NeurIPS 2019 Reproducibility Challenge-Kernel-Based Approaches for Sequence Modeling: Connections to Neural Methods

Palak Goenka
Indian Institute of Technology, Roorkee
goenkapalak11@gmail.com

Ashutosh Bhushan Bharambe
Indian Institute of Technology, Roorkee
a.bharambe123@gmail.com

Kartikey Pandey
Indian Institute of Technology, Roorkee
pandeykartikey99@gmail.com

Subham Sahoo
Indian Institute of Technology, Roorkee
subhamsahoo4444@gmail.com

Abstract

Motivated by the importance of kernel-machines for the development of Deep Learning models, we have represented an extensive study of the work done in the paper *Kernel-Based Approaches for Sequence Modeling:Connections to Neural Methods* [1] as a part of NeurIPS Reproducibility Challenge 2020. In this report, we have tried to judge the reproducibility of the original paper by comparing our results with the ones reported by the authors. Along with every minute explanation required for the experimentation defined in the original paper [1], we have also tried to provide insights for the incorporation of various training techniques mentioned by the authors. Our complete codebase is available at GitHub.

1 Introduction

The property of Deep Learning (DL), which says that deep learning can be viewed as a feature mapping from $\psi_\theta(x)$ to the weight ω, has opened up a new question of bridging the gap between deep learning and kernel machines. Insights obtained about neural networks from the perspective of kernel machines have proved to provide a better explanation of deep learning models [2].

Prior work in this domain has been predominantly in the field of image analysis. For instance, in [3], authors have presented an analysis for Convolutional Neural Networks (CNN) as Hierarchical Kernel Machines. Furthermore, there is significant work on the use of recurrent kernel machines (RKMs) for sequential data [4]. Inspired by the previous work, the authors have derived connections between recurrent neural networks (RNNs) and recurrent kernel machines (RKMs).

The paper constructs RNN in terms of RKMs by using simple filters. These kernel machines have memory cells that are updated while the sequential data comes in. The authors introduce a method of adaptive-gating for the adaptivity of memory of the memory cells of Recurrent Kernel Machines. And it is observed that the obtained RKM is closely related to the LSTM. Furthermore, Gated CNN and RAN are obtained from this LSTM-like framework by turning off some elements.

In this reproducibility report, we study in detail, the model architecture proposed in the original paper, run the experiments, provide insights and suggestions for replicating the result, and analyze the results obtained in comparison with the ones reported by the original paper.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Model	Parameters	Input	Cell	Output
LSTM [15]	$(nm+d)(4d)$	$z_t' = [x_t, h_{t-1}']$	$c_t = \eta_t \odot \tanh(\tilde{c}_t) + f_t \odot c_{t-1}$	$h_t' = o_t \odot \tanh(c_t)$
RKM-LSTM	$(nm+d)(4d)$	$z_t' = [x_t, h_{t-1}']$	$c_t = \eta_t \odot \tilde{c}_t + f_t \odot c_{t-1}$	$h_t' = o_t \odot c_t$
RKM-CIFG	$(nm+d)(3d)$	$z_t' = [x_t, h_{t-1}']$	$c_t = (1-f_t) \odot \tilde{c}_t + f_t \odot c_{t-1}$	$h_t' = o_t \odot c_t$
Linear Kernel w/ o_t	$(nm+d)(2d)$	$z_t' = [x_t, h_{t-1}']$	$c_t = \sigma_i^2 \tilde{c}_t + \sigma_f^2 c_{t-1}$	$h_t' = o_t \odot c_t$
Linear Kernel	$(nm+d)(d)$	$z_t' = [x_t, h_{t-1}']$	$c_t = \sigma_i^2 \tilde{c}_t + \sigma_f^2 c_{t-1}$	$h_t' = \tanh(c_t)$
Gated CNN [10]	$(nm)(2d)$	$z_t' = x_t$	$c_t = \sigma_i^2 \tilde{c}_t$	$h_t' = o_t \odot c_t$
CNN [18]	$(nm)(d)$	$z_t' = x_t$	$c_t = \sigma_i^2 \tilde{c}_t$	$h_t' = \tanh(c_t)$

Figure 1: Model Detail.

