Kalpa Publications in Computing
Volume 20, 2024, Pages 74–80
Proceedings of 2024 Concurrent Processes Architectures and Embedded Systems Hybrid Virtual Conference

Benchmarking Python Deep Learning Frameworks for Language Modeling on GPUs

Vijaya Laxmi Pachva

vaishnavivijayalaxmi.p@gmail.com

Abstract. Neural networks are omnipresent in natural language processing (NLP). We benchmark three popular Python frameworks (DyNet, TensorFlow, and Theano) on the standard NLP task of language modeling, and find that DyNet is signifi- cantly faster on this task. We also discuss other bottlenecks beyond performance, such as ease of use, that may impact the selection of a neural network framework.

1 Introduction

Deep learning is witnessing an extraordinary surge in popularity due to its performance gains over traditional systems on many tasks. Several deep learning frameworks have been released in response to this, each claiming niche improvements over others, but it is not necessarily clear which frameworks are better suited for which types of applications. Previous comparisons between frameworks may now be out-of-date (older versions; lack of newer frameworks) or were run against simpler tasks (which might have hidden other interesting performance characteristics).
In this project, we benchmark deep learning frameworks on a complex natural lan- guage processing (NLP) task and rank them based on different performance metrics. Specifically, we look at the following frameworks:

– TensorFlow [3]
– Theano [7], using the Keras wrapper [4]
– DyNet [1]

Recurrent neural networks (RNNs), by working well with the sequential nature of language, are proving to be a promising technique in research activities related to NLP. Benchmarks on RNNs have been run before, but most tasks were benchmarked on toy tasks which do not necessarily reflect performance on real data. For instance, GitHub user `glample` generated random data to run through their RNN [2], which is not realistic. A standard approach in NLP is to transform words into vocabulary-sized one-hot vectors and then embed them as hidden-sized vectors (i.e., dimensionality re- duction). Sparse learning of the embedding matrix, which is usually in the order of
$[20,000 \times 200]$, is likely to severely impact performance.

2 Background

2.1 Language Modeling

The task of language modeling is about estimating probabilities of particular sequences of words. It is used as an invaluable component in various real-world applications such as speech recognition and statistical machine translation. More formally, if $s =$
$\{START, x_1, x_2, \ldots, x_n, STOP\}$ is an n-word sentence, language modelling will try to es- timate the probability of each word x_i given its history: $p(x_i | START, x_1, x_2, \ldots, x_{i-1})$.
Historically, this task was done by simply using an l-th order Markov model:

$$p(x_i | START, x_1, x_2, \ldots, x_{i-1}) =$$
$$p(x_i | x_{i-l}, x_{i-l+1}, \ldots, x_{i-1})$$

However, the Markov assumption does not need to be made with RNNs, as they are capable of encoding longer histories. With RNNs, we normalize (using softmax) the output of a nonlinear function f:

$$f(p(x_i | START, x_1, x_2, \ldots, x_{i-1})) =$$
$$softmax(f(x_i, START, x_1, x_2, \ldots, x_{i-1}))$$

2.2 Frameworks

All three frameworks discussed - TensorFlow, Theano, and DyNet - utilize symbolic computation graphs for modeling neural networks. This approach maps operations and variables to nodes in a graph structure, facilitating efficient computation. However, since the publication of this paper, several new deep learning frameworks have emerged, each with its unique features and advantages.

– PyTorch: PyTorch has gained significant traction in the deep learning community due to its dynamic computation graph, which is similar to DyNet's approach. It allows for intuitive model construction and easy debugging, making it a popular choice for researchers and developers alike.
– JAX (and Flax): JAX, along with its high-level API Flax, has become increasingly popular for its combination of flexibility and performance. JAX allows for compos- able function transformations and automatic differentiation, making it suitable for both research and production-level deployments.
– Hugging Face Transformers: While not a traditional deep learning framework, Hug- ging Face Transformers provides pre-trained models and pipelines for natural lan- guage processing tasks, leveraging frameworks like PyTorch and TensorFlow. Its simplicity and extensive model zoo have made it a go-to choice for many NLP practitioners.
– Swift for TensorFlow: Swift for TensorFlow, an experimental project by Google, combines the flexibility of Swift programming language with the power of Tensor- Flow. It aims to provide a more intuitive and expressive interface for building deep learning models.
– MXNet: MXNet, although not as widely used as TensorFlow or PyTorch, remains a strong contender in the deep learning landscape. Its support for dynamic compu- tation graphs and efficient distributed training makes it suitable for a wide range of applications.

In addition to TensorFlow, Theano, and DyNet, researchers and practitioners may also consider these newer frameworks when selecting tools for their deep learning projects. Each framework has its strengths and weaknesses, and the choice ulti- mately depends on the specific requirements of the task at hand.

