SK hynix i-TAP
반도체 Data Scientist를 위한
ML/DL 심화 커리큘럼

Ernest K. Ryu (류경석)

2020.11.04

커리큘럼 초안

- 1강 CNN architectures
- 2강 data augmentation, regularization
- 3강 network visualization
- 4강 image segmentation
- 5강 adversarial attacks and out-of-distribution samples
- 6강 variational autoencoders
- 7강 generative adversarial networks
- 8강 VAE, GAN 관련 논문/적용 사례 공유
- 9강 Training & generalization of modern overparameterized neural networks
- 10강 Deep-learning based image processing: image denoising and super-resolution
- 11강 DL-based image processing 논문 review
- 12강 model pruning and efficient compressed models
- 13강 reinforcement learning 개론
- 14강 Transfer Learning, Online Learning, AutoML
- 15강 Data Interpretation(XAI)
- 16강 AIOps 관련 논문 Review 및 적용 Case Study

커리큘럼 목적

- Deep Learning의 기초 및 최신동향 공부
  - 요청 주제가 있으면 말씀 해주세요.

- 이론과 함께 code demonstration

- SK hynix data science 팀과 deep learning 활용 탐구

0강. Supervised Learning Foundation

Prerequisite to deep learning with convolutional neural networks:

- Supervised learning setup
- Multi-layer perceptron
- Gradient computation (backpropagation)
- Training via SGD
- GPU computing
- Python and PyTorch programming

We discuss these today.

Traditional Machine Learning

- Theory focus (driven by academic publishing)
- Leveraged domain knowledge
- Predictable performance

Modern Deep Learning (2014~)

- Empirical performance focus (driven by industry academia)
- Leveraged large data and computation; less domain knowledge
- Unpredictable performance and unexpected failures

Modern Deep Learning Formula

Deep learning success = big data + big computation + large (deep) neural network + rapid prototyping

- Big data sometimes comes from internet. Also a matter of focusing on the use of big data. (Example. AlphaGo uses large volume of “data” generated through self-play.)
- Big computation by GPU computing.
- Large (deep) neural network. Architecture design is an art, rather than a science. Based on know-how.
- Rapid prototyping via high-level frameworks: PyTorch, TensorFlow, Caffe2, Chainer, Gluon, Keras, Mxnet

Machine learning = function approximation?

Machine learning is function approximation ≈ Internet is a connection of computers

ML is more than just function approximation, but it is an important fundamental view.

Supervised Learning Setup

Assume there is an unknown function

$$f: \mathcal{X} \to \mathcal{Y}$$

with sample space $\mathcal{X}$ and label space $\mathcal{Y}$.

Ex1) Input image, output numerical label.

Ex2) Input email, output spam / not spam.

Goal: find f.

Supervised Learning Formulation

Issue #1

Search among all functions is intractable.

Solution. Restrict search to a class of parameterized functions f_θ where $\theta \in \mathbb{R}^p$, i.e., only consider functions in $\{f_\theta | \theta \in \Theta\}$ where $\Theta \subseteq \mathbb{R}^p$.

- **Goal: ~~find~~ ~~f~~ find $f_\theta \approx f$, where f_θ is a set of functions parametrized by θ.**

Supervised Learning Formulation

Issue #2
We must define $f_\theta \approx f$.

Define $\text{cost}(y_1, y_2)$ that measures error between labels y_1 and y_2.

If goal is "do well all of the time", then define $g \approx f$ to be:
$\text{cost}\big(g(x), f(x)\big)$ small for all x.

Optimization problem

$$\underset{g \in \{f_\theta | \theta \in \Theta\}}{\text{minimize}} \quad \underset{x \in X}{\text{maximum}} \ \text{cost}(g(x), f(x))$$

equivalent to

$$\underset{\theta \in \Theta}{\text{minimize}} \quad \underset{x \in X}{\text{maximum}} \ \text{cost}(f_\theta(x), f(x))$$

Supervised Learning Formulation

However, maximum is too pessimistic and maximum is computationally harder to work with.

Updated Goal: do well ~~all of the time~~ on average.

$$\underset{\theta \in \Theta}{\text{minimize}} \quad \mathbb{E}_X \text{cost}(f_\theta(X), f(X))$$

- X represents a random (unseen) new data

In ML, "training" is solving an optimization problem.

