ISSN: 2393-8528

Contents lists available at www.ijicse.in

International Journal of Innovative Computer Science & Engineering

Volume 5 Issue 3; May-June 2018; Page No. 06-13

Text Completion, Classification and Correction Using Different Types of Neural Networks

Manav Mehra and Riya Sahal

B.M.S College of Engineering

ABSTRACT

This article reviews the various methods used in the process of completing and correcting text on a word based as well as a sentence based approach. Firstly, we look at the process of understanding the text from scratch which requires the application of deep learning to text that facilitates the understanding the inputs at a character level and other concepts related to abstract texts, and all this helps us in classifying texts. For the above said purposes we use different types of Artificial Neural Networks which include Convolutional Neural Networks and Recurrent Neural Networks. Both Convolutional and Recurrent Convolutional neural networks can be individually used in the classification of texts and Recurrent Neural Networks along with Convolutional Networks are used in the completion and correction. The result of using temporal and character level Convolutional network for classification are also shown to be competitive to the results produced by traditional techniques. The model for correction and completion enables both the processes to occur simultaneously by ciphering the hidden representations at a character level and decoding the revised sequence.

Keywords: Machine Learning, Artificial Neural Network, Convolutional Neural Network, Recurrent Neural Network, Computer Vision, Supervised Learning

1. Introduction

In recent years, deep learning has embarked successfully on computer vision and language modelling. The field of natural language processing is shifting from statistical methods to neural network methods.In natural language processing, deep learning methods involves learning word vector representations through neural language models and arranging them to perform classification.Word vector means converting words into vectors, which deep learning algorithms can understand, process and produce a better understanding of natural language. Convolutional neural networks (CNN) is a deep, feed-forward artificial neural networks used in computer vision. It has been successful in semantic parsing , search query retrieval sentence modeling and various other NLP tasks. Recurrent Neural Network (RNN), unlike feed-forward networks, make use of their internal state to perform a sequence of operation on the input sequence which enables it to perform tasks such as unsegmented, connected handwriting recognition or speech recognition. Analogous to the human brain, being super efficient in understanding texts which is achieved through years of training, RNN is formulated upon a similar concept. Text classification forms a base for many applications, such as sentiment analysis,web searching and information filtering. The RNN inspects a text word by word and making a note of the semantics of the previously entered text in a fixed-sized hidden layer. It has the capability to capture provisional information but, RNN being a biased-model, later words tend to weigh more than earlier words and thereby, reducing the efficiency if used to analyze the semantics of a whole document where key words can appear anywhere than in the end. To overcome the problem of RNN, Convolution Neural Network, an unbiased network is used to tackle an NLP task which uses a max-pooling layer to determine different phrases in a text. But in a simple CNN, kernels such as fixed windows are used which pose a difficulty to predicting the window sizes, selecting a large window would make it difficult to train the neural network and using a small size might result in a loss of important information. Both CNN and RNN have a time complexity of O(n).To address the confinement of the above models, a Recurrent Convolutional Neural Network (RCNN) and apply it to text classification. A bidirectional recurrent network is used as it introduces less noise compared to CNN to capture contextual information. The model can hold a bigger scope of the word requesting when learning portrayals of writings. Max-pooling layer automatically decides features which an important role in text

**Corresponding author: Manav Mehra*

classification and hence, capturing major features in the text. As this model combines both, recurrent structure and max-pooling layer, it deploys the advantages of both recurrent neural and convolution neural models. It also exhibits a time complexity of O(n) which depends on length of the text. Touch-screen mobile text interfaces are omnipresent and so is text error completion and correction which cannot be tackled using the traditional language models without the assistance of deep repetitive error patterns in training data. Error patterns depend on the user's typing style and native language, and the various aspects of the interface such as the design, text entry method and vocabulary restriction. Recurrent neural network (RNN) have been successful in a wide variety of sequence problem and efficient in modelling the current natural language and hence, recurrent neural networks, long short-term memory networks and gated recurrent neural networks form the backbone for language modelling and machine translation and more.

2. Understanding and Classification

We discuss two different approaches to this process the first one uses the convolutional neural network and the other uses recurrent convolutional neural network.

2.1 Convolutional Neural Network Design.

This section gives an introduction for the design of Convolutional Networks that is used in the process of understanding and classification. The modular design obtains its gradient by back propagation and performs optimisation.

