ROBUST TEXT CLASSIFIER ON TEST-TIME BUDGETS

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we design a generic framework for learning a robust text classification model that achieves accuracy comparable to standard full models under test-time budget constraints. We take a different approach from existing methods and learn to dynamically delete a large fraction of unimportant words by a low-complexity selector such that the high-complexity classifier only needs to process a small fraction of important words. In addition, we propose a new data aggregation method to train the classifier, allowing it to make accurate predictions even on fragmented sequence of words. Our end-to-end method achieves state-of-the-art performance while its computational complexity scales linearly with the small fraction of important words in the whole corpus. Besides, a single deep neural network classifier trained by our framework can be dynamically tuned to different budget levels at inference time.

1 INTRODUCTION

Recent advances in deep neural networks (DNN) has improved the performance of natural language processing tasks such as document classification, question answering, and sentiment analysis (Wu et al., 2017; Seo et al., 2016; Socher et al., 2011; Yu et al., 2017). These approaches process the entire text and construct representations of words and phrases in order to perform target tasks. While these models do realize high accuracy, their computational-time scales linearly with the size of the documents, which can be slow for documents containing many sentences. In this context, various approaches based on modifying the existing RNN or LSTM architecture have been proposed Seo et al. (2017); Yu et al. (2017) to speed-up processing. However, processing is still fundamentally sequential, which in turn requires loading entire documents to process, limiting compute gains.

Contributions: We propose a novel test-time prediction method for efficient text classification on long documents that mitigates sequential processing as seen in Fig. 1. Our method is a general framework consisting of a *selector* and a *classifier*. The *selector* performs a coarse one-shot selection deleting unimportant words and choosing important words in the input document. The collection of fragmented sentences is input into the *classifier*, which then performs the target task.

The problem is challenging due to competing goals and requires joint training of *selector* and *classifier* functions. First, *selector* must have negligible overhead while being compatible with the terminal classification task, since uncontrolled word-deletions cannot be handled during classification. We adopt an architecture that integrates dual embeddings, one based on word-embeddings and the other based on bag-of-words. Second, the challenge encountered by the *classifier* is that its input is a sequence of fractured sentences that is incompatible with standard RNN/LSTM inputs and when used without modification leads to significant performance degradation. One potential solution is to train classifiers with a diverse collection of sentence fragments but this is not meaningful since there are combinatorially many possibilities. A different approach rooted in so-called "blanking-noise," that randomly blanks out text, leads to marginalized feature distortion (Maaten et al., 2013) but this also leads to poor accuracy. This is because DNNs leverage word combinations and word sequences, which the marginalized distortion approach does not account for. We propose a data aggregation framework (DAG) that augments the training corpus with outputs from *selector*s at different budget levels. By training the *classifier* on the aggregated *structured* blank-out text, the *classifier* learns to fuse fragmented sentences into a feature representation that mirrors the representation obtained on full sentences and thus realizes high-accuracy. We show the effectiveness of the proposed approach through comprehensive experiments on real-world datasets.

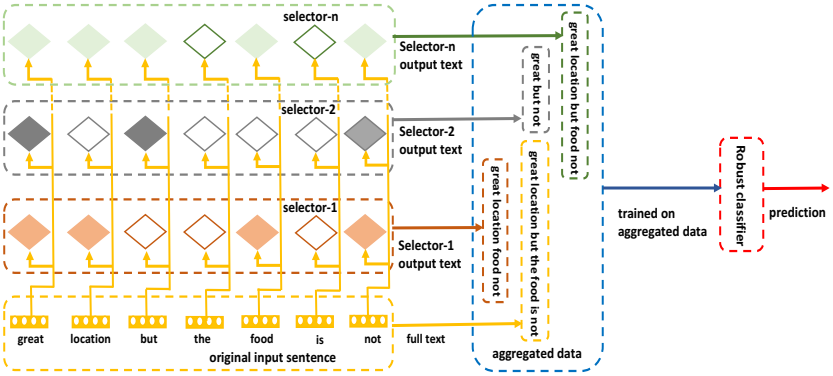


Figure 1: An illustration of the proposed framework. A *selector* is designed to select words that are relevant to the target task. These words are input into the *classifier* for processing. We aggregate output text from different *selector*s and train the *classifier* on the aggregated data.

2 Related Work

Fast Reading Text: Recent works have proposed test-time speed-up for DNNs. Wu et al. (2017) and Choi et al. (2017) propose CNN based approaches to speed up question answering. Of particular relevance are LSTM-jump (Yu et al., 2017) and skim-RNN (Seo et al., 2017), which are based on modifying existing RNN/LSTM architectures. LSTM-jump learns to completely skip words deemed to be irrelevant and a variant, skim-RNN, uses a low-complexity LSTM to skim words rather than skipping. In contrast, we adopt existing *classifier* architectures but modify the training method.

