Length Adaptive Recurrent Model for Text Classification

Zhengjie Huang
Department of Computer Science
Sun Yat-sen University
Guangzhou, China
hzhengj@mail2.sysu.edu.cn

Zi Ye
Department of Computer Science
Sun Yat-sen University
Guangzhou, China
yezi7@mail2.sysu.edu.cn

Shuangyin Li
iPIN
Shenzhen, China
shuangyinli@ipin.com

Rong Pan*
Department of Computer Science
Sun Yat-sen University
Guangzhou, China
panr@sysu.edu.cn

ABSTRACT

In recent years, recurrent neural networks have been widely used for various text classification tasks. However, most of the recurrent architectures will not assign a class label to a text until they read the last word, while human beings are able to determine the text class before reading the whole text. In this paper, we propose a Length Adaptive Recurrent Model (LARM) which can automatically determine the minimum text length that is necessary to perform the classification. With three parts including *Reader*, *Predictor* and *Agent*, our model is designed to read a text word by word, and terminate the process when the adequate information has been caught for the text classification task. The experimental results show that our model has comparable or even better performance compared to the vanilla LSTM when both are fed with partial text input. Besides, we can speed up text classification by truncating the text when sufficient evidence is found for classification. Furthermore, we also visualize our model and show that our model works like human beings, who can gradually come up with the general idea of a text while reading texts sequentially.

CCS CONCEPTS

• **Computing methodologies** **Supervised learning by classification**; **Sequential decision making**; **Neural networks**;

KEYWORDS

Recurrent Neural Network, Text classification

*Corresponding author.

CIKM'17 , November 6–10, 2017, Singapore, Singapore

ACM ISBN 978-1-4503-4918-5/17/11…$15.00
https://doi.org/10.1145/3132847.3132947

1 INTRODUCTION

As one of the most studied topics in natural language processing, text classification is a task of assigning labels to documents, including sentiment analysis and topic assignment [2]. One of the most common approaches is to use handcraft features to represent the documents, such as unigrams or bigrams, and then feed the features to a linear classifier, which actually ignores the word order [22]. In the recent decades, Recurrent Neural Networks (RNN) and its variations, such as Long Short-Term Memory Networks (LSTM) [11] and Gated Recurrent Neural Networks (GRU) [4], have been widely applied to modeling sequential data [15]. For text classification tasks, RNN represents a document with distributed features by applying a recurrent function on words, which can not only capture the semantic of input texts but also take the word order into consideration.

It is worth noting that a major drawback of these methods, including RNN and its variations, is that they need to read all the words of a text for classification. Firstly, these models assume all the words of a text is processed together, regardless of the feasibility of getting all information, which means that these models feed the entire passage into a classifier with the document features, such as *bag-of-words*. Secondly, the complexity of these methods usually grows tragically with the increase of the length of the input texts, especially for RNN and its variations. Although recurrent models process a text sequentially, they can not adapt their reading length to truncate a long text input yet. Thirdly, the performance of RNN may drop on a long texts dataset [10]. The main reason is that RNN is a bias model with the tail words much more dominant than the head words, and it can not efficiently extract features from the head words of a passage, such as news whose important features are stored in the title.

To address the issues above, it may be wise to adapt the reading length to the dataset instead of processing the whole text. Imagining when human beings read a passage word by word, the main idea may gradually come to our minds and it is not necessary for us to process the whole passage. If the recurrent model can simulate this reading scheme and adapt the reading length automatically, it can not only reduce the complexity but also be beneficial to some classification tasks.

Therefore, we propose an innovative length adaptive recurrent model called LARM, which attempts to adapt the input text length

for text classification when reading word by word. The model can determine when it should stop reading and then output the final answer based on the processed text with dropping the remaining words. Our model consists of three parts, *Reader* that reads word by word and generates a representation of processed text, *Predictor* which is a task-specific classifier and *Agent* that determines when to stop reading and give corresponding commands to other components.

To summarize, the main contribution in this paper is that we introduce a novel length adaptive recurrent model for text classification tasks. Our model can read texts sequentially and adapt its input text length to the dataset. We report our empirical experimental results to show that our model can not only adapt its input text length to the data resources but also speed up by truncating input text. And with the same limited resources, our model outperforms the vanilla LSTM to some extent. Furthermore, our model is more robust, which can adapt its input text length to go through the useless or misleading information. We further analyze the cooperation mode in our model by visualizing and tracking the dynamic states of the texts being processed.

Our paper is organized as follows. In Section 2, we review the previous works about recurrent models relevant to our work. Then we describe the details of our model in Section 3. Training algorithm and inference process are presented in Section 4. Section 5 is mainly about the experiments and their results. Finally, Section 6 concludes.

2 RELATED WORK

As for the structure of our adaptive recurrent models, there were several relevant works in recent years. [16] presented a model that had a similar structure as ours. Although [16] aims to classify the images rather than texts, but its model can adaptively select the features as well as our model. Similar to our framework consisting of *Reader*, *Predictor* and *Agent*, there is a glimpse network used to extract features from partial images in [16], including a location network used to decide where to glimpse, and a core network for assigning labels. Furthermore, several other studies have followed these 3-step strategies. For example, [3] incorporated this strategy with speech recognition, [23] separated image caption into 3 steps including showing, attending and telling. All of these models learn the hard or soft attentions to focus on the specific positions of their inputs, which helps to improve their ability to track importance features.