2 Proposed Method

To establish a relation between RNNs and RKMs. The authors proposed that $y_t = Uh_t$ where h_t is given by

$$h_t = f(W_t^{(x)}) + W_t^{(h)} - 1 + b)$$

Here x_t is the embedding vector for the t-th word (w_t) in a sequence of words. Now further U is factorized as $U = AE$. Thus writing $y_t = Ah_t^{'}$ where $h_t^{'} = Eh_t$.

The paper focuses on the kernel which is of the form

$$k_\theta(\tilde{z}, z_t) = q_\theta(z^T z_t) = \tilde{h_1^T} h_t$$

To evaluate the kernel, the authors used the recursive relationship. The overall model with added feedback was then given by

$$h_t^{'} = q_\theta(c_t), c_t = \tilde{c}_t + q_\theta(c_{t-1}), \tilde{c}_t = \tilde{X}x_t + \tilde{H}h_{t-1}$$

Here c_t is the memory cell at time t and row i of $\tilde{X}$ corresponds to $\tilde{x_i^T}$. The function q_θ can take many forms the simplest of which is a linear time invariant kernel which is modeled by

$$h_t^{'} = c_t, c_t = \sigma_i^2 \tilde{c}_t + \sigma_f^2 c_{t-1}, \tilde{c}_t = \tilde{X}x_t + \tilde{H}h_{t-1}$$

The authors then introduced dynamic forms of σ_i^2 and σ_f^2 and derive LSTM like model called RKM-LSTM and for the linear kernel special cases of σ_i^2 and σ_f^2 yield CNN and Gated-CNN . The above changes are summarized in Figure 1. Complete detail on the proposed method could be found in the original paper

3 Implementation Details

To meet the requirements of the replication task, we implemented the architecture in PyTorch[1], solemnly based on the description provided in the paper. Similar to the original paper, we aim to determine the relation between kernel machines and recurrent model on three tasks, Document Classification, Language Modelling, and Local Field Potential (LFP) classification. However, due to the public unavailability of the LFP dataset, we were able to validate the results on only the Document Classification and the Language Modelling task. Our complete codebase for the reproducibility of the original paper along with installation guidelines could be found at github. In the repository, we have also included the checkpoints for each task. All the experiments were conducted on NVIDIA GeForce GTX 1080 Ti and Google Colab[2].

To replicate the model structure defined in Figure 1, we implemented a general LSTM cell (Figure 2). All the variations were thus obtained by changing the activation functions and gated layers in this cell. As specified in the original paper, we took the values of σ_i^2 and σ_f^2 to be 0.5 for all the four cases i.e. Linear Kernel w/o_t, Linear Kernel, Gated CNN and CNN. A detailed experimental setup for the two tasks can be found in the underneath section.

[1] https://pytorch.org/

[2] https://colab.research.google.com

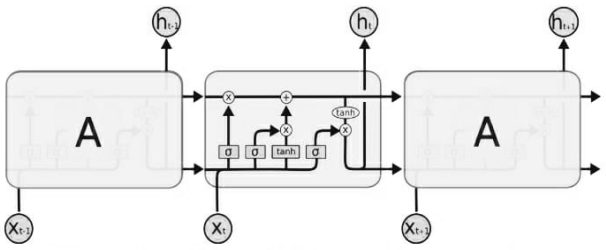

Figure 2: LSTM cell structure.

