Introduction to Deep Learning

Ups and Downs of Deep Learning

- 1958, perceptron (linear model)
- 1969: Perceptron has limitation
- 1980: Multi-layer perceptron (X)
    - Do not have significant difference from DNN today
- 1986: Backpropagation
    - Usually more than 3 hidden layers is not helpful
- 1989: 1 hidden layer is 'good enough', why deep learning?
- 2006: RBM initialization (breakthrough)
- 2009: GPU
- 2011: Start to be popular in speech recognition
- 2012: win ILSVRC image competition

Three Steps for Deep Learning

- Define a set of function (Neural Network)
- Evaluate the performance of function
- Pick up the best function

Step1:

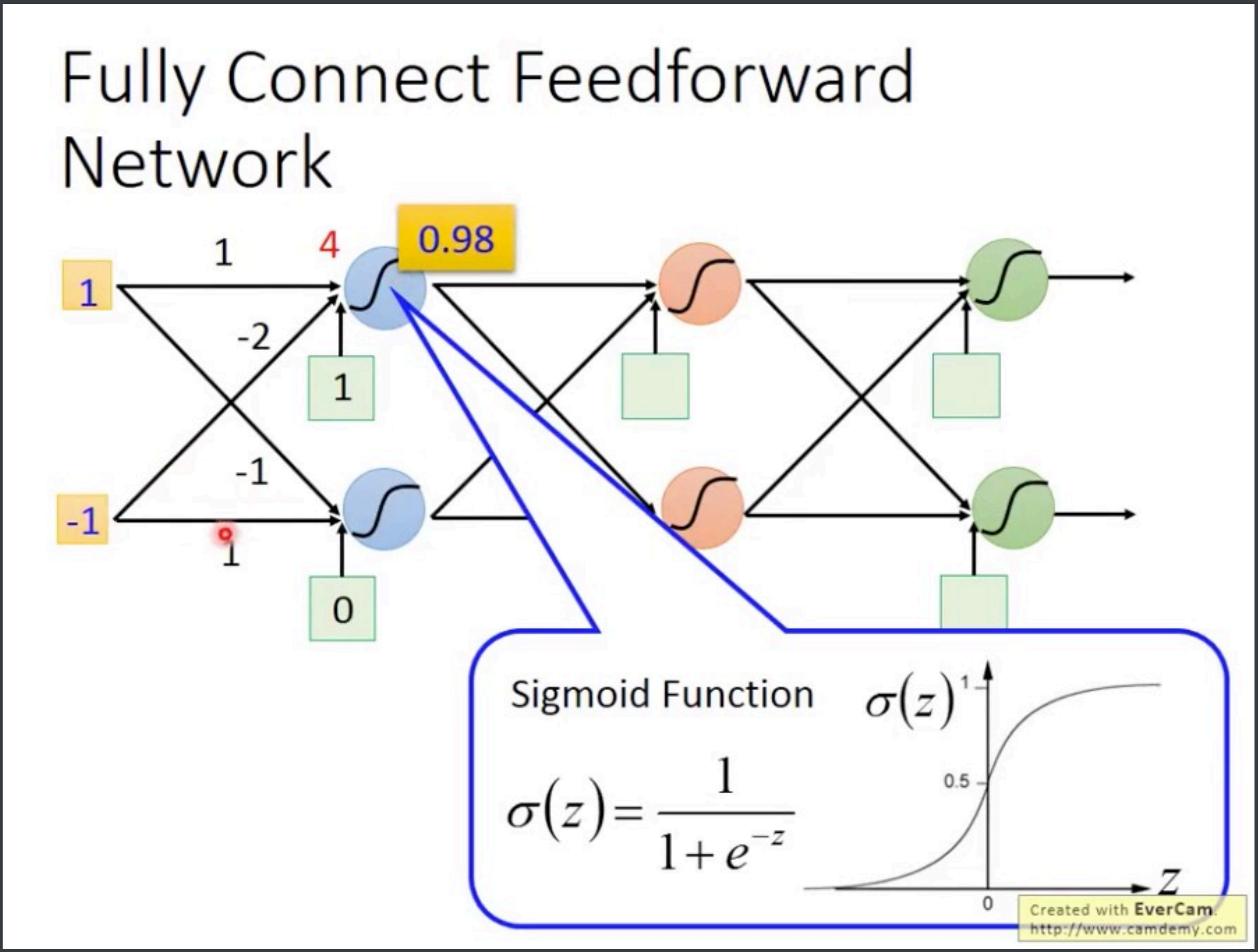
Neural Network

Concatnate different logistic regressions, each logistic regression has its own weight and bias.

Different connection leads to dfferent network structures:

Network parameter theta: all the weights and biases in the 'neurons'

How to connect them?



