# INT3405 - Machine Learning

## Lecture 9: Deep Learning

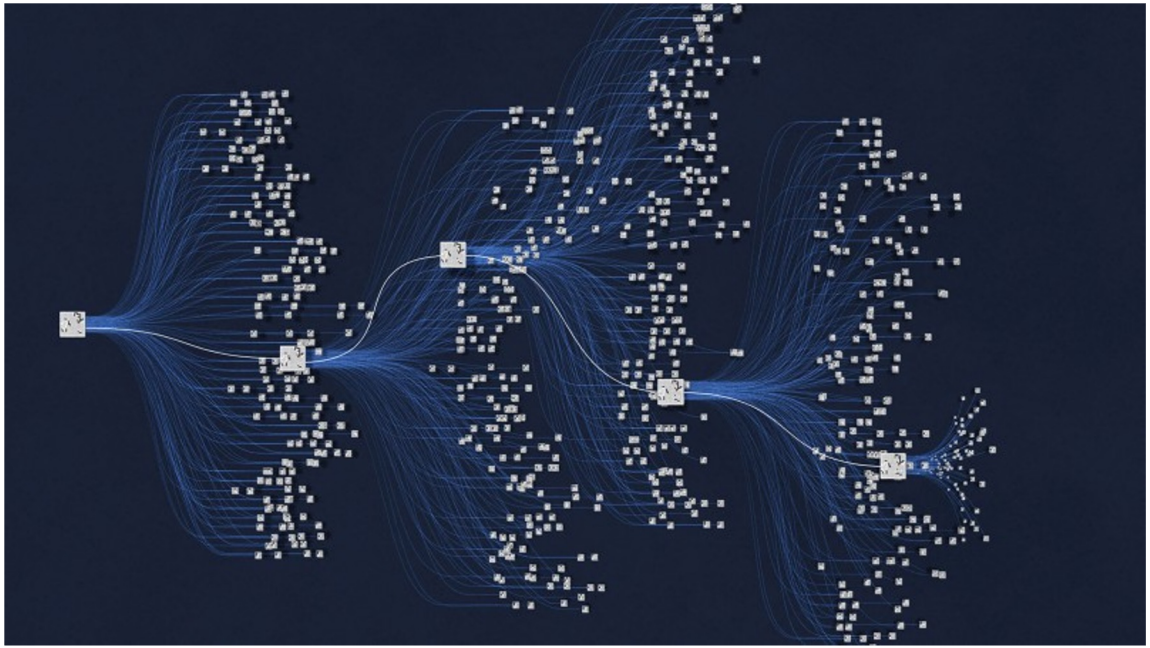
**Duc-Trong Le & Viet-Cuong Ta**

**Hanoi, 09/2023**

# Outline

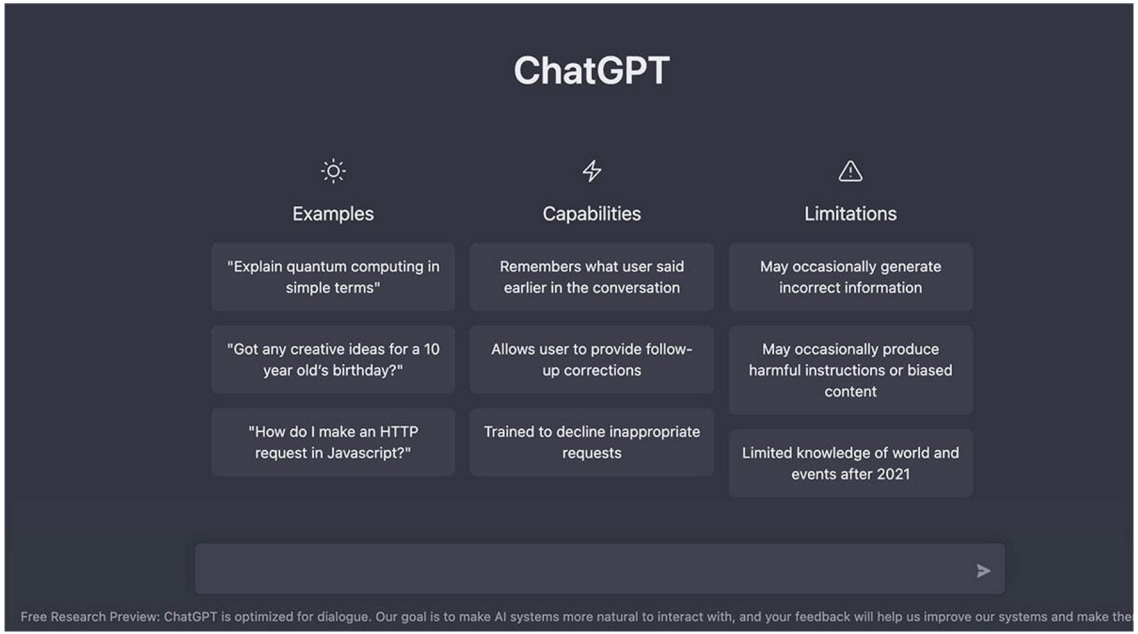
- Deep Learning
- Learning with Gradient Descent
- Back-propagation

# AlphaGo



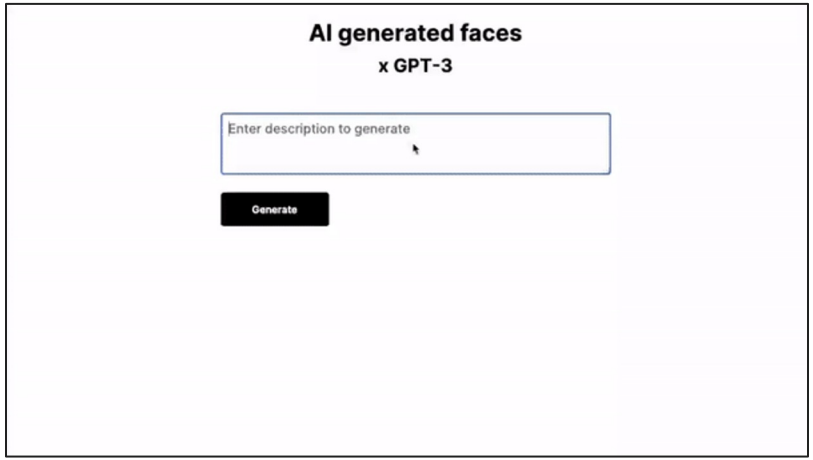
https://www.youtube.com/watch?v=8tq1C8spV_g