Key Modules Temporal Convolution module, being the main component, computes one dimensional convolution.A discrete input function, assumed to be, g(x) ∈ [1, l] → R and a discrete kernel function, assumed to be, f(x) ∈ [1, k] → R. The convolution h(y) ∈ [1, b(l − k)/dc + 1] → R between f(x) and g(x) with stride d is defined as,

$$h(y) = \sum_{x=1}^{k} f(x) \cdot g(y \cdot d - x + c),$$

where c = k − d + 1 is an offset constant. Just as in traditional convolutional networks in vision, the module is parameterized by a set of such kernel functions fij (x) (i = 1, 2, . . . , m and j = 1, 2, . . . , n) which we call weights, on a set of inputs gi(x) and outputs hj (y). We call each gi (or hj) input (or output) features, and m (or n) input (or output) feature size. The outputs hj (y) is obtained by a sum over i of the convolutions between gi(x) and fij (x). One key module that helped us to train deeper models is temporal max-pooling. It is the 1-D version of the max-pooling module used in computer vision. Given a discrete input function g(x) ∈[1, l] → R, the max-pooling function h(y) ∈ [1, b(l − k)/dc + 1] → R of g(x) is defined as

$$h(y) = \max_{x=1}^{k} g(y \cdot d - x + c),$$

where c = k − d + 1 is an offset constant. This very pooling module enabled us to train ConvNets deeper than 6 layers, where all others fail. The analysis by might shed some light on this. The non-linearity used in our model is the rectifier or thresholding function h(x) = max{0, x}, which makes our convolutional layers similar to rectified linear units (ReLUs). The algorithm used is stochastic gradient descent (SGD) with a mini batch of size 128, using momentum 0.9 and initial step size 0.01 which is halved every 3 epoches for 10 times. Each epoch takes a fixed number of random training samples uniformly sampled across classes. This number will later be detailed for each dataset separately. The implementation is done using Torch 7.

Character Quantization - Our models acknowledge a succession of encoded characters as information. The encoding is finished by endorsing a letters in order of size m for the information dialect, and after that quantize each character utilizing 1-of-m encoding (or on the other hand "one-hot" encoding). At that point, the succession of characters is changed to an arrangement of such m estimated vectors with settled length l0. Any character surpassing length l0 is overlooked, and any characters that are not in the letter set including clear characters are quantized as each of the zero vectors. The character quantization arrange is in reverse so the most recent perusing on characters is constantly set close to the start of the yield, making it simple for completely associated layers to connect weights with the most recent perusing. The letters in order utilized as a part of the greater part of our models comprises of 70 characters, including 26 English letters, 10 digits, 33 different characters and the new line character.
The non-space characters are:

abcdefghijklmnopqrstuvwxyz0123456789 -
,;.!?:’’’/\|_@#$%ˆ&*˜‘+-=<>()[]{}

Model Design - For our requirements we design two convolutional networks having difference in their size, large and small, and both of them have the same configuration that consists of 9 layers deep having 6 convolutional layers along with 3 fully connected layers as shown,

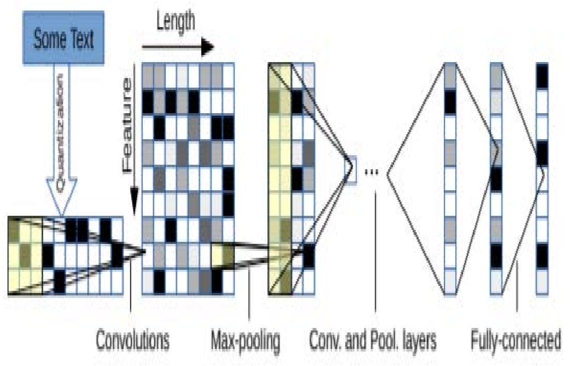

Figure 1:

The input has number of highlights equivalent to 70 because of our character quantization strategy, and the information full length is 1014. It appears that 1014 characters could as of now catch a large portion of the writings of intrigue. We additionally embed 2 dropout modules in the middle of the 3 completely associated layers to regularize. They have dropout likelihood of 0.5. Table 1 records the designs for convolutional layers, and table 2 records the setups for completely associated (direct) layers.