Interpretability of Neural Networks: Our framework resembles Lei et al. (2016), who propose to find snippets of input-text to serve as justification (rationales) for text classification. Their framework also consists of a *selector* in cascade with a *classifier*. However, in their proposed embodiment, both of these modules have similar complexity and in turn, require similar processing times during run-time. In contrast, our goal is speed-up and we show that a simple *selector* works as well as a complex one as long as the *classifier* can account for fragmented text.

Feature Selection in Text Classification: While text preprocessing such as stop-word removal is conventional, they require pre-defined word lists and are not learned in conjunction with targeted tasks. Various feature selection approaches (Chandrashekar & Sahin, 2014) have been discussed in the literature. The most relevant to ours is to employ lasso (Tibshirani, 1996) or group lasso (Faruqui et al., 2015) for learning sparse features. Different from these approaches, we directly learn a *selector* along with the *classifier*. Besides, our *selector* chooses salient words of an instance (long sentence). These words serve as input to a *classifier* (e.g., LSTM). This is very different from feature subspace selection methods, such as PCA or other dimensionality reduction methods, that map an instance into low dimension space as this representation is not aligned with required LSTM input.

Data Aggregation: Aggregating data or models to improve the performance of a *classifier* has been studied under various contexts. Bagging (Breiman, 1996) has been proposed to aggregate models learned from different set of training samples. Here, we aggregate the output from *selector* instead of models. Similar to us the DAGGER algorithm (Ross et al., 2010) has been proposed to account for distorted inputs in reinforcement learning and imitation learning. DAGGER is iterative; at each iteration, it updates its policy by training a *classifier* in a different reinforcement learning context. In contrast, our blank-out datasets originate from the given training data and we aggregate these datasets only once, as a means to obtain a rich collection of fragmented sentences.

Budgeted Learning: The literature on budgeted learning is vast but much of it focuses on a different set of applications and problems than ours (see Viola & Jones (2001); Karayev et al. (2013); Xu et al. (2013); Trapeznikov & Saligrama (2013); Strubell et al. (2015); Weiss & Taskar (2013); He et al. (2013). Of relevance are methods for speed-up in DNN architectures Bengio et al. (2015); Leroux et al. (2017); Lin et al. (2017); Bolukbasi et al. (2017). Different from our method, those methods focus on gating different layers of an existing DNN towards conditional computation.

Algorithm 1: Data Aggregated Training Schema

Input: Training corpus $\mathcal{X}$, a set of *selector*s $\mathcal{S} = \{S_b\}$, *classifier* class C
Output: A robust *classifier* $C_{\mathcal{T}}$

Initialize the aggregated corpus: $\mathcal{T} \leftarrow \mathcal{X}$
for $S_b \in \mathcal{S}$ **do**
  $S_b \leftarrow$ Train a *selector* $S_b \in S$ with budget level b on $\mathcal{X}$ (see Sec 3.2)
  Generate a blank-out dataset $\mathcal{I}(\mathcal{X}, S_b)$
  Aggregate data: $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{I}(\mathcal{X}, S_b)$
$C_{\mathcal{T}} \leftarrow$ Train a *classifier* C on aggregated data $\mathcal{T}$
return $C_{\mathcal{T}}$

3 TEXT CLASSIFICATION ON A TEST-TIME BUDGET

Our goal is to build a robust *classifier* along with a suite of selectors to achieve good performance under test-time budgets. Formally, a *classifier* $C(\hat{x})$ takes a sequence of words $\hat{x}$ as input and predicts the corresponding output label y, and a *selector* $S_b(x)$ with test-time budget b takes an input word sequence $x = \{w_1, w_2, \ldots, w_N\}$ and generates a binary sequence $S_b(x) = \{z_{w_1}, z_{w_2}, \ldots, z_{w_N}\}$ where $z_{w_k} \in \{0, 1\}$ representing if the corresponding word w_k is selected or not. We denote the sub-sequence of words generated by the *selector* as $I\big(x, S_b(x)\big) = \{w_k : z_{w_k} = 1, \forall w_k \in x\}$. Our framework aims to train a *classifier* C and *selector*s S_b such that $I\big(x, S_b(x)\big)$ is sufficient to make accurate prediction on the output label (i.e., $C\Big(I\big(x, S_b(x)\big)\Big) \approx C(x)$). Here, the test-time budget b can be viewed as a hyper-parameter of the *selector* to control the trade-off between test-time speed and accuracy. Note that in contrast to some existing frameworks (e.g., Yu et al. (2017)), we build a single *classifier* for different budgets. This design choice is due to a practical reason. The learned parameters of a *classifier* is often much larger than of a *selector* (e.g., the number of parameters in one of the *classifier*s used in our experiment is more than 88 million, while the size of the *selector* is 300). As a result, storing different *classifier*s for different budgets is impractical.

Our learning framework is designed to overcome two main challenges: 1) how to train a *classifier* C such that it can work with *selector*s S_b with different budget levels and different architectures? 2) How to train a *selector* without explicit annotations about which words should be selected? For the former, we propose a data aggregation framework (DAG) to augment blank-out outputs $I(x, S_b(x))$ from different *selector*s and trained the *classifier* C on the aggregated data. For the latter, we train the selectors by leveraging the feedback from task labels. We discuss details below.

