Accelerator Architectures for Machine Learning

Lecture 1: Course Introduction

Tsung Tai Yeh

Tuesday: 3:30 – 6:20 pm

Classroom: ED-302

Acknowledgements and Disclaimer

- Slides was developed in the reference with
  Joel Emer, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, ISCA 2019 tutorial
  Efficient Processing of Deep Neural Network, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, Morgan and Claypool Publisher, 2020
  Yakun Sophia Shao, EE290-2: Hardware for Machine Learning, UC Berkeley, 2020
  CS231n Convolutional Neural Networks for Visual Recognition, Stanford University, 2020

Outline

- Course overview
- References and text books
- Schedule
- Rating
- AI Accelerator basics

Course overview

- Instructor: **Tsung Tai Yeh**
- TA team^{+}:
  - Zhi-Duan Jiang
- Lecture: T789
- Location: ED-302
- Office Hour: 5 – 6 pm Monday
- My Office: EC 707
- Course web site:
  - https://reurl.cc/q0z7K0

Course website QR Code

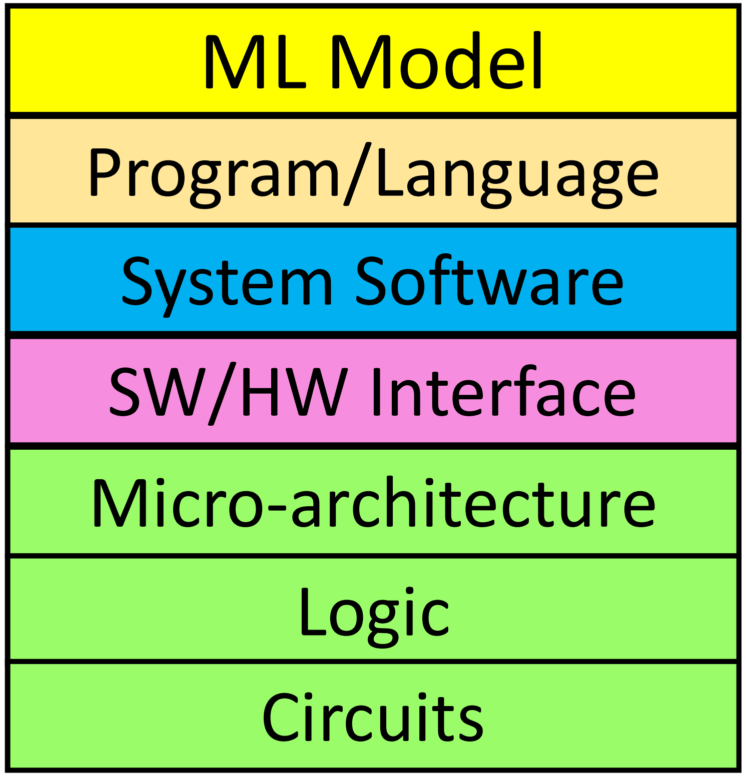
Discussion Forum

- Students should join our class discord discussion forum
- Discord forum
  - Course Announcement
  - Lab
  - Final Project

Discord Forum QR Code

Course overview

- **Efficient Inference**
  - Basics of Deep Learning
  - Quantization + Model Pruning
- **AI Accelerator**
  - Digital/Analog AI Accelerators
- **Edge AI Acceleration**
  - TinyML Acceleration Architecture
- **Lecture + laboratory**
  - Class lecture + 5 labs



Intended Lecture Outcomes (ILOs)

- AAML Course Intended Lecture Outcomes
  - **Understanding** the construction of DNN models
  - **Describing** details of AI accelerators
  - **Implementing** dataflow AI accelerator on Google CFU Playground
  - **Designing** AI accelerator to improve the performance of DNN models

What will you need to do in this course?

- Paper presentation (5%)
  - Groups of students present paper
  - Paper summary writing
- 5 Lab projects (55%) , Lab 1-2 (5%), Lab3-5 (15%)
  - Google CFU Playground
- 2 Quiz (10%)
- 1 Final Project (30%)
  - Optimize a Deep Neural Network Model on CFU Playground
  - Rule: 2 – 3 people/group

Prerequisites

- **Courses:**
  - Basic Programming , Computer Organization, Advanced Computer Architecture
- **You should:**
  - Basic understanding of computer architecture and digital logic design
  - Comfortable with programming in C/C++, Verilog and Python

Lecture

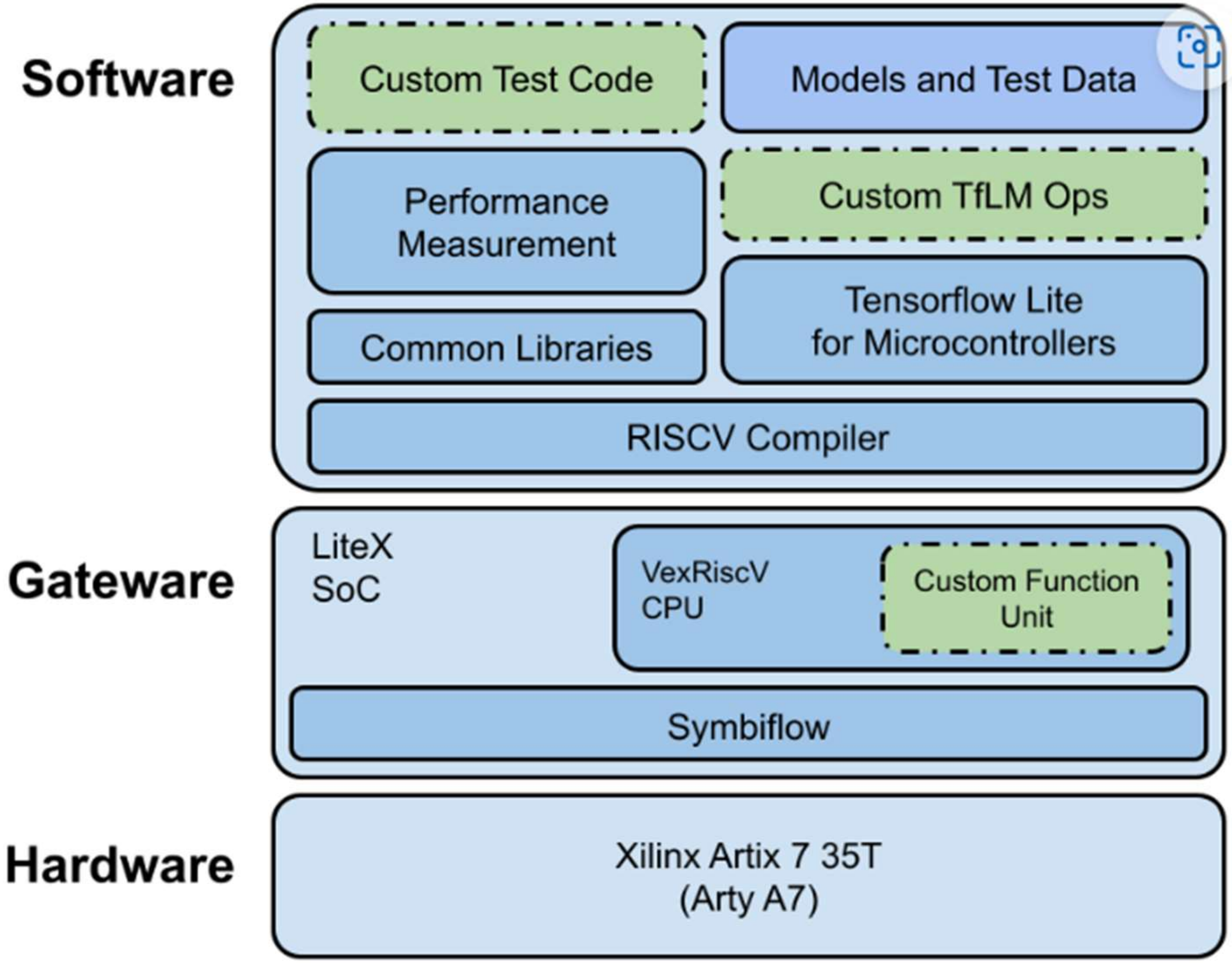
- **Class lecture**
  - This lecture also covers three topics about AI accelerators and DNN models
  - Lecture (2 hours)– summarize course materials of each topic
  - Lab preview or paper presentation (1 hour)
  - Lecture materials have shown on the class website

Lab

- **Platform**
  - **Google CFU Playground on Nexys A7-100T FPGA Board**
- Overview of AAML Labs
  - Build CFU + Run a model
  - CFU + (SIMD + Quantization)
  - Systolic Array Implementation (Verilog)
  - CFU + systolic array
  - Double buffing on systolic array

CFU Playground

- **Google CFU Playground**
  - TensorFlow Lite for Microcontrollers (TFLM)
  - RISC-V CPU + Custom Function Unit
  - Simulation on FPGA



CFU Playground Overview

Lab

- One lab every two weeks
  - Lab 1-2 takes 5% each, 3-5 takes 15% each
- **Lab Demo**
  - Biweekly demonstration
  - Time: 5:20 – 6:20 pm on Tuesday
  - Location: ED 302 or EC222

Final Project

- The final project take **30%** score
- Problem:
  - How to optimize a Deep Neural Network Model on CFU Playground
  - Designing an AI accelerator to improve the performance of a DNN model by using CFU playground

Paper Presentation

- **Paper Presentation**
  - 7 papers, 4 - 5 students are responsible for the presentation of one paper
  - Peer review feedback form – students need to fulfill **10** times attendance, 1% score off when you less than 10 times attendance
  - Each paper presentation takes **5 %** of the total score

Paper Presentation Slide

- The paper presentation slide should include:
  - Paper Title
  - The origin of the paper and year
  - Name of presenters
  - Research problems
  - Contributions and outcome
  - Methodology
  - Evaluation

Schedule

Week	Date	Lecture Topics	Paper Report	Lab Deadline
1	9/12	Course Introduction		
2	9/19	Basics of Deep Learning		
			Efficient Inference	
3	9/26	Quantization	Domain-Specific Accelerator, ACM Comm., 2020 [pdf]	
4	10/3	Pruning and Sparsity		[Lab 1]
			AI Accelerator	
5	10/10	Holiday		
6	10/17	Systolic Accelerator		Lab 2
7	10/24	Digital AI Accelerator	TPU v4, ISCA, 2023 [pdf]	
8	10/31	GPGPU Architecture		Lab 3
9	11/7	GPU Tensor Core	MTIA, ISCA 2023 [pdf]	
10	11/14	Sparse DNN Accelerator		Lab 4
11	11/21	Chiplet Accelerator	Sparse Tensor Core, ISCA 2021 [pdf]	
12	11/28	Analog ML Accelerator	Simba Arch., MICRO 2019 [pdf]	Lab 5
			Edge AI Acceleration	
13	12/5	Basics of TinyML	SNAFU, ISCA 2021 [pdf]	
14	12/12	TinyML Acceleration Architecture	Intelligence Beyond the edge, ASPLOS 2019 [pdf]	
15	12/19	Invited talk		
16	12/26	Final Project		
17	1/2	Final Project		

Textbook

- Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, **Efficient Processing of Deep Neural Network**, Morgan and Claypool Publisher, 2020
- You can download the e-book from NYCU library through EBSCOhost E-book database within NYCU campus

