



元智大學

卓越・務實・宏觀・圓融



Deep Learning – Concepts and PyTorch Development

Tien-Lung Sun, Dept. of Industrial Engineering and Management, Yuan-Ze University

孫天龍 元智大學 工業工程與管理系 教授

My GitHub

The screenshot shows a web browser window displaying the GitHub profile of TienLungSun. The browser's address bar shows the URL `github.com/TienLungSun`. The page header includes the GitHub logo, a search bar, and navigation links for Pull requests, Issues, Marketplace, and Explore. The profile section on the left features a circular profile picture of a man with glasses, the name **Tien-Lung Sun**, the username `TienLungSun`, an `Edit profile` button, and statistics showing 3 followers, 0 following, and 0 stars. The main content area is titled **Popular repositories** and lists six repositories in a grid. Each repository card includes the repository name, a brief description, the file type (Jupyter Notebook), and star/fork counts. The repositories are: `2020-PyTorch-Colab` (1 star, 1 fork), `AI-Lecture-notes` (1 star), `2017-PyTorchDL` (Jupyter Notebook), `RL-Mobile-Robot` (Jupyter Notebook), `RL-Robot-Arm` (Jupyter Notebook), and `Orange-for-ML` (Jupyter Notebook). A `Customize your pins` link is visible in the top right of the repository grid.

TienLungSun (Tien-Lung Sun) x +

github.com/TienLungSun

應用程式 Google 學術搜尋 YouTube Colaboratory GitHub TienLungS... 李弘毅 ML 元智大學個人portal

Search or jump to... Pull requests Issues Marketplace Explore

Overview Repositories 6 Projects Packages

Popular repositories Customize your pins

2020-PyTorch-Colab
This repository contains PyTorch programs for my AI Deep Learning courses. These programs are designed to run in Colab. Please see "0. How to run my GitHub programs from Colab".
Jupyter Notebook ☆ 1 🔗 1

AI-Lecture-notes
☆ 1

2017-PyTorchDL
Jupyter Notebook

RL-Mobile-Robot
Jupyter Notebook

RL-Robot-Arm
Jupyter Notebook


Orange-for-ML
Jupyter Notebook

Tien-Lung Sun
TienLungSun
Edit profile
3 followers · 0 following · ☆ 0

<http://github.com/TienLungSun>

Acknowledgement



Machine Learning is so simple





Machine Learning (Hung-yi Lee, NTU)


36 videos • 1,837,804 views • Last updated on Feb 17, 2021


課程網頁: <http://speech.ee.ntu.edu.tw/~tlkagk/c...>


 Hung-yi Lee SUBSCRIBED 


- 


1 ML Lecture 0-1: Introduction of Machine Learning
Hung-yi Lee
WATCHED 38:57
- 

2 ML Lecture 0-2: Why we need to learn machine learning?
Hung-yi Lee
WATCHED 1:20
- 

3 ML Lecture 1: Regression - Case Study
Hung-yi Lee
WATCHED 1:18:35
- 

4 ML Lecture 1: Regression - Demo
Hung-yi Lee
WATCHED 6:53
- 

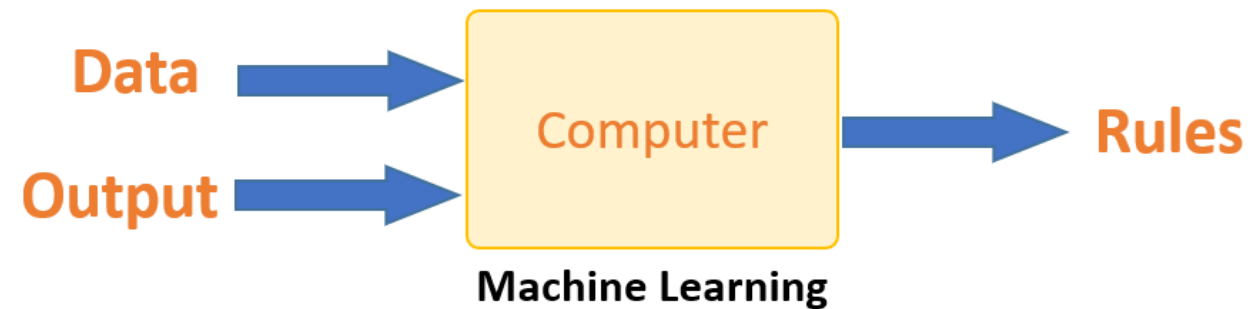
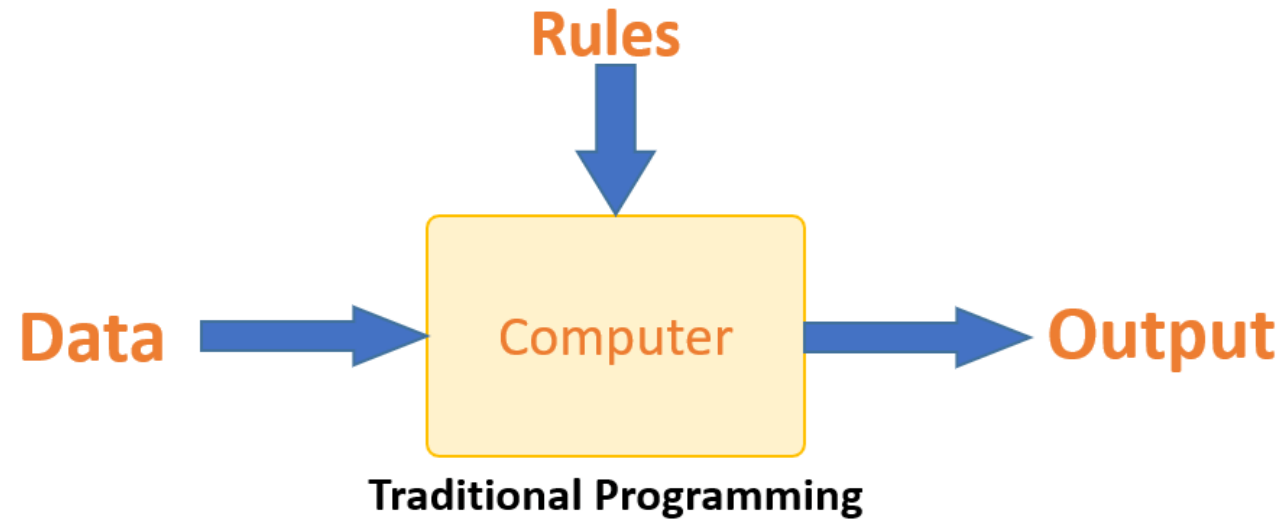
5 ML Lecture 2: Where does the error come from?
Hung-yi Lee
WATCHED 43:15
- 

