MagmaDNN
Integration and Applications

Stephen Qiu (University of Tennessee)
Julian Halloy (University of Tennessee)

Introduction

- MAGMA is a collection of Linear Algebra (LA) routines for heterogeneous architectures which take advantage of GPUs as well as multi-core CPUs for faster and more efficient computation

Introduction

- MagmaDNN is a deep learning framework that utilizes the high performance calculations of MAGMA for common Neural Network calculations.
- Currently they are two separate packages. MagmaDNN is dependent on MAGMA to run, but MAGMA can be run independently.

Research Goals

- Prove MagmaDNN works correctly given the MLP and CNN examples
- Optimize MagmaDNN
- Integrate MagmaDNN into MAGMA without losing the speed and functionality that each had before
- Make MagmaDNN readily accessible to researchers utilizing MAGMA by creating a DNN submodule similar to MAGMA-sparse

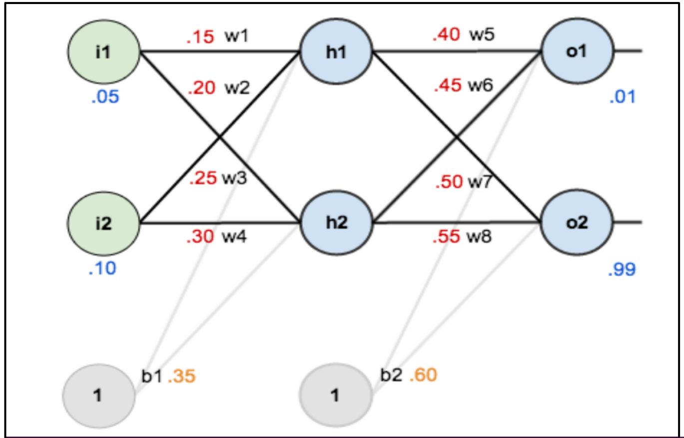
MLP

```
using T = float;
```

```
memory_t mem;

auto i = Tensor<T> ({1,2,1}, {NONE, {}}, mem);
i.set({0,0,0}, 0.05f);
i.set({0,1,0}, 0.10f);

auto target = Tensor<T> ({1,2,1}, {CONSTANT, {0.01f}}, mem);
target.set({0,0,0}, 0.01f);
target.set({0,1,0}, 0.99f);
```

```
// Initialize our model parameters
model::nn_params_t params;
params.batch_size = 1;
params.n_epochs = 1;
params.learning_rate = 0.25;
params.momentum = 0;
```

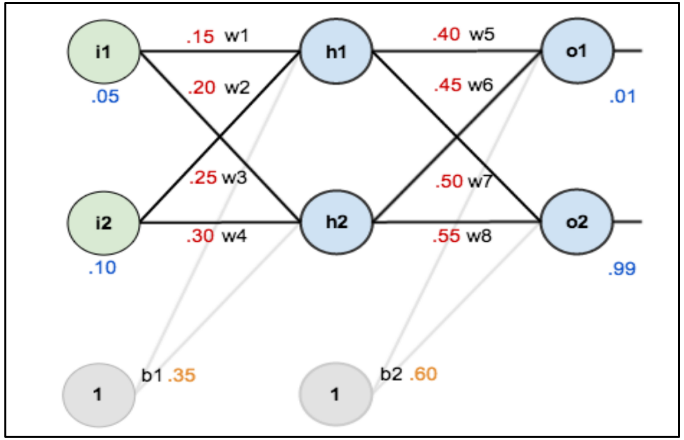
```
// n_features) This will serve as the input to our network.
auto x_batch = op::var<T>(
                "x_batch",
                {params.batch_size, 1, 1, 2},
                {NONE, {}}, training_memory_type);
```




```
// Initialize the layers in our network
auto input = layer::input(x_batch);

auto flatten = layer::flatten(input->out());
auto fc1 = layer::fullyconnected(flatten->out(), 2, true);

fc1->get_weights()[0]->eval()->set({0,0}, 0.15f);
fc1->get_weights()[0]->eval()->set({1,0}, 0.2f);
fc1->get_weights()[0]->eval()->set({0,1}, 0.25f);
fc1->get_weights()[0]->eval()->set({1,1}, 0.3f);
fc1->get_weights()[1]->eval()->set(0, 0.35f);

auto act1 = layer::activation(fc1->out(), layer::SIGMOID);
auto fc2 = layer::fullyconnected(act1->out(), 2, true);

fc2->get_weights()[0]->eval()->set({0,0}, 0.40);
fc2->get_weights()[0]->eval()->set({1,0}, 0.45);
fc2->get_weights()[0]->eval()->set({0,1}, 0.50);
fc2->get_weights()[0]->eval()->set({1,1}, 0.55);
fc2->get_weights()[1]->eval()->set(0, 0.60);

auto act2 = layer::activation(fc2->out(), layer::SIGMOID);

auto output = layer::output(act2->out());
```

