Lecture Series on Hardware for Deep Learning

Part 1: Introduction to Deep Learning

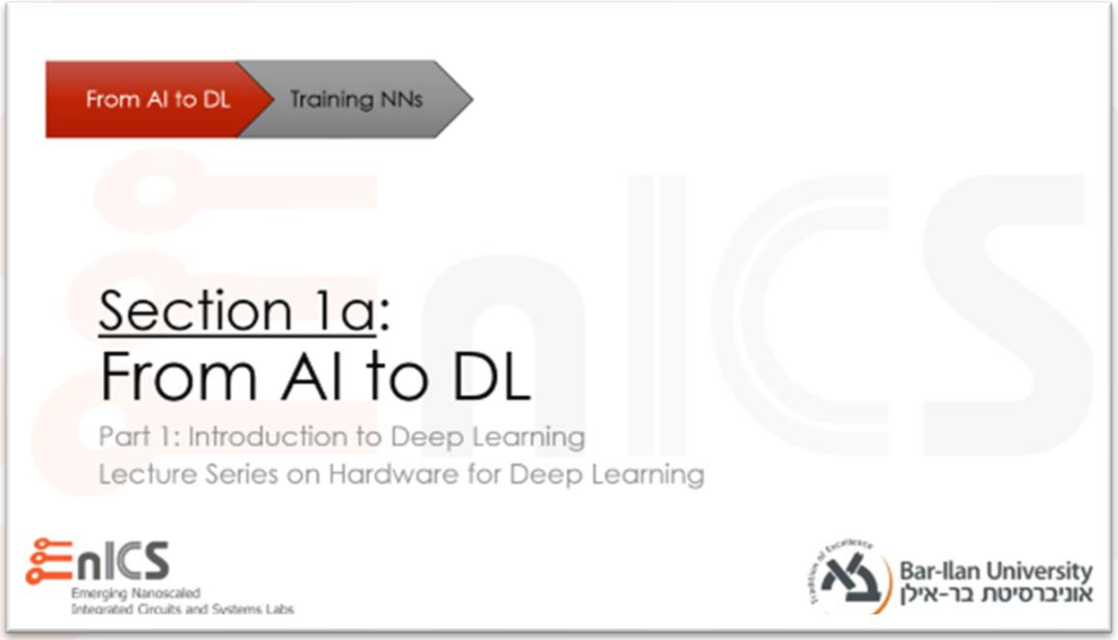
Dr. Adam Teman
EnICS Labs, Bar-Ilan University

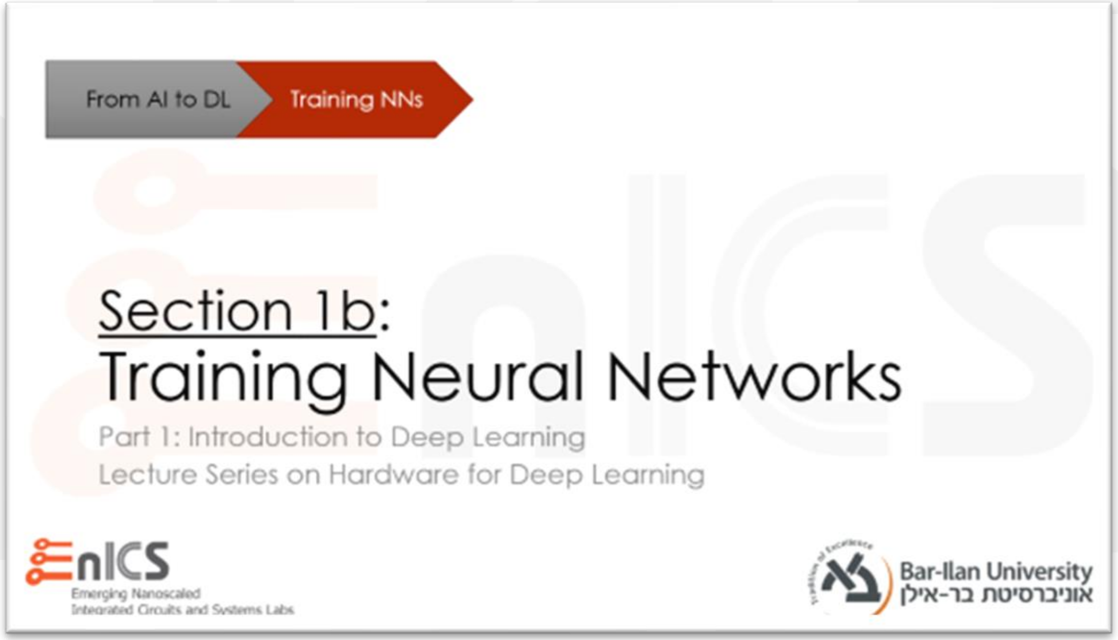
15 March 2020

EnICS
Emerging Nanoscaled Integrated Circuits and Systems Labs

Tradition of Excellence
Bar-Ilan University
אוניברסיטת בר-אילן

Outline


From AI to DL
Training NNs
Section 1a:
From AI to DL
Part 1: Introduction to Deep Learning
Lecture Series on Hardware for Deep Learning
EnICS
Emerging Nanoscaled
Integrated Circuits and Systems Labs
Bar-Ilan University
אוניברסיטת בר-אילן


From AI to DL
Training NNs
Section 1b:
Training Neural Networks
Part 1: Introduction to Deep Learning
Lecture Series on Hardware for Deep Learning
EnICS
Emerging Nanoscaled
Integrated Circuits and Systems Labs
Bar-Ilan University
אוניברסיטת בר-אילן