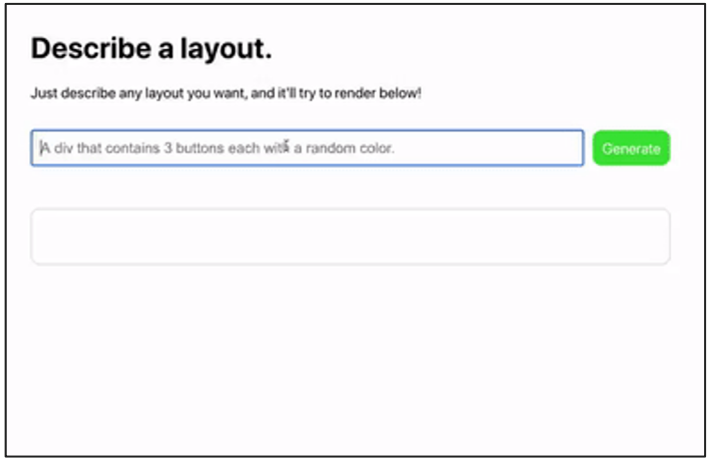
# ChatGPT




https://chat.openai.com/

# GPT Applications




https://www.youtube.com/watch?v=_x9AwxfjxvE

# Tesla X


https://twitter.com/elonmusk/status/1695247110030119054

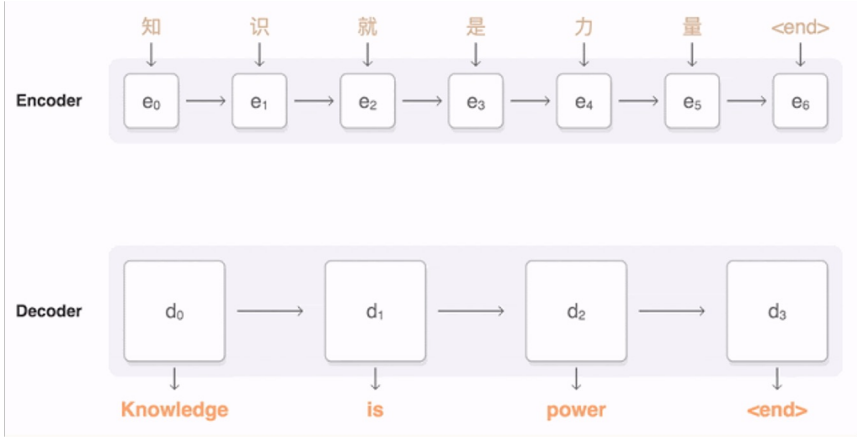
# Emotion Detection

- Anger
- Disgust
- Fear
- Happiness
- Neutral
- Sadness
- Surprise


**Check out**
**https://faceinmotion.preferred.ai**

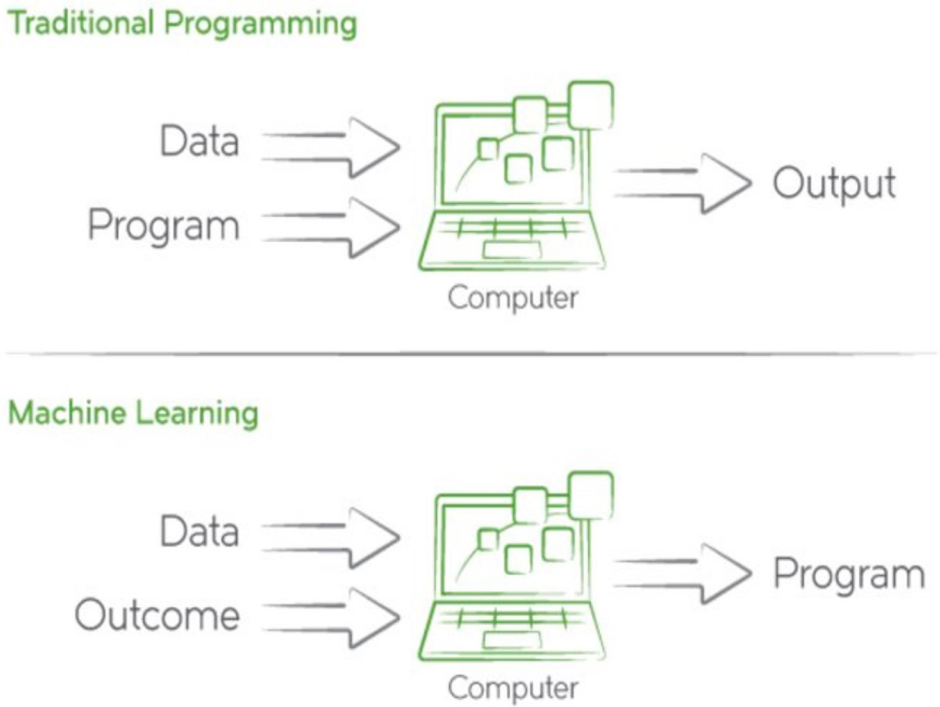
https://www.freecodecamp.org/news/facial-emotion-recognition-develop-a-c-n-n-and-break-into-kaggle-top-10-f618c024faa7/

# Google Translation




https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html

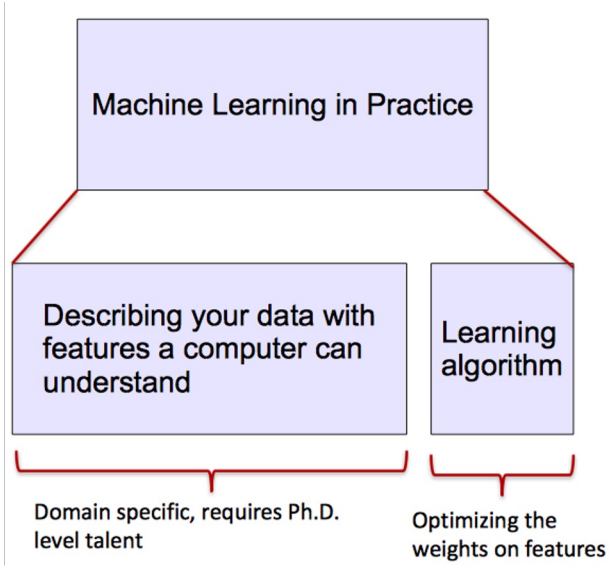
# Machine Learning Basis



# Traditional Machine Learning

Traditional ML methods work well because of **human-designed representations** and **input features**

ML becomes just **optimizing weights** to best make a final prediction

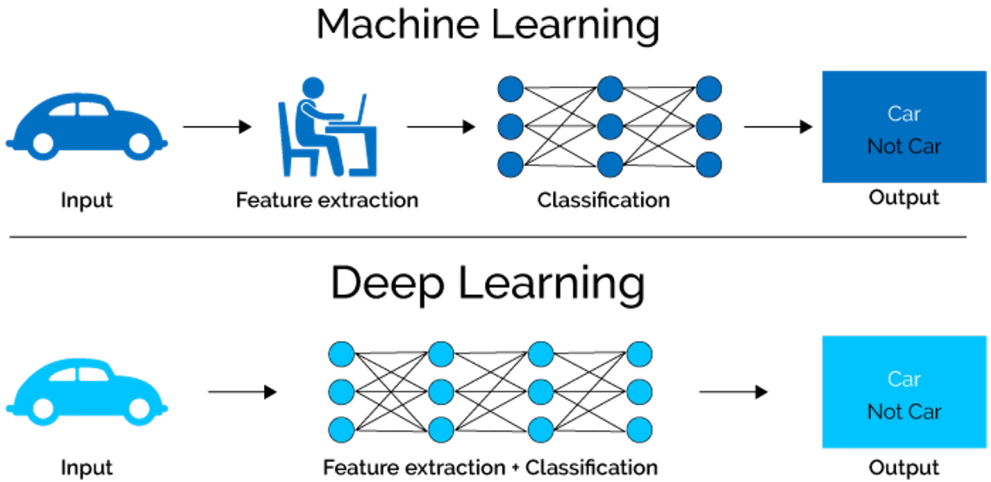



| Feature | NER |
|---|---|
| Current Word | ✓ |
| Previous Word | ✓ |
| Next Word | ✓ |
| Current Word Character n-gram | all |
| Current POS Tag | ✓ |
| Surrounding POS Tag Sequence | ✓ |
| Current Word Shape | ✓ |
| Surrounding Word Shape Sequence | ✓ |
| Presence of Word in Left Window | size 4 |
| Presence of Word in Right Window | size 4 |

# What is Deep Learning?

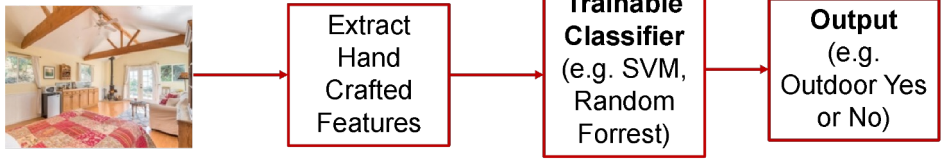
A machine learning subfield of learning **representations** of data. Exceptional effective at **learning patterns**.

