Hardware Support for Neural Networks and Deep Learning

Taylor Bergeron, Clayton Dembski, Nathan Dennler, Mikayla Fischler, Zoraver Kang, Matthew LeMay, Floris van Rossum

Overview

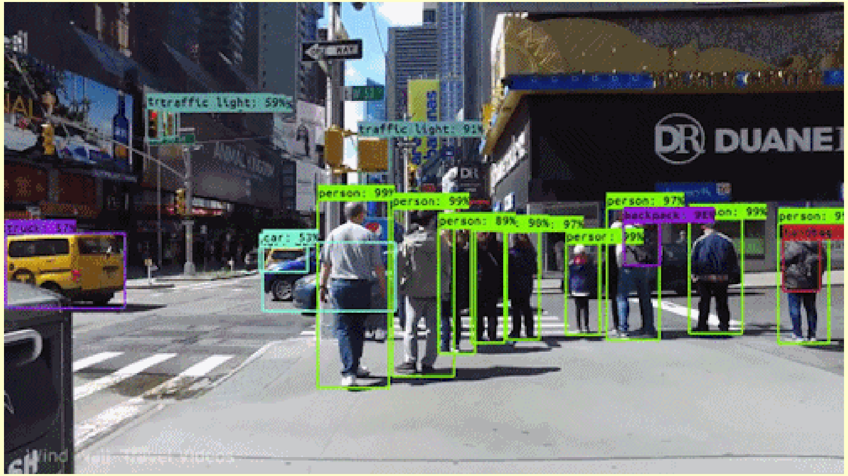
1. What are Neural Networks?
2. How can we speed up Neural Network training?
3. What kind of hardware exists that supports neural networks?
4. What is the future of hardware support for Neural Networks?

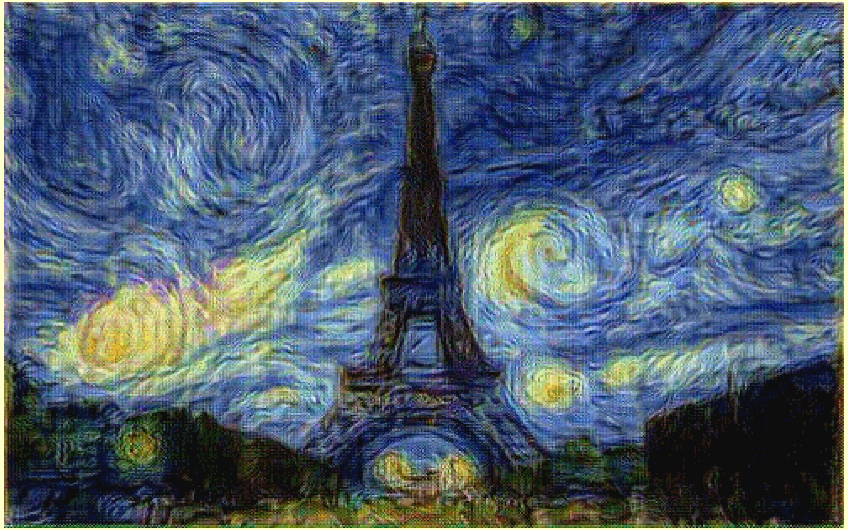
Overview

1. **What are Neural Networks?**
    a. Use-Cases
    b. How they classify
    c. How they are trained
2. **How can we speed up Neural Network training?**
3. **What kind of hardware exists that supports neural networks?**
4. **What is the future of hardware support for Neural Networks?**

Uses

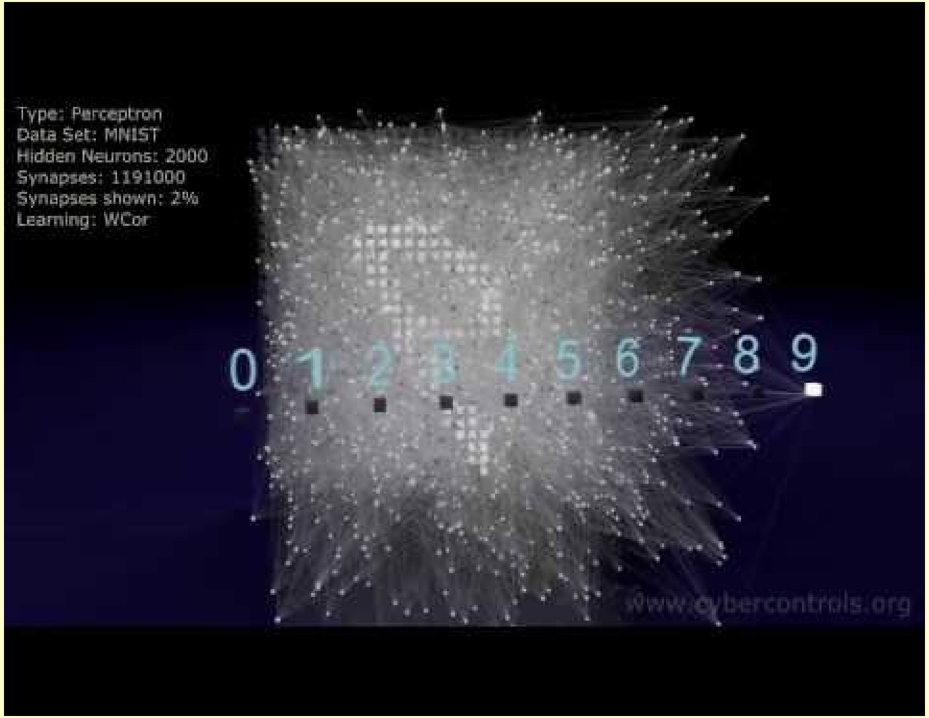
- Image Processing and Object Detection
    - Facial Recognition
    - Object Labeling
        - Cars -- see self driving cars
        - Dogs, cats, trees, ect
    - Generating new data from given data
        - Generative Additive Networks
- Forecasting
    - Predicting weather
    - Predicting trends in stocks
- Natural Language Processing
    - Siri, Ok Google, Alexa





What are Neural Networks?

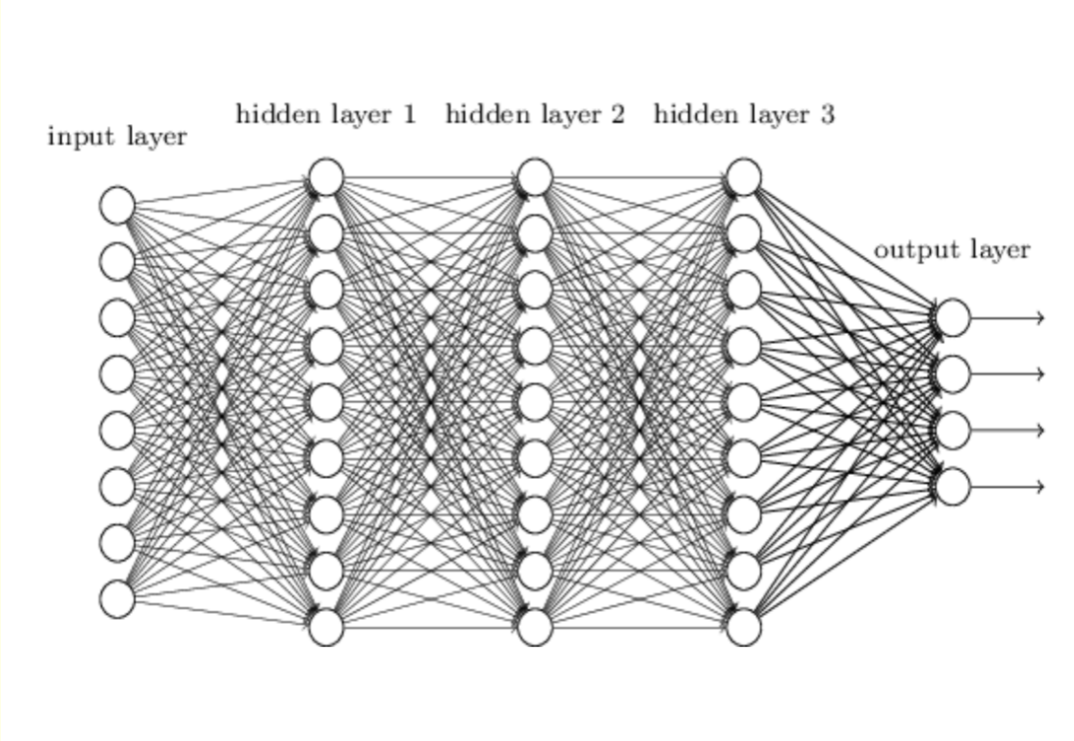
- An umbrella term for a series of Machine Learning Algorithms that help us Cluster and Classify data using methods of statistics and supervised learning
- Less formally: a model trained to recognize trends and patterns in data, and use these trends to predict the results of new data
- Example was trained on MNIST:
  - Dataset made up of handwritten numbers and the corresponding label
  - Goal is number recognition



How does a Neural Network Function?

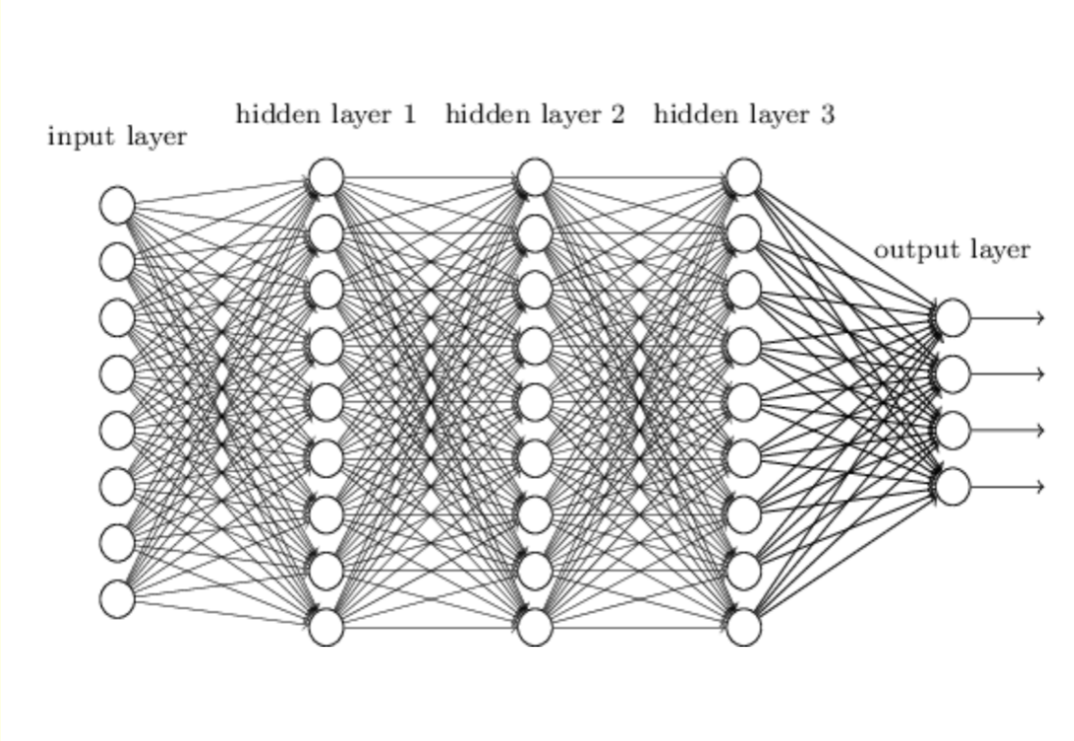
Made up of

- **Input Layer**
  - This is where data is entered
- **Hidden layers**
  - This is where the data is processed
  - The number of layers and feed into method are application specific
- **Output Layer**
  - This is where the prediction is made
- Each Layer is made up of perceptrons
  - These take in individual pieces of data and classify it in some novel way
- We can process each perception in a given layer ***in parallel***



Example

- **Input Layer**
  - A 48 by 48 array of float pixel intensities representing the number 9
- **Hidden layer 1**
  - Data Classified as Either White or Black based on pixel intensity
- **Hidden Layer 2:**
  - Regions where black pixels meet white pixels classified as edges
- **Hidden layer 3**
  - Edges classified by relation as a straight line and a circle
- **Output Layer**
  - A circle over a straight line classified as a 9. Output the value TRUE at the output node corresponding to 9

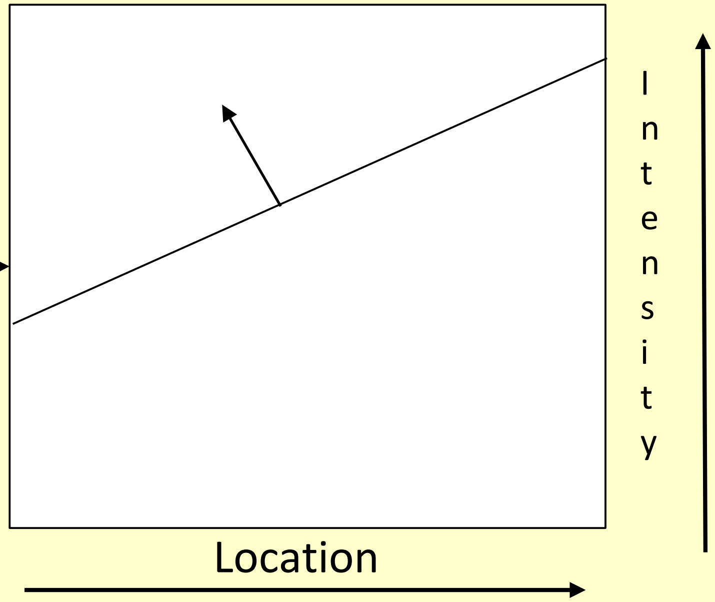


For this example, each Data Point is made up of an X location and a Pixel Intensity

Our classification function is linear and set by a decision boundary of f(x) = mx +b
m and b are our weights: they control the position and orientation of our boundary.
Anything above the line is classified as type 1
Anything below is classified as type 2

Data1, Data2, Data3, Data4, Data5 Data6, Data7

Feed the data into the perceptron



We can do this all **in Parallel** by feeding in our data as a matrix:

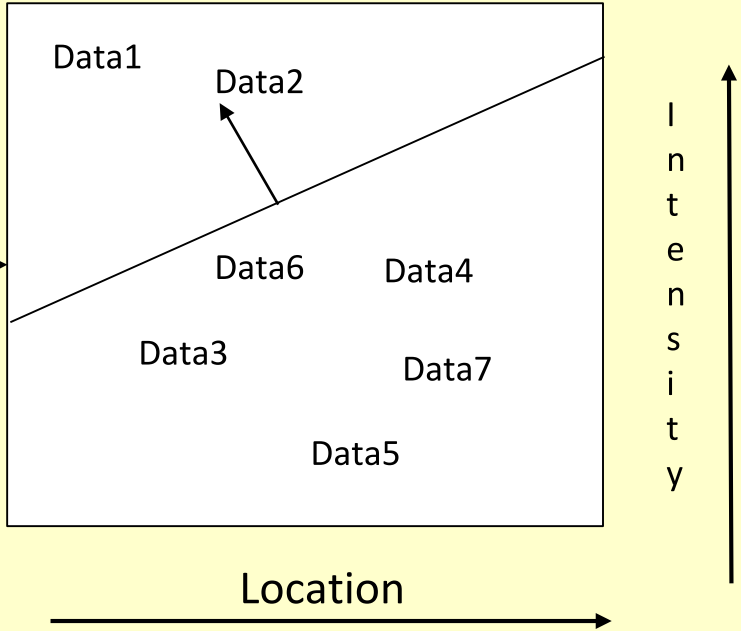
$$[\text{Data1, Data2, ...}] * \begin{bmatrix} W1, \\ W2 \\ \dots \end{bmatrix}$$

Data1, Data2, Data3, Data4, Data5 Data6, Data7

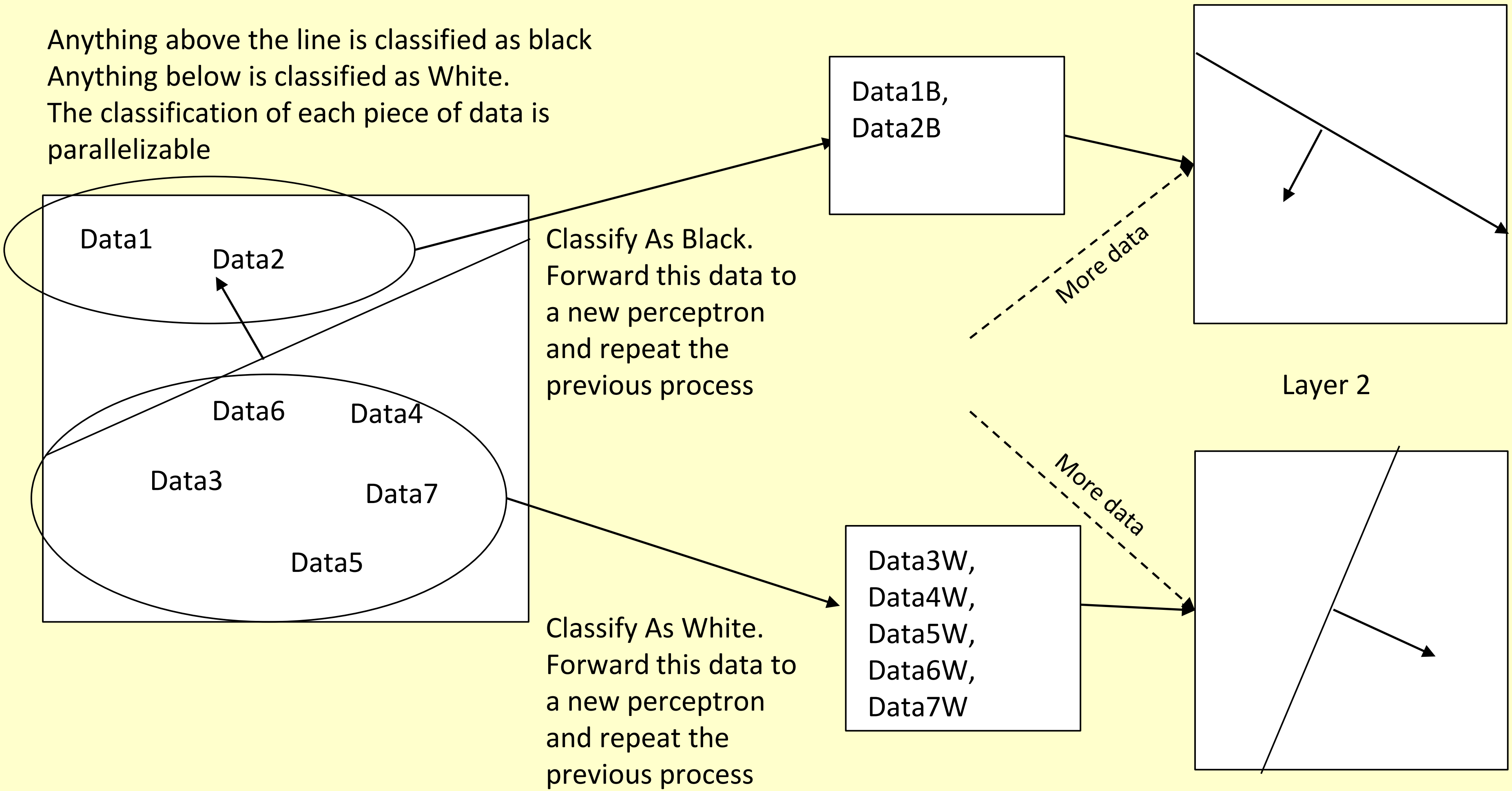
Feed the data into the perceptron

Anything above the line is classified as type 1
Anything below is classified as type 2

This is a Binary Classifier



Anything above the line is classified as black
Anything below is classified as White.
The classification of each piece of data is parallelizable

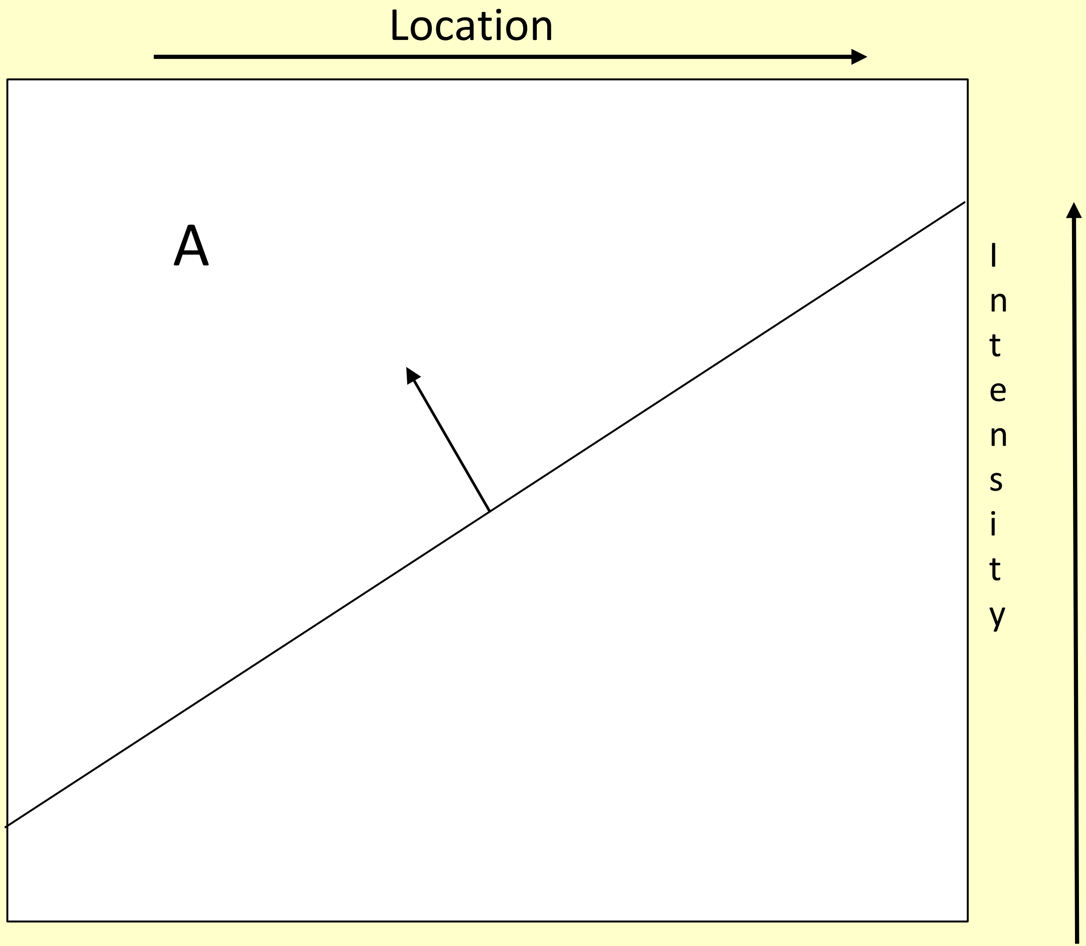


How does an individual perceptron classify the data given to it?