6 ML Lecture 3-1: Gradient Descent
Hung-yi Lee
WATCHED 1:01:52
- 

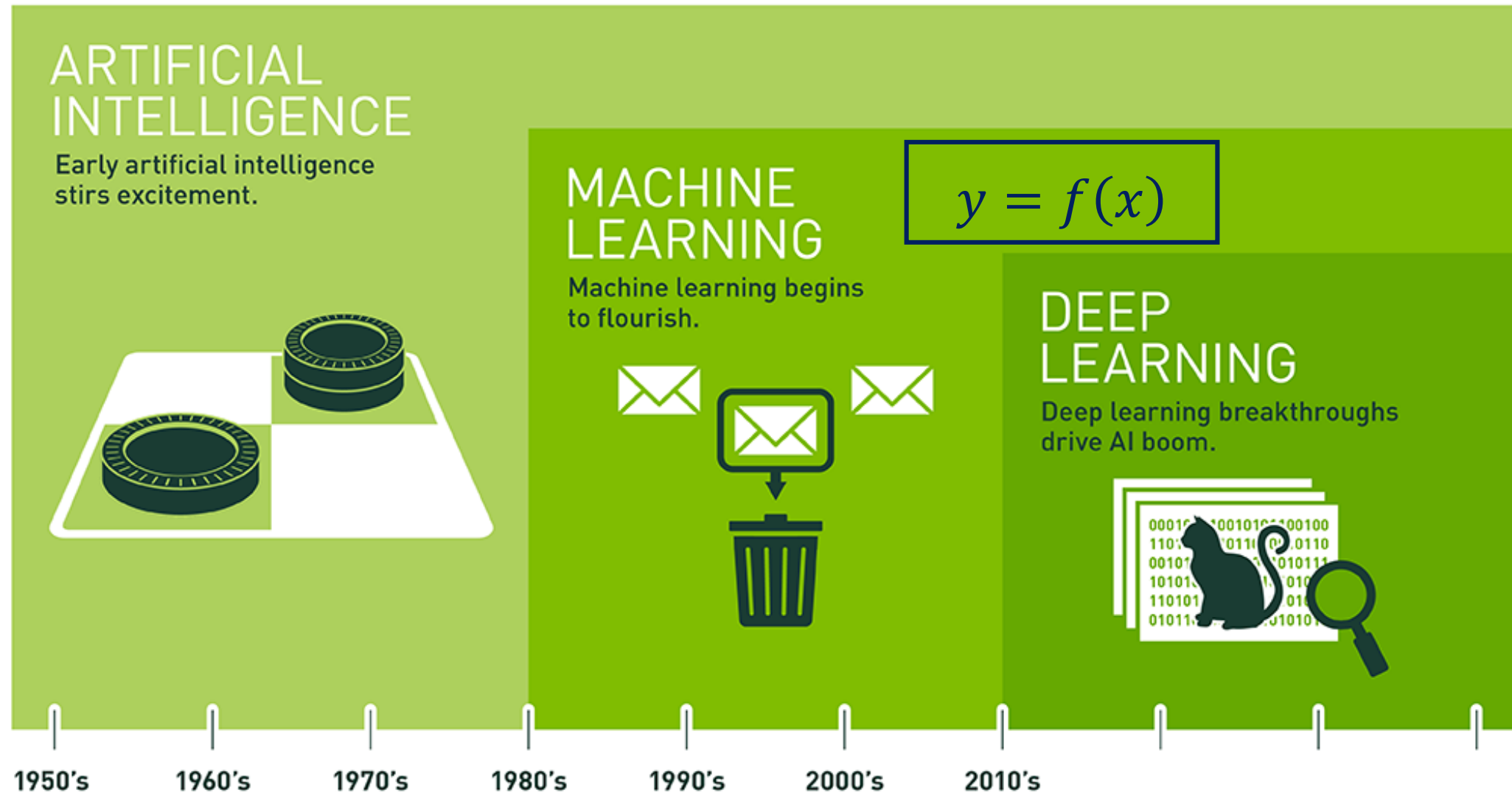
7 ML Lecture 3-2: Gradient Descent (Demo by AOE)
Hung-yi Lee

https://www.youtube.com/playlist?list=PLJV_el3uVTsPy9oCRY30oBPNLCo89yu49

ML vs programming approach to let computer have intelligence



AI, ML and DL



Deep Learning - Machine learns a connected network



Geoffrey Hinton spent 30 years hammering away at an idea most other scientists dismissed as nonsense. Then, one day in 2012, he was proven right. Canada's most influential thinker in the field of artificial intelligence is far too classy to say I told you so

<https://torontolife.com/tech/ai-superstars-google-facebook-apple-studied-guy/>

For more than 30 years, Geoffrey Hinton hovered at the edges of artificial intelligence research, an outsider clinging to a simple proposition: that computers could think like humans do—using intuition rather than rules.

Geoffrey Hinton 多年來堅持着一個簡單的觀點：電腦可以像人類一樣思考-用直覺而不是規則。Hinton 一直好奇的是，電腦能不能像人類大腦一樣的工作：信息通過一個巨大的，由神經元圖譜連接起來的細胞網絡傳播，在多達十億條的路徑上發射、連接和傳輸。

Machine learning mechanism

1. Define a function to be learned: $y^n = f(x^n)$
2. Define a loss function $\mathcal{L}(f)$ to describe the error between y^n and \hat{y}^n
3. Find the optimal parameters that minimize $\mathcal{L}(f)$

Learning strategies

	Supervised Learning	Self-supervised Learning	Reinforcement Learning
1. Function to be learned	MLP, CNN families	AE/VAE, GAN	Actor
	$y = f(x)$	$\hat{x} = f(x)$	$a = f(s)$
2. Loss function $\mathcal{L}(f)$	MSE, CE	MSE, CE, KLD, JSD	MSE, KLD
3. Minimize $\mathcal{L}(f)$	Gradient decent, Maximum Likelihood		

Intelligent skills learned by SL, SSL and RL

	Supervised Learning	Self-supervised Learning	Reinforcement Learning
Function learned	MLP, CNN families	AE/VAE, GAN	Actor
	$y = f(x)$	$\hat{x} = f(x)$	$a = f(s)$
Intelligence	Recognition		Interact with dynamic, adversarial environment
Skills learned	<ul style="list-style-type: none">• Regression• Classification• CV tasks (Object detection, tracking)	<ul style="list-style-type: none">• Feature extraction• Image generation• Anomaly detection	<ul style="list-style-type: none">• Play chess• Play video game• Robot (arm) that can play with people

More...

- Labelling cost in SL
- SSL is particularly suitable in domains that have
 - Large-amount of data, e.g., FB, Youtube, ...
 - Unbalanced data, e.g., manufacturing, health-care
- RL has to consider
 - Prepare training environment, e.g., Unity and ML Agent
 - Reward engineering
 - Other training settings
 - Timed delay and adversarial interaction
- Integrate pre-trained models with application software, e.g., flask, app
- Deployed to edge computing devices (Jetson Nano, Xavier)

Challenges in learning $a=f(s)$

- Define a function to be learned: $a = f(s)$
- Define a loss function $\mathcal{L}(f)$ to describe the error between y^n and

