Stochastic Computing for Deep Neural Networks

MEng Final Year Project

Adamos Solomou

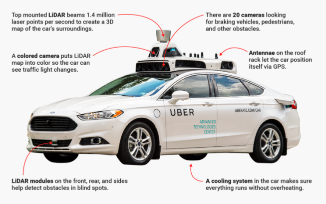
June 27, 2018

Imperial College London
Circuits and Systems Research Group

Motivation & Background

The Deep Learning Era

- Deep Neural Networks have been revolutionizing machine learning research and applications.
- Surpassed human perception in many recognition tasks.



Computational Complexity of DNNs

- Deep architectures require more computations compared to typical machine learning techniques.
- Lead to long training times and large amount of computational resources.
- Currently, large scale DNNs are deployed on high performance computing clusters.
- Restricts deployment of DNNs in **resource constraint systems**: Mobile devices and embedded systems.

Configuration	Time to complete 100 epochs
1 GPU	10.5 days
2 GPUs Model parallelism	6.6 days
4 GPUs model + data parallelism	4.8 days

Table information adopted by: Yadan et al. Multi-GPU Training of ConvNets [1]

Stochastic Computing

- We consider **Stochastic Computing** (SC) as a novel, low-cost alternative to conventional binary computing.
- SC uses random pulse sequences as information carriers.
- A **probability number** $p \in [0, 1]$ is represented by a bit-stream X of chosen length

$$X = (X_{n-1}, X_{n-2}, ..., X_0)$$

  where $P(X_i = 1) = p$.
- The encoded quantity x is **estimated** by:

$$x = \frac{1}{n} \sum_{i=0}^{n-1} X_i$$

- Can be extended to represent numbers in the range $[-1, 1]$ through the linear mapping $y = 2p - 1$

Example

The bit-streams $X = (0, 1, 1, 1) \quad Y = (1, 0, 1, 1) \quad Z = (0, 1, 1, 1, 1, 1, 1, 0)$ represent $p = 0.75$

Arithmetic Units in Stochastic Computing

Representation systems based on SC are **redundant**.

- Enables very **low-complexity** arithmetic units.

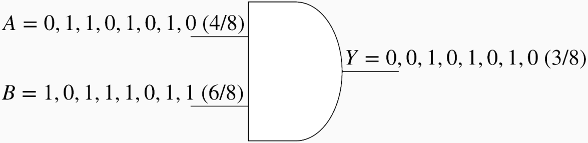
Multiplication

Inputs: A and B

Output: $Y = A \cdot B$

$P_Y = P_A \times P_B$

$\Rightarrow y = a \times b$



Scaled Addition

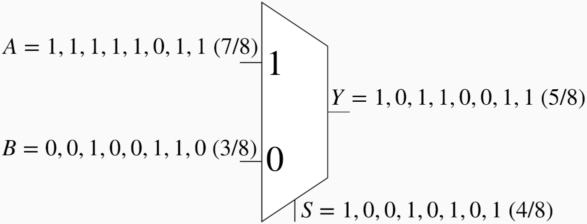
Inputs: A and B

Output: Y

Select line: $P_S = 0.5$

$P_Y = (P_A + P_B)/2$

$\Rightarrow y = (a + b)/2$



Related Work

Previous attempts to implement DNNs using SC:

- Yuan Ji et al. [2015] implement a RBF neural network using SC.
- Ao Ren et al. [2017] present SC hardware designs for the implementation of CNNs. Focus is given on:
  - Weight storage schemes
  - Structure optimization techniques

  Aiming to minimize hardware area and power consumption.

Existing work only considers neural network **inference** using SC.

- Only the stochastic implementation of the hyperbolic tangent is considered.

There exists no work investigating how SC can be incorporated within the **training** stage of a DNN.

Main Contributions

The main contributions of this project are:

- The introduction of a stochastic rectified linear unit (ReLU).