In another way, the automatical determination of input length can be regarded as architecture adaptation. There are some studies related to this topic with the intuition that we can automatically adapt the network structure to the practical tasks. For instances, considering that the amount of computing time should be flexible based on the difficulty of tasks, [8] introduced Adaptive Computation Time algorithm that allowed recurrent neural networks to learn how many computational steps to take between receiving an input and emitting an output. The author applied this algorithm to training language model and got dramatical improvement. With the same idea, [5] modified traditional neural network structure into a Directed Acyclic Graph (DAG) like architecture, in which different decision paths were selected according to diverse inputs with each path defined as a different sequence of transformations. To a certain degree, our model can be regarded as a special case of a DAG architecture that every node in our model is connected to the next node or final output and all the out-degree except the last step are two. Furthermore, [20] proposed ReasoNet to undertake machine comprehension tasks. The ReasoNet could adapt their reading turns during comprehension process and generate answers when sufficient evidence was found.

There are some other closed ideas such as [6, 7, 25], which take text classification problem as a reading process. [6] attempted to tackle document classification task with a sequential reading approach, which looks similar to our work to some extent. However, the algorithm in [6] processed the document in a sentence-level basis rather than the word-level basis, and regarded the reading approach as a Markov Decision Process (MDP) learned by Reinforcement Learning. [7] extended the Bayes methods in sequential ways to solve text classification problem, and called this problem as early text classification task. With the same intuition that not all text data are useful, [25] proposed a model to skim words for text classification and question answering, which gained speed-up in these machine comprehension tasks.

3 MODEL

As illustrated in the right part of Figure 1, our model consists of three different parts:

- *Reader*, a long short-term memory network encoder, which is used to read words sequentially from text and generate representations for what it has read;
- *Predictor*, a linear classifier, which is in charge of assigning the labels for the input texts based on the final representations generated by *Reader*;
- *Agent*, constructed by a two-layer fully connected neural network, which is in charge of interacting with the above two parts, giving instructions to them, determining whether to keep reading or stop to predict.

The process of the proposed model is explained as follows. When it comes to a document, *Reader* reads the text word by word. Everytime the *Reader* receives a word, the *Agent* will decide whether the *Reader* should continue reading or not. If *Agent* thinks the words processed are not adequate for classification, which means that *Reader* ought to read more words, it will instruct *Reader* to read the next word. If *Agent* thinks the words processed are sufficient at some step, it will instruct *Reader* to stop reading. Then *Predictor* will analyze the representation encoded by *Reader* based on the words processed and try to assign the appropriate label to the document. It is worth noting that the representation of a document delivered to *Predictor* from *Reader* may be generated by just a part of the text. Because when *Agent* tells *Reader* to stop, *Reader* may not have reached the last word of the document. When *Reader* stops, those remaining words in the document which have not been processed will be discarded. The details of the architecture will be discussed in this section.

3.1 *Reader*: Long Short-Term Memory Network Encoder

Reader is designed to read words and learn representations for the current input texts. We take Long Short-Term Memory Networks

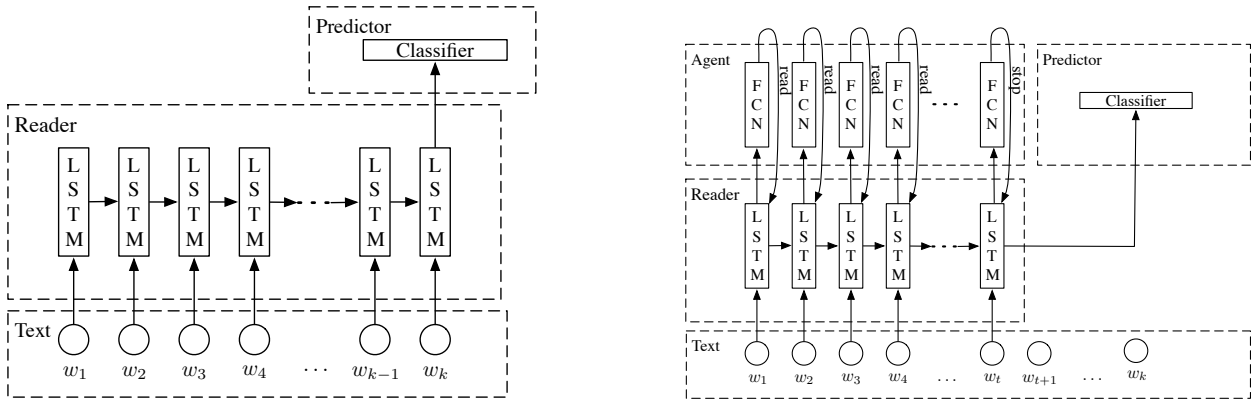

Figure 1: The left part is the vanilla LSTM for text classification, which force the *Reader* to read the whole text and then feed the last hidden state to a *Predictor*. And the right part is the model we use for text classification, which add an *Agent* that tracks the dynamic state of *Reader* and instructs it to read next word or stop reading.

