SPIKING CONVOLUTIONAL NEURAL NETWORKS FOR TEXT CLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Spiking neural networks (SNNs) offer a promising pathway to implement deep neural networks (DNNs) in a more energy-efficient manner since their neurons are sparsely activated and inferences are event-driven. However, there have been very few works that have demonstrated the efficacy of SNNs in language tasks partially because it is non-trivial to represent words in the forms of spikes and to deal with variable-length texts by SNNs. This work presents a "conversion + fine-tuning" two-step method for training SNN for text classification and proposes a simple but effective way to encode pre-trained word embeddings as spike trains. We show empirically that after further fine-tuning with surrogate gradients, the converted SNNs achieve comparable results to their DNN counterparts across multiple datasets for Both English and Chinese. We also demonstrate that such SNNs are more robust against adversarial attacks than DNNs.

1 INTRODUCTION

Inspired by the biological neuro-synaptic framework, modern deep neural networks are successfully used in various applications (Krizhevsky et al., 2017; Graves & Jaitly, 2014; Mikolov et al., 2013b). However, the amount of computational power and energy required to run state-of-the-art deep neural models is considerable and continues to increase in the past decade. For example, a neural language model of GPT-3 (Brown et al., 2020) consumes roughly $190,000$ kWh to train (Dhar, 2020; Anthony et al., 2020), while the human brain performs perception, recognition, reasoning, control, and movement simultaneously with a power budget of just 20 W (Cox & Dean, 2014). Like biological neurons, spiking neural networks (SNNs) use discrete spikes to compute and transmit information, which are more biologically plausible and also energy-efficient than deep learning models. Spike-based computing fuelled with neuromorphic hardware provides a promising way to realize artificial intelligence while reducing energy consumption.

Although many studies have shown that SNNs can produce competitive results in vision (mostly classification) tasks (Cao et al., 2015; Diehl et al., 2015; Rueckauer et al., 2017; Shrestha & Orchard, 2018; Sengupta et al., 2019), there are very few works that have demonstrated their effectiveness in natural language processing (NLP) tasks (Diehl et al., 2016; Rao et al., 2022). SNNs offer a promising opportunity for processing sequential data. Rao et al. (2022) showed that long-short term memory (LSTM) units can be implemented by spike-based neuromorphic hardware with the spike frequency adaptation mechanism. They tested the performance of such spike-based networks (called RelNet) on a question-answering dataset (Weston et al., 2015), and observed that RelNet could solve 16 out of the 17 toy tasks. A task is considered to be solved if the network has an error rate at most 5% on unseen instances of the task. In their design, each word is encoded as a one-hot vector, and a sentence is also fed into the network in the form of one-hot coded spikes. Such a one-hot encoding schema limits the size of the vocabulary that could be used (otherwise, very high-dimensional vectors are required to represent words used in a language). Besides, it is impossible for spike-based networks to leverage the word embeddings learned from a large amount of text data. Diehl et al. (2016) used pre-trained word embeddings in their TrueNorth implementation of a recurrent neural network and achieved 74% accuracy in a question classification task. However, an external projection layer is required to project word embeddings to the vectors with positive values that can be further converted into spike trains. Such a projection layer cannot be easily implemented by spike-based networks, and in fact, they used a hybrid architecture that combines artificial and spiking neural networks, which is less biologically plausible.

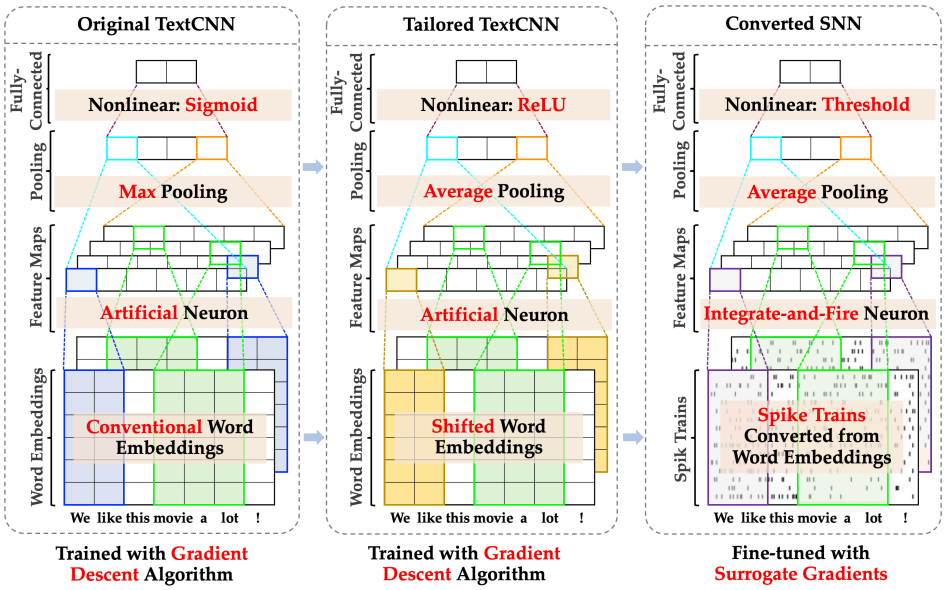


Figure 1: An illustration of a two-step method (conversion + fine-tuning) for training spiking neural networks for text classification: initialize an SNN with the weights of a tailored network trained with the gradient descent, and perform backpropagation with surrogate gradients on the converted SNN. The tailored network is obtained by replacing the max-pooling operation with average-pooling, the Sigmoid activation function with ReLU, and the word embeddings with their positive equivalents.