Supervised Learning Formulation

Issue #3

Expectation over $X \in \mathcal{X}$ difficult to evaluate

Take a finite sample $x_1, \ldots, x_N \in \mathcal{X}$, $y_1, \ldots, y_N \in \mathcal{Y}$ and solve

$$\underset{\theta \in \Theta}{\text{minimize}} \quad \frac{1}{N} \sum_{i=1}^{N} \text{cost}(f_\theta(x_i), y_i)$$

Note $y_i = f(x_i)$

Hopefully, f_θ will provide accurate prediction for new (unseen) data points.

Least-Squares Fitting

$\mathcal{X} = \mathbb{R}^p$, $\mathcal{Y} = \mathbb{R}$ and $\text{cost}(y_1, y_2) = (y_1 - y_2)^2$.

Least-squares problem

$$\underset{\theta \in \Theta}{\text{minimize}} \quad \sum_{i=1}^{N} (f_\theta(x_i) - y_i)^2$$

Restrict f_θ to be linear, i.e., $f_\theta(x_i) = \theta^T x_i$.

$$\underset{\theta \in \mathbb{R}^p}{\text{minimize}} \quad \sum_{i=1}^{N} \left(x_i^T \theta - y_i\right)^2$$

(Because f_θ is linear, also called linear regression.)

Least Squares Fitting

Equivalent to

$$\underset{\theta \in \mathbb{R}^p}{\text{minimize}} \quad \|X\theta - y\|^2$$

Solution is $\theta^{\text{sol}} = (X^T X)^{-1} X^T y$. (When $X^T X$ is not invertible, use pseudo-inverse.)

Linear Classification and Perceptron

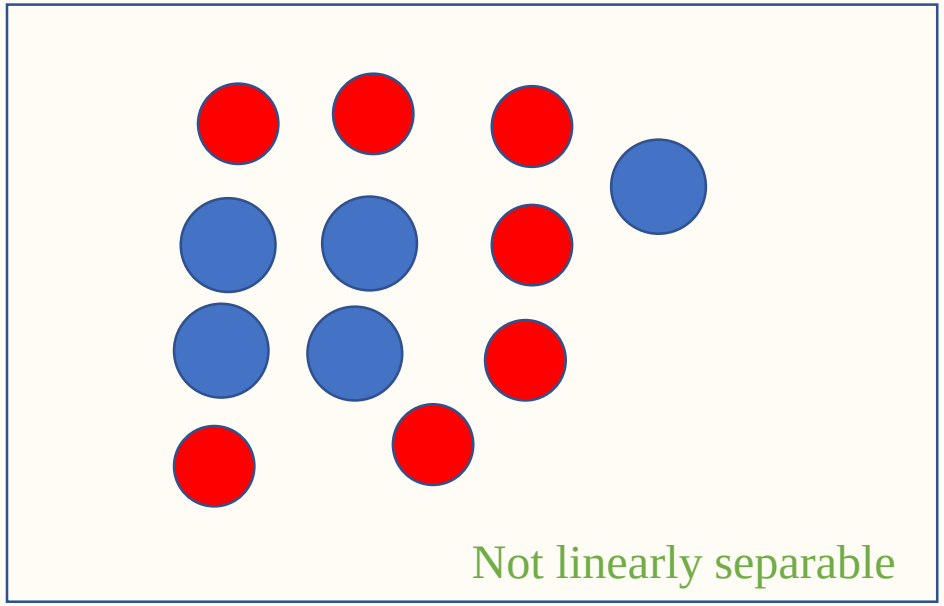
Consider $\mathcal{X} = \mathbb{R}^p$, $\mathcal{Y} = \{-1, +1\}$

