THE OHIO STATE UNIVERSITY

Real-time Inference of Generative Models on TX1 at Traffic Intersections

Menna El-Shaer

el-shaer.1@osu.edu

Training GTX1080

Images

Batch Splitting

Epoch Loops

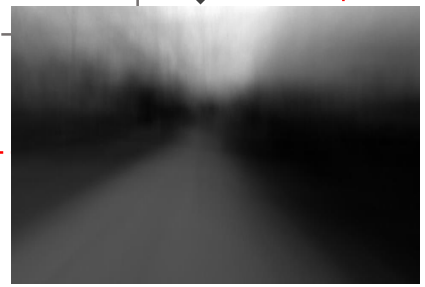
Inference TX1

Images

Iterative Loop

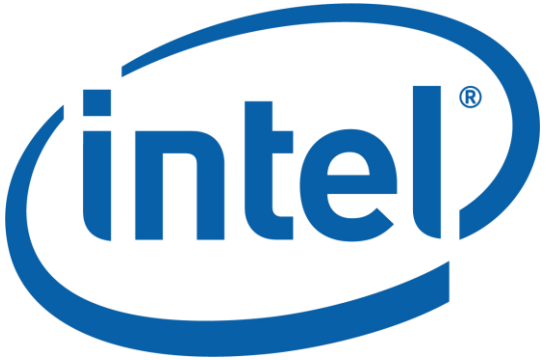
Gibbs Step

Optimizations



Data Collection

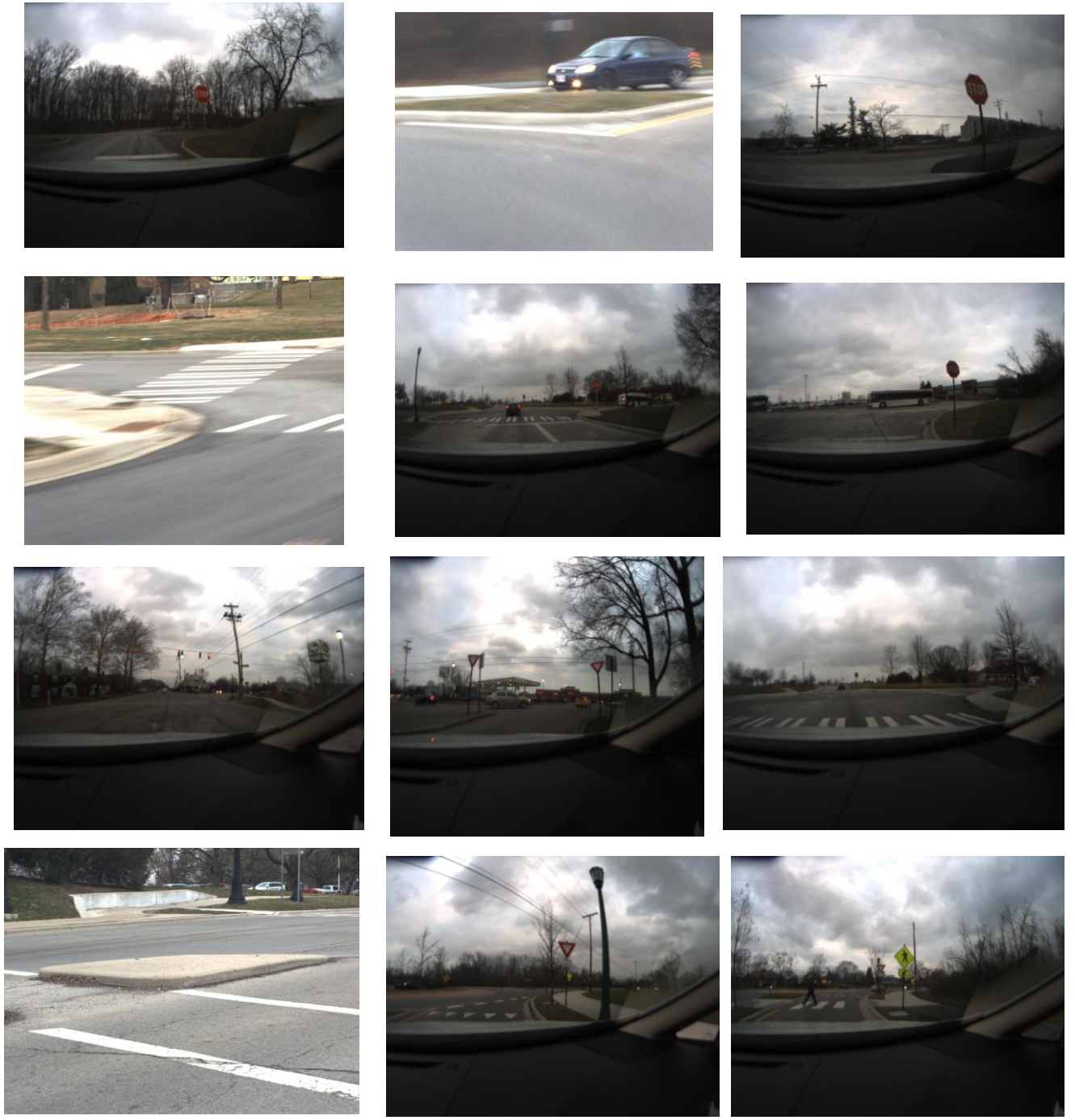
- Jetson TX1 as processing unit
- Two Point Grey GigE cameras:
  - FL3-GE-13S2C-CS
- Intel Network GigE PCIe Adapter 82576