Basics of AI Accelerators

Outline

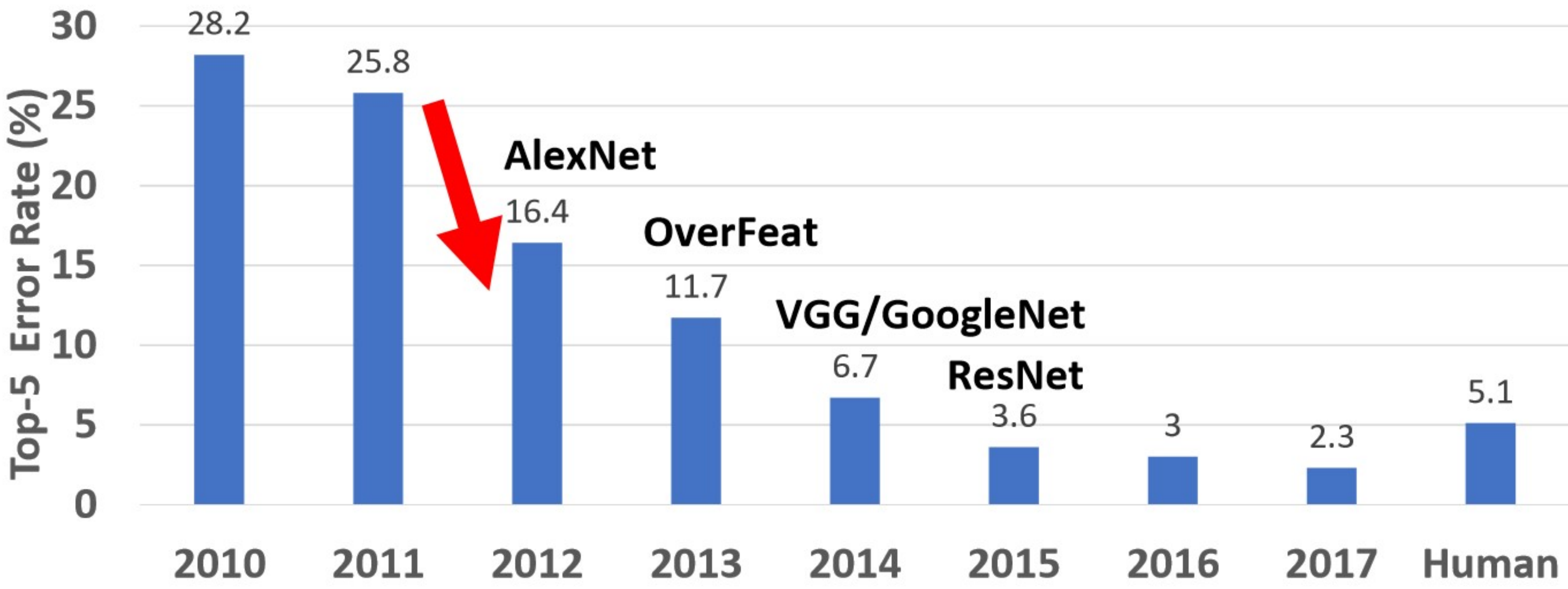
- Dennard Scaling vs Dark Silicon
- Artificial Neural Network (ANN)
- Spiking Neural Network (SNN)
- Neuromorphic architectures
- Digital vs Analog Accelerators

Why do we need accelerators ?

- Previously
  - We focused on designing general-purpose processors
- Why do accelerators have become attractive in recent years?
  - **Dennard Scaling** has ended
    - Dennard Scaling allowed voltage to shrink with transistor size
    - Without Dennard Scaling, we need to find other ways to keep **power** in check
  - **Dark Silicon**
    - Not turn on all transistors on the chip
    - The success of application's accelerators (encryption, compression …)
    - Applications only use subset of processors/accelerators at a time, such a heterogeneous architecture meets dark silicon phenomenon

Why Deep Neural Network become popular?

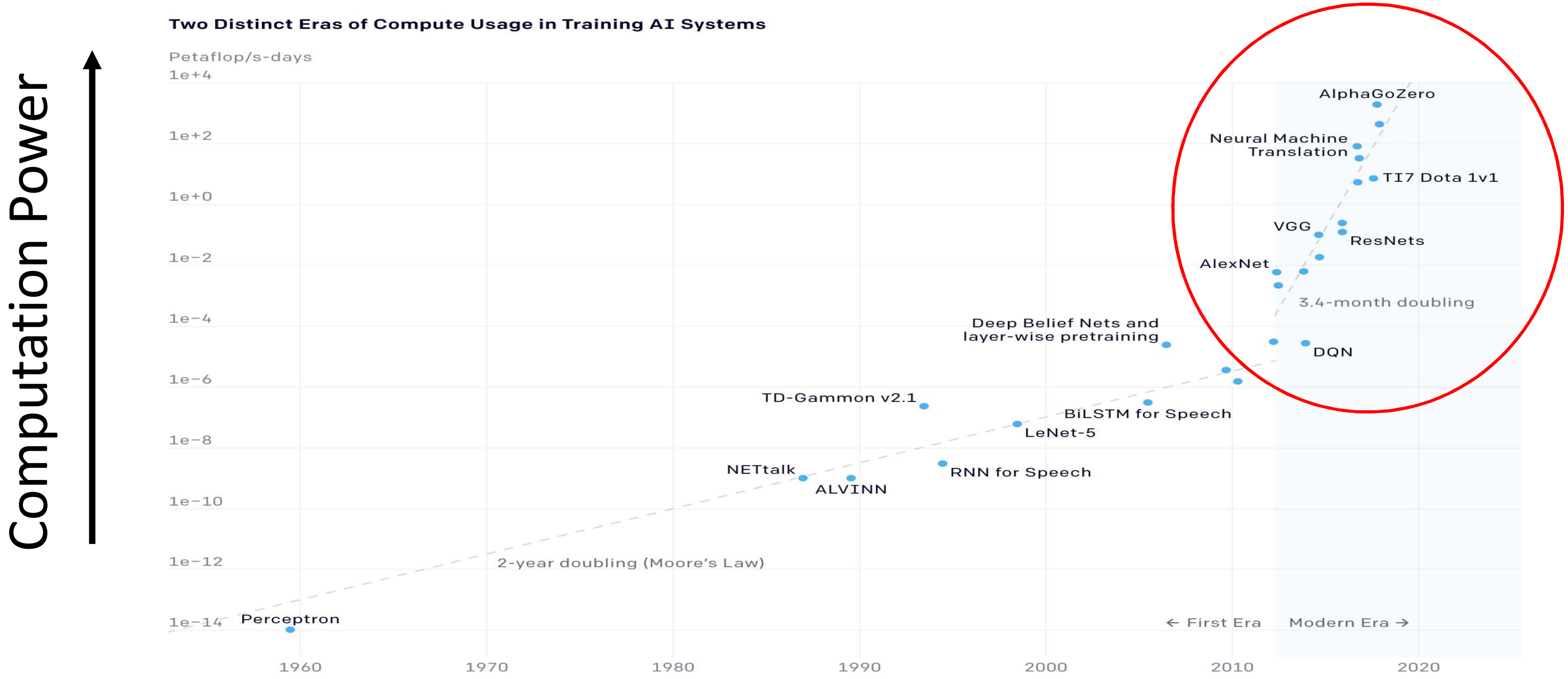
- DNN model outperforms human-being on the ImageNet Challenge



https://arxiv.org/ftp/arxiv/papers/1911/1911.05289.pdf

No free lunch on DNN computation

- AlexNet to AlphaGo Zero: A 300,000 x Increase in Compute



https://arxiv.org/ftp/arxiv/papers/1911/1911.05289.pdf

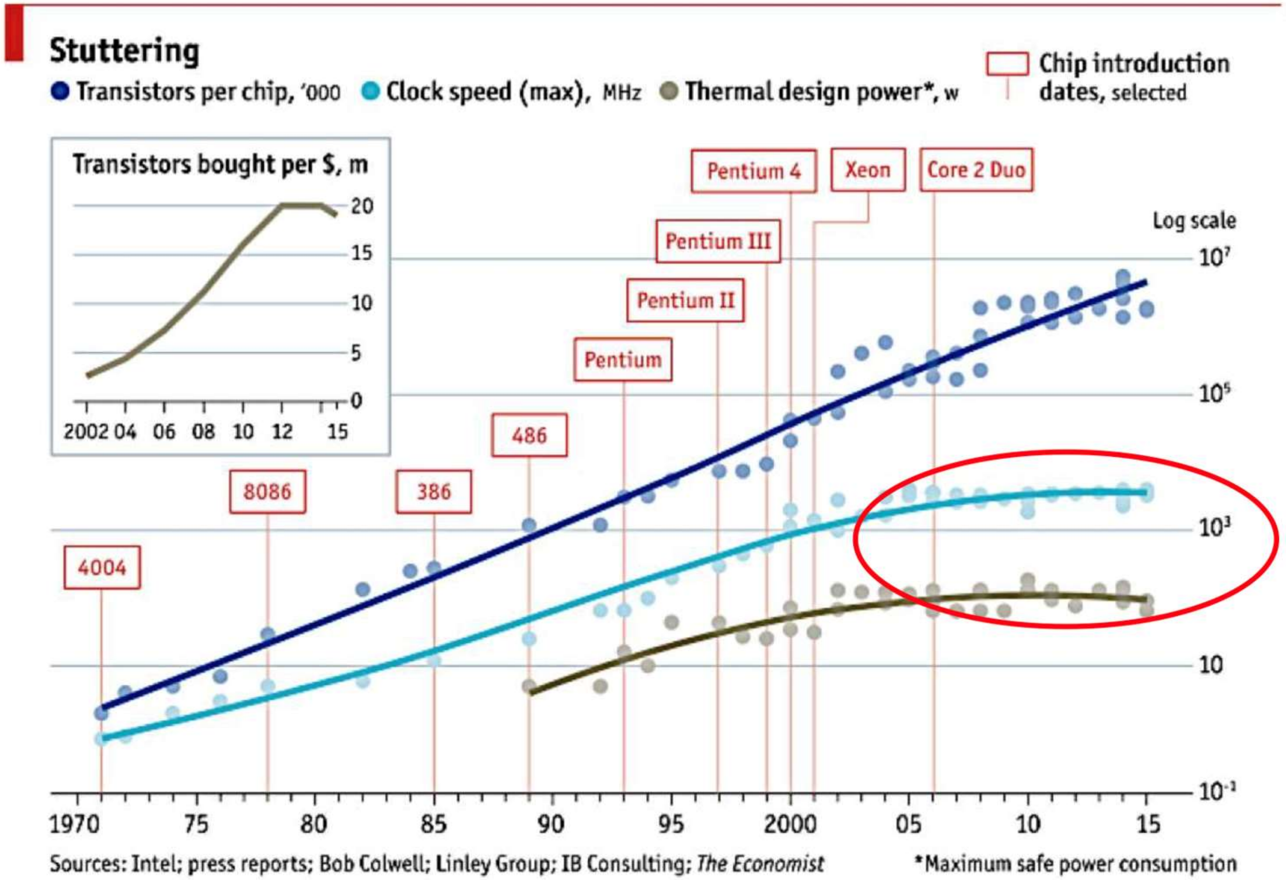
Hardware trends

- **Stagnant single and multi-thread performance on general-purpose cores**
- **Why the emphasis on accelerators ?**
  - Dark silicon (emphasis on power-efficient throughput)
  - End of scaling
- **Emergence of machine learning**
  - Accelerators consumes silicon area and expensive
  - Facilitate the pervasive of hardware acceleration as machine learning emerges as a solution for “everything”.

