LAPENNA Program
LECTURE 3

Kwai Wong, Stan Tomov
Julian Halloy, Stephen Qiu, Eric Zhao

University of Tennessee, Knoxville

February 17, 2022

Acknowledgements:

- Support from NSF, UTK, JICS, ICL, NICS
- LAPENNA, www.jics.utk.edu/lapenna, NSF award #202409
- www.icl.utk.edu, cfdlab.utk.edu, www.xsede.org, www.jics.utk.edu/recsem-reu,
- MagmaDNN is a project grown out from the RECSEM REU Summer program supported under NSF award #1659502
- Source code: www.bitbucket.org/icl/magmadnn
- www.bitbucket.org/cfdl/opendnnwheel

INNOVATIVE COMPUTING LABORATORY

Overview of LAPENNA

The major goal of The Linear Algebra Preparation for Emergent Neural Network Architectures (LAPENNA) program is to provide essential knowledge to advance literacy in AI to sustain the growth and development of the workforce in the cyberinfrastructure (CI) ecosystem. This program aims to prepare college faculty, researchers, and industrial practitioners to design, enable and direct their own course curricula, collaborative projects, and training programs for in-house data-driven sciences programs. The LAPENNA program focuses on delivering computational techniques, numerical algorithms and libraries, and implementation of AI software on emergent CPU and GPU platforms.

Ecosystem Linear Algebra Preparation for Emergent Neural Network Architectures (LAPENNA)

Modeling, Numerical Linear Algebra, Data Analytics, Machine Learning, DNN, GPU, HPC

Session 1	Session 2	Session 3	Session 4
7/2020 - 12/2020	1/2021- 6/2021	7/2021 - 1/2022	1/2022 - 6/2022
16 participants	16 participants	16 participants	16 participants
Faculty/Students	Faculty/Students	Faculty/Students	Faculty/Students
10 webinars	10 webinars	10 webinars	10 webinars
4 Q & A	4 Q & A	4 Q & A	4 Q & A

Colleges courses, continuous integration, online courses, projects, software support

Web-based resources, tutorials, webinars, training, outreach

Teams and Participants of LAPENNA

- ✓ **PIs : Kwai Wong (JICS), Stan Tomov (ICL), University of Tennessee, Knoxville**
  - **Stephen Qiu, Julian Halloy, Eric Zhao (Students)**

- ✓ **Team : Clemson University**
- ✓ **Teams : University of Arkansas**
- ✓ **Team : University of Houston, Clear Lake**
- ✓ **Team : Miami University, Ohio**
- ✓ **Team : Boston University**
- ✓ **Team : West Virginia University**
- ✓ **Team : Louisiana State University, Alexandria**
- ✓ **Teams : Jackson Laboratory**
- ✓ **Team : Georgia State University**
- ✓ **Teams : University of Tennessee, Knoxville**
- ✓ **Teams : Morehouse College, Atlanta**
- ✓ **Team : North Carolina A & T University**
- ✓ **Team : Clark Atlanta University, Atlanta**
- ✓ **Team : Alabama A & M University**
- ✓ **Team : Slippery Rock University**
- ✓ **Team : University of Maryland, Baltimore County**

Meeting Time

- ✓ **Webinar Meeting time. Thursday 8:00 – 10:00 pm ET,**
- ✓ **Tentative schedule, www.jics.utk.edu/lapenna --> Spring 2022**

Topic: LAPENNA Spring 2022 Webinar
Time: Feb 3, 2022 08:00 PM Eastern Time (US and Canada)
Every week on Thu, 12 occurrence(s)
Feb 3, 2022 07:30 PM
Feb 10, 2022 07:30 PM
Feb 17, 2022 07:30 PM
Feb 24, 2022 07:30 PM
Mar 3, 2022 07:30 PM
Mar 10, 2022 07:30 PM
Mar 17, 2022 07:30 PM
Mar 24, 2022 07:30 PM
Mar 31, 2022 07:30 PM
Apr 7, 2022 07:30 PM
Apr 14, 2022 07:30 PM
Apr 21, 2022 07:30 PM

Join from PC, Mac, Linux, iOS or Android: https://tennessee.zoom.us/j/94140469394
Password: 708069

Schedule of LAPENNA Spring 2022
Thursday 8:00pm -10:00pm Eastern Time

The goal of The Linear Algebra Preparation for Emergent Neural Network Architectures (LAPENNA) program is to provide essential knowledge to advance literacy in AI to sustain the growth and development of the workforce in the cyberinfrastructure (CI) ecosystem for data-driven applications.

Month	Week	Date	Topics
February	Week 01	3	Logistics, High Performance Computing
	Week 02	10	Computational Ecosystem, Linear Algebra
	Week 03	17	Introduction to DNN, Forward Path (MLP)
	Week 04	24	Backward Path (MLP), Math, Example
March	Week 05	3	CNN Computation
	Week 06	10	CNN Backpropagation, Example
	Week 07	17	CNN Network, Linear Algebra
	Week 08	24	Segmentation, Unet,
	Week 09	31	Object Detection, RC vehicle
April	Week 10	7	RNN, LSTEM, Transformers
	Week 11	14	DNN Computing on GPU
	Week 12	21	Overview, Closing
June or July	Workshop	To be arranged	4 Days In Person at UTK

Topics of Webinar Three

- ✓ **Unit 2 : Computational Ecosystem**
- ✓ **Big Science and Big Data Computational Ecosystem**
  - ➢ **Hardware System, Hardware, Performance**
  - ➢ **Computational Models - Mathematics, Solvers, Numerical Linear Algebra**
  - ➢ **Software Implementation - Computer Sciences, Implementations**
  - ➢ **Data Computing – Machin Learning Models, Frameworks**

- ✓ **Unit 2 : Numerical Linear Algebra**
  - ➢ **Linear Algebra Operations**
  - ➢ **Basic Linear Algebra Subprogram (BLAS)**
  - ➢ **Tensorflow 2.0, Tensor operations**

✓ **Unit 3 : Neural Network Computation**

Modeling Cycle and Emergent Technology

Big Data → Deep Learning → Computing capacity
Combining the Learned the pattern and behavior hidden in sensor or computed data to predict complicated phenomena

A growing field with evolving technology
Statistical Machine Learning →
Deep Neural Network → Numerical Linear Algebra Supercomputing (HPC) → Real Time Applications

Data- + model-based simulation --> Big Iron (supercomputer)

Fast enough to get approximate results (time scale)
Enough memory to process the huge amount of data (length scale)

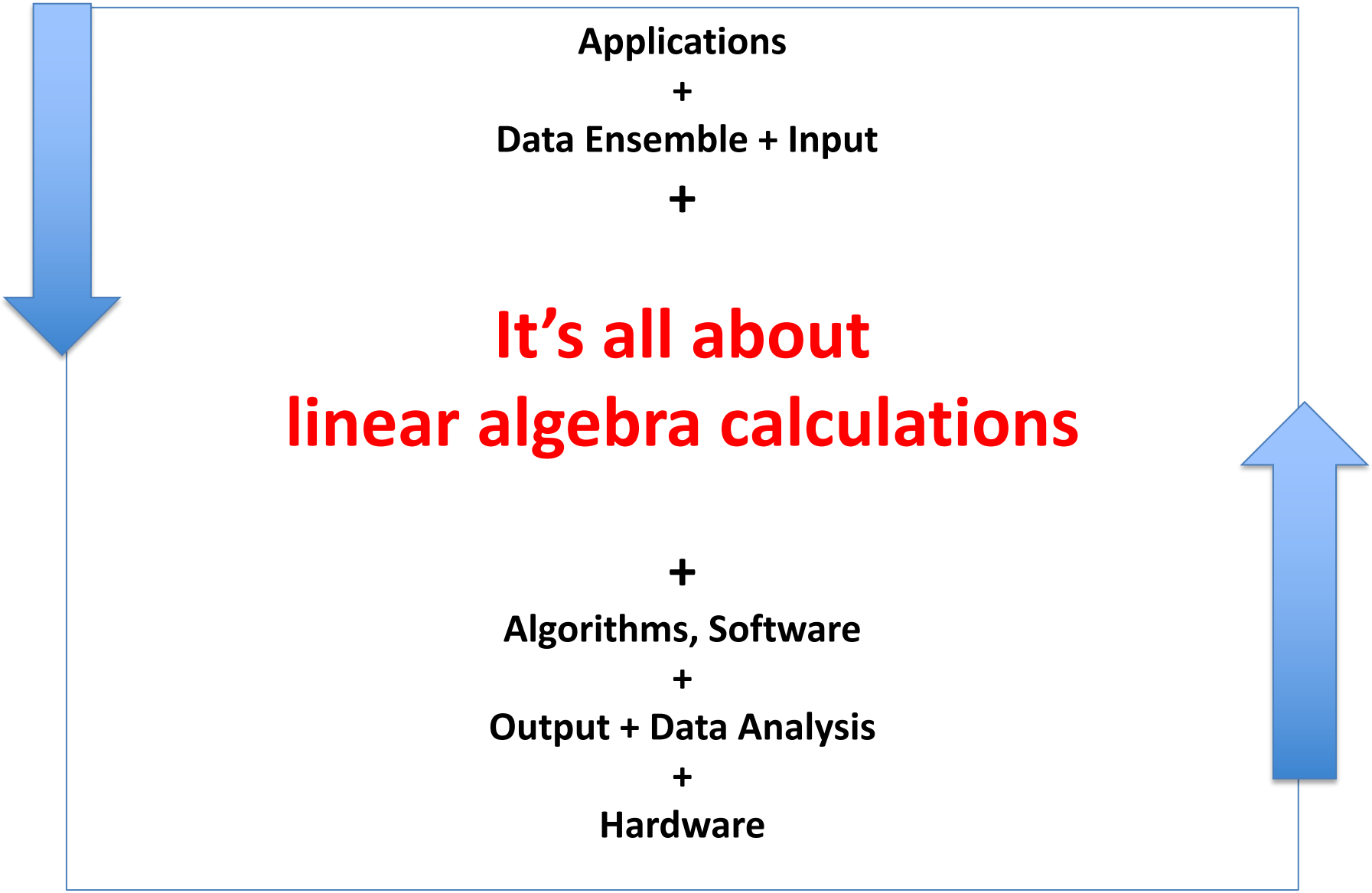
Computational Ecosystem

Computational Ecosystem

- **Advance Computing system**
  - Top500, FLOPS and Performance
  - DOE Exascale Road Map
  - NSF infrastructure
  - Desktop, Accelerators, GPUs
  - Google COLAB
- **Predictive Simulation Science**
  - Equation based simulation sciences
  - Mathematics Essentials, Calculus
  - Linear Algebra, BLAS

- **Computer Sciences and Software tools**
  - Linux OS
  - Computational thinking
  - Workflow
  - Languages
- **Data Intensive Sciences**
  - Statistical Learning
  - Data mining
  - Deep Learning
  - Framework

DNN Model

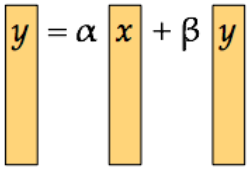

Applications
+
Data Ensemble + Input
+
It's all about
linear algebra calculations
+
Algorithms, Software
+
Output + Data Analysis
+
Hardware

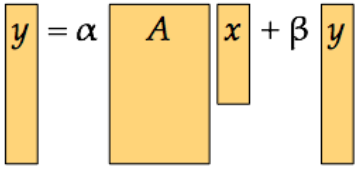
Basic Linear Algebra Subprograms (BLAS)

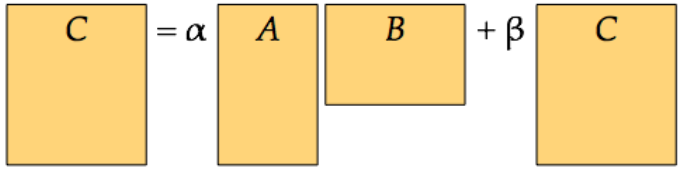
- **BLAS is a library of standardized basic linear algebra computational kernels created to perform efficiently on serial computers taking into account the memory hierarchy of modern processors.**
- **BLAS1 does vectors-vectors operations.**
  - **Saxpy = y(i) = a* x(i) + y(i), ddot= $\sum$x(i) *y(i)**
- **BLAS2 does matrices - vectors operations.**
  - **MV : y = A x + b (dgemv)**
- **BLAS3 operates on pairs or triples of matrices.**
  - **MM : C = αAB + βC, Triangular Solve : X = $\alpha T^{-1}X$**
- **Level3 BLAS is created to take full advantage of the fast cache memory. Matrix computations are arranged to operate in block fashion. Data residing in cache are reused by small blocks of matrices. dgemm**
- **Atlas, openBLAS, MKL, ESSL, libsci, ACML, cuBLAS, BLIS**

Computational Intensity = FLOPS/ Memory Access

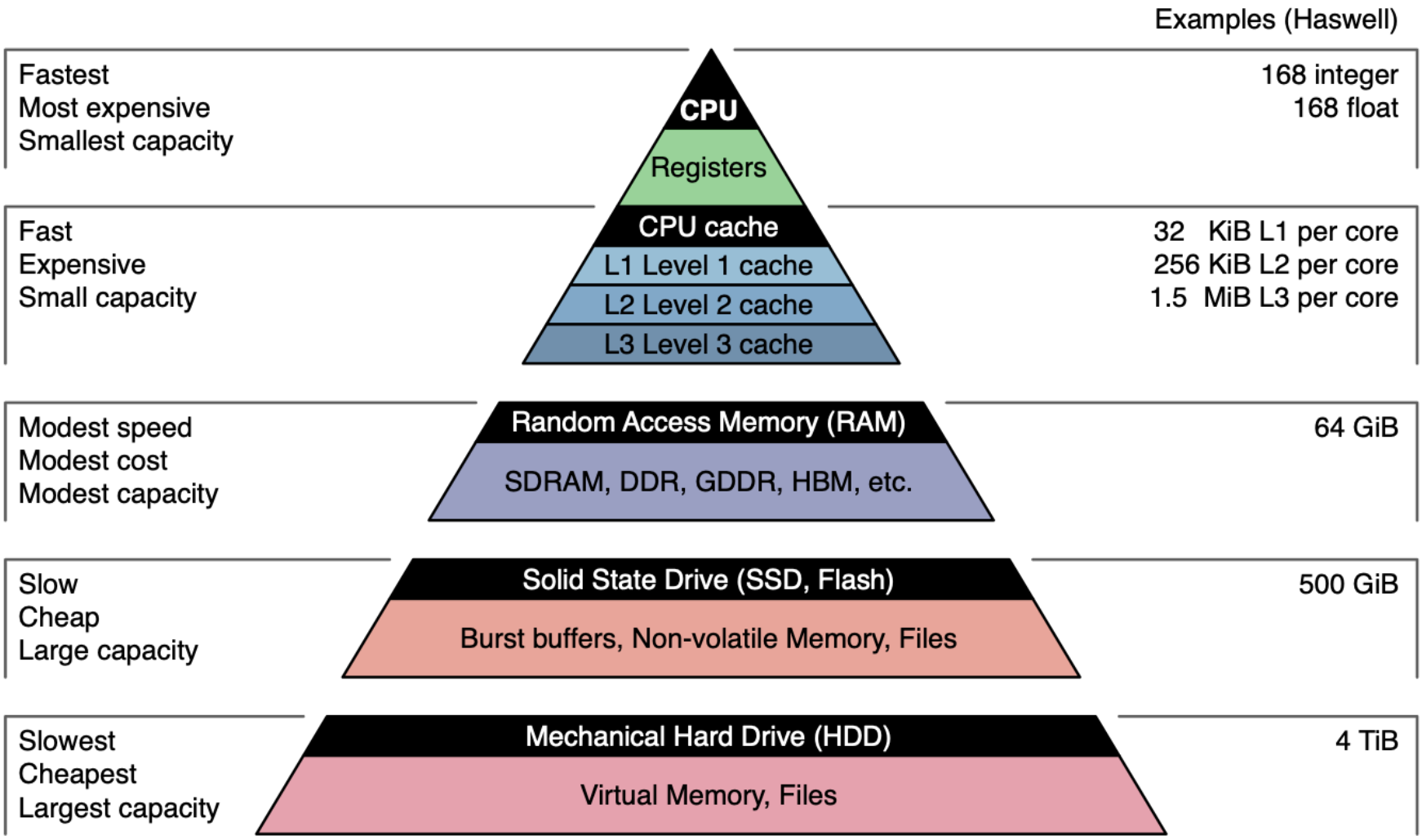
- ✓ Level 1 BLAS — vector operations
  - ✓ $O(n)$ data and flops (floating point operations)
  - ✓ Memory bound:
    $O(1)$ flops per memory access
- ✓ Level 2 BLAS — matrix-vector operations
  - ✓ $O(n^2)$ data and flops
  - ✓ Memory bound:
    $O(1)$ flops per memory access
- ✓ Level 3 BLAS — matrix-matrix operations
  - ✓ $O(n^2)$ data, $O(n^3)$ flops
  - ✓ Surface-to-volume effect
  - ✓ Compute bound:
    $O(n)$ flops per memory access







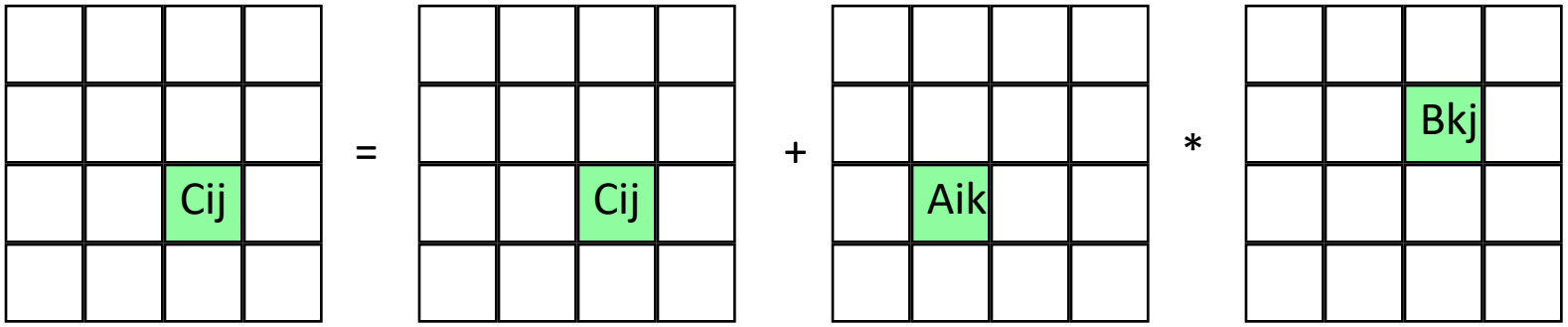
Memory hierarchy



Adapted from illustration by Ryan Leng

Block MM

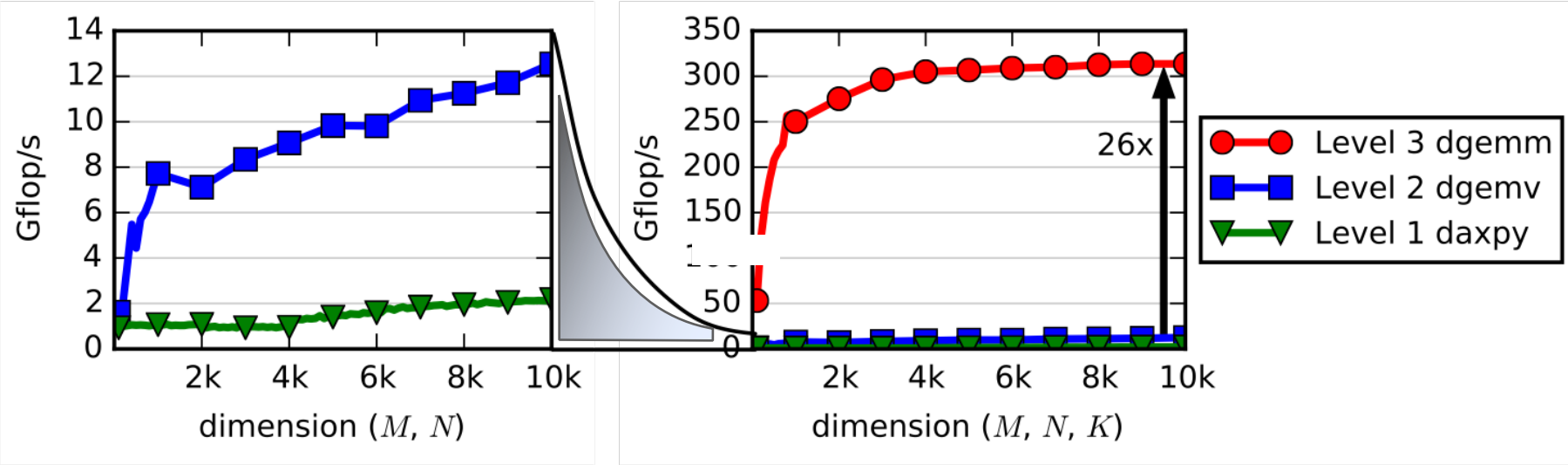
- q = f/m = (2*n^3) / ((2*N + 2) * n^2) ~ n / N
- If N is equal to 1, the algorithm is ideal. However, N is bounded by the amount of fast cache memory. However, N can be taken independently to the size of matrix, n.
- The optimal value of N = sqrt (size of fast memory / 3)



$$C_{ij} = C_{ij} + \sum_{k=1}^{n} A_{ik} * B_{kj}$$

BLAS: Basic Linear Algebra Subroutines

- Intel Sandy Bridge (2 socket E5-2670)
  - Peak 333 Gflop/s = 2.6 GHz * 16 cores * 8 double-precision flops/cycle †
  - Max memory bandwidth 51 GB/s ‡

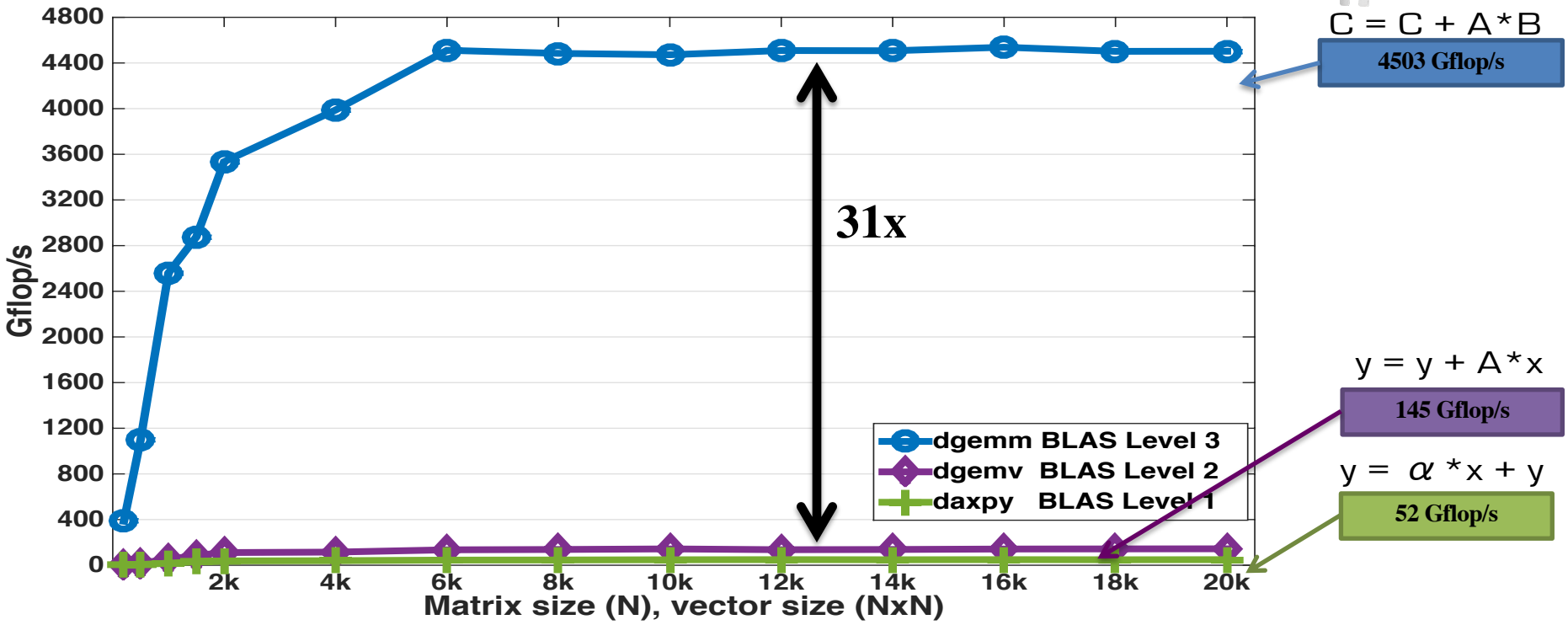


† http://stackoverflow.com/questions/15655835/flops-per-cycle-for-sandy-bridge-and-haswell-sse2-avx-avx2

‡ http://ark.intel.com/products/64595/Intel-Xeon-Processor-E5-2670-20M-Cache-2_60-GHz-8_00-GTs-Intel-QPI

Level 1, 2 and 3 BLAS

Nvidia **P100**, 1.19 GHz, Peak DP = 4700 Gflop/s



C = C + A*B

4503 Gflop/s

y = y + A*x

145 Gflop/s

y = α *x + y

52 Gflop/s

Nvidia P100
The theoretical peak double precision is 4700 Gflop/s
CUDA version 8.0

1. Write a python code to compute C=C+A x B and plot a curve of the FLOPS against the matrix size N when the computation is done on a CPU. Do the same on the GPU.
2. 2 * (N^3) FLOP / time = ? FLOPS, 2 * 5 *5 * 5 /7.4 = 33.8 GFLOPS
3. Run the same problem again using single precision.
4. Repeat question #5 using R

```
[18] import numpy as np

     A = np.random.rand(5000, 5000).astype('float64')
     B = np.random.rand(5000, 5000).astype('float64')

     %timeit np.dot(A, B)
```

```
1 loop, best of 3: 7.39 s per loop
```

```
%%R
library(dplyr)
A<-matrix(runif(25000000),nrow=5000)
B<-matrix(runif(25000000),nrow=5000)
system.time(C<-A%*%B)
```

```
   user  system elapsed
 15.449   0.025   7.840
```

LAB 2
Problem 4

Write a python code to compute C=AXB and plot a curve of the FLOPS against the matrix size N (take N=2000, 4000, 6000) using single precision when the computation is done on a CPU. DO the same on the GPU

```
import numpy as np
import time

A = np.random.rand(2000, 2000).astype('float32')
B = np.random.rand(2000, 2000).astype('float32')
%timeit np.dot(A,B)
timestt = time.time()
C=np.dot(A,B)
timeend = time.time()
tmm2000 = timeend - timestt
gf2000 = 2*2*2*2/tmm2000
print('time = ',tmm2000)
print('GFLOPS = ', gf2000)

A = np.random.rand(4000, 4000).astype('float32')
B = np.random.rand(4000, 4000).astype('float32')
timestt = time.time()
C=np.dot(A,B)
timeend = time.time()
tmm4000 = timeend - timestt
gf4000 = 2*4*4*4/tmm4000
print('time = ',tmm4000)
print('GFLOPS = ', gf4000)
```

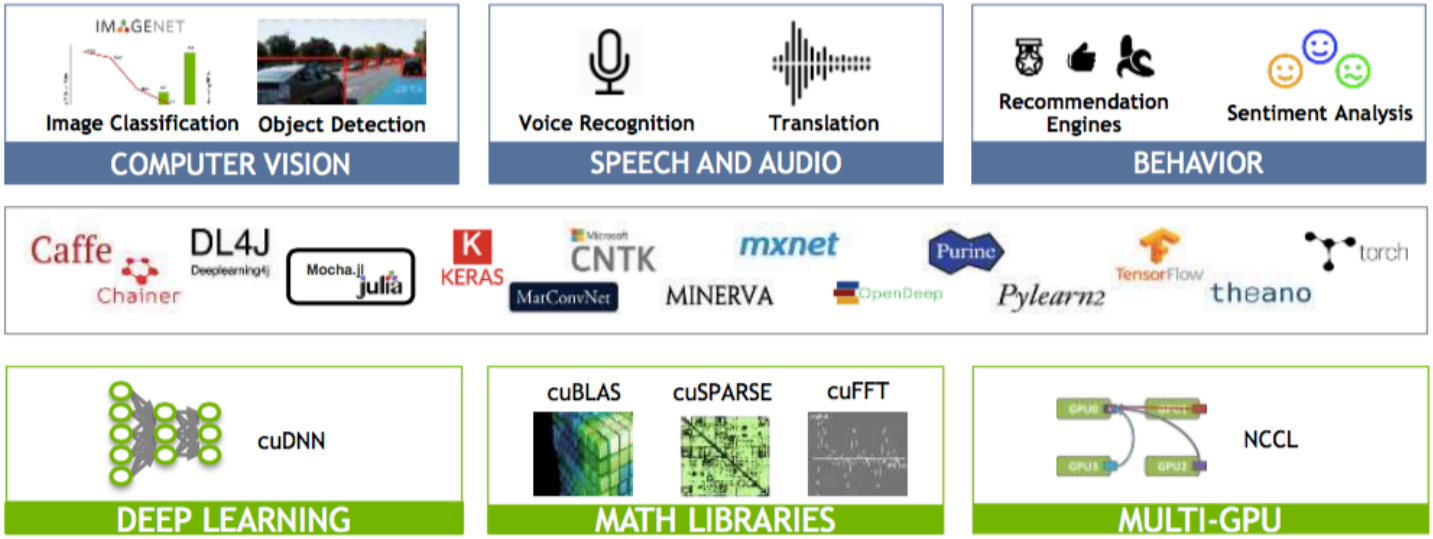
```
A = np.random.rand(6000, 6000).astype('float32')
B = np.random.rand(6000, 6000).astype('float32')
timestt = time.time()
C=np.dot(A,B)
timeend = time.time()
tmm6000 = timeend - timestt
gf6000 = 2*6*6*6/tmm6000
print('time = ',tmm6000)
print('GFLOPS = ', gf6000)

import torch
A = torch.randn(6000, 6000).cuda()
B = torch.randn(6000, 6000).cuda()
timestt = time.time()
C=torch.matmul(A,B)
timeend = time.time()
tmm6000 = timeend - timestt
gf6000 = 2*6*6*6/tmm6000
print('time = ',tmm6000)
print('GFLOPS = ', gf6000)
```

```
1 loop, best of 5: 228 ms per loop
time =  0.2360849380493164
GFLOPS =  67.7722184744277
time =  1.8578176498413086
GFLOPS =  68.8980428251037
time =  6.155170440673828
GFLOPS =  70.18489644824643
```

```
1 loop, best of 5: 214 ms per loop
time =  0.22870469093322754
GFLOPS =  69.95921218192831
time =  1.8288803100585938
GFLOPS =  69.98817762759944
time =  6.09820103645324 7
GFLOPS =  70.84056386754575
```

```
time =  6.029452800750732
GFLOPS =  71.6482928510878
time =  0.06104087829589844
GFLOPS =  7077.224510202169
```

Topics of Unit Three

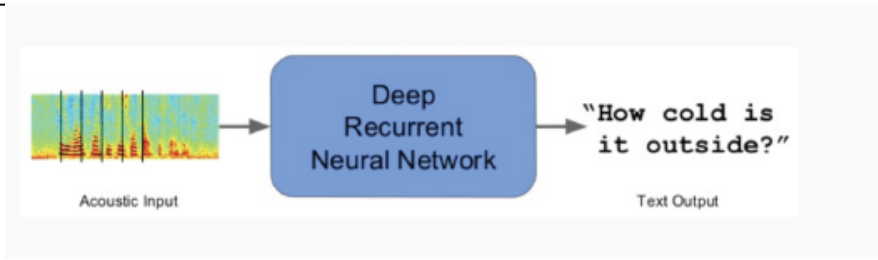


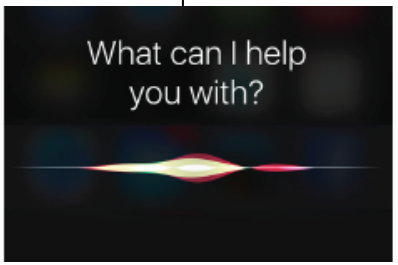
- Overview
- MNIST Test case , TensorFlow example
- Neural Network Computation, Forward path
- computational graph
- Fundamental Mathematical Theory, Loss function
- Example of Forward path calculation,
- Complete the cycle
- Tensorflow Tensor recap
- Backpropagation next

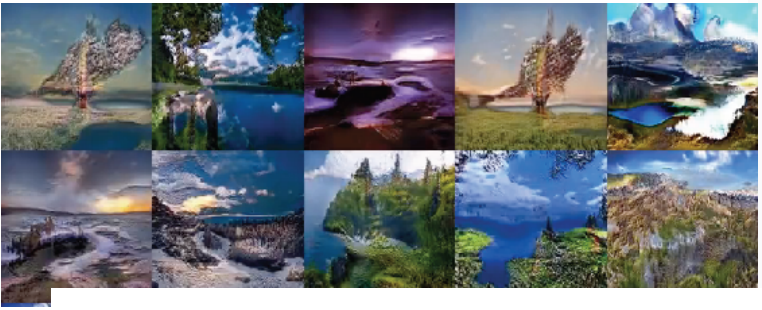
DNN is Ubiquitous

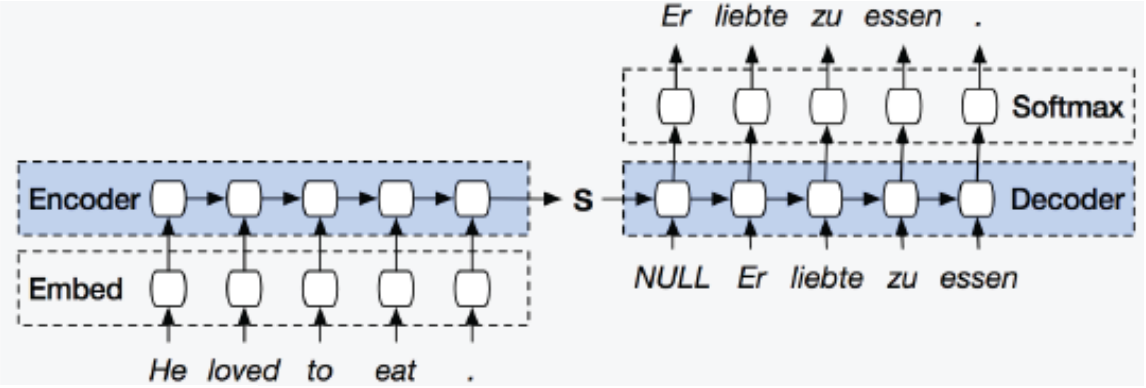
DNN: now providing breakthrough results in many applications

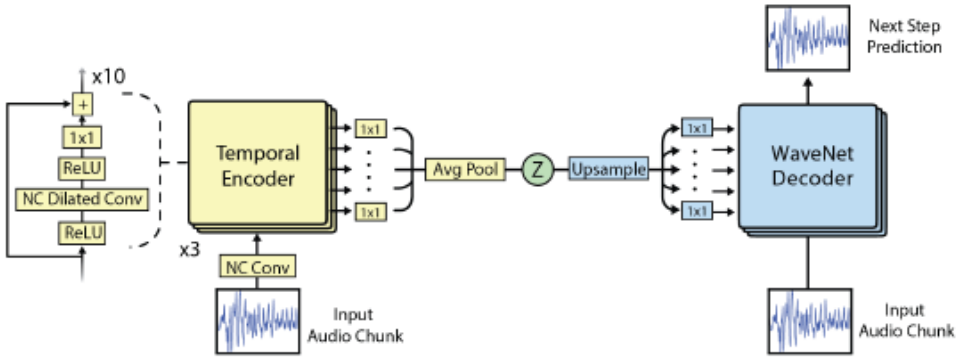
- Speech recognition
- Machine translation
- Image analysis
- Audio synthesis
- Question answering

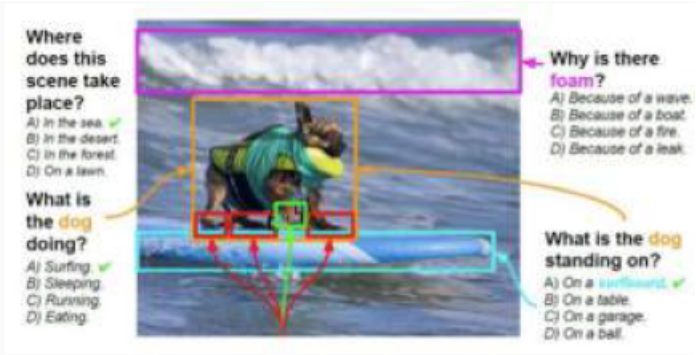












https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491

M. Telgarsky: the benefit of depth in neural networks, 2016

Historical Cycle of Artificial Intelligence (AI)

AI is based on the assumption that the process of human thought can be modelled and formulated.