(LSTM) as our *Reader* here. The main reason is that Recurrent Neural Networks (RNN) are beneficial to tracking the dynamic states of sequential data, and LSTM is an outstanding variation of RNN [11], which can deal with long dependencies of sequential data and suffer less gradient vanishing problems. Actually, we base our work on the following LSTM formulation. At each timestep t, LSTM takes input $\mathbf{x}_t$, $\mathbf{h}_{t-1}$, $\mathbf{c}_{t-1}$ and calculates $\mathbf{h}_t$, $\mathbf{c}_t$ via the following formulation:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}^{(i)}\mathbf{x}_t + \mathbf{U}^{(i)}\mathbf{h}_{t-1} + \mathbf{b}^{(i)}),\\
\mathbf{o}_t &= \sigma(\mathbf{W}^{(o)}\mathbf{x}_t + \mathbf{U}^{(o)}\mathbf{h}_{h-1} + \mathbf{b}^{(o)}),\\
\mathbf{f}_t &= \sigma(\mathbf{W}^{(f)}\mathbf{x}_t + \mathbf{U}^{(f)}\mathbf{h}_{t-1} + \mathbf{b}^{(f)})\\
\mathbf{u}_t &= \tanh(\mathbf{W}^{(u)}\mathbf{x}_t + \mathbf{U}^{(u)}\mathbf{h}_{t-1} + \mathbf{b}^{(u)}),\\
\mathbf{c}_t &= \mathbf{i}_t \odot \mathbf{u}_t + \mathbf{f}_t \odot \mathbf{c}_{t-1},\\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned} \tag{1}$$

where $\odot$ denotes element-wise multiplication, $\sigma(\cdot)$ and $\tanh(\cdot)$ are the element-wise sigmoid and hyperbolic tangent functions. And $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{o}_t$ are referred to as *input*, *forget* and *output* gates. In experiments, $\mathbf{h}_0$ and $\mathbf{c}_0$ are initialized as zero vectors. In most cases, we ignore the intermediate production $\mathbf{c}_t$ then we can rewrite the LSTM calculation in a recursive form:

$$\mathbf{h}_t = \begin{cases} f(\mathbf{x}_t, \mathbf{h}_{t-1}) & , t > 0 \\ \mathbf{0} & , t = 0 \end{cases}. \tag{2}$$

For our *Reader*, we take word sequence $[\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k]$ as input, where k is the length of the sequence and $\mathbf{w}_i$ denotes the word vector for the i-th word. At each step when *Reader* receives a read request from *Agent*, it will process the next word, and produce next hidden state by applying the LSTM recursive function on the former state and the current input:

$$\mathbf{h}_t = f_r(\mathbf{w}_t, \mathbf{h}_{t-1}). \tag{3}$$

Here we denote f_r as the LSTM structure for *Reader*. So we could get a hidden state sequence $[\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k]$.

3.2 *Predictor*: Linear Classifier

Predictor aims to classify the current input texts with the representations given by *Reader*. In this work, we simply apply a linear classifier for prediction. When it comes to t-th word and *Agent* judges that it is time to stop, our *Predictor* will take $\mathbf{h}_t$ from *Reader* as the input, and assign corresponding labels based on the distribution calculated as:

$$\hat{\mathbf{y}_t} = \text{softmax}(\mathbf{W}^{(p)}\mathbf{h}_t + \mathbf{b}^{(p)}). \tag{4}$$

In a vanilla LSTM like the left part in Figure 1, it is always the last hidden state $\mathbf{h}_k$ to be used for the downstream task. The main drawback is that the distance from inner hidden states to final output may cause gradient vanishing problems, which has aroused wide discussion. In recent years, several studies such as [24, 27] have used different techniques to tackle the gradient vanishing or exploding problem. They utilize attention schema or max-pooling to better aggregate information from all parts of the hidden states. And in our work, we propose a length adaptive structure based on the original form of LSTM, which can tackle the gradient vanishing or exploding problem as well, because our model can randomly connect the middle inner inputs to the final outputs.

Although our model is designed based on vanilla LSTM, it is easy to apply our idea on other novel models, such as some hybrid models of LSTM and Convolution Neural Networks (CNN), by rewriting their formula in the recursive form as shown above in Equation 3.

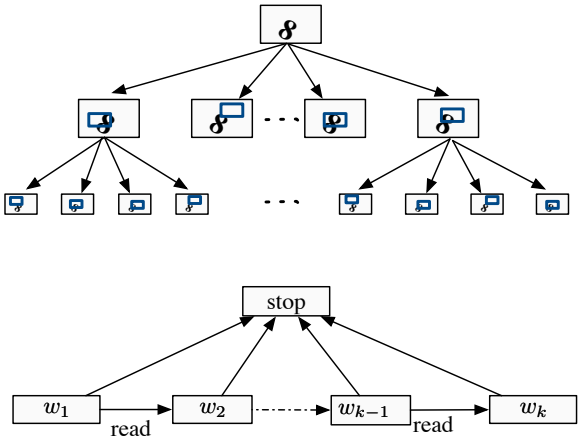

Figure 2: The first one shows the decision paths in [5] whose number of states grows exponentially. While the second one shows the sequential decision path in our model, and the number of states grows linearly with the number of words in a sequence.

