Yu, Adams Wei, Hongrae Lee, and Quoc V. Le. "Learning to skim text." *arXiv preprint arXiv:1704.06877* (2017).

Byeongjo Kim

0. Abstract

Recurrent Neural Networks are showing much promise in many sub-areas of natural language processing, ranging from document classification to machine translation to automatic question answering. Despite their promise, many recurrent models have to read the whole text word by word, making it slow to handle long documents. For example, it is difficult to use a recurrent network to read a book and answer questions about it. In this paper, we present an approach of reading text while skipping irrelevant information if needed. The underlying model is a recurrent network that learns how far to jump after reading a few words of the input text. We employ a standard policy gradient method to train the model to make discrete jumping decisions. In our benchmarks on four different tasks, including number prediction, sentiment analysis, news article classification and automatic Q&A, our proposed model, a modified LSTM with jumping, is up to 6 times faster than the standard sequential LSTM, while maintaining the same or even better accuracy.

1. Introduction

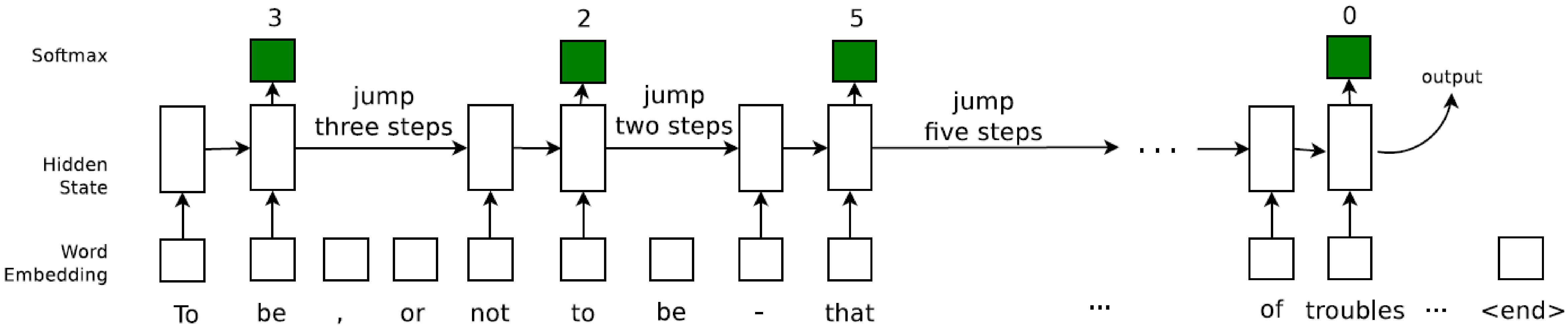
Seo et al., 2016; Xiong et al., 2016). An important characteristic of all these models is that they read all the text available to them. While it is essential for certain applications, such as machine translation, this characteristic also makes it slow to apply these models to scenarios that have long input text, such as document classification or automatic Q&A. However, the fact that texts are usually written with redundancy inspires us to think about the possibility of reading selectively.

In this paper, we consider the problem of understanding documents with partial reading, and propose a modification to the basic neural architectures that allows them to read input text with skipping. The main benefit of this approach is faster inference because it skips irrelevant information. An unexpected benefit of this approach is that it also helps the models generalize better.

In our approach, the model is a recurrent network, which learns to predict the number of jumping steps after it reads one or several input tokens. Such a discrete model is therefore not fully differentiable, but it can be trained by a standard policy gradient algorithm, where the reward can be the accuracy or its proxy during training.

In our experiments, we use the basic LSTM recurrent networks (Hochreiter and Schmidhuber, 1997) as the base model and benchmark the proposed algorithm on a range of document classification or reading comprehension tasks, using various datasets such as Rotten Tomatoes (Pang

2. Methodology



2.1 Model Overview

The main architecture of the proposed model is shown in Figure 1, which is based on an LSTM recurrent neural network. Before training, the number of jumps allowed N, the number of tokens read between every two jumps R and the maximum size of jumping K are chosen ahead of time. While K is a fixed parameter of the model, N and R are hyperparameters that can vary between training and testing. Also, throughout the paper, we would use $d_{1:p}$ to denote a sequence $d_1, d_2, ..., d_p$.

