Introduction to Deep Learning

8. Numerical Stability, Hardware

STAT 157, Spring 2019, UC Berkeley

Alex Smola and Mu Li

courses.d2l.ai/berkeley-stat-157

Numerical Stability

Wikipedia

Gradients for Neural Networks

- Consider a network with *d* layers

$$\mathbf{h}^t = f_t(\mathbf{h}^{t-1}) \quad \text{and} \quad y = \ell \circ f_d \circ \ldots \circ f_1(\mathbf{x})$$

- Compute the gradient of the loss ℓ w.r.t. $\mathbf{W}_t$

$$\frac{\partial \ell}{\partial \mathbf{W}^t} = \frac{\partial \ell}{\partial \mathbf{h}^d} \underbrace{\frac{\partial \mathbf{h}^d}{\partial \mathbf{h}^{d-1}} \ldots \frac{\partial \mathbf{h}^{t+1}}{\partial \mathbf{h}^t}}_{\text{Multiplication of } d\text{-}t \text{ matrices}} \frac{\partial \mathbf{h}^t}{\partial \mathbf{W}^t}$$

Two Issues for Deep Neural Networks

$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^{i}}$$

Gradient Exploding

$$1.5^{100} \approx 4 \times 10^{17}$$

Gradient Vanishing

$$0.8^{100} \approx 2 \times 10^{-10}$$

Example: MLP

- Assume MLP (without bias for simplicity)

$$f_t(\mathbf{h}^{t-1}) = \sigma(\mathbf{W}^t \mathbf{h}^{t-1})$$ σ is the activation function

$$\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = \text{diag}\left(\sigma'(\mathbf{W}^t \mathbf{h}^{t-1})\right)(W^t)^T$$ σ' is the gradient function of σ

$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})\right)(W^i)^T$$

Gradient Exploding

- Use ReLU as the activation function

$$\sigma(x) = \max(0,x) \quad \text{and} \quad \sigma'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Elements of $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})\right)(W^i)^T$ may from $\prod_{i=t}^{d-1} (W^i)^T$
  - Leads to large values when *d-t* is large

$$1.5^{100} \approx 4 \times 10^{17}$$

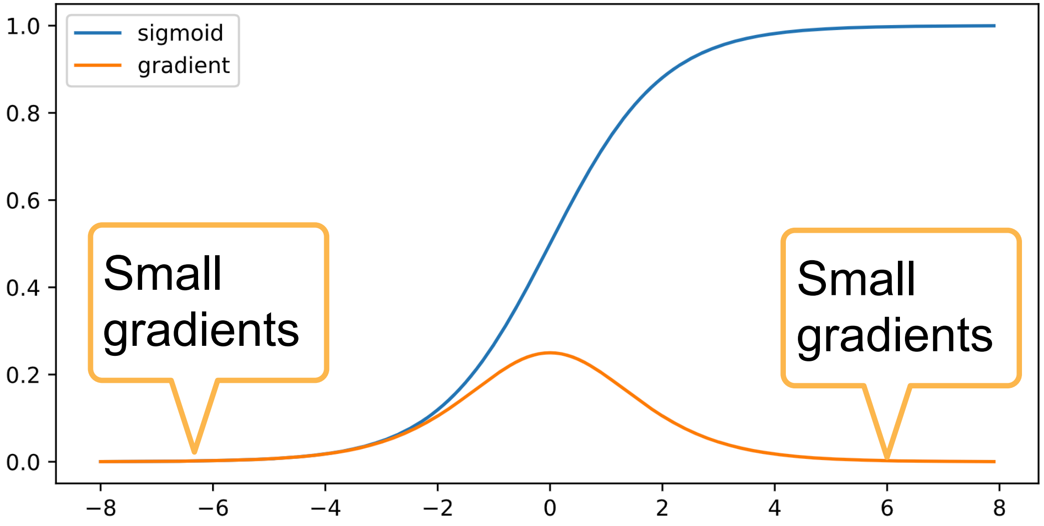
Issues with Gradient Exploding

- Value out of range: infinity value
  - Severe for using 16-bit floating points
    - Range: 6e-5 - 6e4
- Sensitive to learning rate (LR)
  - Not small enough LR -> large weights -> larger gradients
  - Too small LR -> No progress
  - May need to change LR dramatically during training

Gradient Vanishing

- Use sigmoid as the activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$



Gradient Exploding

- Use sigmoid as the activation function

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \sigma'(x) = \sigma(x)(1-\sigma(x))$$

- Elements $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})\right)(W^i)^T$ are products of *d-t* small values

$$0.8^{100} \approx 2 \times 10^{-10}$$

Issues with Gradient Vanishing

- Gradients with value 0
  - Severe with 16-bit floating points
- No progress in training
  - No matter how to choose learning rate
- Severe with bottom layers
  - Only top layers are well trained
  - No benefit to make networks deeper

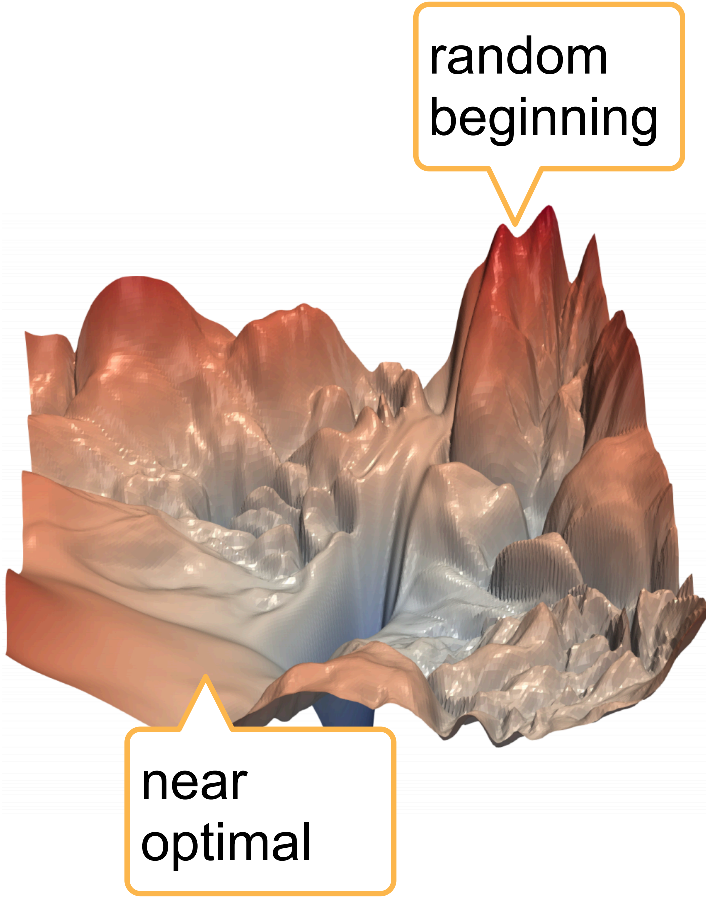
Stabilize Training

Stabilize Training

- Goal: make sure gradient values are in a proper range
    - E.g. in [1e-6, 1e3]
- Multiplication -> plus
    - ResNet, LSTM
- Normalize
    - Batch Normalization, Gradient clipping
- Proper weight initialization and activation functions