From AI to DL | Training NNs

Section 1a:
From AI to DL

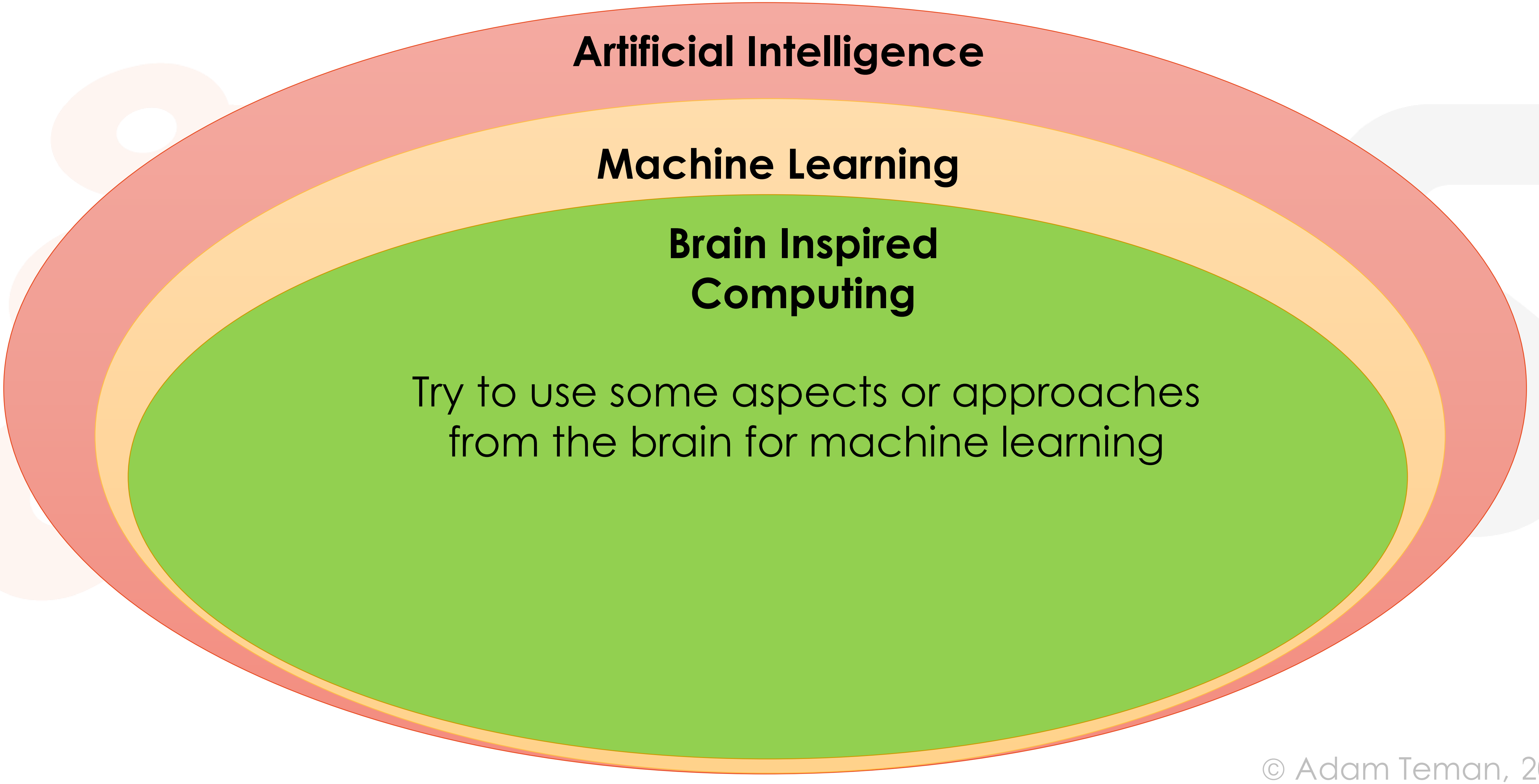
Part 1: Introduction to Deep Learning

Lecture Series on Hardware for Deep Learning

Emerging Nanoscaled
Integrated Circuits and Systems Labs

Tradition of Excellence
Bar-Ilan University
אוניברסיטת בר-אילן

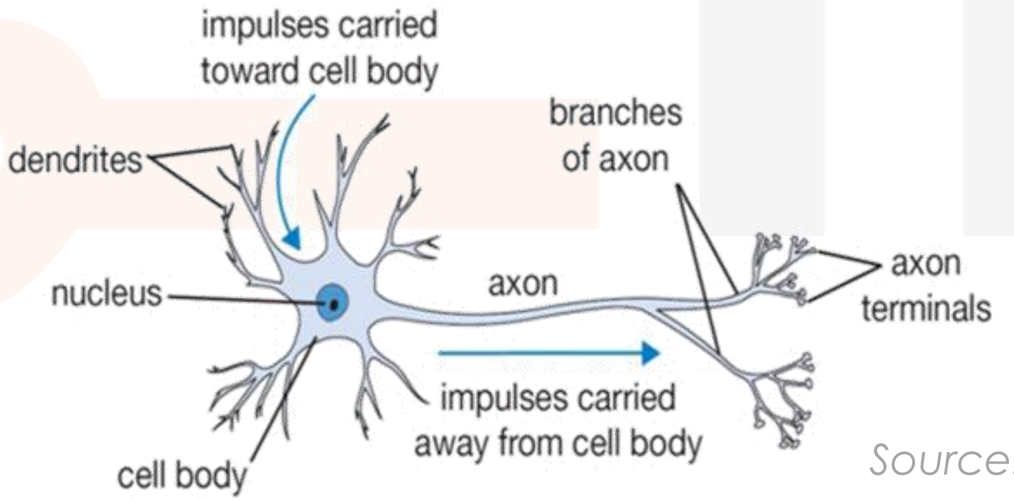
Artificial Intelligence



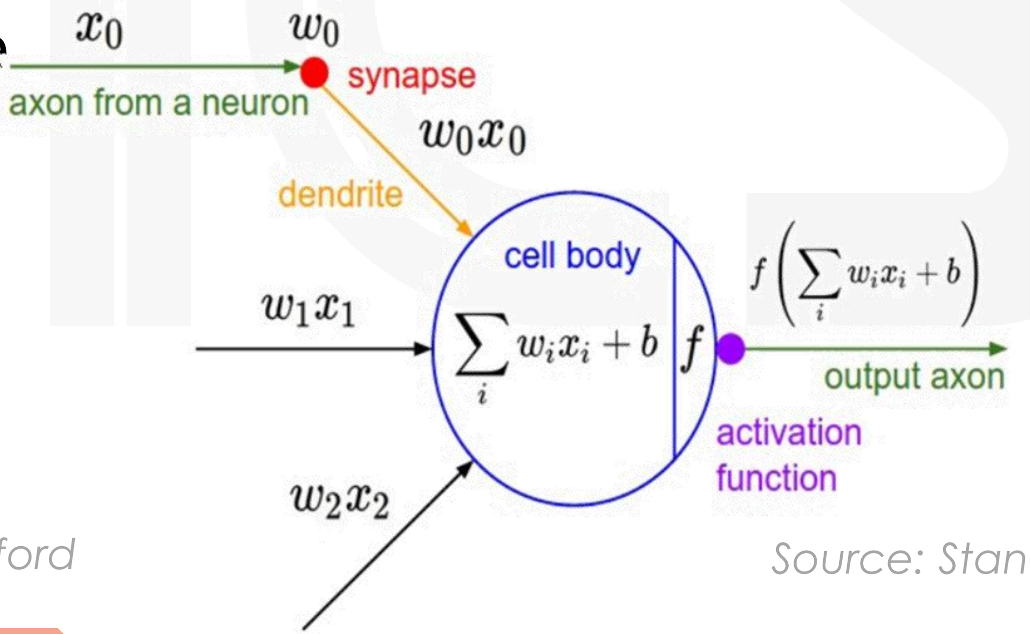
The Neuron

- **There are ~86 Billion neurons in the human brain**
  - Dendrites – inputs to neurons
  - Axons – outputs from neurons
  - Synapse – connection between axon and dendrite
  - Activation – signal propagating between neurons
  - Weight – scaling of a signal by a synapse

