Deep Learning : Representation Power of Functions

Lecture - 7

राष्ट्रीय प्रौद्योगिकी संस्थान सिक्किम
NATIONAL INSTITUTE OF TECHNOLOGY SIKKIM

Course Instructor:
Dr. Bam Bahadur Sinha
Assistant Professor
Computer Science & Engineering
National Institute of Technology
Sikkim

Parameter update Rule & Stopping Criteria

Parameter Update Rule

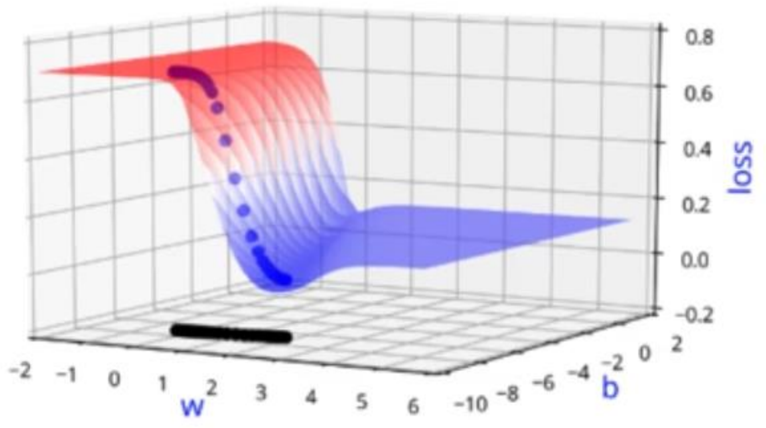
$$w_{t+1} = w_t - \eta \Delta w_t$$
$$b_{t+1} = b_t - \eta \Delta b_t$$

$$where\ \Delta w_t = \frac{\partial \mathscr{L}(w,b)}{\partial w}_{at\ w=w_t, b=b_t}, \Delta b_t = \frac{\partial \mathscr{L}(w,b)}{\partial b}_{at w=w_t, b=b_t}$$

i. Either set the number of passes (eg. 100 or 1000)
ii. Set any Epsilon value to denote the loss
iii. Check ‘w’ and ‘b’ across two iterations, if there is no much change between w(t+1) and w(t) && b(t+1) and b(t). Then we can STOP.

Full Learning Algorithm – Full Code (Reference)

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x):
    #sigmoid with parameters w, b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error(w, b):
    err = 0.0
    for x, y in zip(X, Y):
        fx = f(w, b, x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta = -2, -2, 1.0
    max_epochs = 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



Emergency Room Visits	Narcotics	Pain	Total Visits	Medical Claims	PoorCare
0	2	6	11	53	1
1	1	4	25	40	0
0	0	5	10	28	0
1	3	5	7	20	1

What happens when we have more than 2 parameters ?

$$z = w_1 * ER\_visits + w_2 * Narcotics + w_3 * Pain + w_4 * TotalVisits + w_5 * MedicalClaims + b$$

$$z = w_1 * x_{i1} + w_2 * x_{i2} + w_3 * x_{i3} + w_4 * x_{i4} + w_5 * x_{i5} + b$$

$$\hat{y} = \frac{1}{1+e^{-z}}$$

$$\hat{y} = \frac{1}{1+e^{-(w_1 * x_{i1} + w_2 * x_{i2} + w_3 * x_{i3} + w_4 * x_{i4} + w_5 * x_{i5} + b)}}$$

Initialise $w_1, w_2, \ldots, w_5, b$

Iterate over data:

$$w_1 = w_1 - \eta \Delta w_1$$

$$w_2 = w_2 - \eta \Delta w_2$$

$$\vdots$$

$$w_5 = w_5 - \eta \Delta w_5$$

$$b = b - \eta \Delta b$$

till satisfied

$$\Delta w = \sum_{i=1}^{m} (f(x) - y) * f(x) * (1 - f(x)) * x$$

$$\Delta w_1 = \sum_{i=1}^{m} (\hat{y} - y) * \hat{y} * (1 - \hat{y}) * x_{i1}$$

$$\Delta w_2 = \sum_{i=1}^{m} (\hat{y} - y) * \hat{y} * (1 - \hat{y}) * x_{i2}$$

$$\Delta w_j = \sum_{i=1}^{m} (\hat{y} - y) * \hat{y} * (1 - \hat{y}) * x_{ij}$$

```python
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x):
    #sigmoid with parameters w, b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error(w, b):
     err = 0.0
     for x, y in zip(X, Y):
        fx = f(w, b, x)
        err += 0.5* (fx - y) ** 2
     return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta = -2, -2, 1.0
    max_epochs = 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

```python
def f(w, b, x):
    #sigmoid with parameters w, b
    return 1.0 / (1.0 + np.exp(-(np.dot(w, x) + b)))
```

