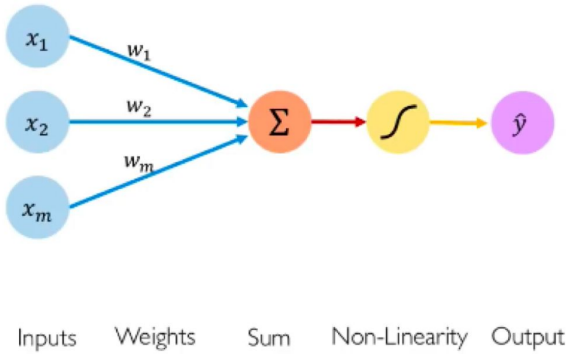
Simple Fully Connected Neural Network Using CUDA

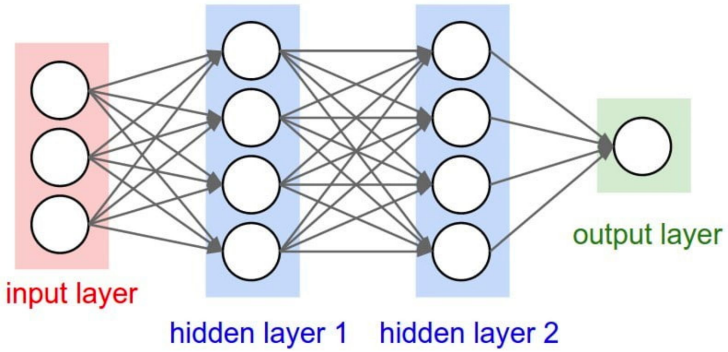
Zhouyuan Yuan

Model Architecture

Each Layer has

1. A weight matrix that linearly projects vectors from input dimension to output dimension.
2. An activation function, usually tanh or ReLU, that introduces non-linearity to the network.





Mathematics (Forward Pass)

Let X be the input with shape NxI, where N is the number of images, and I is the image dimension.

Let W1 be first layer's weight matrix with shape IxJ.

W2 be the second layer's weight matrix with shape JxK, where K is the number of classes.

Then, the forward pass of the two-layer network works as follow:

$A1 = X W1 + b; Z1 = ReLU(A1);$

$A2 = Z1 W2; Y = softmax(A2);$

$Prediction = argmax(Y)$

Mathematics (Backward Pass)

Let T be the one-hot-encoded ground truth labels with shape NxK.

Let α be the learning rate.

Then the backward pass of the two-layer network works as follow:

$$W2 = W2 - \alpha * A2^T (Y - T)$$

$$W1 = W1 - \alpha * Z1^T (A1>0)^T * (Y - T)\ W2^T$$

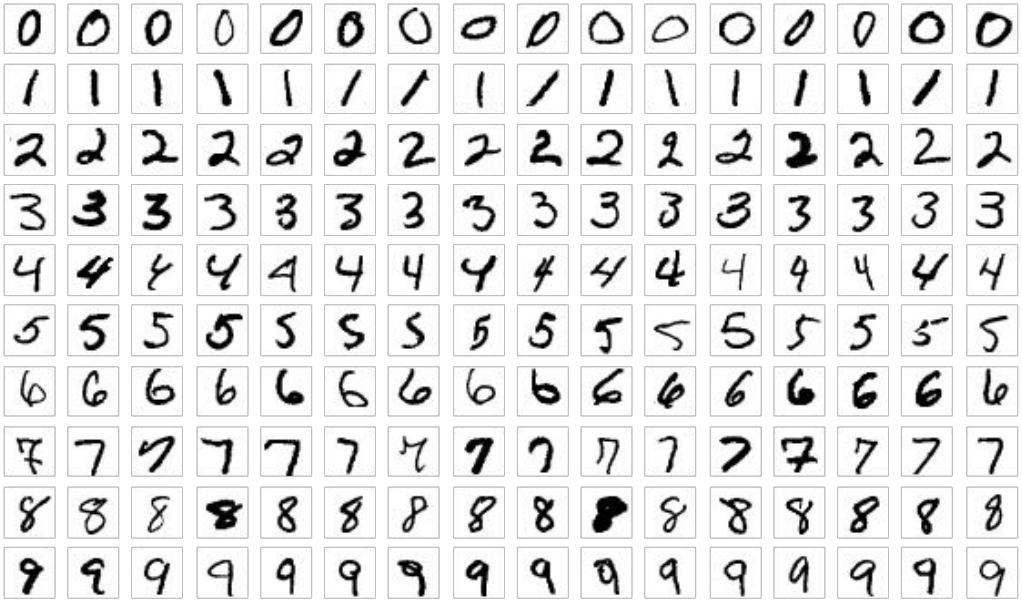
MNIST Dataset

Handwritten digits from 0-9.

Each Image has 28*28 pixels.

60000 Training Images

10000 Test Images



CUDA Kernels

1. /* Added = [ori 1], append column of ones */

```
__global__ void add_bias(float* added, float* ori, int nrows, int ncols)
```

2. /* out = AB */

```
__global__ void mat_mul( float* out, float* A, float* B, int m, int n, int k)
```

3. /* in = softmax(in) “/

```
__global__ void softmax( float* in, int nrows, int ncols )
```

CUDA Kernels

4. /* A = A - B */

__global__ void mat_substract(float* A, float* B, int total)

5. /* A = A - B */

__global__ void mat_substract_bool(float* A, bool* B, int total)

6. /* A = gamma * A */

__global__ void element_wise(float* A, float gamma, int total)

CUDA Kernels

7. /* out = A^T B */

__global__ void mat_mul_T(float* out, float* A, float* B, int m, int n, int k)

8. /* out = argmax(A, axis=1) == argmax(B, axis=1) */

__global__ void equal(bool* out, float* A, bool* B, int rows)

Main Challenge - Read MNIST in C++

Binary Files

Image: 4 integer bytes + unsigned chars for each pixel in images

Label: 2 integer bytes + unsigned chars for each image label

/*Modified on https://stackoverflow.com/questions/8286668/how-to-read-mnist-data-in-c, which only read images*/

void ReadMNIST(char* img_file, char* label_file, int NumberOfImages, int DataOfAnImage, float* arr, int* arr_lbl)

Main Challenge - Data types

For Fast Data Transfer from Host to Device, I tried to keep the data type minimal:

Let one-hot-encoded ground-truth labels be Bool (1 byte)

Let weight matrix and input Images be float (4 bytes)

However, during softmax, exp() will make the number too small and become nan if stored in float type, so cast to double (8 bytes) is needed during softmax operation.

Main Challenge - Debugging

- Coded the whole algorithm in Python first using cpu to make sure the algorithm works
- Compare every __global__ functions' outputs to CPU computed outputs
- Using VS Nsight to step into the __global__ functions
- Start with small subset of the dataset
- Use checkCudaErrors as much as possible.

Result (Single Layer)

```
int train_num = 60000;
int test_num = 10000;
int img_dim = 28 * 28;
int class_num = 10;
float lr = 0.00001;
int num_epochs = 1000;
```

```
Epoch: 0, Train Acc: 0.141167, Test Acc: 0.136300
Epoch: 100, Train Acc: 0.867200, Test Acc: 0.874900
Epoch: 200, Train Acc: 0.889450, Test Acc: 0.894300
Epoch: 300, Train Acc: 0.898967, Test Acc: 0.904300
Epoch: 400, Train Acc: 0.904433, Test Acc: 0.908700
Epoch: 500, Train Acc: 0.907917, Test Acc: 0.911700
Epoch: 600, Train Acc: 0.911167, Test Acc: 0.913500
Epoch: 700, Train Acc: 0.913250, Test Acc: 0.914900
Epoch: 800, Train Acc: 0.914633, Test Acc: 0.916100
Epoch: 900, Train Acc: 0.916200, Test Acc: 0.916200
Epoch: 999, Train Acc: 0.917483, Test Acc: 0.917200
Done ! Total Training time: 21.000000 sec.
```

Run time investigation on number of warps per block