COMPARATIVE STUDY OF CAFFE, NEON, THEANO, AND TORCH FOR DEEP LEARNING

Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, Mohak Shah
Bosch Research and Technology Center
{Soheil.Bahrampour,Naveen.Ramakrishnan,
fixed-term.Lukas.Schott,Mohak.Shah}@us.bosch.com

ABSTRACT

Deep learning methods have resulted in significant performance improvements in several application domains and as such several software frameworks have been developed to facilitate their implementation. This paper presents a comparative study of four deep learning frameworks, namely Caffe, Neon, Theano, and Torch, on three aspects: extensibility, hardware utilization, and speed. The study is performed on several types of deep learning architectures and we evaluate the performance of the above frameworks when employed on a single machine for both (multi-threaded) CPU and GPU (Nvidia Titan X) settings. The speed performance metrics used here include the gradient computation time, which is important during the training phase of deep networks, and the forward time, which is important from the deployment perspective of trained networks. For convolutional networks, we also report how each of these frameworks support various convolutional algorithms and their corresponding performance. From our experiments, we observe that Theano and Torch are the most easily extensible frameworks. We observe that Torch is best suited for any deep architecture on CPU, followed by Theano. It also achieves the best performance on the GPU for large convolutional and fully connected networks, followed closely by Neon. Theano achieves the best performance on GPU for training and deployment of LSTM networks. Finally Caffe is the easiest for evaluating the performance of standard deep architectures.

1 INTRODUCTION

Deep learning methods have recently influenced several application domains, namely computer vision (Krizhevsky et al., 2012; Russakovsky et al., 2014), speech recognition (Ding et al., 2014; Hannun et al., 2014), and nature language processing (Collobert et al., 2011b), where they have enjoyed significant performance improvements compared to state-of-art methods in the respective domains. For the latest list of domains and challenges on benchmark datasets where deep learning performed better than the existing state-of-art, see `http://deeplearning4j.org/accuracy.html`. Most of the successful deep learning architectures are composed of a combination of different types of layers such as fully connected, convolutional, and recurrent layers and are usually trained with a variant of the stochastic gradient descent algorithm along with various regularization techniques such as dropout and weight decay (Bengio et al., 2015). As the popularity of the deep learning methods have increased over the last few years, several deep learning software frameworks have appeared to enable efficient development and implementation of these methods. The list of available frameworks includes, but is not limited to, Caffe, DeepLearning4J, deepmat, Eblearn, Neon, PyLearn, Theano, Torch, etc. Different frameworks try to optimize different aspects of training or deployment of a deep learning algorithm. For instance, Caffe emphasises ease of use where standard layers can be easily configured without hard-coding while Theano provides automatic differentiation capabilities which facilitates flexibility to modify architecture for research and development. Several of these frameworks have received wide attention from the research community and are well-developed allowing efficient training of deep networks with billions of parameters, thanks to their strong GPU backends. Developers have constantly improved these frameworks by adding more features (e.g. by adding support for different types of convolution algorithms) and speed improvements to attract more users and foster research (Bergstra et al., 2011; Collobert et al., 2011a; Vasilache et al., 2014;

Measures	Caffe	DeepLearning4J	Eblearn	Neon	Theano	Torch7
Number of members in Google groups	3517	819	108	48	2512	1617
Number of contributors in GitHub	158	47	NA	26	182	62

Table 1: Community involvements for some of the deep learning frameworks as of 11/18/2015.

Property	Caffe	Neon	Theano	Torch
Core	C++	Python	Python	Lua
CPU	✓	✓	✓	✓
Multi-threaded CPU	✓Blas	**x** Only data loader	✓Blas, conv2D, limited OpenMP	✓**Widely used**
GPU	✓	✓customized Nvidia backend	✓	✓
Multi-GPU	✓(only data parallel)	✓	**x** Experimental version available	✓
Nvidia cuDNN	✓	**x**	✓	✓
Quick deploy. on standard models	✓**Easiest**	✓	**x** Via secondary libraries	✓
Auto. gradient computation	✓	✓Supports Op-Tree	✓**Most flexible** (also over loops)	✓

Table 2: Properties of Caffe, Neon, Theano, and Torch as of 12/21/2015.

Bastien et al., 2012; Jia et al., 2014). Recently, the efficacy of several deep learning frameworks have been evaluated in Chintala (2015a). However, the comparison is only focused on speed performance of the convolutional frameworks. Hence, this paper expands on the previous benchmarks and evaluates four deep learning frameworks, namely: Caffe, Neon, Theano, and Torch. Among the available software frameworks, Caffe, Theano, and Torch are indeed the top three well developed and widely used frameworks by the deep learning community. The reason for including Neon in this study is its recently reported state-of-the-art performance for training several deep learning architectures (Chintala, 2015a). We evaluate these frameworks from the perspective of practitioners, on the following aspects:

- *Extensibility*: Their capability to incorporate different types of deep learning architectures (convolutional, fully-connected, and recurrent networks), different training procedures (unsupervised layer-wise pre-training and supervised learning), and different convolutional algorithms (e.g. FFT-based algorithm).
- *Hardware utilization*: Their efficacy to incorporate hardware resources in either (multi-threaded) CPU or GPU setting.
- *Speed*: Their speed performance from both training and deployment perspectives.