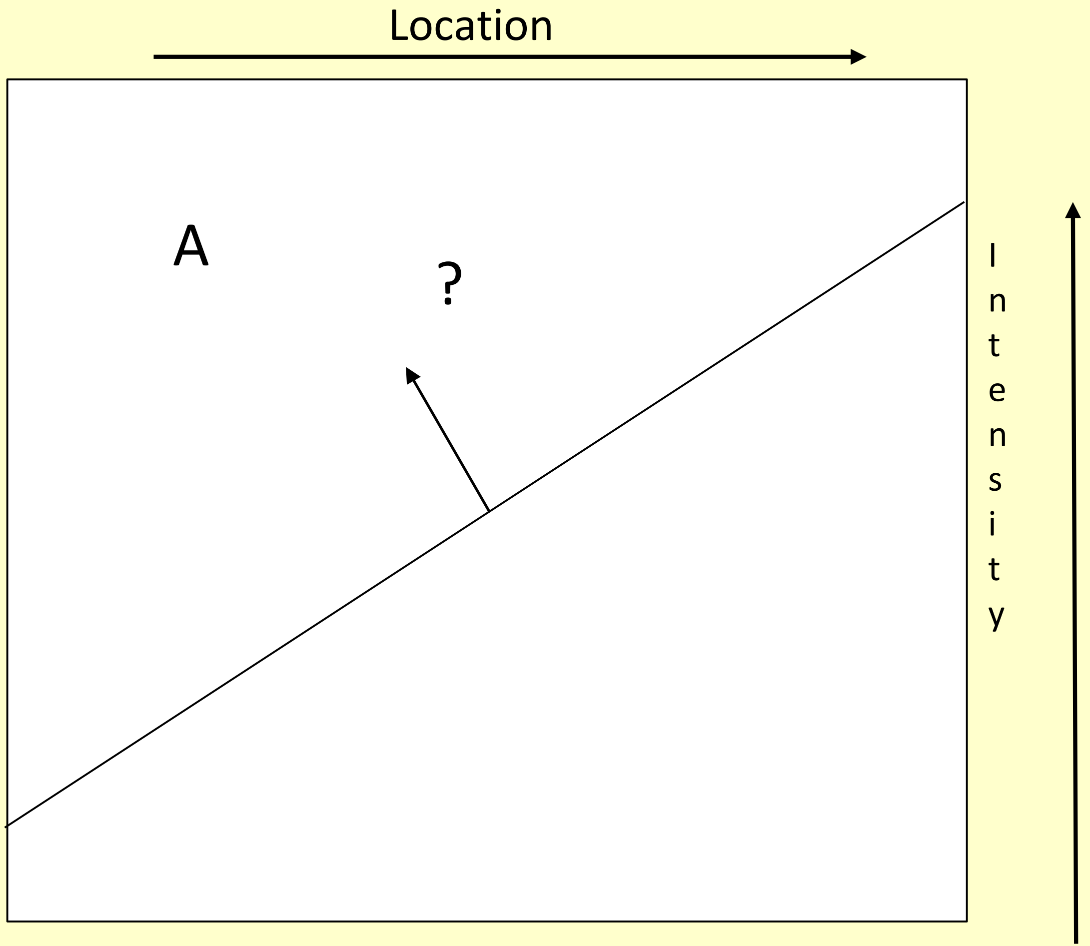
Through some classification (Mathematical) function

- The individual weights and parameters of each function are trained by giving the model real examples followed by the expected result of the example
- Simplest are linear classifiers
  - mx+b
  - This is the example that will be used in the slides to follow
- Can have polynomial and nonlinear classifiers
- Can have hard and soft margin classifiers (SVMs)
  - These are trained to split the data in a way that leaves the largest gap between the data points

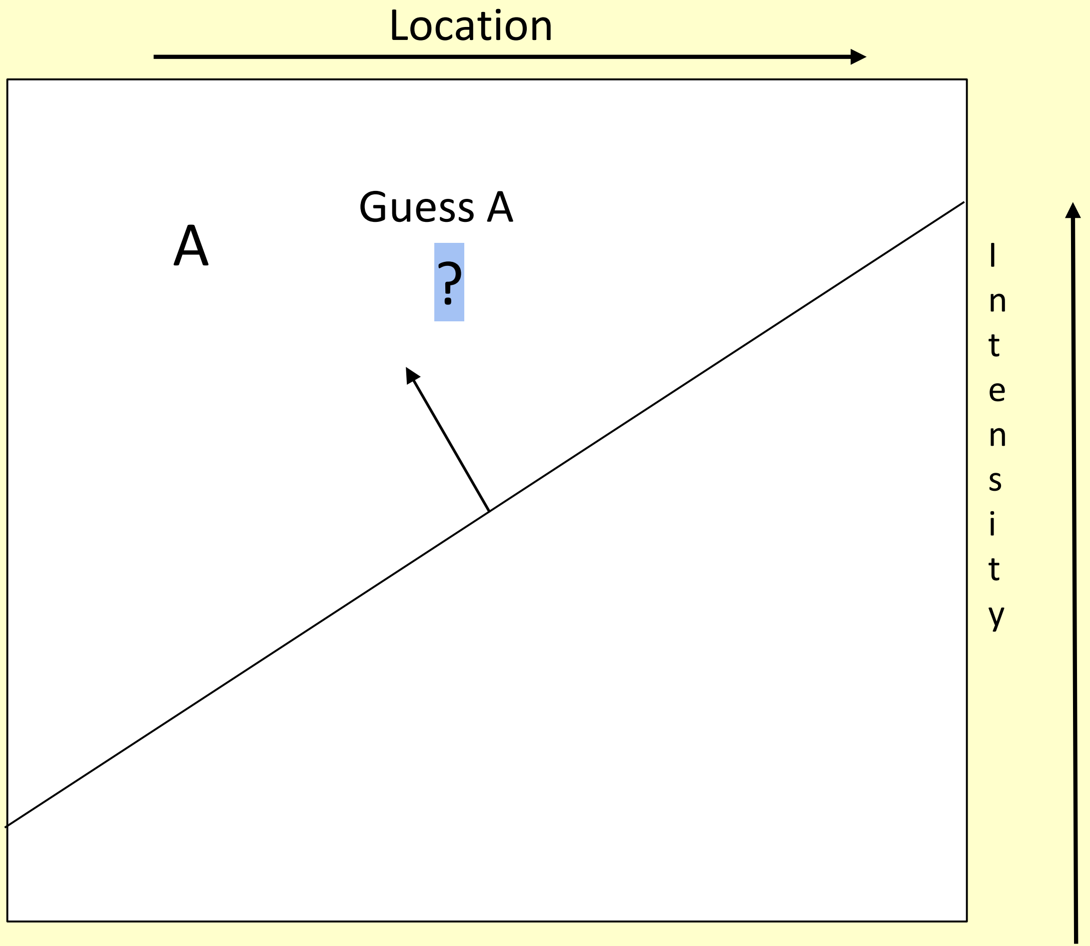
How a perceptron updates its data



We get a new piece of data

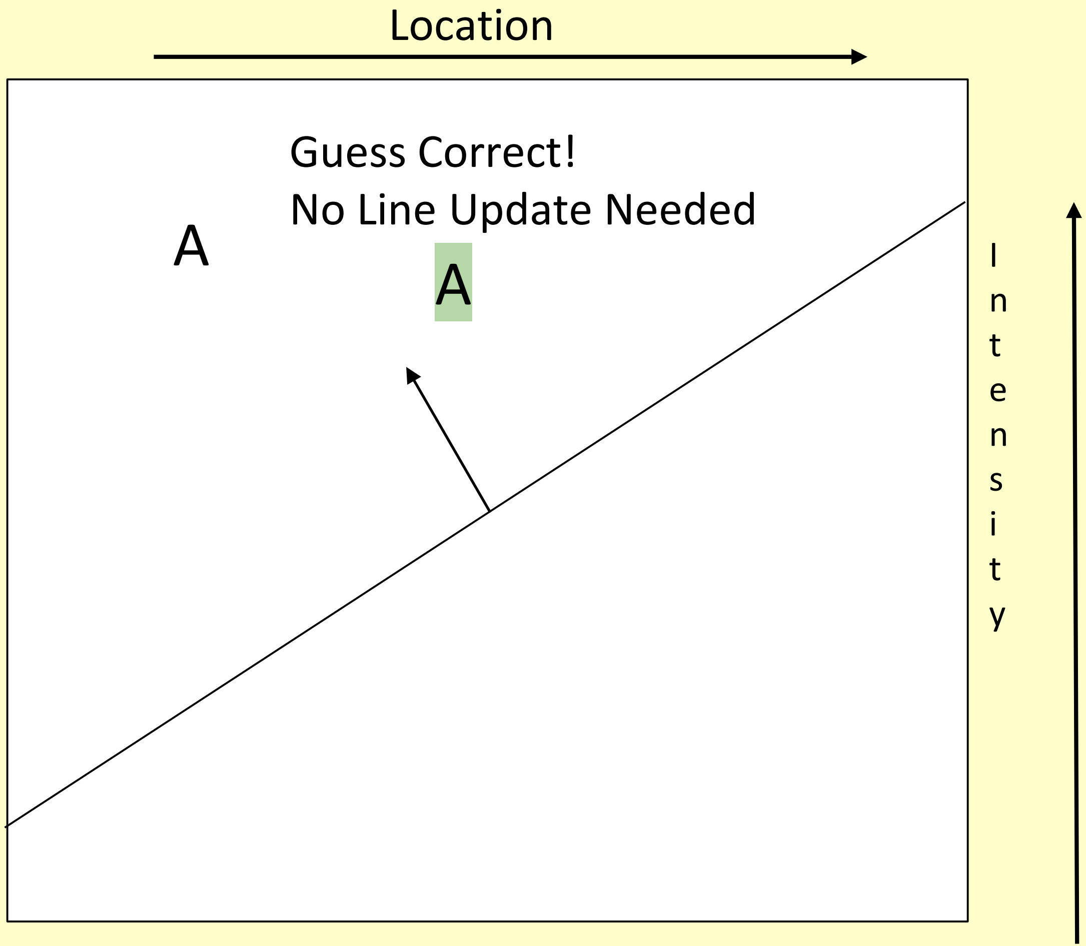


The perceptron guesses, based on its classifier, what the data is

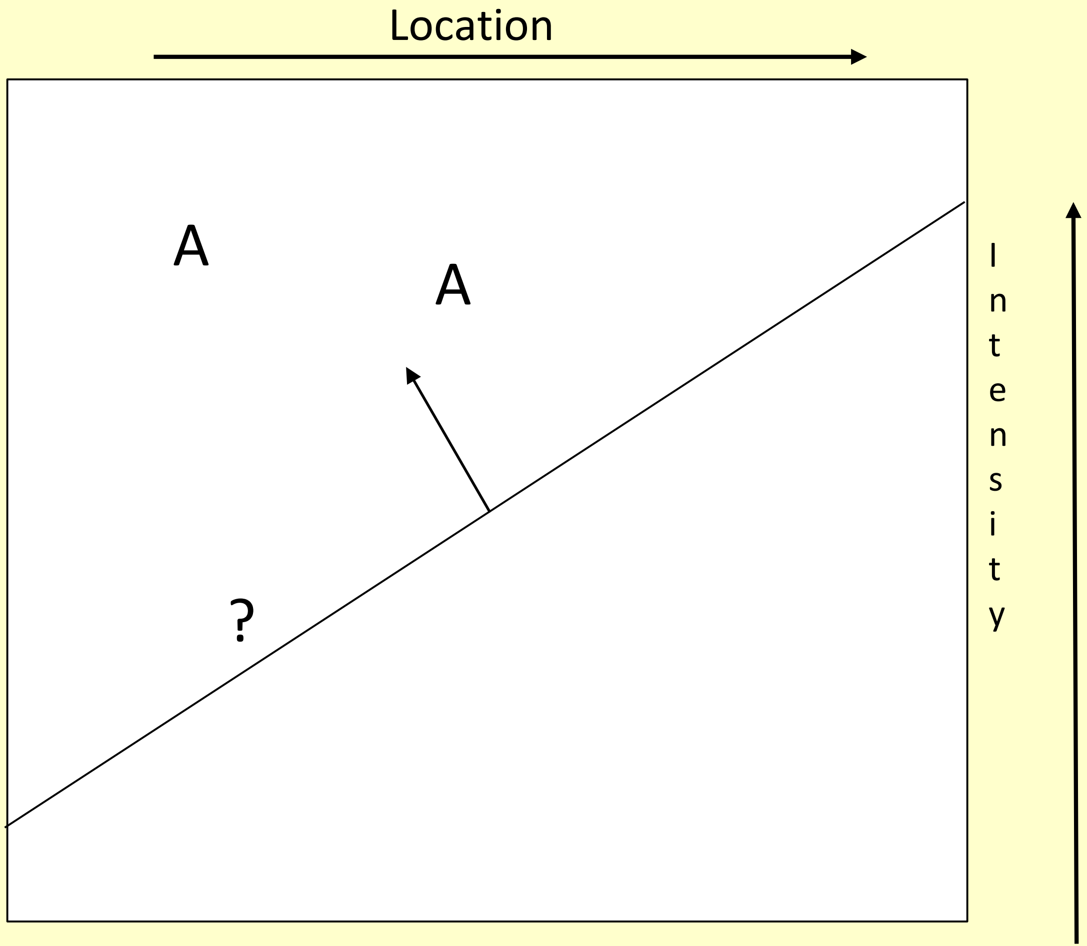


We inform the model if the guess was correct.

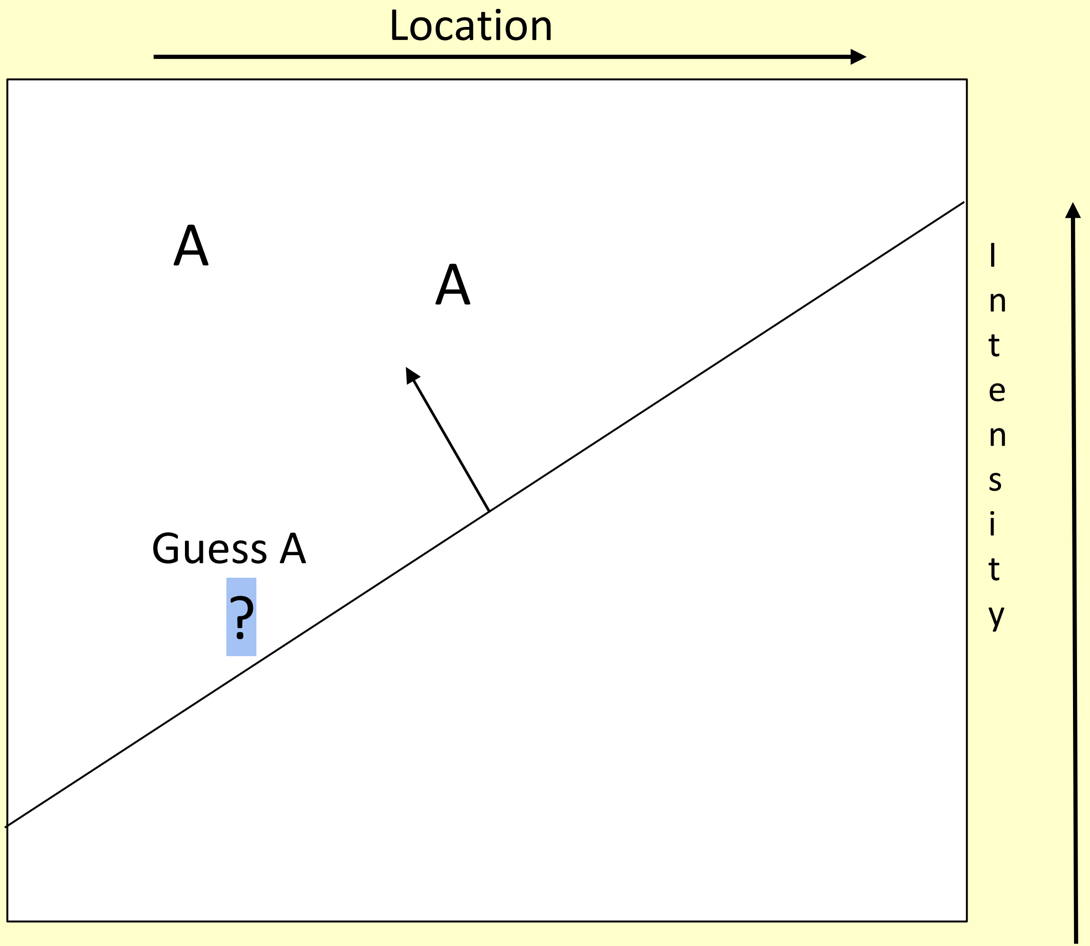
If it is, the model stays as it is



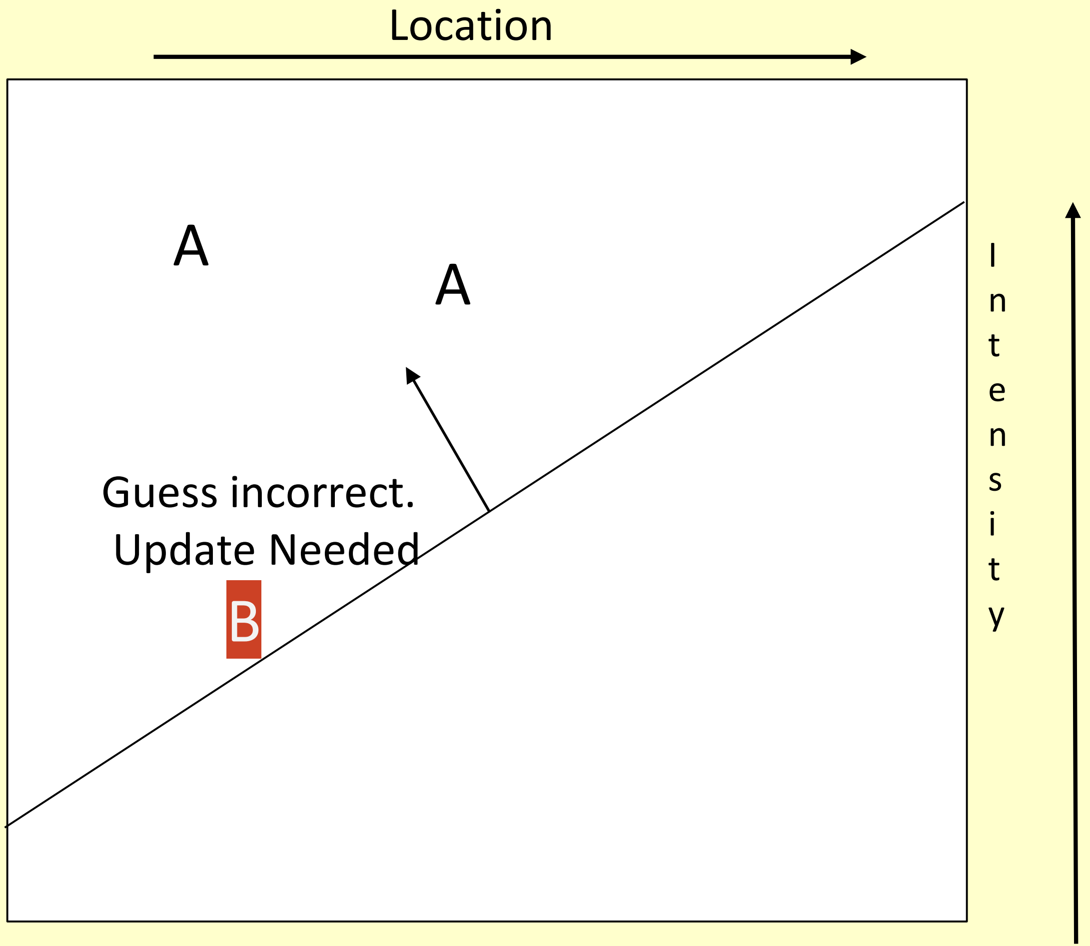
We get a new piece of data



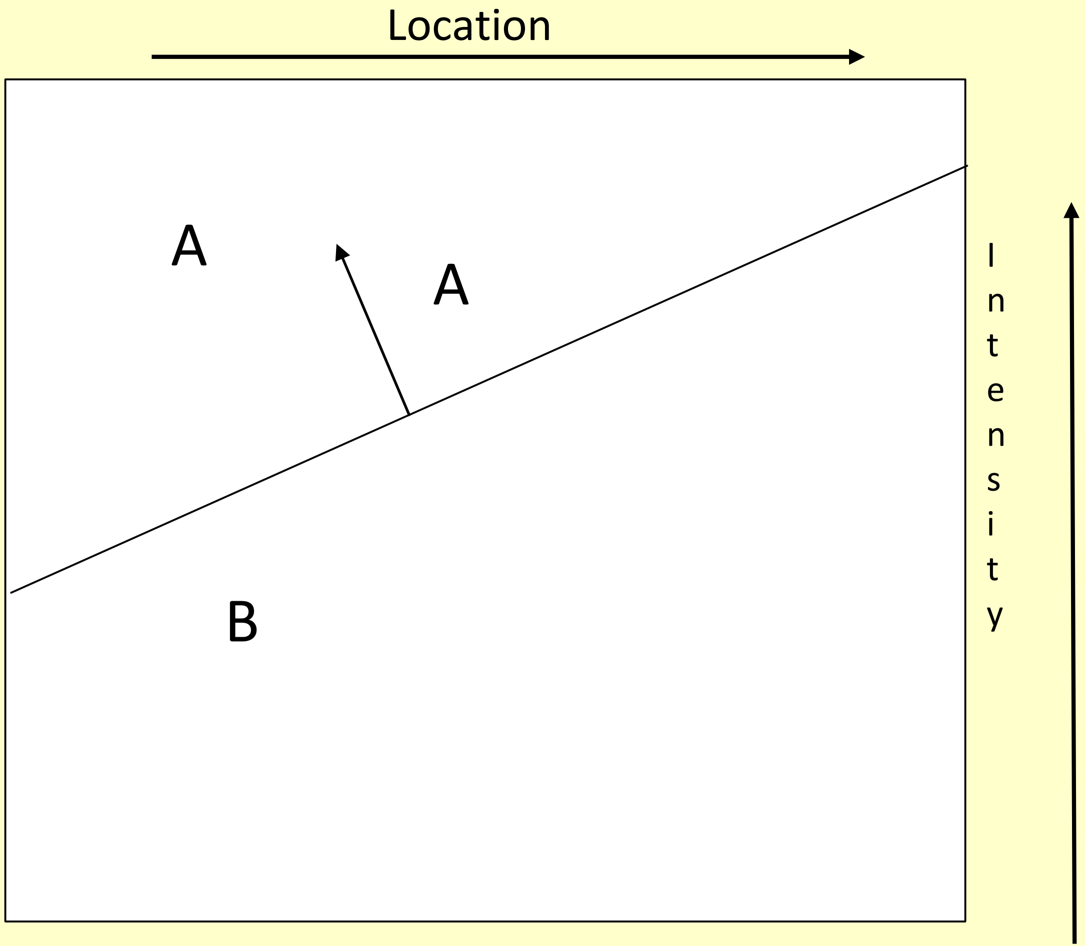
We guess its classification



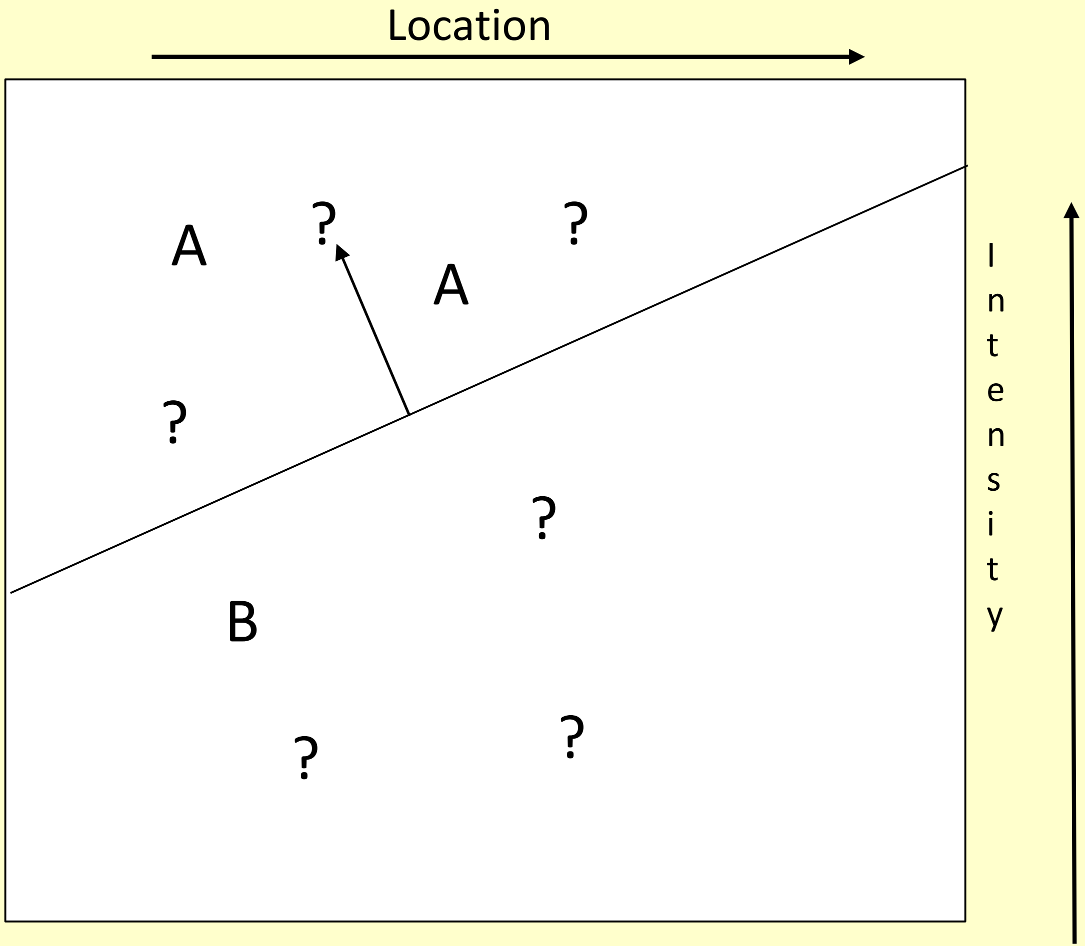
Guess is incorrect. We need to update our line