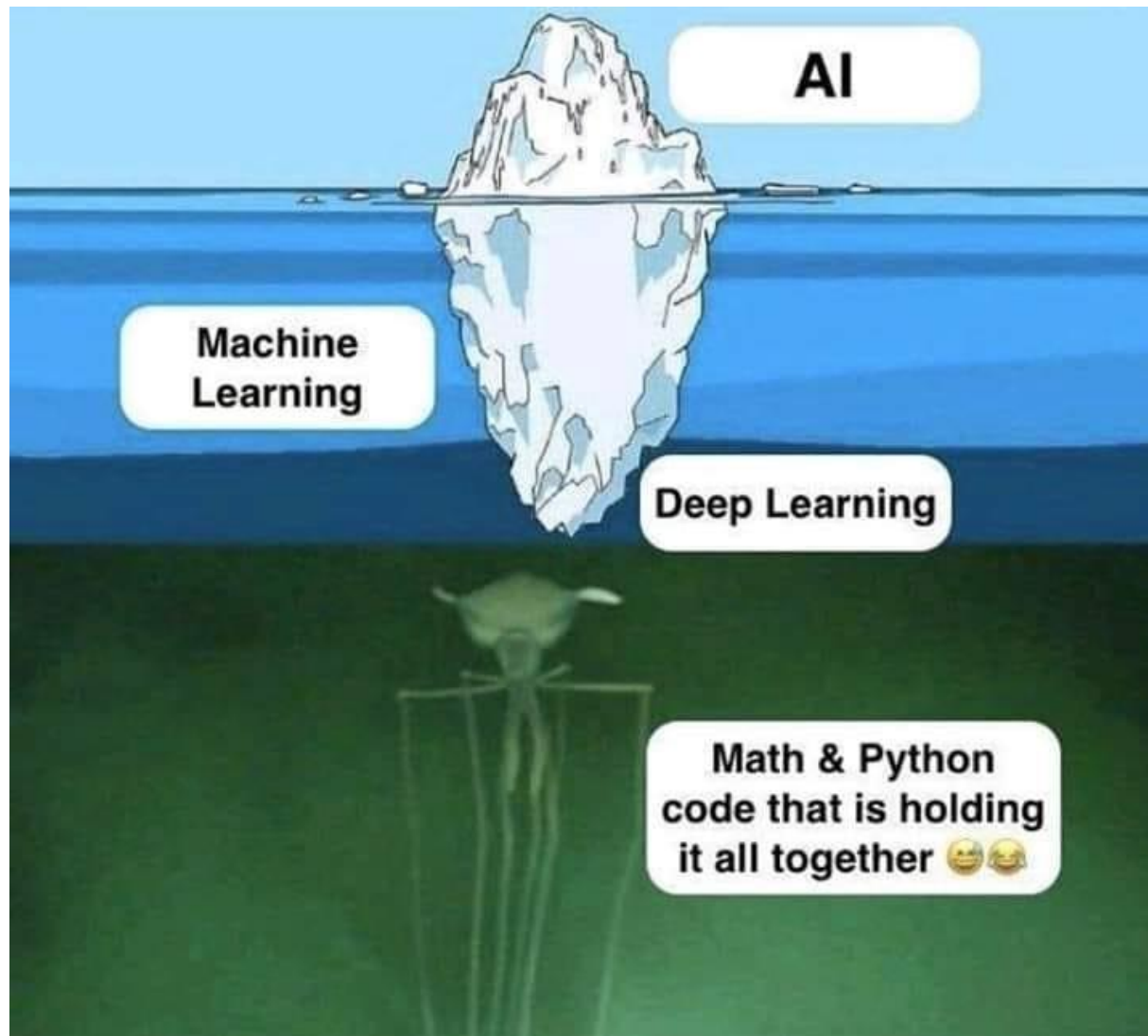
\hat{y}^n

Time-delayed answer – If at time t we perform action a_t under state s_t , we only know the immediate reward r_t and there is a time delay to know the total accumulated reward \hat{y} .

Adversarial interaction – After performing a_t at state s_t , there are infinite number of possibilities for following state- actions $s_{t+1}, a_{t+1}, s_{t+2}, a_{t+2}, \dots, s_{t+T}, a_{t+T}$. It is difficult to estimate the true answer y (the true final reward).

- Find the optimal parameters that minimize $\mathcal{L}(f)$

Math and coding



- Probability and statistics
- Non-linear optimization
- Linear algebra

Python



Run my PyTorch code in GitHub from Colab

The screenshot shows the Google Colaboratory interface with the GitHub integration menu open. The 'GitHub' tab is selected in the top navigation bar. The search bar contains the text 'TienLungSun'. Below the search bar, the repository 'TienLungSun/2020-PyTorch-Colab' is selected. The file '1. 2. MLP regression.ipynb' is highlighted in the list of files. The interface also shows a sidebar with a directory tree and a bottom status bar with the text 'seconds_in_a_day'.

歡迎使用 Colaboratory

檔案 編輯 檢視

範例 最近 Google 雲端硬碟 **GitHub** 上傳

輸入 GitHub 網址或依機構或使用者搜尋 ☐ 包括私人存放區

TienLungSun

存放區: [\[icon\]](#) 分支版本: [\[icon\]](#)

TienLungSun/2020-PyTorch-Colab [\[icon\]](#) main [\[icon\]](#)

路徑

- 1. 1. Understand MLP .ipynb
- 1. 2. MLP regression.ipynb**
- 1. 3. MLP Classification.ipynb
- 2. 1. Understand CNN .ipynb

