Speed Reading: Learning to Read ForBackward via Shuttle

Tsu-Jui Fu
Academia Sinica
No. 128, Sec. 2, Academia Rd.
Taipei 11529, Taiwan
s103062110@m103.nthu.edu.tw

Wei-Yun Ma
Academia Sinica
No. 128, Sec. 2, Academia Rd.
Taipei 11529, Taiwan
ma@iis.sinica.edu.tw

Abstract

We present LSTM-Shuttle, which applies human speed reading techniques to natural language processing tasks for accurate and efficient comprehension. In contrast to previous work, LSTM-Shuttle not only reads shuttling forward but also goes back. Shuttling forward enables high efficiency, and going backward gives the model a chance to recover lost information, ensuring better prediction. We evaluate LSTM-Shuttle on sentiment analysis, news classification, and cloze on IMDB, Rotten Tomatoes, AG, and Children's Book Test datasets. We show that LSTM-Shuttle predicts both better and more quickly. To demonstrate how LSTM-Shuttle actually behaves, we also analyze the shuttling operation and present a case study.

1 Introduction

Recently, recurrent neural networks (RNNs) and long short-term memory (LSTM) cells and gate recurrent unit (GRU) cells have achieved great success and are increasingly being applied in nature language processing tasks, e.g., part-of-speech (POS) tagging (Wang et al., 2015), named-entity recognition (Chiu and Nichols, 2015), sentiment analysis (Zhang et al., 2018), document classification (Kim, 2014; Le and Mikolov, 2014a), cloze (Srinivasan et al., 2018), machine translation (Bahdanau et al., 2015), dialogue modeling (Mei et al., 2017), document summarization (Allahyari et al., 2017), automatic knowledge extraction (Durme and Schubert, 2008), and question answering (Chen et al., 2017).

Those tasks all call for text comprehension techniques. To solve these tasks, the proposed models read all the text available. That is, models read every token or word of the text from beginning to end. However, for some classification tasks, it is not necessary to treat each individual word equally. Take, for example, sentiment analysis: sentences such as "this movie is amazing" or "too boring" are sufficient to judge a sentiment without reading the entire comment. In addition, the fact that texts are often written redundantly also motivates reading selectively, especially for certain NLP tasks.

In terms of human reading habits, although people tend to skim text when reading a newspaper or a novel, this does not significantly impair comprehension. Speed reading, a reading technique, is used to improve one's ability to read quickly. Work has been done on modeling skimming behavior along with the original sequence modeling RNN. LSTM-Jump (Yu et al., 2017) predicts how many words to skim based on the RNN hidden state. They show that neglecting some words in a document does not greatly harm prediction accuracy but does significantly accelerate the process. They also show that for certain tasks such as cloze, skimming even outperforms traditional methods. In addition, (Yu et al., 2018) use RNN hidden states to decide when to stop. If the RNN judges it has achieved sufficient comprehension of the context, it stops early and produce the answer directly.

However, strictly speakly, simply skimming and stopping early is not speed reading. For example, imagine that during a reading test, we first read the question and then the long article. We read quickly and skip some information. What do we do if we encounter text that we don't understand? We go back, read the previous text, and try to fill in the gaps in our understanding. For speed reading, going back – or "reading backward" – is likewise important as it helps us to recover lost information or correct misunderstandings, leading to better comprehension of long documents. In fact, reading backward increases rather than decreases reading speed: given the opportunity to go back to correct

Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 4439–4448
Brussels, Belgium, October 31 - November 4, 2018.

misunderstandings, we skim more words and thus read faster without reducing our comprehension.

In this paper, we propose LSTM-Shuttle, which teaches the RNN model to speed read by modeling both forward and backward reading behavior. We evaluate the proposed method on sentiment analysis, document classification, and cloze tasks. We use IMDB (L. et al., 2011) and Rotten Tomatoes (Pang and Lee, 2005) as sentiment analysis datasets, AG New (Shang et al., 2015) as a document classification dataset, and Facebook Children's Book Test (Hill et al., 2015) as a cloze dataset. The experiments show that the proposed method achieves better prediction performance and reads faster at the same time, in comparison with the LSTM baseline (Hochreiter and Schmidhuber, 1997) and LSTM-Jump. We also analyze the shuttling behavior under different settings, proving that reading forward and backward does help in reading.

2 Related Work

The proposed method is inspired mainly by LSTM-Jump (Yu et al., 2017), which predicts how many words should be neglected, accelerating the reading process using RNN. Both their work and ours is related to the idea of learning visual attention per (Mnih et al., 2015), where a recurrent model is used to decide which image part to watch seriatim. They train the model end-to-end using the REINFORCE algorithm (Williams, 1992) and sample from a continuous Gaussian distribution. The difference between their and our methods is that we sample from a discrete distribution to reflect the properties of text and image.

