Non-linear function evaluation reusing matrix-vector multipliers

Ce Guo[1], Wayne Luk[1] and Wenguang Xu[2]
[1]Imperial College London [2]Huawei Technologies
Email: [1]{c.guo, w.luk}@imperial.ac.uk [2] xuwenguang@huawei.com

Abstract

This paper presents a method to extend matrix-vector multipliers to support the evaluation of non-linear functions. The proposed approach introduces non-linearity by optionally overriding the input signals of the matrix-vector multiplier. The method aims to reduce the idleness of hardware resources during computation, to maximise the reuse of arithmetic units and internal structures in existing matrix-vector multipliers, and to reduce the effort in adding additional functions. From our analysis on a case with eight non-linear functions, the proposed design consumes fewer components for addition, multiplication, division, exponentiation and logarithm than a reference design with dedicated function evaluation facilities.

1 Introduction

Two types of operations consume the majority of execution time in various data processing tasks. One type of operation is linear combination that takes the weighted sum of a vector against a dense weight matrix. The other type of operation is non-linear function evaluation that evaluates a function for all entries in a vector. The sequential dependencies between the calculations only allow the execution of one type of operation at the same time. For instance, the forward and backward propagation operations for fully connected layers in neural networks follow this pattern.

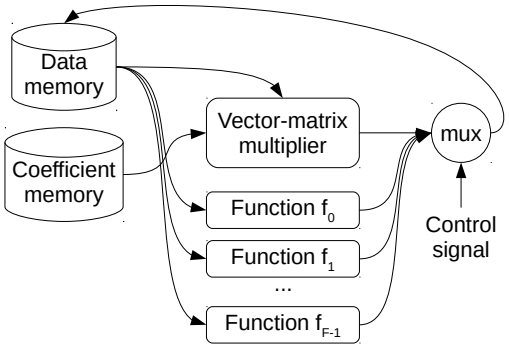
Linear combination and non-linear function evaluation are fundamentally different regarding arithmetic operations. A straightforward way to design hardware is to create a separate arithmetic block for each operation [3]. When the arithmetic block for one operation is active, the blocks for other operations are idle. In other words, only a small fraction of logic units work at the same time, resulting in wastage of hardware resources. Admittedly, it is possible to use reconfigurable hardware to time-multiplex hardware resources for different operations [1]. However, reconfigurable hardware runs at lower frequencies, and run-time reconfiguration may take a considerable amount of execution time.

This paper presents a method to extend matrix-vector multipliers to support non-linear function evaluation to counter drawbacks brought by dedicated function evaluations blocks while avoiding run-time reconfiguration.

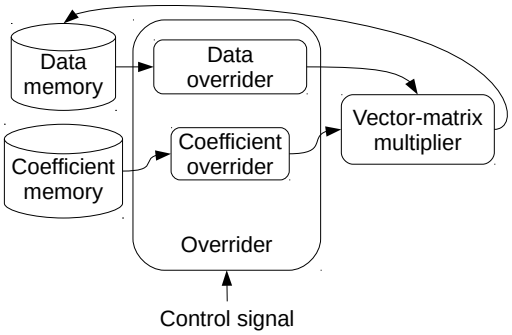
2 Motivation

Matrix-vector multiplication is a well-studied topic in hardware design [2]. A matrix-vector multiplier for arbitrary-width inputs often uses fixed-width matrix-vector multipliers as computational kernels. A scheduling logic partitions high-dimensional tasks into tiles that fit the fixed-width matrix-vector multiplier.

The transformation that the matrix-vector multiplier can apply must be linear. Purely linear models may not be able to capture the pattern of real-life data. It is often desirable for a data processing system to evaluate non-linear functions in addition to the calculation of linear combination. An obvious way to evaluate non-linear functions on hardware is to design a dedicated datapath for each non-linear function, as shown in design D in Figure 1(a). A multiplexer selects one result vector among the outputs of the matrix-vector multiplier and a group of non-linear functions. This method has three drawbacks. First, the datapath ignored by the multiplexer should either stay idle or compute useless results. At any moment, at most one branch among the matrix-vector multiplier and the non-linear functions contributes to the final result. Second, a practical design needs to support multiple non-linear functions. Each of these function may be complicated so that a data path for accurate evaluation may occupy a large area on the chip. Third, when it is necessary to support addi-



(a) Design D: non-linearity with dedicated datapaths



(b) Design P: proposed method with overriders

Figure 1: Architectures for linear combination and non-linear function evaluation

tional non-linear functions or change the definition of existing ones, it is necessary to redesign the datapaths, which may result in considerable development cost.