Weight Initialization

- Initialize weights with random values in a proper range
- The beginning of training easily suffers to numerical instability
    - The surface far away from an optimal can be complex
    - Near optimal may be flatter
- Initializing according to $\mathcal{N}(0,\ 0.01)$ works well for small networks, but not guarantee for deep neural networks



Constant Variance for each Layer

- Treat both layer outputs and gradients are random variables
- Make the mean and variance for each layer's output are same, similar for gradients

Forward

$$\mathbb{E}[h_i^t] = 0$$

$$\mathrm{Var}[h_i^t] = a$$

Backward

$$\mathbb{E}\left[\frac{\partial \ell}{\partial h_i^t}\right] = 0 \quad \mathrm{Var}\left[\frac{\partial \ell}{\partial h_i^t}\right] = b \qquad \forall i, t$$

a and *b* are constants

Example: MLP

- Assumptions
  - i.i.d $w_{i,j}^t$, $\mathbb{E}[w_{i,j}^t] = 0, \ \mathrm{Var}[w_{i,j}^t] = \gamma_t$
  - h_i^{t-1} is independent to $w_{i,j}^t$
  - identity activation: $\mathbf{h}^t = \mathbf{W}^t \mathbf{h}^{t-1}$ with $\mathbf{W}^t \in \mathbb{R}^{n_t \times n_{t-1}}$

$$\mathbb{E}[h_i^t] = \mathbb{E}\left[\sum_j w_{i,j}^t h_j^{t-1}\right] = \sum_j \mathbb{E}[w_{i,j}^t]\mathbb{E}[h_j^{t-1}] = 0$$

Forward Variance

$$\mathrm{Var}[h_i^t] = \mathbb{E}[(h_i^t)^2] - \mathbb{E}[h_i^t]^2 = \mathbb{E}\left[\left(\sum_j w_{i,j}^t h_j^{t-1}\right)^2\right]$$

$$= \mathbb{E}\left[\sum_j \left(w_{i,j}^t\right)^2 \left(h_j^{t-1}\right)^2 + \sum_{j \neq k} w_{i,j}^t w_{i,k}^t h_j^{t-1} h_k^{t-1}\right]$$

$$= \sum_j \mathbb{E}\left[\left(w_{i,j}^t\right)^2\right] \mathbb{E}\left[\left(h_j^{t-1}\right)^2\right]$$

$$= \sum_j \mathrm{Var}[w_{i,j}^t]\mathrm{Var}[h_j^{t-1}] = n_{t-1}\gamma_t \mathrm{Var}[h_j^{t-1}]$$

$$\Rightarrow \quad n_{t-1}\gamma_t = 1$$

Backward Mean and Variance

- Apply forward analysis as well

$$\frac{\partial \ell}{\partial \mathbf{h}^{t-1}} = \frac{\partial \ell}{\partial \mathbf{h}^{t}} \mathbf{W}^{t} \quad \text{leads to} \quad \left(\frac{\partial \ell}{\partial \mathbf{h}^{t-1}}\right)^T = (W^t)^T \left(\frac{\partial \ell}{\partial \mathbf{h}^{t}}\right)^T$$

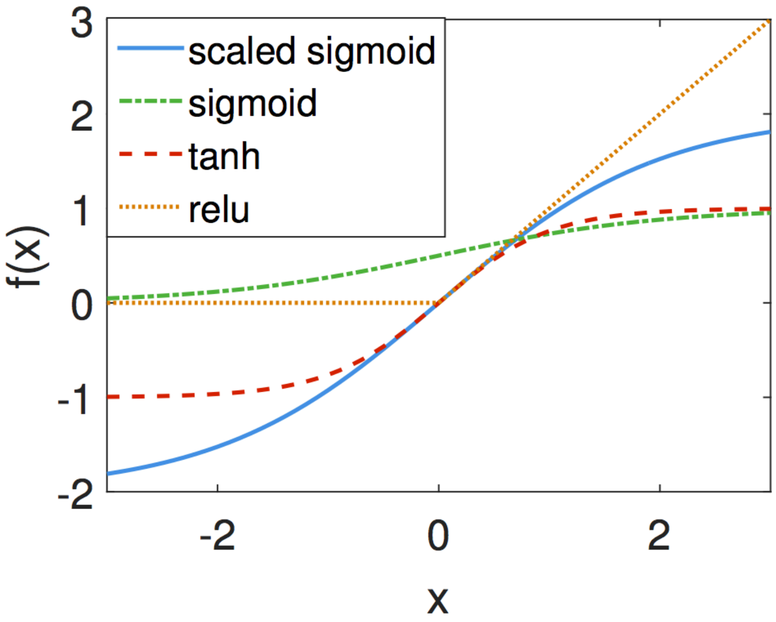
$$\mathbb{E}\left[\frac{\partial \ell}{\partial h_i^{t-1}}\right] = 0$$

$$\mathrm{Var}\left[\frac{\partial \ell}{\partial h_i^{t-1}}\right] = n_t \gamma_t \mathrm{Var}\left[\frac{\partial \ell}{\partial h_j^{t}}\right] \Rightarrow n_t \gamma_t = 1$$

Xavier Initialization

- Conflict goal to satisfies both $n_{t-1}\gamma_t = 1$ and $n_t\gamma_t = 1$
- Xavier $\gamma_t(n_{t-1} + n_t)/2 = 1 \quad \rightarrow \quad \gamma_t = 2/(n_{t-1} + n_t)$
  - Normal distribution $\mathcal{N}\left(0, \sqrt{2/(n_{t-1} + n_t)}\right)$
  - Uniform distribution $\mathcal{U}\left(-\sqrt{6/(n_{t-1} + n_t)}, \sqrt{6/(n_{t-1} + n_t)}\right)$
    - Variance of $\mathcal{U}[-a, a]$ is $a^2/3$
- Adaptive to weight shape, especially when n_t varies

Activation



A Simple Linear Activation Function

- Assume $\sigma(x) = \alpha x + \beta$

$$\mathbf{h}' = \mathbf{W}^t \mathbf{h}^{t-1} \quad \text{and} \quad \mathbf{h}^t = \sigma(\mathbf{h}')$$

$$\mathbb{E}[h_i^t] = \mathbb{E}\left[\alpha h_i' + \beta\right] = \beta \quad \Rightarrow \quad \beta = 0$$

$$\begin{aligned} \mathrm{Var}[h_i^t] &= \mathbb{E}[(h_i^t)^2] - \mathbb{E}[h_i^t]^2 \\ &= \mathbb{E}[(\alpha h_i' + \beta)^2] - \beta^2 \\ &= \mathbb{E}[\alpha^2 (h_i')^2 + 2\alpha\beta h_i' + \beta^2] - \beta^2 \\ &= \alpha^2 \mathrm{Var}[h_i'] \end{aligned} \quad \Rightarrow \quad \alpha = 1$$