Main Contributions

The main contributions of this project are:

- The introduction of a stochastic rectified linear unit (ReLU).
- The formulation and implementation of saturation arithmetic in SC.

Main Contributions

The main contributions of this project are:

- The introduction of a stochastic rectified linear unit (ReLU).
- The formulation and implementation of saturation arithmetic in SC.
- A thorough analysis of the scaling scheme used for DNN inference in SC

Main Contributions

The main contributions of this project are:

- The introduction of a stochastic rectified linear unit (ReLU).
- The formulation and implementation of saturation arithmetic in SC.
- A thorough analysis of the scaling scheme used for DNN inference in SC
- An investigation of DNN training using stochastic arithmetic.

Main Contributions

The main contributions of this project are:

- The introduction of a stochastic rectified linear unit (ReLU).
- The formulation and implementation of saturation arithmetic in SC.
- A thorough analysis of the scaling scheme used for DNN inference in SC
- An investigation of DNN training using stochastic arithmetic.
  - Modified neuron architectures are introduced.

Main Contributions

The main contributions of this project are:

- The introduction of a stochastic rectified linear unit (ReLU).
- The formulation and implementation of saturation arithmetic in SC.
- A thorough analysis of the scaling scheme used for DNN inference in SC
- An investigation of DNN training using stochastic arithmetic.
  - Modified neuron architectures are introduced.
  - An optimization-based scaling scheme is proposed to learn *optimal* saturation levels during training.

Neural Network Inference in SC

DNN Architecture

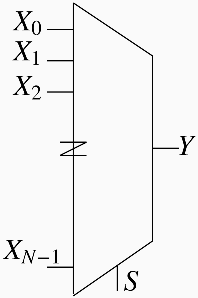
- A DNN consists of a chain of fully connected layers.
- The activations in a fully connected network are calculated as:

$$h^{(i)} = \phi(W^T h^{(i-1)} + b)$$

  where ϕ is the activation function.
- The main operation of a fully connected layer is the **inner product**.
- **ReLU** is a widely used activation function, $\phi(z) = max\{0, z\}$.

Stochastic Inner Product

- Inner product is not a closed operation on the interval $[-1, 1]$
- The computation is performed in a **scaled** manner.
- In each clock cycle:
  - Based on a probability distribution, select one of the inputs at random.
  - Connect it to the output.



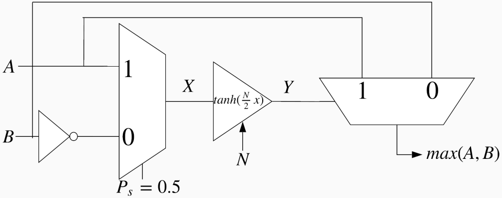
- A MUX which randomly selects an input i with some probability α_i such that $\sum_i \alpha_i = 1$ computes

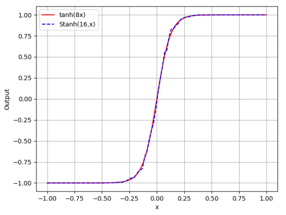
$$y = \sum_{i=0}^{N-1} \alpha_i x_i$$

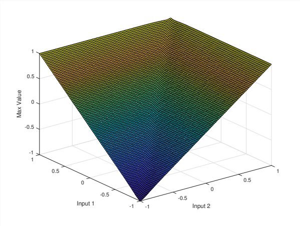
- Can be extended to facilitate arbitrary inner product computations $y = \mathbf{w}^T \mathbf{x}$, where $x_i \in [-1, 1]$.
- The result is down-scaled by $s_{dot} = \sum_i |w_i|$.

