# Deep Learning and Quantum Many Body Systems

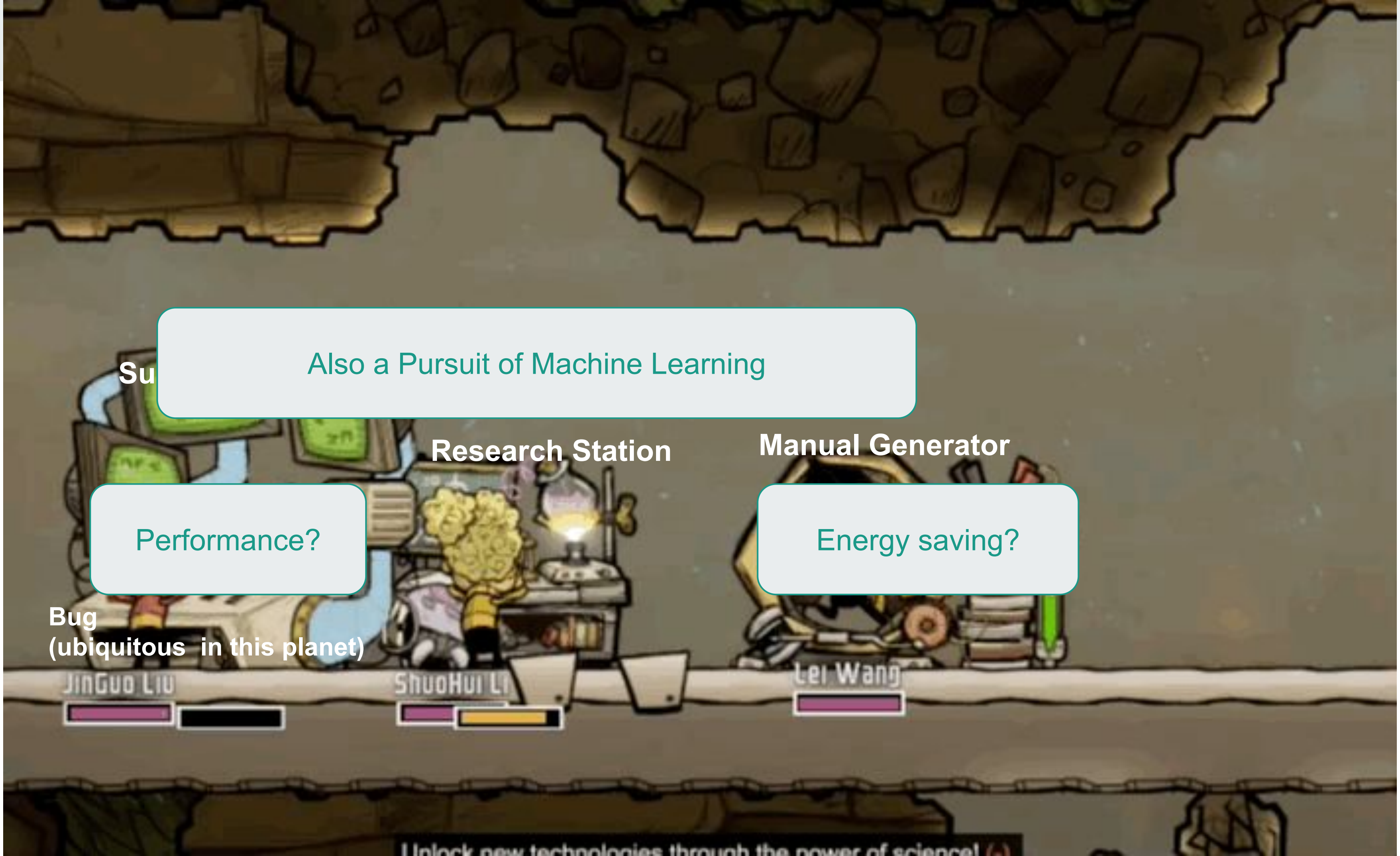
A programming guide

刘金国 (Jin-Guo Liu)

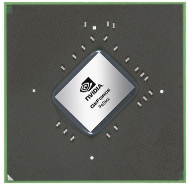
# Setup your workplace

Following the guide
https://github.com/GiggleLiu/marburg/

If you cloned this repository or lecture notes days before, pull for updates please!