MLP

```
// Wrap each layer in a vector of layers to pass to the model
std::vector<layer::Layer<T> *> layers =
{input,
        flatten,

        fc1, act1,

        fc2, act2,
        output};
```

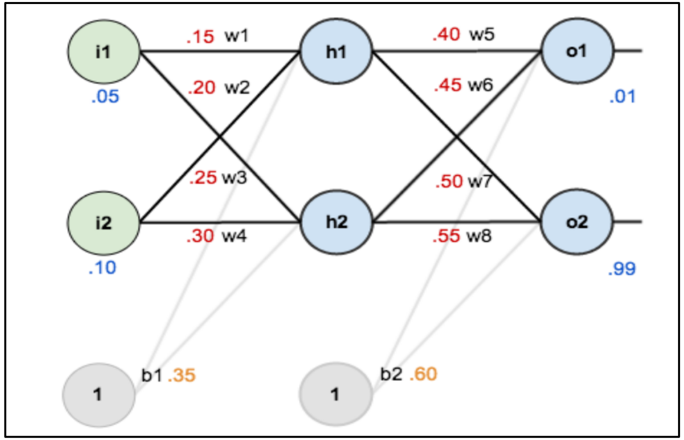



```
model::NeuralNetwork<T> model(layers, optimizer::CROSS_ENTROPY, optimizer::SGD, params);

// metric_t records the model metrics such as accuracy, loss, and
// training time
model::metric_t metrics;



model.fit(&i, &target, metrics, true);
model.summary();
```

MLP Output

```
Name                         Output Shape        # Params
=========================================================
InputLayer                   (1, 1, 2, 1)               0
FlattenLayer                       (1, 2)               0
FullyConnected                     (1, 2)               6
FullyConnected                     (1, 2)               6
OutputLayer                        (1, 2)               0
```

```
InputLayer[0] output = 0.05
InputLayer[1] output = 0.1
FlattenLayer[0] output = 0.05
FlattenLayer[1] output = 0.1
FullyConnected[0] output = 0.372791
FullyConnected[1] output = 0.387787
Activation[0] output = 0.592133
Activation[1] output = 0.59575
FullyConnected[0] output = 1.05536
FullyConnected[1] output = 1.20536
Activation[0] output = 0.741804
Activation[1] output = 0.769477
OutputLayer[0] output = 0.741804
OutputLayer[1] output = 0.769477
w1 = 0.149890
w2 = 0.199781
w3 = 0.249876
w4 = 0.299751
w5 = 0.379458
w6 = 0.429333
w7 = 0.505651
w8 = 0.555685
b1 = 0.345319
b2 = 0.574900
loss = 0.298371
```



w1+: 0.1497807161327648

w2+: 0.1997807161327648

w3+: 0.2497511436323716

w4+:

Print Gradients

neuralnetwork.cpp -> optimizer.cpp -> gradtable.cpp

```
template <typename T>
void GradTable<T>::print() {
   printf("GradTable\n");
   printf("%s\n", std::string(30, '=').c_str());
   printf("Number of tensors = %i\n", _table.size());
   int itr = 0;
   for (auto &entry : this->_table) {
             itr++;
             printf("%s Tensor %i\n%i value(s): ", entry.first->get_name().c_str(), itr, entry.second->get_size());
             for (unsigned int i = 0; i < entry.second->get_size(); i++){
                          printf("%.5g%s", entry.second->get(i), (i == entry.second->get_size()-1) ? "\n" : ", ");
       }
   }
   printf("\n");
}
```