Stochastic ReLU

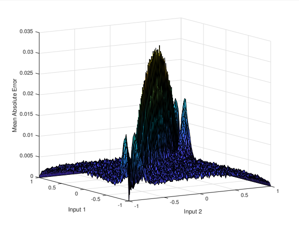
Based on the approximation of the **max function** in SC





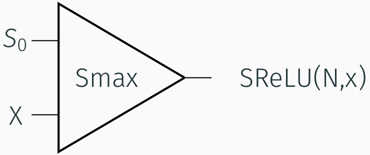


(a) Result of *Smax*

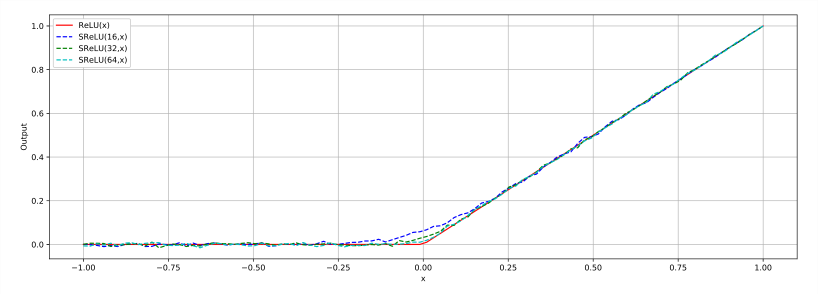


(b) Mean absolute error

Stochastic ReLU



S_0 is a stochastic sequence representing the value zero



Observations

- *SReLU* is a function of the parameter N
- The approximation is poorest around $x = 0$

Conversion Process

Conventional DNN

- Floating-point inputs
- Floating-point coefficients
- Floating-point operators

SC Compatible DNN

- n-bit inputs
- Compatible coefficients
- SC operators

Conversion process consists of the following steps:

1. Convert floating-point input data to stochastic bit-streams.

Conversion Process

Conventional DNN

- Floating-point inputs
- Floating-point coefficients
- Floating-point operators

SC Compatible DNN

- n-bit inputs
- Compatible coefficients
- SC operators

Conversion process consists of the following steps:

1. Convert floating-point input data to stochastic bit-streams.
2. Convert floating-point network coefficients to stochastic bit-streams.

Conversion Process

Conventional DNN

- Floating-point inputs
- Floating-point coefficients
- Floating-point operators

SC Compatible DNN

- *n*-bit inputs
- Compatible coefficients
- SC operators

Conversion process consists of the following steps:

1. Convert floating-point input data to stochastic bit-streams.
2. Convert floating-point network coefficients to stochastic bit-streams.
3. Construct the equivalent SC computational graph for the original neural network.

Conversion Process

Conventional DNN

- Floating-point inputs
- Floating-point coefficients
- Floating-point operators

SC Compatible DNN

- n-bit inputs
- Compatible coefficients
- SC operators

Conversion process consists of the following steps:

1. Convert floating-point input data to stochastic bit-streams.
2. Convert floating-point network coefficients to stochastic bit-streams.
3. Construct the equivalent SC computational graph for the original neural network.
    - **Scaling Scheme**: Internal scalings need to be computed

Conversion Process

Conventional DNN

- Floating-point inputs
- Floating-point coefficients
- Floating-point operators

SC Compatible DNN

- *n*-bit inputs
- Compatible coefficients
- SC operators

Conversion process consists of the following steps:

1. Convert floating-point input data to stochastic bit-streams.
2. Convert floating-point network coefficients to stochastic bit-streams.
3. Construct the equivalent SC computational graph for the original neural network.
    - **Scaling Scheme**: Internal scalings need to be computed
4. Convert the output of the SC network to its floating-point representation and compute the loss function.

Network I

Train a feedforward network on the MNIST dataset.

Network Architecture		
Input Layer	Number of Units	784
Hidden Layer	Number of Units Hidden Unit	128 Linear
Output Layer	Number of Units Output Unit Loss Function	10 Softmax Cross Entropy Error

Training Accuracy: 87.05% Testing Accuracy: 87.03%

Worst-case scalings:

Node	Input	Weight Product 1^{st} Layer	Bias Addition 1^{st} Layer	Weight Product Output Layer	Output
Scaling	1	2^{10}	2^{11}	2^{17}	2^{18}

$\Rightarrow$ Output lies within $[-2^{18}, 2^{18}]$.