Also a Pursuit of Machine Learning
Su
Research Station
Manual Generator
Performance?
Energy saving?
Bug
(ubiquitous in this planet)
JinGuo Liu
ShuoHui Li
Lei Wang

# Use the Right Device



How to calculate FLOPS?
See notebook *gpu/hardware.ipynb* in our github repository for an example.

**CPU**
Ultra-low power
Intel Core i5-7200U (15 W)

**Graphics Card**
NVIDIA GeForce 940MX (23 W)

Parallelism

| | CPU | Graphics Card |
|---|---|---|
| Parallelism | 2 cores (2.5GHZ), AVX2 SIMD | 384 CUDA cores (1.189GHZ) |
| | 40 GFLOPS | 913 GFLOPS |

# AVX2 acceleration of saxpy function

$$\mathbf{y} = a\mathbf{x} + \mathbf{y}$$

Not vectorized

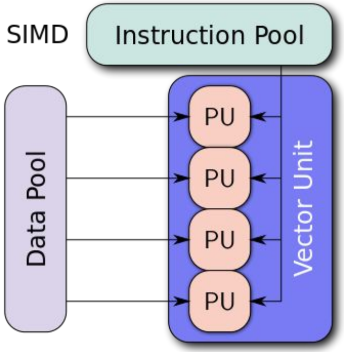
```c
void saxpy(int n, float a, float *x, float *y) {
    for (int i=0; i<n; i++)
        y[i] = a*x[i] + y[i];
}
```

AVX2 vectorized

```c
void saxpy(int n, float a, float *x, float *y){
    __m256 x_vec, y_vec, a_vec, res_vec;      //define the registers used
    a_vec = _mm256_set1_ps(a); // Vector of 8 alpha values
    for (int i=0; i<n; i+=8) {
        x_vec = _mm256_loadu_ps(&x[i]); // Load 8 values from X
        y_vec = _mm256_loadu_ps(&y[i]); // Load 8 values from Y
        res_vec = _mm256_fmadd_ps(a_vec, x_vec, y_vec); // Compute
        _mm256_store_ps(&y[i], res_vec);
    }
}
```

Load data to SIMD register

Read result from SIMD register



Source

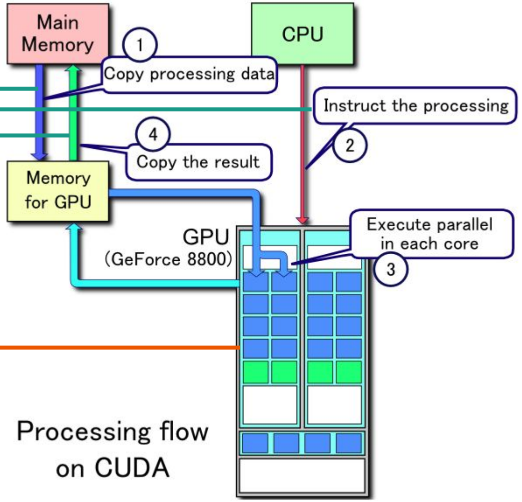
https://devblogs.nvidia.com/easy-introduction-cuda-c-and-c/

