Knowledge-Aware Leap-LSTM: Integrating Prior Knowledge into Leap-LSTM towards Faster Long Text Classification

Jinhua Du, Yan Huang, Karo Moilanen
Investments AI, AIG (American International Group, Inc.)
{jinhua.du, yan.huang, karo.moilanen}@aig.com

Abstract

While widely used in industry, recurrent neural networks (RNNs) are known to have deficiencies in dealing with long sequences (e.g. slow inference, vanishing gradients etc.). Recent research has attempted to accelerate RNN models by developing mechanisms to skip irrelevant words in input. Due to the lack of labelled data, it remains as a challenge to decide which words to skip, especially for **low-resource** classification tasks. In this paper, we propose Knowledge-Aware Leap-LSTM (KALL), a novel architecture which integrates prior human knowledge (created either manually or automatically) like in-domain keywords, terminologies or lexicons into Leap-LSTM to partially supervise the skipping process. More specifically, we propose a knowledge-oriented cost function for KALL; furthermore, we propose two strategies to integrate the knowledge: (1) the Factored KALL approach involves a keyword indicator as a soft constraint for the skipping process, and (2) the Gated KALL enforces the inclusion of keywords while maintaining a differentiable network in training. Experiments on different public datasets show that our approaches are $1.1x \sim 2.6x$ faster than LSTM with better accuracy and $23.6x$ faster than XLNet in a resource-limited CPU-only environment.

Introduction

Recurrent neural networks, including vanilla RNN, Long Short-term Memory (LSTM) (Hochreiter and Schmidhuber 1997) and Gated Recurrent Unit (GRU) (Chung et al. 2014), are suitable for sequential natural language processing (NLP) tasks. They have been widely used in many industrial areas such as finance (Han et al. 2018) and healthcare (Miotto et al. 2018). Typically, RNNs read in either all or a fixed length of text sequentially, and output a distributed representation for each token. However, while such process is essential for some applications like machine translation (Bahdanau, Cho, and Bengio 2015), it makes RNNs slow to process long input text common in e.g. document classification (Liu et al. 2015) or question answering (Tan, Xiang, and Zhou 2015). Meanwhile, humans read text by making a sequence of fixations and saccades, i.e. fixating some words and skip others (Hahn and Keller 2016), because texts are usually written with redundancy and can be understood by partial reading (Yu, Lee, and Le 2017). Some research shows that not all input tokens are equally important in many NLP tasks (Seo et al. 2017).

To accelerate RNNs in processing long text, recent research has been focusing on mechanisms to skip/skim irrelevant, unimportant or redundant words in input for RNNs (Yu, Lee, and Le 2017; Campos et al. 2017; Seo et al. 2017; Liu et al. 2018; Hansen et al. 2019; Huang, Shen, and Deng 2019). We define such models as "*selective* RNNs".

Representative selective RNNs include LSTM-Jump (Yu, Lee, and Le 2017), Skip-RNN (Campos et al. 2017), Skim-RNN (Seo et al. 2017) and Leap-LSTM (Huang, Shen, and Deng 2019). While they involve different modifications of standard RNNs/LSTM, selective RNNs share a crucial component: a decision making network predicting whether a word should be skipped/skimmed or not. In general, the decision making component is jointly trained with the classification targets in a supervised learning framework. However, due to the absence of labelled ground truths on whether a word should be skipped or not, there is no direct supervision signal for the training of the decision making component, resulting in inaccurate skipping process.

To improve the accuracy of skipping process, we propose a novel architecture using two strategies to integrate prior human knowledge to partially supervise the skipping process. For the prior human knowledge, we use existing keywords, terminologies or lexicons, which has been beneficial for many NLP classification tasks such as document categorisation, topic modelling, sentiment analysis etc. In the following sections, we use the term *keyword or keywords* to refer to prior human knowledge. We define our general architecture as Knowledge-Aware Selective RNNs for long text and document classification.

The contributions of this paper include:

- We propose a novel architecture to integrate prior knowledge to selective RNNs. We illustrate this architecture with the state-of-the-art (SOTA) Leap-LSTM (Huang, Shen, and Deng 2019), and refer to the architecture as knowledge-aware Leap-LSTM (KALL) in the rest of the paper. Two different integration strategies are proposed, namely the factored KALL and gated KALL;
- The factored KALL uses a vector to indicate keywords and concatenates the indicator with word embeddings for

skip prediction, which is straight-forward but effective;

- Since the factored KALL uses a *soft* constraint which does not guarantee to keep the keywords, we further propose the gated KALL which utilises a gate mechanism to enforce inclusion of the keywords while maintaining a differentiable network in training;
- We propose a knowledge-oriented cost function to supervise the decision making component;
- We conduct experiments on four public datasets and verify the effectiveness of the proposed architecture.

Related Work

In this section, we will look at some representative work regarding selective RNN/LSTM models, including LSTM-Jump, Skip-RNN, Skim-RNN and Leap-LSTM. These models select and use only parts of texts from input to speed up process, achieving better performance than the standard RNN/LSTM models in many tasks and datasets.