Saturation Arithmetic in Stochastic Computing

Worst-case scalings are overly pessimistic.

- They accommodate the full-dynamic range of a computation.
- Large scalings may **degrade precision** in the computations.

Saturation arithmetic can be applied to reduce the scaling parameters.

- The scaled adder in SC computes $y = a \oplus b = \frac{(a+b)}{2}$
- A stochastic adder with **saturation** computes

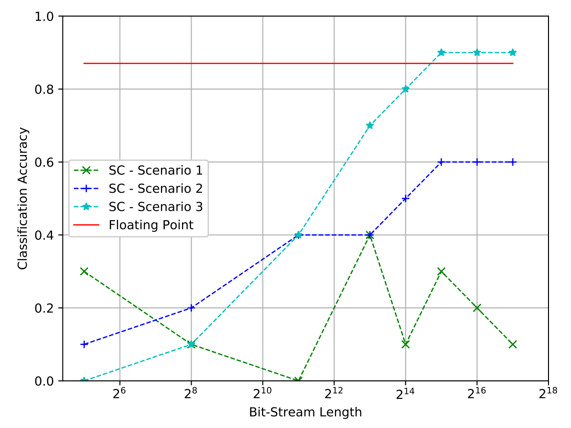
$$\hat{y} = a \hat{\oplus} b = \begin{cases} 1, & \frac{1}{2}(a+b) \geq M \\ \frac{1}{2M}(a+b), & \frac{1}{2}|a+b| < M \\ -1, & \frac{1}{2}(a+b) \leq -M \end{cases}$$

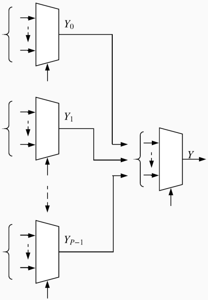
  where $M < 1$.
- **Compression error** is introduced if $a \oplus b \notin [-M, M]$.

Network I Inference in Stochastic Computing

Three scenarios are considered:

1. No saturation arithmetic is applied.
2. Saturation arithmetic is applied.
3. Saturation arithmetic and a decomposed inner product is used.





Training Stochastic Computing Neural Networks

Overview

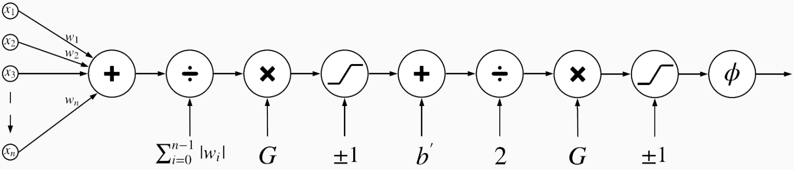
We propose an alternative training procedure, namely **SC compatible training**.

- Aims to capture the **limitations** of SC **during** the **training** phase.
  - Small dynamic range of SC.
  - SC implements scaled addition and inner product.
- Is there any benefit from following such a training procedure?

SC Neuron Architecture

A neuron architecture is proposed to **model** the main *features* of SC

- Scaled computations
- Saturation arithmetic



Stochastic arithmetic affects both the **forward** and **backward** propagation of the network.

- Software frameworks allow to model both using the proposed SC neuron architecture.

Learning Compatible Coefficients and Optimal Saturation Levels

Constrain network's coefficients to lie within the representable range

$$\begin{aligned} \underset{w,b}{\text{minimize}} \quad & \mathcal{J}(w, b) \\ \text{subject to} \quad & -1 \leq w \leq 1 \\ & -1 \leq b \leq 1 \end{aligned}$$

where $\mathcal{J}$ is the loss function.