3 Experiments

3.1 Data & Preprocessing

We chose to focus on the Los Angeles Times subset from 2009 of the GigaWord corpus [6]. Each of these documents is a news article, so we used NLTK's [5] sentence splitter on the articles. Every sentence is then tokenized using NLTK's `casual_tokenize` function, limiting ourselves to sentences that are $<$ 100 tokens long. In order to limit overfitting, we replace all words that occurred less than 150 times by a special `OOV` symbol (this allows the model to be more robust when encountering previously unseen words). We ended up with $1,191,848$ sentences, $27,269,856$ total word tokens and a vocabulary size of $35,642$.

3.2 Implementation & Hyperparameters

We implemented the same RNN language model in each framework. The model con- sists of a matrix of input word embeddings, which are passed through a RNN to get predicted word embeddings and then compared (with a loss function) to a correspond- ing set of output embeddings. All these elements are available in all frameworks (in Theano's case, the RNN is provided by the Keras wrapper). Our implementations are made available on GitHub.[1]
In our experiments, we optimized for a standard language modeling objective: *cross- entropy*. The built-in cross-entropy loss functions in each framework were slightly dif- ferent, which could potentially affect the number of iterations required for convergence. Table 1 summarizes the hyperparameters chosen for the experiments; there was no cross-validation done since the evaluative performance of the model trained is irrele-vant.

hyperparam	value
RNN type	`LSTM`
hidden size	256
optimizer	`AdamOptimizer`
learning rate	0.003
batch size	25

Table 1. Hyperparameters used in the experiments

Traditionally, neural models are trained until their performance on a *development* set stops improving. To determine if we should terminate, each full epoch of training (where all the training data is used) was followed by a predictive pass through the development data.

[1] `https://github.com/lucylin/neural-benchmarks`

4 Benchmarks

4.1 Time

The first benchmark to look at is how long training takes. Table 2 summarizes various breakdowns. Overall, DyNet seems to converge the fastest of the three; its epoch time is dramatically shorter, but it also took more epochs to converge.

Table 2. Total runtime on various subtasks

Task	DyNet	TensorFlow	Theano
Convergence	26h21m	35h25m	-
Train epoch	2h20m	6h57m	6h35m
Test epoch	3m4s	7m45s	24m
Train batch	1s	5s	1s
Test batch	<1s	<1s	<1s

4.2 Kernel function usage

Using NVidia's built-in GPU profiler, nvprof, we first looked into which kernel func- tion calls were most heavily utilized by profiling training on a single batch. Table 3 shows how ample runtime is spent on matrix multiplication (specifically, a variant of `magma lds 128 sgemm kernel()`) for all frameworks. This is expected given that neural network operations are heavily dependent on matrix multiplies.

Table 3. Percentage of runtime spent on the most common matrix multiply call used and on CUDA memset operations.

Function	DyNet	TensorFlow	Theano
Matrix multiply	29.64%	68.87%	49.90%
CUDA memset	40.13%	0.00%	0.24%

4.3 CUDA profiler metrics

Given the high usage of the matrix multiply operation `magma lds128 sgemm kernel()`, we then looked into a variety of CUDA profiler metrics for this operation. We collected data for various metrics as summarized in Table 4.

We found that all of the measurements were within similar orders of magnitude; nothing at this level directly points to why DyNet performance was considerably faster than the other frameworks.

Table 4. Various CUDA profiler metrics for the task of training a single batch.

Metric	**DyNet**	**TensorFlow**	**Theano**
Achieved Occupancy	0.062	0.062	0.062
SM Efficiency (%)	16.64	22.58	16.04
Warp Efficiency (%)	100.0	100.0	99.99
Warp Nonpred Efficiency (%)	100.0	99.95	99.91
Global Hit Rate	2.88%	0.00%	0.00%
Local Hit Rate	0.00%	0.00%	0.00%
DRAM Read Throughput (GB/s)	10.07	11.07	9.97
DRAM Write Throughput (MB/s)	357.63	411.99	503.54

5 Discussion

Selecting a neural network framework involves several considerations. In this section, we describe how performance and various aspects of the programming experience may impact such a selection.

5.1 Performance

As discussed in section 4, we found that DyNet was the fastest at performing our spe- cific language modeling task. DyNet seems to make certain optimizations that are ad- vantageous in this setting, perhaps in the memory management choices it makes (as discussed in **??**). We also suspect (but could not empirically confirm based on metrics) that because DyNet dynamically creates computation graphs, it might be better able to adapt to different sentence lengths in a way that TensorFlow and Theano cannot.
Despite our performance findings, we note that the comparative performance be- tween frameworks is likely dependent on the task, data set, choice of optimizer, and other parameters. Therefore, these results may not extrapolate to other settings, and if tuning training or prediction performance is a serious consideration, we recommend benchmarking on the specific task at hand.