LSTM-Jump learns to predict the number of jumping steps to skip irrelevant information after it reads one or several input tokens, resulting in faster inference (Yu, Lee, and Le 2017). Given a training example $x_{1:T}$, LSTM-Jump reads the embedding of the first R tokens $x_{1:R}$, and uses the resulting hidden state to compute a softmax that determines a distribution over the jumping steps between 1 and K. To solve the non-differentiable jumping action parameters θ_a, reinforcement learning and policy gradient method (Williams 1992) are used. The experiments on different NLP tasks show that the selective reading approach speeds up the base model by *two* to *six* times, beating the standard LSTM in accuracy.

Skip-RNN augments RNN with a binary *state update gate*, $u_t \in \{0, 1\}$ which determines whether the state of the RNN will be updated ($u_t = 1$) or copied from the previous time step ($u_t = 0$) (Campos et al. 2017). At a time step t, the probability $\tilde{u}_{t+1} \in [0, 1]$ of performing a state update at $t + 1$ is pre-calculated. If the model decides to omit a state update, the pre-calculated state update gate for the following time step, $\tilde{u}_{t+1}$, is incremented by $\Delta\tilde{u}_t$. Alternatively, if the model decides to perform a state update, the accumulated value is flushed and $\tilde{u}_{t+1} = \Delta\tilde{u}_t$. The whole model is differentiable except for a binary function. Skip-RNN uses a straight-through estimator (Bengio, Léonard, and Courville 2013) and approximates the binary function by the identity when computing gradients during the backward pass. Evaluation on six different sequence learning tasks shows that Skip-RNN provides faster and more stable training for long sequences and complex models, reducing the number of floating point operations (FLOPs).

Structural-Jump-LSTM (Hansen et al. 2019) combines the advantages of Skip-RNN and LSTM-Jump: it can skip and jump text. The model consists of two agents: one is capable of skipping single words when reading, and one is capable of exploiting punctuation structure (sub-sentence separators (,:), sentence end symbols (.!?), or end of text markers) to jump ahead after reading a word. JUMPER (Liu et al. 2018), inspired by the cognitive process of text reading, scans a piece of text sequentially and makes classification decisions when there is enough evidence, reducing total text reading by 30∼40%.

Skim-RNN *skims* rather than skips tokens (Seo et al. 2017). Skimming refers to spending little time on parts of the text that do not affect the reader's main objective. This is operationalised as using a smaller RNN to update only a fraction of the hidden state. Alternatively, for parts of text that should be read fully, Skim-RNN updates the hidden state with a larger RNN. To solve the non-differentiable issue of the decision function, Skim-RNN uses Gumbel-Softmax (Jang, Gu, and Poole 2017) to estimate the gradient of the function. Experiments show that Skim-RNN can significantly reduce FLOPs, achieving faster inference on CPUs and higher accuracy than skipping tokens.

Most of the aforementioned models use only preceding context (e.g. hidden state at time step $t - 1$) in the decision making component. The resulting networks are difficult to converge and unstable. Leap-LSTM improves over these models by including information from the current word and following context (Huang, Shen, and Deng 2019). Furthermore, leap-LSTM provides a controllable skipping rate by adding a penalty term during training. Experiments show that skipping about 60% or 90% words leads to only insignificant decline in accuracy compared to standard LSTM. Furthermore, Leap-LSTM model can achieve better accuracy with a speed-up ranging from 1.5x∼1.7x. However, Leap-LSTM looks at the words one by one to decide whether to skip or keep, which slows down the processing to some extent. Pointer-LSTM uses a pointer network to go through all words first and then select top-K important words for the final classification (Du, Huang, and Moilanen 2020). Results on four data sets show that Pointer-LSTM is much faster than Leap-LSTM at the skip rate 0.9.

Since Leap-LSTM is the latest work regarding selective RNNs and achieves state-of-the-art results on various public datasets, we illustrate our approach by enhancing Leap-LSTM. We propose a knowledge-aware framework which can utilise prior knowledge (i.e. keywords) to supervise the training of the decision network. The main difference of our work with previous models is that we fully utilise human knowledge to guide the skipping process.

Knowledge-Aware Leap-LSTM (KALL)

Existing selective RNNs achieves significant speed-up, but only marginal improvement in performance. We investigated the skipping decisions of these models and found that many informative or important words are skipped due to the lack of direct supervision on skipping during training.

Since human knowledge can provide complementary signal to neural networks, we propose a knowledge-aware framework to improve the performance of selective RNNs. More specifically, we take Leap-LSTM as the backbone and illustrate the integration of human knowledge to the model. A major challenge here is how to maintain the differentiability of a selective RNN when integrating a discrete variable about keyword knowledge into it, so that back-propagation can be used to train the neural network. We propose two integration approaches as described below.