Many recent natural language processing applications have explored the idea of filtering irrelevant content. As in our work, instead of skimming some words, (Seo et al., 2018) consider all words but use a small RNN for irrelevant words and the original large RNN for relevant ones. (Campos et al., 2018) also attempt to dynamically control the RNN's computational costs, but they instead control the number of units to update at each time step. In our method, in contrast, we skim words, directly setting the amount of computation to be zero, which streamlines the reading process.

From another perspective, (Yu et al., 2018) attempt to model early stopping behavior, deciding whether the model can answer confidently based on the hidden states. This is very effective for tasks such as question answering. To ensure accuracy, (Shen et al., 2016) focuses on early stopping after multiple passes, and (Shen et al., 2016) also using reinforcement learning to attempt to learn to reason. Both early stopping and our method adequately take into account research on sufficient comprehension. While, early stopping is not fast enough for classification, we can do better with text shuttling.

3 Main Idea

In this section, we describe the proposed LSTM-Shuttle, first presenting its architecture. Then, we show that due to the nondifferentiability of the shuttle mechanism, we apply a policy gradient (Sutton et al., 1999) to train it end-to-end. Finally, we present the implementation details and the inference approach.

3.1 Overview

As Fig. 1 illustrates, LSTM-Shuttle is based on an LSTM recurrent neural network to which is added an additional fully connected layer to predict forward or backward steps after a softmax distribution.

Given a text which denoted as $x_1, x_2, \ldots, x_L$ or $x_{1:L}$, LSTM-Shuttle first reads a fixed number of words sequentially and outputs the hidden state. Then, based on the hidden state, LSTM-Shuttle computes the shuttle softmax distribution over the forward or backward steps on $[-K, K]$. Given a negative step value, LSTM-Shuttle goes back to correct misunderstandings, and with a positive step value, LSTM-Shuttle speed reads, skimming unimportant words. After shuttling, LSTM-Shuttle reads words sequentially and then proceeds to shuttle again, iteratively. This process continues until one of the following occurs:

- The shuttle softmax samples a 0
- The number of shuttles exceeds the predefined upper limit
- The model has arrived at the last word

After stopping, the last hidden state is used to predict the desired task. The post-processing depends on the task. For instance, for classification, the hidden state generates a softmax distribution over the target class, and for cloze, it is used to find the correlation between the question article and each candidate answer. The detailed settings of each task are described in Section 4.

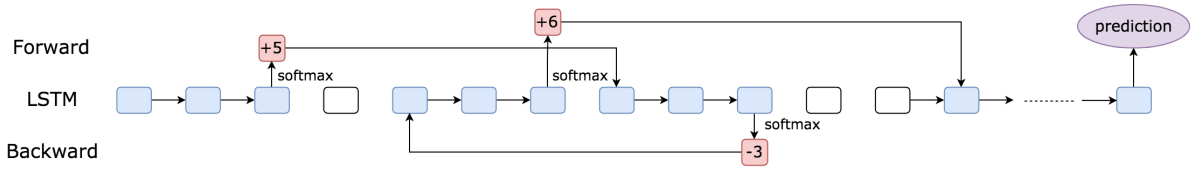

Figure 1: An overview of LSTM-Shuttle. In this example, we set the number of words read sequentially before shuttling $R = 2$, and the maximum shuttle size $K = 10$. The shuttle action is sampled from $[-K, K]$. After reading the entire text, the last hidden state is used to answer prediction. Note that when going backward, the shuttle step is counted before reading sequentially.

As with LSTM-Jump (Yu et al., 2017), we use the following notation:

- N: total number of allowed shuttles
- R: number of words to read before shuttling
- K: maximum shuttle size

Whereas K is a fixed hyperparameter during training, N and R can vary between training and testing. Note that as LSTM-Shuttle proceeds not only forwards but also backwards, the output shuttle size is $2K + 1$: K forward, K backward, and 1 for stopping. In contrast to LSTM-Jump (Yu et al., 2017), when going back, our shuttle step is counted before reading sequentially. In the example in Fig. 1, we set $R = 2$ and $K = 10$.

3.2 Training via Policy Gradient

In LSTM-Shuttle, there are two main parameter sets to compute: θ_R and θ_U. θ_R includes the RNN along with the output prediction parameters, and θ_U represents the parameters of the shuttle mechanism.

We compute θ_R via backpropagation directly by minimizing $J_1(\theta_R)$, the cross entropy loss, which is differentiable over θ_R and is the target objective function of the classification task.

However, this does not work for θ_U. Since cross entropy isn't differentiable over θ_U, we cannot use backpropagation to compute θ_U. Therefore, we recast it as a reinforcement learning problem and apply a policy gradient to train it: we seek to maximize the reward function over θ_U via the following formulation.