There are approximately 10^{14}-10^{15} synapses in the average human brain

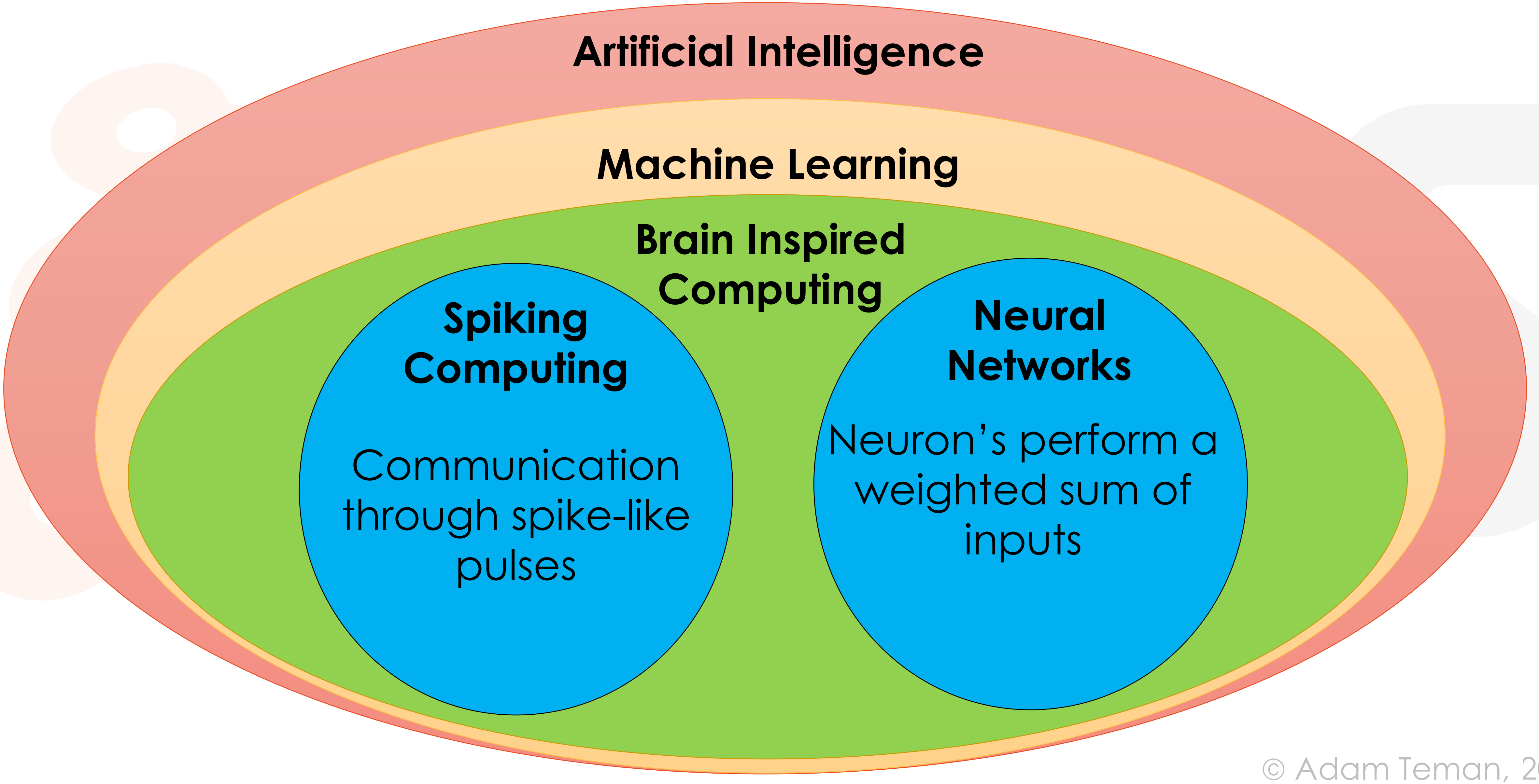


Source: Stanford

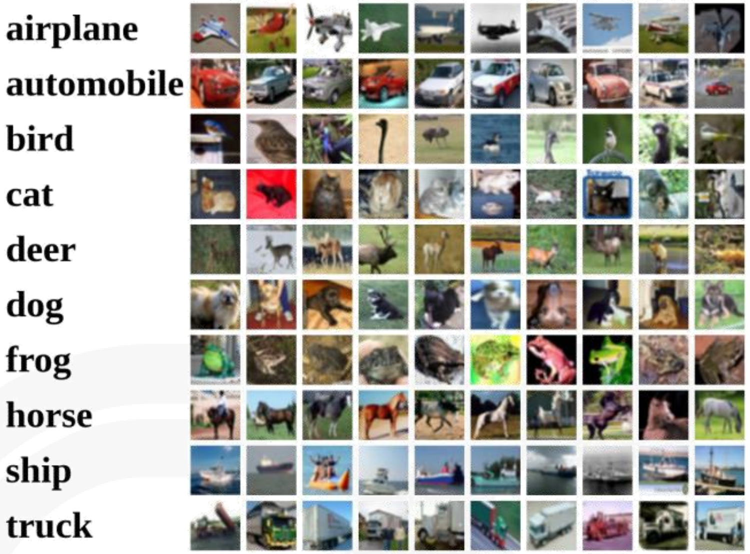


Source: Stanford

Artificial Intelligence



Linear Classification



CIFAR10
50,000 images of 32x32x3
Source: Stanford 321n

- **Given a 32x32 RGB image from the CIFAR10 database, can we use a linear approach to classify the image?**

$$f(x,W) = Wx + b$$

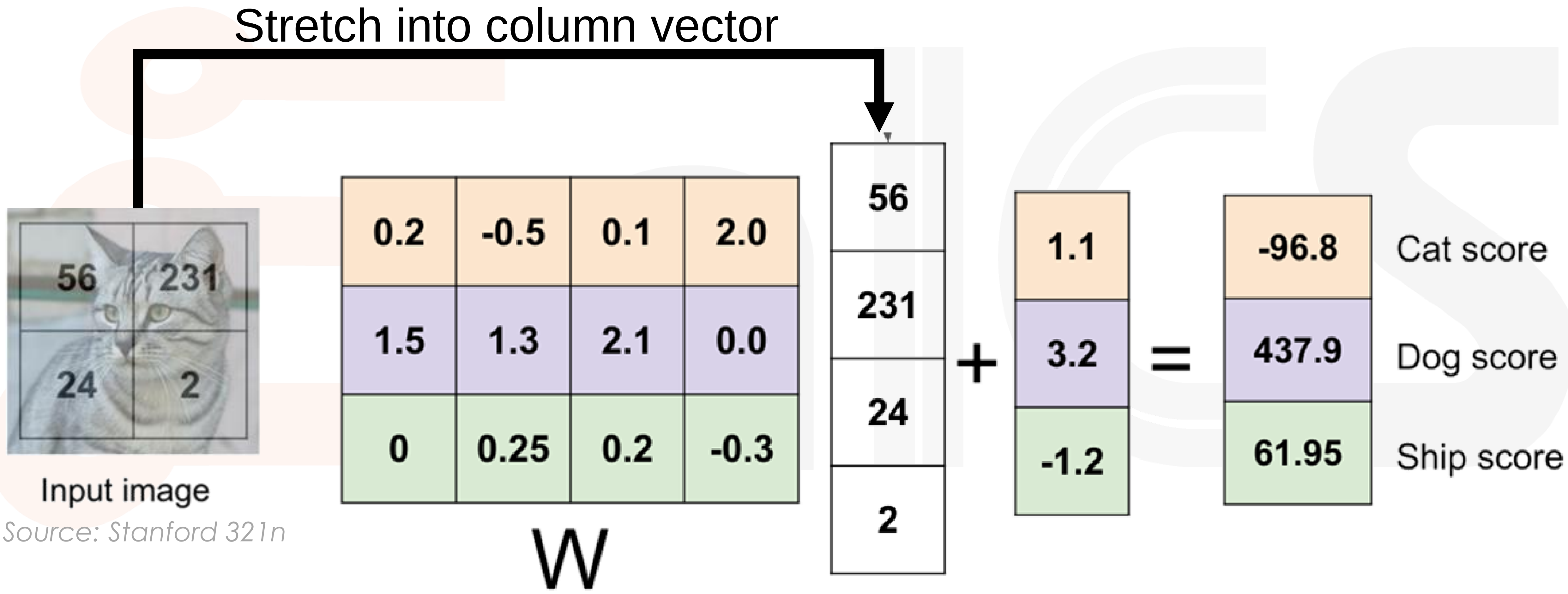
$f(x,W)$: 10x1 — W: 10x3072 — x: 3072x1 — b: 10x1

Array of 32x32x3 numbers (3072 numbers total)

→ $f(x,W)$ → 10 numbers giving class scores

W Parameters or weights

Example with 4 pixels and 3 classes



Source: Stanford 321n

But Linear Classifiers are Limited

- **Hard cases for a linear classifier**
  - Can you draw a line to separate the two classes?