Can be *approximately* solved using penalty functions

- L^1 and L^2 Regularization
- External penalty function

Can be extended to **learn** *optimal* saturation levels during training

$$\begin{aligned} \underset{w,b,g}{\text{minimize}} \quad & \mathcal{J}(w, b, g) \\ \text{subject to} \quad & -1 \leq w \leq 1 \\ & -1 \leq b \leq 1 \end{aligned}$$

where g denotes collectively the **gain** coefficients.

Experiments

Regularized Network Inference in SC

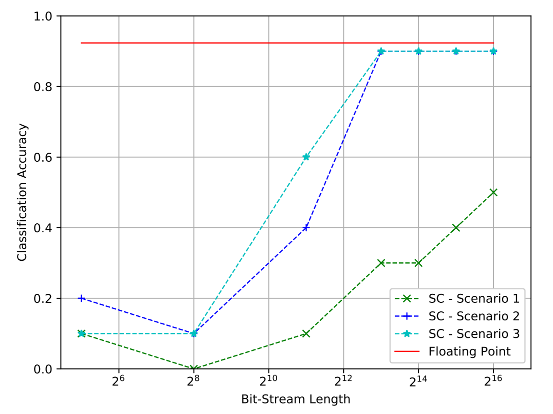
Training with different regularization techniques

	L^1 **Regularization**	L^2 **Regularization**	**Custom Penalty Function**
Training Accuracy	89.74 %	92.6 %	92.73 %
Testing Accuracy	89.88 %	92.34 %	92.1 %

Worst-case scaling parameters (L^2 regularization)

Node	Input	Weight Product 1^{st} Layer	Bias Addition 1^{st} Layer	Weight Product Output Layer	Output
Scaling	1	2^5	2^6	2^{11}	2^{12}

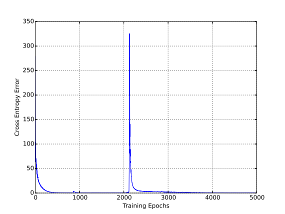
Regularized Network Inference in SC



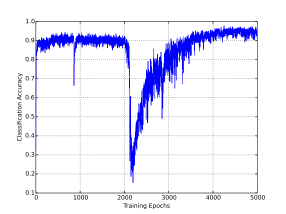
Observations

- Signal scalings are significantly smaller
- Gain coefficients (with saturation) are smaller
- A shorter bit-stream is needed in both scenarios two and three
  $\Rightarrow$ Faster execution

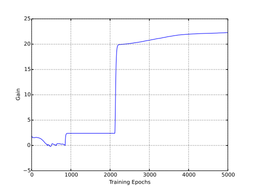
Training SC Compatible Neural Networks



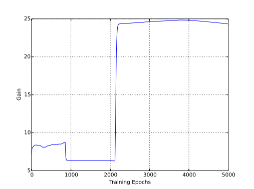
(a) Training loss versus epochs



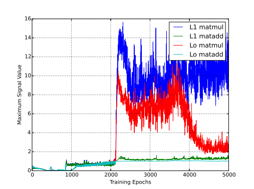
(b) Training accuracy versus epochs



(c) Gain parameter in the hidden layer



(d) Gain parameter in the output layer



(e) Maximum signal value prior clipping to ± 1

Training Accuracy: 96.52%

Testing Accuracy: 95.76%

Conclusions

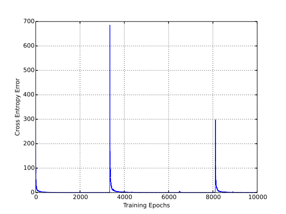
- Neural network **inference** in SC can indeed be achieved provided **saturation arithmetic** is applied.
- Convergence of the SC network accuracy is of the form $\sqrt{n}$, where n is the bit-stream length.
- Neural network inference in SC can significantly **benefit** by appropriately **regularizing** the **weights** of the network.
- The proposed **training** technique allows the **network** to develop its own **knowledge** regarding both the recognition task and the **stochastic representation**.
- The network identifies the limitations of SC.
    - Worst-case scalings are overly pessimistic.
    - Increases gain coefficients allowing non-zero saturation error.
    - **Learns** *optimal* **saturation levels** to avoid loosing precision.
- The proposed technique can even **improve** network's **accuracy**.