取消

seconds_in_a_day

在這裡輸入文字來搜尋

下午 09:10 2021/2/25

Notations

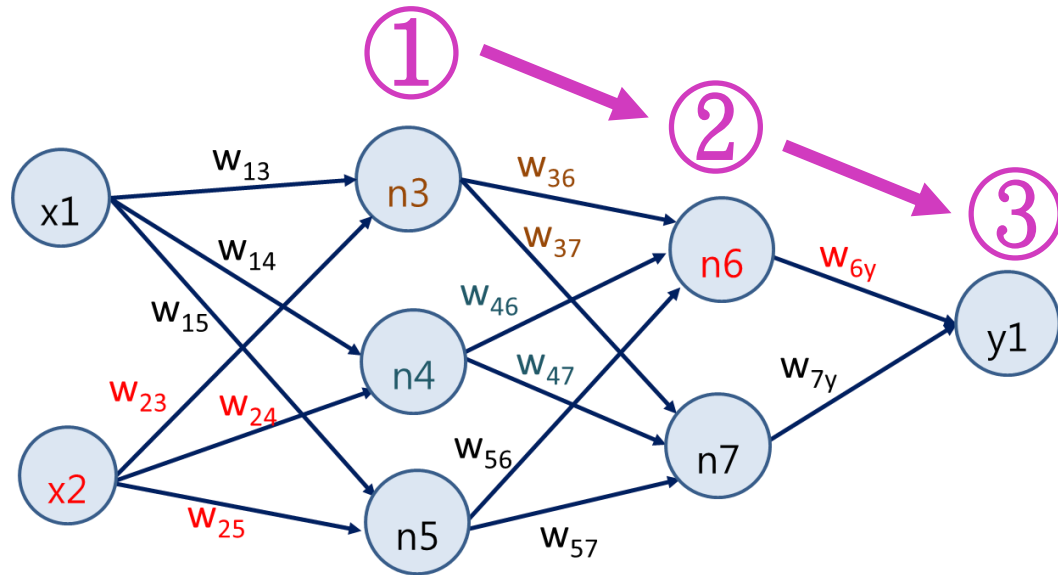
x_i

No	age	t1	t2	t3	t4	t5	t6	time	Step frequency	n1	n2	n3	n4	n5	n6	px	py	pz	Steps	Gender	TUG	y1	BBS	y2
1	70	1.76	2.64	6.24	7.02	10	12.8	11	2.285	80	120	282	317	453	575	11.67	1.809	-1.99	13	F	11	0	26	0
2	86	1.64	2.6	5.82	7.27	10.4	12.6	11	1.934	75	118	263	328	470	570	11.14	2.302	-4.651	12	F	11	0	24	0
3	76	1.76	2.93	6.27	7.04	10.3	12.8	11	2.109	80	133	283	318	465	575	11.53	2.169	-3.253	14	F	11	0	22	1
4	70	2.38	3.29	5.58	6.47	9.02	10.4	8	2.461	108	149	252	292	407	468	11.6	1.838	-3.138	12	F	8	0	24	0
5	66	3.09	4.07	6.6	7.4	10.2	12.1	9	2.461	140	184	298	334	462	545	11.55	2.531	-2.742	12	F	9	0	26	0
6	79	1.76	2.91	5.87	6.6	10.2	12.8	11	2.109	80	132	265	298	462	575	x_i^n	1.788	-1.349	13	F	11	0	26	0
7	85	1.2	2.33	5.42	8.31	12.1	17.2	16	2.988	55	106	245	375	545	775		2.203	-4.89	17	M	16	1	18	1
8	81	1.64	2.93	5.98	7.47	10.9	13.6	12	1.758	75	133	270	337	493	615		2.667	-4.594	10	F	12	0	24	0
9	82	0.64	1.47	4.76	5.76	9.36	11.6	11	2.109	30	67	215	260	422	525	11.26	4.1	-2.693	14	M	11	0	24	0
10	69	1.64	2.49	5.02	5.98	9.82	12.6	11	2.637	75	113	227	270	443	570	11.27	3.292	-3.522	13	F	11	0	20	1
11	84	0.64	1.4	5.67	7.29	11.5	14.6	14	1.934	30	64	256	329	520	660	11.53	2.335	-2.999	15	M	14	1	26	0
12	69	1.09	1.98	5	5.62	8.38	10.1	9	2.109	50	90	226	254	378	455	11.15	1.919	-4.608	11	M	9	0	26	0
13	73	1.09	2.13	6.78	8.38	12.4	17.1	16	3.691	50	97	306	378	558	770	11.46	2.264	-3.333	16	F	16	1	14	1
14	81	0.64	1.87	9.24	11.2	19	22.6	22	1.934	30	85	417	507	857	1020	11.58	2.511	-2.157	27	M	22	1	24	0
15	80	0.76	1.71	3.98	5	7.58	9.76	9	2.109	35	78	180	226	342	440	11.33	2.821	-3.595	10	M	9	0	26	0
16	88	0.98	2.13	6.31	7.44	11.5	14	13	1.934	45	97	285	336	518	630	11.38	2.498	-3.702	16	M	14	1	26	0
17	81	1.09	2.09	4.18	5.16	7.76	10.1	9	2.285	50	95	189	233	350	455	11.21	2.241	-4.337	10	M	9	0	28	0
18	76	1.76	2.64	5.87	6.98	9.98	12.8	11	1.406	80	120	265	315	450	575	11.33	2.679	-3.736	10	M	11	0	26	0
19	69	0.36	3.76	13.3	16.7	24.2	29.4	29	3.691	17	170	598	753	1090	1322	11.31	1.361	-4.171	28	F	29	1	10	1
20	75	1.98	2.93	5.98	7.91	12.2	15	13	1.934	90	133	270	357	551	675	11.5	2.202	-1.495	14	M	13	0	28	0
21	87	1.53	3.2	10.9	13.8	21.3	26.5	25	2.9	70	145	492	624	960	1195	11.6	2.199	-2.54	19	F	25	1	16	1
22	72	0.2	1.02	3.36	4.11	7.42	10.2	10	1.758	10	47	152	186	335	460	11.52	2.658	-2.081	9	M	10	0	28	0
23	109	0.64	1.93	5.04	5.71	9.13	10.6	10	2.285	30	88	228	258	412	480	11.51	2.056	-3.158	15	F	10	0	28	0

\hat{y}^n

Multiple-layer perceptron (MLP)

1. Define a function to be learned: $y^n = f(x^n)$



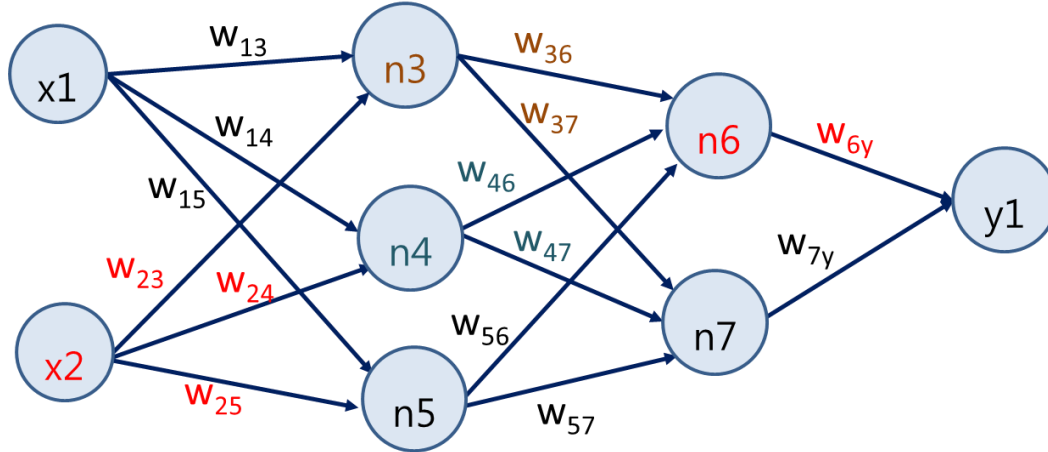
$$\begin{aligned} n_3 &= \sigma(x_1 * w_{13} + x_2 * w_{23} + b_3) \\ \textcircled{1} \quad n_4 &= \sigma(x_1 * w_{14} + x_2 * w_{24} + b_4) \\ n_5 &= \sigma(x_1 * w_{15} + x_2 * w_{25} + b_5) \end{aligned}$$

$$\begin{aligned} \textcircled{2} \quad n_6 &= \sigma(n_3 * w_{36} + n_4 * w_{46} + n_5 * w_{56} + b_6) \\ n_7 &= \sigma(n_3 * w_{37} + n_4 * w_{47} + n_5 * w_{57} + b_7) \end{aligned}$$

$$\textcircled{3} \quad y_1 = \sigma(n_6 * w_{6y} + n_7 * w_{7y} + b_y)$$

Calculate error

2. Define a loss function $\mathcal{L}(f)$ to describe the error between y^i and \hat{y}^i



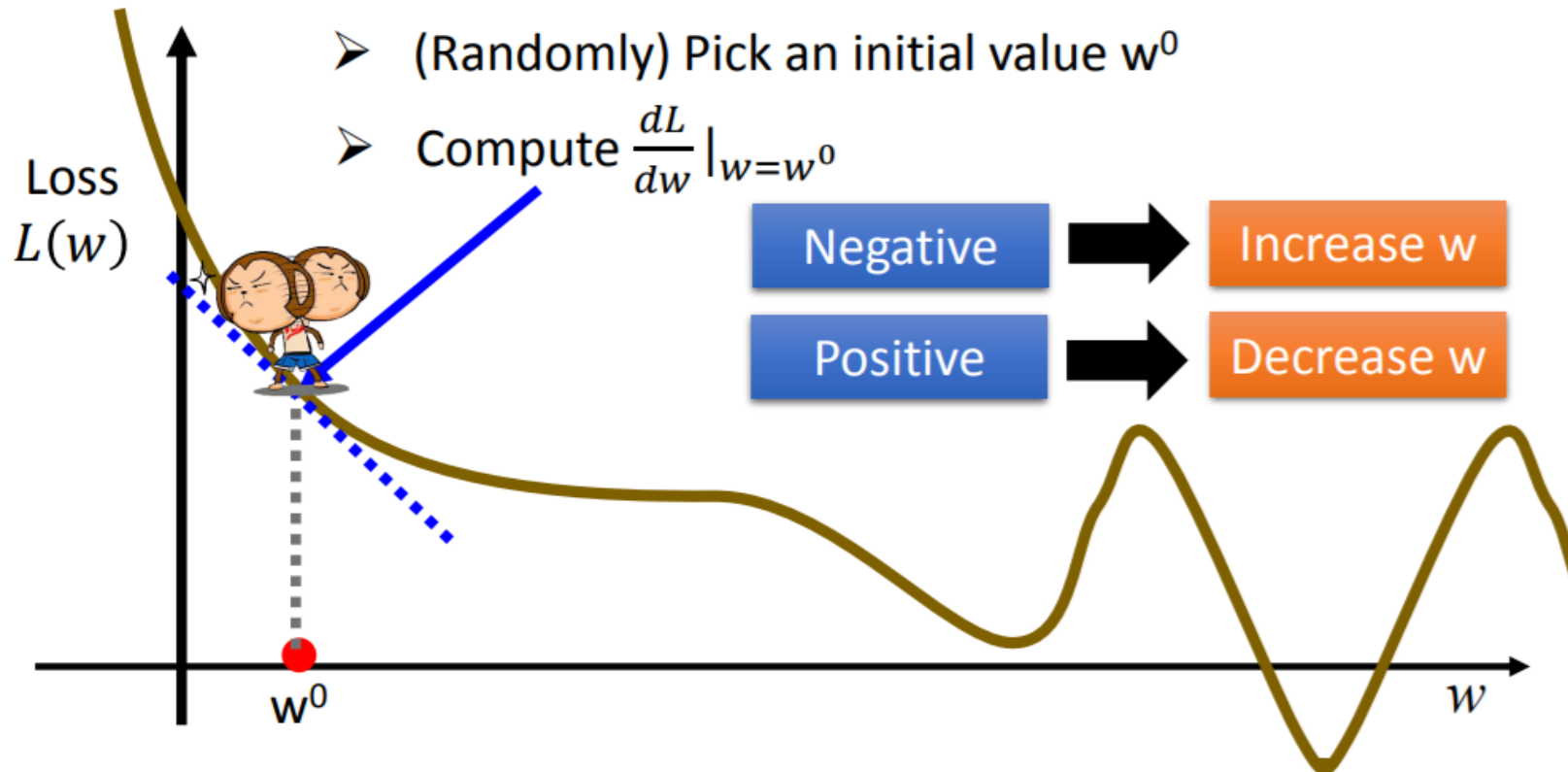
$$L = \frac{1}{N} \sum_{i=1}^N (y^i - \hat{y}^i)^2$$

