TRAINING DEEP NEURAL NETWORKS WITH LOW PRECISION MULTIPLICATIONS

Matthieu Courbariaux & Jean-Pierre David
École Polytechnique de Montréal
{matthieu.courbariaux,jean-pierre.david}@polymtl.ca

Yoshua Bengio
Université de Montréal, CIFAR Senior Fellow
yoshua.bengio@gmail.com

ABSTRACT

Multipliers are the most space and power-hungry arithmetic operators of the digital implementation of deep neural networks. We train a set of state-of-the-art neural networks (Maxout networks) on three benchmark datasets: MNIST, CIFAR-10 and SVHN. They are trained with three distinct formats: floating point, fixed point and dynamic fixed point. For each of those datasets and for each of those formats, we assess the impact of the precision of the multiplications on the final error after training. We find that *very low precision is sufficient* not just for running trained networks but *also for training them*. For example, it is possible to train Maxout networks with **10** bits multiplications.

1 INTRODUCTION

The *training* of deep neural networks is very often limited by hardware. Lots of previous works address the best exploitation of general-purpose hardware, typically CPU clusters (Dean *et al.*, 2012) and GPUs (Coates *et al.*, 2009; Krizhevsky *et al.*, 2012a). Faster implementations usually lead to state of the art results (Dean *et al.*, 2012; Krizhevsky *et al.*, 2012a).

Actually, such approaches always consist in adapting the algorithm to best exploit state of the art general-purpose hardware. Nevertheless, some dedicated deep learning hardware is appearing as well. FPGA and ASIC implementations claim a better power efficiency than general-purpose hardware (Kim *et al.*, 2009; Farabet *et al.*, 2011; Pham *et al.*, 2012; Chen *et al.*, 2014a;b). In contrast with general-purpose hardware, dedicated hardware such as ASIC and FPGA enables to build the hardware from the algorithm.

Hardware is mainly made out of memories and arithmetic operators. Multipliers are the most space and power-hungry arithmetic operators of the digital implementation of deep neural networks. The objective of this article is to assess the possibility to reduce the precision of the multipliers for deep learning:

- We train deep neural networks with low precision multipliers and high precision accumulators (Section 2).
- We carry out experiments with three distinct formats:
  1. Floating point (Section 3)
  2. Fixed point (Section 4)
  3. Dynamic fixed point, which we think is a good compromise between floating and fixed points (Section 5)
- We use a higher precision for the parameters during the updates than during the forward and backward propagations (Section 6).
- Maxout networks (Goodfellow *et al.*, 2013a) are a set of state-of-the-art neural networks (Section 7). We train Maxout networks with slightly less capacity than Goodfellow *et al.* (2013a) on three benchmark datasets: MNIST, CIFAR-10 and SVHN (Section 8).

- For each of the three datasets and for each of the three formats, we assess the impact of the precision of the multiplications on the final error of the training. We find that *very low precision multiplications are sufficient* not just for running trained networks but *also for training them* (Section 9). We made our code available [1].

2 MULTIPLIER-ACCUMULATORS

Multiplier (bits)	Accumulator (bits)	Adaptive Logic Modules (ALMs)
32	32	504
16	32	138
16	16	128

Table 1: Cost of a fixed point multiplier-accumulator on a Stratix V Altera FPGA.

Algorithm 1 Forward propagation with low precision multipliers.

for all layers **do**
  Reduce the precision of the parameters and the inputs
  Apply convolution or dot product (with high precision accumulations)
  Reduce the precision of the weighted sums
  Apply activation functions
end for
Reduce the precision of the outputs

Applying a deep neural network (DNN) mainly consists in convolutions and matrix multiplications. The key arithmetic operation of DNNs is thus the multiply-accumulate operation. Artificial neurons are basically multiplier-accumulators computing weighted sums of their inputs.

The cost of a fixed point multiplier varies as the square of the precision (of its operands) for small widths while the cost of adders and accumulators varies as a linear function of the precision (David *et al.*, 2007). As a result, the cost of a fixed point multiplier-accumulator mainly depends on the precision of the multiplier, as shown in table 1. In modern FPGAs, the multiplications can also be implemented with dedicated DSP blocks/slices. One DSP block/slice can implement a single 27×27 multiplier, a double 18×18 multiplier or a triple 9×9 multiplier. Reducing the precision can thus lead to a gain of 3 in the number of available multipliers inside a modern FPGA.