3.3 *Agent*: Two-Layer Fully Connected Neural Network

Agent in our model plays an important role in providing guidance to both *Reader* and *Predictor*, determining whether to keep reading or stop processing the following words. At time step t, *Agent* performs an action a_t. And here we define $a_t = 1$ as to stop reading while $a_t = 0$ as to keep reading. Then the action policy is estimated as follows:

$$\begin{aligned} p_t &= p(a_t = 1|\mathbf{h}_t) \\ &= \sigma(\mathbf{W}_1^{(a)}\tanh(\mathbf{W}_2^{(a)}\mathbf{h}_t + \mathbf{b}_2^{(a)}) + \mathbf{b}_1^{(a)}), \end{aligned} \tag{5}$$

where p_t denotes the probability of stop and give prediction at step t, $\sigma(\cdot)$ and $\tanh(\cdot)$ are sigmoid and hyperbolic tangent functions. This part is actually a two-layer fully connected neural networks with sigmoid function applied on its last layer, so the output is in the range from 0 to 1 to indicate the probability. At each time step, *Agent* tracks the dynamic state of *Reader* and gives an instruction to *Reader* and *Predictor*, as illustrated in left part of Figure 1. Imagining if we remove *Agent*, *Reader* will therefore read the whole text and feed the last hidden state $\mathbf{h}_k$ to *Predictor*, as shown in the right part of Figure 1, it is exactly the vanilla RNN architecture for text classification.

4 MODEL TRAINING AND INFERENCE

In this section, we first discuss a differentiable training mechanism of our model in detail. Then we introduce a time penalty term, which helps to force our model to truncate input texts adaptively. Besides, we further illustrate the specific inference process in testing phases.

4.1 Differentiable Training Algorithm for Sequential Decision

The process of reading, judging and predicting in this paper can be considered as a Partially Observable Markov Decision Process (POMDP) problem [17]. Several studies have considered POMDP problems as non-differentiable ones. They usually applied policy-gradient based methods for training, sampling approximation to estimate the intractable expectation of reward [5, 20]. For example, as shown in the first part of Figure 2, in the model of [5] which has a similar model structure to our work, the model sensor glimpses a part of the image at each step and the location for each glimpse can be various, so the number of decision paths grows exponentially. However, as demonstrated in the second part of Figure 2, the sequential reading approach in our model can read next word or stop to terminate a decision path, so the number of decision paths is increasing linearly. Then we can calculate the conditional probability of labels as below.

Let $A_i = \{a_1 = 0, a_2 = 0, \ldots, a_i = 1\}$ be a decision path denoting that the model stops at the i-th step. If we have a text with k words, then there will be exactly k decision paths. Therefore, it is easy to enumerate the k decisions and calculate the expectation of stop position in linear time. The probability of $p(A_i|\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k)$ can be written as

$$p(A_i|\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k) = \prod_{j=1}^{i-1}(1 - p_j)p_i. \tag{6}$$

We can estimate the marginal log-likelihood of observing labels as follows and then optimize the log-likelihood with a gradient-based algorithm.

$$\begin{aligned} L &= \log p(y|\mathbf{w}_1, \ldots, \mathbf{w}_k) \\ &= \log \sum_{i=1}^{k} p(y|\mathbf{w}_1, \ldots, \mathbf{w}_i)p(A_i|\mathbf{w}_1, \ldots, \mathbf{w}_k). \end{aligned} \tag{7}$$

Since $p(A_i|\mathbf{w}_1, \ldots, \mathbf{w}_k)$ has a term $\prod_{j=1}^{i-1}(1 - p_j)$ whose value will decrease exponentially, the model may stuck after some steps without much more exploration. To prevent this issue, we randomly set p_t to 0 during training with probability ρ, which can force *Agent* to keep reading and explore more information. In all our following experiments, we set ρ to 0.9.

Moreover, recalling the attention mechanism in [24, 28] that take weighted average features for the classification task. In attention mechanism, we always have the weighted value summed up to 1. In our model, we also have $\sum_{i=1}^{k} p(A_i) = 1$, which can be considered as a special case of attention technique. In most attention-based models, the attentive weights can only be estimated over the whole sequence and the value may be more smooth. However, the attentive weights in our model can be estimated at each time step and the value can be much sharper. Furthermore, this kind of weighting strategies can be regarded as attaching the inner hidden state to the final output, which may suffer less gradient exploding or vanishing problem as mentioned before.

4.2 Time Penalty Regularization

In our work, we introduce a penaly term to make constrains on the *Agent* to force our model to give early prediction. Apparently,

Table 1: Classification Dataset Details

Dataset	Ave. Len	Max Len	#Classes	#Train:#Test
AG's News	34	211	4	120,000:7,600
⌞ Pre-Padding	97	266		
⌞ Post-Padding	97	275		
DBPedia	49	1,485	14	560,000:70,000
⌞ Pre-Padding	108	1,565		
⌞ Post-Padding	108	1,522		

since the probability of stop position can be written in a explicit formulation, we can utilize the expectation of consuming steps as an extra regularization. The regularization term and the total loss term are shown below

$$E_{step} = \sum_{i=1}^{k} ip(A_i), \tag{8}$$

$$L_s = L + \tau E_{step}. \tag{9}$$

There is a trade-off between expectation steps and predicting accuracy. Since keeping the number of classification steps lower will drop some of the information, the accuracy may be affected in some ways. However, if some useless information is skipped, then the accuracy can be maintained and the text length needed for classification can be shortened.

4.3 Inference in Testing Phase

In training phase, it is possible to obtain the whole sequence for training. However, in testing phase, considering the high-cost of processing the whole sequence, we can do stochastic inference to get the classification result with less resource. We describe the details of our inference algorithm in Algorithm 1. At each time step, we determine whether to terminate the reading process by sampling from the policy distribution given by *Agent*. Instead of getting full text data in testing phase, we can truncate the input sequence when adequate information is found, so less computation resource will be wasted.