Use gradient decent to find optimal parameters

3. Find the optimal parameters that minimize $\mathcal{L}(f)$

$$w^* = \arg \min_w L(w)$$

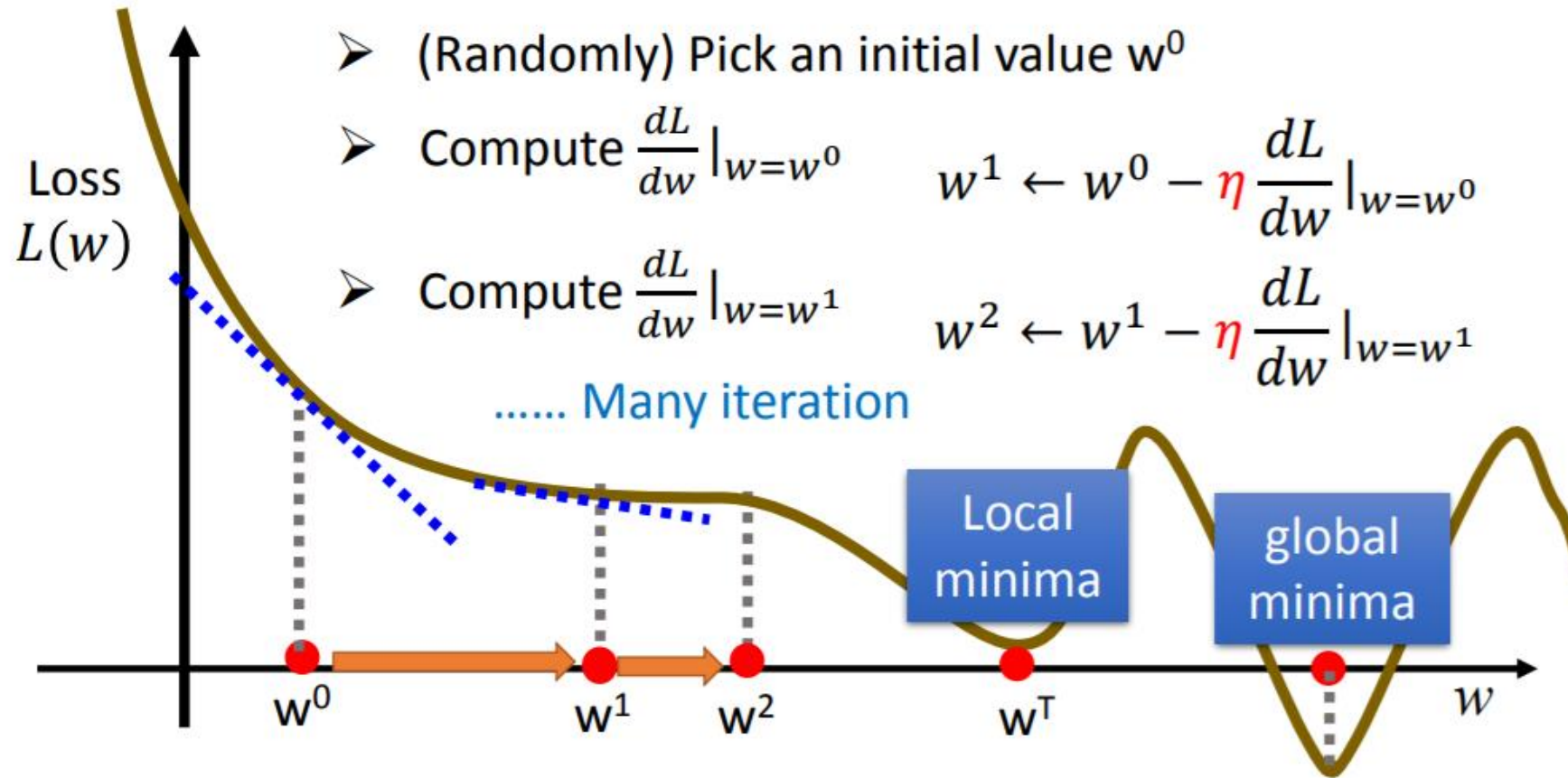
- Consider loss function $L(w)$ with one parameter w :



Use gradient decent to find optimal parameters

$$w^* = \arg \min_w L(w)$$

- Consider loss function $L(w)$ with one parameter w :



Gradient decent to find two parameters w^* and b^*

- How about two parameters? $w^*, b^* = \arg \min_{w, b} L(w, b)$

➤ (Randomly) Pick an initial value w^0, b^0

➤ Compute $\frac{\partial L}{\partial w} \big|_{w=w^0, b=b^0}, \frac{\partial L}{\partial b} \big|_{w=w^0, b=b^0}$

$$\begin{bmatrix} \frac{\partial L}{\partial w} \\ \frac{\partial L}{\partial b} \end{bmatrix} \text{gradient}$$