Dataset

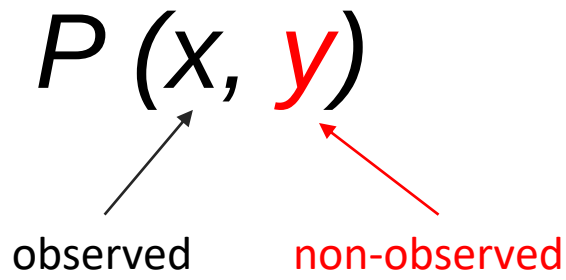
4883 total training images from both cameras
976 total test images from both cameras
5 types of intersections:

1. Four-way stop sign controlled.
2. Traffic light signal controlled.
3. Cross walks without stop signs.
4. Roundabout-type.
5. Three-way intersections.



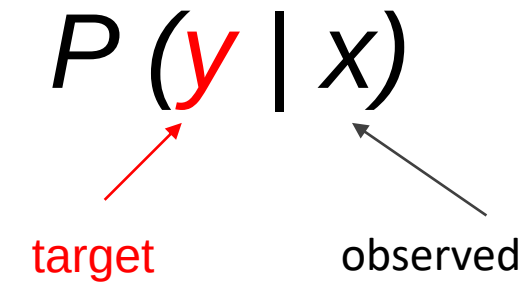
Generative Models

Examples: Naïve-Bayes, Autoencoders,
RBMs, Topic Models, Mixture Models, .. etc



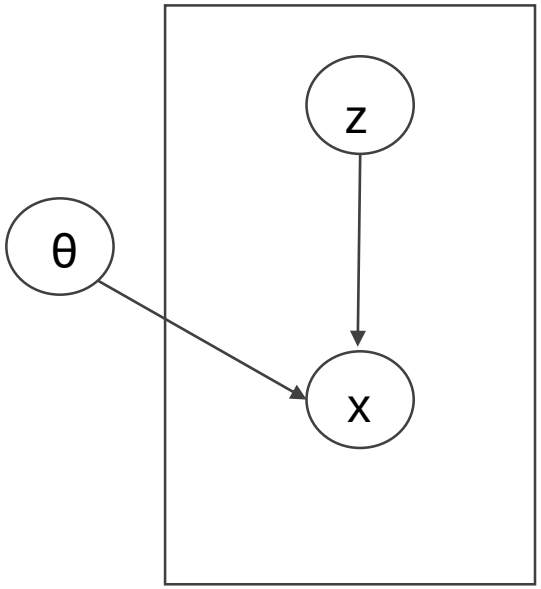
Discriminative Models

Examples: Classifiers, Logistic Regression,
Neural Networks, .. etc

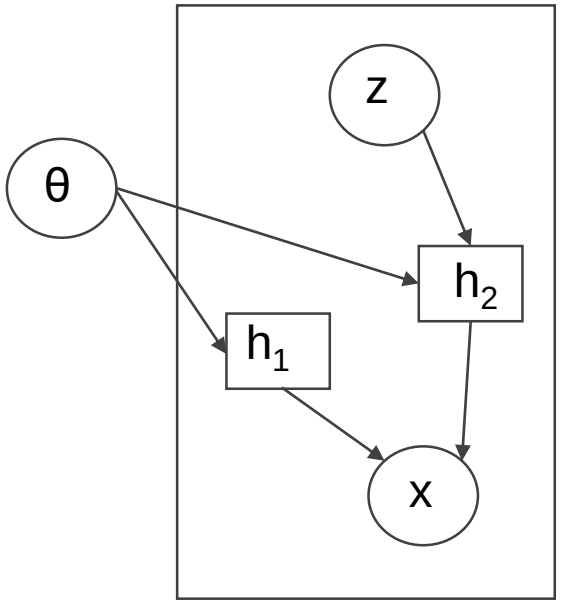


Generative Models

Model Observed Data

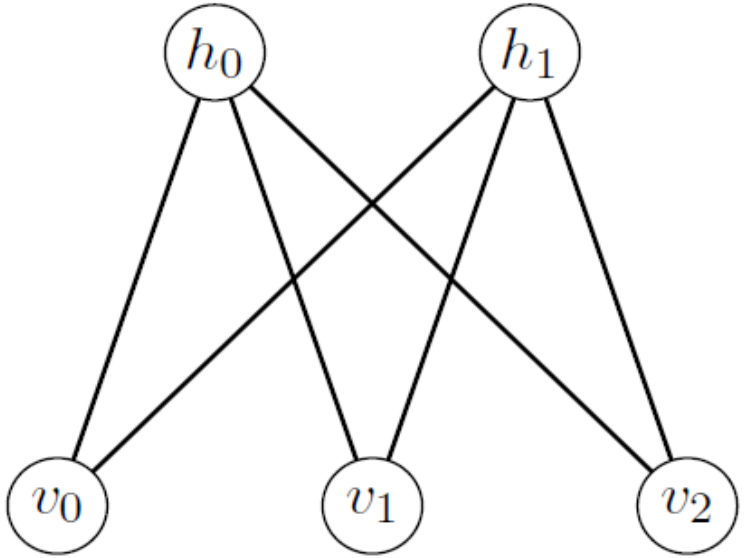


Latent Variable Models



Training

Sampling from model distribution → Alternating Gibbs Sampling: updating all hidden units in parallel followed by updating all visible units in parallel