Fully Connect Feedforward Network

Sigmoid Function

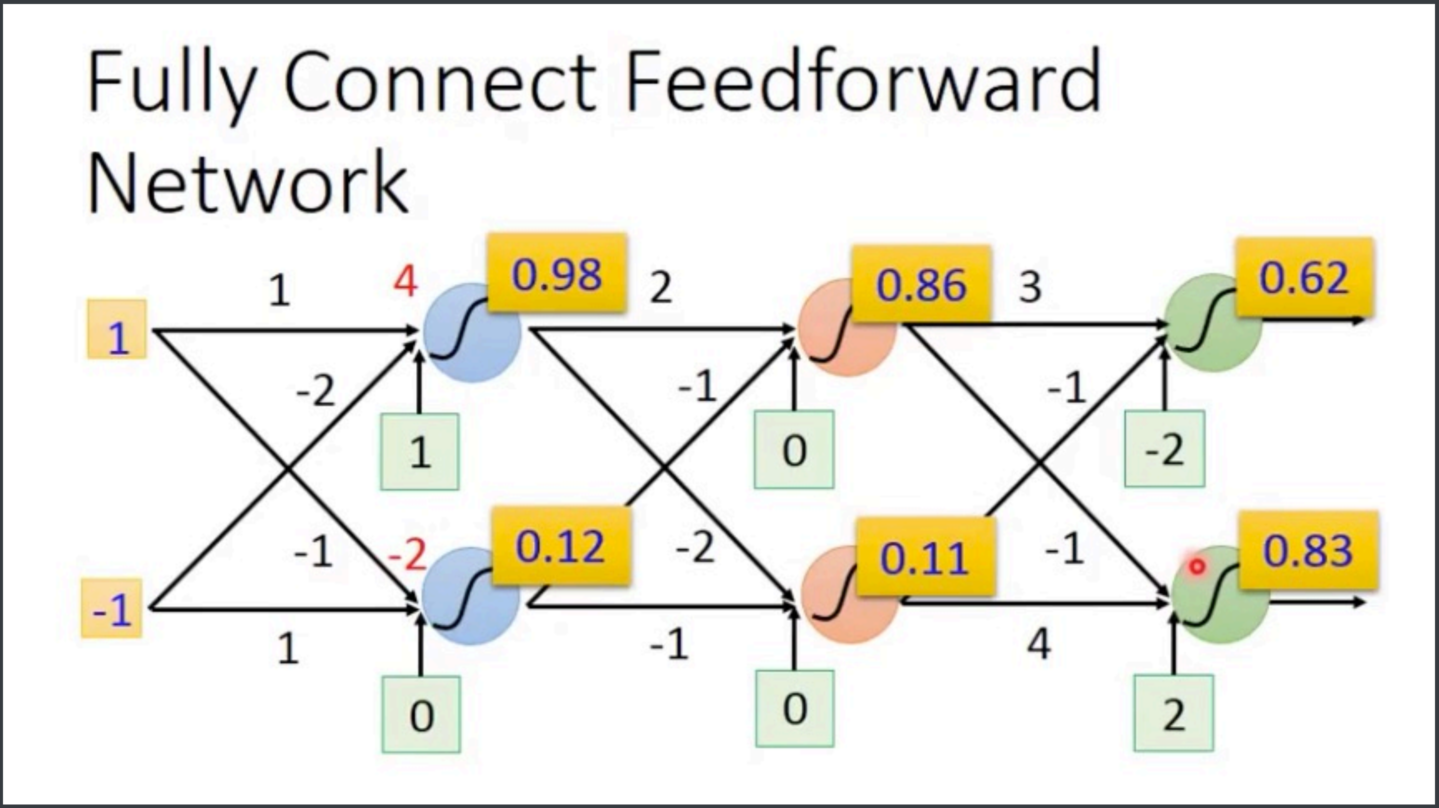
$$\sigma(z)=\frac{1}{1+e^{-z}}$$

weight and bias are from training data.

for example, blue top neuron, weight (1, -2), bias(1); blue bottom neuron, weight (-1, 1), bias(0) top blue output: 1x1 + (-1) x (-2) + 1 (bias) = 4 >>> Pass the sigmoid function >>> 0.98

bottom blue output: 1x(-1) + (-1)x1 + 0 = -2 >>> Pass the sigmoid function >>> 0.12

Continuing ...