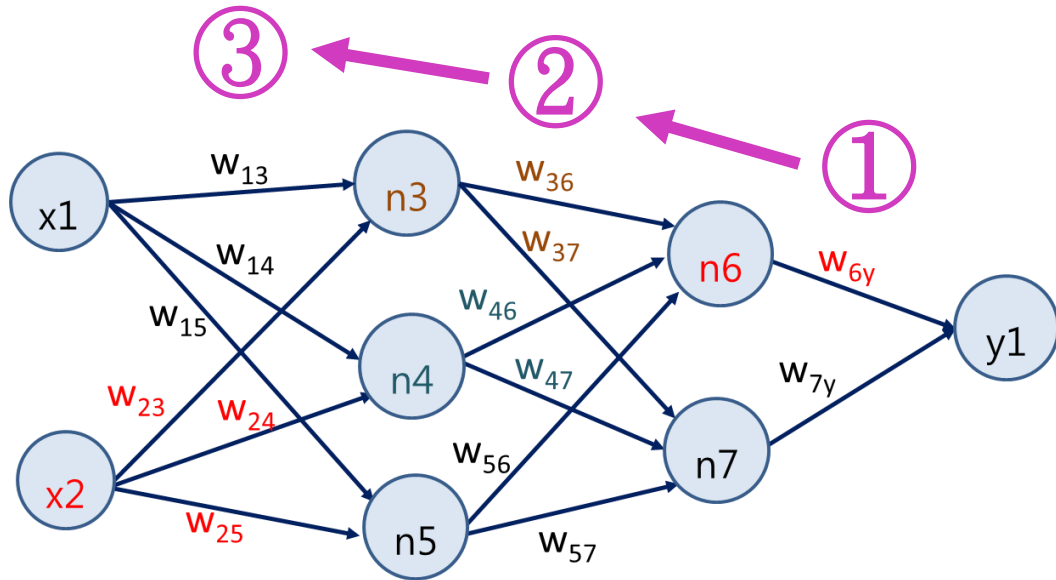
$$w^1 \leftarrow w^0 - \eta \frac{\partial L}{\partial w} \big|_{w=w^0, b=b^0} \quad b^1 \leftarrow b^0 - \eta \frac{\partial L}{\partial b} \big|_{w=w^0, b=b^0}$$

➤ Compute $\frac{\partial L}{\partial w} \big|_{w=w^1, b=b^1}, \frac{\partial L}{\partial b} \big|_{w=w^1, b=b^1}$

$$w^2 \leftarrow w^1 - \eta \frac{\partial L}{\partial w} \big|_{w=w^1, b=b^1} \quad b^2 \leftarrow b^1 - \eta \frac{\partial L}{\partial b} \big|_{w=w^1, b=b^1}$$

Use gradient decent to find optimal NN weights

3. Find the optimal parameters that minimize $\mathcal{L}(f)$



$$w_i \leftarrow w_i - \eta \frac{\partial e}{\partial w_i}$$

$$L = g(y - \hat{y}) \quad y = \sigma(n_6 * w_{6y} + n_7 * w_{7y} + b_y)$$

①

$$w_{6y} \leftarrow w_{6y} - \eta \frac{\partial L}{\partial w_{6y}} \quad \frac{\partial L}{\partial w_{6y}} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{6y}}$$

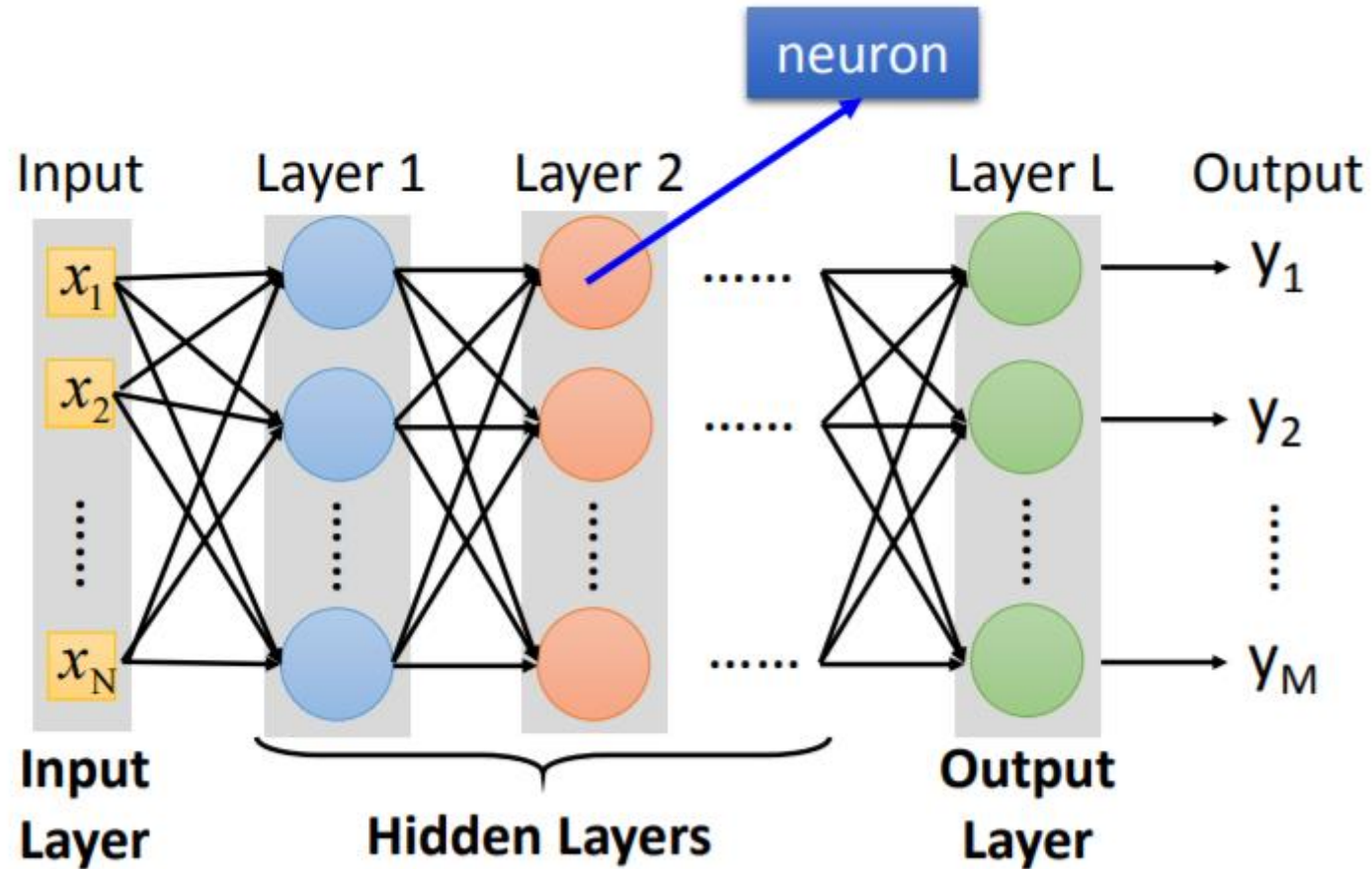
$$w_{7y} \leftarrow w_{7y} - \eta \frac{\partial L}{\partial w_{7y}} \quad \frac{\partial L}{\partial w_{7y}} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{7y}}$$

②

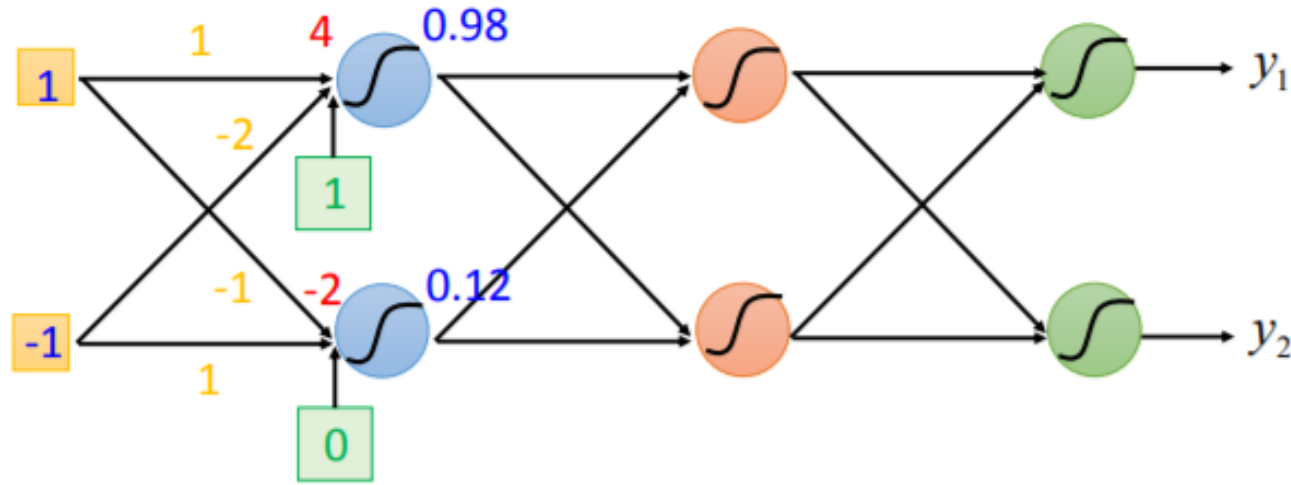
$$w_{57} \leftarrow w_{57} - \eta \frac{\partial L}{\partial w_{57}} \quad \frac{\partial L}{\partial w_{57}} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial n_7} \frac{\partial n_7}{\partial w_{57}}$$

