurrent neural networks is that it is unable to learn long distance dependency in the inputs due to the vanishing gradient (Bengio, Simard, and Frasconi 1994). To address the problem, Hochreiter and Schmidhuber (1997) introduced long short-term memory (LSTM), which is able to learn long-term dependencies by adding a memory cell (c). The memory cell has an ability to discard or keep previous information in its state. Technically, the LSTM architecture is given by the following equations,

$$i_n = \sigma\left(W_i\left[h_{n-1}, x_n\right] + b_i\right) \tag{1}$$

$$f_n = \sigma\left(W_f\left[h_{n-1}, x_n\right] + b_f\right) \tag{2}$$

$$o_n = \sigma\left(W_o\left[h_{n-1}, x_n\right] + b_o\right) \tag{3}$$

$$g_n = \sigma\left(W_g\left[h_{n-1}, x_n\right] + b_g\right) \tag{4}$$

$$c_n = f_n \odot c_{n-1} + i_n \odot g_n \tag{5}$$

$$h_n = o_n \odot \tanh\left(c_n\right) \tag{6}$$

where σ is the (element-wise) sigmoid function and $\odot$ is the element-wise multiplication.

While a standard input vector for RNN derives from either a word or a character, the input vector in scRNN consists of three sub-vectors (b_n, i_n, e_n) that correspond to the characters' position.

$$x_n = \begin{bmatrix} b_n \\ i_n \\ e_n \end{bmatrix} \tag{7}$$

The first and third sub-vectors (b_n, e_n) represent the first and last character of the n-th word. These two sub-vectors are therefore one-hot representations. The second sub-vector (i_n) represents a bag of characters of the word without the initial and final positions. For example, the word "University" is represented as $b_n = \{U = 1\}$, $e_n = \{y = 1\}$, and $i_n = \{e = 1, i = 2, n = 1, s = 1, r = 1, t = 1, v = 1\}$, with all the other elements being zero. The size of sub-vectors (b_n, i_n, e_n) is equal to the number of characters (N) in our language, and x_n has therefore the size of $3N$ by concatenating the sub-vectors.

Regarding the final output (i.e. predicted word y_n), the hidden state vector (h_n) of the LSTM is taken as input to the following softmax function layer with a fixed vocabulary size (v).

$$y_n = \frac{\exp\left(W_h \cdot h_n\right)}{\sum_v \exp\left(W_h \cdot h_n\right)} \tag{8}$$

We use the cross-entropy training criterion applied to the output layer as in most LSTM language modeling works; the model learns the weight matrices (W) to maximize the likelihood of the training data. This should approximately correlate with maximizing the number of exact word match in the predicted outputs. Figure 1 shows a pictorial overview of scRNN.

In order to check if the scRNN can recognize the jumbled words correctly, we test it in spelling correction experiments. If the hypothesis about the robust word processing mechanism is correct, scRNN will also be able to read sentences with jumbled words robustly.

Character-based Neural Network

Another possible approach to deal with reading jumbled words by neural networks is (pure) character-level neural network (Sutskever, Martens, and Hinton 2011), where both input and output are characters instead of words. The character-based neural networks have been investigated and used for a variety of NLP tasks such as segmentation (Chrupala 2013), dependency parsing (Ballesteros, Dyer, and Smith 2015), machine translation (Ling et al. 2015), and text normalization (Chrupała 2014).

For spelling correction, Schmaltz et al. (2016) uses character-level convolutional neural networks (CharCNN) proposed by Kim et al. (2015), in which the input is character but the prediction is at the word-level. More technically, according to Kim et al. (2015), CharCNN concatenates the character embedding vectors into a matrix $P_n \in \mathbb{R}^{d\times l}$ whose k-th column corresponds to the k-th character embedding vector (size of d) of n-th word which contains l characters. A narrow convolution is applied between P and *filter* $H \in \mathbb{R}^{d\times w}$ of width w, and then *feature map* $f_n \in \mathbb{R}^{l-w+1}$ is obtained by the following transformation[3] with a bias b.

$$f_n = \tanh(\mathrm{Tr}(P_n[:, k : k + w - 1]H^T) + b) \tag{9}$$

This is interpreted as a process of capturing important feature f with filter H to maximize the predicted word representation y_n by the *max-over-time*:

$$y_n = \max_k f_n[k] \tag{10}$$

Although CharCNN and scRNN have some similarity in terms of using a recurrent neural network, CharCNN is able to store richer representation than scRNN. In the following section, we compare the performance of CharCNN and scRNN with respect to jumbled word recognition task.