Deep learning algorithms attempt to learn (multiple levels of) representation by using a **hierarchy of multiple layers**

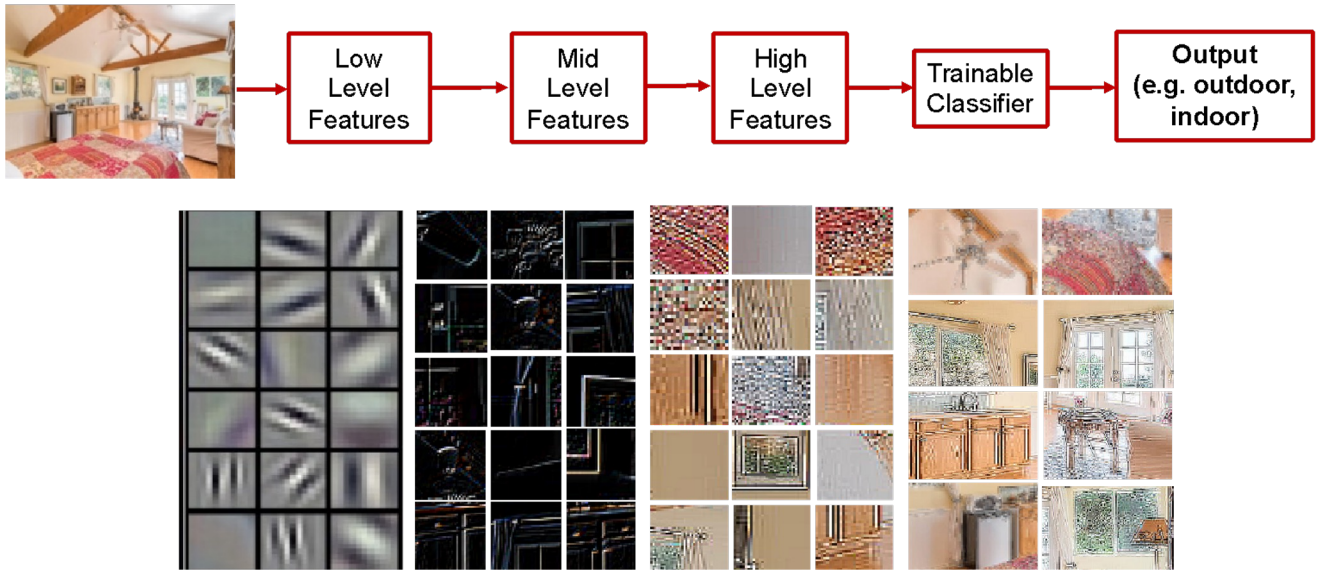



*https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png*

## Traditional ML:

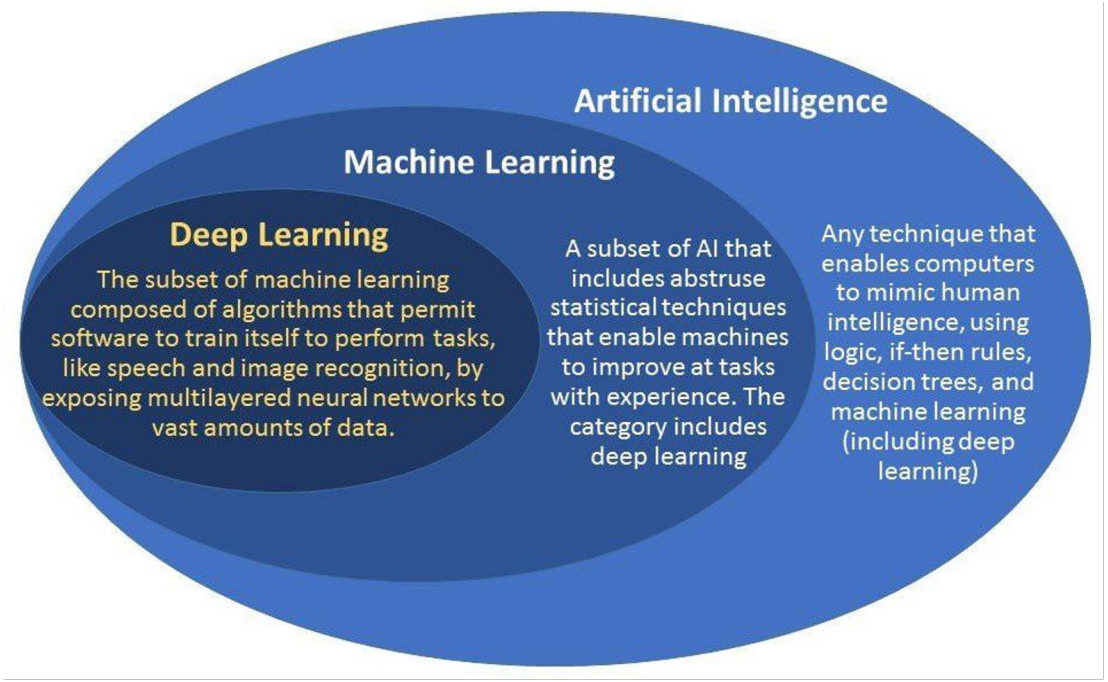



## Deep Learning:



*“Deep Learning doesn’t do different things,* ***it does things differently”****.*

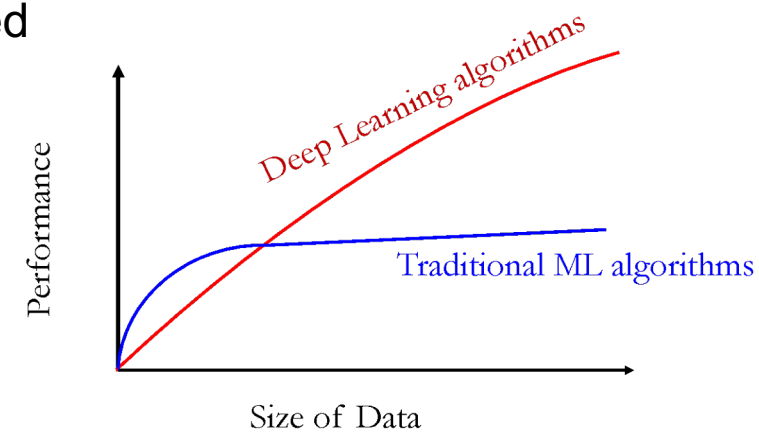
# Machine Learning vs. Deep Learning



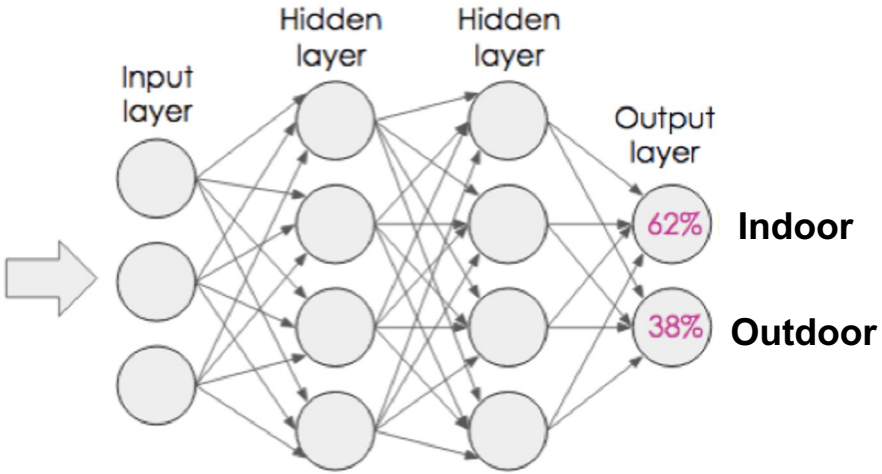

*https://lerablog.org/technology/ai-artificial-intelligence-vs-machine-learning-vs-deep-learning/*

# Why is DL useful?

- Manually designed features are often **over-specified**, **incomplete** and take a **long time to design** and validate
- Learned features are **easy to adapt**, **fast** to learn
- Deep learning provides a very **flexible**, (almost?) **universal**, learnable framework for representing world, visual and linguistic information.
- Can learn both unsupervised and supervised
- Effective **end-to-end** joint system learning
- Utilize large amounts of training data

