FIT3181 Deep Learning

Week 02: Prelude to Deep Learning and Feed-forward Neural Networks

Lecturer: Lim Chern Hong

Email: lim.chernhong@monash.edu

GROUP OF EIGHT AUSTRALIA

Department of Data Science and AI
Faculty of Information Technology, Monash University, Australia

Outline

- Prelude to Deep Learning
- Multi-layered feed-forward neural networks (NNs)
  - Evolution of feed-forward NNs
  - Parameterization of feed-forward NNs
  - Modern activation function
  - Forward propagation
- Computational graph of feed-forward neural networks
- Train feed-forward NNs

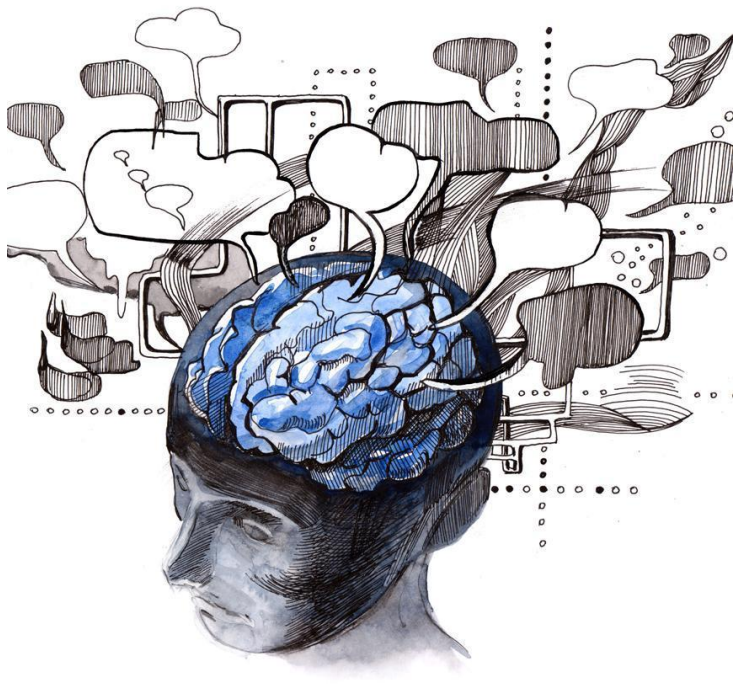
- **Further reading recommendations**
  - Dive into Deep Learning – Chapter 4: https://d2l.ai/chapter_multilayer-perceptrons/index.html
  - Deep Learning - Chapter 6
  - Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow – Chapter 10

Applications of Deep Learning

Prelude to Deep Learning

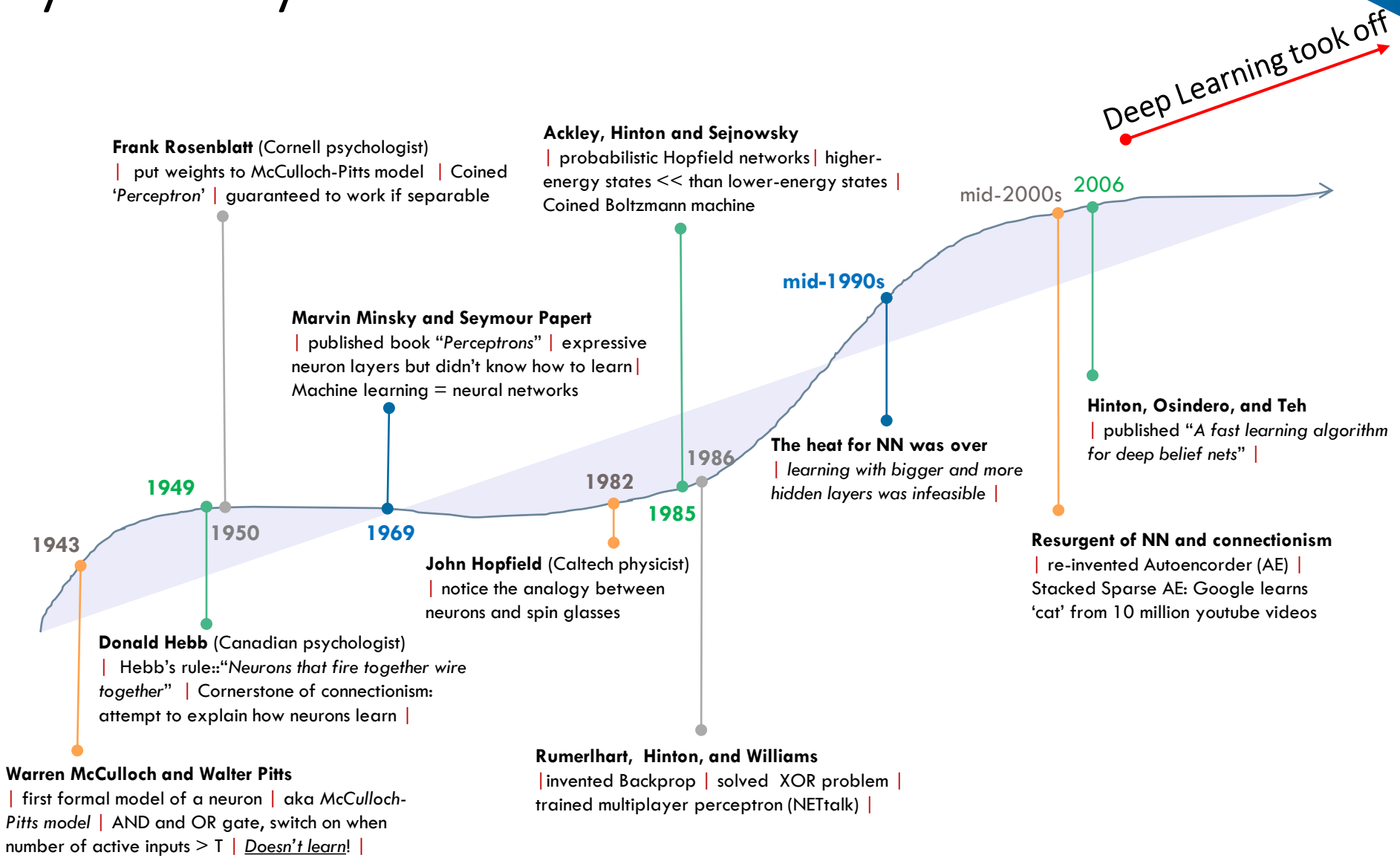
A Prelude to Deep Learning

- A young and fast moving field, but with history dated back to 40s (connectionism)!
- Rooted in machine learning and early pattern recognition theory.
- Drawn heavily on knowledge of human brain, statistics and applied maths
- Central technology to build Artificial Intelligence systems.



[Image courtesy: @DeepLearningIT]

Early history of DL

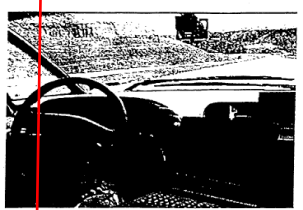


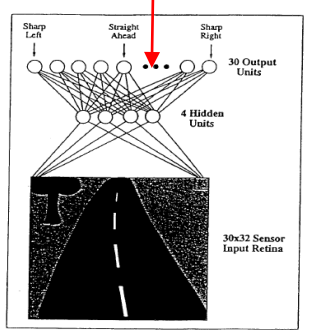
A prelude to deep learning (Forum discussion)

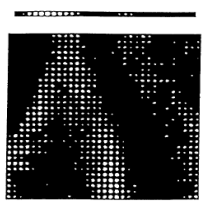
1990 — Dark age of DL — 2010

4 nodes, 3 layers

.. and TODAY, 152 – 1000 layers (Resnet)
GPT-3: 172 billions parameters







[*Machine Learning*, Tom Mitchell, 1997]

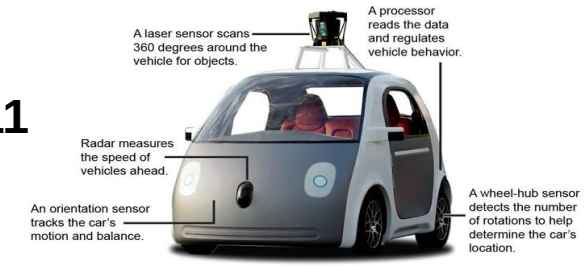
Early day of NN application

Advancements in hardware: GPU, TPU

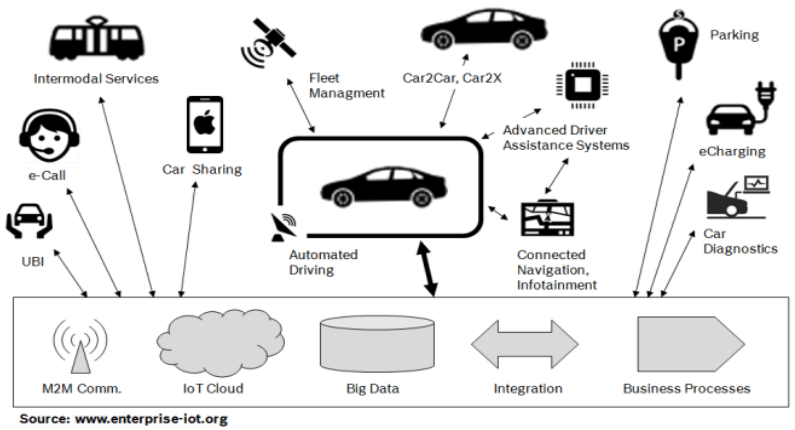
Massive datasets: ImageNet, MSCoco

Better techniques to train DL models:
- Adam, RMSProp
- ResNet, DenseNet
- BatchNorm, DropOut, MixUp

AlexNet won ImageNet 2011

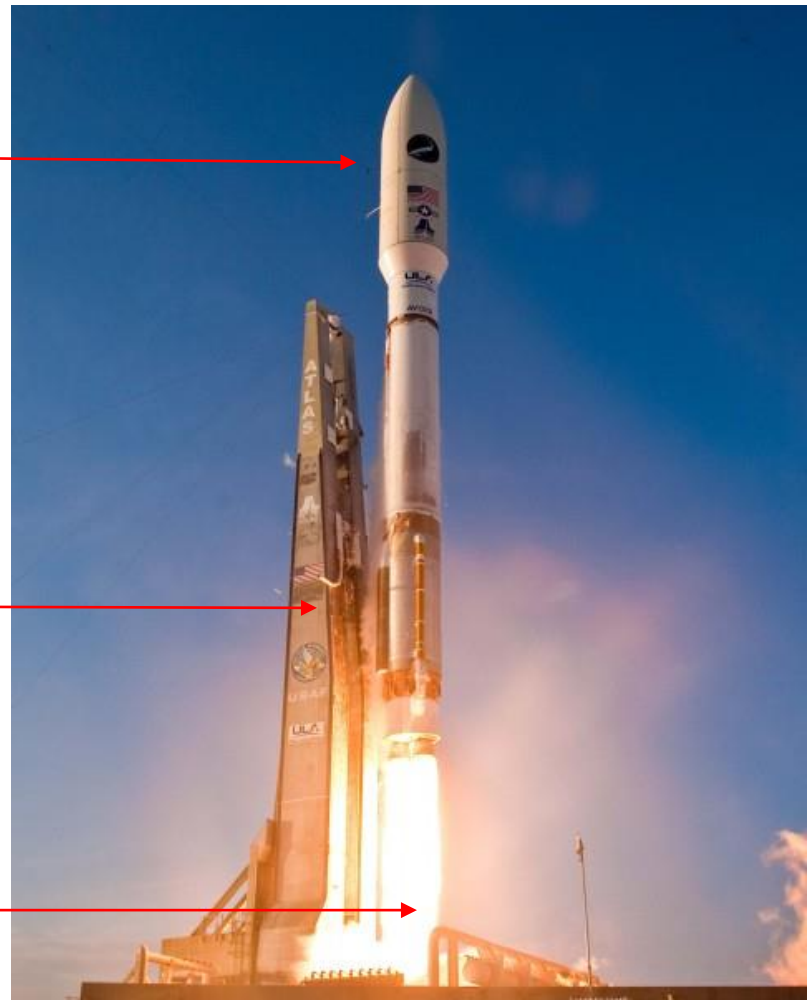


Truck,
Train,
Boat,
Uber,
UAV,
.....



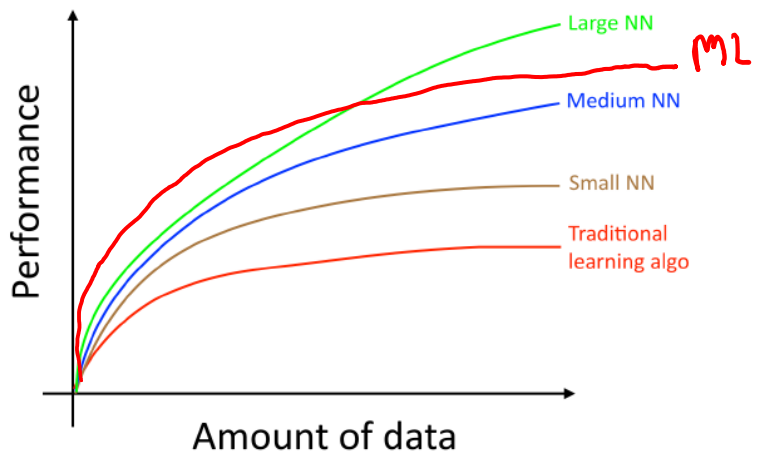
When DL works? (Forum discussion)

deep learning models



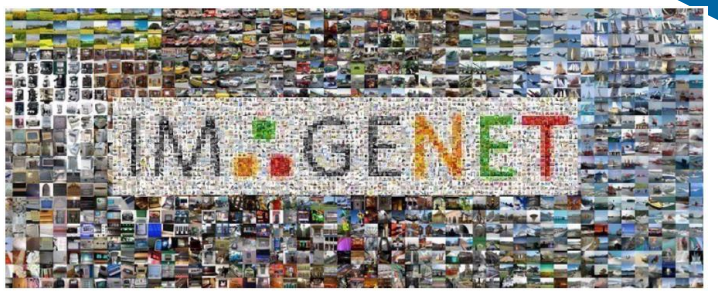
Computational infrastructure, researcher, environment and policy as the launch pad

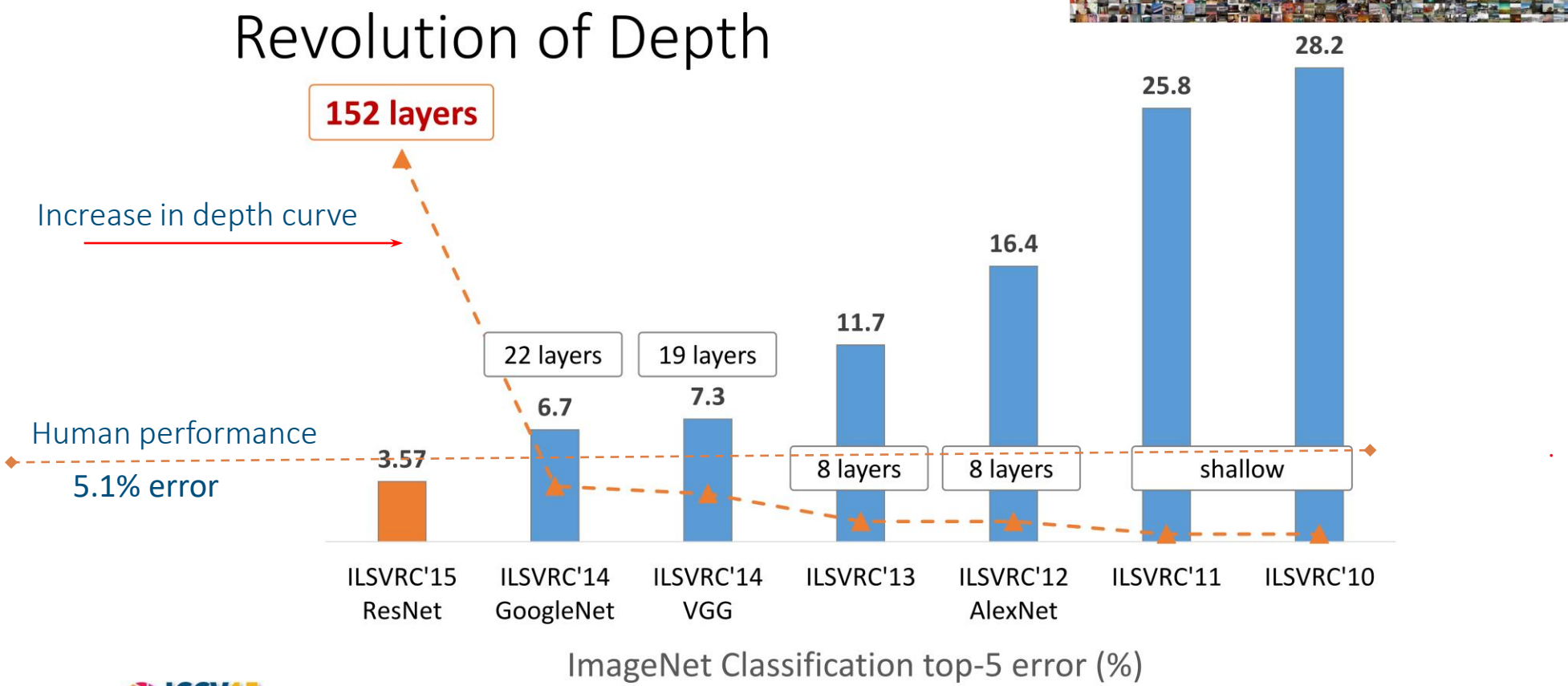
Big data as energy resource



Andrew Ng's Analogy

Breakthrough in object recognition





ICCV15 International Conference on Computer Vision

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

ResNet 2015 beats human performance in object recognition task!