Commercial Hardware for Machine Learning

- Google TPU (inference and training)
- Nvidia Tensor/transformer cores (Ampere, Hopper)
- Microsoft Brainwave and Catapult
- Intel Loihi NPU
- Cambricon
- Graphcore (training)
- Cerebras (Training)
- Tesla (FSD, Dojo)
- …

Increasing transistors is not getting efficient



General purpose processor is not getting faster and power-efficient because of **Slowdown of Moore's Law and Dennard Scaling**

Dennard Scaling

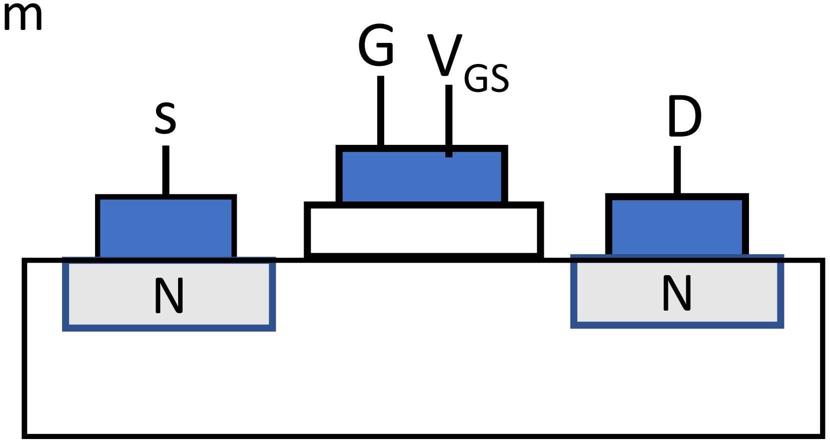
- Dennard scaling allowed voltage to shrink with transistor size
  - E.g. 180 nm -> 1.8 V, 130 nm -> 1.3 V
  - All 4 cores (45 nm) can be worked in full speed
  - Could all 8 cores (28 nm) be worked in full speed, too ? Why ?

Power = alpha x CFV^2
alpha: percent time switched
C: capacitance
F: Frequency
V: Voltage

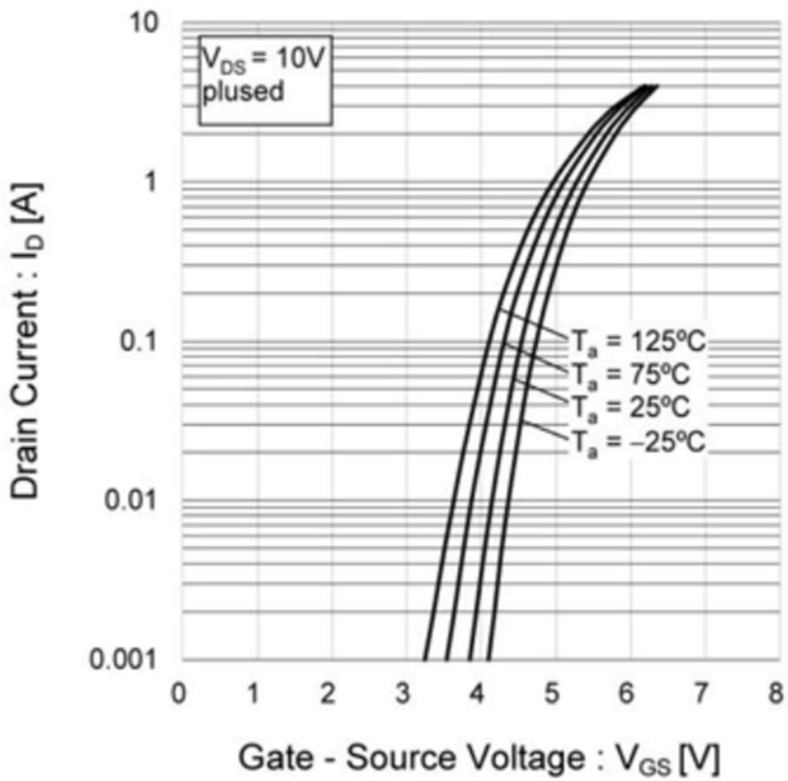
1. Typically, the transistor size reduces K (~1.4) times
2. In the same chip area, **the number of transistor** increases **K^2** times, the **frequency** increases **K** times
3. **The size of capacitance** shrinks **K** times as the reduction of transistor size, and **the voltage** reduces **K^2** times
4. So, we can boost performance of the chip without any compensation of the power

Voltage threshold of MOSFET

- Temperature affects the value of V_{GS} and I_D
  - Ta = 25 d, I_D = 1A and Ta = 75 d, I_D = 1.5 A when fixing V_{GS}
  - Due to $V_{GS\ (TH)}$ constraint, difficult to keep reducing voltage to be proportional to the transistor size below 28 nm



Nch MOSFET



https://techweb.rohm.com.tw/knowledge/si/s-si/03-s-si/5277

What can we do ?

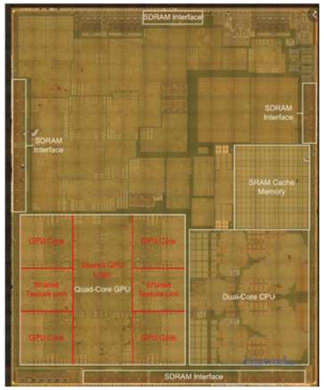
Power = alpha x CFV^2
alpha: percent time switched
C: capacitance
F: Frequency
V: Voltage

- **Dark silicon**
  - Below 28 nm, the voltage is hard to be changed
  - K^2 (transistor size) x frequency (K) v.s. K times capacitance size
  - The power increases K^2 times
  - Therefore, **not turn on all transistor on the chip**
  - What is the percentage of inactive transistors ?
  - 20 nm: 33%, 16 nm: 45%, 10 nm: 56%, 7 nm: 75%, 5 nm: 80%
- **Dim silicon**
  - Turn all transistor on at low clock speeds

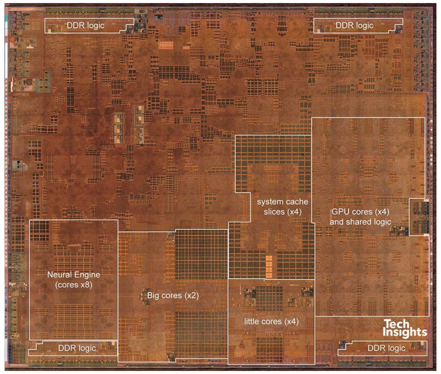
Heterogeneous SoC

- **Post-Moore era and dark silicon**
  - A suite of accelerators on chip are rising
  - Applications will only use a subset of processors/accelerators at a time
  - Such a heterogeneous architecture is compatible with dark silicon

2010 Apple A4
65 nm TSMC 53 mm^2
4 accelerators



2014 Apple A8
20 nm TSMC 89 mm^2
28 accelerators

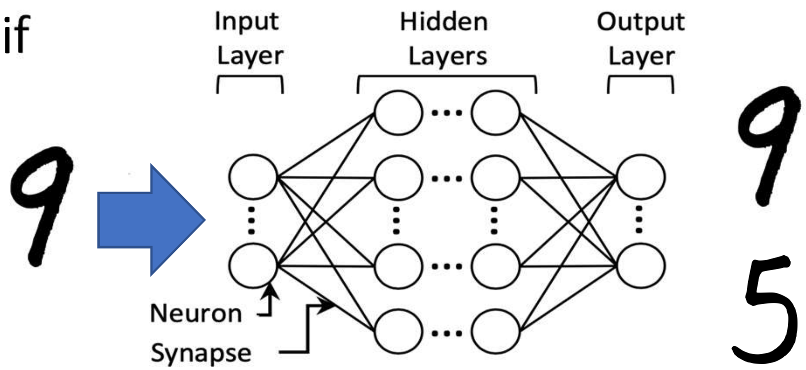


2019 Apple A12
7 nm TSMC 83 mm^2
42 accelerators

https://edge.seas.harvard.edu/files/edge/files/alp.pdf

Artificial Neural Network (ANN)

- **Most machine learning algorithms**
  - Perceptron or artificial neuron
  - Receiving synchronous inputs, and performs math, then produce outputs
  - Measuring the **“strength” (z)** of weighted inputs
  - **Z = x1 * w1 + x2 * w2** where (x is the input of the neuron, w is the weight (determined by training))
  - **Activation function** a = f(z) to decide if a neuron should fire or not
  - **Training performs back-propagation** with gradient descent



Probability: 0.8

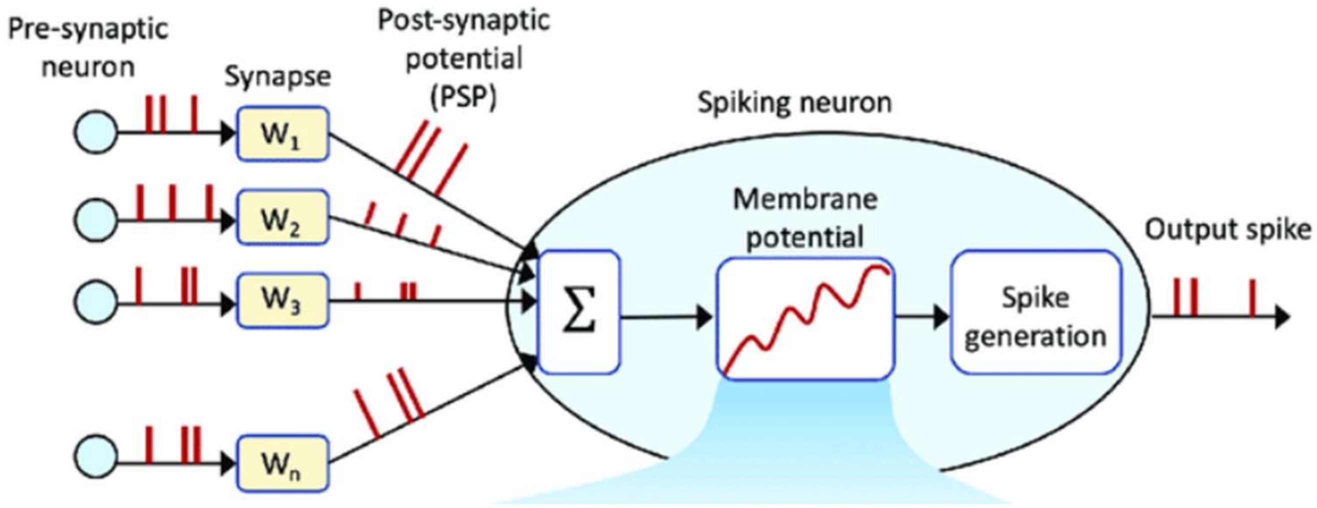
Probability: 0.3

Multilayer perceptrons (MLPs)

https://arxiv.org/pdf/2005.01467.pdf

Spiking Neural Network (SNN)

- Spiking neurons resembles chemical reactions in our brains
  - **A neuron has a certain potential** that represents inputs received
  - The potential **rises and falls** depending on the relative importance of those inputs and leaks away when no receiving inputs
  - When the potential reaches a **threshold**, **the neuron fires**
  - All inputs/outputs are in the form of **binary spikes**



https://www.researchgate.net/figure/A-Schematic-of-a-spiking-neural-network-consisting-of-an-array-of-plastic-synapses_fig1_342414706