The golden years 1956–1974: foundation, logic, model, formulation

- ✓ 1956, The Dartmouth Summer Research Project on Artificial Intelligence, organized by John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon
- ✓ **Funding from DARPA and other government agencies**
- ✓ The rise of symbolic methods and logical approaches, work on natural languages, preceptron
- ✓ 1973, the Lighthill report by James Lighthill, "Artificial Intelligence: A General Survey" automata, robotics, neural network

The first AI winter 1974–1980 : Limited successes, computing power, approaches, funding

Boom 1980–1987: Expert Systems – knowledge learning

- ✓ Neural networks, perceptron again, backpropagation by Geoffrey Hinton and David Rumelhart
- ✓ VSLI, LISP machine, prolog, parallel computing,
- ✓ Funding from Japanese government and Strategic Computing Initiatives from DARPA

Bust: the second AI winter 1987–1993 : Economic bubble!! Desktops for knowledge model

AI 1993–2011:

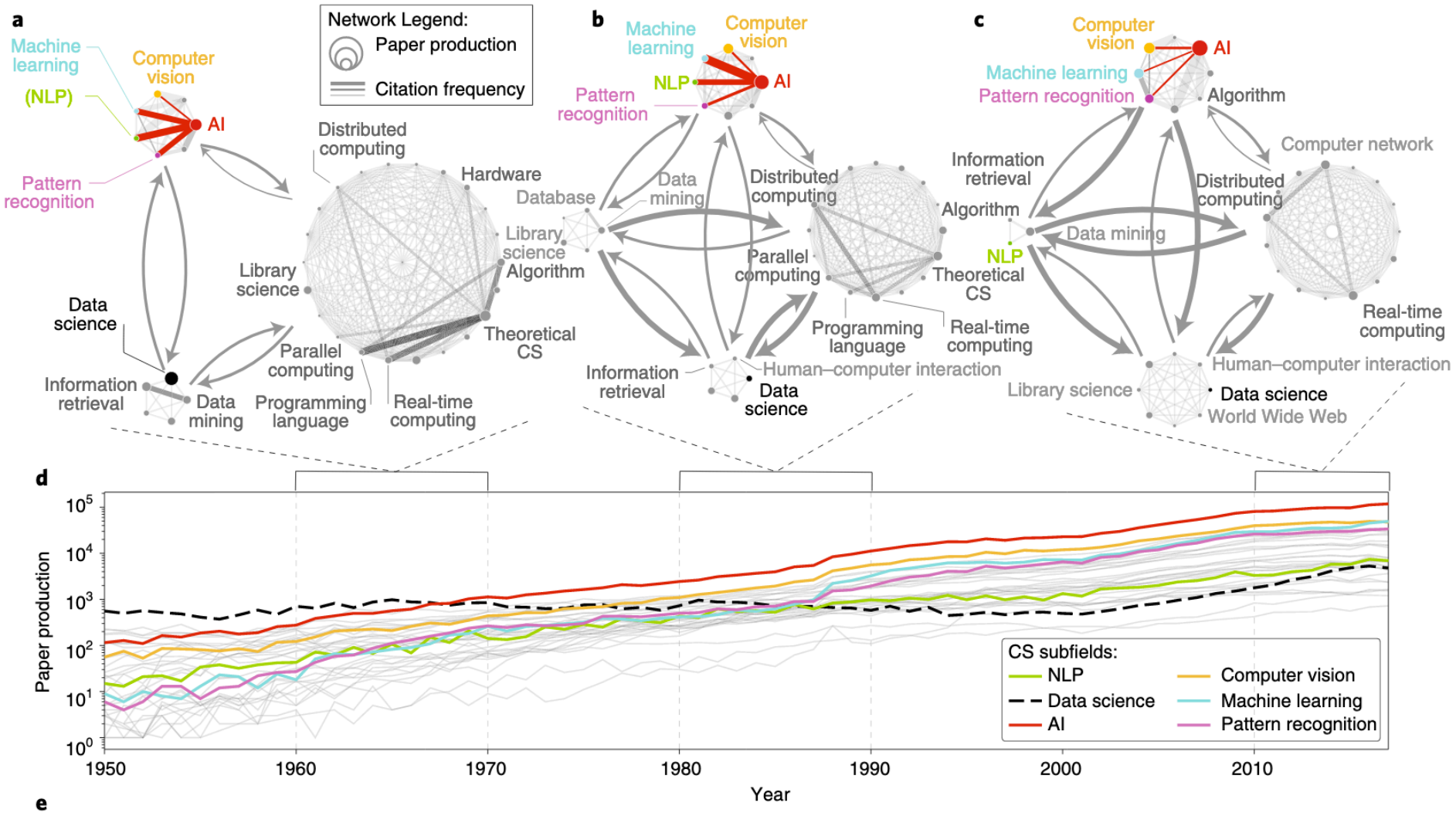
- ✓ Private industries, internet, Google, NLP
- ✓ Netflix challenge ($US 1 million), DARPA Grand Challenge (autonomous vehicle)
- ✓ 2006, Hinton (U. of Toronto), Bingio (U. of Montreal), LeCun (NYU) : Deep Neural Network (DNN)

Deep learning, big data and artificial general intelligence: 2011–present :

- ✓ 2012, ImageNet by Fei-Fei Li (2010-2017) and AlexNet
- ✓ Industries, software startups, phone systems, sensors, big data, initiatives
- ✓ Supercomputers, GPUs, Math, Algorithm, research, social sciences to applications (transformative)
- ✓ DeepMind, AlphaGo, Compute- and data drive modelling, GAN

https://en.wikipedia.org/wiki/Dartmouth_workshop, https://en.wikipedia.org/wiki/Lighthill_report
https://en.wikipedia.org/wiki/History_of_artificial_intelligence

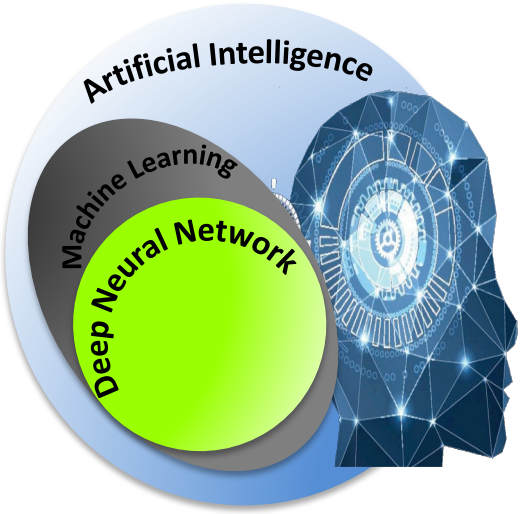
The evolution of citation graphs in artificial intelligence researchhttps://web.media.mit.edu/~mrfrank/papers/nmi2019.pdf

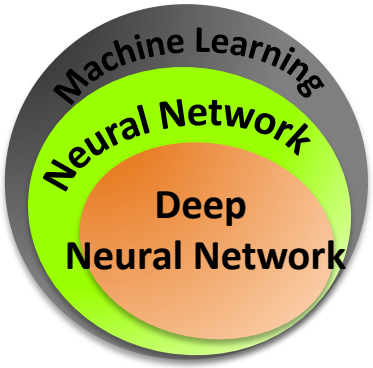


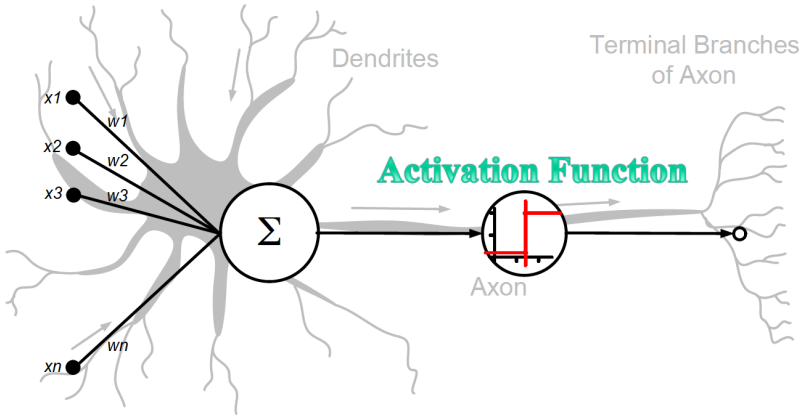
Citation patterns among CS subfields identify areas of AI-related research. We examine the rate of citations between CS subfields based on journal and conference publications from three different decades: the 1960s (a), the 1980s (b) and the 2010s through 2017 (c)

https://analyticsindiamag.com/top-machine-learning-research-papers-released-in-2020/

AI, ML, NN, DNN

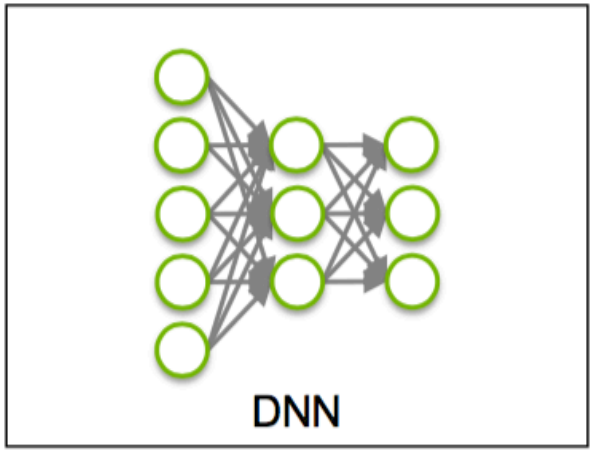


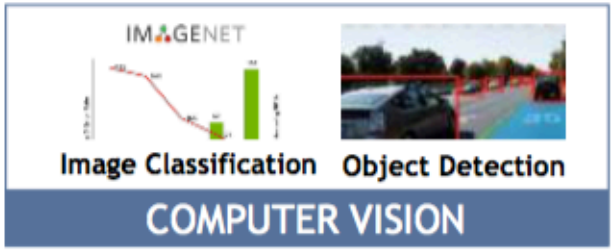


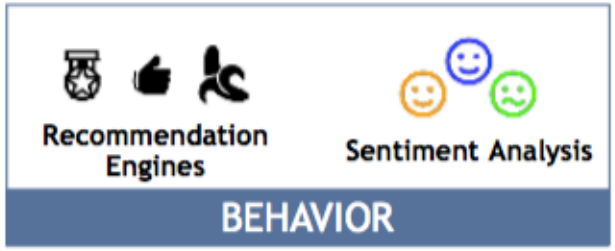


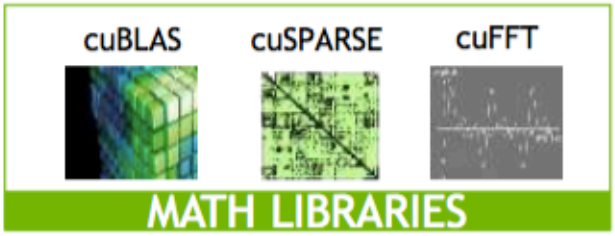
- ✓ **Artificial Intelligence (AI)**: science and engineering of making intelligent machines to perform the human tasks (John McCarthy,1956). AI applications is ubiquitous.
- ✓ **Machine learning (ML) :** A field of study that gives computers the ability to learn without being explicitly programmed (Arthur Samuel, 1959). A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E (Tom Mitchell,1998).
- ✓ **Neural Network (NN) :** Neural Network modeling, a subfield of **ML** is algorithm inspired by structure and functions of biological neural nets
- ✓ **Deep Neural Network (DNN) :** (aka deep learning): an extension of **NN** composed of many layers of functional neurons, is dominating the science of modern **AI** applications
- ✓ **Supervised Learning (SL)** : A class in ML, dataset has labeled values, use to predict output values associated with new input values.

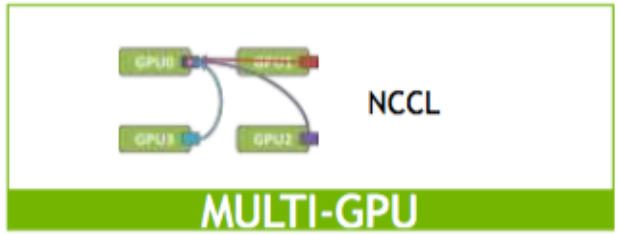
Machine Learning – GPU acceleration











Source captured frin - Julie Bernauer – HPC Advisory Council Stanford Tutorial – 2017/02/07

Supervised Learning

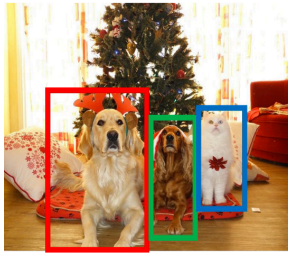
Supervised Learning Data:
(x, y) x is data, y is label

Goal:
Learn a function to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

CAT

Classification

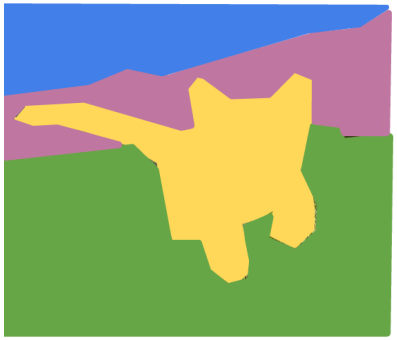


DOG, DOG, CAT

Object Detection

A cat sitting on a suitcase on the floor

Image captioning



GRASS, CAT, TREE, SKY

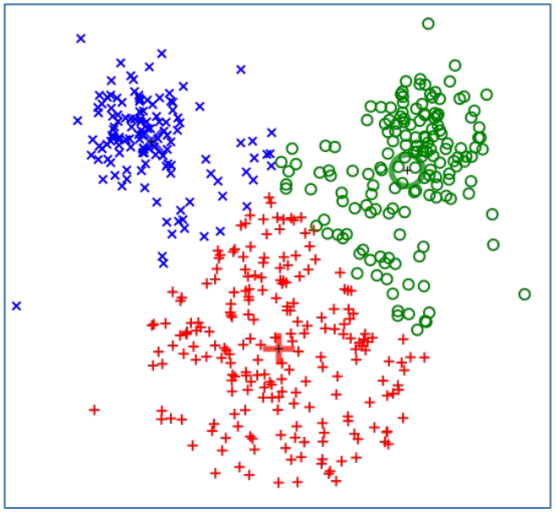
Semantic Segmentation

Unsupervised Learning

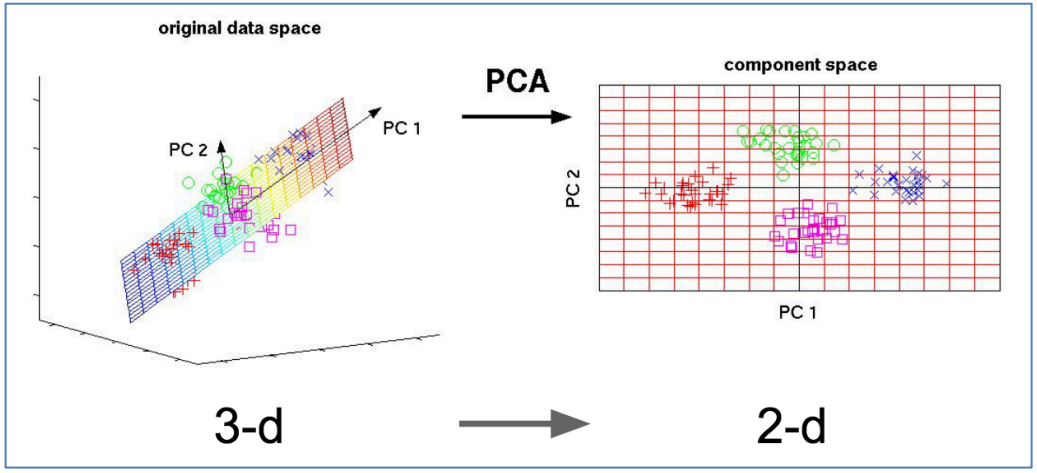
Unsupervised Learning Data:
x Just data, no labels!

Goal:
Learn some underlying hidden structure of the data

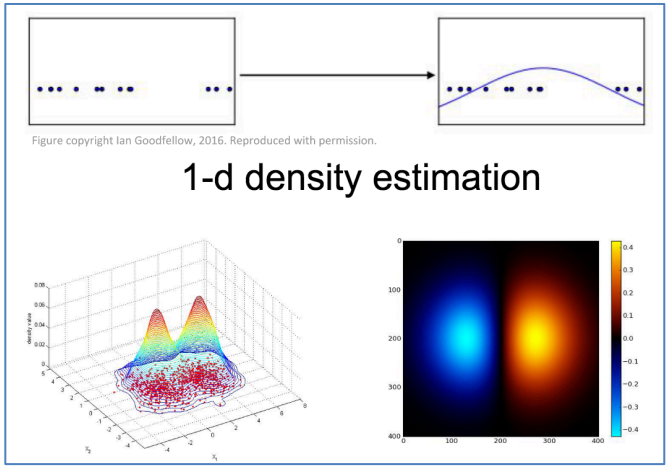
Examples: Clustering, dimensionality reduction, density estimation, etc.



K-means clustering

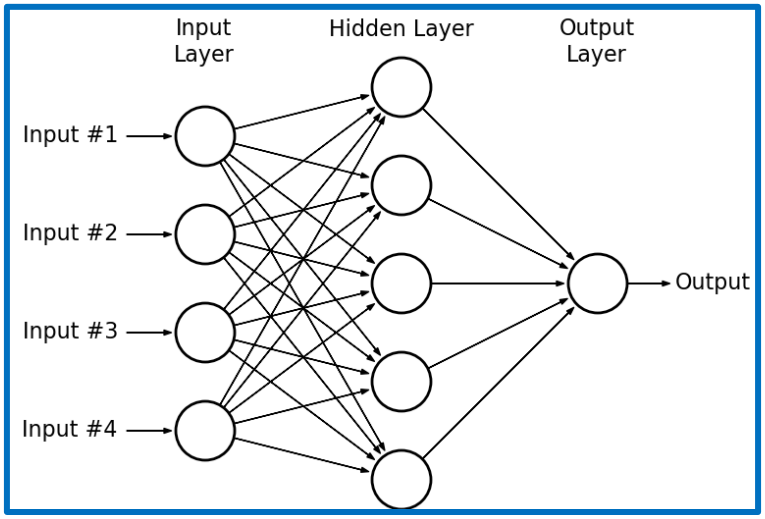


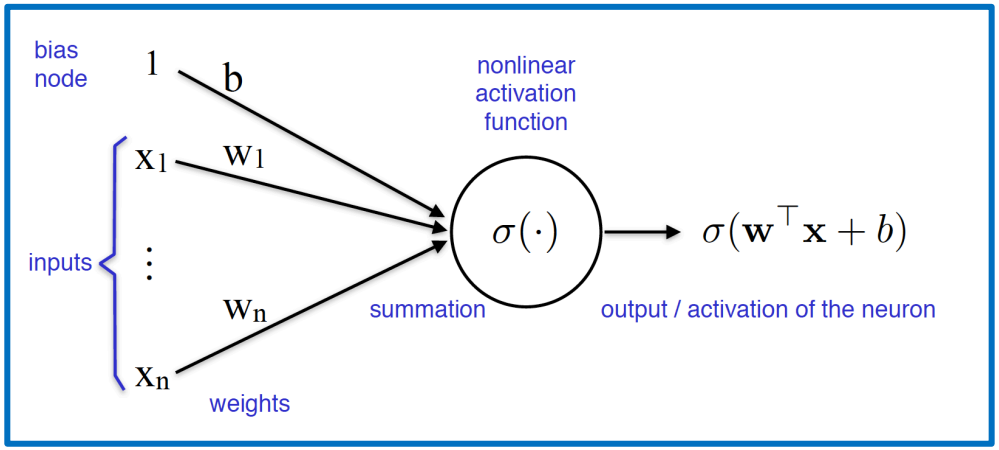
Principal Component Analysis
(Dimensionality reduction)



2-d density estimation

NN Modeling



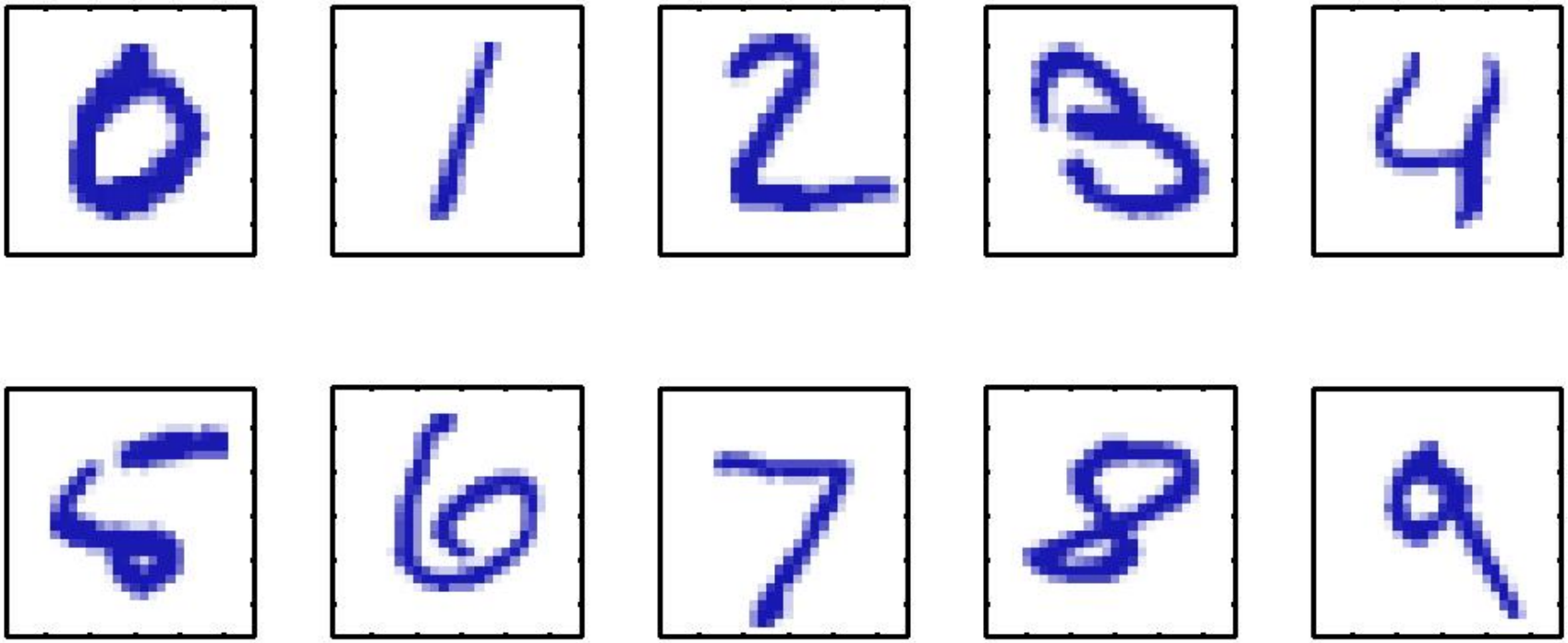


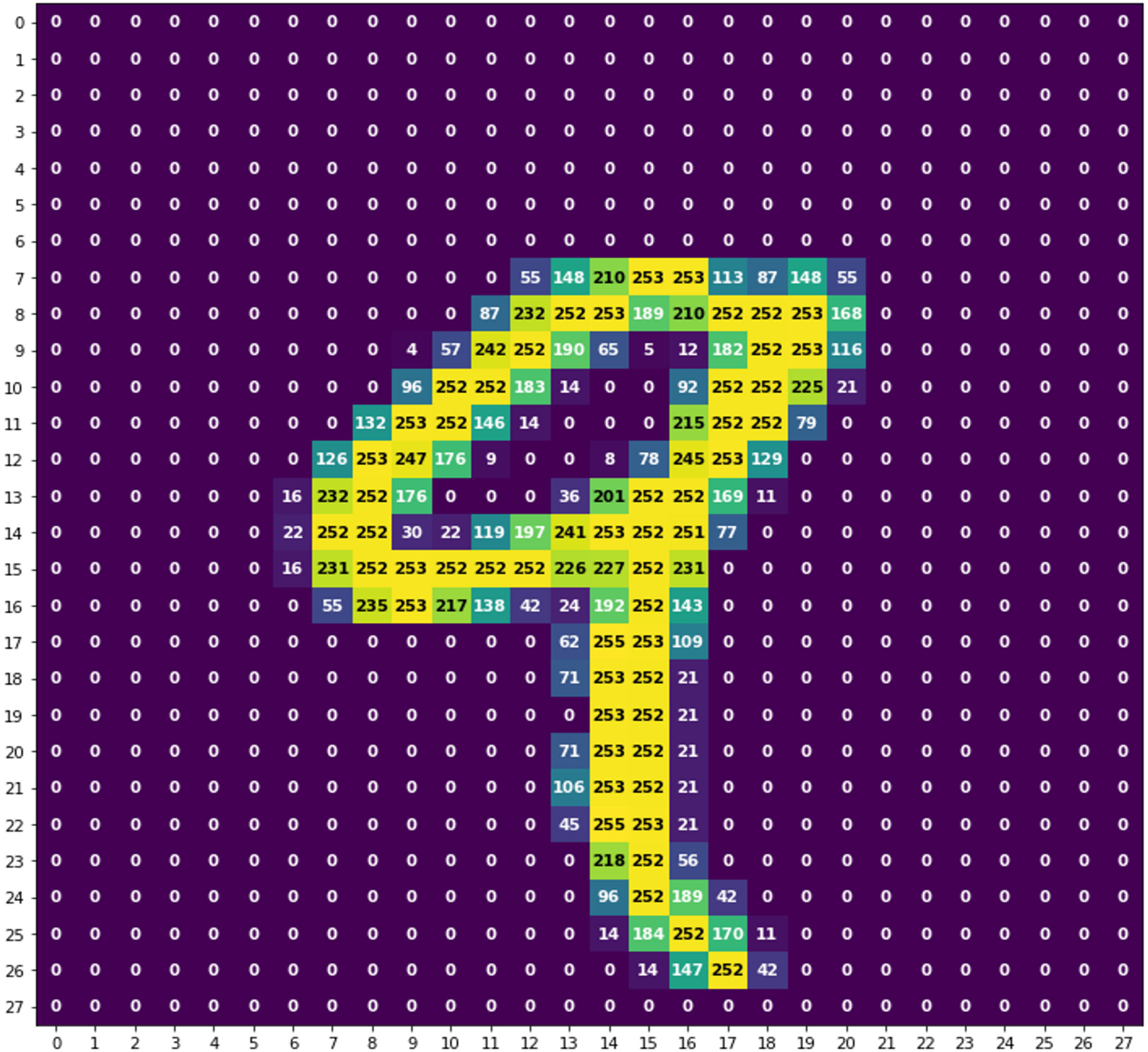
- ✓ A node in the neural network is a mathematical function or activation function which maps input to output values.
- ✓ Inputs represent a set of vectors containing weights (w) and bias (b). They are the sets of parameters to be determined.
- ✓ Many nodes form a **neural layer**, **links** connect layers together, defining a NN model.
- ✓ Activation function (f or σ), is generally a nonlinear data operator which facilitates identification of complex features.

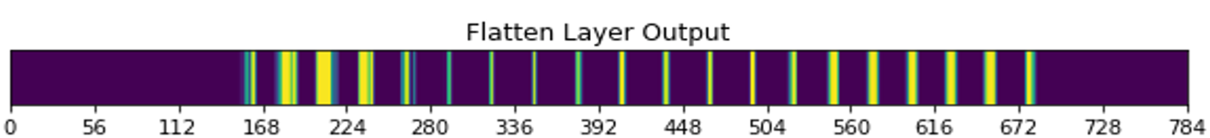
Example: how can computer see images?