3.1 THE DATA AGGREGATION FRAMEWORK

For the ease of discussion, given a set of training data $\mathcal{X} = \{(x_1, y_1), .., (x_t, y_t), .., (x_m, y_m)\}$, we assume we have a set of selectors $\mathcal{S} = \{S_b\}$ with different budget levels. We will discuss how to obtain these selectors in Section 3.2. To generate an aggregated corpus, we first apply each *selector* $S_b \in \mathcal{S}$ on the training set, and generate corresponding blank-out corpus $\mathcal{I}(\mathcal{X}, S_b) = \big\{I\big(x_t, S_b(x_t)\big), \forall x_t \in \mathcal{X}\big\}$. Then, we create an new corpus by aggregating blank-out corpora with different budget level: $\mathcal{T} = \bigcup_{S_b \in \mathcal{S}} \mathcal{I}(\mathcal{X}, S_b)$. Finally, we train the *classifier* $C_{\mathcal{T}}$ on the aggregated corpus $\mathcal{T}$. As $C_{\mathcal{T}}$ is trained on documents with distortions, it learns to make predictions with different budget levels. The data aggregation training framework is summarized in Algorithm 1.

In the following, we discuss two extensions of the data aggregation framework. First, the blank-out data can be generated from different classes of selectors with different features or architectures. In practice, we observed that by aggregating selections from multiple *selector*s, the trained *classifier* $C_{\mathcal{T}}$ is more robust, leading to higher accuracy. Second, in the above discussion, we filter out unimportant words by *selector*s and aggregate the resulting corpora (we call it word-level aggregation (WAG)). However, the blank-out and selection can be done in phrase or sentence level. Specifically, if phrase boundaries are provided, we can leverage this information and design a phrase-level aggregation (PAG) to avoid a *selector* from breaking compound nouns or meaningful phrases (e.g., "New York", "not so bad"). Similarly, for documents consisting of many short sentences, we can enforce

the *selector* to pick the whole sentence if any word in the sentence is selected. In this way, we can design a sentence-level aggregation (SAG) to better capture long phrases.

3.2 LEARNING A *Selector*

A *selector* in our framework should satisfy the following criteria. First, as our goal is to reduce overall test time, the *selector* has to be computationally efficient. Second, the selected words have to be informative for the *classifier* to achieve similar performance using the selected words as the original input. Several existing works (e.g., Lei et al. (2016)) do not satisfy both conditions. For example, Lei et al. (2016) proposed a framework to jointly learn a *selector* with a *classifier*, where they consider the *selector* has the same complexity as the *classifier* as both of components are implemented with RCNN architecture. As a result, the time complexity of running a RCNN *selector* is as high as the *classifier*; therefore, it is not suitable to be used in our framework. In the following, we consider two classes of selectors: 1) a *selector* with word embedding features trained jointly with the *classifier* by a doubly gradient descent method, and 2) a *selector* trained by a L1-regularized logistic regression with bag-of-words features.

Word Embedding (WE) *selector*. To achieve overall speedup gains, we consider a parsimonious word-selector using word embeddings (e.g., GloVe Pennington et al. (2014)) as features to predict if a word should be passed to the *classifier*. Intuitively, word embedding preserves the word semantics. Therefore, for semantic-oriented tasks, word embedding is suitable to identify informative words for predicting target labels.

Formally, for each instance $x = (w_1, w_2, \ldots, w_N)$, the WE *selector* outputs a binary vector z, where z_{w_k} is associated with word w_k. Let $\vec{w}_k \in R^d$ be a word vector of word w_k, where d is the dimension of the word embedding. We assume the informative words can be identified independently by word embedding and consider modeling the probability that a word w_k is selected by

$$P(z_{w_k}|w_k) = \text{sigmoid}(z_{w_k}\theta^T\vec{w}_k) = \frac{1}{1+\exp(-z_{w_k}\theta^T\vec{w}_k)}, \tag{1}$$

where $\theta_S \in R^n$ is the model parameters of the *selector* S_b. Then, the selection of the entire document $x = \{w_1, w_2, \ldots, w_N\}$ is

$$P\big(S_b(x)|x\big) = \prod_{k=1}^{N} P(z_{w_k}|\vec{w}_k).$$

Because we do not have explicit annotations about which words are important, directly optimizing S_b is unfeasible. Instead, we train the *selector* S_b with a *classifier* C. We denote the model parameters of the *classifier* as θ_C. Given a training data $(x_t, y_t) \in \mathcal{X}$ ($\mathcal{X}$ is the training set), the *classifier* C makes predictions based on a word sequences sampled from the *selector* (i.e., $z_t \sim P\big(S_b(x_t)|x_t\big)$). For classification problems, we minimizing the negative log-likelihood (i.e., cross-entropy loss) $l(C, y_t, I(x_t, z_t)) = -\log P_C(y_t; I(x_t, z_t))$, where P_C is the probability distribution over candidate labels predicted by the *classifier* C. For regression problem, we minimizing the squared loss based on L2 distance: $l(C, y_t, I(x_t, z_t)) = \|y_t - C(I(x_t, z_t))\|_2^2$. As in Lei et al. (2016), we consider two ℓ_1-regularizers to promote sparsity and continuity of selections, respectively,