# CUDA programming model

```
__global__ void saxpyDevice(int n, float a, float *x, float *y){
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}

void saxpy(int n, float a, float *x, float *y){
    float *d_x, *d_y;

    cudaMalloc(&d_x, n*sizeof(float));
    cudaMalloc(&d_y, n*sizeof(float));

    cudaMemcpy(d_x, x, n*sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_y, y, n*sizeof(float), cudaMemcpyHostToDevice);

    // Perform SAXPY on GPU
    saxpyDevice<<<(n+255)/256, 256>>>(n, 2.0f, d_x, d_y);

    cudaMemcpy(y, d_y, n*sizeof(float), cudaMemcpyDeviceToHost);

    cudaFree(d_x);
    cudaFree(d_y);
}
```

device code

host code




Processing flow on CUDA

# Free ourselves from low-level programming

Define the Computation $\xrightarrow{\text{lib}}$ Choose the device to run

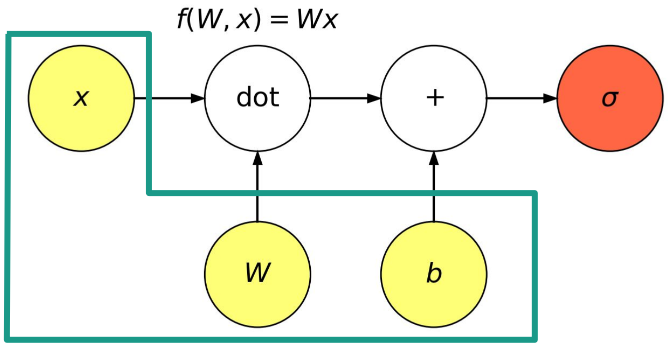
Computation graph

**Neural network version of Feynman Diagram**

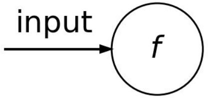
CPU/GPU(s)/TPU

# An Example of Computation Graph

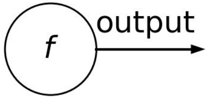
$$f(x) = \sigma(Wx + b)$$




load data to graph




**function argument**



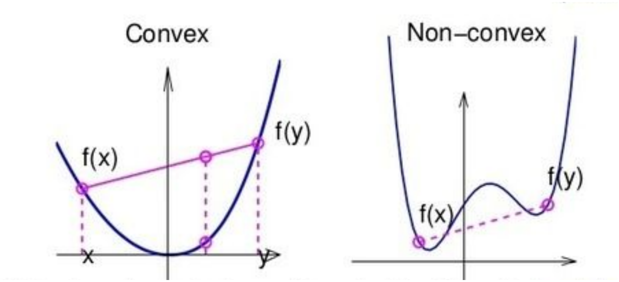

**function output**

A node knows how to compute its **derivative** w.r.t each argument (edge).

Can be used to train a NN Loss function

Choromanska, Anna, et al. "The loss surfaces of multilayer networks." Artificial Intelligence and Statistics. 2015.

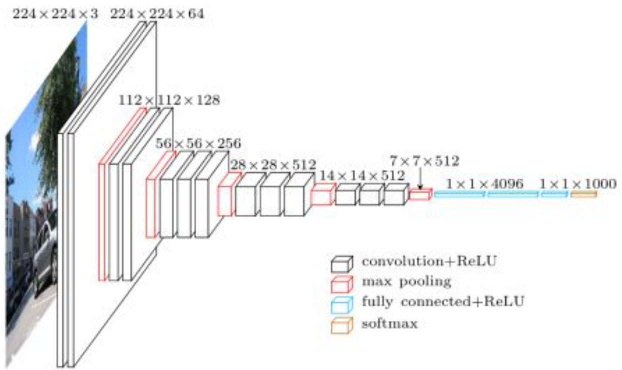
# Losses Expressed as Computation Graphs




theory reality

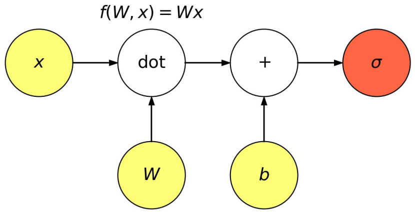
**Convex optimization model**

VGG-16 network (138 M parameters)



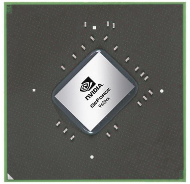

Not to find the global minima, but to obtain a low enough loss efficiently.