The study will provide the users and enterprises a broad picture of the strengths and (current) limitations of the studied deep learning frameworks to enable them to assess suitability in the context of their requirements. Moreover, the discussions highlight the current limitations of the respective frameworks which can be addressed in their future developments [1]. We plan to share the code for all the frameworks in the near future through a publicly available webpage.

The rest of the paper is organized as follows: Section 2 gives a brief overview of the software frameworks we focus on in this paper; Section 3 describes the benchmarking set up which is followed by results and conclusions in Section 4 and Section 5, respectively.

2 Overview of the deep learning frameworks

With deep learning methods gaining popularity in many applications domains over the last few years, there have been quite a lot of interest from many academic (e.g. Univ. of California Berkeley,

[1] Note that most of these frameworks have very active community support that keeps adding new features/functionalities potentially making some of our observations obsolete in the near future.

NYU) and industry groups (e.g. Google, Facebook) to develop software frameworks (e.g. Theano, Caffe) that help easily create and test various deep architectures. At the time this paper was written, some of the widely used software frameworks for deep learning were: Caffe, Theano, Torch, Neon, Chainer, DeepLearning4J, deepmat, Eblearn, MXNet, etc. (for a more complete list of Deep Learning Softwares see `http://deeplearning.net/software_links/`). Many of these frameworks are mature already as of today and are very fast in training deep networks with billions of parameters thanks to their strong CUDA backends. Today, almost every group training deep networks use Graphical Processing Units (GPU) to accelerate the training process and this has led to joint development of software libraries (e.g. cuDNN) between academic (e.g. Berkeley, NYU) and industry players (e.g. Nvidia). Table 1 shows the number of users in Google groups and the number of contributors[2] for each of the frameworks in their corresponding GitHub repositories. It is clear that the top three widely developed and supported deep learning frameworks are Caffe, Theano, Torch, and are thus selected in this paper for the benchmarking purposes. We also evaluate Neon framework from Nervana as it has recently shown the state-of-the-art performance for training convolutional networks (Chintala, 2015a). Table 2 shows the general properties of these four deep learning frameworks. Note that, for this paper, we restrict ourselves to frameworks built for single node (with potentially multiple GPUs) but not distributed deep learning frameworks like DeepLearning4J. For a brief review of the selected frameworks see Appendix.

3 BENCHMARKING SETUP

3.1 EVALUATION METRICS

We use the two following evaluation metrics to obtain a holistic understanding of speed of the four deep learning frameworks under various system scenarios and application domains:

- Forward Time: We measure the time it takes for an input batch of a pre-selected batch size, for a given dataset and network, to flow through the entire network and produce the corresponding output. This is important as it indicates the latency of a deep network when deployed in real-world.
- Gradient Computation Time: We also measure the time it takes to get the gradients for each measurable parameter in the deep network for a given input batch. This is an important indicator of the training time. Note that, for most of the cases (e.g. Torch), this gradient computation time is the summation of the times spent in calling the corresponding *forward* and *backward* functions as these two functions should be called consecutively to compute the gradients. But for Theano, this gradient computation time is measured by calling a Theano function that is compiled to generate the gradients given the inputs to the networks which *implicitly* performs the forward and backward steps through computational graphs. It should be noted that the gradient computation time we report, does not include the time taken to update the network parameters, such as computation of learning rate, weight decay, momentum term, etc.

For Theano, one initially need to compile forward and gradient computation functions before calling them during execution. To provide a complete picture, these compilations times are also reported (See Tabel 8). We also report the GPU memory usage for large networks.

3.2 SYSTEM SETUP

All the experiments are performed on a single machine running on Ubuntu 14.04 with Intel Xeon CPU E5-1650 v2 @ 3.50GHz 12; Nvidia GeForce GTX Titan X/PCIe/SSE2; 32 GiB DDR3 memory; and SSD hard drive. We used openCV 3.0.0, CUDA 7.5, cuDNN v3, and OpenBLAS 0.2.14 with commit ID 3684706. For Caffe, commit ID 8c8e832 is used. For Neon, version 1.0.0.rc1 (2015-09-08) with the commit ID of a6766ff is used. Theano version 0.7.0.dev and Torch7 used here have commit IDs 662ea98 and 8c8e832, respectively. The commit ID for fbcunn is 5bb9785. Data arrays are stored using the float32 format.

[2] We only report the number of the contributors in the main repository of the framework. The numbers do not include any other relevant repositories.