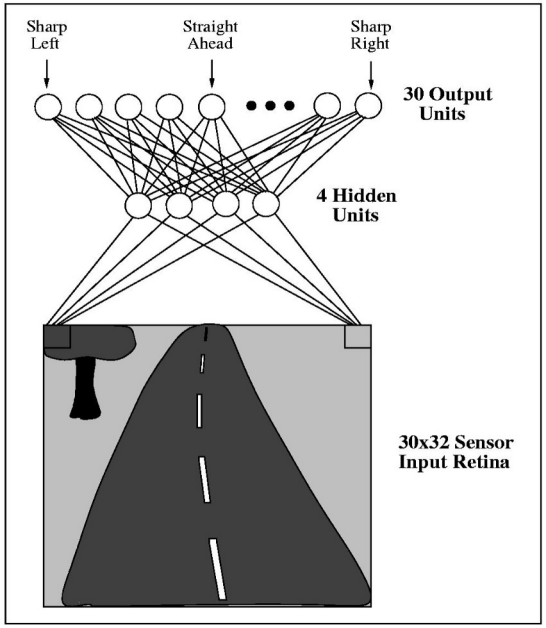


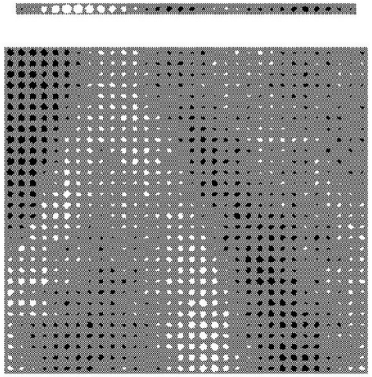
# How does DL learn features?




**Answer:** Indoor

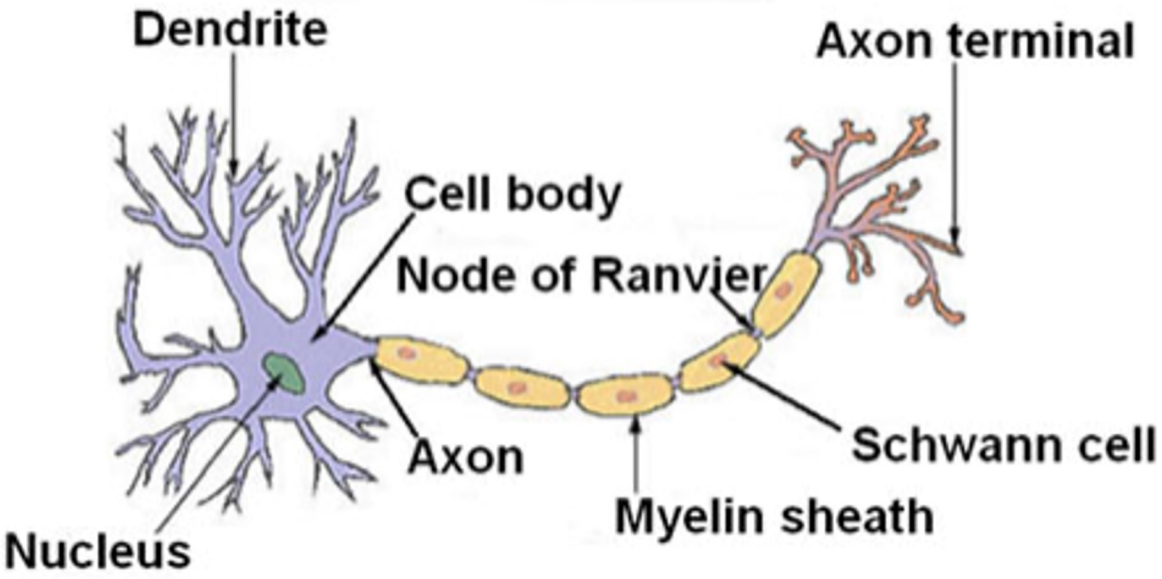
# Self—Driving Car



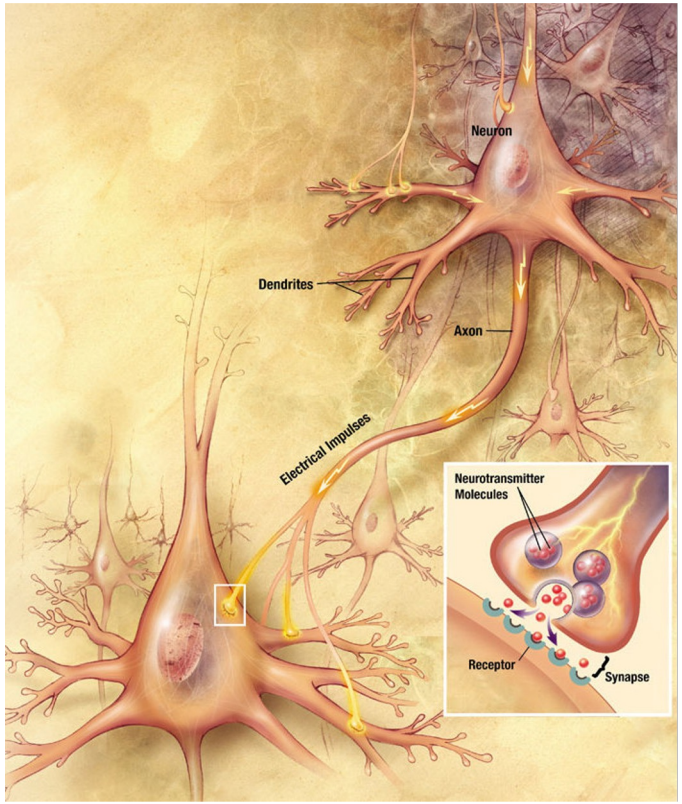



# 2.
# Neural Networks

# Neural in the Brain (1)

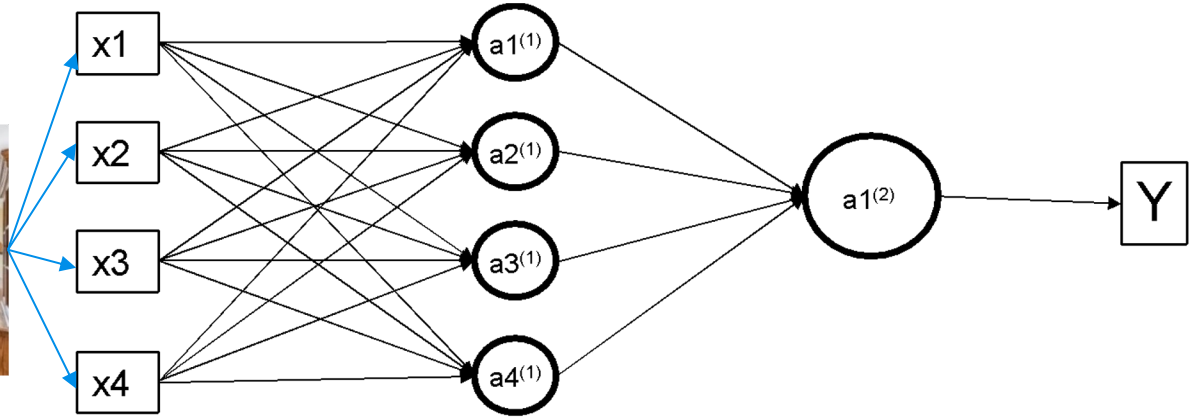


# Neural in the Brain (2)



# Artificial Neural Network

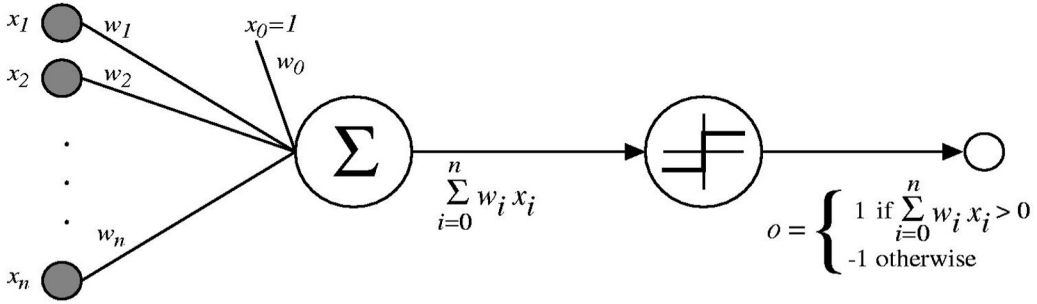
An Artificial Neural Network is an information processing paradigm that is **inspired by the biological nervous systems**, such as the human brain's information processing mechanism.