# So Far



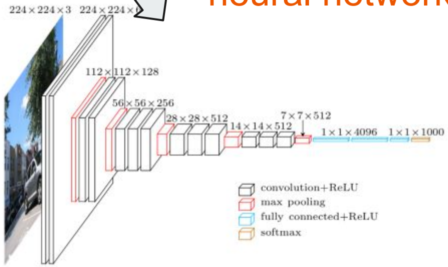

Computation Graphs

Provide unified framework to use computation resources

Devices

Provide efficient training of neural networks




Neural Networks

# Hands on time

with numpy:

build functions used in computation graphs

conquer digits classification problem

★ build your own computation graph node

Notebook

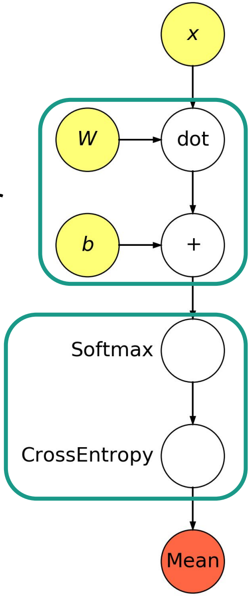
[https://goo.gl/6d2sei](https://goo.gl/6d2sei)

local: notebooks/computation_graph.ipynb

# A Digit Classification Problem

Linear

SoftmaxCrossEntropy

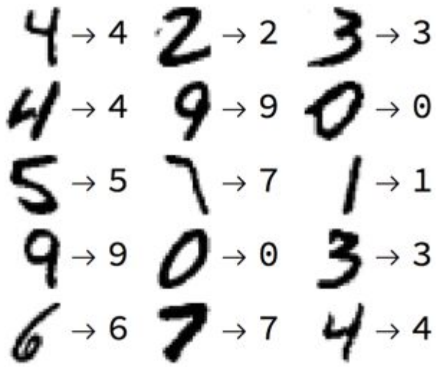



Generate One hot labels

Comparison

**Target: Correctly assign labels, e.g.**

Softmax: [ 0.1, 0, 0, 0, 0.8, 0.1, 0, 0, 0, 0 ]
Label: [ 0 , 0, 0, 0, 1 , 0 , 0, 0, 0, 0 ]




A typical Application
Detect Phase Transition

# Solution

$$y = \sum_i (\log(\sum_j e^{x_j}) - x_i) p_i$$

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\partial \mathcal{L}}{\partial y} \left( \frac{e^{x_i}}{\sum_j e^{x_j}} - p_i \right)$$

# State of Art Python Neural Network Libraries

**TensorFlow™**

| | |
|---|---|
| **Developer** | Google Brain Team |
| **Initial release** | November 9, 2015 |
| **Stable release** | 1.5.0 / January 26, 2018 |
| **Repository** | github.com/tensorflow/tensorflow |
| **Written in** | Python, C++, CUDA |
| **Platform** | Linux, macOS, Windows, Android |
| **License** | Apache 2.0 open source license |
| **Website** | www.tensorflow.org |

**PYTORCH**

| | |
|---|---|
| **Developer** | Facebook's AI research group |
| **Initial release** | October 2016 |
| **Stable release** | 0.3.0 / 5 December 2017 |
| **Repository** | github.com/pytorch/pytorch |
| **Written in** | Python, C, CUDA |
| **Platform** | Linux, macOS |
| **Website** | pytorch.org |

Major difference

| | |
|---|---|
| static graph | dynamic graph |

# Static and Dynamic Graphs

## Static graphs

Define the graph, feed data into the graph, run on its VM and get output.