We first denote $s_{1:T}$ as the shuttle action sequence when training with text $x_{1:L}$. Assuming that h_i is the hidden state of LSTM before the i-th shuttle s_i, it is a function of $s_{i:i-1}$ and thus can be denoted as $h_i(s_{1:i-1})$. Also, the shuttle action can be sampled from the distribution of $p(s_t|h_i(s_{1:t-1}); \theta_U)$, which is determined by the shuttle softmax. We have the final prediction after LSTM-Shuttle processes text $x_{1:L}$ under the current θ_U shuttle strategy. As with (Yu et al., 2017), we set the reward to 1 if the prediction is correct, and -1 otherwise.

$$R = \begin{cases} 1 & \text{if predicted correctly} \\ -1 & \text{otherwise} \end{cases}$$

The objective function of θ_U we seek to maximize is the expected reward under the distribution over θ_U shuttle strategy, i.e.,

$$J_2(\theta_U) = \mathbb{E}_{p(s_{1:T};\theta_U)}[R], \tag{1}$$

where $p(s_{1:T;\theta_U}) = \prod_i p(s_{1:i}|h_i(s_{1:i-1}; \theta_U)$.

To maximize the objective function, we must compute the gradient of Eq. (1). We compute an approximate gradient by running M examples with the REINFORCE algorithm (Williams, 1992):

$$\begin{aligned} \nabla_{\theta_U} J_2(\theta_U) &= \sum_{i=1}^{T} \mathbb{E}_{p(s_{1:T;\theta_U})}[\nabla_{\theta_U} \log p(s_{1:i}|h_i; \theta_U) R] \\ &\approx \frac{1}{M} \sum_{m=1}^{M} \sum_{i=1}^{T} [\nabla_{\theta_U} \log p(s^m_{1:i}|h^m_i; \theta_U) R^m], \end{aligned}$$

where the superscript m denotes that it belongs to the m-th example. Eventually, the term $\nabla_{\theta_U} \log p(s_{1:i}|h_i; \theta_U)$ is computed by backpropagation as usual.

Though the approximation of $\nabla_{\theta_U} J_2(\theta_U)$ is unbiased, it may have very high variance (Williams, 1992). One common way to reduce this variance is to subtract a baseline value b from the reward function R, transforming the approximated gradient into

$$\nabla_{\theta_U} J_2(\theta_U) \approx \frac{1}{M} \sum_{m=1}^{M} \sum_{i=1}^{T} [\nabla_{\theta_U} \log p(s^m_{1:i}|h^m_i; \theta_U) R^m - b^m_i].$$

Here we apply same bias strategy as (Lewis et al., 2017), treating the bias value b as the average reward from then until now.

The final objective function for LSTM-Shuttle to minimize is

$$J(\theta_R, \theta_U) = J_1(\theta_R) - J_2(\theta_U),$$

which is entirely differentiable and can be computed by standard backpropagation.

3.3 Implementation detail and Inference

To simulate negative step selection, which corresponds to reading backward in the shuttle action, we set the shuttle output dimension to $[0, 2K]$, where 0 maps to $-K$, 1 maps to $-(K-1)$, ..., K maps to $0, \ldots, 2K-1$ maps to $+(K-1)$, and $2K$ maps to $+K$.

We used the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 10^{-3} for all experiments. For a fair comparison with (Yu et al., 2017), the dropout rate between LSTM layers was set to 0.2 and the embedding dropout rate to 0.1. We implemented LSTM-Shuttle in PyTorch (Adam et al., 2017) on a GTX 1080Ti gpu.

During inference, we apply greedy sampling: we select the most probable shuttle step from the shuttle softmax distribution.

4 Experiments

In this section, we evaluate the proposed LSTM-Shuttle on three different tasks: sentiment analysis, news classification, and cloze on four different datasets. We use IMDB (L. et al., 2011) and Rotten Tomatoes (Pang and Lee, 2005) for sentiment analysis, AG (Shang et al., 2015) for news article classification, and Children's Book Test (Hill et al., 2015) for cloze. Table 1 contains statistics for the tasks and datasets in our experiments.

To show the improvement in not only accuracy but also efficiency, we compared LSTM-Shuttle with three baselines: vanilla LSTM (Hochreiter and Schmidhuber, 1997), LSTM-Jump (Yu et al., 2017), and bi-directional LSTM-Jump, as shown in Fig. 2. For a fair comparison, we trained LSTM-Shuttle with the same LSTM settings as LSTM-Jump. For example, for sentiment analysis on IMDB, LSTM-Jump was trained with R=20, K=40, and N=80; we trained LSTM-Shuttle with the same parameters. Vanilla LSTM is the traditional recurrent neural network using LSTM cells which reads the entire text and then outputs the prediction. LSTM-Jump has a skim mechanism which neglects some text. Bi-directional LSTM-Jump applies LSTM-Jump twice but starting from different directions, and concatenates the last hidden state for answer prediction. To shorten the presentation, for LSTM-Jump we selected only two results from the original paper directly on each dataset: one with the best accuracy and the other with the highest efficiency. This is to show the difference between reading in two directions and shuttling. The quantitative result of each dataset is shown in the following sections.

In addition to the quantitative results, we sought to investigate how the shuttle mechanism progresses in reality. We present the shuttle statistics for different K settings, and show that because of the backward mechanism which affords a chance to recover lost information, LSTM-Shuffle shuttles with larger steps, increasing the shuttle step size as it grows more and more confident in its prediction.