4 RESULTS AND DISCUSSIONS

The evaluations are performed by training stacked autoencoders and convolutional networks on the MNIST (LeCun et al., 1998) and the ImageNet datasets (Deng et al., 2009) as well as training LSTM network using the IMDB review dataset (Maas et al., 2011). Note that the evaluation metrics can vary drastically based on the CUDA package used along with the native software. For example, in Torch, one can perform the convolution operations using Nvidia cuDNN library or cunn library (a CUDA backend for the nn package) or fbcunn library (deep learning CUDA extensions from Facebook AI Research containing FFT based fast convolutions). In Theano, it is also straightforward to perform convolution using cuDNN or conv-fft. The conv-fft is a FFT-based implementation of convolution operation on Theano. Hence we try to use as many libraries as possible for each of the cases and measure the performance to present the inherent tradeoffs with each of the libraries. We use the same blas library for Caffe, Theano, and Torch which performs majority of the computations when CPU is used. Neon uses its own CPU/GPU backend. Moreover, wherever applicable, we measure the speeds with both GPU and CPU (single and multi-threaded) so as to understand the hardware specific behaviours of these frameworks from both training and deployment perspectives. We perform several iterations of warm-ups before timing the operations. The timings reported here are average of 20-1000 iterations and are controlled to have small standard deviations.

4.1 LENET

The first benchmark is a slightly modified LeNet neural network on the MNIST dataset (LeCun et al., 1998) where the sigmoid activations are replaced with *ReLU* units and softmax logistic loss layer is used instead of the RBF network. It consists of two convolution-pooling layers with the *tanh* activation functions and two fully connected layers. For Caffe, the network is available in Caffe model repository. For Theano, the code is adopted from LISA Lab (2014). For Torch, we used the "mnist" package for easily loading the dataset and wrote our own script for timing purposes. For Neon[3], we adopted the code from the Neon GitHub repository. Neon requires the kernel size of convolutional layers and mini-batch size to be multiples of 4 and 32, respectively when employed on GPU. Thus the second convolution layer for Neon implementation is chosen to have 52 filters instead of 50 filters used in the other frameworks.

Table 3 shows the averaged processing time for gradient computation as well as the time for a forward step obtained by the four frameworks on both CPU and GPU using batch size of 64. For CPU timings, the number of threads used in each experiment is also reported. It should be noted that Neon cannot be configured to use multiple CPU threads and thus its performance on CPU is only reported with one thread. On the other hand, Caffe can be configured only during installation to run on a pre-determined number of threads (12 here) and thus it's performance on CPU is only reported with 12 threads. Theano and Torch are flexible in selecting the number of threads and thus their performances on CPU are reported with multiple settings. We report results for six and twelve threads since our system has six physical cores which can also be used with twelve threads using Hyper-Threading. When GPU is used, the underlying convolution library (e.g. cuDNN) is mentioned along with the framework. Neon uses its own GPU/CPU backend as mentioned before. The processing times clearly show the advantage of GPU over CPU (at least 20×) for training deep convolutional networks. This advantage would be more significant when training more complex models with larger data as will be shown later in this section. Torch results in best performance when comparing CPU times while Neon results in the worst performance on CPU. It is seen in the GPU experiments that cuDNN is faster for this network compared to the conv-fft. In general, the performance gain of using the FFT-based approach is highly dependent on the size of the input and kernel sizes (Mathieu et al., 2013). Theano results in best performance for the gradient computation on GPU while Torch and Theano achieve the best GPU performance for deployment. It should be noted that MNIST is a relatively small dataset and fits on the CPU host memory or the GPU device memory. Therefore, when Theano, Torch, or Neon is employed on GPU, the data is entirely copied into the GPU memory once before the training starts to avoid possible delays caused by communications between GPU and host for copying mini-batches[4].

[3] We directly call the *fprop* and *get_cost* functions to time the forward pass. Similarly, the *get_errors* and *bprop* functions are used to time the backward pass.

[4] This is done on Theano using shared variables, on Torch by calling the *:cuda()* function, and on Neon using the DataIterator class. Copying of the entire dataset into the memory can also be done for Caffe using

Table 3: The averaged processing times using batch size of 64.

		Setting	Gradient (ms)	Forward (ms)
CPU	1	Neon	545.6	172.7
		Theano	141.1	48.3
		Torch	46.1	18.1
	6	Theano	142.7	50.4
		Torch	18.1	5.6
	12	Caffe	66.4	33.7
		Theano	204.3	78.7
		Torch	**16.5**	**4.6**
GPU		Caffe + cuDNN	1.9	0.8
		Neon	2.3	1.0
		Theano + cuDNN	**1.4**	**0.5**
		Theano + conv-fft	5.6	2.7
		Torch + cuDNN	1.7	**0.5**
		Torch + cunn	13.6	5.8
		Torch + fbcunn	2.1	0.9

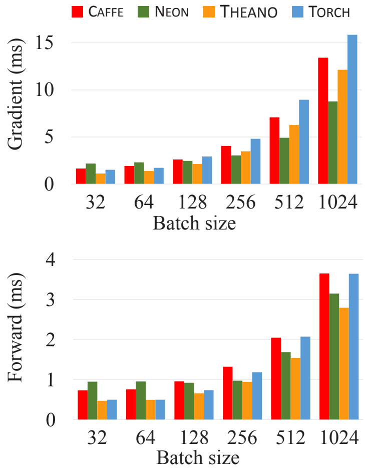

Figure 1: The averaged processing times for LeNet on GPU using different batch sizes. The cuDNN is used for Caffe, Theano, and Torch.