Backward

- Assume $\sigma(x) = \alpha x + \beta$

$$\frac{\partial \ell}{\partial \mathbf{h}'} = \frac{\partial \ell}{\partial \mathbf{h}^t}(W^t)^T \quad \text{and} \quad \frac{\partial \ell}{\partial \mathbf{h}^{t-1}} = \alpha \frac{\partial \ell}{\partial \mathbf{h}'}$$

$$\mathbb{E}\left[\frac{\partial \ell}{\partial h_i^{t-1}}\right] = 0 \quad \Rightarrow \quad \beta = 0$$

$$\mathrm{Var}\left[\frac{\partial \ell}{\partial h_i^{t-1}}\right] = \alpha^2 \mathrm{Var}\left[\frac{\partial \ell}{\partial h_j'}\right] \quad \Rightarrow \quad \alpha = 1$$

Revise Activation Functions

- By the Taylor expansions

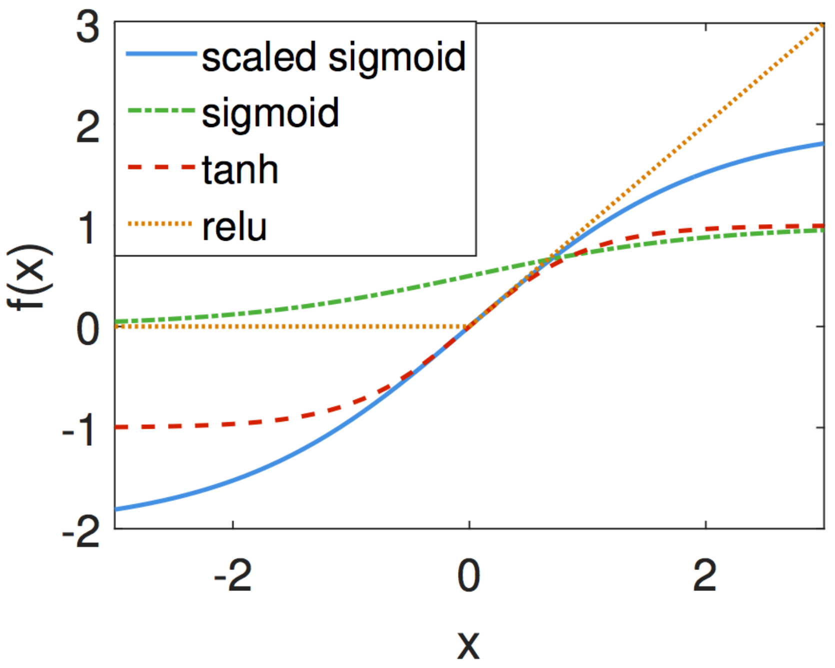
$$\operatorname{sigmoid}(x) = \frac{1}{2} + \frac{x}{4} - \frac{x^3}{48} + O(x^5)$$

$$\tanh(x) = 0 + x - \frac{x^3}{3} + O(x^5)$$

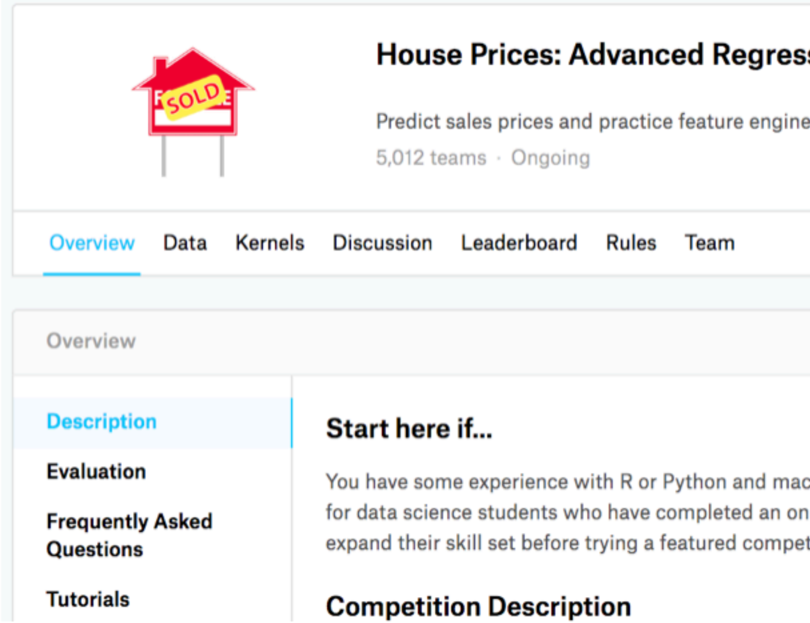
$$\operatorname{relu}(x) = 0 + x \quad \text{for } x \geq 0$$

- "Correct" sigmoid by

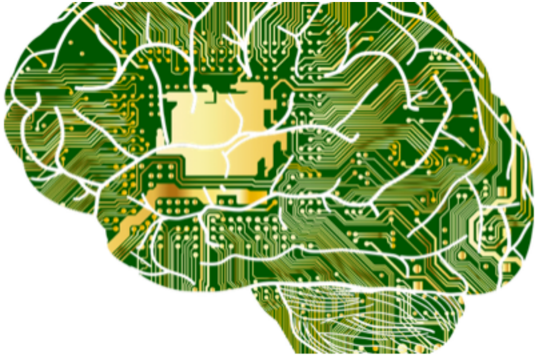
$$4 \times \operatorname{sigmoid}(x) - 2$$



Predicting House Prices on Kaggle



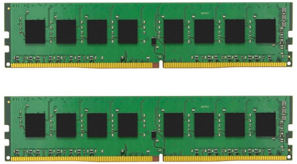
Hardware for Deep Learning



(articleshub360.com)