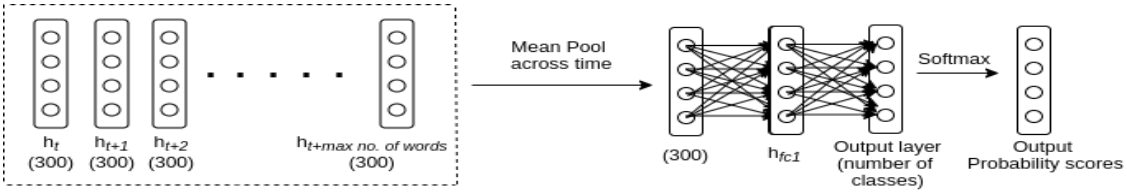

Figure 3: Complete model architecture for Document Classification. Here h_t denotes the output obtained from the recurrent model.

3.1 Document Classification

Figure 3 explains the architecture opted by the original paper for this task. The experimentation for this task was performed on AGNews, Yahoo, DBpedia, and Yelp full dataset. As described in the original paper, we chose GloVe as our word embedding initialization and dimension of hidden states to be 300. The preprocessing procedure was the same as in [5]. Some of the information that was missing in the original paper and our assumed values for them are listed below:

- **The dimension of the h_{fc1} layer (refer Fig 3):** We chose it to be 100 in the case of 1-gram and 200 in the case of 3-gram.
- **Non-Linearity function for the fully connected layers (refer Fig 3):** We took it to be sigmoid.
- **Batch size, number of epoch, and learning rate:** We assumed batch size to vary from 64 to 521 based on the type of dataset. For AGNews, we took it to 64, while for yahoo and DBpedia, we took it to be 256, and 512 for yelp (these values were chosen by taking into account the training time and rate). The number of epoch to be in the range of (20-40) and learning rate to be 0.01 and 0.001.
- **Loss Function and gradient descent optimization algorithm:** We performed the experiments using Pytorch's implementation of cross-entropy loss and [3]Adam as the optimizer.

Besides all these, authors have also mentioned incorporating Layer Normalization [6]. We believed one possible reason for this step could be faster training of recurrent cell. But upon experimenting, we found that another primary reason is to provide stable training. As we can see in Figure 1, tanh activation function is not applied in the output layer of RKM-LSTM, RKM-CIFG, Linear Kernel w/o_t, and Gated CNN, which make these structures vulnerable to exploding gradient problem. To tackle this problem, we tried various other techniques like changing the learning rate, batch size, small initialization of hidden cell, applying gradient clipping, but nothing except layer normalization worked.

3.1.1 Result and Discussion:

Considering Figure 4, we can conclude that our observed training accuracy is similar to that of mentioned in the original paper. Although our values deviate from the original values, we feel that we can neglect those deviations, because the reasoning for these deviations could be the variance in the

[3]https://pytorch.org/docs/stable/optim.html,

Model	Parameters		AGNews					
			1 gram			3 gram		
	1 gram	3 gram	authors	ours	deviation	authors	ours	deviation
LSTM	720K	1.44M	91.82%	91.64%	0.18%	**92.46%**	92.31%	0.15%
RKM-LSTM	720K	1.44M	91.76%	91.75%	0.01%	92.28%	91.89%	0.39%
RKM-CIFG	540K	1.08M	**92.29%**	**91.76%**	0.53%	92.39%	**92.42%**	-0.03%
Linear Kernel w/ot	360K	720K	92.07%	91.42%	0.65%	91.49%	91.21%	0.28%
Linear Kernel	180K	360K	91.62%	89.50%	2.12%	91.50%	89.59%	1.91%
Gated CNN	180K	540K	91.54%	91.15%	0.39%	91.78%	91.65%	0.13%
CNN	90K	270K	91.20%	89.56%	1.64%	91.53%	89.70%	1.83%