In this study, we propose a two-step recipe of "conversion + fine-tuning" to train spiking neural networks for NLP. A normally-trained neural network is first converted to a spiking neural network by simply duplicating its architecture and weights, and then the converted SNN is fine-tuned afterward. Before the conversion, an appropriate tailored network needs to be built and trained. Taking a convolutional neural network for sentence classification, called TextCNN (Kim, 2014), as an example (see Figure 1), the original TextCNN is first modified to a tailored CNN by replacing the max-pooling operation with average-pooling, the Sigmoid activation function with ReLU, and the word embeddings with positive-valued vectors (shifted). After the tailored network is trained on a dataset with the gradient descent algorithm, it is converted to a spiking neural network that is further fine-tuned with the surrogate gradient method (Zenke & Vogels, 2021) on the same datast. The SNNs trained with the proposed two-step training strategy yield comparable results to their DNN counterparts. The contribution of this study can be summarized as follows:

- We present a two-step method for training SNNs for language tasks, which combines the conversion-based approach (also known as shallow training) and the backpropagation using surrogate gradients at the fine-tuning phase.
- We propose a method to convert word embeddings to spike trains, which makes it possible for SNNs to leverage the word embeddings pre-trained from a large amount of text data. The ablation study shows that using pre-trained word embeddings can significantly improve the performance of SNNs.
- This study is among the first to demonstrate that well-trained spiking neural networks can achieve comparable results to their DNN counterparts on 6 text classification datasets and for both English and Chinese languages. We also show that SNNs perform more robustly against adversarial attacks than traditional DNNs.

2 Related Work

Spiking neural networks offer a promising computing paradigm due to their ability to capture the rich temporal dynamics of biological neurons. Several methods have been proposed for training SNNs, and they can be roughly divided into two categories: conversion-based and spike-based approaches. The former is to train a non-spiking neural network first and convert it into an SNN that produces the same input-output mapping for a given task as that of the original network. In the latter approaches, SNNs are trained using spike-timing information in an unsupervised or supervised way.

The advantage of conversion-based approaches is that the non-differentiability of discrete spikes can be circumvented and the burden of training in the temporal domain is partially removed. Cao et al. (2015) proposed an approach for converting a deep CNN into an SNN by interpreting the activations as firing rates. To minimize performance loss in the conversion process, Diehl et al. (2015) presented a new weight normalization method to regulate firing rates, which boosts the performance of SNNs without additional training time. Sengupta et al. (2019) pushed spiking neural networks going deeper by exploring residual architectures and introducing a layer-by-layer weight normalization method. However, the conversion is just an approximation, leading to a decline in the accuracy of converted SNNs. Another drawback of such approaches is that converting high precision activations into spikes requires a long sequence of time steps in simulation which increases latency at the inference. Rathi et al. (2020) proposed a hybrid approach to partially address the issue of long-time sequences, which is most related to this study. Initialized with the weights from a trained neural network, the converted SNN is trained using backpropagation. Although this helps to reduce the number of required time steps, it appears to degrade accuracy in image classification tasks. In contrast, we experimentally show that the fine-tuning with surrogate gradients can further improve the accuracy across multiple text classification datasets and the obtained SNNs are capable of integrating the temporal dynamics of spikes properly derived from the pre-trained word embeddings via backpropagation through time.

Inspired by neuroscience, unsupervised SNN training with local STDP-based rules has drawn great attention (Masquelier et al., 2009). Diehl & Cook (2015) demonstrated that an SNN trained in a completely unsupervised way yields comparable accuracy to deep learning on the MNIST dataset. However, unsupervised trained SNNs generally perform worse than their supervised counterparts. Early works in supervised approaches are the tempotron (Gütig & Sompolinsky, 2006) and ReSuMe (Ponulak & Kasiński, 2010). Most works in this line rely on the gradients estimated by a differentiable approximate function so that gradient descent can be applied with backpropagation using spike times (Bohte et al., 2002; Booij & tat Nguyen, 2005) or backpropagation using spikes (i.e., backpropagation through time) (Shrestha & Orchard, 2018; Hunsberger & Eliasmith, 2015; Bellec et al., 2018; Huh & Sejnowski, 2018). To date, supervised learning has been unable to surpass the conversion-based approaches although it turns out to be more computationally efficient. SNNs have provided competitive results, but mostly in vision-related tasks. In this study, we provide proof of experiments that spiking CNNs can yield competitive accuracies over multiple language datasets by combining the advantages of conversion-based approaches and backpropagation using spikes.

3 METHOD

We describe our method for training SNNs for text classification in this section. We first present the approach to building tailored neural networks and the conversion process by taking TextCNN as an example (Kim, 2014), and then depict the way to fine-tune the converted SNNs with surrogate gradients, where the pre-trained word embeddings were transformed into spike trains produced by a Poisson event-generation function. The entire training procedure is summarized in Algorithm 1.

3.1 CONVERSION-BASED APPROACH

The idea behind the conversion-based approaches is simple—interpreting the activations as firing rates and mapping the magnitude of values output by each unit of a DNN to the frequency of spikes generated by the corresponding neuron of the converted SNN. To enable such conversions, a DNN architecture should be tailored to fit the requirements of SNN by removing some operations that cannot be realized by spike-based computation (Cao et al., 2015). The tailored DNN is trained by the same way as one would with conventional DNN, and the learned weights are then applied to the SNN converted from the tailored DNN. Some weight normalization methods are often applied to regulate firing rates after the conversion Diehl et al. (2015); Rueckauer et al. (2017); Sengupta et al. (2019). However, we found that when the conversion is followed by the fine-tuning step, the effectiveness of weight normalization is negligible for text classification tasks (see Subsection 4.3).

3.1.1 TAILORED NEURAL NETWORK

Although the proposed method is applied to all the deep neural architectures that can be converted to SNNs, we take the convolutional neural networks for sentence classification (TextCNN) (Kim, 2014) as an example architecture for clarity (this architecture is also used in the experiments). As shown in Figure 1, the TextCNN applies a convolution layer with multiple filter widths and feature maps

over word embeddings learn from an unsupervised neural language model, and then summarizes the outputs of the convolution layer over time by a max-pooling (followed by a fully-connected layer) to produce a sentence representation. The TextCNN is trained over the summarized representations by minimizing a given loss function.