This paper tackles the three drawbacks by optionally overriding the inputs of the matrix-vector multiplier, as shown in design P in Figure 1(b). The primary objective is to extend the matrix-vector multiplier to support the evaluation of arbitrary non-linear functions without modifying its internal structure.

3 Mathematical derivation

In this section, we present the mathematical background of the proposed method. The key idea is to find a unified mathematical form for both linear combination and non-linear function evaluation. We aim to support the evaluation of arbitrary real-valued functions without introducing additional multipliers or changing the connections between adders and multipliers. Following this principle, we adopt piecewise linear approximation to compute approximate values for non-linear functions. In general, the piecewise linear approximation technique is mathematically simple. However, the approximation technique in this study is challenging because we aim to reuse the hardware resources in matrix-vector multipliers.

Our main achievement is to derive a mathematical foundation for this approximation technique, given by Eq. 1–11. The following function captures the behaviour of the hardware block that supports both linear and non-linear operations:

$$g(\vec{x}, m, k) = \sigma\big(\phi(W, m, k) \odot \psi(\vec{x}, m, k)\big) \qquad (1)$$

where $\sigma(\cdot)$ is the row-sum function; W is a $b \times b$ matrix; $\odot$ is the entrywise multiplication operator; $\phi(W, m, k)$ and $\psi(\vec{x}, o, k)$ are function that returning $b \times b$ matrices; the entry at position $[i, j]$ in $\phi(W, m, k)$ and $\psi(\vec{v}, m, k)$ are respectively:

$$\phi_{i,j}(W, m, k) = \begin{cases} W_{i,j} & m = 0 \\ \alpha_{k,i} & m = 1 \end{cases} \qquad (2)$$

$$\psi_{i,j}(\vec{x}, m, k) = \begin{cases} x_j & m = 0 \\ 0 & m = 1, x_i \le h_{k,j} \\ x_i & m = 1, h_{k,j} < x_i \le h_{k,j+1} \\ \beta_j & m = 1, h_{k,j+1} < x_i \end{cases} \qquad (3)$$

where m is a Boolean variable controlling the linearity of the operation; $k \in [0..K-1]$ that selects the non-linear function to evaluate; $\vec{\beta} = [\beta_0 \dots \beta_{B-1}]^T$ and $\vec{h} = [h_0 \dots h_b]^T$ are parameter vectors. Note that $\vec{\beta}$ has b entries while $\vec{h}$ has $(b+1)$ entries. The function in Eq. 1 supports linear and non-linear operations as follows. When $m = 0$, $\psi(\vec{x}, m, k)$ contains b copies of $\vec{x}$. Therefore, Eq. 1 downgrades to a function that takes the matrix-vector product $W\vec{x}$. As a result, one may evaluate a linear combination with arbitrary weight matrices by tiling and accumulation. When $m = 1$, x_i appears exactly once in the i-th row in $\psi(\vec{x}, o, k)$ by Eq. 3. The position that x_i appears corresponds to the interval that covers x_i; each entry before x_i takes a predefined value β_j. Each row of W is a copy of $\vec{\alpha}_k$. As a result, Eq. 1 becomes a piece-wise linear function that can approximate a non-linear function.

The set of parameters $\{\vec{h}_k, \vec{\alpha}_k, \vec{\beta}_k, \lambda_k\}$ defines the k-th non-linear function. In particular, $\vec{h}_k$ defines the boundary of the input for the approximation segments; $\vec{\alpha}_k$ consists of slopes of the segments; $\vec{\beta}_k$ carries differentiated intercepts scaled by $1/\vec{\alpha}$; λ gives the baseline intercept.

We present a way to compute the parameter set. Let $\vec{h}'_k = [h'_0 \dots h'_{b-2}]^T$ be an ascending vector of b numbers sampled from the domain of $f(\cdot)$. The pa-

rameters are as follows:

$$\vec{h}_k = \begin{bmatrix} -\infty & h'_{k,0} & \dots & h'_{k,b-2} & +\infty \end{bmatrix} \quad (4)$$

$$\vec{\alpha}_k = \begin{bmatrix} \alpha_\perp & \alpha'_{k,0} & \dots & \alpha'_{k,b-1} & \alpha_\top \end{bmatrix} \quad (5)$$

$$\beta_k = \begin{bmatrix} z_{k,0} & \dots & z_{k,b-2} & 0 \end{bmatrix} \div \vec{\alpha} \quad (6)$$