Your GPU Computer

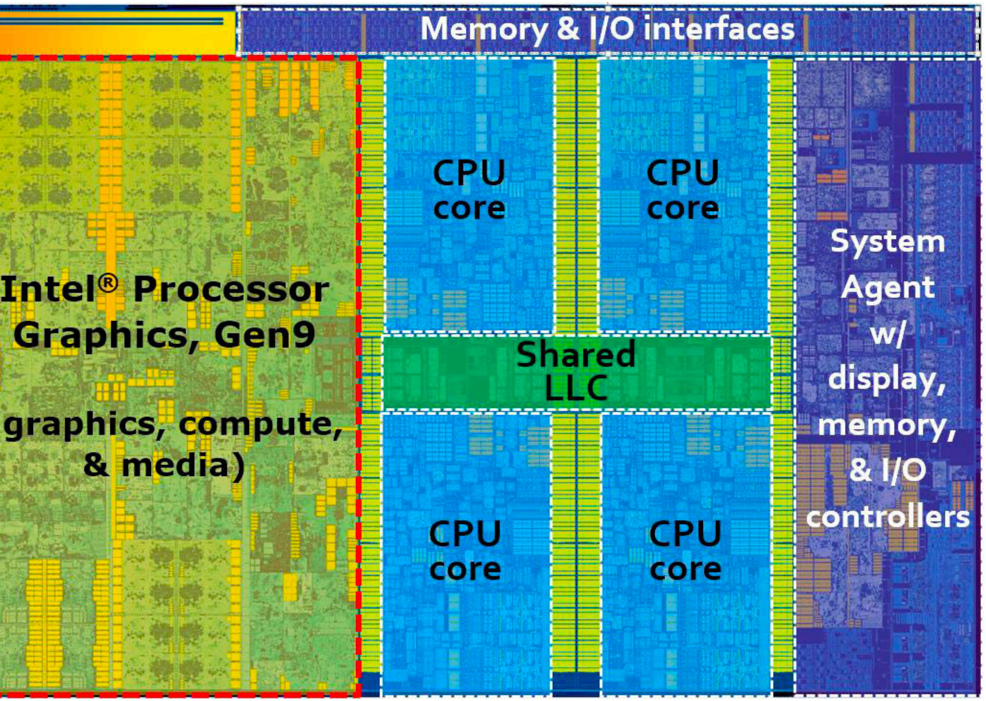
Intel i7
0.15 TFLOPS



DDR4
32 GB

Nvidia Titan X
12 TFLOPS
16 GB

Intel i7-6700K



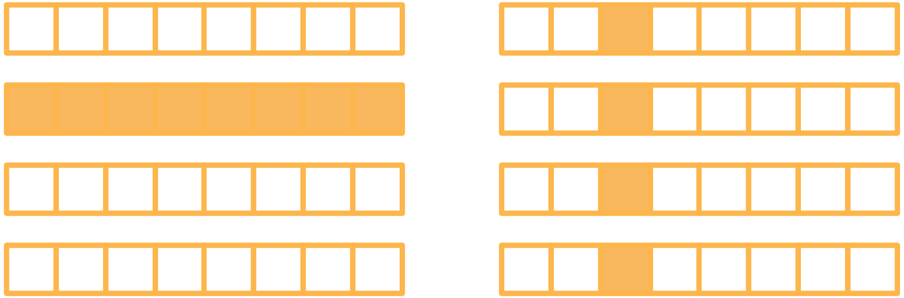
- 4 Physical cores
- Per core
    - 64KB L1 cache
    - 256KB L2 cache
- Shared 8MB L3 cache
- 30 GB/s to main memory

Improve CPU Utilization I

- Before computing *a* + *b*, need to prepare data first
  - Main memory -> L3 -> L2 -> L1 -> registers
    - L1 cache reference time: 0.5 ns
    - L2 cache reference time 7 ns (14 x L1)
    - Main memory reference time 100ns (200 x L1)
- Improve temporal and spatial memory locality
  - Temporal: reuse data so we keep them on cache
  - Spatial: read data sequential so we can pre-fetch data

Case Study

- For a matrix stored in raw-major, accessing a column is slower than accessing a row
    - CPU reads 64 bytes (cache line) each time
    - CPU "smartly" reads the next cache line ahead when it's needed



Improve CPU Utilization II

- Server CPUs may have dozens of cores
  - EC2 P3.16xlarge: 2 Intel Xeon CPUs, 32 physical cores in total
- Parallelization to use all cores
  - Hyper-threading may not help because of shared registers

Case Study

- Left is slower than right (Python)

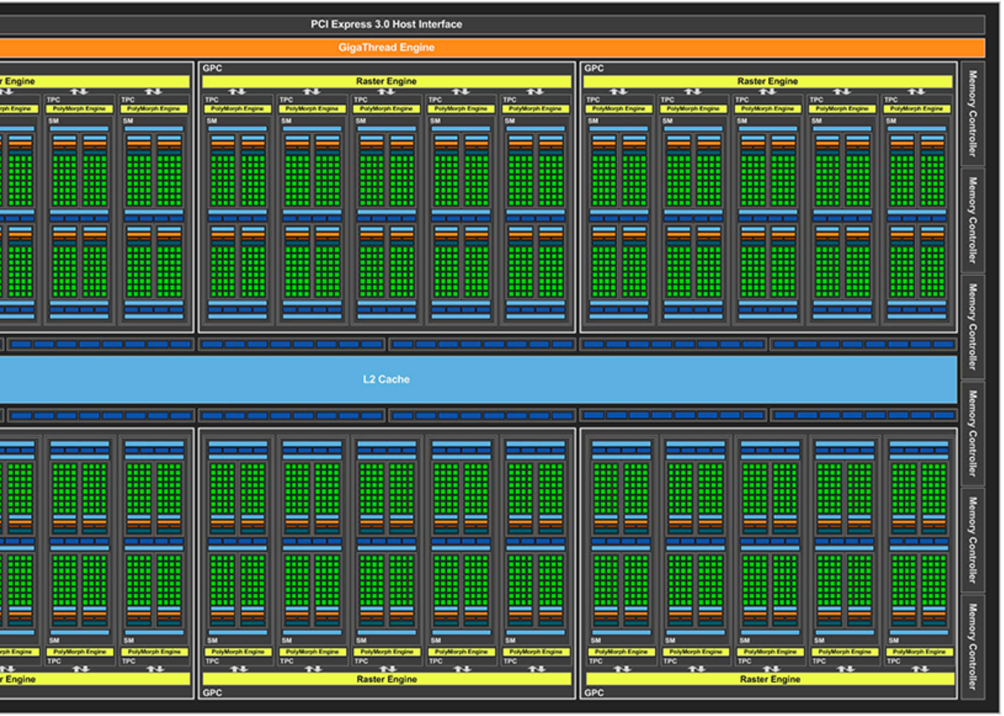
```
for i in range(len(a)):
    c[i] = a[i] + b[i]
```

```
c = a + b
```

  - Left issues *n* calls, each invoking has certain overhead (e.g. several microsecond)
  - Right is easier to be parallelized by a C++ operator

```
#pragma omp for
for (i=0; i<a.size(); i++) {
    c[i] = a[i] + b[i]
}
```

Nvidia Titan X (Pascal)