Class 1: number of pixels > 0 odd

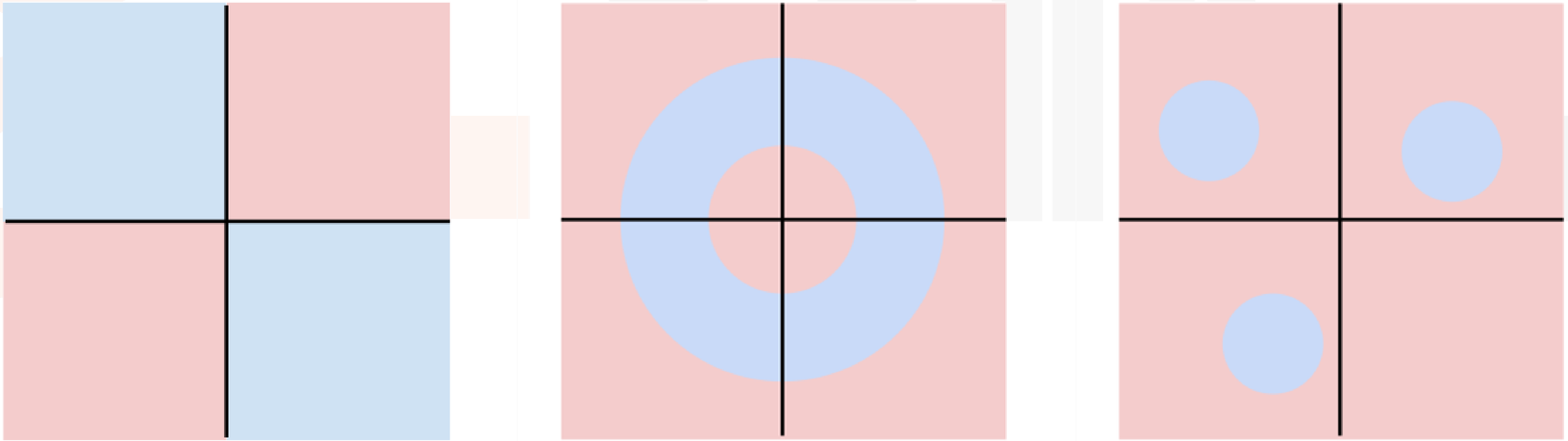
Class 2: number of pixels > 0 even

Class 1: 1 <= L2 norm <= 2

Class 2: Everything else

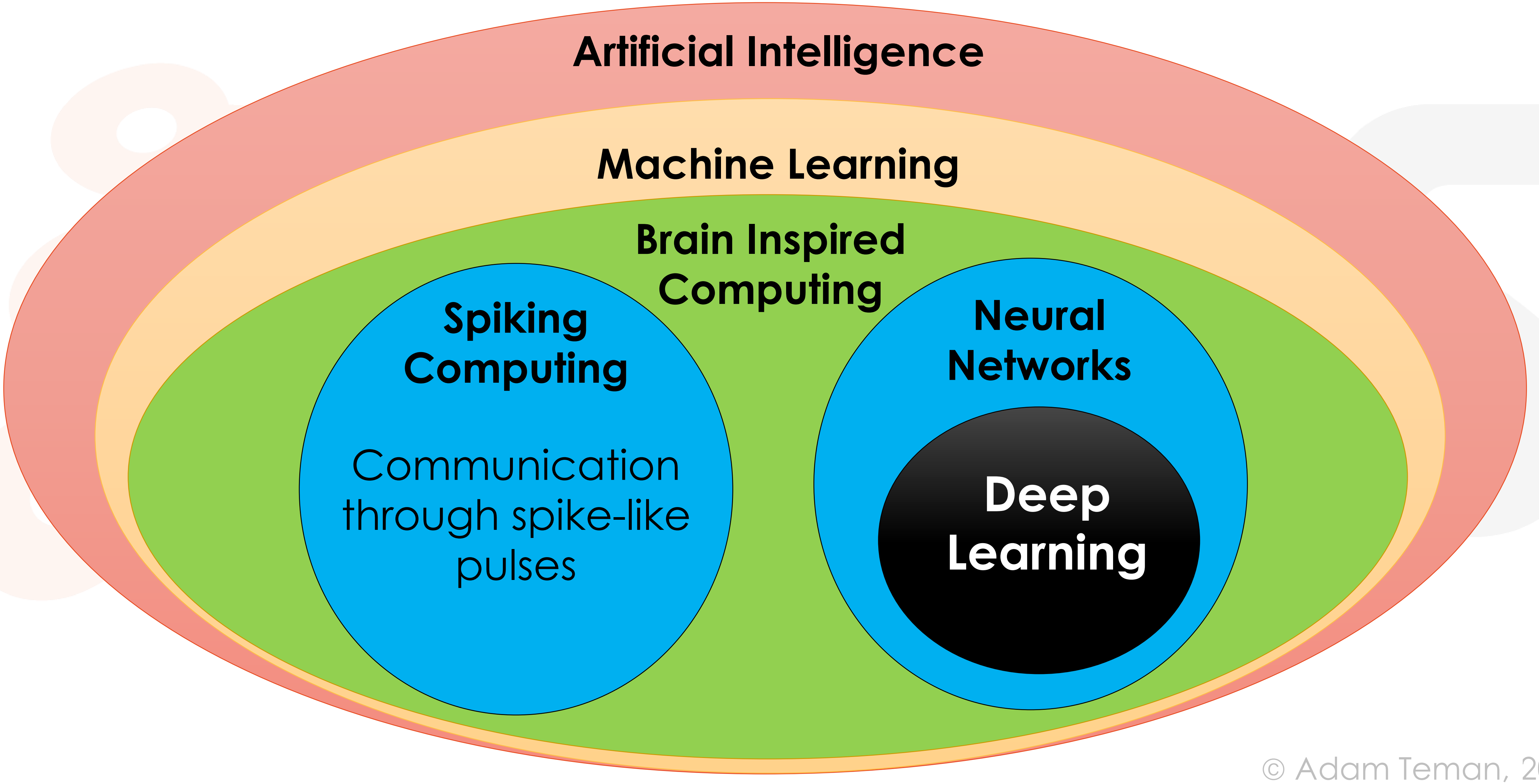
Class 1: Three modes

Class 2: Everything else



Source: Stanford 321n

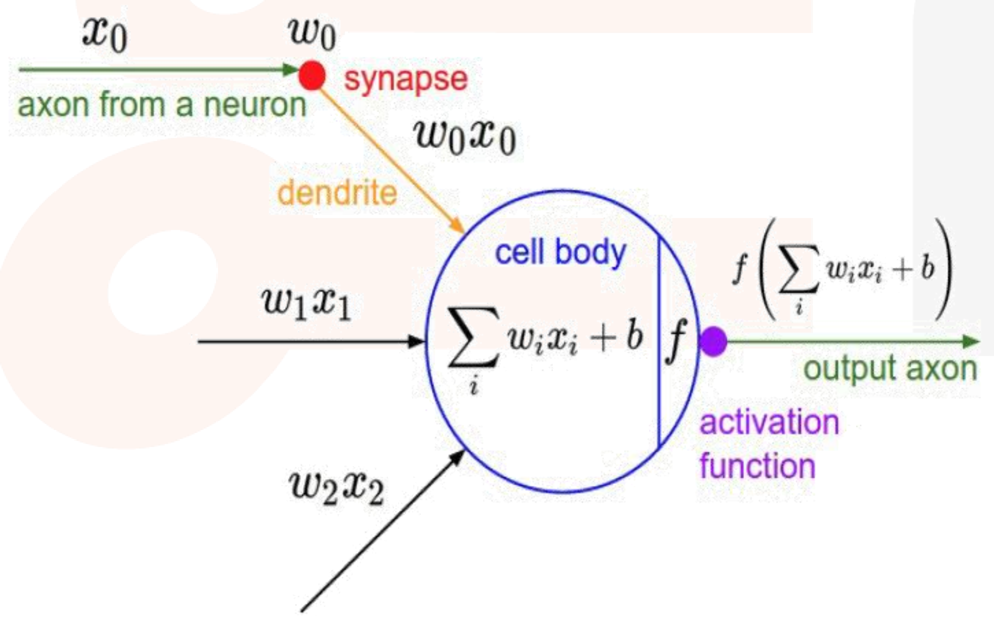
Artificial Intelligence

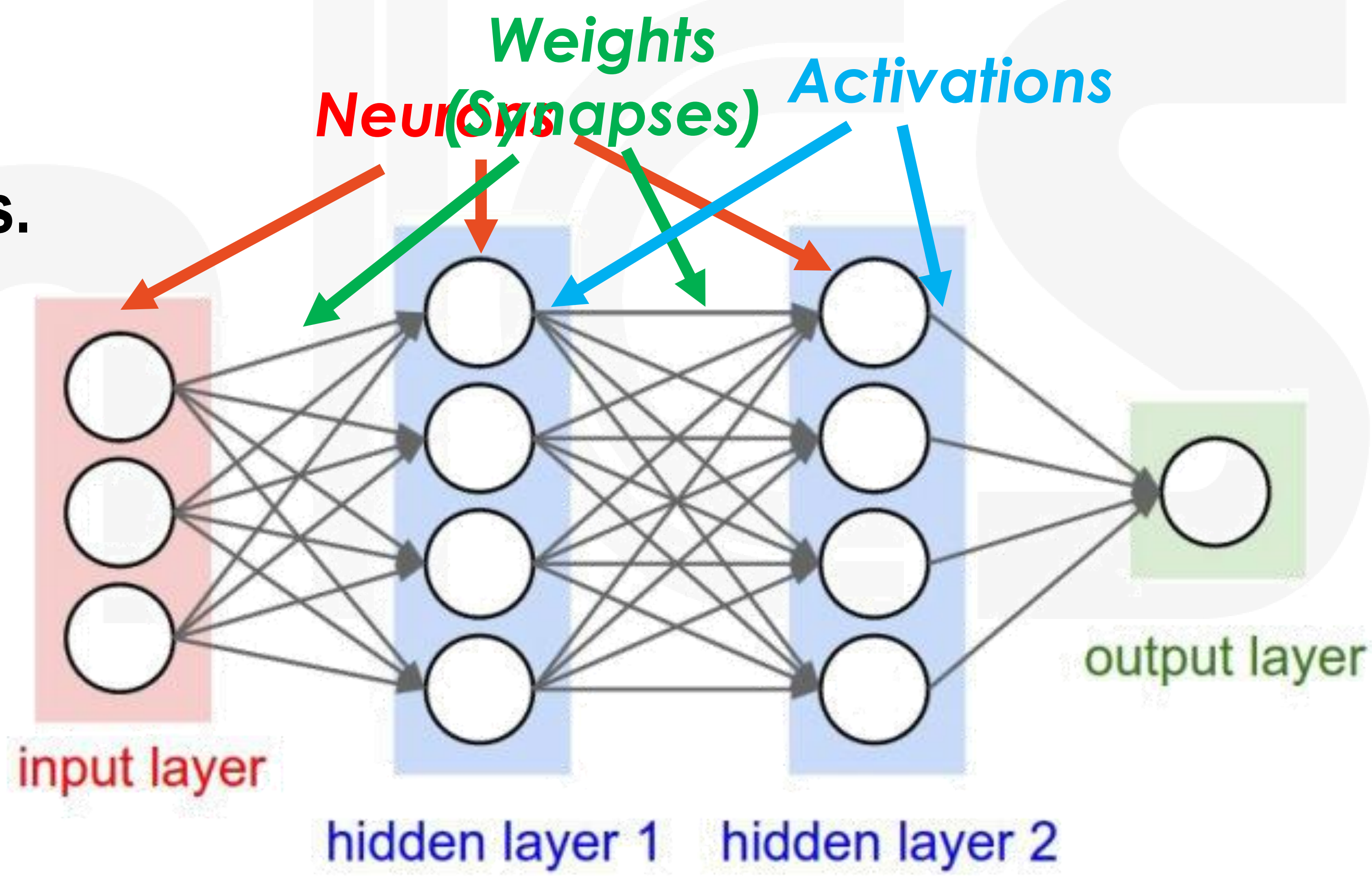


Deep Neural Networks