$$\phi(z) = \lambda_1\|z\|_1 + \lambda_2\sum\nolimits_{k=1}^{n}|z_{w_k} - z_{w_{k-1}}|,$$

where λ_1 and λ_2 are hyper-parameters (a.k.a. budget level) and solve the overall objective

$$\min_{\theta_S,\theta_C} E_{x_t,y_t\sim\mathcal{X}}E_{z_t\sim P(S_b(x_t)|x_t)}\Bigg[l(C, y_t, I(x_t, z_t)) + \phi(z_t)\Bigg],$$

by doubly stochastic gradient descent.

Bag-of-Words *selector*. We also consider a traditional approach to use an ℓ_1-regularized linear model (Zou & Hastie, 2005; Ng, 2004; Yuan et al., 2010) with bag-of-word features to identify important words necessary for a target task. To build intuition, consider binary classification with output labels $y \in \{1, -1\}$. In the bag-of-word model, for each document x, we construct a feature vector $\vec{x} \in \{0, 1\}^{|V|}$, where $|V|$ is the size of the vocabulary. Each element of the feature vector

$\vec{x}_w$ represents if a specific word w appear in the document x. Given a training data set $\mathcal{X}$, the ℓ_1-regularized logistic regression model optimizes

$$\theta^* = \arg\min_{\theta} \sum\nolimits_{(x_t, y_t) \in \mathcal{X}} \log(1 + \exp(-y_t \theta^T \vec{x_t})) + b\|\theta\|_1,$$

where $\theta \in R^{|V|}$ is a weight vector to be learned, θ_w corresponds to word $w \in V$, and b is a hyper-parameter (i.e., selection budget).[1] Based on the optimal solution θ^*, we construct a *selector* that picks word w if the corresponding θ^*_w is non-zero. That is the Bag-of-Words *selector* output, $S_b(x) = \{\delta(\theta_w \neq 0) : w \in x\}$, where δ is an indicator function.

4 EXPERIMENTS

In this section, we evaluate the proposed approach on five real-world text classification datasets. We first compare the proposed approach with existing budget learning methods, then we conduct comprehensive analyses.

Experimental Setup We consider five datasets in the experiments. The statistics of the datasets are summarized in Table 5 in the appendix. Stanford Sentiment Treebank (**SST-2**) (Socher et al., 2013) is a binary classification problem in sentiment analysis. The dataset contains annotations of sentiment labels for entire sentences and phrases. **IMDB** is described in Maas et al. (2011). Each instance in the dataset is a paragraph of a movie review. **Multi-Aspect** is collected by Lei et al. (2016). For this dataset, we use word embeddings provided with the dataset and apply both RCNN and WE *selector* to aggregate the data. To compare our model with other approaches, we follow Lei et al. (2016) to model this problem as a regression problem and use mean square error (MSE) as the evaluation metric. **AGNews:** We collect the dataset from a public repository[2] Zhang et al. (2015). Each instance consists of a title and a small paragraph. **Yelp** is used in Conneau et al. (2016). Each instance is a short paragraph of a restaurant review. For all tasks, we apply the word-level aggregation (WAG). For datasets consisting of documents with multiple sentences (YELP, IMDB) or have phrase boundary annotations, we also consider sentence-aggregation (SAG) and phrase-aggregation (PAG) schemes.

Our framework is generic and can leverage different types of *classifiers*. Therefore, we evaluate our framework with the following two neural network architectures: **Biattentive Classification Network (BCN):** BCN is a generic text classification model (McCann et al., 2017). It comprises Bi-LSTM, Bi-attention, and Maxout networks. BCN provides a strong baseline on many datasets, including SQuAD, SST, IMDB, and several others. We use the implementation in AllenNLP (`https://allennlp.org/`). **LSTM:** LSTM model is widely used for text classification (Zhang et al., 2015; Seo et al., 2016). LSTM sequentially reads words in a passage and updates its hidden state to capture features from the text. Both LSTM-jump and Skim-RNN are built upon LSTM. Besides evaluating our framework with the BCN and the LSTM classifiers, we also analyze the data aggregation framework with **Recurrent Convolution Neural Network (RCNN)**. RCNN is a refined local n-gram convolutional neural network model. The recurrent part learns the average features in a dynamic fashion and the convolution part learns the n-gram features that are not necessarily contiguous. For *selector*s, we consider both selectors discussed in Sec. 3.2. To demonstrate the speed and quality of the selectors, we compare them with an RCNN *selector* used in Lei et al. (2016). By default, we use WAG selection scheme and aggregate the data using different WE *selector*s with budgets (i.e., fraction of text to select) $\{0.5,0.6,..,1.0\}$ (See Section 3.1, 3.2), Glove (Pennington et al., 2014) word embeddings) and evaluate in terms of accuracy or error (error = 1 - accuracy) unless stated otherwise. [3]

4.1 EVALUATION

First, we compare our framework with the following approaches: 1) *Baseline*: the original classifier. 2) *LSTM-jump* Yu et al. (2017): accelerating LSTM inference by skipping words. 3) *Skim-RNN* Seo et al. (2017): applying a low-complexity LSTM to model unimportant words. Seo et al. (2017) report

[1] In our experiment, we use the implementation in scikit-learn.