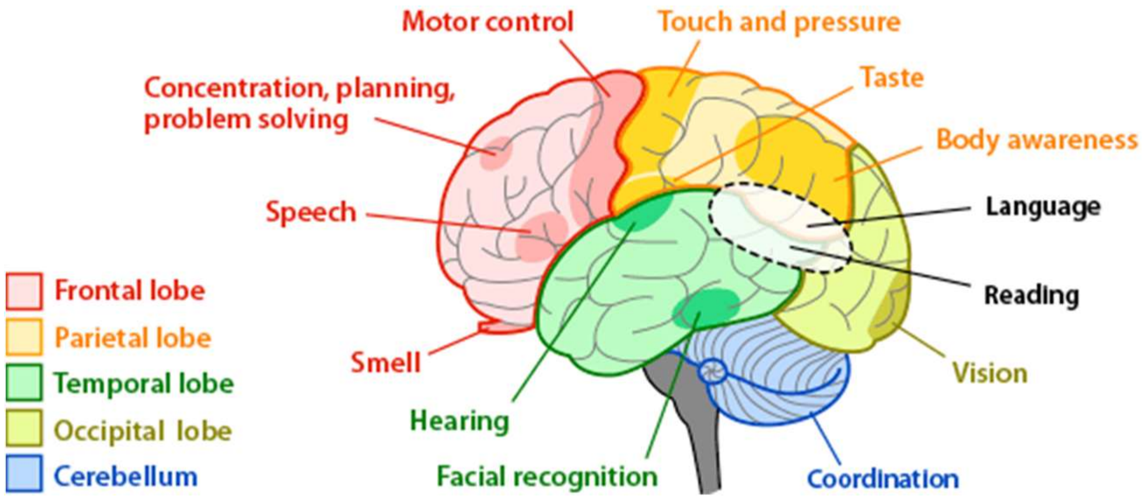
ANN vs. SNN

- **ANN**
  - Perceptron, 8-bit or 16-bit multiplications, complex activation functions
  - High accuracy, supervised learning (inference and training)
- **SNN**
  - Don’t achieve very high accuracies, not well understood
  - A neuron has state that is a more powerful construct for applications that have a notion of time, e.g. video and language analysis
  - Carry a large amount of information in a few bits
  - Unsupervised learning

Uncover Your Brain

2400 kcal/24 hr = 100 kcal/hr = 27.8 cal/sec = 116.38 J/s = 116 W
20% x 116 W = 23.3 W

- The computer as a brain that comprises **specialized accelerators**
- **Low power** – the brain consumes only about 20W
- **Fault tolerant** – the brain loses neurons all the time



https://askabiologist.asu.edu/sites/default/files/resources/articles/nervous_journey/brain-regions-areas.gif

Yang, Eric. Think Dinner. *Mac Evolution, 1998*

Neuromorphic architectures

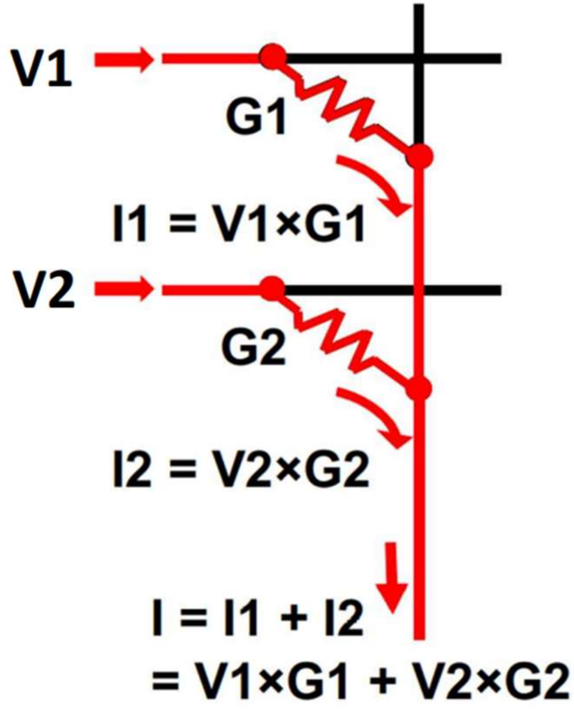
- Architectures inspired by neuron behavior
- **Two major flavors**
  - Artificial Neural Network (ANN)
    - Operations on perceptrons
  - Spiking Neural Network (SNN)
    - Mimic operations in the brain
- **Two major implementation styles**
  - Digital
  - Analog

Neuromorphic Hardware

- **Emulating the human brain**
  - Low power – the brain consumes only 20 W
  - Fault tolerant – the brain loses neurons all the time
  - No programming required – the brain learns by itself
- **Examples:**
  - SpiNNaker, Spikey, TrueNorth

Digital vs. Analog

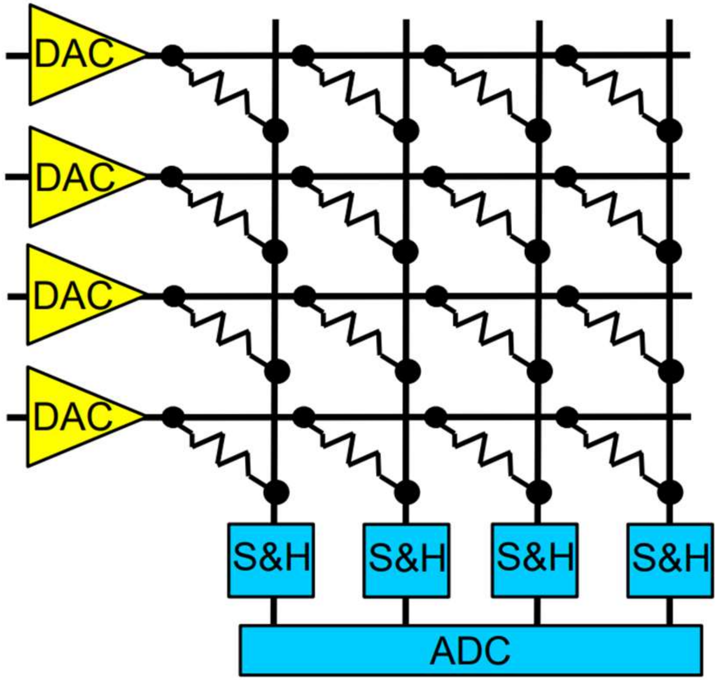
- **A single analog device**
  - Perform multiple multi-bit operations
  - Analog has **challenges, e.g., noise/precision**
  - The **current** in a wire or the **charge** in a capacitor represent **a rational number**
  - Perform **addition** by merging the currents in two wires
  - **Multiplication** can be represented by the current that emerges when a voltage is applied to a conductor
  - **Instability** as temperature changes, currents change
- **Digital device**
  - Use CMOS transistors and gates, exclusively deal with 0s and 1s



ISAAC, ISCA 2016

Crossbar for vector-matrix multiplication

- A grid of **resistances** and horizontal and vertical **wires**
  - The **input voltages** are provided on the horizontal wires (**wordlines**)
  - Each **column** represents a different **neuron**
  - Each column computes a different **dot-product based on conductances** in that column
  - Analog current is sent through an analog-to-digital converter (**ADC**). Why ?
  - **S&H** is the sample-and-hold circuit that feeds signals sequentially to the ADC



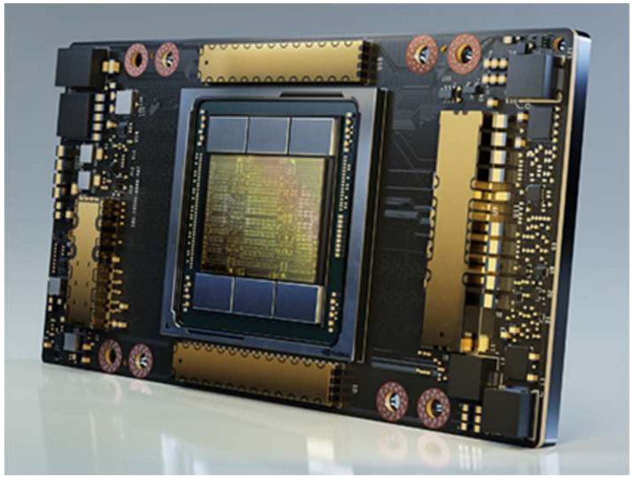
ISAAC, ISCA 2016

Challenges of analog devices

- **High ADC/DAC area/energy**
  - Long stay in analog needs expensive **analog buffering**, introduces **significant noise** that accumulates across network layers
  - Some ADC overheads increase exponentially with resolution
    - The number of bits coming out of a bitline is a function of **the bits of info in the voltage (v)**
    - **The bits of info in the weight (w)**
    - **The number of rows (R)** being added
  - To increase the parallelism and storage density – high v, w, and R
    - Demanding an expensive high-resolution ADC
  - SNN is amenable to analog, why ?

Digital (I) GPU

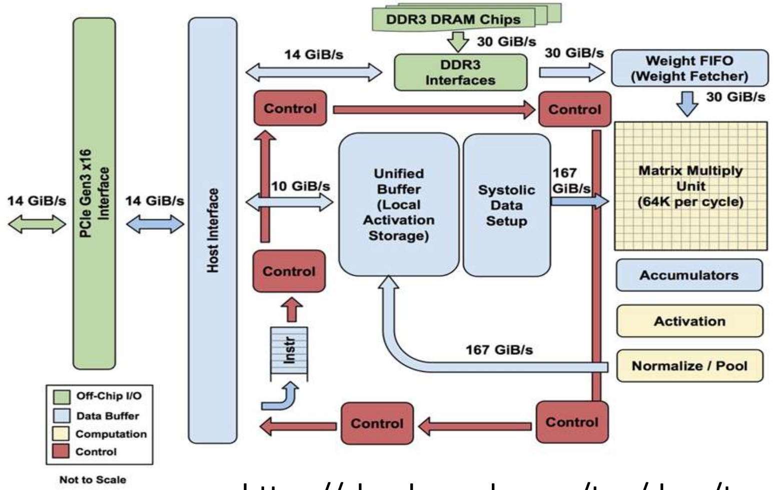
	Nvidia V100 GPU (2019)	Nvidia A100 GPU (2020)	
Transistor count	21 billion	54 billion	2.57 X
FP32 performance	15.7 TFLOP/s	19.5 TFLOP/s	1.24 X
Tensor FP32	125 TFLOP/s	156 TFLOP/s	1.25 X
TDP	300 W	250 W	
Die size	815 mm^2	862 mm^2	
	TSMC 12 nm	TSMC 7 nm	



https://www.nvidia.com/en-us/data-center/a100/

Digital (II) Google Tensor Processing Unit (TPU)

- Systolic-array accelerator
  - V1: Inference only
  - V2: Training with bfloat
  - V3: 2X powerful than v2
- Edge TPU
  - Coral Dev Board
  - 4 TOPS
  - 2 TOPS/Watt
  - Support TensorFlow Lite

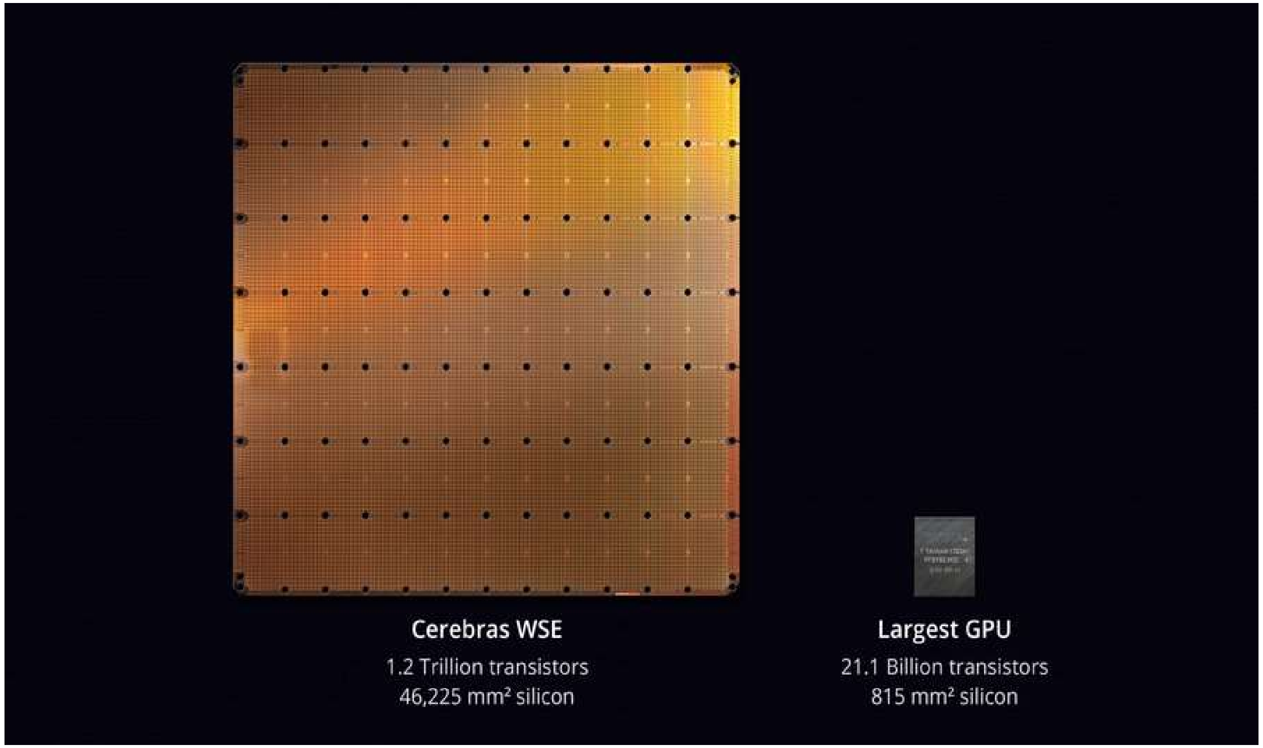


https://cloud.google.com/tpu/docs/tpus

https://coral.ai/products/

Digital (III) Cerebras: Wafer-Scale DL Engine

- Largest DL Chip Ever Built!!
- 46225 mm^2 (WoW !!)
- 1.2 trillion transistor
- 400,000 optimized AI cores
- 18 GB on-chip memory
- TSMC 16 nm process