LSTM을 사용

N : number of jumps,
fixed parameter

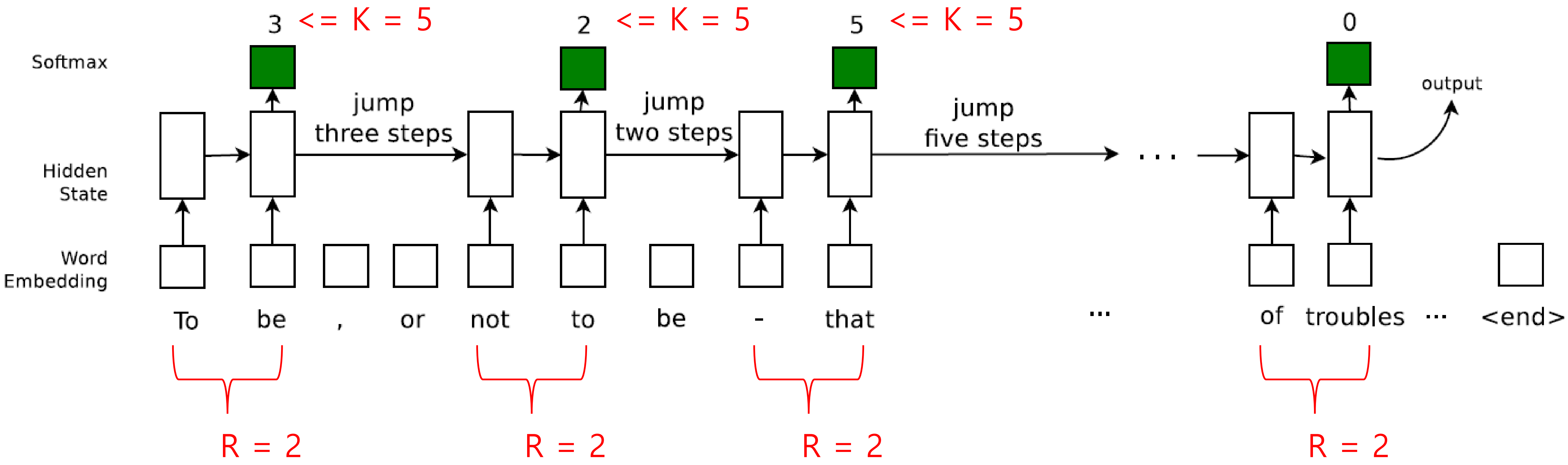
R : number of tokens read between every two jumps
hyperparameters

K : maximum size of jumping,
hyperparameters

$d_{1:p}$: d_1, d_2, d_3, … d_p sequence

2.1 Model Overview

N = number of jumps



In the following, we describe in detail how the model operates when processing text. Given a training example $x_{1:T}$, the recurrent network will read the embedding of the first R tokens $x_{1:R}$ and output the hidden state. Then this state is used to compute the jumping softmax that determines a distribution over the jumping steps between 1 and K. The model then samples from this distribution a jumping step, which is used to decide the next token to be read into the model. Let κ be the sampled value, then the next starting token is $x_{R+\kappa}$. Such process continues until either

a) the jump softmax samples a 0; or
b) the number of jumps exceeds N; or
c) the model reaches the last token x_T.

After stopping, as the output, the latest hidden state is further used for predicting desired targets.

$x_{1:T}$: x_1, x_2, x_3, ... x_t

jumping steps(κ) = softmax(lstm($x_{1:R}$))
jumping steps(κ) = softmax(lstm($x_{R+\kappa:R+\kappa+R}$))
jumping steps(κ) = softmax(lstm($x_{R+\kappa+R+\kappa:R+\kappa+R+\kappa+R}$

$$\vdots$$

2.2 Training with REINFORCE

Our goal for training is to estimate the parameters of LSTM and possibly word embedding, which are denoted as θ_m, together with the jumping action parameters θ_a. Once obtained, they can be used for inference.

The estimation of θ_m is straightforward in the tasks that can be reduced as classification problems (which is essentially what our experiments cover), as the cross entropy objective $J_1(\theta_m)$ is

However, the nature of discrete jumping decisions made at every step makes it difficult to estimate θ_a, as cross entropy is no longer differentiable over θ_a. Therefore, we formulate it as a reinforcement learning problem and apply policy gradient method to train the model. Specifically, we need to maximize a reward function over θ_a which can be constructed as follows.

θ_m : LSTM과 word embedding의 파라미터들
일반적인 분류 문제와 같다
θ_m로 미분가능한 $J_1(\theta_m)$ → backpropagation 사용

θ_a : jumping action을 결정하는 파라미터들
점프 결정은 분리되어 있어
cross entropy로 θ_a estimate 어려움
reinforcement learning problem으로 해결

2.2 Training with REINFORCE

Let $j_{1:N}$ be the jumping action sequence during the training with an example $x_{1:T}$. Suppose h_i is a hidden state of the LSTM right before the i-th jump j_i,[1] then it is a function of $j_{1:i-1}$ and thus can be denoted as $h_i(j_{1:i-1})$. Now the jump is attained by sampling from the multinomial distribution $p(j_i|h_i(j_{1:i-1});\theta_a)$, which is determined by the jump softmax. We can receive a reward R after processing $x_{1:T}$ under the current jumping strategy.[2] The reward should be positive if the output is favorable or non-positive otherwise. In our experiments, we choose

$$R = \begin{cases} 1 & \text{if prediction correct;} \\ -1 & \text{otherwise.} \end{cases}$$

Then the objective function of θ_a we want to maximize is the expected reward under the distribution defined by the current jumping policy, i.e.,

$$J_2(\theta_a) = \mathbb{E}_{p(j_{1:N};\theta_a)}[R]. \qquad (1)$$

where $p(j_{1:N};\theta_a) = \prod_i p(j_{1:i}|h_i(j_{1:i-1});\theta_a)$.