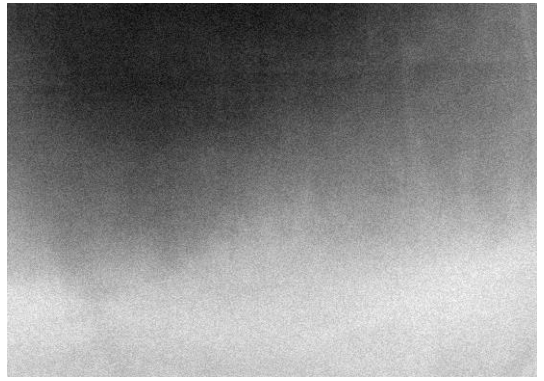


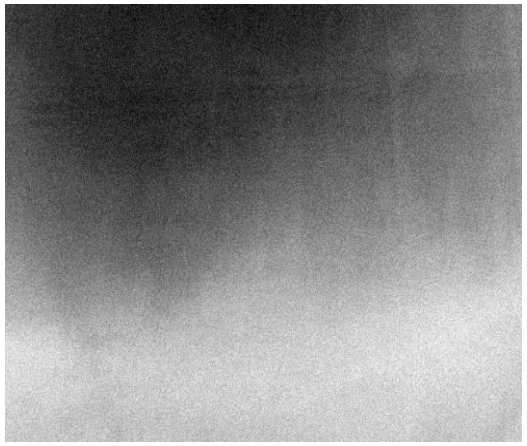
Sample hidden layer given input training vector in parallel using $p(h_j = 1 \mid v) = \sigma(c_j + \sum_i v_i w_{ij})$, then using $p(v_i = 1 \mid h) = \sigma(b_i + \sum_j h_j w_{ij})$, compute probability of each visible unit = 1 i.e. reconstruct the visible layer.

Learning is then done using this reconstruction instead

$$\Delta w_{ij} = learning\_rate(E(v_i, h_j)_{data} - E(v_i, h_j)_{reconstruction})$$

Training Weights (Filters)