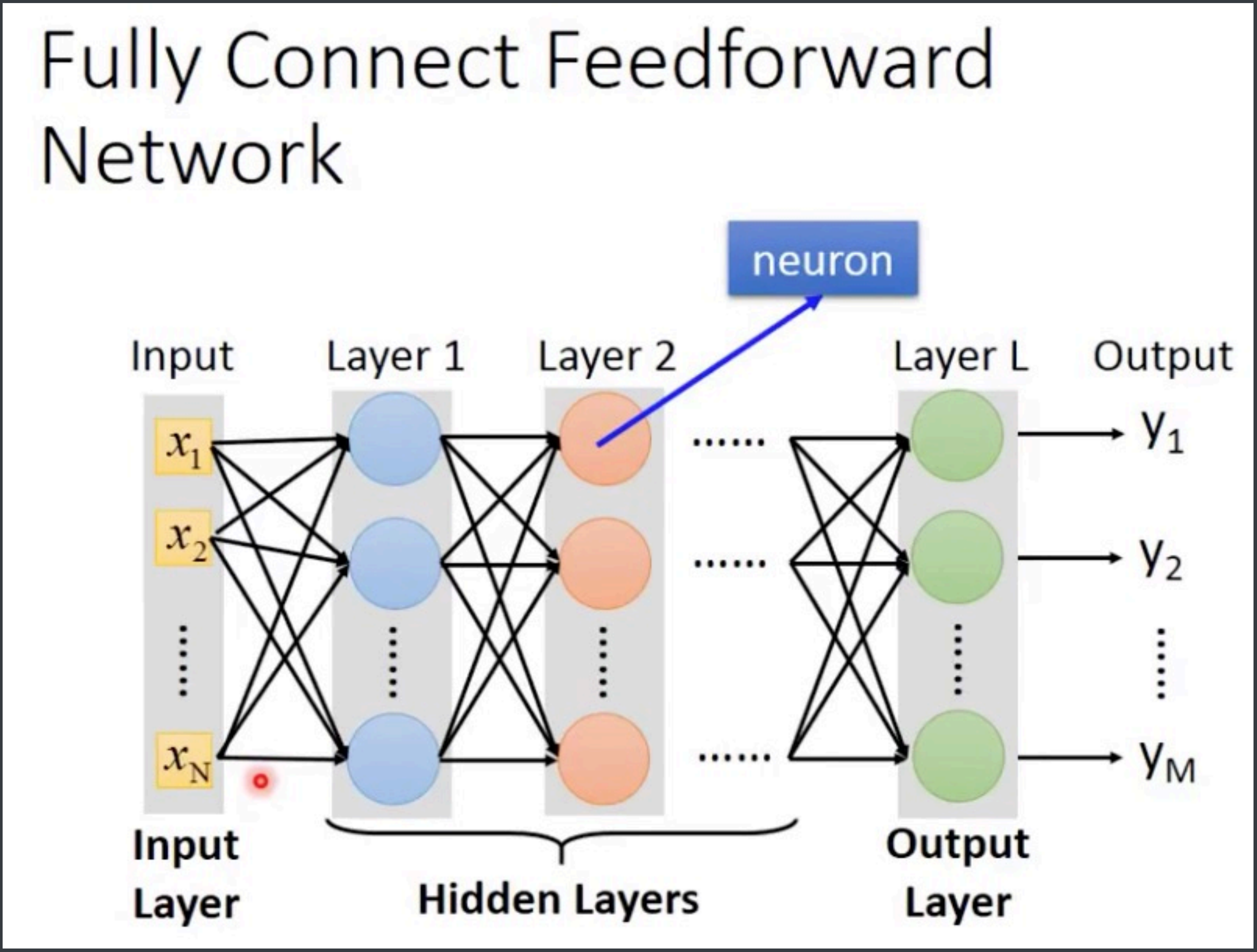


Fully Connect Feedforward Network

Input is a vector (1, -1), output is a vector (0.62, 0.83)