$$\lambda_k = \lim_{x \to -\infty} f(x) \quad (7)$$

where $\alpha_\perp$ and $\alpha_\top$ are respectively the slope of $f(x)$ when x approximates $-\infty$ and $-\infty$; and

$$\vec{y}_k = f(\vec{h}'_k) \quad (8)$$

$$\vec{\alpha'} = \text{diff}(\vec{y}_k) \div \text{diff}(\vec{h}_k) \quad (9)$$

$$\vec{y}''_k = -\vec{\alpha} \odot [h'_{k,0} \dots h'_{k,b2}] + [y_{k,1} \dots y_{k,b-1}] \quad (10)$$

$$\vec{z} = \text{diff}(\begin{bmatrix} y_{k,0} & y''_{k,0} & \dots & y''_{k,b-2} \end{bmatrix}) \quad (11)$$

where $\div$ is the entrywise division operator; $\text{diff}(\cdot)$ is the vector difference function.

4 Hardware design

We design an input overrider for the matrix-vector multiplier that optionally overwrites its inputs to introduce non-linearity.

Table 1: Assignment of control signals $c_{i,j}$

m	$x_i < h_{k,j}$	$x_i < h_{k,j+1}$	$c_{i,j}$	output
0	any	any	00	x_j
1	1	1	01	0
1	0	1	10	x_i
1	1	0	11	$\beta_{k,j}$

The overrider for the vector $\vec{x}$ includes $b \times b$ four-to-one multiplexers. Each multiplexer has an index $[i, j]$ where $i \in [0..b-1]$ and $j \in [0..b-1]$. The multiplexer indexed by $[i, j]$ implements the four cases for $\psi_{i,j}(\vec{x}, m, k)$ in Eq. 3. The assignment of the corresponding two-bit control signal $c_{i,j}$ follows Table 1. The overrider for the matrix W does not need multiplexers because the overridden value is independent of $\vec{x}$. As a result, one may pre-compute and store the overridden coefficient weight matrices rather than computing them on the fly.

For instance, Figure 2 shows a arithmetic block with that extends a matrix-vector multiplier with $b = 2$. The four scalar multipliers and the two adders constitute the original matrix-vector multiplier. The four multiplexers constitute the overrider. The two-bit control signals for multiplexers follow the assignment in Table 1.

The proposed arithmetic block has three properties. Each property address a drawback discussed in Section 2. First, all scalar multipliers work in both linear combination and non-linear function evaluation. The reduction in hardware idleness addresses the first drawback. Second, overriders do not consume multipliers. Therefore, the number of multipliers in the arithmetic blocks is independent of the number of non-linear functions that the system can support, addressing the second drawback. Third, the specification of a non-linear function is a set of numeric parameters. It is possible to define additional functions in memory without touching the arithmetic block, addressing the third drawback. Moreover, the proposed method preserves the internal connections between multipliers and adders inside the vector-multiplier, which makes it possible to preserve related optimisations.

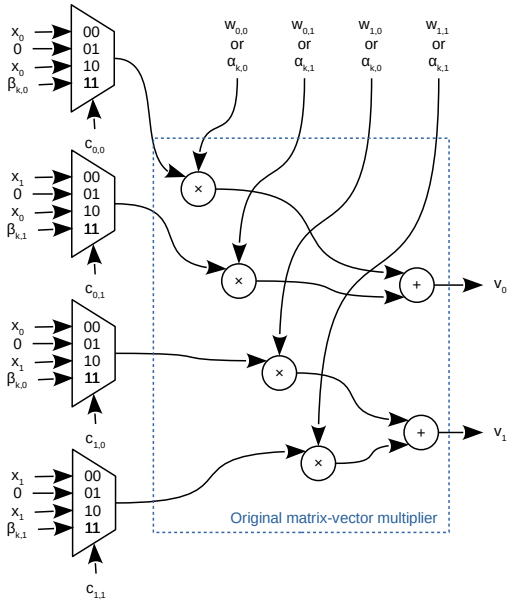


Figure 2: Matrix-vector multiplier with overrider

5 Evaluation

This section evaluates the proposed method by comparing potential resource usage between the reference design D with dedicated function evaluation blocks in Figure 1(a) and the proposed design P with overriders in Figure 1(b).