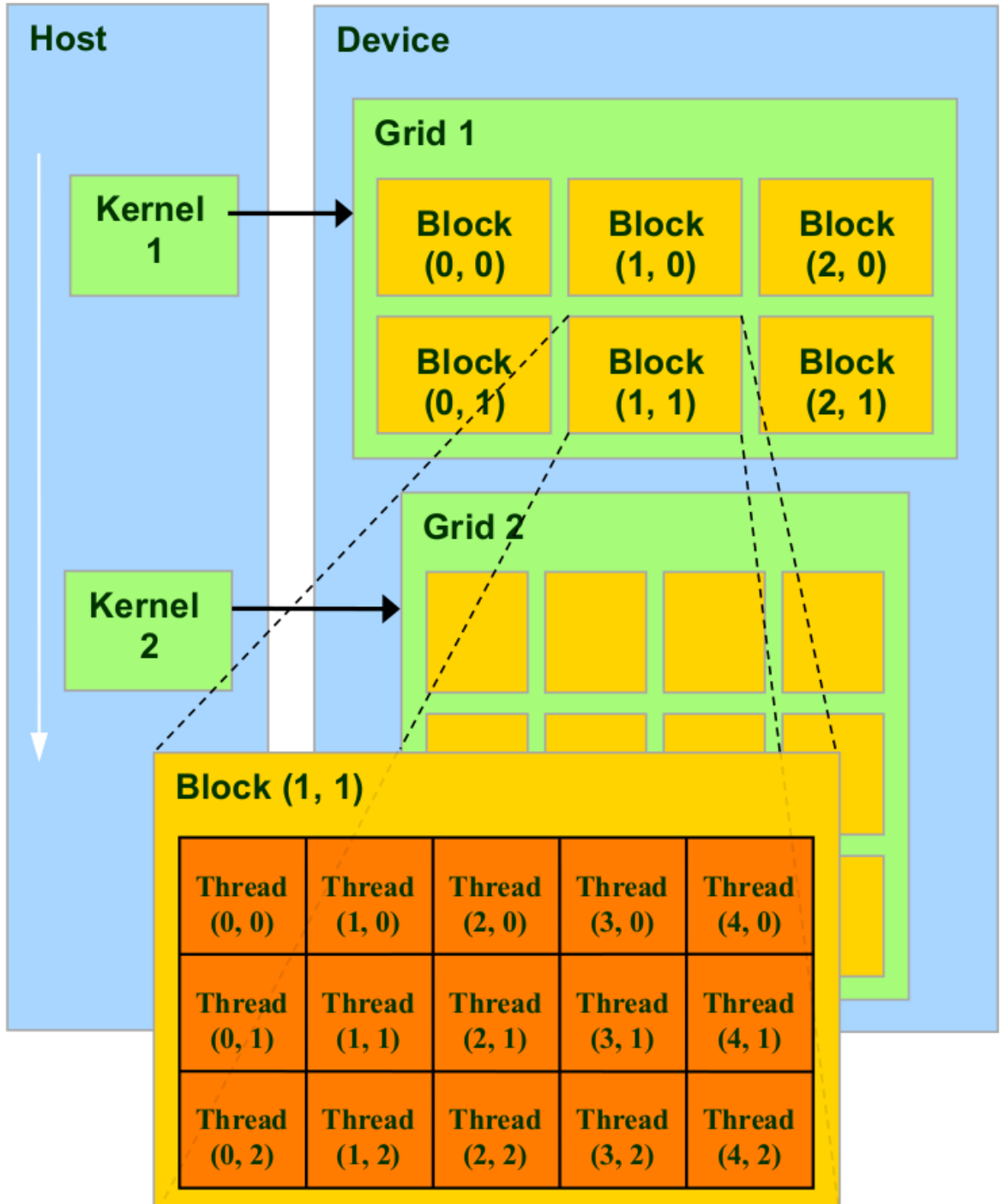


Hyper Parameters

Learning Rate
Momentum
Weight Cost (Penalty)
Sparsity Target
Initial Values of Weights and Biases
Number of Hidden Units
Batch size