Input | Hidden Layers | Output

#parameters: 4*4 + 4 +1

# Perceptron




$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

# Perceptron Training Rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$ is target value
- $o$ is perceptron output
- $\eta$ is small constant (e.g., 0.1) called *learning rate*

Can prove it will converge if

- Training data is linearly separable
- $\eta$ sufficiently small

# Gradient Descent (1)

To understand, consider simpler *linear unit*, where

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Let's learn $w_i$'s that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where $D$ is set of training examples

# Gradient Descent (2)

Gradient:

$$\nabla E[\vec{w}] \equiv \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n} \right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

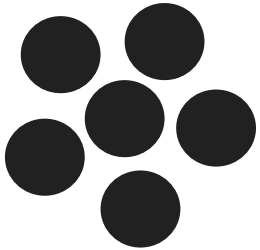
I.e.:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

# Gradient Descent (3)

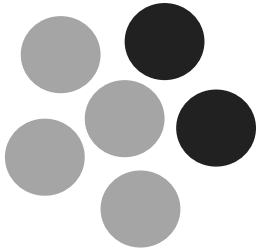
$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}
$$

# Batch, Mini-Batch, Iterative Training



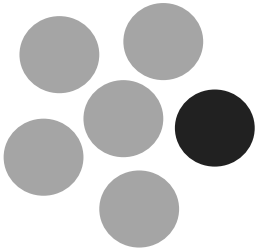
**Batch Training**

Use all training points to compute gradients for each iteration



**Mini-batch Training**

Use subset training points to compute gradients for each iteration



**Iterative Training**

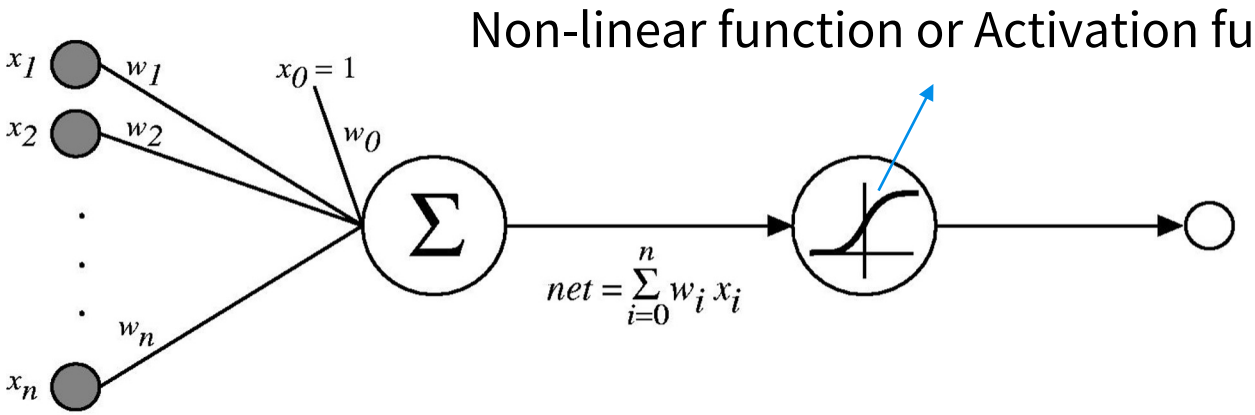
Use a single training point to compute gradients for each iteration

# Iteration, Epoch

An iteration respects to the training for a mini-batch

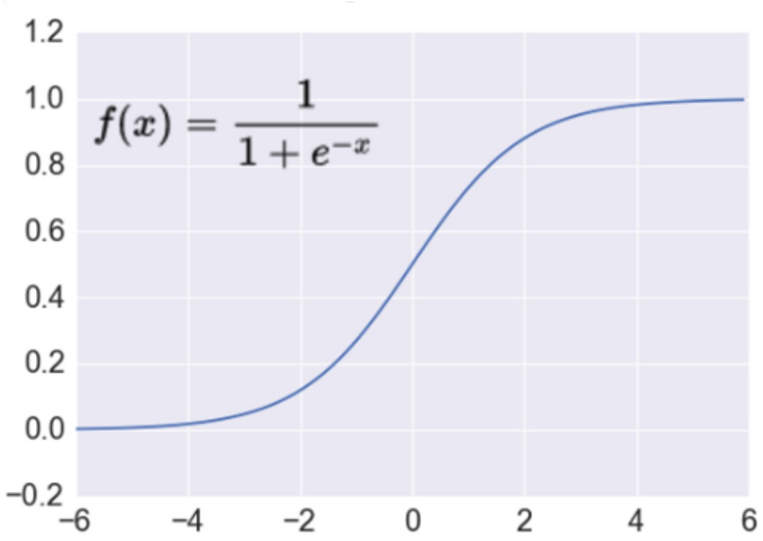
An epoch respects to the training for full dataset

# Neural Networks with Activation Functions




The purpose of the activation function is to introduce ***non-linearity into the network***

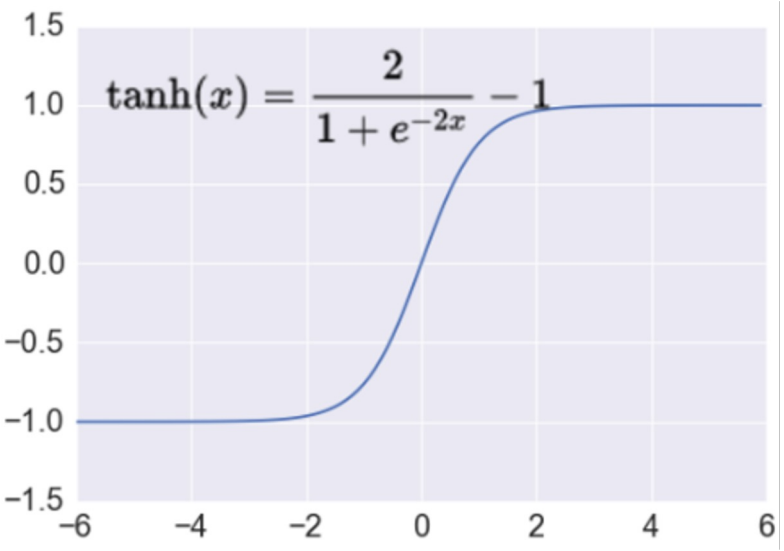
# Activation: Sigmoid




*http://adilmoujahid.com/images/activation.png*

Takes a real-valued number and “squashes” it into range between 0 and 1.

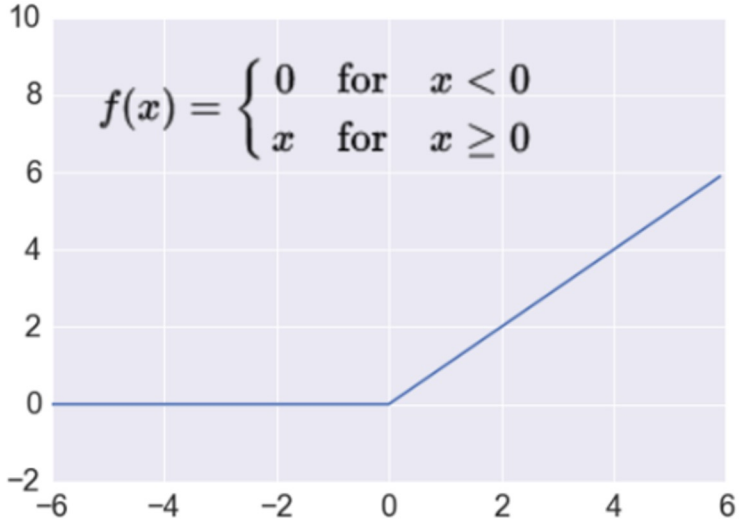
# Activation: Tanh




*http://adilmoujahid.com/images/activation.png*

Takes a real-valued number and “squashes” it into range between -1 and 1.

