

# NEURAL NETWORKS WITH FEW MULTIPLICATIONS

**Zhouhan Lin**

Université de Montréal  
Canada

zhouhan.lin@umontreal.ca

**Matthieu Courbariaux**

Université de Montréal  
Canada

matthieu.courbariaux@gmail.com

**Roland Memisevic**

Université de Montréal  
Canada

roland.umontreal@gmail.com

**Yoshua Bengio**

Université de Montréal  
Canada

## ABSTRACT

For most deep learning algorithms training is notoriously time consuming. Since most of the computation in training neural networks is typically spent on floating point multiplications, we investigate an approach to training that eliminates the need for most of these. Our method consists of two parts: First we stochastically binarize weights to convert multiplications involved in computing hidden states to sign changes. Second, while back-propagating error derivatives, in addition to binarizing the weights, we quantize the representations at each layer to convert the remaining multiplications into binary shifts. Experimental results across 3 popular datasets (MNIST, CIFAR10, SVHN) show that this approach not only does not hurt classification performance but can result in even better performance than standard stochastic gradient descent training, paving the way to fast, hardware-friendly training of neural networks.

## 1 INTRODUCTION

Training deep neural networks has long been computational demanding and time consuming. For some state-of-the-art architectures, it can take weeks to get models trained (Krizhevsky et al., 2012). Another problem is that the demand for memory can be huge. For example, many common models in speech recognition or machine translation need 12 Gigabytes or more of storage (Gulcehre et al., 2015). To deal with these issues it is common to train deep neural networks by resorting to GPU or CPU clusters and to well designed parallelization strategies (Le, 2013).

Most of the computation performed in training a neural network are floating point multiplications. In this paper, we focus on eliminating most of these multiplications to reduce computation. Based on our previous work (Courbariaux et al., 2015), which eliminates multiplications in computing hidden representations by binarizing weights, our method deals with both hidden state computations and backward weight updates. Our approach has 2 components. In the forward pass, weights are stochastically binarized using an approach we call *binary connect* or *ternary connect*, and for back-propagation of errors, we propose a new approach which we call *quantized back propagation* that converts multiplications into bit-shifts.<sup>1</sup>

## 2 RELATED WORK

Several approaches have been proposed in the past to simplify computations in neural networks. Some of them try to restrict weight values to be an integer power of two, thus to reduce all the multiplications to be binary shifts (Kwan & Tang, 1993; Marchesi et al., 1993). In this way, multiplications are eliminated in both training and testing time. The disadvantage is that model performance can be severely reduced, and convergence of training can no longer be guaranteed.

<sup>1</sup>The codes for these approaches are available online at <https://github.com/hantek/BinaryConnect>

Kim & Paris (2015) introduces a completely Boolean network, which simplifies the test time computation at an acceptable performance hit. The approach still requires a real-valued, full precision training phase, however, so the benefits of reducing computations does not apply to training. Similarly, Machado et al. (2015) manage to get acceptable accuracy on sparse representation classification by replacing all floating-point multiplications by integer shifts. Bit-stream networks (Burge et al., 1999) also provides a way of binarizing neural network connections, by substituting weight connections with logical gates. Similar to that, Cheng et al. (2015) proves deep neural networks with binary weights can be trained to distinguish between multiple classes with expectation back propagation.

There are some other techniques, which focus on reducing the training complexity. For instance, instead of reducing the precision of weights, Simard & Graf (1994) quantizes states, learning rates, and gradients to powers of two. This approach manages to eliminate multiplications with negligible performance reduction.

### 3 BINARY AND TERNARY CONNECT

#### 3.1 BINARY CONNECT REVISITED

In Courbariaux et al. (2015), we introduced a weight binarization technique which removes multiplications in the forward pass. We summarize this approach in this subsection, and introduce an extension to it in the next.