The operations that will produce negative values or those not supported in SNNs should be avoided or replaced with other alternatives in tailored neural networks because negative values are quite hard to be precisely represented in SNNs. There is also no simple and good way to implement the max-pooling operation in SNNs, which requires two additional network layers with lateral inhibition and causes a loss in accuracy due to the additional complexity. The biases of neurons also cannot easily be implemented in SNNs since their value could be positive or negative. Like Cao et al. (2015), we create tailored neural networks by making the following changes to their original DNNs:

- Word embeddings are converted into the vectors of the same dimension with positive values by normalization and shifting (discuss later in Subsection 3.1.2).
- All the non-linear activation functions are replaced with ReLU (rectified linear unit) activation function (i.e., $\text{ReLU}(x) = \max(x, 0)$).
- The biases are removed from all the convolutional and fully-connected layers.
- The average-pooling is used instead of the max-pooling, which can be easily implemented in spike-based computation.

3.1.2 Pre-trained Word Embeddings

Pre-trained word embeddings have been successfully used in a wide range of NLP tasks, and they should be useful for SNNs to generalize from a training set with a limited size to possible unseen texts too. The values of word embeddings are not all positive, and therefore an appropriate method is required to convert those word embeddings into the vectors with positive values so that the inputs to the first layer of an SNN are all non-negative. We tried several methods to fulfill this purpose, and found the following one is simple, but effective in preserving the semantic regularities in language captured in the original word embeddings when they are converted and transformed to spike trains. We first calculate the mean value μ and the standard deviation σ of the values of pre-trained word embeddings, clamp all the values within $[\mu + 3\sigma, \mu - 3\sigma]$, perform the normalization by subtracting μ and then dividing by σ, and shift all the components of vectors within the range of $[0, 1]$.

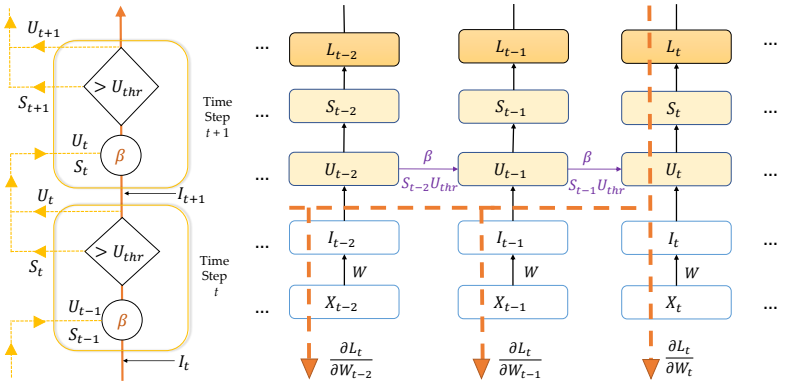
For English, we used 300-dimentional GloVe embeddings (Pennington et al., 2014) trained on a text dataset merged from Wikipedia 2014 and Gigaword 5. For Chinese, the Word2Vec toolkit (Mikolov et al., 2013a) was used to learn word embeddings of 300 dimensions from a text corpus composed of Baidu Encyclopedia, Wikipedia-zh, People's Daily News, Sogou News, Financial News, ZhihuQA, Weibo and Complete Library in Four Sections (Li et al., 2018b). In the supervised training stage of tailored neural networks, the converted (positive-valued) word embeddings are free to modify and their values will be clamped to $[0, 1]$ if they were modified to values greater than 1 or less than 0. SNNs only take spikes as input, and thus a Poisson spike train will be generated for each component of a word embedding with a firing rate proportional to its scale.

3.1.3 Training and Conversion

We use the cross-entropy loss to train the tailored neural networks as usual. As illustrated in Figure 1, the conversion of a tailored network to an SNN is straightforward. All the processing blocks of the converted SNN are inherited from the tailored network except a spike generator that is added to the SNN, which is used to generate Poisson spike trains derived from the learned word embeddings. Each neuron of the tailored network will be replaced with a leaky integrate-and-fire neuron (discuss later in subsection 3.2.1), and the weights for convolutional and fully-connected layers in the tailored network become the synaptic strengths in the converted SNN. The ReLU activation functions are no longer needed since their functionality is implicitly provided by the neuron's membrane threshold.

3.2 Fine-tuning with surrogate gradients

Once an SNN is converted from a tailored network (trained), we can fine-tune the converted SNN by the generalized backpropagation algorithm with surrogate gradients on the same dataset that was used to train the tailored network. The converted weights can be viewed as a good initialization, which contributes to solving the problem of temporal and spatial credit assignment for the SNN. In the fine-tuning stage, the word embeddings present in the form of spike trains are fixed because such



(a) A LIF neuron (b) Backpropagation through time (BPTT)

Figure 2: Computational steps in training SNNs by the generalized backpropagation with surrogate gradients. (a) A recurrent representation of a leaky integrate-and-fire (LIF) neuron. (b) An unrolled computational graph of the LIF neuron where time flows from left to right.

spike trains are generated randomly and there is no one-to-one mapping between a word embedding and its spike train. However, such temporal codes with timing disturbances make SNNs more robust to adversarial attacks (see the experimental results in subsection 4.4) because the trained SNNs are more tolerant to noise appearing in input spike trains randomly generated.