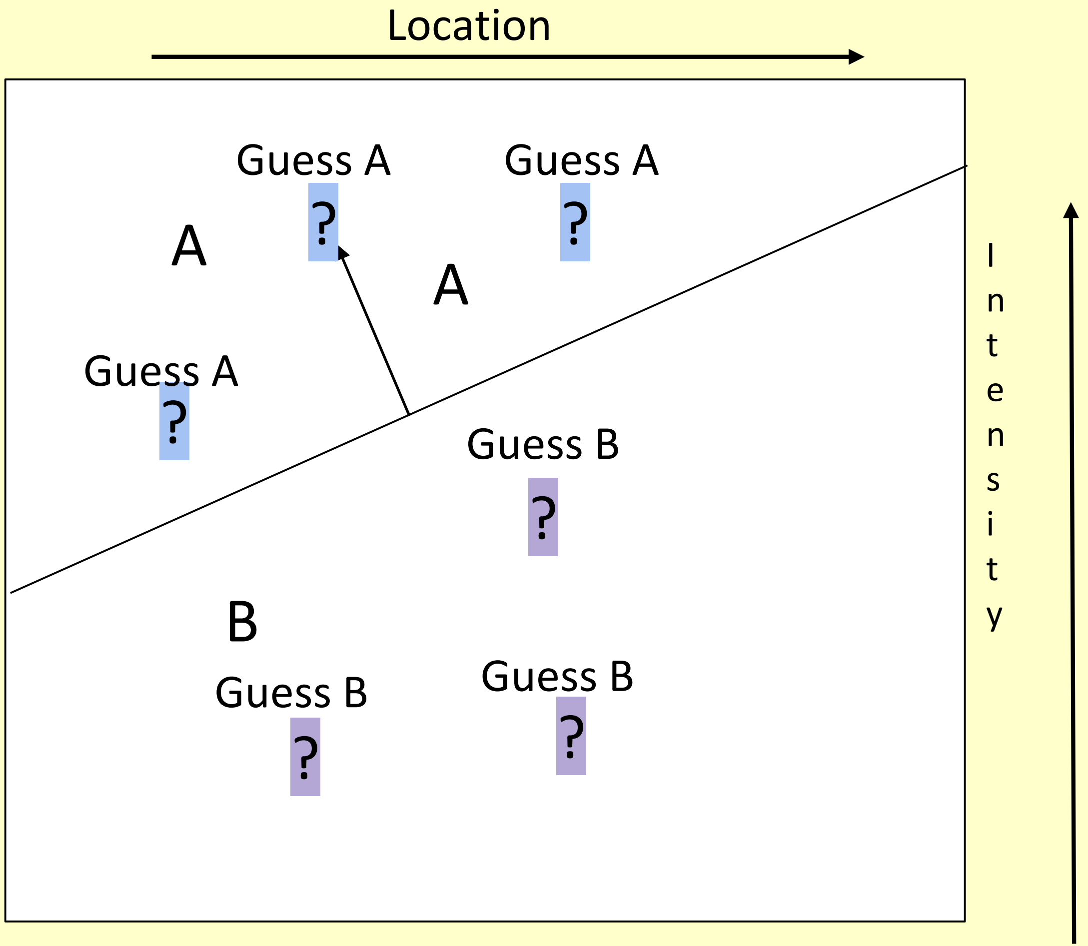
Line updates



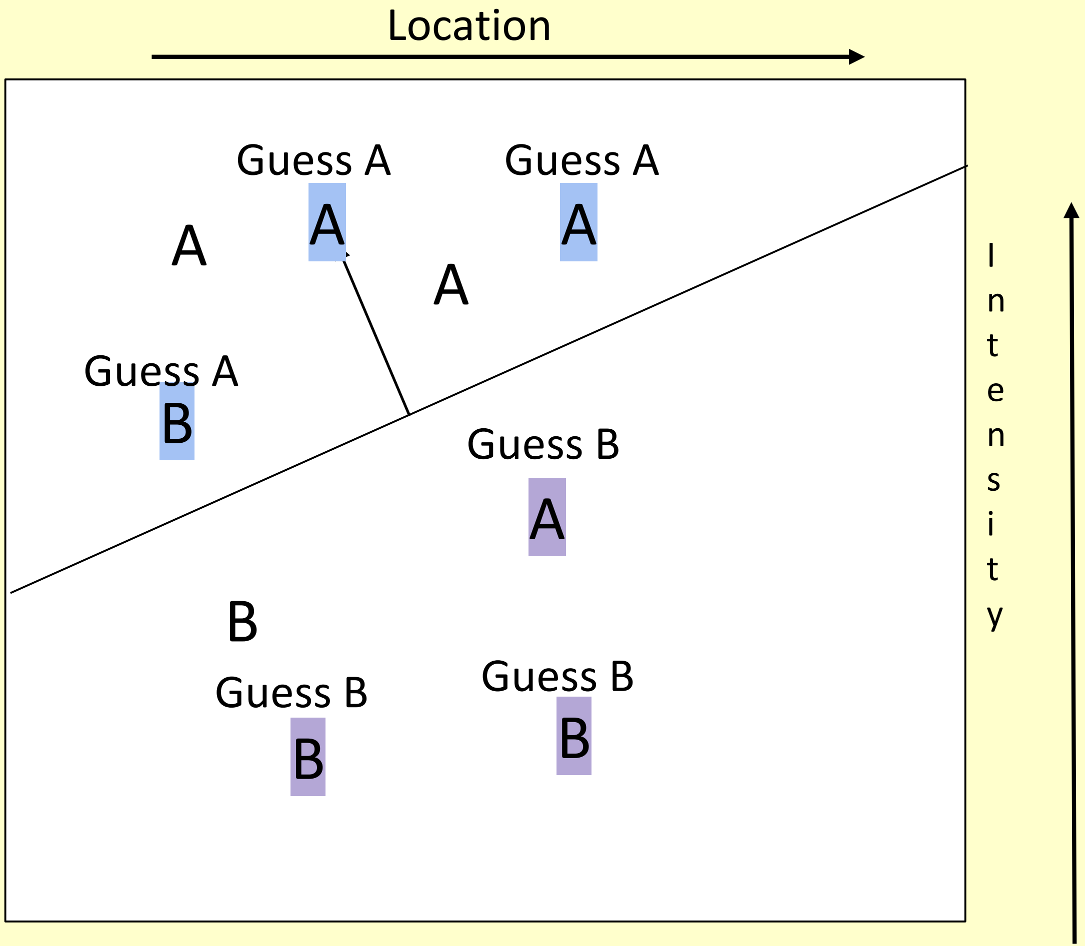
We don't need to just do one piece of data at a time



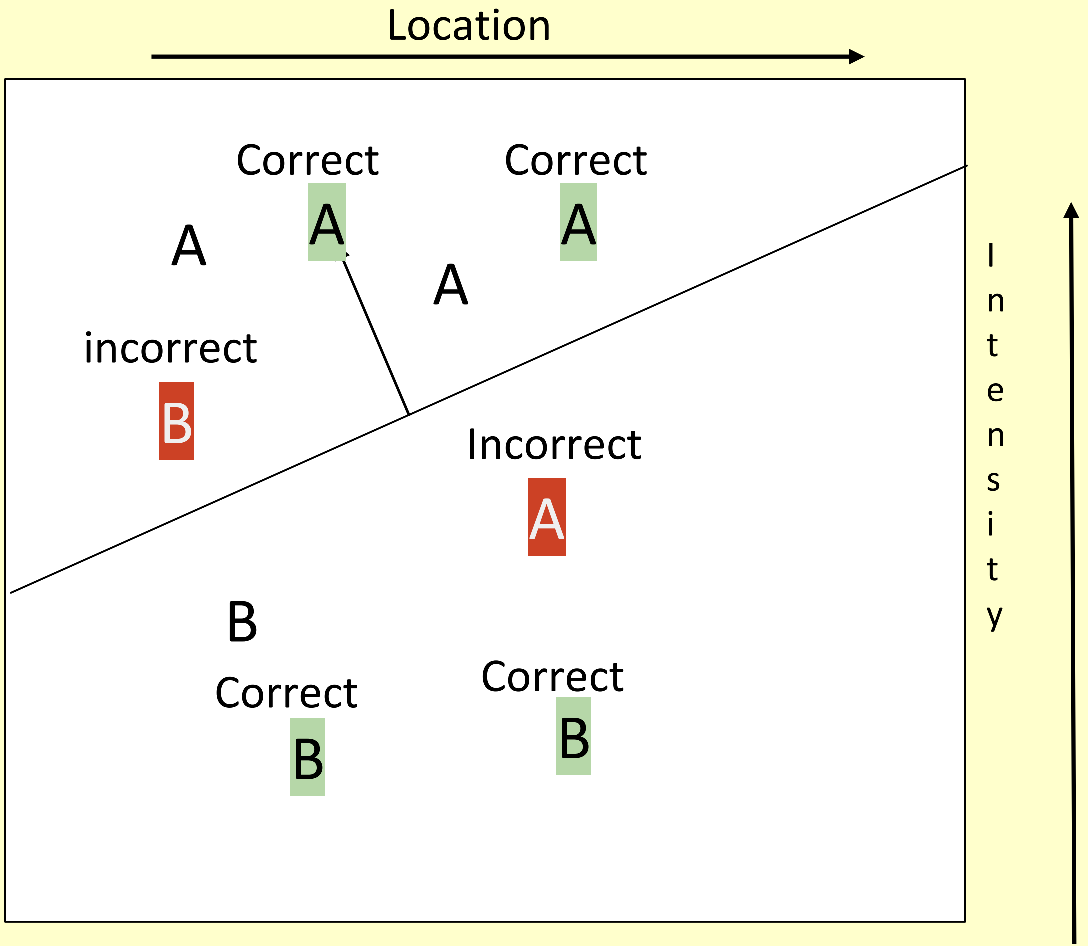
We don't need to just do one piece of data at a time



We don't need to just do one piece of data at a time

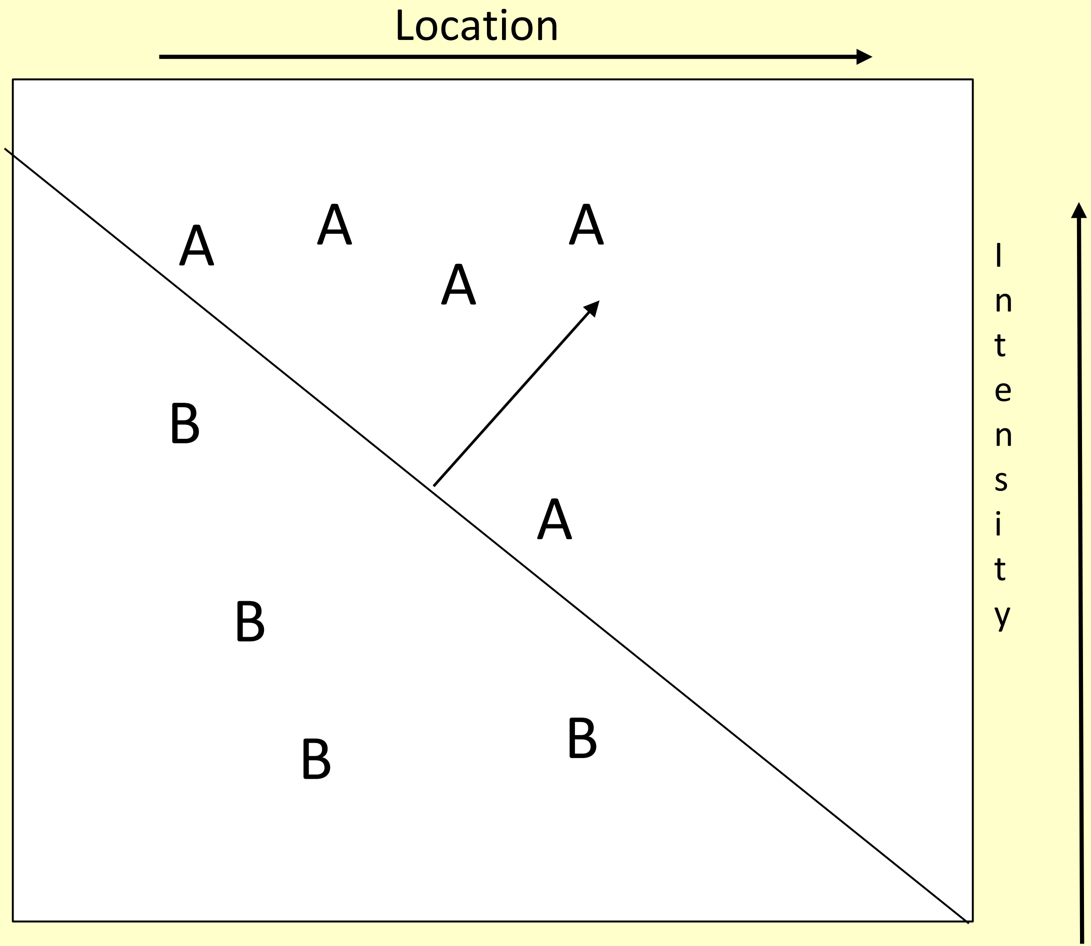


We don't need to just do one piece of data at a time



Update our linear separator

Because we can evaluate, check, and update our separator for large amounts of data at the same time, this is a chance for parallelization



How Do We Train a Perceptron?

- The process of updating our linear separator is done through an algorithm called gradient descent:

- To update our line, we first create a Loss Function.
  This function tells us how accurate our measure is, and is essentially some function of the ratio of the number we have correct vs incorrect compared to the relative positions of the actual markers. There are multiple loss functions for different ways of evaluating your data

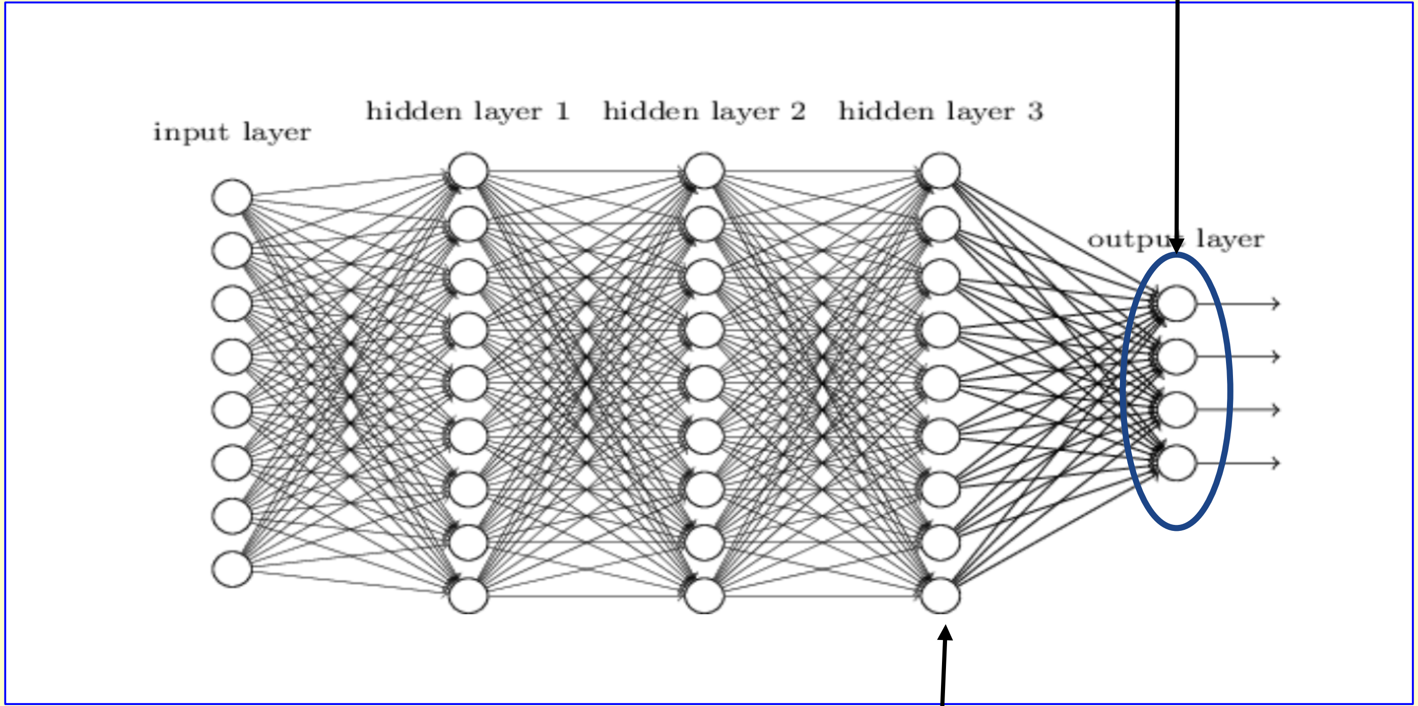
- This function has a local minimum, which represents the smallest loss, or smallest misclassification possible.

- If you remember from your calculus classes, we can find this local extrema by analyzing the slope of the function: we take and analyze the derivative of our loss function.

- Less mathematically, this is what we do:

How Do We Train The **Entire System**

We know what the output
should be for this layer

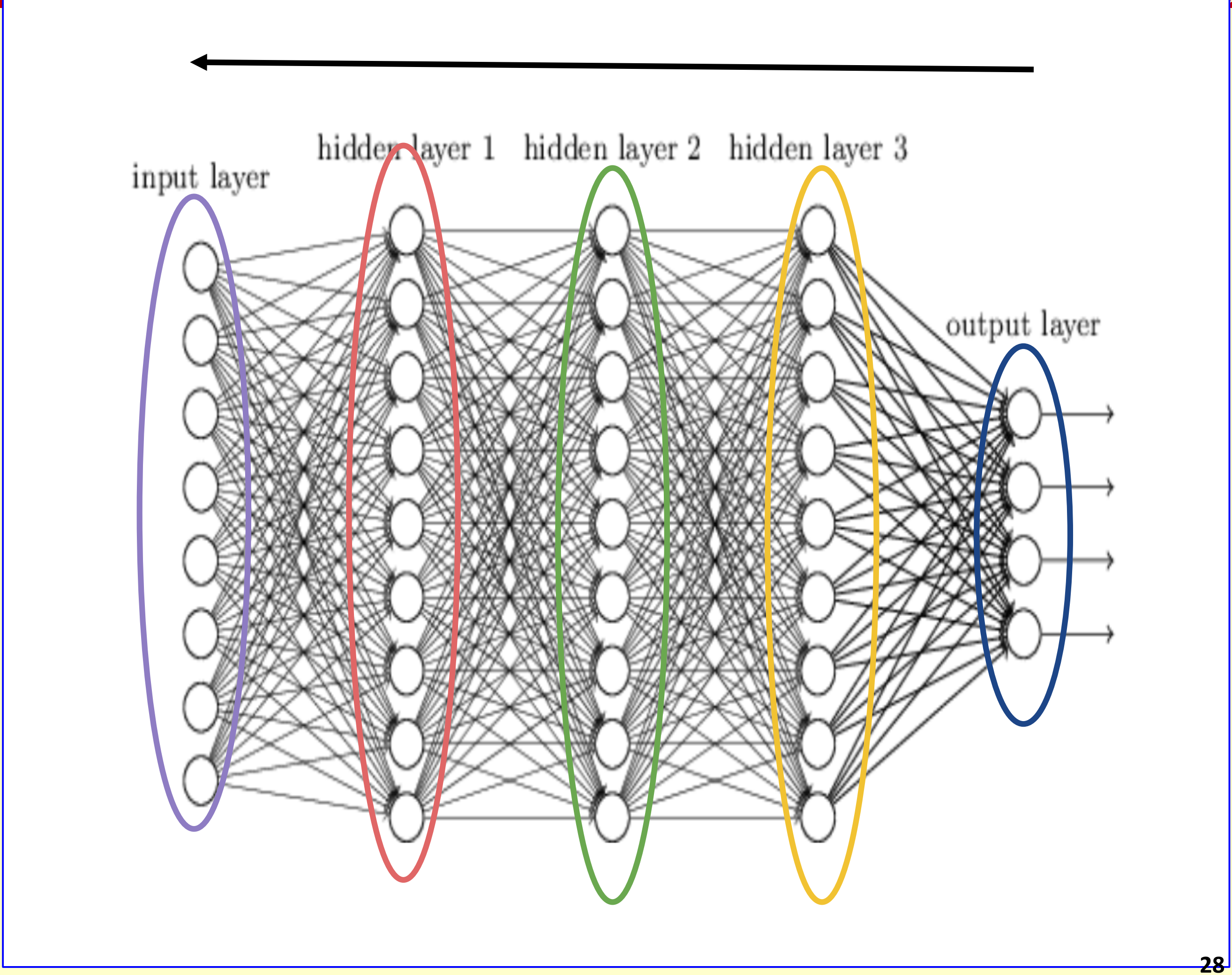


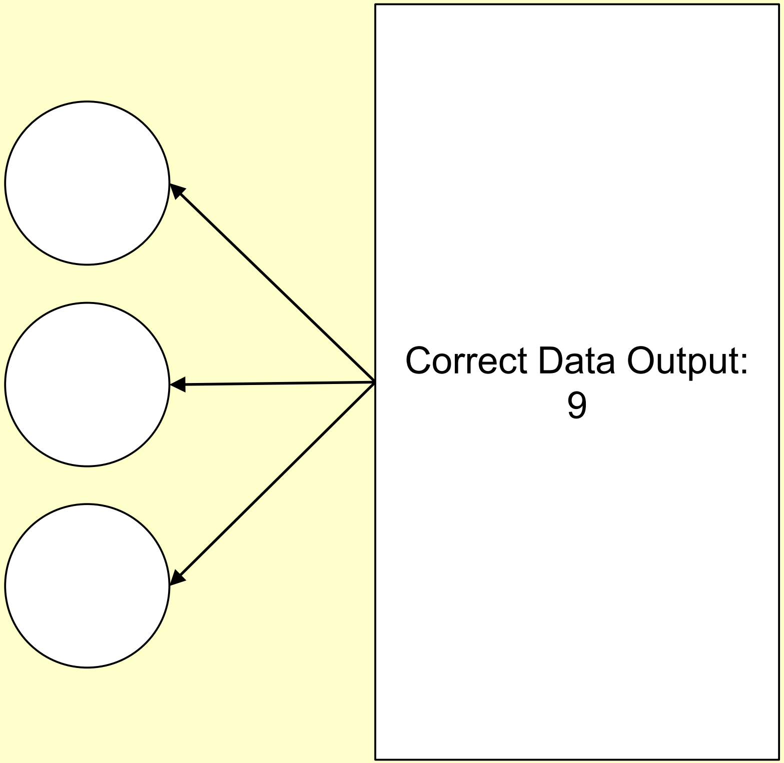
We don't know what the output
should be for this layer
So how do we find this out?