$j_{1:N}$: $x_{1:T}$ 학습 중의 jumping action sequence
h_i : j_i 점프가 있기 바로 직전의 hidden state

$j_i = h_i(j_{1:i-1})$
$p(j_i|h_i(j_{1:i-1});\theta_a)$:
θ_a에 대해 $h_i(j_{1:i-1})$가 주어졌을 때의 j_i

R : 현재의 점프 전략을 이용 했을 때의 보상 (1 or -1)

$J_2(\theta_a) = E_{p(j_{1:N};\theta_a)}[R]$:
$j_{1:N}$ 에 따라 점프를 했을 때의 보상의 기대 값

$p(j_{1:N};\theta_a) = \prod_i p(j_{1:i}|h_i(j_{1:i-1});\theta_a)$

2.2 Training with REINFORCE

Optimizing this objective numerically requires computing its gradient, whose exact value is intractable to obtain as the expectation is over high dimensional interaction sequences. By running S examples, an approximated gradient can be computed by the following REINFORCE algorithm (Williams, 1992):

$$\nabla_{\theta_a} J_2(\theta_a) = \sum_{i=1}^{N} \mathbb{E}_{p(j_{1:N};\theta_a)}[\nabla_{\theta_a} \log p(j_{1:i}|h_i;\theta_a) R]$$
$$\approx \frac{1}{S} \sum_{s=1}^{S} \sum_{i=1}^{N} [\nabla_{\theta_a} \log p(j_{1:i}^s|h_i^s;\theta_a) R^s]$$

where the superscript s denotes a quantity belonging to the s-th example. Now the term $\nabla_{\theta_a} \log p(j_{1:i}|h_i;\theta_a)$ can be computed by standard backpropagation.

$$\nabla_{\theta_a} J_2(\theta_a) = \sum_{i=1}^{N} \mathbb{E}_{p(j_{1:N};\theta_a)}[\nabla_{\theta_a} \log p(j_{1:i}|h_i;\theta_a) R]$$

-> log 취하고 미분 하고 나서,

$$\approx \frac{1}{S} \sum_{s=1}^{S} \sum_{i=1}^{N} [\nabla_{\theta_a} \log p(j_{1:i}^s|h_i^s;\theta_a) R^s]$$

-> 위의 식으로 기대값을 합으로 변환하면서 gradient의 근사값 계산 가능

2.2 Training with REINFORCE

Although the above estimation of $\nabla_{\theta_a} J_2(\theta_a)$ is unbiased, it may have very high variance. One widely used remedy to reduce the variance is to subtract a *baseline* value b_i^s from the reward R^s, such that the approximated gradient becomes

$$\nabla_{\theta_a} J_2(\theta_a) \approx \frac{1}{S}\sum_{s=1}^{S}\sum_{i=1}^{N}[\nabla_{\theta_a} \log p(j_{1:i}^s | h_i^s; \theta)(R^s - b_i^s)]$$

It is shown (Williams, 1992; Zaremba and Sutskever, 2015) that any number b_i^s will yield an unbiased estimation. Here, we adopt the strategy of Mnih et al. (2014) that $b_i^s = w_b h_i^s + c_b$ and the parameter $\theta_b = \{w_b, c_b\}$ is learned by minimizing $(R^s - b_i^s)^2$. Now the final objective to minimize is

$$J(\theta_m, \theta_a, \theta_b) = J_1(\theta_m) - J_2(\theta_a) + \sum_{s=1}^{S}\sum_{i=1}^{N}(R^s - b_i^s)^2,$$

which is fully differentiable and can be solved by standard backpropagation.

