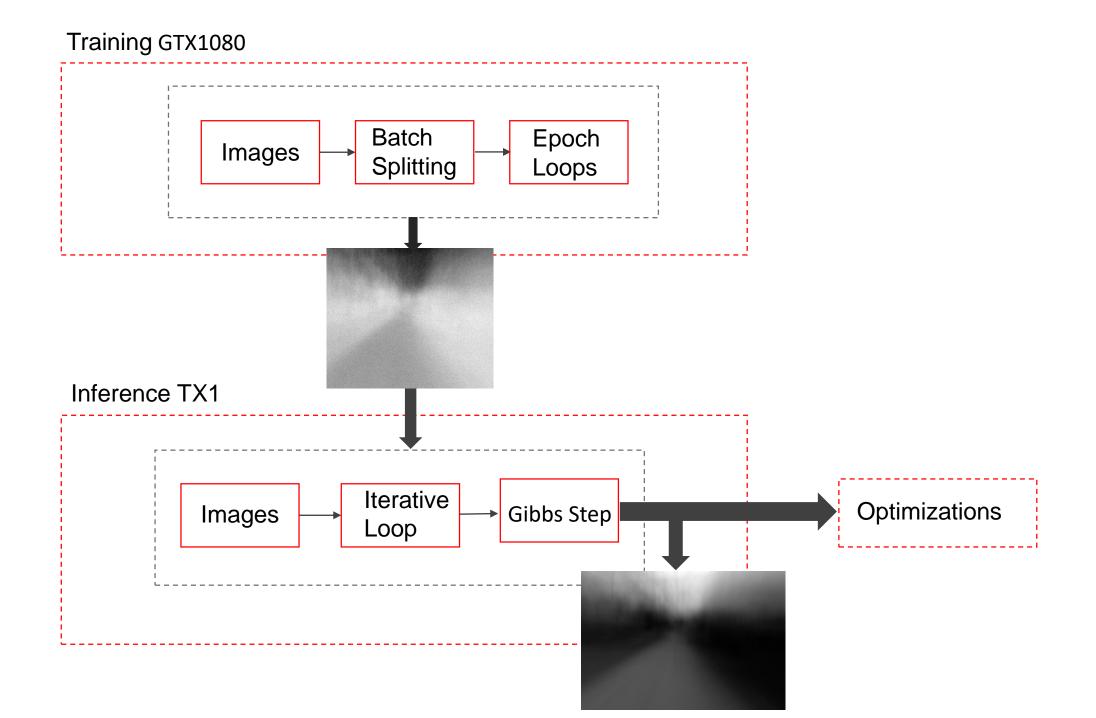


Real-time Inference of Generative Models on TX1 at Traffic Intersections

Menna El-Shaer

el-shaer.1@osu.edu

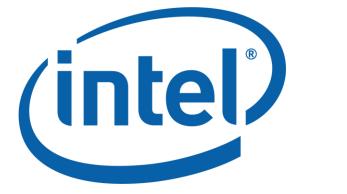


Data Collection

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









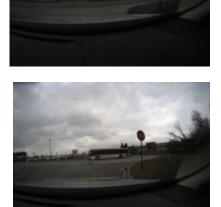
Dataset

- 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

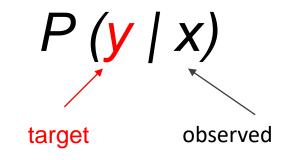
 $P(\mathbf{x}, \mathbf{y})$

observed

non-observed

Discriminative Models

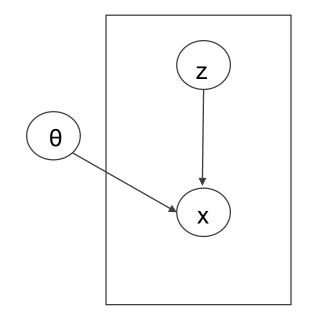
Examples: Classifiers, Logistic Regression, Neural Networks, .. etc

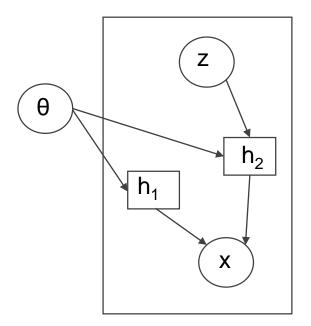


Generative Models

Model Observed Data

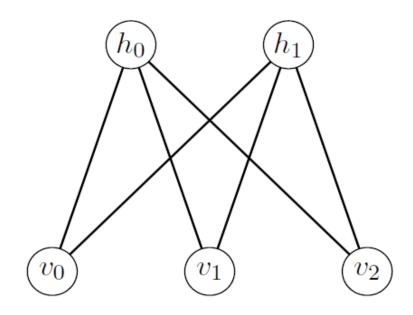
Latent Variable Models





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 $p(h_j = 1 | v) = \sigma(c_j + \sum_i v_i w_{ij})$, then using $p(v_i = 1 | h) = \sigma(b_i + \sum_j h_j w_{ij})$, compute probability of each visible unit = 1 i.e. reconstruct the visible layer.