Experiments

We conducted spelling correction experiments to judge how well scRNN can recognize noisy word sentences. In order to make the task more realistic, we tested three different noise types: *jumble*, *delete*, and *insert*, where the *jumble* changes the internal characters (e.g. Cambridge → Cmbarigde), *delete* randomly deletes one of the internal characters (Cambridge → Camridge), and *insert* randomly inserts an alphabet into an internal position (Cambridge → Cambpridge). None of the noise types change the first and last characters. We used Penn Treebank for training, tuning, and testing.[4]

[3] In the equation, $P_n[:, k : k+w-1]$ means the k-to-$(k+w-1)$-th column of P_n.

[4] Section 2-21 for training, 22 for tuning, and 23 for test `https://catalog.ldc.upenn.edu/ldc99t42`. The data includes 39,832 sentences in training set (898k/950k tokens

Original	Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy , it deos n't mttaer in waht oredr the ltteers in a wrod are , the olny iprmoetnt tihng is taht the frist and lsat ltteer be at the rghit pclae . The rset can be a toatl mses and you can sitll raed it wouthit porbelm . Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef , but the wrod as a wlohe .
Correct	According to a researcher at Cambridge University , it does n't matter in what order the letters in a word are , the only important thing is that the first and last letter be at the right place . The rest can be a total mess and you can still read it without problem . This is because the human mind does not read every letter by itself , but the word as a whole .
CharCNN (Kim et al.)	According to a research at Cambridge Minority , it deck nt mother in wait or the letters in a wood are , the tony Vermont timing is taxi the tourist and sat letter be at the fruit pile . The reset can be a total uses and you can vital rake it worthy parallel . Mips is abuse the human trim deck not rake survey latter by leftist , but the wood as a whole .
Enchant	Ecuadoran to a searcher at Brigade Nerviness , it does n't matter in what order the letters in a word are , the only omnipresent thing is that the freest and slat letter be at the right place . The rest can be a total mess and you can still read it outhit corbel . Tish is Ceausescu , the human mind does not read Hervey letter by leftist , but the word as a whole .
Commercial A	Occurring to a scholarch at Cambridge Inertias , it does n't matter in what order the letters in a word are , the only impotent thing is that the first and last letter be at the right place . The rest can be a total mess and you can still read it outhit problem . This is bcuseae the human mind does not read every letter by istle , but the word as a whole .
Commercial B	Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy , it does n't matter in what order the letters in a word are , the only iprmoetnt thing is that the first and last letter be at the right place . The rest can be a total mess and you can still read it wouthit problem . Tihs is bcuseae the human mind does not read every letter by itself , but the word as a whole .
scRNN (proposed)	According to a research at Cambridge University , it does n't matter in what order the letters in a word are , the only important thing is that the first and last letter be at the right place . The rest can be a total mess and you can still read it without problem . This is because the human mind does not read every letter by itself , but the word as a whole .

Table 2: Example spelling correction outputs for the *Cmabrigde Uinervtisy* sentences. Words that the system failed to correct are underlined. CharCNN stands for the character-based convolutional neural network by Kim et al. (2015).

The input layer of scRNN consists of a vector with length of 76 (A-Z, a-z and 24 symbol characters). The hidden layer units had size 650, and total vocabulary size was set to 10k. We applied one type of noise to every word, but words with numbers (e.g. 1980s) and short words (length $\leq$ 3) were not subjected to jumbling, and therefore these words were excluded in evaluation. We trained the model by running 5 epochs with (mini) batch size 20. We set the backpropagation through time (BPTT) parameter to 3: scRNN updates weights for previous two words (x_{n-2}, x_{n-1}) and the current word (x_n).