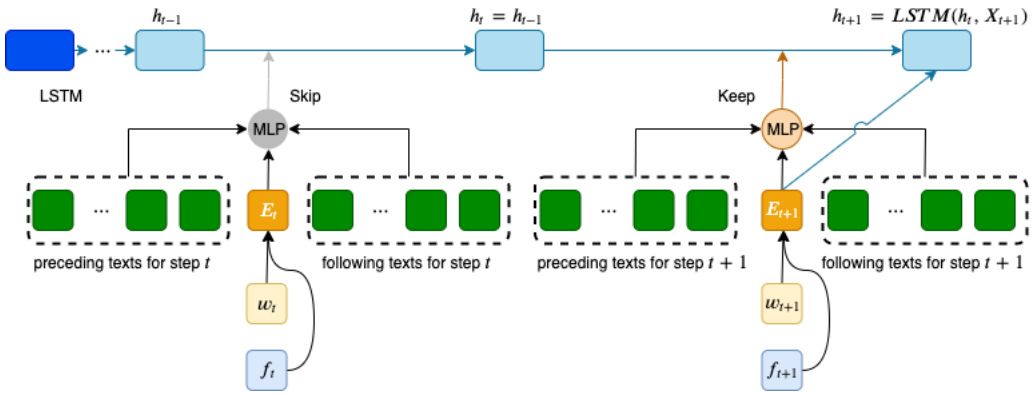

Figure 1: Architecture of Factored KALL

Human Knowledge: Keywords

We define keywords as important and informative words or expressions for an NLP task. They can be extracted automatically or manually. Take binary sentiment analysis as an example, where a given sentence is classified as *positive* or *negative*. Positive sentences tend to include words like "adorable, attractive, distinguished", while negative sentences may contain words like "aggressive, annoying, bored". These words can be regarded as sentiment keywords and used as features in sentiment analysis.

Recent decades have seen a surge of electronic dictionaries, lexicons and terminologies. These human knowledge resources contain abundant keywords useful for augmenting various NLP models. As a result, our knowledge-aware selective RNN framework is widely applicable.

Standard Leap-LSTM

Given an input token x_t at time step t, Leap-LSTM uses a two-layer MLP to calculate a probability distribution over skipping versus keeping the token:

$$s_t = \texttt{RELU}(\boldsymbol{W}_1[x_t; \boldsymbol{f}_{precede}(t); \boldsymbol{f}_{follow}(t)] + \boldsymbol{b}_1) \quad (1)$$

$$\pi_t = \texttt{softmax}(\boldsymbol{W}_2 s_t + \boldsymbol{b}_2) \quad (2)$$

where s_t is the hidden state of the MLP, and π_t represents the aforementioned probability distribution. $\boldsymbol{f}_{precede(t)}$ and $\boldsymbol{f}_{follow(t)}$ are contextual features before and after time step t, respectively. $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are trainable parameters.

A binary decision d_t is then obtained by sampling π_t using Gumbel-softmax, and the hidden state of LSTM is updated as:

$$h_t = \begin{cases} \texttt{LSTM}(h_{t-1}, x_t) & \text{if } d_t = 0 \\ h_{t-1} & \text{if } d_t = 1 \end{cases} \quad (3)$$

where $d_t = 0$ indicates *keep* and $d_t = 1$ indicates *skip*.

The objective function includes a penalty term to control the skipping rate of the model:

$$\mathcal{L} = \mathcal{L}_c + \lambda(r_t - r)^2 \quad (4)$$

where $\mathcal{L}_c$ is the classification loss; r_t is the pre-defined (expected) skip rate while r is the actual skip rate; $\lambda > 0$ is the weight for the penalty term.

In Equation (1), the *preceding* feature $\boldsymbol{f}_{precede}(t)$ is formed by h_{t-1} to encode the information of all processed words. The following feature $\boldsymbol{f}_{follow}(t)$ consists of two parts: local and global contexts. The local context is obtained by applying a convolutional neural network (CNN) to $x_{t+1:t+m}$, where m is the window size of the context. The global feature is produced using a small LSTM on $x_{t+1:T}$, where T is the length of current sequence. Note that the small LSTM reversely reads the texts, i.e. starting from the end of the sequence. The complete *following* feature is constructed as:

$$\boldsymbol{f}_{follow}(t) = \begin{cases} [\texttt{LSTM}_r(t+1); \texttt{CNN}(t+1)] & \text{if } t < T; \\ \boldsymbol{h}_{end} & \text{if } t = T. \end{cases} \quad (5)$$

where h_{end} is a learnable parameter, and $\texttt{LSTM}_r(t+1)$ is the small LSTM reading texts reversely.

The Gumbel-softmax for decision sampling is as:

$$y_i = \frac{exp((\log(\pi_i) + g_i)/\tau}{\sum_{j=1}^{k} exp((\log(\pi_j) + g_j)/\tau)} \quad (6)$$

where $i = \{1, \ldots, k\}$ and k is the number of decisions; τ is the softmax temperature. The update function of Equation (3) is then operationalised as:

$$h_t = [y_t]_0 \cdot \text{LSTM}(h_{t-1}, x_t) + [y_t]_1 \cdot h_{t-1} \quad (7)$$

Knowledge-Oriented Cost Function

Given a set of keywords K, we use k_{w_t} to indicate whether an input token w_t is a keyword. A knowledge-oriented cost function is then created by adding a penalty term to the cost function of Equation (4) as follows:

$$\mathcal{L} = \mathcal{L}_c + \lambda\mathcal{L}_s + \beta\mathcal{L}_k \quad (8)$$

where $\mathcal{L}_s$ represents the loss from the skipping mechanism (i.e. the difference between the pre-defined and actual skip rates), while $\mathcal{L}_k$ represents the cost related to knowledge guidance and weighted by $\beta > 0$.