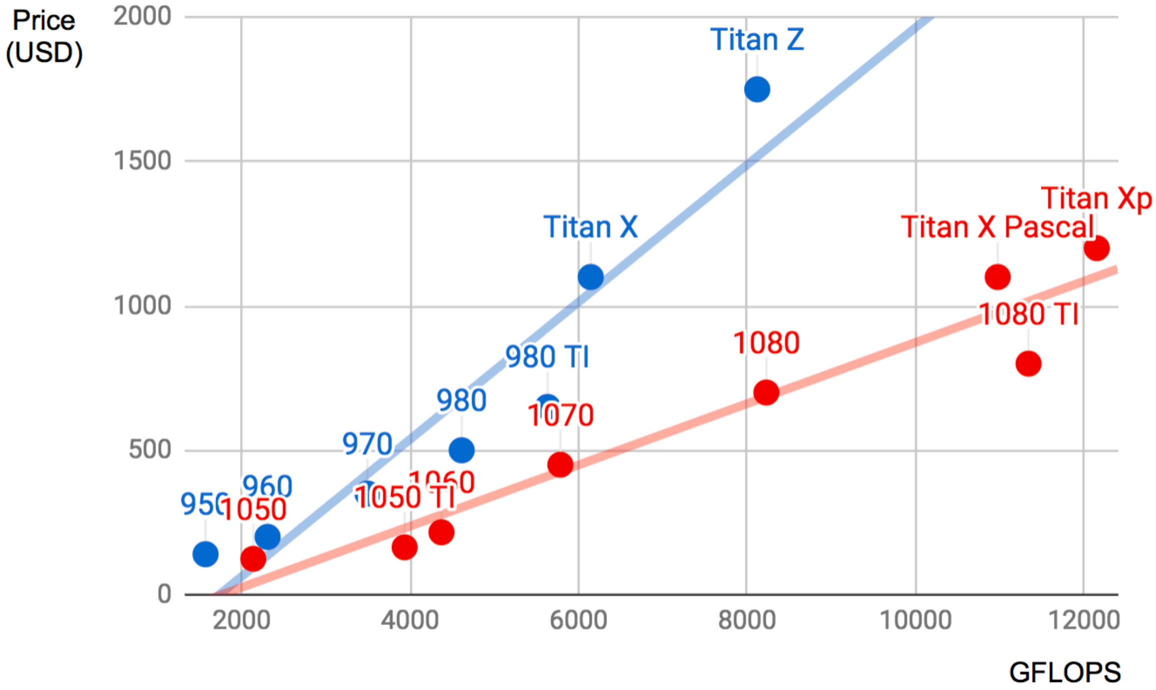


- 2584 cores, each core
    - Multiple arithmetic units
- 3MB L2 cache
- 480 GB/s memory bandwidth

Improve GPU Utilization

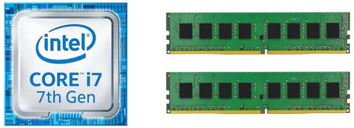
- Parallelization
    - Uses thousands of threads
- Memory locality
    - Simple cache architecture and small cache size
- Simple control flows
    - Very limited support
    - Large synchronization

Buy GPUs



- Each series improve the previous one
  - Buy new models
- Price is linear to power in a series

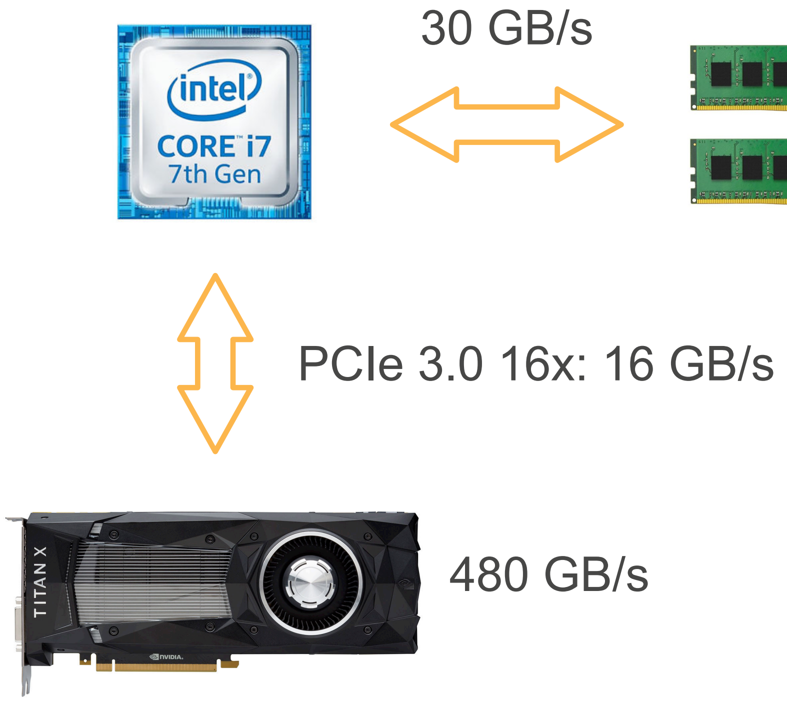
CPU vs GPU



# Cores	6 / 64	2K / 4K
TFLOPS	0.2 / 1	10 / 100
Memory size	32 GB / 1 TB	16 GB / 32 GB
Memory bandwidth	30 GB/s / 100 GB/s	400 GB/s / 1 TB/s
Control flow	Strong	Weak

Typical / High End

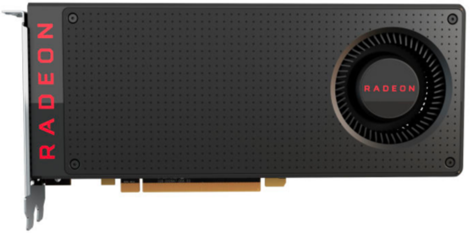
CPU/GPU Bandwidth



- Do not copy data between GPU and CPU frequently
  - Limited PCIe bandwidth
  - Synchronization overhead

More CPUs and GPUs

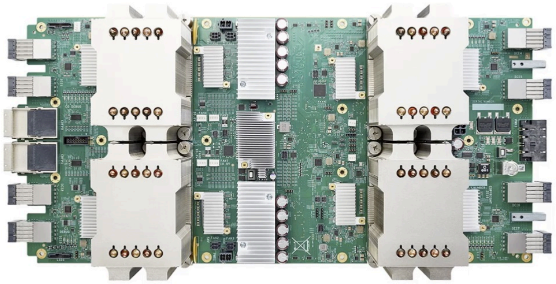
- CPU
  - Desktop/Server: AMD
  - Edge: ARM
- GPU
  - Desktop/Server GPU: AMD, Intel
  - Edge: ARM, Qualcomm