Both designs should fully pipeline the same set of linear and non-linear operations. In particular, each design should (i) calculate the linear combination between a b-dimensional vector $\vec{x}$ and a $b \times b$ matrix W (ii) apply one of the eight non-linear functions in Table 2 to all entries in $\vec{x}$ in parallel. The functions in the table are non-linear activation functions for

Table 2: Non-linear functions in P and D where s is the number of segments in P

Function	$<$	$+$	$\times$	$\div$	exp	log
Sigmoid	0	1	0	1	1	0
Log-sigmoid	1	1	0	1	1	1
Tanh	0	2	2	1	2	0
Tanhshrink	0	2	1	0	1	1
ELU	1	1	1	0	1	1
SELU	1	2	2	0	1	0
Softplus	1	1	0	0	1	1
Softsign	0	1	0	1	0	0
Total for D	4	11	7	4	8	4
Total for P	s	0	0	0	0	0

feed-forward neural networks in the PyTorch 1.0 machine learning framework. In addition to the function names, Table 2 also shows the number of arithmetic operations for comparison, addition, multiplication, division, exponentiation and logarithm.

Each design should include b^2 multipliers and $b(b-1)$ adders for linear combination. The reference design needs to include components to evaluate all functions for b inputs. Let R_A be the count of arithmetic block A. We have $R_< = 4b$, $R_+ = b^2 + 10b$, $R_\times = b^2 + 7b$, $R_\div = 4b$, $R_{\exp} = 8b$ and $R_{\log} = 4b$. In contrast, the proposed design reuses the matrix-vector multiplier and introduces non-linearity with comparators. The counts for the six components in the proposed design are $R_< = bs$, $R_+ = b(b-1)$, $R_\times = b^2$ and $R_\div = R_{\exp} = R_{\log} = 0$ where $s \in [1..b]$ is the number of segments for the piecewise linear approximation. Larger s brings better accuracy for function approximation. We use $s = 16$ in this study as this setting is enough to keep the relative error for all functions in Table 2 to be less than 1%.

Compared with the reference design with dedicated function evaluation blocks, the proposed design always consumes fewer components for addition, multiplication, division, exponentiation and logarithm regardless the value of b, although the proposed design consumes $12b$ more comparators. The comparators in the proposed design and the arithmetic components in the reference design grow linearly with b. However, a set of arithmetic components in the reference design D far higher logic complexity than a comparator in design P.

Assume that (i) we use a fixed-point representation with n bits throughout the design; (ii) an adder or a comparator has αn transistors; (iii) a multiplier or a divider has αn^2 transistors; (iv) an evaluation block for the exponential function or the logarithm functions has $\mu\alpha n^2$ transistors. The ratio between the number of transistors for design D and P is:

$$\frac{T_D}{T_P} = \frac{bn + b + 12\mu n + 11n + 14}{bn + b + s - 1} \tag{12}$$

As a number takes at least one bit, the ratio is always larger than 1. In other words, design D always takes more transistors than design P.

A practical setting with $b = s = 16$ and $\mu = 5$ is sufficient to accurately evaluate all non-linear functions in Table 2 in both designs. In this case, the ratio becomes

$$\frac{T_D}{T_P} = \frac{87n + 30}{16n + 31} \tag{13}$$

A larger n leads to higher precision. The ratio for $n = 16, 32, 64$ are respectively 4.95, 5.18 and 5.31, which means that design D consumes around 5 times more transistors than design P with 16-bit, 32-bit and 64-bit fixed-point numbers. The ratio grows slightly as the number of bits increases. The upper bound of $\frac{T_D}{T_P}$ is 5.4375.

6 Conclusion

This paper presents an arithmetic block extending matrix-vector multipliers to evaluate non-linear functions without changing their internal structures. The method reduces hardware idleness, the number multipliers and potential maintaining costs. From our analysis on eight non-linear functions, a reference design with dedicated function evaluation facilities consumes up to five times more transistors than the proposed design.

7 Acknowledgements

The support of the United Kingdom EPSRC (grant numbers EP/L016796/1, EP/N031768/1, EP/P010040/1 and EP/L00058X/1), Corerain, Maxeler, Intel and Xilinx is gratefully acknowledged.

References

[1] W Zhao et. al. An FPGA-based framework for training convolutional neural networks. In *ASAP*, 2016.

[2] S. Kestur, J. D. Davis, and E. S. Chung. Towards a universal FPGA matrix-vector multiplication architecture. In *FCCM*, 2012.

[3] Y. Li and A. Pedram. Caterpillar: Coarse grain reconfigurable architecture for accelerating the training of deep neural networks. In *ASAP*, 2017.