3.2.1 Integrate-and-fire Neuron

There have been many spiking neuron models that could be used to build SNNs (Izhikevich, 2004), we chose to use a widely-used, first-order leaky integrate-and-fire (LIF) neuron as the building block. Like the artificial neuron model, LIF neurons operate on a weighted sum of inputs, which contributes to the membrane potential U_t of the neuron at time step t. If the neuron is sufficiently excited by the weighted sum and its membrane potential reaches a threshold U_{thr}, a spike S_t will be generated:

$$S_t = \begin{cases} 1, & \text{if } U_t \geq U_{\text{thr}}; \\ 0, & \text{if } U_t < U_{\text{thr}}. \end{cases} \tag{1}$$

The dynamics of the neuron's membrane potential can be modelled as an resistor–capacitor circuit, an approximate solution to the differential equation of this circuit can be represented as follows:

$$\begin{aligned} U_t &= I_t + \beta U_{t-1} - S_{t-1} U_{\text{thr}} \\ I_t &= W X_t \end{aligned} \tag{2}$$

where X_t are inputs to the LIF neuron at time step t, W is a set of learnable weights used to integrate different inputs, I_t is the weighted sum of inputs, β is the decay rate of membrane potential, and U_{t-1} is the membrane potential at the previous time $t-1$. The last term of $S_{t-1}U_{\text{thr}}$ is introduced to account for spiking and membrane potential reset. A LIF neuron model is illustrated in Figure 2.

3.2.2 Surrogate Gradient

Backpropagation through time (BPTT) is one of the most popular approaches for training SNNs (Shrestha & Orchard, 2018; Huh & Sejnowski, 2018; Cramer et al., 2022). This approach applies the generalized backpropagation algorithm to the unrolled computational graph. The gradients flow from the final output of a network to its input layer (indicated by the orange dashed arrows in Figure 2 (b)). In this way, computing the gradients through an SNN is most similar to that of a recurrent neural network. To deal with the spike non-differentiability problem, the Heaviside step function, shown as Equation (1), is applied to determine whether a neuron emits a spike during the forward pass while this function is replaced with a differentiable one during the backward pass. The derivative of the differentiable function is used as a surrogate, and this approach is known as the surrogate gradient. We chose to use Fast-Sigmoid (Zheng & Mazumder, 2018) as the surrogate function $\hat{S}_t$, where k is set to 25 by default:

$$\hat{S}_t \approx \frac{U_t}{1 + k|U_t|} \tag{3}$$

Algorithm 1 The global algorithm of "conversion + fine-tuning" for training spiking neural networks.

Input: E: A set of pre-trained word embeddings;
D: A training set consisting of N instances $\{(x_i, y_i)\}_{i=1}^{N}$;
R: A traditional neural network (architecture only);
M: A spiking neural network with a set of learnable weights W;
β: A decay rate of membrane potential;
U_{thr}: A membrane threshold;
η: a learning rate η used at the fine-tuning phase.

I. Conversion Step:
Transform pre-trained word embeddings E to the vectors with positive values (Subsection 3.1.2);
Build a tailored neural network from a traditional neural network R (Subsection 3.1.1);
Train the tailored neural network on the training set D with the gradient descent algorithm;
Convert the tailored neural network (trained) to the corresponding spiking neural network M.

II. Fine-Tuning Step:
for *each mini-batch B in D* **do**
  Generate a Poisson spike train for each component of all the word embeddings appeared in B;
  Perform a forward pass and record the spikes as well as the membrane potentials at every time step;
  Calculate the derivative of the loss with respect to the weights (see Equation (4));
  Update the weights of the spiking neural network M by $W = W - \eta \frac{\partial L}{\partial W}$ (see Appendix A.1).
end
Return The fine-tuned spiking neural network M.

3.2.3 Loss Function

Since the surrogate function, like Equation (3), is applied when working backward, we can add a softmax layer to the end of SNNs to predict the category labels at each time step t for an instance i, denoted by $\hat{y}_t^i$. The SNNs are fine-tuned by minimizing the cross-entropy spike rate error using the generalized gradient descent. This is equivalent to minimizing the KL-divergence between the prediction distribution $\hat{y}_t^i$ and the target distribution y^i at each time step t for an instance i. We use the 1-of-K coding scheme to represent the target y^i. The loss function for N training instances is:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{T}\sum_{t=1}^{T}\left(y^i \times \log(\hat{y}_t^i)\right)\right) \tag{4}$$

where T is the number of time steps used for training SNNs. Finding the derivative of this loss with respect to the weights allows the use of gradient decent to train SNNs. We list the efficient way to calculate the derivatives in Appendix A.1.

4 Experiments

We conducted four sets of experiments. The first is to evaluate the accuracy of the SNNs trained with the proposed method on 6 different text classification benchmarks for both English and Chinese by comparing to their DNN counterparts. The goal of the second experiment is to see how robust the SNNs would be to defend against existing sophisticated adversarial attacks. The third one is to show that the conversion and fine-tuning sets are essential for training SNNs by ablation study. The last experiment is to see how the performance of SNNs is impacted by the value of decay rate, the number of neurons used to predict for each category, and the value of membrane threshold.

4.1 Dataset

We used the following 6 text classification datasets to evaluate the SNNs trained by the proposed method, four of which are English datasets and the other two are Chinese benchmarks. Those datasets vary in the size of examples and the length of texts. If there is no standard training-test split, we randomly select 10% examples from the entire dataset as the test set.

- **MR**: It consists of movie-review documents labeled with respect to their overall sentiment polarity (positive or negative) or subjective rating (Pang & Lee, 2005).

- **SST-5**: The Stanford Sentiment Treebank 5 comprises $11,855$ sentences extracted from movie reviews for sentiment classification (Socher et al., 2013). There are 5 different classes (very negative, negative, neutral, positive, very positive).
- **SST-2**: It is the binary version of SST-5, and there are just 2 classes (positive and negative).
- **Subj**: The task of this dataset is to classify a sentence as being subjective or objective[1].
- **ChnSenti**: This datasets contains about $7,000$ Chinese hotel reviews annotated with positive or negative labels[2].
- **Waimai**: It consists of $12,000$ Chinese user reviews collected by a food delivery platform for binary sentiment classification (positive and negative)[3].

4.2 Implementation Details

We used TextCNN (Kim, 2014) as the neural network architecture from which the tailored network is built, and filter widths of 3, 4, and 5 with 100 feature maps each. When training the tailored networks, we set the dropout rate to 0.5, the batch size to 32, and the learning rate to $1e-4$.