$$Y_j = \text{activation}\left(\sum_{i=1}^{3} W_{ij} \times X_i\right)$$

- A neuron's computation involves a weighted sum of the input values followed by some non-linearity.
- Deep Neural Networks (DNNs) go through several layers of neurons.

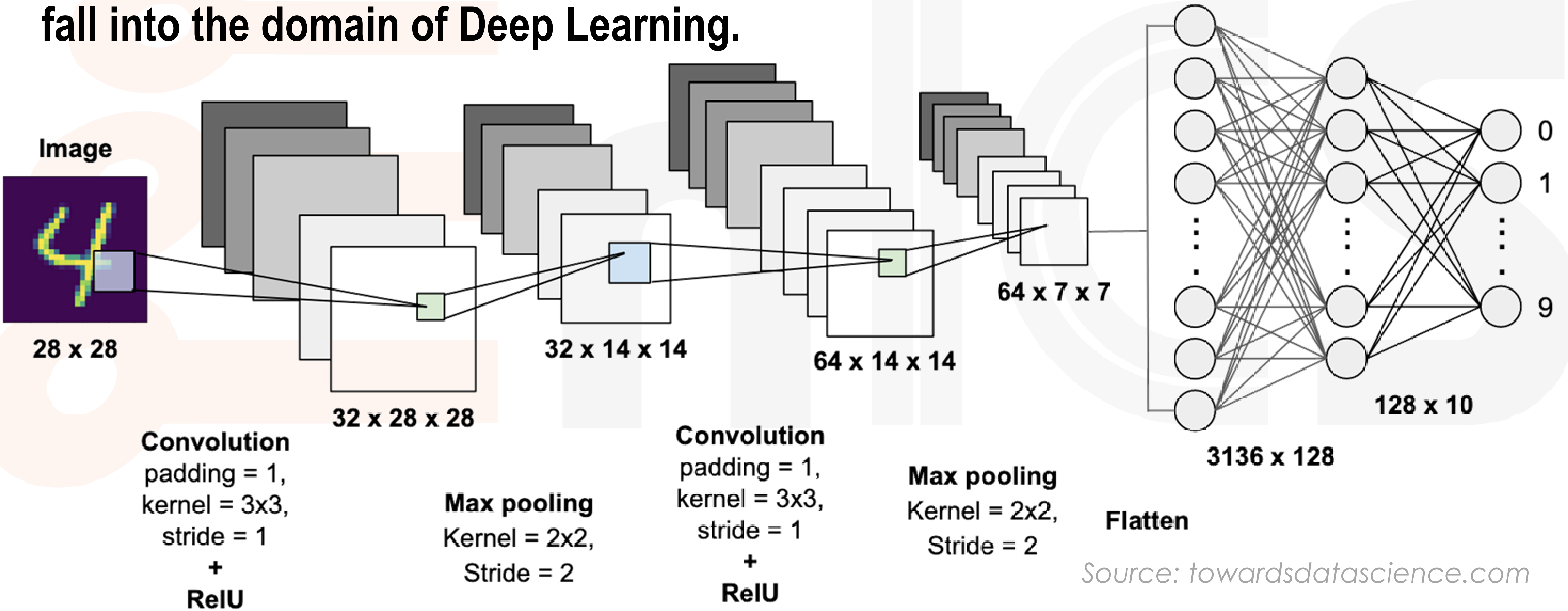




Source: Stanford 321n

Deep Learning

- **Neural networks with many (more than three) layers fall into the domain of Deep Learning.**