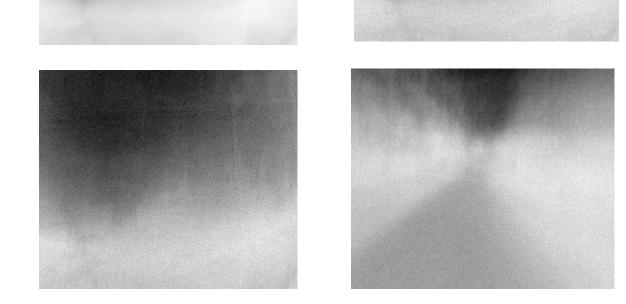
Learning is then done using this reconstruction instead $\Delta w_{ij} = learning_rate(E(v_i, h_j)_{data} - E(v_i, h_j)_{reconstruction})$

Training Weights (Filters)



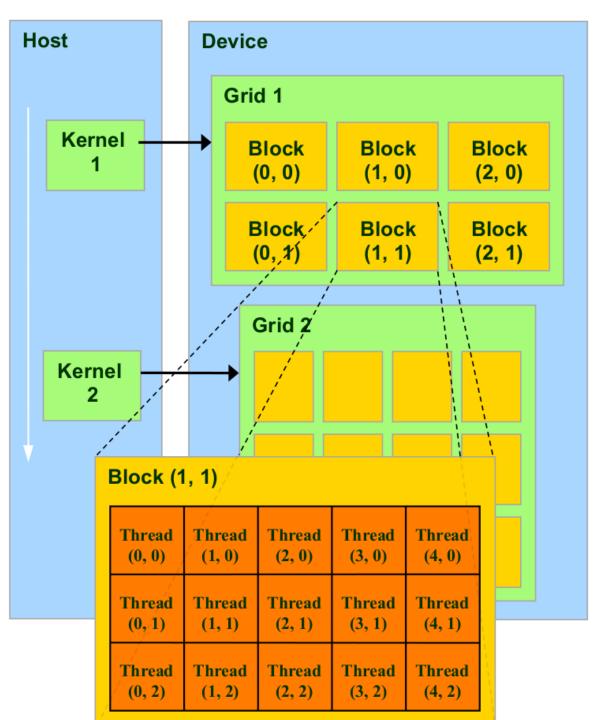
Hyper Parameters

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		

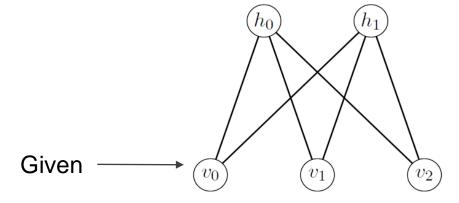
NVIDIA Maxwell Architecture

[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

				P	olyMorpi	h E	Engine 3.	0								
	Verte	x Fetch			Tess	iell	ator		v	iewport T	ransform					
			Attrib	ute Setup				Stream O	utput							
					Instruct	ior	n Cache									
	Ir	nstructio	on Buffe	r		11	Instruction Buffer									
Warp Scheduler							Warp Scheduler Dispatch Unit Dispatch Unit									
Dispatch Unit Dispatch Unit							+ +									
	Registe	er File (1	16,384 x	32-bit)	Register File (16,384 x 32-bit)											
Core	Core	Core	Core	LD/ST	SFU		Core	Core	Core	Core	LD/ST	SFU				
Core	Core	Core	Core	LD/ST	SFU		Core	Core	Core	Core	LD/ST	SFL				
Core	Core	Core	Core	LD/ST	SFU		Core	Core	Core	Core	LD/ST	SFL				
Core	Core	Core	Core	LDIST	SFU		Core	Core	Core	Core	LD/ST	SFL				
Core	Core	Core	Core	LD/ST	SFU		Core	Core	Core	Core	LD/ST	SFL				
Core	Core	Core	Core	LD/ST	SFU		Core	Core	Core	Core	LD/ST	SFU				
Core	Core	Core	Core	LDIST	SFU		Core	Core	Core	Core	LD/ST	SFL				
Core	Core	Core	Core	LD/ST	SFU		Core	Core	Core	Core	LD/ST	SFL				
					Texture	/ L	1 Cache									
	Тех			Tex	Texture	Ī	1 Cache	Tex			Tex					
			on Buffe	_	Texture		1 Cache	_	nstructi							
D		Warp So	cheduler)r					Warp S	cheduler	er	nit				
D	ispatch Un	Warp So	cheduler C	or Dispatch Ur	nit			ispatch Un	Warp Se	cheduler C	or Dispatch U	nt				
D	ispatch Un	Warp So	cheduler C	or Dispatch Ur	nit			ispatch Un	Warp So it	cheduler C	or Dispatch U	nit				
D	ispatch Un	Warp So	cheduler C	or Dispatch Ur	nit			ispatch Un	Warp Se	cheduler C	or Dispatch U					
•	ispatch Uni I Rogist	Warp So at Xer File (*	cheduler c 16,384 x	Dispatch Ur T 32-bit)	nit			ispatch Un T Regist	Warp So it er File (cheduler (16,384 x	Dispatch Ur Tagatch Ur Tagatch Ur Tagatch Ur	SFL				
Core	Regist	Warp So at cor File (* Core	cheduler 16,384 x Core	Dispatch Ur 32-bit) LDIST	nit SFU		Core	Regist	Warp So it er File (' Core	theduler 16,384 x	Dispatch Ur 32-bit) LDrST	SFL				