G. Hinton, “A Practical Guide to Training Restricted Boltzmann Machines (1)”, August 2010

Review: CUDA Programming Model



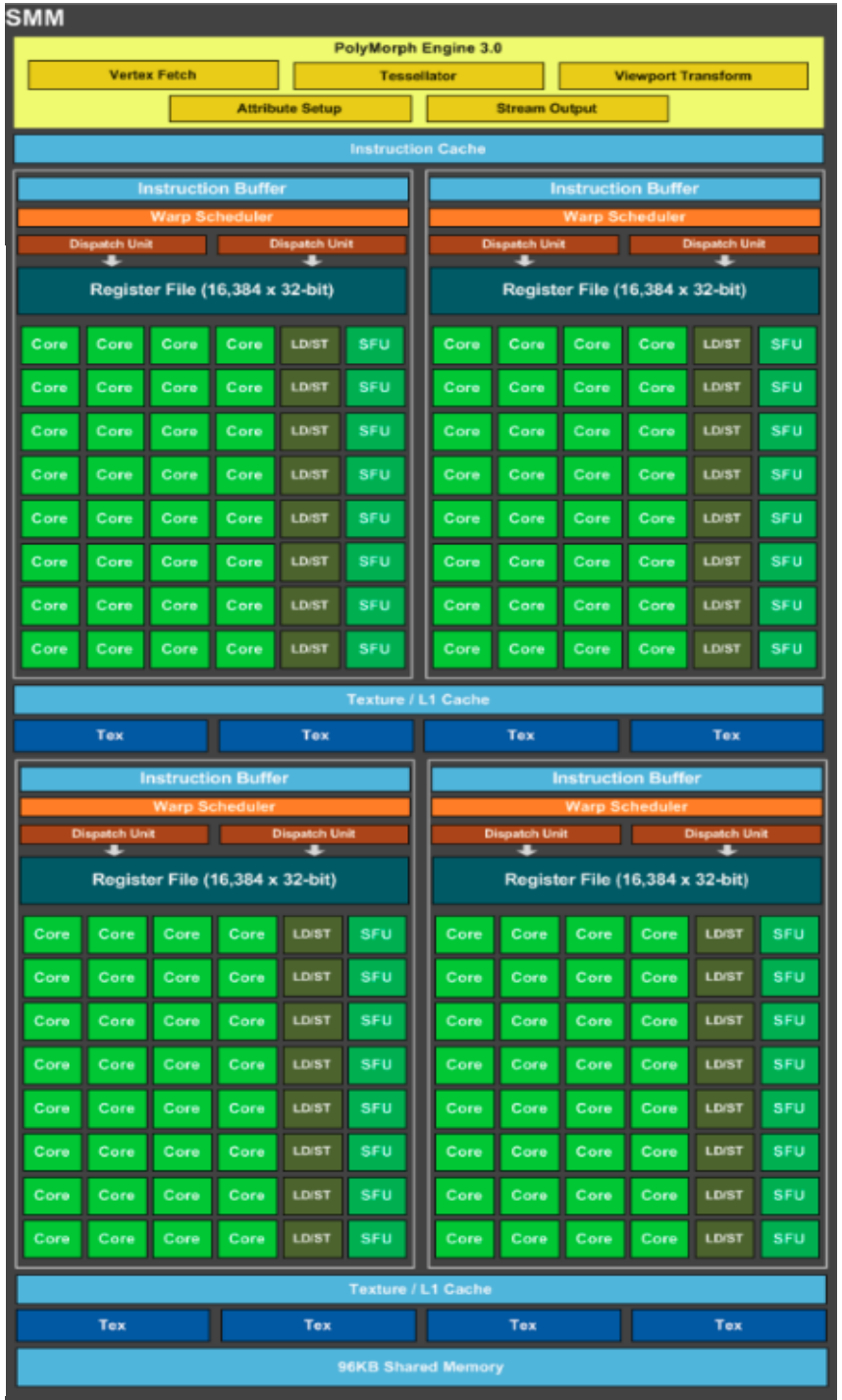
Review: Memory Organization

Global Memory	Shared Memory	Local Memory (not registers)	Texture Memory	Constant Memory
DRAM	DRAM	DRAM	ROM	ROM
- High Latency - Low Throughput	- Low Latency - High Throughput	High Latency	Optimized for 2D spatial locality, enabling one read instruction from cache for multiple threads resulting in better coalescence.	A read costs one memory read from device (global) memory only on a cache miss.
Access is between CPU and device (GPU)	Access is between threads occupying the same block	Access is restricted to individual threads	Located in the global memory on the device. Access is through specialized hardware.	Located on the device. Very small, 64KB+8KB cache for an SM