https://twitter.com/CerebrasSystems/status/1163443985714753537

In summary

- **Learning from History**
  - Neural network (NN) booms, but fades away when it ceases to be fashionable -> support vector machines (SVM) took over
  - General-purpose processors and GPU quickly outpace ASICs
- **Today**
  - NNs > SVM
  - GPPs and GPUs will stagnate in performance, but ML is hot
  - ML accelerators (hardware + ML software perspective) include many implementation operations
  - Neuroscience + emerging technology

Takeaway Questions

- What does dark silicon tell us ?
  - (A) We should turn all transistor on at low clock speeds
  - (B) We cannot turn on all transistors on a chip
  - (C) Allowed voltage to shrink with transistor size
- Why does SNN have the potential for low-energy computations and communication ?
  - (A) SNN is in the form of binary spikes
  - (B) Not involve in multiplications or complex activation functions
  - (C) Skipping connections

Takeaway Questions

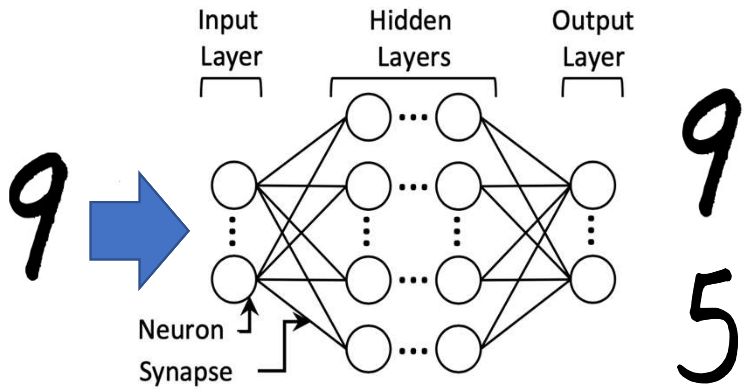
- What are the challenges of analog accelerators ?
  - (A) High ADC/DAC area and energy
  - (B) Limited parallelism
  - (C)Non-programmable

Deep Neural Networks

Outline

- Challenges of Deep Learning
- Non-linear activation
- Gradient vanishing/exploding problem
- Batch normalization
- Feed forward/Backpropagation

Deep Neural Networks

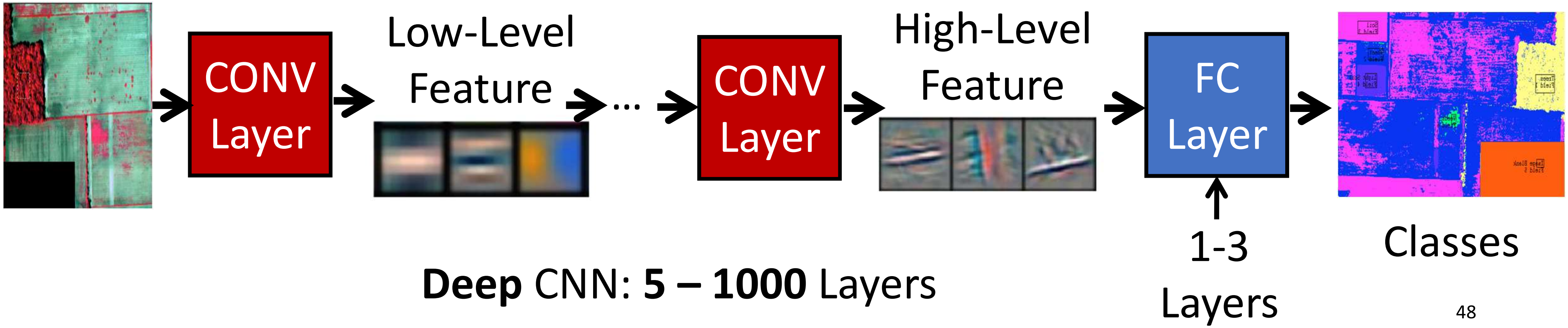


Multilayer perceptrons (MLPs)

Probability: 0.8

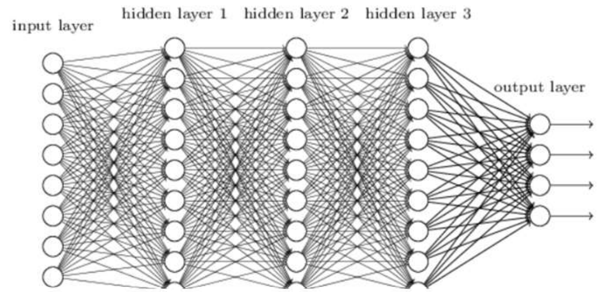
Probability: 0.3

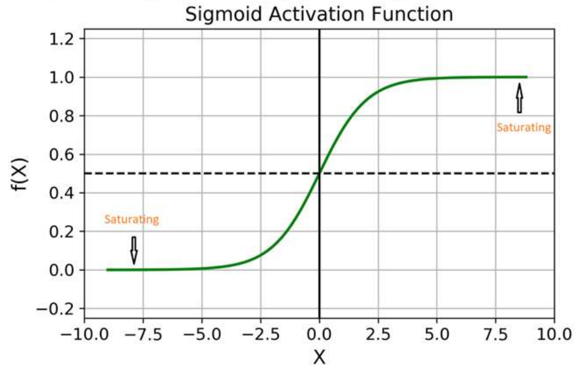
- **Deep vs. Shallow**
  - Learn simple features in early layers
  - More complex features in subsequent layers
  - Instead of recognizing an object with a single magical neuron



Deep CNN: **5 – 1000** Layers

Deep Neural Network





- **A deep neural network**
  - Learn simple features in early layers
  - More complex features in subsequent layers
- **Gradient vanishing**
  - Small gradients are propagated through the last layer to the initial layer in the backpropagation.
  - Activation function leads to gradient vanishing
    - The derivative is close to 0 when the inputs are fairly large or small
    - No update in weights of lower layer
  - Potential solutions
    - Change activation function
    - Batch normalization

https://debuggercafe.com/neural-networks-the-problem-of-vanishing-gradients/

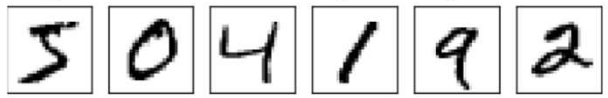
Deep learning challenges

- The creation of trainable deep networks
- **Vanishing/exploding gradients**
  - Different learning rates for different layers to reduce this problem
  - Early layers may have slow learning rate
- **Picking the correct activation function**
- **Good initialization of weights**
  - The choice of weights and activation functions can impact learning rates
- **Choice of network architecture, hyper-parameters, etc.**

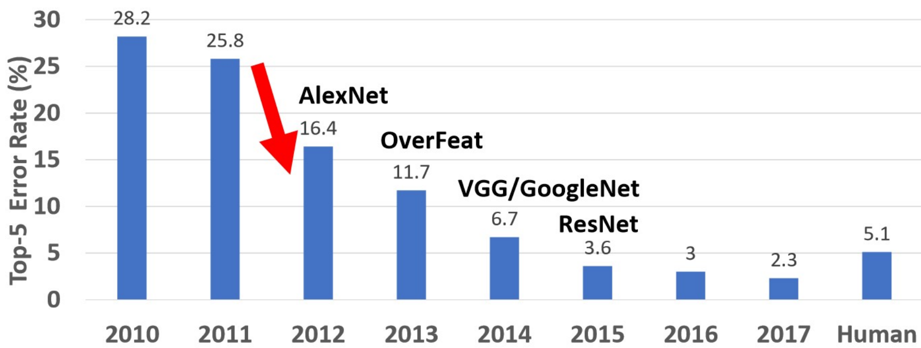
Deep networks for image classification

- Image classification is one of success stories for deep learning
- **MNIST**
  - 784-pixel hand-written image digits; 50K training, 10K testing images
- **ILSVRC**
  - 1000 categories, 1.2 million training images 150K test images
  - Top-5 criterion
    - Make 5 best guesses if one of these matches the label, the prediction is deemed accurate