- Platform independent
- Optimize the graph before running

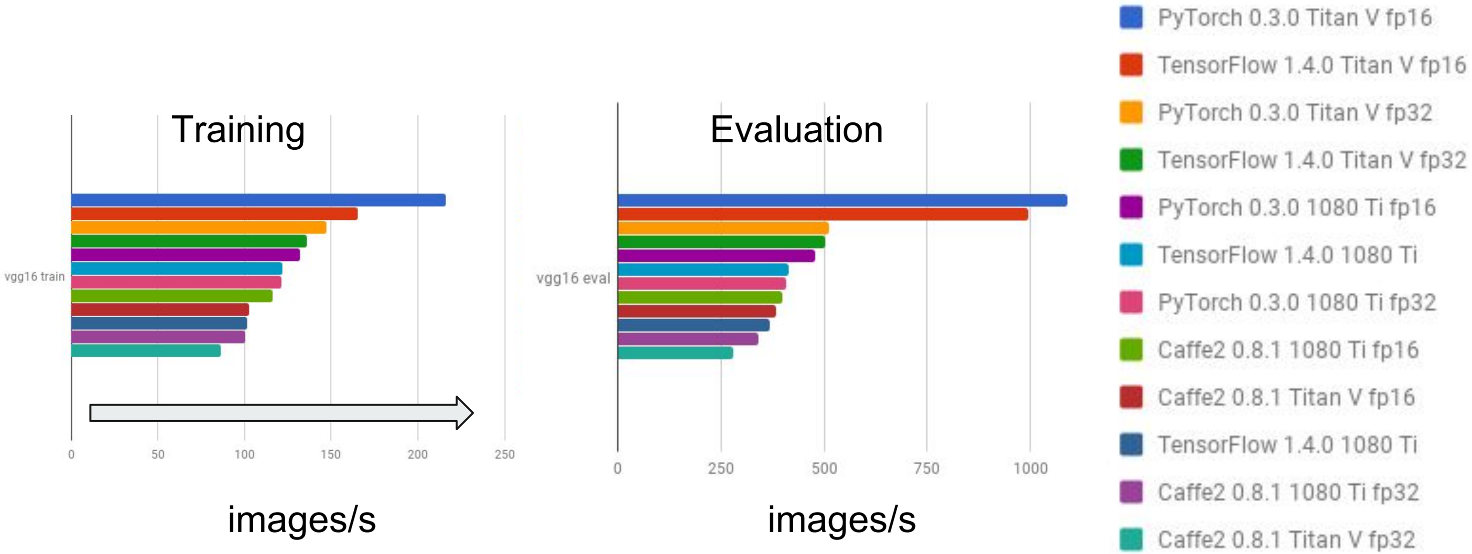
## Dynamic graphs

Build graph while running, remember the graph for back propagation.

- Pythonic control flow
- Rebuild the graph in each run, more flexible

https://github.com/u39kun/deep-learning-benchmark

# A Benchmark on Performance




VGG 16 network

# Pytorch used in this tutorial

Advantages:

Performance
easy for extensions
easy to debug
friendly to beginners

Disadvantages:

No complex number support
Sometimes being unstable and buggy

PYTORCH

http://pytorch.org/

Notebook

https://goo.gl/8Caymh

local: notebooks/nice.ipynb

# Hands on time

Notebook with Solution
https://goo.gl/FhAHRZ

with pytorch:

build a NICE network

solve a sampling problem

★ build a RealNVP network to do it better

A typical application
Accelerate Monte Carlo sampling for $\phi_4$ Model

3:40

# Hands on time

## Experience CUDA acceleration without having Nvidia GPUs

**Open colab notebook**

Edit

- Notebook settings
  - Hardware accelerator
  - Choose GPU

**with pytorch and GPU:**

**build a restricted Boltzmann machine (RBM)**

**write a digits generative model**

**★ recover a broken image**

**RBM as a wave function ansatz for VMC**

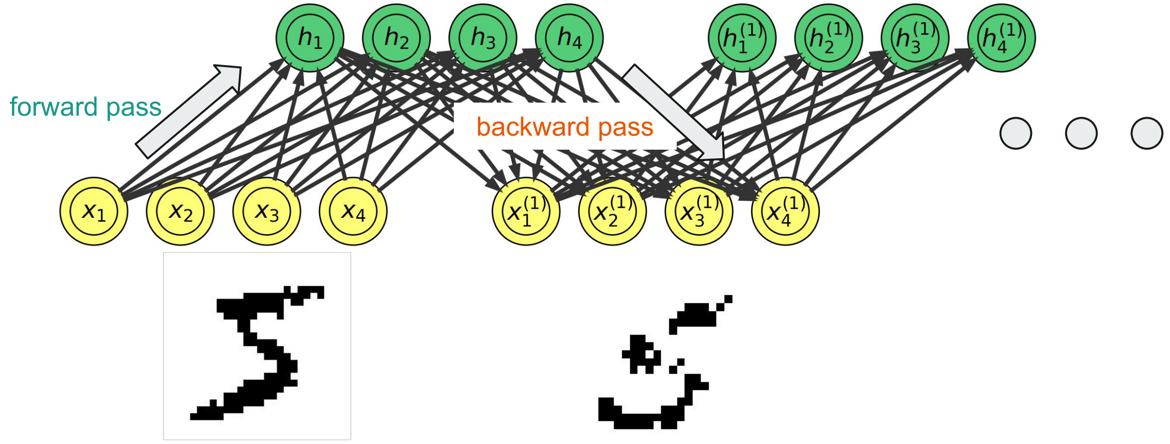
https://goo.gl/d7kPzy

local: notebooks/rbm_generation.ipynb

**Miguel, A., & Hinton, G. E. (n.d.). On Contrastive Divergence Learning, 0.**

$$\frac{\partial \mathcal{L}}{\partial \theta} = \langle \frac{\partial E_\theta(x)}{\partial \theta} \rangle_{x \sim \mathcal{D}} - \langle \frac{\partial E_\theta(x)}{\partial \theta} \rangle_{x \sim p_\theta(x)}$$

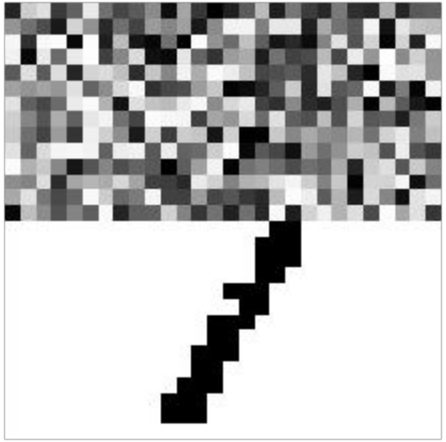
Gibbs sampling process

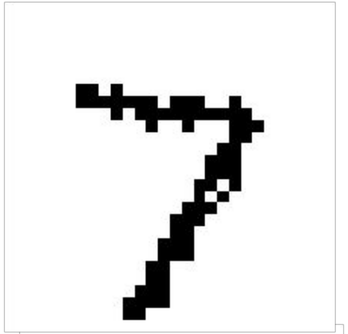



$\sim p_\theta(x)$

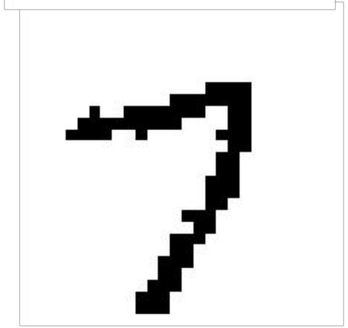
https://goo.gl/VxYYQX

# A Typical Result





Original digit



Generated digit

Takahiro has already introduced it yesterday

# How does VMC Work

Target $\min_{\theta} \frac{\langle \psi_\theta | H | \psi_\theta \rangle}{\langle \psi_\theta | \psi_\theta \rangle}$

Ansatz $\psi(x, \theta)$

knows how to compute probability ratios $p(x'|\theta)/p(x|\theta)$

gradient based optimization, like SGD

Monte Carlo importance sampling

$\{x\}_{\sim p(x|\theta)}$

dataset draw from $p(x|\theta) = |\psi(x,\theta)|^2$

$\langle H \frac{\partial}{\partial\theta} \rangle - \langle H \rangle \langle \frac{\partial}{\partial\theta} \rangle$

"Gradient"

Average local quantities on samples

Calculate gradients with respect to $\theta$

$\langle H \rangle, \langle \frac{\partial}{\partial\theta} \rangle, \langle H \frac{\partial}{\partial\theta} \rangle$

