Large-scale Deep Unsupervised Learning using Graphics Processors

Rajat Raina

Anand Madhavan

Andrew Y. Ng

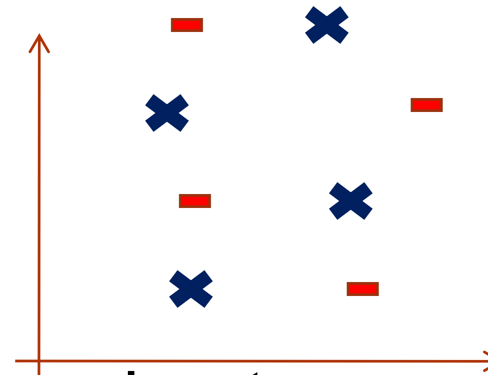
Stanford University

Learning from unlabeled data

Classify

vs.

✖ car — motorcycle



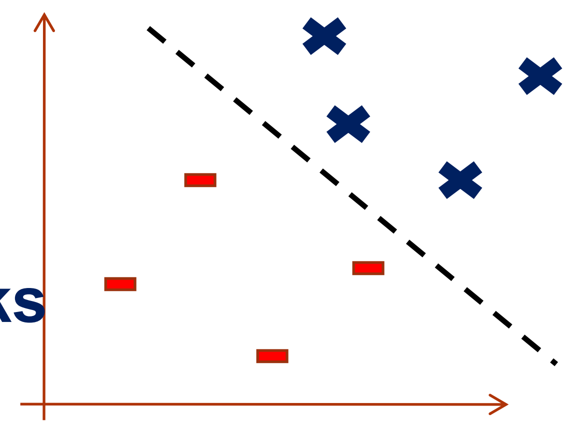
Input space

Unlabeled examples

→ Learn higher-level representation

Deep Belief Networks

Sparse Coding



Higher-level representa

The promise of unsupervised learning

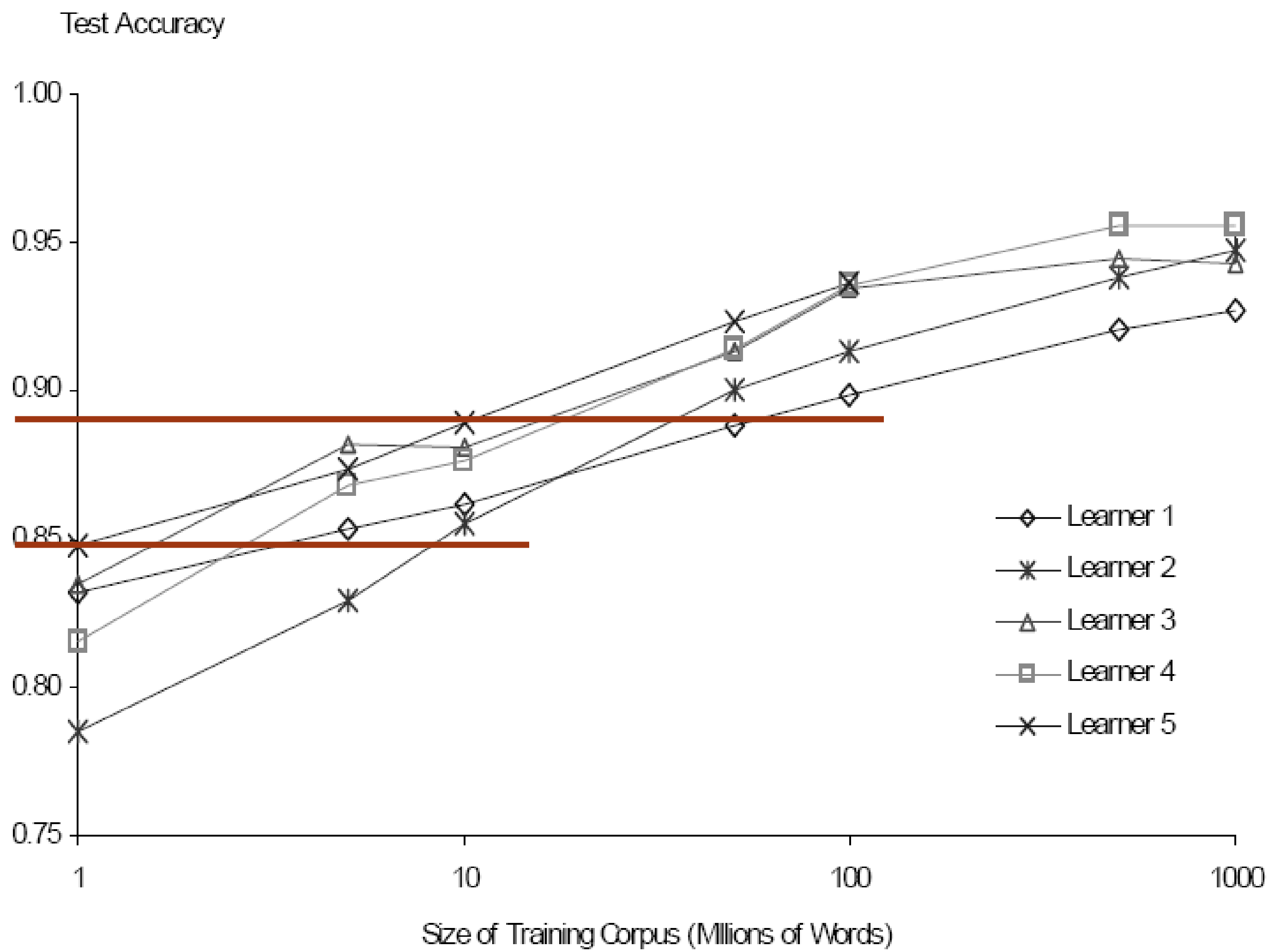
Use **large amounts of unlabeled data** to learn **complex/deep models**, possibly with many parameters.