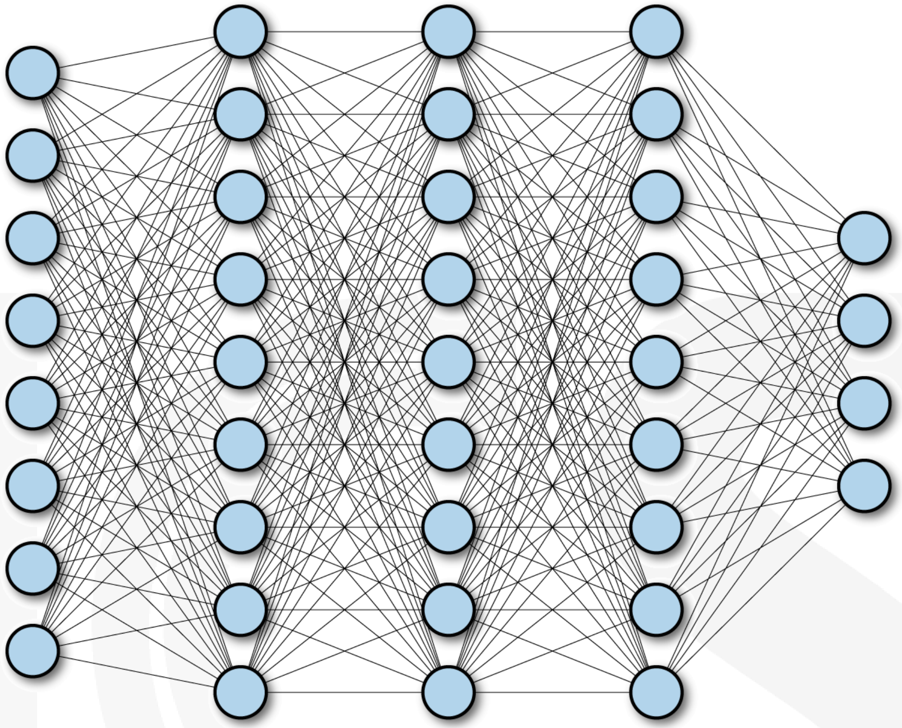


Source: *towardsdatascience.com*

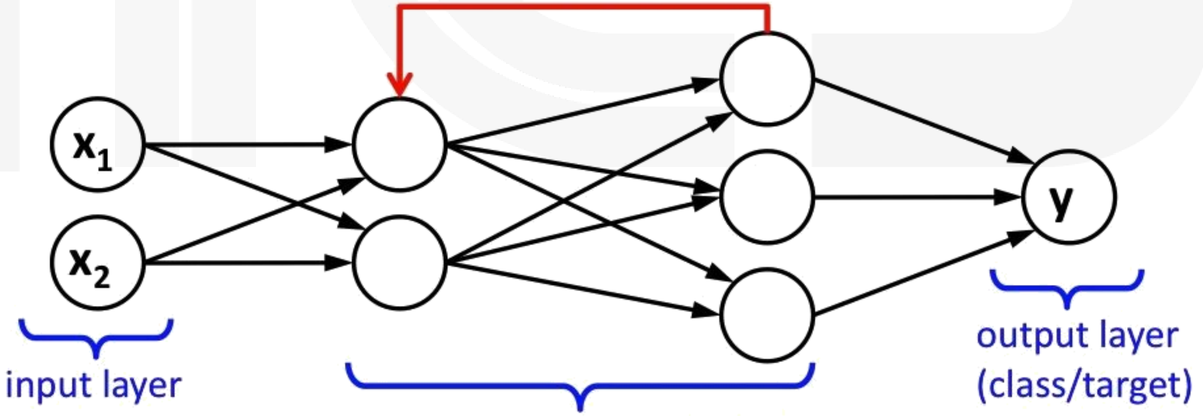
Popular Types of DNNs

- **Fully-Connected NN**
  - Feed forward,
  - a.k.a. multilayer perceptron (MLP)
- **Convolutional NN (CNN)**
  - Feed forward,
  - sparsely-connected w/ weight sharing
- **Recurrent NN (RNN)**
  - Feedback
- **Long Short-Term Memory (LSTM)**
  - Feedback + storage

Fully Connected NN



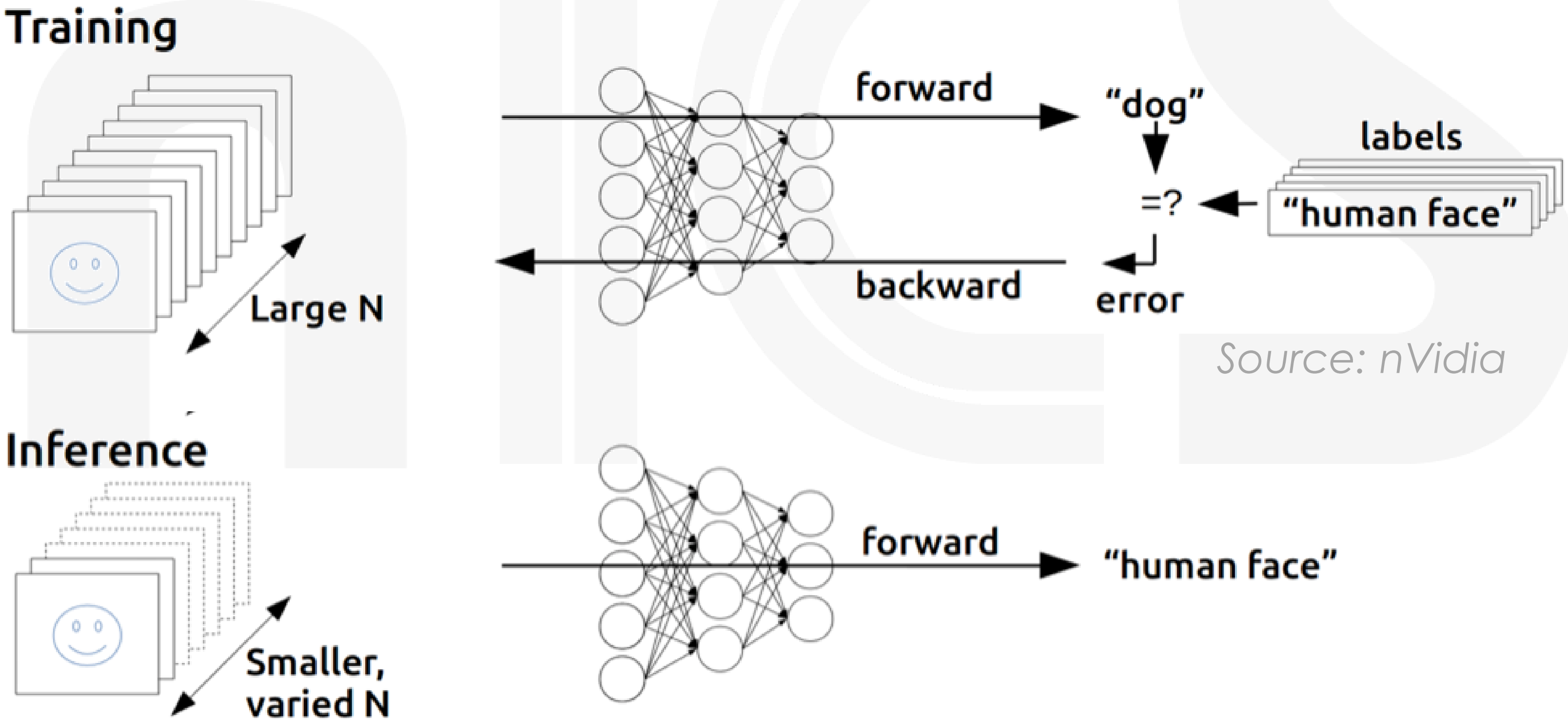
Source: O'Reilly



Source: gabormelli.com

Training and Inference

- **Machine Learning algorithms, such as DNNs, have to *learn* their task.**
- **The learning phase is called *Training* and it involves determining the values of the weights and bias of the network.**
  - Supervised Learning: training set is **labeled**
  - Unsupervised Learning: training set is **unlabled**
  - Reinforcement Learning maximize the **reward**.
- **After learning, running with the learned weights is known as *Inference*.**



Source: nVidia

Datasets

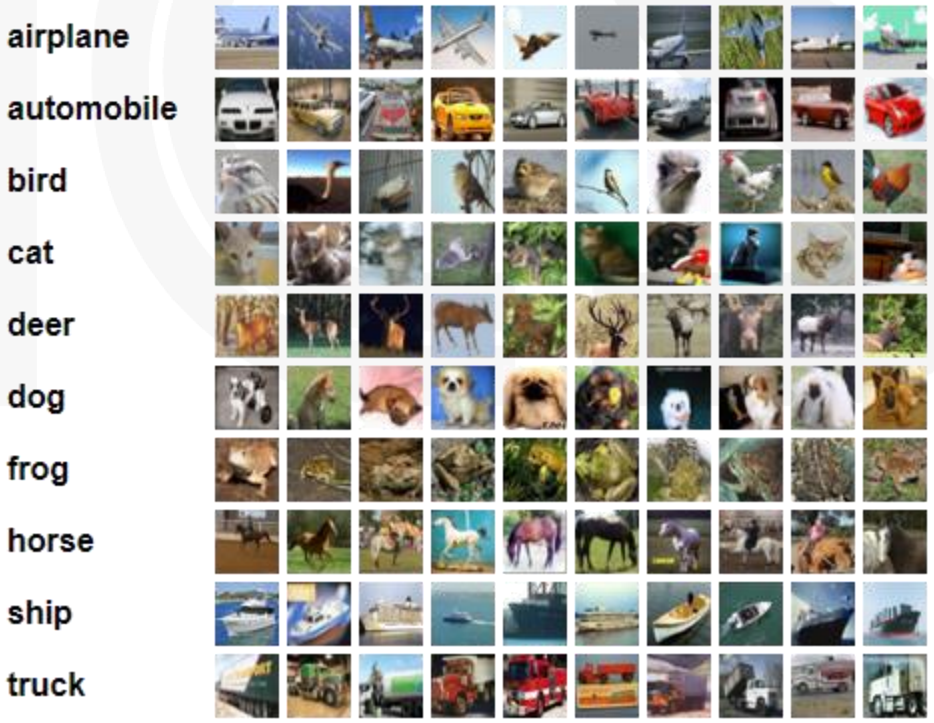
- **The three components that enabled the breakthrough of deep learning are:**
  - Computational Power (Moore's Law)
  - Parallelism for Training (GPUs)
  - Availability of labeled datasets.
- **Several popular datasets for image classification have been developed:**
  - MNIST: digit classification
  - CIFAR-10: simple images
  - ImageNet: many categories of images
  - PASCAL VOC: object detection
  - MS COCO: detection, segmentation, recognition
  - YouTube data set: 8 Million videos
  - Google audio set: 2 Million sound clips