For comparison, we evaluated CharCNN on the same training data[5], and also compared widely-used spelling checkers (Enchant[6], Commercial A, and Commercial B[7]).

Spelling correction results

Table 2 presents example outputs for the *Cmabrigde Uinervtisy* sentence by each model.[8] It may be surprising that CharCNN performs poorly compared with other spelling checkers. This is probably because the CharCNN highly depends on the order of characters in the word and the transposed characters adversely affected the recognition performance. Enchant, Commercial A, and Commercial B tend to fail long word correction. This may be because these models are not designed for severely jumbled input but they

	Jumble	Delete	Insert
CharCNN (Kim et al.)	17.17	21.30	35.00
Enchant	57.15	37.01	88.54
scRNN (proposed)	**98.96**	**85.74**	**96.70**

Table 3: Spelling correction accuracy (%) with different error types on the entire test set.

	Jumble	Delete	Insert
CharCNN (Kim et al.)	16.18	19.76	35.53
Enchant	57.59	35.37	89.63
Commercial A	54.81	60.19	93.52
Commercial B	54.26	71.67	73.52
scRNN (proposed)	**99.44**	**85.56**	**97.04**

Table 4: Spelling correction accuracy (%) with different error types on the subset of test set (50 sentences).

are covered by the top 10k vocabulary), 1,700 sentences in the tuning set (coverage 38k/40k), and 2,416 sentences in test set (coverage 54k/56k).

[5]For CharCNN, we employed the codebase available at `https://github.com/yoonkim/lstm-char-cnn.git`

[6]`http://www.abisource.com/projects/enchant/`

[7]We anonymized the name of the commercial product.

[8]The *Cmabrigde Uinervtisy* sentences contains jumbling as well as deletion, insertion, and replacement of characters. Note that we used a single scRNN (Jumble), and didn't train scRNN separately for each error type in this example.

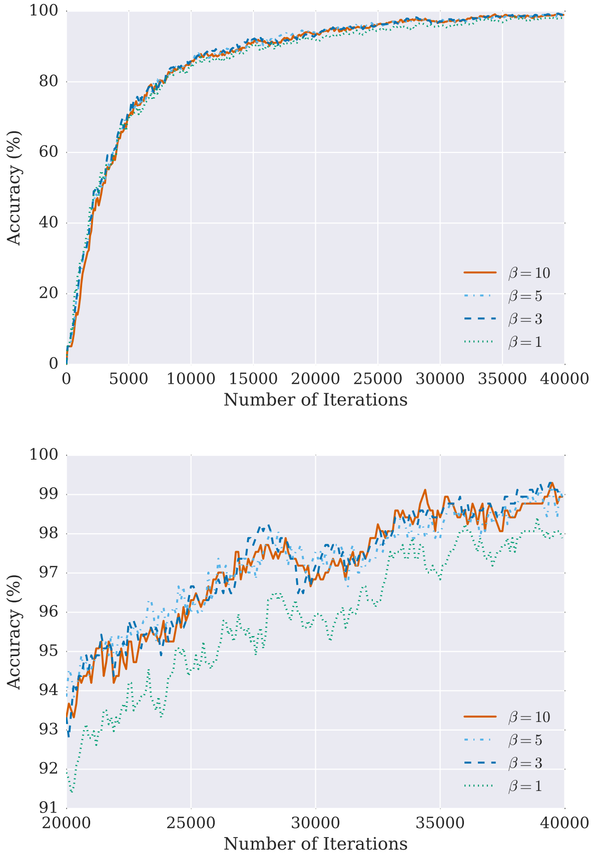

Figure 3: Learning curve of training scRNN with different BPTT parameter (on dev set): first 40k iterations (top) and its enlarged view between 20k and 40k iterations (bottom).