Core Core	Regist Core Core	Warp So at Core Core	Core Core	Pr Dispatch Uv 32-bit) LDIST LDIST	nit SFU SFU		Core	Regist	Warp So it er File (Core Core	Core Core	er Dispatch U 32-bit) LD/ST LD/ST	SFL SFL				
D Core Core Core	Regist Core Core Core	Warp So at Core Core Core	Core Core Core	ar Dispatch Ur 32-bit) LDIST LDIST LDIST	SFU SFU SFU		Core Core Core	Regist Core Core Core	Warp So it er File (Core Core Core	cheduler 16,384 x Core Core Core	Dispatch Ut 32-bit) LD/ST LD/ST	SFL SFL SFL				
D Core Core Core	Registo Core Core Core Core	Warp So at Core Core Core Core	Core Core Core Core Core Core	ar Dispatch Ui 32-bit) LDIST LDIST LDIST	ot SFU SFU SFU SFU		Core Core Core Core	Regist Core Core Core Core	Warp So it Core Core Core Core	Core Core Core Core Core	or Dispatch Ur t 32-bit) LD/ST LD/ST LD/ST	SFL SFL SFL SFL				
D Core Core Core Core	Register Core Core Core Core Core	Warp Solar at File (Core Core Core Core Core	Core Core Core Core Core Core	ar Chispatich Ui 32-bit) LOIST LDIST LDIST LDIST	SFU SFU SFU SFU SFU		Core Core Core Core Core	Regist Core Core Core Core Core	Warp So it Core Core Core Core Core Core	cheduler 16,384 x Core Core Core Core Core	Construction Construction Construction Construction Construction Construction Construction Construction	SFL SFL SFL SFL SFL				
Core Core Core Core Core Core	Register Core Core Core Core Core Core	Warp Sr Ter File (Core Core Core Core Core Core Core	Core Core Core Core Core Core Core Core	Constraints of the second seco	SFU SFU SFU SFU SFU SFU		Core Core Core Core Core Core	Regist Core Core Core Core Core Core	Warp Sr at Core Core Core Core Core Core Core	Core Core Core Core Core Core Core Core	Clispatich Un 232-bit) LD/ST LD/ST LD/ST LD/ST LD/ST	SFL SFL SFL SFL SFL				
Core Core Core Core Core Core	Registr Core Core Core Core Core Core Core	Werp State	Core Core Core Core Core Core Core Core	Claspatch Ud 32-bit) LDIST LDIST LDIST LDIST LDIST LDIST	SFU SFU SFU SFU SFU SFU SFU SFU		Core Core Core Core Core Core	Regist Core Core Core Core Core Core Core Core	Warp St it Core Core Core Core Core Core Core Core Core	Core Core Core Core Core Core Core Core	Construction of the second sec	SFL SFL SFL SFL SFL SFL				

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;
```

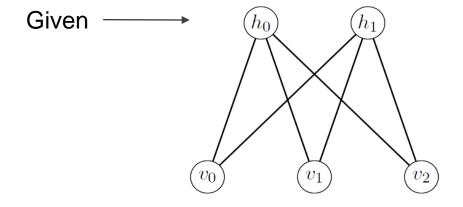
```
for (int ivis = 0; ivis < n_vis; ivis++)
    sum += wtr[ivis*n_hid+ihid] *</pre>
```

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;
}</pre>
```