4.1 Sentiment Analysis on IMDB and Rotten Tomatoes

Sentiment analysis is a classic natural language processing task, in which we read an article and predict its latent sentiment as positive or negative. It is widely implied in many forms or questionnaires such as satisfaction surveys. Here we use IMDB (L. et al., 2011) and Rotten Tomatoes (Pang and Lee, 2005) as our sentiment analysis datasets.

4.1.1 IMDB Results

IMDB (L. et al., 2011), a well-known movie information website, also includes audience comments and their sentiments. It contains 25,000 training data and 25,000 testing data, where the average length is 241 words. Both baselines and LSTM-Shuttle used a single layer and 128 hidden units as LSTM cells. We used pre-trained word2vec embeddings (Le and Mikolov, 2014b) as initial word embeddings and trained it along with LSTM. For a comparison with the baselines, all models were trained under $R = 20$, $K = 40$, and $N = 5$. We also show the result of a larger shuttle step ($K = 75$) version of LSTM-Shuttle.

Table 2 shows the experimental results for IMDB, where the speedup ratio is compared with vanilla LSTM, conducted on a machine with a single GTX 1080Ti GPU. For bi-directional LSTM-Jump we used our own implementation, and the

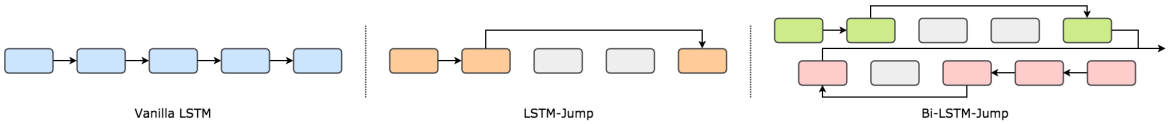

Figure 2: Baseline architectures. **Left:** Vanilla LSTM: reads entire text sequentially. **Middle:** LSTM-Jump: skims to neglect some words. **Right:** Bi-direction LSTM-Jump: skims in two directions, concatenates two latest hidden states.

Task	Dataset	Level	AvgLen	#train	#test	#class
Sentiment analysis	IMDB	word	241 words	21,143	25,000	2
Sentiment analysis	Rotten Tomatoes	word	22 words	8,835	1,030	2
News classification	AG	character	200 characters	101,851	7,600	4
Cloze	CBT-NE	sentence	20 sentences	120,769	2,500	10

Table 1: Tasks and datasets

Method	(R, K, N)	Accuracy	Speedup	Backward
LSTM	-	89.1%	1x	-
LSTM-Jump	(80, 40, 8)	89.4%*	1.64x	-
LSTM-Jump	(100, 40, 1)	88.0%	**2.54x***	-
Bi-LSTM-Jump	(80, 40, 8)	89.6%*	1.12x	-
Bi-LSTM-Jump	(100, 40, 1)	88.4%	2.33x*	-
LSTM-Shuttle	(20, 40, 5)	88.9%	2.43x	0.23
LSTM-Shuttle	(60, 40, 6)	**89.9%***	2.08x	0.26
LSTM-Shuttle	(80, 40, 8)	89.7%	1.49x	0.27
LSTM-Shuttle	(100, 40, 1)	88.6%	2.46x*	0.12
LSTM-Shuttle	(50, **75**, 4)	89.7%*	2.27x	0.29
LSTM-Shuttle	(50, **75**, 2)	89.1%	2.45x*	0.21

Table 2: Sentiment analysis results on IMDB. * means the best accuracy or highest speedup for each method given the same setting of K.

"Backward" column in the table shows the frequency ratio of backward shuttles.

Under the same (R, K, N) setting, LSTM-Shuttle is a little slower than LSTM-Jump since the softmax size of K is larger, but the former yields better prediction. For bi-directional LSTM-Jump, since it applies LSTM-Jump twice, it predicts better than the original. However, we doubt whether it is worth sacrificing so much efficiency for such a small increase in prediction accuracy ($+0.2\%$). Under the same (R, K, N), LSTM-Shuttle is more accurate and faster than bi-directional LSTM-Jump, even though $(80, 40, 8)$ is not in fact the best setting for LSTM-Shuttle. LSTM-Shuttle achieves the highest accuracy (89.9%) with $2.08\times$ acceleration under $(60, 40, 6)$. Due to the shuttle mechanism that can go back, LSTM-Shuttle does not need to read many words before each shuttle, thus accelerating the overall reading process.

In general, the combination of (R, N) represents a trade-off between accuracy and efficiency. If we use a larger (R, N), the model reads more words and predicts better but more slowly. Otherwise, for a smaller (R, N), the model reads faster but yields predictions that are not as accurate. A similar tendency is found when it comes to the backward ratio. A smaller N means LSTM-Shuttle shuttles less often, so the model tends to read through as much as possible, making for a lower backward ratio. On the other hand, LSTM-Shuttle can shuttle many times so it is willing to go back to correct misunderstandings.