are likely to depend on edit distance between the incorrect and correct words. While these existing models struggle with correcting *Cmabrigde Uinervtisy* sentence, we see that scRNN demonstrates significantly better recognition ability. The only error in scRNN may be because the last character (rscheearc**h**) activated the scRNN nodes strongly toward *research* instead of *researcher*.[9]

Table 3 and 4 show the overall result on the test set with respect to noise type. We also tested spelling correction on a small subset (50 sentences) because of the API limits etc. of commercial systems. Overall, as seen in the example above, scRNN significantly outperforms the other spelling checker models across all three noise types. Since scRNN is especially designed for jumbled word recognition, it is not surprising that it performs particularly well on *jumble* noise. However, it is striking that scRNN outperforms the other models in deletion and insertion errors as well.[10] This clearly demonstrates the robustness of scRNN.

The relatively large drop in *delete* in scRNN may be because the information lost by deleting character is significant. For example, when the word *place* has dropped the character *l*, the surface form becomes *pace*, which is also a valid word. Also, the word *mess* with *e* being deleted produces the form of *mss*, which can be recovered as *mess*, *mass*, *miss*, etc. In the *Cmabrigde Uinervtisy* sentences, in both cases, the local context support other phrase such as 'at the right *pace/place*' and 'a total *mass/mess*'. These examples clearly demonstrate that deleting characters harm the word recognition more significantly than other noise types. All the models perform relatively well on *insert* noise, indicating that adding extraneous information by inserting a letter does not change the original information significantly.

β	Accuracy (%)	SD
1	98.69	0.53
3	98.96	0.45
5	98.91	0.40
10	98.95	0.43

Table 5: scRNN accuracy (%) on jumbled word recognition with different BPTT parameters. There were no statistically significant differences among them.

Units	Acc (%)	SD	Size (KB)
5	24.65	2.59	236
10	48.43	3.26	435
15	73.32	3.65	632
20	84.82	2.39	830
30	94.15	1.54	1,255
40	96.90	1.26	1,670
50	98.48	0.94	2,092
60	98.39	0.81	2,514

Table 6: scRNN accuracy (%), the standard deviation, and the size of model file (KB) on jumbled word recognition with respect to the number of units of LSTM.

With respect to a learning curve on scRNN (Figure 3, top), we found that the model achieves 0.9 (in accuracy) at the 15,000-th iteration of the mini batch. This can be made within a hour with a CPU machine, which demonstrates simplicity of scRNN compared with CharCNN.

Figure 3 also shows the effect of BPTT size (β), and the accuracy on test set is presented in Table 5. As explained, β indicates the context length of updates during training. It is surprising that the longer contexts ($\beta = 5, 10$) do not necessarily yield better performance. This is probably because it rarely happens that the context plays an important role on distinguishing ambiguous representation (e.g. anagrams) in scRNN. If we take closer look at the learning curve (Table 3, bottom), however, there is a clear gap in learning efficiency between with and without contexts (i.e. β=1 vs. the rest).

Finally, we reduced the number of units in the hidden layer to see the model size and performance of scRNN. Surprisingly, as Table 6 presents, scRNN with 50 units already achieves comparable results to 650 units (Table 3). The result suggests that 50 units (2 MB) are enough to distinguish 10k English words.

[9] There is also an deletion of 'r'.

[10] It is important to note that some commercial systems have constraints of the model size (Church, Hart, and Gao 2007).

Cond.	Example	Accuracy
INT	As a relust , the lnik beewetn the fureuts and sctok mretkas rpiped arapt .	98.96
END	As a rtelus , the lkni betwene the feturus and soctk msatrek rpepid atarp .	98.68*
BEG	As a lesurt , the lnik bweteen the utufers and tocsk makrtes pipred arpat .	98.12†
ALL	As a strule , the lnik eewtneb the eftusur and okcst msretak ipdepr prtaa .	96.79‡

Table 7: Example sentences and results for spelling correction accuracy by scRNN variants depending on different jumble conditions: INT = internal letters are jumbled; END = letters at word endings are jumbled; BEG = letters at word beginnings are jumbled; ALL = all letters are jumbled. Entries with * have a difference with marginal significance from the condition INT ($p = 0.07$) and those with † and ‡ differ from * and † with $p < 0.01$ respectively.