$$n_7 = f(n_3 * w_{37} + n_4 * w_{47} + n_5 * w_{57} + b_7)$$

MLP is a fully connected feedforward network



Fully connected feed forward network is implemented as matrix operation



$$y = \sigma(w \cdot x + b)$$

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

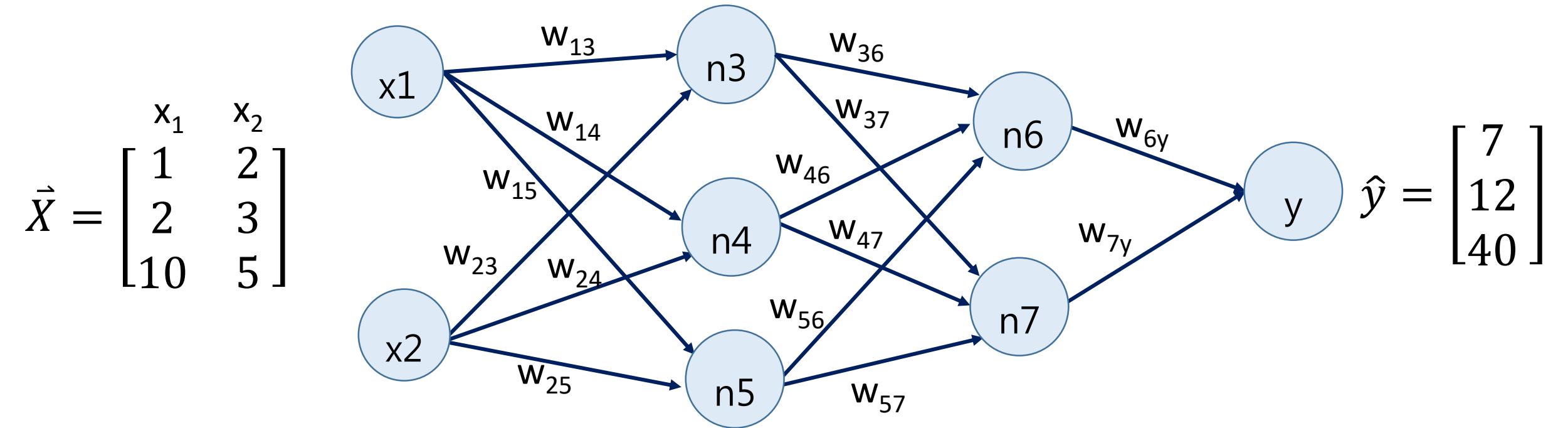
Practice

- Run "1.1 Matrix operation.ipynb"



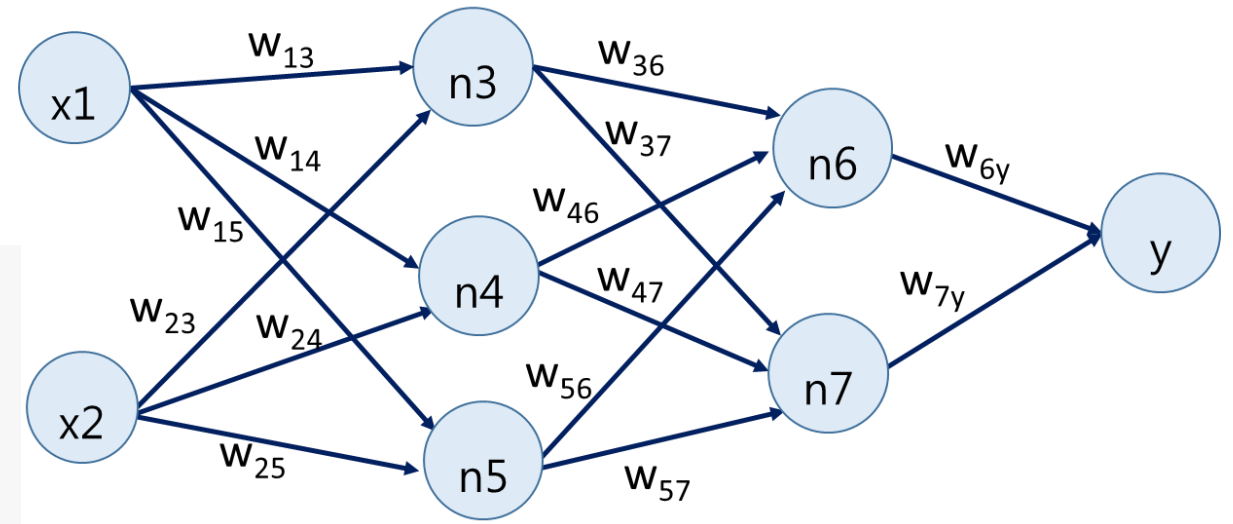
Matrix operation

```
MyNet = nn.Sequential(  
    nn.Linear(2, 3),  
    nn.Linear(3, 2),  
    nn.Linear(2, 1)  
)
```



Matrix operation

```
for param in MyNet.parameters():  
    if param.requires_grad:  
        print(param.data)
```



tensor([[0.4727, -0.5188],
 [-0.5681, -0.6032],
 [-0.0252, -0.3011]])

tensor([-0.6986, -0.6602, -0.4860])

tensor([[-0.5549, 0.2550, 0.4584],
 [0.2930, 0.0849, -0.3146]])

tensor([0.1677, 0.0736])

tensor([[0.4106, -0.3618]])

tensor([-0.2270])

$\begin{bmatrix} w_{13} & w_{23} \\ w_{14} & w_{24} \\ w_{15} & w_{25} \end{bmatrix}$

$[b_3 \quad b_4 \quad b_5]$

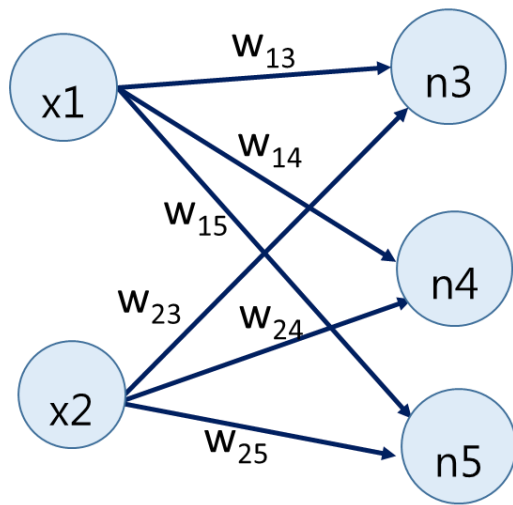
$\begin{bmatrix} w_{36} & w_{46} & w_{56} \\ w_{37} & w_{47} & w_{57} \end{bmatrix}$