Off-chip	On-chip	Off-chip	Off-chip	Off-chip
Uncached	Cached	Uncached	Cached	Cached
Persistent	Has the lifetime	Has the		

NVIDIA Maxwell Architecture

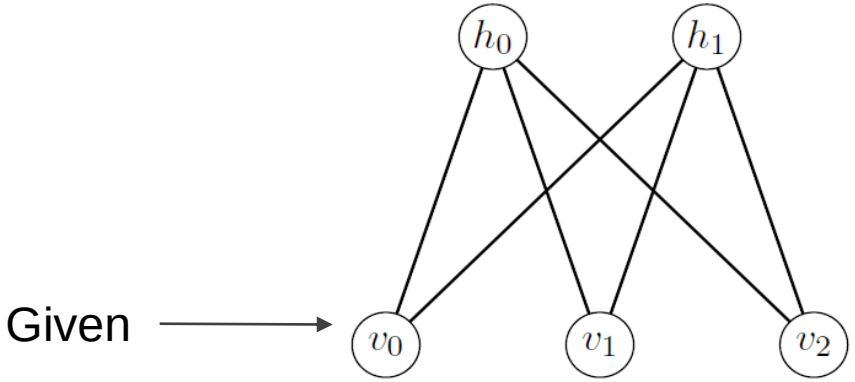
[0] NVIDIA Tegra X1

Compute Capability	5.3
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	96 KiB
Max. Registers per Block	32768
Max. Registers per Multiprocessor	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	32
Half Precision FLOP/s	36.864 GigaFLOP/s
Single Precision FLOP/s	36.864 GigaFLOP/s
Double Precision FLOP/s	1.152 GigaFLOP/s
Number of Multiprocessors	2
Multiprocessor Clock Rate	72 MHz
Concurrent Kernel	true
Max IPC	6
Threads per Warp	32
Global Memory Bandwidth	204 MB/s
Global Memory Size	3.901 GiB
Constant Memory Size	64 KiB
L2 Cache Size	256 KiB
Memcpy Engines	1