In this article, we train deep neural networks with low precision multipliers and high precision accumulators, as illustrated in Algorithm 1.

3 FLOATING POINT

Format	Total bit-width	Exponent bit-width	Mantissa bit-width
Double precision floating point	64	11	52
Single precision floating point	32	8	23
Half precision floating point	16	5	10

Table 2: Definitions of double, single and half precision floating point formats.

Floating point formats are often used to represent real values. They consist in a sign, an exponent, and a mantissa, as illustrated in figure 1. The exponent gives the floating point formats a wide range, and the mantissa gives them a good precision. One can compute the value of a single floating point number using the following formula:

$$value = (-1)^{sign} \times \left(1 + \frac{mantissa}{2^{23}}\right) \times 2^{(exponent-127)}$$

[1] https://github.com/MatthieuCourbariaux/deep-learning-multipliers

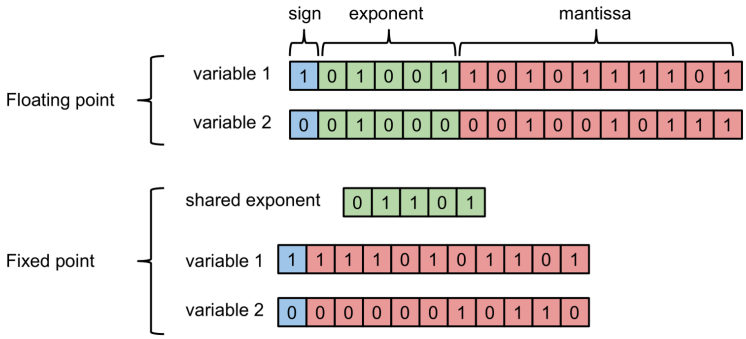

Figure 1: Comparison of the floating point and fixed point formats.

Table 2 shows the exponent and mantissa widths associated with each floating point format. In our experiments, we use single precision floating point format as our reference because it is the most widely used format in deep learning, especially for GPU computation. We show that the use of half precision floating point format has little to no impact on the training of neural networks. At the time of writing this article, no standard exists below the half precision floating point format.

4 FIXED POINT

Fixed point formats consist in a signed mantissa and a global scaling factor shared between all fixed point variables. The scaling factor can be seen as the position of the radix point. It is usually fixed, hence the name "fixed point". Reducing the scaling factor reduces the range and augments the precision of the format. The scaling factor is typically a power of two for computational efficiency (the scaling multiplications are replaced with shifts). As a result, fixed point format can also be seen as a floating point format with a *unique shared fixed exponent* , as illustrated in figure 1. Fixed point format is commonly found on embedded systems with no FPU (Floating Point Unit). It relies on integer operations. It is hardware-wise cheaper than its floating point counterpart, as the exponent is shared and fixed.

5 DYNAMIC FIXED POINT

Algorithm 2 Policy to update a scaling factor.

Require: a matrix M, a scaling factor s_t, and a maximum overflow rate r_{max}.
Ensure: an updated scaling factor s_{t+1}.
  if the overflow rate of $M > r_{max}$ **then**
    $s_{t+1} \leftarrow 2 \times s_t$
  else if the overflow rate of $2 \times M \leq r_{max}$ **then**
    $s_{t+1} \leftarrow s_t/2$
  else
    $s_{t+1} \leftarrow s_t$
  end if

When training deep neural networks,

1. activations, gradients and parameters have *very different ranges*.
2. gradients ranges *slowly diminish* during the training.

As a result, the fixed point format, with its unique shared fixed exponent, is ill-suited to deep learning.

The dynamic fixed point format (Williamson, 1991) is a variant of the fixed point format in which there are *several scaling factors* instead of a single global one. Those scaling factors are *not fixed*. As such, it can be seen as a compromise between floating point format - where each scalar variable owns its scaling factor which is updated during each operations - and fixed point format - where there is only one global scaling factor which is never updated. With dynamic fixed point, a few grouped variables share a scaling factor which is updated from time to time to reflect the statistics of values in the group.