[2] `https://github.com/mhjabreel/CharCNN/tree/master/data/ag_news_csv`

[3] For more details on experimental settings, See Appendix.

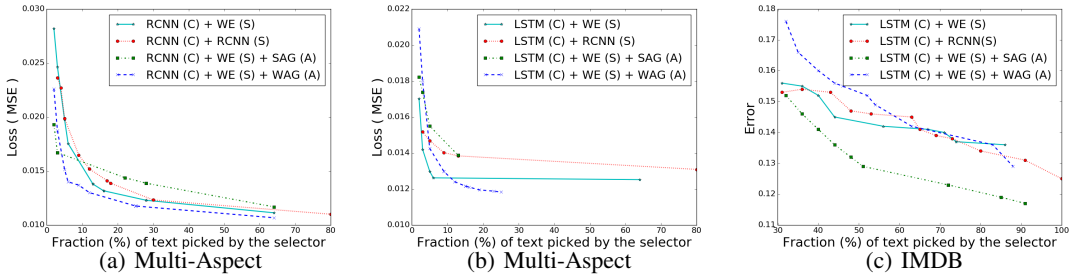
Model	SST-2		IMDB		AGNews		Yelp	
	Acc(%)	speedup	Acc (%)	speedup	Acc (%)	speedup	Acc (%)	speedup
LSTM Classifier								
LSTM-jump	-	-	89.4 (+0.3)	1.6x	89.3 (+1)	1.1x	-	-
skim-RNN-1	85.6 (-0.8)	1.5x	88.7 (-2.4)	1.5x	**93.3** (-0.2)	1x	-	-
skim-RNN-2	**86.4** (0)	**1.7x**	90.9 (-0.2)	**2.7x**	92.5 (-1)	0.8x	-	-
Baseline	85.2	1x	92.0	1x	92.8	1x	**66.7**	1x
Stop-words	84.0 (-1.2)	1x	91.1 (-0.9)	1.8x	92.7 (-0.1)	1x	64.3 (-2.4)	**1.7x**
Bag-of-Words	82.2 (-3.0)	1.3x	89.7 (-2.3)	1.2x	90.1 (-2.7)	1.2x	60.6 (-6.1)	1.5x
Our framework	82.9 (-2.3)	1.3x	91.1 (-0.9)	1.2x	92.3 (-0.5)	1.2x	64.5 (-2.2)	1.5x
	86.4 (+1.2)	1x	**92.1** (+0.1)	1x	92.9 (+0.1)	1x	66.4 (-0.3)	1x
BCN Classifier								
Baseline	85.7	1x	91.0	1x	92.3	1x	66.5	1x
Stop-words	82.2 (-3.5)	1x	91.2 (+0.2)	1.8x	92.8 (+0.5)	1x	64.7 (-1.8)	**1.7x**
Bag-of-Words	78.8 (-6.9)	1.7x	90.4 (-0.7)	1.2x	91.7 (+0.7)	**1.3x**	59.7 (-6.8)	1.6x
Our framework	82.6 (-3.1)	**2x**	92.0 (+1)	1.2x	93.1 (+0.8)	**1.3x**	64.8 (-1.7)	1.6x
	85.3 (-0.4)	1x	**92.1** (+1.1)	1x	93.2 (+0.9)	1x	66.3 (-0.2)	1x

Table 1: Test performance and overall speedup on the Test set. Our reported speedup refers to full pipeline (selection + classification) test-time in comparison with the baseline *classifier*. LSTM-jump and Skim-RNN use a different baseline *classifier*, and we report the difference in accuracy in parentheses. We report the best result of models tuned on the Dev set. All results are the average of 3 runs. For each *classifier* in the table, our framework has two rows of results. First one (top row) denotes the best speedup performance and the second one (bottom row) denotes the best text accuracy achieved by our framework. Best performance and best speed-up are boldfaced.

results with four different parameter settings. As we do not have access to the performance of their model on dev set, we cannot perform model selections. Therefore, we report two of the best results shown in their paper. 4) Stop-Words: We filter out stop-words by the list of stop-words provided by NLTK (`https://www.nltk.org/`). This approach is widely used as a prepossessing step and is viewed as a naive baseline. 5) Bag-of-Words: Filter words by the Bag-of-Word selector in Sec 3.2 and feed the fragments of sentences to the original classifier. This approach has been considered in the context of linear models (e.g., Chang & Lin (2008)).