SnnTorch framework provided by (Eshraghian et al., 2021) was used to train SNNs, which extends the capabilities of PyTorch (Paszke et al., 2019) and can perform gradient-based learning with SNNs. We set the number of time steps to 50, the membrane threshold U_{thr} to 1, the decay rate β to 1, the batch size to 50, and the learning rate to $5e-5$ at the fine-tuning stage of SNNs.

Eshraghian et al. (2021) showed that if we collect the results produced by multiple neurons and count their spikes, it is possible to accurately measure a firing rate from a population of neurons in a very short time window. Therefore, we also used such a commonly-used ensemble method in the tailored networks and the converted SNNs. Specifically, instead of assigning one neuron to each category for prediction, we use h neurons for each category and the prediction results on h spiking neurons are ensembled to get a final output. Unless otherwise specified, we set h to 10 in all the experiments.

4.3 Results

We reported in Table 1 the classification accuracy achieved by the SNNs trained with the "conversion + fine-tuning" method on 4 English and 2 Chinese datasets, compared to several baselines, including the converted SNNs without the fine-tuning, the converted SNNs with two weight normalization methods, and the SNNs directly-trained with surrogate gradients without using the weights of tailored networks for the initialization. Diehl et al. (2015) proposed two ways to perform the weight normalization on the converted SNNs. The first one is called a model-based normalization that considers all possible positive activations and re-scale all the weights by the maximum positive input, and the second is called a data-based normalization in which the weights are normalized according to the maximum activation reached by propagating all the training examples through the network.

The numbers reported in Table 1 show that the SNNs trained with the proposed method outperform all the SNN baselines across 6 text classification datasets. They also achieved comparable results to the original TextCNNs by a small drop of 2.51% on average in accuracy (2.89% difference for English and 1.78% for Chinese respectively). We tried some combinations of "conversion + normalization + fine-tuning" and found that when the conversion is followed by the fine-tuning step, the weight normalization methods contribute a little to the performance of SNNs on the text classification tasks.

4.4 Adversarial Robustness

Deep neural networks have proven to be vulnerable to adversarial examples (Samanta & Mehta, 2017; Wong, 2017; Liang et al., 2018; Alzantot et al., 2018), and the existence and pervasiveness of adversarial examples have raised serious concerns. We believe that SNNs provide a promising means to defend against adversarial attacks due to the non-differentiability of spikes and their tolerance to noise introduced by randomly-generated input spike trains. We evaluated the empirical robustness of SNNs under test-time attacks with four black-box, synonym substitution-based attacks: TextFooler

[1] https://www.cs.cornell.edu/people/pabo/movie-review-data/

[2] https://raw.githubusercontent.com/SophonPlus/ChineseNlpCorpus/master/datasets/ChnSentiCorp_htl_all/ChnSentiCorp_htl_all.csv

[3] https://raw.githubusercontent.com/SophonPlus/ChineseNlpCorpus/master/datasets/waimai_10k/waimai_10k.csv

Table 1: Classification accuracy achieved by different models on 6 datasets. The model obtained by applying the model-based normalization on the converted SNN is denoted as "Converted SNN + MN" and that by applying the data-based normalization as "Converted SNN + DN".

Method	**English Dataset**				**Chinese Dataset**	
	MR	**SST-2**	**Subj**	**SST-5**	**ChnSenti**	**Waimai**
Original TextCNN	77.41	83.25	94.00	45.48	86.74	88.49
Tailored TextCNN	76.94	83.03	91.50	43.48	85.79	88.21
Directly-trained SNN	51.55	75.73	53.30	23.08	63.18	66.42
Converted SNN	74.13	80.07	90.40	41.40	84.16	86.43
Converted SNN + MN	74.70	79.90	89.40	40.59	84.89	85.21
Converted SNN + DN	74.19	80.67	90.30	40.63	83.73	86.33
Converted SNN + Fine-tuning	**75.45**	**80.91**	**90.60**	**41.63**	**85.02**	**86.66**

Table 2: Empirical results on 4 English datasets under 4 different adversarial attack algorithms.

Dataset	**Model**	**Cln**	**TextFooler**		**TextBugger**		**BERT-Attack**		**PWWS**	
			Boa	**Suc**	**Boa**	**Suc**	**Boa**	**Suc**	**Boa**	**Suc**
MR	TextCNN	77.50	8.60	88.11	8.30	88.38	13.30	81.58	5.30	92.79
	SNN	74.30	**16.40**	77.47	**10.40**	85.91	**22.70**	69.45	**12.20**	82.98
SST-2	TextCNN	81.80	8.30	89.51	4.70	94.08	13.30	83.54	4.10	94.75
	SNN	80.20	**14.70**	81.39	**9.20**	88.22	**21.50**	72.37	**11.00**	86.11
Subj	TextCNN	93.50	11.10	87.12	7.40	91.28	12.30	84.60	6.40	92.31
	SNN	90.40	**47.40**	47.09	**39.50**	56.16	**51.00**	42.50	**41.40**	54.41
SST-5	TextCNN	44.80	0.70	98.31	0.30	99.33	1.80	95.74	0.50	98.80
	SNN	41.10	**7.00**	84.49	**5.10**	89.90	**8.30**	81.33	**5.50**	88.51

(Jin et al., 2020), TextBugger (Li et al., 2018a), BERT-Attack (Li et al., 2020), and PWWS (Ren et al., 2019). BERT-Attack generates synonyms dynamically by using BERT (Devlin et al., 2019), and all the other attack algorithms use K nearest neighbor words of GloVe vectors (Pennington et al., 2014) to generate the synonyms of a word. In this experiment, we set the maximum percentage of words that can be modified to 0.3, the size of the synonym set to 15, and the semantic similarity threshold between an original text and the adversarial one to 0.8. The following metrics (Li et al., 2021) are used to report the results of empirical robustness:

- The *clean accuracy* (**Cln**) is the accuracy achieved by a classifier on the clean texts.
- The *robust accuracy* (**Boa**) is the accuracy of a classifier achieved under a certain attack.
- The *success rate* (**Suc**) is the number of texts successfully perturbed by an attack algorithm (causing the model to make errors) divided by all the number of texts to be attempted.