MONASH University

Yann LeCun
Director of AI Research, Facebook

Yoshua Bengio
Head, Montreal Institute for Learning Algorithms

Geoff Hinton
Google Brain Toronto, University of Torronto

The ACM A.M. Turing Award is an annual prize given by the Association for Computing Machinery (ACM) to "an individual selected for contributions of a technical nature made to the computing community". The Turing Award is recognized as the "highest distinction in computer science" and "Nobel Prize of Computing". The award is named after Alan Mathison Turing, mathematician and reader in mathematics at the University of Manchester. From 2007 to 2013, the award was accompanied by a prize of $250,000. Effective 2014, the funding level has been increased to $1 million, i.e. four times the previous amount.

Deep learning

Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

nature

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

DL applications

Computer vision and big data

Speech Recognition & Computer Vision

Drug discovery & Medical Image Analysis

Recommender System

Google IBM Microsoft

MERCK NOVARTIS

amazon.com ebay NETFLIX

DL applications

Reinforcement learning

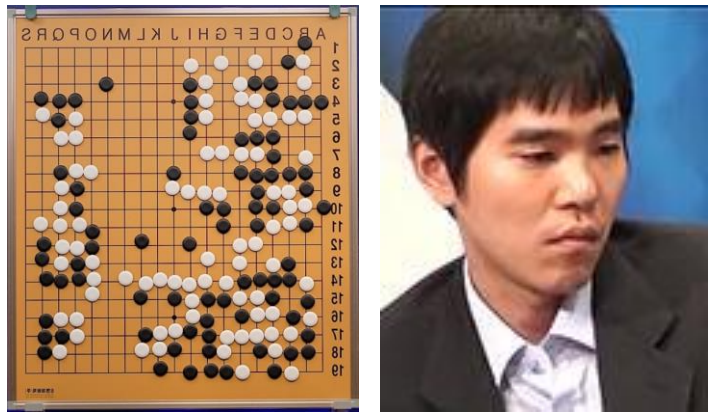
Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

DL applications

Reinforcement Learning

1997



2016

from search heuristics to ...

AUTOMATED LEARNING!

Other applications

Style transferred with CycleGAN

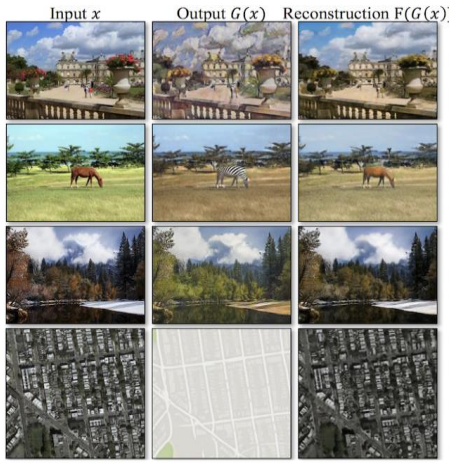
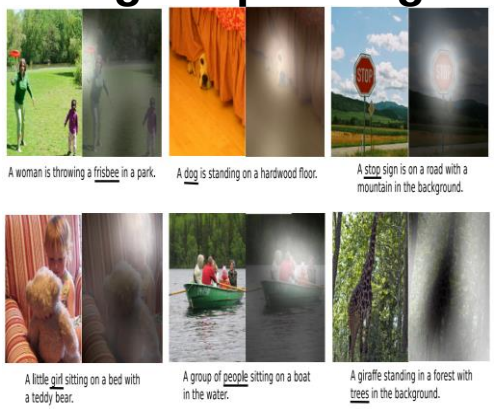
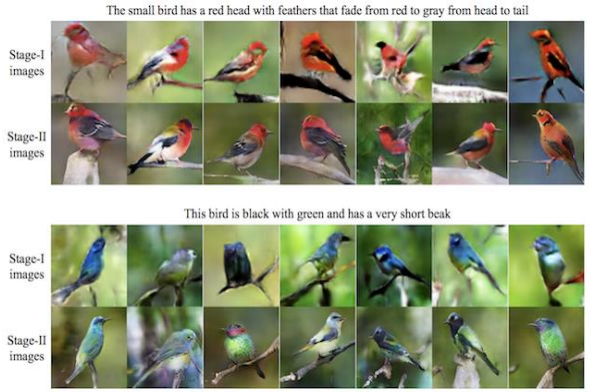


Image generation from Progressive GAN

Image captioning

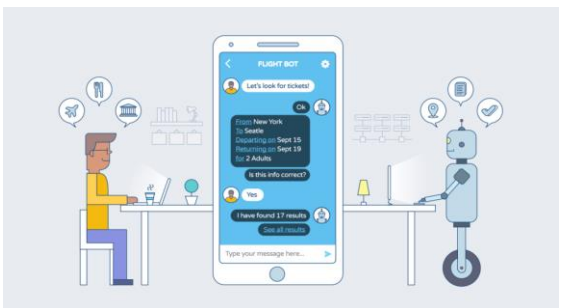


StackGAN: Text to Photo-realistic Image Synthesis



Machine translation with seq2seq

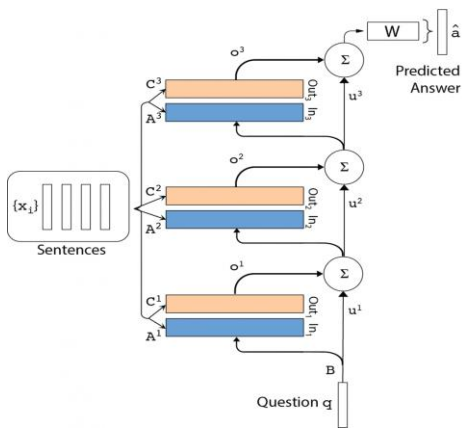
Chatbot



Brian is a lion.
Julius is a lion.
Julius is white.
Bernhard is green.
Q: What colour is Brian?
A: White.

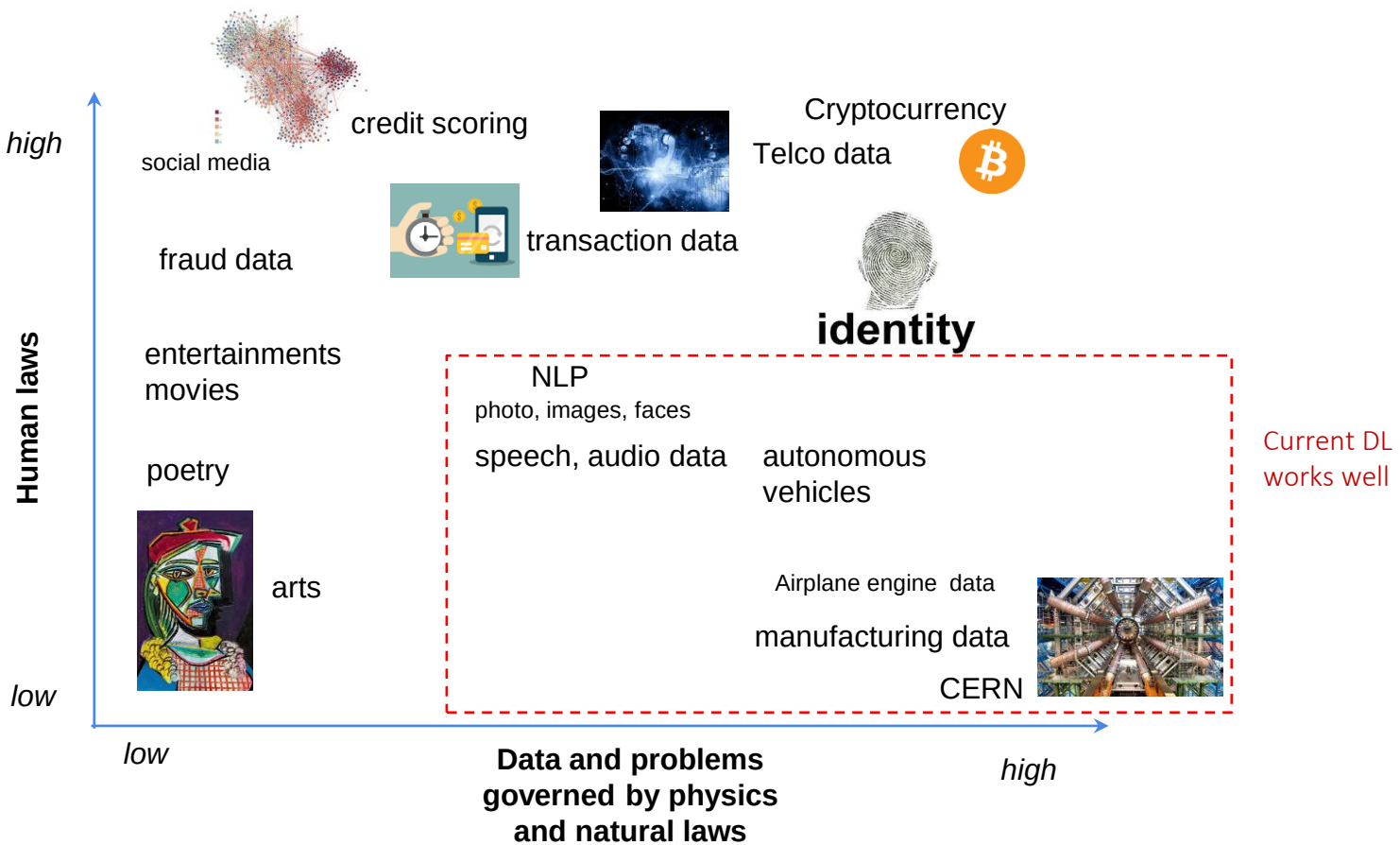
Sam picks up an apple.
Sam walks into the bedroom.
Sam drops the apple.
Q: Where is the apple?
A: Bedroom.

Memory network for reasoning



MONASH University

Where DL works and doesn't work?



Where does DL sit within ML field? (Forum Discussio

Approaches	Prob
Symbolists	Kno
Evolutionary	S
Connectionists	
Bayesian	
Analogizers	Similar

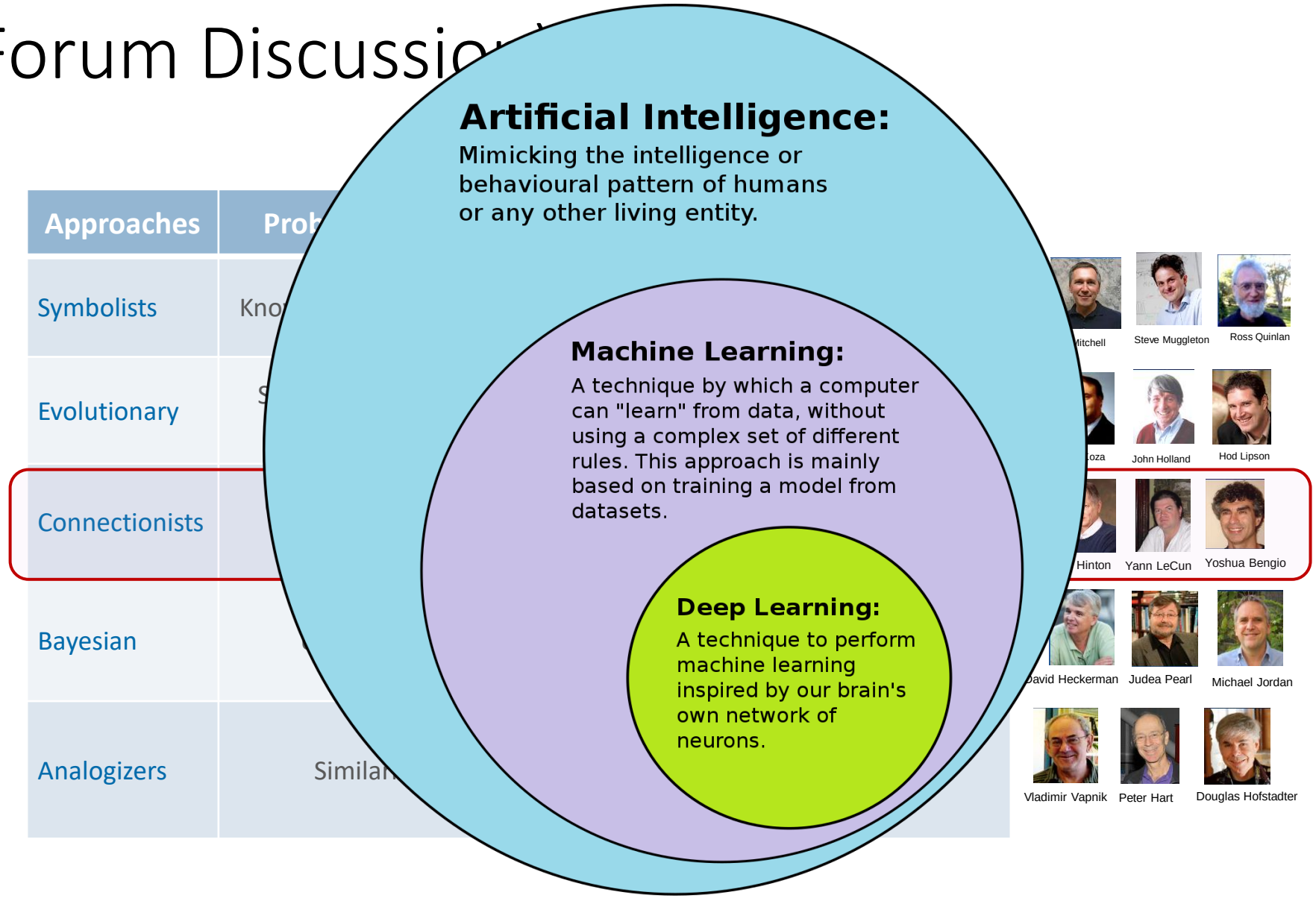


Image credit: Wikipedia

Source: “The Master Algorithm”, Domingo


Q&A
WHO?
WHERE?
WHEN?
WHAT?
How?
Why?
When?
What?
HOW?
Where?