We also show the result for $K = 75$. With the larger shuttle step, fewer words are read before shuttling, which accelerates reading but has little impact on accuracy. LSTM-Shuttle achieves 89.7% with $2.27\times$ speedup under $(50, 75, 4)$ and 89.1% with $2.45\times$ times speedup under $(50, 75, 2)$: both settings yield both high accuracy and efficiency.

4.1.2 Rotten Tomatoes Results

The Rotten Tomatoes dataset (Pang and Lee, 2005) is to IMDB. We chose to use a two-layer LSTM and 256 hidden units, and again used the pre-trained word2vec embeddings (Le and Mikolov, 2014b). We trained all models under $R = 8$, $K = 10$, and $N = 3$.

The experimental results are shown in Table 3. LSTM-Shuttle achieves an accuracy of 79.5% with $1.55\times$ speedup; a higher efficiency version accelerates to $1.89\times$. These results demonstrate a similar trade-off tendency with different (R, N) combinations as those for IMDB. Since the average comment length in Rotten Tomatoes is short, we used lower shuttle times ($N = 2$) for better speed but also maintained high accuracy. Because

Method	(R, K, N)	Accuracy	Speedup	Backward
LSTM	-	79.1%	1x	-
LSTM-Jump	(7, 10, 4)	79.3%*	1.56x	-
LSTM-Jump	(9, 10, 2)	78.3%	1.94x*	-
Bi-LSTM-Jump	(7, 10, 4)	79.4%*	1.32x	-
Bi-LSTM-Jump	(9, 10, 2)	78.9%	1.54x*	-
LSTM-Shuttle	(7, 10, 4)	79.5%	1.52x	0.41
LSTM-Shuttle	(8, 10, 3)	79.5%*	1.55x	0.41
LSTM-Shuttle	(9, 10, 2)	79.3%	1.89x*	0.45
LSTM-Shuttle	(6, **20**, 3)	**79.8%***	1.74x	0.39
LSTM-Shuttle	(6, **20**, 2)	79.2%	**1.97x***	0.48

Table 3: Sentiment analysis results for Rotten Tomatoes. * means the best accuracy or highest speedup for each method given the same setting of K.

Method	(R, K, N)	Accuracy	Speedup	Backward
LSTM	-	88.1%	1x	-
LSTM-Jump	(30, 40, 5)	88.5%*	1.24x*	-
LSTM-Jump	(40, 40, 6)	87.4%	0.83x	-
Bi-LSTM-Jump	(30, 40, 5)	89.5%*	1.08x*	-
Bi-LSTM-Jump	(40, 40, 6)	88.4%	0.81x	-
LSTM-Shuttle	(20, 40, 5)	**90.1%***	1.34x*	0.27
LSTM-Shuttle	(30, 40, 5)	88.9%	1.16x	0.30
LSTM-Shuttle	(40, 40, 6)	88.4%	0.82x	0.34
LSTM-Shuttle	(20, **80**, 4)	89.8%	**1.63x***	0.26
LSTM-Shuttle	(30, **80**, 4)	**90.1%***	1.29x	0.28

Table 4: News classification result on AG. * means the best accuracy or highest speedup for each method given the same setting of K.

of the shorter comments and lower shuttle times, LSTM-Shuttle prefers to shuttle over almost the entire text from the beginning, trying to see the last part of a comment, and then goes back to the middle part; thus the backward ratio is much higher.

We also show the results under a larger K ($K = 20$). On Rotten Tomatoes, a larger shuttle step seems more suitable. LSTM-Shuttle achieves the best accuracy (79.8%) with 1.74× high efficiency. A setting with fewer shuttles further accelerates up to 1.97× while maintaining a high accuracy of 79.2%.

4.2 News Article Classification on AG dataset

News classification is a common application of document comprehension. Given a news article, the model must recognize which field it belongs to. Modern topic classification is applied on different target sources such as blog posts. We used AG (Shang et al., 2015) as news article classification dataset in our experiments.

4.2.1 Result on AG dataset

We used the subset constructed by (Shang et al., 2015) for classification at the character level. AG contains news covering four topics (World, Sports, Business, Sci/Tech), each of which includes 30,000 training and 1,900 testing documents. We used a single-layer LSTM with 64 hidden units. We trained the character embedding with 16 dimensions for 70 characters in total, per LSTM-Jump (Yu et al., 2017). We trained all models using $R = 30$, $K = 40$, and $N = 5$.

As shown in Table 4, LSTM-Shuttle still yields improvement at the character level. For both LSTM-Jump and bidirectional LSTM-Jump, the speedup effect is not obvious because of the computational overhead of skim being larger than when processing the entire text directly. On the other hand, LSTM-Shuttle yields accurate predictions even when reading fewer characters before shuttling, and it clearly yields accelerated performance. We reach an accuracy of 90.1% with 1.34× speedup, both far from LSTM-Jump. Interestingly, $(20, 40, 5)$ reads fewer words than $(30, 40, 5)$, but the former predicts better. This may be due to the character-level nature of the task and because many characters actually mislead the model. This can be seen with LSTM-Jump as well. The backward ratio on the AG dataset is nearly to that of IMDB: almost three times forward with once backward.