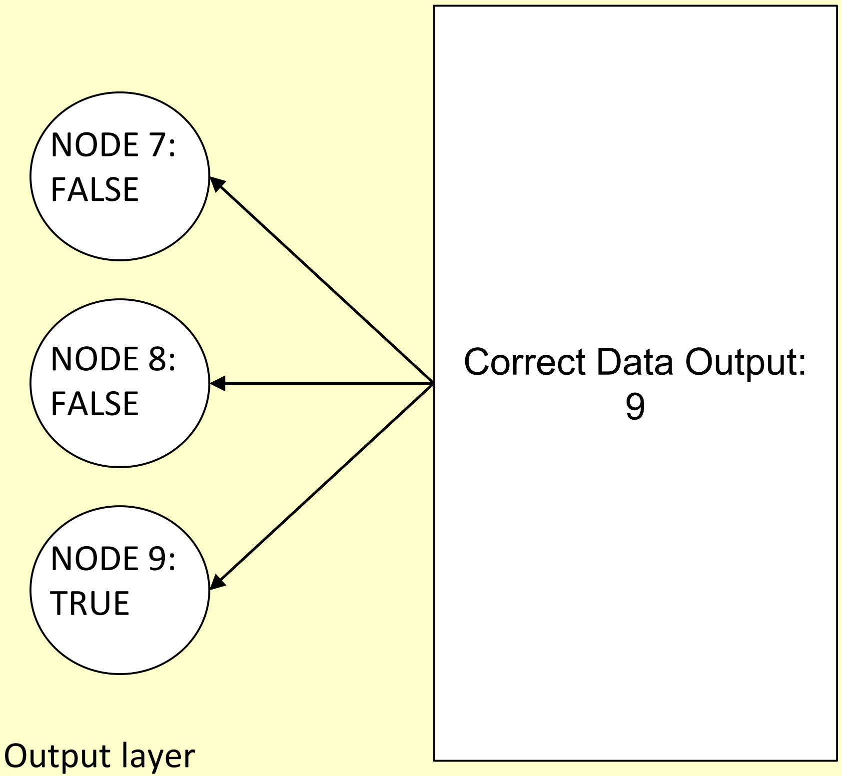
Backpropagation

Each node is the full derivative (the total of All the partials) of the layers above it

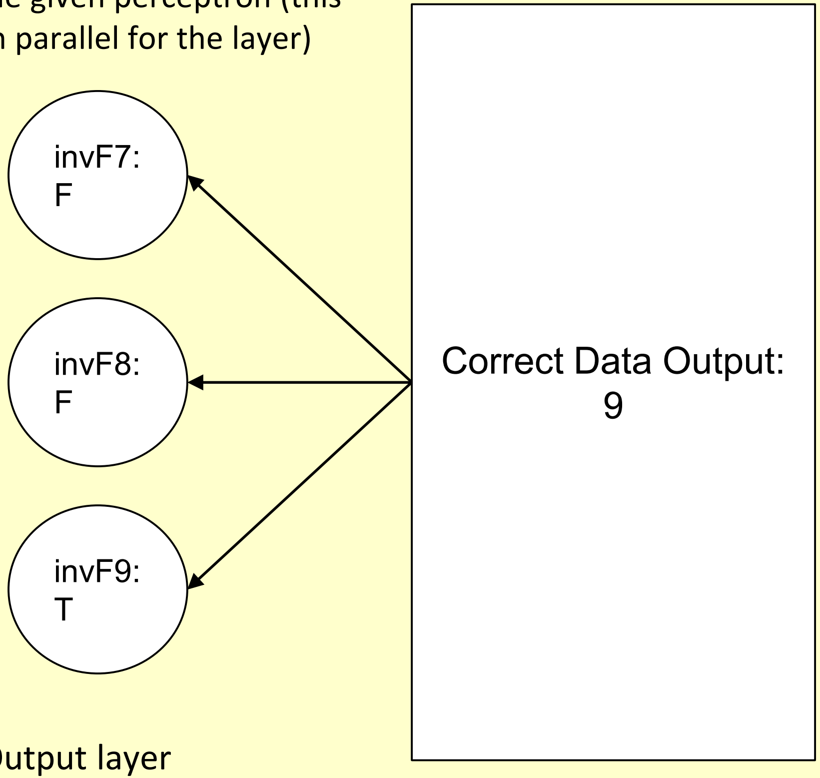
When we feed in data to update, we know the starting and ending points, not the points in between. This means that we don't know what the "correct" results of the hidden layers are. To update Every. Single. Neuron, we must first update the outermost layer, then reverse engineer what the input to that layer should be from the output of that layer to find out what the correct output of the previous layer should be.



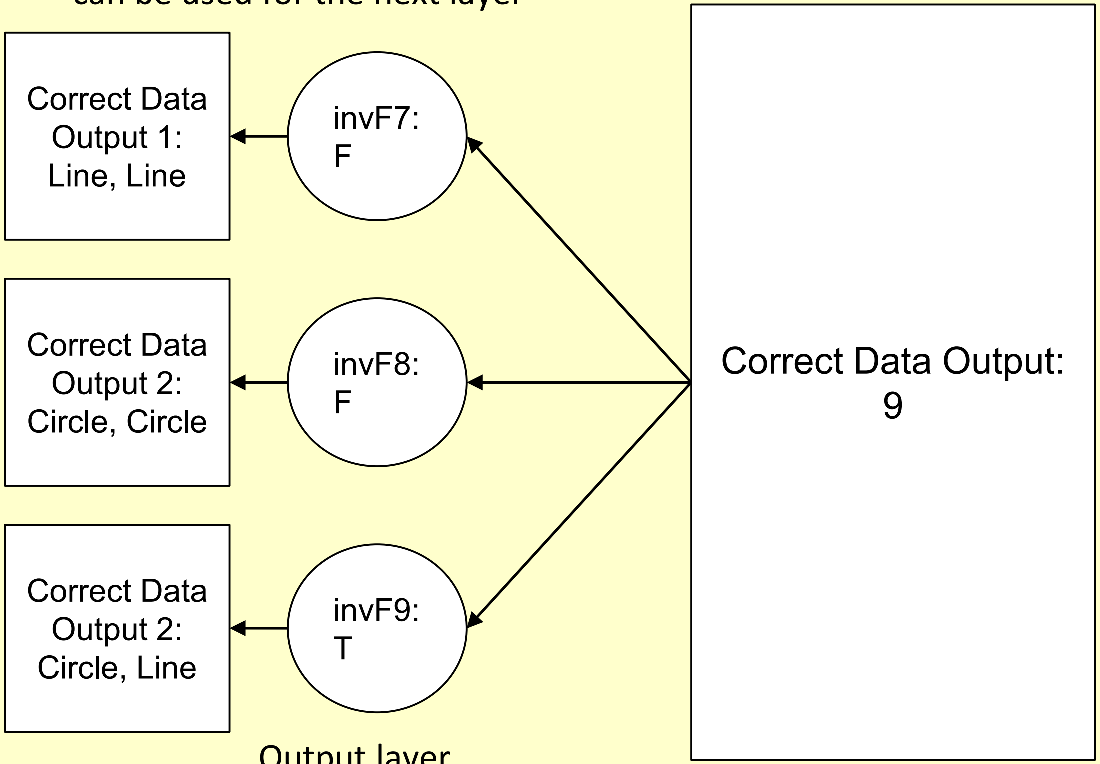

Correct Data Output:
9


NODE 7:
FALSE
NODE 8:
FALSE
NODE 9:
TRUE
Output layer
Correct Data Output:
9

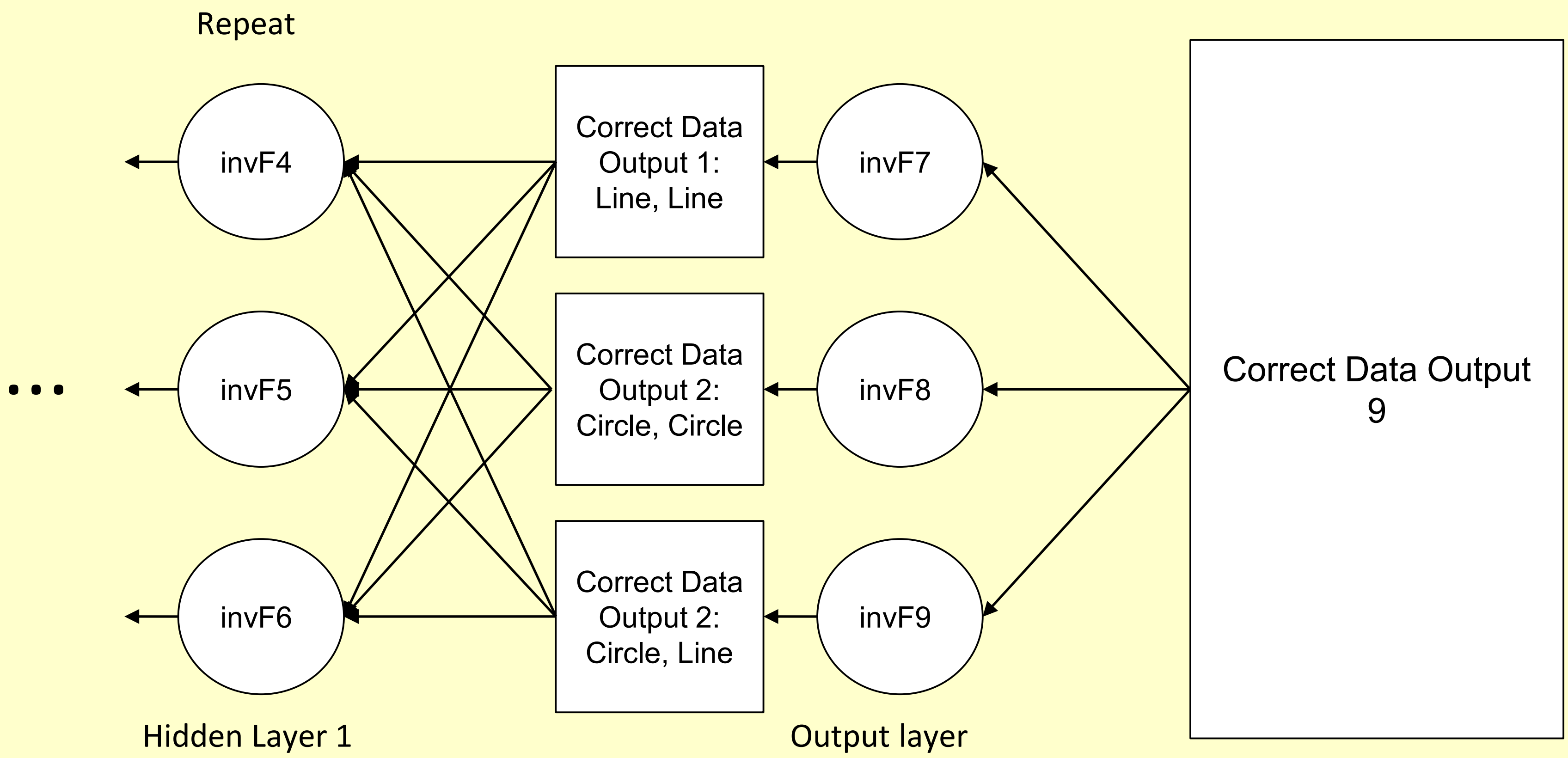
Put the data output through the inverse of the perceptron classification to find the input to the given perceptron (this can be done in parallel for the layer)



Get input to the perceptron, this can be used for the next layer



Output layer


Repeat
invF4
invF5
invF6
Correct Data Output 1: Line, Line
Correct Data Output 2: Circle, Circle
Correct Data Output 2: Circle, Line
invF7
invF8
invF9
Correct Data Output 9
Hidden Layer 1
Output layer

Why do we need hardware support?

- Using neural networks to classify data is **Expensive**
- All of the data inside of an individual neuron can be processed in parallel
  - This becomes a series of matrix multiplications
- All of the data processed in a given layer can be done in parallel
- Multiple data inputs can be processed at once
  - EX: Can classify a handwritten 8, 9, and 1 at the same time
- But Training models is **even more expensive**

Overview

1. **What are Neural Networks?**
    a. Use-Cases
    b. How they classify
    c. How they are trained
2. **How can we speed up Neural Network training?**
3. **What kind of hardware exists that supports neural networks?**
4. **What is the future of hardware support for Neural Networks?**

Neural Networks in Code

- **Matrix operations**
  - Multiplication
  - Addition
  - Binarized Neural Network matrix multiplication (pictured right)

```
#pragma omp parallel for
 for(int i=0;i<n;i+=fBlkI)
  for(int j=0;j<m;j+=fBlkJ)
   for(int k=0;k<_k;k+=fBlkK){
    for(int jj=0;jj<fBlkJ;jj++)
     for(int kk=0;kk<fBlkK;kk++)
      bt[jj][kk]= B[(k + kk)*m +j + jj];
    for(int ii=0;ii<fBlkI;ii+=fBlkII)
     for(int jj=0;jj<fBlkJ;jj+=fBlkJJ)
      for(int kk=0;kk<fBlkK;kk+=fBlkKK){
       ct_00 = C[(i+ii+0)*m+j+jj+0]; ct_01 = C[(i+ii+0)*m+j+jj+1];
       ct_10 = C[(i+ii+1)*m+j+jj+0]; ct_11 = C[(i+ii+1)*m+j+jj+1];
       ct_20 = C[(i+ii+2)*m+j+jj+0]; ct_21 = C[(i+ii+2)*m+j+jj+1];
       ct_30 = C[(i+ii+3)*m+j+jj+0]; ct_31 = C[(i+ii+3)*m+j+jj+1];
       for(int kkk=0;kkk<fBlkKK;kkk++){
        b0 = bt[jj+0][kk+kkk];b1 = bt[jj+1][kk+kkk];
        ct_00 +=popcnt(A[(i+ii+0)*_k +k+kk+kkk]^b0);
        ct_01 +=popcnt(A[(i+ii+0)*_k +k+kk+kkk]^b1);
        ct_10 +=popcnt(A[(i+ii+1)*_k +k+kk+kkk]^b0);
        ct_11 +=popcnt(A[(i+ii+1)*_k +k+kk+kkk]^b1);
        ct_20 +=popcnt(A[(i+ii+2)*_k +k+kk+kkk]^b0);
        ct_21 +=popcnt(A[(i+ii+2)*_k +k+kk+kkk]^b1);
        ct_30 +=popcnt(A[(i+ii+3)*_k +k+kk+kkk]^b0);
        ct_31 +=popcnt(A[(i+ii+3)*_k +k+kk+kkk]^b1);
       }
       C[(i+ii+0)*m+j+jj+0] = ct_00;C[(i+ii+0)*m+j+jj+1] = ct_01;
       C[(i+ii+1)*m+j+jj+0] = ct_10;C[(i+ii+1)*m+j+jj+1] = ct_11;
       C[(i+ii+2)*m+j+jj+0] = ct_20;C[(i+ii+2)*m+j+jj+1] = ct_21;
       C[(i+ii+3)*m+j+jj+0] = ct_30;C[(i+ii+3)*m+j+jj+1] = ct_31;
      }
   }
```

Fig. 5. CPU implementation of binarized matrix multiply (C = A x B).

Measurement Metrics

- Training time can be used as a metric to evaluate the performance of different architectures
- Inference takes much less time (<< 1 second)
  - Measured in predictions per second

Type of data	Problem area	Size of benchmark's training set	DNN architecture	Hardware	Training time
text [1]	word prediction (word2vec)	100 billion words (Wikipedia)	2-layer skip gram	1 NVIDIA Titan X GPU	6.2 hours
audio [2]	speech recognition	2000 hours (Fisher Corpus)	11-layer RNN	1 NVIDIA K1200 GPU	3.5 days
images [3]	image classification	1 million images (ImageNet)	22-layer CNN	1 NVIDIA K20 GPU	3 weeks
video [4]	activity recognition	1 million videos (Sports-1M)	8-layer CNN	10 NVIDIA GPUs	1 month

Figure 7.5 Training set sizes and training time for several DNNs [Iandola 2016]

Overview

1. **What are Neural Networks?**
2. **How can we speed up Neural Network training?**
   a. Software Speedups
   b. Hardware Speedups
3. **What kind of hardware exists that supports neural networks?**
4. **What is the future of hardware support for Neural Networks?**

Non-hardware Specific Speedups

- **Pre-processing data**
- **Reduce number of layers**
- **Parallelization**
  - Clusters
  - Threads
- **More efficient code**
  - Ex: Make better use of caches

Hardware Constraints

- Memory Bandwidth
- Numeric Operations
- Power
- Parallelization

Floating Point Representation

Slide courtesy of Professor Lauer's CS 2011 lecture notes

- **Numerical Form: $(-1)^s \times M \times 2^E$**
  - Sign bit *s* determines whether number is negative or positive
  - Significand *M* normally a fractional value in range [1.0, 2.0).
  - Exponent E weights value by power of two
- **Encoding:**

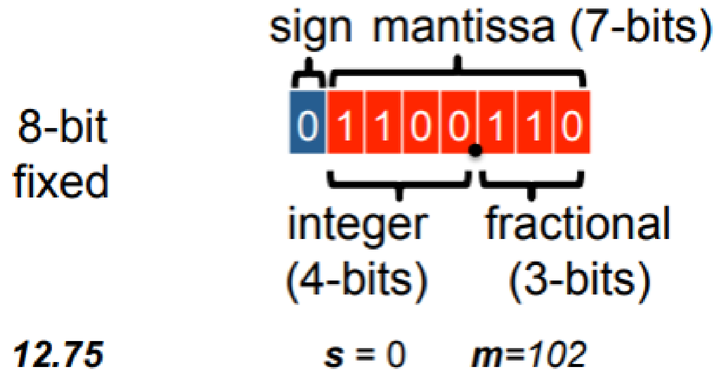
  - MSB *s* is sign bit s
  - *exp* field encodes *E* (but is not equal to E)
  - *frac* field encodes M (but is not equal to M)

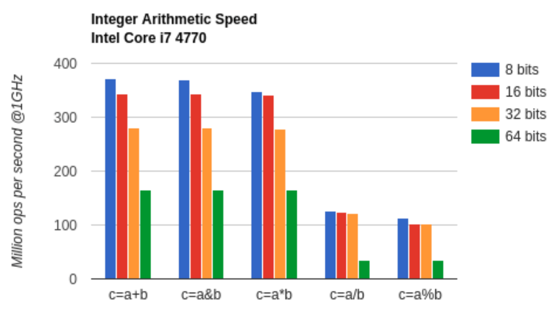
Floating Point Numbers

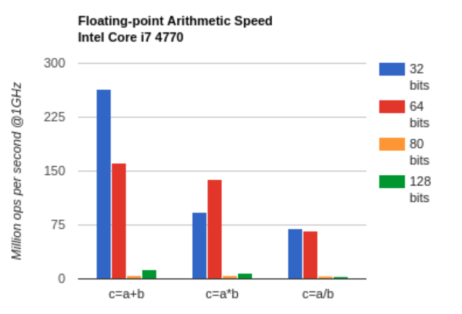
- **Higher Precision**
  - More computing power = More power consumption
  - Slower Computations
- **Lower Precision**
  - Less computer power = Less power consumption
  - Faster Computations
  - Ex: TPU uses half-precision 16-bit floating point numbers

Dynamic Fixed Point Numbers

- Alternate way of representing fractional numbers
- "Dynamic" because the number of fractional bits can vary based on data type and layer





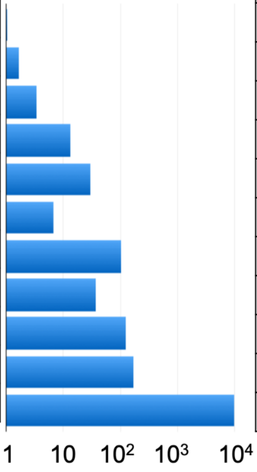


Effects of Reduced Precision

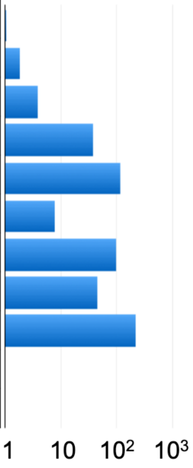
Cost of Operations

Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A

Relative Energy Cost



Relative Area Cost



[Horowitz, "Computing's Energy Problem (and what we can do about it)", ISSCC 2014]

3

Summary of Reduce Precision

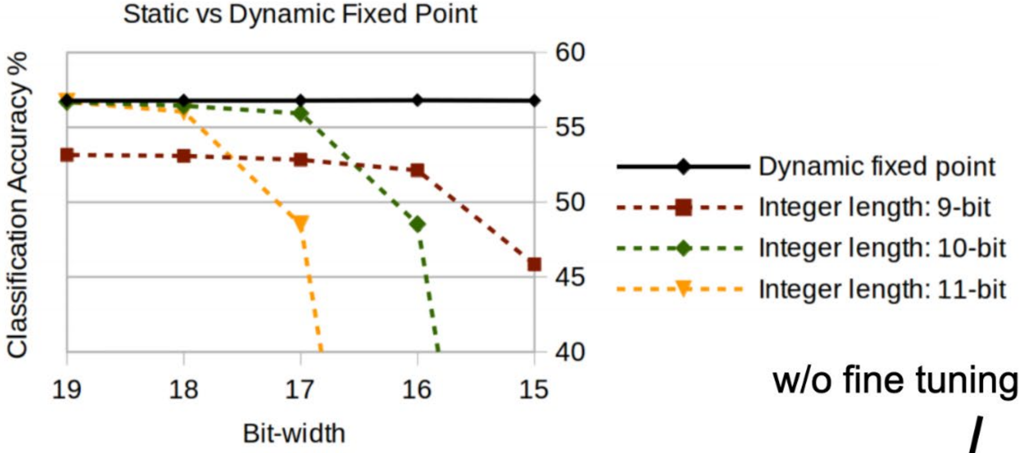
Category	Method	Weights (# of bits)	Activations (# of bits)	Accuracy Loss vs. 32-bit float (%)
Dynamic Fixed Point	w/o fine-tuning	8	10	0.4
	w/ fine-tuning	8	8	0.6
Reduce weight	Ternary weights Networks (TWN)	2*	32	3.7
	Trained Ternary Quantization (TTQ)	2*	32	0.6
	Binary Connect (BC)	1	32	19.2
	Binary Weight Net (BWN)	1*	32	0.8
Reduce weight and activation	Binarized Neural Net (BNN)	1	1	29.8
	XNOR-Net	1*	1	11
Non-Linear	LogNet	5(conv), 4(fc)	4	3.2
	Weight Sharing	8(conv), 4(fc)	16	0

* first and last layers are 32-bit float

Full list @ [Sze et al., arXiv, 2017] 25

Impact on Accuracy

Top-1 accuracy on of CaffeNet on ImageNet



w/o fine tuning

	Layer outputs	CONV parameters	FC parameters	32-bit floating point baseline	Fixed point accuracy
LeNet (Exp 1)	4-bit	4-bit	4-bit	99.1%	99.0% (98.7%)
LeNet (Exp 2)	4-bit	2-bit	2-bit	99.1%	98.8% (98.0%)
Full CIFAR-10	8-bit	8-bit	8-bit	81.7%	81.4% (80.6%)
SqueezeNet top-1	8-bit	8-bit	8-bit	57.7%	57.1% (55.2%)
CaffeNet top-1	8-bit	8-bit	8-bit	56.9%	56.0% (55.8%)
GoogLeNet top-1	8-bit	8-bit	8-bit	68.9%	66.6% (66.1%)

[Gysel et al., Ristretto, ICLR 2016]

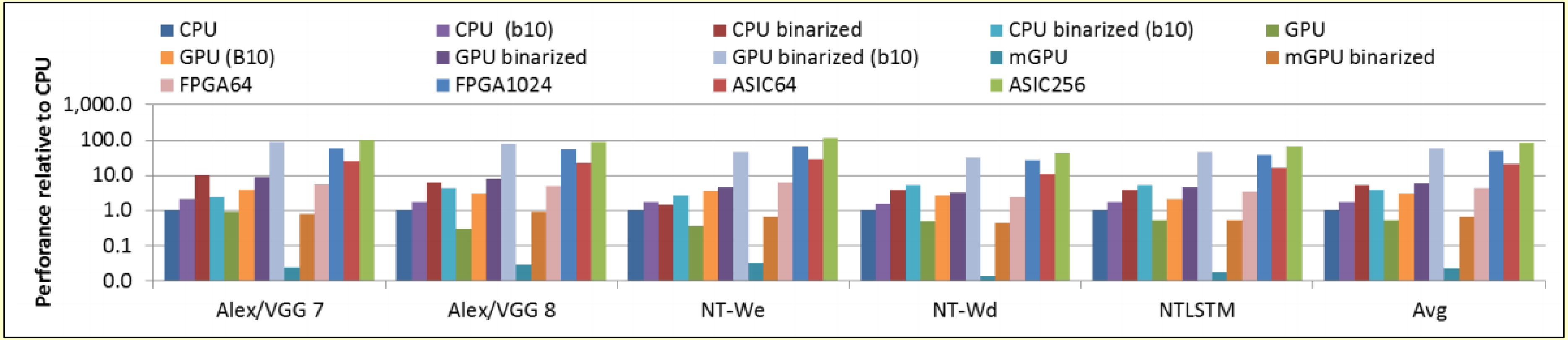
Hardware Specific Speedups

- **Predicated (Speculative) execution**
- **Branch delay slots**
- **Tailored to streaming workloads**
  - Specialized data and memory paths
- **Full support for sparse data structures**
- **Application specific hardware support**
  - Ex: Tensorflow and TPU

Overview

1. **What are Neural Networks?**
2. **How can we speed up Neural Network training?**
3. **What kind of hardware exists that supports neural networks?**
    a. General: CPU, GPU
    b. Specialized: FPGA, ASICs
        i. Tensor Processing Units (TPU)
        ii. Visual Processing Unit (VPU)
4. **What is the future of hardware support for Neural Networks?**

Machine Learning Hardware (Overview)



Neural Networks on Hardware

- **Most NN are still implemented on software on sequential machines**
- **More can be gained from NN implemented on hardware**
  - Exploits parallelism inherent in neural networks without undue costs

Neural Network Hardware (CPUs)

- **Memory locality**
  - Make use of CPU caches
- **Loop Unrolling**
  - Remove hazards
- **Multiple Accumulators**
  - Allow for more pipelining
- **Batched Classification**
  - Group together multiple inputs in the classification phase of a neural network

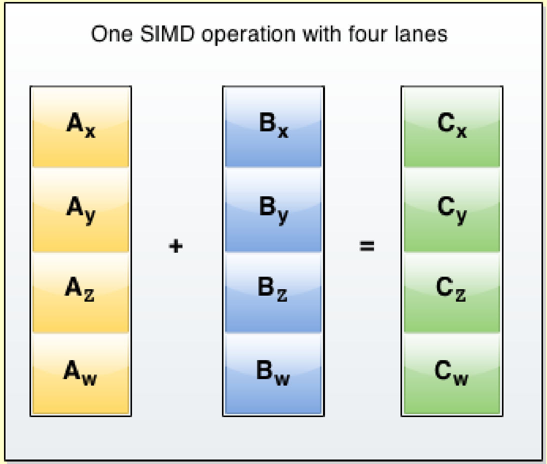
```
c += a[i]*b[i] + a[i+1]*b[i+1] + a[i+2]*b[i+2] + a[i+3]*b[i+3]
```