Feed-forward Neural Networks

Discriminative machine learning (Forum discussion)

Data point x

Ground-truth label

$y = 8 \in \{1, .., 10\}$

feed

ML model h

$h(x)$ | $h_1(x)$, $h_2(x)$, ..., $h_{10}(x)$

softmax

Prediction probabilities

$p(x) = \text{softmax}(h(x))$

$p(x)$ | $p_1(x)$, $p_2(x)$, ..., $p_{10}(x)$

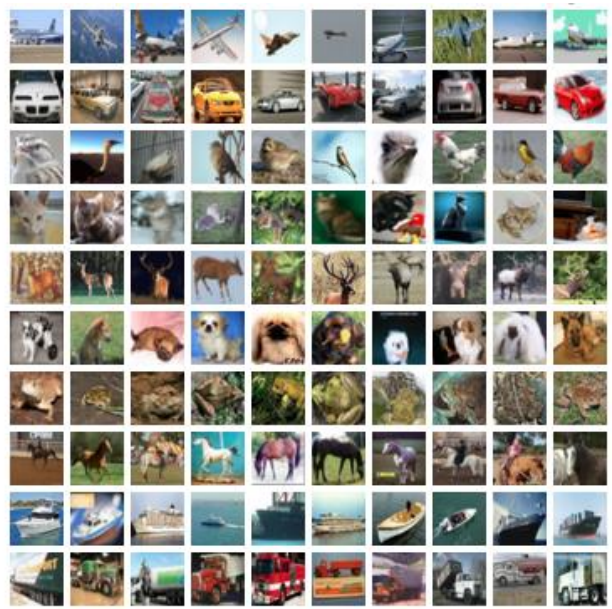
predict

$f(x) = \hat{y} := \text{argmax}_{1 \le m \le 10} p_m(x)$

Suffer **loss** $l(y, \hat{y}) = CE\left(1_y, p(x)\right) = -\log p_y$

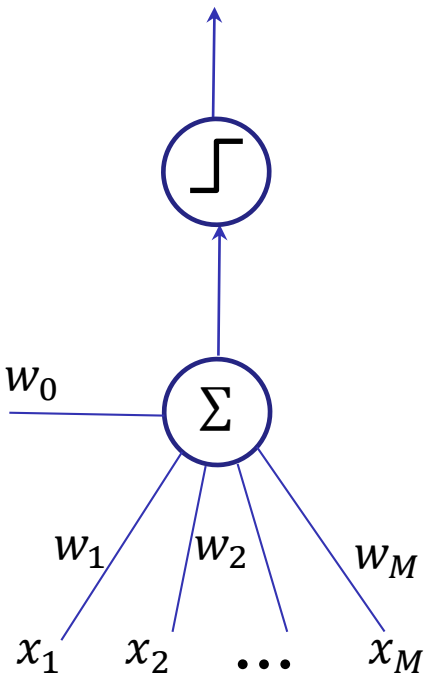
- $h_1 = -1, h_2 = h_3 = h_4 = h_5 = h_6 = 1, h_7 = 5, h_8 = 3, h_9 = h_{10} = 0$
  - Possibilities to assign data example x to classes.
- $p_1 = 0.002, p_2 = p_3 = p_4 = p_5 = p_6 = 0.015, p_7 = 0.8, p_8 = 0.11, p_9 = p_{10} = 0.005$
  - $\hat{y}$ =7≠y=8 → incorrect prediction.
  - $l(y, \hat{y}) = \text{CE}(1_y, p) = -\log p_y = -\log 0.11$

(1) airplane
(2) automobile
(3) bird
(4) cat
(5) deer
(6) dog
(7) frog
(8) horse
(9) ship
(10) truck

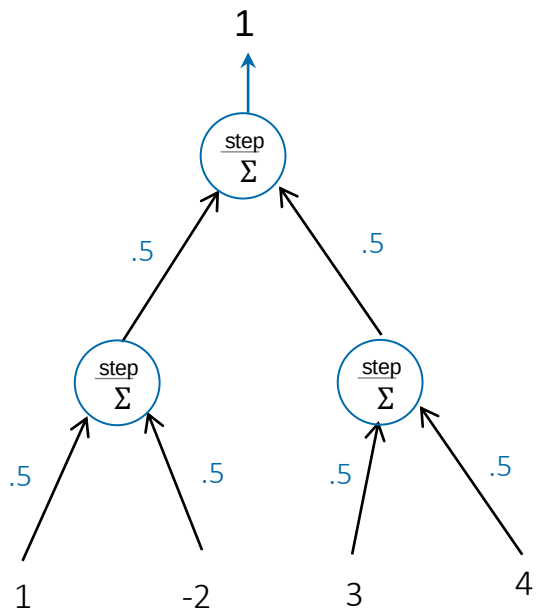


Cifar 10 dataset

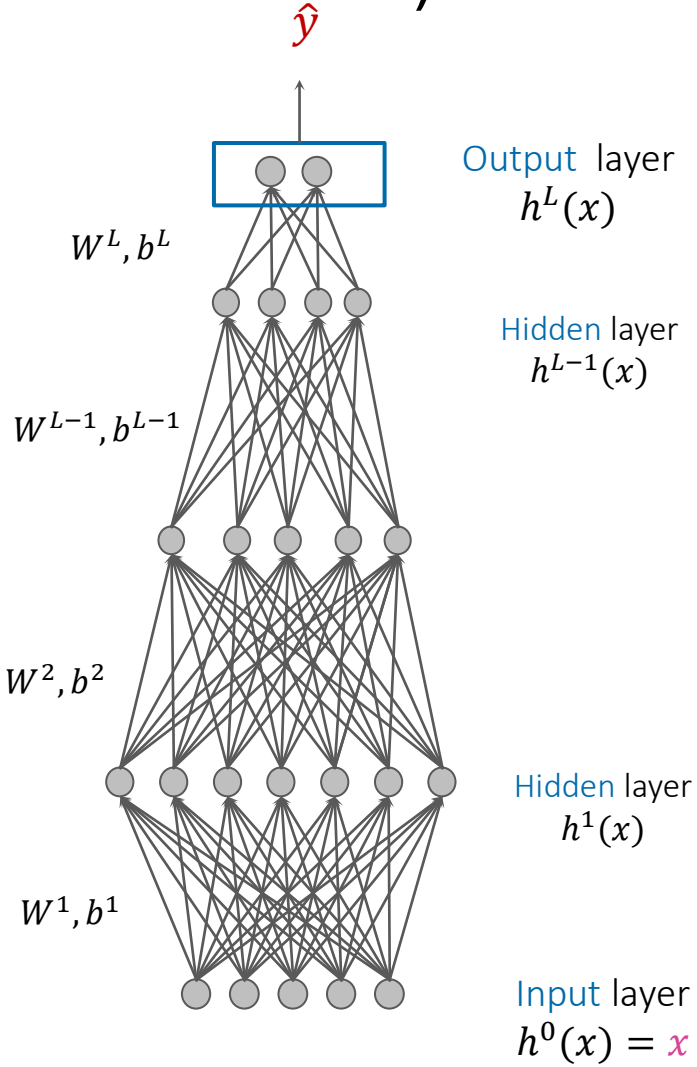
Evolution of deep learning models (Forum discussion)



Single Perceptron

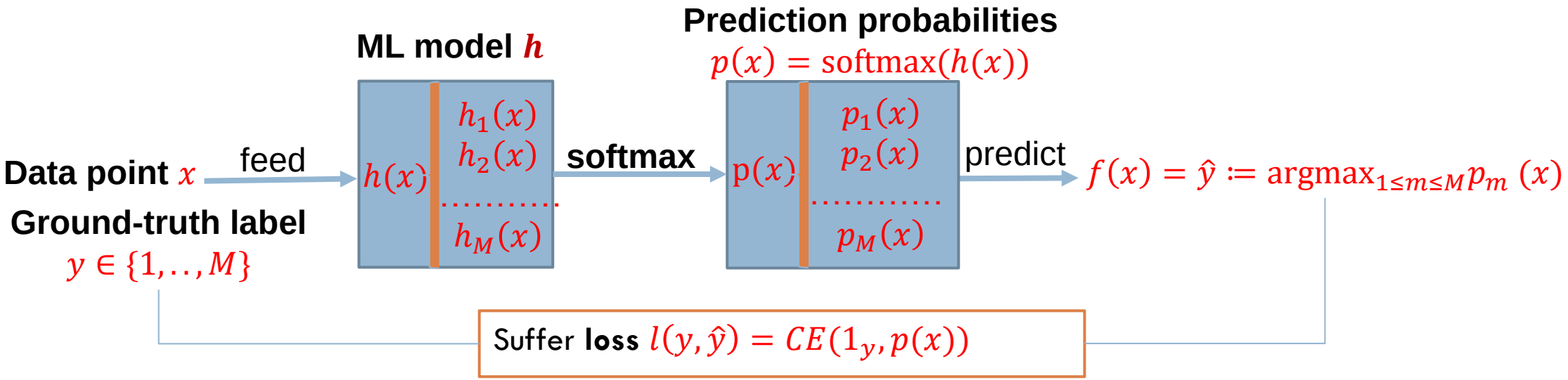


Multi-layered Perceptron

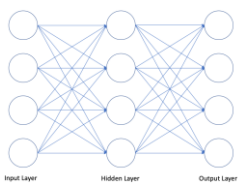


Multi-layered Feedforward NN

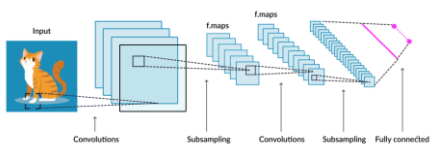
Deep learning (Forum discussion)



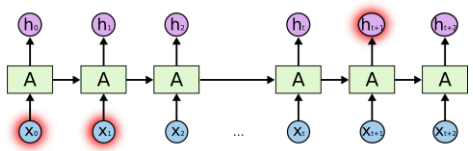
Deep neural networks with multiple neural layers and neurons



Multilayered feedforward nets



Convolutional neural nets



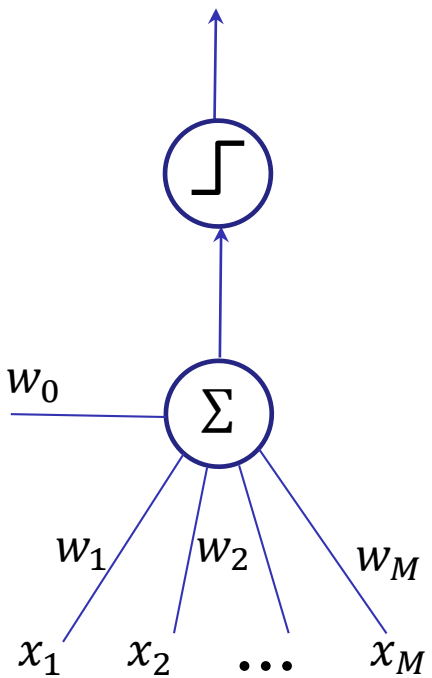
Recurrent neural nets

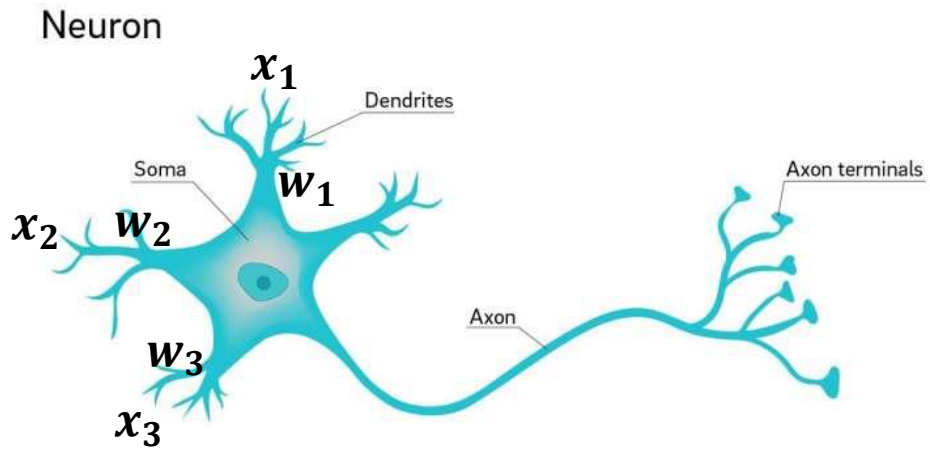
- **Three kinds of deep nets** will be **covered** in this unit:
  - Multilayered feedforward neural nets
    - For working with vectors, 1D tensors
  - Convolutional neural nets
    - For working with images, 2D/3D tensors
  - Recurrent neural nets
    - For working with sequences, sentence

Perceptron

$$step(z) = heaviside(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

output $y = step(\boldsymbol{w}^T.\boldsymbol{x} + \boldsymbol{w_0}) = step(w_0 + \sum_{i=1}^{M} w_i x_i)$





[Image source: medicalxpress.com]

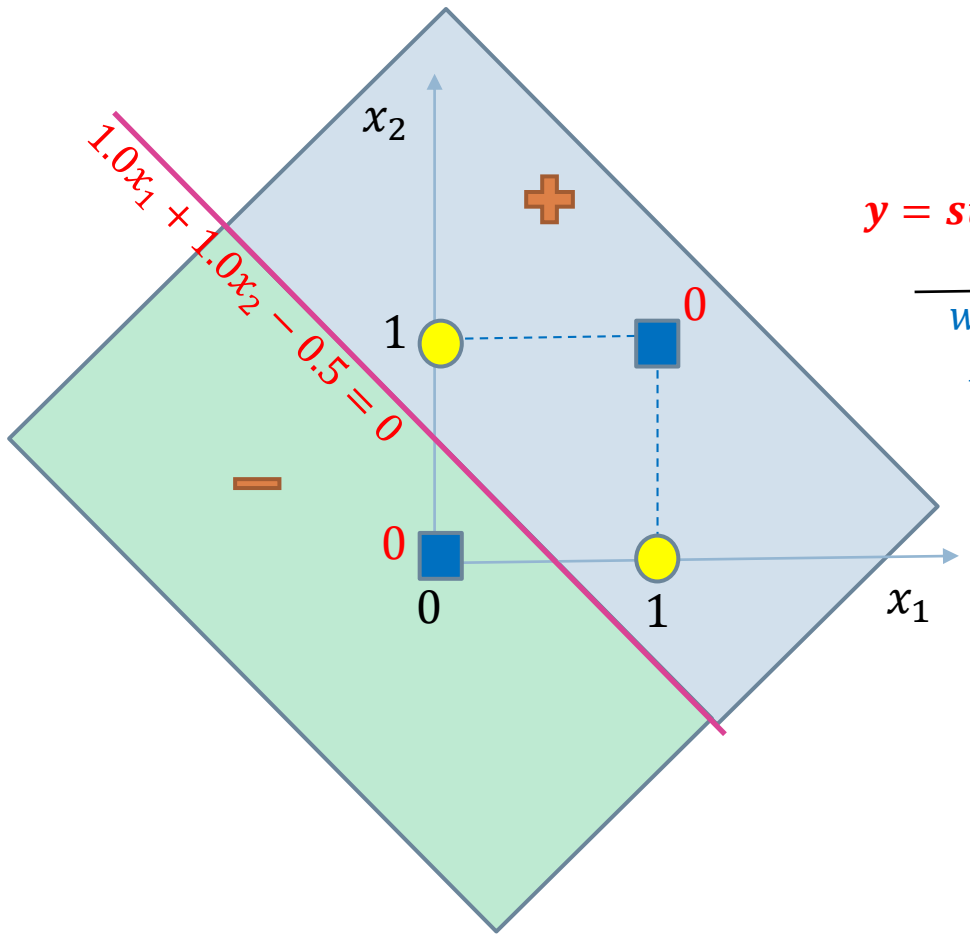
- Invented by Rosenblatt in 1957
- Single 'neuron' unit
- Perceptron guarantees to find a solution as long as data are linearly separable!

Exclusive OR (XOR) problem

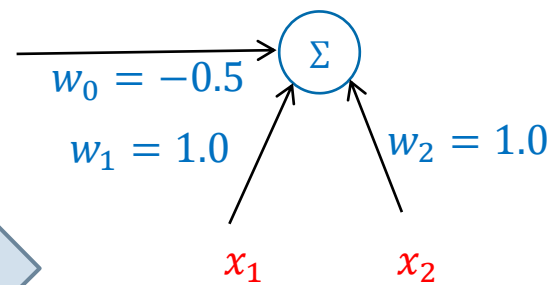
- x_1, x_2 are binary taking 0 or 1.
- x_1 XOR x_2 returns:
  - 1 if $x_1 \neq x_2$
  - 0 otherwise
- Ground-truth table

x_1	x_2	x_1 *XOR* x_2
0	0	0
0	1	1
1	0	1
1	1	0

Can you fit a straight line to separate blue squares and yellow circles?