We also show that a larger setting of K ($K = 80$) yields further acceleration, even at the character level, achieving the highest (1.63×) speedup.

4.3 Cloze on Children's Book Test Name Entity dataset

For the cloze task, we must supply the missing words in an article. In the Children's Book Test (CBT) (Hill et al., 2015), the question includes a complete article and a query from which a specific word is deleted. The model must determine which of the ten candidate words is most suitable. In contrast to previous tasks, which have a fixed class type, CBT provides different candidate words for each question. Thus we cannot train it as with a normal classification problem. Inspired by (Chen et al., 2016), we formulate the task as

$$\text{softmax}(CWh_o) \in \mathbb{R}^{10}, \quad (2)$$

where $C \in \mathbb{R}^{10 \times d}$ is the word embedding matrix, h_o is the latest LSTM hidden state, and W is a trainable weight variable. The output of the above equation is taken as the index of the answer word. We train LSTM-Shuttle to maximize the distribution over a one-hot answer index. Therefore for

Method	(R, K, N)	Accuracy	Speedup	Backward
LSTM	-	45.3%	1x	-
LSTM-Jump	(1, 5, 5)	46.8%*	3.05x	-
LSTM-Jump	(1, 5, 1)	45.2%	**6.28x***	-
Bi-LSTM-Jump	(1, 5, 5)	47.0%*	2.64x	-
Bi-LSTM-Jump	(1, 5, 1)	45.3%	6.19x*	-
LSTM-Shuttle	(1, 5, 5)	**47.2%***	2.98x	0.31
LSTM-Shuttle	(1, 5, 1)	46.0%	6.16x*	0.18
LSTM-Shuttle	(1, **10**, 5)	47.1%*	2.91x	0.36
LSTM-Shuttle	(1, **10**, 1)	46.6%	6.13x*	0.27

Table 5: Cloze result on CBT-NE. * means the best accuracy or highest speedup for each method given the same setting of K.

different candidate words, we concatenate them as an embedding matrix, feed this into the above equation, and generate the prediction distribution. We used the named-entity (NE) part of CBT as the cloze dataset when evaluating LSTM-Shuttle.

4.3.1 Result on CBT-NE

CBT-NE includes 120,769 questions for training and 2,500 for testing. We trained all models using Eq. (2) with a two-layer LSTM and 256 hidden units. Pre-trained word2vec embeddings were again applied directly. We trained them under $R = 1$, $K = 5$, and $N = 5$ at the sentence level, which means that LSTM-Shuttle read one sentence and shuttled several sentences five times. Vanilla LSTM, LSTM-Jump, Bi-LSTM-Jump, and LSTM-Shuttle all read the query, but only vanilla LSTM read the entire question article. Others decided how to skim or shuttle.

The result is reported in Table 5. LSTM-Shuttle's best accuracy is 47.2% with 2.98× speedup, and the highest efficiency version achieved 46.0% accuracy with 6.16× speedup. Despite the modest acceleration effect, LSTM-Shuttle yields consistently better prediction accuracy with minimal drops in efficiency. Acceleration is only modest because the average number of article sentences was only 20, and it thus did not need to read many sentences ($R = 1$) before shuttling or shuttling so many times ($N = 5$). That is also why the backward ratio here is low in CBT-NE.

LSTM-Shuttle maintains accurate prediction and high efficiency under a larger K: 46.6% accuracy with 6.13× speedup. To demonstrate the proposed method, we offer a case study in Section 4.5.

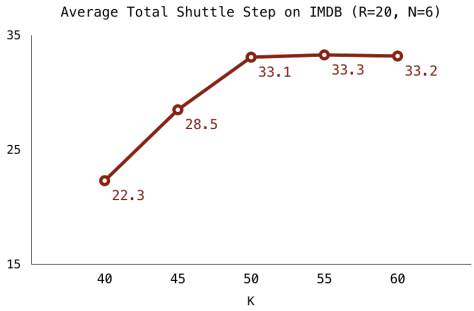

Figure 3: Average total shuttle steps

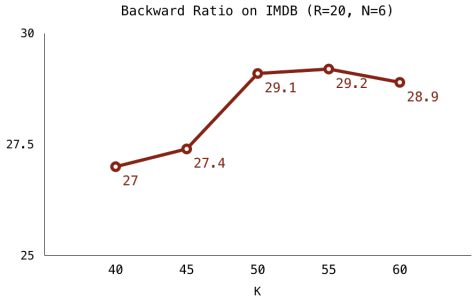

Figure 4: Backward ratio

4.4 Analysis of Shuttle Mechanism

Here we analyze how LSTM-Shuttle actually operates. We compute the total average shuttle steps and the average shuttle steps for each shuttle action under same $R = 20$ and $N = 6$ but different K on the IMDB datasets.