Consider a neural network layer with  $N$  input and  $M$  output units. The forward computation is  $\mathbf{y} = h(W\mathbf{x} + \mathbf{b})$  where  $W$  and  $\mathbf{b}$  are weights and biases, respectively,  $h$  is the activation function, and  $\mathbf{x}$  and  $\mathbf{y}$  are the layer’s inputs and outputs. If we choose ReLU as  $h$ , there will be no multiplications in computing the activation function, thus all multiplications reside in the matrix product  $W\mathbf{x}$ . For each input vector  $x$ ,  $NM$  floating point multiplications are needed.

Binary connect eliminates these multiplications by stochastically sampling weights to be  $-1$  or  $1$ . Full precision weights  $\bar{w}$  are kept in memory as reference, and each time when  $y$  is needed, we sample a stochastic weight matrix  $W$  according to  $\bar{w}$ . For each element of the sampled matrix  $W$ , the probability of getting a  $1$  is proportional to how “close” its corresponding entry in  $\bar{w}$  is to  $1$ . i.e.,

$$P(W_{ij} = 1) = \frac{\bar{w}_{ij} + 1}{2}; \quad P(W_{ij} = -1) = 1 - P(W_{ij} = 1) \quad (1)$$

It is necessary to add some edge constraints to  $\bar{w}$ . To ensure that  $P(W_{ij} = 1)$  lies in a reasonable range, values in  $\bar{w}$  are forced to be a real value in the interval  $[-1, 1]$ . If during the updates any of its value grows beyond that interval, we set it to be its corresponding edge values  $-1$  or  $1$ . That way floating point multiplications become sign changes.

A remaining question concerns the use of multiplications in the random number generator involved in the sampling process. **Sampling an integer has to be faster than multiplication for the algorithm to be worth it.** To be precise, in most cases we are doing mini-batch learning and the sampling process is performed only once for the whole mini-batch. Normally the batch size  $B$  varies up to several hundreds. So, as long as one sampling process is significantly faster than  $B$  times of multiplications, it is still worth it. Fortunately, efficiently generating random numbers has been studied in Jeavons et al. (1994); van Daalen et al. (1993). Also, it is possible to get random numbers according to real random processes, like CPU temperatures, etc. We are not going into the details of random number generation as this is not the focus of this paper.

#### 3.2 TERNARY CONNECT

The binary connect introduced in the former subsection allows weights to be  $-1$  or  $1$ . However, in a trained neural network, it is common to observe that many learned weights are zero or close to zero. Although the stochastic sampling process would allow the mean value of sampled weights to be zero, this suggests that it may be beneficial to explicitly allow weights to be zero.

To allow weights to be zero, some adjustments are needed for Eq. 1. We split the interval of  $[-1, 1]$ , within which the full precision weight value  $\bar{w}_{ij}$  lies, into two sub-intervals:  $[-1, 0]$  and  $(0, 1]$ . If a

weight value  $\bar{w}_{ij}$  drops into one of them, we sample  $\bar{w}_{ij}$  to be the two edge values of that interval, according to their distance from  $\bar{w}_{ij}$ , i.e., if  $\bar{w}_{ij} > 0$ :

$$P(W_{ij} = 1) = \bar{w}_{ij}; \quad P(W_{ij} = 0) = 1 - \bar{w}_{ij} \quad (2)$$

and if  $\bar{w}_{ij} \leq 0$ :

$$P(W_{ij} = -1) = -\bar{w}_{ij}; \quad P(W_{ij} = 0) = 1 + \bar{w}_{ij} \quad (3)$$

Like binary connect, ternary connect also eliminates all multiplications in the forward pass.

#### 4 QUANTIZED BACK PROPAGATION

In the former section we described how multiplications can be eliminated from the forward pass. In this section, we propose a way to eliminate multiplications from the backward pass.