Our design of $\mathcal{L}_k$ follows the idea that the skipping decision d_t from the decision making component should be consistent with the keyword indicator. More specifically, when

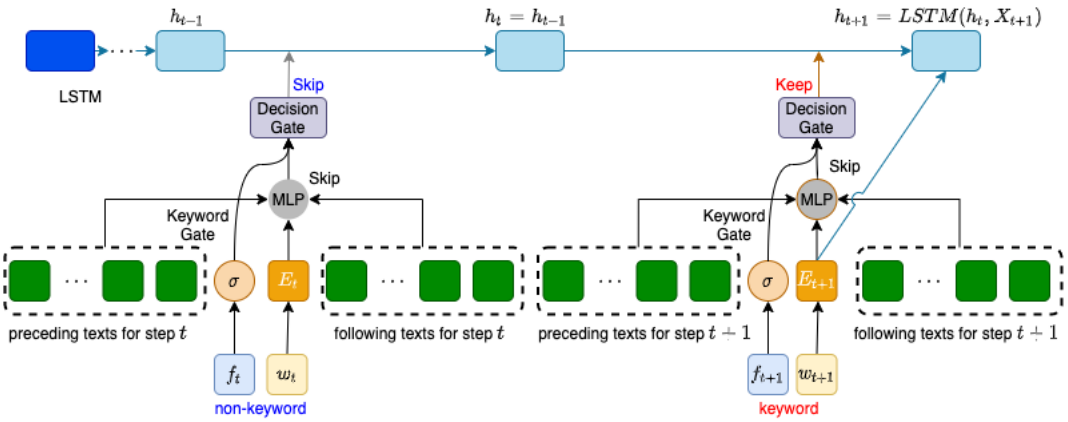

Figure 2: Architecture of Gated KALL

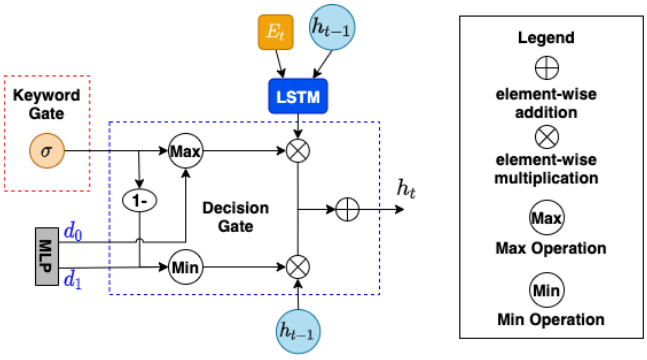

Figure 3: Gating mechanism in gated knowledge integration

a token is indicated as a keyword ($k_{w_t} = 1$), the network should decide to keep the token (i.e. predicting $d_t = 0$ for Equation 3), otherwise the model should be penalised. As a result, we define $\mathcal{L}_k$ as below:

$$\mathcal{L}_k = -\frac{1}{T}\sum_{t=1}^{T} u_t \log P(d_t = 0) \tag{9}$$

$$u_t = \begin{cases} 1 & \texttt{if } d_t = 0 \texttt{ or } k_{w_t} = 1; \\ 0 & \texttt{otherwise} \end{cases} \tag{10}$$

where $P(d_t = 0)$ is the probability of $d_t = 0$ calculated from Equation (2).

The penalisation works as follows: given a keyword input token ($k_{w_t} = 1$), if the network predicts that the token should be kept ($d_t = 0$), the probability of $d_t = 0$ approaches 1 (i.e. the log probability approaches 0), and no penalty is incurred. Alternatively, if the network predicts that $d_t = 1$, the probability of $d_t = 0$ is low, resulting in a large loss. We do not penalise any other scenarios (e.g. when the input token is not indicated as a keyword but the model decides to keep the token) so as to maintain the flexibility of the model to learn and discover useful words for a task.

Factored KALL: Knowledge as Factor in Word Embeddings

To draw further benefit from human knowledge, we integrate keyword information to the skip prediction process.

Our first approach is to represent keyword information as a factor in the word embedding input. More specifically, we define a vector f_t to represent whether w_t is a keyword. While there are many ways to define f_t (e.g. using a binary indicator, a one-hot vector or embedding etc.), we use a multi-hot vector to capture rich keyword information: $f_t = [d_0, d_1, d_2, d_3]$, in which d_0 represents whether w_t is a single keyword or part of a multi-word keyword, while d_1, d_2 and d_3 indicate whether w_t is the start, middle and end of the multi-word keyword, respectively. For example, $f_t = [1, 0, 1, 0]$ indicates that the current word w_t is a keyword, and it is in the middle of a multi-word keyword. Furthermore, in scenarios that keywords have different importance, we can prioritise each keyword by setting f_t to different positive numbers, e.g. > 1 indicates more important, and < 1 represents less important. In our experiments, we set f_t as a multi-hot vector with equal weights.

Figure 1 shows the architecture of factored KALL, where w_t is the word embedding, E_t is the concatenation of f_t and the word embedding w_t. We feed E_t as well as the *preceding* and *following* contexts of w_t to an MLP network to predict whether w_t should be kept or not.