Inference

$$p(h_j = 1 \mid v) = \sigma(c_j + \sum_i v_i w_{ij})$$

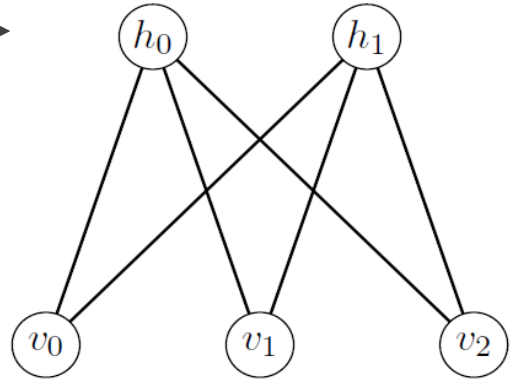



```
__global__ void vis_to_hid_kernel()
{
    int ihid = blockIdx.x * blockDim.x +
threadIdx.x;
    int ichain = blockIdx.y;

    float sum = hid_bias[ihid];
    int randnum;

    for (int ivis = 0; ivis < n_vis; ivis++)
        sum += wtr[ivis*n_hid+ihid] *

        vis_layer[ichain*n_vis+ivis];

    float act_Q = 1.0f / (1.0f + __expf(-
sum));
    hid_layer[ichain*n_hid+ihid] = (frand <
act_Q) ? 1.0 : 0.0;
}
```

Inference

$$p(v_i \mid h) = \sigma(b_i + \sum_j h_j w_{ij})$$

Given →



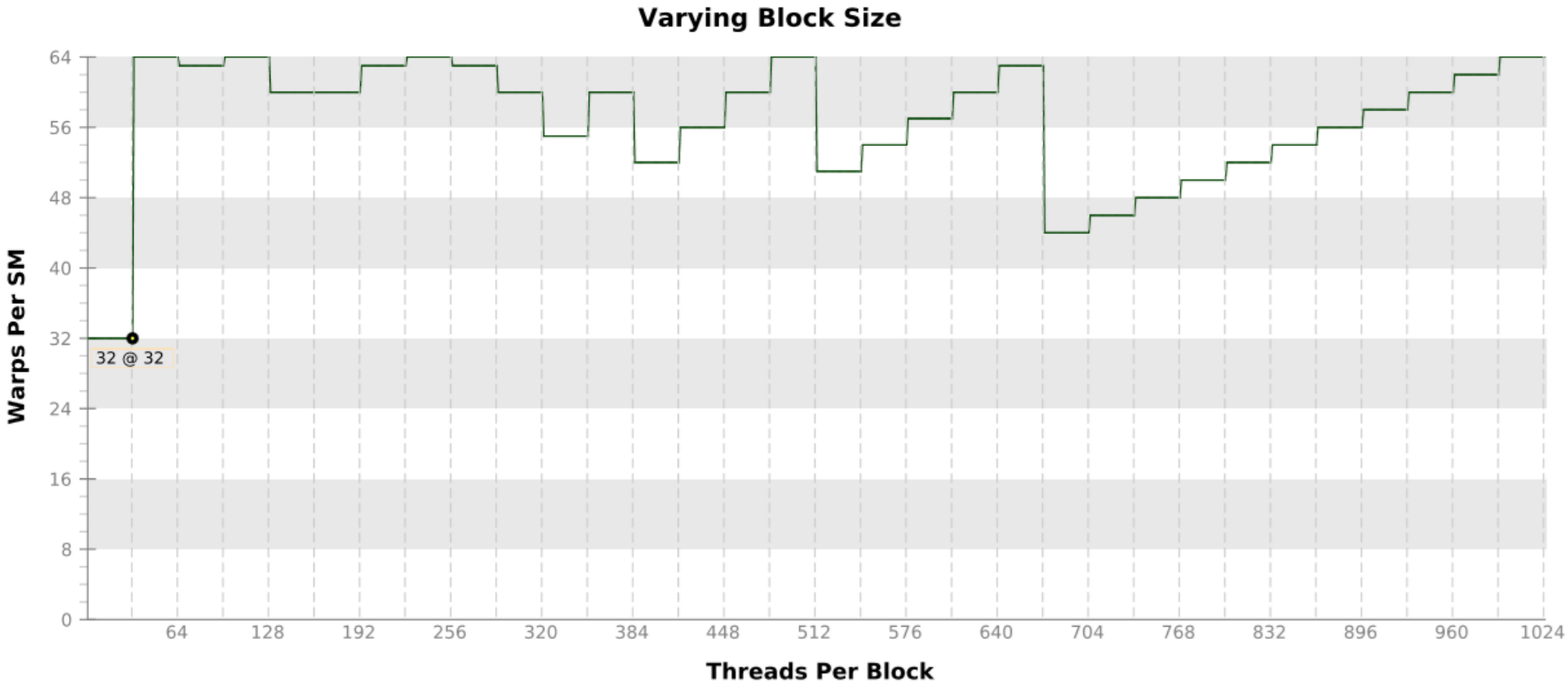

```
__global__ void hid_to_vis_kernel()
{
        int ivis = blockIdx.x *
blockDim.x + threadIdx.x;
        int ichain = blockIdx.y;
        float sum = vis_bias[ivis];
        for (int ihid = 0; ihid < n_hid;
ihid++){
                sum += w[ihid*n_vis+ivis]
* hid_layer[ichain*n_hid+ihid];

        vis_layer[ichain*n_vis+ivis] =
1.0 / (1.0 + __expf(-sum));
}
```