Handwritten Digit Recognition (MNIST data set)

The **MNIST database** (*Modified National Institute of Standards and Technology database*) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.[4][5] It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels. The MNIST database contains 60,000 training images and 10,000 testing images. – from Wikipedia







Grayscale image of 28 x 28 pixels

same as a 28 x 28 matrix

values of 0 - 255

Flatten the

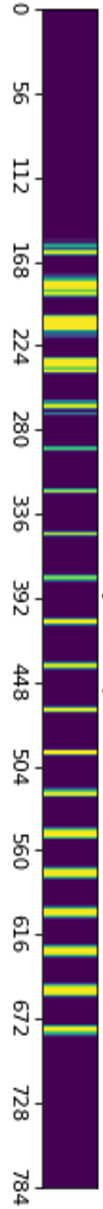
28 x 28 matrix

to a vector of 28 x 28 elements

784 elements

Simple MNIST MLP Network

Google Colab Code



Flatten Layer Output

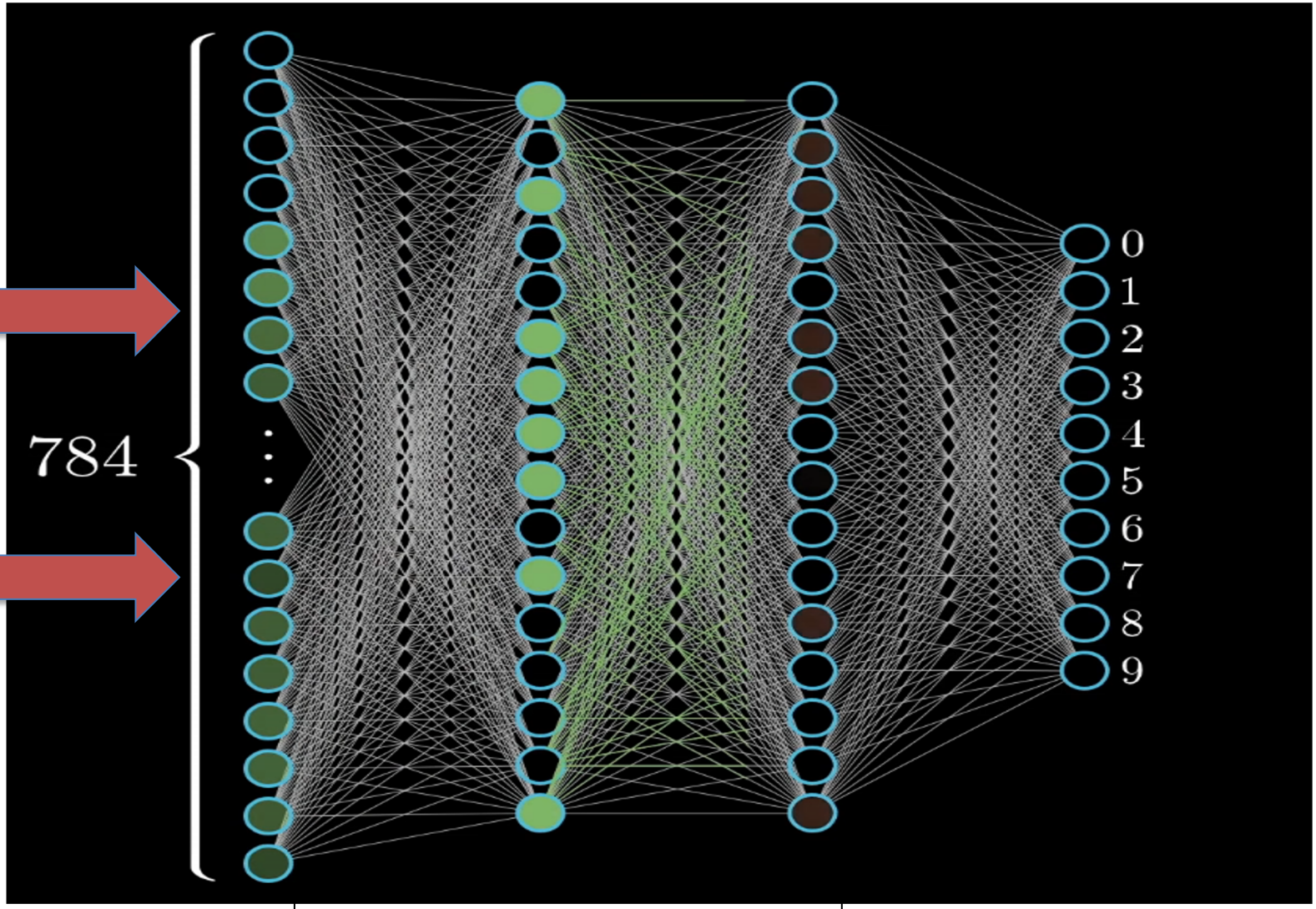
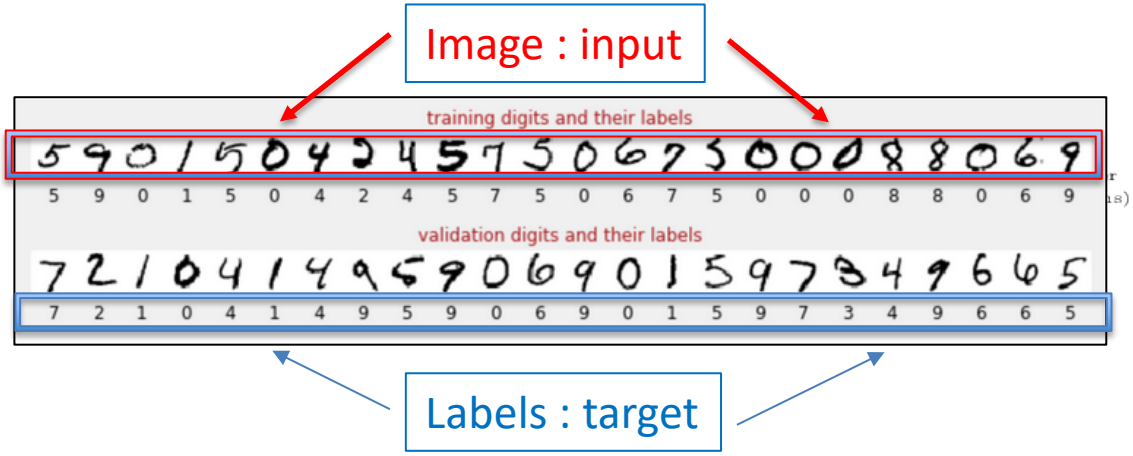
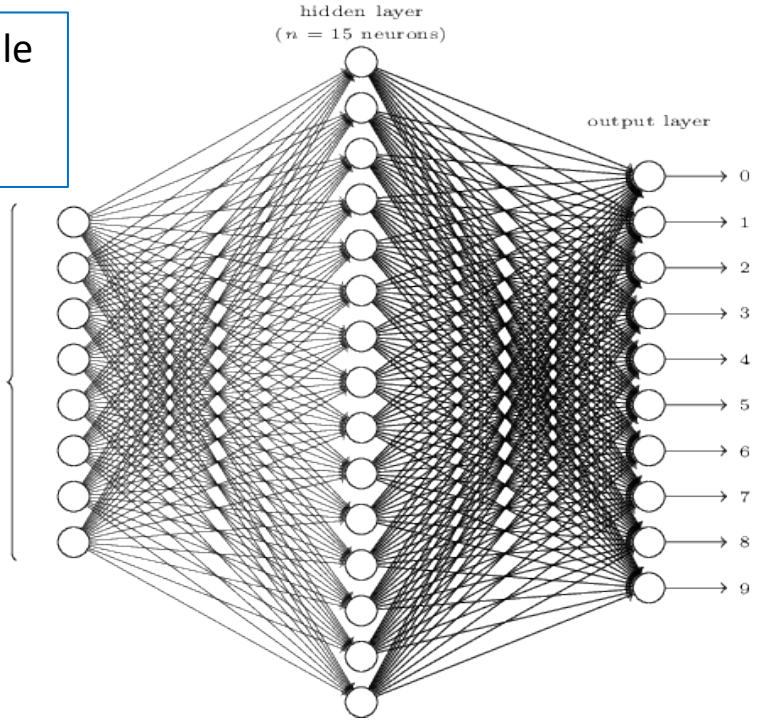


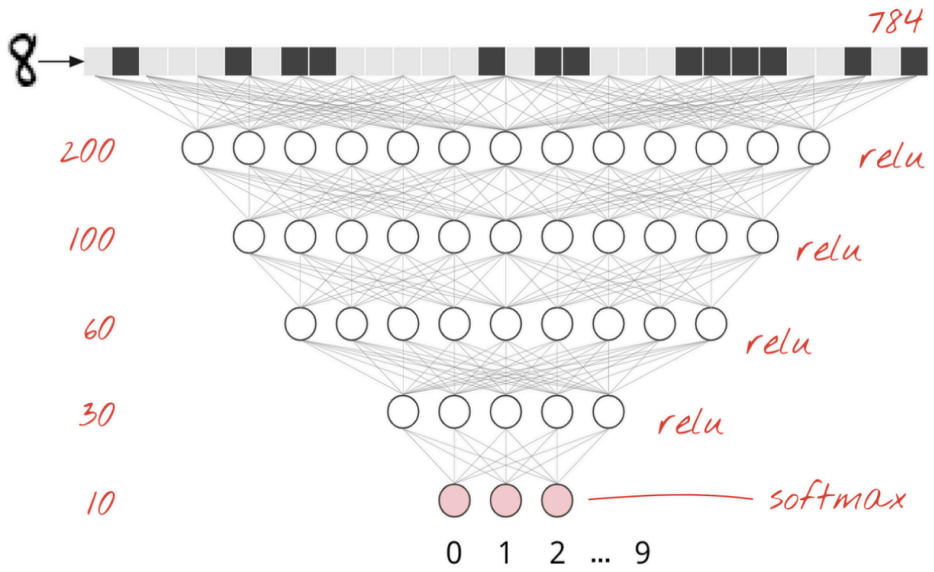
Image:https://www.3blue1brown.com/

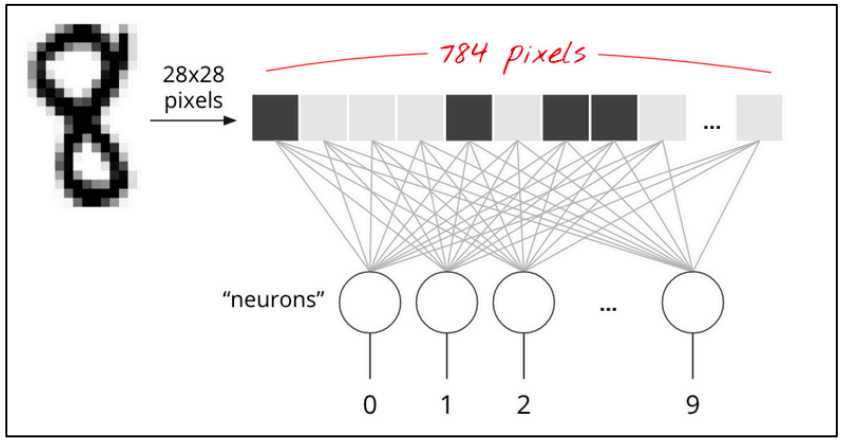
MNIST Example (28x28 pixels)

Handwritten digits in the MNIST dataset are 28x28 pixel grayscale images. The simplest approach for classifying them is to use the 28x28=784 pixels as inputs for a 1-layer neural network.









·Flow, Keras and deep learning, without a PhD
https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist#0

Tensorflow Example

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(16, activation='relu'),
 tf.keras.layers.Dense(16, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

Tensorflow is imported

Sets the mnist dataset to variable "mnist"

Loads the mnist dataset

Builds the layers of the model
4 layers in this model

```
model.compile(optimizer='sgd', loss='SparseCategoricalCrossentropy',
metrics=['accuracy'])

model.summary()

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-
%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
```

Loss Function

Compiles the model with the SGD optimizer

Print summary

Use tensorborad

```
model.fit(x_train, y_train, epochs=5, batch_size=10,
validation_data=(x_test, y_test),  callbacks=[tensorboard_callback])

model.evaluate(x_test,  y_test, verbose=2)
%tensorboard --logdir logs
```

Adjusts model parameters to minimize the loss
Tests the model performance on a test set

Tensorflow Example

The model is trained in about 10 seconds completing 5 epochs with a Nvidia GTX 1080 GPU and has an accuracy of around 97-98%

```
2020-07-29 19:53:40.592860: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1108]      0
2020-07-29 19:53:40.592871: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1121] 0:   N
2020-07-29 19:53:40.593073: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from
SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2020-07-29 19:53:40.593535: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from
SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2020-07-29 19:53:40.593973: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from
SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2020-07-29 19:53:40.594387: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1247] Created TensorFlow device (/job:lo
calhost/replica:0/task:0/device:GPU:0 with 7219 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1080, pci bus
id: 0000:26:00.0, compute capability: 6.1)
2020-07-29 19:53:40.623075: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x55d981198b80 initialized fo
r platform CUDA (this does not guarantee that XLA will be used). Devices:
2020-07-29 19:53:40.623096: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): GeForce GTX
1080, Compute Capability 6.1
2020-07-29 19:53:52.215159: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
 library libcublas.so.10
Epoch 1/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2982 - accuracy: 0.9127
Epoch 2/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1433 - accuracy: 0.9571
Epoch 3/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1097 - accuracy: 0.9663
Epoch 4/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0893 - accuracy: 0.9730
Epoch 5/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0777 - accuracy: 0.9750
313/313 - 0s - loss: 0.0727 - accuracy: 0.9776
```

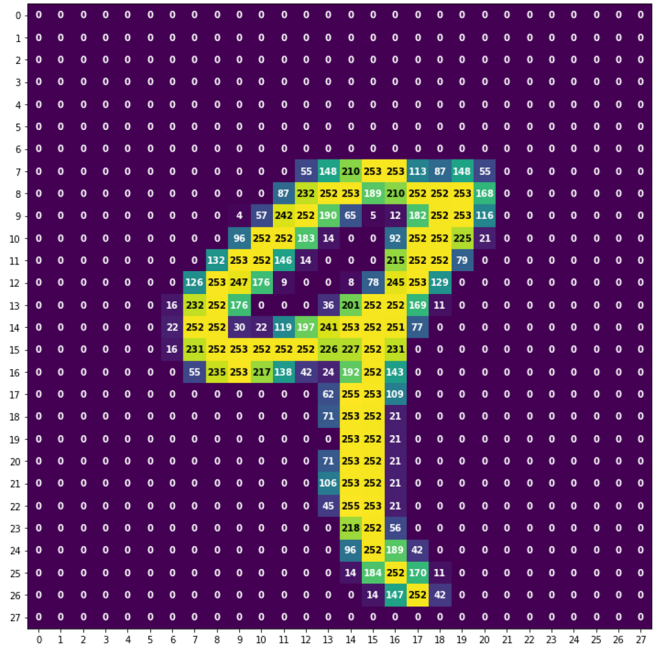
TensorFlow 2.0

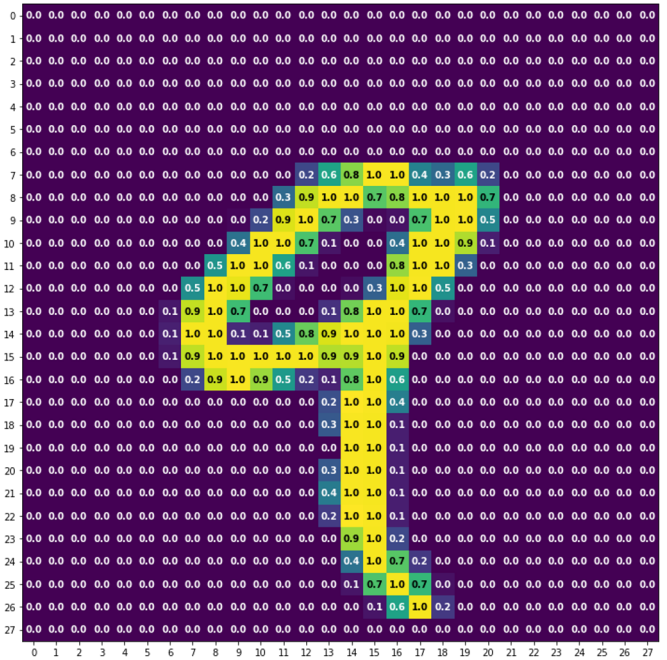
Loading and Preparing the MNIST Dataset

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
#download the images
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
x_train, x_test = x_train / 255.0,
x_test / 255.0 #normalize
```

Each Image is a 28x28 image of a handwritten digit

x 60,000 images





TensorFlow 2.0 : NN Network

Let's use this simple network to demonstrate what TensorFlow is doing behind the scenes

```
model = tf.keras.models.Sequential([
 tf.keras.layers.Flatten(input_shape=(28, 28)),
 tf.keras.layers.Dense(16, activation='relu'),
 tf.keras.layers.Dense(16, activation='relu'),
 tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='sgd',
              loss='Catagorical_Crossentropy,
              metrics=['accuracy'])
Model.summary
model.fit(x_train, y_train, epochs=3, batch_size=10)
```

Simple MLP Network

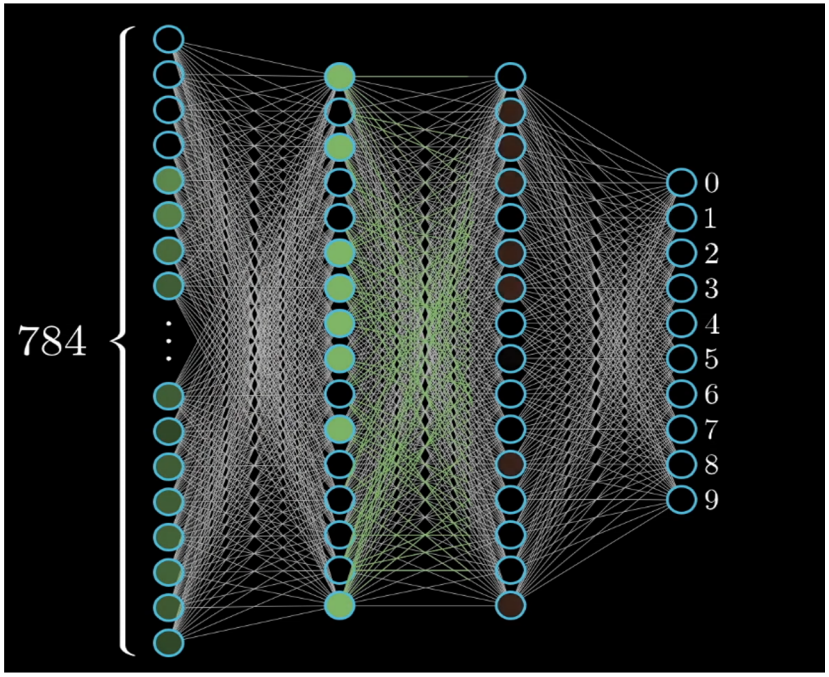
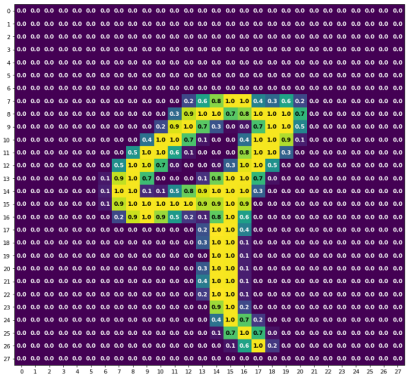


Image:
https://www.3blue1brown.com/

- ✓ **Batch Size**: how many images do we pass through the network for each iteration (10)
- ✓ **Epochs**: how many times we pass over the entire dataset (3)
- ✓ **# Iterations per Epoch** = number of images / batch size (60,000/10=6,000)

https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#0

TensorFlow 2.0 : Forward Pass



```
tf.reshape(tensor, shape)
#example
tf.reshape(x_train[0], [784])
```

Flatten a single image

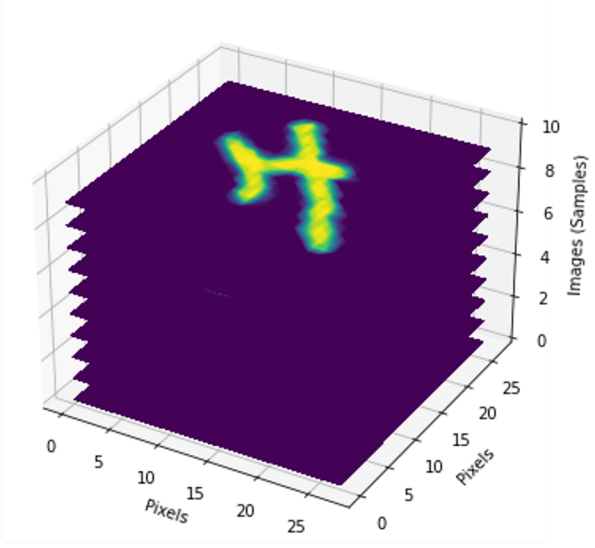
```
tf.reshape(tensor, shape)
#example
tf.reshape(x_train[0:10], [10: 784])
```

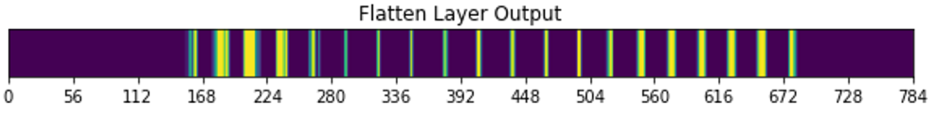
Flatten a batch of images

Flatten

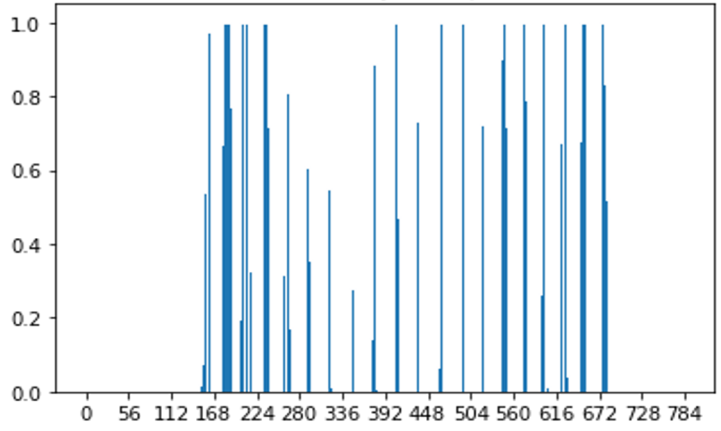
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28))
```

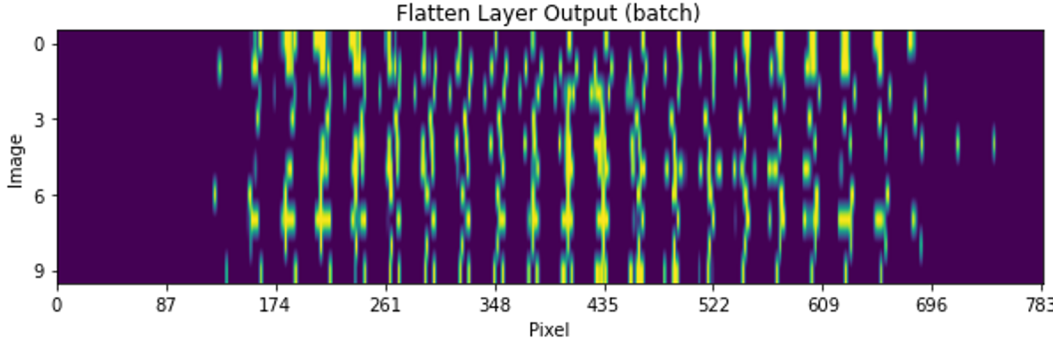
Network Input Per Iteration





=

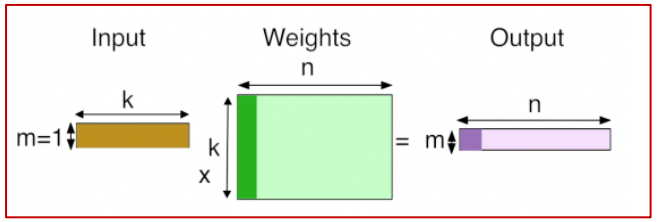




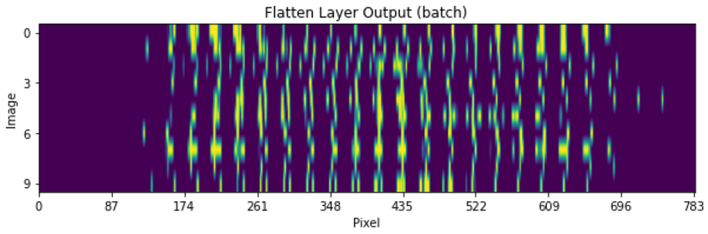
TensorFlow 2.0 : Forward Pass

1ST Fully Connected Layer with 16 neurons
Matrix multiplication

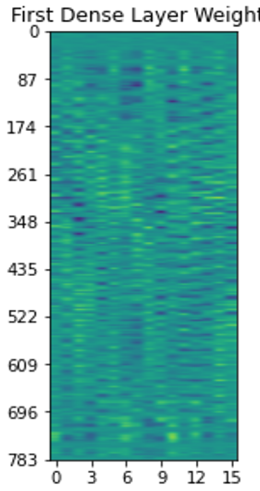
```
...
 tf.keras.layers.Flatten(input_shape=(28, 28)),
 tf.keras.layers.Dense(16, activation='relu'),
...
```



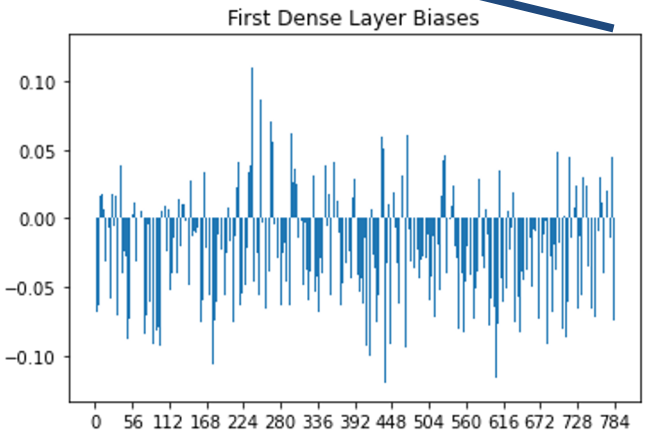
$$a^1 = \text{reLU}(X * W^1 + b^1)$$



*



+



Weights are initialized with random values

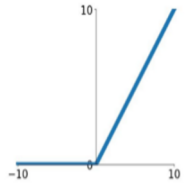
TensorFlow 2.0 : Forward Pass

Second Fully Connected Layer with 16 neurons

```
model = tf.keras.models.Sequential([
 tf.keras.layers.Flatten(input_shape=(28, 28)),
 tf.keras.layers.Dense(16, activation='relu'),
 tf.keras.layers.Dense(16, activation='relu'),
 ...
```

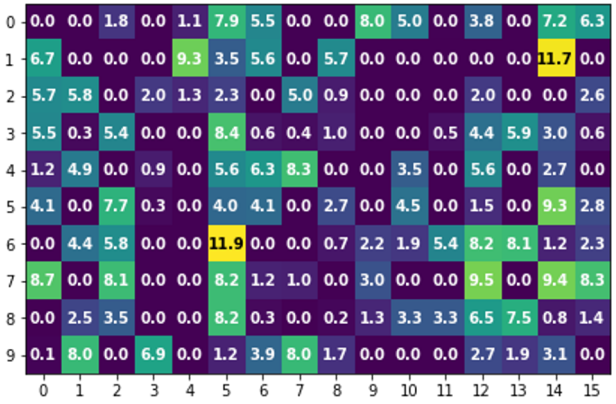
ReLU

$\max(0, x)$



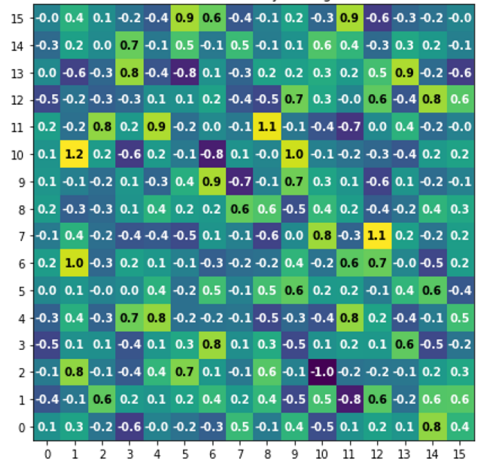
$a^2 = \text{reLU}(a^1 * W^2 + b^2)$

First Dense Layer Output (batch)



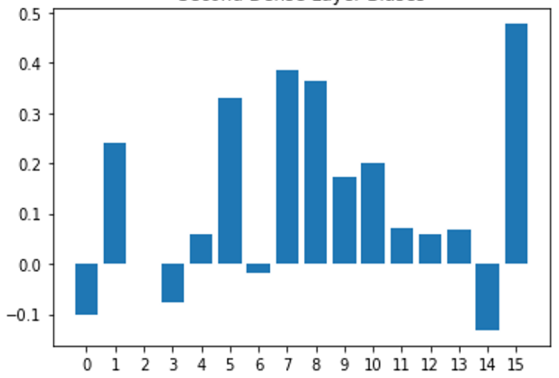
*

Second Dense Layer Weights



+

Second Dense Layer Biases



TensorFlow 2.0 : Forward Pass

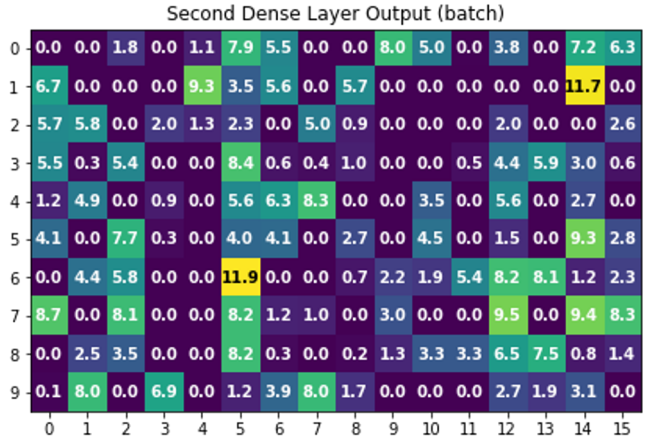
Final Fully Connected Layer with 10 neurons : 10 classes

Softmax = $\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

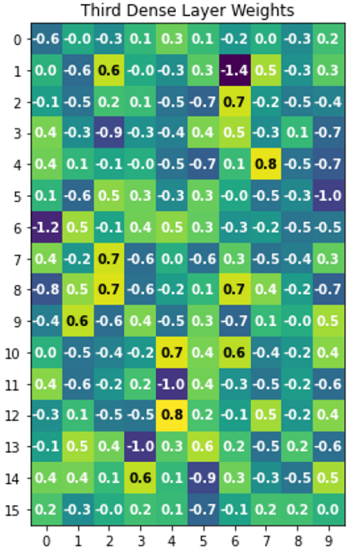
- ✓ The softmax function is a function that turns a vector of K real values into values between 0 and 1, and the values sum up to 1, so that they can be interpreted as probabilities.
- ✓ This is because the softmax is a generalization of logistic regression that can be used for multi-class classification.

https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

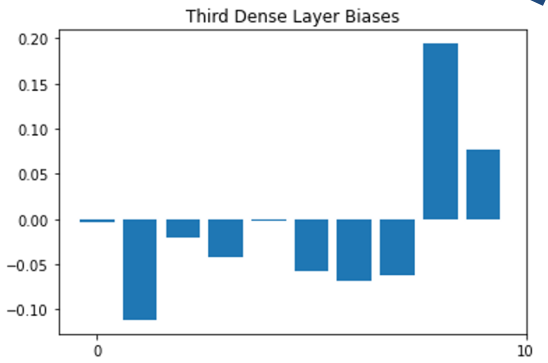
$a^3 = \text{softmax}(a^2 * W^3 + b^3)$



*



+



TensorFlow 2.0 : Forward Pass

`tf.losses.mean_squared_error(y_true_tf, y_pred_tf)`

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Evaluating the Error

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

Probabilities must be >= 0

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Probabilities must sum to 1

$$L_i = -\log P(Y = y_i|X = x_i)$$

cat	3.2	exp	24.5	normalize	0.13	compare 1.00
car	5.1		164.0		0.87	0.00
frog	-1.7		0.18		0.00	0.00