$[b_6 \quad b_7]$

$[w_{6y} \quad w_{7y}]$

$[b_y]$

$$\vec{X} = \begin{bmatrix} x_1 & x_2 \\ 1 & 2 \\ 2 & 3 \\ 10 & 5 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 10 & 5 \end{bmatrix} \begin{bmatrix} w_{13} & w_{14} & w_{15} \\ w_{23} & w_{24} & w_{25} \end{bmatrix} + \begin{bmatrix} b_3 & b_4 & b_5 \end{bmatrix}$$

$$\begin{bmatrix} k_3^1 & k_4^1 & k_5^1 \\ k_3^2 & k_4^2 & k_5^2 \\ k_3^3 & k_4^3 & k_5^3 \end{bmatrix} + \begin{bmatrix} b_3 & b_4 & b_5 \\ b_3 & b_4 & b_5 \\ b_3 & b_4 & b_5 \end{bmatrix}$$

$$\begin{bmatrix} n_3^1 & n_4^1 & n_5^1 \\ n_3^2 & n_4^2 & n_5^2 \\ n_3^3 & n_4^3 & n_5^3 \end{bmatrix}$$

Use Excel to verify

```
W1 = MyNet[0].weight
b1 = MyNet[0].bias
print(W1, W1.shape, b1)
```

Parameter containing:

```
tensor([[ 0.4727, -0.5188],
        [-0.5681, -0.6032],
        [-0.0252, -0.3011]],
        tensor([-0.6986, -0.6602, -0.4860], r
```

```
#Calculate n3, n4, n5
HiddenLayer1 = MyNet[0](tensorX)
print(HiddenLayer1)
```

```
tensor([[ -1.2635, -2.4348, -1.1135],
        [-1.3097, -3.6061, -1.4398],
        [ 1.4340, -9.3577, -2.2441]],
```

```
#Calculate n3, n4, n5 using Pytorch matrix operation
HiddenLayer1 = tensorX.mm(torch.transpose(W1, 1, 0)) + b1
print(HiddenLayer1)
```

```
tensor([[ -1.2635, -2.4348, -1.1135],
        [-1.3097, -3.6061, -1.4398],
        [ 1.4340, -9.3577, -2.2441]], grad_fn=<AddBackward0>)
```

```
#Calculate n6, n7 using PyTorch matrix operation
W2 = MyNet[1].weight
b2 = MyNet[1].bias
HiddenLayer2 = HiddenLayer1.mm(torch.transpose(W2, 1, 0)) + b2
print(HiddenLayer2)
```

```
tensor([[ -0.2625, -0.1530],
        [-0.6852, -0.1632],
        [-4.0429,  0.4054]], grad_fn=<AddBackward0>)
```

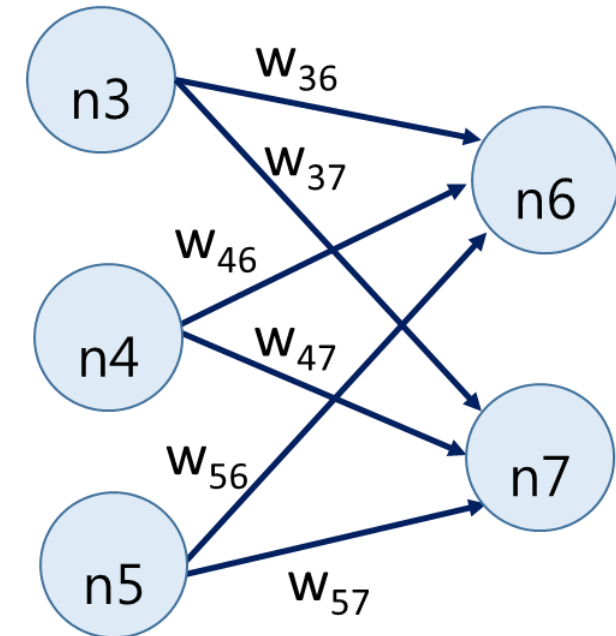
$$\begin{bmatrix} n_3^1 & n_4^1 & n_5^1 \\ n_3^2 & n_4^2 & n_5^2 \\ n_3^3 & n_4^3 & n_5^3 \end{bmatrix}$$

$$\begin{bmatrix} -1.2635 & -2.4348 & -1.1135 \\ -1.3097 & -3.6061 & -1.4398 \\ 1.4340 & -9.3577 & -2.2441 \end{bmatrix} \begin{bmatrix} w_{36} & w_{37} \\ w_{46} & w_{47} \\ w_{56} & w_{57} \end{bmatrix} + [b_6 \quad b_7]$$

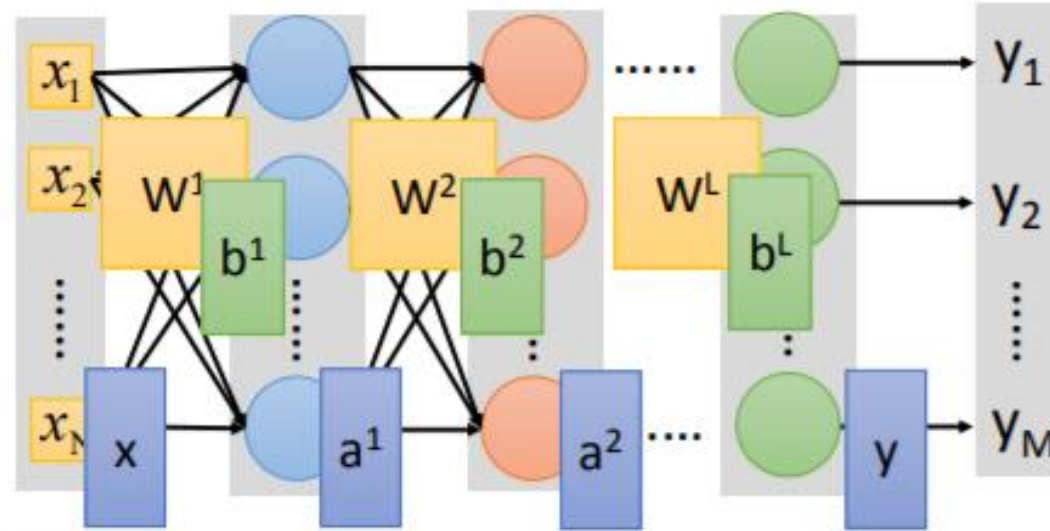
$$\begin{bmatrix} k_6^1 & k_7^1 \\ k_6^2 & k_7^2 \\ k_6^3 & k_7^3 \end{bmatrix} + \begin{bmatrix} b_6 & b_7 \\ b_6 & b_7 \\ b_6 & b_7 \end{bmatrix}$$

$$\begin{bmatrix} n_6^1 & n_7^1 \\ n_6^2 & n_7^2 \\ n_6^3 & n_7^3 \end{bmatrix}$$

Use Excel to
verify



Use parallel computing to speed up matrix operation



$$y = f(x)$$

Using parallel computing techniques
to speed up matrix operation

$$= \sigma(W^L \dots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \dots + b^L)$$

Use parallel computing to speed up matrix operation

```
In [2]: if(torch.cuda.is_available()):  
        device = torch.device("cuda")  
        print(device, torch.cuda.get_device_name(0))  
    else:  
        device= torch.device("cpu")  
        print(device)
```

cuda Tesla P100-PCIE-16GB

```
tensorX = torch.FloatTensor(trainX).to(device)  
tensorY_hat = torch.FloatTensor(trainY_hat).to(device)  
print(tensorX.shape, tensorY_hat.shape)
```

torch.Size([128, 2]) torch.Size([128, 1])

```
conv1_out = conv1(imageTensor.to(device))  
conv1_out.shape  
#output image (feature map) has 64 channels
```

torch.Size([1, 64, 55, 55])