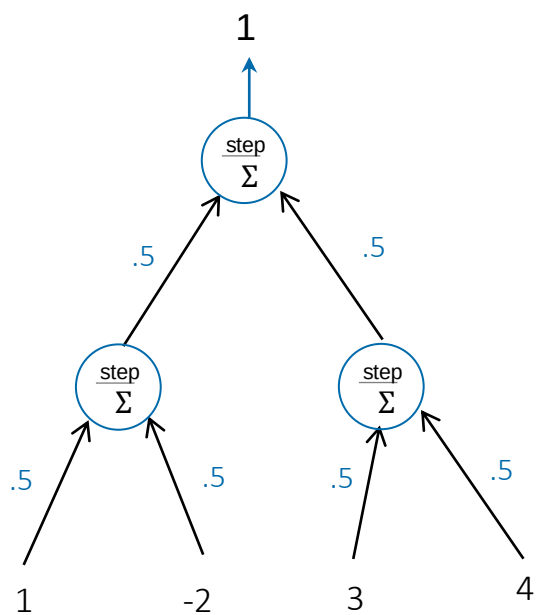
$$y = step(1.0x_1 + 1.0x_2 - 0.5)$$



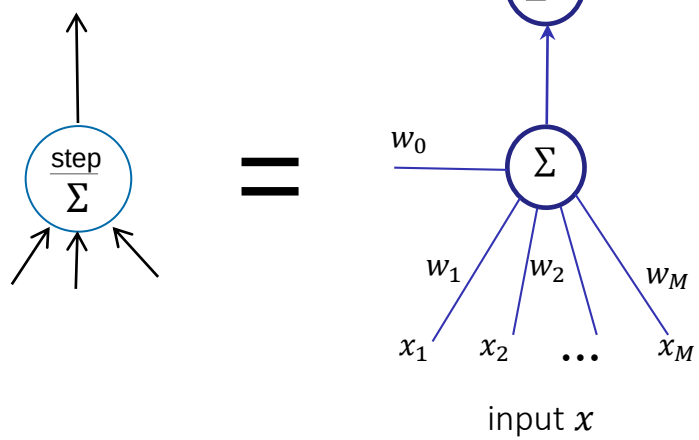
From perceptron to deep NNs

- Build deeper and richer models:
  - By stacking multiple perceptron on top of each others

The step function $\sum$ (activation function) in between is very crucial because it brings non-linearity to our network.



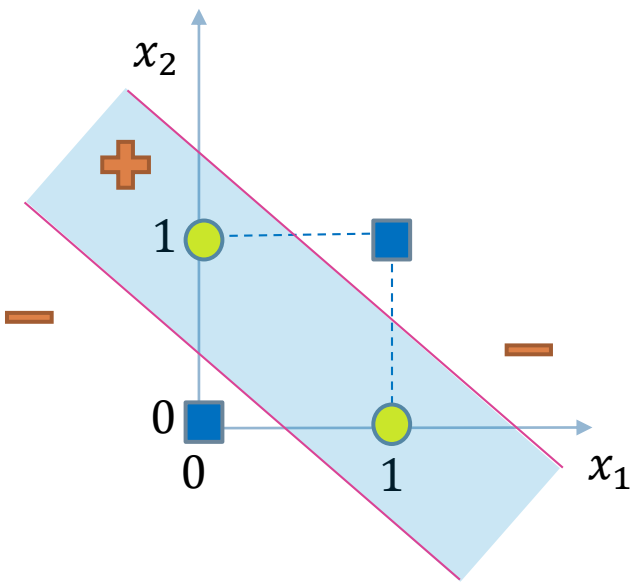
$$\text{e.g., heavidside}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

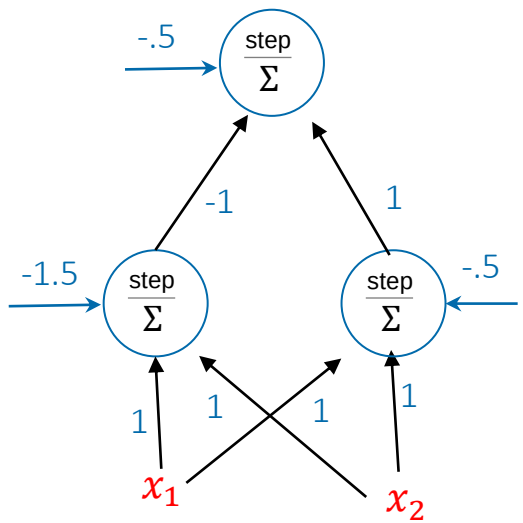


From perceptron to deep NNs

- Build deeper and richer models:
  - By stacking multiple perceptron on top of each others
  - This is called Multi-layer Perceptron (MLP)
  - MLP can solve the XOR problem!

$$step(-step(x_1 + x_2 - 1.5) + step(x_1 + x_2 - 0.5) - 0.5)$$

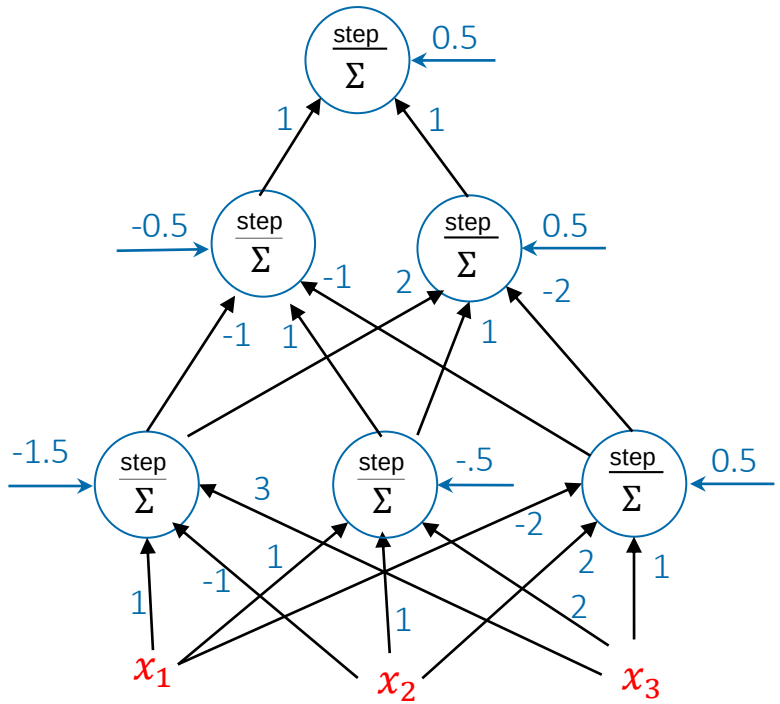




Exercise: *Verify that the MLP architecture above solves the XOR problem*!

From perceptron to deep NNs

- What are **more suitable** activation functions?
  - **step function** is hard to train and non-smooth.
- How to **parameterize** a deep NN and **do computation** more efficiently?



From step to sigmoid activation

1986

Learning Internal Representations by Error Propagation

D. E. RUMELHART, G. E. HINTON, and R. J. WILLIAMS

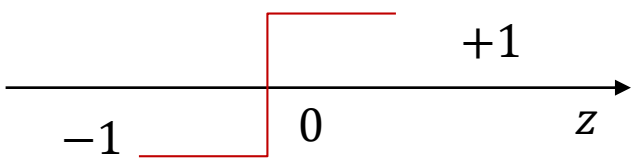
THE PROBLEM

We now have a rather good understanding of simple two-layer associative networks in which a set of input patterns arriving at an input layer are mapped directly to a set of output patterns at an output layer. Such networks have no *hidden* units. They involve only *input* and *output* units. In these cases there is no *internal representation*. The coding provided by the external world must suffice. These networks have proved useful in a wide variety of applications (cf. Chapters 2, 17, and 18). Perhaps the essential character of such networks is that they map similar input patterns to similar output patterns. This is what allows these networks to make reasonable generalizations and perform reasonably on patterns that have never before been presented. The similarity of patterns in a PDP system is determined by their overlap. The overlap in such networks is determined outside the learning system itself—by whatever produces the patterns.

- Deep Neural Networks

$$sign(z) = \begin{cases} -1 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

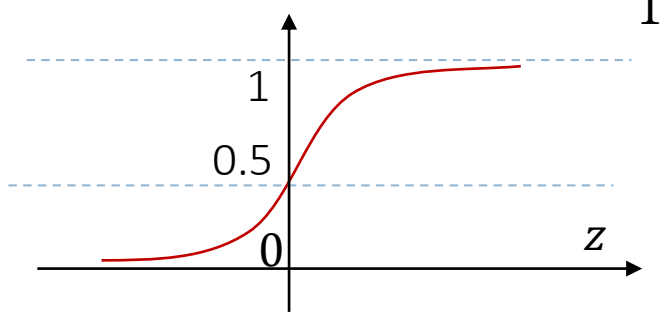
$\text{sign}(z)$

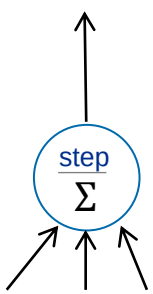


small step, but big leap in modelling

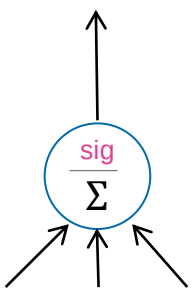
derivative is now well-defined everywhere!

$\text{sigmoid}(z)$ $s(z) = \frac{1}{1+e^{-z}}$



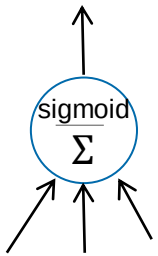


$$\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$$



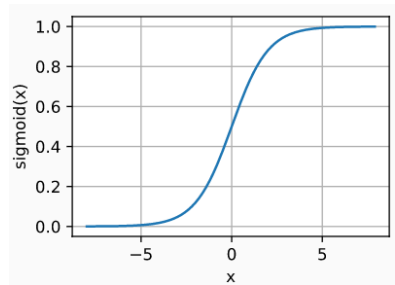
Exercise: *Re-derive the result for the derivative for the sigmoid function above.*

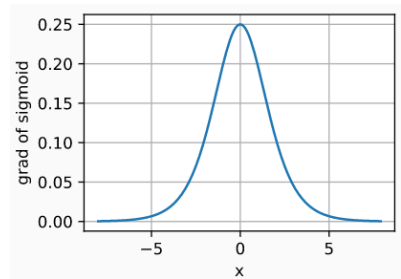
Modern Activation Functions

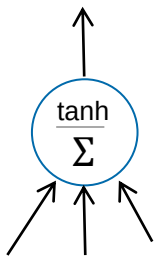


- The ***sigmoid*** function
  - $s(z) = \frac{1}{1+e^{-z}}$ `tf.math.sigmoid(z)`
  - Logistic S-shaped, continuous, and differentiable
  - Output range: 0 to 1

$$\sigma(z) = s(z) = \frac{1}{1+\exp\{-z\}} \qquad \sigma'(z) = \sigma(z)(1-\sigma(z))$$

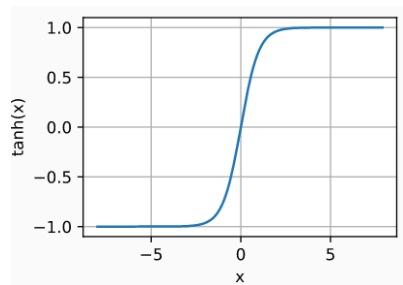


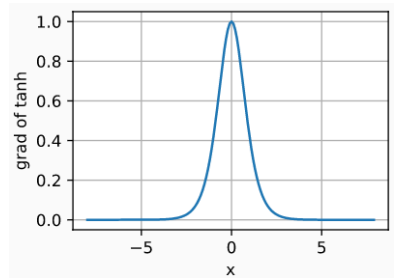


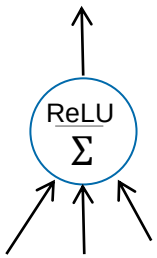


- The **hyperbolic *tangent*** function
  - $\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$ `tf.math.tanh(z)`
  - Logistic S-shaped, continuous, and differentiable
  - Output range: -1 to 1, hence help speed up convergence.

$$\sigma(z) = tanh(z) = \frac{\exp\{z\}-\exp\{-z\}}{\exp\{z\}+\exp\{-z\}} \qquad \sigma'(z) = 1-\sigma^2(z)$$

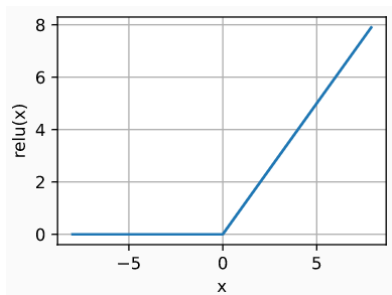


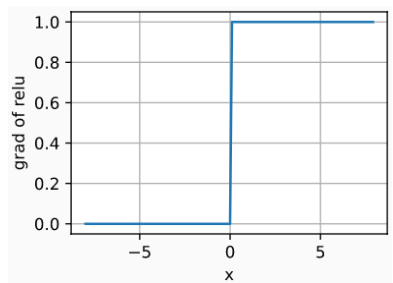




- The ***rectified linear function*** (ReLU)
  - $\mathrm{ReLU}(z) = \max(0, z)$ `tf.nn.relu(z)`
  - Continuous, but not differentiable at **0**
  - Hence, can make GD bounce around
  - Fast to compute, works very well in practice!
  - No cap on output value, hence help with gradient vanishing problem.

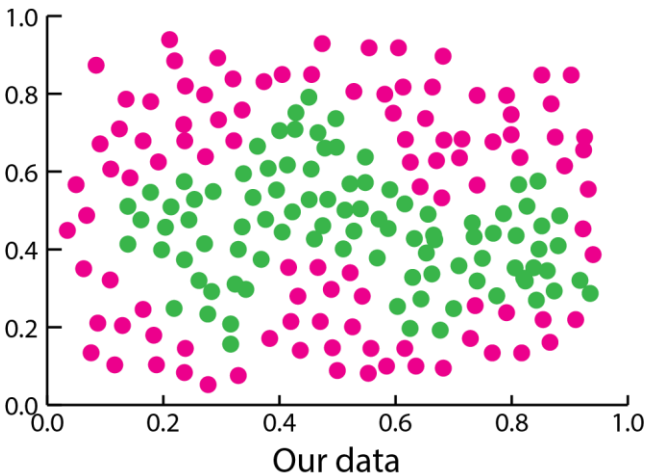
$$\sigma(z) = ReLU(z) = \max\{0, z\} \qquad \sigma'(z) = \begin{cases} 1 & if\ z \geq 0 \\ 0 & otherwise \end{cases}$$



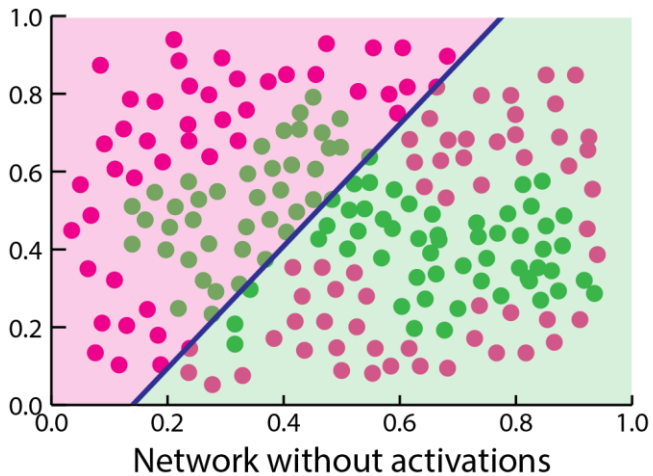


Importance of Activation Functions

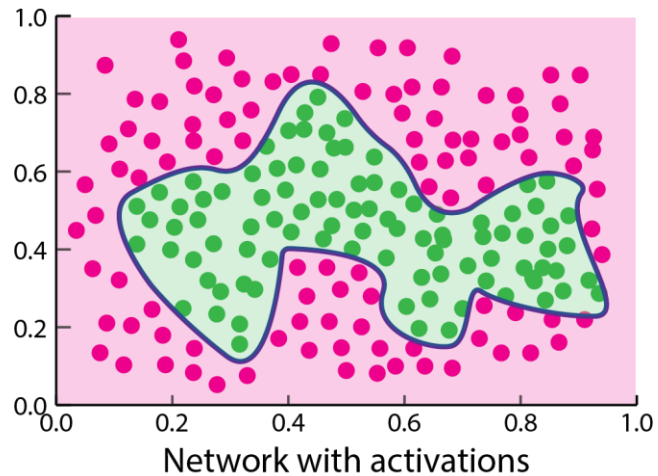
Aims to bring **non-linearities** to neural networks