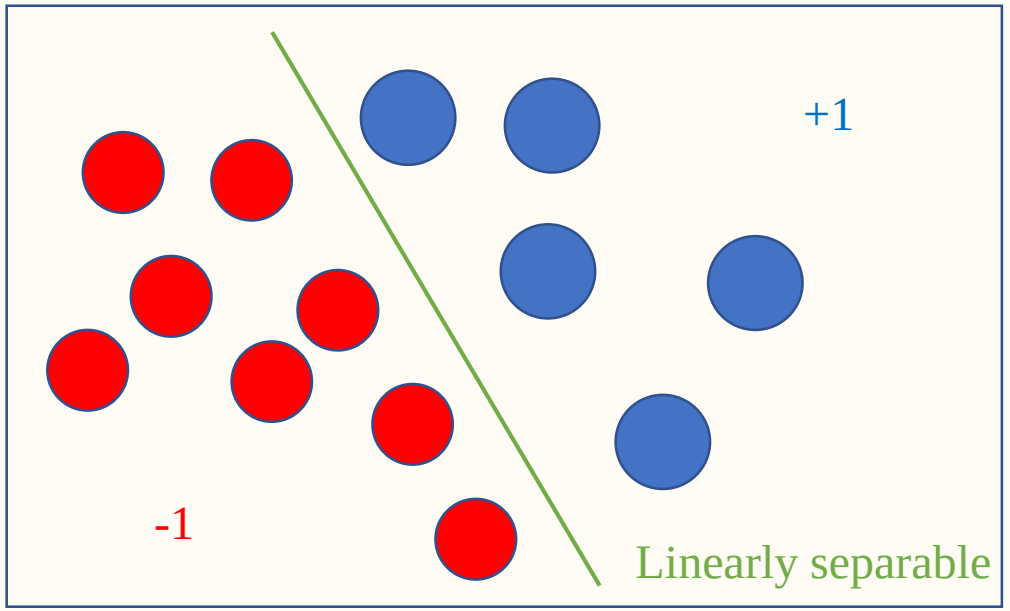
Assume data is linearly separable, i.e., there exists a hyperplane defined by $(a_{\text{true}}, b_{\text{true}})$ such that

$$f(x) = \begin{cases} +1 \text{ if } {a_{\text{true}}}^T x + b_{\text{true}} > 0 \\ -1 \qquad \text{otherwise} \end{cases}$$

Linear Classification and Perceptron

In 2-D space, linear separability looks like:





$f(x) = (a_{true}x + b_{true})$ (Note, ● and ● are length n array, not the x-coordinate)

Linear Classification and Perceptron

Use 0-1 loss function

$$\text{cost}(y_1, y_2) = \begin{cases} 1 & \text{if } y_1 \neq y_2 \\ 0 & \text{if } y_1 = y_2 \end{cases}$$

Consider linear classifiers

$$f_{a,b}(x) = \begin{cases} +1 & \text{if } a^T x + b > 0 \\ -1 & \text{otherwise} \end{cases}$$

Solve

$$\underset{a \in \mathbb{R}^p, b \in \mathbb{R}}{\text{mininimize}} \quad \sum_{i=1}^{N} \text{cost}(f_{a,b}(x_i), y_i)$$

Has solution with optimal value 0 (by linear separation assumption)

Perceptron Algorithm [1]

- $a^0 = 0, b^0 = 0$
- while True:

    if there is (x, y) misclassified:

        pick misclassified (x, y)

    else:

        break

    $a^{k+1} = a^k + yx$

    $b^{k+1} = b^k + y$

(convergence of algorithm can be proved)

Current status of perceptron algorithm:
-not a good algorithm (there are better ones)
-not a good model of biological neurons

[1] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review,*

Improved Supervised Learning Setup

Labels are randomly generated based on data.

Assume there is an unknown function

$$f: \mathcal{X} \to \mathcal{P}(\mathcal{Y})$$

$\mathcal{P}(\mathcal{Y})$ is a probability distribution on the labels $\mathcal{Y}$.

Example. An image may have a 50% chance of being a dog and a 50% chance of being a cat.

Image source: Elsayed, Shreya Shankar, Cheung, Papernot, Kurakin, Goodfellow, Sohl-Dickstein, Adversarial Examples that Fool both Computer Vision and Time-Limited Humans, NeurIPS, 2018.

KL-Divergence

The KL-divergence of $p \in \mathbb{R}^n$ to $q \in \mathbb{R}^n$, where p, q represent probability mass functions ($p_1 + \cdots + p_n = 1$, $p_i \geq 0$ for $i = 1, \ldots, n$) is defined as

$$D_{\mathrm{KL}}(\mathrm{p}||\mathrm{q}) = \sum_{i=1}^{n} p_i \log(p_i/q_i) = \underbrace{-\sum_{i=1}^{n} p_i \log q_i}_{=H(p,q)} + \underbrace{\sum_{i=1}^{n} p_i \log p_i}_{=H(p)}$$

$=H(p,q)$: cross entropy of p relative to q

$=H(p)$: entropy of p

Properties

1. Not symmetric, i.e., $D_{\mathrm{KL}}(\mathrm{p}||\mathrm{q}) \neq D_{\mathrm{KL}}(\mathrm{q}||\mathrm{p})$
2. $D_{\mathrm{KL}}(\mathrm{p}||\mathrm{q}) > 0$ if $p \neq q$ and $D_{\mathrm{KL}}(\mathrm{p}||\mathrm{q}) = 0$ if $p = q$
3. $D_{\mathrm{KL}}(\mathrm{p}||\mathrm{q}) = \infty$ possible.