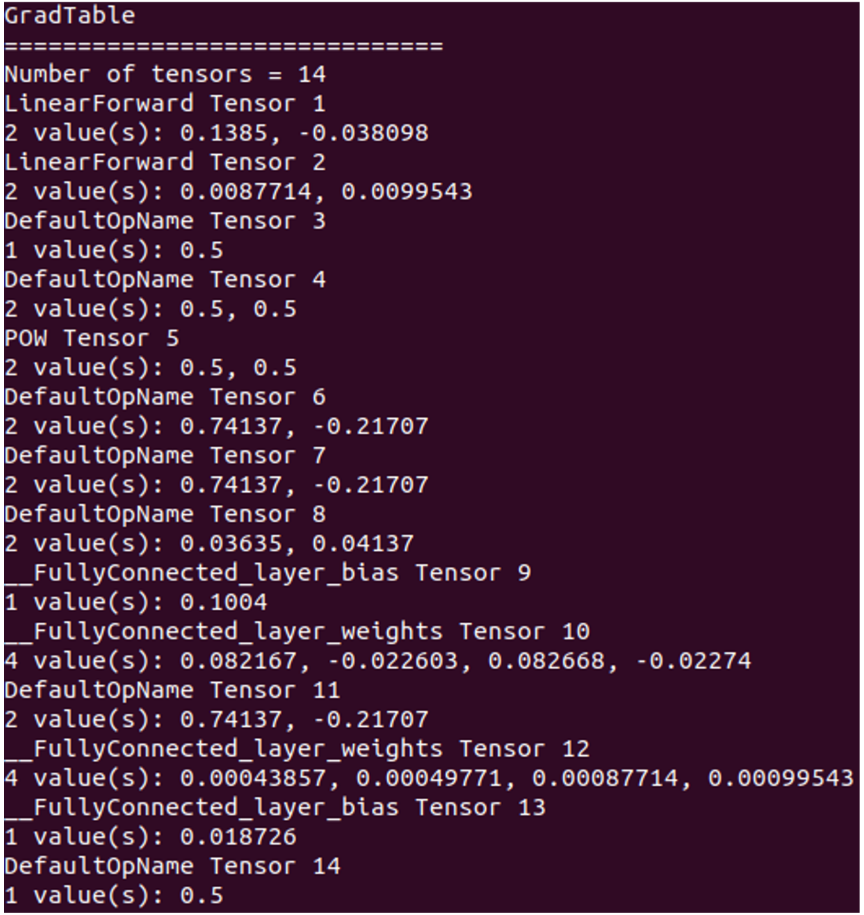
Print Gradients

```
void GradientDescent<T>::minimize(op::Operation<T> *obj_func, const std::vector<op::Operation<T> *> &wrt, bool print) {
   typename std::vector<op::Operation<T> *>::const_iterator vit;


   this->_obj_func = obj_func;


   /* evaluate if need be */
   this->_obj_func->eval(false);


   /* build the gradients */
   this->table.clear();
   op::get_grad_table(wrt, this->_obj_func, this->table);


   if (print){
       table.print();
   }
   /* now update each one */
   for (vit = wrt.begin(); vit != wrt.end(); vit++) {
       this->update((*vit), table.get(*vit));
   }
}
```

Print Gradients

```
virtual magmadnn_error_t fit(Tensor<T> *x,
Tensor<T> *y, metric_t &metric_out, bool
verbose = false, bool print = false);
```

grad_o1: 0.7413650695475076
grad_o2: -0.21707153468357898
grad_d1: 0.13849856162945076
grad_d2: -0.03809823651803844
grad_w5: 0.08216704056448701
grad_w6: 0.08266762784778263
grad_w7: -0.022602540478279 04
grad_w8: -0.02274024221678477
grad_h1: 0.036350306392761086
grad_d11: 0.00877135468940779
grad_w1: 0.000438567734470 3895
grad_w2: 0.0004385677344703895
grad_h2: 0.04137032264833171
grad_d22: 0.009954254705134271
grad_w3: 0.0004977127352567136
grad_w4: 0.0009954254705134271



```
GradTable
==============================
Number of tensors = 14
LinearForward Tensor 1
2 value(s): 0.1385, -0.038098
LinearForward Tensor 2
2 value(s): 0.0087714, 0.0099543
DefaultOpName Tensor 3
1 value(s): 0.5
DefaultOpName Tensor 4
2 value(s): 0.5, 0.5
POW Tensor 5
2 value(s): 0.5, 0.5
DefaultOpName Tensor 6
2 value(s): 0.74137, -0.21707
DefaultOpName Tensor 7
2 value(s): 0.74137, -0.21707
DefaultOpName Tensor 8
2 value(s): 0.03635, 0.04137
__FullyConnected_layer_bias Tensor 9
1 value(s): 0.1004
__FullyConnected_layer_weights Tensor 10
4 value(s): 0.082167, -0.022603, 0.082668, -0.02274
DefaultOpName Tensor 11
2 value(s): 0.74137, -0.21707
__FullyConnected_layer_weights Tensor 12
4 value(s): 0.00043857, 0.00049771, 0.00087714, 0.00099543
__FullyConnected_layer_bias Tensor 13
1 value(s): 0.018726
DefaultOpName Tensor 14
1 value(s): 0.5
```