Our data



Networks without activations



Networks with activations

Neuron and Layer views

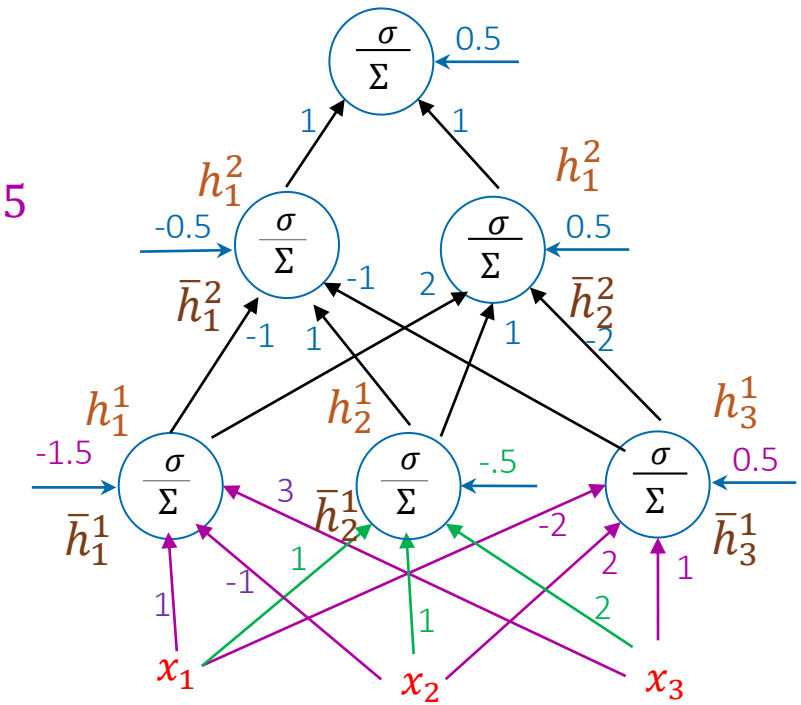
- Neuron (micro) view

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$W_1^1 = [1, -1, 3], b_1^1 = -1.5$$
$$\bar{h}_1^1 = W_1^1 x + b_1^1$$
$$h_1^1 = \sigma(\bar{h}_1^1)$$

$$W_2^1 = [1, 1, 2], b_2^1 = -0.5$$
$$\bar{h}_2^1 = W_2^1 x + b_2^1$$
$$h_2^1 = \sigma(\bar{h}_2^1)$$

$$W_3^1 = [-2, 2, 1], b_3^1 = 0.5$$
$$\bar{h}_3^1 = W_3^1 x + b_3^1$$
$$h_3^1 = \sigma(\bar{h}_3^1)$$



Neuron and Layer views

Neuron (micro) view

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$W_1^1 = [1, -1, 3], b_1^1 = -1.5$

$\bar{h}_1^1 = W_1^1 x + b_1^1$

$h_1^1 = \sigma(\bar{h}_1^1)$

$W_2^1 = [1, 1, 2], b_1^1 = -0.5$

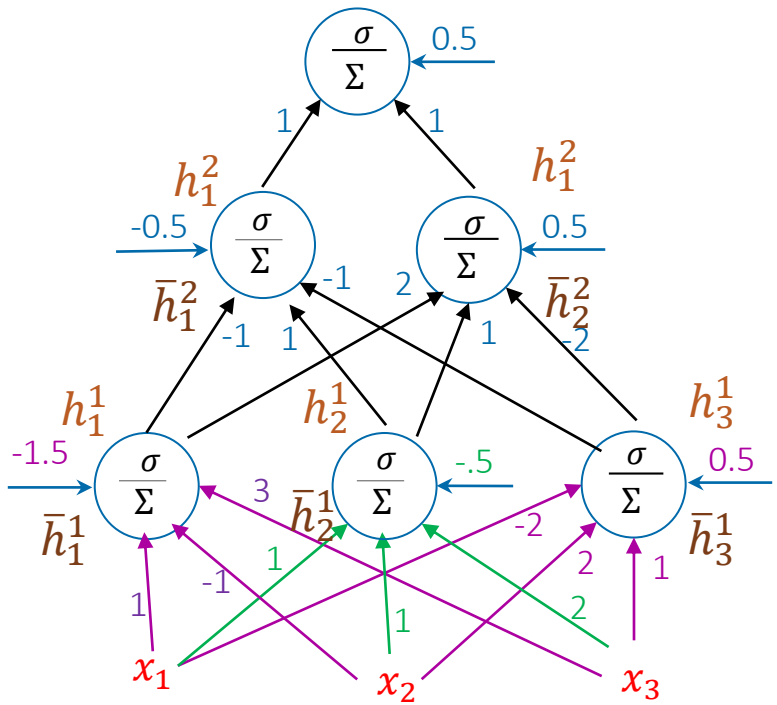
$\bar{h}_2^1 = W_2^1 x + b_2^1$

$h_2^1 = \sigma(\bar{h}_2^1)$

$W_3^1 = [-2, 2, 1], b_3^1 = 0.5$

$\bar{h}_3^1 = W_3^1 x + b_3^1$

$h_3^1 = \sigma(\bar{h}_3^1)$



Layer (macro) view

$$W^1 = \begin{bmatrix} 1 & -1 & 3 \\ 1 & 1 & 2 \\ -2 & 2 & 1 \end{bmatrix}, b^1 = \begin{bmatrix} -1.5 \\ -0.5 \\ 0.5 \end{bmatrix}$$

$$\bar{h}^1 = \begin{bmatrix} \bar{h}_1^1 \\ \bar{h}_2^1 \\ \bar{h}_3^1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 3 \\ 1 & 1 & 2 \\ -2 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} -1.5 \\ -0.5 \\ 0.5 \end{bmatrix}$$

$$= W^1 x + b^1$$

$$h^1 = \sigma(\bar{h}^1)$$

$$W^2 = \begin{bmatrix} * & * & * \\ * & * & * \end{bmatrix}, b^2 = \begin{bmatrix} * \\ * \end{bmatrix}$$

Neuron and Layer views

Neuron (micro) view

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$W_1^1 = [1, -1, 3], b_1^1 = -1.5$

$\bar{h}_1^1 = W_1^1 x + b_1^1$

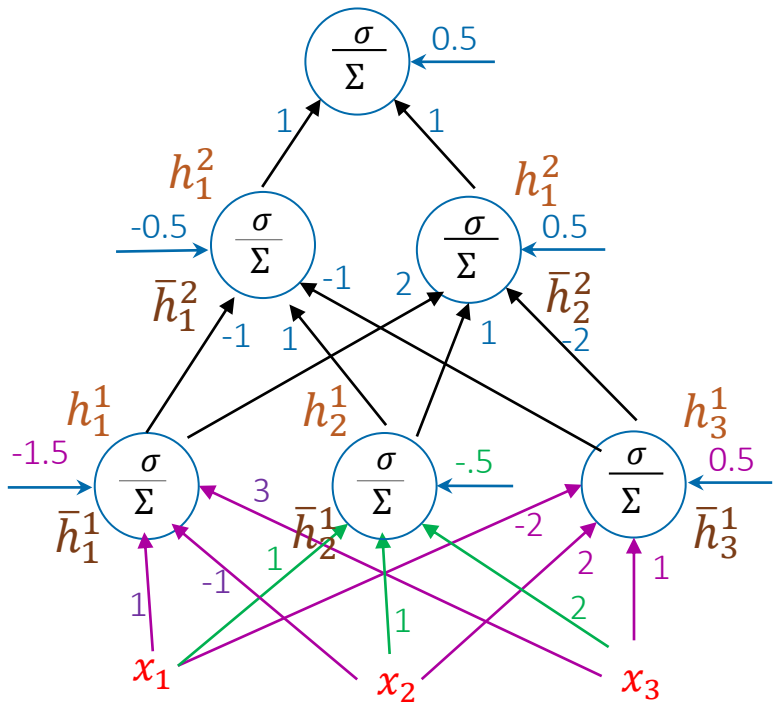
$h_1^1 = \sigma(\bar{h}_1^1)$

$\bar{h}_2^1 = W_2^1 x + b_2^1$

$h_2^1 = \sigma(\bar{h}_2^1)$

$\bar{h}_3^1 = W_3^1 x + b_3^1$

$h_3^1 = \sigma(\bar{h}_3^1)$



Layer (macro) view

$$W^1 = \begin{bmatrix} 1 & -1 & 3 \\ 1 & 1 & 2 \\ -2 & 2 & 1 \end{bmatrix}, b^1 = \begin{bmatrix} -1.5 \\ -0.5 \\ 0.5 \end{bmatrix}$$

$$\bar{h}^1 = \begin{bmatrix} \bar{h}_1^1 \\ \bar{h}_2^1 \\ \bar{h}_3^1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 3 \\ 1 & 1 & 2 \\ -2 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} -1.5 \\ -0.5 \\ 0.5 \end{bmatrix}$$

$$= W^1 x + b^1$$

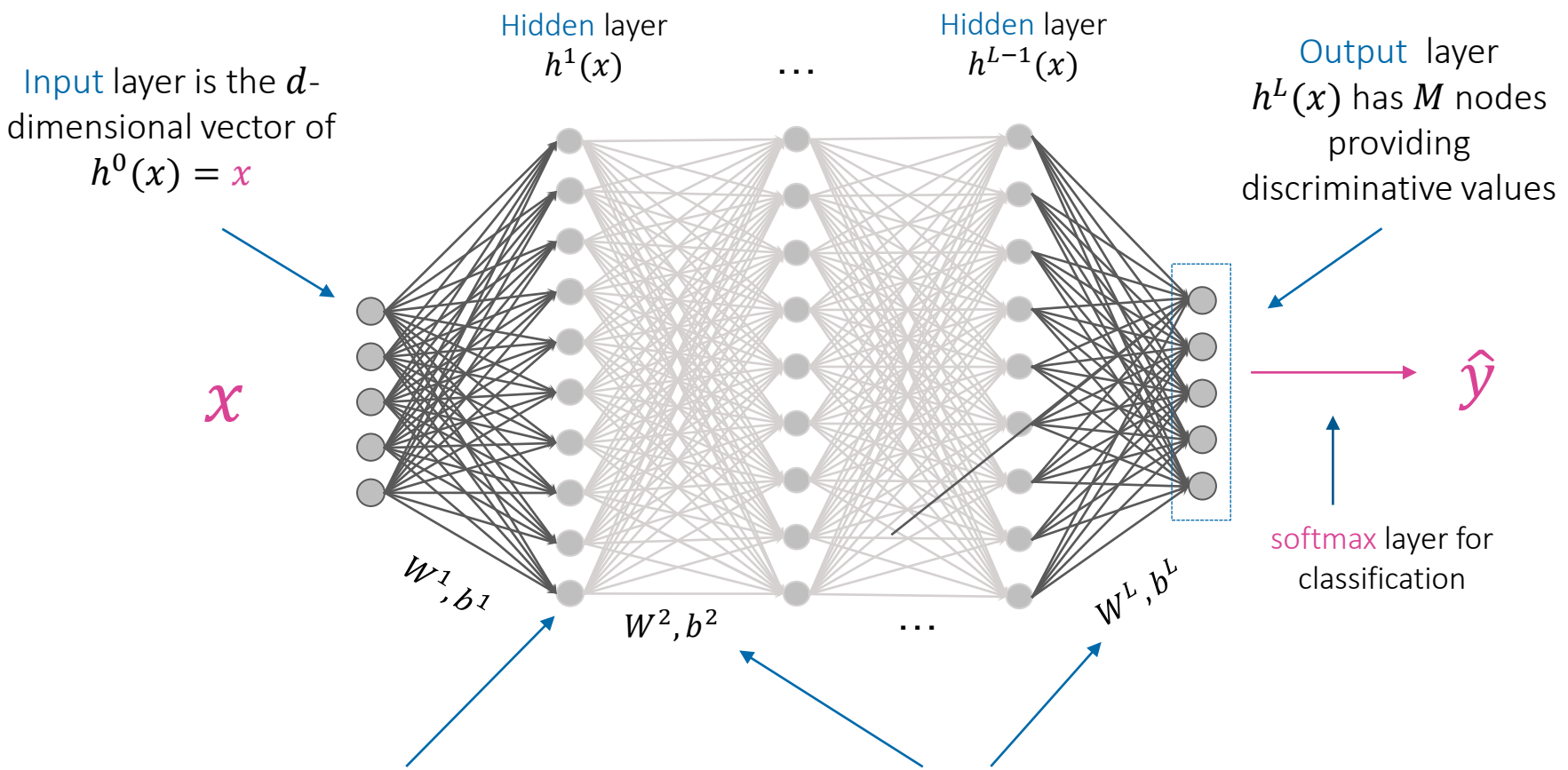
$$h^1 = \sigma(\bar{h}^1)$$

$$W^2 = \begin{bmatrix} -1 & 1 & -1 \\ 2 & 1 & -2 \end{bmatrix}, b^2 = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix}$$

$$\bar{h}^2 = \begin{bmatrix} \bar{h}_1^2 \\ \bar{h}_2^2 \end{bmatrix} = W^2 h^1 + b^2$$

$$h^2 = \sigma(\bar{h}^2)$$

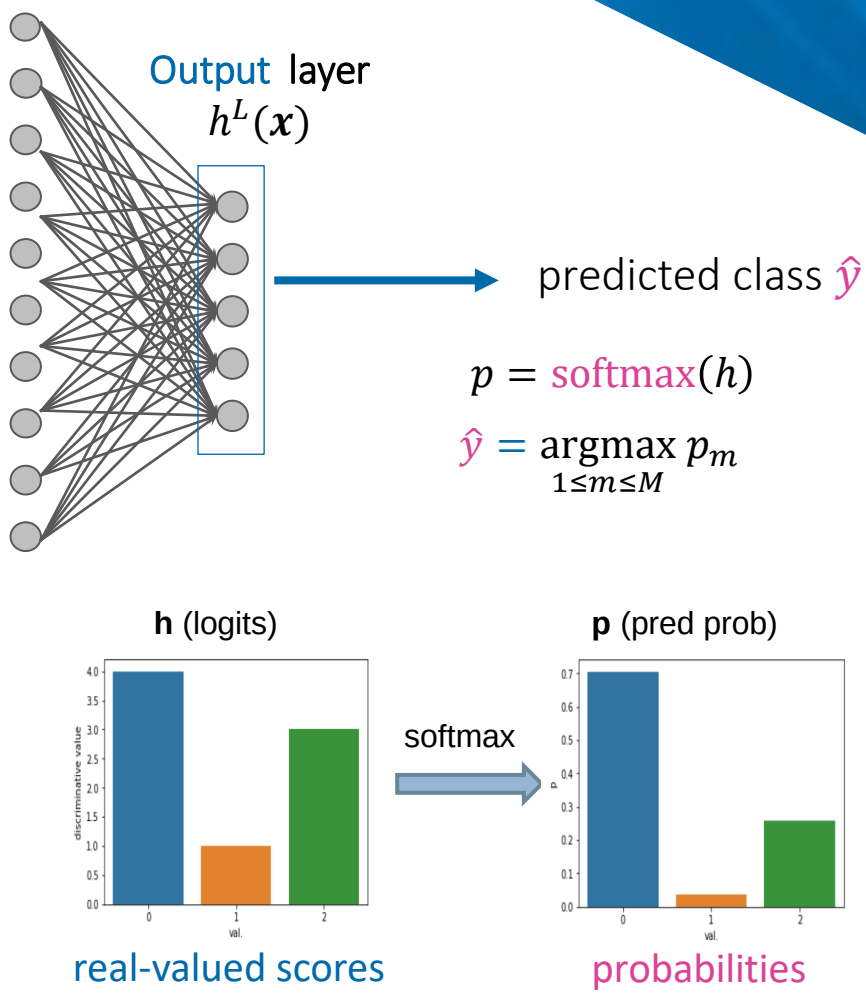
Deep NNs: parameterisation



- Layer k has n_k neurons
  - $n_0 = d$ and $n_L = M$
- Weight matrices and biases
  - $W^k \in \mathbb{R}^{n_k \times n_{k-1}}$ and $b^k \in \mathbb{R}^{n_k}$ for $k = 1,2,\dots,L$

What is a softmax layer?