Algorithm 1 Stochastic Inference in Testing Phase

Input: Initial state $\mathbf{h}_0$; Step $t = 1$; A word sequence $[w_1, w_2, \ldots, w_k]$;

Output: Termination Step T and Classification Result c;

1: Generate hidden state $\mathbf{h}_t = f_r(\mathbf{w}_t, \mathbf{h}_{t-1})$;
2: Generate random variable $a_t \sim Uniform(0, 1)$;
3: If $a_t > \sigma(\mathbf{W}_1^{(a)} \tanh(\mathbf{W}_2^{(a)} \mathbf{h}_t + \mathbf{b}_2^{(a)}) + \mathbf{b}_1^{(a)})$ goto step 4; otherwise goto Step 5;
4: If $t < k$ set $t = t + 1$ and goto Step 1; Otherwise goto Step 5;
5: Generate answer $c = \underset{i}{\text{argmax}}\, \text{softmax}_i(\mathbf{W}^{(p)} \mathbf{h}_t + \mathbf{b}^{(p)})$;
6: Return $T = t$ and c;

5 EXPERIMENTS

In this section, we will show the performance of our models by the following experiments. In the first part of experiments, we test our model on text classification tasks and show its ability of length adaptation. We also tune hyperparameter τ to control the strength of time penalty, which can further illustrate the trade-off among stopping position, speed-up performance and the prediction accuracy in our model. In the second part of experiments, to further explore adaptiveness characteristic and robustness of our model, we evaluate our model on noise-padding data without time penalty term. In this case, we are going to show how our model can automatically change its reading length to avoid the interference. Finally, in the last part of experiments, we visualize the dynamic inner states of our model and analyze how each part of the framework cooperates with others actually.

All the experiments are trained and tested on two large text classification datasets. The first one, **AG's news**[1], has categorized news articles from more than 2,000 news sources. We select the 4 largest classes constructed in [26] for our experiments, including World, Sports, Business and Sci/Tech. Each class contains 30,000 training samples and 1,900 testing samples. And the total number of training samples is 120,000 and testing 7,600. The second dataset is **DBPedia ontology dataset**, a crowd-sourced community effort to extract structured information from Wikipedia [14]. In our experiments, We pick out the dataset constructed in [26] consisting of 14 non-overlapping classes from DBPedia 2014. And each class is made of 40,000 training samples and 5,000 testing samples. Besides, we modify both two datasets to gain some new datasets, and the details will be shown below. The average and the max word counts for all datasets are reported in the Table 1. In the following experiments, since in test phase, we take a stochastic inference to generate the classification result. At each time, we randomly draw 100 classification results with stochastic inference in Algorithm1, then calculate the accuracy and the average number of steps for classification.

Train Details

For all the experiments, we split the training data with 80% for training and 20% for validation randomly. And we take pre-train word embeddings GLOVE[2] as the input for all our models [19], replacing the words with token <UNK>if they do not appear in the pre-train data. The dimension of word embedding is 300, then we set both the size of the hidden state in our LARM and LSTM to 500. During training, We perform early stopping strategy on expectation accuracy of validation data and then tune learning rate. Moreover, we use Adam to perform parameter optimization [13]. We also used gradient clipping in which the gradient norm is limited to 5 [18].

[1] http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html
[2] http://nlp.stanford.edu/projects/glove/

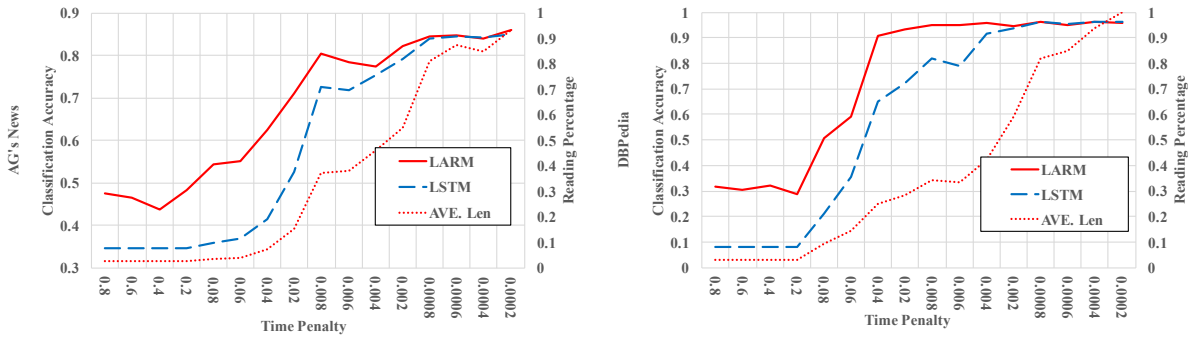

Figure 3: Trade-off between time penalty and classification accuracy

All our implements are based on Tensorflow[1] and run on one Titan X GPU.

5.1 Evaluation on Text Classification

In this part of experiments, we apply our model on text classification to show its ability of length adaptation. We force our model to perform text classification tasks by optimizing both the log-likelihood with the time penalty term. We test our model in both **AG's News** and **DBPedia** datasets. Then we tune the parameter τ to see the trade-off between decision steps and accuracy. We also compare our model with the vanilla LSTM. For comparison, we generate classification result with the vanilla LSTM fed with input texts truncated by our *Agent*. Figure 3 shows the empirical results and the trade-off between the length of the processed sequence and the classification accuracy. Although both models are trained using full-text data, our method can effectively make use of the incomplete information, compared to vanilla LSTM whose performance drops rapidly with fewer words.