Fig. 3 shows the average total shuttle steps for different K, taking into consideration both forward and backward steps. For backward shuttling, we use the absolute value as its shuttle steps. For instance, shuttling -5 means it goes back 5 words, and the shuttle step is 5 indeed. We can see that a larger K, and thus a larger shuttle space, tends to shuttle larger steps, but also converges for large enough values of K. We see the same thing in the backward ratio. Fig. 4 shows the backward ratio

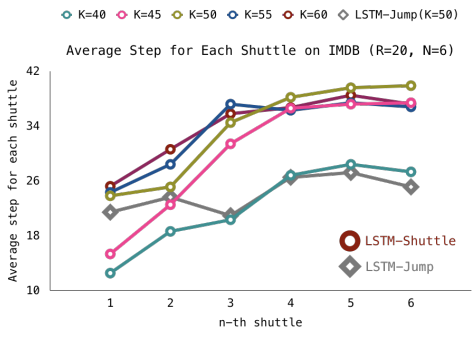

Figure 5: Average steps per shuttle

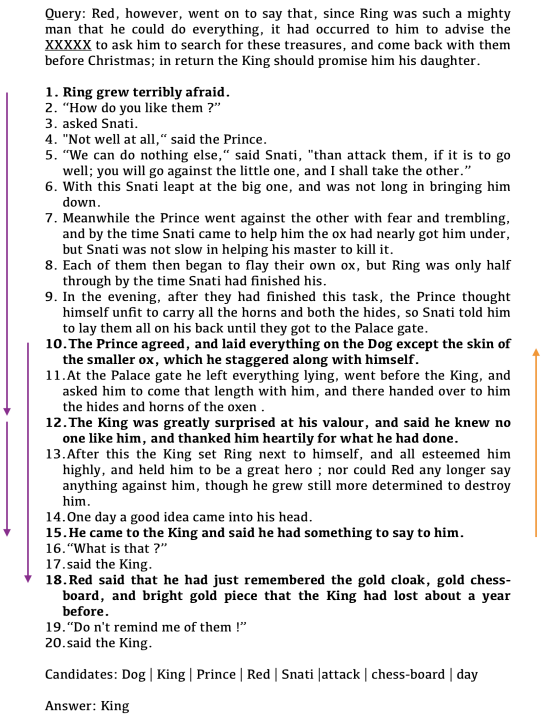

Figure 6: Example 1 for $R = 1$, $K = 15$, and $N = 4$, where read sentences are shown in bold

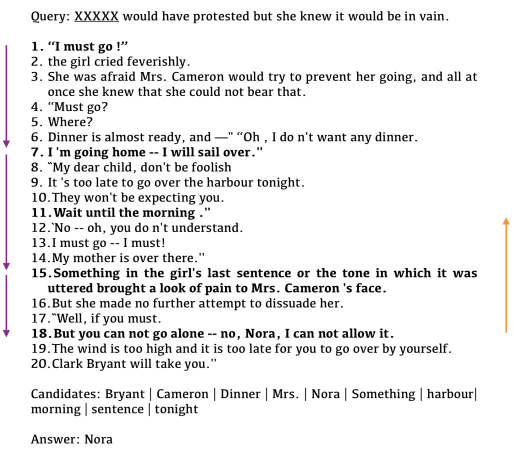

Figure 7: Example 2 for $R = 1$, $K = 15$, and $N = 4$, where read sentences are shown in bold

under different K. Since a larger K shuttles more both forward and backward, it has more chances to go back. Also, it converges when K is large enough.

In addition to the total average, we seek to understand how LSTM-Shuttle shuttles for each. As above, both forward and backward are taken into consideration and the absolute value is used for the backward steps. We show each shuttle record for a total of 6 shuttles under K between $[40, 50]$. As shown in Fig. 5, all settings of LSTM-Shuttle shuttle more steps after shuttling more times, but LSTM-Jump skims at an almost fixed frequency. Thus the model reads more words after more shuttles since it tends to read sequential words before each shuttle, in turn yielding better comprehension for the model. For LSTM-Shuttle, with its backward mechanism to recover lost information, it shuttles with larger and larger steps. However, for LSTM-Jump, as it cannot go back, it reads more carefully and maintains a constant skim step.

4.5 Case Study

Below, we show two examples of LSTM-Shuttle shuttling on the CBT-NE dataset. Example 1 in Fig. 6 illustrates a simple case. Based only on the query, "King should promise him his daughter", the deleted word clearly should be "King". LSTM-Shuttle shows more confidence providing the answer given only this query, so it shuttles with large steps to read the last part of the article, from sentence 1 to 12. Also, it goes back to confirm its prediction: from sentence 15 to 10. Example 2 in Fig. 7 is a more difficult task because the answer word "Nora" appears only once in the entire article. Thus LSTM-Shuttle must read more carefully. The shuttle steps are all smaller than 8 before discovering the answer in sentence 18, after which it goes back to see if it missed something. From these examples, we see that the shuttle mechanism is used in diverse manners for queries with different difficulties. For simple queries, LSTM-Shuttle shuttles in large steps, while for difficult queries, it shuttles more conservatively. In both cases we witness the ability to go back if necessary to make sure it understands correctly.