- Softmax function transforms **real-valued discriminative scores h** to **discrete probabilities p**.
- Input: vector of dimension M
  - e.g., $h = [h_1, h_2, h_3] = [1, 2, 3]$, M $= 3$
- Output: a discrete distribution of dimension M
  - e.g., $p = [p_1, p_2, p_3] = [0.09, 0.24, 0.67]$
- How to calculate p_i?
  - $h = [h_m]_{m=1}^{M} \rightarrow p = \text{softmax}(h) := \left[\frac{\exp\{h_m\}}{\sum_{i=1}^{M}\exp\{h_i\}}\right]_{m=1}^{M}$
  - $p = [p_m]_{m=1}^{M}$ becomes a distribution over classes $\{1, \dots, M\}$
    - $p_m \geq 0$ $(1 \leq m \leq M)$ and $\sum_{m=1}^{M} p_m = 1$.



transforms real-valued discriminative scores (ranged in $(-\infty, +\infty)$) to probability values (ranged in $[0, 1]$) but preserving the order.

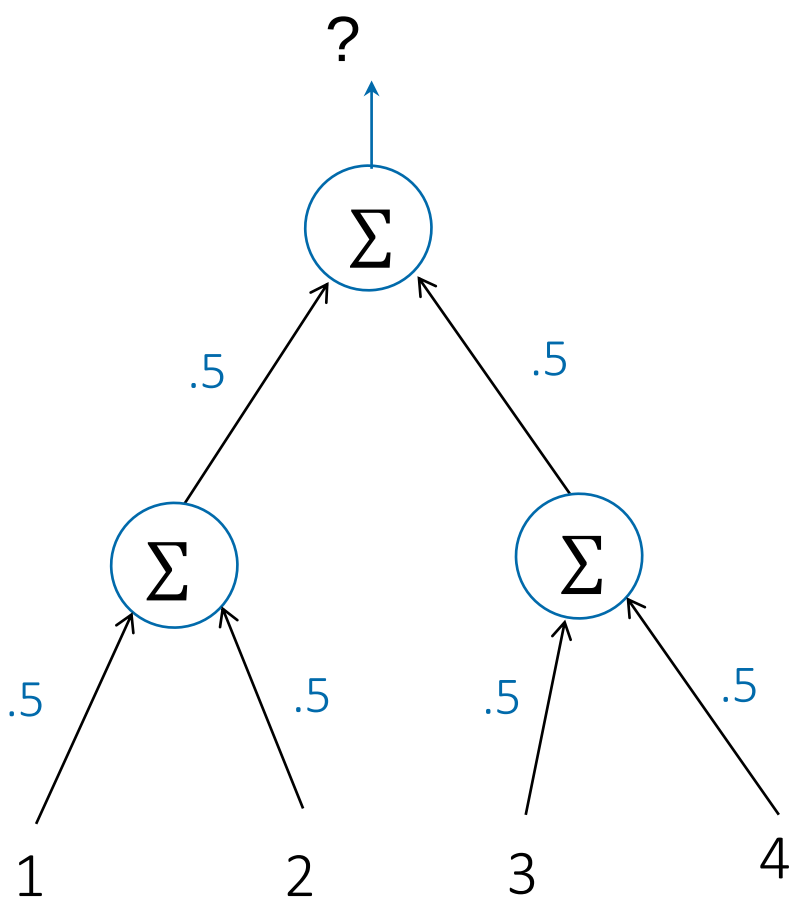
- Prediction step: return the prediction for the class that has the highest probability

$$\hat{y} = \underset{1 \leq m \leq M}{\text{argmax}}\, p_m$$

  - i.e., return 3 in above example (why?)

DNNs: making prediction via FP

Forward propagation

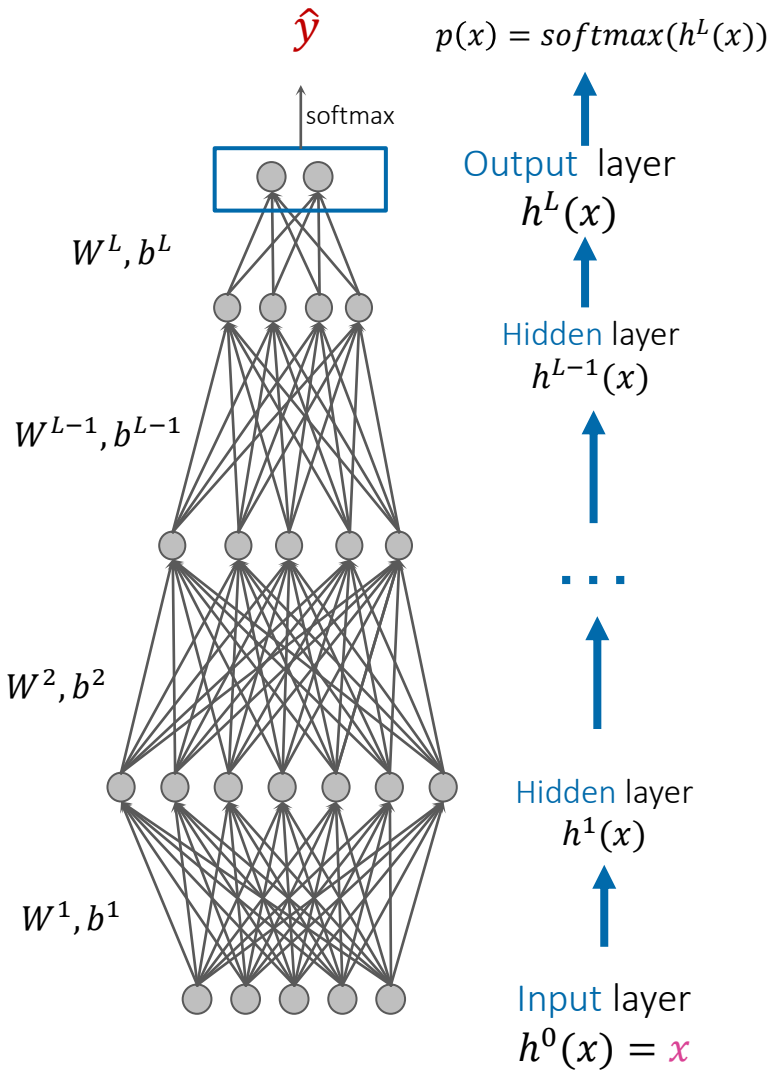


- Supposed we know models and all of its parameters
  - E.g., deploy a trained DNNs to real world
- Given an input x, make a prediction $\hat{y}$ given the model above.

DNNs: making prediction via FP

Forward propagation- Classification

$h^0(x) = x$
for $k = 1$ **to** $L - 1$ **do**
  $\bar{h}^k = W^k h^{k-1}(x) + b^k$ //linear operation
  $h^k(x) = \sigma(\bar{h}^k(x))$ //activation
$h^L(x) = W^L h^{L-1}(x) + b^L$
$p(x) = softmax(h^L(x))$ //prediction probabilities

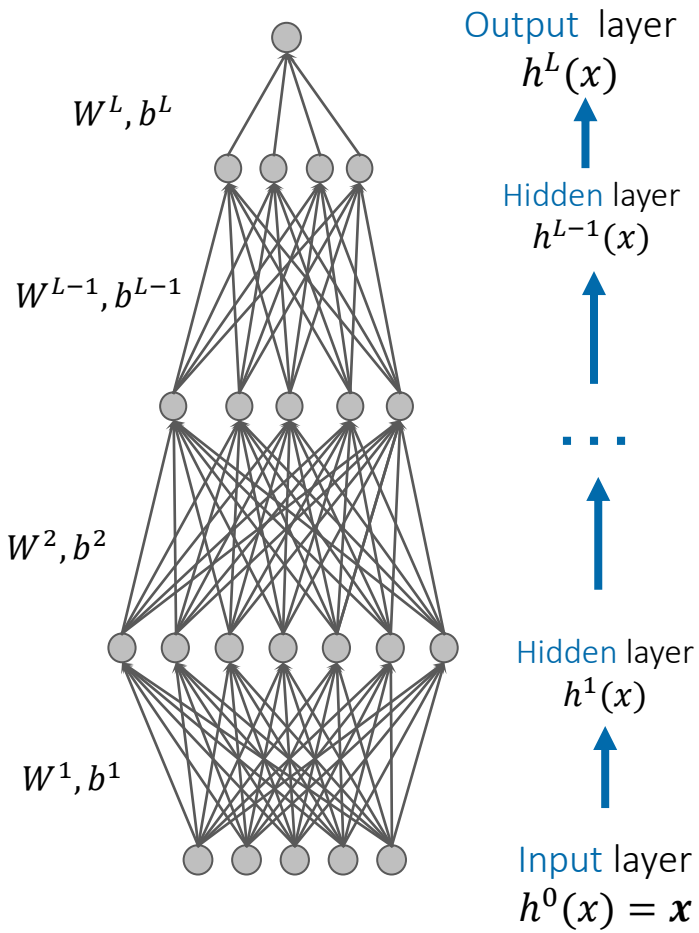


- Parameter: $\theta = (W^k, b^k)_{k=1}^L$ is given
- $\sigma: \mathbb{R} \to \mathbb{R}$ is the activation function
  - Sigmoid, tanh, ReLU
- Output: $p(x) = \text{softmax}(h^L(x))$
  - $\Pr(y = k|x) = p_k(x) = \frac{\exp\{h_k^L(x)\}}{\sum_{j=1}^M \exp\{h_j^L(x)\}}$
  - $\hat{y} = \underset{1 \le k \le M}{\text{argmax}}\, p_k(x)$

DNNs: making prediction via FP

Forward propagation- Regression

$h^0(x) = x$
for $k = 1$ **to** $L - 1$ **do**
  $\bar{h}^k = W^k h^{k-1}(x) + b^k$ //linear operation
  $h^k(x) = \sigma(\bar{h}^k(x))$ //activation
$h^L(x) = W^L h^{L-1}(x) + b^L$



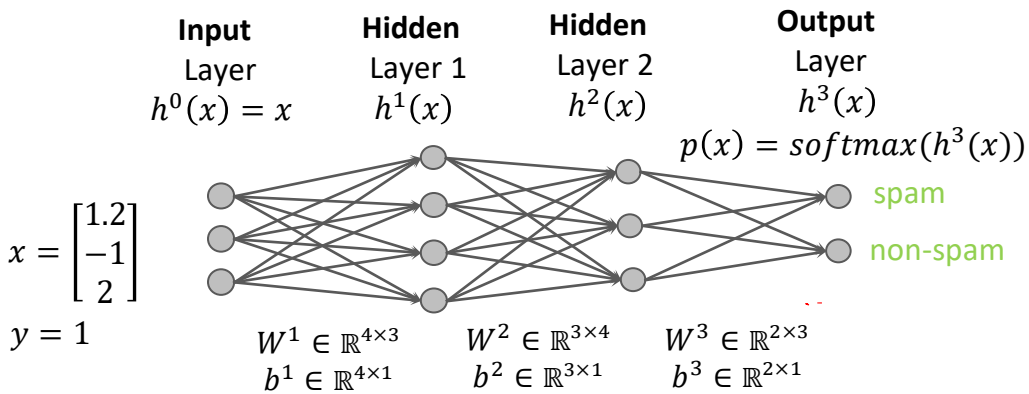
- Parameter: $\theta = \left(W^k, b^k\right)_{k=1}^{L}$
- $\sigma: \mathbb{R} \to \mathbb{R}$ is the activation function
  - Sigmoid, tanh, ReLU
- Output:
  - $\hat{y} = h^L(x)$

DNNs: making prediction via FP

Example of spam email detection

- From emails, assume that we extract **three features**, hence data points $x \in \mathbb{R}^3$
  - $x = [x_1, x_2, x_3]^T \in \mathbb{R}^3$
  - There are **two** classes and labels: **spam** $(y = 1)$ and **non-spam** $(y = 2)$

- Network architecture:
  - $3 \to 4(sigmoid) \to 3(sigmoid) \to 2(softmax)$
- $h^0(x) = x = [1.2, -1, 2.2]^T \in \mathbb{R}^{3\times 1}$
- Hidden layer 1
  - $\bar{h}^1(x) = W^1 h^0(x) + b^1 \in \mathbb{R}^{4\times 1}$ *(linear operation)*
  - $h^1(x) = sigmoid\left(\bar{h}^1(x)\right) = \left[sigmoid\left(\bar{h}_i^1(x)\right)\right]_{i=1}^{4} \in \mathbb{R}^{4\times 1}$ *(activation)*
- Hidden layer 2
  - $\bar{h}^2(x) = W^2 h^1(x) + b^2 \in \mathbb{R}^{3\times 1}$ *(linear operation)*
  - $h^2(x) = sigmoid\left(\bar{h}^2(x)\right) \in \mathbb{R}^{3\times 1}$ *(element-wise activation)*
- Output layer
  - $h^3(x) = W^3 h^2(x) + b^3 \in \mathbb{R}^{2\times 1}$ *(linear operation)*
  - $p(x) = softmax\left(h^3(x)\right) \in \mathbb{R}^{2\times 1}$ *(softmax activation)*
- Assume that $p(x) = [0.3, 0.7]^T$
  - Prediction $\hat{y} = 2 \neq y = 1 \to$ ***incorrect*** *prediction*



$h^0(x) = x$
for $k = 1$ **to** 2 **do**
  $\bar{h}^k = W^k h^{k-1}(x) + b^k$ //linear operation
  $h^k(x) = \sigma\left(\bar{h}^k(x)\right)$ //activation
$h^3(x) = W^3 h^2(x) + b^2$
$p(x) = softmax(h^3(x))$ //prediction probabilities

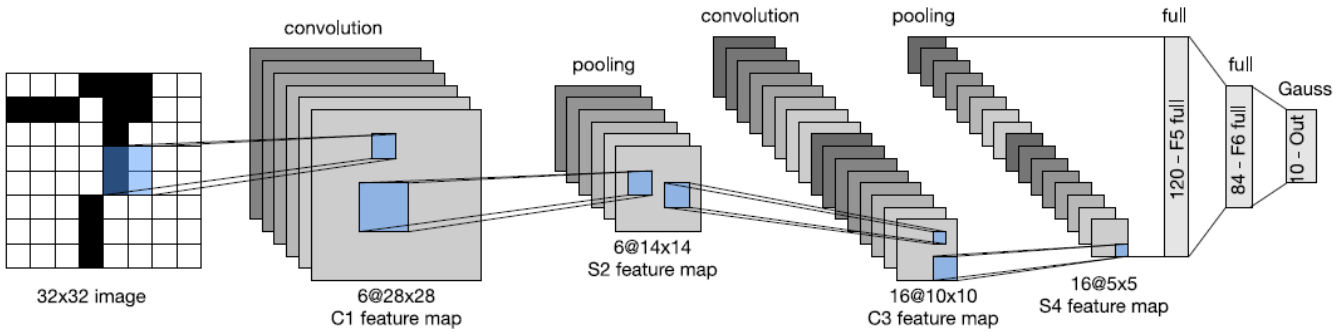
Exercise: *check all the dimensions for matrix multiplication consistency!*
What is the cross-entropy loss in this case?

Modern deep NNs

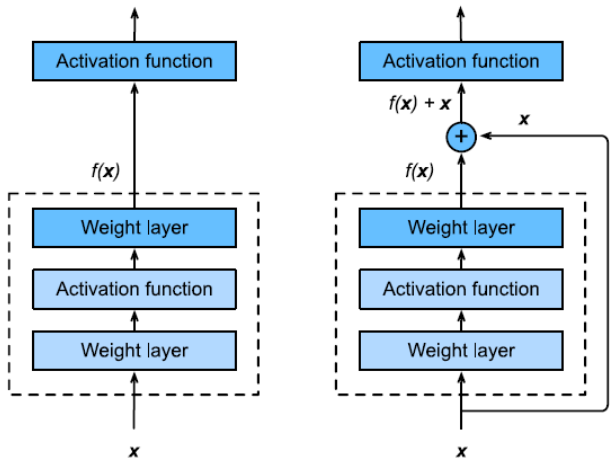
- Deep neural networks (DNN)
  - 1986 Backpropagation was introduced!
  - Step activation function was replaced with sigmoid function
  - Derivative exists everywhere, gradient method can be applied!
- Modern Deep NNs
  - Deep NNs can easily overfit
    - Before big data time: less data to train
    - Can't go deep: easier to overfit and gradient vanishing!
  - Modern techniques:
    - Activation functions plays a vital role
      - ReLU, Tanh, Sigmoid
    - New strategies to combat overfitting and gradient vanishing: regularization
      - dropout, batch-norm,
    - Better optimization methods for nonconvex problems: Adam, RMSProp, etc.