MNIST and CIFAR-10

- **MNIST** (1998)
  - Handwritten digits
  - 28x28x1 (B&W) pixels
  - 10 classes (0-9)
  - 60,000 Training
  - 10,000 Testing

http://yann.lecun.com/exdb/mnist/

- **CIFAR-10** (2009)
  - Simple color images
  - 32x32x3 (RGB) pixels
  - 10 mutually exclusive classes
  - 50,000 Training
  - 10,000 Testing



https://www.cs.toronto.edu/~kriz/cifar.html

ImageNet (2010)

- **ImageNet:**
  - A large scale image data set
  - 256x256x3 (RGB) pixels
  - 1000 classes
    e.g., 120 different breeds of dogs
  - 1.3 Million Training images
  - 100,000 Testing images
  - 50,000 Validation images

- **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**
  - Accuracy of classification reported based on **top-1** and **top-5** error

http://www.image-net.org/challenges/LSVRC/

From AI to DL | Training NNs

Section 1b: Training Neural Networks

Part 1: Introduction to Deep Learning

Lecture Series on Hardware for Deep Learning

Emerging Nanoscaled Integrated Circuits and Systems Labs

Bar-Ilan University
אוניברסיטת בר-אילן
Tradition of Excellence

Loss Function

- **A loss function tells how good our current classifier is.**
  - Given a dataset of examples $\{(x_i, y_i)\}_{i=1}^N$
    Where x_i is image and y_i is (integer) label
  - Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

- **Popular loss functions:**
  - Multiclass SVM loss $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$
  - SoftMax

SoftMax Classifier

- Treat the scores (outputs) of our classifier as ***unnormalized log probabilities of the classes:*** $P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}}$ $s=f(x_i;W)$
- We want to ***maximize the log likelihood*** or alternatively ***minimize the negative log likelihood*** of the correct class: $L_i=-\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$

	Unnormalized log probabilities	exp	normalize: probabilities
cat	3.2	24.5	0.13
car	5.1	164.0	0.87
frog	-1.7	0.18	0.00

-log on correct category → 0.89

Optimization through Gradient Descent

- **We want to reduce our loss to reach a minimum**
  - Advance in the opposite direction of the derivative
- **We could find the derivative of the loss function at the current point numerically**

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- **But that's a lot of work!**
  - Instead, just find the analytic gradient… $\nabla_W L$

current W:	**W + h** (first dim):	**gradient dW:**
[0.34,	[0.34 + **0.0001**,	[-2.5,
-1.11,	-1.11,	?,
0.78,	0.78,	?,
0.12,	0.12,	
0.55,	0.55,	
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

(1.25322 - 1.25347)/0.0001
= -2.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

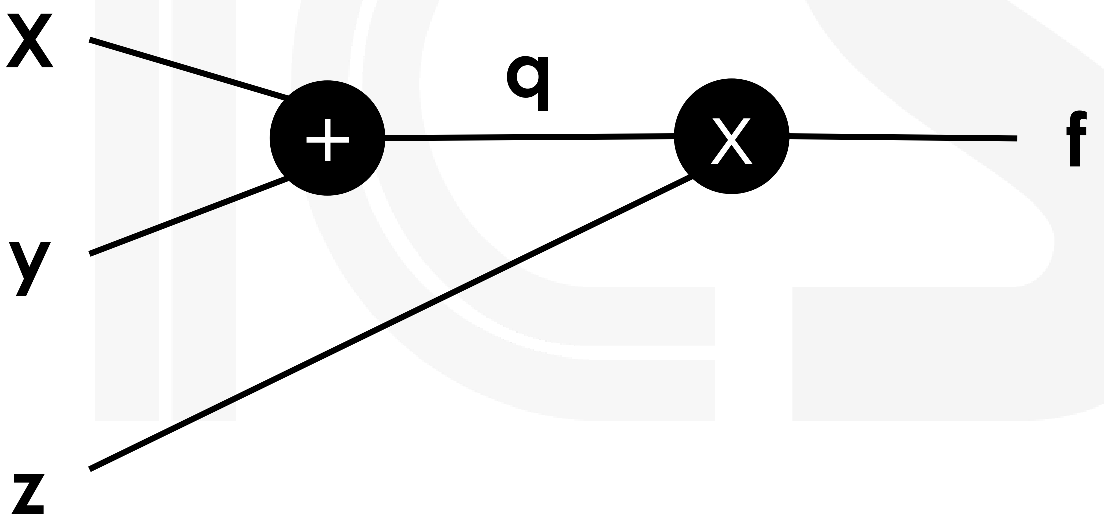
Backpropagation

- **We can use backpropagation ("*backprop*") to find the analytic derivative.**
- **Let's use a simple example:**
  - Our function: $f(x, y, z) = (x + y)z$
- **Let's build a computational graph:**

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

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

- **Remember the chain rule?** $\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$

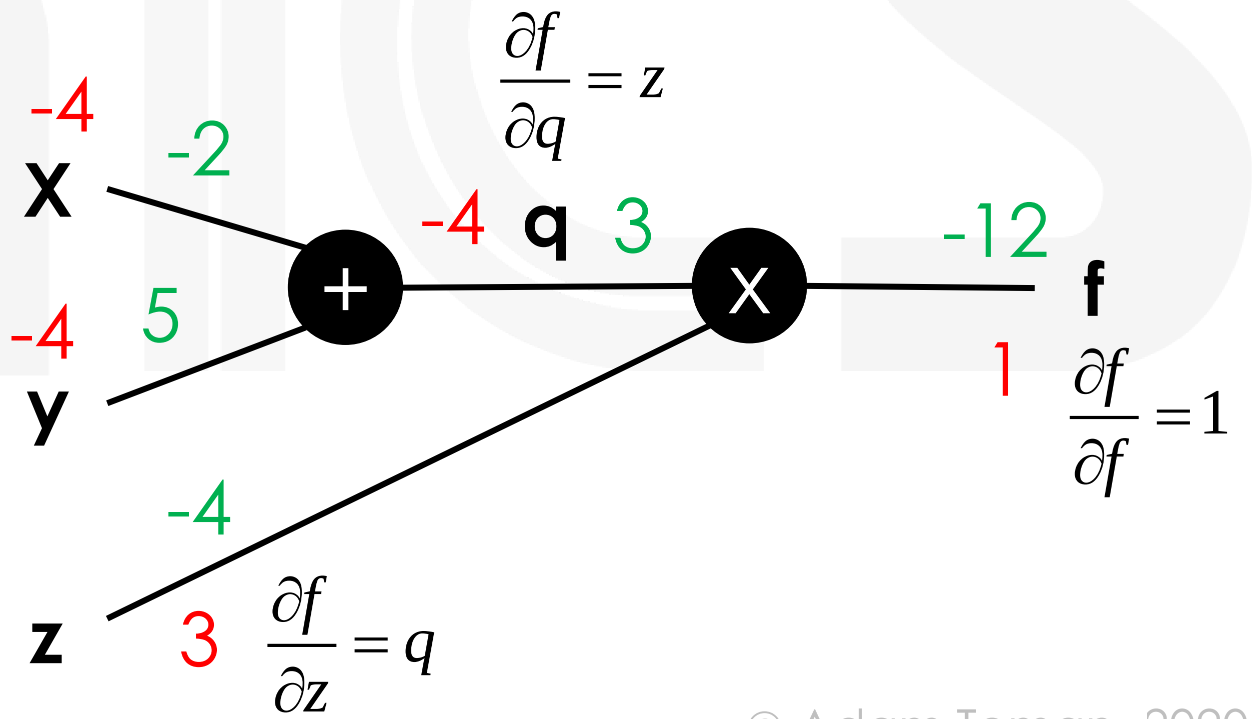


Backpropagation

- So, let's assume we have some current value in our network:
  - e.g., x=-2, y=5, z=-4
- We will first run inference through the network.
- Now find the derivative of the output to each input