In practice, we associate each layer's weights, bias, weighted sum, outputs (post-nonlinearity) and the respective gradients vectors and matrices with a different scaling factor. Those scaling factors are initialized with a global value. The initial values can also be found during the training with a higher precision format. During the training, we update those scaling factors at a given frequency, following the policy described in Algorithm 2.

6 UPDATES VS. PROPAGATIONS

We use a higher precision for the parameters during the updates than during the forward and backward propagations, respectively called fprop and bprop. The idea behind this is to be able to accumulate small changes in the parameters (which requires more precision) and while on the other hand sparing a few bits of memory bandwidth during fprop. This can be done because of the implicit averaging performed via stochastic gradient descent during training:

$$\theta_{t+1} = \theta_t - \epsilon \frac{\partial C_t(\theta_t)}{\partial \theta_t}$$

where $C_t(\theta_t)$ is the cost to minimize over the minibatch visited at iteration t using θ_t as parameters and ϵ is the learning rate. We see that the resulting parameter is the sum

$$\theta_T = \theta_0 - \epsilon \sum_{t=1}^{T-1} \frac{\partial C_t(\theta_t)}{\partial \theta_t}.$$

The terms of this sum are not statistically independent (because the value of θ_t depends on the value of θ_{t-1}) but the dominant variations come from the random sample of examples in the minibatch (θ moves slowly) so that a strong averaging effect takes place, and each contribution in the sum is relatively small, hence the demand for sufficient precision (when adding a small number with a large number).

7 MAXOUT NETWORKS

A Maxout network is a multi-layer neural network that uses maxout units in its hidden layers. A maxout unit outputs the maximum of a set of k dot products between k weight vectors and the input vector of the unit (e.g., the output of the previous layer):

$$h_i^l = \max_{j=1}^{k}(b_{i,j}^l + w_{i,j}^l \cdot h^{l-1})$$

where h^l is the vector of activations at layer l and weight vectors $w_{i,j}^l$ and biases $b_{i,j}^l$ are the parameters of the j-th filter of unit i on layer l.

A maxout unit can be seen as a generalization of the rectifying units (Jarrett *et al.*, 2009; Nair and Hinton, 2010; Glorot *et al.*, 2011; Krizhevsky *et al.*, 2012b)

$$h_i^l = \max(0, b_i^l + w_i^l \cdot h^{l-1})$$

which corresponds to a maxout unit when $k = 2$ and one of the filters is forced at 0 (Goodfellow *et al.*, 2013a). Combined with dropout, a very effective regularization method (Hinton *et al.*, 2012), maxout networks achieved state-of-the-art results on a number of benchmarks (Goodfellow *et al.*, 2013a), both as part of fully connected feedforward deep nets and as part of deep convolutional nets. The dropout technique provides a good approximation of model averaging with shared parameters across an exponentially large number of networks that are formed by subsets of the units of the original noise-free deep network.

8 BASELINE RESULTS

We train Maxout networks with slightly less capacity than Goodfellow *et al.* (2013a) on three benchmark datasets: MNIST, CIFAR-10 and SVHN. In Section 9, we use the same hyperparameters as in this section to train Maxout networks with low precision multiplications.

Dataset	Dimension	Labels	Training set	Test set
MNIST	784 (28×28 grayscale)	10	60K	10K
CIFAR-10	3072 (32×32 color)	10	50K	10K
SVHN	3072 (32×32 color)	10	604K	26K

Table 3: Overview of the datasets used in this paper.

Format	Prop.	Up.	PI MNIST	MNIST	CIFAR-10	SVHN
Goodfellow *et al.* (2013a)	32	32	0.94%	0.45%	11.68%	2.47%
Single precision floating point	32	32	1.05%	0.51%	14.05%	2.71%
Half precision floating point	16	16	1.10%	0.51%	14.14%	3.02%
Fixed point	20	20	1.39%	0.57%	15.98%	2.97%
Dynamic fixed point	10	12	1.28%	0.59%	14.82%	4.95%

Table 4: Test set error rates of single and half floating point formats, fixed and dynamic fixed point formats on the permutation invariant (PI) MNIST, MNIST (with convolutions, no distortions), CIFAR-10 and SVHN datasets. **Prop.** is the bit-width of the propagations and **Up.** is the bit-width of the parameters updates. The single precision floating point line refers to the results of our experiments. It serves as a baseline to evaluate the degradation brought by lower precision.