Figure 1 shows the gradient computation time and forward step time of the four frameworks on GPU using different batch sizes. It is seen that Theano has the best gradient computation time for small batches while Neon has the best performance for large batches. Theano consistently has the minimum forward time, specially for the large batch sizes. It is seen that Torch and Caffe performances drop more rapidly as the batch size increases.

4.2 AlexNet

In this section, we train AlexNet (Krizhevsky et al., 2012) on the ImageNet dataset. Note that there have been many recent, larger networks like GoogleNet, OxfordNet, etc. but we stick with AlexNet as it is the first network that significantly improved performance on ImageNet and is very popular. The network consists of five convolution layers, out of which three of them use grouping which restrict the connectivity of filters, and two have local response normalization (LRN) layers. The networks also has three pooling layers, two fully connected layers with ReLU activation units and dropout, and a softmax logistic loss. Each image is cropped to have dimension of 224. The data augmentation using random cropping or transformation is not performed[5]. For Caffe and Neon, the network is available from the corresponding GitHub repository. Neon currently does not support grouping and LRN layers. For Theano, the code of Ding et al. (2014) is adopted without performing data parallelization. We updated the implementation to avoid unnecessary *dimshuffle* operations. The convolution on GPU on Theano is performed by calling either the $dnn.dnn\_conv$ function from cuDNN library or the corresponding function from pylearn2 cuda-convent wrapper[6][7]. The latter is referred as cuconv in the results. For Torch, in addition to cuDNN library, we report the timings on GPU using both cunn and fbcunn libraries. Note that fbcunn does not support stride lengths greater than 1. So when reporting fbcunn results, we use the cuDNN-based convolution for the first layer of AlexNet and fbcunn-based convolutions for the rest. Furthermore, cunn and fbcunn do not support grouping. In addition to reporting the results for the exact AlexNet implementation, we also report the results without LRN layers and with grouping set to one to make the comparison transparent.

MemoryData layer. Here we used efficient LMDB database for Caffe and the communication overhead is not significant. The combined averaged forward and backward computational time of the data layer of LeNet in Caffe is about 1/1220 (1/30) of the total computational time of the batch when using CPU (GPU). This includes the time to rescale the images of the batch to the unit range.

[5] The ImageNet data is accessed in Caffe using the LMDB database, in Neon using $ImgMaster$ class, in Theano using Hickle, and in Torch using a multithreaded data loader provided in Chintala (2015b) that creates a pre-specified number of threads for parallel data loading from disk.

[6] The cuda-convnet (Krizhevsky et al., 2012) is a fast implementation of convolution but has some restrictions on input and kernel shapes with a different memory layout compared to Theano convolution operator.

[7] The Theano implementation does not use the standard convolution function ($conv.conv2d$) as it does not implement the type of padding used in AlexNet, known as "same" padding. Thus, it is not possible to perform the AlexNet Theano experiment on CPU or using conv-fft as they can be accessed through the conv2d function.

Table 4: The averaged processing times for AlexNet as well as peak GPU memory usage with batch size of 256.

	Settings	Gradient (ms)	Forward (ms)	GPU RAM (GB)
CPU	Caffe (12 threads)	43 152	19 817	-
	Neon (1 thread)*†	100 987	28 828	-
	Torch (6 threads)*	11 977	4383	-
	Torch (12 threads)*	**8421**	**2746**	-
GPU	Caffe + cuDNN	422.4	111.7	4.1
	Theano + cuDNN	529.8	162.8	**3.3**
	Theano + cuconv	684.9	156.1	5.6
	Torch + cuDNN	**390.2**	**92.5**	3.7
	Caffe + cuDNN*†	521.2	130.4	2.7
	Neon*†	290.5	**96.3**	**2.4**
	Theano + cuDNN*†	561.2	172.3	2.7
	Theano + cuconv*†	698.8	211.1	6.8
	Torch + cuDNN*†	405.9	100.7	2.8
	Torch + cunn*†	915.7	365.3	2.9
	Torch + fbcunn*†	**286.3**	98.4	4.8

*Without local response normalization layers.
† No grouping is performed in convolutional layers.