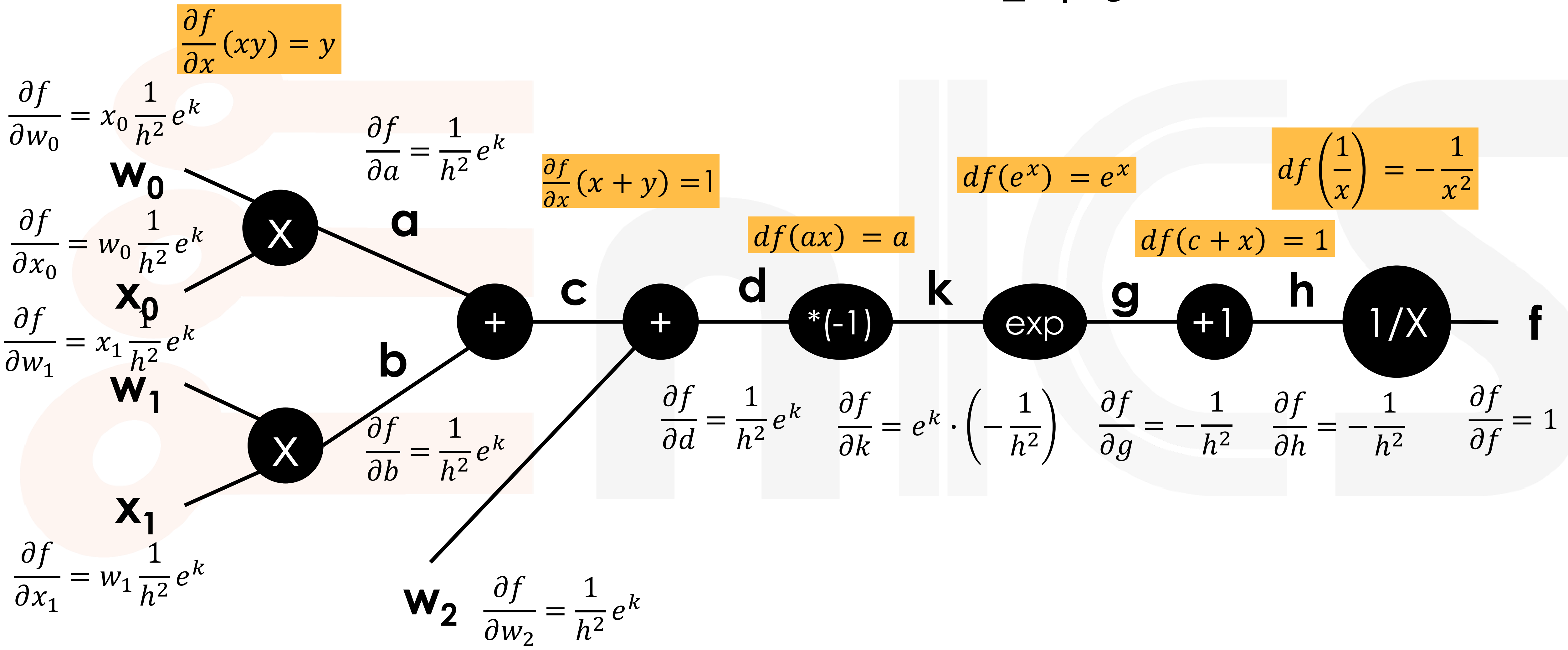
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x} = -4 \cdot 1$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y} = -4 \cdot 1$$



Another Example

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



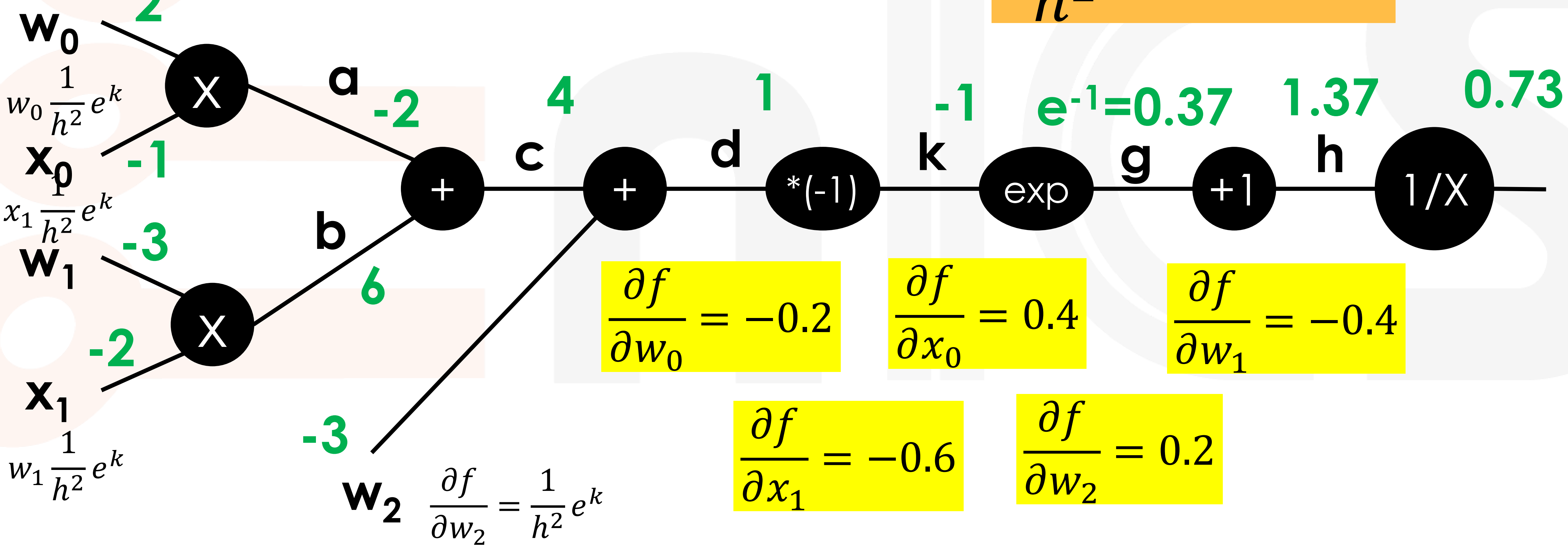
Another Example

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

$h = 1.37$

$k = -1$

$\frac{1}{h^2}e^k = 0.2$



$$\frac{\partial f}{\partial w_0} = x_0\frac{1}{h^2}e^k$$

$$\frac{\partial f}{\partial x_0} = w_0\frac{1}{h^2}e^k$$

$$\frac{\partial f}{\partial w_1} = x_1\frac{1}{h^2}e^k$$

$$\frac{\partial f}{\partial x_1} = w_1\frac{1}{h^2}e^k$$

$$\frac{\partial f}{\partial w_2} = \frac{1}{h^2}e^k$$

$\frac{\partial f}{\partial w_0} = -0.2$

$\frac{\partial f}{\partial x_0} = 0.4$

$\frac{\partial f}{\partial w_1} = -0.4$

$\frac{\partial f}{\partial x_1} = -0.6$

$\frac{\partial f}{\partial w_2} = 0.2$

Batch Training

- **The straightforward way to update weights:**
  - Run inference on all training samples
    and update weights with the average loss.
  - But this is very costly!
- **Instead, calculate the loss for a *batch*** of inputs.
  - Select a *batch size*, i.e., *n*-sample subset of the entire training set.
  - Calculate the average loss for the batch of samples.
  - Backprop and update weights.
- **A run through the full training set is called an *epoch*.**
- **Another option to reduce training time is called *Transfer Learning***
  - Start with a trained model and adjust to the new model or data set.

Main References

- Stanford C231n, 2017
- Sze, et al. “Efficient Processing of Deep Neural Networks: A Tutorial and Survey”, Proceedings of the IEEE, 2017
- Sze, et al. ISCA Tutorial 2019