Using knowledge as a factor in word embeddings provides a *soft* signal to skip prediction. There is no guarantee that the model keeps every keyword. Nevertheless, the knowledge-oriented function introduced in Section can alleviate this problem by driving the model to include more keywords through weight optimisation.

Gated KALL: Gated Knowledge Integration

To enforce the model to keep all keywords while maintaining the differentiability of the network, we propose the second approach of integrating keyword information to the skip prediction: a gating mechanism as shown in Figures 2 and 3.

We define two gates: a *Keyword Gate* and a *Decision Gate*. The *Keyword Gate* uses a pre-defined probability to

indicate the degree to which a word is considered as a keyword. Without loss of generality, we apply a *Sigmoid* function to a predefined keyword weight f_t, setting it as a positive integer $\gg 1$ when w_t is a keyword, and a negative number $\ll -1$ otherwise:

$$h_f(t) = \sigma(f_t) = \begin{cases} \approx 1; & f_t \geq C \ \texttt{if} \ w_t \in K; \\ \approx 0; & f_t \leq -C \ \texttt{if} \ w_t \notin K. \end{cases} \quad (11)$$

where $h_f(t)$ is the probability produced by the Keyword Gate; $C \gg 1$ is the minimum absolute value for f_t. In our experiments, we set $f_t = 6$ for all keywords, and $f_t = -6$ for all non-keywords. Different thresholds may be set to different keywords to indicate varied degree of importance.

The *Decision Gate* controls whether a word is kept or not which performs a *soft switch* operation to combine $h_f(t)$ and the probability distribution π_i on skipping versus keeping from Equation (2), resulting in a new update function for the hidden state of LSTM:

$$h_t = \begin{cases} \texttt{LSTM}(h_{t-1}, x_t) & \text{if } d_t = 0 \ \texttt{or} \ h_f(t) >= \theta \\ h_{t-1} & \text{if } d_t = 1 \ \texttt{and} \ h_f(t) < \theta \end{cases} \quad (12)$$

where $d_t = 0$ indicates a decision to keep the token as produced by the decision network, and $d_t = 1$ otherwise; θ is a pre-defined threshold for the Keyword Gate (e.g. 0.5). To make the neural network differentiable, we rewrite the hidden state update function as:

$$\begin{aligned} h_t = & \max(h_f(t), [y_t]_0) \cdot \text{LSTM}(h_{t-1}, x_t) \\ & + \min(1 - h_f(t), [y_t]_1) \cdot h_{t-1} \end{aligned} \quad (13)$$

Figure 3 presents a detailed diagram of the gating mechanism. The gating mechanism guarantees that the model keeps all keywords, and uses the decision network to decide whether to keep or skip a *non-keyword*. To illustrate, when the Keyword Gate indicates that w_t is a *non-keyword* (e.g. as shown in Figure 2), then $h_f(t) \approx 0$ and the decision component suggests to skip the token (i.e. $d_t = 1$), the update function becomes $h_t \approx \min(1 - h_f(t), [y_t]_1) \cdot h_{t-1}$, i.e. the network skips w_t by copying h_{t-1} to h_t. Alternatively, when Keyword Gate signals that w_t should be *kept* (e.g. a *keyword* as shown in Figure 2), even though the decision function suggests to skip the token, the update function becomes $h_t \approx \max(h_f(t), [y_t]_0) \cdot \text{LSTM}(h_{t-1}, x_t)$, i.e. the network keeps the token and feeds it to LSTM to update h_t.

Note that Eq. (13) describes the strategy for updating h_t during training. Since back-propagation is not required during inference, We directly use Eq. (12) for inference.

Experiments

We use four datasets to evaluate KALLs: (1) two large-scale public datasets with automatically generated keywords as "human knowledge", for experiments in a general scenario; (2) two customised datasets with only long sequences and human generated keywords, for experiments in a low-resource scenario.

We evaluate the overall accuracy of the models (i.e. F1 score for multi-class classification), and conduct an ablation test to investigate the performance gain from the knowledge-oriented cost function, the factored and the gated mechanisms. Furthermore, we evaluate the speed-up of KALLs in terms of inference time.

Our baselines include LSTM, standard Leap-LSTM, distilBERT (Sanh et al. 2019) and the SOTA pre-trained model XLNet (Yang et al. 2019). DistilBERT reduces the size of a BERT model by 40%, while retaining 97% of its language understanding capabilities and being 60% faster. XLNet is an extension of the Transformer-XL (Dai et al. 2019) model which introduces a recurrence mechanism without the sequence length limit during the training (nevertheless, the maximum length limit still needs to be set when fine-tuning downstream tasks). All systems are trained on GPUs (AWS g3.16xlarge, NVIDIA Tesla M60), and the inference time is evaluated in a CPU-only industrial scenario.

Datasets

The four datasets are AGNews, DBpedia, Yelp Review Full (Yelp-Full) and Yelp Review Polarity (Yelp-Polarity)[1]. AGNews and DBpedia are the same as in (Huang, Shen, and Deng 2019), containing both short and long sequences, whereas Yelp-Full and Yelp-Polarity are customised to include long sequences to simulate a low-resource scenario. More specifically, we first selecte sequences of more than 100 tokens from the original train, validation and test sets, respectively, and then randomly sample 10% of these sequences respectively from the selected train, validation and test sets to form small datasets. Table 1 shows the statistics of the datasets. *Full Small* and *Polarity Small* are the customised datasets, as opposed to the original ones. *MaxLen* and *AveLen* denote the maximum and average sequence length of a training set. As we can see, the sequences can be very long ($>$ 1000 tokens), with an average length of more than 200 in our constructed datasets.