Cond.	Examples of errors (correct/wrong)
INT	Once/once, Under/under, Also/also, there/three, form/from, fares/fears, trail/trial, Broad/Board
END	being/begin, quiet/quite, bets/best, stayed/steady, heat/hate, lost/lots + same errors in INT
BEG	Several/reveal, Growth/worth, host/shot, creditors/directors, views/wives + same errors in INT
ALL	Under/trend, center/recent, licensed/declines, stop/tops + same errors in INT, END, & BEG

Table 8: Error analysis of scRNN variants.

Corroboration with psycholinguistic experiments

As seen in the literature review in psycholinguistics, the position of jumbled characters affects the cognitive load of human word recognition. We investigate this phenomenon with scRNN by manipulating the structure of input vector. We replicate the experimental paradigm in Rayner et al. (2006), but using scRNN rather than human subjects. We trained scRNN variants depending on different jumble conditions: INT, END, BEG, and ALL. INT is the same model as explained in the previous section ($x_n = [b_n, i_n, e_n]^T$). END represents an input word as a concatenation of the initial character vector (b) and a vector for the rest of characters ($x_n = [b_n, i_n + e_n]^T$). In other words, in END model, the internal and last characters are subject to jumbling. BEG model combines a vector for the final character (e) and a vector for the rest of characters ($x_n = [b_n + i_n, e_n]^T$). In other words, initial and internal characters are subject to jumbling. In ALL model, all the letters are subject to jumble (e.g. research vs. eesrhrca) and represented as a single vector ($x_n = [b_n + i_n + e_n]^T$). This is exactly the same as bag of characters. We trained all the scRNN variants with $\beta = 3$, the number of hidden layer units being 650, and total vocabulary size to be 10k.

Table 7 shows the result. While all the variants of scRNN achieve high accuracy, the statistical test revealed that INT and END have a difference with statistically marginal significance ($p = 0.07$). There are statistically significant differences ($p < 0.01$) both in END&BEG and BEG&ALL. From the results, the word recognition difficulty of different jumbled types is summarized as INT $\leq$ END $<$ BEG $<$ ALL, which is the same order as the finding in Table 1. It is not surprising that INT outperform the other variants because it has richer representation in x_n (twice or three times larger than the other variants). However, it is interesting to see that END outperforms BEG both in Rayner et al. (2006) and our experiment despite that the size of x_n between END and BEG models are equal. This suggests the scRNN replicates (at least a part of) the human word recognition mechanism, in which the first letter is more important and informative than the last one in English.

For qualitative analysis, Table 8 shows some errors (correct/wrong) that each variant made. All the scRNN variants often fail to recognize capitalized first character (e.g. Once/once, Under/under), specifically when the word is at the beginning of the sentence. Other than the capitalization errors, most errors come from anagrams. For example, errors in INT (the original scRNN) are internally anagrammable words (e.g. there/three, form/from). END model made errors on words that are anagrammable with the first character being fixed (e.g. being/begin, quiet/quite). BEG model, on the other hand, failed to recognize anagrammable words with the last character being the same (e.g. creditors vs. directors, views vs. wives). In addition, BEG model often ignores the first (upper-cased) character of the word (e.g. Several/reveal, Growth/worth). Finally, ALL model failed to recognize anagrammable words (e.g. center/recent, licensed vs. declines). Although scRNN generally disambiguated anagrammable words successfully from context, all these examples from the error analysis are straightforward and convincing when we consider the characteristics of each variant of scRNN.