Unnormalized log-probabilities / logits

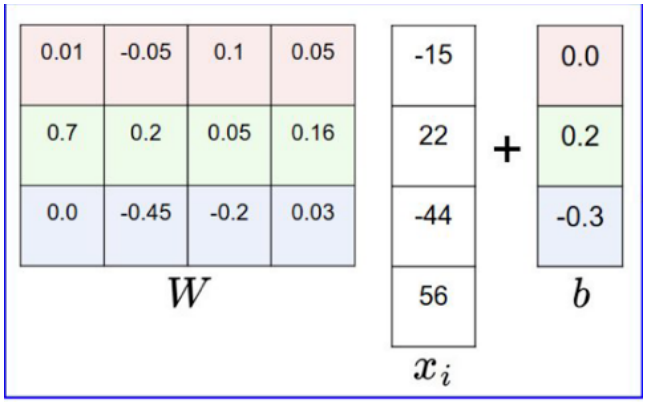
unnormalized probabilities

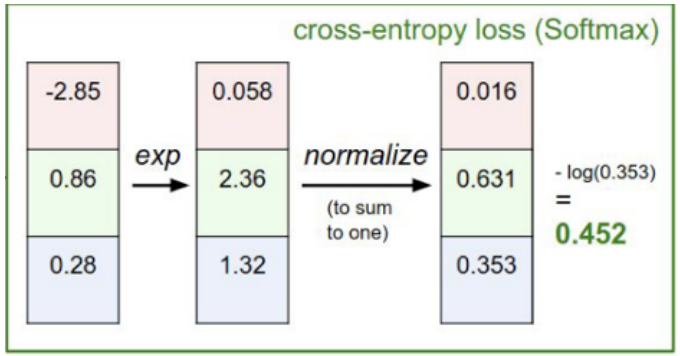
probabilities

Cross Entropy

$$H(P, Q) = H(p) + D_{KL}(P\|Q)$$

Correct probs



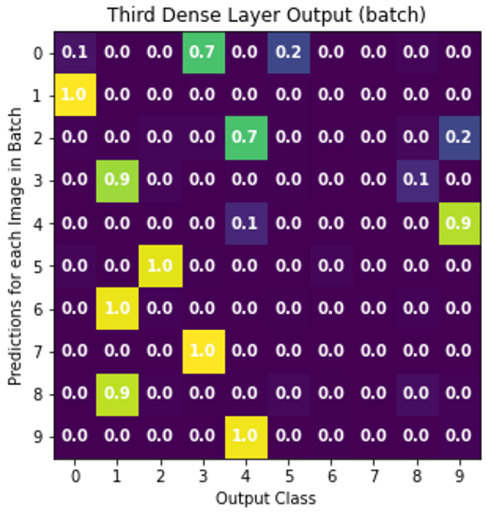


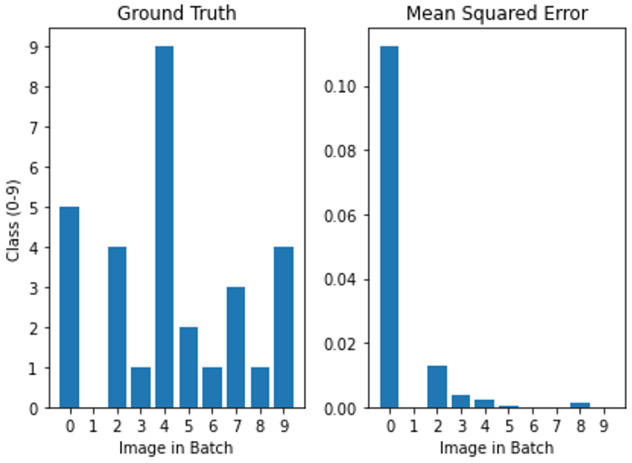
TensorFlow 2.0 : Forward Pass

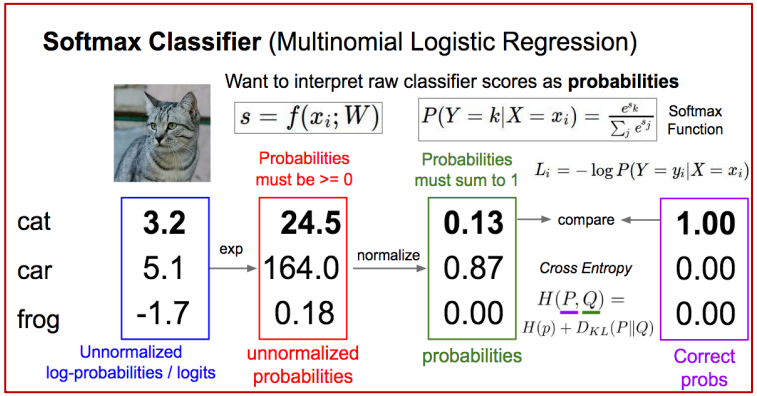
tf.losses.mean_squared_error(y_true_tf, y_pred_tf)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

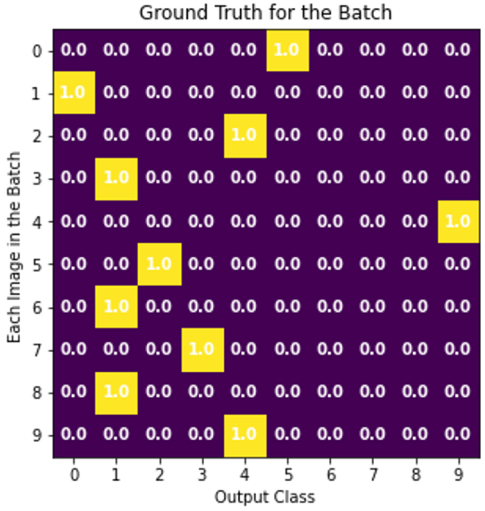
Evaluating the Error

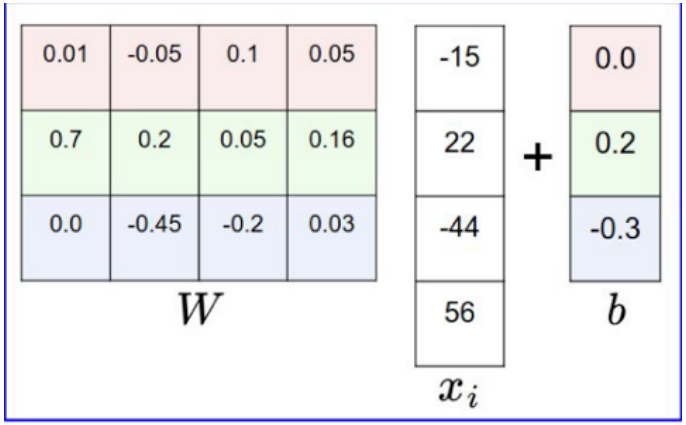


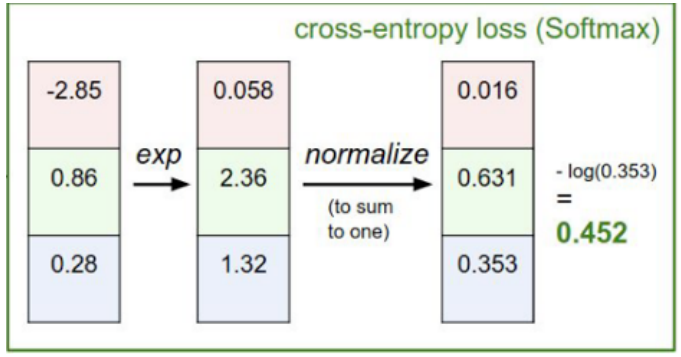




$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$







Simple MNIST MLP Neural Network

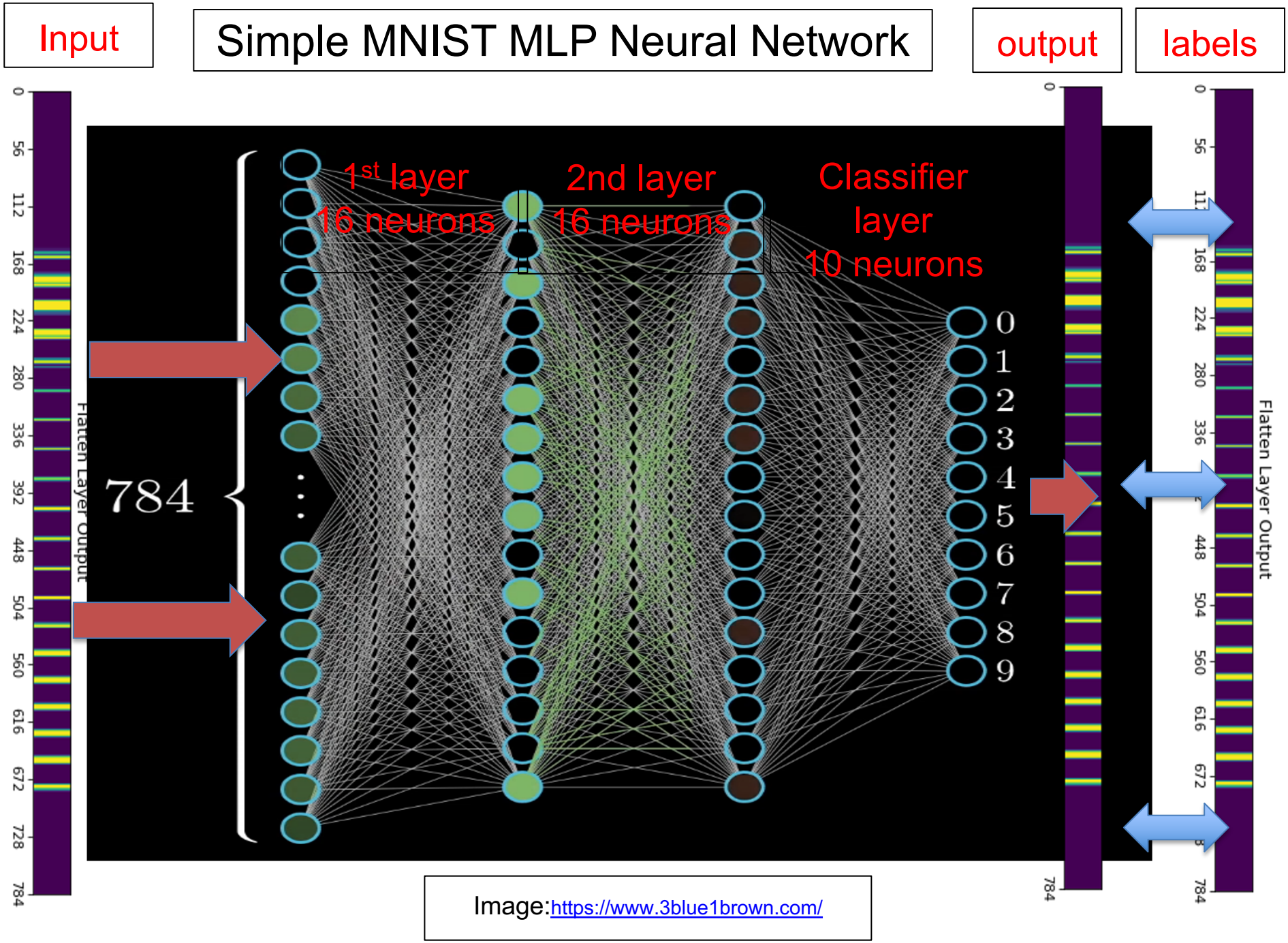
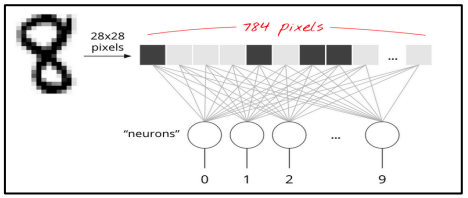


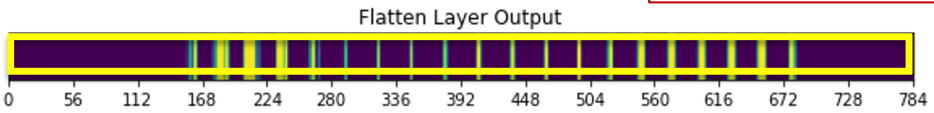
Image:https://www.3blue1brown.com/

Summary : Flow of MLP Dense layer NN

1 image = a vector of 784



An image of 28x28 pixels



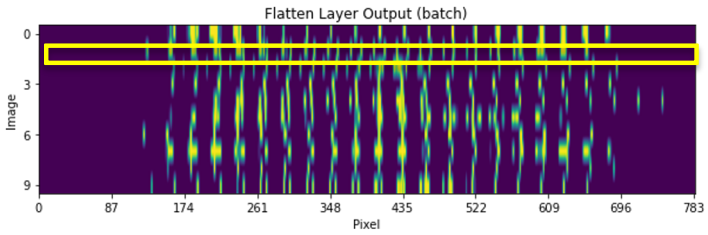
Input to 1st layer

10 images : batch of 10
10 x 784 matrix

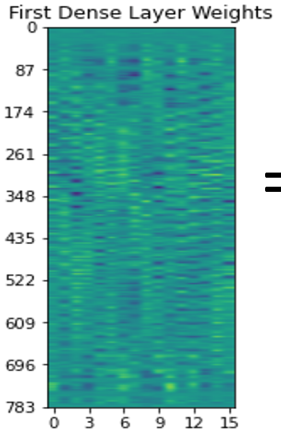
16 neurons
784 x 16 W matrix

10 x 16 output matrix

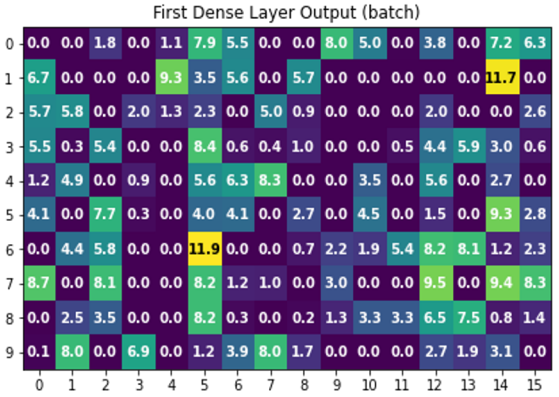
Input to 2nd layer



*

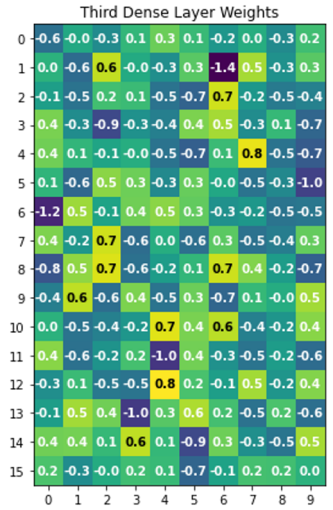


=



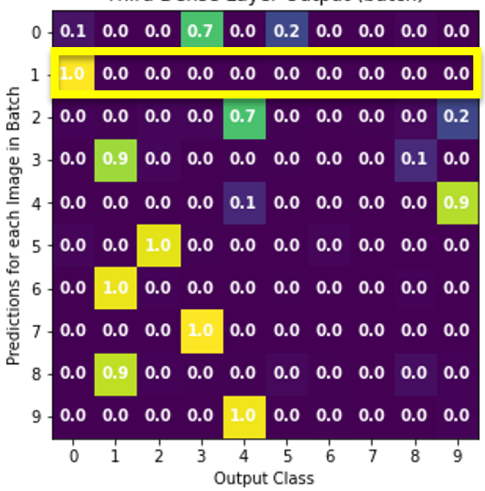
10 x 16 matrix

*

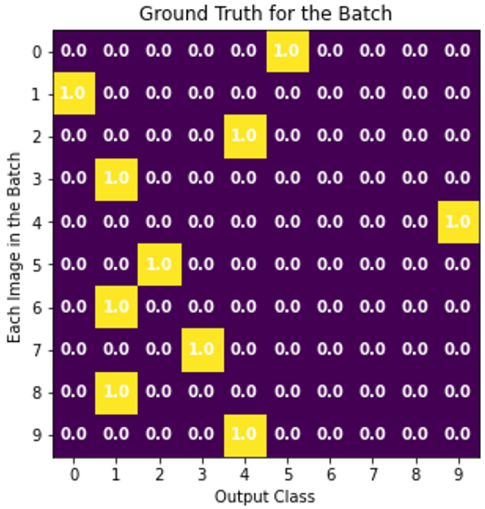


16 neurons
16 x 10
W matrix

=

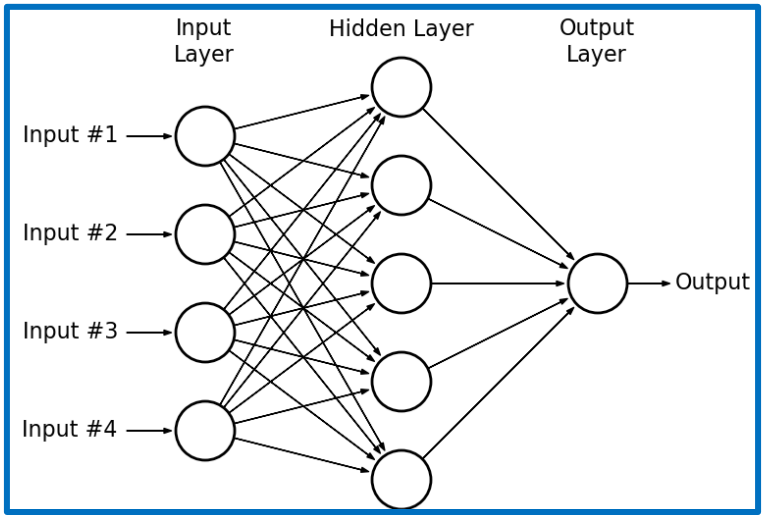


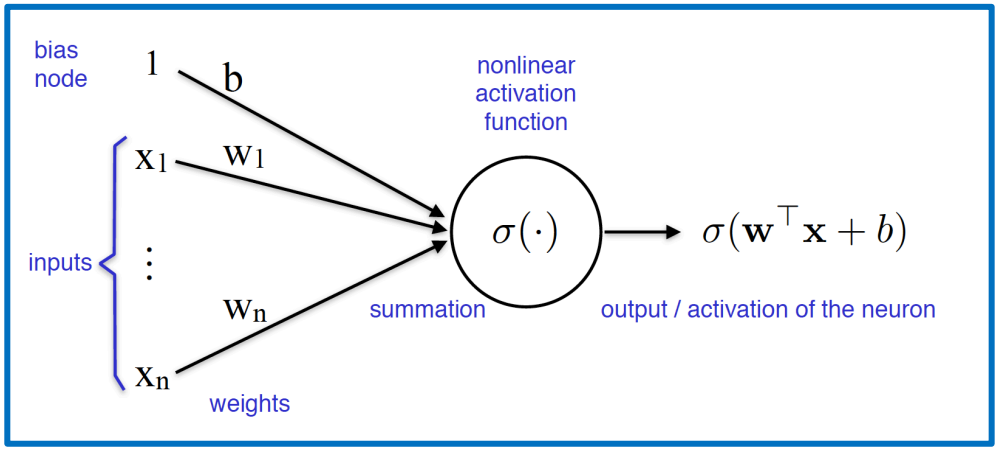
10 x 10 output matrix



10 x 10 labeled matrix

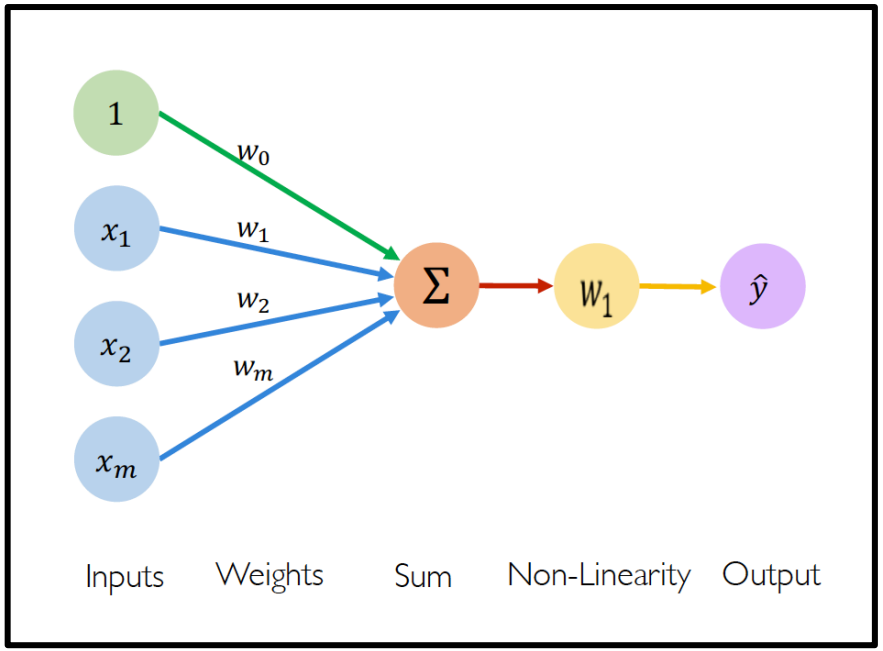
NN Modeling





- ✓ A node in the neural network is a mathematical function or activation function which maps input to output values.
- ✓ Inputs represent a set of vectors containing weights (w) and bias (b). They are the sets of parameters to be determined.
- ✓ Many nodes form a **neural layer**, **links** connect layers together, defining a NN model.
- ✓ Activation function (f or σ), is generally a nonlinear data operator which facilitates identification of complex features.

DNN MLP Forward Steps



$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Output

Linear combination of inputs

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

Non-linear activation function

Bias

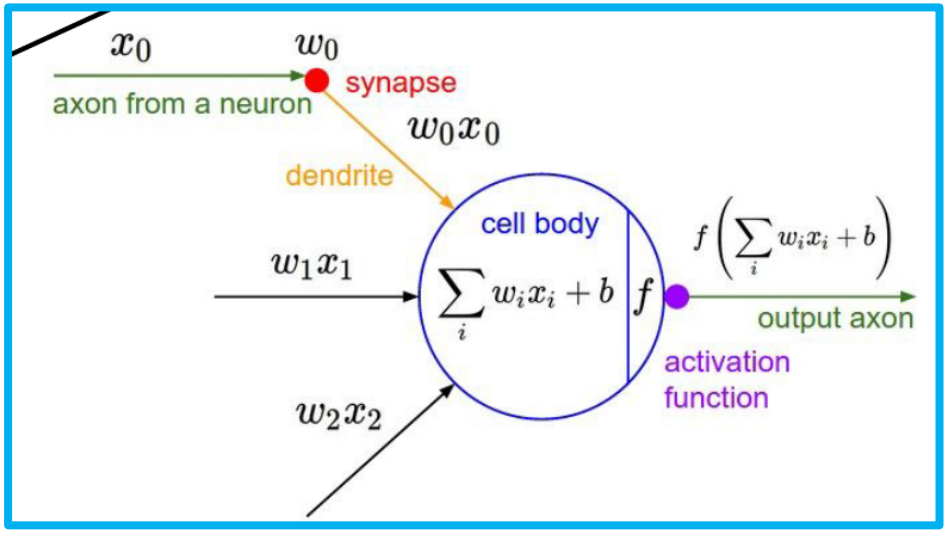
$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

- Example: sigmoid function

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

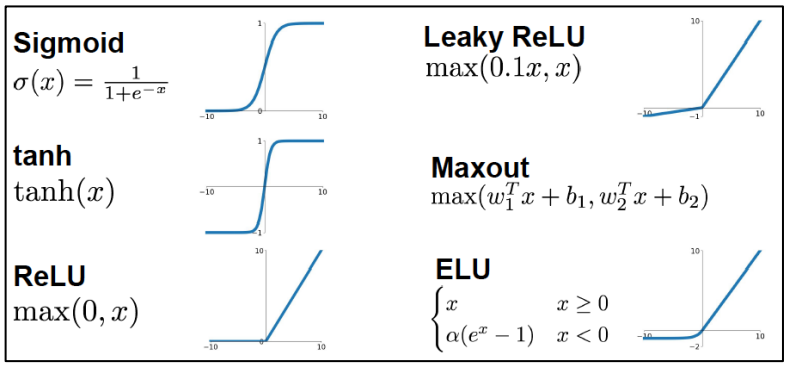
Perceptron

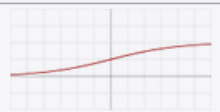
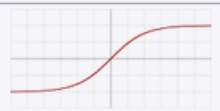
Perceptrons were developed in the 1950s and 1960s by the scientist Frank Rosenblatt. A perceptron takes several binary inputs, *x*1,*x*2,…, and produces a single binary output. By varying the weights and the threshold, we can get different models of decision-making.



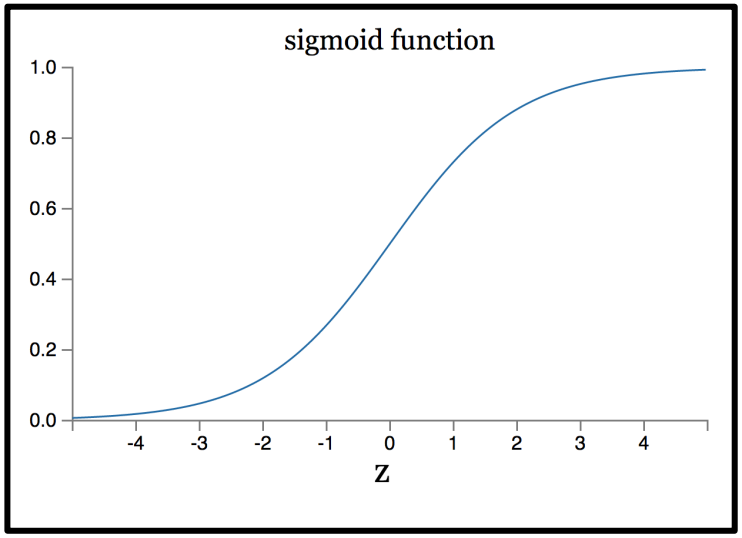
$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

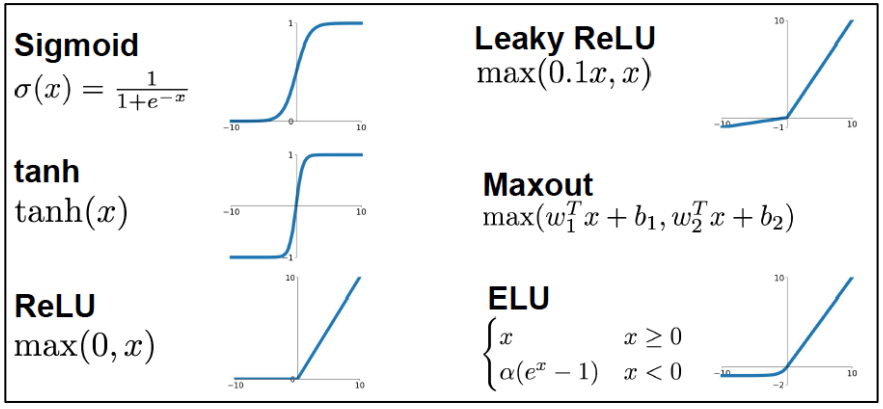


Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1+\lvert x\rvert}$
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$

Activation Function



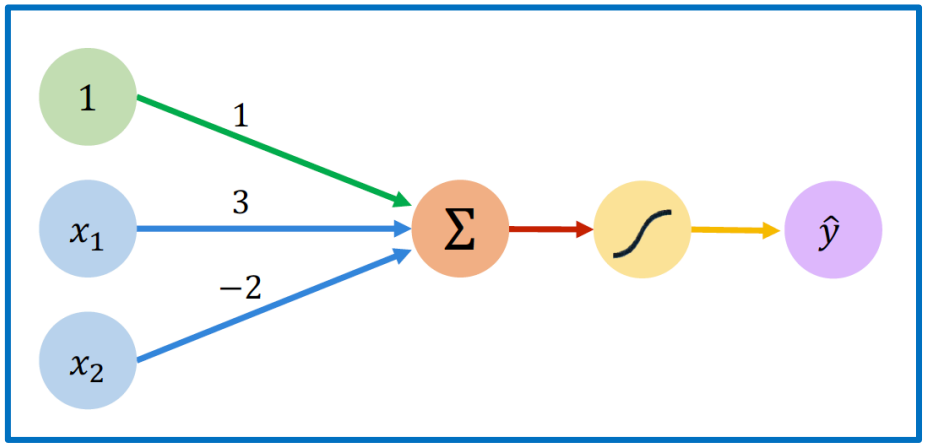
$$\sigma(z) \equiv \frac{1}{1+e^{-z}}.$$



- ✓ In much modern work on neural networks, the main neuron model used is one called the *sigmoid neuron*.
- ✓ Sigmond (logistic) function, σ, is a smooth out preceptron operating on a set of data, $\sigma(w \cdot x+b)$. The smoothness of σ means that small changes Δw in the weights and Δb in the bias will produce a small change Δoutput in the output from the neuron.
- ✓ If $z \equiv w \cdot x+b$ is a large positive number, then $exp(-z) \approx 0$ and so $\sigma(z) \approx 1$.
- ✓ If $z=w \cdot x+b$ is very negative, then $exp(-z) \rightarrow \infty$, and $\sigma(z) \approx 0$. So when $z=w \cdot x+b$ is very negative.
- ✓ It's only when $w \cdot x+b$ is of modest size that there's much deviation from the perceptron model.

neuralnetworksanddeeplearning.com

Activation Function Example



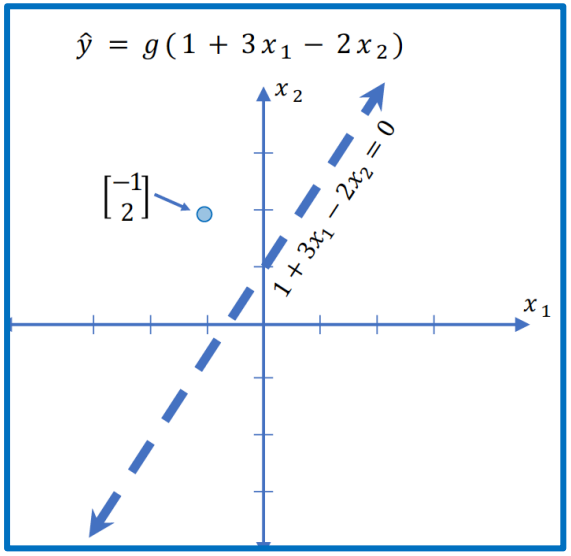
We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

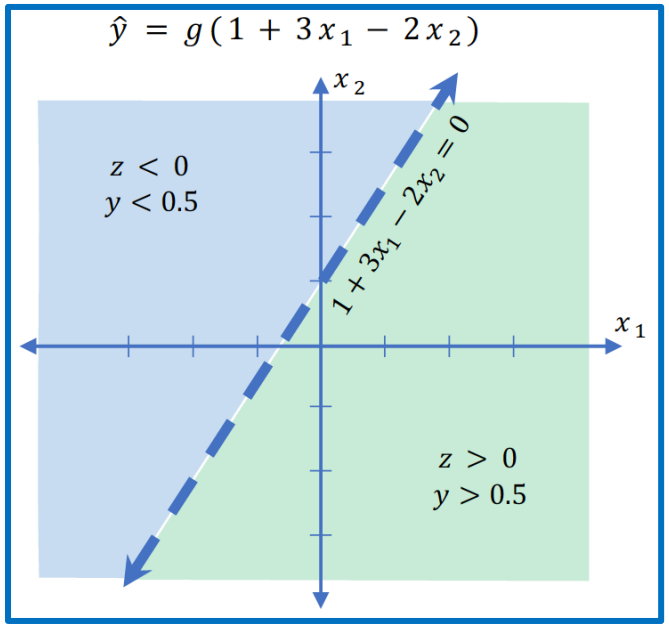
$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$

This is just a line in 2D!