Modern Deep NNs

LeNet



Residual Networks (ResNet)



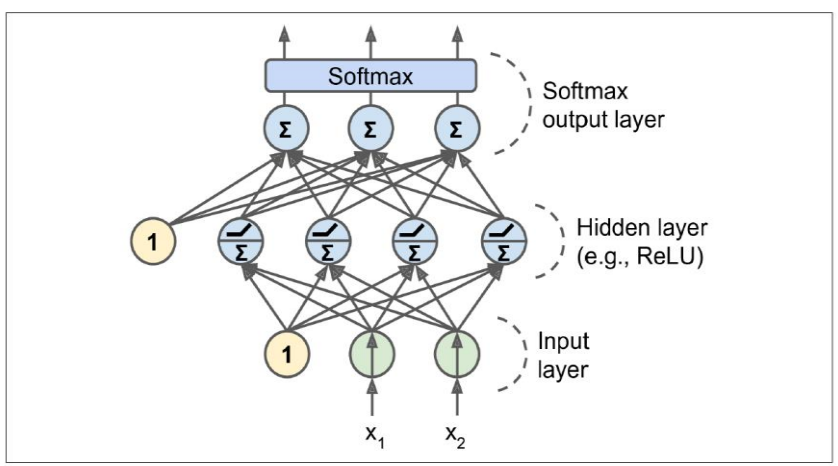
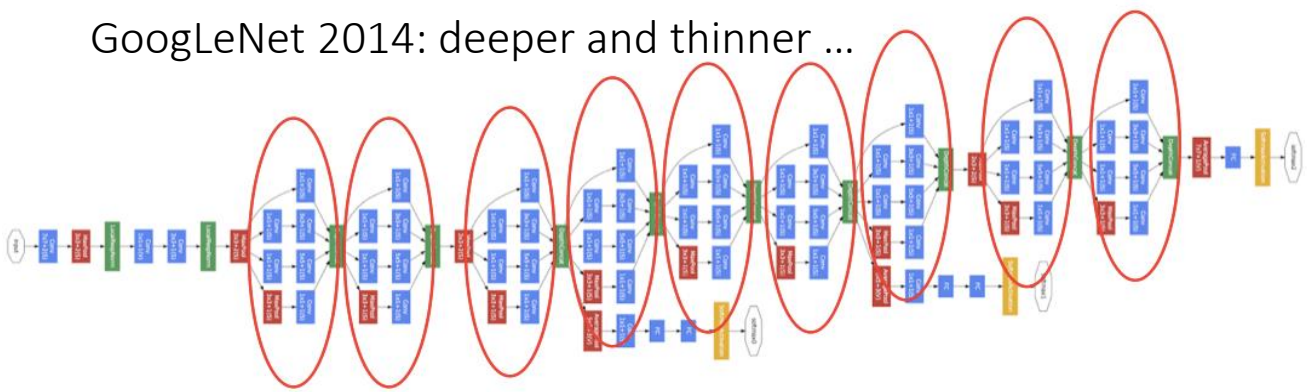


Figure 10-9. A modern MLP (including ReLU and softmax) for classification

GoogLeNet 2014: deeper and thinner …



9 **Inception** modules

Network in a network in a network...

Convolution
Pooling
Softmax
Other

many many more ….


Q&A
WHO?
WHERE?
WHEN?
WHAT?
How?
Why?
When?
Where?
What?
HOW?

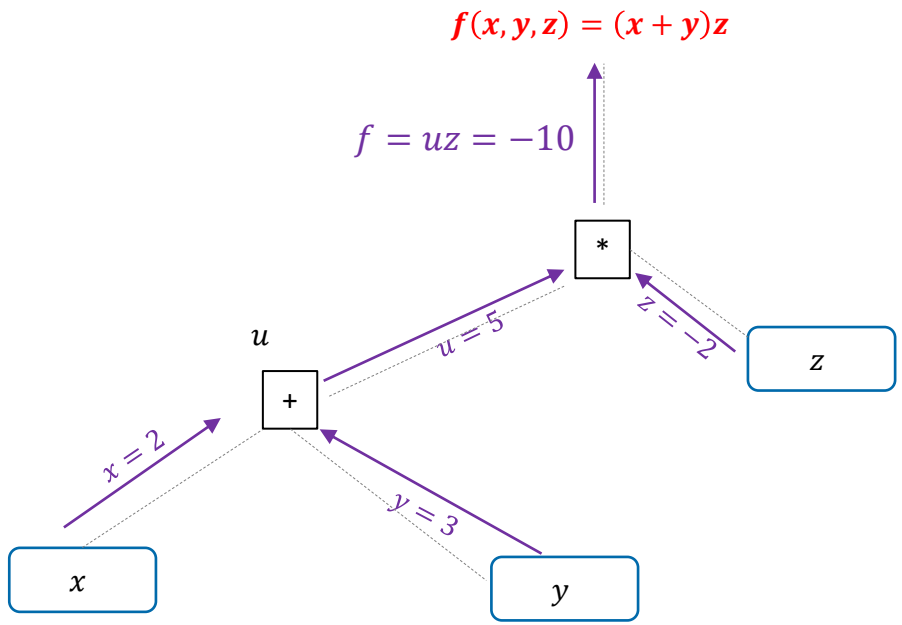
Computational graph

Computational graph (used in TensorFlow)

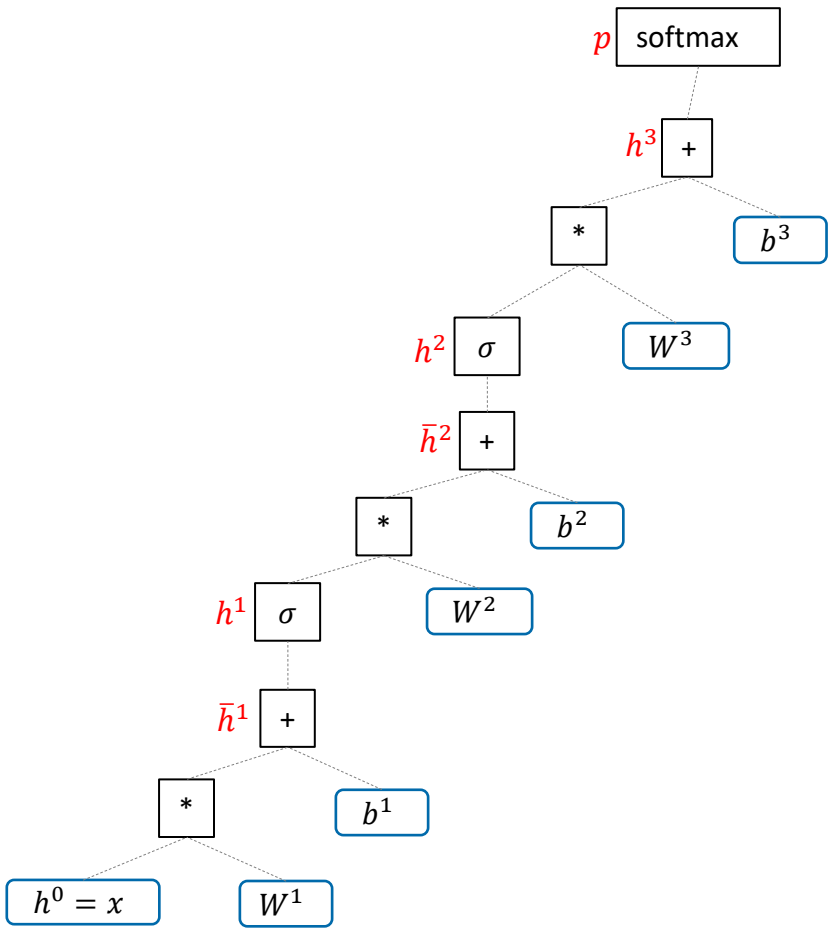
Problem: $f(x, y, z) = (x + y)z$
What is the output evaluated at $x = 2, y = 3, z = -2$?

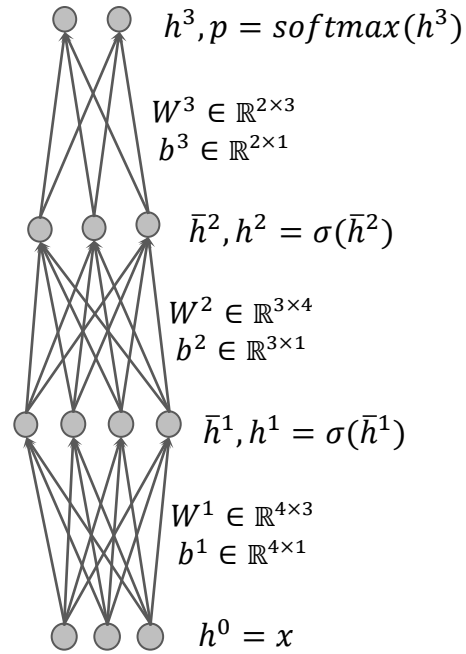
Step 1:

a) construct computational graph
b) forward propagation
c) record value at each node



Computational graph of feedforward nets





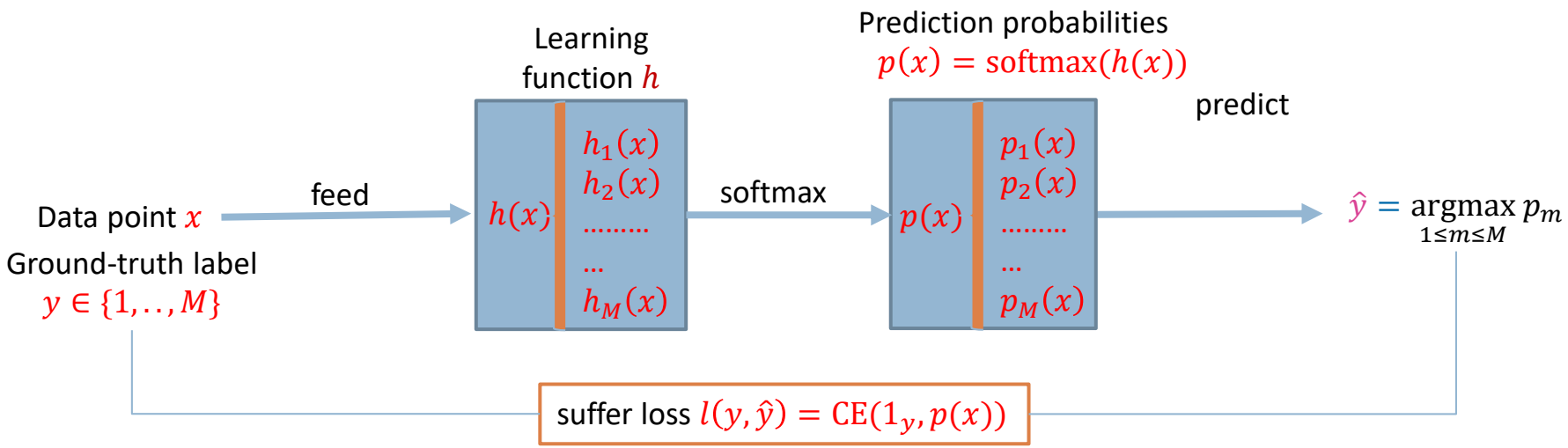
$h^0(x) = x$
for $k = 1$ **to** 2 **do**
  $\bar{h}^k = W^k h^{k-1}(x) + b^k$ //linear operation
  $h^k(x) = \sigma(\bar{h}^k(x))$ //activation
$h^3(x) = W^3 h^2(x) + b^2$
$p(x) = softmax(h^3(x))$ //prediction probabilities

Training deep neural networks

Training a deep NN

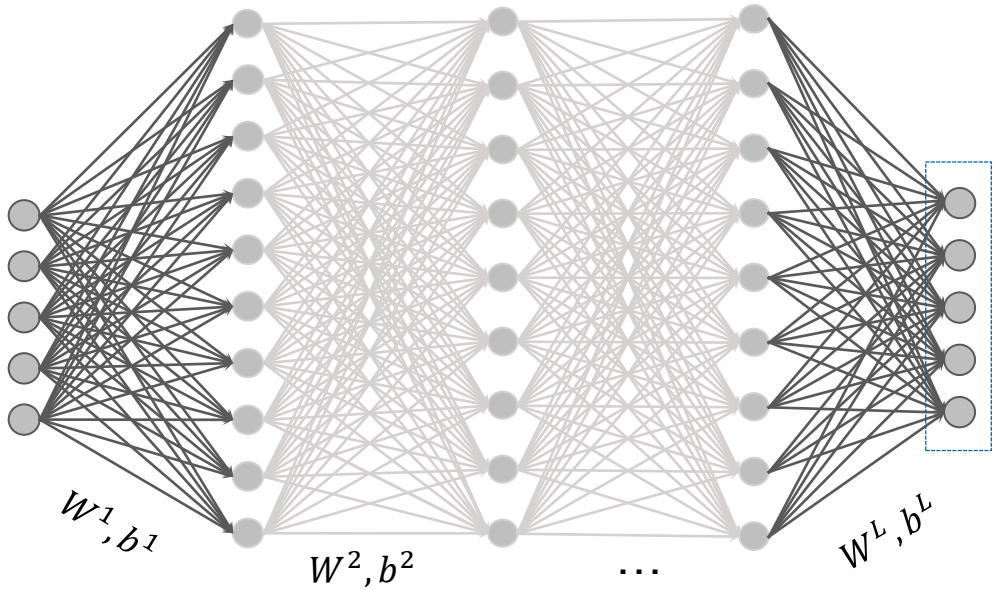
- We have learned how to make prediction if the model had already trained.
- Machine learning is all about how to train these models from data!
  - Training set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
- Back Propagation is currently the best approach to train a DNN.