Some recent work on DBNs

Published Source	Domain	Number of free parameters
Hinton et al.	Handwritten digits	1.6 million
Hinton & Salakhutdinov	Face images	3 million
Salakhutdinov & Hinton	Information retrieval	2.6 million
Ranzato & Szummer	Text documents	3.6 million
Our DBN model over images		**100 million**

(Similar situation for sparse coding.)

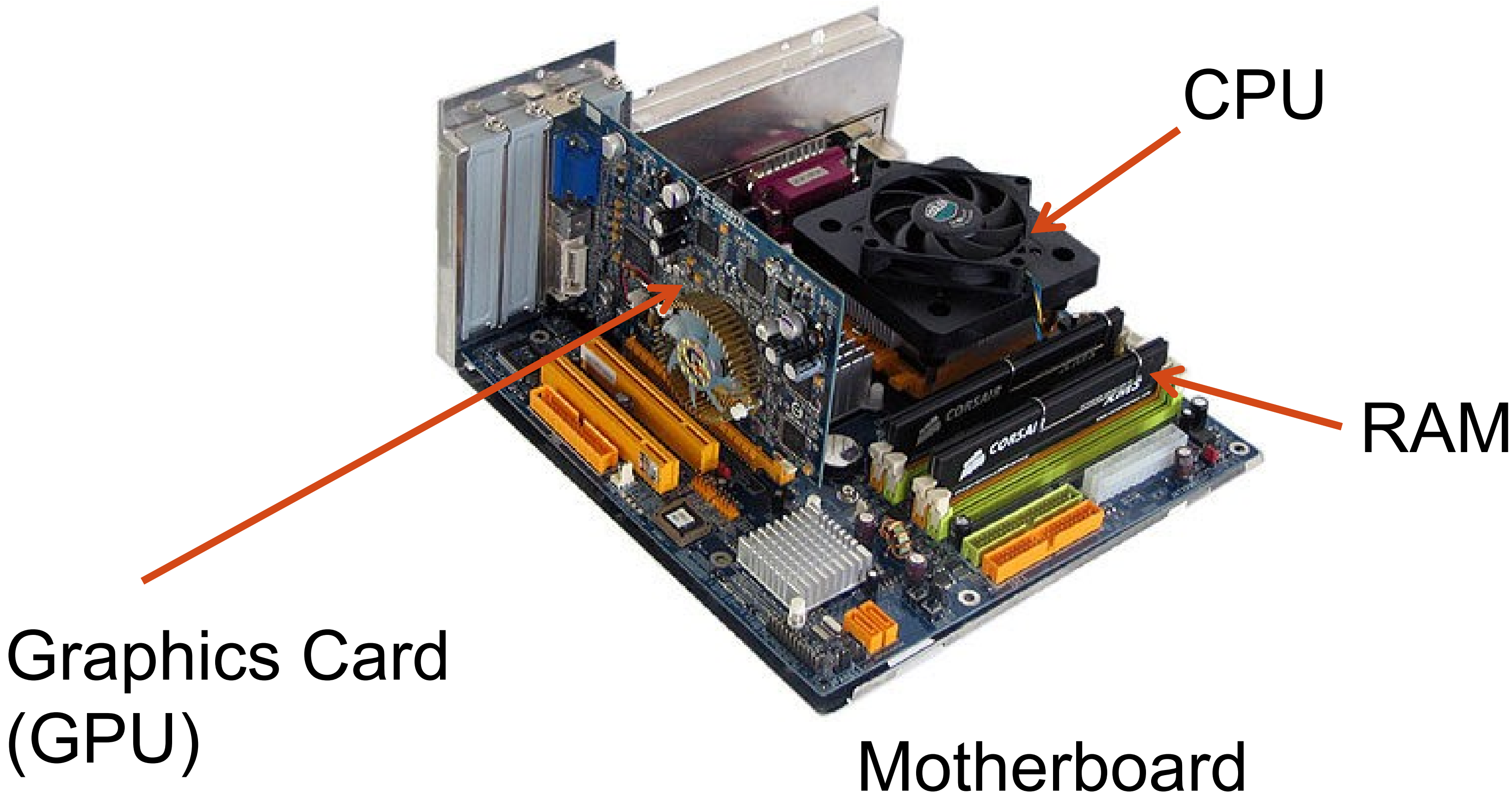
Large-scale learning [Banko & Brill, 2001]