Table 1:

Layer	Large Feature	Small Feature	Kernel	Pool
1	1024	256	7	3
2	1024	256	7	3
3	1024	256	3	N/A
4	1024	256	3	N/A
5	1024	256	3	N/A
6	1024	256	3	3

Table 2

Layer	Output Units Large	Output Units Small
7	2048	1024
8	2048	1024
9	Depends on the problem	

For various issues the information lengths might be extraordinary (for instance for our situation l0 = 1014), as are the casing lengths. From our model outline, it is anything but difficult to realize that given information length l0, the yield outline length after the last convolutional layer (yet before any of the completely associated layers) is l6 = (l0 − 96)/27. This number duplicated with the casing size at layer 6 will give the information measurement the principal completely associated layer acknowledges.

2.2 Recurrent Convolutional Neural Network Design.

We propose a deep neural model to catch the semantics of the content of text. The following figure shows the network structure of the model. The contribution of the system is an archive D, which is an arrangement of words t1, t2 . . . the output of the network contains class elements. We use p(k|D, θ) to denote the probability of the document being class k, where θ is the parameters in the network.

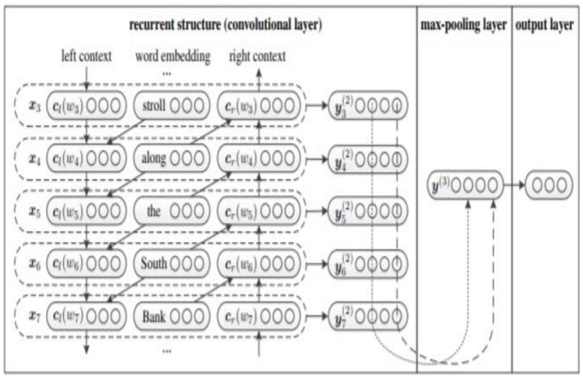

Figure 2: The structure of the recurrent neural network

Word Representation Learning - The combination of word and its context gives a word which helps in giving a clear meaning of the word. The bidirectional nature of the recurrent neural network captures the meaning of the word.

We characterize $P_l(t_i)$ as the left setting of word t_i and $P_r(t_i)$ as the right setting of word t_i. $P_l(t_i)$ and $P_r(t_i)$ are dense vectors with |t| as real value elements. The left-side context $P_l(t_i)$ of word t_i is found using equation (a), where s(ti−1) is the word embedding of word (ti−1), which is also a dense vector with |s| as real value elements. Pl(ti−1) is the left-side setting of the previous word wi−1. The left-side setting for the first word in any archive uses the same shared parameters Pl(t1). M(l) is a matrix that transforms the hidden layer (context) into the next hidden layer. $M(s_l)$ is a matrix that is used to combine the semantic of the current word with the next word's left context. F is a non-linear activation function. The right-side context $P_r(t_i)$ is calculated in a similar manner using equation (b). The right-side contexts of the last word in a document share the parameters $P_r(t_n)$.

$P_l(t_i) = F(M(l)P_l(t_i-1) + M(sl)e(t_i-1))$ ---(a)

$P_r(t_i) = F(M(r)P_r(t_i+1) + M(sr)e(t_i+1))$ ---(b)

As appeared in equations (a) and (b), the setting vector catches the semantics of all left-and right-side settings.

For instance, in Figure 1,$P_l(t_7)$ encodes the semantics of the left-side setting "walk around the South" alongside every single past content in the sentence, and $P_r(t_7)$ encodes the semantics of the right-side setting "bears a . . . ". At that point, we characterize the portrayal of word t_7 in equation (c), which is the link of the left-side setting vector $P_l(t_i)$, the word implanting $s(t_i)$ and the right-side setting vector $P_r(t_i)$.In this way, utilizing this logical data, our model might be better ready to disambiguate the importance of the word t_i in contrast with traditional neural models that lone utilize a settled window (i.e., they just utilize fractional data about writings).

$X_i = [P_l(t_i); s(t_i); P_r(t_i)]$--- (c)

The intermittent structure can acquire all P_l in a forward sweep of the content and P_r in a regressive output of the content. The time complexity is O(n). After we obtain the representation X_i of the word t_i, we apply a straight change together with the tanh actuation capacity to X_i and send the outcome to the next layer.

$Z_i^{(2)} = \tanh(X_i M^{(2)} + B^{(2)})$ --- (d)

$Z_i^{(2)}$ is a latent semantic vector, in which each semantic factor will be investigated to decide the most helpful factor for representing the content.