↓

```
c0 += a[i]*b[i];
c1 += a[i+1]*b[i+1];
c2 += a[i+2]*b[i+2];
c3 += a[i+3]*b[i+3];
c = c0 + c1 + c2 + c3;
```

Neural Network Hardware (CPU)

- **SIMD**
  - Perform multiple operations in parallel on contiguous data
  - High Memory Locality required
  - Memory alignment required
- **Without batching, CPU is not a good hardware platform for certain neural networks**



Neural Network Hardware (GPUs)

- **Massive Parallelism**
  - Use Neural Network independence
- **Large amount of ALUs**
  - For fast, simple instructions
- **Batching**
  - Linear improvements from batching processes

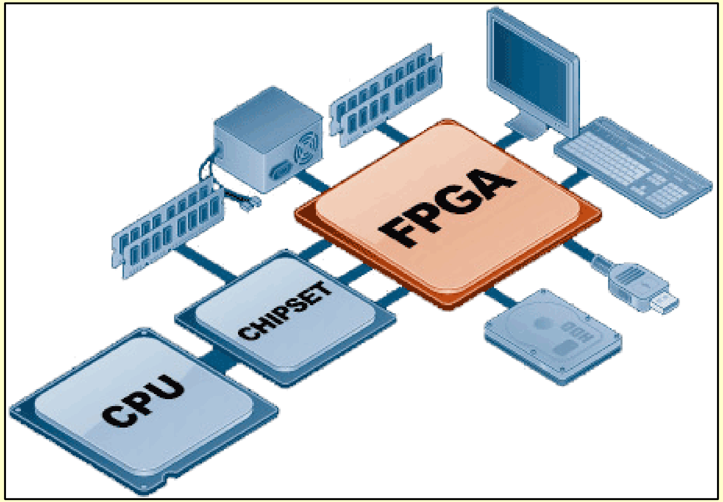
Table 3: GPU Implementation

	Batch size	Processing 1s of speech
CPU	1	1360 ms
GPU	1	490 ms
	2	250 ms
	4	125 ms
	8	66 ms
	128	20 ms

Neural Network Hardware (CPU vs. GPU)

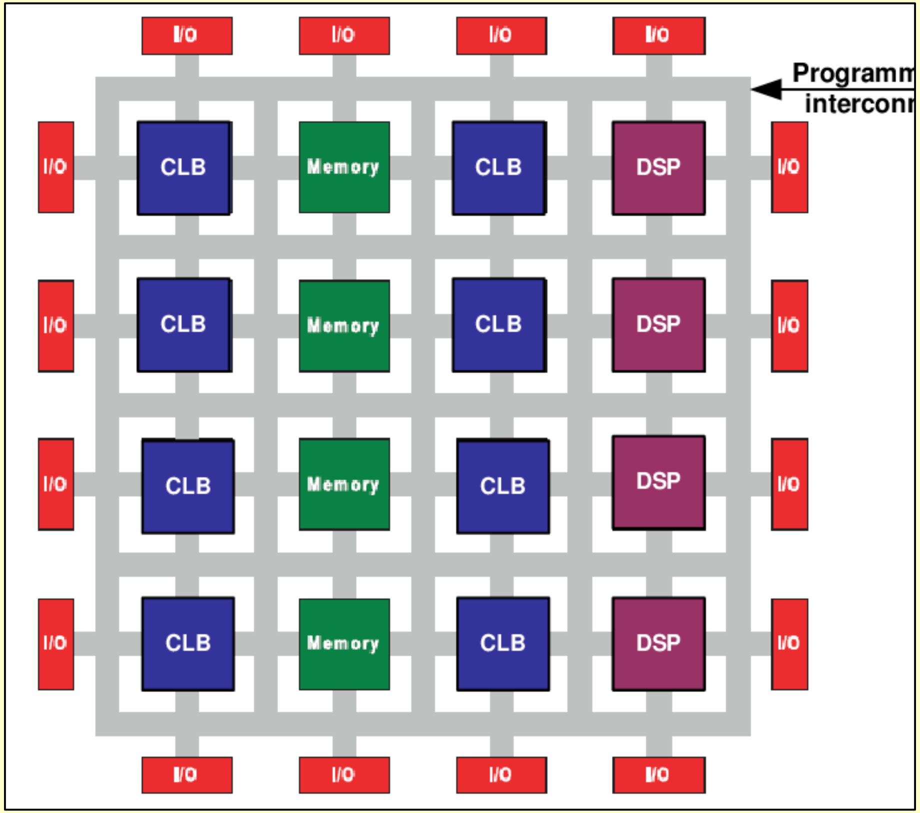
- **Depends on the type of Neural Network**
  - Batching possible: GPU
    - Speech-recognition Neural Network
  - Batching not possible: CPU
    - More performance for power
    - Limited parallelization possible
- **Both suffer from under-utilization**
  - Limited data reuse due to varying weight-matrices

Field Programmable Gate Arrays (FPGAs)



- **Devices that permit implementation of digital systems**
  - array of logic components configured by bitstream
- **Current research attempting to make digital processing techniques resemble the biological structure of the brain**

FPGA Architecture



Neural Networks (FPGA)

- **Until 1990's FPGAs were not considered**
  - Never large enough or fast enough for NN applications
- **Current FPGAs most realistic alternative for NN**
- **FPGAs cannot match ASIC processors in performance**
  - However, have a better cost:performance ratios for applications

Neural Networks (FPGA)

- **Have capacity for reconfiguration, so can have range of applications - meaning many types of NN**
- **Can implement NN models:**
  - Multi-layer perceptrons
  - Kohonen's self-organizing feature map
  - Associative memory networks
- **NN exhibit several types of parallelism**
  - Except for very small networks, fully parallel implementation in hardware is not feasible, so virtual parallelism necessary

Neural Networks (FPGA)

- **Parallelism that works on FPGAs**
  - Training parallelism
    - Different training sequences run in parallel (on SIMD or MIMD processors)
    - Level of parallelism usually medium, fully mapped onto large FPGAs
  - Layer Parallelism
    - In multilayer network, multiple layers processed in parallel
    - Level of parallelism usually low, so limited value. Can be fully mapped onto FPGAs

Neural Networks (FPGA)

- **Parallelism that works on FPGAs (cont.)**
  - *Node Parallelism*
    - Corresponds to individual neurons
    - Most important level of parallelism, because if fully exploited, parallelism of above higher levels also fully exploited
    - Since number of neurons can be in millions, may not be possible to fully exploit
    - Matches FPGAs very well, typical FPGA has many “cells” that operate in parallel, onto which neurons can be mapped

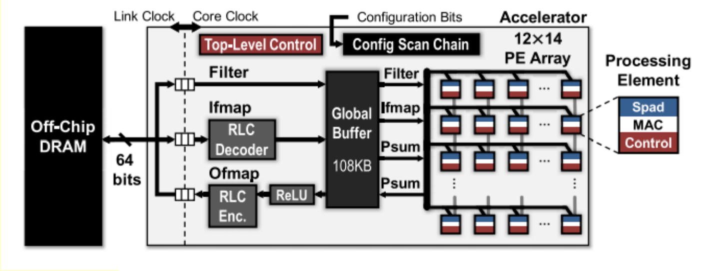
Application Specific Integrated Circuit (ASIC)

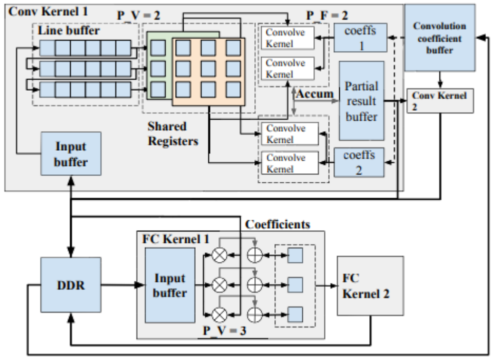
- **Integrated circuit (IC) customized for a particular use**
  - rather than intended for general-purpose use
- **Over 100 million logic gates**
- **Modern ASICs often include entire microprocessors, memory blocks (ROM, RAM, EEPROM), flash memory, and and other large building blocks**
  - These types of ASICs are termed SoC (system-on-chip)

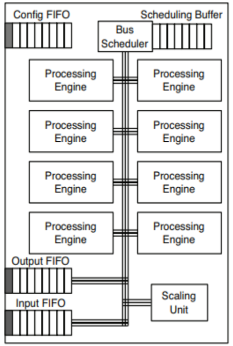
Neural Networks (ASICs)

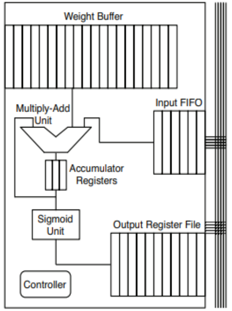
- **Numerous Neural Network ASIC implementations**
  - TPU
  - Dell-Graphcore IPU
    - “improving performance and efficiency by between 10x to 100x”
  - Numerous other proposed ASICs
- **Optimized to fit requirements**
  - Power efficiency
  - Cost
  - High performance

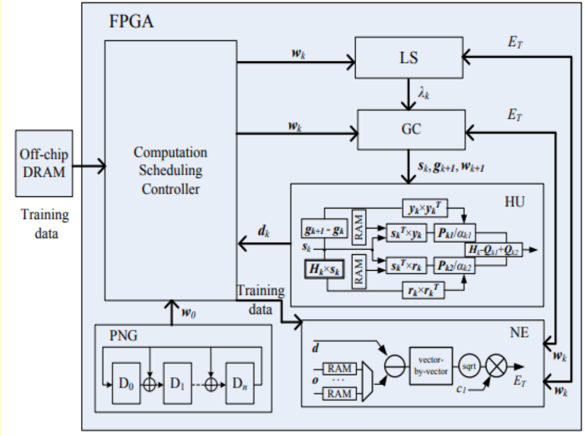
Neural Networks Research (FPGA & ASICs)

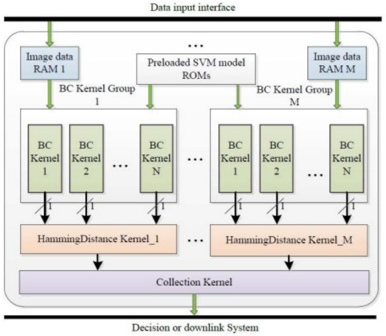


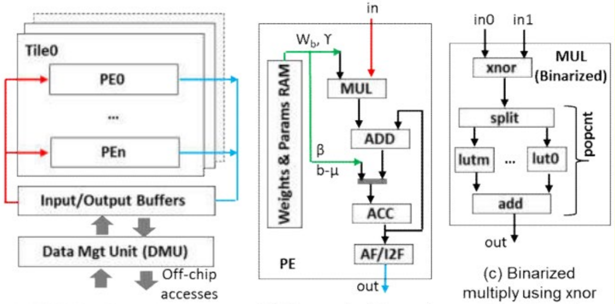






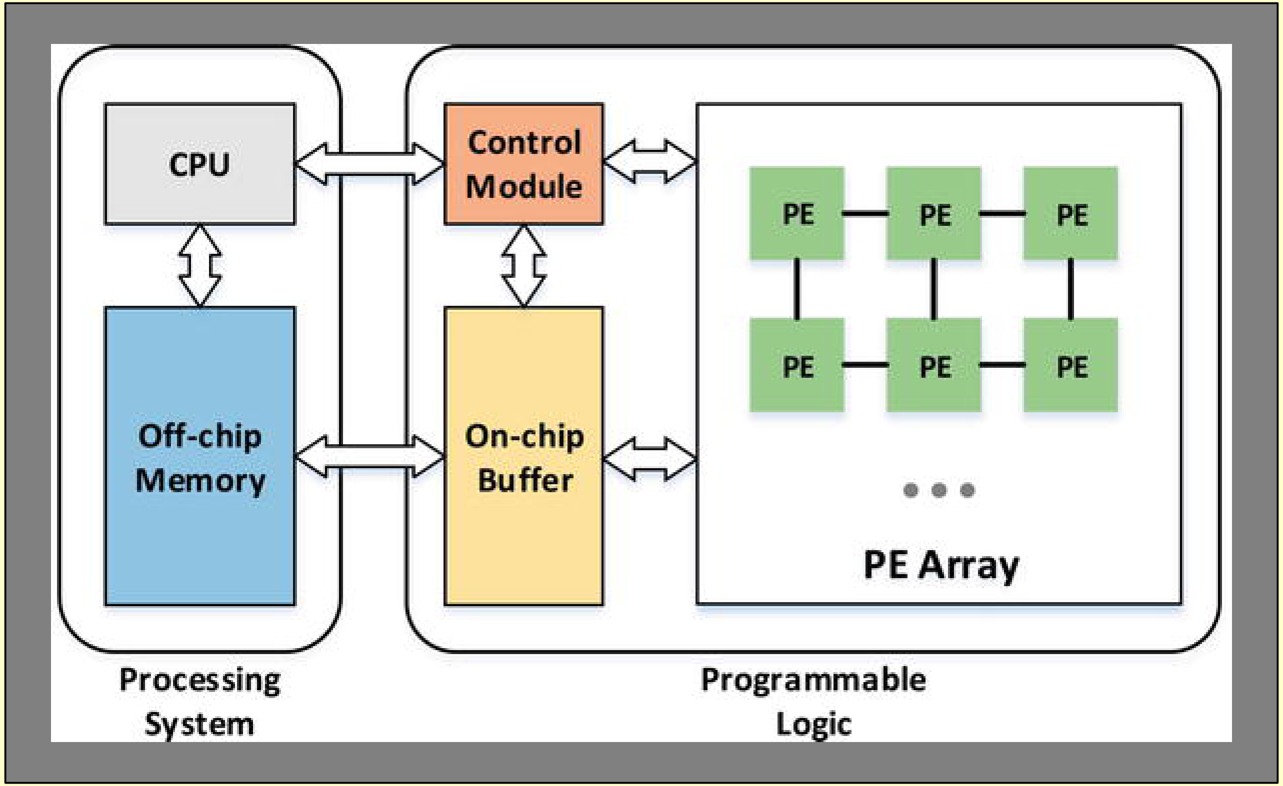






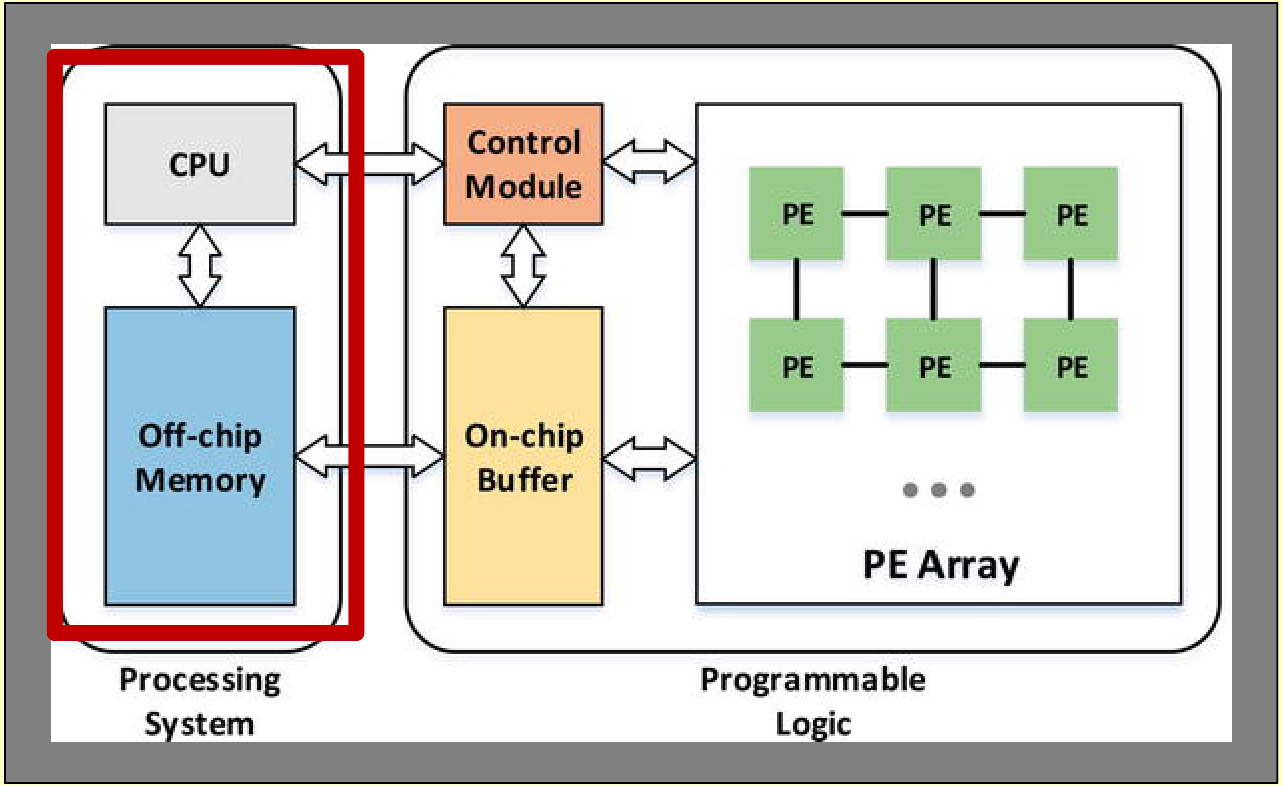
(a) BNN Hardware Accelerator (b) Processing Element (c) Binarized multiply using xnor and popcount

CNN Accelerator



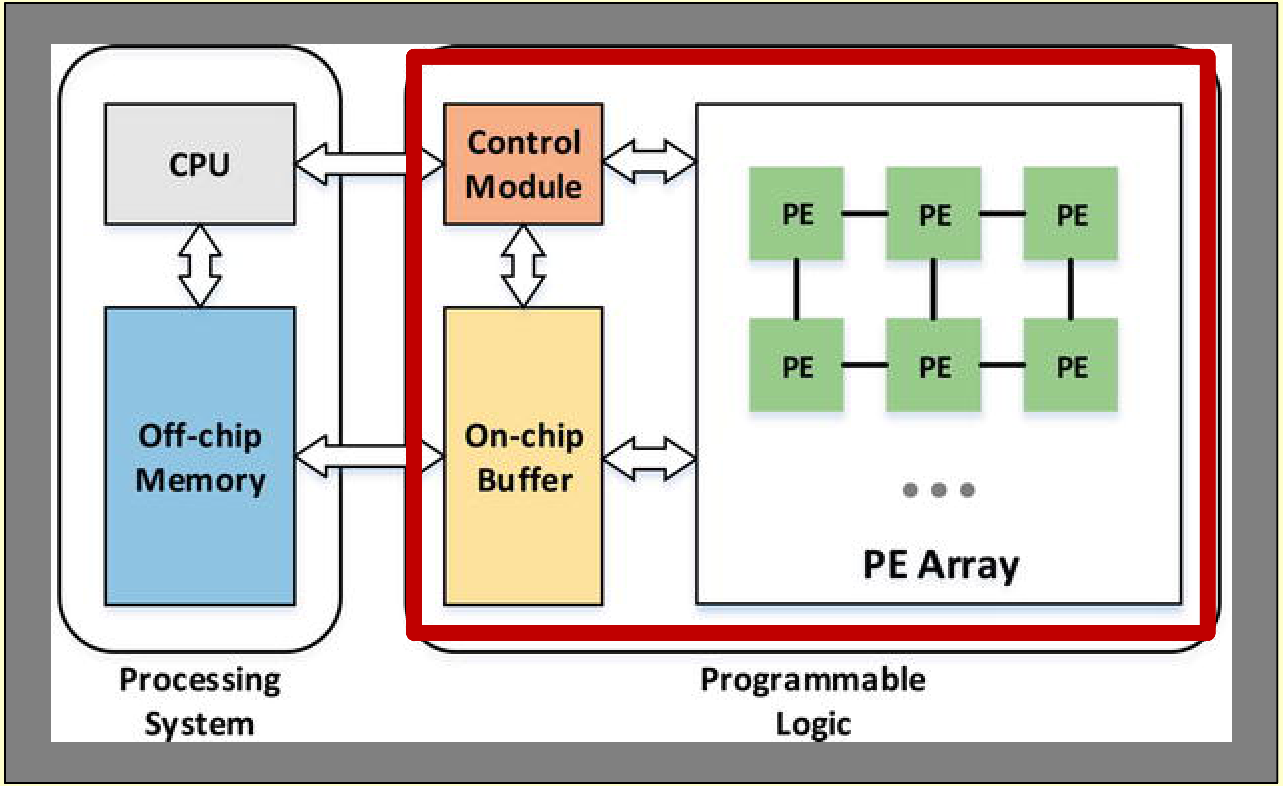
CPU and FPGA Convolutional Neural Network accelerator

CNN Accelerator



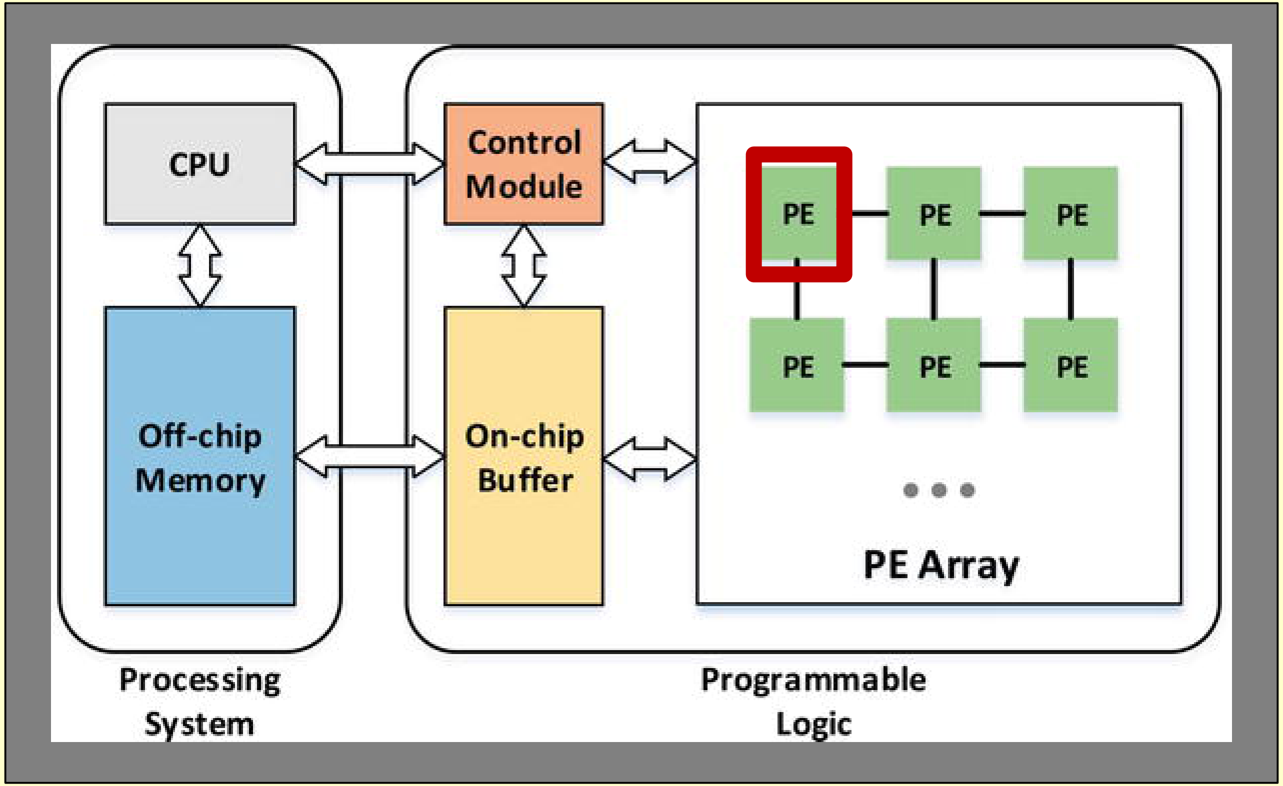
CPU and FPGA Convolutional Neural Network accelerator