Given network structure, define a function set

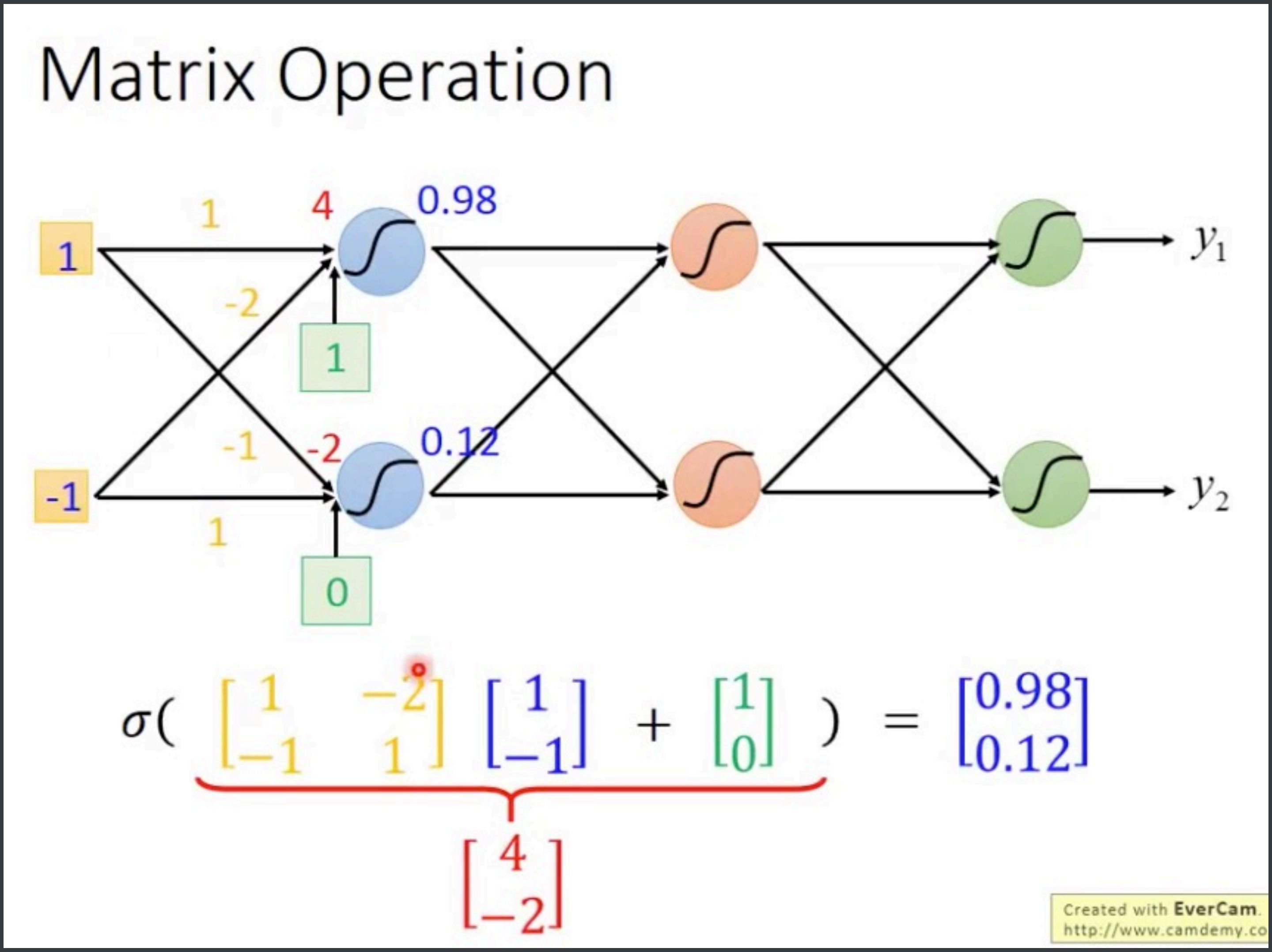
Types of connections



Fully Connect Feedforward Network

Deep = many hidden layers

Matrix Operation



Matrix Operation

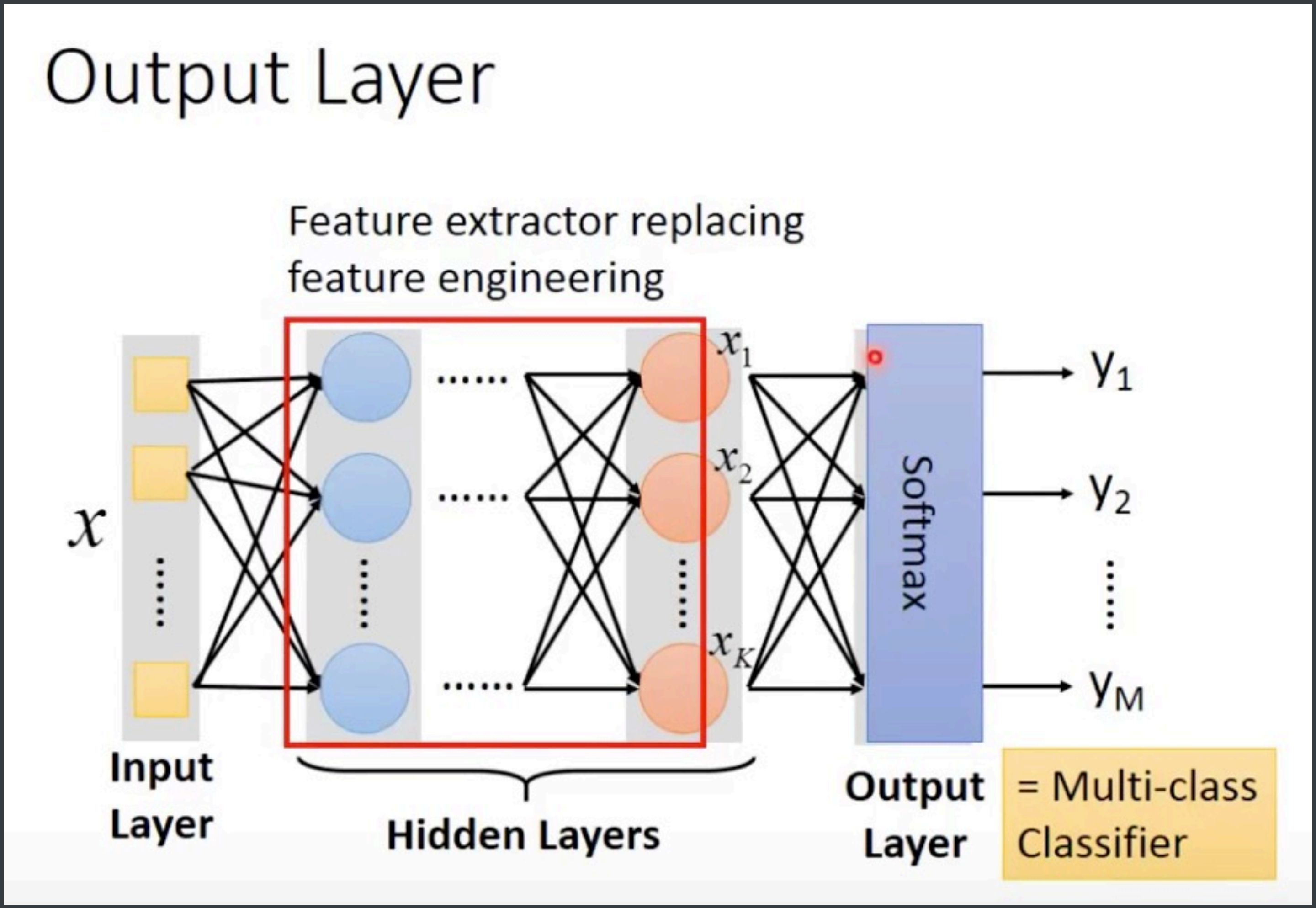
$$\underbrace{\sigma\left(\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right)}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}} = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