Suppose the  $i$ -th layer of the network has  $N$  input and  $M$  output units, and consider an error signal  $\delta$  propagating downward from its output. The updates for weights and biases would be the outer product of the layer's input and the error signal:

$$\Delta W = \eta \left[ \delta \odot h' (W\mathbf{x} + b) \right] \mathbf{x}^T \quad (4)$$

$$\Delta b = \eta \left[ \delta \odot h' (W\mathbf{x} + b) \right] \quad (5)$$

where  $\eta$  is the learning rate, and  $\mathbf{x}$  the input to the layer. The operator  $\odot$  stands for element-wise multiply. While propagating through the layers, the error signal  $\delta$  needs to be updated, too. Its update taking into account the next layer below takes the form:

$$\delta = [W^T \delta] \odot h' (W\mathbf{x} + b) \quad (6)$$

There are 3 terms that appear repeatedly in Eqs. 4 to 6:  $\delta$ ,  $h' (W\mathbf{x} + b)$  and  $\mathbf{x}$ . The latter two terms introduce matrix outer products. To eliminate multiplications, we can quantize one of them to be an integer power of 2, so that multiplications involving that term become binary shifts. The expression  $h' (W\mathbf{x} + b)$  contains downflowing gradients, which are largely determined by the cost function and network parameters, thus it is hard to bound its values. However, bounding the values is essential for quantization because we need to supply a fixed number of bits for each sampled value, and if that value varies too much, we will need too many bits for the exponent. This, in turn, will result in the need for more bits to store the sampled value and unnecessarily increase the required amount of computation.

While  $h' (W\mathbf{x} + b)$  is not a good choice for quantization,  $\mathbf{x}$  is a better choice, because it is the hidden representation at each layer, and we know roughly the distribution of each layer's activation.

Our approach is therefore to eliminate multiplications in Eq. 4 by quantizing each entry in  $\mathbf{x}$  to an integer power of 2. That way the outer product in Eq. 4 becomes a series of bit shifts. Experimentally, we find that allowing a maximum of 3 to 4 bits of shift is sufficient to make the network work well. This means that 3 bits are already enough to quantize  $\mathbf{x}$ . As the float32 format has 24 bits of mantissa, shifting (to the left or right) by 3 to 4 bits is completely tolerable. We refer to this approach of back propagation as "quantized back propagation."

If we choose ReLU as the activation function, and since we are reusing the  $(W\mathbf{x} + b)$  that was computed during the forward pass, computing the term  $h' (W\mathbf{x} + b)$  involves no additional sampling or multiplications. In addition, quantized back propagation eliminates the multiplications in the outer product in Eq. 4. The only places where multiplications remain are the element-wise products. In Eq. 5, multiplying by  $\eta$  and  $\sigma$  requires  $2 \times M$  multiplications, while in Eq. 4 we can reuse the result of Eq. 5. To update  $\delta$  would need another  $M$  multiplications, thus  $3 \times M$  multiplications

are needed for all computations from Eqs. 4 through 6. Pseudo code in Algorithm 1 outlines how quantized back propagation is conducted.

**Algorithm 1** Quantized Back Propagation (QBP).  $C$  is the cost function.  $\text{binarize}(W)$  and  $\text{clip}(W)$  stands for binarize and clip methods.  $L$  is the number of layers.

**Require:** a deep model with parameters  $W$ ,  $b$  at each layer. Input data  $x$ , its corresponding targets  $y$ , and learning rate  $\eta$ .

---

```

1: procedure QBP(model,  $x$ ,  $y$ ,  $\eta$ )
2:   1. Forward propagation:
3:   for each layer  $i$  in range( $1, L$ ) do
4:      $W_b \leftarrow \text{binarize}(W)$ 
5:     Compute activation  $a_i$  according to its previous layer output  $a_{i-1}$ ,  $W_b$  and  $b$ .
6:   2. Backward propagation:
7:   Initialize output layer's error signal  $\delta = \frac{\partial C}{\partial a_L}$ .
8:   for each layer  $i$  in range( $L, 1$ ) do
9:     Compute  $\Delta W$  and  $\Delta b$  according to Eqs. 4 and 5.
10:    Update  $W$ :  $W \leftarrow \text{clip}(W - \Delta W)$ 
11:    Update  $b$ :  $b \leftarrow b - \Delta b$ 
12:    Compute  $\frac{\partial C}{\partial a_{k-1}}$  by updating  $\delta$  according to Eq. 6.