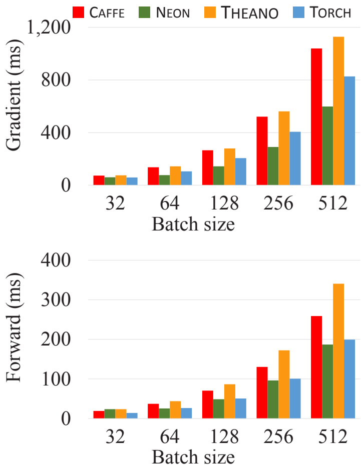

Figure 2: The averaged processing times for AlexNet on GPU using different batch sizes. The cuDNN is used for Caffe, Theano, and Torch. Experiments are without normalization layers and grouping.

Table 4 shows the performance of the four frameworks on the AlexNet using batch size of 256.To have a better performance comparison across the frameworks, the time required for data loading and processing (mean normalization) in each batch is excluded from time of forward and backward steps in all our experiments. We also report the peak GPU memory consumption to illustrate the efficacy of the frameworks in implementing deep networks[8]. In the CPU setting, Torch results in the best speed performance, similar to the LeNet results. The speed-up obtained by using GPU instead is more significant (at least 25×) here compared to the LeNet. When employed on GPU, Torch results in the best performance on the exact AlexNet implementation. When LRN layers are dropped and grouping are set to one, Torch using fbcunn results in the best gradient computation performance while Neon results in the best forward pass performance followed closely by Torch. Figure 2 shows the performance of the four frameworks on GPU using different batch sizes when no LRN layers are used here and grouping for all convolutional layers are set to one. Note that Neon and Torch have consistent superior performances for different batch sizes for the forward pass but Neon results in best performance for the gradient computation time. In terms of GPU memory consumption, similar efficient usage are observed across Caffe, Theano, and Torch when cuDNN is used while Neon has the most memory efficient usage. We also noticed from our experiments that the LMDB database used in Caffe has significantly better performance than the other data access layers used in other frameworks as it supports concurrent reads. Caffe also uses pre-fetching to eliminates IO latency. This can be an area of future developments for Neon, Theano, and Torch to make the LMDB database (or other efficient databases) and pre-fetching available in their frameworks. Pre-fetching and multi-thread processing can also be implemented in Torch as has been done for the Imagenet example in Chintala (2015b).

4.3 STACKED AUTOENCODERS

To benchmark a scenario with layer-wise pre-training procedure,we choose stacked autoencoders. This also provides a better picture of the performances of different frameworks when only fully-connected layers are used. We train three autoencoders (AEs) where each AE has a encoder and a corresponding decoder layer with tied weights, i.e. the decoder weights are transpose of the encoder weights.The sigmoid activation functions are used. The network is trained on the MNIST dataset in two steps: layer-wise unsupervised training and supervised fine-tuning. The unsupervised layer-wise training step is performed similar to the procedure in Bengio et al. (2007) using mean squared loss function. The AE1 is first trained on the raw images and then its weights are fixed. The AE2 is then trained on the resulting outputs of the first encoder and this procedure is repeated until all AEs

[8] We used *nvidia-smi* to monitor the GPU memory consumption.

			Gradient (ms)					Forward (ms)
		Setting	AE1	AE2	AE3	Total pre-training	SE	SE
CPU threads	1	Neon	14.6	11.5	10.5	35.6	9.7	4.8
		Theano	14.8	10.5	8.4	33.7	8.2	6.4
		Torch	13.7	8.7	6.5	28.9	8.2	5.0
	6	Theano	5.8	3.9	2.5	12.2	**2.6**	**1.8**
		Torch	**5.0**	**3.0**	**2.3**	**10.3**	3.3	1.9
	12	Caffe	11.7	10.6	8.6	30.9	7.4	6.1
		Theano	6.2	4.4	4.0	14.6	3.7	2.8
		Torch	9.8	4.2	3.2	17.2	3.8	2.3
GPU		Caffe + cuDNN	0.7	0.8	0.8	2.3	1.0	0.6
		Neon	1.1	1.5	1.7	4.3	1.8	0.9
		Theano + cuDNN	**0.6**	**0.4**	**0.3**	**1.3**	**0.4**	**0.2**
		Torch + cuDNN	**0.6**	0.5	0.5	1.6	0.7	0.3

Table 5: The averaged processing times of the stacked autoencoders (AE) for both pre-training and fine-tuning steps using batch size of 64. The encoder dimensions for AE1, AE2, and AE3 are 400, 200, and 100, respectively. For the unsupervised pre-training step, the gradient computation times are reported for the individual AEs along with the total gradient computation. For the supervised fine-tuning step of the stacked enocoders (SE), both gradient computation and forward pass times are reported. Caffe and Neon implementations do not have tied weights.

are trained. Note that once an AE is trained, its encoder outputs are not computed and recorded for the entire dataset in the memory to be used for the following AE[9], rather each batch is separately processed. Thus the forward pass for AE2, for example, includes a pass from raw image data to the first encoder and AE2 before loss is computed. In the supervised fine-tuning step, the training is performed on the stacked encoders (SE) of each AE with a softmax layer of size 10 and a cross entropy loss function. The decoders are not present in this fine-tuning step.

