# Efficient Sparse Deep Neural Network Computation on GPU with TVM


**Lillian Wang, Avik Malladi, Yuede Ji**

Department of Computer Science and Engineering, University of North Texas

UNT | TAMS Texas Academy of Mathematics & Science


## Introduction

- Sparse Deep Neural Networks (SpDNNs) provide unique scalability difficulties in which optimizations and advancements can be made [1].
- Apache TVM [2] is a machine learning compiler framework for CPUs and GPUs which has shown promising improvements in the optimizations of networks [3].
- To evaluate its effectiveness, this work presents GPU optimizations using Apache TVM for SpDNNs.

## Problem Definition

**Input:**

- weight matrices $W_l$
- MNIST sparse input data $Y_0$
- bias vector $B_l$
- truth categories

**Inference:** For each layer, we compute the next using:

$$Y_{l+1} = ReLU(Y_l \times W_l + B_l)$$

**Results:**

- time equation for each pass,
- check for accuracy with truth categories
- compute rate for inference using:

*(number of inputs)* × *(number of connections)* ÷ *(time)*

## Motivation

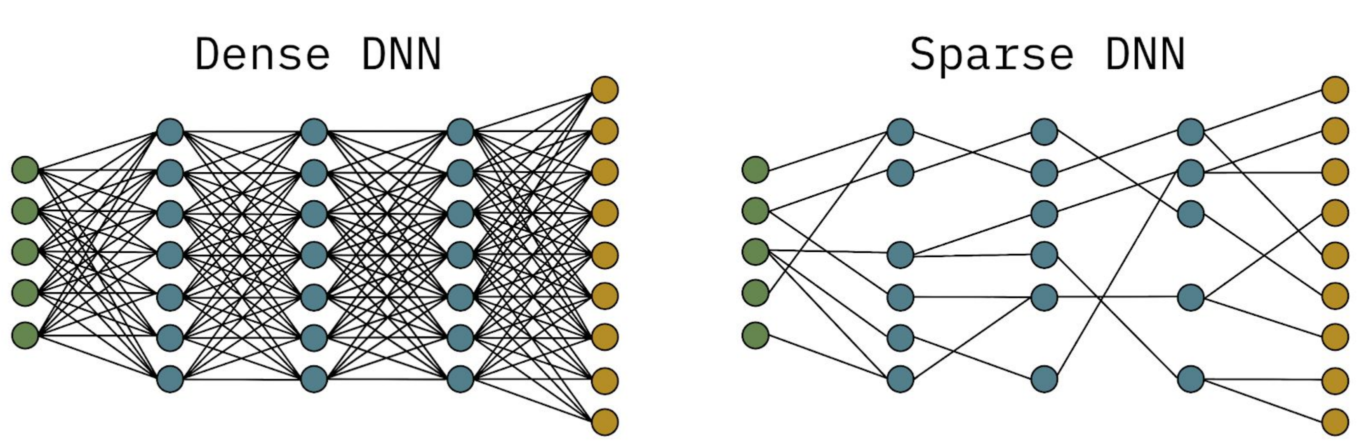


**Figure 1.** This figure describes the difference between sparse and dense DNNs

**Benefits of using SpDNN:**

- pruning DNNs increases sparsity and improves generalization results
- High sparsity (more zeros) results in high potential for more efficient storage and computation.
  - compatible with devices with low processing power

**Challenges with SpDNN's:**

- Most current developments do not accommodate high sparsity and are thus inefficient
- Dense DNNs more easily use information like historical statistics or previous predictions to define features/connections than sparse DNNs.
- Most current SpDNNs are built using C++ and CUDA rather than python which TVM uses

## Methodology

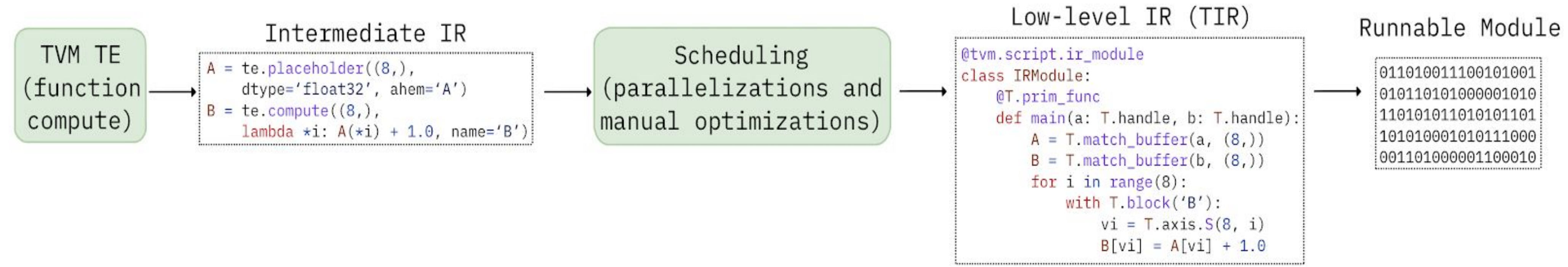


**Figure 2.** This figure provides an overview of our TVM implementation.

Our implementation has two main parts:

**TVM Tensor Expression.** TVM's TE, a namespace that TVM's optimizations build off of, is used to write the inference function through te.compute and related functions

- Code may be written with PyTorch, TensorFlow, etc. before converting to an IR Module

**Scheduling.** We use TE and TVM Relay, a namespace containing the Intermediate Representation (IR) definition and compiler, to partition each layer into equal sizes and use TVM's built in scheduling functions to parallelize the partitions in CPU and GPU.

- **Low-level IR.** We convert the model into a low-level IRModule using Relay. The second code block in figure 2 is the generated IRModule script, which is more
  - Further optimizations on the module may be made during this step
- **Runnable Module.** This is the final compiled module. The input parameters for the module are the input tensors and the output tensors.

## Experimental Setup

| Component Type | Component |
|---|---|
| Server | Runs Rocky Linux 8.6 with hyper-threading enabled |
| CPU | Intel Xeon Silver 4309Y CPU containing 8 cores |
| GPU | A40 NVIDIA GPU with 48GB running CUDA Toolkit 11.7 |

## Results

| | Layers | Edges | CPU Time (s) | CPU Edges/Second | GPU Time (s) | GPU Edges/Second |
|---|---|---|---|---|---|---|
| TVM | 120 | 3,932,160 | 46.19 | 5.11 e+09 | 42.56 | 5.55 e+09 |
| MATLAB | | | 72.42 | 3.26 e+09 | | |
| Sparse library | | | 43.09 | 5.48 e+09 | 68.99 | 3.42 e+09 |
| TVM | 480 | 15,728,640 | 184.07 | 5.13 e+09 | 172.91 | 5.46 e+09 |
| MATLAB | | | 289.61 | 3.26 e+09 | | |
| Sparse library | | | 163.13 | 5.79 e+09 | 275.97 | 3.42 e+09 |
| TVM | 1920 | 62,914,560 | 733.46 | 5.15 e+09 | 717.60 | 5.26 e+09 |
| MATLAB | | | 1220.09 | 3.09 e+09 | | |
| Sparse library | | | 658.55 | 5.73 e+09 | 1105.53 | 3.41 e+09 |

**Table 1.** Single CPU and GPU performance for the deep neural network computation with 1024 neurons and varying layer size.

## Key Findings

- For CPU, our TVM implementation achieves a 1.6x speed up over the benchmark code in matlab.
- The sparse library outperforms our TVM implementation in CPU by 1.1x. However, our algorithm for TVM is very basic compared the algorithm provided by the sparse library, and our TVM implementation is still able to compensate for the deficiencies with our partitioning a scheduling optimizations.
- In GPU, we achieve a 1.6x speed up over our sparse library implementation, indicating TVM's potential for efficient architecture usage.
- GPU runtime is most efficient with 64 partitioned blocks

## Conclusions

- **Sparse deep neural networks provide unique challenges** and opportunities. We pursue this with Apache TVM, a machine learning compiler framework for various computer architectures.
- **TVM's scheduling and tuning optimizations improve upon** the baseline and shows promise compared to other sparse libraries.
- **Further research may be done** to apply TVM's autoTVM or AutoScheduler to tune the inference, explore TVM's accommodations for other Python machine learning libraries such as Pytorch and TensorFlow, or learn how to optimize multiplication algorithm for multiplication between two sparse matrices

## References


**[1]** Mauro Bisson and Massimiliano Fatica. 2019. A GPU implementation of the sparse deep neural network graph challenge. In 2019 IEEE High Performance Extreme Computing Conference (HPEC). IEEE, 1–8.

**[2]** Chen, Tianqi, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan et al. "{TVM}: An automated {End-to-End} optimizing compiler for deep learning." In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 578-594. 2018.

[3] Hu, Yuwei, Zihao Ye, Minjie Wang, Jiali Yu, Da Zheng, Mu Li, Zheng Zhang, Zhiru Zhang, and Yida Wang. "Featgraph: A flexible and efficient backend for graph neural network systems." In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1-13. IEEE, 2020.