```

---

Like in the forward pass, most of the multiplications are used in the weight updates. Compared with standard back propagation, which would need  $2MN + 3M$  multiplications at least, the amount of multiplications left is negligible in quantized back propagation. Our experiments in Section 5 show that this way of dramatically decreasing multiplications does not necessarily entail a loss in performance.

## 5 EXPERIMENTS

We tried our approach on both fully connected networks and convolutional networks. Our implementation uses Theano (Bastien et al., 2012). We experimented with 3 datasets: MNIST, CIFAR10, and SVHN. In the following subsection we show the performance that these multiplier-light neural networks can achieve. In the subsequent subsections we study some of their properties, such as convergence and robustness, in more detail.

### 5.1 GENERAL PERFORMANCE

We tested different variations of our approach, and compare the results with Courbariaux et al. (2015) and full precision training (Table 1). All models are trained with stochastic gradient descent (SGD) without momentum. We use batch normalization for all the models to accelerate learning. At training time, binary (ternary) connect and quantized back propagation are used, while at test time, we use the learned full resolution weights for the forward propagation. For each dataset, all hyper-parameters are set to the same values for the different methods, except that the learning rate is adapted independently for each one.

Table 1: Performances across different datasets

	Full precision	Binary connect	Binary connect + Quantized backprop	Ternary connect + Quantized backprop
MNIST	1.33%	1.23%	1.29%	1.15%
CIFAR10	15.64%	12.04%	12.08%	12.01%
SVHN	2.85%	2.47%	2.48%	2.42%

### 5.1.1 MNIST

The MNIST dataset (LeCun et al., 1998) has 50000 images for training and 10000 for testing. All images are grey value images of size  $28 \times 28$  pixels, falling into 10 classes corresponding to the 10 digits. The model we use is a fully connected network with 4 layers: 784-1024-1024-1024-10. At the last layer we use the hinge loss as the cost. The training set is separated into two parts, one of which is the training set with 40000 images and the other the validation set with 10000 images. Training is conducted in a mini-batch way, with a batch size of 200.

With ternary connect, quantized backprop, and batch normalization, we reach an error rate of 1.15%. This result is better than full precision training (also with batch normalization), which yields an error rate 1.33%. If without batch normalization, the error rates rise to 1.48% and 1.67%, respectively. We also explored the performance if we sample those weights during *test time*. With ternary connect at test time, the same model (the one reaches 1.15% error rate) yields 1.49% error rate, which is still fairly acceptable. Our experimental results show that despite removing most multiplications, our approach yields a comparable (in fact, even slightly higher) performance than full precision training. The performance improvement is likely due to the regularization effect implied by the stochastic sampling.

Taking this network as a concrete example, the actual amount of multiplications in each case can be estimated precisely. Multiplications in the forward pass is obvious, and for the backward pass section 4 has already given an estimation. Now we estimate the amount of multiplications incurred by batch normalization. Suppose we have a pre-hidden representation  $h$  with mini-batch size  $B$  on a layer which has  $M$  output units (thus  $h$  should have shape  $B \times M$ ), then batch normalization can be formalized as  $\gamma \frac{h - \text{mean}(h)}{\text{std}(h)} + \beta$ . One need to compute the  $\text{mean}(h)$  over a mini-batch, which takes  $M$  multiplications, and  $BM + 2M$  multiplication to compute the standard deviation  $\text{std}(h)$ . The fraction takes  $BM$  divisions, which should be equal to the same amount of multiplication. Multiplying that by the  $\gamma$  parameter, adds another  $BM$  multiplications. So each batch normalization layer takes an extra  $3BM + 3M$  multiplications in the forward pass. The backward pass takes roughly twice as many multiplications in addition, if we use SGD. These amount of multiplications are the same no matter we use binarization or not. Bearing those in mind, the total amount of multiplications invoked in a mini-batch update are shown in Table 2. The last column lists the ratio of multiplications left, after applying ternary connect and quantized back propagation.