We use the publicly available Opinion Lexicon[2] (Hu and Liu 2004) as the human knowledge for Yelp-Full and Yelp-Polarity. The opinion lexicon contains 6,800 English positive and negative sentiment/opinion words. While no keywords are generated by humans for AGNews and DBpedia, we illustrate the wide applicability of our models by using automatically generated keywords as "human knowledge". We first use sklearn's tf-idf[3], TextRank (Mihalcea and Tarau 2004) and Rake (Rose et al. 2010) to generate three separate keyword sets with n-gram range of $[1, 3]$ throughout; then we intersect them and obtain 3,006 keywords for AGNews and 3,938 for DBpedia.

Settings and Hyper-parameters

Table 3 shows the common settings of the hyper-parameters across KALLs and LSTM-based baselines. Specific settings for KALLs and the standard learp-LSTM models are listed in Table 4, where λ is the weight of the penalty term for skip prediction in Equations (4) and (8). We set the weight of the knowledge-oriented penalty term as $\beta = 2$ for both factored and gated KALLs[4]. We use "xlnet-base-cased" pre-

[1] https://github.com/zhangxiangxiao/Crepe.

[2] https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#lexicon

[3] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[4] We set $\beta = 4$ when the skip rate is 0.9 for Gated KALL.

Dataset	#Train	#Validation	#Test	MaxLen	AveLen	#Label
Full Original	600,000	50,000	50,000	1,231	156	5
Polarity Original	540,000	20,000	38,000	1,194	154	2
Full Small	67,045	5,558	5,604	1,187	235	5
Polarity Small	59,259	2,189	4,159	1,164	236	2
AGNews	110,000	10,000	76,00	250	45	4
DBpedia	540,000	20,000	70,000	1,500	55	14

Table 1: Statistics of the datasets.

Model	Full Small			Polarity Small			AGNews			DBpedia		
	Settings: Fixed Length											
Standard LSTM	*ALL*	*200*	*100*	*ALL*	*200*	*100*	*ALL*	*100*	*25*	*ALL*	*100*	*25*
	61.63	57.60	51.75	**95.14**	92.67	87.93	93.53	93.46	92.11	99.05	99.01	98.11
	256	*128*	–	*256*	*128*	–	*256*	*128*	–	*256*	*128*	–
XLNet	***64.97***	60.30	–	***96.27***	91.53	–	***94.50***	93.05	–	***99.40***	98.91	–
DistilBERT	***64.63***	–	–	***96.08***	–	–	***94.29***	–	–	***99.25***	–	–
	Settings: Skipping Rate											
	0.25	*0.6*	*0.9*	*0.25*	*0.6*	*0.9*	*0.25*	*0.6*	*0.9*	*0.25*	*0.6*	*0.9*
Standard Leap-LSTM	61.56	59.48	54.08	94.69	93.65	89.06	93.85	93.54	93.04	99.10	99.06	98.84
Factored KALL w/o $\mathcal{L}_k$	61.55	59.90	55.73	94.57	93.77	90.69	–	–	–	–	–	–
Gated KALL w/o $\mathcal{L}_k$	61.67	58.90	53.25	94.73	93.41	89.03	–	–	–	–	–	–
Factored KALL + $\mathcal{L}_k$	61.85	60.64	56.50	94.97	94.21	91.56	93.93	93.64	93.23	99.14	99.09	99.01
Gated KALL + $\mathcal{L}_k$	**62.33**	**60.72**	**57.48**	**95.02**	**94.42**	**92.71**	**94.00**	**93.75**	**93.34**	**99.35**	**99.16**	**99.10**

Table 2: Micro F1 scores of all systems

Parameter	**Value**
optimiser	Adam
dimension of Glove pre-trained word embedding	100
size of hidden states	200
learning rate	0.001
number of layers of LSTM	1
epochs	25
batch size	32
dropout rate	0.4
lowercase	True

Table 3: Common settings of all systems.

Parameter	**Value**
λ	1.0
size of the first CNN filter	[3, 300, 1, 60]
size of the second CNN filter	[4, 300, 1, 60]
size of the third CNN filter	[5, 300, 1, 60]
size of small LSTM for global feature	20
size of MLP for decision making	20
dropout of small LSTM for global feature	0.5
temperature τ for gumbel-softmax	0.1

Table 4: Settings of Leap-LSTM and KALL

trained model for XLNet and "distilbert-base-uncased" for distilBERT. The batch size of fine-tuning XLNet is set to 8 and 4 when the maximum length limit is 128 and 256, respectively; the batch size of distilBERT is set to 10. All other hyper-parameters are kept as default.

In the implementation of standard Leap-LSTM[5] and our KALL models, the states of small LSTM and CNNs are pre-computed and cached after each data batch is fed into the model; and the main LSTM retrieves relevant states from the cache at each time step for skip decision making.