We conduct experiments on all datasets except Multi-Aspect, as we do not have performances of LSTM-jump and Skim-RNN on it. We will use Multi-Aspect to analyze the proposed approach and compare with Lei et al. (2016). We evaluate our framework with two widely used text classification models, LSTM, and BCN. As both Skim-RNN and LSTM-jump are designed specifically for accelerating the LSTM model, we only compare them with our model with LSTM *classifier*. Besides both of these models built upon LSTM with slighlty different baseline accuracy. To make the comaprison fair, we also report the difference in accuracy with respect to each of their baseline model.

The accuracy and speed-up of all the methods are shown in Table 1. The results show that our framework achieves competitive performance to both LSTM-jump, and skim-RNN. In particular, despite skim-RNN performs well in SST-2 and IMDB, it is unstable and is hard to control the trade-off between performance and the test-time budget. For example, Skim-RNN-2 is slower than the baseline method with significant accuracy drops. In contrast, our model is more stable and achieves reasonable performance under different budgets (details will be demonstrated in Sec. 4.2). Besides, our model allows to naturally incorporate fine-grained annotations in word and phrase levels. For example, if we leverage the sentiment annotations for phrases in SST-2, our model achieves 86.4 with 1.3x speedup for LSTM and 86.7 with 1.7x speedup for BCN. Although Stop-words achieves notable speedup, it sometimes comes with a significant performance drop. This is due to the Stop-words used for filtering text are not learned with the class labels; therefore, some meaningful words (e.g., "but") are filtered out even if they play a very significant role in determining the polarity of the full sentence. Besides, we are not able to control the budget in the Stop-words approach. Compared to Bag-of-Words, our framework achieves better performance, highlighting the fact that the issue of *classifier* incompatibility is real. By training *classifier* with the proposed aggregation framework, the model is robust to the distortions and achieves better performance. Finally, we observed that the classifiers trained with data aggregation improves both the baselines with LSTM and BCN on full-text. By aggregating fragments picked by selectors, the model can put more emphasis on important words and be more robust to the noise in the input document.



(a) Multi-Aspect (b) Multi-Aspect (c) IMDB

Figure 2: The performance versus the fraction of selected words on Multi-Aspect and IMDB datasets. We present results using RCNN and LSTM *classifier* and varying the sparsity, and coherent hyper-parameters (see Equation 3.2) of the corresponding *selector*. (C), (S), and (A) denote *classifier*, *selector* and data aggregation scheme. Results demonstrate that with the data aggregation framework (SAG/WAG), a simple WE *selector* is competitive with a complex RCNN *selector*.

4.2 Analysis

In the rest of this section, we provide comprehensive analyses. To compare with (Lei et al., 2016), we conduct experiments on Multi-Aspect and IMDB, but conclusions are similarly on other datasets.

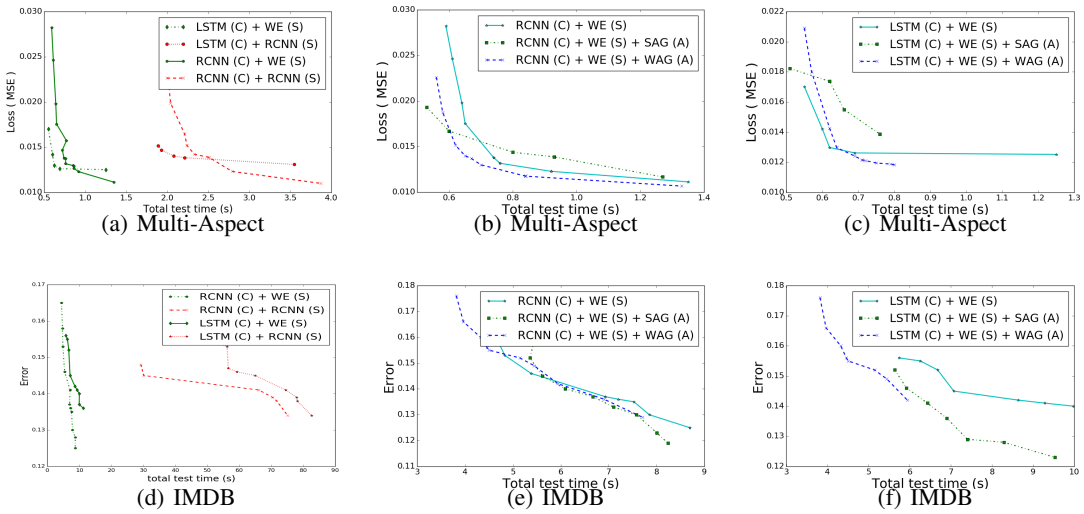
Performance vs. Selected Words. Figure 2 demonstrates the trade-off between the performance and the fraction of words selected by each setting. Overall, the error increases when the fraction of the text selected is lower. On the Multi-Aspect dataset (see Figure 2(a)), the performance of the proposed WE *selector* is competitive with the complex RCNN *selector*. With training the *classifier* with word-level data aggregation strategy, the model further improves and requires only 12% of selected text to achieves an error rate within 0.1% of full-text. Similarly, the WE *selector* and its variant perform well when the *classifier* is an LSTM model (see Figure 2(b)). The mean square error (MSE) of a standard LSTM *classifier* on Multi-Aspect dataset is 0.01250 and our framework outperforms it achieving MSE 0.01188 with only 28% of text. On the IMDB data (see Figure 2(c)), WE *selector* has similar performance trade-off as the RCNN *selector* and further confirms that a simple *selector* is sufficient for identifying rationales. With sentence-level data aggregation, the model performs the best and achieves lower error rate than the baseline RCNN model. To achieve the same accuracy, our approach needs much smaller fraction of text.