Table 2: Estimated number of multiplications in MNIST net

	Full precision	Ternary connect + Quantized backprop	ratio
without BN	$1.7480 \times 10^9$	$1.8492 \times 10^6$	0.001058
with BN	$1.7535 \times 10^9$	$7.4245 \times 10^6$	0.004234

### 5.1.2 CIFAR10

CIFAR10 (Krizhevsky & Hinton, 2009) contains images of size  $32 \times 32$  RGB pixels. Like for MNIST, we split the dataset into 40000, 10000, and 10000 training-, validation-, and test-cases, respectively. We apply our approach in a convolutional network for this dataset. The network has 6 convolution/pooling layers, 1 fully connected layer and 1 classification layer. We use the hinge loss for training, with a batch size of 100. We also tried using ternary connect at test time. On the model trained by ternary connect and quantized back propagation, it yields 13.54% error rate. Similar to what we observed in the fully connected network, binary (ternary) connect and quantized back propagation yield a slightly higher performance than ordinary SGD.

### 5.1.3 SVHN

The Street View House Numbers (SVHN) dataset (Netzer et al., 2011) contains RGB images of house numbers. It contains more than 600,000 images in its extended training set, and roughly 26,000 images in its test set. We remove 6,000 images from the training set for validation. We use 7 layers of convolution/pooling, 1 fully connected layer, and 1 classification layer. Batch size is also

set to be 100. The performances we get is consistent with our results on CIFAR10. Extending the ternary connect mechanism to its test time yields 2.99% error rate on this dataset. Again, it improves over ordinary SGD by using binary (ternary) connect and quantized back propagation.

## 5.2 CONVERGENCE

Taking the convolutional networks on CIFAR10 as a test-bed, we now study the learning behaviour in more detail. Figure 1 shows the performance of the model in terms of test set errors during training. The figure shows that binarization makes the network converge slower than ordinary SGD, but yields a better optimum after the algorithm converges. Compared with binary connect (red line), adding quantization in the error propagation (yellow line) doesn't hurt the model accuracy at all. Moreover, having ternary connect combined with quantized back propagation (green line) surpasses all the other three approaches.

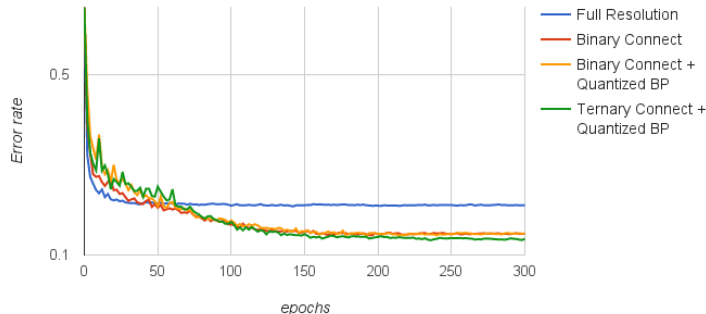


Figure 1: Test set error rate at each epoch for ordinary back propagation, binary connect, binary connect with quantized back propagation, and ternary connect with quantized back propagation. Vertical axis is represented in logarithmic scale.