8.1 MNIST

The MNIST (LeCun *et al.*, 1998) dataset is described in Table 3. We do not use any data-augmentation (e.g. distortions) nor any unsupervised pre-training. We simply use minibatch stochastic gradient descent (SGD) with momentum. We use a linearly decaying learning rate and a linearly saturating momentum. We regularize the model with dropout and a constraint on the norm of each weight vector, as in (Srebro and Shraibman, 2005).

We train two different models on MNIST. The first is a permutation invariant (PI) model which is unaware of the structure of the data. It consists in two fully connected maxout layers followed by a softmax layer. The second model consists in three convolutional maxout hidden layers (with spatial max pooling on top of the maxout layers) followed by a densely connected softmax layer.

This is the same procedure as in Goodfellow *et al.* (2013a), except that we do not train our model on the validation examples. As a consequence, our test error is slightly larger than the one reported in Goodfellow *et al.* (2013a). The final test error is in Table 4.

8.2 CIFAR-10

Some comparative characteristics of the CIFAR-10 (Krizhevsky and Hinton, 2009) dataset are given in Table 3. We preprocess the data using global contrast normalization and ZCA whitening. The model consists in three convolutional maxout layers, a fully connected maxout layer, and a fully connected softmax layer. We follow a similar procedure as with the MNIST dataset. This is the same procedure as in Goodfellow *et al.* (2013a), except that we reduced the number of hidden units and that we do not train our model on the validation examples. As a consequence, our test error is slightly larger than the one reported in Goodfellow *et al.* (2013a). The final test error is in Table 4.

8.3 STREET VIEW HOUSE NUMBERS

The SVHN (Netzer *et al.*, 2011) dataset is described in Table 3. We applied local contrast normalization preprocessing the same way as Zeiler and Fergus (2013). The model consists in three convolutional maxout layers, a fully connected maxout layer, and a fully connected softmax layer. Otherwise, we followed the same approach as on the MNIST dataset. This is the same procedure as in Goodfellow *et al.* (2013a), except that we reduced the length of the training. As a consequence, our test error is bigger than the one reported in Goodfellow *et al.* (2013a). The final test error is in Table 4.

9 LOW PRECISION RESULTS

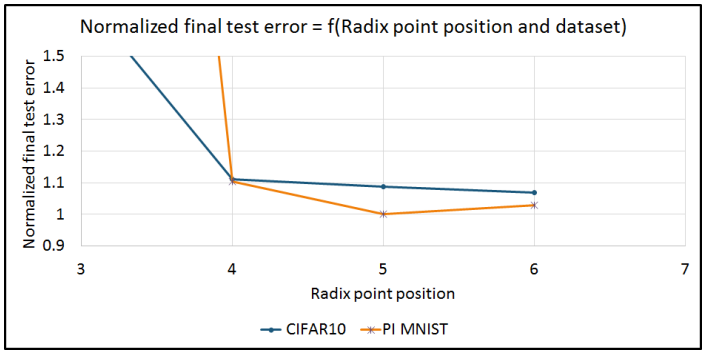

Figure 2: Final test error depending on the radix point position (5 means after the 5th most significant bit) and the dataset (permutation invariant MNIST and CIFAR-10). The final test errors are normalized, that is to say divided by the dataset single float test error. The propagations and parameter updates bit-widths are both set to 31 bits (32 with the sign).