Model Accuracy

Table 2 shows the performance of KALLs and the baselines at different input lengths. ALL means that LSTM reads full sequences. We use the commonly-used cut-off lengths 200 and 100 for Standard LSTM; To speed up the transformers, we use half (256) and a quarter (128) of the original maximum length 512 for XLNet and distilBERT. The skipping rates of all Leap-LSTM systems are set to 0.25, 0.6 and 0.9. All models are evaluated using Micro F1 score.

As we can see from Table 2, the two KALL models with the knowledge-oriented loss function $\mathcal{L}_k$ perform the best on all LSTM-based systems but one scenario: the standard LSTM with the full length input on the Polarity Small dataset[6]. Specifically, the gated KALL + $\mathcal{L}_k$ achieves best overall F1 scores. The performance gain is especially large when the skipping rate is high.

Our ablation tests show that the two KALL models without $\mathcal{L}_k$ perform differently under different skipping rates. Factored KALL w/o $\mathcal{L}_k$ outperforms standard Leap-LSTM, showing that the factored method as a *soft* integration of knowledge gives the decision network more flexibility to de-

[5]The authors of Leap-LSTM released their codes at: https://github.com/ht1221/leap-lstm in Tensorflow, and we re-implemented it with Pytorch.

[6]Nevertheless, for this scenario, when we lowered the skipping rate to 0.15, gated KALL achieved a higher score.

Model	Full Small		Polarity Small		AGNews		DBpedia	
	0.25/L	0.9/L	0.25/L	0.9/L	0.25/L	0.9/L	0.25/L	0.9/L
Leap-LSTM	24.33/179	83.50/39	27.21/171	90.24/23	26.04/33	86.00/6	25.88/41	87.38/7
Factored KALL	23.03/182	83.19/40	28.31/169	84.07/37	25.03/33	85.42/7	25.32/41	86.24/8
Gated KALL	25.45/176	80.0/47	23.81/179	81.25/44	23.94/34	83.67/7	24.50/42	84.36/9

Table 5: Actual skipping rates and average sequence length of Leap-LSTM and KALL

Model	Full Small			Polarity Small			AGNews			DBpedia		
	0.25	*0.6*	*0.9*	*0.25*	*0.6*	*0.9*	*0.25*	*0.6*	*0.9*	*0.25*	*0.6*	*0.9*
Leap-LSTM	1.4x	1.6x	1.9x	1.8x	2.1x	2.7x	1.1x	1.5x	2.3x	1.2x	1.7x	2.8x
Factored Knowledge-aware	1.4x	1.6x	1.9x	1.8x	2.1x	2.5x	1.1x	1.4x	2.3x	1.2x	1.7x	2.6x
Gated Knowledge-aware	1.3x	1.6x	1.9x	1.7x	2.1x	2.3x	1.1x	1.4x	2.3x	1.2x	1.7x	2.5x

Table 6: Inference speed-up relative to standard LSTM.

cide a keyword to keep or skip. By contrast, Gated KALL w/o $\mathcal{L}_k$ performs worse when the skipping rates are 0.6 and 0.9. We argue that forcing the network to keep the keyword without adjusting the decision network to learn from the knowledge will deteriorate system performance, especially when skipping more words.

Furthermore, the addition of the knowledge-oriented cost function improves the performance of KALLs, showing that the cost function contributes to the gain of KALLs by better optimising the parameters, especially for Gated KALL where the loss function helps to adapt the decision network to be consistent with the enforcement of keywords.

XLNet (256) and distilBERT (256)[7] achieve better results on all datasets compared to our KALLs. However, when the maximum length limit is 128, the F1 scores of XLNet dramatically decrease and are significantly worse than our Gated KALLs when the skipping rate is 0.6. We argue that in practical scenarios where resource is limited or low latency is required, fast LSTM-based models which can achieve comparable performance can be more preferable than cumbersome transformers to deploy. Furthermore, our skipping model, which can scan any long sequence and keep important words while reducing the information loss, does not suffer from the inherent maximum input length limitation which hamstrings deep architectures. Our results show that knowledge-aware skipping models can balance accuracy/speed trade-off better than existing selective-RNNs without excessive architectural modifications.

Skipping Rates and Inference Speed

Table 5 shows the actual skipping rates of the three Leap-LSTM models and the average sequence length L under each actual skipping rate on the test set. As we can see, when the skipping rate is 0.9, on average, KALLs select fewer than 50 tokens from an input sequence of the two Yelp datasets, and fewer than 10 tokens from an input sequence of AGNews and DBpedia. These models significantly outperform the standard LSTM which uses a fixed-length input of 100 tokens on Yelp data and 25 tokens on AGNews and DBpedia. This shows that for an NLP task with long sequences, simple truncation strategy introduces a large risk of deteriorating model performance, and KALLs are effective in mitigating that risk.

Table 6 shows the speed-up of inference time of KALLs and Leap-LSTM relative to the standard LSTM. The standard Leap-LSTM is 1.1x$\sim$2.7x faster than the standard LSTM across various skipping rates. The two KALL models are slower than the standard Leap-LSTM in some cases, meaning our knowledge integration has traded off some inference speed for model performance. Nevertheless, KALLs are still 1.1x$\sim$2.6x faster than the standard LSTM.