Training a deep NN



- A data point x receives discriminative values $h_1(x), \dots, h_M(x)$ from the model
  - $h_m(x)$ represents the **possibility** to **classify** x to **class** $\boldsymbol{m}$ for $1 \le m \le M$
- We use these discriminative values to predict the label $\hat{y}$ as
  - $\hat{y} = \text{argmax}_{1 \le m \le M} h_m(x)$, meaning the class with highest discriminative value
- The prediction x with the predicted label $\hat{y}$ suffers a loss
  - $l(\hat{y}, y)$ where l is a loss function (if $\hat{y} = y$ then $l(\hat{y}, y) = 0$).
- Given a training set $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$, the loss incurred is
  - $\frac{1}{N} \sum_{i=1}^{N} \boldsymbol{l(y_i, \hat{y}_i)}$

Training deep nets



suffer loss $l(y, \hat{y}) = \mathrm{CE}(1_y, p(x))$

Training set

$$D = \{(x_1, y_1), \ldots (x_N, y_N)\}$$

Loss function

$$L(D;\theta) \coloneqq \frac{1}{N}\sum_{i=1}^{N} CE\left(1_{y_i}, p(x_i)\right) = -\frac{1}{N}\sum_{i=1}^{N} \log p_{y_i}(x_i)$$

(negative log likelihood)

- DNN parameters: $\theta \coloneqq \{(W^l, b^l)\}_{l=1}^{L}$
- Find model parameters (weight matrices and biases) so that the model predictions fit the training set as much as possible:

$$\min_{\theta} L(D;\theta)$$

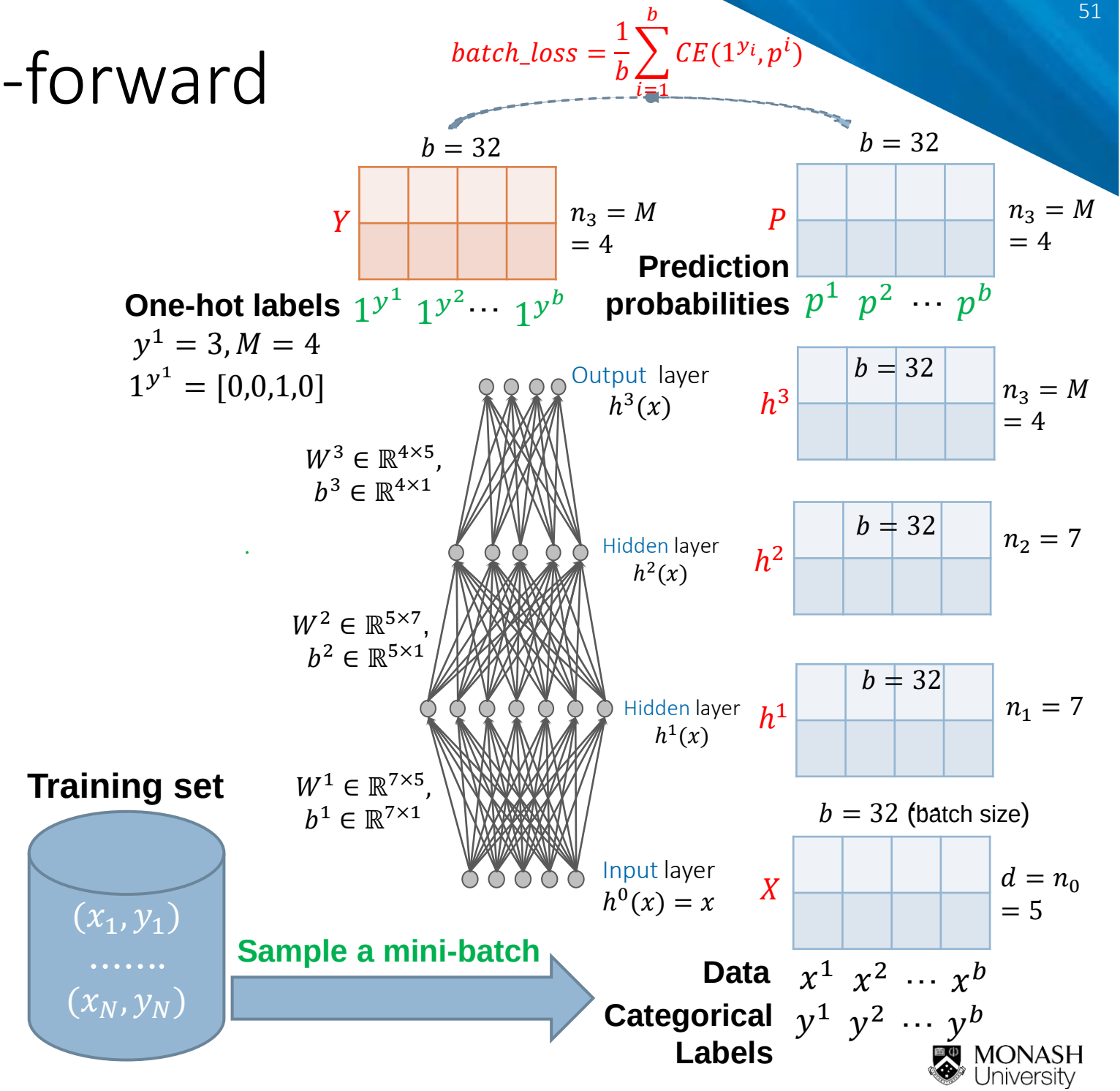
- Use optimizers SGD, Adagrad, Adam, RMSProp to update the model parameters (lectures 3 and 5)

TF Implementation

- tf.keras.optimizers.SGD
- tf.keras.optimizers.Adam
- tf.keras.optimizers.Adadelta
- tf.keras.optimizers.Adagrad
- tf.keras.optimizers.RMSProp

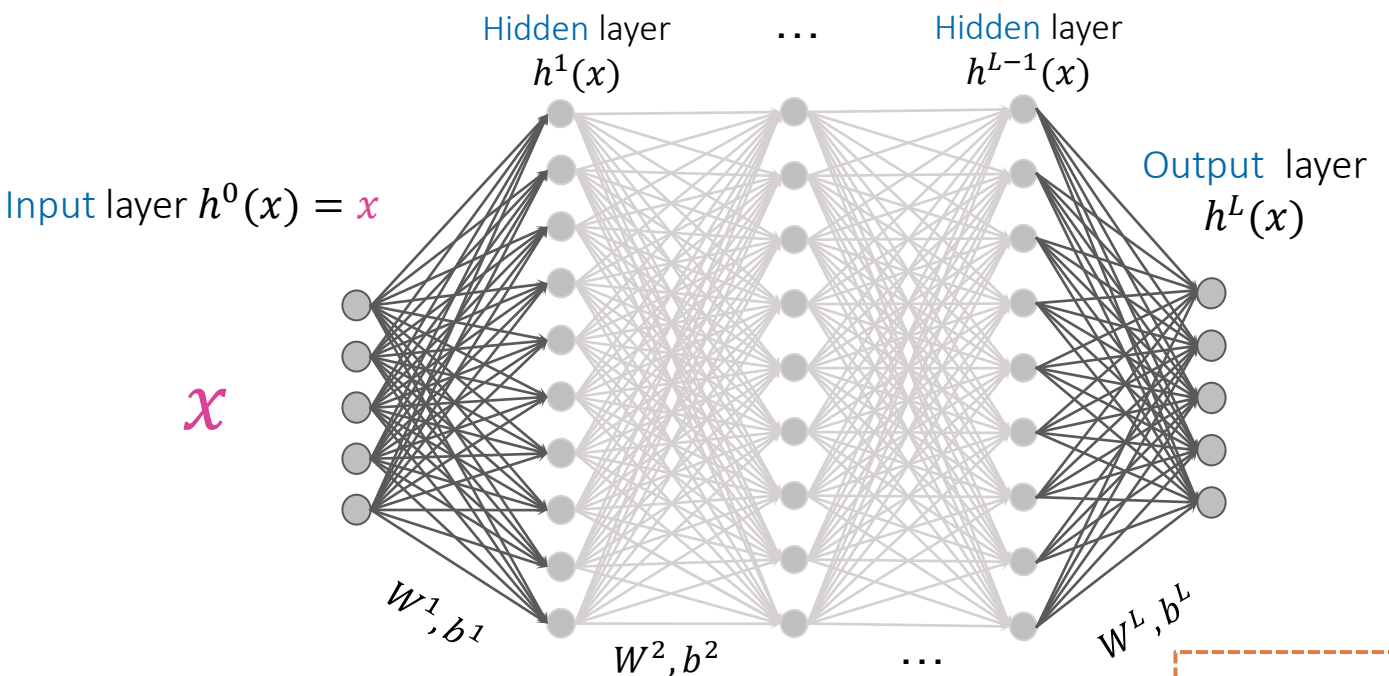
Mini-batch feed-forward

- Input
  - Tensor X: $[n_0 = d, b]$ (b is the batch size)
- Hidden layer 1
  - Tensor $[n_1, b]$
- ……
- Output layer
  - Tensor P: $[n_L = M, b]$
- The loss of the batch
  - $\frac{1}{b}\sum_{i=1}^{b} CE(1^{y_i}, p^i) = -\frac{1}{b}\sum_{i=1}^{b} \log p^i_{y_i}$
  - Update weight matrices and biases to **minimize** the batch loss.

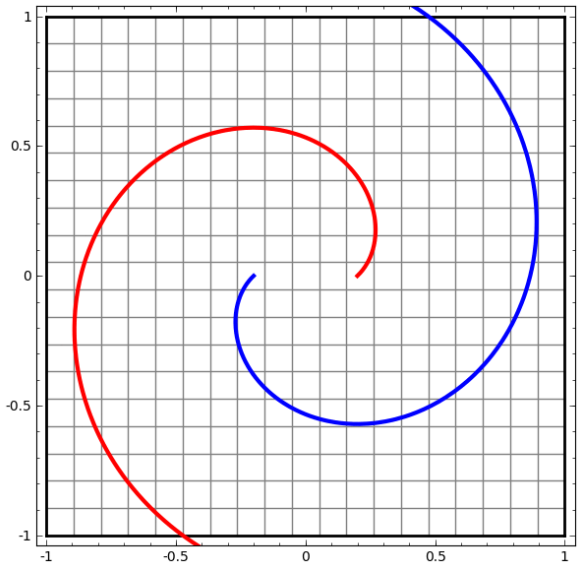


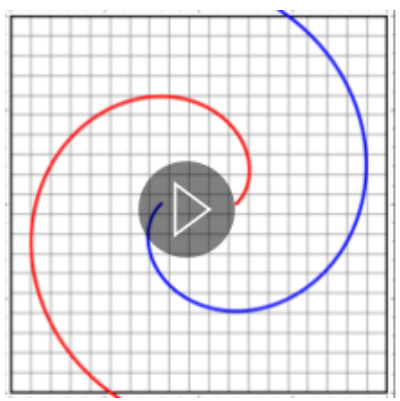
MONASH University

Representation learning of deep nets

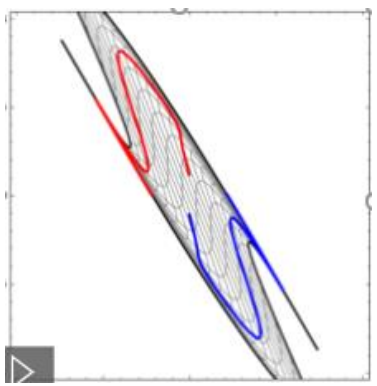


[Source: colab.com]





Data at layer 0



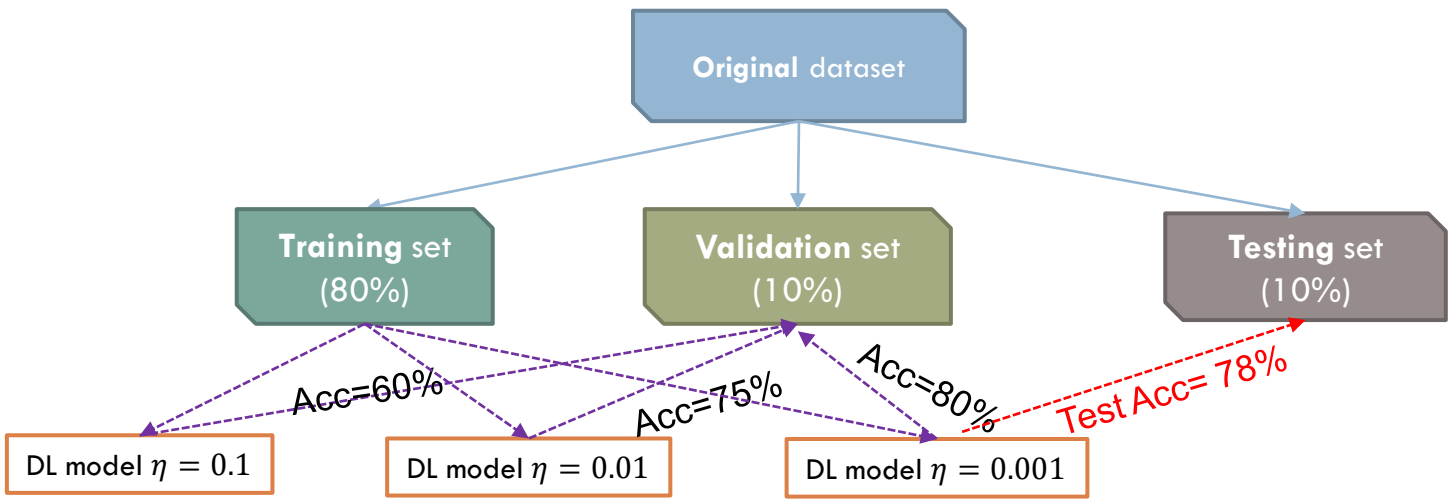
Data at layer $L-1$

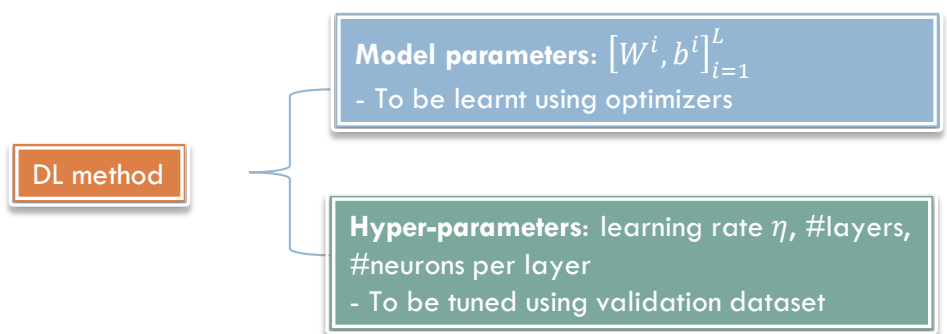
- We can think about feed-forward NN as a map $\boldsymbol{x} \mapsto \boldsymbol{\Phi}(\boldsymbol{x}) = \boldsymbol{h^{L-1}(x)}$
  - Feed-forward NN provides new representation for data
- We **supervisedly train** the network in such a way
  - In new representation layer, represented data can be **linearly separated**

Deep learning pipeline (Forum discussion)

Tuning hyper-parameters

- We want to train our DL model on a **training set** such that the **trained model** can predict well **unseen data** in a **separate testing set**.





Implementation of Feedforward NN (Forum discussion)

- Split dataset into **80%** for **training**, **10%** for **validation**, **10%** for **testing**

```
X_train, X_valid, X_test, y_train, y_valid, y_test = train_valid_test_split(X_data, y_data,
                                                                            train_size=0.8,
                                                                            test_size=0.1)

y_train= y_train.reshape(-1)
y_test= y_test.reshape(-1)
y_valid= y_valid.reshape(-1)
print(X_train.shape, X_valid.shape, X_test.shape)
print(y_train.shape, y_valid.shape, y_test.shape)
print("lables: {}".format(np.unique(y_train)))
```

```
(12000, 16) (1500, 16) (1500, 16)
(12000,) (1500,) (1500,)
lables: [ 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]
```

- Declare a feedforward NN **16 → 10 (ReLU) → 20 (ReLU) → 15 (ReLU) → 26 (softmax)**

```
dnn_model = Sequential()
dnn_model.add(Dense(units=10,  input_shape=(16,), activation='relu'))
dnn_model.add(Dense(units=20, activation='relu'))
dnn_model.add(Dense(units=15, activation='relu'))
dnn_model.add(Dense(units=n_classes, activation='softmax'))
```

- Declare the **loss**, **metric**, and **optimization** and train a deep learning model

```
dnn_model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```

```
history = dnn_model.fit(x=X_train, y=y_train, batch_size=32,
                        epochs=20,
                        validation_data=(X_valid, y_valid))
```

```
dnn_model.evaluate(X_test, y_test)  #return loss and accuracy
```

```
47/47 [==============================] - 0s 1ms/step - loss: 0.9410 - accuracy: 0.7253
[0.9410395622253418, 0.7253333330154419]
```

```
Epoch 1/20
375/375 [==============================] - 1s 2ms/step - loss: 2.9334 - accuracy: 0.1297 - val_loss: 2.3989 - val_accuracy: 0.2607
Epoch 2/20
375/375 [==============================] - 0s 1ms/step - loss: 1.9917 - accuracy: 0.3918 - val_loss: 1.7672 - val_accuracy: 0.4807
Epoch 3/20
375/375 [==============================] - 0s 1ms/step - loss: 1.6242 - accuracy: 0.5191 - val_loss: 1.5474 - val_accuracy: 0.5413
Epoch 4/20
375/375 [==============================] - 0s 1ms/step - loss: 1.4590 - accuracy: 0.5692 - val_loss: 1.4292 - val_accuracy: 0.5827
```

Summary

- Prelude to deep learning
  - Feed-forward NN (vector data), Convolutional NN (image, 2D-3D grid data), Recurrent NN (sequential data).
- Perceptron
  - Stack perceptron and activation function → deep neural nets.
- Feed-forward neural networks
  - Parameterization
  - Forward propagation and compute the cross-entropy loss
  - Computational graph of deep nets
  - Train deep nets
  - Representation learning of deep nets

Thanks for your attention!