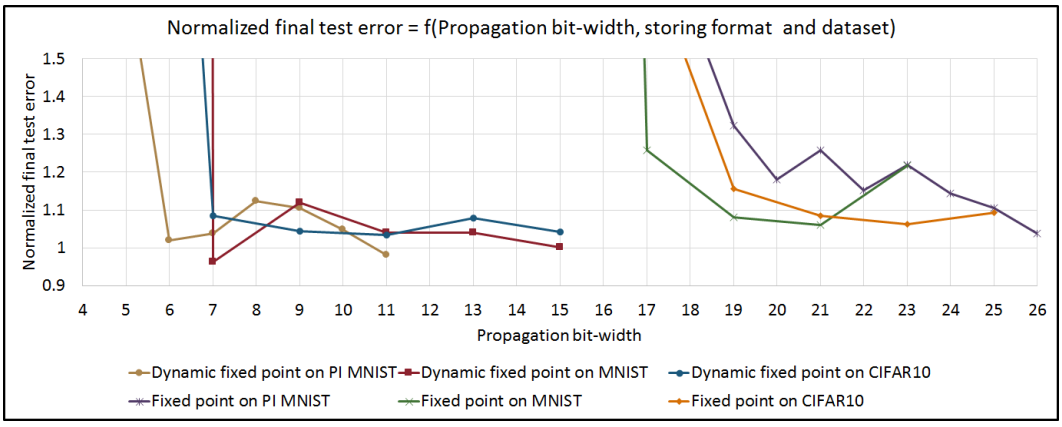

Figure 3: Final test error depending on the propagations bit-width, the format (dynamic fixed or fixed point) and the dataset (permutation invariant MNIST, MNIST and CIFAR-10). The final test errors are normalized, which means that they are divided by the dataset single float test error. For both formats, the parameter updates bit-width is set to 31 bits (32 with the sign). For fixed point format, the radix point is set after the fifth bit. For dynamic fixed point format, the maximum overflow rate is set to 0.01%.

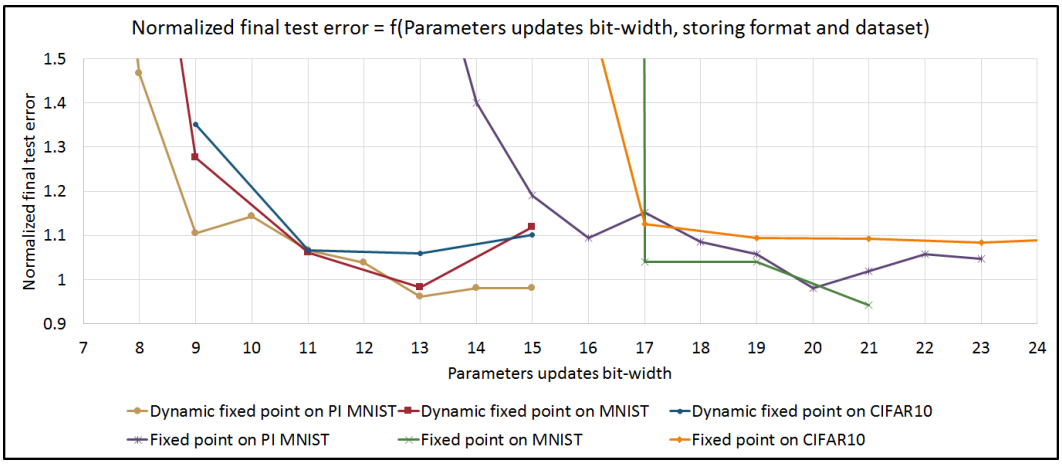

Figure 4: Final test error depending on the parameter updates bit-width, the format (dynamic fixed or fixed point) and the dataset (permutation invariant MNIST, MNIST and CIFAR-10). The final test errors are normalized, which means that they are divided by the dataset single float test error. For both formats, the propagations bit-width is set to 31 bits (32 with the sign). For fixed point format, the radix point is set after the fifth bit. For dynamic fixed point format, the maximum overflow rate is set to 0.01%.

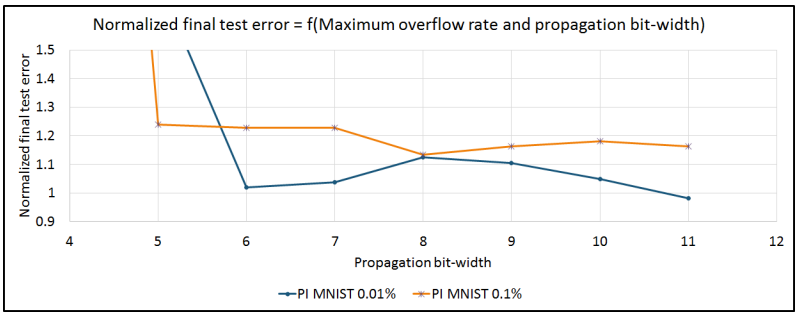

Figure 5: Final test error depending on the maximum overflow rate and the propagations bit-width. The final test errors are normalized, which means that they are divided by the dataset single float test error. The parameter updates bit-width is set to 31 bits (32 with the sign).