Large-scale unsupervised learning

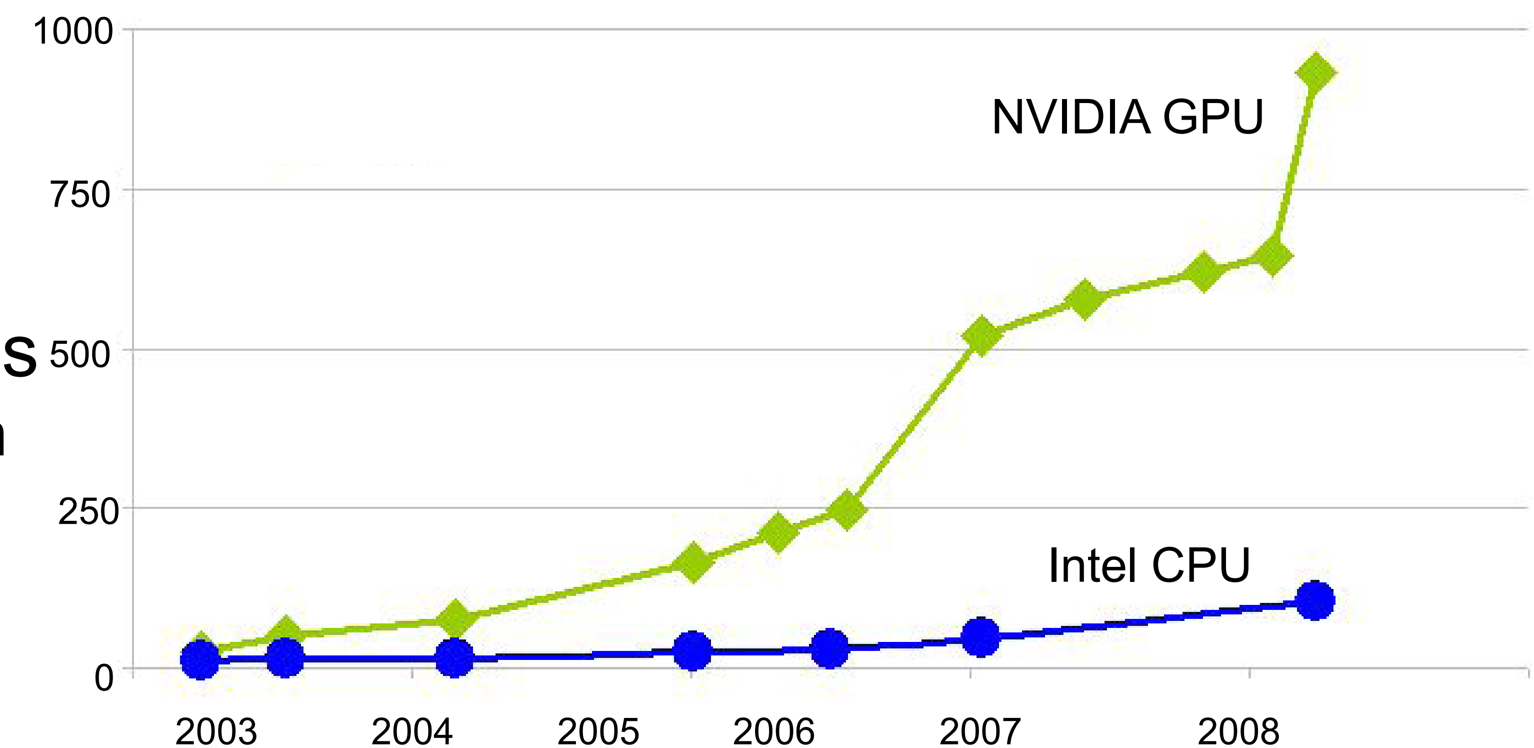
- Current models: 1000s of input dimensions, 1000s of hidden units. 10^6 parameters.
- Our desired model: 10^8 parameters

Graphics Processors



Why graphics processors?

Peak Gflops (billion ops / sec)



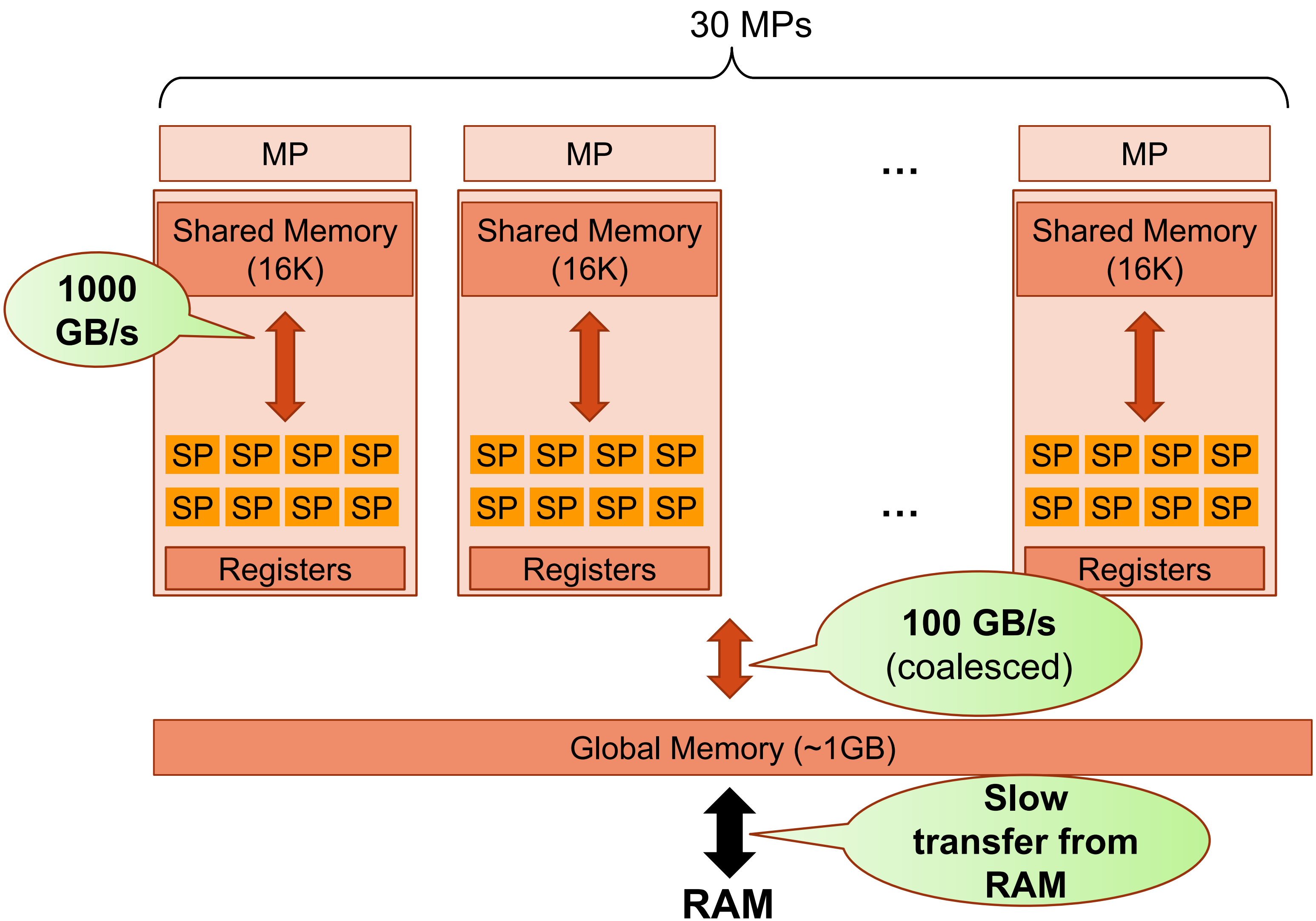
(Source: NVIDIA CUDA Programming Guide)