Text Representation Learning: The convolutional neural network represents text. From the perspective of convolutional neural networks, the recurrent structure we previously mentioned is the convolutional layer.

When all of the representations of words are calculated, we apply a max-pooling layer.

$$y^{(3)} = \max_{i=1}^{n} y_i^{(2)}$$

The max function is an element-wise function. The k-th element of $\square^{(3)}$ is the maximum in the k-th elements of $\square_{\square}^{(2)}$.

The pooling layer changes over writings with different lengths into a settled length vector. With the pooling layer, the data is caught in the whole content. There are different kinds of pooling layers, for example, average pooling layers . Average pooling isn't made to utilize on the grounds that lone a couple of words and their mix are helpful for catching the importance of any file.The max-pooling layer endeavors to locate the most essential dormant semantic factors in the archive. The pooling layer uses the yield of the repetitive structure as the information. The time complexity of the pooling layer is O(n).The general model is a course of the recurrent structure and a max-pooling layer, accordingly, the time many-sided quality of the model is still O(n).

The last piece of the model is a yield layer. Like conventional neural systems, it is characterized as

$\square^{(4)} = \square^{(4)} \square^{(3)} + \square^{(4)}$

Finally, the softmax function is applied to $\square^{(4)}$. It can convert the output numbers into probabilities.

$$p_i = \frac{\exp\left(y_i^{(4)}\right)}{\sum_{k=1}^{n} \exp\left(y_k^{(4)}\right)}$$

3. Correction and Completion

3.1 Sequence-to-Sequence model

Sequence-to-sequence models have demonstrated huge potential in enhancing vanilla repetitive system for input-yield progression mapping endeavors, machine translation. An info side encoder catches the portrayals in the information, while the decoder gets the portrayal from the encoder alongside the information and yields a relating mapping to the objective dialect. This structural set-up appears to normally fit the administration of mapping noisy contribution to denoised output, where the correction expectation can be dealt with as an alternate dialect and the task. The drawback of this mapping architecture is the fixed input and output sizes and the basic sequence-to-sequence model cannot create (test) new content or anticipate the following word. It is an essential component for keyboard decoder to precisely anticipate intended completion. The correction and completion can be achieved by having a separate corrector network as an encoder and an implicit dialect model as a decoder in a sequence-to-sequence architecture that trains end-to-end. A sequence-to-sequence text Correction and Completion Encoder Decoder Attention network (CCEAD) is outlined in the architectural diagram in Figure 1. This model trains a combination of dataset to capture the noise to correct patterns along with learning the implicit dialect model. The main points about the model are:

1. A sequence to sequence architecture deployed with a convolution neural network(CNN) - gated recurrent unit (GRU) encoder based on character is used to catch error representation in noisy text.
2. The decoder in the model is a gated recurrent unit in light of words, it gets the starting state

from the encoder and behaves like a dialect model.

3. The model learns conversational content adjustment and culminations for use in any such content based framework and is freethinker to the input.

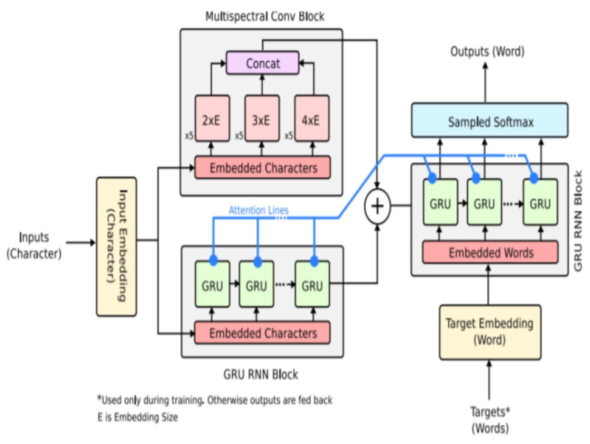

Fig. 1: Architectural diagram of our character based convolutional gated recurrent encoder with word based gated recurrent decoder with attention (CCED).

Figure 3:

3.2 Model Description

The general design of our model is outlined in Fig 1. Our model has the comparative basic design of the sequence-to-sequence models utilized as a part of machine interpretation. Sequence learning issues are trying as Deep Neural Networks (DNN) require the dimensionality of the information sources and focuses to be settled. Further, issues emerge if the data sources are character linked representations while the results are word level representations. Hence, in our model the encoder is made out of both recurrent and feed-forward units and works at a character level though the decoder works at a word level. Our model is made on two primary modules: Error Corrector Encoding Layer and Language Decoding Layer.