Future Work

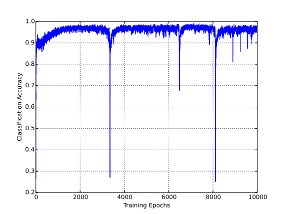
- A **hardware implementation** is required. It will allow to quantitatively assess:
    - Latency
    - Throughput
    - Power consumption
    - Saturation arithmetic overheads
- **Convolutional neural networks** should be considered.
    - The necessary computational units have been developed in this project.
- **Simulate training** using **SC** processing units
    - It will allow to conduct further research such the dynamic selection of the bit-stream length

Questions?

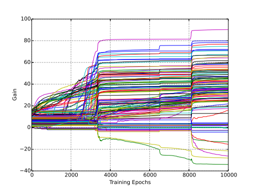
Training SC Compatible Neural Networks



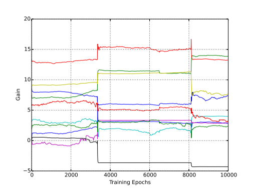
(a) Training loss versus epochs



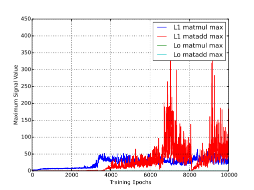
(b) Training accuracy versus epochs



(c) Gain parameter in the hidden layer



(d) Gain parameter in the output layer



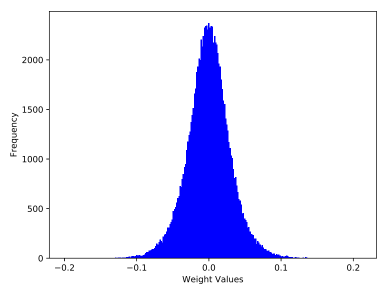
(e) Maximum signal value prior clipping to ± 1

Network Scaling Scheme

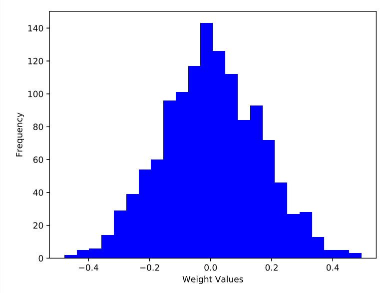
- SC can only represent numbers within the range $[-1, 1]$.
- Numbers outside this range need to be down-scaled accordingly.
- For the purpose of neural network inference, internal scalings can be computed in advance.
- The process is termed **scaling scheme**.
- The proposed scheme is based on **forward propagation** of known information on **data ranges** through the network graph.
- Data ranges are derived from **worst-case scalings**.

Weights i

Distribution of optimal weights using L^2 regularization



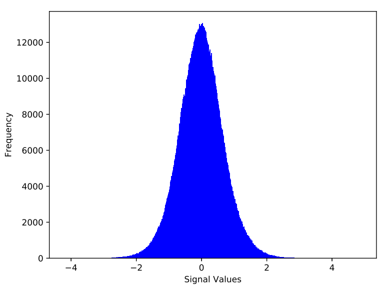
(a) Hidden Layer



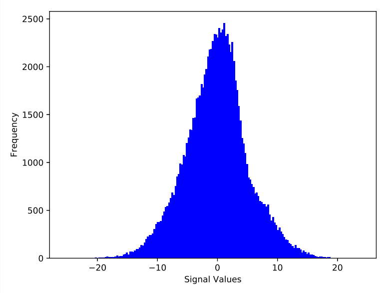
(b) Output Layer

Signal Values i

Signal values determined through simulation



(a) Hidden Layer



(b) Output Layer

O. Yadan, K. Adams, Y. Taigman, and M. Ranzato.
Multi-gpu training of convnets.
CoRR, abs/1312.5853, 2013.