Why graphics processors?

IBM ASCI White
Supercomputer
Cost: $110 million
Space: 2 basketball courts

13 graphics cards

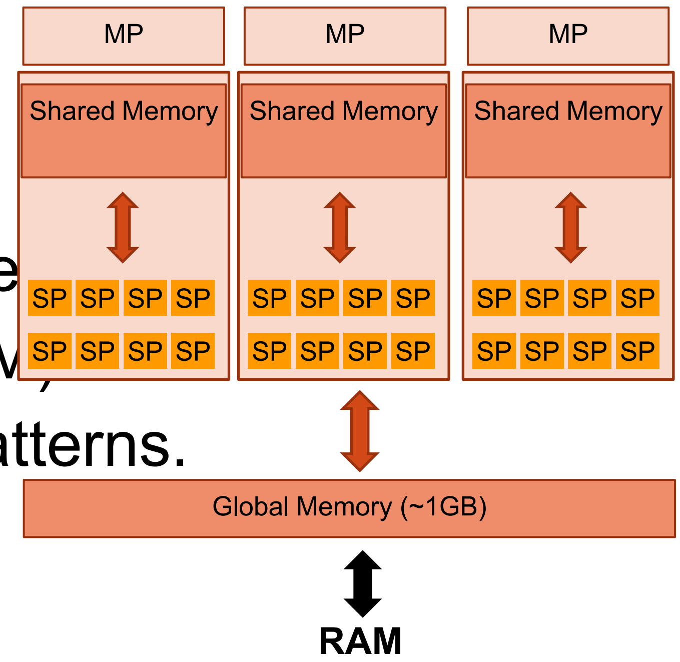
GPU Schematic



(Note: Some additional features not displayed.)

GPU Programming

- Two-level parallelism
- Split task into blocks, blocks into thre
  - Access to global memory (not RAM)
  - Restrictions on memory access patterns.



- Main bottleneck:
  - Getting data into GPU memory, and accessing it in efficient ways.
- NVIDIA CUDA
  - High-level routines to allocate/copy GPU memory.
  - Good GPU matrix libraries that suffice for many machine learning tasks.

Unsupervised learning on GPUs

Initialize parameters in global memory.

while convergence criterion is not satisfied

  Periodically transfer a large number of unlabeled examples into global memory.

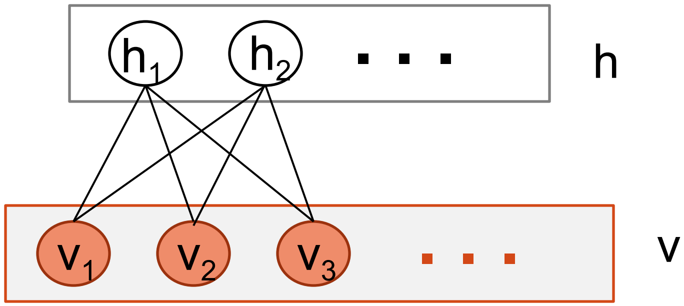
  Pick a few of the unlabeled examples at a time, and compute the updates in parallel using the GPU's two-level parallelism (blocks and threads) or GPU matrix libraries.

end

Transfer learnt parameters from global memory.

Deep Belief Networks

Restricted Boltzmann Machine (RBM)