The above pre-training and fine-tuning steps are implemented in Theano LISA Lab (2014) and Torch. For Caffe, pre-training step is implemented using a few tricks. We have four configuration files in which three of them handle training of the individual AEs and one handles the fine-tuning step on the SE. We set the learning rates of the layers that should not be updated during pre-training step to zero[10]. For example, when training the second AE, the learning rate for the weights of first encoder are set to zero[11]. Our Neon implementation is very similar to Caffe implementation and Multiple optimizers are used to set the learning rates of the layers that should not get updated to zero. It should be noted that Caffe and Neon do not yet support tied weights and thus, different from our Theano and Torch implementations, have independent parameters for encoders and decoders. The performance of the four frameworks are shown in Table 5 where the encoders of the three AE layers have 400, 200 and 100 hidden layers, respectively. It is seen that Torch and Theano results in superior performance and Neon results in the worst performance for both CPU and GPU settings. We have also evaluated the frameworks in a different setting, where the number of hidden layers of encoders of AE1, AE2 and AE3 are 800, 1000 and 2000, respectively. For this larger network, Caffe results in better performance compared to Theano on GPU but Torch again achieves the best performance. The results of this experiment are shown in Table 7 in the Appendix.

4.4 LSTM

In this section, we train a LSTM network (Graves et al., 2012) for the task of sentiment analysis on IMDB dataset. In this task, each sentence is considered as a (varying-length) sequence of words. The network architecture is the same as the one used in LISA Lab (2014). It consists of an embedding layer followed by an LSTM layer. The outputs of the LSTM layer are then averaged and fed to a linear fully connected layer with softmax logistic regression for binary classification. The sequences

[9] Saving the outputs of trained encoder for the entire input would improve computational time but is not a memory-efficient procedure, specially for large datasets, and therefore is not employed here.

[10] One can use PyCaffe and the Caffe "net surgery" procedure to transfer the learned weights of each trained AE to the following AE. This is not performed here as we are only interested in the computational performance.

[11] It should be noted that Caffe detects the zero learning rates and does not perform unnecessary calculations.

	Setting	Gradient (ms)	Forward (ms)
CPU	Theano (6 thread)	205.77	96.24
	Torch (6 threads)	**117.18**	**54.8**
GPU	Theano + cuDNN	**16.72**	**4.66**
	Torch + cuDNN	98.74	29.2

Table 6: The averaged processing times of the LSTM using batch size of 16.

within each batch are padded to have the same size as the largest sequence within the batch and a masking array is used to make sure the recursive computations of the LSTM layer remain valid. For Torch, we use the LSTM layer from the “rnn” package Leonard (2015) along with the *MaskZero* and *LookupTableMaskZero* modules for handling the varying length scenario.

Caffe does not yet officially support cyclic architectures, and in particular LSTM, and thus its performance is not reported here[12]. While Neon has LSTM layers and has the option to pad data to fixed sizes, it does not accept variable length inputs within a batch and thus is not used here. It should be noted that one of the main advantages of recurrent networks are their capabilities in handling variable length inputs without the need to make the window size constant (Graves et al., 2012).

We used 124 iterations, one entire epoch, to average the computational time for different padding sizes. Also shuffling is not performed on the training set to make sure different frameworks receive the same sequence of batches and thus have the same number of flops. As the dataset is small, it is initially loaded into the device or host memory. Table 6 shows the performance of Theano and Torch for the LSTM network.As with previous cases, Torch performs best for CPU but with a GPU, Theano results in better performance.

5 Conclusions

We evaluated four of the top deep learning frameworks, namely Caffe, Neon, Theano and Torch for a variety of settings on a single machine. Here are our main observations:

- Theano and Torch are the most extensible frameworks both in terms of supporting various deep architectures but also in terms of supported libraries. The symbolic differentiation is one of the most useful features that Theano offers for implementing non-standard deep architectures. Torch community is trying to fill this gap[13].
- For CPU-based training and deployment of *any* tested deep network architecture, Torch performs the best followed by Theano, and Neon has the worst performance.
- For GPU-based deployment of trained convolutional and fully connected networks, Torch is best suited, followed by Theano.
- For GPU-based training of convolutional and fully connected networks, we noticed Theano is fastest for small networks and Torch is fastest for larger networks. Neon is very competitive on GPU for large convolutional networks.
- For GPU-based training and deployment of recurrent networks (LSTM), Theano results in the best performance.
- Torch could greatly benefit from expanded documentation of its libraries and capabilities and better error debugging tools.

References

Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Bergstra, James, Goodfellow, Ian, Bergeron, Arnaud, Bouchard, Nicolas, Warde-Farley, David, and Bengio, Yoshua. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

Bengio, Yoshua, LeCun, Yann, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5), 2007.

[12] Recently, a pull request is submitted to the official Caffe repository which adds the support for RNN and LSTM. See `http://jeffdonahue.com/lrcn/` for more information.