Summary

We have presented a semi-character recurrent neural network model, scRNN, which is inspired by the robust word recognition mechanism known in psycholinguistics literature as the *Cmabrigde Uinervtisy* effect. Despite the model's simplicity compared to character-based convolutional neural networks (CharCNN), it significantly outperforms widely used spelling checkers with respect to various noise types. We also have demonstrated a similarity between scRNN and human word recognition mechanisms, by showing that scRNN replicates a psycholinguistics experiment about word recognition difficulty in terms of the position of jumbled characters.

There are a variety of potential NLP applications for scRNN where robustness plays an important role, such as normalizing social media text (e.g. *Cooooolll* $\rightarrow$ *Cool*), post-processing of OCR text, and modeling morphologically rich languages, which could be explored with this model in future work.

References

Ballesteros, M.; Dyer, C.; and Smith, N. A. 2015. Improved transition-based parsing by modeling characters instead of words with lstms. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 349–359. Lisbon, Portugal: Association for Computational Linguistics.

Bengio, Y.; Simard, P.; and Frasconi, P. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5(2):157–166.

Chrupala, G. 2013. Text segmentation with character-level text embeddings. *arXiv preprint arXiv:1309.4628*.

Chrupała, G. 2014. Normalizing tweets with edit scripts and recurrent neural embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 680–686. Baltimore, Maryland: Association for Computational Linguistics.

Church, K.; Hart, T.; and Gao, J. 2007. Compressing trigram language models with Golomb coding. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 199–207. Prague, Czech Republic: Association for Computational Linguistics.

Davis, M. 2003. Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy. *http://www.mrc-cbu.cam.ac.uk/people/matt.davis/cmabridge/*.

Forster, K. I.; Davis, C.; Schoknecht, C.; and Carter, R. 1987. Masked priming with graphemically related forms: Repetition or partial activation? *The Quarterly Journal of Experimental Psychology* 39(2):211–251.

Grainger, J.; Granier, J.-P.; Farioli, F.; Van Assche, E.; and van Heuven, W. J. 2006. Letter position information and printed word perception: the relative-position priming constraint. *Journal of Experimental Psychology: Human Perception and Performance* 32(4):865.

Guerrera, C., and Forster, K. 2008. Masked form priming with extreme transposition. *Language and Cognitive Processes* 23(1):117–142.

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

Humphreys, G. W.; Evett, L. J.; and Quinlan, P. T. 1990. Orthographic processing in visual word identification. *Cognitive Psychology* 22(4):517 – 560.

Johnson, R. L.; Perea, M.; and Rayner, K. 2007. Transposed-letter effects in reading: Evidence from eye movements and parafoveal preview. *Journal of Experimental Psychology: Human Perception and Performance* 33(1):209.

Kim, Y.; Jernite, Y.; Sontag, D.; and Rush, A. M. 2015. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*.

Ling, W.; Trancoso, I.; Dyer, C.; and Black, A. W. 2015. Character-based neural machine translation. *arXiv preprint arXiv:1511.04586*.

Mikolov, T.; Karafiát, M.; Burget, L.; Cernocký, J.; and Khudanpur, S. 2010. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, 1045–1048.

Perea, M., and Lupker, S. J. 2004. Can CANISO activate casino? transposed-letter similarity effects with nonadjacent letter positions. *Journal of Memory and Language* 51(2):231 – 246.

Peressotti, F., and Grainger, J. 1999. The role of letter identity and letter position in orthographic priming. *Perception & Psychophysics* 61(4):691–706.

Rayner, K.; White, S. J.; Johnson, R. L.; and Liversedge, S. P. 2006. Raeding wrods with jubmled lettres: There is a cost. *Psychological Science* 17(3):192–193.

Schmaltz, A.; Kim, Y.; Rush, A. M.; and Shieber, S. 2016. Sentence-level grammatical error identification as sequence-to-sequence correction. In *Proceedings of the 11th Workshop on Innovative Use of NLP for Building Educational Applications*, 242–251. San Diego, CA: Association for Computational Linguistics.

Sutskever, I.; Martens, J.; and Hinton, G. E. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 1017–1024.

Van Assche, E., and Grainger, J. 2006. A study of relative-position priming with superset primes. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 32(2):399.