$$p(v,h) \propto e^{-E(v,h)}$$

$$E(v,h) = -(\sum_{i,j} v_i W_{ij} h_j + \sum_i c_i v_i + \sum_j b_j h_j)$$

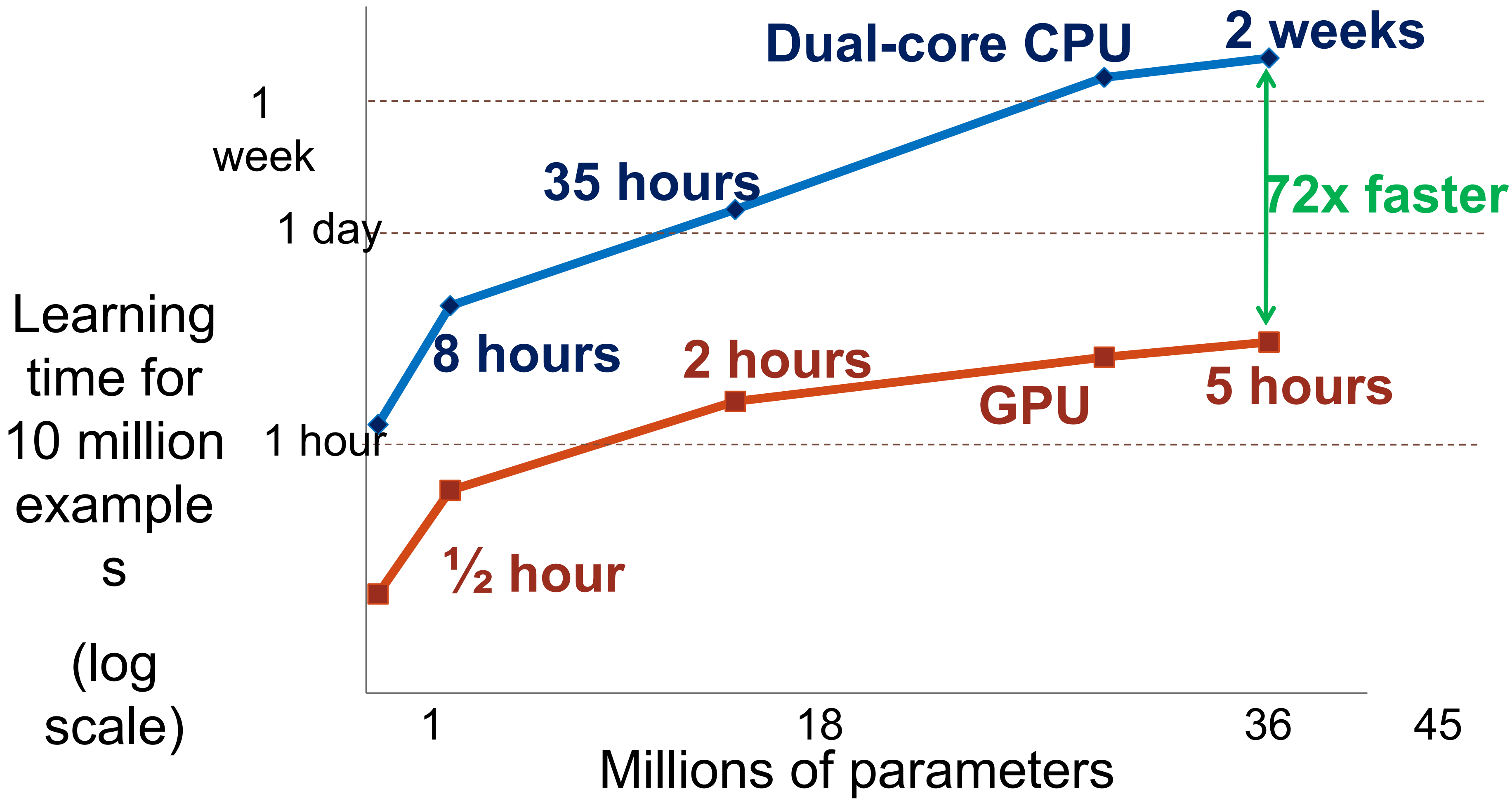
Contrastive divergence learning via conditional distributions: $p(h \mid v) = g(W^T v + b)$

$$p(v \mid h) = g(Wh + c)$$

Experimental setup

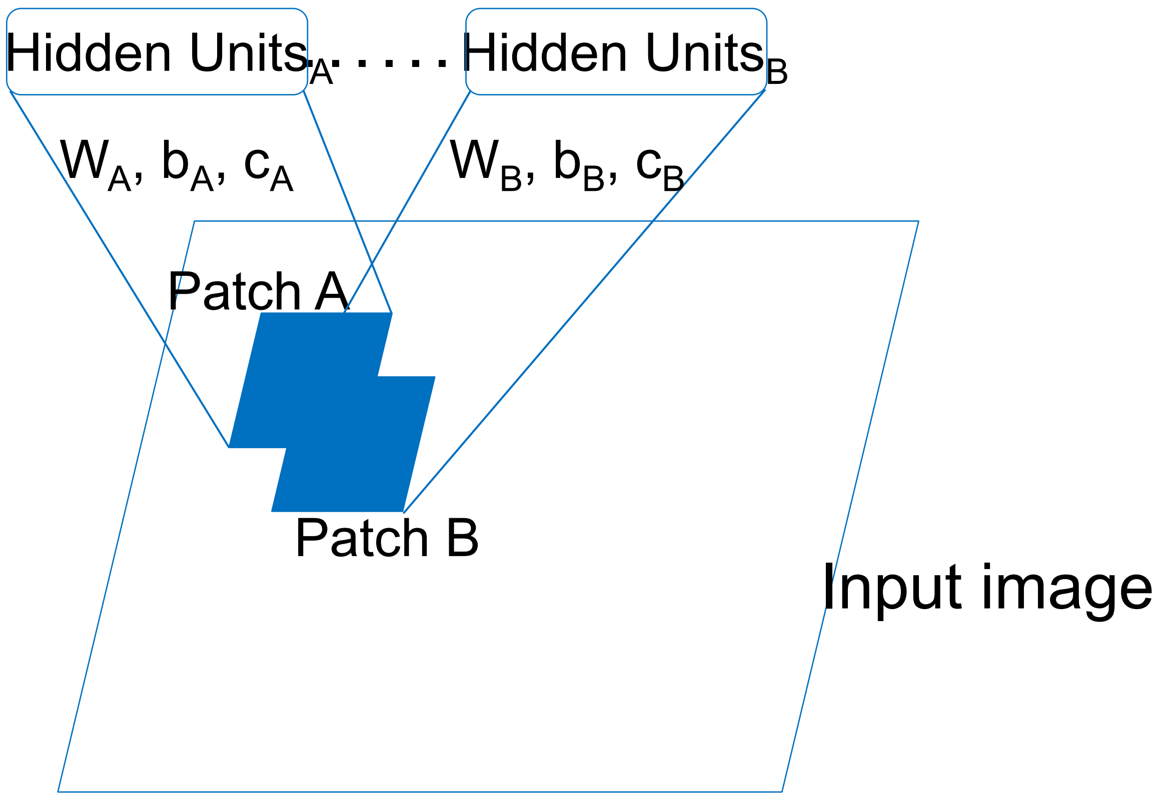
- Single graphics card: Nvidia GTX 280
  - 1GB on-board memory, 240 cores.
  - Current price: US $250.
- CPU:
  - Two cores, each @3.16GHz.