Besides, compared to vanilla LSTM whose input is full-text, our model can speed up the classification progress in testing phase by truncating input text. In the experiments, we utilize the number of processed word to estimate their speed. For the same classification tasks, the model who has dealt with less number of words, will spend fewer recourses and have a higher speed. And we take the speed of vanilla LSTM as the baseline (speed 1x). The results of speed up with different settings of τ are shown in Table 2. With a large penalty, our model will gain a lot of speed-up but drop a lot on accuracy. However, with appropriate settings of τ like 0.002, we can get almost 2 times speed-up (half of the text are processed while the remaining half are truncated by our model) with small effects on accuracy. The results suggest that not all words of texts are needed for classification and the partial texts may be enough for achieving a comparable result. Definitely, it makes sense since human beings can determine the topic of an article or long paper with a few glances.

5.2 Length Adaptation Performance

In the second part of experiments, we test the model robustness on normal and modified datasets to show if our model can effectively adapt their reading length to the useless or misleading information.

Table 2: Speed up performance with different time penalty τ

	AG New's		DBPedia	
Time Penalty(τ)	**ACC.**	**Speed Up**	**ACC.**	**Speed Up**
0.0002	85.86%	1.07x	95.98%	1.00x
0.0004	83.96%	1.17x	96.29%	1.06x
0.0006	84.79%	1.14x	94.94%	1.18x
0.0008	84.43%	1.23x	94.94%	1.22x
0.002	82.16%	1.81x	94.59%	1.68x
0.004	77.31%	2.16x	94.67%	2.37x
0.006	78.34%	2.62x	95.04%	2.99x
0.008	80.52%	2.68x	95.07%	2.92x
0.02	71.17%	6.54x	93.16%	3.54x
0.04	62.35%	13.84x	90.87%	4.01x

Since the original text may not have strong interference, in order to enhance the misleading information, we modify the two original datasets by padding the useless tokens as noise. We randomly pad 20-100 tokens, denoted as <noise>, at the end of sequence for the post-padding data and at the beginning for the pre-padding data. Some examples are shown in Table 3. Since the time constraint may affect the performance of our model, in this experiment, we only use log-likelihood loss for training without the additional time penalty term. For comparison, we take vanilla LSTM as the baseline, where LSTM is used to encode a document and take the last hidden state to a classifier. This experiment shows if our adaptive length model can notice the existence of useless information and get rid of their interference.

Table 4 shows the classification accuracies of all the datasets. From this table, we can see that the vanilla LSTM suffers a lot from the post-padding interference and its performance drops rapidly to even making random guesses. However, our model can adapt itself to the post-padding noises. As for pre-padding datasets, both models keep their performances. To explain this phenomenon, we explore some relative records for the training process of our model and show them in Figure 4. We can find that the growth of reading length in post-padding data is much slower than the one in pre-padding data. It can be easily inferred that our model first attaches the front words to *Predictor*. So for the pre-padding data, it tries to read more words

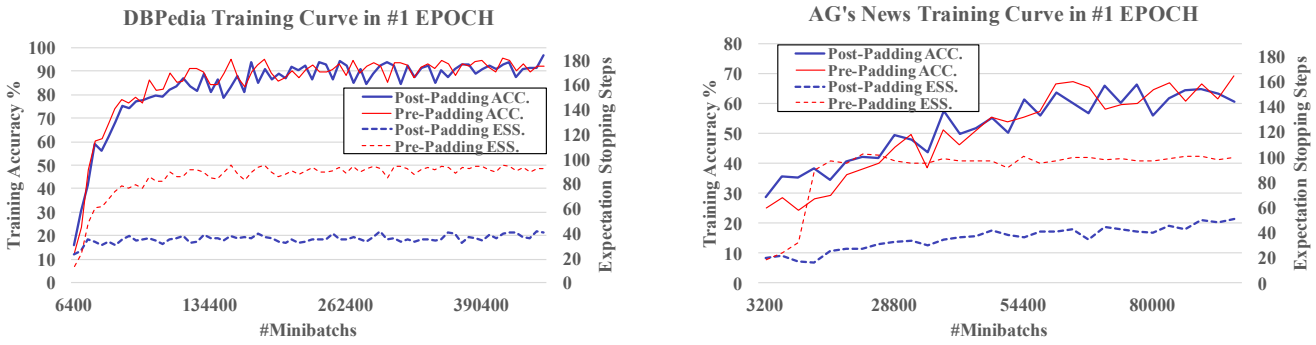

Figure 4: The curve about training accuracy (ACC.) and expectation stopping steps (ESS.) in the 1st epoch.