ILSVRC dataset



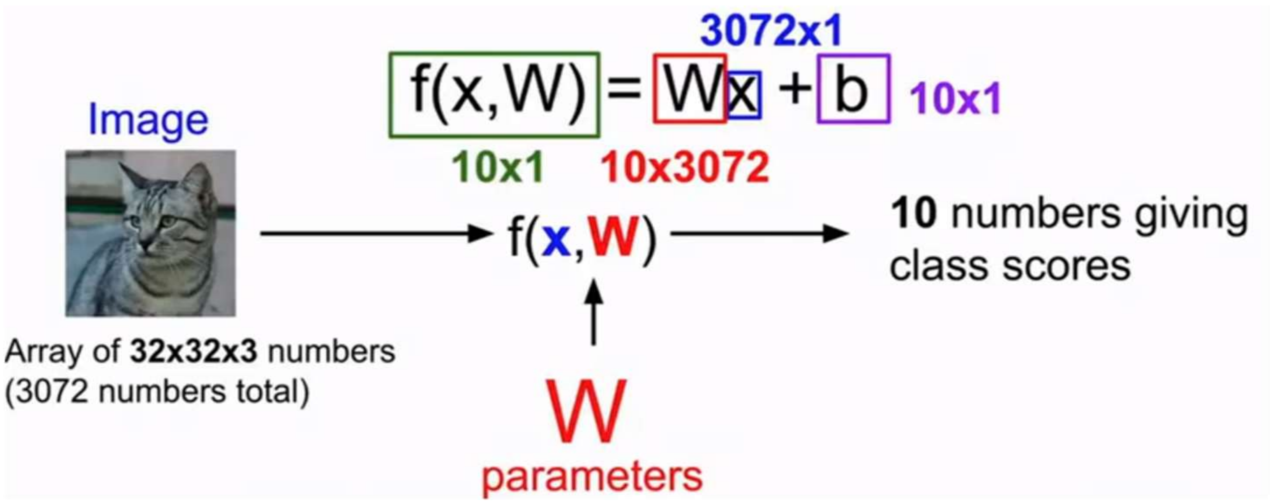
MNIST dataset



https://arxiv.org/ftp/arxiv/papers/1911/1911.05289.pdf

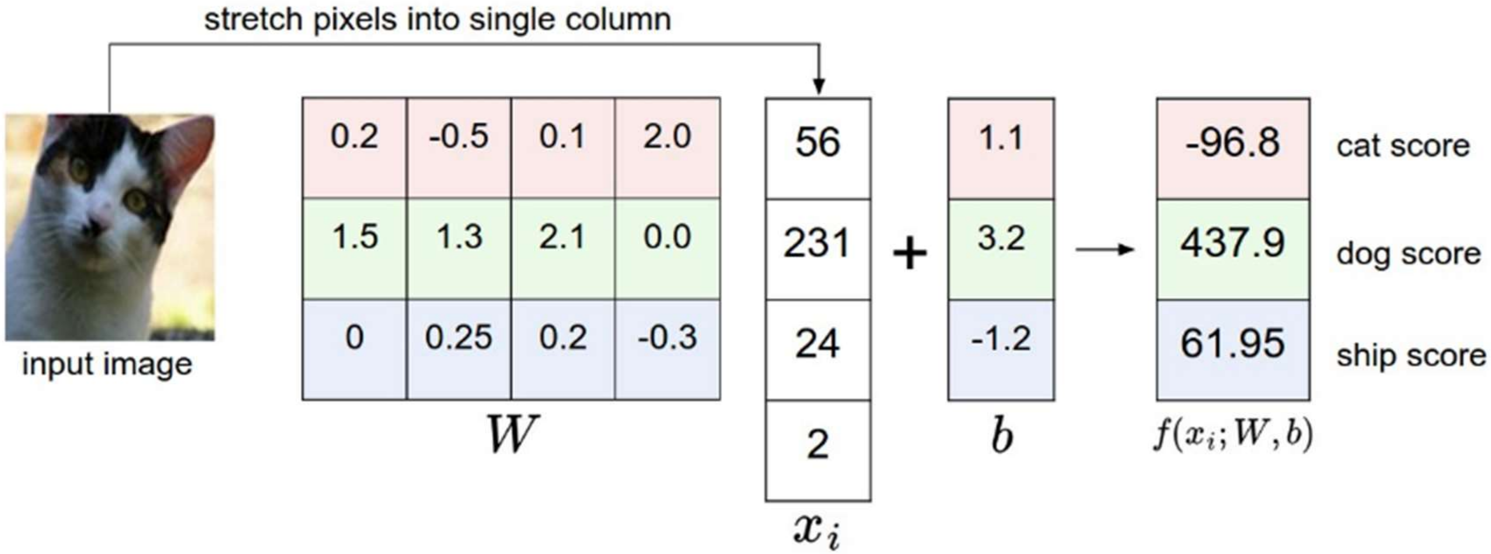
Linear Classification

- **Parametric approach**
  - x: input image (CIFAR-10 input is the array of 32 x 32 x 3)
  - W: weights (each class has its specific weights, CIFAR-10 has 10 classes -> CIFAR-10 weight has (32 x 32 x 3) x 10)
  - b: bias (fine-tuning the model)



Linear Classification

- Linear classifiers get weights (W) from training sets
- Weights contain properties of each class



How to Initialize Weights in Practice?

- **Zero initialization**
  - What happen if weights are initialized with 0 number ?
    - Undergo the exact same parameter updates
    - No source of asymmetry between neurons
    - How about weights are initialized with 1 number? -> symmetry breaking
- **Random initialization**
  - If weights have very large/small random numbers, what's wrong?
    - The slope of gradient changes slowly and learning takes a lot of time
- **Calibrating the variances**
  - W = np.random.randn(n) / sqrt(n), where n is the number of its input
  - Ensure all neurons in the network initially have approximately the same output distribution -> improve the rate of convergence

https://cs231n.github.io/neural-networks-2/#init

What is bias in neural networks?

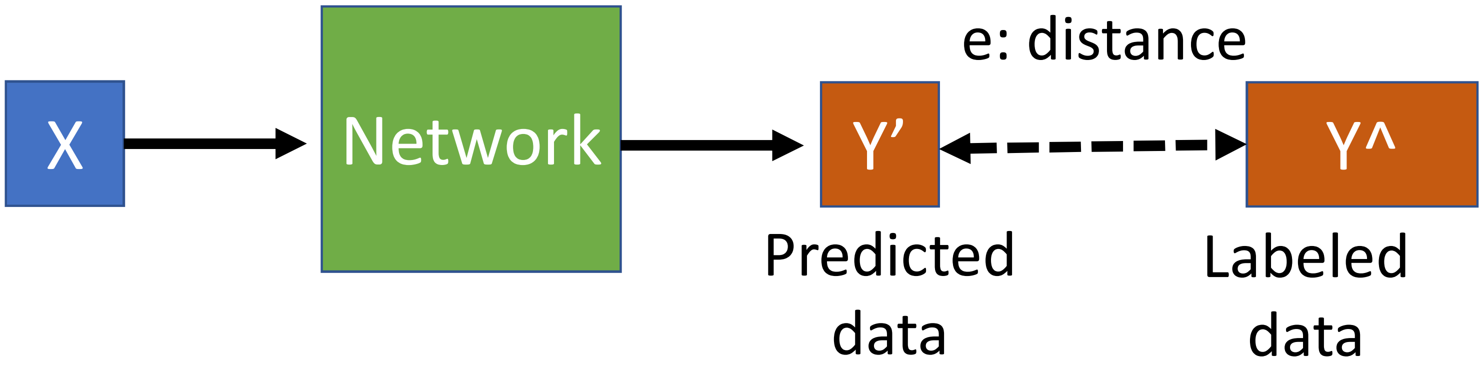
$$\boxed{\text{Output = activation\_function(dot\_product(weights, inputs) + bias)}}$$

- **Bias** shifts activation function to better fit the data
- Why can we initialize the biases to be zero?
  - The asymmetry breaking is provided by the small random numbers in weights
- The bias only impacts the output values
  - A node with a large bias, the output value tends to be high
  - Negative bias value -> sigmoid outputs are near to 0
  - Very small bias (or 0) -> weights and inputs dominate the outputs
- Could we set bias value to **0** initially?

$$L = \frac{1}{N}\sum_{n} e_n$$

Loss Function

- The difference between the correct and predicted data
- We want to set weights during training
- Making the predicted scores are consistent with the ground true labels in the training data.



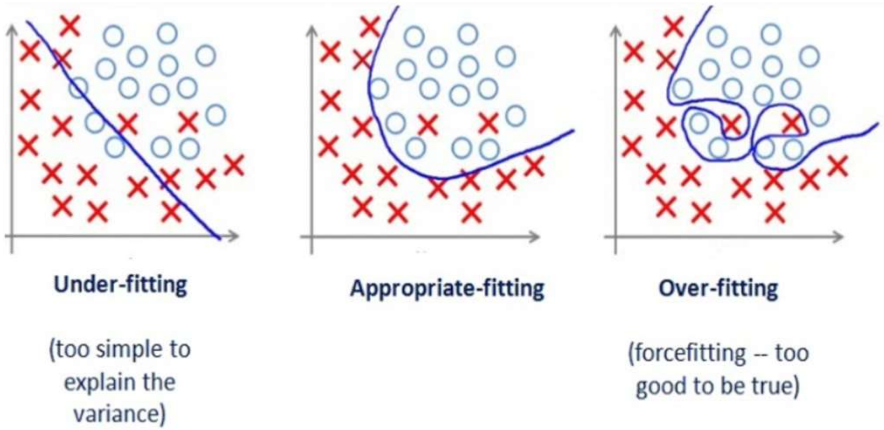
- Mean Square Error (MSE):
- Cross Entropy:

$$e = -\sum_{i} \hat{\boldsymbol{y}}_i \ln \boldsymbol{y}'_i$$

$$e = \sum_{i} (\hat{\boldsymbol{y}}_i - \boldsymbol{y}'_i)^2$$

Overfitting on DNN Training

- **Generalization problem**: how well the model maintains the accuracy between training and unseen data (testing dataset)
- **Overfitting:** Low error rates in the training, but high error rates in the test data
- Our goal doesn’t create a model only fitting the training data
  - We want to create a model targets for new data

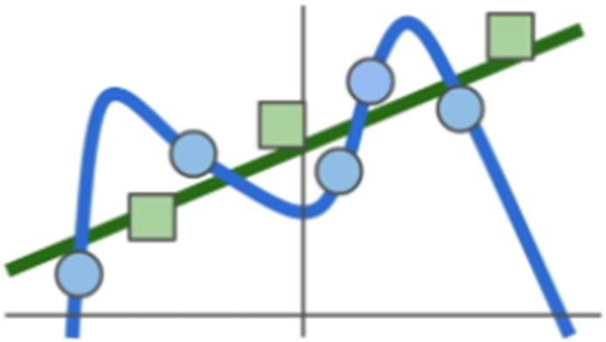


https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

How to Fix Overfitting

- How to help the generalization in the training results?
  - Large, diverse dataset
  - Regularization: adding constraints to the model during training such as smoothness, prior distribution

Data loss: Model predictions should match training data



Regularization: Model should be "simple", so it works on test data

Occam's Razor:
"Among competing hypotheses, the simplest is the best"
William of Ockham, 1285 - 1347

https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

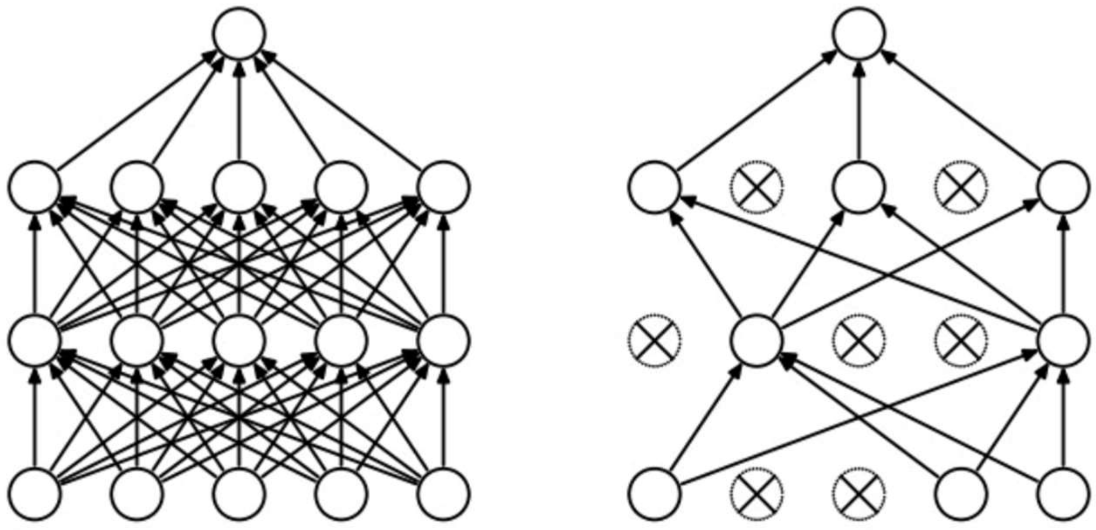
Regularization

- **To overcome the overfitting problem**
- Adding the magnitude of the penalty (P) in all parameters
- **L1 regularization**
  - For each weight, add ƛw to the objective
  - Weight vectors become sparse (very close to exactly 0)
  - Using only a sparse subset of their most important inputs
- **L2 regularization**
  - For each weight, adding 1/2 ƛw^2 to the objective
  - Heavy penalizing peaky to diffuse weight vectors (small number)
  - Not remove improper features, but rather minimize their impacts

https://cs231n.github.io/neural-networks-2/#init

What is the dropout in NN training?