Character Error Correcting Encoding Layer -

- *Character Error Context Understanding*

Recurrent Neural Networks, are extremely efficient in capturing contextual patterns. For a sequence of inputs (x1, ..., xT), a classical RNN computes a sequence of outputs (y1, ..., yT) iteratively using the following equation:

$$\mathbf{h}_t = \sigma(\mathbf{W}^{hx}\mathbf{x}_t + \mathbf{W}^{hh}\mathbf{h}_{t-1} + \mathbf{b}^h)$$
$$\mathbf{y}_t = \mathbf{W}^{yh}\mathbf{h}_t.$$

The power of RNNs lie in the ease with which it can map input sequence to target sequence, when the alignment between sequence is known ahead of time. In case of unaligned sequences, to circumvent the problem, the input and target sequences are padded to fixed length vectors, then one RNN is used as the encoder to map the padded input vector to a different fixed sized target vector using another RNN. However, RNNs struggle to cope with long term dependency in the data due to vanishing gradient problem. This problem is solved using Long Short Term Memory (LSTM) recurrent neural networks. However, for purposes of error correction, medium to short term dependencies are more useful. Therefore, our candidate for contextual error correction encoding layer is the Gated Recurrent Network (GRU) which has similar performance to that of LSTM. We evaluated a vanilla sequence to sequence model that we trained on exhaustive set of synthetic noisy to true mappings, without any context. The models could not generalize to all types of errors especially insertion and deletion. The example in the following table,

Table 3:

Erroneous	Truth
Say yello to him	Say hello to him
Don't yello at me	Don't yell at me
I prefer yello	I prefer yellow

Highlights the importance of context for each of the sentences that will require different corrections. Gated Recurrent Unit (GRU) are the fundamental units of our model. In GRU, the input and the hidden states are of the same size that enables better generalization. The main motivation in using GRU over Long Short Term Memory networks (LSTM) as our fundamental unit is the size equality between input and state which is not the case with LSTM. Also, GRU and LSTM have exhibited similar performance across many tasks. If the input vector is x representing a character, and the current state vector is s, then the output is given as in Kaiser et. al.:

$$\mathbf{h}_t = \mathbf{u}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{u}_t) \odot \tanh\left(\mathbf{W}^{hx}\mathbf{x}_t + \mathbf{U}^{hh}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}^h\right),$$

where,

$$\mathbf{u}_t = \sigma(\mathbf{W}^{ux}\mathbf{x}_t + \mathbf{U}^{uu}\mathbf{h}_{t-1} + \mathbf{b}^u)$$

and,

$$\mathbf{r}_t = \sigma(\mathbf{W}^{rx}\mathbf{x}_t + \mathbf{U}^{rr}\mathbf{h}_{t-1} + \mathbf{b}^r).$$

In the GRU equations above, W* s and the U * s are weight matrices with respect to the gates and hidden state while the b * s are the bias vectors; where both of these type of parameters are learnt by the model. u and r are the gates as their elements have values between [0, 1] - u is the update gate whereas r is the reset gate. At every time step in the recurrent neural network, a GRU unit passes the result as the new state to the next GRU and to the output of the current time step.

- *Character Error Representation*

The convolutional layers are good at capturing the representations for insertion and deletion spelling errors. Both the GRU and the CNN are character based. For a sequence of, the input to the GRU and the CNN is simultaneous and in the form of a concatenated and padded fixed sized vector corresponding to the sequence length. CNN architecture applied to natural language processing typically model temporal rather than spatial convolutions. The characters considered in our model consists of 68 characters, including 26 English letters, padding symbol for aligning the sequences across batches of data input to our model, beginning of sequence marker and end of sequence marker among other special characters as listed below:

{0: ’\t’, 1: ’\n’, 2: ’\r’, 3: ’ ’, 4: ’!’, 5: ’"’, 6: ’#’, 7: ’\$’, 8: ’\%’, 9: ’\&’, 10: "’", 11: ’(’, 12: ’)’, 13: ’*’, 14: ’+’, 15: ’,’, 16: ’.’, 17: ’/’, 18: ’0’, 19: ’1’, 20: ’2’, 21: ’3’, 22: ’4’, 23: ’5’, 24: ’6’, 25: ’7’, 26: ’8’, 27: ’9’, 28: ’:’, 29: ’;’, 30: ’=’, 31: ’>’, 32: ’?’, 33: ’@’, 34: ’[’, 35: ’]’, 36: ’_’, 37: ’‘’, 38: ’a’, 39: ’b’, 40: ’c’, 41: ’d’, 42: ’e’, 43: ’f’, 44: ’g’, 45: ’h’, 46: ’i’, 47: ’j’, 48: ’k’, 49: ’l’, 50: ’m’, 51: ’n’, 52: ’o’, 53: ’p’, 54: ’q’, 55: ’r’, 56: ’s’, 57: ’t’, 58: ’u’, 59: ’v’, 60: ’w’, 61: ’x’, 62: ’y’, 63: ’z’, 64: ’{’, 65: ’|’, 66: ’}’, 67: ’’, 68: ’’}

Suppose the length of the character sequence is given by l for the kth word wk, Vc is the vocabulary of characters and Ec is the character embedding matrix of embedding dimension d such that Ec ∈ R |Vc|×d . If word wk is made up of characters [c1, c2, . . . , cl], then the character level representation for word wk is V k c ∈ R l×d . Central to temporal convolutions is 1D convolution. For input function g(x) ∈ [1, l] → R and kernel function f(x) ∈ [1, k] → R, the convolution is h(y) ∈ [1, b(l − k + 1)/dc] → R, for stride d is defined as:

$$h(y) = \sum_{x=1}^{k} f(x) \cdot g(y.d - x + c),$$

where c is the offset constant given by c = k − d + 1, and k is the width of the filter. There will be multiple filters of particular widths to produce the feature map, which is then concatenated and flattened for further processing. Here, sequence length signifies how many characters define the context for the error for the GRU and is about 5 characters or time steps. This sequence size will get adjusted to fixed sized length with padding to accommodate the shorter sequences. The number of neurons or cell size in the GRU is set to 256. The CNN consists of 5 filters with sizes varying in the range of [2, 3, 4]. The batch size indicating the number of instances to train on per batch is fixed to 100. The character vocabulary size is 30 including the character used for padding, newline, space, and start of sequence. The number of layers in the GRU is fixed to 1.

Word Level Decoder -

The decoder is also a GRU recurrent network that does word based processing. The output from the encoder is a linear transformation between the final hidden state of the char based GRU in the encoder and the output of the fully connected layer of the CNN. This state is then used to initialize the state of the decoder. The decoder sees as input a padded fixed length vector of integer indexes corresponding to the word in the vocabulary. The input sequence is a sequence of words prefixed with the start token. The sequence length is set to 5 which will then be padded for shorter sequences to generate fixed sized sequence. The vocabulary size for word based decoder is only about 3000 words. The encoder-decoder is trained end-to-end having a combined char-word based representation on the encoder and decoder side respectively. The size of the GRU cell is 256 with one layer.

Context based Attention

The decoder constructs the context by attending to the encoder states according to attention mechanisms . If the context is given by ci , decoder state, previous encoder states by hi by si−1, then for the sequence i, . . . , T:

$$\mathbf{c}_i = \sum_{j=1}^{T} \boldsymbol{\alpha}_{ij} \mathbf{h}_j$$

$$\boldsymbol{\alpha}_{ij} = \frac{\exp(\mathbf{d}_{ij})}{\sum_{k=1}^{T} \exp(\mathbf{d}_{ik})}$$

$$d_{ij} = \tanh(\mathbf{W}_s \mathbf{s}_{i-1} + \mathbf{W}_h \mathbf{h}_j + \mathbf{b})$$