5 Conclusion

We present LSTM-Shuttle to use human speed reading techniques for text comprehension. In addition to reading forward and skimming over text to accelerate, LSTM-Shuttle goes back to recover lost information or double-check its grasp of the text's meaning. We evaluate LSTM-Shuttle on sentiment analysis, news classification, and cloze on IMDB, Rotten Tomatoes, AG, and Children's Book Test datasets. We show that LSTM-Shuttle predicts better on all datasets with higher efficiency. We also analyze LSTM-Shuttle's behavior under different shuttle step restrictions, and provide case studies that reveal the specific shuttle operations; these show how the model comprehends the context to achieve specific goals.

References

Paszke Adam, Gross Sam, Chintala Soumith, Chanan Gregory, Yang Edward, DeVito Zachary, Lin Zeming, Desmaison Alban, Antiga Luca, and Lerer Adam. 2017. Automatic differentiation in pytorch. In *Proceedings of Neural Information Processing Systems*.

Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saeid Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, and Krys Kochut. 2017. Text summarization techniques: A brief survey. In *arXiv preprint arXiv:1707.02268*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*.

Victor Campos, Brendan Jou, Xavier Giro i Nieto, Jordi Torres, and Shih-Fu Chang. 2018. Skip rnn: Learning to skip state updates in recurrent neural networks. In *International Conference on Learning Representations*.

Danqi Chen, Jason Bolton, and Christopher D. Manning. 2016. A thorough examination of the cnn/daily mail reading comprehension task. In *Meeting of the Association for Computational Linguistics*.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. In *Meeting of the Association for Computational Linguistics*.

Jason P.C. Chiu and Eric Nichols. 2015. Named entity recognition with bidirectional lstm-cnns. In *Transactions of the Association for Computational Linguistics*.

Benjamin Van Durme and Lenhart Schubert. 2008. Open knowledge extraction through compositional language processing. In *Conference on Semantics in Text Processing*.

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. 2015. The goldilocks principle: Reading children's books with explicit memory representations. In *arXiv preprint arXiv:1511.02301*.

Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. In *Neural computation*.

Yoon Kim. 2014. Convolutional neural net- works for sentence classification. In *arXiv preprint arXiv:1408.5882*.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *arXiv preprint arXiv:1412.6980*.

Maas Andrew L., Daly Raymond E., Pham Peter T., Huang Dan, Ng Andrew Y., and Potts Christopher. 2011. Learning word vectors for sentiment analysis. In *Meeting of the Association for Computational Linguistics*.

Quoc V. Le and Tomas Mikolov. 2014a. Distributed representations of sentences and documents. In *International Conference on Machine Learning*.

Quoc V. Le and Tomas Mikolov. 2014b. Distributed representations of sentences and documents. In *International Conference on Machine Learning*.

Mike Lewis, Denis Yarats, Yann Dauphin, Devi Parikh, and Dhruv Batra. 2017. Deal or no deal? end-to-end learning of negotiation dialogues. In *Proceedings of Empirical Methods in Natural Language Processing*.

Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. 2017. Coherent dialogue with attention-based language models. In *Association for the Advancement of Artificial Intelligence*.

Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. 2015. Recurrent models of visual attention. In *Proceedings of Neural Information Processing Systems*.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Meeting of the Association for Computational Linguistics*.

Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. 2018. Neural speed reading via skim-rnn. In *International Conference on Learning Representations*.

Lifeng Shang, Zhengdong Lu, and Hang Li. 2015. Neural responding machine for short-text conversation. In *Meeting of the Association for Computational Linguistics*.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2016. Reasonet: Learning to stop reading in machine comprehension. In *Knowledge Discovery and Data Mining*.

Siddarth Srinivasan, Richa Arora, and Mark Riedl. 2018. A simple and effective approach to the story cloze test. In *arXiv preprint arXiv:1803.05547*.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of Neural Information Processing Systems*.

Peilu Wang, Yao Qian, Frank K. Soong, Lei He, and Hai Zhao. 2015. Part-of-speech tagging with bidirectional long short-term memory recurrent neural network. In *Meeting of the Association for Computational Linguistics*.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*.

Adams Wei Yu, Hongrae Lee, and Quoc V. Le. 2017. Learning to skim text. In *Meeting of the Association for Computational Linguistics*.

Keyi Yu, Yang Liu, Alexander G. Schwing, and Jian Peng. 2018. Fast and accurate text classification: Skimming, rereading and early stopping. In *International Conference on Learning Representations*.

Lei Zhang, Shuai Wang, and Bing Liu. 2018. Deep learning for sentiment analysis : A survey. In *arXiv preprint arXiv:1801.07883*.