9.1 Floating point

Half precision floating point format has little to no impact on the test set error rate, as shown in Table 4. We conjecture that a high-precision fine-tuning could recover the small degradation of the error rate.

9.2 Fixed point

The optimal radix point position in fixed point is after the fifth (or arguably the sixth) most important bit, as illustrated in Figure 2. The corresponding range is approximately [-32,32]. The corresponding scaling factor depends on the bit-width we are using. The minimum bit-width for propagations in fixed point is 19 (20 with the sign). Below this bit-width, the test set error rate rises very sharply, as illustrated in Figure 3. The minimum bit-width for parameter updates in fixed point is 19 (20 with the sign). Below this bit-width, the test set error rate rises very sharply, as illustrated in Figure 4. Doubling the number of hidden units does not allow any further reduction of the bit-widths on the

permutation invariant MNIST. In the end, using 19 (20 with the sign) bits for both the propagations and the parameter updates has little impact on the final test error, as shown in Table 4.

9.3 DYNAMIC FIXED POINT

We find the initial scaling factors by training with a higher precision format. Once those scaling factors are found, we reinitialize the model parameters. We update the scaling factors once every 10000 examples. Augmenting the maximum overflow rate allows us to reduce the propagations bit-width but it also significantly augments the final test error rate, as illustrated in Figure 5. As a consequence, we use a low maximum overflow rate of 0.01% for the rest of the experiments. The minimum bit-width for the propagations in dynamic fixed point is 9 (10 with the sign). Below this bit-width, the test set error rate rises very sharply, as illustrated in Figure 3. The minimum bit-width for the parameter updates in dynamic fixed point is 11 (12 with the sign). Below this bit-width, the test set error rate rises very sharply, as illustrated in Figure 4. Doubling the number of hidden units does not allow any further reduction of the bit-widths on the permutation invariant MNIST. In the end, using 9 (10 with the sign) bits for the propagations and 11 (12 with the sign) bits for the parameter updates has little impact on the final test error, with the exception of the SVHN dataset, as shown in Table 4. This is significantly better than fixed point format, which is consistent with our predictions of Section 5.

10 RELATED WORKS

Vanhoucke *et al.* (2011) use 8 bits linear quantization to store activations and weights. Weights are scaled by taking their maximum magnitude in each layer and normalizing them to fall in the [-128, 127] range. The total memory footprint of the network is reduced by between $3\times$ and $4\times$. This is very similar to the dynamic fixed point format we use (Section 5). However, Vanhoucke *et al.* (2011) only *apply* already trained neural networks while we actually *train* them.

Training neural networks with low precision arithmetic has already been done in previous works (Holt and Baker, 1991; Presley and Haggard, 1994; Simard and Graf, 1994; Wawrzynek *et al.*, 1996; Savich *et al.*, 2007) [2]. Our work is nevertheless original in several regards:

- We are the first to train deep neural networks with the dynamic fixed point format.
- We use a higher precision for the weights during the updates.
- We train some of the latest models on some of the latest benchmarks.

11 CONCLUSION AND FUTURE WORKS

We have shown that:

- Very low precision multipliers are sufficient for training deep neural networks.
- Dynamic fixed point seems well suited for training deep neural networks.
- Using a higher precision for the parameters during the updates helps.

Our work can be exploited to:

- Optimize memory usage on general-purpose hardware (Gray *et al.*, 2015).
- Design very power-efficient hardware dedicated to deep learning.

There is plenty of room for extending our work:

- Other tasks than image classification.

[2] A very recent work (Gupta *et al.*, 2015) also trains neural networks with low precision. The authors propose to replace round-to-nearest with stochastic rounding, which allows to reduce the numerical precision to 16 bits while using the fixed point format. It would be very interesting to combine dynamic fixed point and stochastic rounding.

- Other models than Maxout networks.
- Other formats than floating point, fixed point and dynamic fixed point.