Mean Squared Error

Sums error over the batch and divides by the batch size

Does not sum both errors or divide by 2 for our MLP example

Solution: add reducesum to MSE for the other dimension

1	2
3	4
5	6
7	8

3
7
11
15

Process of Integration

- Cloned the MAGMA bitbucket and created our own private github repository

Integration into MAGMA - File Organization

- magma/
  - docs/
  - example/
  - fortran/
  - include/
  - interface_cuda/
  - magmablas/
  - magmadnn/
  - src/
  - etc.

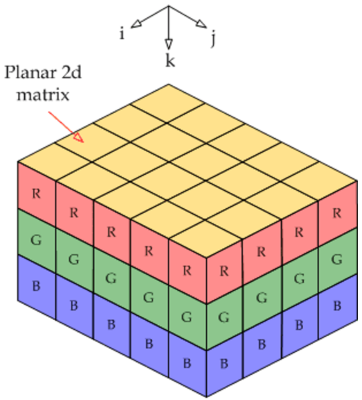
- /usr/local/magma/lib
  - libmagma.a
  - libmamga.so
  - libmagmadnn.a
  - libmagmadnn.so
  - pkgconfig/

Integration into MAGMA

- Since most of MAGMA is written primarily in C and MagmaDNN is mostly written in C++, the code is generally compatible with each other, allowing us to easily merge the two together.
- Issues:
    - MagmaDNN has its own tensor class to store data, MAGMA is only matrices
    - Row major vs Column major

Data Storage

- MAGMA stores data only as matrices
- MagmaDNN uses its own tensor class to store the data
    - it is very common for the data neural networks interact with to be 3 or more dimensions

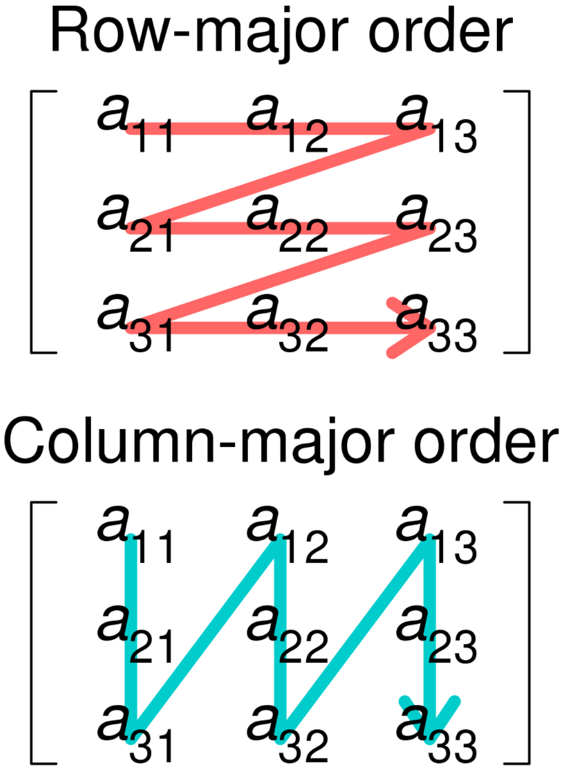


Graphical presentation of RGB 3d matrix

Ex.
RGB data
Batched data
2d Conv with multiple filters

Row vs Column major

- Affects how the matrix is stored in memory
- Affects how the code is supposed to access the data through incrementing.