Model	Parameters		DBPedia					
			1 gram			3 gram		
	1 gram	3 gram	authors	ours	deviation	authors	ours	deviation
LSTM	720K	1.44M	98.98%	98.83%	0.15%	98.97%	**98.97%**	0.00%
RKM-LSTM	720K	1.44M	98.97%	**98.98%**	-0.01%	99%	**98.97%**	0.03%
RKM-CIFG	540K	1.08M	**98.99%**	98.93%	0.06%	**99.05%**	98.92%	0.13%
Linear Kernel w/ot	360K	720K	98.96%	98.92%	0.04%	98.94%	98.95%	-0.01%
Linear Kernel	180K	360K	98.65%	98.32%	0.33%	98.77%	98.59%	0.18%
Gated CNN	180K	540K	98.37%	97.93%	0.44%	98.77%	98.78%	-0.01%
CNN	90K	270K	98.17%	98.39%	-0.22%	98.52%	97.88%	0.64%

Model	Parameters		Yahoo!					
			1 gram			3 gram		
	1 gram	3 gram	authors	ours	deviation	authors	ours	deviation
LSTM	720K	1.44M	**77.74%**	**76.39%**	1.35%	77.72%	76.52%	1.20%
RKM-LSTM	720K	1.44M	77.70%	76.30%	1.40%	77.72%	76.64%	1.08%
RKM-CIFG	540K	1.08M	77.71%	76.13%	1.58%	**77.91%**	76.69%	1.22%
Linear Kernel w/ot	360K	720K	77.41%	76.26%	1.15%	77.53%	**76.70%**	0.83%
Linear Kernel	180K	360K	76.93%	74.41%	2.52%	76.53%	75.61%	0.92%
Gated CNN	180K	540K	72.92%	63.92%	9.00%	76.66%	71.57%	5.09%
CNN	90K	270K	72.51%	73.77%	-1.26%	75.97%	74.29%	1.68%

Model	Parameters		Yelp Full					
			1 gram			3 gram		
	1 gram	3 gram	authors	ours	deviation	authors	ours	deviation
LSTM	720K	1.44M	**66.27%**	64.15%	2.12%	66.37%	64.93%	1.44%
RKM-LSTM	720K	1.44M	65.92%	64.14%	1.78%	**66.43%**	**65.68%**	0.75%
RKM-CIFG	540K	1.08M	65.93%	**64.80%**	1.13%	65.92%	65.54%	0.38%
Linear Kernel w/ot	360K	720K	65.35%	61.62%	3.73%	65.94%	65.30%	0.64%
Linear Kernel	180K	360K	61.18%	61.37%	-0.19%	62.11%	61.18%	0.93%
Gated CNN	180K	540K	60.25%	62.07%	-1.82%	64.30%	64.50%	-0.20%
CNN	90K	270K	59.77%	55.19%	4.58%	62.08%	61.02%	1.06%

Figure 4: Document classification accuracy for original paper and our implementation on various models. Total parameters of each model are shown, excluding word embedding and the classifier

choice of hyperparameters like learning rate and batch size. Along with this, Figure 4 also justifies the following claims done in the original paper.

- RKM-LSTM and RKM-CIFG performs comparably to LSTM across all datasets.
- Classification accuracy decreases as the recurrent model become less sophisticated regarding gating and memory cell. A subtle decrement is observed in the case of yelp, while for AGNews and DBpedia, 1-gram CNN performs significantly well.
- N-gram (where n>1) archives performs well as compared to 1-gram.

3.1.2 Reproducibility Cost:

The datasets AGNews and DBpedia required lesser time in comparison to the Yahoo! and Yelp-Full datasets. Even though the size of Yahoo! dataset was twice as big as the Yelp dataset, the latter took maximum training time per epoch due to the complexity in its structure. The required time not only varies with model-structure choice but also due to the change in the number of parameters. Refer Figure 5 for the exact values of the average epoch time on NVIDIA GeForce GTX 1080 Ti and Google Colab.