[13] For more information see `https://blog.twitter.com/2015/autograd-for-torch`

Bengio, Yoshua, Goodfellow, Ian J., and Courville, Aaron. Deep learning. Book in preparation for MIT Press, 2015. URL `http://www.iro.umontreal.ca/~bengioy/dlbook`.

Bergstra, James, Bastien, Frédéric, Breuleux, Olivier, Lamblin, Pascal, Pascanu, Razvan, Delalleau, Olivier, Desjardins, Guillaume, Warde-Farley, David, Goodfellow, Ian, Bergeron, Arnaud, et al. Theano: Deep learning on gpus with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*, 2011.

Chintala, Soumith. convnet-benchmarks. `https://github.com/soumith/convnet-benchmarks`, 2015a. Accessed: 2015-10-30.

Chintala, Soumith. Imagenet multi-gpu. `https://github.com/soumith/imagenet-multiGPU.torch`, 2015b. Accessed: 2015-10-30.

Collobert, Ronan, Kavukcuoglu, Koray, and Farabet, Clément. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011a.

Collobert, Ronan, Weston, Jason, Bottou, Léon, Karlen, Michael, Kavukcuoglu, Koray, and Kuksa, Pavel. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011b.

Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Ding, Weiguang, Wang, Ruoyan, Mao, Fei, and Taylor, Graham. Theano-based large-scale visual recognition with multiple gpus. *arXiv preprint arXiv:1412.2302*, 2014.

Graves, Alex et al. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer, 2012.

Hannun, Awni, Case, Carl, Casper, Jared, Catanzaro, Bryan, Diamos, Greg, Elsen, Erich, Prenger, Ryan, Satheesh, Sanjeev, Sengupta, Shubho, Coates, Adam, et al. Deepspeech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pp. 675–678. ACM, 2014.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Leonard, Nicholas. Rnn library for torch7. `https://github.com/Element-Research/rnn`, 2015. Accessed: 2015-10-30.

LISA Lab. Deep learning tutorial. *University of Montreal*, 2014.

Maas, Andrew L, Daly, Raymond E, Pham, Peter T, Huang, Dan, Ng, Andrew Y, and Potts, Christopher. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 142–150. Association for Computational Linguistics, 2011.

Mathieu, Michael, Henaff, Mikael, and LeCun, Yann. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, pp. 1–42, 2014.

Vasilache, Nicolas, Johnson, Jeff, Mathieu, Michael, Chintala, Soumith, Piantino, Serkan, and LeCun, Yann. Fast convolutional nets with fbfft: A gpu performance evaluation. *arXiv preprint arXiv:1412.7580*, 2014.

6 Appendix

6.1 Caffe

Caffe is a deep learning tool developed by the Berkeley Vision and Learning Center and by community contributors and is released under BSD 2-Clause license (Jia et al., 2014). It is developed in C++ with expression, speed, and modularity in mind which uses CUDA for GPU computation and has commandline, Python, and Matlab interfaces for training and deployment purposes. It separates the definition of the network architecture from actual implementation allowing to conveniently and quickly explore different architectures and layers on either CPU or GPU. Caffe can use LMDB database that allocates memory on the host and device automatically and lazily based on demand for efficient memory usage and high-throughput. The LMDB database supports concurrent reads.

Several types of layers and loss functions are already implemented which can be configured in the form of arbitrary directed acyclic graphs in a configuration file. There are also pre-trained models for popular networks such as AlexNet (with non-commercial license) which allows reproducible research. At the time of writing this report, Caffe supports various layers such as convolution, fully connected and pooling layers, etc. The convolution operation can be computed using either a native implementation (by dense matrix multiplications using Blas) or Nvidia cuDNN, if it is installed, where latter usually results in faster computation.

6.2 Theano

Theano is a free Python symbolic manipulation library, under a BSD license, aiming to improve execution time and development time for machine learning algorithms (Bergstra et al., 2011; Bastien et al., 2012). It has specifically been utilized for the gradient-based methods such as deep learning that require repeated computation of the tensor-based mathematical expressions. Such mathematical expressions can be rapidly coded in Theano using a high-level description language similar to a functional language that can be compiled and executed on either a CPU or a GPU.

Theano uses CUDA library to define arrays located on an Nvidia GPU memory with Python bindings. Theano includes many custolllrrm C and CUDA code generators tailored for different types, sizes, and shapes of inputs which optimizes the computation of the complicated tensor computations. Theano benefits from a large user community that contribute to its development partly due to the ease of development offered by Python language and its scientific computing stack. Examples of the deep learning algorithms implemented using Theano can be found at LISA Lab (2014). In the latest version of Theano used here (Theano 0.7), the convolution operation automatically uses the optimized Nvidia cuDNN library, if installed, to perform the convolution. It also provides two additional implementations for the convolution operation, an FFT-based implementation (Mathieu et al., 2013) and an implementation based on the open-source code of Alex Krizhevsky (Krizhevsky et al., 2012).While Theano is a general mathematical expression library and may have a relatively steep learning curve for writing efficient code and debugging, several libraries (e.g. Pylearn2, Keras, and Lasagne) have been developed on top it which are specifically tailored for deep learning algorithm providing building blocks for fast experimentation of the well-known methods.