Feedforward Network Calculation Process:

(input multiply by matrix (weight) + bias) >> into sigmoid (any activitation function) >>> output

$$\sigma(W^! * x + b^1)$$
$$\sigma(W^2 * a^1 * b^2)$$
$$\cdots$$

a series of matrix computations



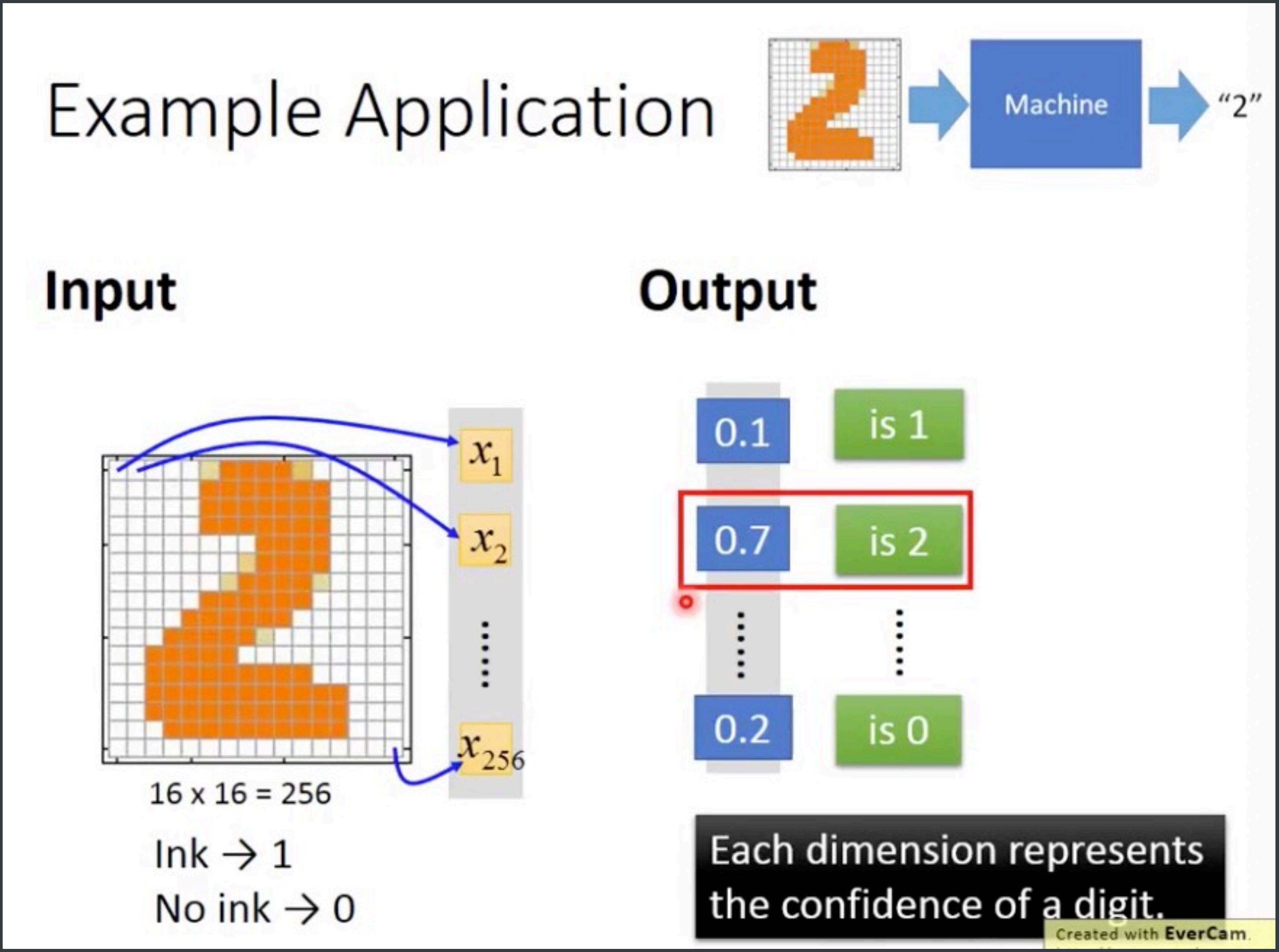
Output Layer

Feature extractor replacing feature engineering

= Multi-class Classifier

output layer = multi-class classifier

outpub layer: after the hidden layers conducted complex transformation, it (output layer) takes a set of features



Example Application

Input

16 x 16 = 256

Ink → 1

No ink → 0

Output

Each dimension represents the confidence of a digit.