CNN Accelerator



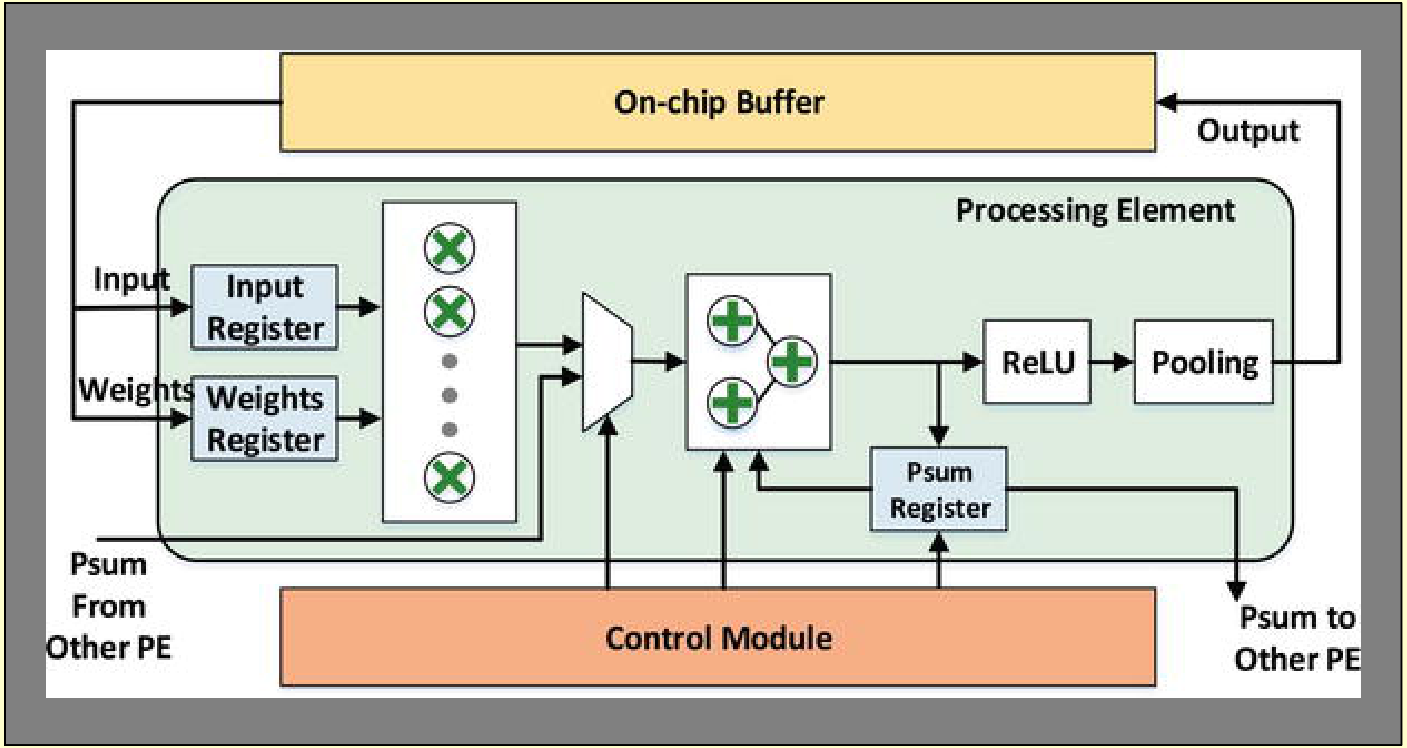
CPU and FPGA Convolutional Neural Network accelerator

CNN Accelerator



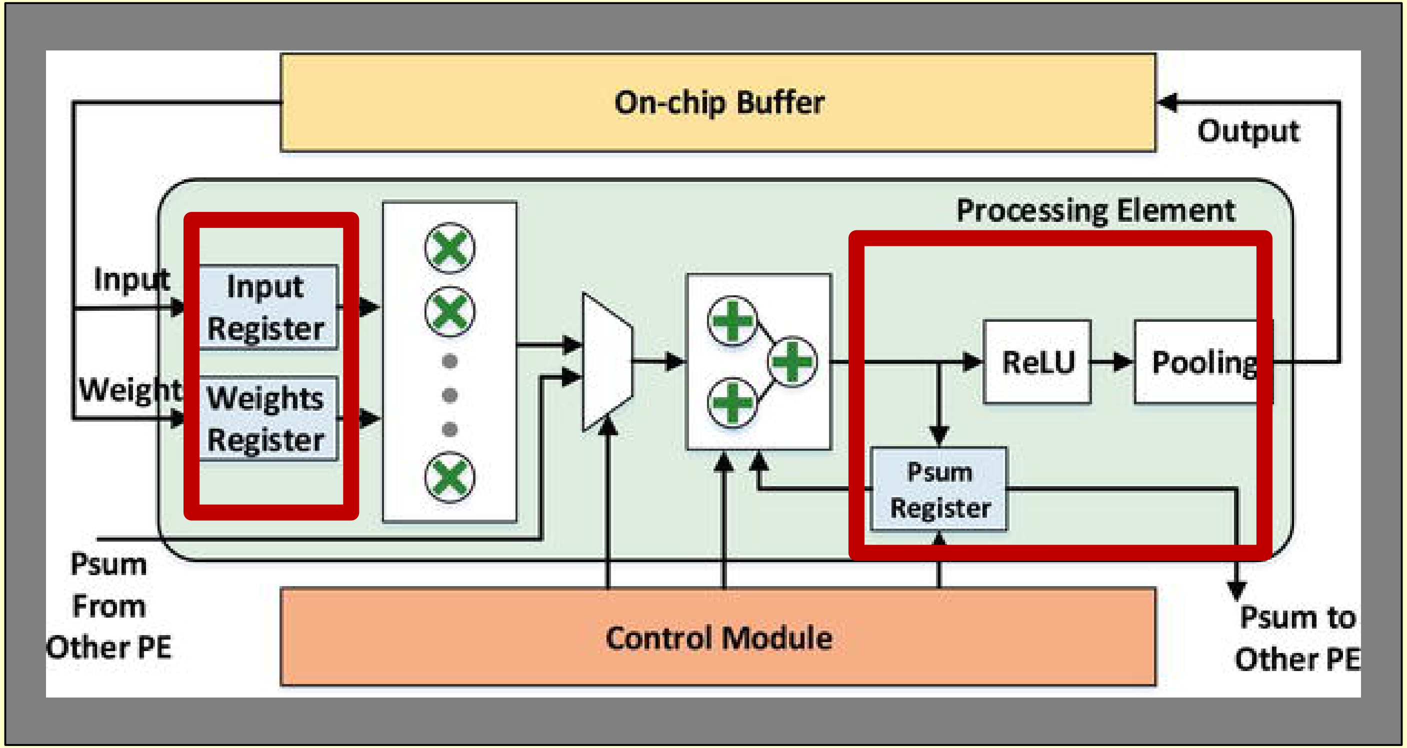
CPU and FPGA Convolutional Neural Network accelerator

CNN Accelerator



Processing Element of the CPU and FPGA accelerator

CNN Accelerator

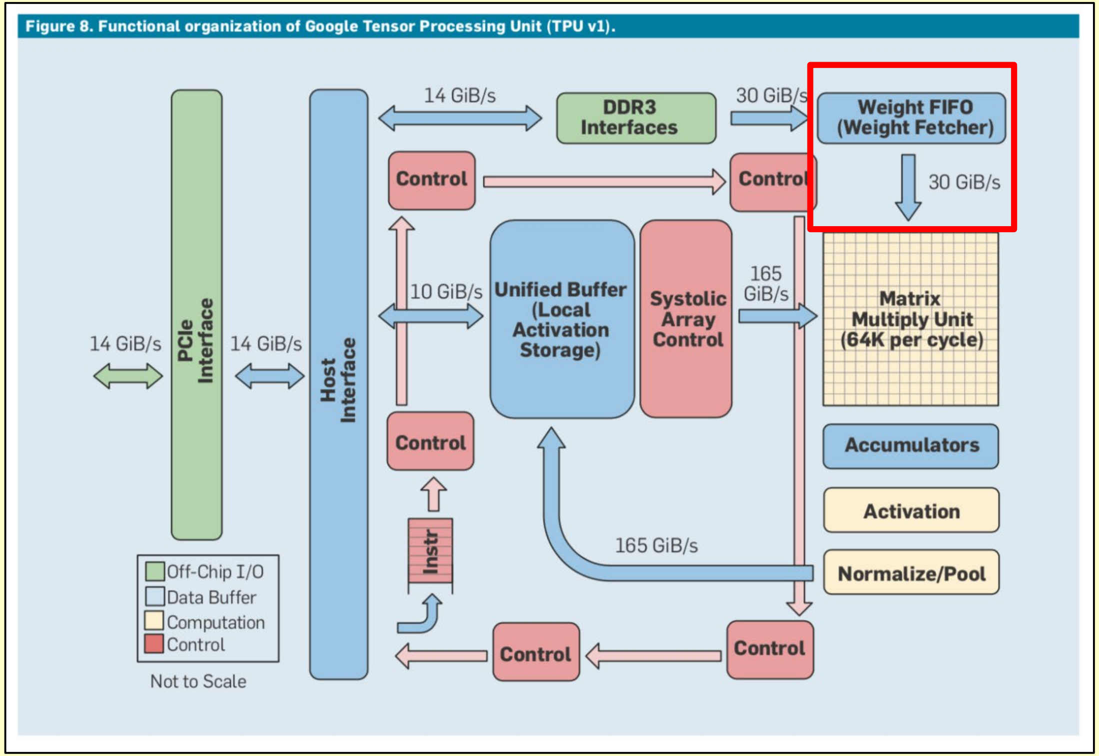


Processing Element of the CPU and FPGA accelerator

Tensor Processing Units (TPUs)

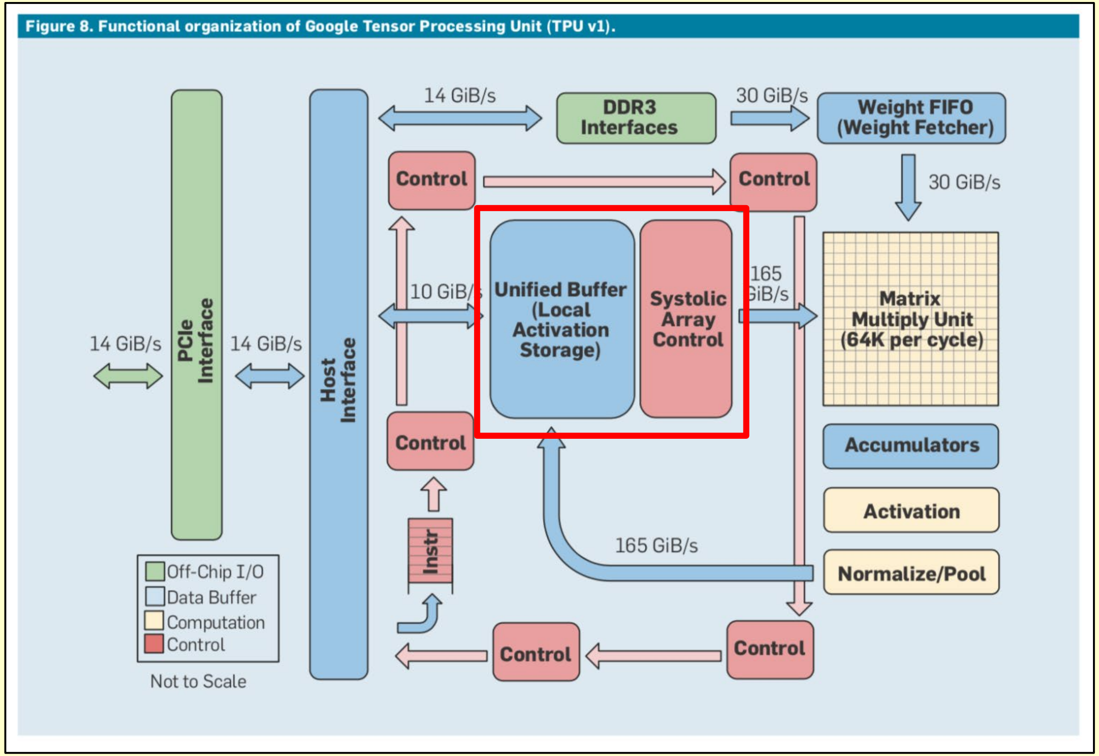
- **Developed first by Google in 2013**
- **Made to keep up with the growing demand for responses from Neural Nets as their popularity increases**
  - high throughput of low-precision multiplications
- **Used mainly in inference (or prediction) of the neural network**
  - GPUs still mainly used for training
  - The technology is still proprietary (only used in Google)

Tensor Processing Unit Organization



Gathers the weights to multiply the input with

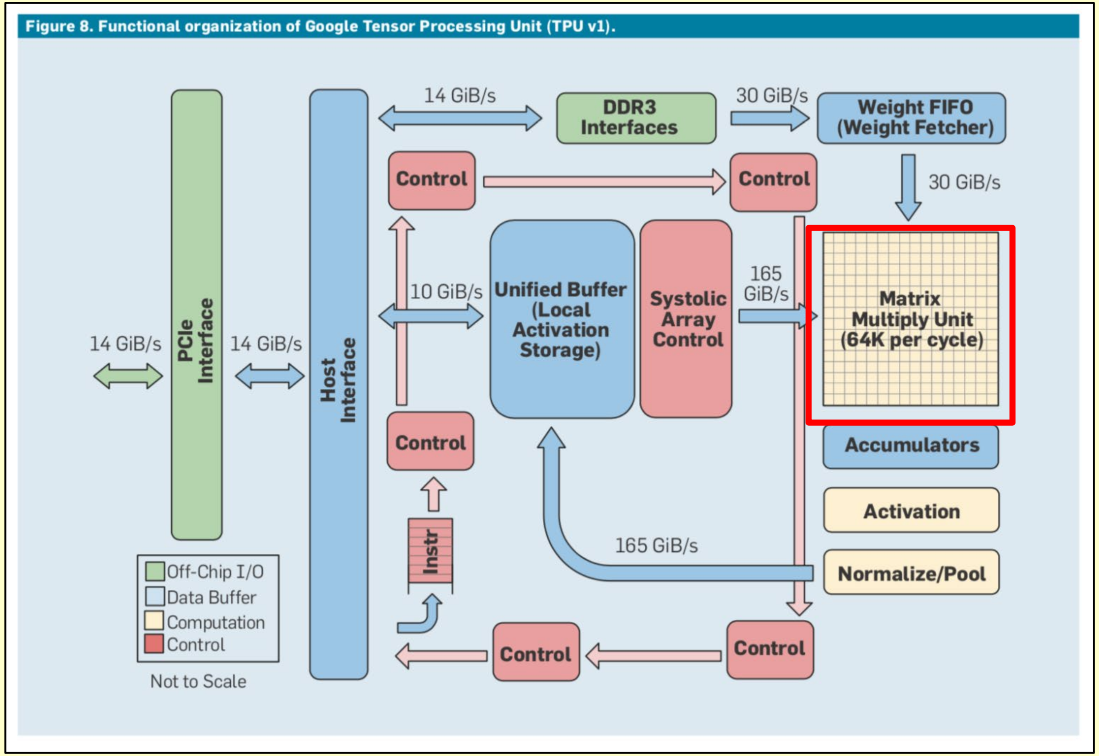
Tensor Processing Unit Organization



Prepares the inputs to be multiplied

The unified buffer stores the activation functions for the neurons

Tensor Processing Unit Organization



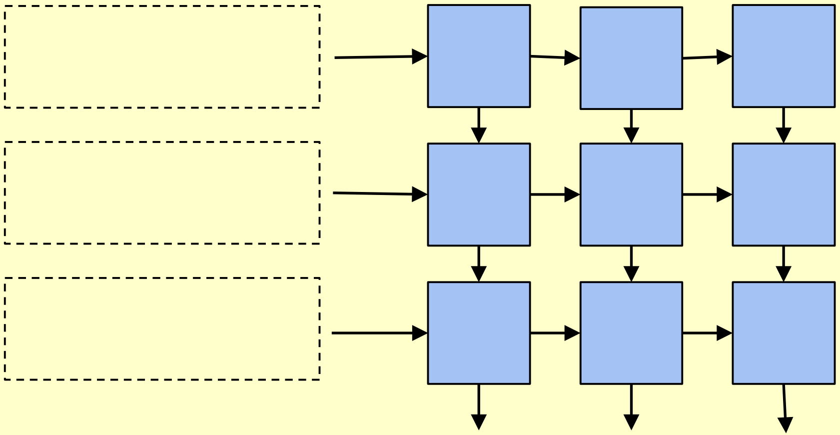
- **Heart of the TPU**
  - Can perform 256 x 256 8-bit operations in one cycle
  - Stores result in Accumulator (situated below)

Matrix Multiply Unit (Systolic Array)

- A 256 x 256 matrix is loaded into the MMU
- A *B* x 256 matrix is multiplied to this matrix vector by vector
- This will take *B* pipelined cycles to complete (with some latency)

Example: A matrix of 10 x 256 is multiplied with a 256 x 256 matrix will take 10 cycles.

Flow of Data Through the TPU

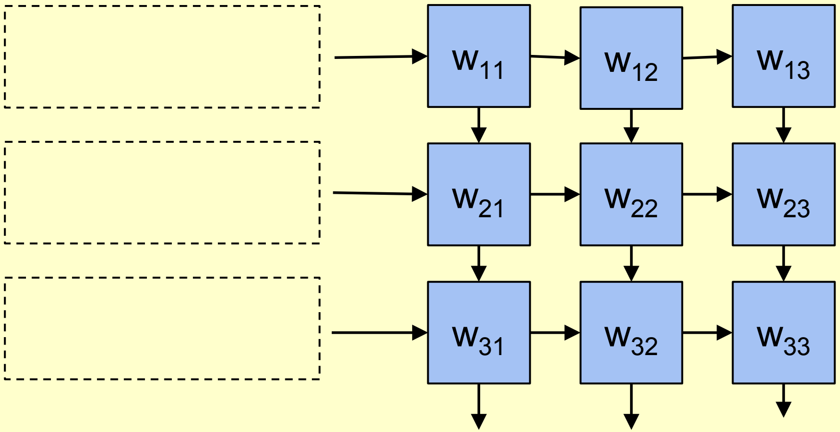


Imagine we want to multiply matrix A with a matrix W. The result will be matrix Y.

i.e.

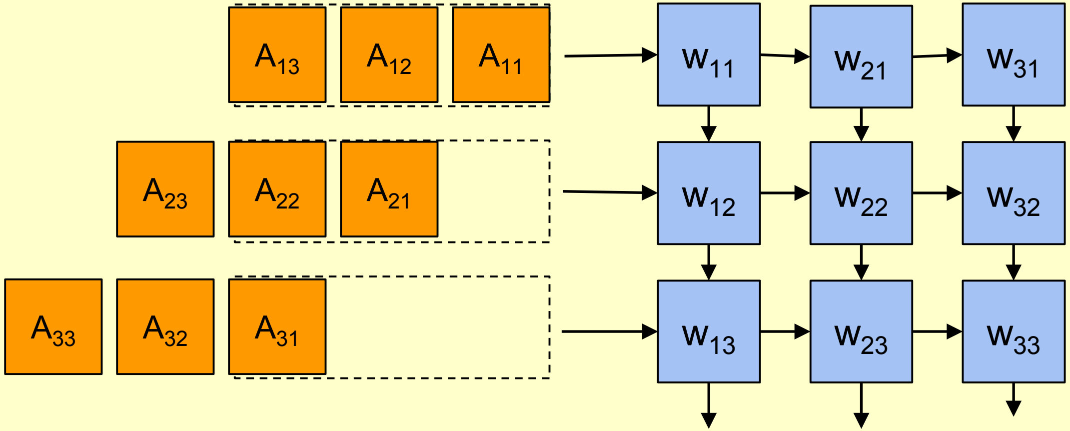
A x W = Y

Flow of Data Through the TPU



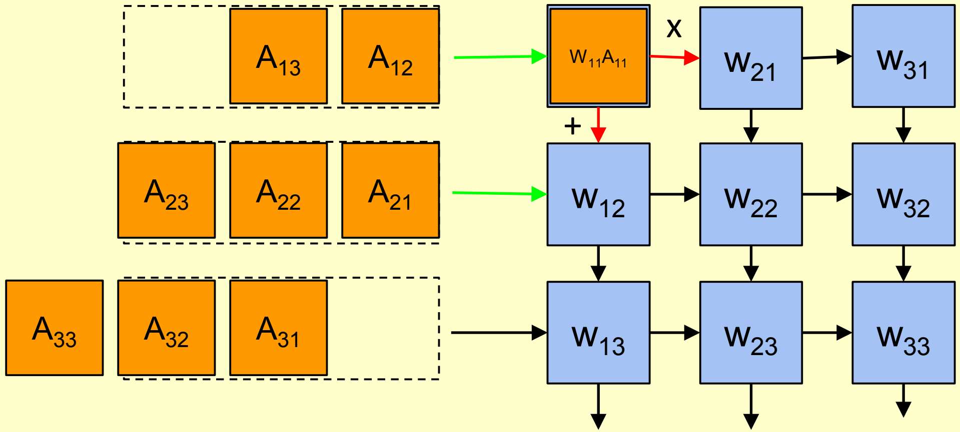
Step 1: Load weights into systolic array

Flow of Data Through the TPU



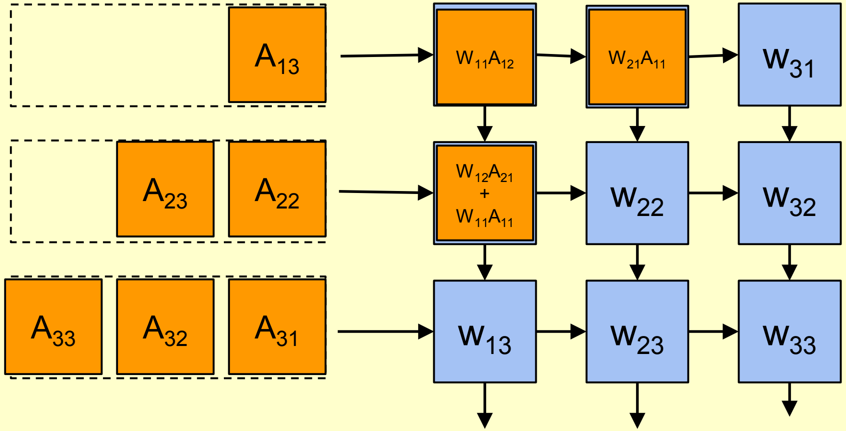
Step 2: Load inputs into systolic array
array control