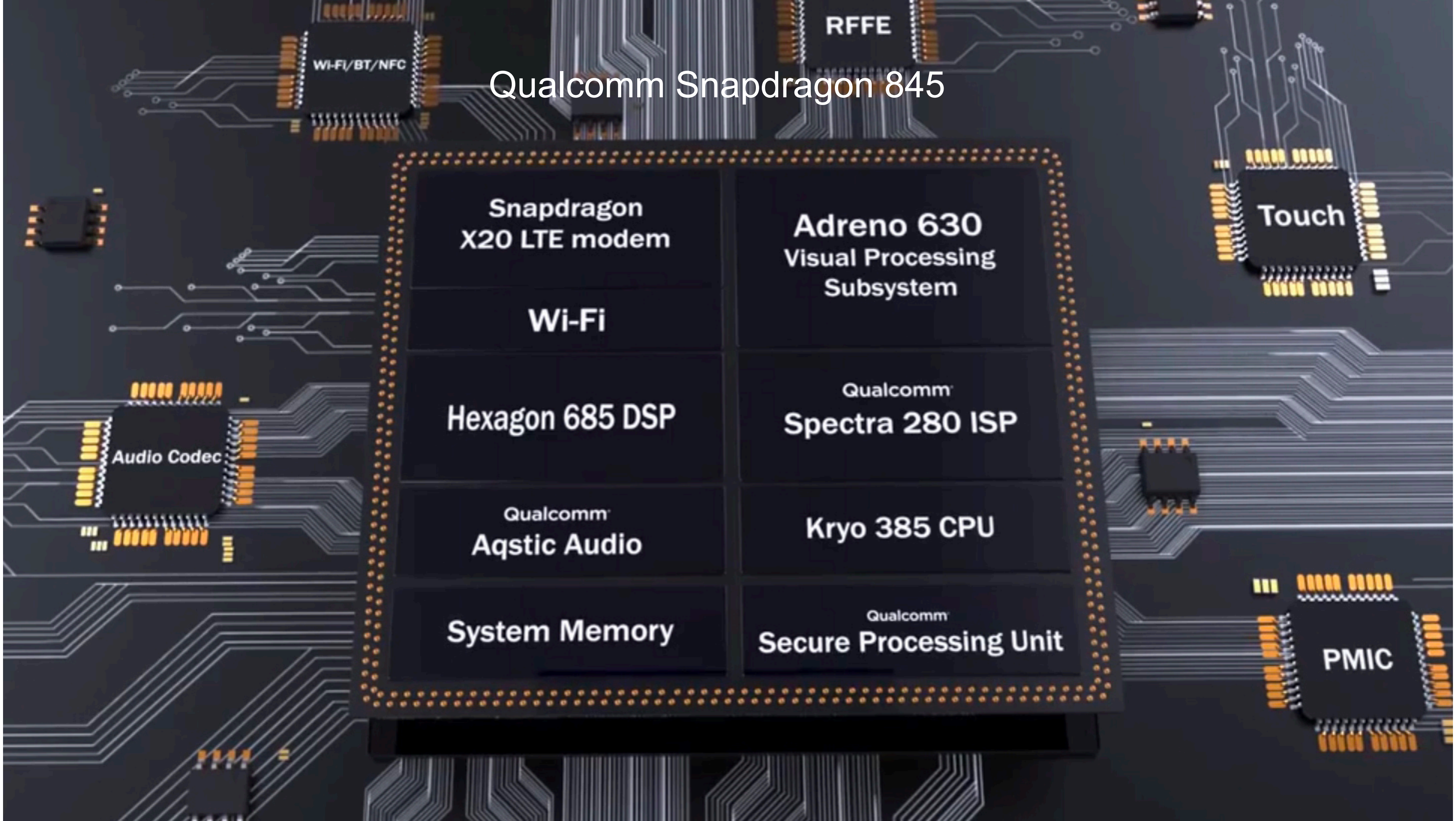
Programming on CPU/GPU

- CPU: C++ or any other high-performance language
    - Mature compilers with performance guaranteed
- GPU
    - CUDA for Nvidia
        - Rich features, mature complier and driver
    - OpenCL for others
        - Quality varies for chip vendors

More Promising Hardware



Qualcomm Snapdragon 845



Digital Signal Processor

- Designed for digital processing algorithms
  - Dot product, Convolution, FFT
- Low power and high performance
  - 5x faster than mobile GPUs, uses less power
- VLIW: Very long instruction word
  - Hundreds multiply-accumulates in a single instruction
- Hard to program and debug
- Compiler toolchain quality varies from chip vendors

Field-Programmable Gate Array (FPGA)

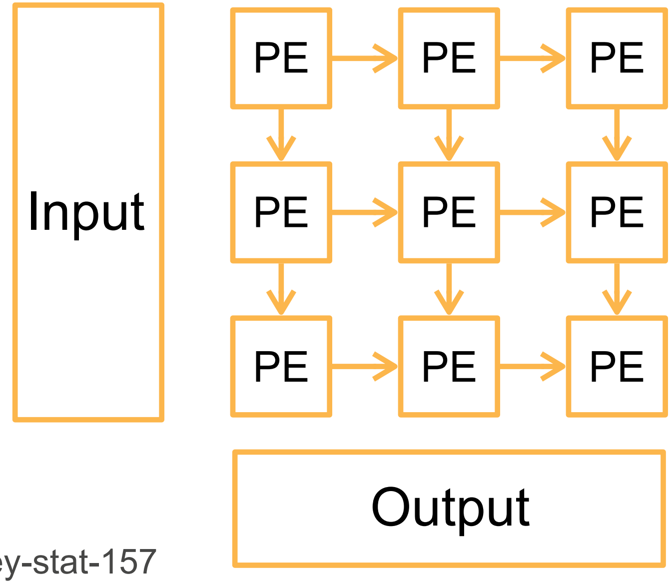
- Contains a large number of programmable logic blocks and reconfigurable interconnects
- Can be configured to perform complex functions
    - Common languages: VHDL and Verilog
- Often has high efficiency than general purpose hardware
- Toolchain qualities vary
- Each “compilation” may take several hours

AI ASIC

- Hot topic in deep learning
  - Every major company is building their chips (Intel, Qualcomm, Google, Amazon, Facebook, …)
- Google TPU is a landmark chip
  - Matches high-end Nvidia GPU’s performance, but 10x-100x cheaper
  - Widely deployed in Google
  - The core is a systolic array

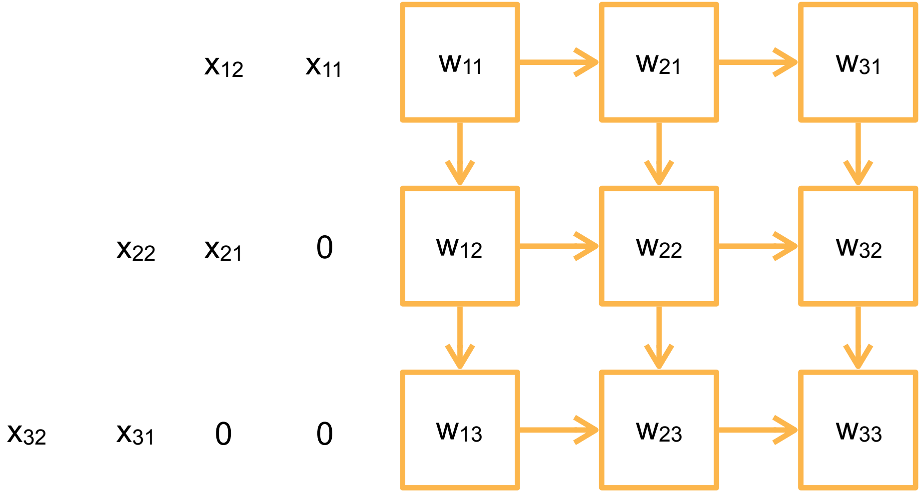
Systolic Array

- An array of processing elements (PE)
- Good at performing matrix-matrix multiplication
- Relative easy/cheaper to build