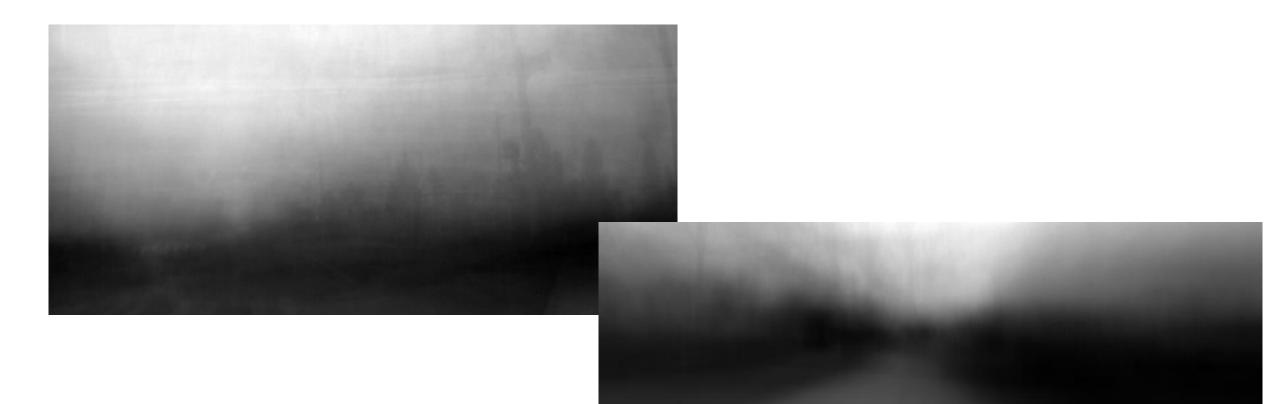
Inference

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



```
_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;</pre>
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));
```

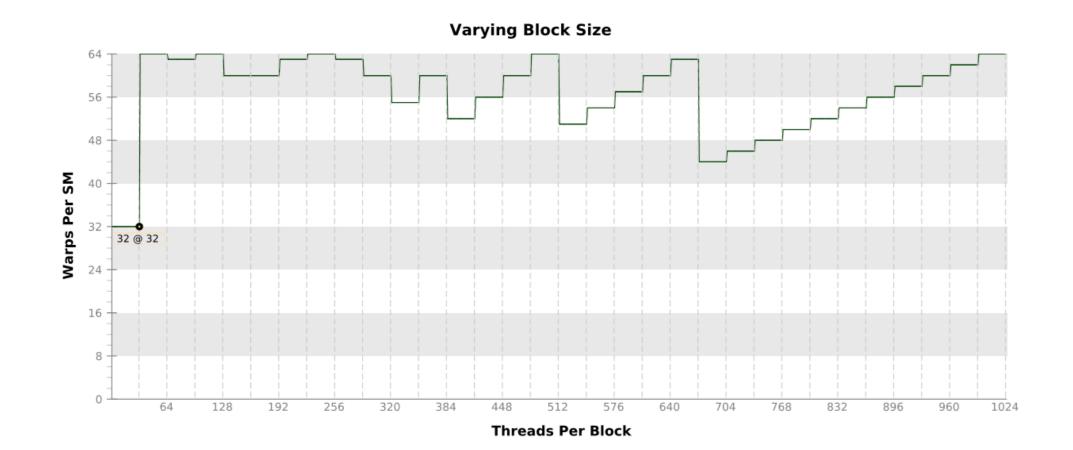
Inference Examples after training



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 S	ize: [1,40	,1](40 blo	cks) E	Block	Size:	[32,1,	1](3	2 thr	eads
Occupancy Per SM	i .		1												
Active Blocks		32	32	0	3	6	9	12	15	18	21	24	27	30	32
Active Warps	19.99	32	64	0	7	14	L ;	1 21	28	35	42	49	56)	6 6 4
Active Threads		1024	2048	0	256	5	512	768	10	24	1280	1536	179)2	204
Occupancy	31.2%	50%	100%	0%		2	25%	(5	0%		75%	0		10
Warps															
Threads/Block		32	1024	0	128	2	256	384	51	12	640	768	89	6	102
Warps/Block		1	32	0	3	6	9	12	15	18	21	24	27	30	32
Block Limit		64	32	0	3	6	9	12	15	18	21	24	27	30	32
Registers															
Registers/Thread		32	32768	0	4096	6 8	192	1228	8 163	384	20480	24576	6 286	72 3	27
Registers/Block		1024	65536	0			16k		3	2k		48k		_	64
Block Limit		64	32	0	3	6	9	12	15	18	21	24	27	30	32
Shared Memory															
Shared Memory/Block		0	98304	0				32k			64	k			96
Block Limit			32												