KL-Divergence

KL-divergence, entropy, cross entropy originate from information theory, but this information theoretic meaning is not used in ML often.

We use KL-divergence as a distance measure between p and q.

Cross entropy is used as a formula.

Logistic Regression

Key 1: Choose model

$$f_{w,b}(x) = \frac{1}{1+e^{w^T x+b}}\begin{bmatrix}1\\ e^{w^T x+b}\end{bmatrix} = \begin{bmatrix}\frac{1}{1+e^{w^T x+b}}\\ \frac{1}{1+e^{-(w^T x+b)}}\end{bmatrix}$$

(Output of $f_{w,b}(x)$ is probability on labels $y=-1$ and $y=1$.)

Define "empirical distribution"

$$\mathcal{P}(y) = \begin{cases}\begin{bmatrix}1\\0\end{bmatrix} & \text{if } y=-1\\ \begin{bmatrix}0\\1\end{bmatrix} & \text{if } y=+1\end{cases}$$

Logistic Regression

Key 2: minimize average KL-divergence from the empirical distribution $\mathcal{P}(y_i)$ to model $f_{w,b}(x_i)$

$$\underset{w\in\mathbb{R}^p, b\in\mathbb{R}}{\text{minimize}} \quad \sum_{i=1}^{N} D_{\mathrm{KL}}(\mathcal{P}(y_i)||f_{w,b}(x_i))$$