매우 high variance 함

-> R^s에서 b_i^s(baseline)를 빼면서 variance를 줄임

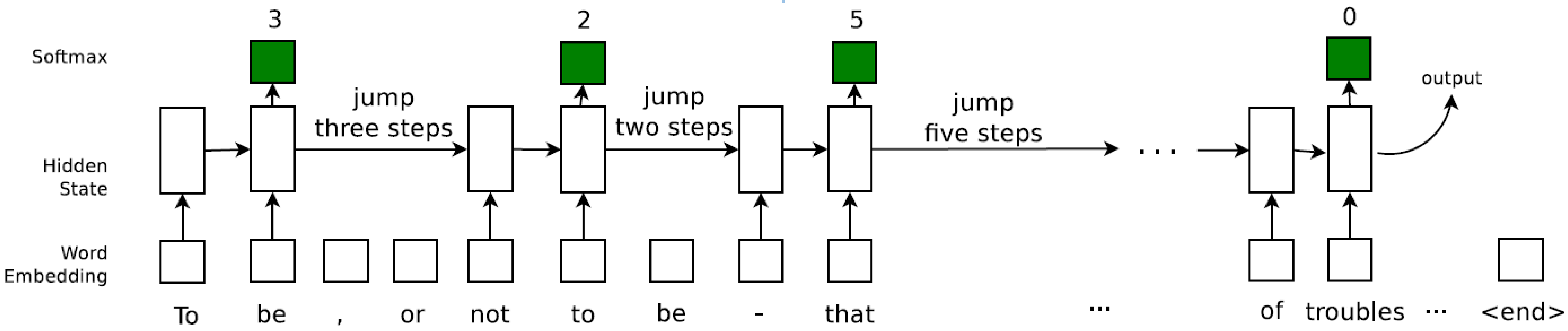
$b_i^s = w_b h_i^s + c_b$로 결정

-> θ_b parameter 또한 학습 해야함

$$J(\theta_m, \theta_a, \theta_b) = J_1(\theta_m) - J_2(\theta_a) + \sum_{s=1}^{S}\sum_{i=1}^{N}(R^s - b_i^s)^2,$$

2.3 Inference

During inference, we can either use sampling or greedy evaluation by selecting the most probable jumping step suggested by the jump softmax and follow that path. In the our experiments, we will adopt the sampling scheme.



3. Experimental Results

Task	Dataset	Level	Vocab	AvgLen	#train	#valid	#test	#class
Number Prediction	synthetic	word	100	100 words	1M	10K	10K	100
Sentiment Analysis	Rotten Tomatoes	word	18,764	22 words	8,835	1,079	1,030	2
Sentiment Analysis	IMDB	word	112,540	241 words	21,143	3,857	25,000	2
News Classification	AG	character	70	200 characters	101,851	18,149	7,600	4
Q/A	Children Book Test-NE	sentence	53,063	20 sentences	108,719	2,000	2,500	10
Q/A	Children Book Test-CN	sentence	53,185	20 sentences	120,769	2,000	2,500	10

3. Experimental Results

Number Prediction with a Synthetic Dataset

Seq length	LSTM-Jump	LSTM	Speedup
Test accuracy			
10	**98%**	96%	n/a
100	**98%**	96%	n/a
1000	**90%**	80%	n/a
Test time (Avg tokens read)			
10	**13.5s (2.1)**	18.9s (10)	1.40x
100	**13.9s (2.2)**	120.4s (100)	8.66x
1000	**18.9s (3.0)**	1250s (1000)	**66.14x**

Word Level Sentiment Analysis with Rotten Tomatoes and IMDB datasets

Model	(R, N)	Accuracy	Time	Speedup
LSTM-Jump	(9, 2)	0.783	**6.3s**	**1.98x**
	(8, 3)	0.789	7.3s	1.71x
	(7, 4)	**0.793**	8.1s	1.54x
LSTM	n/a	0.791	12.5s	1x

Model	(R, N)	Accuracy	Time	Speedup
LSTM-Jump	(80, 8)	**0.894**	769s	1.62x
	(80, 3)	0.892	764s	1.63x
	(70, 3)	0.889	673s	1.85x
	(50, 2)	0.887	585s	2.12x
	(100, 1)	0.880	**489s**	**2.54x**
LSTM	n/a	0.891	1243s	1x

Character Level News Article Classification with AG dataset

Model	(R, N)	Accuracy	Time	Speedup
LSTM-Jump	(50, 5)	0.854	102s	0.80x
	(40, 6)	0.874	98.1s	0.83x
	(40, 5)	0.889	83.0s	0.98x
	(30, 5)	0.885	**63.6s**	**1.28x**
	(30, 6)	**0.893**	74.2s	1.10x
LSTM	n/a	0.881	81.7s	1x

Sentence Level Automatic Question Answering with Children's Book Test dataset

Model	(R, N)	Accuracy	Time	Speedup
Children's Book Test - Named Entity				
LSTM-Jump	(1, 5)	**0.468**	40.9s	3.04x
	(1, 3)	0.464	30.3s	4.11x
	(1, 1)	0.452	**19.9s**	**6.26x**
LSTM	n/a	0.438	124.5s	1x
Children's Book Test - Common Noun				
LSTM-Jump	(1, 5)	0.493	39.3s	3.09x
	(1, 3)	0.487	29.7s	4.09x
	(1, 1)	**0.497**	**19.8s**	**6.14x**
LSTM	n/a	0.453	121.5s	1x