5.2 Programmer Experience

While implementing language models in these frameworks, we found that it is also important to consider the development experience. No one framework wins or loses out; instead, there are several tradeoffs to consider. We describe some of our experiences below.

Installation/dependencies Unlike a standard Python module which just requires a simple `pip install`, neural network frameworks are typically dependent on exter- nal fast low-level linear algebra libraries to perform efficiently. Therefore there is often linking or configuration required to wire things together.

DyNet We encountered some difficulties in installing DyNet. The library failed to build with the recommended development version of Eigen, so we had to revert to an earlier commit. We also had to use Python 2 for the Python interface, though the documentation says it supports Python 3.

TensorFlow TensorFlow provides very extensive installation guidelines, and pip in- stall takes care of installing it and its dependencies. However, one inconvenience is that it requires separate installations for CPU and GPU usage, which can cause issues if not using a virtual environment of some sort.

Theano Our experience with Theano installation was relatively straightforward: pip install of relevant modules and specification of the BLAS install location.

Ease of use

DyNet DyNet includes built-in constructors for various NLP-relevant models, in- cluding several varieties of RNNs (e.g., tree LSTMs, encoder-decoders). This makes it easy for the programmer to build many common kinds of models. DyNet also allows for dynamic computation graphs, allowing nodes (representing parameter tensors) to be combined "on the fly" to adapt to, for example, data sequences of different lengths. While our neural language model used a static computation graph, DyNet made graph construction easy.
DyNet's primary drawback is the relative lack of documentation and support. The development team and userbase are both fairly small, so there is less information avail- able to help users debug. There is some documentation available online, but it is incom- plete.

TensorFlow TensorFlow is relatively easy to use, as it comes with a myriad of tutorials and examples. TensorFlow's large userbase, polyvalence, and large amount of support from its creator (Google) have made it a very solid option for developers.
There is plenty of support for RNNs and many other standard models. However, writing the computation graph requires some amount of know-how, since there is a lot of TensorFlow-specific syntax, and debugging the symbolic graph can be difficult since there are no values. TensorFlow also sometimes requires a decent amount of boiler- plate code; for example, the computation graph being statically constructed requires that sequences or mini-batches be padded, which can make RNN computation some- what cumbersome.

Theano In contrast to DyNet and TensorFlow, Theano is "lower-level" in that it re- quires that the programmer define the shared variables propagation steps, and so on manually.[2] Theano therefore offers a great deal of control and flexibility in model im- plementation, which is a benefit if one is implementing neural net layers not supported by other frameworks.

[2] An example of the relative implementation complexity can be seen in code for the LSTM Theano tutorial at `http://deeplearning.net/tutorial/lstm.html`.

However, this also presents a steeper learning curve (despite the extensive docu- mentation) and a much higher prototyping/development cost. We originally attempted to implement the Theano language model using just Theano and found the learning/de- bugging costs to be high. Because our language model consists of very standard neural net components, we instead switched to using Keras, which supplies an API with built- in RNN components implemented using Theano.

6 Conclusion

We implemented a simple language modeling task in three deep learning frameworks and measured their performance when run on a GPU. We found that DyNet converged faster than TensorFlow or Theano, but performance analysis on a fine-grained level showed that all frameworks used the underlying GPU resources with comparable effi- ciency.
Through our experience implementing language models in these frameworks, we also found that efficiency is only one of several factors researchers should consider in choosing a deep learning framework. Users should also consider ease of installation and use, level of support available, and the suitability of the framework for their particular task.

References

1. DyNet. https://github.com/clab/dynet (2016), accessed: 2016-11-28
2. RNN benchmarks. https://github.com/glample/rnn-benchmarks (2016), accessed: 2016-12-08
3. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane´, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Vie´gas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), http://tensorflow.org/, software available from tensorflow.org
4. Chollet, F.: Keras. https://github.com/fchollet/keras (2016)
5. Loper, E., Bird, S.: Nltk: The natural language toolkit. In: Proceedings of the ACL-02 Work- shop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1. pp. 63–70. ETMTNLP '02, Association for Com- putational Linguistics, Stroudsburg, PA, USA (2002). https://doi.org/10.3115/ 1118108.1118117, http://dx.doi.org/10.3115/1118108.1118117
6. Parker, R., Graff, D., Kong, J., Chen, K., Maeda, K.: English gigaword fifth edition. Linguistic Data Consortium, Philadelphia (2011), lDC2011T07
7. Theano Development Team: Theano: A Python framework for fast computation of mathemat- ical expressions. arXiv e-prints **abs/1605.02688** (May 2016), http://arxiv.org/abs/ 1605.02688