## 5.3 THE EFFECT OF BIT CLIPPING

In Section 4 we mentioned that quantization will be limited by the number of bits we use. The maximum number of bits to shift determines the amount of memory needed, but it also determines in what range a single weight update can vary. Figure 2 shows the model performance as a function of the maximum allowed bit shifts. These experiments are conducted on the MNIST dataset, with the aforementioned fully connected model. For each case of bit clipping, we repeat the experiment for 10 times with different initial random instantiations.

The figure shows that the approach is not very sensible to the number of bits used. The maximum allowed shift in the figure varies from 2 bits to 10 bits, and the performance remains roughly the same. Even by restricting bit shifts to 2, the model can still learn successfully. The fact that the performance is not very sensitive to the maximum of allowed bit shifts suggests that we do not need to redefine the number of bits used for quantizing  $x$  for different tasks, which would be an important practical advantage.

The  $x$  to be quantized is not necessarily distributed symmetrically around 2. For example, Figure 3 shows the distribution of  $x$  at each layer in the middle of training. The maximum amount of shift to the left does not need to be the same as that on the right. A more efficient way is to use different values for the maximum left shift and the maximum right shift. Bearing that in mind, we set it to 3 bits maximum to the right and 4 bits to the left.

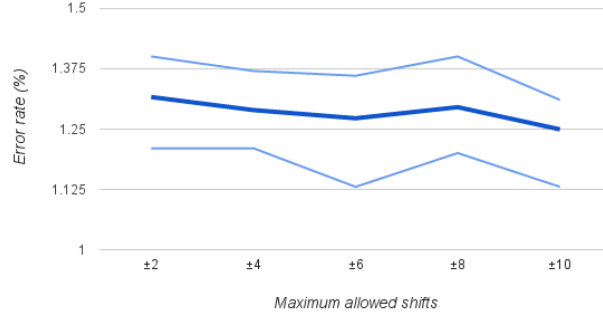


Figure 2: Model performance as a function of the maximum bit shifts allowed in quantized back propagation. The dark blue line indicates mean error rate over 10 independent runs, while light blue lines indicate their corresponding maximum and minimum error rates.

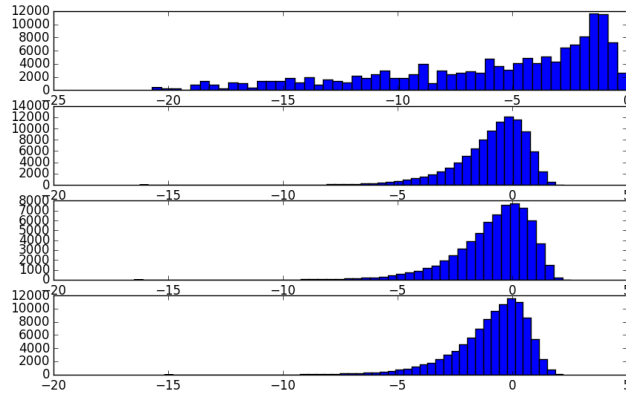


Figure 3: Histogram of representations at each layer while training a fully connected network for MNIST. The figure represents a snap-shot in the middle of training. Each subfigure, from bottom up, represents the histogram of hidden states from the first layer to the last layer. The horizontal axes stand for the exponent of the layers' representations, i.e.,  $\log_2 x$ .

## 6 CONCLUSION AND FUTURE WORK

We proposed a way to eliminate most of the floating point multiplications used during training a feedforward neural network. This could make it possible to dramatically accelerate the training of neural networks by using dedicated hardware implementations.

A somewhat surprising fact is that instead of damaging prediction accuracy the approach tends improve it, which is probably due to several facts. First is the regularization effect that the stochastic sampling process entails. Noise injection brought by sampling the weight values can be viewed as a regularizer, and that improves the model generalization. The second fact is low precision weight values. Basically, the generalization error bounds for neural nets depend on the weights precision. Low precision prevents the optimizer from finding solutions that require a lot of precision, which correspond to very thin (high curvature) critical points, and these minima are more likely to correspond to overfitted solutions than broad minima (there are more functions that are compatible with such solutions, corresponding to a smaller description length and thus better generalization). Similarly,

Neelakantan et al. (2015) adds noise into gradients, which makes the optimizer prefer large-basin areas and forces it to find broad minima. It also lowers the training loss and improves generalization.