Assume we have input: $\boldsymbol{X} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

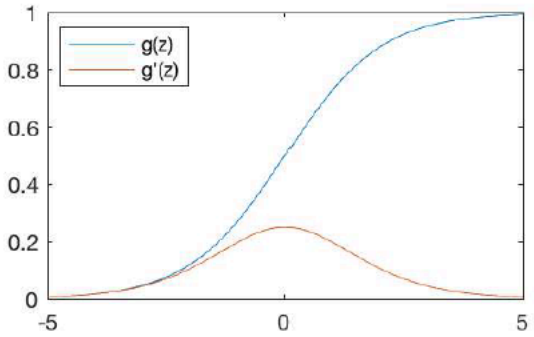
$$\hat{y} = g\left(1 + (3 * -1) - (2 * 2)\right)$$
$$= g(-6) \approx 0.002$$





Nonlinear Activation Function

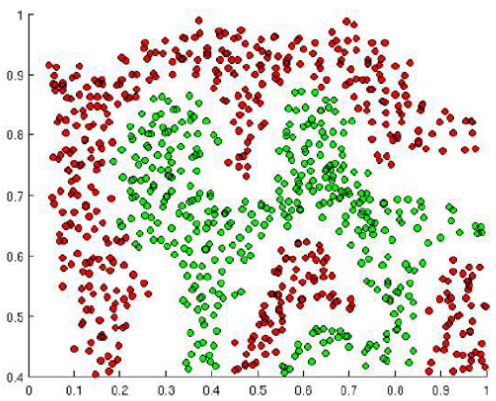
Sigmoid Function

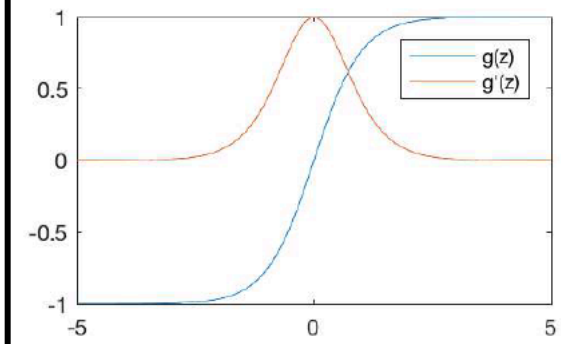


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

tf.nn.sigmoid(z)

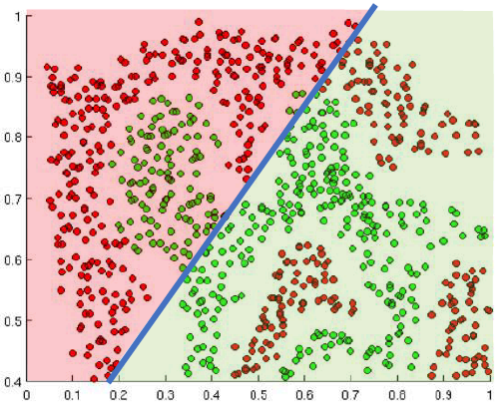


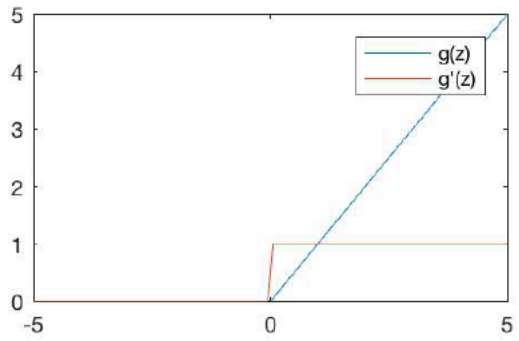


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

tf.nn.tanh(z)

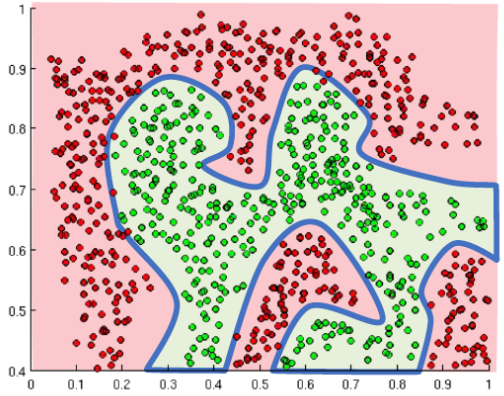




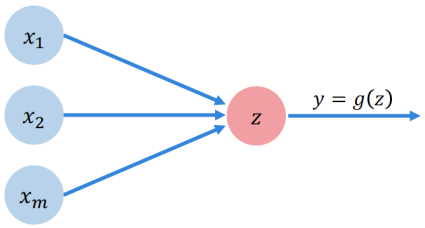
$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

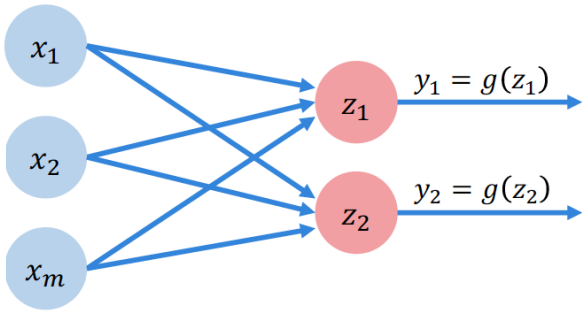
tf.nn.relu(z)



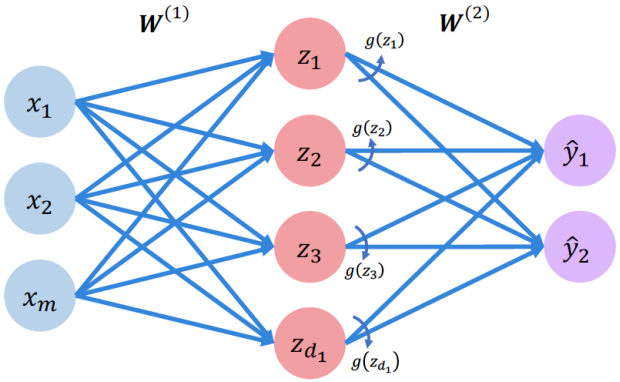
Multilayer Perceptron : Forward Path Calculation



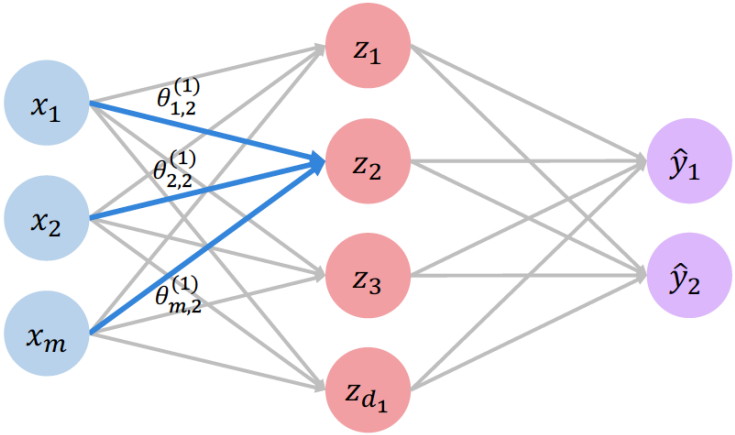
$$z = w_0 + \sum_{j=1}^{m} x_j w_j$$



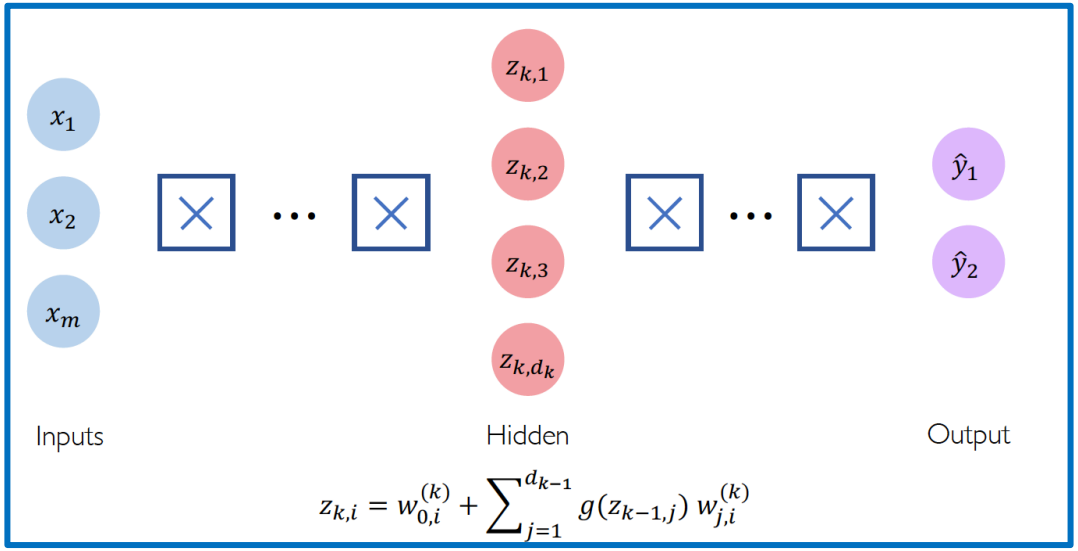
$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j w_{j,i}$$



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)}\right)$$

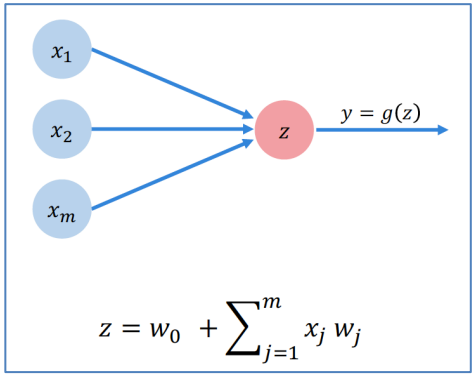


$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j w_{j,2}^{(1)}$$
$$= w_{0,2}^{(1)} + x_1 w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + x_m w_{m,2}^{(1)}$$



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Forward Calculation represented as a graph



Sigmoid function

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

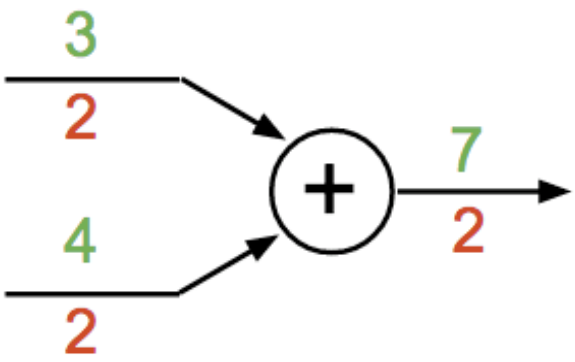
Z = (x1*w1 + x2*w2) + b = (-1.0*-2.0 + -2.0*3.0) + -3.0 = 4.0 – 3.0 = 1.0

Y = σ (z) = 1 / (1 + e^{-Z}) = 1 / (1 + e^{-1}) = 0.731

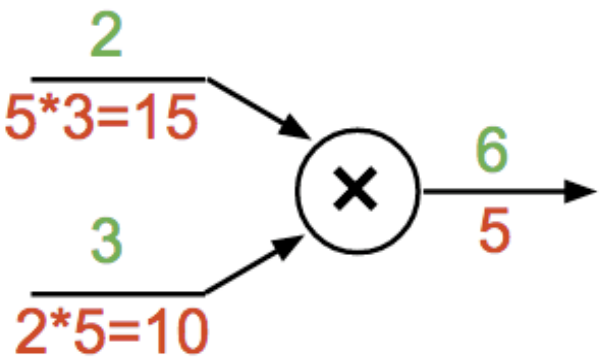
Forward computational Operators

Forward path (Green) : backward path

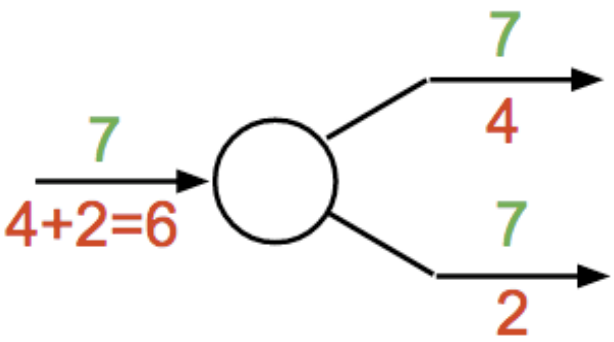
add gate: gradient distributor



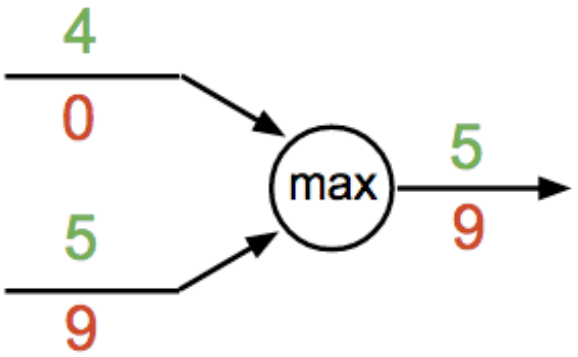
mul gate: "swap multiplier"



copy gate: gradient adder



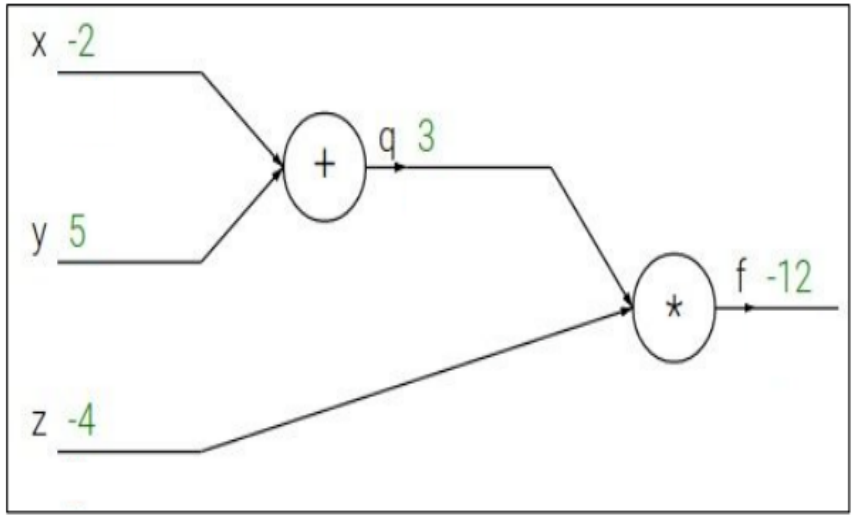
max gate: gradient router



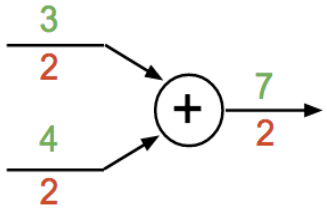
http://cs231n.stanford.edu/slides/2020/lecture_4.pdf

Computational Graph

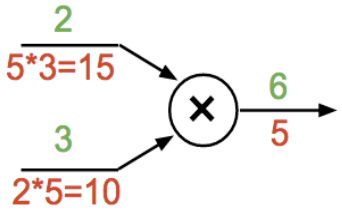
http://cs231n.stanford.edu/syllabus.html



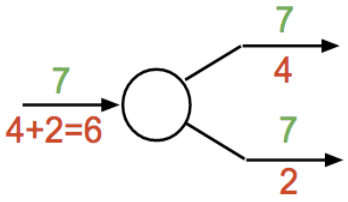
add gate: gradient distributor



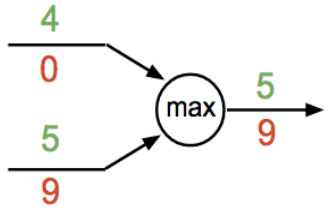
mul gate: "swap multiplier"



copy gate: gradient adder



max gate: gradient router



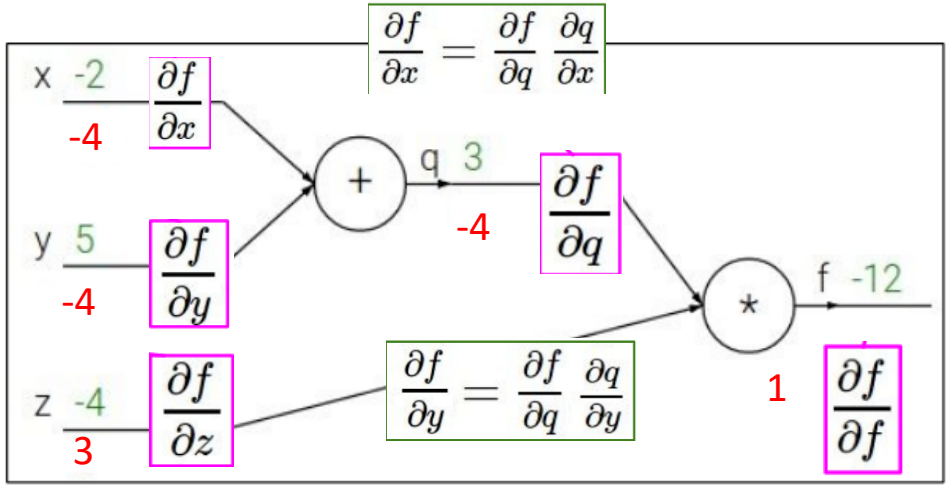
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$f(x, y, z) = (x + y)z$

e.g. x = -2, y = 5, z = -4

$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$

$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

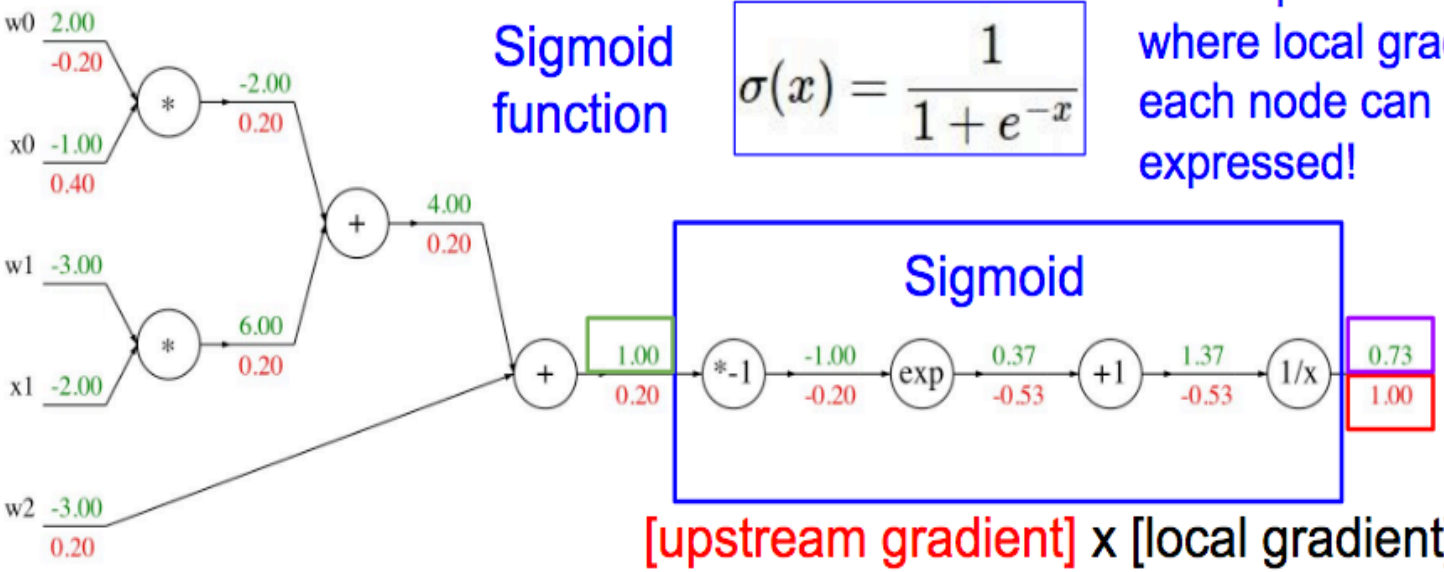


Computational Graph

http://cs231n.stanford.edu/syllabus.html

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



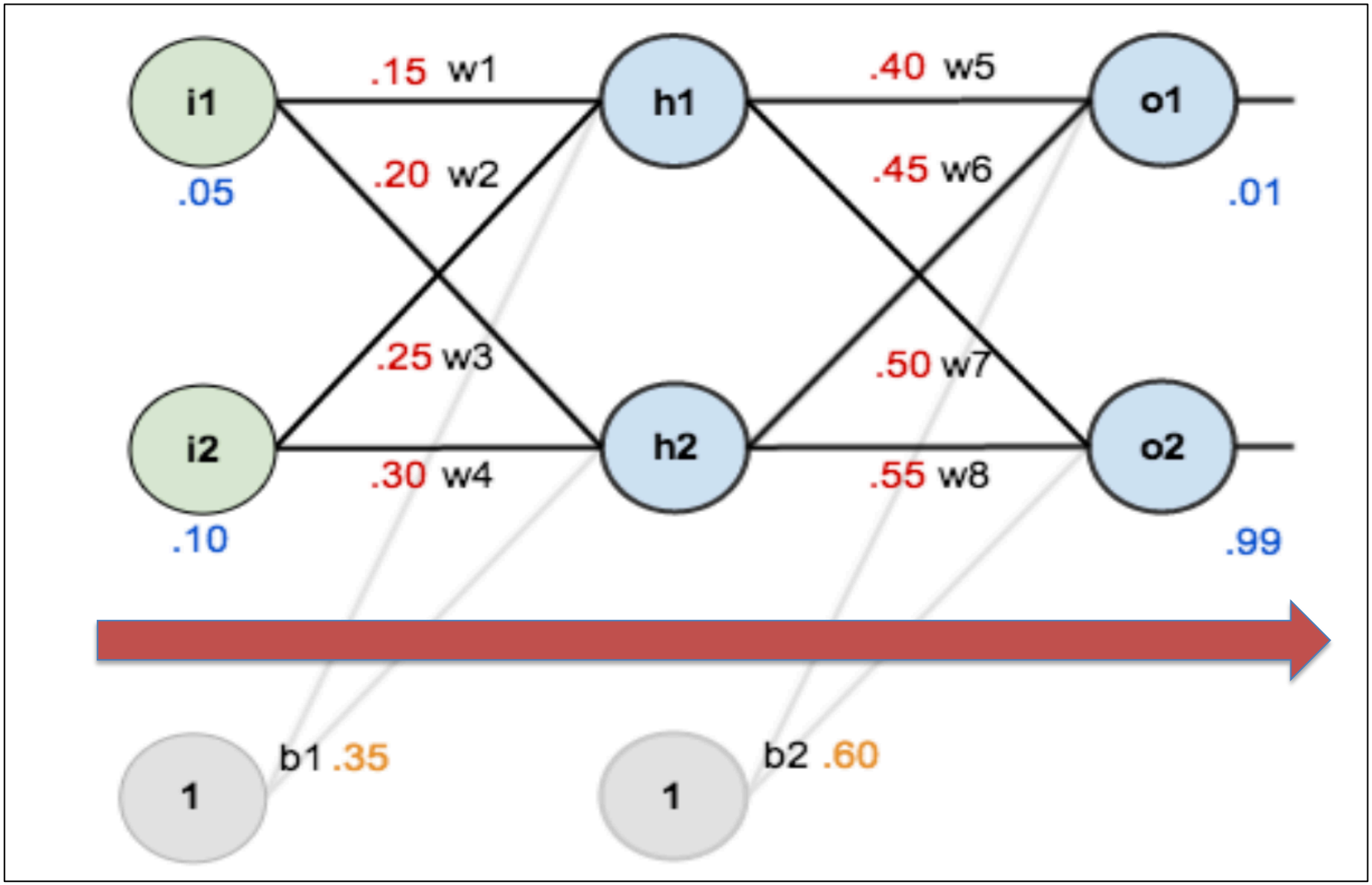
Sigmoid function $$\sigma(x) = \frac{1}{1+e^{-x}}$$

[upstream gradient] x [local gradient]

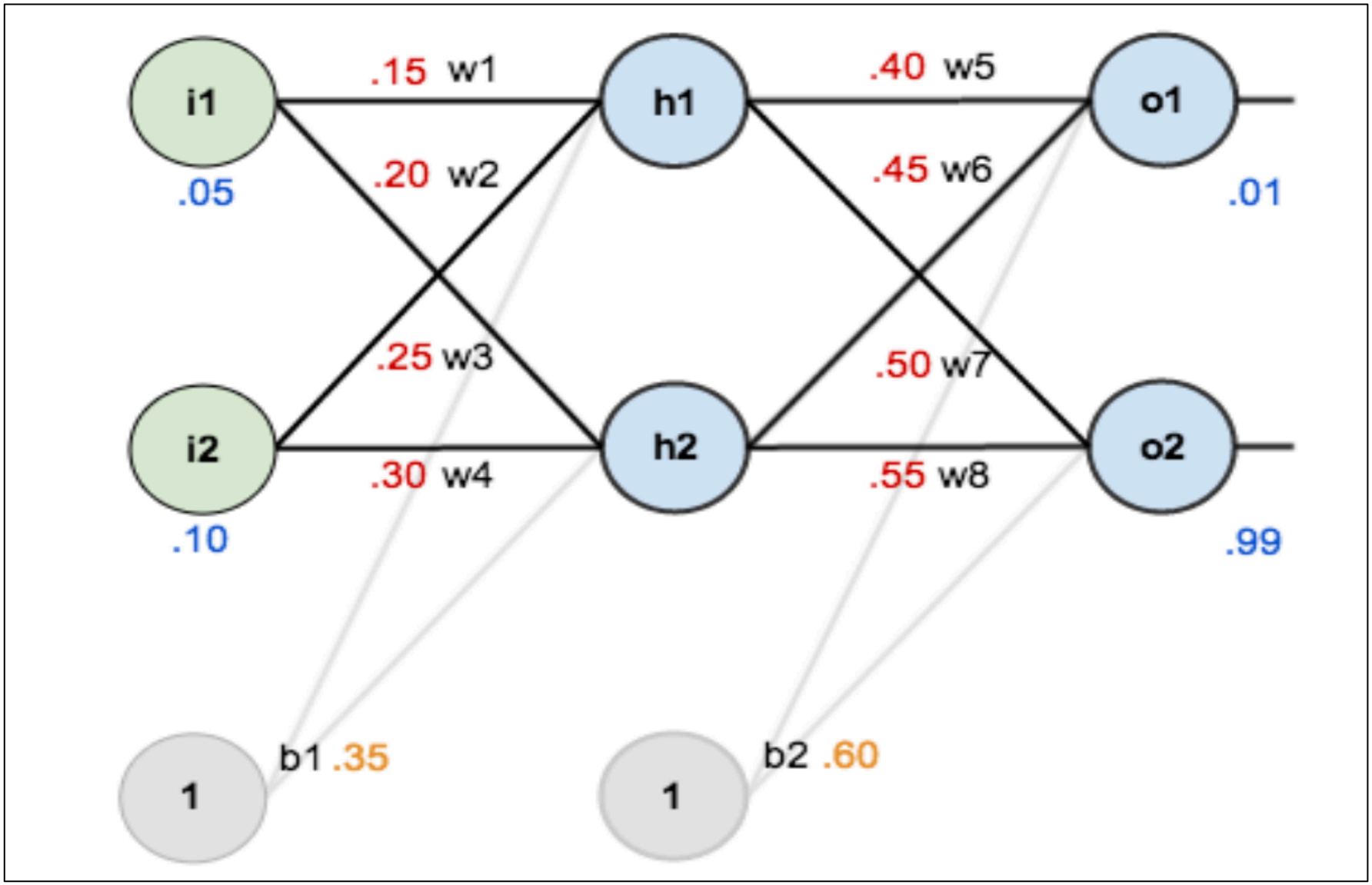
[1.00] x [(1 - 0.73) (0.73)] = 0.2

Sigmoid local gradient: $$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

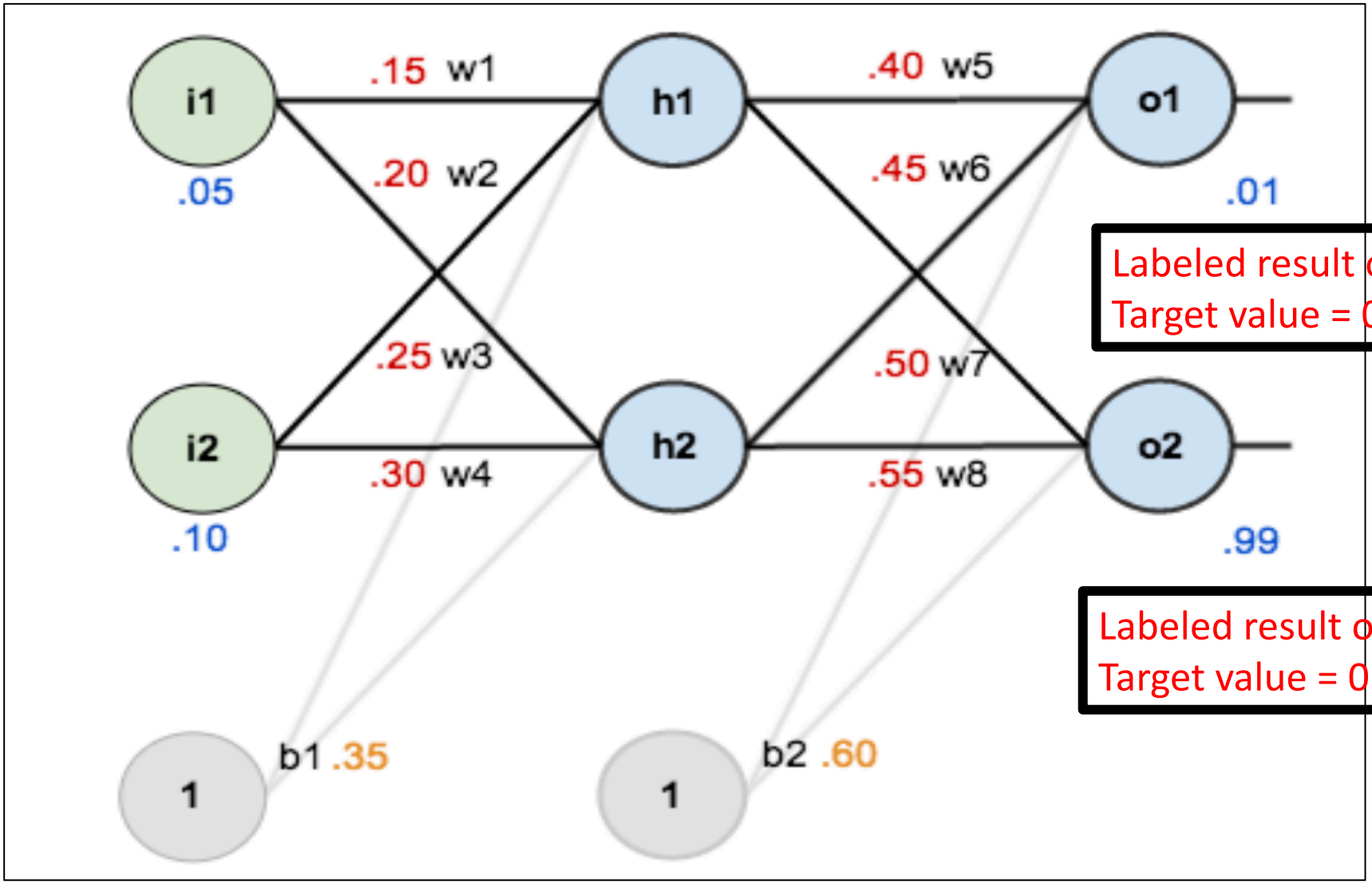
Multilayer Perceptron : Forward Path Calculation



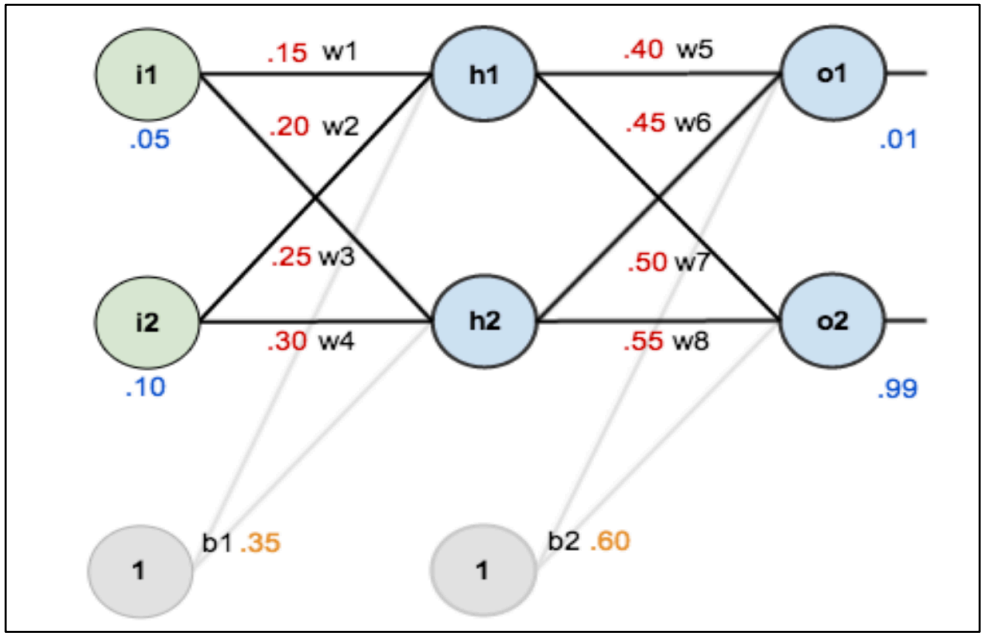
Multilayer Perceptron : Forward Path Calculation