12 ACKNOWLEDGEMENT

We thank the developers of Theano (Bergstra *et al.*, 2010; Bastien *et al.*, 2012), a Python library which allowed us to easily develop a fast and optimized code for GPU. We also thank the developers of Pylearn2 (Goodfellow *et al.*, 2013b), a Python library built on the top of Theano which allowed us to easily interface the datasets with our Theano code. We are also grateful for funding from NSERC, the Canada Research Chairs, Compute Canada, and CIFAR.

REFERENCES

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., and Bengio, Y. (2012). Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., and Temam, O. (2014a). Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*, pages 269–284. ACM.

Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., Li, L., Chen, T., Xu, Z., Sun, N., *et al.* (2014b). Dadiannao: A machine-learning supercomputer. In *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, pages 609–622. IEEE.

Coates, A., Baumstarck, P., Le, Q., and Ng, A. Y. (2009). Scalable learning for object detection with gpu hardware. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4287–4293. IEEE.

David, J., Kalach, K., and Tittley, N. (2007). Hardware complexity of modular multiplication and exponentiation. *Computers, IEEE Transactions on*, **56**(10), 1308–1319.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Le, Q., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. (2012). Large scale distributed deep networks. In *NIPS'2012*.

Farabet, C., Martini, B., Corda, B., Akselrod, P., Culurciello, E., and LeCun, Y. (2011). Neuflow: A runtime reconfigurable dataflow processor for vision. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 109–116. IEEE.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *AISTATS'2011*.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013a). Maxout networks. Technical Report Arxiv report 1302.4389, Université de Montréal.

Goodfellow, I. J., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., Bergstra, J., Bastien, F., and Bengio, Y. (2013b). Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*.

Gray, S., Leishman, S., and Koster, U. (2015). Nervanagpu library. Accessed: 2015-06-30.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. (2015). Deep learning with limited numerical precision. *CoRR*, **abs/1502.02551**.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. Technical report, arXiv:1207.0580.

Holt, J. L. and Baker, T. E. (1991). Back propagation simulations using limited precision calculations. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume 2, pages 121–126. IEEE.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision (ICCV'09)*, pages 2146–2153. IEEE.

Kim, S. K., McAfee, L. C., McMahon, P. L., and Olukotun, K. (2009). A highly scalable restricted Boltzmann machine FPGA implementation. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 367–372. IEEE.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012a). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012b). ImageNet classification with deep convolutional neural networks. In *NIPS'2012*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324.

Nair, V. and Hinton, G. (2010). Rectified linear units improve restricted Boltzmann machines. In *ICML'2010*.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. Deep Learning and Unsupervised Feature Learning Workshop, NIPS.

Pham, P.-H., Jelaca, D., Farabet, C., Martini, B., LeCun, Y., and Culurciello, E. (2012). Neuflow: dataflow vision processing system-on-a-chip. In *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on*, pages 1044–1047. IEEE.

Presley, R. K. and Haggard, R. L. (1994). A fixed point implementation of the backpropagation learning algorithm. In *Southeastcon'94. Creative Technology Transfer-A Global Affair., Proceedings of the 1994 IEEE*, pages 136–138. IEEE.

Savich, A. W., Moussa, M., and Areibi, S. (2007). The impact of arithmetic representation on implementing mlp-bp on fpgas: A study. *Neural Networks, IEEE Transactions on*, **18**(1), 240–252.

Simard, P. and Graf, H. P. (1994). Backpropagation without multiplication. In *Advances in Neural Information Processing Systems*, pages 232–239.

Srebro, N. and Shraibman, A. (2005). Rank, trace-norm and max-norm. In *Proceedings of the 18th Annual Conference on Learning Theory*, pages 545–560. Springer-Verlag.

Vanhoucke, V., Senior, A., and Mao, M. Z. (2011). Improving the speed of neural networks on cpus. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*.

Wawrzynek, J., Asanovic, K., Kingsbury, B., Johnson, D., Beck, J., and Morgan, N. (1996). Spert-ii: A vector microprocessor system. *Computer*, **29**(3), 79–86.

Williamson, D. (1991//). Dynamically scaled fixed point arithmetic. pages 315 – 18, New York, NY, USA. dynamic scaling;iteration stages;digital filters;overflow probability;fixed point arithmetic;fixed-point filter;.

Zeiler, M. D. and Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. In *International Conference on Learning Representations*.