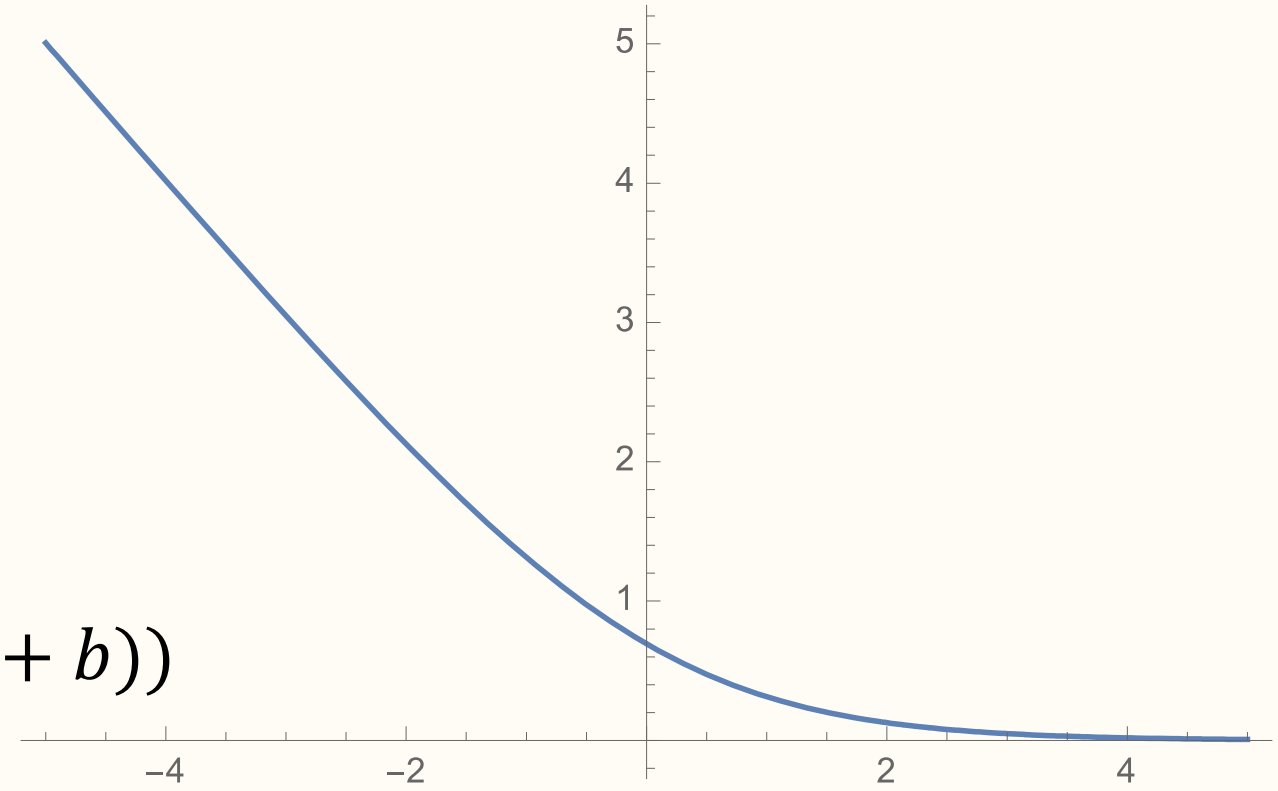
which is equivalent to

$$\underset{w\in\mathbb{R}^p, b\in\mathbb{R}}{\text{minimize}} \quad \sum_{i=1}^{N} \log(1+\exp(-y_i(w^T x_i + b)) + (\text{terms independent of } w, b)$$

Logistic Regression

Define logistic loss

$$\ell(z) = \log(1 + e^{-z})$$



Then we solve

$$\underset{w \in \mathbb{R}^p, b \in \mathbb{R}}{\text{mininimize}} \quad \sum_{i=1}^{N} \ell(y_i(w^T x_i + b))$$

Logistic Regression PyTorch demonstration

Generalize Logistic Regression

- Generalize key 2: Losses other than cross-entropy can be used. [1]

- Generalize key 1: Use model $f_{w,b}$ with more depth.

[1] Hui, Belkin. Evaluation of Neural Architectures Trained with Square Loss vs Cross-Entropy in Classification Tasks, arXiv, 2020.

Multi-layer perceptron PyTorch demonstration

Computing derivatives

Programming

$$f(x,y,z) = \sin\left(\frac{\cosh\left(y^2 + \frac{x}{z}\right) + \tanh(xyz)}{\log(1+e^x)}\right)$$

in Python is easy.

Programming $\frac{\partial}{\partial x} f(x,y,z)$ would be time-consuming and error prone, if done manually.

Automatic Differentiation (Backpropagation)

Automates gradient computation! Only need to specify how to evaluate function.

Gradient costs roughly 5 × computation cost* of evaluating function.

AutoDiff is not
- Finite differencing
- Symbolic differentiation.

AutoDiff ≈ chain rule of vector calculus

* Depends on computational structure of function, for 5X difference is mostly true for functions (neural networks) used in machine learning.

Automatic Differentiation (Backpropagation)

Backprop is one version of AutoDiff, and it is the most commonly used AutoDiff in ML.

One can use Backprop in PyTorch without understanding it, but there are cases where a shallow level of AutoDiff is helpful. (Example. 3. network visualization and 5. adversarial attacks)

I can explain backprop further upon request.

Training with SGD

To solve

$$\underset{\theta}{\text{minimie}} \sum_{i=1}^{N} h(\theta; x_i, y_i)$$

use stochastic gradient descent (SGD)

$$\theta^{k+1} = \theta^k - \alpha_k \nabla_\theta h\left(\theta^k; x_{i(k)}, y_{i(k)}\right)$$

where $i(k)$ is a randomly chosen index.

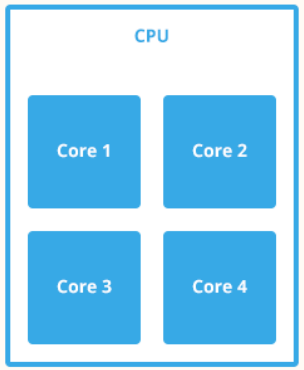
Training with SGD

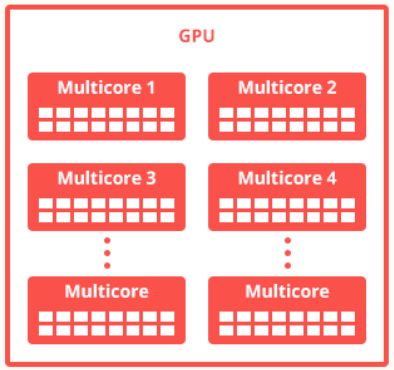
There are many variants of SGD, and they are referred to as “optimizers” in PyTorch.