DNN Example


i1
.05
i2
.10
.15 w1
.20 w2
.25 w3
.30 w4
h1
h2
.40 w5
.45 w6
.50 w7
.55 w8
o1
.01
o2
.99
Labeled result of out1
Target value = 0.01
Labeled result of out2
Target value = 0.99
1
b1 .35
1
b2 .60

DNN Example (1)



w1=0.15

i1 = 0.05

*

w2=0.2

i2 = 0.1

*

+

+

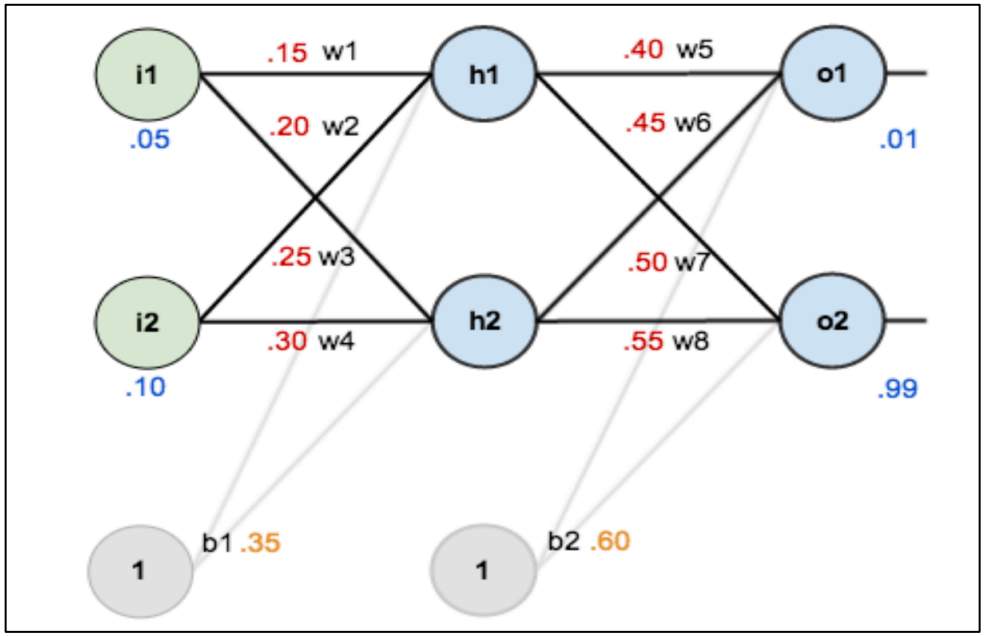
σ

h1 = 0.59326992

b1=0.35

1

DNN Example (2)



w3=0.25

i1 = 0.05

*

w4=0.3

i2 = 0.1

*

+

+

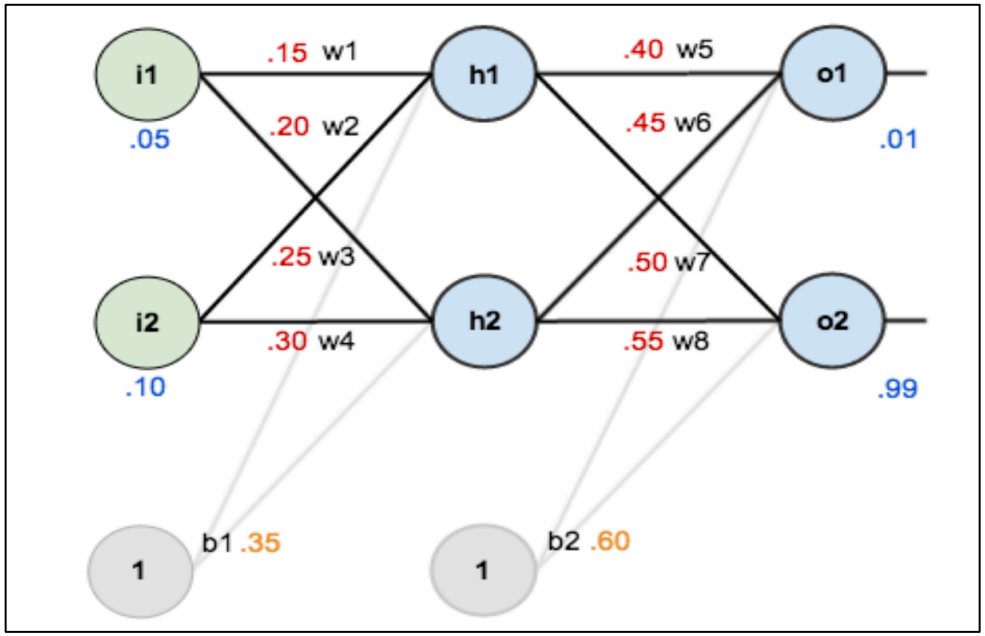
σ

h2 = 0.596884378

b1=0.35

1

DNN Example (3)



w5=0.4

h1= 0.59327

*

w6=0.45

h2 = 0.59688

*

+

+

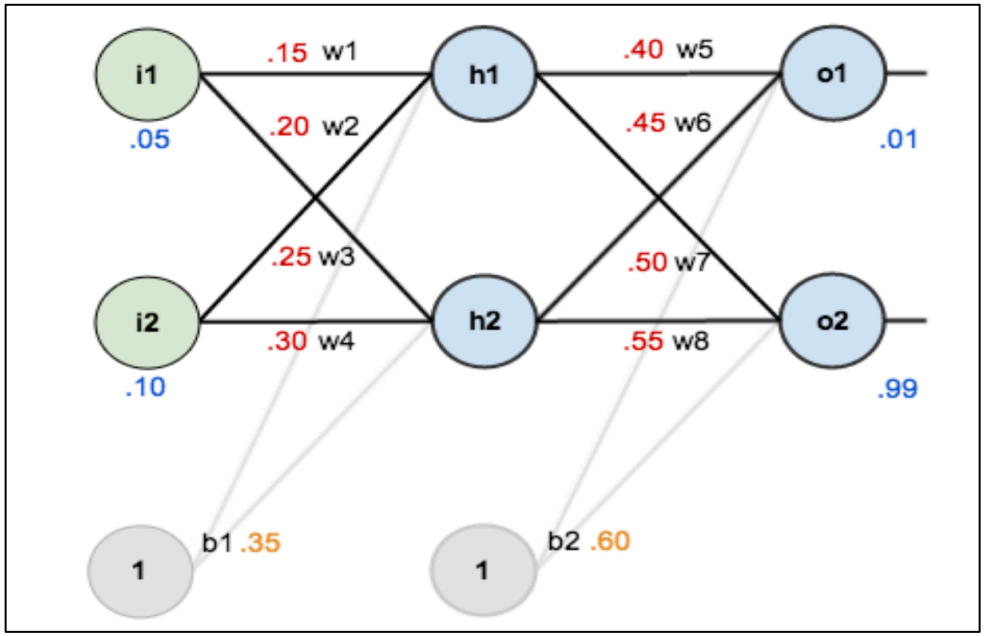
σ

o1 = 0.75136

b2=0.6

1

DNN Example (4)



w7=0.5

h1= 0.59327

*

w8=0.55

h2 = 0.59688

*

+

+

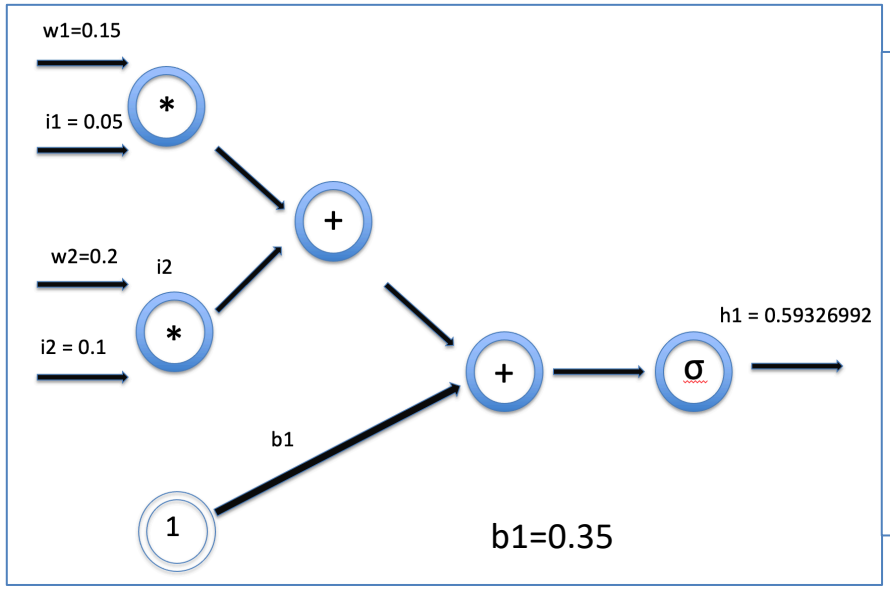
σ

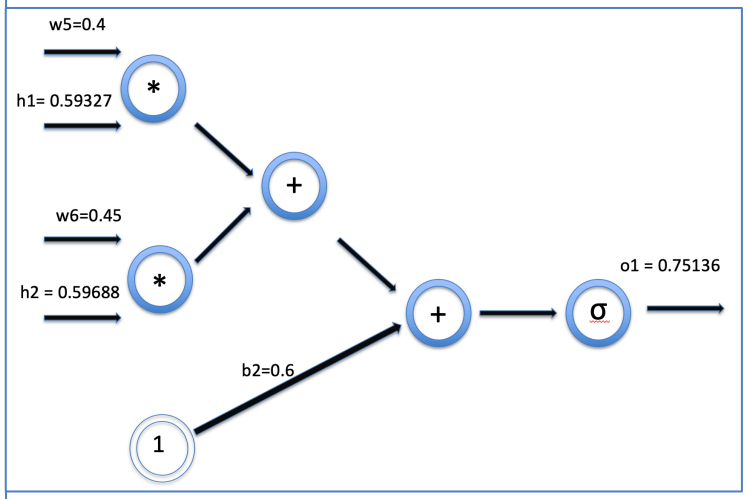
o2 = 0.77293

B2=0.6

1

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$





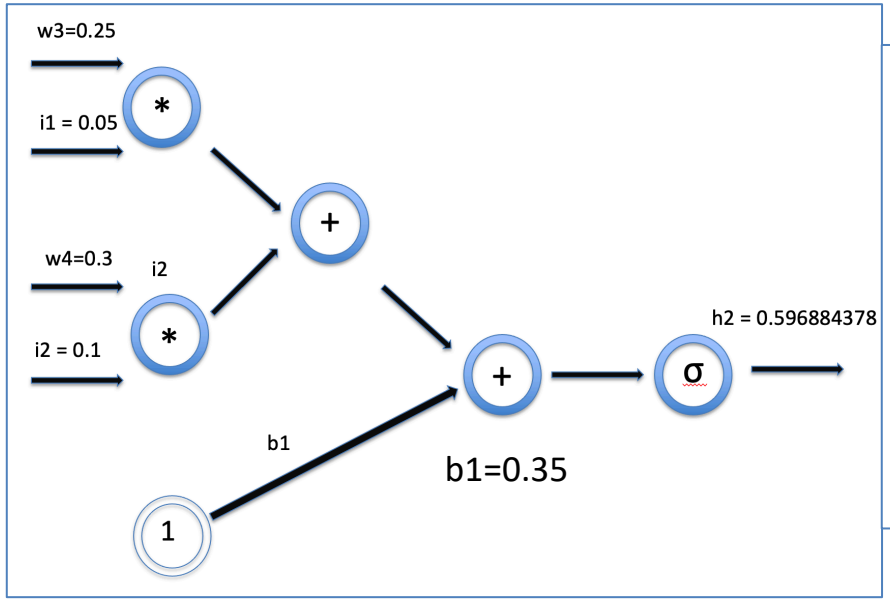
target=0.01

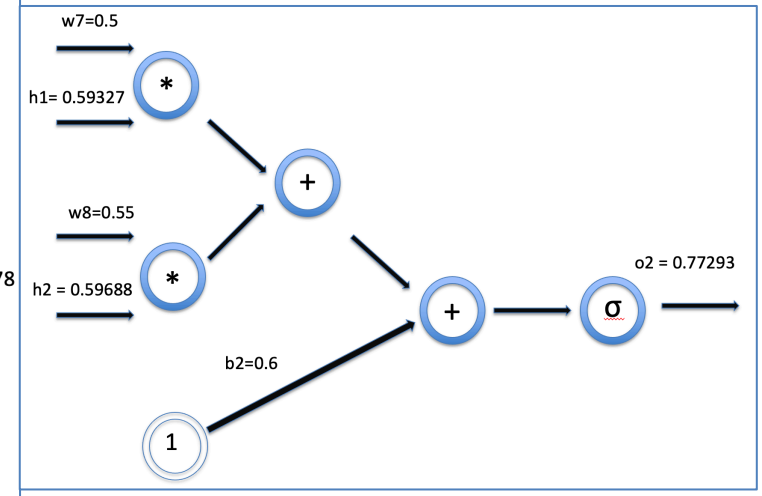
o1=0.75136

Eo1=0.27481

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Etotal = Eo1 + Eo2 = 0.298371109





Eo2=0.02356

o2=0.77293

target=0.99

$$E_{o2} = 0.023560026$$

Training a Neural Network

"Training a neural network"
What does it mean?
What does the network train?
What are the training parameters
How does it work?

Supervised Learning

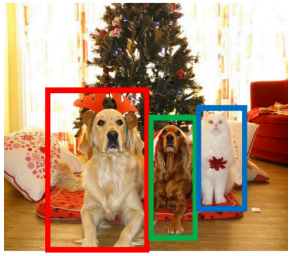
Supervised Learning Data:
(x, y) x is data, y is label

Goal:
Learn a function to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

CAT

Classification

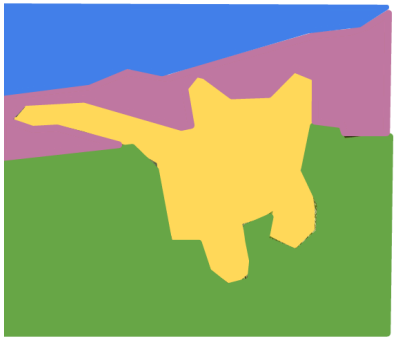


DOG, DOG, CAT

Object Detection

A cat sitting on a suitcase on the floor

Image captioning



GRASS, CAT, TREE, SKY

Semantic Segmentation

Simple MNIST MLP Neural Network

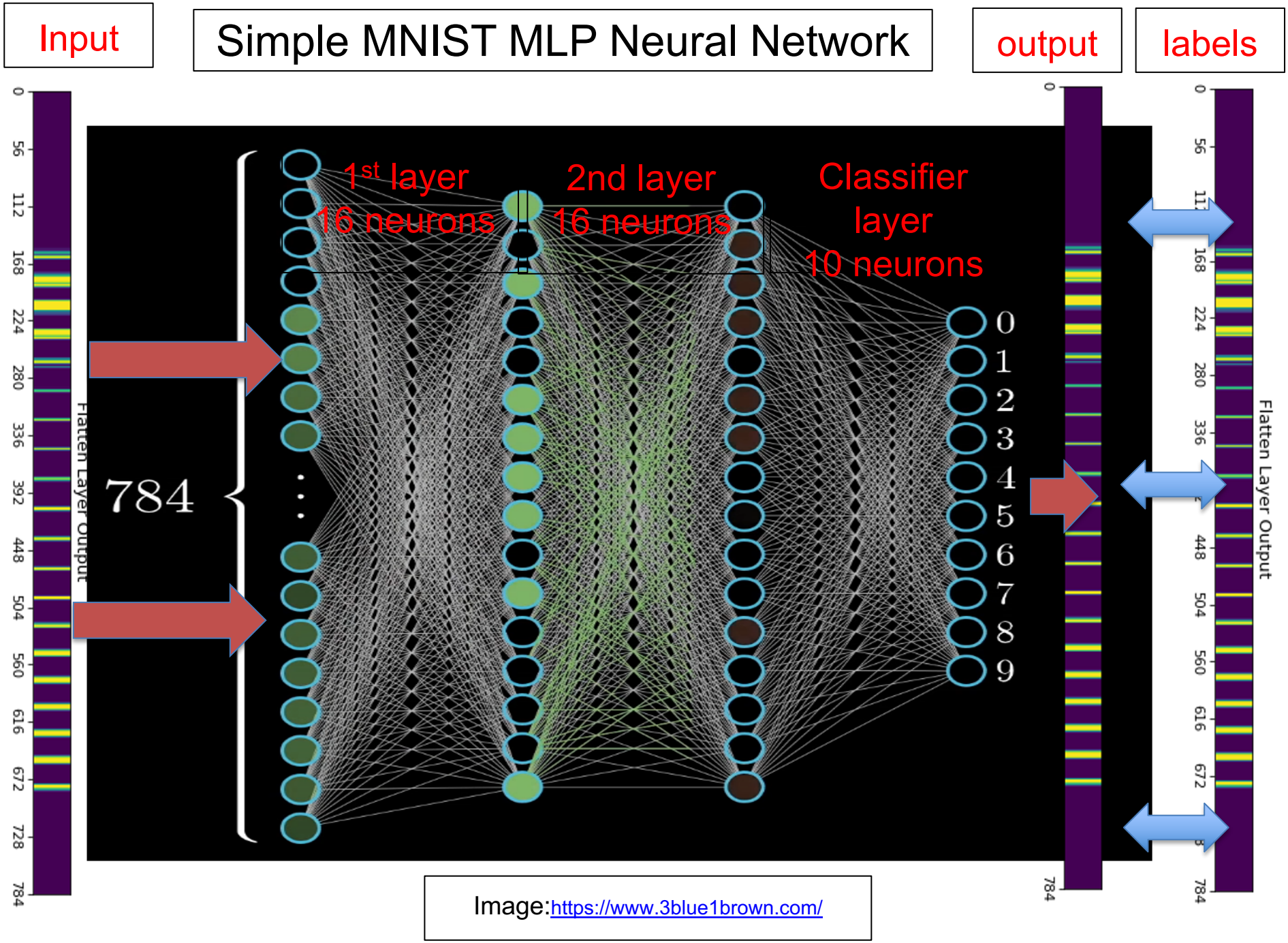


Image:https://www.3blue1brown.com/

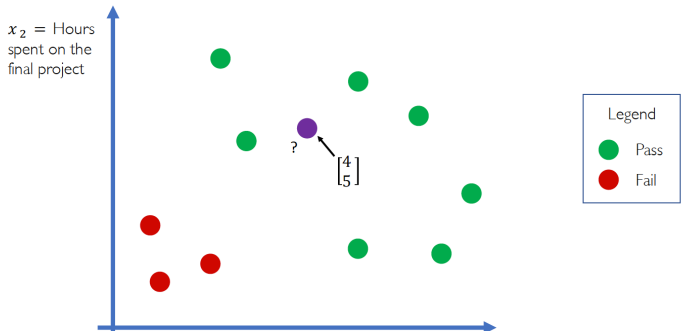
Training a Neural Network

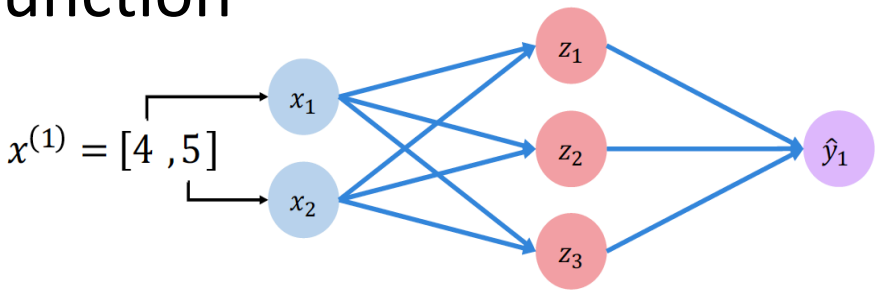
"Training a neural network"
What does it mean?
What does the network train?
What are the training parameters
How does it work?

- ✓ Neural network is composed of many unknown parameters (weights and bias), initialize them randomly in the beginning
- ✓ Perform forward path computations
- ✓ Computed results are compared to ground truth (labels) and errors are calculated → Need to have a cost (loss) function for error evaluation
- ✓ Use the error to adjust the weights and bias (backpropagation)
- ✓ Repeated the process (training) until the model is good enough

Loss Function

Example Problem: Will I pass this class?



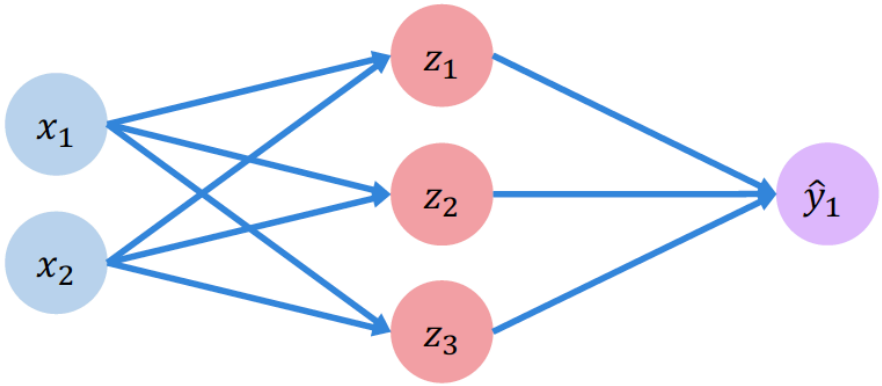


$$\mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

*The **empirical loss** measures the total loss over our entire dataset*

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$f(x) = \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \qquad y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

Also known as:
- Objective function
- Cost function
- Empirical Risk

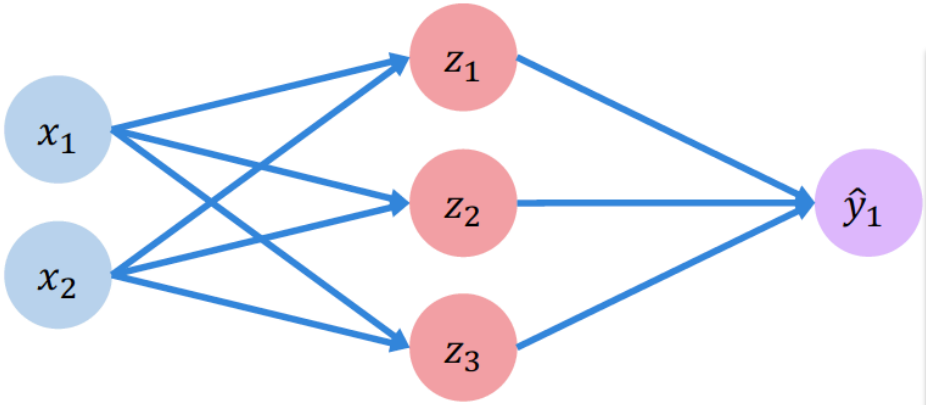
$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

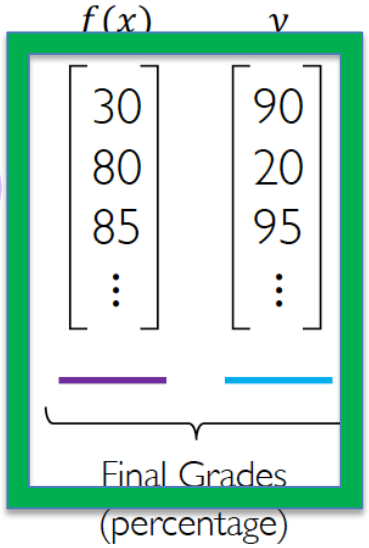
Predicted Actual

Mean Squared Error Loss

Mean squared error loss *can be used with regression models that output continuous real numbers*

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$





$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}; \boldsymbol{W}\right)\right)^2$$

Actual Predicted

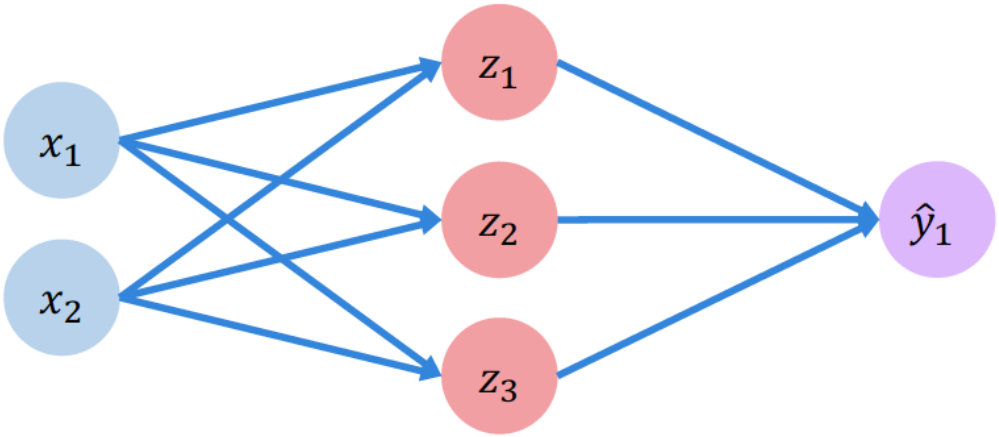
```
loss = tf.reduce_mean( tf.square(tf.subtract(model.y, model.pred) )
```

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Cross Entropy Loss

Cross entropy loss *can be used with models that output a probability between 0 and 1*

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



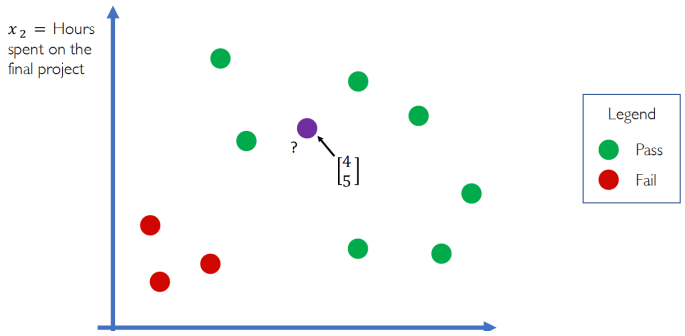
$$f(x) = \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \qquad y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

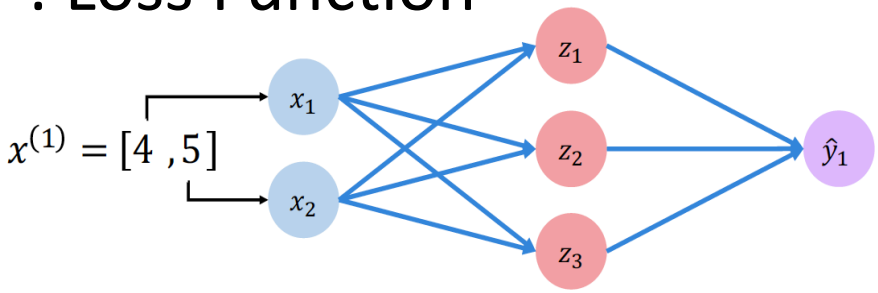
$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log\Big(\underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big) + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log\Big(1 - \underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big)$$