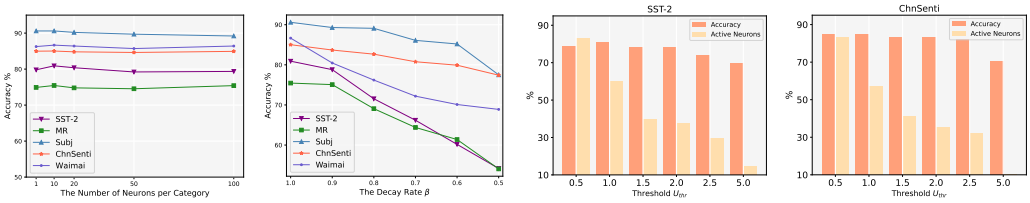
Table 2 shows the clean accuracy, robust accuracy (i.e., accuracy under attack), and attack success rate achieved by the SNNs under four sophisticated attacks on all 4 English datasets, compared to the original TextCNNs. From these numbers, we can see that the SNNs consistently perform better than the original neural networks under all four attack algorithms on the robust accuracy and attack success rate while suffering little performance drop on the clean data. The SNNs can improve the robust accuracy under attack by 13.55% on average and lower the attack success rate by 17% on average. On the Subj dataset, the robust accuracy can even be increased by a fairly significant margin of 38.7% with 42.1% decrease in the attack success rate under the test-time BERT-Attack.

4.5 Ablation Study and Impact of Hyper-Parameters

We conducted an ablation study over all the considered datasets on the SNNs obtained by two variants of training methods to analyze the necessity of the shallow training (i.e., conversion step) and the pre-trained word embeddings. One is to train SNNs directly with the surrogate gradients without the conversion step. As we can see from the row indicated by "Directly-trained SNN" from Table 1, the SNNs trained directly perform considerably worse than those trained with the two-step method by a significant margin of 21.17% accuracy on average. It shows that the conversion step is indispensable in training SNNs. Another is to use randomly-generated word embeddings to train SNNs. We report these results in Table 3, where the models indicated by "RWE" were initialized with the word embeddings randomly generated, while those by "PRE" were with pre-trained word embeddings. No matter what training method is used, a dramatic drop in accuracy is observed in all the SNNs.

Table 3: Classification accuracy achieved by the models using randomly-generated word embeddings (denoted as "RWE") and those using pre-trained word embeddings (denoted as "PWE"). The model obtained by applying the model-based normalization on the converted SNN is denoted as "Converted SNN + MN" and that by applying the data-based normalization as "Converted SNN + DN".

Method	**English Dataset**				**Chinese Dataset**	
	MR	**SST-2**	**Subj**	**SST-5**	**ChnSenti**	**Waimai**
Original TextCNN (RWE)	75.16	80.23	91.20	39.32	82.36	81.72
Tailored TextCNN (RWE)	51.55	49.92	54.40	23.08	68.93	66.42
Converted SNN (RWE)	49.39	51.53	52.80	25.93	68.93	66.42
Converted SNN + MN (RWE)	48.45	50.80	51.90	25.24	66.84	67.50
Converted SNN + DN (RWE)	51.55	50.92	52.80	23.08	67.93	66.24
Converted SNN + Fine-tuning (RWE)	48.36	49.97	52.20	25.24	66.57	66.42
Converted SNN + Fine-tuning (PWE)	**75.45**	**80.91**	**90.60**	**41.63**	**85.02**	**86.66**



(a) (b) (c) (d)

Figure 3: The impact of hyper-parameters. (a) Accuracy versus the number of neurons used per category. (b) Accuracy versus the decay rate of β. (c) and (d) Accuracy and the proportion of active neurons influenced by different values of membrane thresholds U_{thr} on SST-2 and ChnSenti datasets.

Comparing the numbers shown in the last two rows of Table 3, we noticed a considerable difference of 25.25% in accuracy between the SNNs with or without using pre-trained word embeddings, indicating their effectiveness in improving the performance of SNNs.

We want to understand how the performance and estimated energy consumption of SNNs are impacted by the choice of three important hyper-parameters: the number of neurons per category, the value of decay rate β, and the membrane threshold U_{thr}. As we mentioned in Subsection 4.2, more than one spiking neuron can be assigned to each category to improve the accuracy of prediction. Figure 3 (a) shows that the classification accuracy is generally insensitive to the number of neurons used for prediction, and the highest accuracy can be achieved around 10 neurons per category. It appears that if more neurons are used, the SNNs suffer from the problem of over-fitting. As we can see from Figure 3 (b), if the conversion-based method is used the value of β should be set to 1.0, otherwise the accuracy will be severely degraded. A single spike only consumes a constant amount of energy (Cao et al., 2015). The amount of energy consumption heavily depends on the number of spikes and the number of time steps used at the inference stage. Figure 3 (c) and (d) show that we can reduce the number of active neurons (approximately the number of spikes) by increasing the value of membrane threshold while suffering little or no performance drop, which implies the possibility of about 50% energy saving by carefully adjusting the values of membrane threshold (say $U_{\text{thr}} = 2$).

5 Conclusion

We found that it is hard to train spiking neural networks for language tasks using the error back-propagation through time although SNNs are supposed to be suitable for modeling time-varying data due to their temporal dynamics. To address this issue, we suggested a two-step training recipe: start with an SNN converted from a normally-trained tailored network, and perform backpropagation on the converted SNN. We also proposed a method to make use of pre-trained word embeddings in SNNs. Pre-trained word embeddings are projected into vectors with positive values after proper normalization and shifting, which can be used to initialize tailored networks and converted to spike trains as input of SNNs. Through extensive experimentation on 6 text classification datasets, we demonstrated that the SNNs trained with the proposed method achieved competitive results on both English and Chinese datasets. Such SNNs were also proven to be less vulnerable to textual adversarial examples than traditional neural counterparts. It would be interesting to see if we can pre-train SNNs unsupervisedly using masked language modeling with a large text set in the future.