Model	AGNews		DBPedia		Yahoo!		Yelp Full!	
	1 gram	3 gram	1 gram	3 gram	1 gram	3 gram	1 gram	3 gram
LSTM	2 mins	2 mins	4 mins	8 mins	7 mins	10 mins	40 mins	26 mins
RKM-LSTM	2 mins	2 mins	25 mins	8 mins	11 mins	17 mins	40 mins	26 mins
RKM-CIFG	2 mins	2 mins	4 mins	7 mins	10 mins	16 mins	1 hr 1 min	24 mins
Linear Kernel w/ ot	1 min	2 mins	5 mins	6 mins	8 mins	7 mins	52 mins	22 mins
Linear Kernel	1 min	1 mins	2 mins	5 mins	7 mins	6 mins	18 mins	19 mins
Gated CNN	1 min	3 mins	3 mins	4 mins	7 mins	6 mins	20 mins	21 mins
CNN	1 min	1 min	2 mins	3 mins	3 mins	4 mins	19 mins	19 mins

Figure 5: Average Epoch Time for document classification.

3.2 Language Modeling

Similar to the author's approach, We also adopted the AWD-LSTM [7] as our base model[4] and replaced the standard LSTM layer with the models defined in Figure 1. To perform experiments for Language Modeling, we wrote a wrapper around the modified cell (RKM-LSTM, RKM-CIFG, LSTM, Linear Kernel w/o_t). The wrapper provided an interface similar to the LSTM layer implementation in Pytorch. The authors had mentioned using the default parameters of the base model, and the default value of the number of epoch was 8000. On experimentation, we found that the average training time per epoch for this model was around 45 seconds, which made this experiment computationally expensive. As a consequence of this, we were unable to perform all the variations defined for this task. Our result for this part only includes training of the RKM-CIFG model on the Penn Tree Bank dataset. For this case, we got 173.63 test PPL and 195.98 valid PPL, which didn't match with the values given in the original paper.

4 Conclusion & Future Work

Due to the lack of time, computing resources, and unavailability of a dataset, we reproduce results from some of the selected experiments from the original paper. Based on the empirical results obtained from the document classification task, we affirm the author's claim of modifying recurrent kernel machines to a model that is closely related to the LSTM model. Moreover, we would also like to add that in spite, we obtained a major deviation for the result in the language modeling task, we can't draw a conclusion from that. Because we believe that it can be improved by increasing the number of epochs and tuning the hyperparameters with discussion with the authors.

As the author said, the decrease in classification accuracy of the model with a decrease in the sophistication of gated and memory cells is because of the complex structure of the Yelp dataset. So in future we would like to observe the model's performance on other more complex tasks like machine translation, question answering. The authors have incorporated only mercer's kernel. So, we would also like to investigate the effect of other different kernels like Polynomial, Gaussian [8] on bridging the gap between kernel machines and recurrent models.

References

[1] Kevin Liang, Guoyin Wang, Yitong Li, Ricardo Henao, and Lawrence Carin. Kernel-based approaches for sequence modeling: Connections to neural methods. In *Advances in Neural Information Processing Systems*, pages 3387–3398, 2019.

[2] Alberto Bietti and Julien Mairal. Invariance and stability of deep convolutional representations. In *Advances in neural information processing systems*, pages 6210–6220, 2017.

[3] Fabio Anselmi, Lorenzo Rosasco, Cheston Tan, and Tomaso Poggio. Deep convolutional networks are hierarchical kernel machines. *arXiv preprint arXiv:1508.01084*, 2015.

[4] Michiel Hermans and Benjamin Schrauwen. Recurrent kernel machines: Computing with infinite echo state networks. *Neural Computation*, 24(1):104–133, 2012.

[4] https://github.com/salesforce/awd-lstm-lm

[5] Guoyin Wang, Chunyuan Li, Wenlin Wang, Yizhe Zhang, Dinghan Shen, Xinyuan Zhang, Ricardo Henao, and Lawrence Carin. Joint embedding of words and labels for text classification. *arXiv preprint arXiv:1805.04174*, 2018.

[6] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[7] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

[8] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.