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(model.y, model.pred) )
```

Quantify Error : Loss Function

Example Problem: Will I pass this class?



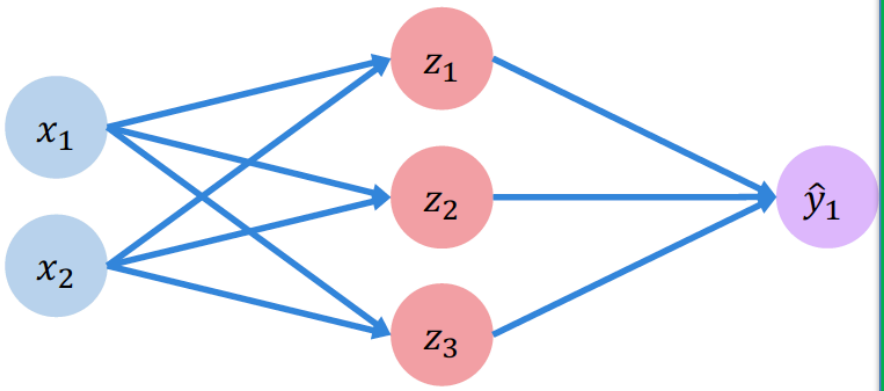


$$\mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

*The **empirical loss** measures the total loss over our entire dataset*

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$f(x)$ y

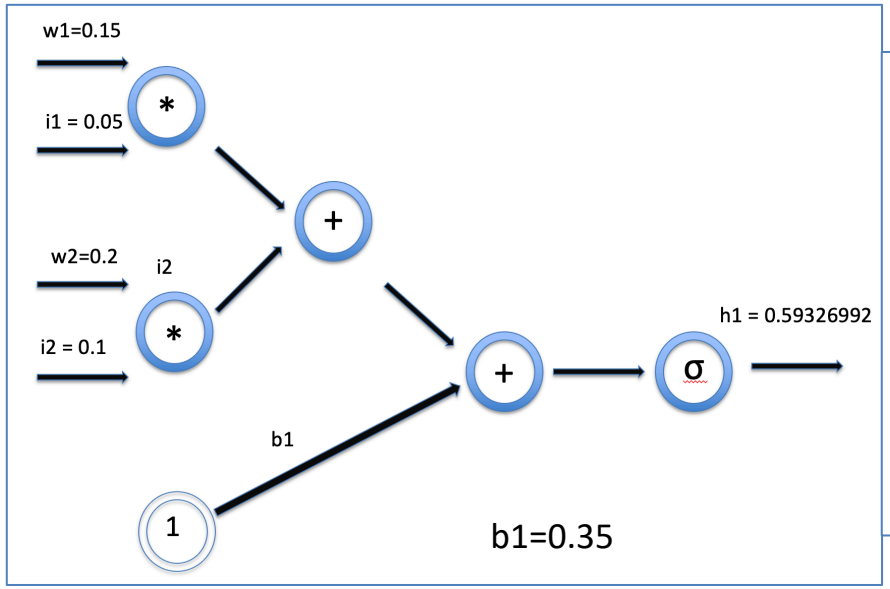
$$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \quad \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

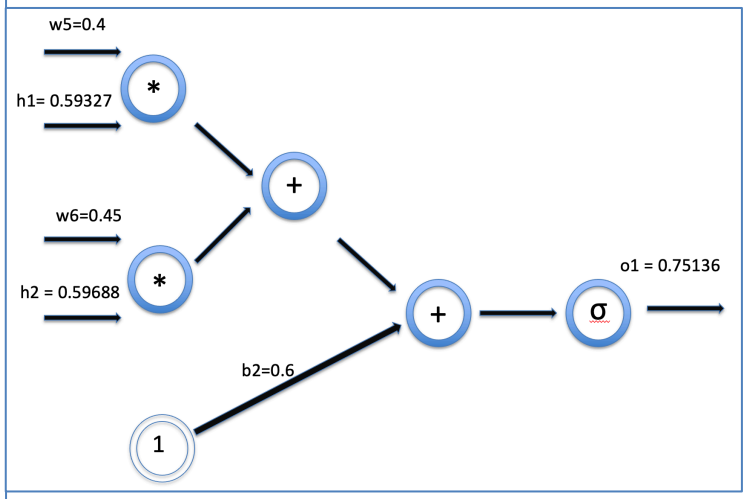
Also known as:
- Objective function
- Cost function
- Empirical Risk

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$





target=0.01

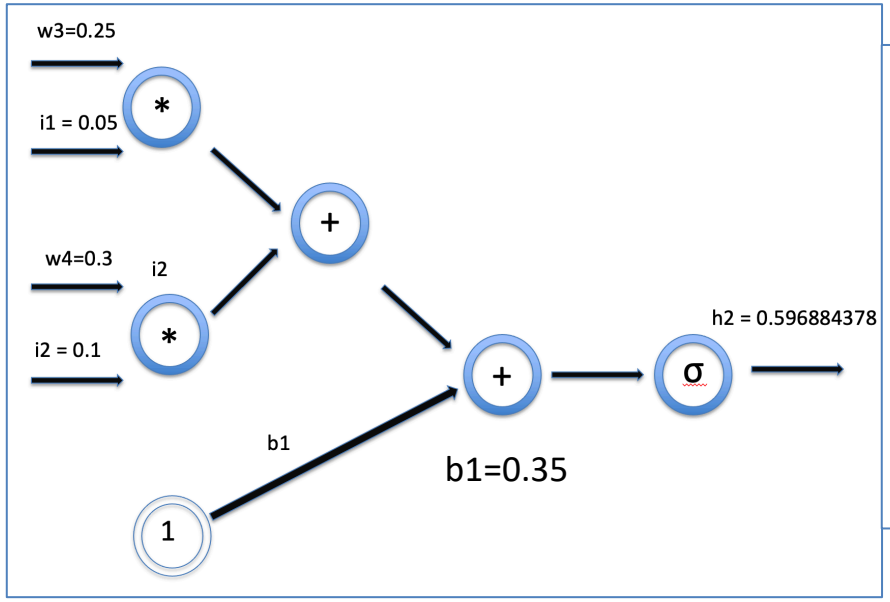
o1=0.75136

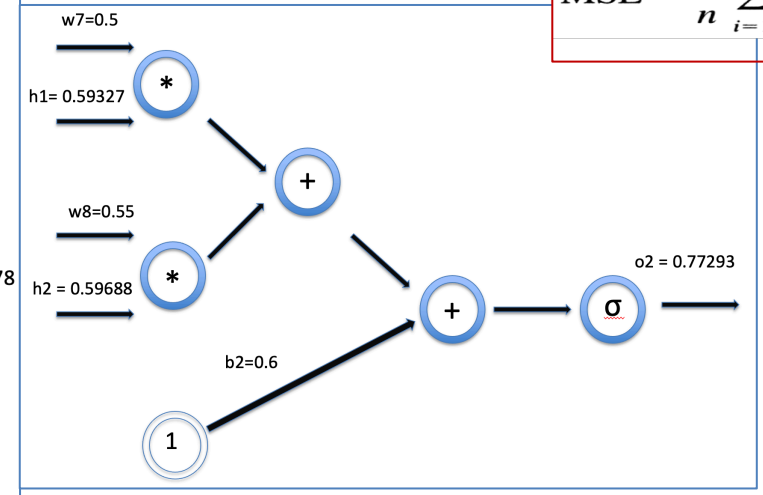
Eo1=0.27481

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Etotal = Eo1 + Eo2 = 0.298371109

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$





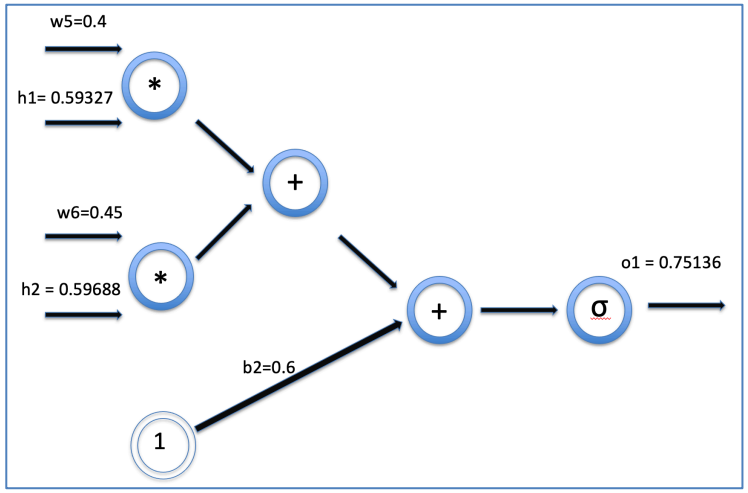
Eo2=0.02356

o2=0.77293

target=0.99

$$E_{o2} = 0.023560026$$

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$



target=0.01 | o1=0.75136 | Eo1=0.27481

Forward Path to Objective Function

target1 = 0.01

o1 = 0.75136

Cost f

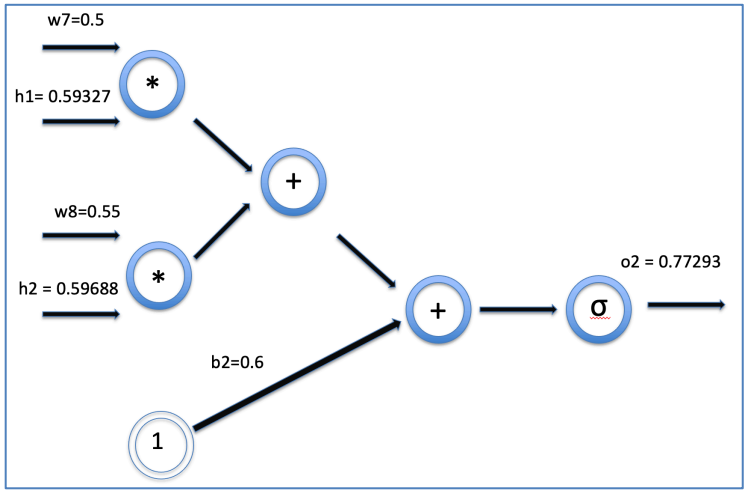
Eo1 = 0.27481

ET = 0.298371

grad ET = 1.0

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Etotal = Eo1 + Eo2 = 0.298371109



o2 = 0.77293

Cost f

Eo2 = 0.02356

target2 = 0.99

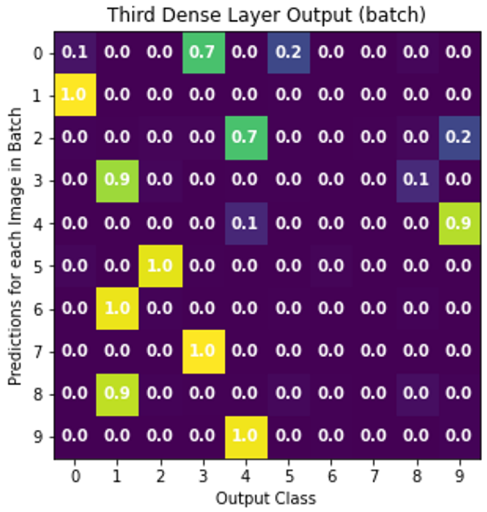
target=0.99 | o2=0.77293 | Eo2=0.02356

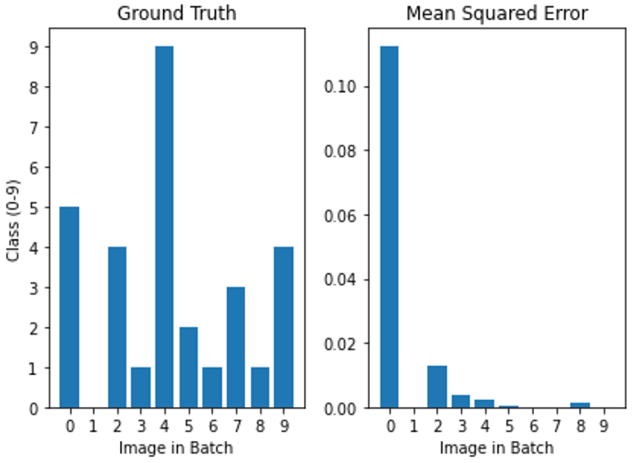
TensorFlow 2.0 : Forward Pass

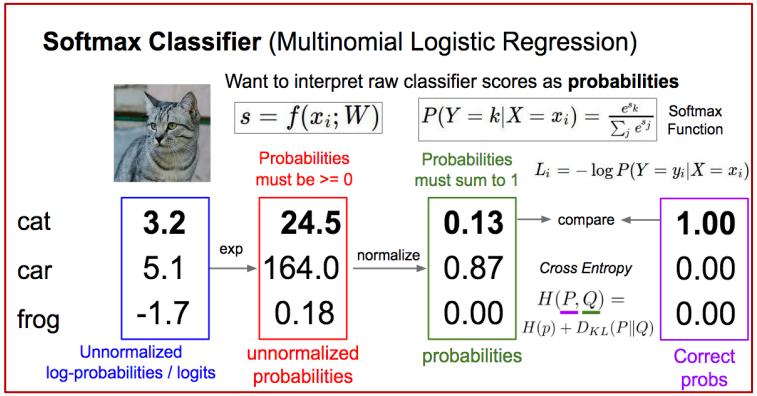
tf.losses.mean_squared_error(y_true_tf, y_pred_tf)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

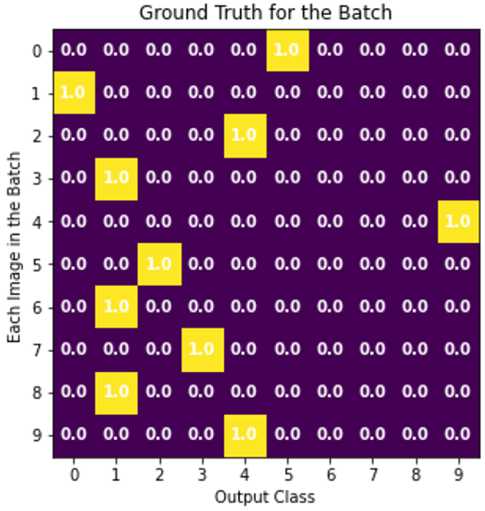
Evaluating the Error

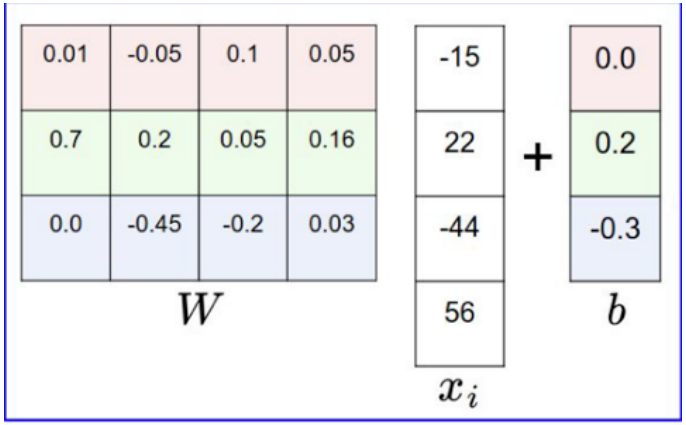


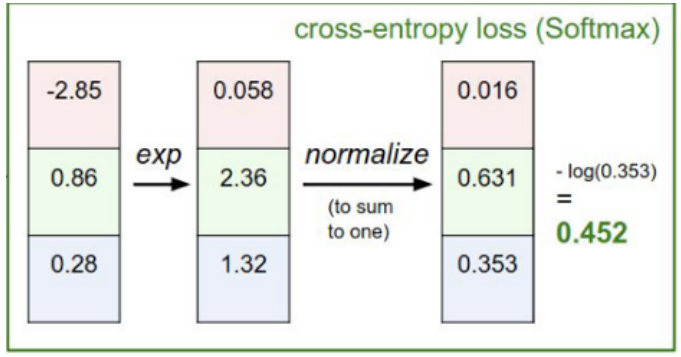




$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

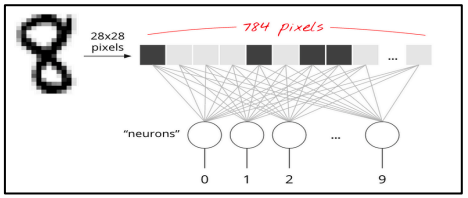




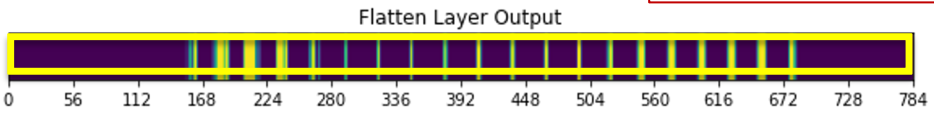


Summary : Flow of MLP Dense layer NN

1 image = a vector of 784



An image of 28x28 pixels



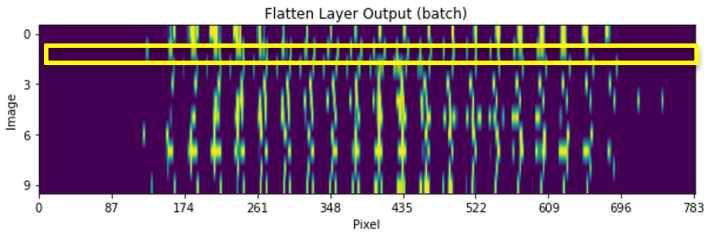
Input to 1st layer

10 images : batch of 10
10 x 784 matrix

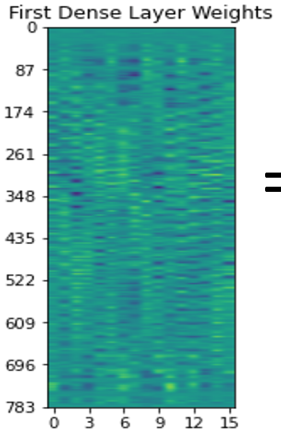
16 neurons
784 x 16 W matrix

10 x 16 output matrix

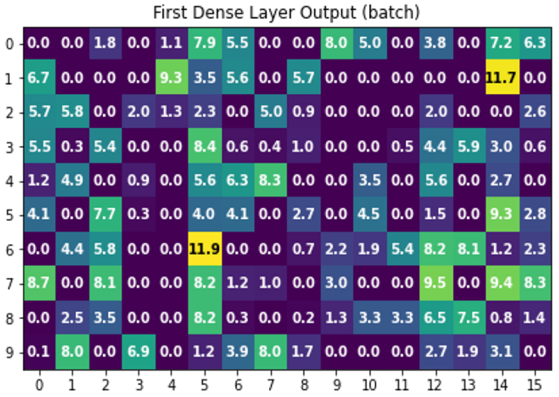
Input to 2nd layer



*

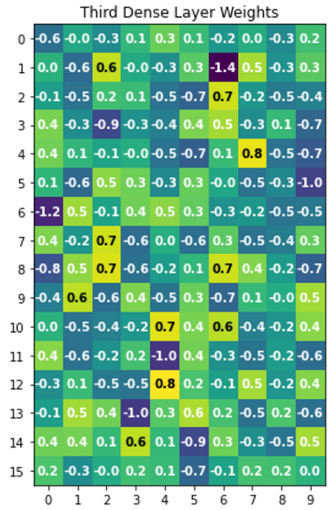


=



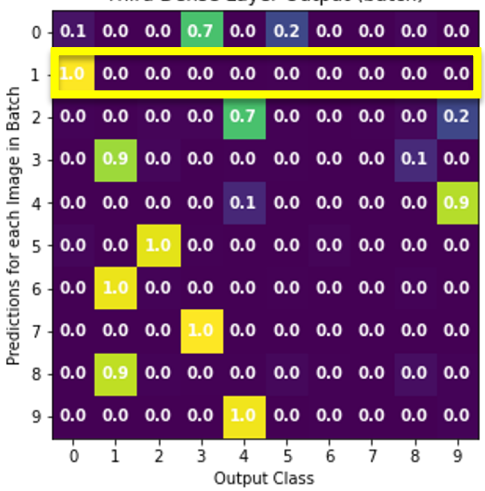
10 x 16 matrix

*

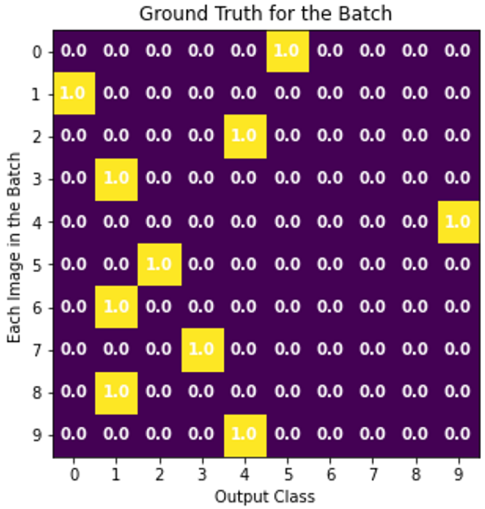


16 neurons
16 x 10
W matrix

=



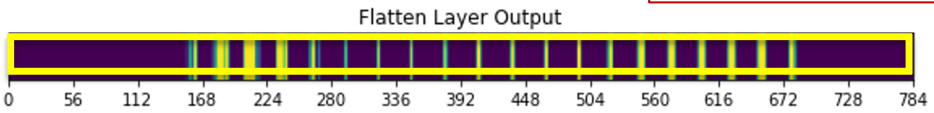
10 x 10 output matrix

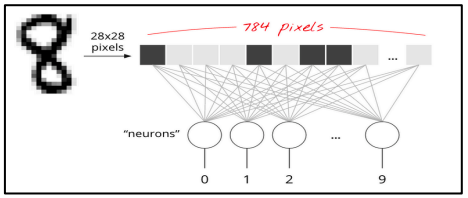


10 x 10 labeled matrix

Summary : Flow of MLP Dense layer NN

1 image = a vector of 784

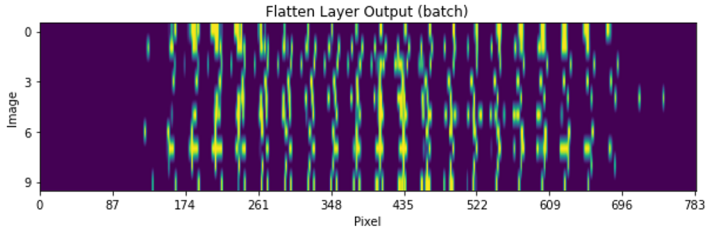




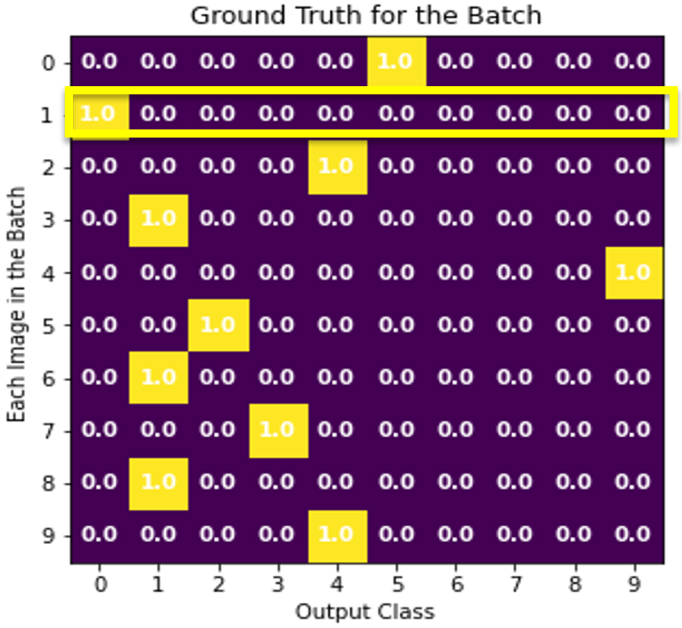
An image of 28x28 pixels

Input to 1st layer

10 images : batch of 10
10 x 784 matrix

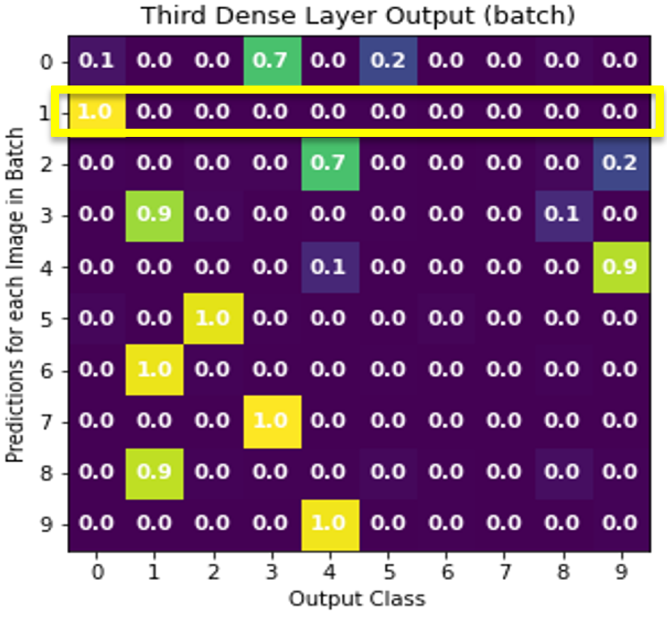


10 x 10 labeled matrix



Comparing Computed NN results To Labeled results (target)

10 x 10 output matrix



```
# e constant
e = 2.7182818284
# initial values
i1 = 0.05
i2 = 0.10
# initial weights
w1 = 0.15
w2 = 0.20
w3 = 0.25
w4 = 0.30
w5 = 0.40
w6 = 0.45
w7 = 0.50
w8 = 0.55
# bias
b1 = 0.35
b2 = 0.60
# targets
To1 = 0.01
To2 = 0.99
```

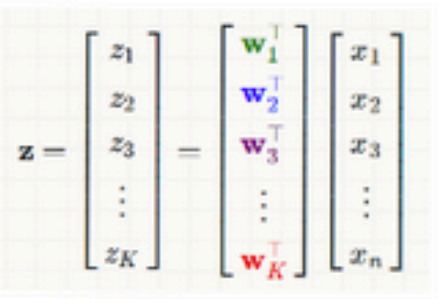
```
# forward propagation
h1 = 1/(1+e**(-(w1*i1 + w2*i2+b1)))
print("h1: " + str(h1))

h2 = 1/(1+e**(-(w3*i1 + w4*i2+b1)))
print("h2: " + str(h2))

o1 = 1/(1+e**(-(w5*h1 + w6*h2+b2)))
print("o1: " + str(o1))

o2 = 1/(1+e**(-(w7*h1 + w8*h2+b2)))
print("o2: " + str(o2))
```

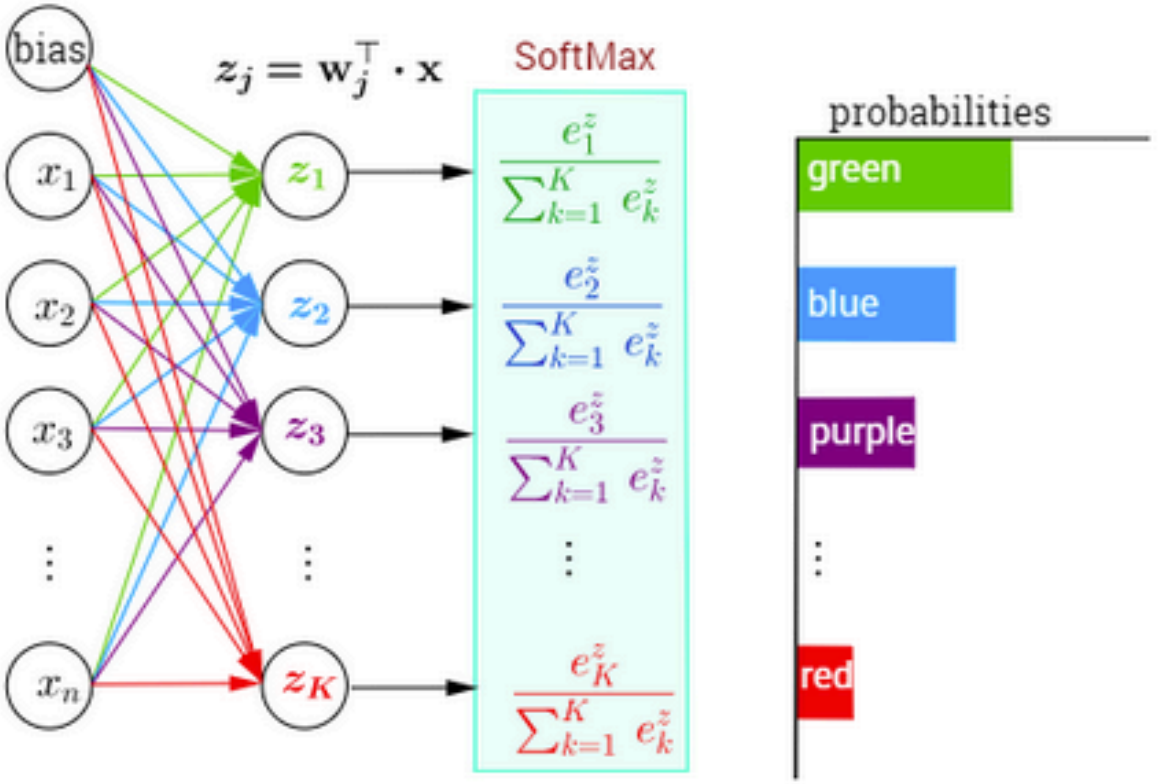
```
h1: 0.5932699921052087 h2: 0.5968843782577157
o1: 0.7513650695475076 o2: 0.772928465316421
```

```
# Error
Eo1 = 0.5*(To1-o1)**2
print("Error o1: " + str(Eo1))
Eo2 = 0.5*(To2-o2)**2
print("Error o2: " + str(Eo2))

E = Eo1 + Eo2
print("Total Error: " + str(E))
```

```
Error o1: 0.2748110831725904
Error o2: 0.023560025584942117
Total Error: 0.29837110875753253
```

Multi-Class Classification with NN and SoftMax Function



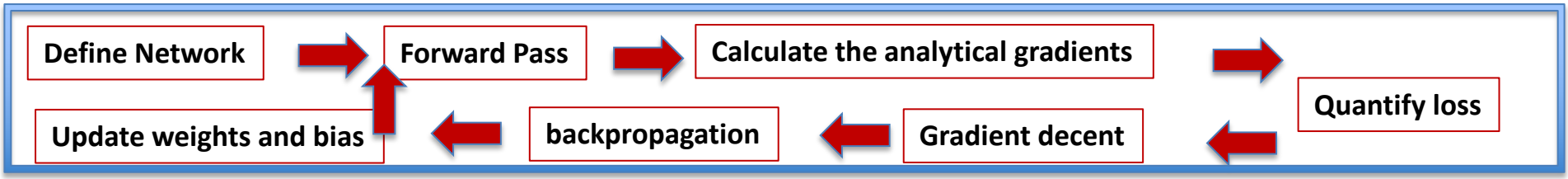


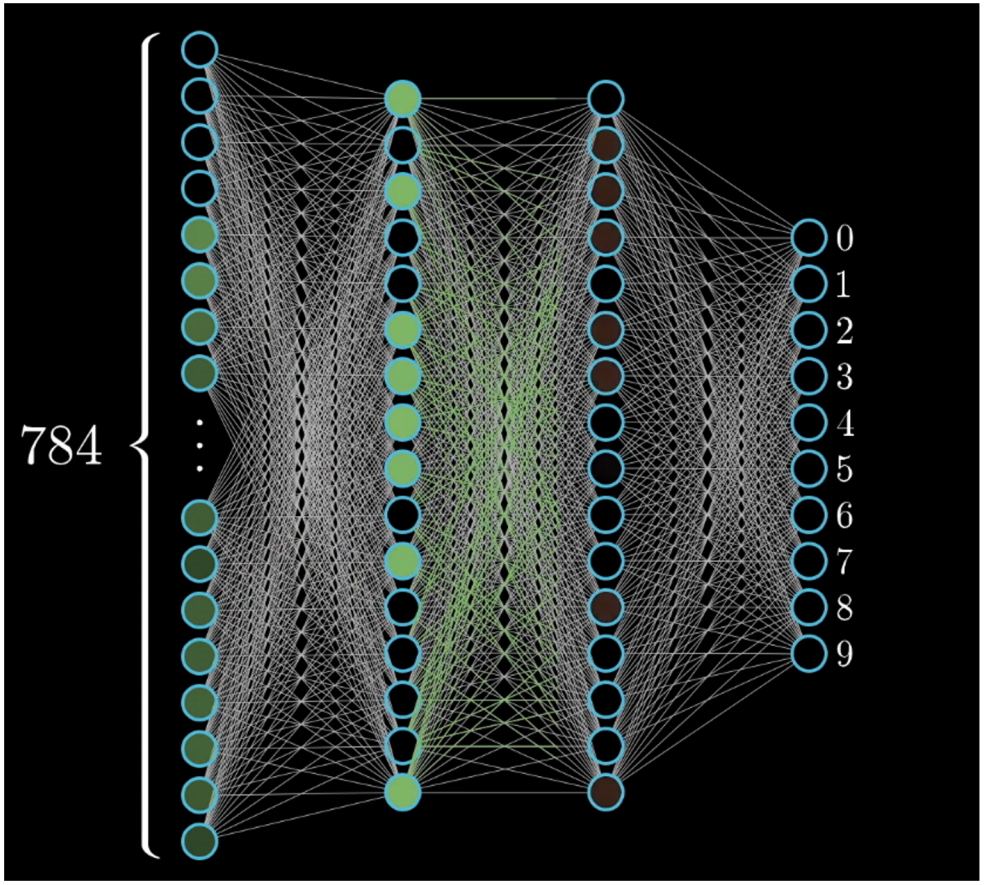
Softmax Function

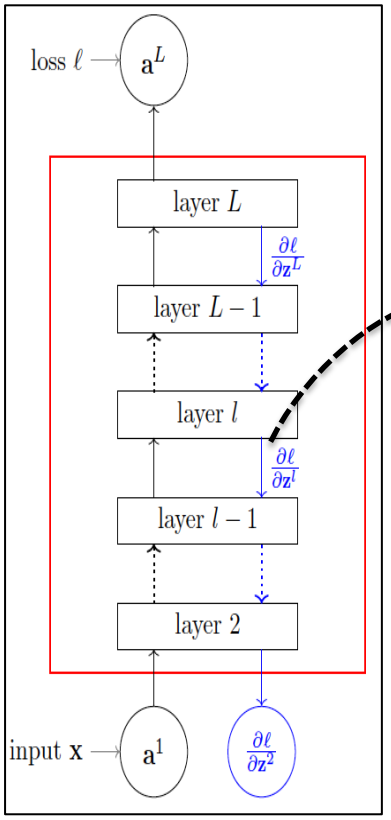
$$\sigma(j) = \frac{\exp(\mathbf{w}_j^\top \mathbf{x})}{\sum_{k=1}^K \exp(\mathbf{w}_k^\top \mathbf{x})} = \frac{\exp(z_j)}{\sum_{k=1}^K \exp(z_k)}$$

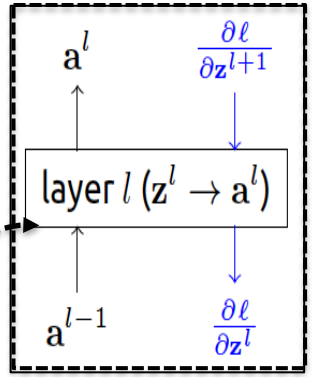
Summary : Flow of NN

```
model.fit(x_train, y_train, epochs=3, batch_size=10)
```





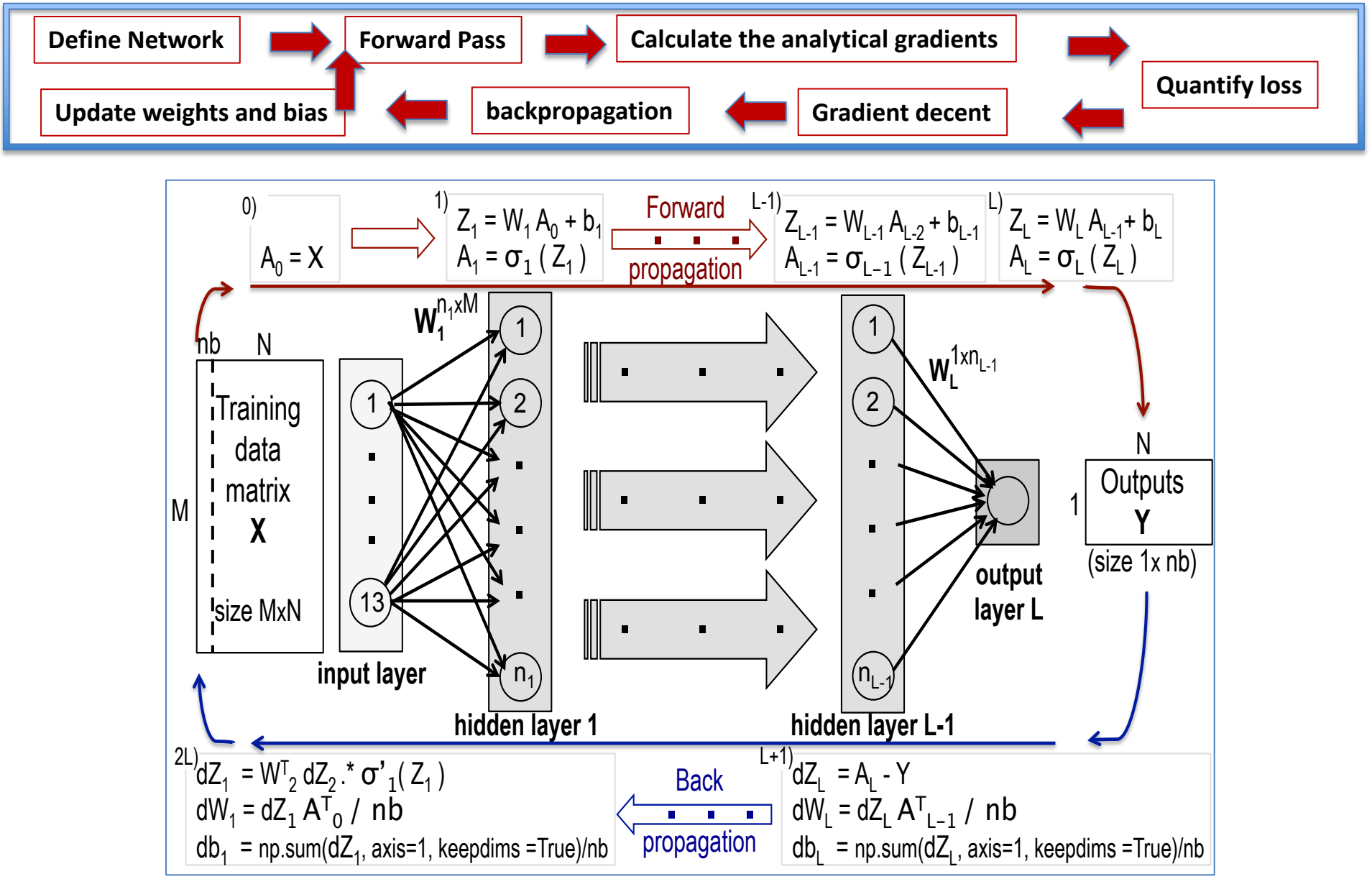




$$\frac{\partial \ell}{\partial z^l} = \frac{\partial \ell}{\partial z^{l+1}} \cdot \frac{\partial z^{l+1}}{\partial z^l} = \frac{\partial \ell}{\partial z^{l+1}} \cdot \frac{\partial z^{l+1}}{\partial a^l} \cdot \frac{\partial a^l}{\partial z^l} = \frac{\partial \ell}{\partial z^{l+1}} \cdot W^{l+1} \cdot \frac{\partial a^l}{\partial z^l}$$

Summary : Flow of NN