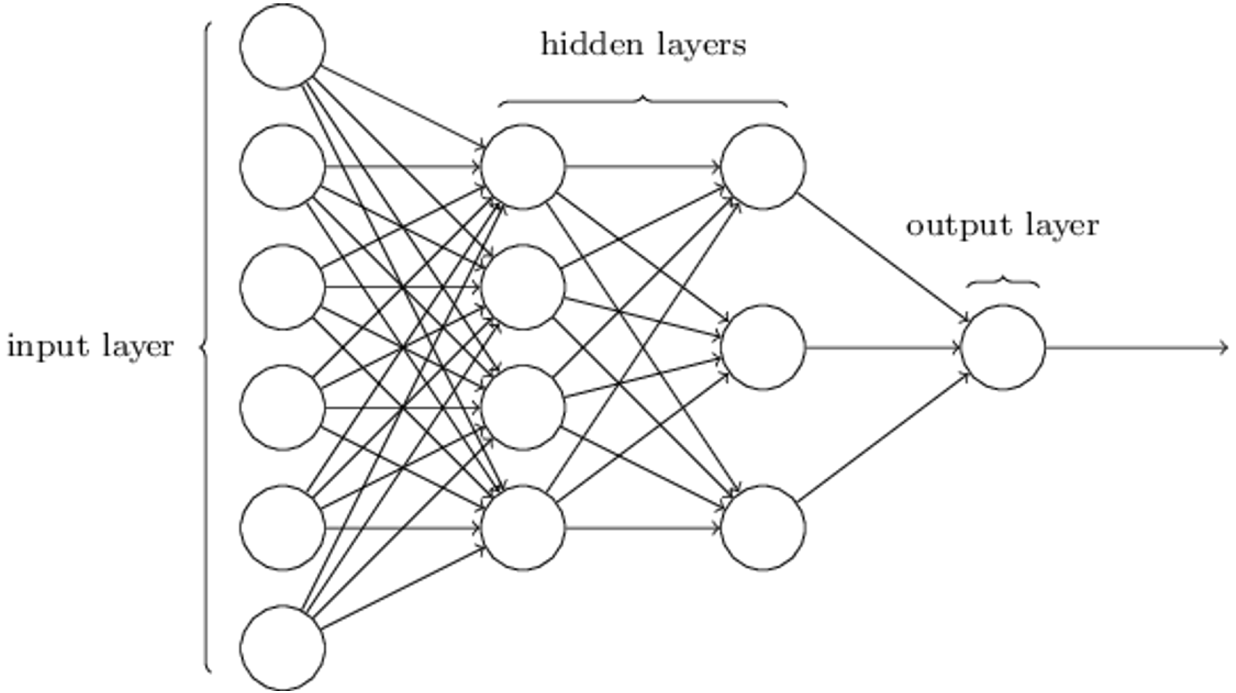
# Activation: ReLu



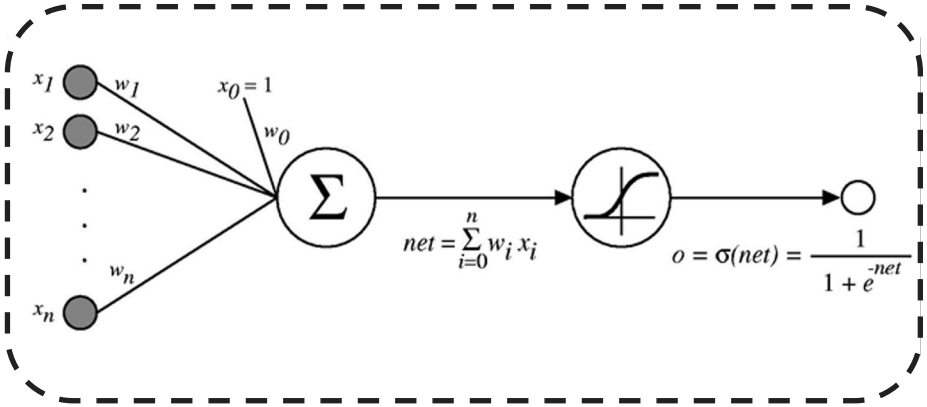

*http://adilmoujahid.com/images/activation.png*

Takes a real-valued number and thresholds it at zero

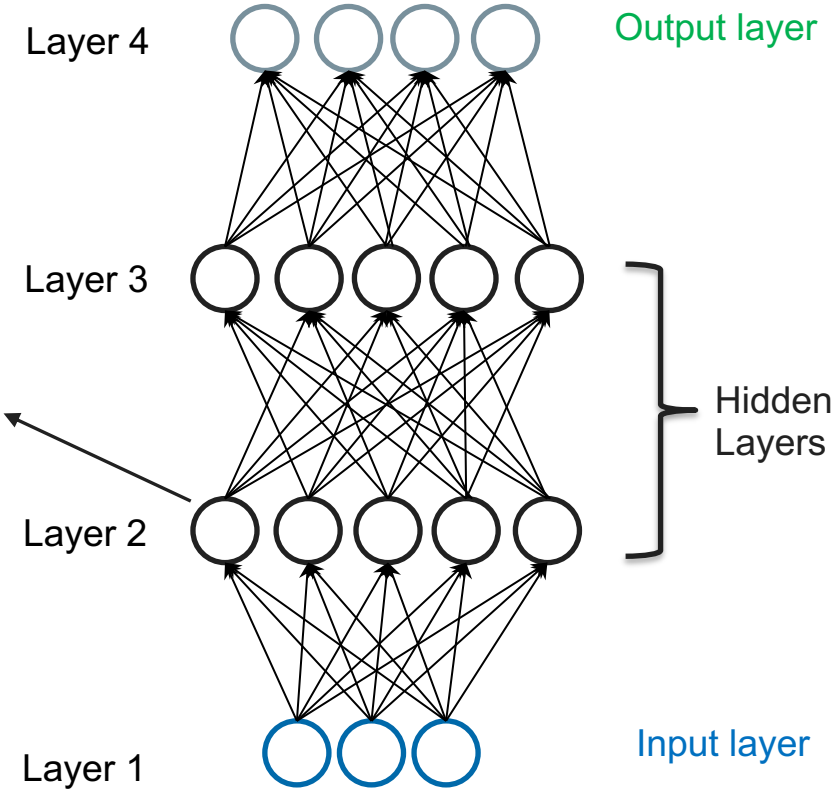
# Multi Layer Perceptron



# Multi-layer Neural Networks with Sigmoid




Each node (“neuron”) is the sigmoid unit



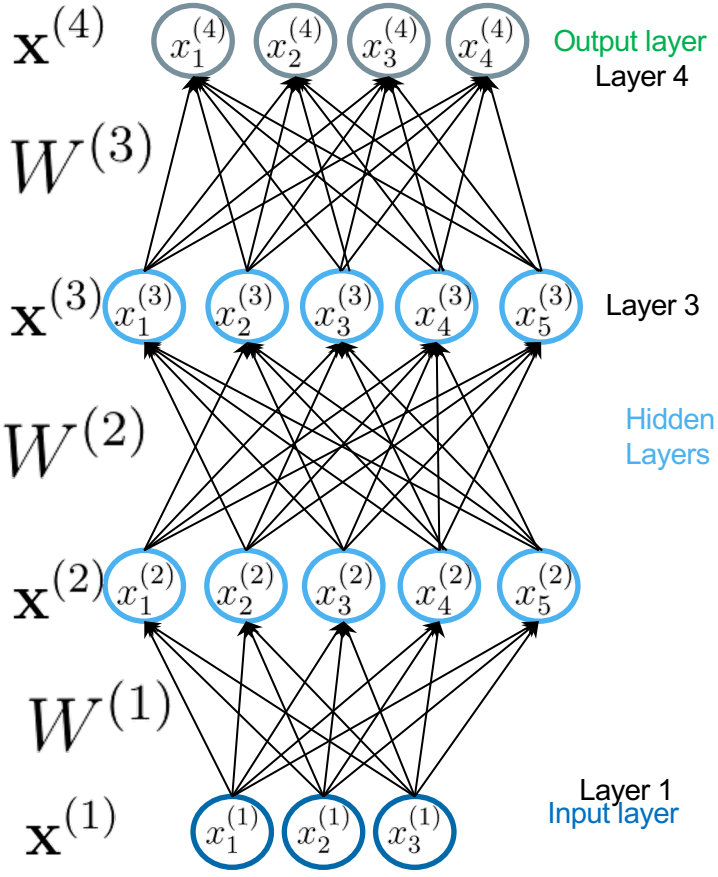
# Forward Propagation

- Notations
  - Input vector at level $l$ $\mathbf{x}^{(l)}$
  - Weight matrix of level $l$ $W^{(l)}$
- Forward-propagation