Though XLNet and distilBERT performs better on many NLP tasks, they are impractical for many industry applications due to its high resource consumption and low speed. We take the Full Small data set as an example: on our CPU-only platform, XLNet (256) uses 802 seconds on the test set of Full Small, distilBERT (256) takes 337 seconds, while our Gated KALL ($r = 0.9$) uses only 34 seconds, which is 23.6x and 9.9x faster than XLNet (256) and distilBERT (256), respectively. Even on the GPU server, XLNet (256) takes 371 seconds and distilBERT (256) uses 60 seconds, which are respectively 10.9x and 1.76x slower than our Gated KALL ($r = 0.9$). We obtained similar results in terms of speed comparison on other data sets.

Conclusions

This paper proposes a novel framework (KALL) to integrate human knowledge of keywords into the skipping process of Leap-LSTM, a SOTA selective RNN which is fast in processing long sequences. We design two integration strategies: the factored KALL which integrates a keyword vector as a factor in word embedding, and the gated KALL which uses a gated mechanism to enforce the model to keep the keywords. We also propose a knowledge-oriented cost function which can better optimise the parameters in the decision network for predicting whether to keep or skip the word. Our evaluation on four public datasets shows that: (1) KALLs significantly outperform standard LSTM and Leap-LSTM in accuracy, while maintaining a fast processing speed; (2) KALLs are not only faster, but also achieve comparable performance in a resource-limited circumstance compared to XLNet and distilBERT.

[7] 256 is the maximum input length for XLNet and distilBERT.

Acknowledgements

The authors would like to thank the reviewers for their insightful comments, Guruprasad Sethurathinam for infrastructure support, and internal AIG Investments AI reviewers for their feedback.

References

Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations (ICLR)*.

Bengio, Y.; Léonard, N.; and Courville, A. C. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *CoRR* abs/1308.3432.

Campos, V.; Jou, B.; Gir'o i Nieto, X.; Torres, J.; and Chang, S. 2017. Skip RNN: Learning to Skip State Updates in Recurrent Neural Networks. *CoRR* abs/1708.06834.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*.

Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J.; Le, Q.; and Salakhutdinov, R. 2019. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Du, J.; Huang, Y.; and Moilanen, K. 2020. Pointing to Select: A Fast Pointer-LSTM for Long Text Classification. In *Proceedings of the 28th International Conference on Computational Linguistics*, 6184–6193. Barcelona, Spain (Online).

Hahn, M.; and Keller, F. 2016. Modeling Human Reading with Neural Attention. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 85–95.

Han, J.; Barman, U.; Hayes, J.; Du, J.; Burgin, E.; and Wan, D. 2018. NextGen AML: Distributed Deep Learning based Language Technologies to Augment Anti Money Laundering Investigation. In *Proceedings of ACL 2018, System Demonstrations*, 37–42.

Hansen, C.; Hansen, C.; Alstrup, S.; Simonsen, J. G.; and Lioma, C. 2019. Neural Speed Reading with Structural-Jump-LSTM. In *7th International Conference on Learning Representations (ICLR)*.

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

Hu, M.; and Liu, B. 2004. Mining and summarizing customer reviews. In *KDD*, 168–177.

Huang, T.; Shen, G.; and Deng, Z. 2019. Leap-LSTM: Enhancing Long Short-Term Memory for Text Categorization. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 5017–5023.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparameterization with Gumbel-Softmax. In *The 5th International Conference on Learning Representations (ICLR)*.

Liu, P.; Qiu, X.; Chen, X.; Wu, S.; and Huang, X. 2015. Multi-Timescale Long Short-Term Memory Neural Network for Modelling Sentences and Documents. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2326–2335. Lisbon, Portugal.

Liu, X.; Mou, L.; Cui, H.; Lu, Z.; and Song, S. 2018. Jumper: Learning When to Make Classification Decision in Reading. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, 4237–4243.

Mihalcea, R.; and Tarau, P. 2004. TextRank: Bringing Order into Text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, 404–411.

Miotto, R.; Wang, F.; Wang, S.; Jiang, X.; and Dudley, J. T. 2018. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics* 19: 1236–1246.

Rose, S.; Engel, D.; Cramer, N.; and Cowley, W. 2010. Automatic Keyword Extraction from Individual Documents. *Text Mining: Applications and Theory* 1 – 20.

Sanh, V.; Debut, L.; Chaumond, J.; and Wolf, T. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *The 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS 2019*.

Seo, M. J.; Min, S.; Farhadi, A.; and Hajishirzi, H. 2017. Neural Speed Reading via Skim-RNN. *CoRR* abs/1711.02085.

Tan, M.; Xiang, B.; and Zhou, B. 2015. LSTM-based Deep Learning Models for non-factoid answer selection. *CoRR* abs/1511.04108.

Williams, R. J. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* 8: 229–256.

Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J.; Salakhutdinov, R. R.; and Le, Q. V. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Advances in Neural Information Processing Systems 32*, 5753–5763.

Yu, A. W.; Lee, H.; and Le, Q. V. 2017. Learning to Skim Text. *CoRR* abs/1704.06877.