Matrix Multiplication with Systolic Array

Time 0



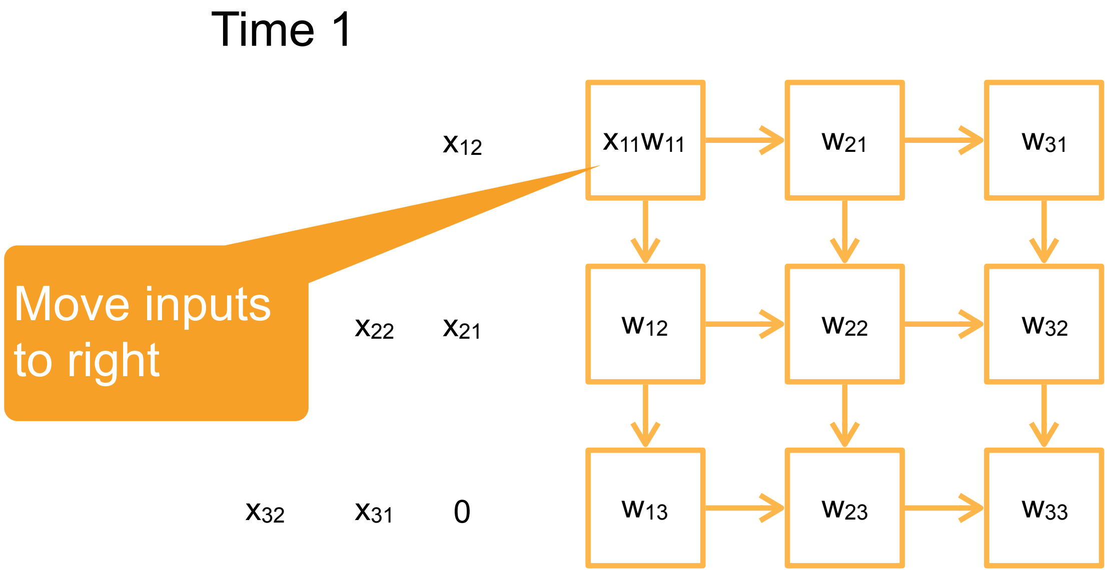
$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

3x2 3x3 3x2

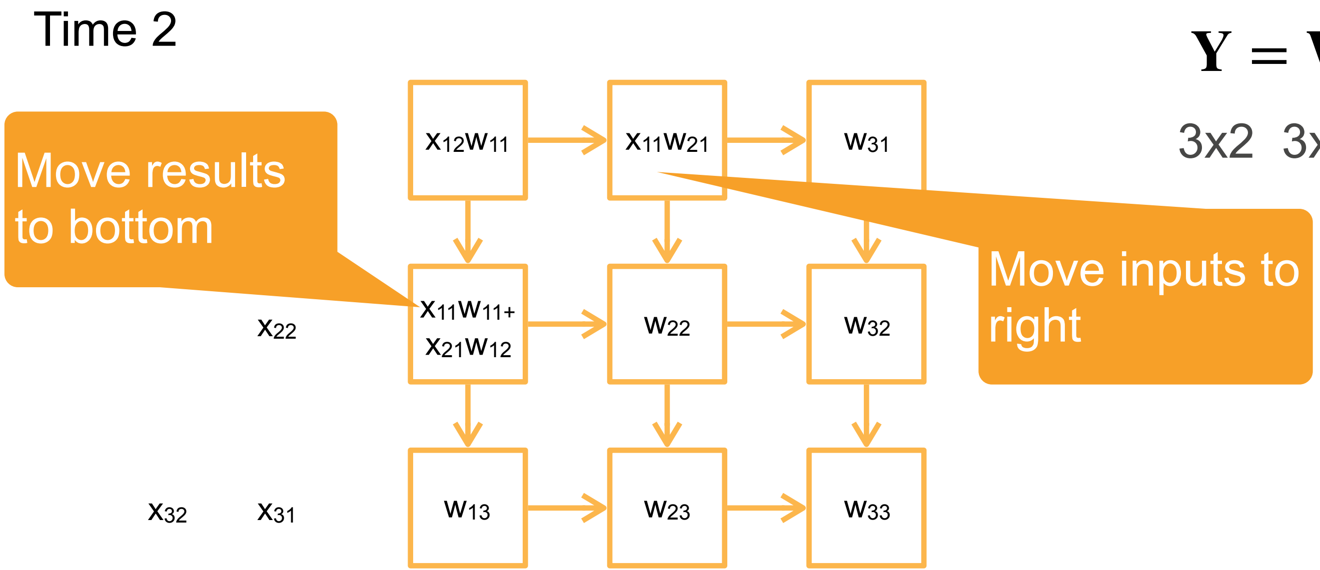
Matrix Multiplication with Systolic Array

