MACHINE LEARNING

ERNEST YEUNG ERNESTYALUMNI@GMAIL.COM

Contents

Abstract. Everything about Machine Learning.

Part 1. Data; Data Wrangling, Data cleaning, Web crawling, Data input

1. Sample, example data; input data

1.1. **sklearn, from sci-kit learn, sample data, datasets.** cf. `sampleinputdataX_sklearn.ipynb`
For $j = 0, 1, \dots d-1$, $d =$ number of "features",

$$x_i^{(j)} \in (\mathbb{R}^N)^d = \underbrace{\mathbb{R}^N \times \mathbb{R}^N \times \cdots \times \mathbb{R}^N}_{d}$$

e.g. $N = 442$ (number of given observations/data)
$y_i \in \mathbb{R}^N$ (represents target or result)
Given data $(x_i^{(j)}, y_i) \in (\mathbb{R}^N)^d \times \mathbb{R}^N$,
we can restrict data $(x_i^{(j)}, y_i)$ to subsets to train and test, for training and testing.
So let $I_{\text{train}}, I_{\text{test}} \subset \{0, 1, \dots N-1\}$ s.t. $I_{\text{train}} \cap I_{\text{test}} = \emptyset$.

Date: 24 avril 2016.
Key words and phrases. Machine Learning, statistical inference, statistical inference learning.
[1]nlab FinSet https://ncatlab.org/nlab/show/FinSet

Want:

$$(x_i^{(j)}, y_i)_{i \in I_{\text{train}}} \mapsto \theta_\alpha$$
$$(\mathbb{R}^{|I_{\text{train}}|})^d \times \mathbb{R}^{|I_{\text{train}}|} \to \mathbb{R}^{|d|}$$

and so further, I think the idea is

$$(x_i^{(j)}, y_i)_{i \in I_{\text{test}}} \xrightarrow{L_{\theta_\alpha}} L_{\theta_\alpha}(\theta_\alpha(x_i^{(j)}, y_i))$$
$$(\mathbb{R}^{|I_{\text{test}}|})^d \times \mathbb{R}^{|I_{\text{test}}|} \to \mathbb{R}$$

Part 2. Introduction

1.1.1. *Terminology.*
inputs ≡ independent variables ≡ predictors (cf. statistics) ≡ features (cf. pattern recognition)
outputs ≡ dependent variables ≡ responses
cf. Chapter 2 Overview of Supervised Learning, Section 2.1 Introduction of Hastie, Tibshirani, and Friedman (2009) [1]
cf. Chapter 2 Overview of Supervised Learning, Section 2.2 Variable Types and Terminology of Hastie, Tibshirani, and Friedman (2009) [1]

1.1.2. *FinSet.*
The category FinSet $\in$ Cat is the category of all finite sets (i.e. Obj(FinSet) $\equiv$ all finite sets) and all functions in between them; note that FinSet $\subset$ Set [1]

Recall that the FinSet *skeletal* is

1.2. **Supervised Learning.** cf. http://cs229.stanford.edu/notes/cs229-notes1.pdf

Consider data to belong to the category of all possible data:

$$\text{Data} \equiv \text{Dat} = (\text{Obj}(\text{Dat}), \text{MorDat}, 1, \circ), \qquad \text{Dat} \in \text{Cat}$$

Consider the **training set**:

$$\text{training set} := \{(x^{(i)}, y^{(i)}) | i = 1 \dots m, x^{(i)} \in \mathcal{X}, y^{(i)} \in \mathcal{Y}\}$$

where $\mathcal{X}$ is a manifold (it can be topological or smooth, EY:20160502 I don't know exactly because I need to check the topological and/or differential structure); $\mathcal{Y} \in \text{Obj}(\text{FinSet})$, or ($\mathcal{Y} \in \text{Obj}(\text{Top})$(or $\mathcal{Y} \in \text{Obj}(\text{Man})$)).

So training set $\subset \mathcal{X} \times \mathcal{Y} \in \text{Obj}(\text{Dat})$.

I propose that there should be a functor H that represents the "learning algorithm":

$$\text{Dat} \xrightarrow{H} \text{ML}$$

s.t.

$$H : \mathcal{X} \times \mathcal{Y} \to \text{Hom}(\mathcal{X}, \mathcal{Y})$$
$$H(\text{training set}) = H(\{(x^{(i)}, y^{(i)}) | i = 1 \dots m\}) = h$$

When $\mathcal{Y} \in \text{Obj}(\text{FinSet})$, *classification.*
When $\mathcal{Y} \in \text{Obj}(\text{Top})$ (or Obj(Man)), *regression.*

1.2.1. *Linear Regression.* Keeping in mind

$$\text{Dat} \xrightarrow{H} \text{ML}$$

Consider

$$h : \mathbb{R}^p \to \text{Hom}(\mathcal{X}, \mathcal{Y})$$
$$h : \theta \mapsto h_\theta$$

s.t.

$$h_\theta : \mathcal{X} \to \mathcal{Y}$$

so (possibly) $h \in \text{Obj}ML$ (or is h part of the functor H?)

Consider the cost function J

$$J : \mathbb{R}^p \to \text{Hom}(\mathfrak{X} \times \mathfrak{Y}, \mathbb{R}) = C^\infty(\mathcal{X} \times \mathcal{Y})$$
$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

1.2.2. *LMS algorithm (least mean square (or Widrow-Hoff learning rule)).* Define **gradient descent** algorithm:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

with := being assignment (I'll use := for "define", in mathematical terms, use context to distinguish the 2), where α is the *learning rate.*

Rewriting the above,

$$\theta := \theta - \alpha \text{grad} J(\theta)$$

where grad : $C^\infty(M) \to \mathfrak{X}(M)$, with M being a smooth manifold.

This is *batch gradient descent*:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) = \theta_j - \alpha \frac{\partial}{\partial \theta_j} \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 = \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \left(\frac{\partial h_\theta(x^{(i)})}{\partial \theta} \right)$$

Simply notice how the entire training set of m rows is used.

I will expound on the so-called distinguished object $1 \xrightarrow{P} X$ on pp. 8, in Section 2 The Category of Conditional Probabilities of Culbertson and Sturtz (2013) [2] because it wasn't clear to me in the first place (the fault is mine; the authors wrote a very lucid and very fathomable, pedagogically-friendly exposition).

$\forall Y$ with indiscrete σ-algebra $\Sigma_Y = \{Y, \emptyset\}$
(remember, $((Y, \Sigma_Y), \mu_Y)$, $\mu_Y(\phi) = 0$, $\mu_Y(Y) = 1$),

$\exists!$ unique morphism in Mor$\mathcal{P}$, $X \to Y$, since
$\forall P : X \to Y$, $P \in \text{Mor}\mathcal{P}$, P_x must be a probability measure on Y, because

$$(X, \Sigma_X) \xrightarrow{P} (Y, \Sigma_Y)$$
$$P : \Sigma_Y \times X \to [0, 1]$$
$$P(\cdot|x) : \Sigma_Y \to [0, 1] \equiv \begin{array}{l} P_x : \Sigma_Y \to [0, 1] \text{ s.t.} \\ P_x(\emptyset) = 0,\ P_x(Y) = 1 \end{array}$$

i.e. EY: 20160503, Given $x \in X$ occurs, Y must occur.

By def. of terminal object ($\forall (X, \Sigma_X) \in \text{Obj}\mathcal{P}$, $\exists!$ morphism P s.t. $(X, \Sigma_X) \xrightarrow{P} (Y, \Sigma_Y)$, Y *terminal* object, and denote unique morphism $!_X : X \to Y$, $!_X \in \text{Mor}\mathcal{P}$.

Up to isomorphism, canonical terminal object is 1-element set denoted by $1 = \{*\}$, with the only possible σ-algebra ($\mu(*) = 1$, $\mu(\emptyset) = 0$),

$$\forall P : 1 \to X,\ P \in \text{Mor}\mathcal{P},\ P \in \text{Hom}_{\mathcal{P}}(1, X),\ \forall X \in \text{Mor}\mathcal{P}$$

P is an "absolute" probability measure on X because "there's no variability (conditioning) possible within singleton set $1 = \{*\}$." [2]

Now

$$P : \Sigma_X \times 1 \to [0, 1]$$
$$P(\cdot|*) : \Sigma_X \to [0, 1]$$

where $P(\cdot|*) : \Sigma_X \to [0, 1]$ perfect probability measure on X, $P(\cdot|*) : \Sigma_X \to [0, 1] \equiv P_*$, i.e. $P(\cdot|*) = p(\cdot)$ (usual probability on X).

$\forall A \in \Sigma_X$, $P(A|\cdot) : 1 \to [0, 1]$, but $P(A|*) = P(A)$, $P(A|\emptyset) = 0$.

Refer to

$$1 \xrightarrow{P} X$$

morphism $P : 1 \to X \in \text{Mor}\mathcal{P}$ as probability measure or distribution on X.

2. Deep Learning

Deep Learning Tutorial [6]

3. Parallel Computing

3.1. **Udacity Intro to Parallel Programming : Lesson 1 - The GPU Programming Model.** Owens and Luebki pound fists at the end of this video. =)))) Intro to the class.

3.1.1. *Running CUDA locally.* Also, Intro to the class, in Lesson 1 - The GPU Programming Model, has links to documentation for running CUDA locally; in particular, for Linux: http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/index.html. That guide told me to go download the NVIDIA CUDA Toolkit, which is the https://developer.nvidia.com/cuda-downloads.

For *Fedora*, I chose Installer Type `runfile (local)`.

Afterwards, installation of CUDA on Fedora 23 workstation had been nontrivial. Go see either my github repository ML-grabbag (which will be updated) or my wordpress blog (which may not be upgraded frequently).

$P = VI = I^2R$ heating.

3.1.2. *Definitions of Latency and throughput (or bandwidth).* cf. Building a Power Efficient Processor

Latency vs Bandwidth

latency [sec]. From the title "Latency vs. bandwidth", I'm thinking that throughput = bandwidth (???). throughput = job/time (of job).

Given total task, velocity v,

total task $/v$ = latency. throughput = latency/(jobs per total task).

Also, in Building a Power Efficient Processor. Owens recommends the article David Patterson, "Latency..."

cf. GPU from the Point of View of the Developer

$n_{\text{core}} \equiv$ number of cores

$n_{\text{vecop}} \equiv (n_{\text{vecop}}-$wide axial vector operations/*core* core)

$n_{\text{thread}} \equiv$ threads/core (hyperthreading)

$$n_{\text{core}} \cdot n_{\text{vecop}} \cdot n_{\text{thread}} \text{ parallelism}$$

There were various websites that I looked up to try to find out the capabilities of my video card, but so far, I've only found these commands (and I'll print out the resulting output):

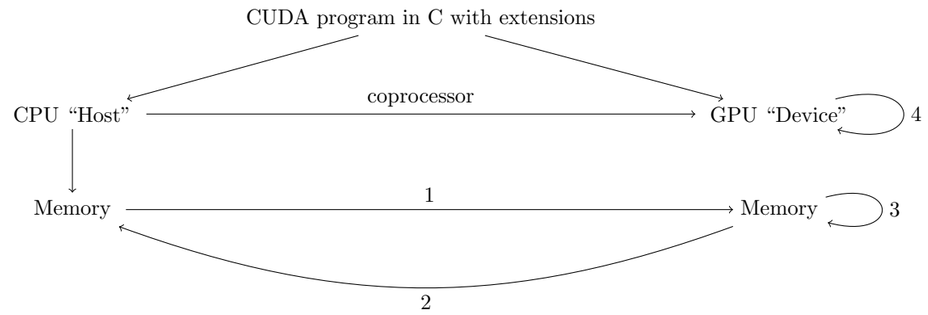
```
$ lspci -vnn | grep VGA -A 12
03:00.0 VGA compatible controller [0300]: NVIDIA Corporation GM200 [GeForce GTX 980 Ti] [10de:17c8] (rev a1) (prog-if 00 [VGA controller])
        Subsystem: eVga.com. Corp. Device [3842:3994]
        Physical Slot: 4
        Flags: bus master, fast devsel, latency 0, IRQ 50
        Memory at fa000000 (32-bit, non-prefetchable) [size=16M]
        Memory at e0000000 (64-bit, prefetchable) [size=256M]
        Memory at f0000000 (64-bit, prefetchable) [size=32M]
        I/O ports at e000 [size=128]
        [virtual] Expansion ROM at fb000000 [disabled] [size=512K]
        Capabilities: <access denied>
        Kernel driver in use: nvidia
        Kernel modules: nouveau, nvidia

$ lspci | grep VGA -E
03:00.0 VGA compatible controller: NVIDIA Corporation GM200 [GeForce GTX 980 Ti] (rev a1)

$ grep driver /var/log/Xorg.0.log
[    18.074] Kernel command line: BOOT_IMAGE=/vmlinuz-4.2.3-300.fc23.x86_64 root=/dev/mapper/fedora-root ro rd.lvm.lv=fedora/root rd.lvm.lv=fedora/swap rhgb quiet LANG=en_US.UTF-8 nouveau.modeset=0 rd.driver.blacklist=nouveau nomodeset gfxpay
[    18.087] (WW) Hotplugging is on, devices using drivers 'kbd', 'mouse' or 'vmmouse' will be disabled.
[    18.087]    X.Org XInput driver : 22.1
[    18.192] (II) Loading /usr/lib64/xorg/modules/drivers/nvidia_drv.so
[    19.088] (II) NVIDIA(GPU-0): Found DRM driver nvidia-drm (20150116)
[    19.102] (II) NVIDIA(0):     ACPI event daemon is available, the NVIDIA X driver will
[    19.174] (II) NVIDIA(0): [DRI2]   VDPAU driver: nvidia
[    19.284]    ABI class: X.Org XInput driver, version 22.1
...

$ lspci -k | grep -A 8 VGA
03:00.0 VGA compatible controller: NVIDIA Corporation GM200 [GeForce GTX 980 Ti] (rev a1)
        Subsystem: eVga.com. Corp. Device 3994
        Kernel driver in use: nvidia
        Kernel modules: nouveau, nvidia
03:00.1 Audio device: NVIDIA Corporation GM200 High Definition Audio (rev a1)
        Subsystem: eVga.com. Corp. Device 3994
        Kernel driver in use: snd_hda_intel
        Kernel modules: snd_hda_intel
05:00.0 USB controller: VIA Technologies, Inc. VL805 USB 3.0 Host Controller (rev 01)
```

CUDA Program Diagram



CPU "host" is the boss (and issues commands) -Owen.

Coprocessor : CPU "host" $\rightarrow$ GPU "device"

Coprocessor : CPU process $\mapsto$ (co)-process out to GPU

With

1 data cpu $\rightarrow$ gpu

2 data gpu $\rightarrow$ cpu (initiated by cpu host)

1., 2., uses `cudaMemcpy`

3 allocate GPU memory: `cudaMalloc`

4 launch kernel on GPU

Remember that for 4., this launching of the kernel, while it's acting on GPU "device" onto itself, it's initiated by the boss, the CPU "host".

Hence, cf. Quiz: What Can GPU Do in CUDA, GPUs can respond to CPU request to receive and send Data CPU $\rightarrow$ GPU and Data GPU $\rightarrow$ CPU, respectively (1,2, respectively), and compute a kernel launched by the CPU (3).

A CUDA Program A typical GPU program

- `cudaMalloc` - CPU allocates storage on GPU
- `cudaMemcpy` - CPU copies input data from CPU $\rightarrow$ GPU
- *kernel launch* - CPU launches kernel(s) on GPU to process the data
- `cudaMemcpy` - CPU copies results back to CPU from GPU

Owens advises minimizing "communication" as much as possible (e.g. the `cudaMemcpy` between CPU and GPU), and do a lot of computation in the CPU and GPU, each separately.

Defining the GPU Computation

Owens circled this

BIG IDEA	This is Important

Kernels look like serial programs

Write your program as if it will run on **one** thread

The GPU will run that program on **many** threads

Squaring A Number on the CPU

Note

(1) Only 1 thread of execution: ("thread" := one independent path of execution through the code) e.g. the `for` loop
(2) no explicit parallelism; it's serial code e.g. the `for` loop through 64 elements in an array

GPU Code A High Level View

CPU:

- Allocate Memory
- Copy Data to/from GPU
- Launch Kernel - species degree of parallelism

GPU:

- Express Out = In · In - says *nothing* about the degree of parallelism

Owens reiterates that in the GPU, everything looks serial, but it's only in the CPU that anything parallel is specified.
pseudocode: CPU code: square kernel <<< 64 >>> (outArray,inArray)

Squaring Numbers Using CUDA Part 3

From the example

```
// launch the kernel
square<<<1, ARRAY_SIZE>>>(d_out, d_in)
```

we're introduced to the "CUDA launch operator", initiating a kernel of 1 block of 64 elements (`ARRAY_SIZE` is 64) on the GPU. Remember that `d_` prefix (this is naming comvention) tells us it's on the device, the GPU, solely.

With CUDA launch operator $\equiv$<<<>>>, then also looking at this explanation on `stackexchange` (so surely others are confused as well, of those who are learning this (cf. CUDA kernel launch parameters explained right?). From Eric's answer,

threads are grouped into blocks. all the threads will execute the invoked kernel function.

Certainly,

$$<<<>>>: (n_{\text{block}}, n_{\text{threads}}) \times \text{kernelfunctions} \mapsto \text{kernelfunction} <<< n_{\text{block}}, n_{\text{threads}} >>> \in \text{End} : \text{Dat}_{\text{GPU}}$$

$$<<<>>>: \mathbb{N}^+ \times \mathbb{N}^+ \times \text{Mor}_{\text{GPU}} \to \text{EndDat}_{\text{GPU}}$$

where I propose that GPU can be modeled as a category containing objects Dat_{GPU}, the collection of all possible data inputs and outputs into the GPU, and Mor_{GPU}, the collection of all kernel functions that run (exclusively, and this *must* be the class, as reiterated by Prof. Owen) on the GPU.

Next,

$$\text{kernelfunction} <<< n_{\text{block}}, n_{\text{threads}} >>>: \text{din} \mapsto \text{dout} \qquad \text{(as given in the "square" example, and so I propose)}$$

$$\text{kernelfunction} <<< n_{\text{block}}, n_{\text{threads}} >>>: (\mathbb{N}^+)^{n_{\text{threads}}} \to (\mathbb{N}^+)^{n_{\text{threads}}}$$

But keep in mind that dout, din are pointers in the C program, pointers to the place in the memory.

`cudaMemcopy` is a functor category, s.t. e.g. $\text{Obj}_{\text{CudaMemcopy}} \ni$ cudaMemcpyDevicetoHost where

$$\text{cudaMemcopy}(-,-,n_{\text{thread}}, \text{cudaMemcpyDeviceToHost}) : \text{Memory}_{\text{GPU}} \to \text{Memory}_{\text{CPU}} \in \text{Hom}(\text{Memory}_{\text{GPU}}, \text{Memory}_{\text{CPU}})$$

Squaring Numbers Using CUDA 4

Note the C language construct *declaration specifier* - denotes that this is a kernel (for the GPU) and not CPU code. Pointers need to be allocated on the GPU (otherwise your program will crash spectacularly -Prof. Owen).

3.1.3. *What are C pointers?* Is $\langle$ type $\rangle *$, a pointer, then a mapping from the category, namely the objects of types, to a mapping from the specified value type to a memory address?

e.g.

$$\langle\rangle * : \text{float} \mapsto \text{float} *$$

$$\text{float} * : \text{din} \mapsto \text{ some memory address}$$

and then we pass in mappings, not values, and so we're actually declaring a square *functor*.

What is `threadIdx`? What is it mathematically? Consider that $\exists$ 3 "modules":

$$\text{threadIdx}.x$$
$$\text{threadIdx}.y$$
$$\text{threadIdx}.z$$

And then the line

```
int idx = threadIdx.x;
```

says that idx is an integer, "declares" it to be so, and then assigns idx to threadIdx.x which surely has to also have the same type, integer. So (perhaps)

$$idx \equiv \text{threadIdx}.x \in \mathbb{Z}$$

is the same thing.

Then suppose threadIdx $\subset$ FinSet, a subcategory of the category of all (possible) finite sets, s.t. threadIdx has 3 particular morphisms, $x, y, z \in \text{Mor}\mathit{threadIdx}$,

$$x : \text{threadIdx} \mapsto \text{threadIdx}.x \in \text{Obj}_{\text{FinSet}}$$
$$y : \text{threadIdx} \mapsto \text{threadIdx}.x \in \text{Obj}_{\text{FinSet}}$$
$$z : \text{threadIdx} \mapsto \text{threadIdx}.x \in \text{Obj}_{\text{FinSet}}$$

Configuring the Kernel Launch Parameters Part 1

$n_{\text{blocks}}, n_{\text{threads}}$ with $n_{\text{threads}} \geq 1024$ (this maximum constant is GPU dependent). You should pick the $(n_{\text{blocks}}, n_{\text{threads}})$ that makes sense for your problem, says Prof. Owen.

3.1.4. *Memory layout of blocks and threads.* $\forall (n_{\text{blocks}}, n_{\text{threads}}) \in \mathbb{Z} \times \{1 \dots 1024\}$, $\{1 \dots n_{\text{block}} \times \{1 \dots n_{\text{threads}}\}$ is now an ordered index (with lexicographical ordering). This is just 1-dimensional (so possibly there's a 1-to-1 mapping to a finite subset of $\mathbb{Z}$).

I propose that "adding another dimension" or the 2-dimension, that Prof. Owen mentions is being able to do the Cartesian product, up to 3 Cartesian products, of the block-thread index.

Quiz: Configuring the Kernel Launch Parameters 2

Most general syntax:

Configuring the kernel launch

```
kernel<<<grid of blocks, block of threads >>>(...)

// for example

square<<<dim3(bx,by,bz), dim3(tx,ty,tz), shmem>>>(...)
```

where `dim3(tx,ty,tz)` is the grid of blocks $bx \cdot by \cdot bz$
`{dim3}(tx,ty,tz)` is the block of threads $tx \cdot ty \cdot tz$
`shmem` is the shared memory per block in bytes

Problem Set 1 "Also, the image is represented as an 1D array in the kernel, not a 2D array like I mentioned in the video."
Here's part of that code for squaring numbers:

```
__global__ void square(float *d_out, float *d_in) {
  int idx = threadIdx.x;
  float f = d_in[idx];
  d_out[idx] = f*f;
  }
```

3.1.5. *Grid of blocks, block of threads, thread that's indexed; (mathematical) structure of it all.* Let

$$\text{grid} = \prod_{I=1}^{N} (\text{block})^{n_I^{\text{block}}}$$

where $N = 1, 2, 3$ (for CUDA) and by naming convention $\begin{aligned} I &= 1 \equiv x \\ I &= 2 \equiv y \\ I &= 3 \equiv z \end{aligned}$

Let's try to make it explicity (as others had difficulty understanding the grid, block, thread model, cf. colored image to greyscale image using CUDA parallel processing, Cuda gridDim and blockDim) through commutative diagrams and categories (from math):

$$\begin{array}{ccc} \prod_{I=1}^{N} \mathbb{Z}^+ & & \ni (N_x^{\text{blocks}}, N_y^{\text{blocks}}, N_z^{\text{blocks}}) \\ \text{gridDim} \circlearrowleft \downarrow \text{dim3} & & \text{dim3} \downarrow \circlearrowright (\text{gridDim}.x,\ \text{gridDim}.y,\ \text{gridDim}.z) \\ \text{grid} & & \ni \text{gridSize}(N_x^{\text{blocks}}, N_y^{\text{blocks}}, N_z^{\text{blocks}}) \end{array}$$

$$\begin{array}{ccc} \text{grid} & & \ni \texttt{d\_rgbaImage} \\ \downarrow \text{blockIdx} & & \downarrow (\text{blockIdx}.x, \text{blockIdx}.y, \text{blockIdx}.z) \\ \prod_{I=1}^{N} \mathbb{Z} \supset \prod_{I=1}^{N} \{1 \dots N_I^{\text{blocks}}\} & & \ni (i^{\text{blocks}}, j^{\text{blocks}}, k^{\text{blocks}}) \end{array}$$

and then similar relations (i.e. arrows, i.e. relations) go for a block of threads:

$$\begin{array}{ccc} \prod_{I=1}^{N} \mathbb{Z}^+ & & \ni (N_x^{\text{threads}}, N_y^{\text{threads}}, N_z^{\text{threads}}) \\ \text{blockDim} \circlearrowleft \downarrow \text{dim3} & & \text{dim3} \downarrow \circlearrowright (\text{blockDim}.x,\ \text{blockDim}.y,\ \text{blockDim}.z) \\ \text{block} & & \ni \text{blockSize}(N_x^{\text{threads}}, N_y^{\text{threads}}, N_z^{\text{threads}}) \end{array}$$

$$\begin{array}{ccc} \text{block} & & \ni \text{block} \\ \downarrow \text{threadIdx} & & \downarrow (\text{threadIdx}.x, \text{threadIdx}.y, \text{threadIdx}.z) \\ \prod_{I=1}^{N} \mathbb{Z} \supset \prod_{I=1}^{N} \{1 \dots N_I^{\text{threads}}\} & & \ni (i^{\text{threads}}, j^{\text{threads}}, k^{\text{threads}}) \end{array}$$

gridsize help assignment 1 Pp explains how threads per block is variable, and remember how Owens said Luebki says that a GPU doesn't get up for more than a 1000 threads per block.

3.1.6. *Generalizing the model of an image.* Consider vector space V, e.g. $\dim V = 4$, vector space V over field $\mathbb{K}$, so $V = \mathbb{K}^{\dim V}$.

Each pixel represented by $\forall\, v \in V$.

Consider an image, or space, M. $\dim M = 2$ (image), $\dim M = 3$. Consider a local chart (that happens to be global in our case):

$$\varphi : M \to \mathbb{Z}^{\dim M} \supset \{1 \dots N_1\} \times \{1 \dots N_2\} \times \cdots \times \{1 \dots N_{\dim M}\}$$
$$\varphi : x \mapsto (x^1(x), x^2(x), \dots, x^{\dim M}(x))$$

$$\begin{array}{ccc} E & \xrightarrow{\varphi} & M \times V \\ \pi \downarrow & \swarrow & \\ M & & \end{array} \qquad \begin{array}{ccc} E & \xrightarrow{\varphi} & \text{grid} \times \ \text{block of threads} \\ \pi \downarrow & \swarrow & \\ \text{grid} & & \end{array}$$

Consider a "coarsing" of underlying M:

$$\begin{array}{ccc} M \times V & \xrightarrow{\qquad\qquad \text{proj} \qquad\qquad} & \text{proj}(M) \times \text{proj}(V) \\ \pi \downarrow & & \downarrow \text{proj}(\pi) \\ M = \{1 \dots N_1\} \times \{1 \dots N_2\} \times \cdots \times \{1 \dots N_{\dim M}\} & \xrightarrow{\text{proj}} & \text{proj}(M) = \{1 \dots \frac{N_1}{N_1^{\text{threads}}}\} \times \{1 \dots \frac{N_2}{N_2^{\text{threads}}}\} \times \cdots \times \{1 \dots \frac{N_{\dim M}}{N_{\dim M}^{\text{threads}}}\} \end{array}$$

e.g. $N_1^{\text{thread}} = 12$

$N_2^{\text{thread}} = 12$

Just note that in terms of syntax, you have the "block" model, in which you allocate blocks along each dimension. So in

$$\textit{const dim3 blockSize}(n_x^b, n_y^b, n_z^b)$$
$$\textit{const dim3 gridSize}(n_x^{\text{gr}}, n_y^{\text{gr}}, n_z^{\text{gr}})$$

Then the condition is $n_x^b/\dim V, n_y^b/\dim V, n_z^b/\dim V \in \mathbb{Z}$ (condition), $\qquad (n_x^{\text{gr}} - 1)/\dim V, n_y^{\text{gr}}/\dim V, n_z^{\text{gr}}/\dim V \in \mathbb{Z}$

Transpose Part 1

Now

$$\text{Mat}_{\mathbb{F}}(n, n) \xrightarrow{T} \text{Mat}_{\mathbb{F}}(n, n)$$
$$A \mapsto A^T \text{ s.t. } (A^T)_{ij} = A_{ji}$$
$$\text{Mat}_{\mathbb{F}} \xrightarrow{T} \mathbb{F}^{n^2}$$
$$A_{ij} \mapsto A_{ij} = A_{in+j}$$

$$\begin{array}{ccc} \text{Mat}_{\mathbb{F}}(n, n) & \longrightarrow & \mathbb{F}^{n^2} \\ T \downarrow & & \downarrow T \\ \text{Mat}_{\mathbb{F}}(n, n) & \longrightarrow & \mathbb{F}^{n^2} \end{array} \qquad \begin{array}{ccc} A_{ij} & \longmapsto & A_{in+j} \\ T \downarrow & & \downarrow T \\ (A^T)_{ij} = A_{ji} & \longmapsto & A_{jn+i} \end{array}$$

Transpose Part 2

Possibly, transpose is a functor.

Consider struct as a category. In this special case, Objstruct = {arrays} (a struct of arrays). Now this struct already has a hash table for indexing upon declaration (i.e. "creation"): so this category struct will need to be equipped with a "diagram" from the category of indices J to struct: $J \to$ struct.

So possibly

$$\begin{aligned} \text{struct} &\xrightarrow{T} \text{array} \\ \text{ObjStruct} = \{ \text{ arrays } \} &\xrightarrow{T} \text{Objarray} = \{ \text{ struct } \} \\ J \rightarrow \text{ struct} &\xrightarrow{T} J \rightarrow \text{ array} \end{aligned}$$

Quiz: What Kind Of Communication Pattern This quiz made a few points that clarified the characteristics of these so-called communication patterns (amongst the memory?)

- map is bijective, and map : Idx → Idx
- gather - not necessarily surjective
- scatter - not necessarily surjective
- stencil - surjective
- transpose (see before)

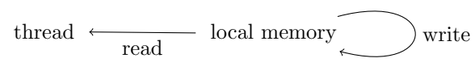
Parallel Communication Patterns Recap

- map - bijective
- transpose - bijective
- gather - not necessarily surjective, and is many-to-one (by def.)
- scatter - one-to-many (by def.) and is not necessarily surjective
- stencil - several-to-one (not injective, by definition), and is surjective
- reduce - all-to-one
- scan/sort - all-to-all

Programmer View of the GPU
thread blocks: group of threads that cooperate to solve a (sub)problem
Thread Blocks And GPU Hardware
CUDA GPU is a bunch of SMs:
Streaming Multiprocessors (SM)s
SMs have a bunch of simple processors and memory.
Dr. Luebki:

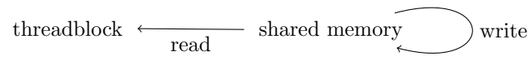
> Let me say that again because it's really important
> GPU is responsible for allocating blocks to SMs

Programmer only gives GPU a pile of blocks.
Quiz: What Can The Programmer Specify
I myself thought this was a revelation and was not intuitive at first:
Given a single kernel that's launched on many thread blocks include X, Y, the programmer cannot specify the sequence the blocks, e.g. block X, block Y, run (same time, or run one after the other), and which SM the block will run on (GPU does all this).
Quiz: A Thread Block Programming Example
Open up `hello blockIdx.cu` in Lesson 2 Code Snippets (I got the repository from github, repo name is cs344).
At first, I thought you can do a single file compile and run in Eclipse without creating a new project. No. cf. Eclipse creating projects every time to run a single file?.
I ended up creating a new CUDA C/C++ project from File -¿ New project, and then chose project type Executable, Empty Project, making sure to include Toolchain CUDA Toolkit (my version is 7.5), and chose an arbitrary project name (I chose cs344single). Then, as suggested by Kenny Nguyen, I dragged and dropped files into the folder, from my file directory program.
I ran the program with the "Play" triangle button, clicking on the green triangle button, and it ran as expected. I also turned off Build Automatically by deselecting the option (no checkmark).
GPU Memory Model



Then consider threadblock ≡ thread block

$$\begin{aligned} \text{Objthreadblock} &\supset \{ \text{ threads } \} \\ \text{FinSet} &\xrightarrow{\text{threadIdx}} \text{thread } \in \text{Morthreadblock} \end{aligned}$$



∀ thread,

Synchronization - Barrier
Quiz: The Need For Barriers
3 barriers were needed (wasn't obvious to me at first). All threads need to finish the write, or initialization, so it'll need a barrier.
While

```
array[idx] = array[idx+1];
```

is 1 line, it'll actually need 2 barriers; first read. Then write.
So *actually* we'll need to *rewrite* this code:

```
int temp = array[idx+1];
__syncthreads();
array[idx] = temp;
__syncthreads();
```

kernels have implicit barrier for each.
Writing Efficient Programs
(1) Maximize *arithmetic intensity* arithmetic intensity := $\frac{\text{math}}{\text{memory}}$
video: Minimize Time Spent On Memory
local memory is fastest; global memory is slower

$$\text{local} > \text{ shared} >> \text{global} >> \text{CPU}$$

kernel we know (in the code) is tagged with `__global__`
quiz: A Quiz on Coalescing Memory Access
Work it out as Dr. Luebki did to figure out if it's coalesced memory access or not.
Atomic Memory Operations
Atomic Memory Operations
atomicadd atomicmin atomicXOR atomicCAS Compare And Swap

4. Pointers in C; Pointers in C categorified (interpreted in Category Theory)

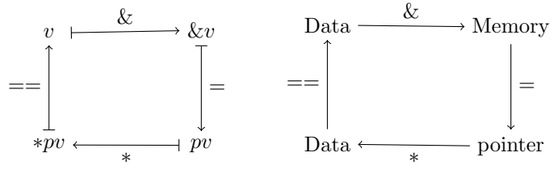
Suppose $v \in$ ObjData, category of data **Data**,
e.g. $v \in$ Int $\in$ ObjType, category of types Type.

$$\text{Data} \xrightarrow{\&} \text{Memory}$$
$$v \overset{\&}{\mapsto} \&v$$

with address $\&v \in$ Memory.
With
assignment $pv = \&v$,

$pv \in$ Objpointer, category of pointers, pointer
$pv \in$ Memory (i.e. not $pv \in$ Dat, i.e. $pv \notin$ Dat)

$$\text{pointer} \ni pv \overset{*}{\mapsto} *pv \in \text{Dat}$$

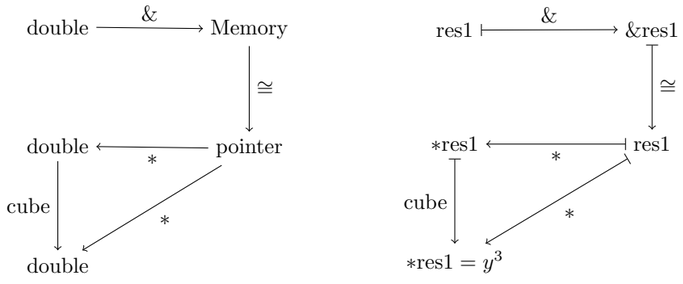


Examples. Consider `passfunction.c` in Fitzpatrick [5].
Consider the type `double`, `double` $\in$ ObjTypes.
fun1, fun2 $\in$ MorTypes namely
fun1, fun2 $\in$ Hom(double, double) $\equiv \text{Hom}_{\text{Types}}$(double, double)
Recall that

$$\text{pointer} \xrightarrow{*} \text{Dat}$$
$$\text{pointer} \xrightarrow{\&} \text{Memory}$$

$*, \&$ are functors with domain on the category pointer.
Pointers to functions is the "extension" of functor $*$ to the codomain of MorTypes:

$$\text{pointer} \xrightarrow{*} \text{MorTypes}$$
$$\text{fun1} \overset{*}{\mapsto} *\text{fun1} \in \text{Hom}_{\text{Types}}(\text{double}, \text{double})$$



It's unclear to me how `void cube` can be represented in terms of category theory, as surely it cannot be represented as a mapping (it acts upon a functor, namely the $*$ functor for pointers). It doesn't return a value, and so one cannot be confident to say there's explicitly a domain and codomain, or range for that matter.
But what is going on is that

$$\text{pointer , double , pointer} \xrightarrow{\text{cube}} \text{pointer , pointer}$$
$$\text{fun1}, x, \text{res1} \overset{\text{cube}}{\mapsto} \text{fun1}, \text{res1}$$

s.t. $*\text{res1} = y^3 = (*\text{fun1}(x))^3$
So I'll speculate that in this case, `cube` is a functor, and in particular, is acting on $*$, the so-called deferencing operator:

$$\begin{array}{ccc} \text{pointer} \xrightarrow{*} \text{float} \in \text{Data} & \xrightarrow{\text{cube}} & \text{pointer} \xrightarrow{\text{cube}(*)} \text{float} \in \text{Data} \\ \text{res1} \overset{*}{\mapsto} *\text{res1} & & \text{res1} \overset{\text{cube}(*)}{\mapsto} \text{cube}(*\text{res1}) = y^3 \end{array}$$

cf. Arrays, from Fitzpatrick [5]

$$\text{Types} \xrightarrow{\text{declaration}} \text{arrays}$$

If $x \in$ Objarrays,

$$\&x[0] \in \text{Memory} \xrightarrow{==} x \in \text{ pointer (to 1st element of array)}$$

cf. Section 2.13 Character Strings from Fitzpatrick [5]

```
char word[20] = ''four''
char *word = ''four''
```

cf. C++ extensions for C according to Fitzpatrick [5]

- simplified syntax to pass by reference pointers into functions
- inline functions
- variable size arrays

```
int n;
double x[n];
```

- complex number class

4.0.7. *Need a CUDA, C, C++, IDE? Try Eclipse!* This website has a clear, lucid, and pedagogical tutorial for using Eclipse: Creating Your First C++ Program in Eclipse. But it looks like I had to pay. Other than the well-written tips on the webpage, I looked up stackexchange for my Eclipse questions (I had difficulty with the Eclipse documentation).

Part 3. Machine Learning

cf. Machine Learning - Introduction, from Coursera. Dr. Andrew Ng.

(1) Week 1
- Linear Regression with One Variable
  - Model and Cost Function
    - Model Representation
    - Cost Function
    - Cost Function - Intuition I
    - Cost Function - Intuition II
  - Parameter Learning
    - Gradient Descent
    - Gradient Descent Intuition
    - Gradient Descent For Linear Regression

cf. Linear Regression with One Variable

cf. Model Representation; Week 1 Linear Regression with 1 Variable, Coursera Machine Learning, Ng

For hypothesis h,

$$h_\theta : \mathbb{R}^d \to \mathbb{R}$$
$$h_\theta : x \mapsto h_\theta(x) \qquad \text{(prediction of } y \text{ for } x)$$

$h_\theta \in L(\mathbb{R}^d, \mathbb{R})$

$$h_\theta : \mathbb{R}^{|\theta|} \to L(\mathbb{R}^d, \mathbb{R})$$
$$\theta \mapsto h_\theta$$

Cost Function; Week 1, Coursera, Machine Learning, Ng

So for parameters

$$\theta \in \mathbb{R}^{|\theta|}$$

define a *cost function*

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 \tag{1}$$

Find

$$\min_\theta J(\theta) = ?(???)$$

for

$$J : \mathbb{R}^{|\theta|} \to \mathbb{R}$$

Actually,

$$J(\theta, (x_i, y_i)_{i \in I_{\text{train}}})$$
$$J : \mathbb{R}^{|\theta|} \times (\mathbb{R}^d)^m \times \mathbb{R}^m \to \mathbb{R} \tag{2}$$

m = number of training examples $= |I_{\text{train}}|$.

Considering

$$H(\theta + \Delta\theta) \approx J(\theta) + \text{grad}J(\theta) \cdot \Delta\theta + \frac{1}{2t}\|\Delta\theta\|^2$$

Suppose $\Delta\theta \equiv \Delta\theta(t) = t\Delta\theta$

$\Delta\theta \approx -\gamma \text{grad}J(\theta)$ is an ansatz, γ small enough.

Then assume J convex, use this ansatz by plugging in, with Lipshitz condition

$$\|\text{grad}J(\theta + \Delta\theta) - \text{grad}J(\theta)\| \leq L\|\Delta\theta\|$$

some constant $L > 0$,

$$\theta^i_{n+1} = \theta^i_n - \gamma_n(\text{grad}J(\theta))^i$$
$$\gamma_n = \frac{(\theta^i_n - \theta^i_{n-1})(\text{grad}_\theta J(x_n) - \text{grad}_\theta J(x_{n-1}))^i}{\|\text{grad}_\theta J(x_n) - \text{grad}_\theta J(x_{n-1})\|^2} = \frac{(\theta_n - \theta_{n-1}) \cdot (\text{grad}_\theta J(x_n) - \text{grad}_\theta J(x_{n-1}))}{\|\text{grad}_\theta J(x_n) - \text{grad}_\theta J(x_{n-1})\|^2} \tag{3}$$

or as Ng points out in the Gradient Descent lesson recap, the correct way is to store in temporary variables first:

$$\text{temp} = \theta^i_n - \gamma_n(\text{grad}J(\theta))^i$$
$$\theta^i_{n+1} = \text{temp} \tag{4}$$

where temp $\in \mathbb{R}^{|\theta|}$

In the lesson recap for Gradient Descent Intuition, Ng denotes the learning rate $\alpha \in \mathbb{R}$ with α, but note that it's denoted as γ or `gamma` for `sci-kit learn`. So be aware of different notations. Nevertheless, the learning rate can be a constant, but even then, choosing it is nontrivial.

4.0.8. *Testing many hypotheses at the same time, via refactoring the matrix.* In Linear Algebra Review of Week 1, Matrix Matrix Multiplication, Ng provided a useful tip in refactoring the matrix of hypotheses h_θ so to test multiple number of hypotheses at the same time on the same input data, X.

Mathematically, beginning with

$$h : \mathbb{R}^{|\theta|} \longrightarrow L(\mathbb{R}^d, \mathbb{R})$$
$$\theta \longmapsto h_\theta$$

Consider testing H different hypotheses, $\underbrace{\mathbb{R}^{|\theta|} \times \cdots \times \mathbb{R}^{|\theta|}}_{H} \equiv \otimes_{i=1}^H \mathbb{R}^{|\theta|}$,

so treat

$$\otimes_{i=1}^H \mathbb{R}^{|\theta|} = \text{Mat}_{\mathbb{R}}(|\theta|, H)$$

and so

$$h : \otimes_{i=1}^H \mathbb{R}^{|\theta|} = \text{Mat}_{\mathbb{R}}(|\theta|, H) \longrightarrow \otimes_{i=1}^H L(\mathbb{R}^d, \mathbb{R})$$
$$\theta^{(i)} \longmapsto h_{\theta^{(i)}}$$

cf. Week 4, Non-linear Hypotheses video of Motivations for Coursera's Machine Learning by Ng

For a sigmoid function g, consider

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

If n large (Ng's notation), $d = \dim\mathbb{R}^d$, number of features for training (data) set,
for including quadratic features,

$$x_1^2, x_1x_2, x_1x_3, x_1x_4 \dots x_1x_{100}$$
$$x_2^2, x_1x_3, \dots$$
$$\approx \mathcal{O}(n^2) \approx \frac{n^2}{2} \qquad (\mathcal{O}(d^2) \approx \frac{d^2}{2})$$

e.g. computer vision,
e.g. 50×50 pixel images,
$n = 2500$
pixel intensity $\in [0, 255]$
rgb $\in [0, 255]^3$

$$g : \mathbb{R}^{|\theta|} \to L(\mathbb{R}^d, \mathbb{R})$$
$$\theta \mapsto g(\theta) \equiv g_\theta$$

$n \equiv d = 2$.

Consider

$$\sum_{\substack{a_1, a_2 = 0 \\ i = a_1 + 2a_2}} \theta^{(i)} x_1^{a_1} x_2^{a_2}$$

and so for this example

$$g(\theta)(x_1, x_2) = g\left(\sum_{\substack{a_1, a_2 = 0 \\ i = a_1 + 2a_2}} \theta^{(i)} x_1^{a_1} x_2^{a_2}\right)$$

For computer vision, consider

$$x \in \mathbb{R}^d \text{ with } d = n^x \times n^y$$

and in particular, given pixel intensity or rgb range,

$$x \in [0, 255]^d$$

$$x \in [0, 255]^{3d}$$

cf. Model Representation I of Week 4, Coursera's Machine Learning Introduction with Ng
The notes at the end of each video segment **help very much**.
For input

$$\mathbf{x} \in \mathbb{R}^d$$

e.g. $d = 1, 2, 3,$ or $4, \dots$
$x_0 =$ "bias unit", input node 0, $x_0 = 1$ always (Ng).
Sigmoid (logistic) activation function $\equiv a$.

$$a_i^{(j)} \equiv \text{ "activation" of unit } i \text{ in layer } j$$

$j \in \{2, \dots, N-1\}$, $j = 1$ is input layer, $j = N$ is output layer.

$$a_i^{(j)} = g(\Theta_{ik}^{(j-1)} x_k)$$

$$j \xrightarrow{\Theta^{(j)}} j+1$$

$\Theta^{(j)}$ matrix of weights controlling function mapping from layer j to layer $j+1$.

$$h_\Theta(x) = a_1^{(N)} = g(\Theta_{1k}^{(N-1)} a_k^{(N-1)})$$

$\forall$ layer j, $\exists$ matrix of weights $\Theta^{(j)}$.
If s_j units in layer j, s_{j+1} units in layer $j+1$, $\dim\Theta^{(j)} = s_{j+1} \times (s_j + 1)$
If $N = 2$, (1 neuron or only 1 hidden layer)

$$x = (x_i)_{i=1\dots d} \in \mathbb{R}^d, \qquad y \in \mathbb{R},\ x_0 = 1$$
$$y = h(\Theta_{1k}^{(1)} x_k^{(1)}) = h(\Theta_{1k}^{(1)} x_k) = h(\Theta^{(1)})(x)$$

e.g. $h(z) = \frac{1}{1+e^z}$ logistic function.
Neural Network, input layer, output layer, and hidden layers.

$$\Theta_{ik}^{(j)} x_k \mapsto g a_i^{(j+1)} \qquad \begin{matrix} k = 0, 1, \dots s_j \\ i = 1, 2, \dots s_{j+1} \end{matrix} \tag{5}$$

Note that y can be $y \in \mathbb{R}^M$, not just $M = 1$.
Model Representation II
$z_i^{(j)}$, $i = 1, \dots s_j$, layer $j = 1, \dots N$.

$$g : z_i^{(j)} \mapsto a_i^{(j)} \tag{6}$$

e.g. $z_i^{(j)} = \Theta_{ik}^{(j-1)} x_k$, $k = 0, 1 \dots d$.
Set $x = a^{(1)}$ for input layer.

$$\Theta^{(j-1)} \in \mathrm{Mat}_{\mathbb{R}}((d+1), s_j)$$
$$\Theta^{(j-1)} : a^{(j-1)} \in \mathbb{R}^{d+1} \mapsto z^{(j)} \in \mathbb{R}^{s_j} \xrightarrow{g} a^{(j)} \in \mathbb{R}^{s_j} \xrightarrow{a_0^{(j)}=1} a^{(j)} \in \mathbb{R}^{s_j+1} \tag{7}$$

For the $j = N$ case, "output" layer,

$$\Theta^{(N-1)} : a^{(N-1)} \mapsto z^N \in \mathbb{R} \xrightarrow{g} g(z^N) = a^N = h_\Theta(x) \in \mathbb{R} \qquad \Theta^{(N-1)} \in \mathrm{Mat}_{\mathbb{R}}(s_{N-1} + 1, 1) \tag{8}$$

In general,

$$\Theta^{(N-1)} : a^{(N-1)} \mapsto z^N \in \mathbb{R} \xrightarrow{g} g(z^N) = a^N = h_\Theta(x) \in \mathbb{R}^M \qquad \Theta^{(N-1)} \in \mathrm{Mat}_{\mathbb{R}}(s_{N-1} + 1, M)$$

cf. Learning With Large Datasets, Quiz of Week 10, Gradient Descent with Large Datasets; Learning with Large Datasets.
Suppose you are facing a supervised learning problem and have a very large dataset ($m = 100,000,000$). How can you tell if using all of the data is likely to perform much better than using a small subset of the data (say $m = 1,000$)?
Plot a learning curve ($J_{\text{train}}(\theta)$ and $J_{CV}(\theta)$, plotted as a function of m) for a range of values of m and verify that the algorithm has high variance when m is small.

Part 4. Notes

Restricted Boltzmann machine - estimate a probability distribution
Recurrent neural network - creates an internal state of the network which allows it to exhibit dynamic temporal behavior

References

[1] Trevor Hastie, Robert Tibshirani, Jerome Friedman. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction**, Second Edition (Springer Series in Statistics) 2nd ed. 2009. Corr. 7th printing 2013 Edition. ISBN-13: 978-0387848570. https://web.stanford.edu/~hastie/local.ftp/Springer/OLD/ESLII_print4.pdf

[2] Jared Culbertson, Kirk Sturtz. *Bayesian machine learning via category theory.* arXiv:1312.1445 [math.CT]

[3] John Owens. David Luebki. *Intro to Parallel Programming. CS344.* **Udacity** http://arxiv.org/abs/1312.1445 Also, https://github.com/udacity/cs344

[4] CS229 Stanford University. http://cs229.stanford.edu/materials.html

[5] Richard Fitzpatrick. "Computational Physics." http://farside.ph.utexas.edu/teaching/329/329.pdf

[6] LISA lab, University of Montreal. Deep Learning Tutorial. http://deeplearning.net/tutorial/deeplearning.pdf September 2015.