$$\mathbf{x}^{(2)} = \sigma(W^{(1)}\mathbf{x}^{(1)})$$
$$5 \times 1 \quad 5 \times 3 \quad 3 \times 1$$

$$\mathbf{x}^{(3)} = \sigma(W^{(2)}\mathbf{x}^{(2)})$$
$$5 \times 1 \quad 5 \times 5 \quad 5 \times 1$$

$$\mathbf{x}^{(4)} = \sigma(W^{(3)}\mathbf{x}^{(3)})$$
$$4 \times 1 \quad 4 \times 5 \quad 5 \times 1$$



# Back Propagation

- Error at level $l$: $\boldsymbol{\delta}^{(l)}$
- Error at last level (L=4)

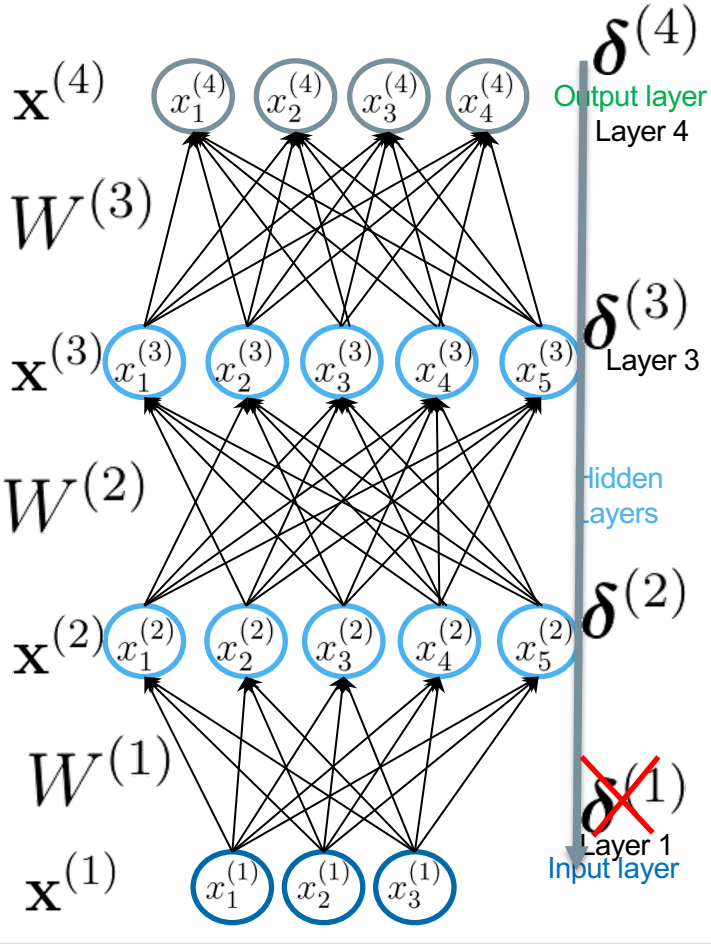
$$\boldsymbol{\delta}^{(4)} = (\mathbf{y} - \mathbf{x}^{(4)}) .* \mathbf{x}^{(4)} .* (1 - \mathbf{x}^{(4)})$$

$$\boldsymbol{\delta}^{(3)} = (W^{(3)})^T \boldsymbol{\delta}^{(4)} .* \mathbf{x}^{(3)} .* (1 - \mathbf{x}^{(3)})$$

$$\boldsymbol{\delta}^{(2)} = (W^{(2)})^T \boldsymbol{\delta}^{(3)} .* \mathbf{x}^{(2)} .* (1 - \mathbf{x}^{(2)})$$

$$\Delta W^{(l)} = \eta \boldsymbol{\delta}^{(l+1)} (\mathbf{x}^{(l)})^T$$

$$\mathbf{W}^{(l)} := \mathbf{W}^{(l)} + \Delta W^{(l)}$$



# More example

Find parameters $\mathbf{W} = (\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_L)$ such that the error function $\ell$ is minimum.

$$\ell = -\mathbf{y}^T \log(\mathcal{S}(\phi_L(\mathbf{W}_L\phi_{L-1}(\mathbf{W}_{L-1}\ldots\phi_1(\mathbf{W}_1\mathbf{x})))))$$

*Solution*: Update parameters using gradients:

$$\delta_{\mathbf{W}_l} = \frac{\partial \ell}{\partial \mathbf{W}_l}, \forall l = 1, 2, \ldots L$$

# More example

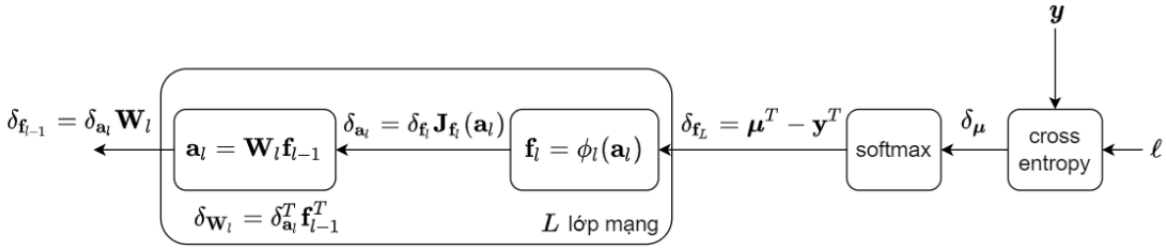
- Univariate function $(f \circ g)(x) = f(g(x))$ have derivative $(f \circ g)'(x) = f'(g(x))g'(x)$
- Multivariate multi-value function $\mathbf{f}(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^n$ have Jacobian derivative matrix:

$$\mathbf{J}_\mathbf{f}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_d} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_d} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_d} \end{bmatrix} \in \mathbb{R}^{n \times d}$$

- The nested function: $(\mathbf{f} \circ \mathbf{g})(\mathbf{x}) = \mathbf{f}(\mathbf{g}(\mathbf{x}))$ with $\mathbf{x} \in \mathbb{R}^d, \mathbf{y} = \mathbf{g}(\mathbf{x}) \in \mathbb{R}^n, \mathbf{f}(\mathbf{y}) \in \mathbb{R}^m$

$$\mathbf{J}'_{\mathbf{f} \circ \mathbf{g}}(\mathbf{x}) = \mathbf{J}'_\mathbf{f}(\mathbf{g}(\mathbf{x}))\mathbf{J}'_\mathbf{g}(\mathbf{x})$$