- A simple approach to prevent neural networks from overfitting
- **Turning off** neurons with a predetermined probability p (e.g. 50%) while training
- Every iteration uses different sample of the models' parameters -> robust features
- Does the dropout reduce the training time? Why?
  - Increase training time
  - Wait for model's convergence



(a) Standard Neural Net (b) After applying dropout.

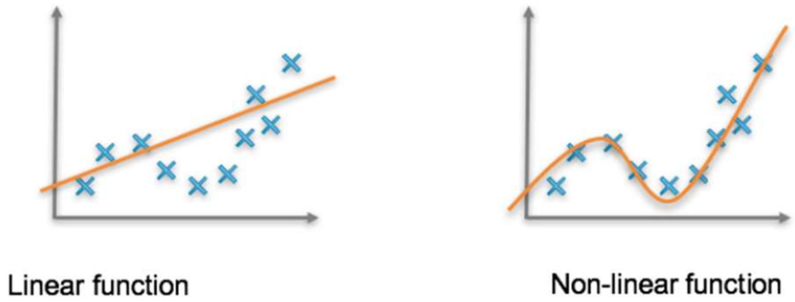
https://cs231n.github.io/neural-networks-2/#init

Learning in Deep Networks

- **When training deep networks**
  - The early layers may have a very slow learning rate (why?)
    - Vanishing gradient problem
  - **Exploding gradient problem**
    - Early layers learn much faster than later layers
- **The unstable gradient problem**
  - The learning rates of different layers tend to be wildly different

Non-linear Neuron matters?

- **Linear function**
  - A change in the first variable corresponds to a constant change in the second variable
  - Y = az, where a is a constant value
- **Problems**
  - The linear function alone doesn't capture complex patterns
  - No support backpropagation (Why?)
    - The derivative of the function is a constant
  - Collapse relations in each layer
    - The last layer is the linear function of the first one

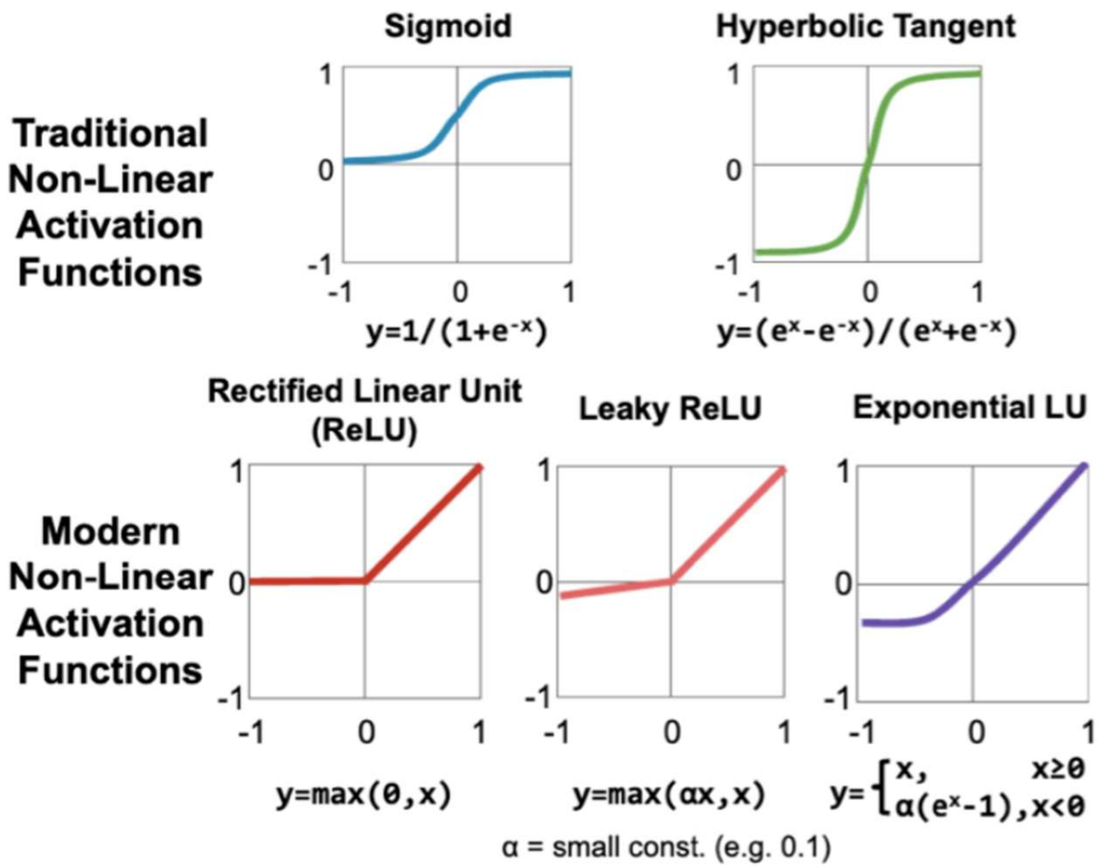


Best fit linear and non-linear models

https://srnghn.medium.com/deep-learning-overview-of-neurons-and-activation-functions-1d98286cf1e4

Non-Linear activation function in NN

- **Activation function**
  - Determine whether the neuron should be “fired” or not
  - Non-linearity -> Help for solving complex problems
- **Rectified Linear Units (ReLU)**
  - Y(x) = max {0, x}
- How to choose a proper activation function in NN models?
  - Automatic activation function selection ?



https://drive.google.com/file/d/154Y_n81-gpFy_cyHXSdeu9btOlus_o-7/view

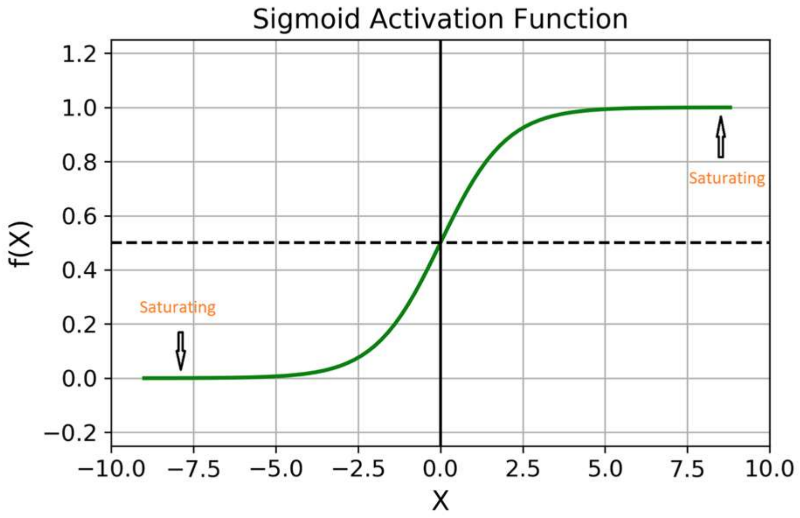
Pros and Cons of activation functions

Sigmoid	Hyperbolic Tangent	ReLU
Advantages	**Advantages**	**Advantages**
1. Smooth gradient: prevent jumps in outputs	1. Zero centered: help for inputs having diff features	1. Computationally efficient
2. Bounded outputs: between 0 and 1, normalizing outputs		2. Non-linear: its derivative function allows for backpropagation
Disadvantages	**Disadvantages**	**Disadvantages**
1. Vanishing gradient: no changes for very high or low value of inputs	1. Like Sigmoid	1. Dying ReLU: wreck the backpropagation in zero and negative inputs (gradient results = 0)

https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

Gradient vanishing problem

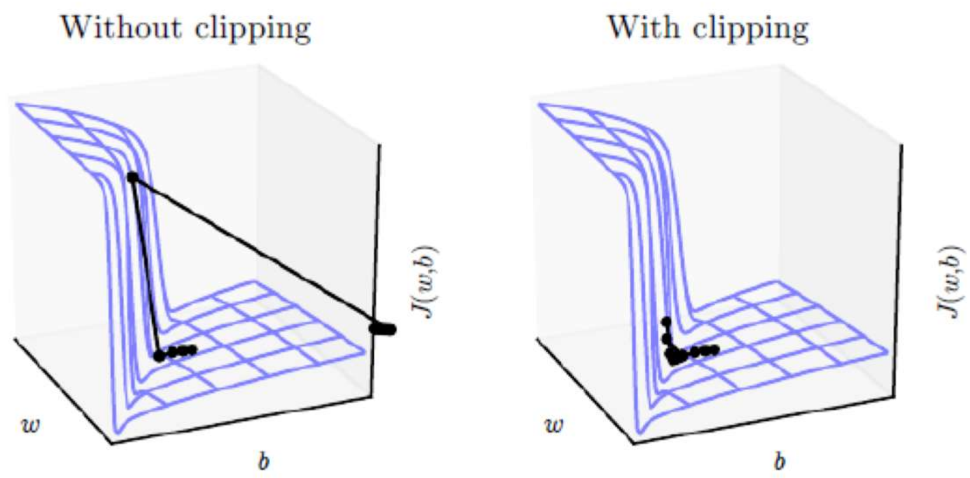
- **Gradient vanishing problem**
  - Small gradients are propagated through the last layer to the initial layer in the backpropagation.
  - These small gradients cause the weights in lower layer are not updated (Why?)
- **Gradient vanishing in Sigmoid function**
  - When the inputs are fairly large or small
  - The derivative becomes close to **0**
  - **Such small gradients -> no update in lower layer weights**
  - Problem is worse in deep network (Why?)
- **Solutions**
  - Change activation function
  - Batch normalization



https://debuggercafe.com/neural-networks-the-problem-of-vanishing-gradients/

Gradient exploding problem

- Large gradient propagates through layers in a neural network
- The model loss will be **NaN** during training
- **Solutions**
  - Gradient clipping
  - Limits the magnitude of the gradient
  - SGD without gradient clipping overshoots the landscape to minimum
  - SGD with gradient clipping descends into the minimum

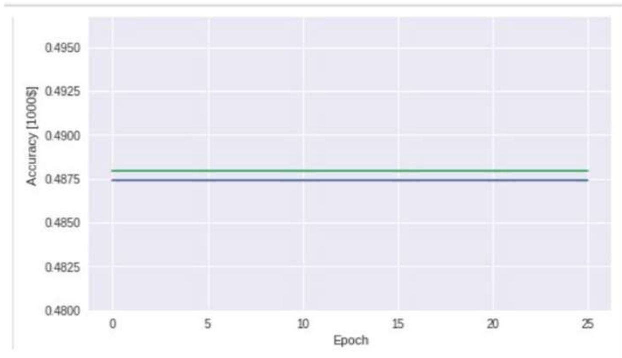


Ian Goodfellow et. al, “Deep Learning”, MIT press, 2016

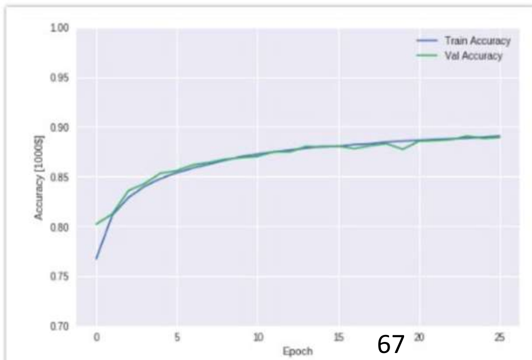
Training stability and accuracy further

- The low accuracy of the trained model without normalizing data
- What is the data normalization?
  - Data matrix [N x D] (N: the number of data, D is their dimensions)
  - Changing scales of data dimensions to the common ones
- When do we need to normalize data?
  - Only when features have different ranges
- How?