REFERENCES

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2018.

Lasse F Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *arXiv preprint arXiv:2007.03051*, 2020.

Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. *Advances in neural information processing systems*, 31, 2018.

Sander M Bohte, Joost N Kok, and Han La Poutre. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4):17–37, 2002.

Olaf Booij and Hieu tat Nguyen. A gradient descent rule for spiking neurons emitting multiple spikes. *Information Processing Letters*, 95(6):552–558, 2005.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, 2015.

David Daniel Cox and Thomas Dean. Neural networks and neuroscience-inspired computer vision. *Current Biology*, 24(18):R921–R929, 2014.

Benjamin Cramer, Sebastian Billaudelle, Simeon Kanya, Aron Leibfried, Andreas Grübl, Vitali Karasenko, Christian Pehle, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, et al. Surrogate gradients for analog neuromorphic computing. *Proceedings of the National Academy of Sciences*, 119(4):e2109194119, 2022.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

Payal Dhar. The carbon impact of artificial intelligence. *Nat. Mach. Intell.*, 2(8):423–425, 2020.

Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, 9:99, 2015.

Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International joint conference on neural networks (IJCNN)*, pp. 1–8. IEEE, 2015.

Peter U Diehl, Guido Zarrella, Andrew Cassidy, Bruno U Pedroni, and Emre Neftci. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8. IEEE, 2016.

Jason Kamran Eshraghian, Max Ward, Emre O. Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Bennamoun, Doo Seok Jeong, and Wei D. Lu. Training spiking neural networks using lessons from deep learning. *ArXiv*, abs/2109.12894, 2021.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning*, pp. 1764–1772. PMLR, 2014.

Robert Gütig and Haim Sompolinsky. The tempotron: a neuron that learns spike timing–based decisions. *Nature neuroscience*, 9(3):420–428, 2006.

Dongsung Huh and Terrence J Sejnowski. Gradient descent for spiking neural networks. *Advances in neural information processing systems*, 31, 2018.

Eric Hunsberger and Chris Eliasmith. Spiking deep networks with lif neurons. *arXiv preprint arXiv:1510.08829*, 2015.

Eugene M Izhikevich. Which model to use for cortical spiking neurons? *IEEE transactions on neural networks*, 15(5):1063–1070, 2004.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is BERT really robust? a strong baseline for natural language attack on text classification and entailment. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8018–8025, Apr. 2020. doi: 10.1609/aaai.v34i05.6311.

Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. Textbugger: Generating adversarial text against real-world applications. *Network and Distributed Systems Security (NDSS) Symposium 2019*, 2018a.

Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. BERT-ATTACK: Adversarial attack against BERT using BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6193–6202, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.500.

Shen Li, Zhe Zhao, Renfen Hu, Wensi Li, Tao Liu, and Xiaoyong Du. Analogical reasoning on chinese morphological and semantic relations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 138–143. Association for Computational Linguistics, 2018b.

Zongyi Li, Jianhan Xu, Jiehang Zeng, Linyang Li, Xiaoqing Zheng, Qi Zhang, Kai-Wei Chang, and Cho-Jui Hsieh. Searching for an effective defender: Benchmarking defense against adversarial word substitution. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. Deep text classification can be fooled. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2018.

Timothée Masquelier, Rudy Guyonneau, and Simon J Thorpe. Competitive stdp-based spike pattern learning. *Neural computation*, 21(5):1259–1276, 2009.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *ICLR*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013b.

Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *ACL*, 2005.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162.

Filip Ponulak and Andrzej Kasiński. Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting. *Neural computation*, 22(2):467–510, 2010.

Arjun Rao, Philipp Plank, Andreas Wild, and Wolfgang Maass. A long short-term memory for ai applications in spike-based neuromorphic hardware. *Nature Machine Intelligence*, 4(5):467–479, 2022.

Nitin Rathi, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. *arXiv preprint arXiv:2005.01807*, 2020.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1085–1097, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1103.

Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience*, 11:682, 2017.

Suranjana Samanta and Sameep Mehta. Towards crafting text adversarial samples. *Computing Research Repository*, arXiv: 1707.02812, 2017.

Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: VGG and residual architectures. *Frontiers in neuroscience*, 13:95, 2019.

Sumit B Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. *Advances in neural information processing systems*, 31, 2018.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, A. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart Van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

Catherine Wong. DANCin SEQ2SEQ: Fooling text classifiers with adversarial text example generation. *Computing Research Repository*, arXiv: 1712.05419, 2017.

Friedemann Zenke and Tim P Vogels. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural computation*, 33(4):899–925, 2021.

Nan Zheng and Pinaki Mazumder. A low-power hardware architecture for on-line supervised learning in multi-layer spiking neural networks. *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2018.

A Appendix

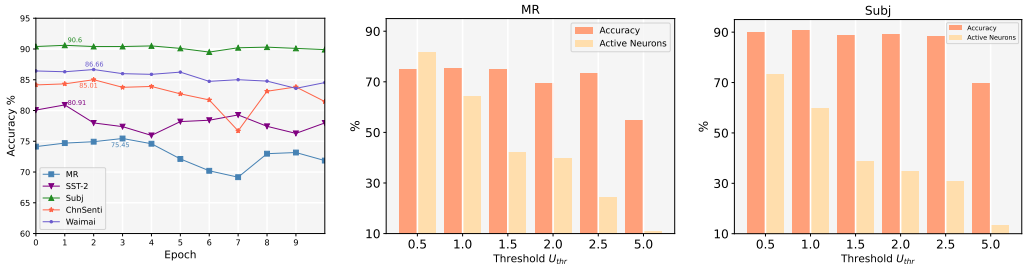
A.1 The derivative of the loss with respect to the weights