Let $\square_{\square}$ be the fixed size vocabulary of words. The decoder in our model behaves like an implicit language model by specifying a conditional distribution over $\square_{\square+1}$ consistent with the sequence seen so far $\square_{1:\square}$ = [$\square_1$, . . . , $\square_{\square}$]. The GRU recurrent unit achieves this by the application of the affine transformation of the hidden state followed by projection onto the word vocabulary by performing a softmax:

$$Pr(w_{t+1} = j | w_{1:t}) = \frac{\exp(\mathbf{h}_t \cdot \mathbf{P}_j + b_j}{\sum_{i \in \mathcal{V}_w} \exp(\mathbf{h}_t \cdot \mathbf{P}_i + b_i)},$$

where □□ is the j-th column of □□□×|□|□ , known as the output embedding and b is the bias. Similar to our encoder side character embedding, our decoder takes word embeddings as inputs, for word □□ at time t, □□ = k, the input to the decoder is the k column of the word embedding matrix □□∈□|□|□× □. During optimization the loss is evaluated on the output of the decoder at the word level. The loss function is the cross-entropy loss per time step summed over the output sequence y:

$$L(x,y) = -\sum_{t=1}^{T} \log P(y_t|x, y_{<t})$$

5. Conclusion

In this paper, we review the various different techniques that can be used for the purpose of text classification, completion and correction. In the model we propose to use a recurrent convolutional network for text classification, in which our model captures contextual information with recurrent structure and constructs the representation of text. For completion and correction we suggest a combination of recurrent and convolutional neural sequence-to-sequence model. The models used traditionally require a large number of words, almost billions, for training and also take a huge amount of time to train, whereas, our model requires only 50000 words and can be trained within a day. The combination of the above techniques can help deliver a predictive keyboard technology that is efficient and effective.

References

1. Bengio, R. Ducharme, P. Vincent. 2003. Neural Probabilistic Language Model. *Journal of Machine Learning Research.*
2. Dong, F. Wei, S. Liu, M. Zhou, K. Xu. 2014. A Statistical Parsing Framework for Sentiment Classification. *CoRR, abs/1401.6330.*
3. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR, abs/1207.0580.*
4. Kalchbrenner, E. Grefenstette, P. Blunsom. 2014. A Convolutional Neural Network for Modelling Sentences. In *Proceedings of ACL 2014.*
5. Shen, X. He, J. Gao, L. Deng, G. Mesnil. 2014. Learning Semantic Representations Using Convolutional Neural Networks for Web Search. In *Proceedings of WWW 2014.*
6. Yoon Kim, *Convolutional Neural Networks for Sentence Classification.*
7. Boureau, Y-L, Bach, Francis, LeCun, Yann, and Ponce, Jean. Learning mid-level features for recognition. In *Computer Vision and Pattern Recognition (CVPR)*, 2010 *IEEE Conference* on, pp. 2559–2566. IEEE, 2010a.
8. dos Santos, Cicero and Gatti, Maira. Deep convolutional neural networks for sentiment analysis of short texts. In Proceedings of COLING 2014, *the 25th International Conference on Computational Linguistics: Technical Papers*, pp. 69–78, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics.
9. Frome, Andrea, Corrado, Greg S, Shlens, Jon, Bengio, Samy, Dean, Jeff, Mikolov, Tomas, et al. Devise: A deep visual-semantic embedding model. In *Advances in Neural Information Processing Systems*, pp. 2121–2129, 2013.
10. Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In NIPS, pp. 1106–1114, 2012.
11. Xiang Zhang,Yann LeCun, *Text Understanding From Scratch.*
12. Aggarwal, C. C., and Zhai, C. 2012. A survey of text classification algorithms. In *Mining text data*. Springer. 163–222.
13. Baroni, M.; Dinu, G.; and Kruszewski, G. 2014. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In ACL, 238–247.
14. Bengio, Y.; Ducharme, R.; Vincent, P.; and Jauvin, C. 2003. A *Neural Probabilistic Language Model*. JMLR 3:1137–1155.
15. Siwei Lai, Liheng Xu, Kang Liu, Jun Zhao, Recurrent Convolutional Neural Networks for Text Classification. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence.*
16. K. Vertanen, H. Memmi, J. Emge, S. Reyal, and P. O. Kristensson, "Velocitap: Investigating fast mobile text entry using sentence-based decoding of touchscreen keyboard input," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '15. New York, NY, USA: ACM, 2015, pp. 659–668. [Online]. Available: http://doi.acm.org/10.1145/2702123.2702135.
17. P. Rodriguez, J. Wiles, and J. L. Elman, "A recurrent neural network that learns to count," *Connection Science*, vol. 11, no. 1, pp. 5–40, 1999.
18. S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
19. I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural

networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

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

21. Shaona Ghosh, Per Ola Kristensson, Neural Networks for Text Correction and Completion in Keyboard Decoding. In *Journal of Latex Class Files*, VOL. 14, No. 8, August 2015