Expectation values of local quantities

$$\Delta_{\text{loc}}(x) = \frac{\partial \log \psi(x,\theta)}{\partial\theta}$$

$$E_{\text{loc}}(x) = \frac{\langle x | H | \psi(\theta) \rangle}{\langle x | \psi(\theta) \rangle}$$

$$\langle H \rangle = \langle E_{\text{loc}} \rangle_{\{x\}}$$

$$\langle \frac{\partial}{\partial\theta} \rangle = \langle \Delta_{\text{loc}} \rangle_{\{x\}}$$

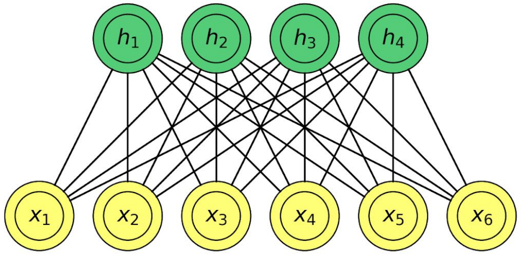
Notebook

Carleo, Giuseppe, and Matthias Troyer. "Solving the quantum many-body problem with artificial neural networks." Science 355.6325 (2017): 602-606.
Cai, Zi, and Jinguo Liu. "Approximating quantum many-body wave functions using artificial neural networks." Physical Review B 97.3 (2018): 035116.

https://goo.gl/vPFtdU

local: notebooks/rbm_ansatz.ipynb

# RBM as a Wave Function Ansatz

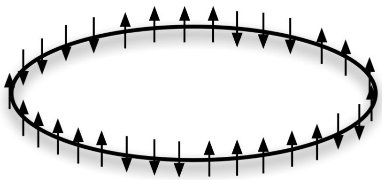



$$x_i = \pm 1$$

$$p(x|\theta) = e^{\sum_i a_i x_i} \prod_{i=1}^{M} 2\cosh(b_i + \textstyle\sum_j W_{ij} x_j)$$

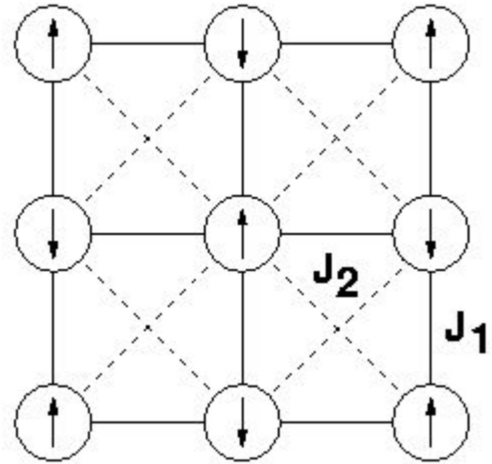
positive

$$p(x) \propto \langle x|\psi\rangle$$



$$H = J \sum_{i=1}^{N} S_i^z S_{i+1}^z - \frac{1}{2}(S_i^+ S_{i+1}^- + S_i^+ S_{i+1}^-)$$

# Ground State for Frustrated Quantum Spin Systems An Open Question yet



# Thanks


**Lei Wang**

Being a level 12 scientist, Lei Wang's brain is filled with crazy ideas.

**Shuo-Hui Li**

Besides being strong at robotics and coding stuff, he has a bottomless stomach.

**Jin-Guo Liu**

Digging and coding are similar, they both produce bugs. Mole hands, coding speed + 50%.