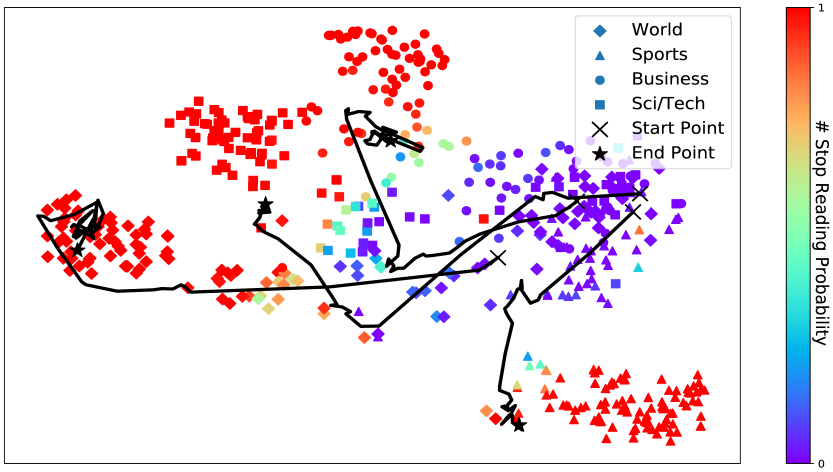

Figure 5: This figure shows a 2-dimensional t-sne projection of the LARM inner hidden states on AG's News dataset. Colors of the points indicate the stop reading probability given by the *Agent*; black lines are traces of inner states for given texts; and different markers are used for denoting the ground truth for classification.

to go through the noises since accuracy is unstable at first, but slow down the growth of the expectation stopping steps when it is well trained. While for the post-padding data, our model first builds a shortcut from front states to *Predictor* and the training accuracy grows steadily. However, when classifying the post-padding data, the vanilla LSTM is used to directly taking the last hidden state for prediction and it is hard to explore the long dependency between the front words and the final outputs with a gap caused by the post-padding noise. While it can just ignore pre-padding noises by simply feeding the *Predictor* with the last hidden state, so the vanilla LSTM keeps its performance with pre-padding noise data.

5.3 Visualization and Analysis of LARM

For further analysis, we are going to visualize our model and track the inner states to see how our model perform text classification with truncated inputs. Several other studies like [12, 21] attempt to visualize RNN to get a better understanding of how it works. [12] visualizes the gate activation in each LSTM cell to explore the long dependencies behavior in LSTM. [21] tries to track the dynamic states in LSTM to discover the special pattern in datasets containing nesting, phrase structure, and chord progressions. In this study, we are going to explore the manifold inside the hidden states of LSTM and show how it works for classification tasks. Besides, to our best knowledge, there is seldom work that explore the manifold of LSTM for text classification.

In this part of experiments, we take the model we have trained with **AG's News** in the Section 5.1 with $\tau = 0.01$ in comparison with the vanilla LSTM. To visualize the inner states of the model, we randomly sample 500 hidden states along with all time-steps from all sequences. We reduce the dimension of hidden states into 2 by t-sne [9].

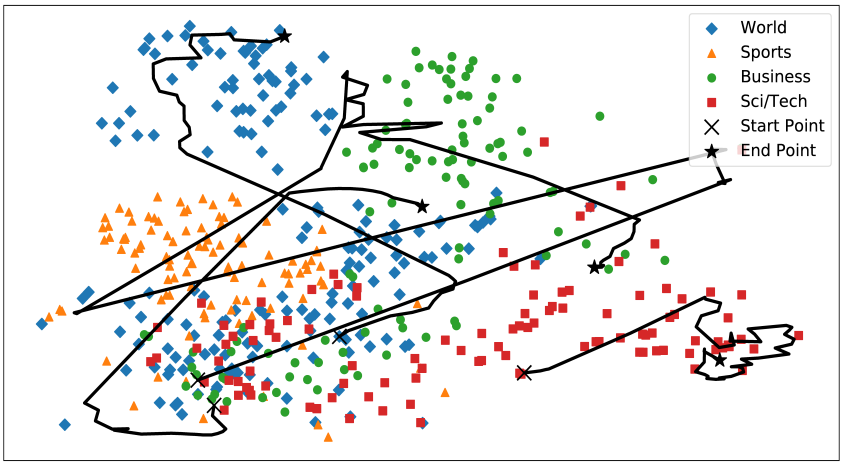

Figure 6: This figure shows a 2-dimensional t-sne projection of the vanilla LSTM inner hidden states on AG's News dataset. Black lines are traces of inner states for some certain texts; and different markers and colors are used for denoting the ground truth.

Table 3: Noise Padding Examples

Origin	Microsoft strips Longhorn of WinFS NEW YORK, September 1 (New Ratings) - Microsoft (MSFT.NAS) reportedly intends to ship Longhorn, its next version of Windows operating system, without WinFS, the next-generation file system.
Pre-Padding	<noise><noise><noise><noise>Microsoft strips Longhorn of WinFS NEW YORK, September 1 (New Ratings) - Microsoft (MSFT.NAS) reportedly intends to ship Longhorn, its next version of Windows operating system, without WinFS, the next-generation file system.
Post-Padding	Microsoft strips Longhorn of WinFS NEW YORK, September 1 (New Ratings) - Microsoft (MSFT.NAS) reportedly intends to ship Longhorn, its next version of Windows operating system, without WinFS, the next-generation file system. <noise><noise><noise><noise>

Table 4: Results on Text Classification

Dataset	LSTM ACC. %	LARM ACC. %
AG's NEWS	87.08	**87.18**
⌞ Pre-Padding	**86.81**	86.70
⌞ Post-Padding	25.00	**86.50**
DBPedia	97.87	**97.88**
⌞ Pre-Padding	**97.85**	97.84
⌞ Post-Padding	7.96	**98.05**