Inference Examples after training

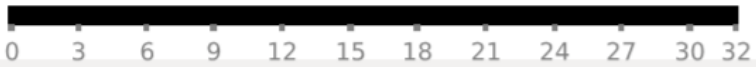
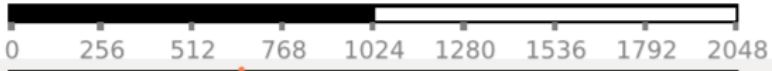
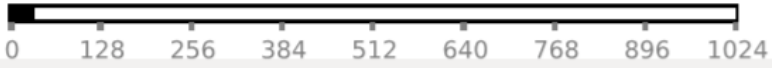
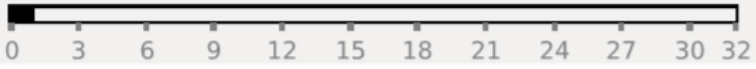
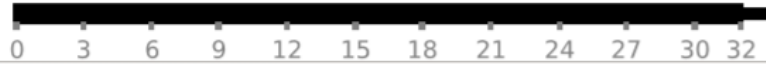
Optimizations

- Unified Memory
- Row-major order global coalescing memory reads/writes
- Different types of memory to use
- Concept of active warps and thread reuse
- Occupancy Calculations

Occupancy Charts



GPU Utilization

Variable	Achieved	Theoretical	Device Limit	Grid Size: [1,40,1] (40 blocks) Block Size: [32,1,1] (32 threads)
Occupancy Per SM				
Active Blocks		32	32	
Active Warps	19.99	32	64	
Active Threads		1024	2048	
Occupancy	31.2%	50%	100%	
Warps				
Threads/Block		32	1024	
Warps/Block		1	32	
Block Limit		64	32	
Registers				
Registers/Thread		32	32768	
Registers/Block		1024	65536	
Block Limit		64	32	
Shared Memory				
Shared Memory/Block		0	98304	
Block Limit			32	