Performance vs. Test Time. Next, we report the performance versus test running time in Figure 3. While RCNN *selector* performs well in identifying important words, its complexity is too high and the overall test-time is 2X higher in all cases (see Figure 3(a), 3(d)). In Figures 3(b), 3(c), we show that our framework with data aggregation achieves around 2.5x speed up for RCNN and LSTM *classifier* on the Multi-Aspect dataset at accuracy level of MSE=0.012. Similar results on IMDB are also demonstrated in Figures 3(e), 3(f).

Robust Sentence Representation.

The DNN classifier can be viewed as a representation learner in cascade with a linear classifier (the last softmax layer). Our data aggregation schema enables the representation learner to be robust to the distortions in the input sentences and effectively estimate the representation of a full sentence when only given its fragments. To demonstrate this, we output the latent feature vectors produced by the representation learner and estimate the differences between the vectors when full documents and the fragmented documents are inputted. Results show that, on the AGNews test corpus, the differences in average cosine distances are 0.81 and 0.56 when using the original *classifier* and the *classifier* trained with DAG, respectively. This confirms the proposed approach has an effect of extrapolating to features obtained with full-text even when many words are deleted.

Qualitative Analysis. One advantage of the proposed framework is that the output of the *selector* is interpretable. In Table 2, we present three examples from the AGNews dataset. Results demonstrate that our framework correctly identifies words such as “Nokia”, “nuclear”, “plant”, “Shane Warne”, “software” and phrases such as “searched by police”, “takes six but India established handy lead” as important to the document classification task. It also learns to filter out words (e.g., “Aug.”, “products”, “users”) that are less predictive to the classification labels.



(a) Multi-Aspect (b) Multi-Aspect (c) Multi-Aspect

(d) IMDB (e) IMDB (f) IMDB

Figure 3: The performance versus test running time on Multi-Aspect and IMDB datasets. Results demonstrate that RCNN *selector* is significantly slower than WE due to its high complexity. The data aggregation framework (SAG/WAG) achieves better performance given the same test-time budget.

World News	**Japanese** nuclear **plant** searched . Kansai Electric Power #39;s **nuclear** power **plant** in Fukui, Japan, **was searched by police** Saturday **during an** investigation **into** an Aug. 9 mishap.
Sports	Warne **takes six but India establish handy lead** (Reuters) . Reuters**- World** test **wicket record** holder **Shane Warne** grabbed **six wickets as India** established a handy 141-run **first innings lead in the second** test **on Saturday**.
Sci/Tech	**Handset Makers Raising** Virus Defenses (Reuters) . Reuters - **Software security** companies **and** handset makers, including Finland's **Nokia** (NOK1V.HE), **are** gearing **up to launch** products **intended to** secure **cell** phones **from variants of the Internet viruses that have become a scourge for personal computer** users.

Table 2: Examples of the WE *selector* output on AGNews. Bold words are selected by the *selector*, while the remainder are filtered out. Although words like *"during an"* seem unimportant, appearing in phrases like "bomb exploded *during an* Independence Day parade" (World-News) and "undefeated *during an* entire season" (Sports-News), provide a hint to understand the sentences.

Latency Analysis. In contrast to skim-RNN and LSTM-Jump that sequentially visit the words in a passage. Our model design allows the WE, and Bag-of-Words *selector*s to process words in a passage in parallel. In practice, as the computation involved in our proposed *selector*s is simple, the running time of the *selector* can be negligible. For example, the WE *selector* takes overall only 14s seconds to identify important words on the Yelp dataset, and the LSTM models take up to 316.5 seconds to process the selected words. The benefit is more obvious when the text classification model is employed in a cloud computing setting. The local devices (e.g., smart watches or mobile phones) do not have sufficient memory and computational power to execute a complex classifier. Therefore, the test instance has to be sent to a cloud server and classified by the model on the cloud. In this setting, our approach can employ the *selector* in the local device, and send only important words to the cloud server. In contrast, skim-RNN and LSTM-jump, which process the text in a sequential nature must either send the entire text to the server or require multiple rounds of communication between the server and local devices. In either case, the network latency and bandwidth may restrict the speed of the classification framework. For WE, and Bag-of-Words *selector*, selection depends only on the embedding, and the unigram word itself respectively. Instead, we can cache the predictions and store only a list of important words to save memory.