Created with EverCam.

Softmax >>> y is a 10-dim vector, each dimension is corresponding to a number

The only constraint for this hand-written image recognition is the 256-dim input, and 10-dim output.

The number of layers and how they connected is unlimited, you can design this part.

Q: How many layers? How many neurons for each layer?

A: Trail and Error + Intuition

Q: Can the structure be automatically detetermined?

A: e.g. Evolutionary Artificial Neural Networks

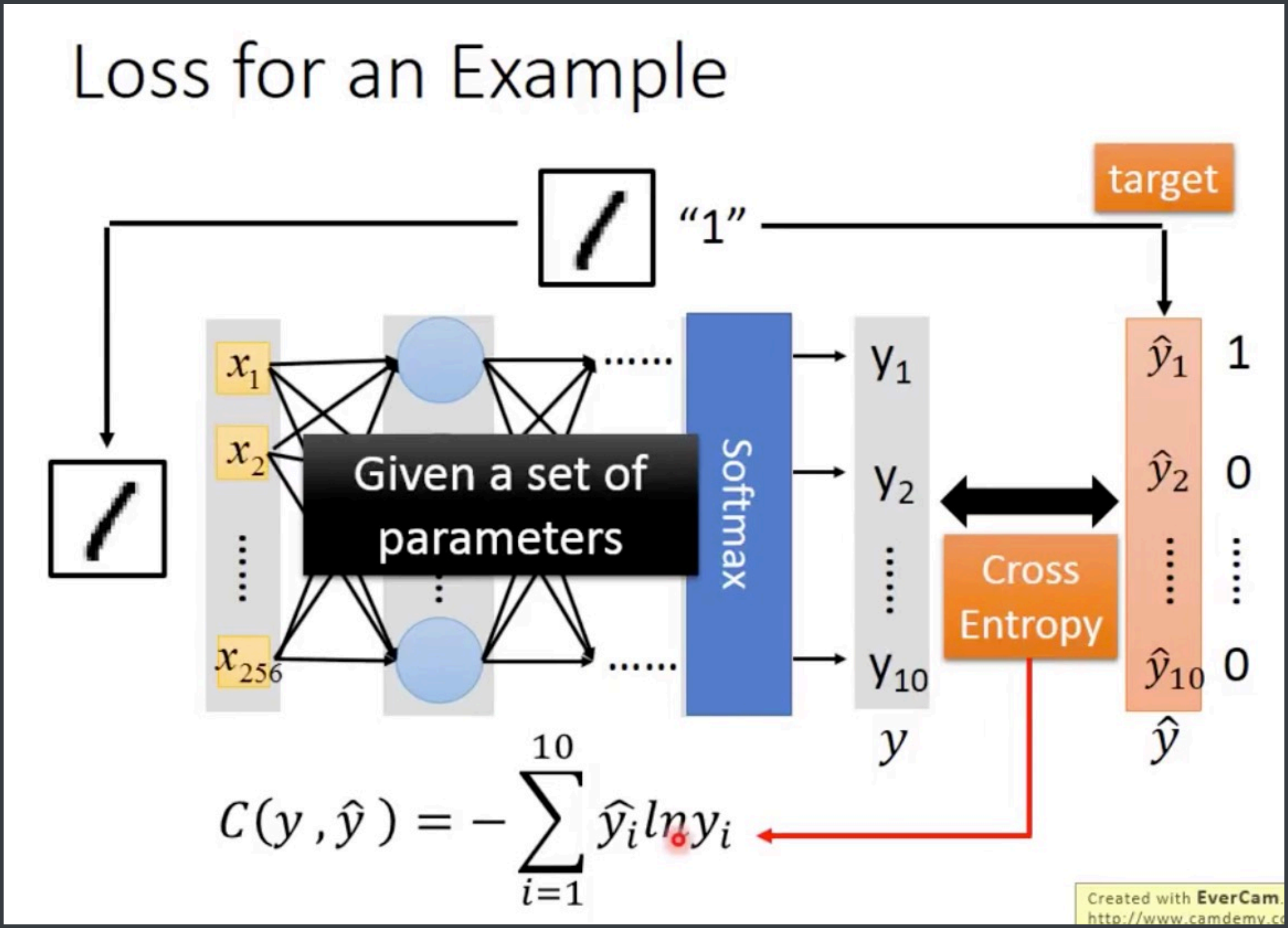
Q: Can we design the network structure?

A: CNN

Note: From ML to DL, the question transformed from 'Feature Extraction' to 'Network Design'

Step2:

Loss function



Loss for an Example

$$C(y,\hat{y}) = -\sum_{i=1}^{10} \hat{y}_i \ln y_i$$

Calculate hte cross entropy of y and y^, find the parameter that minimum the loss.