Solutions

Choose to create a new tensor class within MAGMA

or

Create an interface between MAGMA and MagmaDNN

Interface

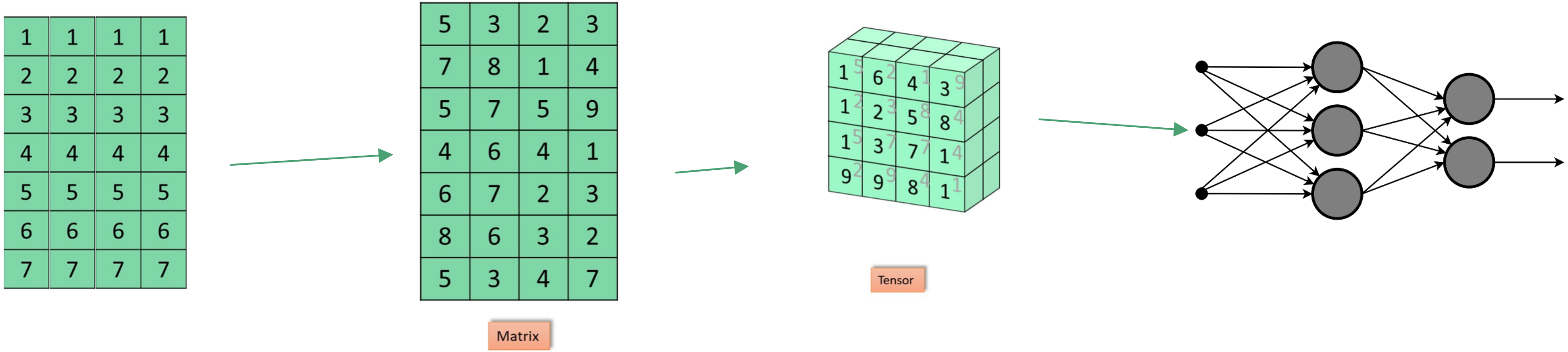
- Matrix to tensor interface
- Allows the reading of a column major matrix and storing it as a row major tensor with user defined parameters
- Takes the matrix address and dimension of the wanted tensor as inputs
- Compatible for 1-4 dimension tensors

```cpp
template <typename T>
Tensor<T>* matrix_to_tensor(T* matrix_addr, unsigned int num_dims, memory_t mem, std::vector<unsigned int> dims) {
    assert(num_dims == dims.size());
    Tensor<T>* new_tensor;
    new_tensor = new Tensor<T>(dims, {CONSTANT, {1}}, mem);

    unsigned int M, N, K, Q;

    switch (num_dims) {
        case 1:
            M = dims.at(0);
            for (unsigned int i = 0; i < M; i++) {
                new_tensor->set(std::vector<unsigned int>{i}, *(matrix_addr + i));
            }
            break;
        case 2:  //(MxN)

            M = dims.at(0);
            N = dims.at(1);
            for (unsigned int i = 0; i < M; i++) {
                for (unsigned int j = 0; j < N; j++) {
                    new_tensor->set(std::vector<unsigned int>{i, j}, *(matrix_addr + i + M * j));
                    printf("%d, ", i + M * j);
                }
            }
            break;
        case 3:  //(QxMxN)
            Q = dims.at(0);
            M = dims.at(1);
            N = dims.at(2);
            for (unsigned int k = 0; k < Q; k++) {
                for (unsigned int i = 0; i < M; i++) {
                    for (unsigned int j = 0; j < N; j++) {
                        new_tensor->set(std::vector<unsigned int>{k, i, j}, *(matrix_addr + i + M * j + M * N * k));
                        printf("%d, ", i + M * j + M * N * k);
                    }
                }
            }
            break;
        case 4:  //(QxKxMxN)
            Q = dims.at(0);
            K = dims.at(1);
            M = dims.at(2);
```

Interface

- The matrix to tensor interface will allow data from MAGMA to be easily used with MagmaDNN to implement some neural network algorithms

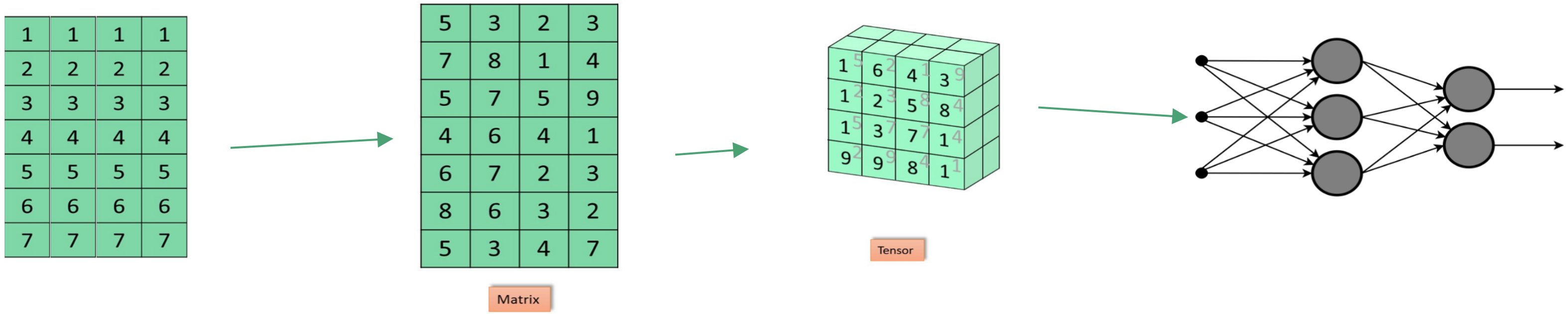
Input ---> MAGMA ---> manipulated data ---> M_to_T --> Tensor ---> MAGMADNN --> NN --> training or inference