Given a loss function defined in Equation (4), the losses at every time step can be summed together to give the following global gradient, as illustrated in Figure 2 (b):

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L_t}{\partial W} = \sum_i \sum_{j \leq i} \frac{\partial L_i}{\partial W_j} \frac{\partial W_j}{\partial W} \tag{5}$$

where i and j denote different time steps, and L_t is the loss calculated at the step t. No matter which time step is, the weights of an SNN are shared across all steps. Therefore, we have $W_0 = W_1 = \cdots = W$, which also indicates that $\frac{\partial W_j}{\partial W} = 1$. Thus, Equation (5) can be written as follows:

$$\frac{\partial L}{\partial W} = \sum_i \sum_{j \leq i} \frac{\partial L_i}{\partial W_j} \tag{6}$$



(a) (b) (c)

Figure 4: (a) Classification accuracy versus the number of epochs used to fine-tune SNNs (b) and (c) Accuracy and the proportions of active neurons influenced by different values of membrane thresholds U_{thr} on MR and Subj dataset.

Based on the chain rule of derivatives, we obtain:

$$\begin{aligned}\frac{\partial L}{\partial W} &= \sum_i \sum_{j \leq i} \frac{\partial L_i}{\partial S_i} \frac{\partial S_i}{\partial U_i} \frac{\partial U_i}{\partial W_j} \\ &= \sum_i \frac{\partial L_i}{\partial S_i} \frac{\partial S_i}{\partial U_i} \sum_{j \leq i} \frac{\partial U_i}{\partial W_j}\end{aligned} \tag{7}$$

where $\frac{\partial L_i}{\partial S_i}$ is the derivative of the cross-entropy loss at the time step i with respect to S_i, and $\frac{\partial S_i}{\partial U_i}$ can be easily derived from Equation (3). As to the last term of $\sum_{j \leq i} \frac{\partial U_i}{\partial W_j}$, we can split it into two parts:

$$\sum_{j \leq i} \frac{\partial U_i}{\partial W_j} = \frac{\partial U_i}{\partial W_i} + \sum_{j \leq i-1} \frac{\partial U_i}{\partial W_j} \tag{8}$$

From Equation (2), we know that $\frac{\partial U_i}{\partial W_i} = X_i$. Therefore, Equation (5) can be simplified as follows:

$$\frac{\partial L}{\partial W} = \sum_i \underbrace{\frac{\partial L_i}{\partial S_i} \frac{\partial S_i}{\partial U_i}}_{\text{constant}} \left(\underbrace{\frac{\partial U_i}{\partial W_j}}_{\text{constant}} + \sum_{j \leq i-1} \frac{\partial U_i}{\partial W_j} \right) \tag{9}$$

By the chain rule of derivatives over time, $\frac{\partial U_i}{\partial W_j}$ can be factorized into two parts:

$$\frac{\partial U_i}{\partial W_j} = \frac{\partial U_i}{\partial U_{i-1}} \frac{\partial U_{i-1}}{\partial W_j} \tag{10}$$

It is easy to see that $\frac{\partial U_i}{\partial U_{i-1}}$ is equal to β from Equation (2), and Equation (5) can be written as:

$$\frac{\partial L}{\partial W} = \sum_i \underbrace{\frac{\partial L_i}{\partial S_i} \frac{\partial S_i}{\partial U_i}}_{\text{constant}} \left(\underbrace{\frac{\partial U_i}{\partial W_j}}_{\text{constant}} + \sum_{j \leq i-1} \underbrace{\frac{\partial U_i}{\partial U_{i-1}}}_{\text{constant}} \frac{\partial U_{i-1}}{\partial W_j} \right) \tag{11}$$

We can treat $\frac{\partial U_{i-1}}{\partial W_j}$ recurrently as Equation (8). Finally, we can update the weights W by the rule of $W = W - \eta \frac{\partial L}{\partial W}$, where η is a learning rate.

A.2 More Analysis on the Impact of Hyper-parameters

Figure 4 (a) shows how the accuracy of SNNs varies as the number of epochs grows at the fine-tuning phase. Although the highest accuracy is achieved with different numbers of epochs for different datasets, the peak accuracy is always reached within 5 epochs, indicating that just a few epochs are required to fine-tune the converted SNNs and such additional training time and effort are acceptable. For each value of membrane threshold U_{thr}, the corresponding accuracy and the proportion of active

Table 4: Classification accuracy achieved by different models on 6 datasets. These models were trained without using the dropout technique.

Method	**English Dataset**				**Chinese Dataset**	
	MR	**SST-2**	**Subj**	**SST-5**	**ChnSenti**	**Waimai**
Original TextCNN	76.29	82.70	89.50	42.40	84.33	87.43
Tailored TextCNN	75.91	82.26	92.50	42.67	85.11	87.82
Converted SNN	74.68	73.41	86.50	35.29	69.61	84.63
Converted SNN + MN	57.83	73.75	89.50	29.10	68.93	80.46
Converted SNN + DN	74.73	75.38	87.00	35.48	69.61	84.35
Converted SNN + Fine-tuning	**75.16**	**75.45**	**90.50**	**38.87**	**83.99**	**86.04**

spiking neurons on both MR and Subj datasets are reported in Figure 4 (b) and (c) respectively. We found similar trends as those for SST-2 and ChnSenti datasets shown in Figure 3 (c) and (d), which confirms that the energy consumption can be further reduced by about 50% without suffering much performance loss. We also investigate the impact of the dropout technique on the performance of the resulting SNNs. Table 4 shows the accuracy on 6 text classification datasets achieved by different neural models that were trained without using the dropout technique (Srivastava et al., 2014). By comparing the numbers reported in Tables 1 (where the dropout rate was set to 50% during the training process) and 4, it is clear that the dropout technique is still quite useful to train spike neural networks although its contribution to the performance of SNNs is slightly smaller than that of DNNs.