Total loss:

$$L = \sum_{n=1}^{N} C^n$$

Goal:

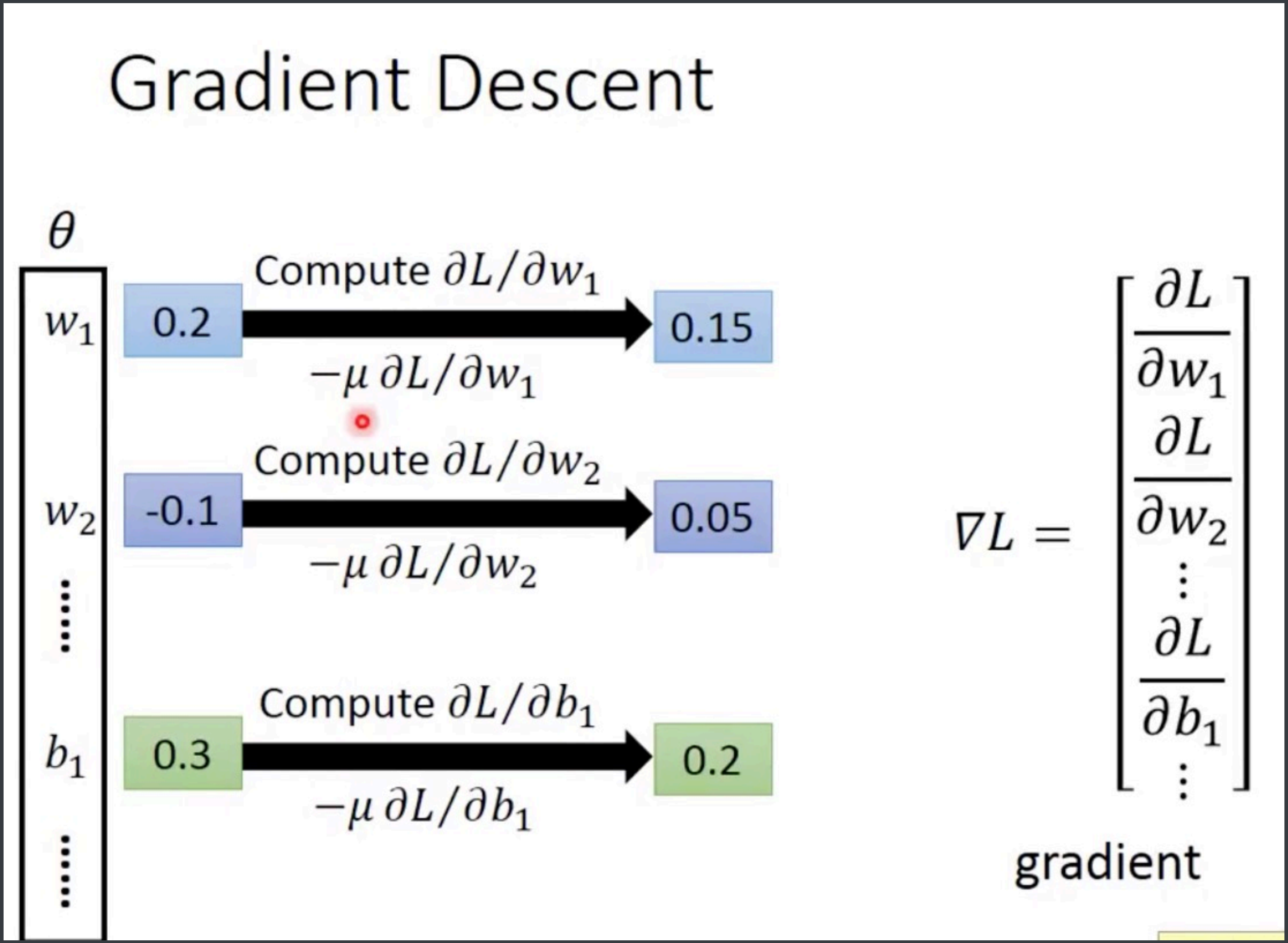
Find a function in function set that miminizes total loss L

OR

Find the network parameter theta* that minimize total loss L

Method:

Gradient Descent



Gradient Descent

θ

w_1 $\quad 0.2 \xrightarrow[-\mu\,\partial L/\partial w_1]{\text{Compute } \partial L/\partial w_1} 0.15$

w_2 $\quad -0.1 \xrightarrow[-\mu\,\partial L/\partial w_2]{\text{Compute } \partial L/\partial w_2} 0.05$