5 Conclusion

We proposed a budgeted learning framework for learning a robust *classifier* under test-time budget constraints. We demonstrated that training *classifier*s with data aggregation work well with low-complexity selectors based on word-embedding or bag-of-word model and achieve good performance with fragmented input. The future work includes applying the proposed framework to other text reading tasks and improving the data aggregation strategy by applying learning to search approaches (Chang et al., 2015).

REFERENCES

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 527–536, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. Learning to search better than your teacher. In *ICML*, 2015.

Yin-Wen Chang and Chih-Jen Lin. Feature ranking using linear svm. In *Causation and Prediction Challenge*, pp. 53–64, 2008.

Eunsol Choi, Daniel Hewlett, Jakob Uszkoreit, Illia Polosukhin, Alexandre Lacoste, and Jonathan Berant. Coarse-to-fine question answering for long documents. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pp. 209–220, 2017.

Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for natural language processing. *arXiv preprint arXiv:1606.01781*, 2016.

Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah A Smith. Sparse overcomplete word vector representations. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pp. 1491–1500, 2015.

He He, Hal Daumé III, and Jason Eisner. Dynamic Feature Selection for Dependency Parsing. In *EMNLP*, pp. 1455–1464, 2013.

Sergey Karayev, Mario Fritz, and Trevor Darrell. Dynamic feature selection for classification on a budget. In *International Conference on Machine Learning (ICML): Workshop on Prediction with Sequential Models*, 2013.

Tao Lei, Regina Barzilay, and Tommi S. Jaakkola. Rationalizing neural predictions. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

Sam Leroux, Steven Bohez, Elias De Coninck, Tim Verbelen, Bert Vankeirsbilck, Pieter Simoens, and Bart Dhoedt. The cascading neural network: building the internet of smart things. *Knowledge and Information Systems*, pp. 1–24, 2017.

Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 2181–2191. Curran Associates, Inc., 2017.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

Laurens Maaten, Minmin Chen, Stephen Tyree, and Kilian Weinberger. Learning with marginalized corrupted features. In *International Conference on Machine Learning*, pp. 410–418, 2013.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pp. 6297–6308, 2017.

Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 78. ACM, 2004.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010. URL http://arxiv.org/abs/1011.0686.

Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016. URL http://arxiv.org/abs/1611.01603.

Min Joon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. Neural speed reading via skim-rnn. *CoRR*, abs/1711.02085, 2017. URL http://arxiv.org/abs/1711.02085.

Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the conference on empirical methods in natural language processing*, pp. 151–161. Association for Computational Linguistics, 2011.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Emma Strubell, Luke Vilnis, Kate Silverstein, and Andrew McCallum. Learning Dynamic Feature Selection for Fast Sequential Prediction. *arXiv preprint arXiv:1505.06169*, 2015.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

Kirill Trapeznikov and Venkatesh Saligrama. Supervised sequential classification under budget constraints. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, pp. 581–589, 2013.

Paul Viola and Michael Jones. Robust real-time object detection. *International Journal of Computer Vision*, 4, 2001.

David J Weiss and Ben Taskar. Learning adaptive value of information for structured prediction. In *Advances in Neural Information Processing Systems*, pp. 953–961, 2013.

Felix Wu, Ni Lao, John Blitzer, Guandao Yang, and Kilian Q. Weinberger. Fast reading comprehension with convnets. *CoRR*, abs/1711.04352, 2017. URL http://arxiv.org/abs/1711.04352.

Zhixiang Xu, Matt Kusner, Minmin Chen, and Kilian Q Weinberger. Cost-Sensitive Tree of Classifiers. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 133–141, 2013.

Adams Wei Yu, Hongrae Lee, and Quoc Le. Learning to skim text. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pp. 1880–1890, 2017.

Guo-Xun Yuan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *Journal of Machine Learning Research*, 11(Nov):3183–3234, 2010.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pp. 649–657, 2015.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

APPENDIX

We report the statistics of the datasets in Table 3.

Dataset	#class	Classification Task	Vocabulary	Size (Train/Dev/Test)	Avg. Len
SST	2	Sentiment Analysis	13,750	6,920 / 872 / 1,821	19
IMDB	2	Sentiment Analysis	61,046	21,143/3,857/25,000	240
AGNews	4	News Classification	60,088	101,851/18,149/7,600	43
Yelp	5	Sentiment Analysis	1,001,485	600k/50k/50k	149
Multi-Aspect	10	Sentiment Analysis	147,761	51,675/1,000/1,000	144

Table 3: Statistics of the datasets we evaluate our framework on.

MORE EXPERIMENTAL SETTINGS

For the data aggregation on Multi-Aspect dataset, we use budgets $\{0.1, 0.2, .., 1.0\}$ with WE *selector*s. For data aggregation on SST-2, we use the Bag-of-Words *selector*s. Both for the BCN and LSTM *classifier*, we use Allennlp implementation, Adam optimizer, learning rate 0.001, 300 dimensional Glove embeddings with dropout 0.25, batch size 32, and the corresponding architecture of the *classifier* is 2 layered (i.e., encoder, integrator), a 3 feed-forward linear layers with dropout [0.2, 0.3, 0.0], maxpolling and softmax layer.