# More example

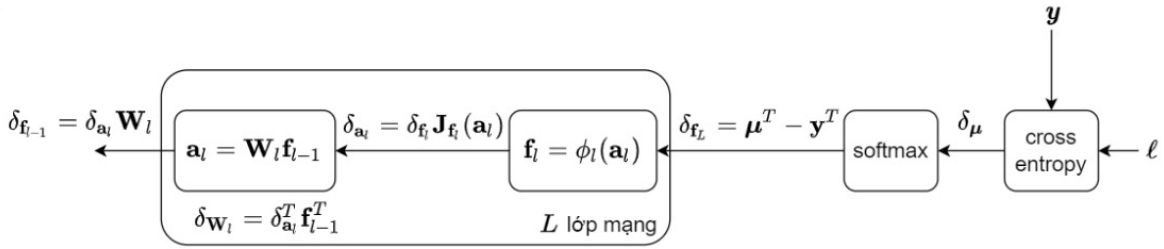



Quá trình lan truyền ngược đạo hàm

1. (Forward propagation) Loop $L$ time, convention $f_0 = x$ with $l = 1, 2, \ldots L$

$$\mathbf{a}_l = \mathbf{W}_l\mathbf{f}_{l-1} \in \mathbb{R}^{p_l}$$
$$\mathbf{f}_l = \phi_l(\mathbf{a}_l) \in \mathbb{R}^{p_l}$$

# More example

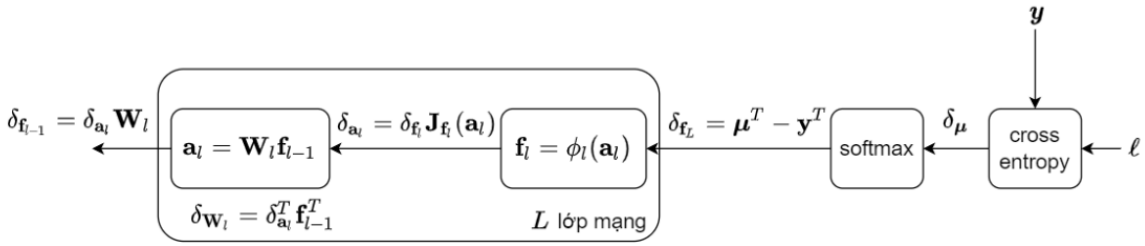



Quá trình lan truyền ngược đạo hàm

2. (Output) Final Logits, probability (softmax) and loss

$$\mathbf{f} = \mathbf{f}_L \in \mathbb{R}^C$$
$$\boldsymbol{\mu} = \mathcal{S}(\mathbf{f}) \in \mathbb{R}^C$$
$$\ell = -\mathbf{y}^T \log(\boldsymbol{\mu}) \in \mathbb{R}$$

# More example



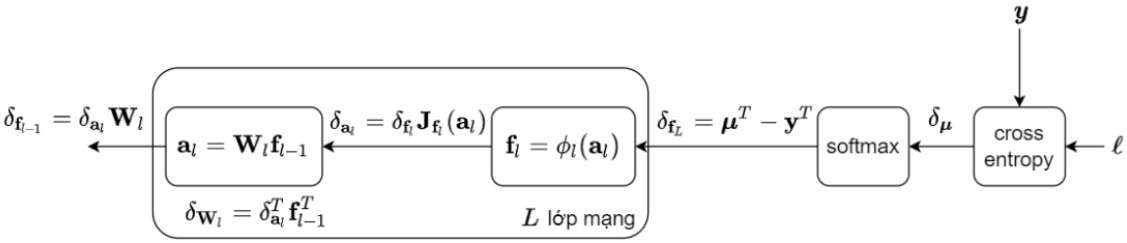

Quá trình lan truyền ngược đạo hàm

3. (Output derivative) Use derivative of loss function (the last layer)

$$\delta_{\mathbf{f}_L=}\delta_{\mathbf{f}} = \delta_{\boldsymbol{\mu}}\mathbf{J}_{\boldsymbol{\mu}}(\mathbf{f}) \in \mathbb{R}^{1\times C}$$

In case of soft-max and cross-entropy $\delta_{\mathbf{f}} = \boldsymbol{\mu}^T - \mathbf{y}^T$

# More example




Quá trình lan truyền ngược đạo hàm

4. (Weight derivative - back propagation) Loop $L$ times with $l = L, L-1, \ldots, 1$

$$\delta_{\mathbf{a}_l} = \delta_{\mathbf{f}_l}\mathbf{J}_{\mathbf{f}_l}(\mathbf{a}_l) \in \mathbb{R}^{1\times p_l}$$
$$\delta_{\mathbf{W}_l} = \delta_{\mathbf{a}_l}^T\mathbf{f}_{l-1}^T \in \mathbb{R}^{p_l\times p_{l-1}}$$
$$\delta_{\mathbf{f}_{l-1}} = \delta_{\mathbf{a}_l}\mathbf{W}_l \in \mathbb{R}^{1\times p_{l-1}}$$

The results: $\delta_{\mathbf{a}_l}, \delta_{\mathbf{W}_l}, \delta_{\mathbf{f}_l}$ with $l = 1, 2, \ldots L$, specially $\delta_{\mathbf{x}} = \delta_{\mathbf{f}_0}$

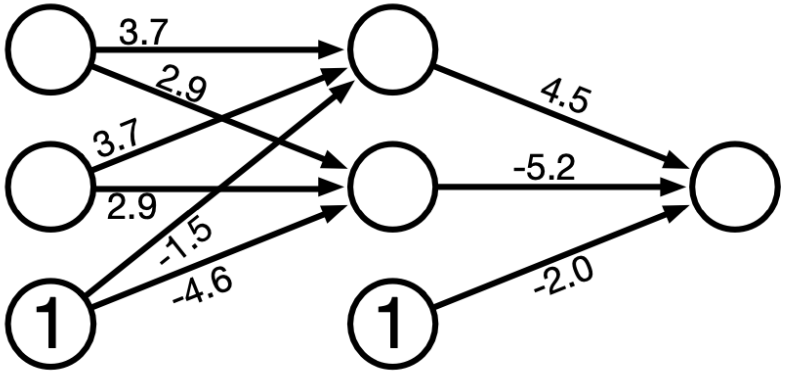
# More example

- $\delta_{\mathbf{W}_l}$: use for training
- $\delta_{\mathbf{x}}$: use for generate data with constraint
- $\delta_{\mathbf{f}_l}, \delta_{\mathbf{a}_l}$ for debugging, interpreting results, and visualizing the output of the network and its layers

**Read more**

- Yes you should understand backprop - Andrej Karpathy
- Cs231 understanding backprop
- Vector, Matrix, and Tensor Derivatives

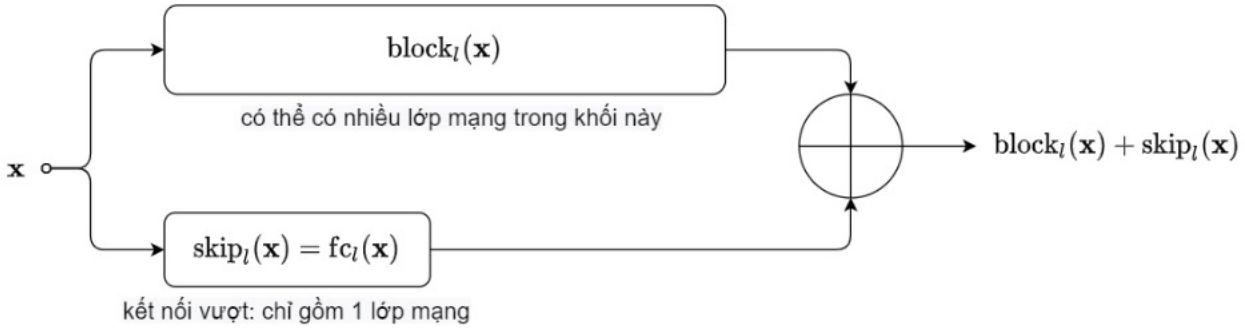
# More example




1. Compute the output with sigmoid as the activation function, directly and matrix-based

2. Do on step update with x = (1, 0) y = 1, with binary cross entropy loss, directly and matrix-based

# More example




Một khối mạng phần dư (residual block)

Compute the derivative with skip connection

# Summary

- Gradient Descent
- Back Propagation