One can use Optimizers in PyTorch without understanding them, but there are cases where a shallow level of understanding can be helpful.

I can explain optimizers further upon request.

GPU as Parallel Processors





- Rendering graphics involves computing many small tasks in parallel. Graphics cards provide many small processors to render graphics.
- In 1999, Nvidia released GeForce 256 and introduced programmability in the form of vertex and pixel shaders. First graphics card to be named a 'Graphical Processing Unit (GPU)'.
- Researchers quickly learned how to implement linear algebra by mapping matrix data into textures and applying shaders.

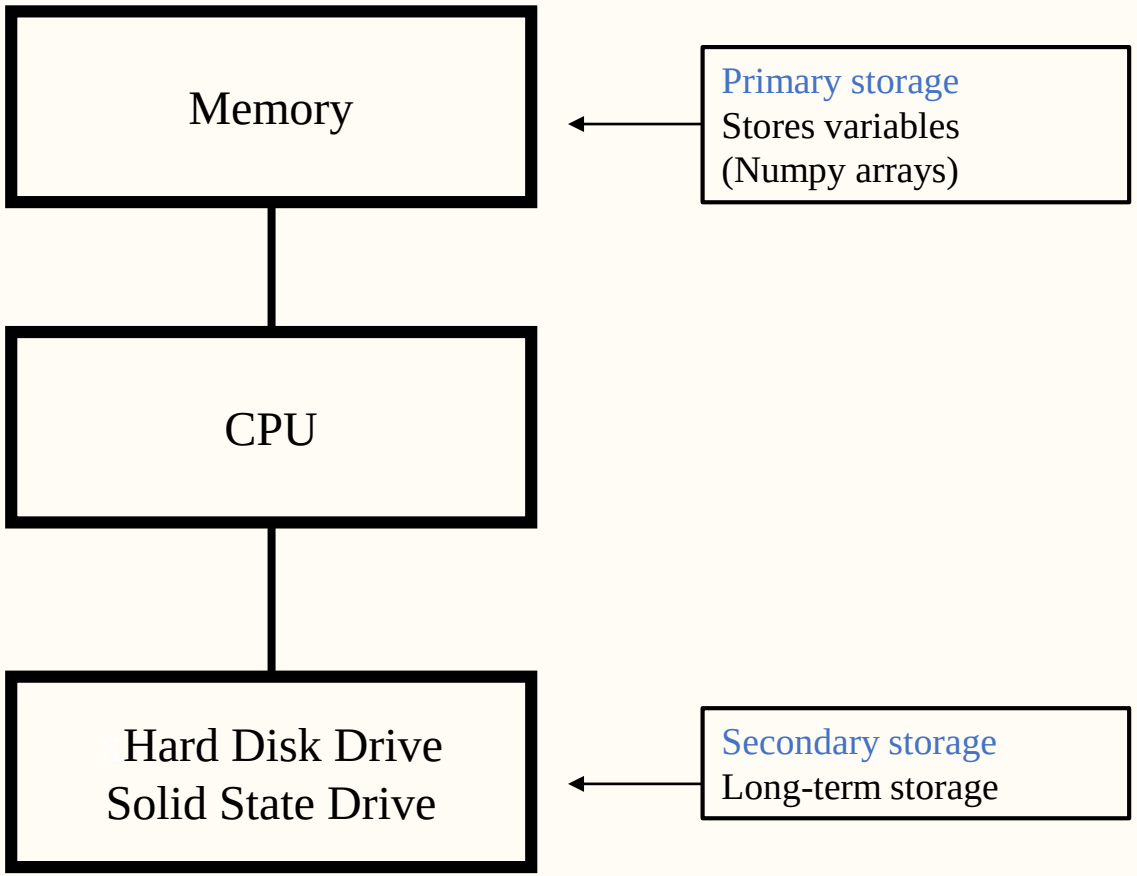
General Purpose GPUs (GPGPU)

- In 2007, Nvidia released Compute Unified Device Architecture (CUDA), which enabled general purpose computing on a CUDA-enabled GPUs.
- Unlike CPUs which provide fast serial processing, GPUs provide massive parallel computing with its numerous slower processors.
- The 2008 financial crisis hit Nvidia very hard since GPUs were luxury items used for games. This encouraged Nvidia to invest further in GPGPUs and create a more stable consumer base.