Flow of Data Through the TPU



Step 3: Propagate through systolic array
T = 1

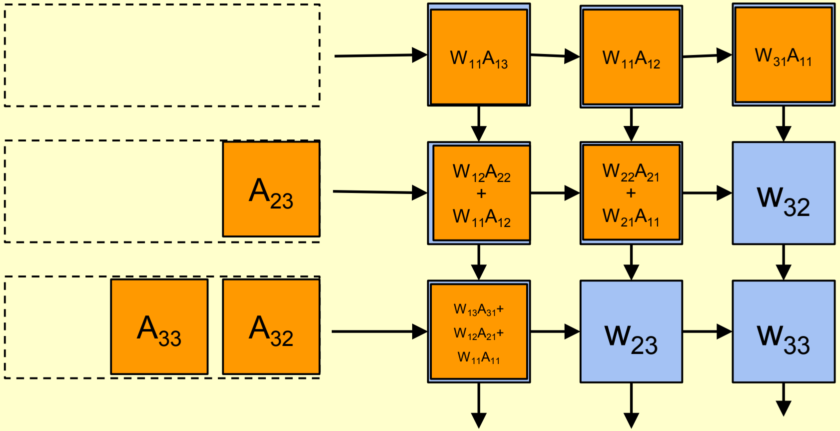
Flow of Data Through the TPU



Step 3: Propagate through systolic array

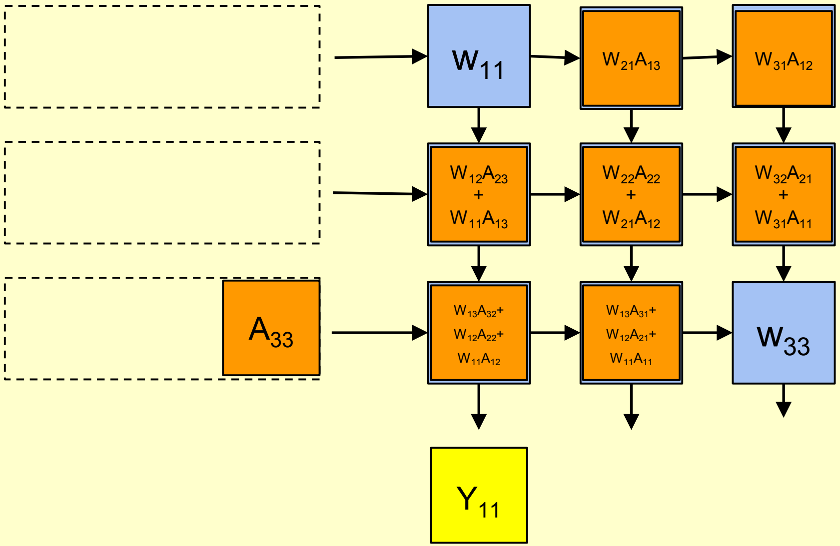
T = 2

Flow of Data Through the TPU



Step 3: Propagate through systolic array
T = 3

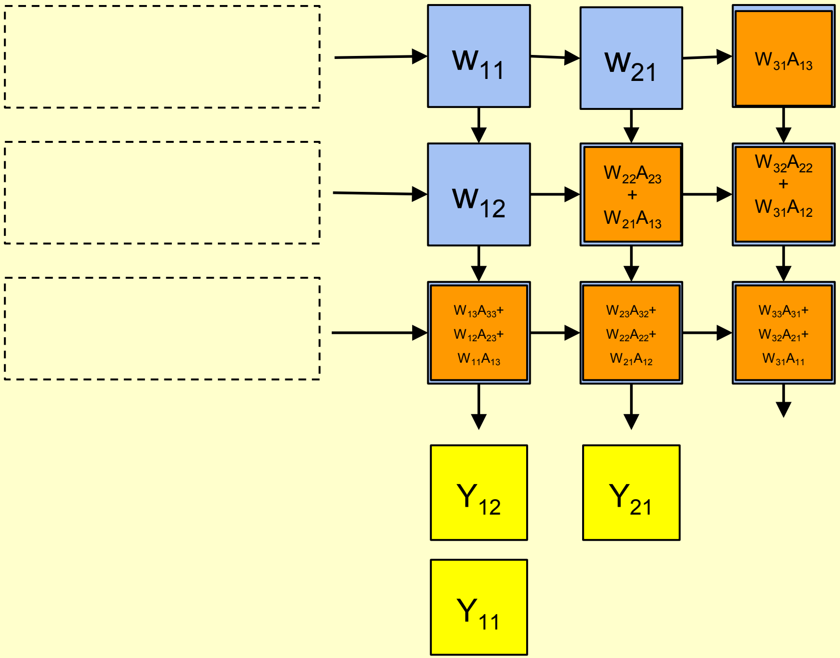
Flow of Data Through the TPU



Step 3: Propagate through systolic array

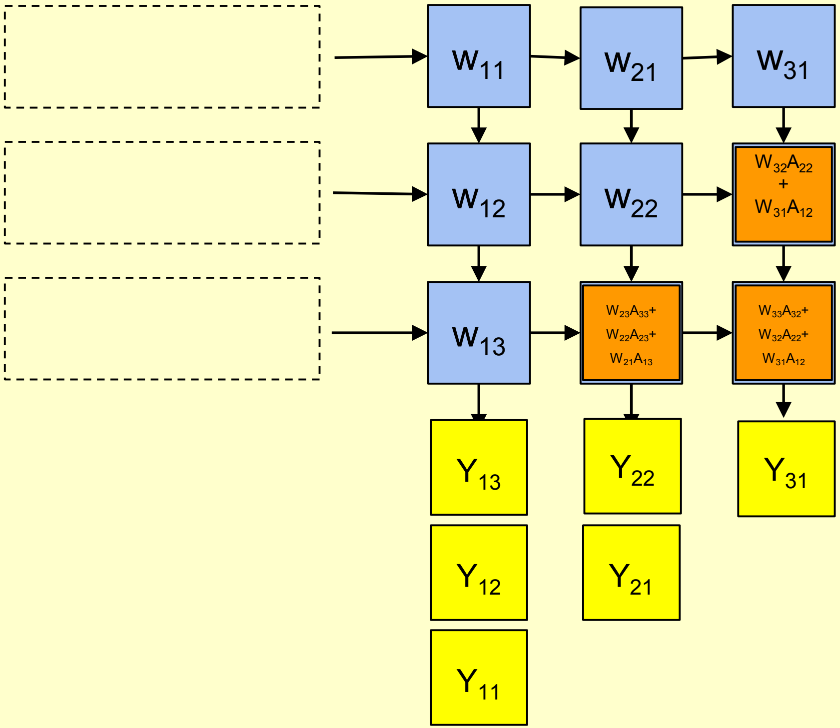
T = 4 (first element emerges)

Flow of Data Through the TPU



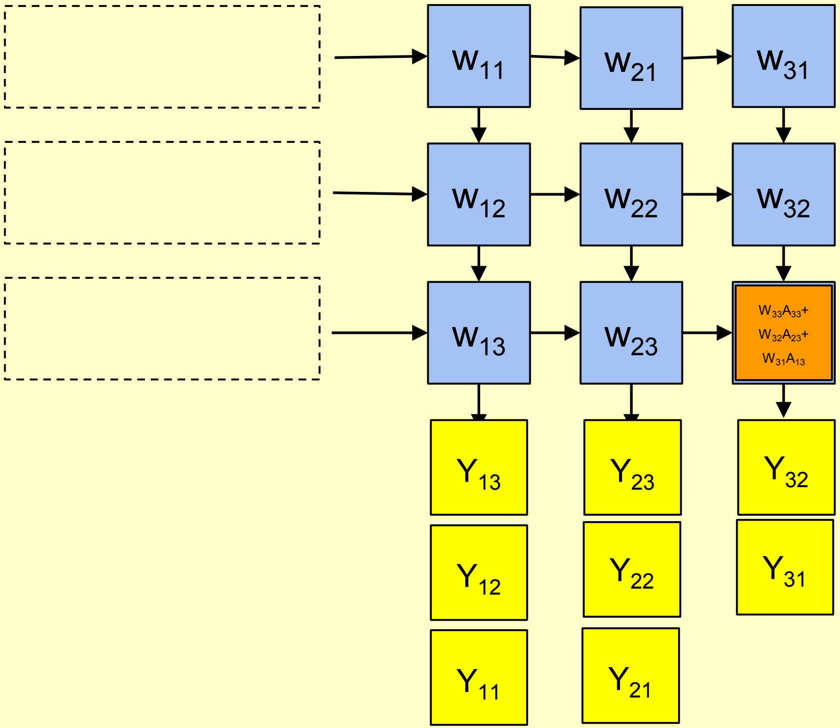
Step 3: Propagate through systolic array
T = 5

Flow of Data Through the TPU



Step 3: Propagate through systolic array
T = 6

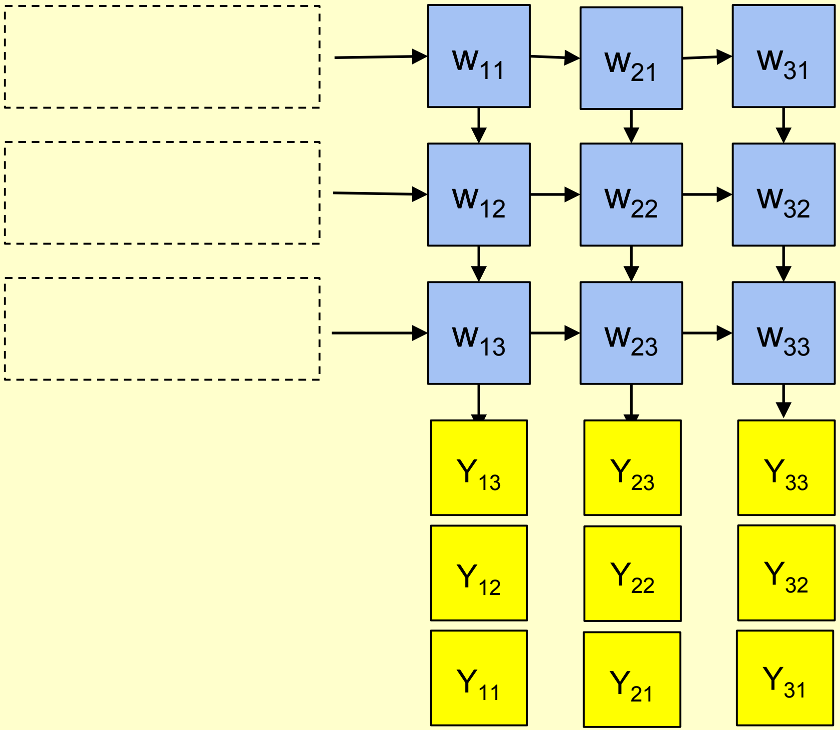
Flow of Data Through the TPU



Step 3: Propagate through systolic array

T = 7

Flow of Data Through the TPU



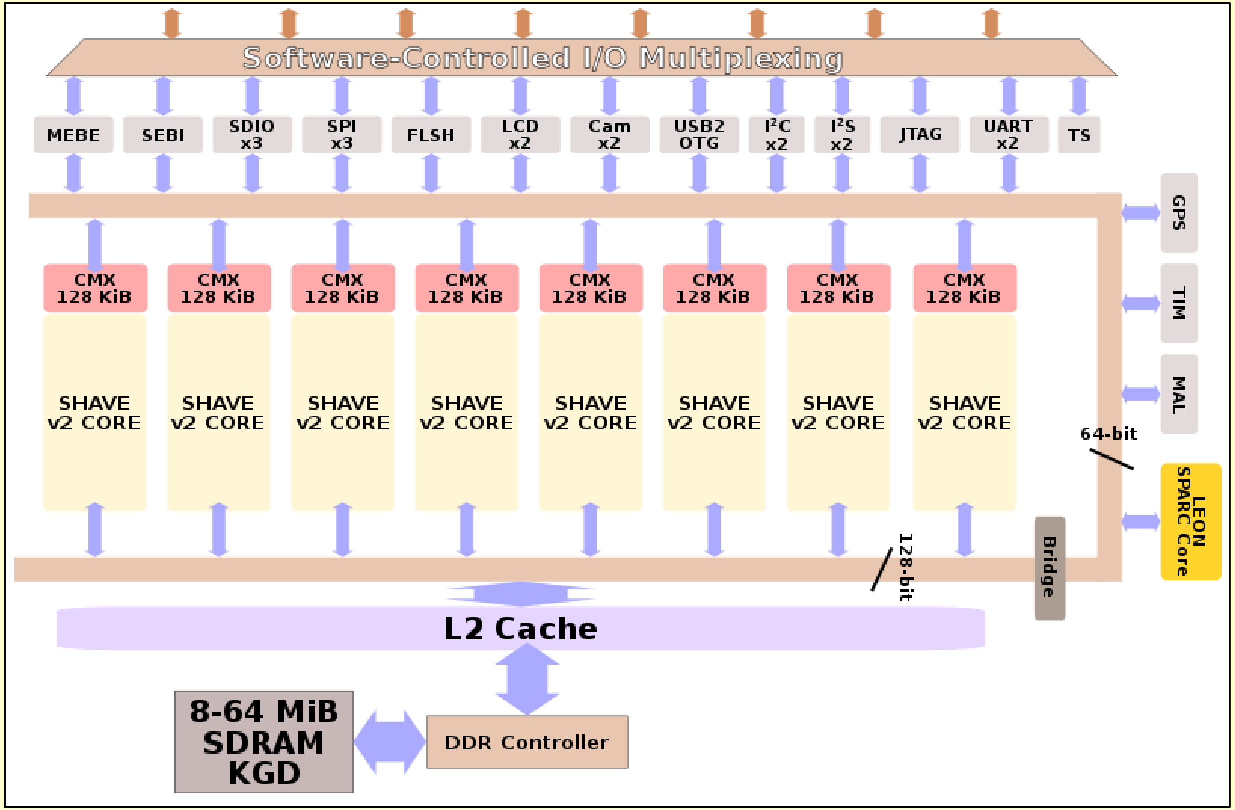
Step 3: Propagate through systolic array

T = 8

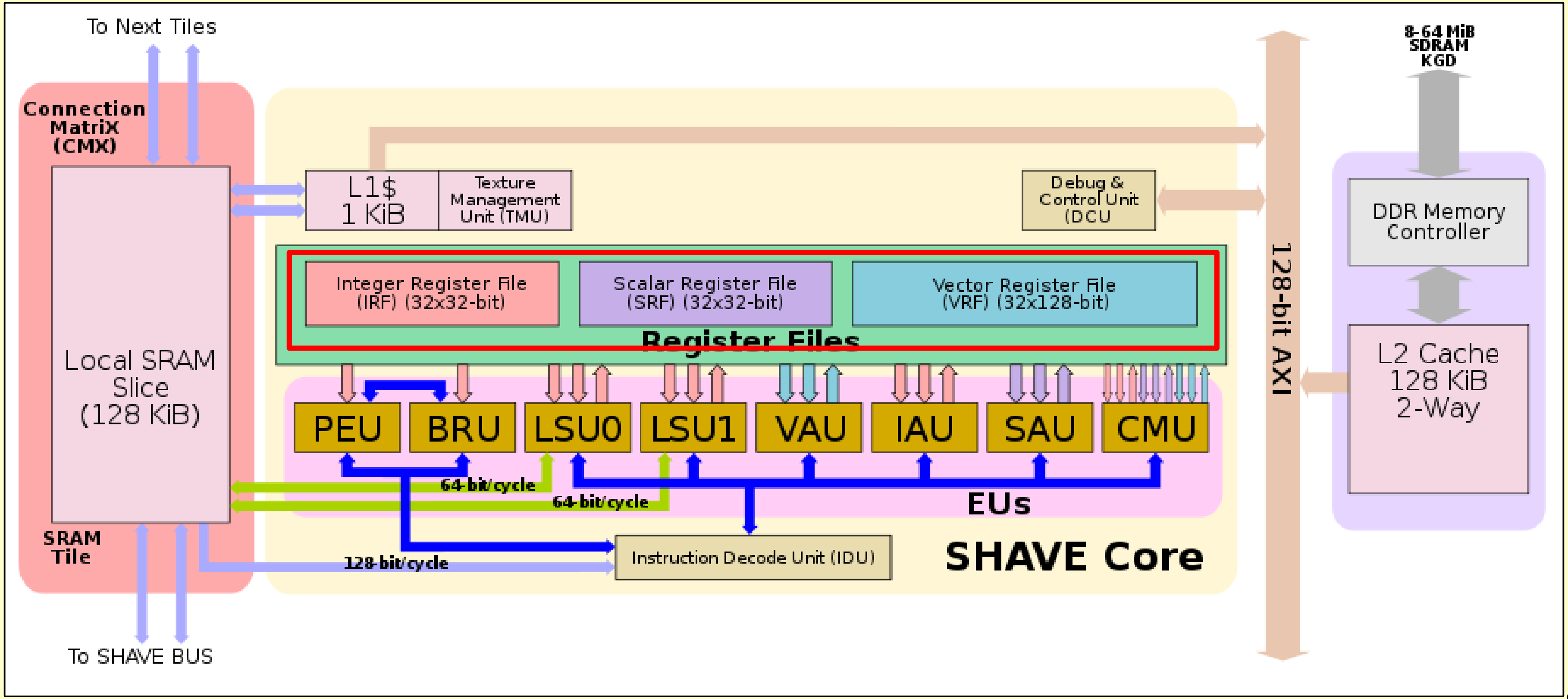
Visual Processing Units (VPUs)

- **Commercially available for applications in computation on “the edge"**
- **Primarily used for inference (prediction)**
- **Best suited for low-power image / video processing**
  - Handles sparse arrays
  - Runs at about .9 Watts (5V at around 180mA)
    - Compare this to ~80 W for a typical CPU
    - Or ~160 W for a typical GPU