Learning Large RBMs

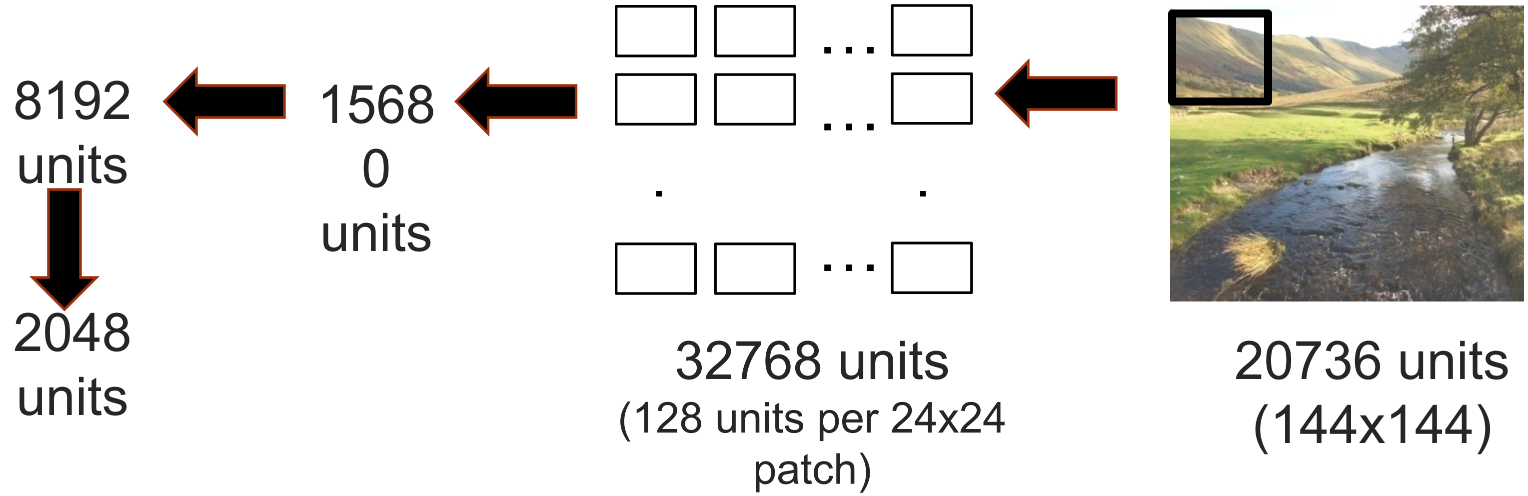


Rajat Raina, Anand Madhavan, Andrew Y. Ng

Overlapping patches DBN



Overlapping patches DBN example

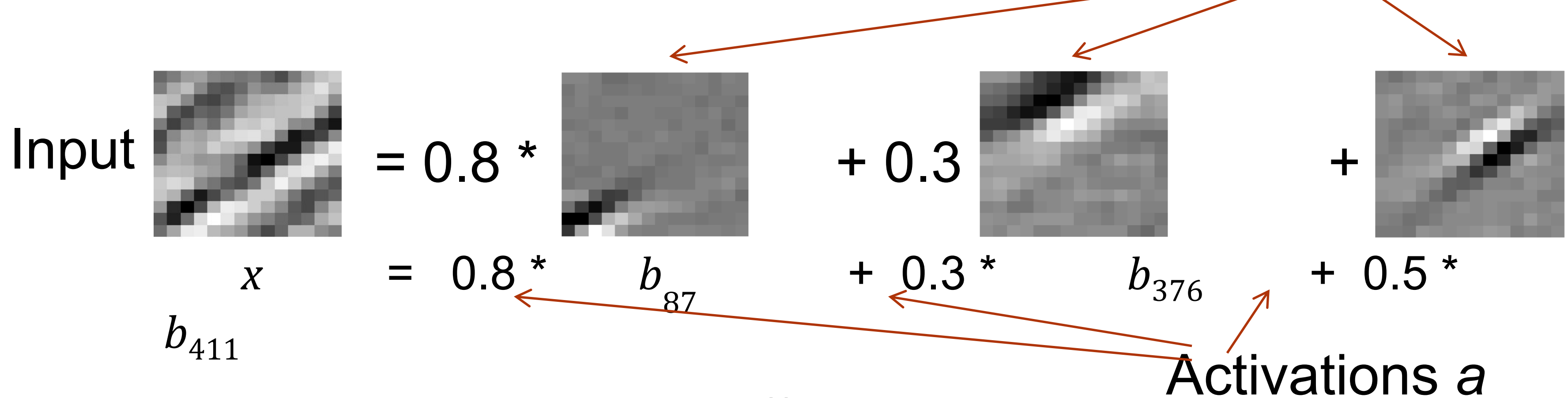


- **110 million parameters.**

All layers can be learnt in about 1 day on a GPU.

Sparse Coding

Sparse coding



Given unlabeled data $x^{(i)}$, obtain b by solving:

$$\min_{b,a} \sum_i \| x^{(i)} - \sum_j a_j^{(i)} b_j \|_2^2 + \beta \sum_i \| a^{(i)} \|_1$$

$$\forall j: \ \| b_j \| \leq 1$$

Alternating minimization

- Keep *a* fixed, find optimal *b*.
- Keep *b* fixed, find optimal *a*.