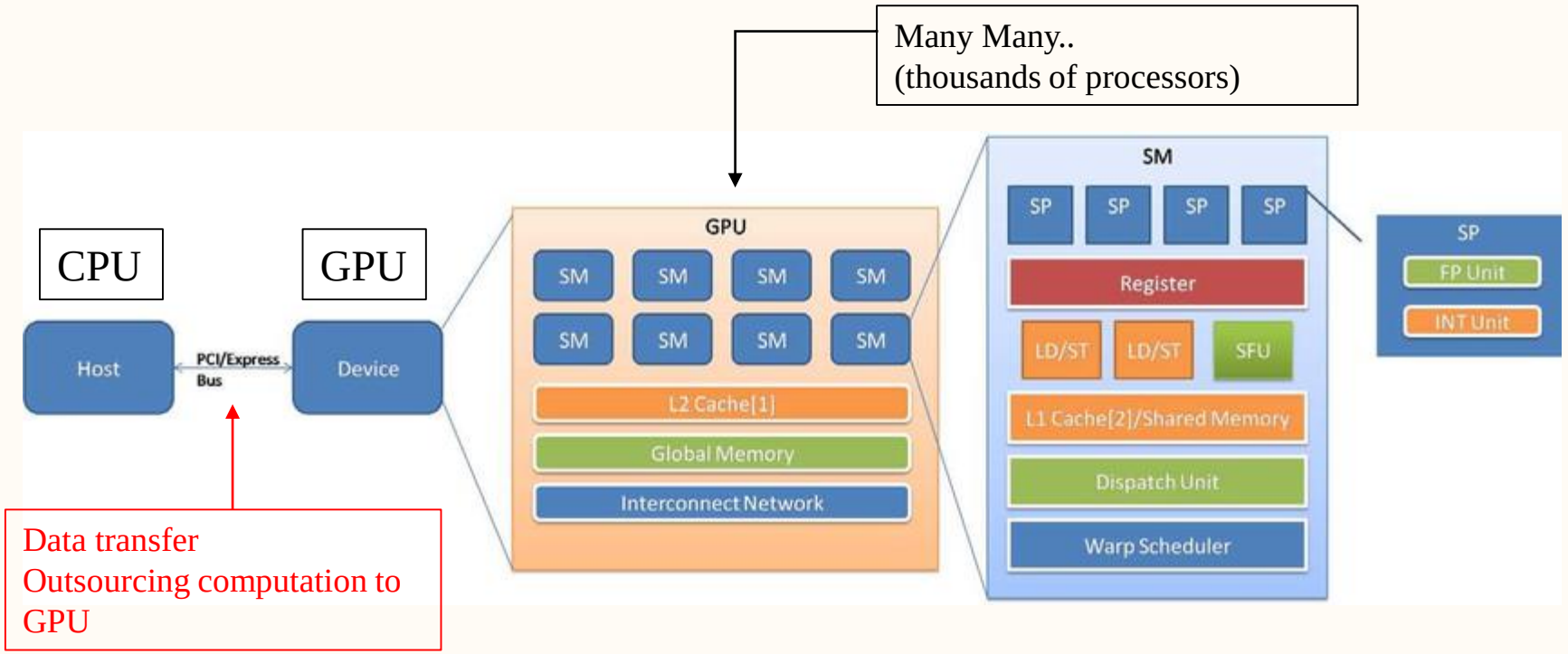
GPUs in Machine Learning

- The 2009 paper "Large-scale Deep Unsupervised Learning using Graphics Processors" by Raina, Madhavan, and Andrew Ng demonstrated that GPUs can be used to train large neural networks.
- (This was not the first to use GPUs in machine learning, but it was one of the most influential.)
- The basis of machine learning is large data and computation. GPUs provided the computational power.

CPU Computing Model



GPU Computing Model



GPU Computing Model

Computing

$$x^{100} = A^{100}x^0$$

with a GPU involves code resembling the following.

```
send A from host (CPU) to device (GPU)
send x=x0 from host (CPU) to device (GPU)
for _ in range(100):
  tell GPU to compute x=A*x
send x from device (GPU) to host (CPU)
```

In this example and deep learning, GPU accelerates computation since:

- Amount of computation ≫ data communication.
- Large data resides in the GPU, and CPU issues commands to perform computation on the data. (A in this example, neural network architecture in deep learning.)