# ASL Spring 2025 Course Project: Sparse Ternary Matrix Multiplication

Supervising TA: Shien Zhu (shien.zhu@inf.ethz.ch)

### Problem Definition:

Given a dense matrix X of size (M, K), a ternary sparse matrix W of size (K, N), and a dense bias vector b of size N, your task is to calculate **Y = XW + b** and **Y = PReLU(XW + b)**, where PReLU refers to the PyTorch Framework for the PReLU implementation [1].

**The ternary sparse matrix W only contains {-1, 0, +1}** and can be preprocessed to a specific data format. **You are allowed, and encouraged, to design and modify the ternary sparse matrix format and the sparse matrix multiplication algorithm**. The lecture on sparse linear algebra gives you a good starting point to explore different formats [2].

In the following, we give examples of the data format and sparse GEMM algorithm.

### Example Project Template:

One example project template is available at github.com/yiweifengya/ASL_SparseGEMM. You can implement the code on an X86 CPU or ARM CPU.

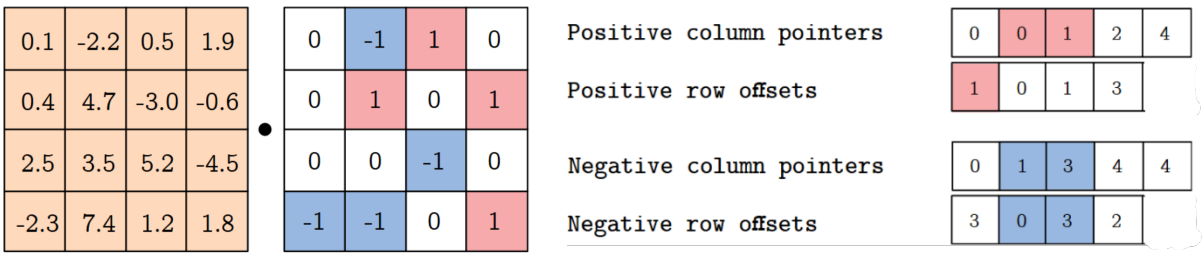
### Example Sparse Matrix Format:

```
class SparseFormat {
public:
        vector<int> col_start_pos; //start point of this column in row index
        vector<int> col_start_neg;
        vector<int> row_index_pos; //the row index of +1
        vector<int> row_index_neg; //the row index of -1
}
```

Sparse formats use specialized data structures to store only the nonzero values of a matrix. As an example, we use here a variant of the Compressed Sparse Column (CSC) format, called Ternary CSC (TCSC). TCSC compresses the data by storing the indices of positive and negative values separately. To do so we need

- The row_index array stores the indices of the rows of the nonzero values for each column,
- The col_start arrays indicates where each new column starts in the row_index array.

Then, we simply store the column offsets and row indices for 1 and -1 separately, as the following figure shows.



## Example Algorithm:

We can calculate **Y = XW + b** similar to general matrix multiplication by using three for loops on M, N, K. The only difference is that we selectively add or sub X[m,k] according to the index of +1 and -1 in W, resulting in two for loops on K.

```
void sparseGEMM(float* X, SparseFormat W, float* b, float* Y, int M, int N, int K) {
    for (int m = 0; m < M; m++) {
        for (int n = 0; n < N; n++) {
            float y = 0;
            // Add all X(m,k) where W(k,n) is +1
            for (int k = W.col_start_pos[n]; k < W.col_start_pos[n + 1]; k++) {
                y += X[m * K + W.row_index_pos[k]];
            }
            // Sub all X(m,k) where W(k,n) is -1
            for (int k = W.col_start_neg[n]; k < W.col_start_neg[n + 1]; k++) {
                y -= X[m * K + W.row_index_neg[k]];
            }
            Y[m*N+n] = y + b[n];
        }
    }
}
```

The PReLU can be implemented as a standalone function or fused into the sparseGEMM function to reduce the memory loading overhead, depending on your benchmarking results.

## Benchmarking Configurations:

Here is a reference list of matrix sizes for benchmarking. For each M[i], K[j] and N[j] are always selected in pairs. Thus, there are 8x8=64 matrix sizes at max. Note that some matrix shapes may be too large to execute depending on your computer hardware. You can cut down these cases and add some smaller configurations if needed.

int M[] = { 1, 16, 64, 256, 1000, 4000, 16000, 64000 };

int K[] = { 512, 1024, 2048, 4096, 2048, 4096, 8192, 16384 };

int N[] = { 2048, 4096, 8192, 16384, 512, 1024, 2048, 4096 };

The sparsity level should be tested in four cases: 50%, 75%, 87.5%, and 93.75%, meaning 1/2, 1/4, 1/8, and 1/16 of the W matrix are non-zero values.

## References


[1] https://pytorch.org/docs/stable/generated/torch.nn.PReLU.html

[2] https://acl.inf.ethz.ch/teaching/fastcode/2024/slides/11-sparse.pdf