Parallel Sparse Coding

$$\min_{b,a} \sum_i \| x^{(i)} - \sum_j a_j^{(i)} b_j \|_2^2 + \beta \sum_i \| a^{(i)} \|_1$$

$$\forall j: \ \| b_j \| \le 1$$

- Alternating minimization
  - Keep *a* fixed, find optimal *b*. Easy on GPU (projected grad descent).
  - Keep *b* fixed, find optimal *a*. Not as straightforward.
- Need to parallelize:

$$\min_a \| x - \sum_j a_j b_j \|_2^2 + \beta \| a \|_1$$

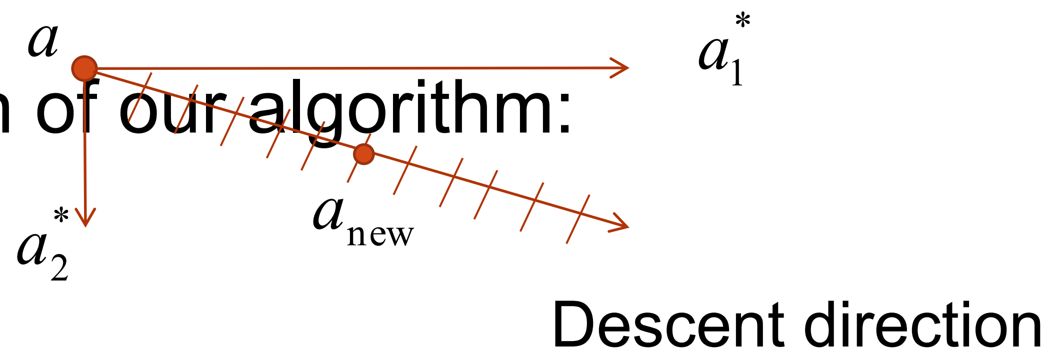
Parallel Sparse Coding

$$\min_a \| x - \sum_j a_j b_j \|_2^2 + \beta \| a \|_1$$

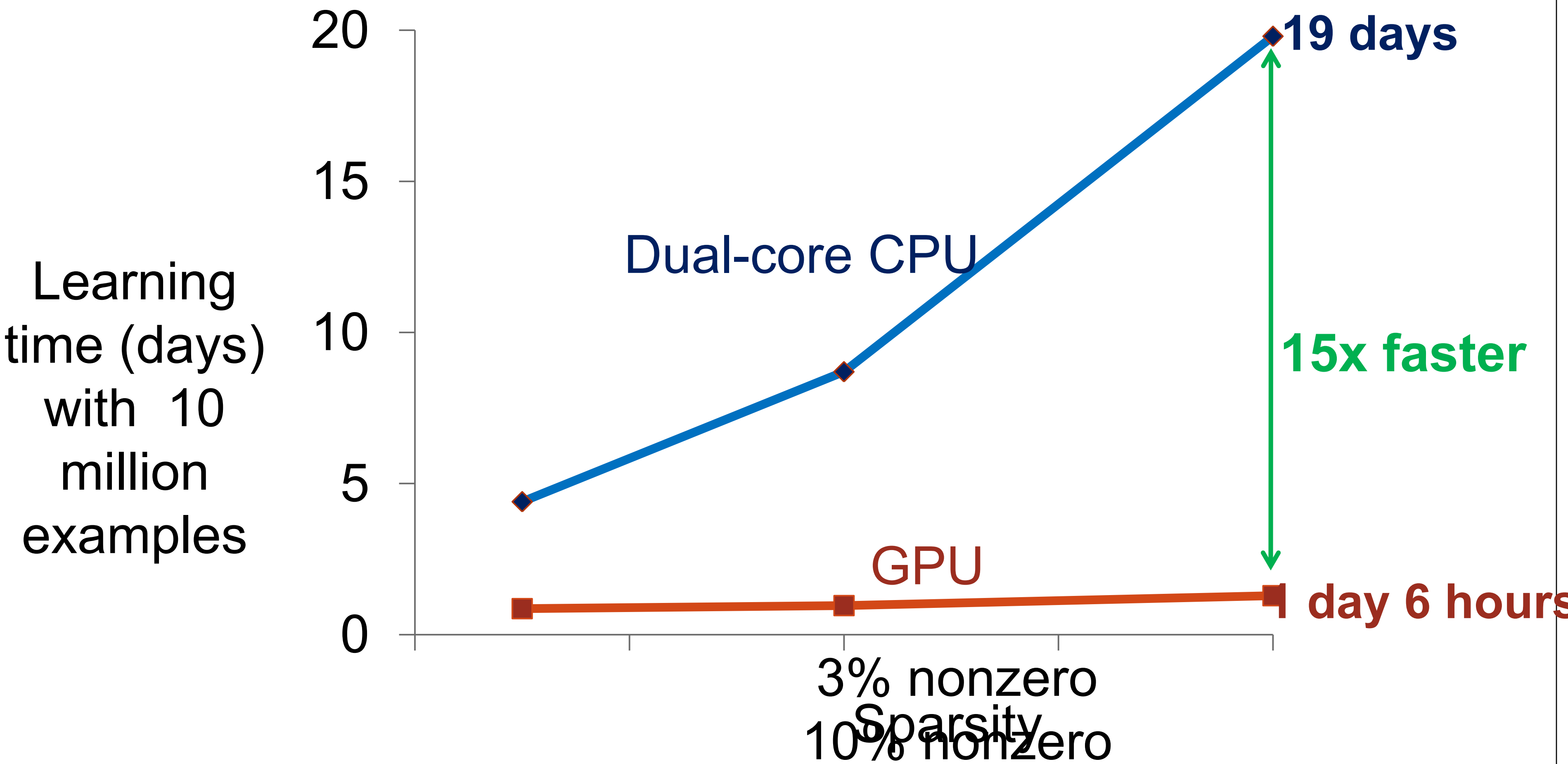
- Easy to optimize for one coordinate (keeping the others fixed).