```
model.fit(x_train, y_train, epochs=3, batch_size=10)
```



TensorFlow 2.0

TensorFlow 1.X requires users to manually stitch together an abstract syntax tree (the graph) by making tf.* API calls.

Since these graphs are data structures, they can be saved, run, and restored all without the original Python code.

It then requires users to manually compile the abstract syntax tree by passing a set of output tensors and input tensors to a session.run() call.

TensorFlow 2.0 executes eagerly (like Python normally does) and in 2.0, graphs and sessions should feel like implementation details. This means TensorFlow operations are executed by Python, operation by operation, and returning results back to Python. It is also easy to debug. For some users, you may never need or want to leave Python. TF2.0 makes it more like numpy

Running TensorFlow op-by-op in Python prevents a host of accelerations otherwise available. If you can extract tensor computations from Python, you can make them into a *graph*.

A session.run() call is almost like a function call: You specify the inputs and the function to be called, and you get back a set of outputs.

In TensorFlow 2.0, you can decorate a Python function using tf.function() to mark it for JIT compilation so that TensorFlow runs it as a single graph (Functions 2.0 RFC).

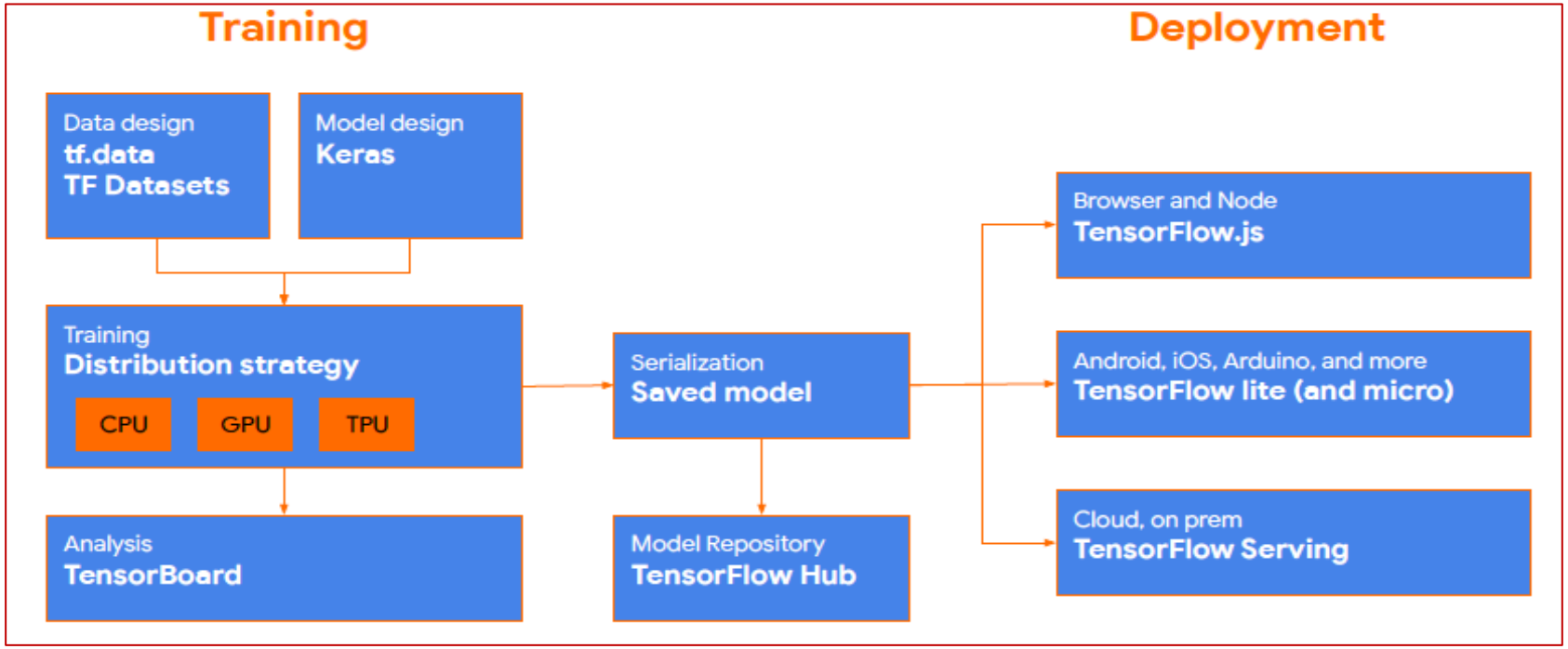
A common usage pattern in TensorFlow 1.X was the "kitchen sink" strategy, where the union of all possible computations was preemptively laid out, and then selected tensors were evaluated via session.run().

In TensorFlow 2.0, users should refactor their code into smaller functions that are called as needed. In general, it's not necessary to decorate each of these smaller functions with tf.function; only use tf.function to decorate high-level computations - for example, one step of training or the forward pass of your model.

TensorFlow 2.0 : model treatment

- ✓ Run like python
- ✓ Easier Tensor expression and reshape,
- ✓ Eager execution
- ✓ graph representation run on function,
- ✓ Use case for user
- ✓ Start with sequential API stack
- ✓ Scale up to function API or subclass
- ✓ Keros API for more
- ✓ Function API, DAG

- ✓ A wonderful thing about Deep Learning: ability to mix data types (text, Dense, images, time series, structured data) in a single model, treat Image to a vector and a question to a vector
- ✓ Run different application the same way in TF
- ✓ TF provides interfaces to other software platforms

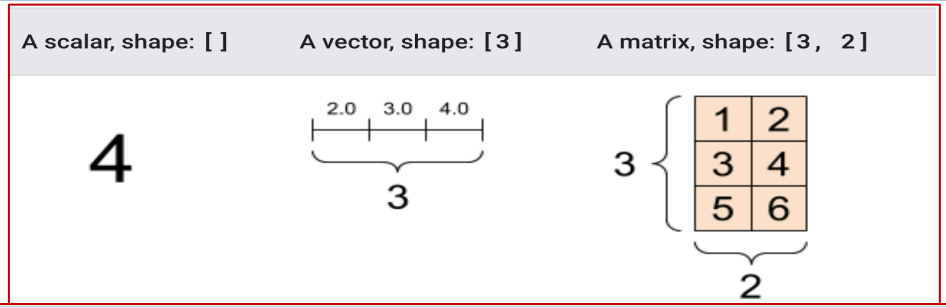


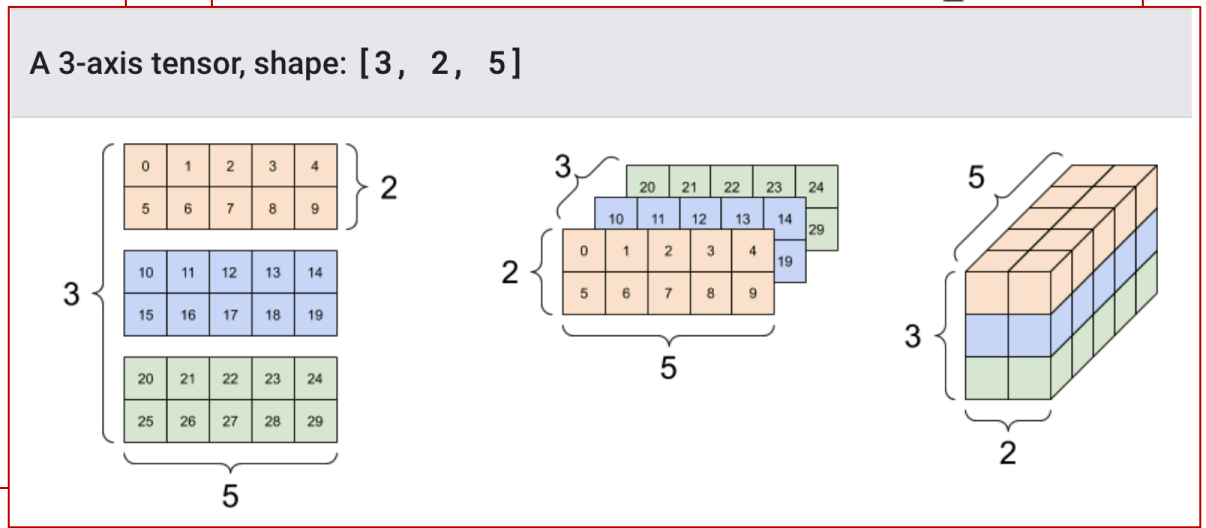
TensorFlow : Tensor

Tensors are multi-dimensional arrays with a uniform type (called a dtype).

```
import tensorflow as tf
import numpy as np
rank_0_tensor = tf.constant(4)
print(rank_0_tensor)

rank_1_tensor = tf.constant([2.0, 3.0, 4.0])
print(rank_1_tensor)

rank_2_tensor = tf.constant([[1, 2],
                             [3, 4],
                             [5, 6]], dtype=tf.float16)
print(rank_2_tensor)

rank_3_tensor = tf.constant([
  [[0, 1, 2, 3, 4],
   [5, 6, 7, 8, 9]],
  [[10, 11, 12, 13, 14],
   [15, 16, 17, 18, 19]],
  [[20, 21, 22, 23, 24],
   [25, 26, 27, 28, 29]],])
print(rank_3_tensor)
```

```
tf.Tensor(4, shape=(), dtype=int32)
tf.Tensor([2. 3. 4.], shape=(3,), dtype=float32)
tf.Tensor( [[1. 2.] [3. 4.] [5. 6.]], shape=(3, 2), dtype=float16)
tf.Tensor( [[[ 0 1 2 3 4] [ 5 6 7 8 9]] [[10 11 12 13 14] [15 16
17 18 19]] [[20 21 22 23 24] [25 26 27 28 29]]], shape=(3, 2,
5), dtype=int32)
```





tf.TensorShape

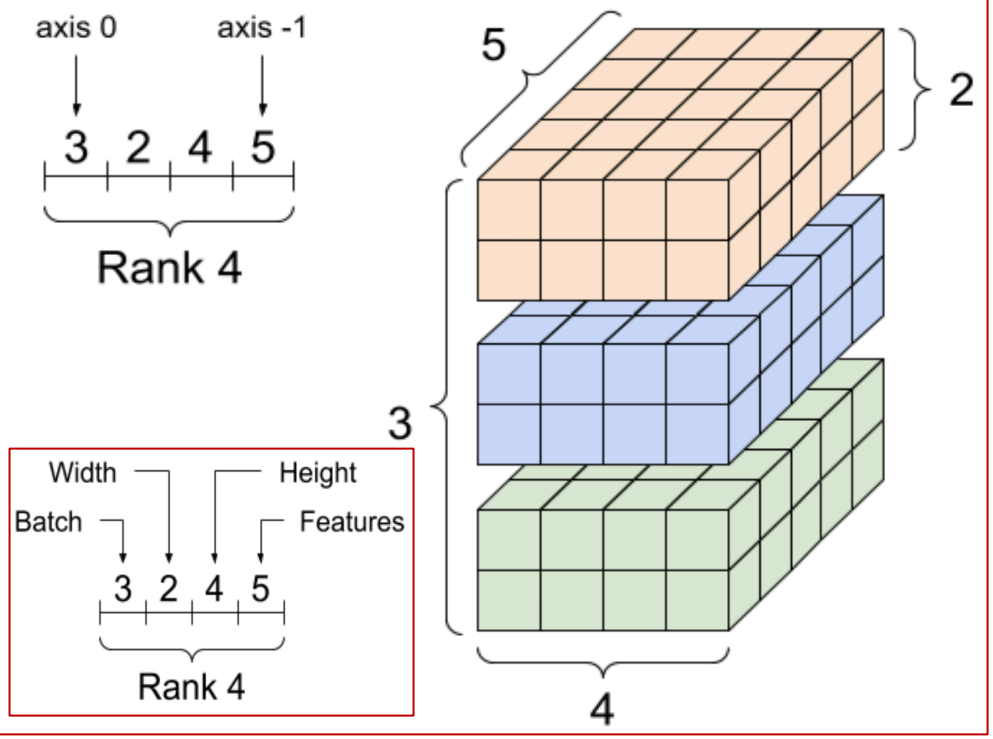
Joint Institute for Computational Sciences

INNOVATIVE COMPUTING LABORATORY

Tensors have shapes.

- ✓ **Shape: The length (number of elements) of each of the dimensions of a tensor.**
- ✓ **Rank: Number of tensor dimensions. A scalar has rank 0, a vector has rank 1, a matrix is rank 2.**
- ✓ **Axis or Dimension: A particular dimension of a tensor.**
- ✓ **Size: The total number of items in the tensor, the product shape vector**

A rank-4 tensor, shape: [3, 2, 4, 5]




```
print("Type of every element:",
      rank_4_tensor.dtype)
print("Number of dimensions:",
      rank_4_tensor.ndim)
print("Shape of tensor:",
      rank_4_tensor.shape)
print("Elements along axis 0 of tensor:",
      rank_4_tensor.shape[0])
print("Elements along the last axis of tensor:",
      rank_4_tensor.shape[-1])
print("Total number of elements (3*2*4*5): ",
      tf.size(rank_4_tensor).numpy())
```

```
rank_4_tensor = tf.zeros([3, 2, 4, 5])
```

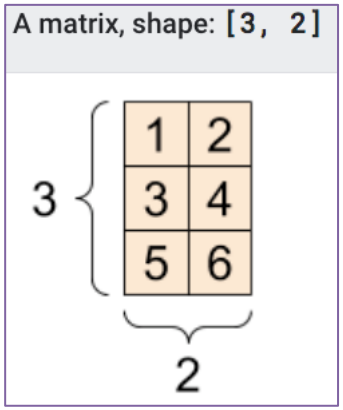
```
Type of every element: <dtype: 'float32'>
Number of dimensions: 4
Shape of tensor: (3, 2, 4, 5)
Elements along axis 0 of tensor: 3
Elements along the last axis of tensor: 5
Total number of elements (3*2*4*5): 120
```

Shape of Tensors

TensorFlow follows standard Python indexing rules, similar to indexing a list or a string in Python, and the basic rules for NumPy indexing.

- ✓ indexes start at 0
- ✓ negative indices count backwards from the end
- ✓ colons, :, are used for slices: start:stop:step

rank_1_tensor
[0 1 1 2 3 5 8 13 21 34]




```
print("Everything:", rank_1_tensor[:].numpy())
print("Before 4:", rank_1_tensor[:4].numpy())
print("From 4 to the end:", rank_1_tensor[4:].numpy())
print("From 2, before 7:", rank_1_tensor[2:7].numpy())
print("Every other item:", rank_1_tensor[::2].numpy())
print("Reversed:", rank_1_tensor[::-1].numpy())
```

```
Everything: [ 0 1 1 2 3 5 8 13 21 34]
Before 4: [0 1 1 2]
From 4 to the end: [ 3 5 8 13 21 34]
From 2, before 7: [1 2 3 5 8]
Every other item: [ 0 1 3 8 21]
Reversed: [34 21 13 8 5 3 2 1 1 0]
```

```
# Get row and column tensors
print("Second row:", rank_2_tensor[1, :].numpy())
print("Second column:", rank_2_tensor[:, 1].numpy())
print("Last row:", rank_2_tensor[-1, :].numpy())
print("First item in last column:", rank_2_tensor[0, -1].numpy())
print("Skip the first row:")
print(rank_2_tensor[1:, :].numpy(), "\n")
```

```
Second row: [3. 4.]
Second column: [2. 4. 6.]
Last row: [5. 6.]
First item in last column: 2.0
Skip the first row: [[3. 4.] [5. 6.]]
```

tf.Tensor and tf.reshape objects

Joint Institute for Computational Sciences

INNOVATIVE COMPUTING LABORATORY

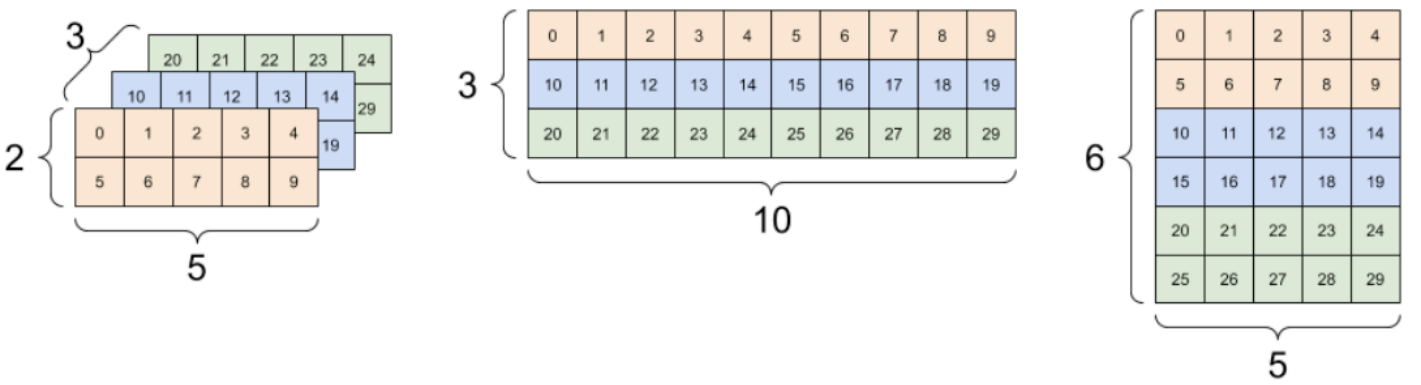
```
print(rank_3_tensor[:, :, 4])
```

```
tf.Tensor( [[ 4 9] [14 19] [24 29]],
shape=(3, 2), dtype=int32)
```

Selecting the last feature across all locations in each example in the batch


```
# A `-1` passed in the `shape` argument
says "Whatever fits", flatten a tensor
print(tf.reshape(rank_3_tensor, [-1]))
```

```
tf.Tensor( [ 0 1 2 3 4 5 6
7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29],
shape=(30,),
dtype=int32)
```




```
print(tf.reshape(rank_3_tensor, [3*2, 5]), "\n")
print(tf.reshape(rank_3_tensor, [3, -1]))
```

```
tf.Tensor( [[ 0 1 2 3 4] [ 5 6 7 8 9] [10 11 12 13 14]
[15 16 17 18 19] [20 21 22 23 24] [25 26 27 28 29]],
shape=(6, 5), dtype=int32)
tf.Tensor( [[ 0 1 2 3 4 5 6 7 8 9] [10 11 12 13 14 15
16 17 18 19] [20 21 22 23 24 25 26 27 28 29]],
shape=(3, 10), dtype=int32)
```

Tensors math operation : NLA

Joint Institute for Computational Sciences

INNOVATIVE COMPUTING LABORATORY

The base tf.Tensor class requires tensors to be "rectangular"---that is, along each axis, every element is the same size. However, Ragged tensors, Sparse tensors can handle different shapes.

addition, element-wise multiplication, and matrix multiplication.

```
a = tf.constant([[1, 2], [3, 4]])
b = tf.constant([[1, 1], [1, 1]])

print(tf.add(a, b), "\n")
print(tf.multiply(a, b), "\n")
print(tf.matmul(a, b), "\n")

print(a + b, "\n") # element-wise addition
print(a * b, "\n") # element-wise multiplication
print(a @ b, "\n") # matrix multiplication

c = tf.constant([[4.0, 5.0], [10.0, 1.0]])

# Find the largest value
print(tf.reduce_max(c))
# Find the index of the largest value
print(tf.argmax(c))
# Compute the softmax
print(tf.nn.softmax(c))
```

```
tf.Tensor( [[2 3]
            [4 5]], shape=(2, 2), dtype=int32)
tf.Tensor( [[1 2]
            [3 4]], shape=(2, 2), dtype=int32)
tf.Tensor( [[3 3]
            [7 7]], shape=(2, 2), dtype=int32)
```

```
tf.Tensor( [[2 3] [4 5]], shape=(2, 2), dtype=int32)
tf.Tensor( [[1 2] [3 4]], shape=(2, 2), dtype=int32)
tf.Tensor( [[3 3] [7 7]], shape=(2, 2), dtype=int32)
```

```
tf.Tensor(10.0, shape=(), dtype=float32)
tf.Tensor([1 0], shape=(2,), dtype=int64)
tf.Tensor( [[2.6894143e-01 7.3105860e-01]
[9.9987662e-01 1.2339458e-04]],
           shape=(2, 2), dtype=float32)
```

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Softmax

$$\sigma(\vec{z})_1 = \sigma(3) = \frac{e^3}{e^3 + e^0} = 0.953$$

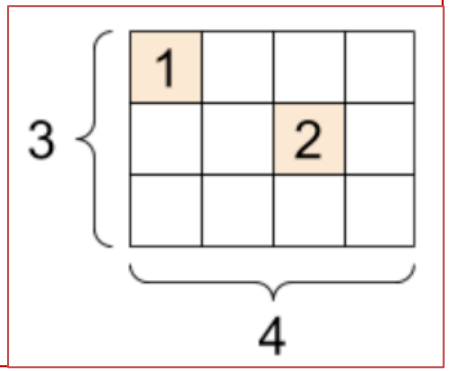
$$\sigma(\vec{z})_2 = \sigma(0) = \frac{e^0}{e^3 + e^0} = 0.0474$$

Sigmoid

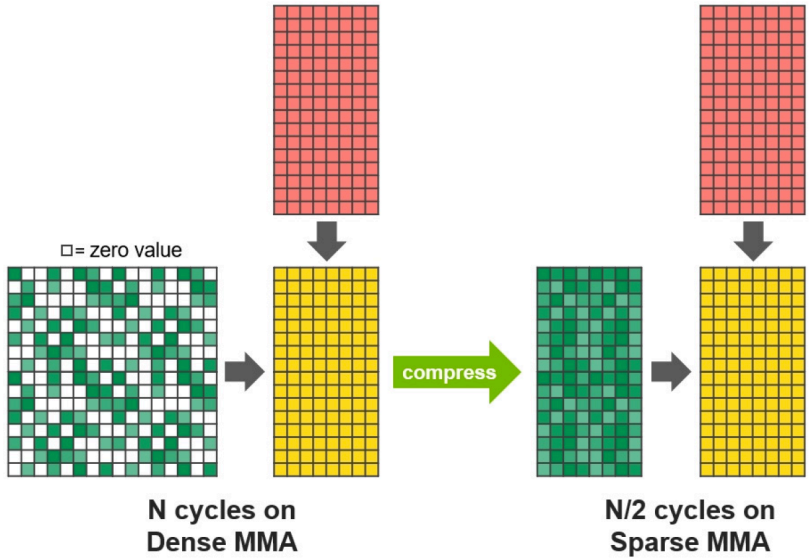
$$S(x) = S(3) = \frac{1}{1 + e^{-3}} = \frac{1}{1 + e^{-3}} = 0.953$$

Sparse Tensors

```
# Sparse tensors store values by index in a memory-efficient manner
sparse_tensor = tf.sparse.SparseTensor(indices=[[0, 0], [1, 2]],
                                       values=[1, 2],
                                       dense_shape=[3, 4])
print(sparse_tensor, "\n")

# We can convert sparse tensors to dense
print(tf.sparse.to_dense(sparse_tensor))
```




```
SparseTensor(indices=tf.Tensor( [[0 0] [1 2]],
shape=(2, 2), dtype=int64),
values=tf.Tensor([1 2], shape=(2,),
dtype=int32), dense_shape=tf.Tensor([3 4],
shape=(2,), dtype=int64))

tf.Tensor( [[1 0 0 0] [0 0 2 0] [0 0 0 0]],
shape=(3, 4), dtype=int32)
```



Example Dense MMA and Sparse MMA operations using 16x16 sparse matrix (Matrix A), multiplied by a dense 16x8 matrix (Matrix B). Sparse MMA operation on right doubles throughput by skipping compute of zero values

Figure 13. Example Dense MMA and Sparse MMA operations

TensorFlow tf.Variable objects

```
my_tensor = tf.constant([[1.0, 2.0], [3.0, 4.0]])
my_variable = tf.Variable(my_tensor)

# Variables can be all kinds of types, just like tensors
bool_variable = tf.Variable([False, False, False, True])
complex_variable = tf.Variable([5 + 4j, 6 + 1j])
```

```
print("A variable:",my_variable)
print("\nViewed as a tensor:",
tf.convert_to_tensor(my_variable))
print("\nIndex of highest value:", tf.argmax(my_variable))

# This creates a new tensor; it does not reshape the variable.
print("\nCopying and reshaping: ", tf.reshape(my_variable,
([1,4])))
```

```
A variable: <tf.Variable 'Variable:0' shape=(2, 2) dtype=float32,
numpy= array([[1., 2.], [3., 4.]], dtype=float32)>
Viewed as a tensor: tf.Tensor( [[1. 2.] [3. 4.]], shape=(2, 2),
dtype=float32) Index of highest value: tf.Tensor([1 1], shape=(2,),
dtype=int64)
Copying and reshaping: tf.Tensor([[1. 2. 3. 4.]], shape=(1, 4),
dtype=float32)
```

```
a = tf.Variable([2.0, 3.0])
# Create b based on the value of a
b = tf.Variable(a)
a.assign([5, 6])

# a and b are different
print(a.numpy())
print(b.numpy())

# There are other versions of assign
print(a.assign_add([2,3]).numpy())  # [7. 9.]
print(a.assign_sub([7,9]).numpy())  # [0. 0.]
```

```
[5.  6.]
[2.  3.]
[7.  9.]
[0.  0.]
```

```
with tf.device('CPU:0'):
  a = tf.Variable([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
  b = tf.Variable([[1.0, 2.0, 3.0]])

with tf.device('GPU:0'):
  # Element-wise multiply
  k = a * b

print(k)
```

```
tf.Tensor( [[ 1. 4. 9.] [ 4. 10. 18.]], shape=(2, 3), dtype=float32)
```

TensorFlow Gradient objects

Joint Institute for Computational Sciences

INNOVATIVE COMPUTING LABORATORY

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

TensorFlow provides the tf.GradientTape API for automatic differentiation; that is, computing the gradient of a computation with respect to some inputs, usually tf.Variables. TensorFlow "records" relevant operations executed inside the context of a tf.GradientTape onto a "tape". TensorFlow then uses that tape to compute the gradients of a "recorded" computation using reverse mode differentiation.

```
x = tf.Variable(3.0)
with tf.GradientTape() as tape:
  y = x**2
```

```
# dy = 2x * dx
dy_dx = tape.gradient(y, x)
dy_dx.numpy()
6
```

In most cases, you will want to calculate gradients with respect to a model's trainable variables. Since all subclasses of tf.Module aggregate their variables in the Module.trainable_variables property, you can calculate these gradients in a few lines of code:

```
w = tf.Variable(tf.random.normal((3, 2)), name='w')
b = tf.Variable(tf.zeros(2, dtype=tf.float32), name='b')
x = [[1., 2., 3.]]

with tf.GradientTape(persistent=True) as tape:
  y = x @ w + b
  loss = tf.reduce_mean(y**2)
```

```
[dl_dw, dl_db] = tape.gradient(loss, [w, b])
```

```
layer = tf.keras.layers.Dense(2, activation='relu')
x = tf.constant([[1., 2., 3.]])

with tf.GradientTape() as tape:
  # Forward pass
  y = layer(x)
  loss = tf.reduce_mean(y**2)

# Calculate gradients with respect to every trainable variable
grad = tape.gradient(loss, layer.trainable_variables)
```

TensorFlow Graph objects : tf.function

- ✓ TensorFlow graph can be used in environments that don't have a Python interpreter.
- ✓ TensorFlow uses graphs as the format for saved models when it exports them from Python.
- ✓ Graphs are extremely useful and let your TensorFlow run **fast**, run **in parallel**, and run efficiently **on multiple devices**. Graphs are also easily optimized. There is an entire optimization system, Grappler, to perform this and other speedups.
- ✓ Define your machine learning models in Python for convenience, and then automatically construct graphs when you need them.
- ✓ The way you create a graph in TensorFlow is to use tf.function, either as a direct call or as a decorator.

```
import tensorflow as tf
import timeit
from datetime import datetime
```

```
# Define a Python function
def function_to_get_faster(x, y, b):
  x = tf.matmul(x, y)
  x = x + b
  return x

# Create a `Function` object that contains a graph
a_function_that_uses_a_graph =
    tf.function(function_to_get_faster)

# Make some tensors
x1 = tf.constant([[1.0, 2.0]])
y1 = tf.constant([[2.0], [3.0]])
b1 = tf.constant(4.0)
# It just works!
a_function_that_uses_a_graph(x1, y1, b1).numpy()
```

```
# Create an oveerride model to classify pictures
class SequentialModel(tf.keras.Model):
  def __init__(self, **kwargs):
    super(SequentialModel, self).__init__(**kwargs)
    self.flatten = tf.keras.layers.Flatten(input_shape=(28, 28))
    self.dense_1 = tf.keras.layers.Dense(128, activation="relu")
    self.dropout = tf.keras.layers.Dropout(0.2)
    self.dense_2 = tf.keras.layers.Dense(10)

  def call(self, x):
    x = self.flatten(x)
    x = self.dense_1(x)
    x = self.dropout(x)
    x = self.dense_2(x)
    return x

input_data = tf.random.uniform([60, 28, 28])

eager_model = SequentialModel()
graph_model = tf.function(eager_model)

print("Eager time:", timeit.timeit(lambda: eager_model(input_data), number=10000))
print("Graph time:", timeit.timeit(lambda: graph_model(input_data), number=10000))
```

```
Eager time: 4.800515108999889
Graph time: 2.0497353999999177
```

Acknowledgements and References

This portion of the tutorial contains many extracted materials from many online websites and courses. Listed below are the major sites. This portion of the materials is not intended for public distribution. Please visit the sites websites for detail contents. If you are beginner users of DNN, I suggest to read the following list of websites in its order.

1) http://neuralnetworksanddeeplearning.com

2) https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist

3) https://www.deeplearning.ai/deep-learning-specialization/

4) http://cs231n.stanford.edu/

5) MIT 6.S191, https://www.youtube.com/watch?v=njKP3FqW3Sk

6) https://livebook.manning.com/book/deep-learning-with-python/about-this-book/

7) https://www.fast.ai/

8) https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/

9) https://www.nersc.gov/users/training/gpus-for-science/gpus-for-science-2020/

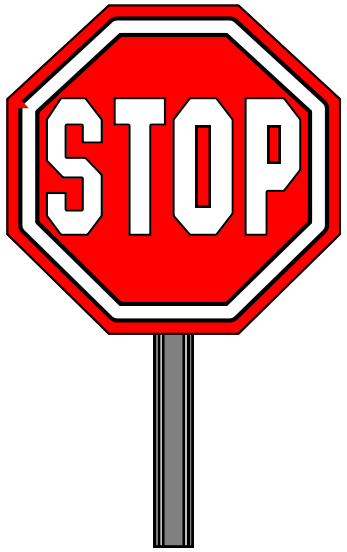
10) https://oneapi-src.github.io/oneDNN/

11) http://www.cs.cornell.edu/courses/cs4787/2020sp/

12) More sites and free books are listed in www.jics.utk.edu/actia → ML

13) https://machinelearningmastery.com

The End



- The End!