https://medium.com/analytics-vidhya/mathematics-and-statistics-behind-machine-learning-part-1-eede0e152d57

Test Interface

- Combination of MAGMA code and Magmadnn code.
    - Reads in the MNIST data and stores into a magma matrix
    - Read the image data from a matrix into a Magmadnn tensor
    - Train a model/Load in a pretrained model and pass the tensor through for inference
    - Return the predicted value



Interface Example

```
(base) sqiu4@lapenna1:~/Downloads/magma/example/dnn_func$ ./mnist
Preparing to read 60000 images with size 28 x 28 ...
finished reading images.
Finished Importing Model
Image index to predict: 28

      80 189 254 255 254 254 254 174 101  31  50  12
  80 242 253 253 253 253 253 253 253 253 216 226 206 200 200  58
 101 253 253 253 253 253 253 253 253 253 253 253 253 253 253 227  53
 251 253 253 253 253 253 253 253 253 253 253 253 253 253 253 253 249 181  17
 122 214 214 158  61  61 113 214 214 250 253 253 253 253 253 253 253 253  45
                                     105 115 115 237 253 253 253 253 253 129
                                                  13  24 168 241 253 253 199
                                                       2 102 243 253 253  87
                                                      16 253 253 253 197  22
                                                  22 182 253 253 251 101
                                          3  99 198 253 253 247 129
                                         99 253 253 253 253 191
                                 117 224 244 253 253 239  30  23
                          58 169 213 253 253 253 197  79
                          86 253 253 253 242 137  16
                         216 253 253 253 141  62   8
                       5 239 253 253 253 253 253 172 162 162 162  64   8   7
                      80 247 253 253 253 253 253 253 253 253 253 253 253 199  66
                          95 199 227 253 253 253 253 253 253 253 220 230 201 235
                                  52  99  99 174 253 253 253 122  39  57  22  99

Predicted number is 2
```

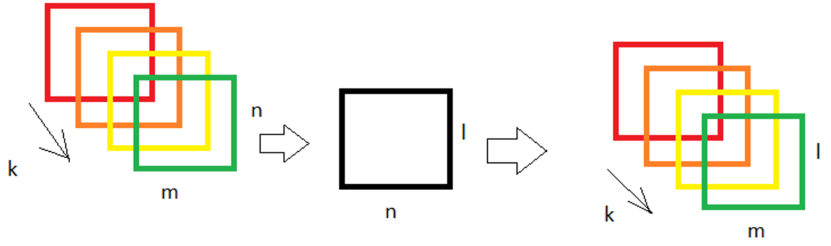
Interface Results

- The accuracy of the code depends on how well the trained/loaded model is.

- Test code shows that the function written can be used interchangeably between MAGMA and MagmaDNN.

Tensor operations

- Created a tensor matrix multiplication function to allow for higher order mathematical calculations.
- [k, m, n] x [n, l] -> [k, m, l]
- Current confined to 3D tensor x 2D matrix with one similar axis to contract upon
- Speed is comparable to the np.einsum() with the tensor matmul function reaching an average speed of 27 µs vs einsum’s speed of 25 µs on a [4, 50, 50] x [50, 50] tensor multiplication



Applications

- Neural network calculations
- Physics and engineering
- Basis for other similar tensor operations

Future Works

Some simple universal DNN functions such as MLP and CNN

Python interface / GUI for MagmaDNN to allow for easier use to those with a smaller programming background

Edge device implementation - utilizing the speed advantages of MagmaDNN on a device such as the Jetson Nano

Thanks