(Friedman et al., 2007)

- One iteration of our algorithm:



Sparse coding with 10^6 parameters



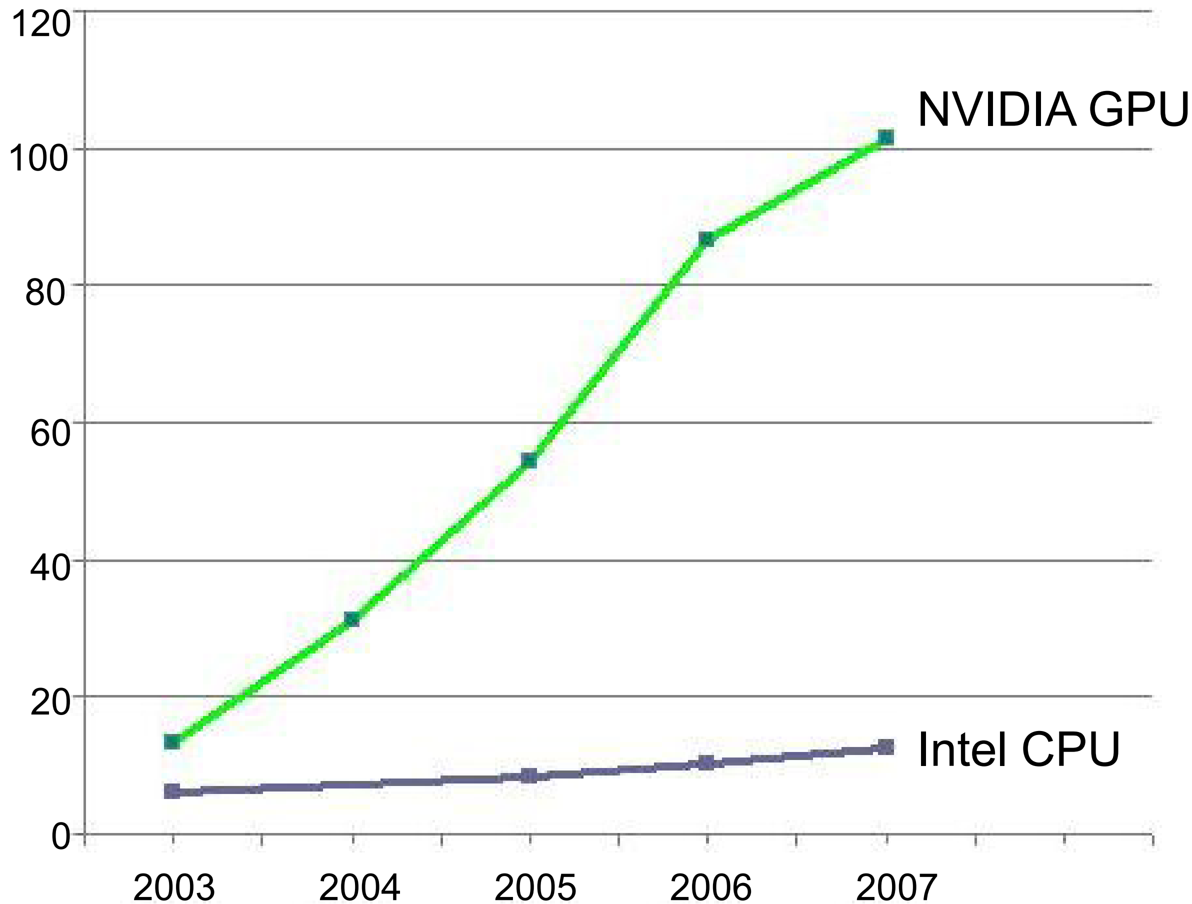
Summary

- Large-scale unsupervised learning.
  - Ten-times more data might transform an OK algorithm into a good algorithm.
  - Working at smaller-scale risks confounding the effects of the model itself, with the effect of scale.
- GPUs are a powerful tool for machine learning.
  - Easy to program (no low-level programming).
  - Especially useful for stochastic learning methods.
- Learning algorithms for DBNs and sparse coding can be an order-of-magnitude faster.

THE END

Why graphics processors?

Bandwidth from memory to processor (GB/s)



(Source: NVIDIA CUDA Programming Guide)

GPU Programming: A=A+B

GPU

```
__global__ void vecAdd(float* A, float* B){
    int my = [          ] + [          ] * 128;
    A[my]=A[my]+B[my];
}
```

CPU

```
int main(int argc, char** argv){
    float A[SIZE], B[SIZE];
    float* d_A, * d_B;
    cudaMalloc((void**)&d_A,SIZE_BYTES);
    cudaMalloc((void**)&d_B,SIZE_BYTES);
    cudaMemcpy(d_A,A,SIZE_BYTES,cudaMemcpyHostToDevice);
    cudaMemcpy(d_B,B,SIZE_BYTES,cudaMemcpyHostToDevice);
    [                                                    ]
    cudaThreadSynchronize();
    cudaMemcpy(A,d_A,SIZE_BYTES,cudaMemcpyDeviceToHost);
}
```

(Adapted from http://www.cs.technion.ac.il/~marks/docs/LinuxClubGPGPU.pdf)