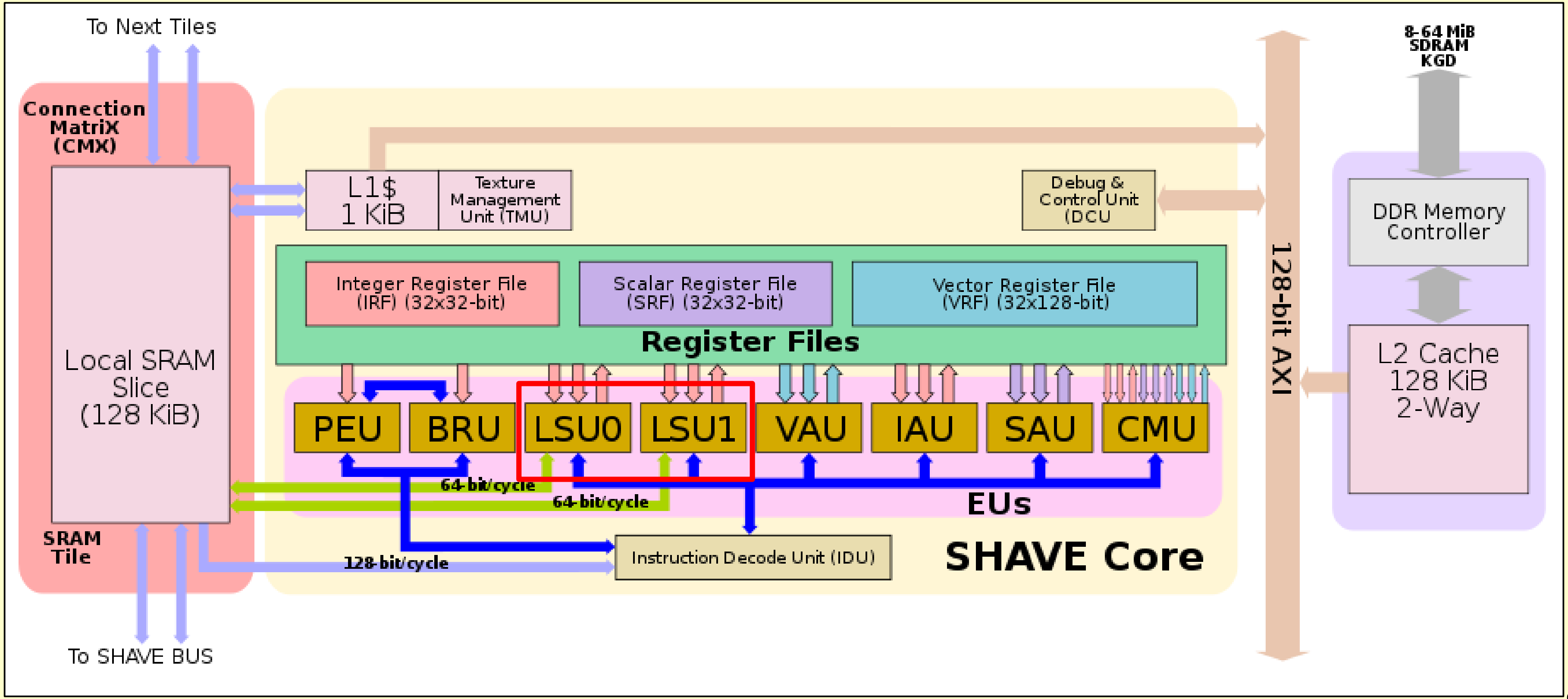
VPU Organization



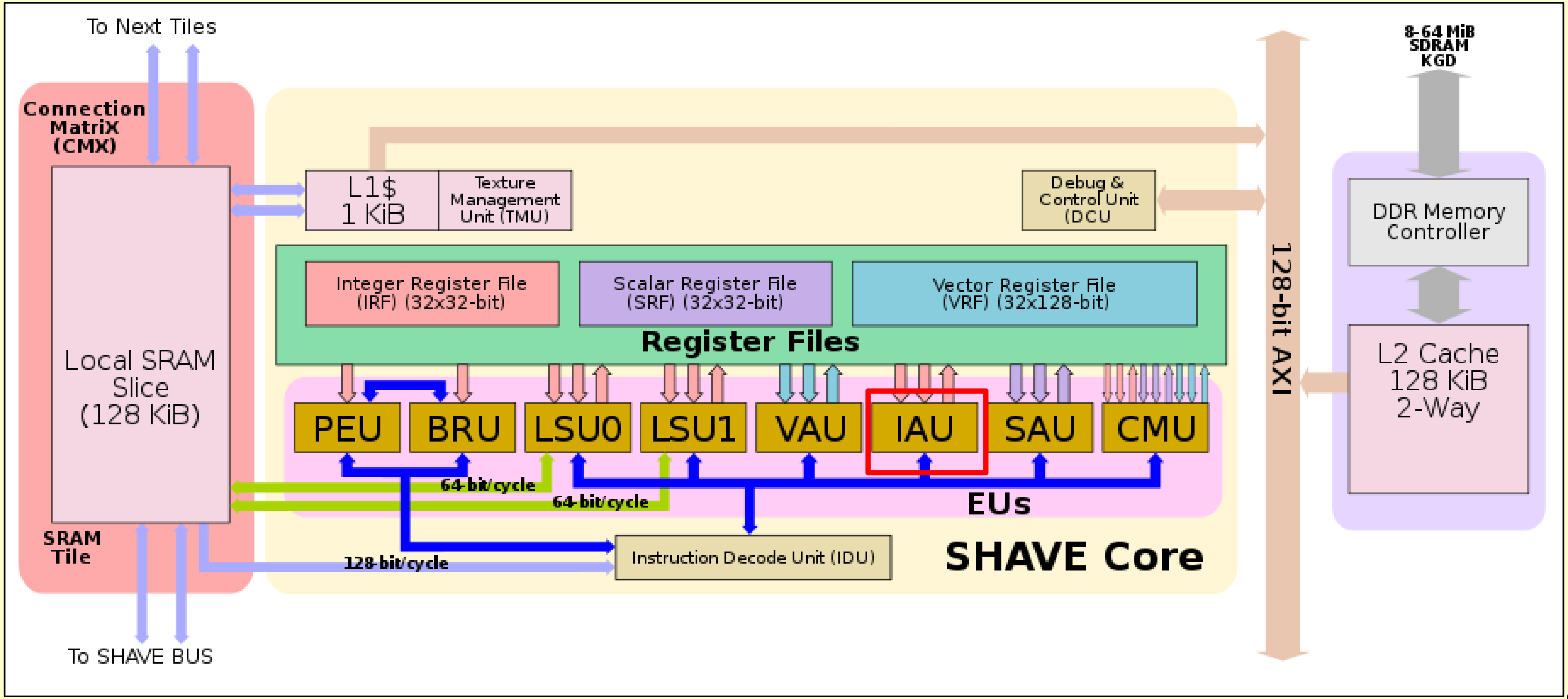
SHAVE Core Organization



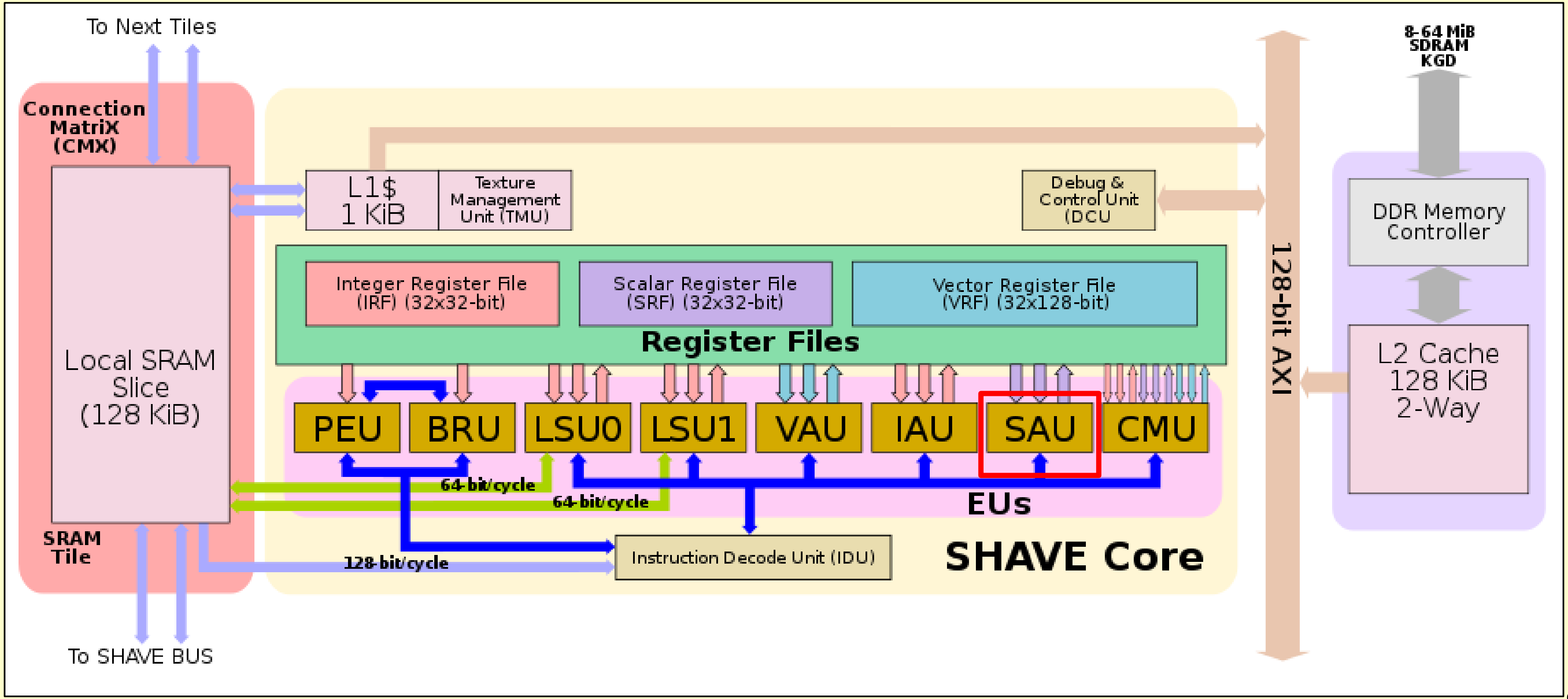
SHAVE Core Organization



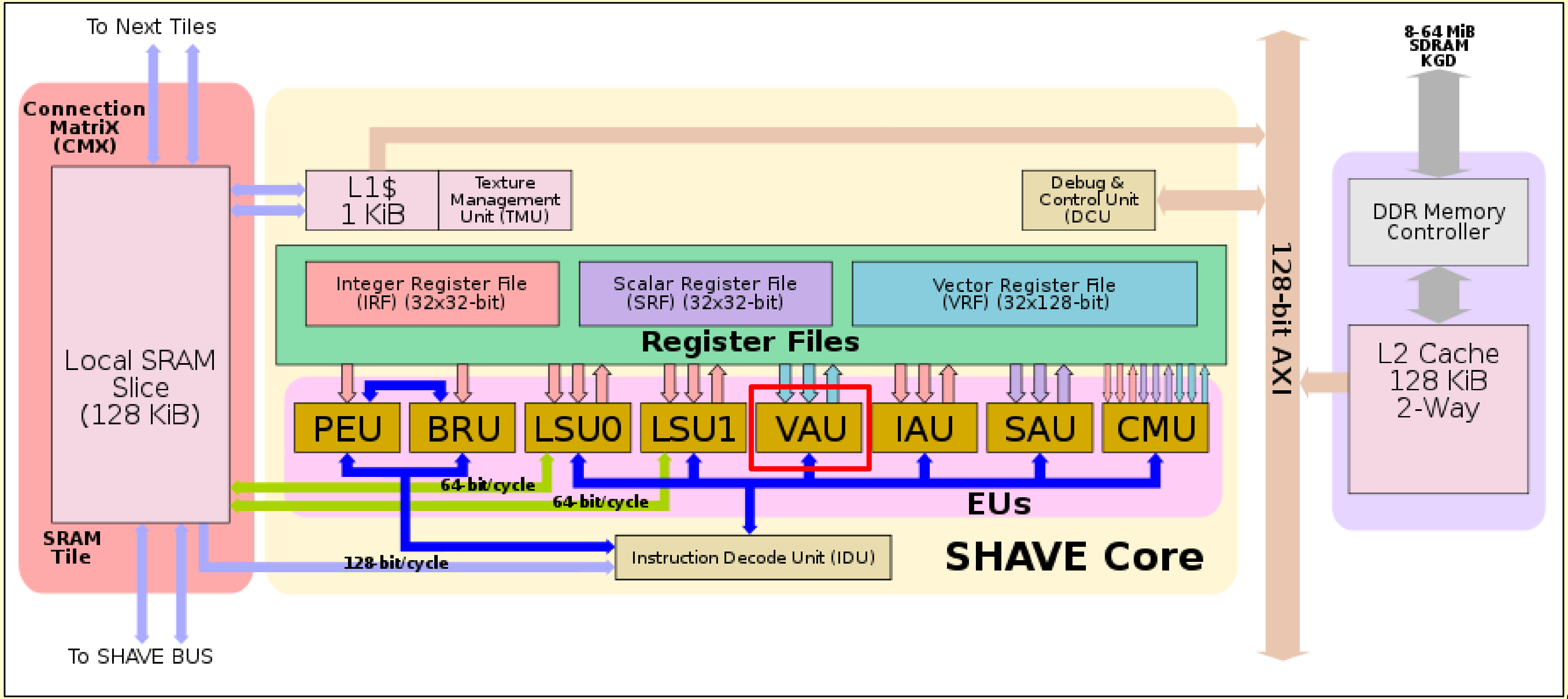
SHAVE Core Organization



SHAVE Core Organization



SHAVE Core Organization



Demo!

Overview

1. **What are Neural Networks?**
2. **How can we speed up Neural Network training?**
3. **What kind of hardware exists that supports neural networks?**
4. **What is the future of hardware support for Neural Networks?**
    a. Current Trends
    b. New Developments

The Future of Neural Networks

- Cloud Computing
- Edge Computing
- Quantum Computing

Cloud Computing for Neural Networks?

Advantages

- Scalability
- Flexibility
- Renting vs Purchasing
- Cloud Benefits

Disadvantages

- Availability
- Customizability
- Security
- Latency
- Bandwidth

Cloud Computing for Neural Networks?

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- IBM Cloud (...IBM Cloud)

Graphic Credits to TensorFlow

Amazon Web Services

- **Amazon SageMaker**
  - AI Development Workflow Software
- **Deep Learning AMIs**
  - Pre-configured Deep Learning Applications
- **TensorFlow and PyTorch Solutions**
  - Using SageMaker or AMIs

Graphic Credits to Amazon

Google Cloud Platform

- **AI Hub**
  - Repository of AI Components
- **AI Building Blocks**
  - Tools for AI Development
- **AI Platform**
  - Code-based Dev. Environment
  - VMs and Hardware

Graphic Credits to Google

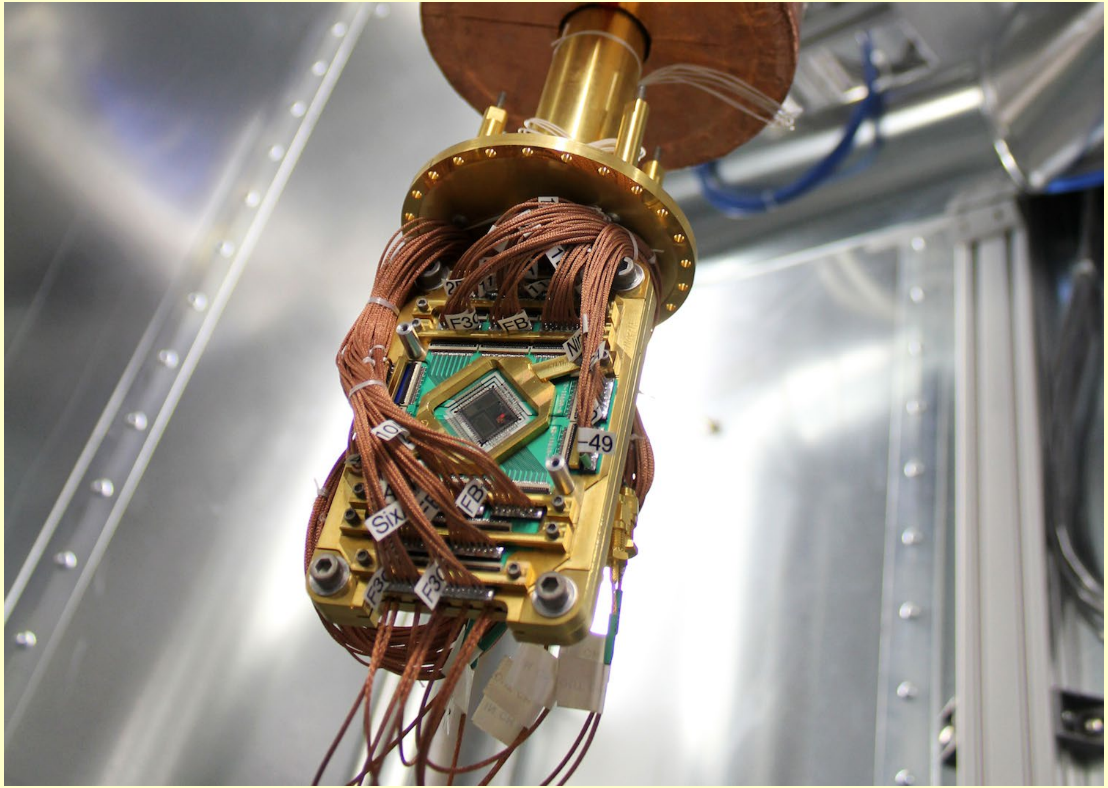
IBM Cloud

- **IBM Watson and IBM Watson Studio**
  - Open, Multi-Cloud AI Platform
- **On-Site with z/OS**
  - Watson ML for Mainframes
- **IBM SPSS® Modeler**
  - Graphical Analytics Platform

Graphic Credits to IBM

Edge Computing

Quantum Computing



Quantum Computing in 2 Minutes or Less

- **Quantum Superposition for QC**
  - Need a two-state quantum system
    - A spin and magnetic field!
    - A photon can be vertically or horizontally charged
  - While unobserved, has a superposition of probabilities for being in either state A or B, so for computing, 1 or 0
  - Once measured, it collapses into state A or B
- **Qbits**
  - Quantum Bits, due to probability, it is a bit that can be 0, 1, or both 0 and 1 at the same time

Quantum Computing in 2 Minutes or Less

- **Bits vs Qbits**
  - 4 bits can only be in one state at once
  - 4 qbits can be in all the possible states of 4 bits at once
  - So, 4 qbits can represent the amount of configurations equal to the amount of possibilities of 4 bits (16)
  - 2^(n) configurations **at once**, since bits have 2^(n) configurations **possible**
- **Even faster with Entanglement**
  - Only need to measure one Qbit in an entangled pair

Quantum Computing in 2 Minutes or Less

- **Logic Gates vs Quantum Gates**
  - Logic Gates have 1 or 2 inputs and 1 definite output
  - Quantum Gates takes in superpositions, manipulates probabilities, and produces a superposition as an output; a probability instead of a definite output
  - Eventually, we measure it, collapsing the Qbits to Bits
    - You just got all possible calculations done at once, giving you the answer that is **probably** right, but checking may be necessary

D-Wave Systems

- They heard all that and said “let’s do that”
- D-Wave QPU
- D-Wave 2000Q System

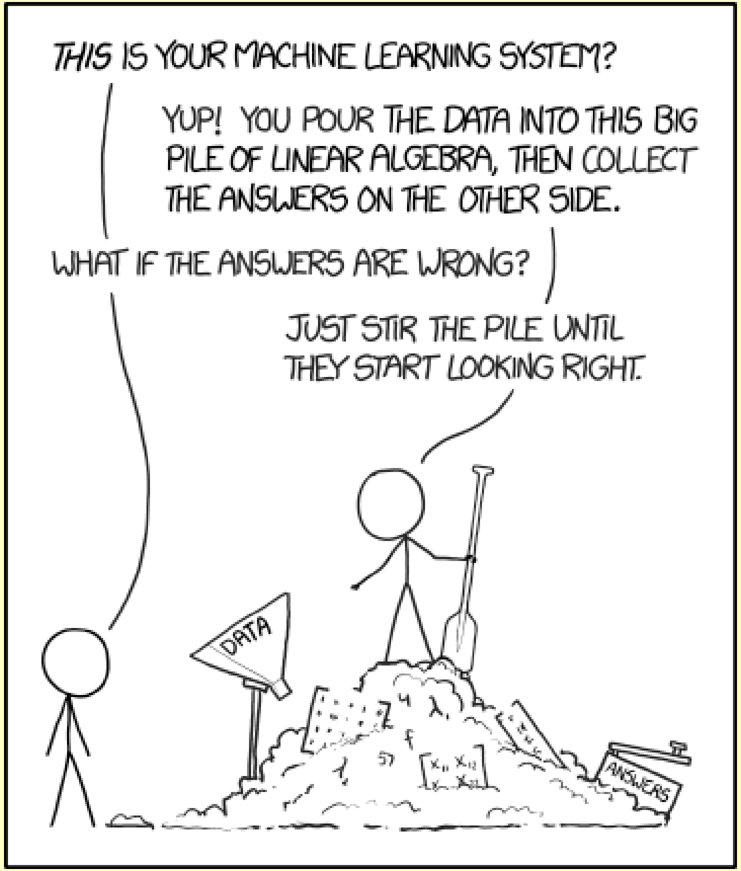
The Quantum Computing Company™

Graphic Credits to D-Wave

Quantum Computing for Neural Networks

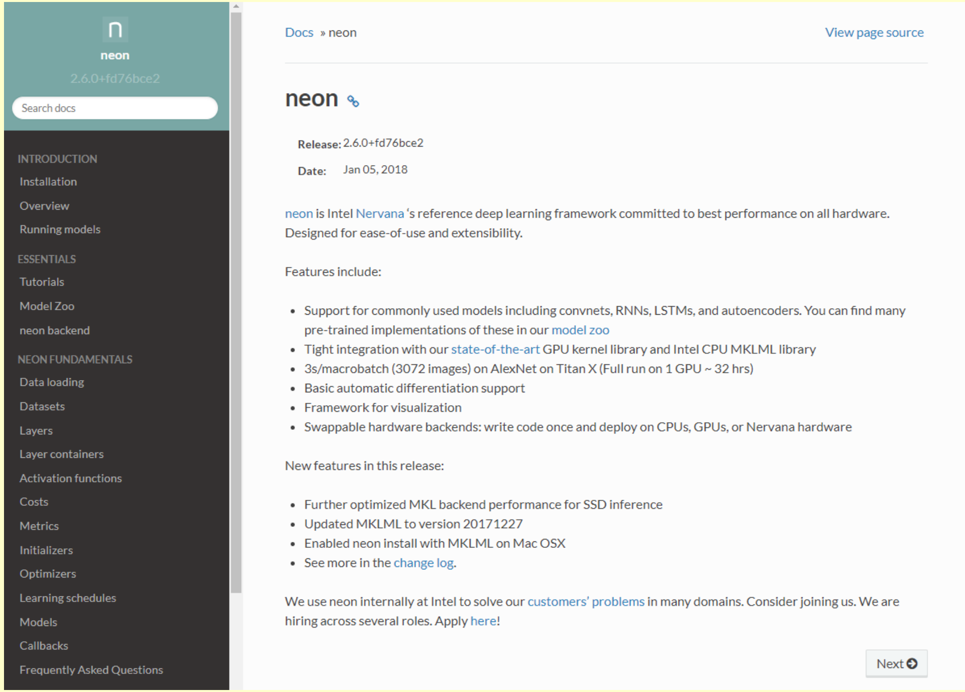
- **Supercomputers vs Quantum Computers: “Comparing apples and fish"**
- **Detecting patterns extremely faster**
  - Experiment with IBM Q system with Raytheon BBN’s team
  - 100x faster with 5 qbits than non-quantum algorithm

Questions?


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

Supporting Slides

Optimization Software



neon

2.6.0+fd76bce2

Search docs

INTRODUCTION
- Installation
- Overview
- Running models

ESSENTIALS
- Tutorials
- Model Zoo
- neon backend

NEON FUNDAMENTALS
- Data loading
- Datasets
- Layers
- Layer containers
- Activation functions
- Costs
- Metrics
- Initializers
- Optimizers
- Learning schedules
- Models
- Callbacks
- Frequently Asked Questions

Docs » neon   View page source

neon

Release: 2.6.0+fd76bce2

Date: Jan 05, 2018

neon is Intel Nervana 's reference deep learning framework committed to best performance on all hardware. Designed for ease-of-use and extensibility.

Features include:

- Support for commonly used models including convnets, RNNs, LSTMs, and autoencoders. You can find many pre-trained implementations of these in our model zoo
- Tight integration with our state-of-the-art GPU kernel library and Intel CPU MKLML library
- 3s/macrobatch (3072 images) on AlexNet on Titan X (Full run on 1 GPU ~ 32 hrs)
- Basic automatic differentiation support
- Framework for visualization
- Swappable hardware backends: write code once and deploy on CPUs, GPUs, or Nervana hardware

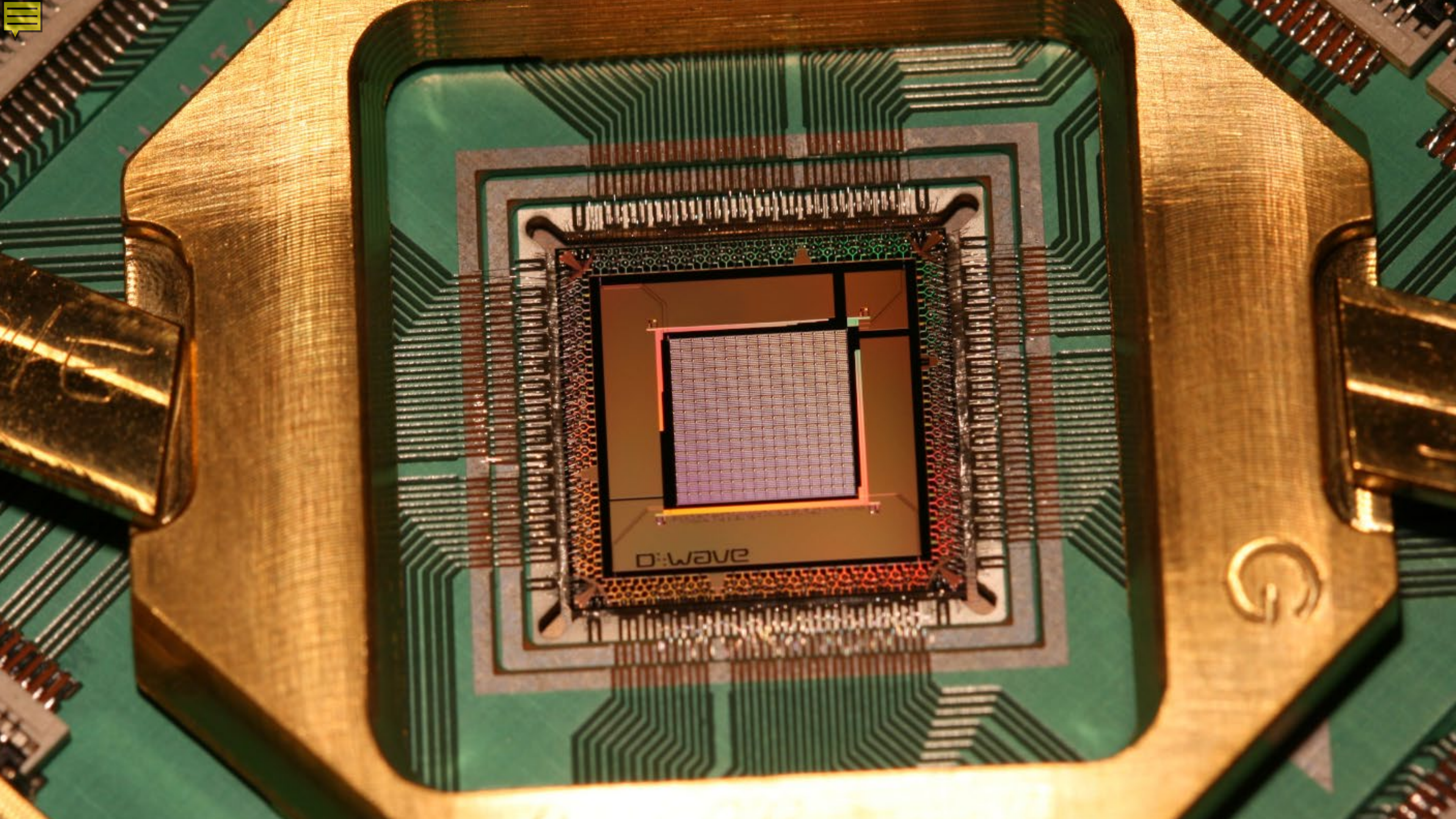
New features in this release:

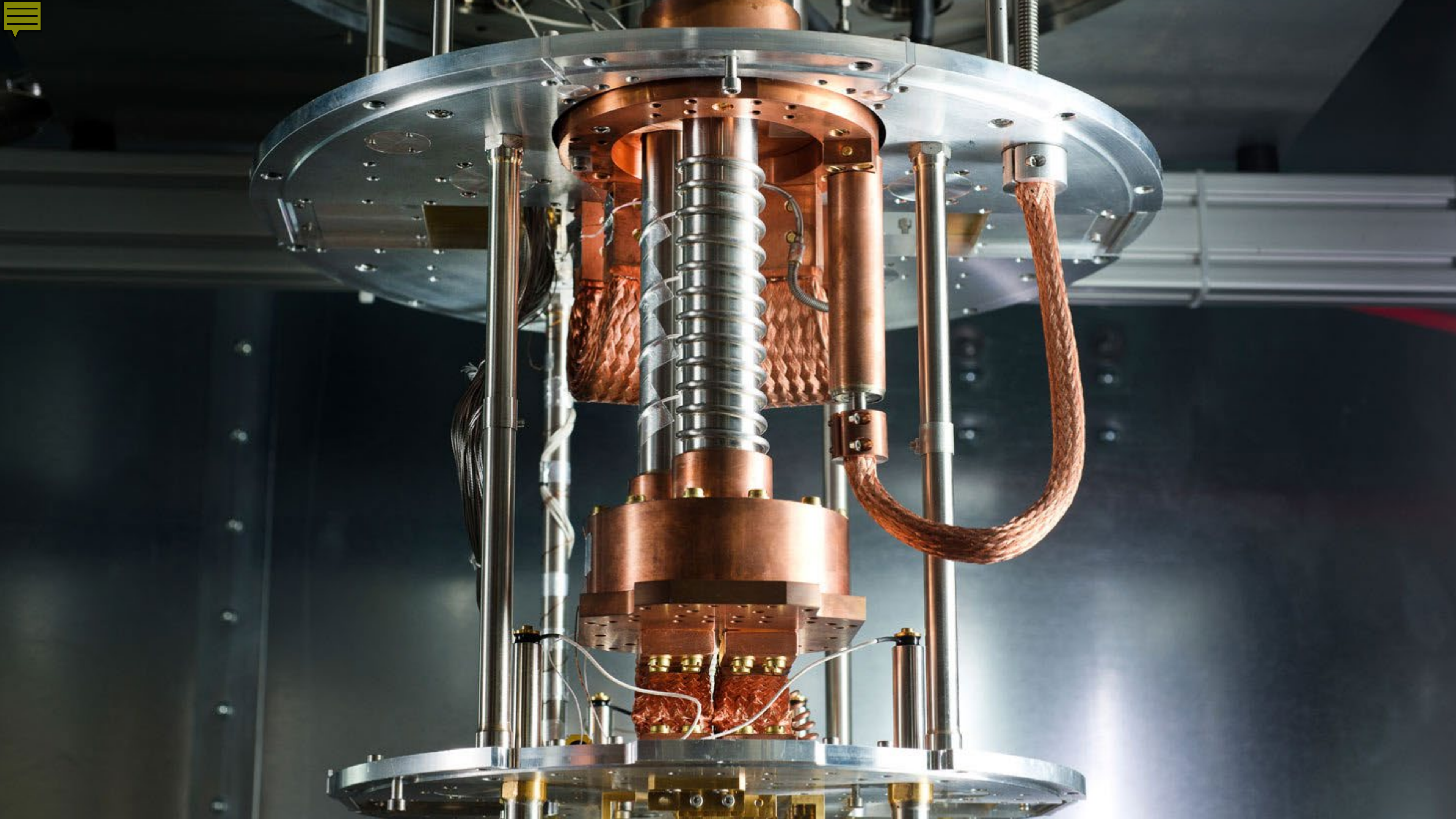
- Further optimized MKL backend performance for SSD inference
- Updated MKLML to version 20171227
- Enabled neon install with MKLML on Mac OSX
- See more in the change log.

We use neon internally at Intel to solve our customers' problems in many domains. Consider joining us. We are hiring across several roles. Apply here!

Next

Software for Neural Network Optimization





Bibliography

1. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/37631.pdf
2. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7929192
3. http://delivery.acm.org/10.1145/1560000/1553486/p873-raina.pdf?ip=130.215.210.57&id=1553486&acc=ACTIVE%20SERVICE&key=7777116298C9657D%2E71E5F5E88B9A3E17%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1554212705_bcfb08af50369dfddd85c3fdf9305560
4. https://en.wikichip.org/wiki/movidius/microarchitectures/shave_v2.0#Floorplan
5. https://medium.com/@antonpaquin/whats-inside-a-tpu-c013eb51973e
6. http://www.telesens.co/2018/07/30/systolic-architectures/
7. https://medium.com/@EskoKilpi/neural-networks-as-the-architecture-of-human-work-3f9d20f019a3
8. https://towardsdatascience.com/artistic-style-transfer-b7566a216431
9. https://www.google.com/search?q=self+driving+car+object+identification&client=ubuntu&hs=W48&channel=fs&source=lnms&tbm=isch&sa=X&ved=0ahUKEwiU-LiP2NLhAhUBVd8KHdQABrsQ_AUIDigB&biw=1329&bih=704#imgrc=Q8KE4T7YDVH1qM: