UNDER REVIEW

RESCIENCE C

Replication / ML Reproducibility Challenge 2022

[Reproducibility Report] 8-bit Optimizers via Block-wise Quantization

Anastasia Blitsi[1], Dimitrios Sainidis[1, ID] Nikolaos Dragatis[1]

[1] Aristotle University of Thessaloniki

Edited by
(Editor)

Reviewed by
(Reviewer 1)
(Reviewer 2)

Received
29 January 2023

Published
–

DOI
–

Reproducibility Summary

The following, is a reproducibility report for the paper titled "8-bit Optimizers via Block-wise Quantization" published in the the **International Conference** on **Learning Representations 2022** (ICLR 2022) [1] as part of the ML Reproducibility Challenge 2022 [2]. The code and the data used are available through GitHub.

Scope of Reproducibility – In the original paper [3] the authors develop 8-bit optimizers (e.g., Adam, AdamW, and Momentum) to replace their 32-bit counterparts. They claim their optimizers can reduce video memory usage (VRAM) and training time by a significant amount while maintaining the performance of 32-bit optimizers. They test the performance of 8-bit optimizers on a range of publicly available benchmarks on Natural Language Processing (NLP) and Computer Vision (CV) tasks. We applied their 8-bit optimizer Adam on NLP-only tasks using different datasets and models.

Methodology – We reimplement the publicly available code provided by the authors on PyPi using smaller datasets and smaller models due to our lack of access to similar hardware (GPUs). We used Google Colab to run our code.

Results – We verify 2 of the 3 claims of the original paper namely the savings in memory usage and training time. We failed to prove the claim regarding the equal performance of 32-bit and 8-bit optimizers.

What was easy – The code is available as a Python package on PyPI. It is easy to install and well-documented with provided tutorials. The datasets and models used by the authors are publicly available.

What was difficult – The authors use powerful and expensive hardware (multiple V100 and A100 GPUs) making reproducibility with the same data and models impossible.

Communication with original authors – There was no communication with the original authors.

Correspondence should be addressed to Dimitrios Sainidis (dsainidis@csd.auth.gr)
The authors have declared that no competing interests exists.
Code is available at https://github.com/nikosdraga/8-Bit-Optimizers-Reproducability-Study.

1 Introduction

The state of the art is shifting towards increasingly larger models with millions or even billions of trainable parameters. Training these large models requires storing the model itself, the gradients, and the state of the optimizer all in working memory. This problem is usually addressed with distributed systems and parallel training. The authors of the original paper focus on developing 8-bit optimizers to replace their 32-bit counterparts, stating that this could help reduce the memory footprint by up to 75% and also reduce the training time without sacrificing performance.
In this work, we test the newly developed 8-bit optimizer on three NLP tasks using different models and different data than the ones used by the original authors. We perform text classification using LSTM, sentiment analysis using LSTM, and sentiment analysis using BERT. Across our testing, the only differentiating factor between runs is the optimizer and we take appropriate steps to eliminate any stochastic variable. We average the **total training time**, the **maximum memory used**, and an **appropriate metric** relevant to the task across 5 runs. We present our findings and compare them with the ones from the original paper.

2 Scope of reproducibility

In the original paper, the authors present and attempt to compare their 8-Bit Adam optimizer with other 32-Bit optimizers like Adam, Adafactor, and Momentum using a variety of models and datasets on the NLP and CV tasks. In this work, we investigate the reproducibility of the following claims:

- Reduction of VRAM usage during training.
- Reduction of time of completion during training.
- No degradation in performance during training.

We validate 2 of the 3 claims above, namely the reduction of memory used and time, and fail to validate the claim regarding performance. We use models and datasets available through Kaggle [4] to reproduce the results presented in the original paper. We focus only on NLP tasks (Text Classification and Sentiment Analysis) and experiment with the hyper-parameters of the models to strengthen our results.

3 Methodology

We implement the authors' code, which is publicly available on PyPI using datasets and models from Kaggle. This is done due to our limited access to capable GPUs. The authors use multiple V100 and A100 GPUs and train their models in the span of days. We use Google Colab, which limited our ability to perform diverse experiments with more complex models.

3.1 Model descriptions

The three models used are described below

LSTM Text Classification - PyTorch The neural network in this training example comprises of an LSTM layer to process vector sequences, followed by a Dense layer and a Sigmoid activation function to map values between 0 and 1. This is done in binary classification tasks where the output is expected to be a probability of the input belonging

to one of two classes.

Sentiment analysis using LSTM - PyTorch

The neural network used in this training example utilizes an LSTM layer to process sequential data. A dropout layer is used next to prevent overfitting by randomly dropping out a certain proportion of the input units during training, helping the model to better generalize. The output of the LSTM layer is passed through a linear layer, which applies a linear transformation to the data. The activation function used is the Softmax function. The combination of LSTM, Dropout, Linear Layer and Softmax function work together to model the sequential data and predict the probability of the output being in a certain class (sentiment).

Sentiment Analysis using BERT | PyTorch

The neural network used in this training example utilizes a BERT (Bidirectional Encoder Representations from Transformers) pretrained model. BERT is a transformer-based neural network architecture that is trained on a massive amount of text data to understand the context of words in a sentence. This pre-trained model is then fine-tuned for the specific task at hand, such as text classification or language understanding. BERT has shown state-of-the-art results in a wide range of NLP tasks and is considered one of the most powerful models in the field.

3.2 Datasets

Sentiment analysis using LSTM - PyTorch

IMDB Dataset of 50K Movie Reviews

The IMDB dataset contains 50,000 movie reviews for NLP or text analytics. It provides a set of 25,000 highly polar movie reviews for training and 25,000 for testing. The dataset is preprocessed to use a BucketIterator which groups similar-length samples together and reduces the need for padding. This approach allows for more efficient processing of the data, as the model does not need to handle sequences of varying lengths. By batching similar-length samples, it reduces the computation time and memory requirements.

LSTM Text Classification - Pytorch

GloVe: Global Vectors for Word Representation

This dataset contains pre-trained English word vectors that have been trained on a large corpus of text data, consisting of the combined Wikipedia 2014 and Gigaword 5th Edition corpora, which totals 6 billion tokens and a vocabulary of 400,000 words. The dataset contains pre-trained word vectors in three different dimensions: 50, 100, and 200. The dataset has been preprocessed to remove all non-word characters, replace runs of whitespaces with no space, replace digits with no space and tokenize the dataset. Padding is added to each sequence to make them of max length. An embedding layer is added to the dataset because there are fewer words in the vocabulary. Instead of using one-hot encoding, the embedding layer is used as a lookup table. This embedding layer can be trained using Word2Vec and then loaded, but it is also possible to create a new layer and use it only for dimensionality reduction, letting the network learn the weights.

SMS Spam Ham Prediction

The dataset contains 5,574 SMS messages with one column beeing the actual message and the other classifying it as either SPAM(trash) or HAM(legitimate). There are 4,827 messages classified as SPAM and 747 messages classified as HAM. Padding tokens were added to the dataset, which help the model be more stable. An embedding layer has been applied to the dataset, which converts the integer sequences to vector sequences. This allows for more efficient processing and representation of the data in the model.

Sentiment Analysis using BERT | PyTorch

SMILE Twitter Emotion Dataset
The dataset contains 3,085 tweets, each representing a text message with an associated emotion. The emotions present in the dataset are anger, disgust, happiness, surprise, and sadness. The dataset has been preprocessed by using the BERT tokenizer, which is a pre-trained tokenizer. The tokenizer is used to encode the data, which helps to prepare the data for further processing. This tokenization step helps to convert the text data into numerical representations that can be used as input for a neural network. The encoding of the data with the tokenizer allows for more efficient processing and better representation of the data in the model.

3.3 Hyperparameters

LSTM Text Classification - Pytorch
In this neural network, the size of the vocabulary is set to the length of the text vocabulary. The embedding dimension is set to 100, which means that each word in the vocabulary is represented by a 100-dimensional vector. The number of hidden nodes in the network is set to 64, and the number of output nodes is set to 1. The number of layers in the network is 2 and bidirectional is set to true. This means that information flows in both directions in the network, allowing for better context representation. A dropout of 0.2 is also applied to the network, which serves as a regularization technique to prevent overfitting. Finally, the number of training epochs is set to 100, which means that the network will be trained on the data for 100 iterations.

Sentiment analysis using LSTM - PyTorch
In this neural network, the number of layers is set to 2. The size of the vocabulary is set to the length of the vocabulary plus one, this extra one is for the padding. The embedding dimension is set to 64, which means that each word in the vocabulary is represented by a 64-dimensional vector. The output dimension is set to 1, and the hidden dimension is also set to 64. The learning rate is set to 0.001, this determines the step size at which the optimizer makes updates to the model's parameters. Finally, the number of training epochs is set to 5, which means that the network will be trained on the data for 5 iterations. This configuration of the hyperparameters allows for a compact model with a moderate number of iterations for training, which may be beneficial for certain use cases.

Sentiment Analysis using BERT | PyTorch
In this neural network, the batch size is set to 4, which means that the model will be trained on 4 samples per iteration. The number of training epochs is set to 10, which means that the network will be trained on the data for 10 iterations. This configuration of the hyperparameters allows for a moderate number of iterations to train the model and a smaller batch size that can be useful when the dataset is not very large. A smaller batch size allows the model to adapt to the variations in the dataset more quickly, but it can also increase the training time.

3.4 Experimental setup and code

The neural network used in this training utilized both LSTM and BERT models for text classification and sentiment analysis tasks. The LSTM model was used for text classification, where it was able to process sequential data and make predictions about the class of the text. The LSTM model was also used for sentiment analysis, where it was able to understand the context of the text and predict the sentiment expressed. BERT, a transformer-based model pre-trained on a massive amount of text data, was also utilized for sentiment analysis. This powerful model was able to understand the context of the text and make accurate predictions about the sentiment expressed. All models were fine-tuned using the 8-bit Adam optimizer on Google Colab's GPU. To compare the

results, this experiment also ran the models with standard Adam optimizer. By using the 8-bit Adam optimizer, the model was able to learn more efficiently and quickly. The use of Google Colab's GPU limited the training process but made it possible to train these models. The results of the comparison showed that using 8-bit Adam optimizer can significantly improve the model's performance.
The metric used to evaluate the performance of an LSTM model, for text classification in Pytorch, was validation loss, which was also the metric used for Sentiment Analysis using LSTM in Pytorch.Finally, for Sentiment Analysis using BERT in Pytorch, the metrics used to evaluate the performance of the model were validation loss and F1 Score (weighted).