# More Slides

https://goo.gl/6d2sei

# Setup your workplace

**Online notebook**

Google Collaborator + Google Drive

https://colab.research.google.com

https://drive.google.com

## Local Setup

1. setup environment according to https://github.com/GiggleLiu/marburg/
2. clone notebooks to local host

   **$ git clone**

   https://github.com/GiggleLiu/marburg.git
3. run it

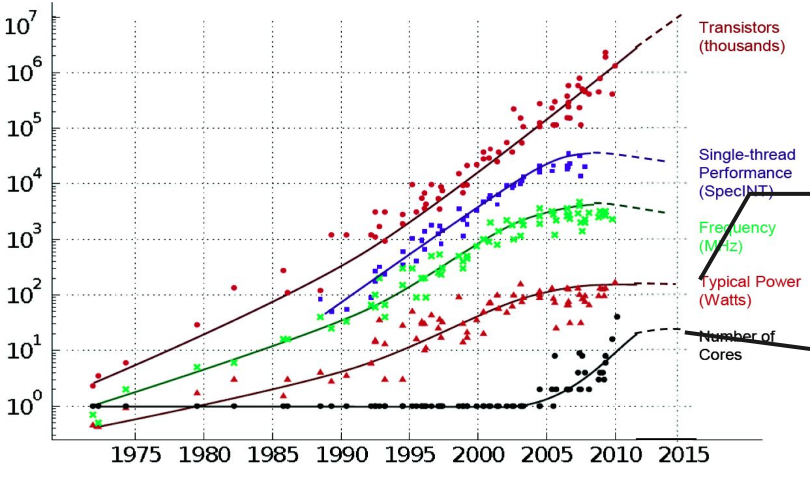
   **$ cd winterchool/notebooks**

   **$ ipython notebook**
4. Open computation_graph.ipynb

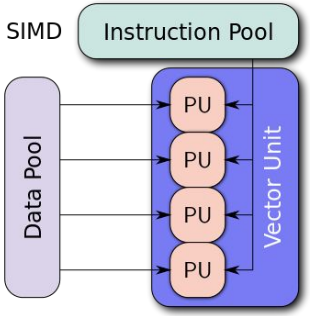
If you cloned this repository or lecture notes days before, pull for updates please!

1. Sign in Google drive
2. Connect Google drive with Google Colaboratory
   a. right click on google drive page
   b. More
   c. Connect more apps
   d. search “Colaboratory” and “CONNECT
3. Open the online notebook link on top right
4. Click “open with colaboratory” on top
5. Save a copy of notebook to your google drive
6. Run/Edit this notebook

# Two Trends in CPU manufacturing




Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore




**Lower Power Consumption**

Intel Core i5-7200U

Ultra-low power

**Parallelism to Increase Computation Speed**

Multi-core CPUs

Single instruction, multiple data (SIMD)

# Parallelism Extremists GPU

GPU Programming?
CUDA programming model (Nvidia)

| | # Cores | Clock Speed | Memory |
|---|---|---|---|
| **CPU** (Intel Core i7-7700k) | 4 (8 threads with hyperthreading) | 4.4 GHz | Shared with system |
| **CPU** (Intel Core i7-6950X) | 10 (20 threads with hyperthreading) | 3.5 GHz | Shared with system |
| **GPU** (NVIDIA Titan Xp) | 3840 | 1.6 GHz | 12 GB GDDR5X |
| **GPU** (NVIDIA GTX 1070) | 1920 | 1.68 GHz | 8 GB GDDR5 |


# Why convex optimization model works

Figure 6 compares SGD with SA.

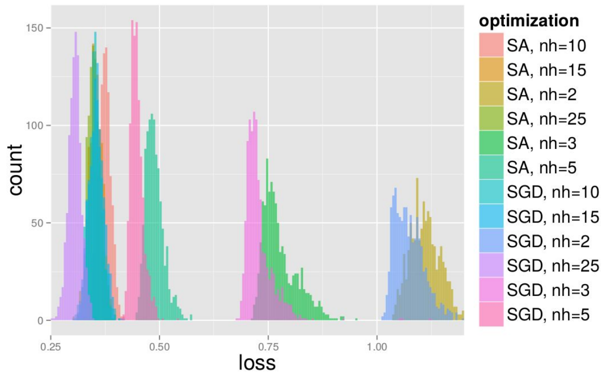



Figure 6: Test loss distributions for SGD and SA for different numbers of hidden units (nh).

- Deeper network: narrower loss distribution
- Global minima: overfit (fit too good for training set, can not be generalized to test set.)