6.3 Torch

Torch is a scientific computational framework built using Lua that runs on Lua (JIT) compiler (Collobert et al., 2011a). It has strong CUDA and CPU backends and contains well-developed, mature machine learning and optimization packages. The Tensor libraries that come with it have very efficient CUDA backend and the neural networks (nn) libraries can be used to build arbitrary acyclic computation graphs with automatic differentiation functionalities i.e. It has a *:forward()* function that computes the output for a given input, flowing the input through the network; and it has a *:backward()* function that will differentiate each parameter in the network w.r.t. the gradient that is passed in. Torch also provides bindings to the latest version of Nvidia cuDNN that gives it access to state-of-art speedups for convolutional operations. The latest version, Torch7, has easy to use multi-GPU support and parallelizing packages that make it very powerful for training deep architectures. Torch has a large community of developers and is being actively used within large organizations like Facebook, Google and Twitter. Specifically, many researchers at NYU and Facebook AI Research (FAIR) lab actively contribute to Torch by making a lot of their code open source. Many companies also have in-house teams to customize Torch for their deep learning platforms that has contributed to its popularity in recent times.

6.4 Neon

Neon is a Python based deep learning framework developed by Nervana. It has recently been open-sourced under an open source Apache 2.0 License. Neon has customized CPU and GPU backends, known as NervanaCPU and NervanaGPU backends, respectively. The NervanaGPU backend consists of kernels written in MaxAs assembler and Python wrappers which is highly optimized for Nvidias Maxwell GPUs (e.g. Titan X). The NervanaCPU backend is built on top of python NumPy library. Neon supports commonly used models such as convnets, MLPs, RNNs, and autoencoders. Compared to above three frameworks, Neon is a relatively young framework. Thus, it has not yet been adopted widely within the deep learning community and many of the features already available in the other frameworks, are still under development for Neon. More discussions on the available and missing features of Neon will be provided in the following sections.

6.5 Supplemental results

			Gradient (ms)					Forward (ms)
		Setting	AE1	AE2	AE3	Total pre-training	SE	SE
CPU threads	1	Neon	24.6	46.1	120.1	190.8	76.5	29.3
		Theano	23.2	36.9	79.0	139.1	65.1	43.2
		Torch	22.9	35.0	79.2	137.1	61.8	34.0
	6	Theano	8.1	13.7	24.8	46.6	24.3	14.6
		Torch	**7.6**	**13.0**	**24.9**	**45.5**	**22.7**	**11.4**
	12	Caffe	17.2	30.0	63.9	111.1	44.3	32.0
		Theano	8.9	15.8	29.0	53.7	25.9	15.8
		Torch	11.4	19.3	37.7	68.4	31.9	16.0
GPU		Caffe + cuDNN	**0.8**	1.1	**1.5**	**3.4**	1.7	0.9
		Neon	1.1	1.5	1.9	4.5	2.0	1.0
		Theano + cuDNN	0.9	1.2	2.2	4.3	**1.1**	0.9
		Torch + cuDNN	**0.8**	**0.9**	1.8	3.5	1.5	**0.7**

Table 7: The averaged processing times of the stacked autoencoders (AE) for both pre-training and fine-tuning steps using batch size of 64. The encoder dimensions for AE1, AE2, and AE3 are 800, 1000, and 2000, respectively. For the unsupervised pre-training step, the gradient computation times are reported for the individual AEs along with the total gradient computation. For the supervised fine-tuning step of the stacked enocoders (SE), both gradient computation and forward pass times are reported. Caffe and Neon implementations do not have tied weights.

	Setting	First compile (s)	Re-compile (s)
CPU	LeNet	25.2	0.7
	Stacked Auto Encoder (small)	19.9	2.0
	LSTM	80.1	12.7
GPU	LeNet	177.7	5.0
	AlexNet	212.0	6.1
	Stacked Auto Encoder (small)	106.8	2.0
	LSTM	283.5	19.7

Table 8: The averaged times required on Theano to compile both gradient and forward functions for the studied deep networks. The cuDNN library is used for the GPU measurements. We report two sets of measurements. The first set shows the compilation times when the Theano cache is clear. The second set shows the times required to re-compile the functions. The re-compilation times, which are significantly faster, are more indicative of times required to fine-tune and cross-validate an architecture and thus are more relevant for practical scenarios. We noticed from our experiments that changing hyperparameters (e.g. number of feature maps or convolutional layers) causes only slight changes in the re-compilation times. For more information see: `http://deeplearning.net/software/theano/extending/pipeline.html`.