3.5 Computational requirements

For the experiments, we used Google Colab with its GPU capabilities, so we used small and simple models. It is important to note that the optimizer needs to run on a Linux operating system. The authors of the study utilized multiple GPUs in order to run the experiments efficiently.

4 Results

We were, only, able to verify 2 of the 3 claims of the original paper, specifically the reduction in memory usage and in training time. Our experiments using the 8-Bit Adam Optimizer, failed to maintain the same performance as its 32-Bit counter part. Also, there is only one case where the training time did not improve, as it is shown in the table below.

4.1 Results reproducing original paper

Result 1 — In this experiment, we used a LSTM model for text classification using a dataset containing SMS messages which are either spam or normal. We used 50 and 100 epochs for training for each optimizer to see if there would be any changes in performance, training time and memory usage. As it is shown in table 1, in both cases using the 8-Bit Adam does decrease the training time and memory usage but does not maintain the performance of the 32-Bit Adam. The differences may seem small but they are depicted by the scale and complexity of the model. Also, we observe that the memory save is proportional to the total memory usage.

Optimizer	Adam 32-Bit		Adam 8-Bit	
Number of Epochs	50	100	50	100
Metric (Val. Loss)	**0.085**	**0.148**	0.102	0.232
Time (Seconds)	32.026	64.274	**29.493**	**58.991**
Memory Used (MB)	196.46	422.36	**182.79**	**402.04**

Table 1. LSTM Text Classification results

Result 2 — In this experiment, we used a LSTM model, different than the one used in the first experiment, for sentiment analysis. The dataset contains positive and negative reviews from IMDB. For each optimizer, we used 2, 4 and 6 layers to see differences in performance, memory saved and training time. We can see that the results resemble those of the first experiment. Here, too, the performance is worst for the 8-Bit Adam in all cases. In training time there is an improvement in 2 of the 3 cases. Namely, when using 4 layers the training time increases with the 8-Bit Adam. Finally, the memory usage is lower in all cases for the 8-Bit Adam albeit the differences are small.

Optimizer	Adam 32-Bit			Adam 8-Bit		
Number of Layers	2	4	6	2	4	6
Metric (Val. Loss)	**0.337**	**0.351**	0.693	0.588	0.693	0.693
Time (Seconds)	143.513	**263.314**	390.358	**138.941**	273.591	**375.265**
Memory Used (MB)	151.36	225.61	301.11	**150.62**	**224.51**	**298.89**

Table 2. LSTM Sentiment Analysis results

Result 3 — For this experiment, we used a pretrained BERT model on a massive amount of text data and fine tuned it for this specific task, sentiment analysis. The dataset contains tweets, each representing a text message associated with an emotion: anger, disgust, happiness, surprise, sadness. From the results, we can see that the performance of the 8-Bit Adam is worse too. Validation loss is quite higher and F1 Score is lower. However, the training time has been improved quite a bit, almost 10% and the memory usage is 23% lower.

Optimizer		Adam 32-Bit	Adam 8-Bit
Metric	Val. Loss	**0.684**	3.586
	F1 Score (weighted)	**0.876**	0.666
Time (Seconds)		737.112	**673.059**
Memory Used (MB)		2816.19	**2171.2**

Table 3. BERT Sentiment Analysis

5 Discussion

Based on our experiments we can verify that 2 of the 3 claims of the original paper are true. Namely the reduction in memory usage and training time. We failed to maintain the same performance when using the 8-Bit Adam optimizer, which can be attributed to a number of factors. Firstly, the models we used were quite small and simple for the tasks given due to hardware limitations.

5.1 What was easy

It was really easy to implement the code of the original paper. It is available as a Python package on PyPi and the installation is pretty straight forward. To use the 8-Bit Adam, one has to just import the class and replace the 32-Bit Optimizer without even changing the parameters.

5.2 What was difficult

The difficult part was making and training the models. Even though we used small models and datasets, the training times in some cases were quite high. Furthermore, because we used Google Colab, we had to upload the datasets each time the environment would get stuck due to an error, which held back our experiments even more.

5.3 Communication with original authors

There was no communication with the authors of the original paper as it was not necessary.

References

1. ICLR. **International Conference on Learning Representations**. Accessed: 2023-01-20. 2023. URL: https://iclr.cc/.
2. MLRC 2022. **ML Reproducibility Challenge 2022**. Accessed: 2023-01-20. 2022. URL: https://paperswithcode.com/rc2022.
3. T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer. "8-bit Optimizers via Block-wise Quantization." In: **arXiv preprint arXiv:2110.02861** (2021).
4. **Kaggle**. Accessed: 2023-01-20. 2022. URL: https://www.kaggle.com/.