From Figure 5, we show the cooperation scheme among *Reader*, *Predictor* and *Agent*. For a certain text, whose inner states are represented by nodes along a black line in this figure, we can see that the corresponding line walks step by step from a position with low stop probability to another position with high conviction. This indicates that *Agent* will persuade *Reader* to read more words initially and stop it eventually. As for all the texts, at the beginning, their inner states are too similar, which is inferred by the nodes with low probability gathering at the same place. That means when the model starts to deal with a text, the states are so uncertain that it is hard for *Predictor* to identify the matching label for them, then *Agent* tends to encourage *Reader* to process more words. Gradually, the inner states of different texts look more and more different and become separated into several groups located in high stop probability regions. Because with the growth of information processed by **Reader**, the gap between the states of different texts get more and more apparent, which means the general idea of them become more and more clear and specific. Then it is much easier for *Predictor* to classify the texts, therefore *Agent* tends to stop the reading process at that time.

As for Figure 6, which illuminates the hidden states of vanilla LSTM, we can see that the traces of the nodes wander around randomly, where the black lines indicate the traces and the nodes denote the hidden states of texts, even though the endpoints finally land in the appropriate places. That means for a certain text, the hidden states are irregular and unstable until the end of the text. Therefore it is hard to avoid a big fall on performance if we truncate the texts in a vanilla LSTM. Compared to vanilla LSTM, our model is likely to quantify the effectiveness of middle outputs of LSTM and arrange the outputs in a more reasonable order rather than irregular order. It seems like human reading behavior in some way that the general idea of the text will gradually come to our minds when we are reading word by word.

6 CONCLUSION

In this paper, we propose a Length Adaptive Recurrent Model that can adapt its reading length to text classification task. We train our model with a differentiable method with time penalty in order to force our model to perform early prediction. The experiment shows our model is robust to useless or misleading information and its performance drop less than the traditional approach with limited resources. Besides, we can gain speed up by truncating the useless tokens in the text. We also visualize the inner states of our model and show the cooperative scheme between *Reader*, *Predictor* and *Agent*, which provide a more intuitive way to explore the dynamic state in LSTM.

Since the performance depends much on time penalty term, in the future, we are going to explore a multi-agent strategy that takes text classification as a quiz in which agents need to give answer faster and more accurate than others, then we may be able to drop the time penalty term. Besides, it is necessary to apply this idea to other state-of-the-art architectures like some other hybrid CNN or LSTM models. Furthermore, it will be interesting to build an analogous model which can determine when to start reading instead of stop reading.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China under Grant 2016YFB0201900, the Fundamental Research Funds for the Central Universities under Grant 17LGJC23.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).
[2] Charu C Aggarwal and ChengXiang Zhai. 2012. A survey of text classification algorithms. In *Mining text data*. Springer, 163–222.
[3] William Chan, Navdeep Jaitly, Quoc V Le, and Oriol Vinyals. 2015. Listen, attend and spell. *arXiv preprint arXiv:1508.01211* (2015).
[4] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).
[5] Ludovic Denoyer and Patrick Gallinari. 2014. Deep sequential neural network. *arXiv preprint arXiv:1410.0510* (2014).
[6] Gabriel Dulacarnold, Ludovic Denoyer, and Patrick Gallinari. 2011. Text classification: a sequential reading approach. (2011).
[7] Hugo Jair Escalante, Manuel Montes-y Gómez, Luis Villaseñor-Pineda, and Marcelo Luis Errecalde. 2015. Early text classification: a Naive solution. *arXiv preprint arXiv:1509.06053* (2015).
[8] Alex Graves. 2016. Adaptive Computation Time for Recurrent Neural Networks. (2016).
[9] G E Hinton. 2008. Visualizing High-Dimensional Data Using t-SNE. *Vigiliae Christianae* (2008).
[10] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. (2001).
[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[12] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2015. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078* (2015).
[13] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[14] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. 2015. DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web* 6, 2 (2015), 167–195.
[15] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model.. In *Interspeech*, Vol. 2. 3.
[16] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. 2014. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*. 2204–2212.
[17] George E Monahan. 1982. State of the artâĂŤa survey of partially observable Markov decision processes: theory, models, and algorithms. *Management Science* 28, 1 (1982), 1–16.
[18] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. *ICML (3)* 28 (2013), 1310–1318.
[19] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162
[20] Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2016. Reasonet: Learning to stop reading in machine comprehension. *arXiv preprint arXiv:1609.05284* (2016).
[21] Hendrik Strobelt, Sebastian Gehrmann, Bernd Huber, Hanspeter Pfister, and Alexander M Rush. 2016. Visual analysis of hidden state dynamics in recurrent neural networks. *arXiv preprint arXiv:1606.07461* (2016).
[22] Sida Wang and Christopher D Manning. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*. Association for Computational Linguistics, 90–94.
[23] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044* 2, 3 (2015), 5.
[24] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of NAACL-HLT*. 1480–1489.
[25] Adams Wei Yu, Hongrae Lee, and Quoc V Le. 2017. Learning to Skim Text. *arXiv preprint arXiv:1704.06877* (2017).
[26] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*. 649–657.
[27] Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. 2016. Text Classification Improved by Integrating Bidirectional LSTM with Two-dimensional Max Pooling. *arXiv preprint arXiv:1611.06639* (2016).
[28] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. 2016. Attention-based bidirectional long short-term memory networks for relation classification. In *The 54th Annual Meeting of the Association for Computational Linguistics*. 207.