Without normalized data



With normalized data



https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029

Backpropagation in training

- Minimize the cost function by adjusting network's weights and biases
- Tuning the weights by using gradient descent
  - A weight is updated by the partial derivative of loss with respect to the weight

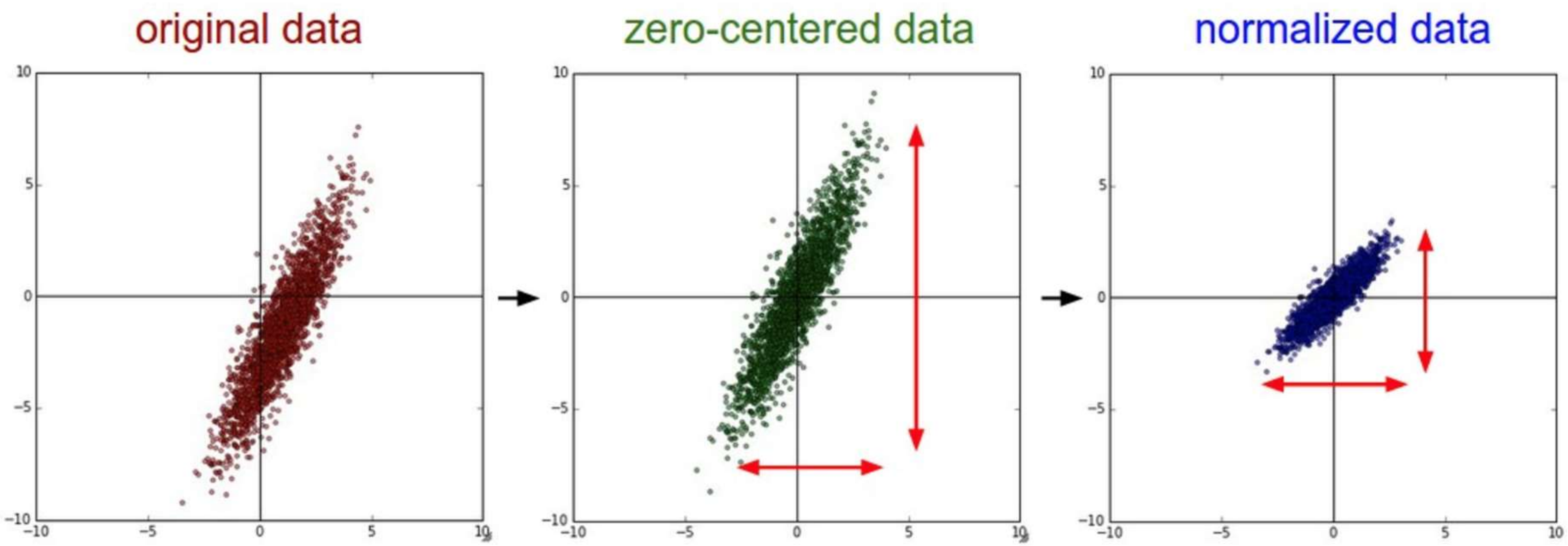
$$w_{ij}^{t+1} = w_{ij}^{t} - \alpha \frac{\partial L}{\partial w_{ij}}$$

α is the learning rate

  - The gradient indicates how the weights should change to reduce the loss
  - The process is repeated iteratively to reduce the overall loss
- What are impacts of high learning rate?
  - A high learning rate increases the step size at each iteration
  - Help speed up the training
  - Result in overshooting the minimum
  - Cause the optimization to not converge

Data normalization

- Why do we need to normalize data?
- Without distorting differences in the ranges of values
- Zero-centered: subtracting mean from each of the data point
- Normalize each dimension, the min/max along the dimension is -1 and 1



https://cs231n.github.io/neural-networks-2/#init

Batch Normalization (BN)

- Provide neural network inputs with zero mean/unit variance
- Adjusting activations in each batch (one batch includes multiple data)
- Subtract the batch mean and divide by the batch standard deviation
- Place BN in the front/back of activation?
  - BN in the front end: avoid saturation region
- Small batch size?
  - No representative mean/sigma -> bad perf.
- How to initialize gamma and beta?
  - Gamma = 1, beta = 0 (Why?)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

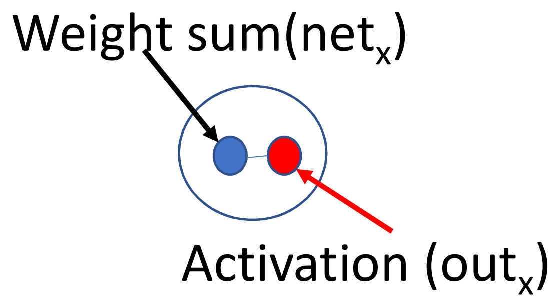
https://arxiv.org/pdf/1502.03167v3.pdf

Batch normalization - benefits

- Accelerating the training speed
- Avoid exploding/vanishing gradients
- Help sigmoid/tanh activation function (trainable network)
- Reduce the impact of initialization
- Reduce the overfitting to increase the accuracy of trained models
- Increasing the learning rate after using BN? Why?

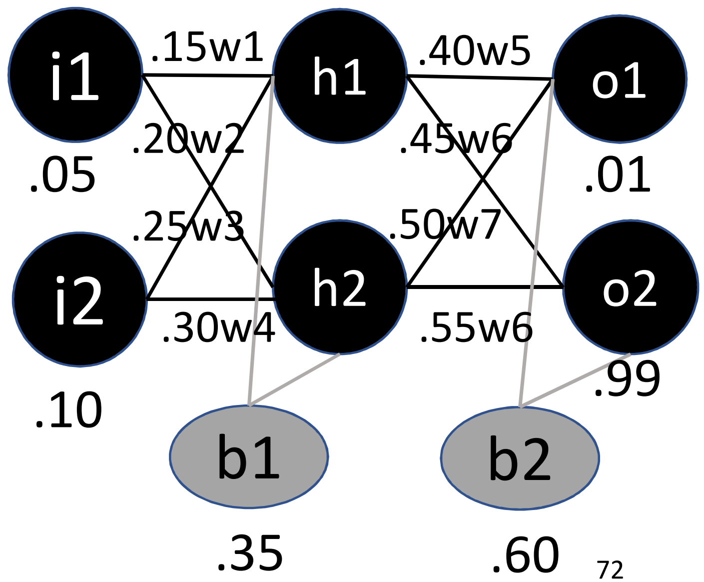
A Simple Feed-forward Neural Network

- What is the net_{h1} and out_{h1} ?
- Activation function is logistic function



$$
\begin{aligned}
net_{h1} &= w1 * i1 + w2 * i2 + b1 \\
&= 0.15 \text{ x } 0.05 + 0.2 \text{ x } 0.1 + 0.35 \\
&= 0.3775
\end{aligned}
$$

$$
\begin{aligned}
Out_{h1} &= 1/1+e\text{^}-net_{h1} \\
&= 1/1+e\text{^}(-0.3775) \\
&= 0.5933
\end{aligned}
$$



The backward Pass

- What is the value of w_5 ?

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_5}$$

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 \times \frac{1}{2}(target_{o1} - out_{o1}) \times -1 + 0$$

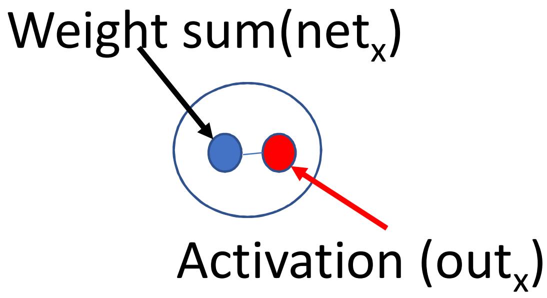
$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

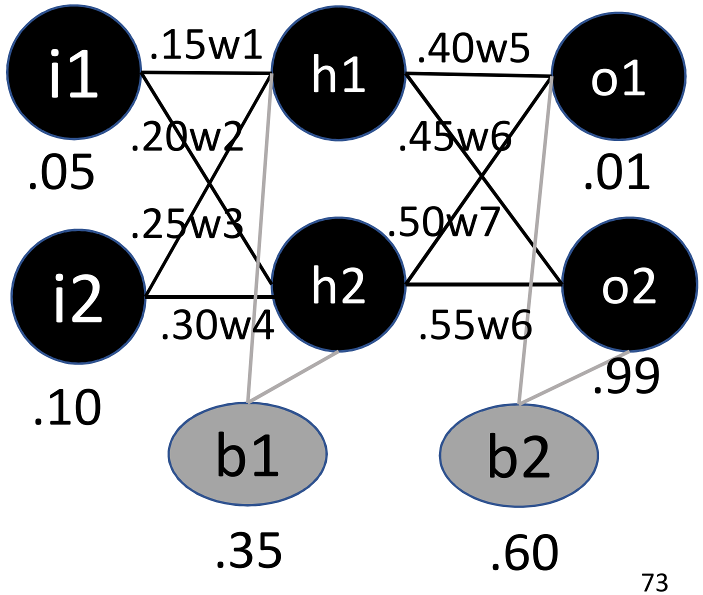
$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1})$$

$$net_{o1} = w_5 \times out_{h1} + w_6 \times out_{h2} + b_2$$

$$\frac{\partial net_{o1}}{\partial w_5} = \mathrm{out}_{h1}$$

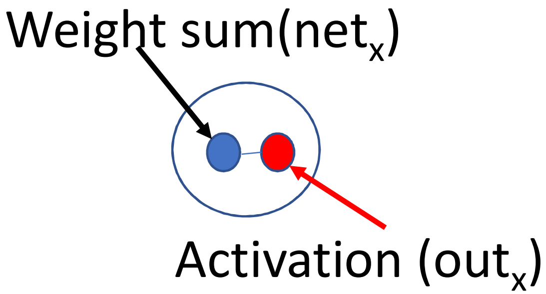
$$w_5^+ = w_5 - \alpha \times \frac{\partial E_{total}}{\partial w_5}$$



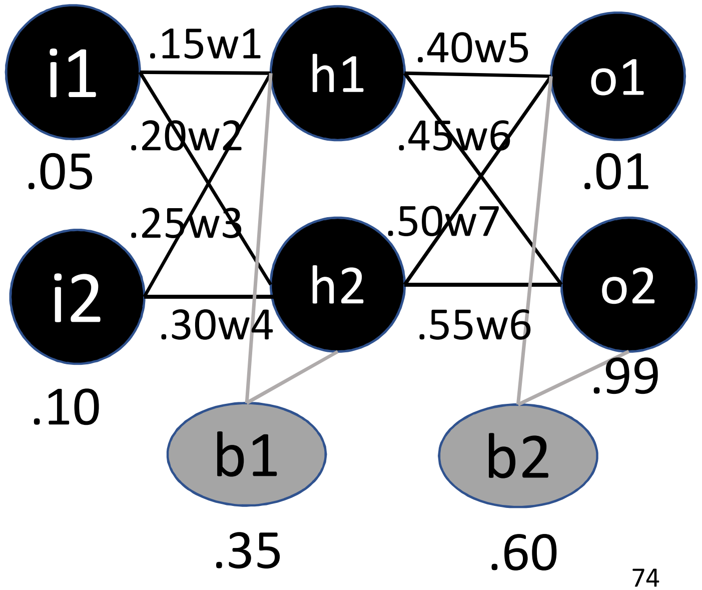


Calculating Total Error

- Using squared error function (E)
- What is E_{01}, E_{02}, E_{total} ?



$$E_{o1} = ½(target_{o1} – out_{o1}) ^2$$
$$= ½(0.01 – 0.7513)^2$$
$$= 0.2748$$
$$E_{total} = E_{01} + E_{02}$$



Takeaway Questions

- What can impact the learning rates ?
  - (A) The selection of datasets
  - (B) Activation function
  - (C) The size of inputs
- What are potential solutions to avoid the gradient vanishing problem ?
  - (A) Changing activation function
  - (B) Using low learning rate
  - (C) Batch normalization