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

3x2 3x3 3x2



Matrix Multiplication with Systolic Array



$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

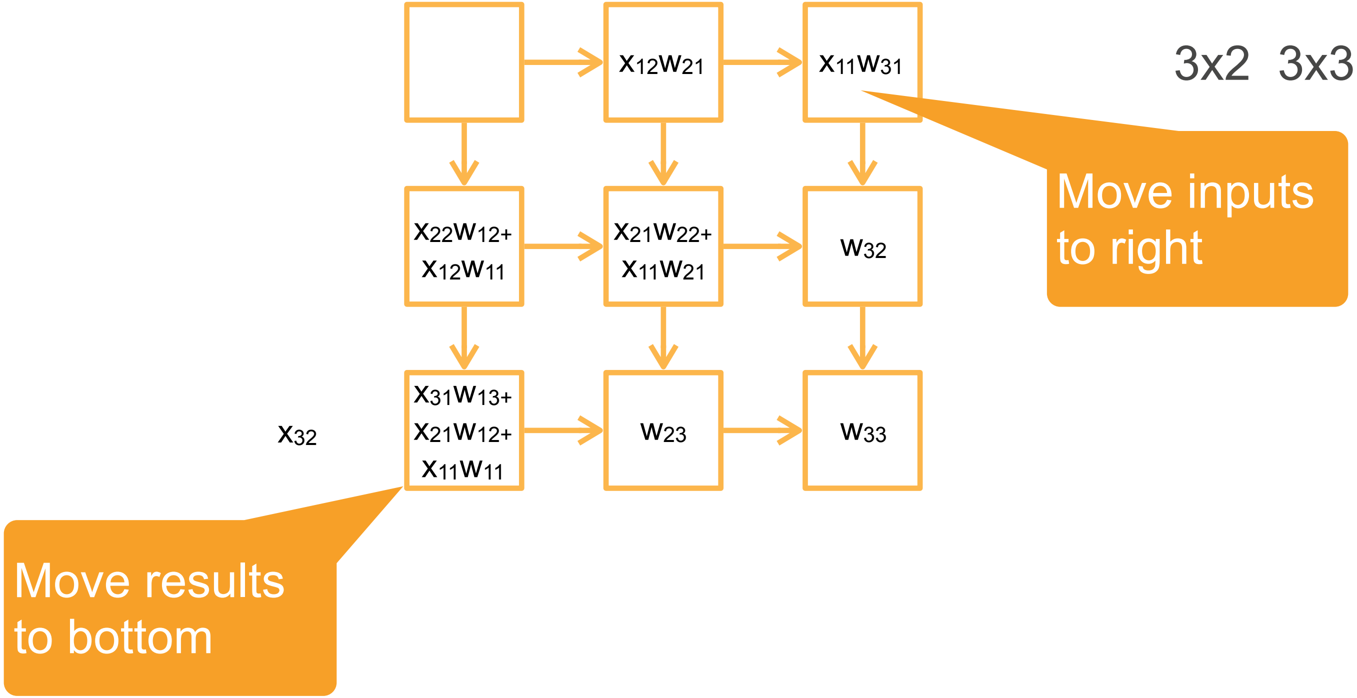
3x2 3x3 3x2

Matrix Multiplication with Systolic Array

Time 3

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

3x2 3x3 3x2

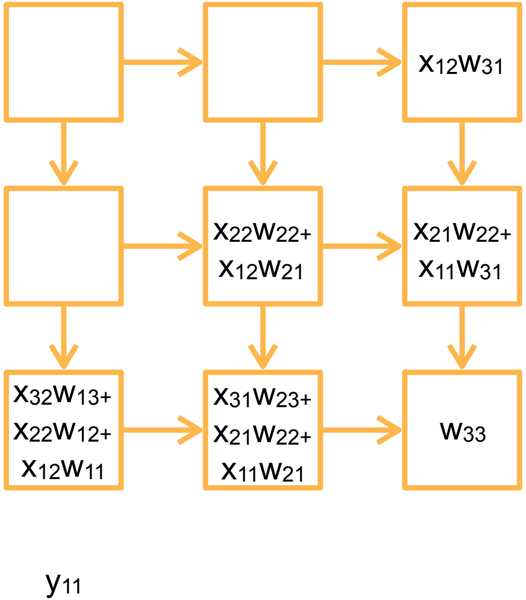


Matrix Multiplication with Systolic Array

Time 4

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

3x2 3x3 3x2

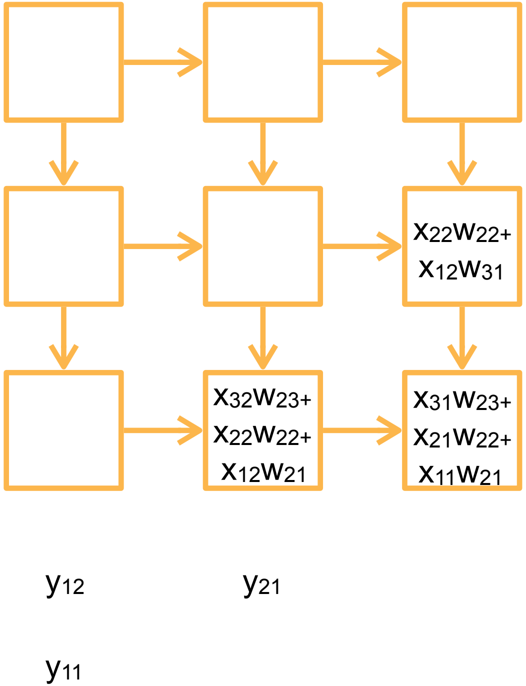


Matrix Multiplication with Systolic Array

Time 5

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

3x2 3x3 3x2

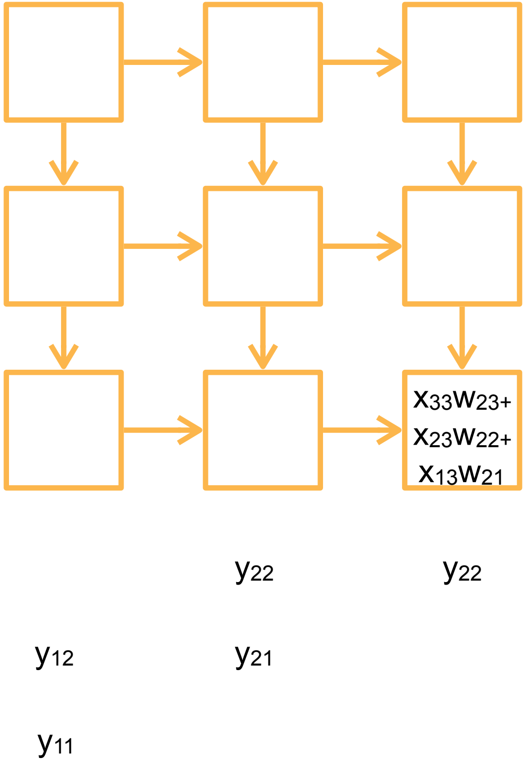


Matrix Multiplication with Systolic Array

Time 6

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

3x2 3x3 3x2

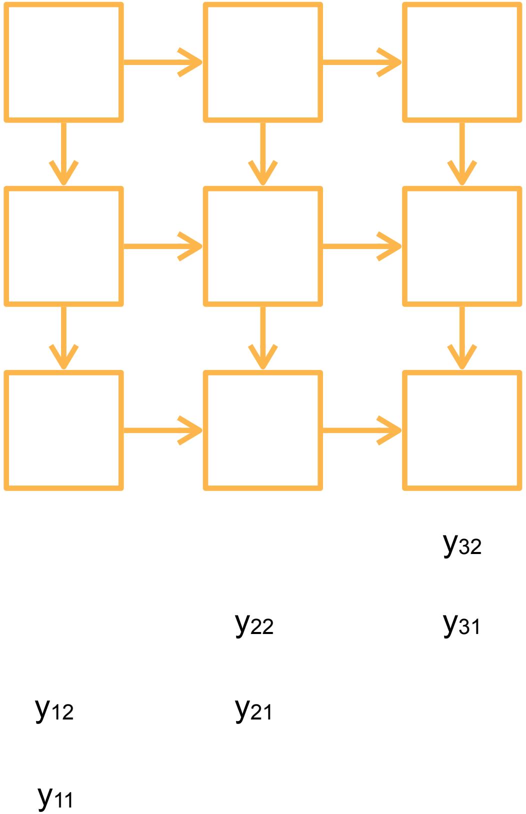


Matrix Multiplication with Systolic Array

Time 7

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

3x2 3x3 3x2



Systolic Array

- For general size matrix multiplication, slice and pad inputs to match the SA size
- Batch inputs to reduce the latency cost
- ASIC has other dedicated components for other NN operations, such as sigmoid


Flexibility and Ease of Use
intel
CORE i7
7th Gen
TITAN X
Hexagon 685 DSP
Xilinx®
FPGA
Performance & Power Efficiency