Directions for future work include exploring actual implementations of this approach (for example, using FPGA), seeking more efficient ways of binarization, and the extension to recurrent neural networks.

#### ACKNOWLEDGMENTS

The authors would like to thank the developers of Theano (Bastien et al., 2012). We acknowledge the support of the following agencies for research funding and computing support: Samsung, NSERC, Calcul Québec, Compute Canada, the Canada Research Chairs and CIFAR.

#### REFERENCES

- Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Bergstra, James, Goodfellow, Ian J., Bergeron, Arnaud, Bouchard, Nicolas, and Bengio, Yoshua. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.
- Burge, Peter S., van Daalen, Max R., Rising, Barry J. P., and Shawe-Taylor, John S. Stochastic bit-stream neural networks. In Maass, Wolfgang and Bishop, Christopher M. (eds.), *Pulsed Neural Networks*, pp. 337–352. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-626-13350-4. URL <http://dl.acm.org/citation.cfm?id=296533.296552>.
- Cheng, Zhiyong, Soudry, Daniel, Mao, Zexi, and Lan, Zhenzhong. Training binary multilayer neural networks for image classification using expectation backpropagation. *arXiv preprint arXiv:1503.03562*, 2015.
- Courbariaux, Matthieu, Bengio, Yoshua, and David, Jean-Pierre. Binaryconnect: Training deep neural networks with binary weights during propagations. *arXiv preprint arXiv:1511.00363*, 2015.
- Gulcehre, Caglar, Firat, Orhan, Xu, Kelvin, Cho, Kyunghyun, Barrault, Loic, Lin, Huei-Chi, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. On using monolingual corpora in neural machine translation. *arXiv preprint arXiv:1503.03535*, 2015.
- Jeavons, Peter, Cohen, David A., and Shawe-Taylor, John. Generating binary sequences for stochastic computing. *Information Theory, IEEE Transactions on*, 40(3):716–720, 1994.
- Kim, Minje and Paris, Smaragdis. Bitwise neural networks. In *Proceedings of The 31st International Conference on Machine Learning*, pp. 0–0, 2015.
- Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images, 2009.
- 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.
- Kwan, Hon Keung and Tang, CZ. Multiplierless multilayer feedforward neural network design suitable for continuous input-output mapping. *Electronics Letters*, 29(14):1259–1260, 1993.
- Le, Quoc V. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 8595–8598. IEEE, 2013.
- 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.
- Machado, Emerson Lopes, Miosso, Cristiano Jacques, von Borries, Ricardo, Coutinho, Murilo, Berger, Pedro de Azevedo, Marques, Thiago, and Jacobi, Ricardo Pezzuol. Computational cost reduction in learned transform classifications. *arXiv preprint arXiv:1504.06779*, 2015.
- Marchesi, Michele, Orlandi, Gianni, Piazza, Francesco, and Uncini, Aurelio. Fast neural networks without multipliers. *Neural Networks, IEEE Transactions on*, 4(1):53–62, 1993.



- Neelakantan, Arvind, Vilnis, Luke, Le, Quoc V, Sutskever, Ilya, Kaiser, Lukasz, Kurach, Karol, and Martens, James. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.
- Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, pp. 5. Granada, Spain, 2011.
- Simard, Patrice Y and Graf, Hans Peter. Backpropagation without multiplication. In *Advances in Neural Information Processing Systems*, pp. 232–239, 1994.
- van Daalen, Max, Jeavons, Pete, Shawe-Taylor, John, and Cohen, Dave. Device for generating binary sequences for stochastic computing. *Electronics Letters*, 29(1):80–81, 1993.