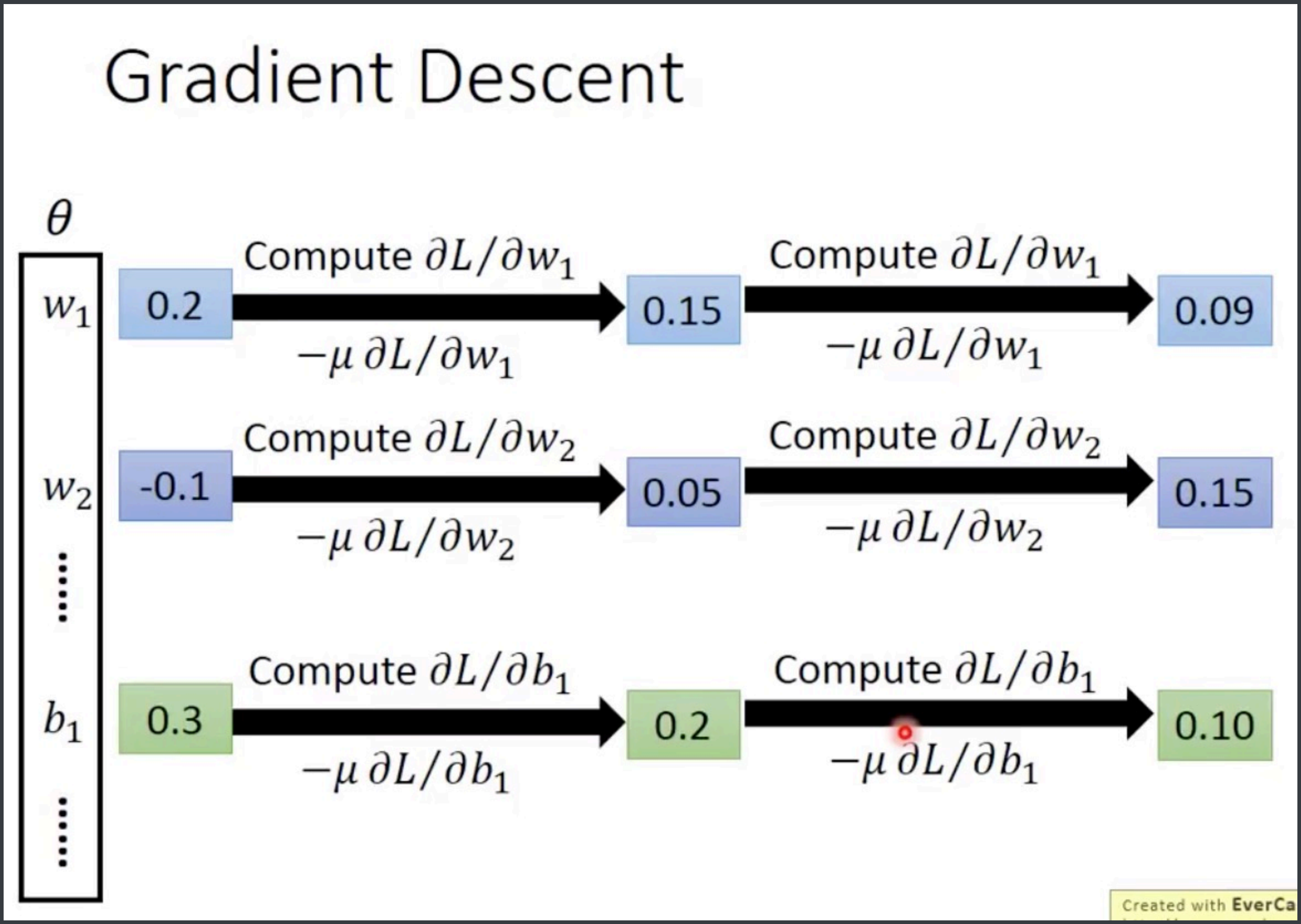
$\vdots$

b_1 $\quad 0.3 \xrightarrow[-\mu\,\partial L/\partial b_1]{\text{Compute } \partial L/\partial b_1} 0.2$

$\vdots$

$$\nabla L = \begin{bmatrix} \dfrac{\partial L}{\partial w_1} \\ \dfrac{\partial L}{\partial w_2} \\ \vdots \\ \dfrac{\partial L}{\partial b_1} \\ \vdots \end{bmatrix}$$

gradient



Gradient Descent

θ

w_1 0.2 Compute $\partial L/\partial w_1$ $-\mu\,\partial L/\partial w_1$ 0.15 Compute $\partial L/\partial w_1$ $-\mu\,\partial L/\partial w_1$ 0.09

w_2 -0.1 Compute $\partial L/\partial w_2$ $-\mu\,\partial L/\partial w_2$ 0.05 Compute $\partial L/\partial w_2$ $-\mu\,\partial L/\partial w_2$ 0.15

b_1 0.3 Compute $\partial L/\partial b_1$ $-\mu\,\partial L/\partial b_1$ 0.2 Compute $\partial L/\partial b_1$ $-\mu\,\partial L/\partial b_1$ 0.10

Backpropagation

An efficient way to compute

$$\frac{dL}{dW}$$

Packages:

TensorFlow, Torch, Theano, Caffe, CNTK, Chainer, DSSTNE, mxnet, liban

Resources on DL:

- ML and having it deep and structured
- Neural Networks and Deep Learning: Michael Nielsen
- Deep Learning: Yoshua Bengio, Ian J. Goodfellow and Aaron Courville