```python
def grad_w_i(w, b, x, y, i):
     fx = f(w, b, x)
     return (fx - y) * fx * (1 - fx) * x[i]
```

Change in the code when we have 2 or more parameters

Evaluation

Training data

Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight	0.19	0.63	0.33	0.99	0.36	0.66	0.1	0.70	0.48
Screen size	0.64	0.87	0.67	0.88	0.7	0.91	0.04	0.98	0.47
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery	0.36	0.51	0.36	0.97	0.34	0.67	0	0.57	0.43
Price	0.09	0.63	0.41	0.19	0.06	0	0.72	0.94	1
Like (y)	**0.9**	**0.3**	**0.85**	**0.2**	**0.5**	**0.98**	**0.1**	**0.88**	**0.23**

Test data

1	0	0	1
0.2	0.73	0.6	0.8
0.2	0.7	0.8	0.9
0	1	0	0
1	0	0	0
0	0	1	0
1	1	1	0
0.83	0.96	0.9	0.2
0.34	0.4	0.6	0.1
0.17	**0.56**	**0.3**	$\boxed{\textbf{0.4}}$
0.24	**0.67**	**0.9**	$\boxed{\textbf{0.3}}$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2}$$

Test data

1	0	0	1
0.2	0.73	0.6	0.8
0.2	0.7	0.8	0.9
0	1	0	0
1	0	0	0
0	0	1	0
1	1	1	0
0.83	0.96	0.9	0.2
0.34	0.4	0.6	0.1
0.17	**0.56**	**0.3**	**0.4**
0	**1**	**1**	**0**

✓ ✓ ✗ ✓

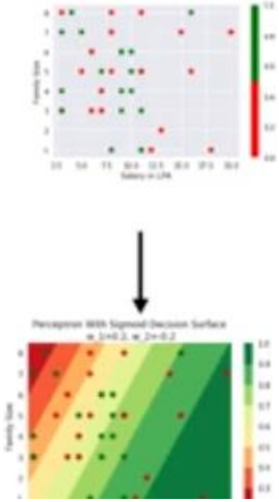
Threshold=0.5

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Summary

Real inputs $\in \mathbb{R}$

Classification /Regression



$Loss = \sum_{i=1}^{n}(y - \hat{y})^2$

$$w = w + \Delta w$$
$$b = b + \Delta b$$

OR

$$w_{t+1} = w_t - \eta \Delta w_t$$
$$b_{t+1} = b_t - \eta \Delta b_t$$

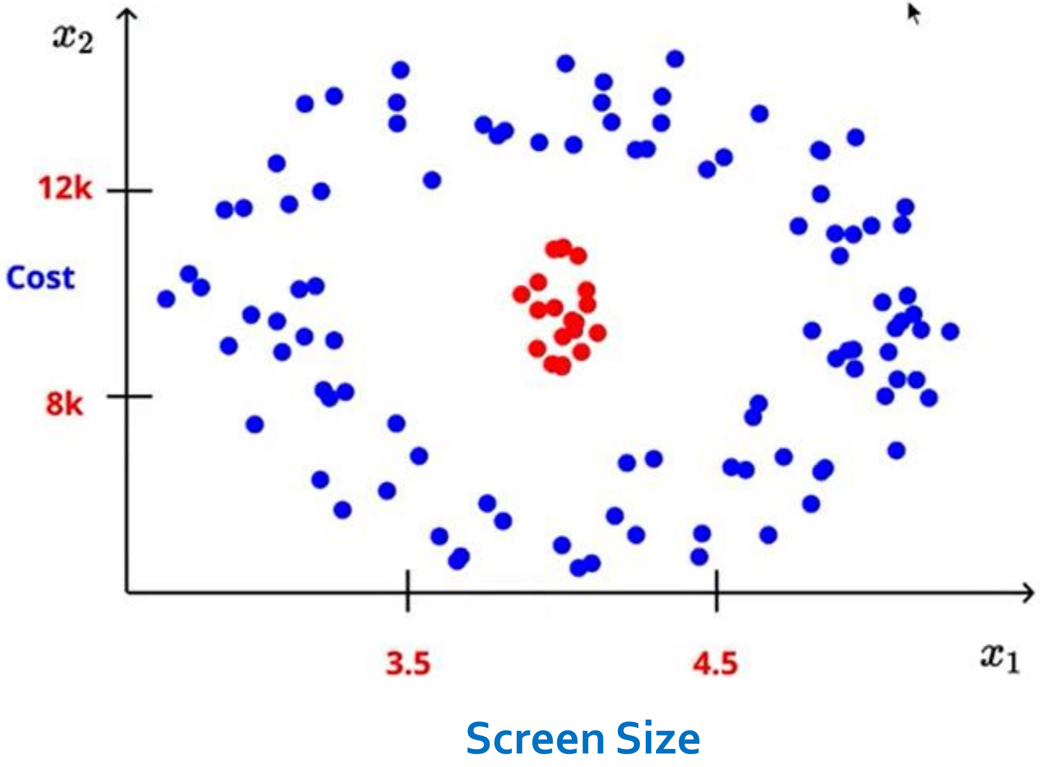
$RMSE$

$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$

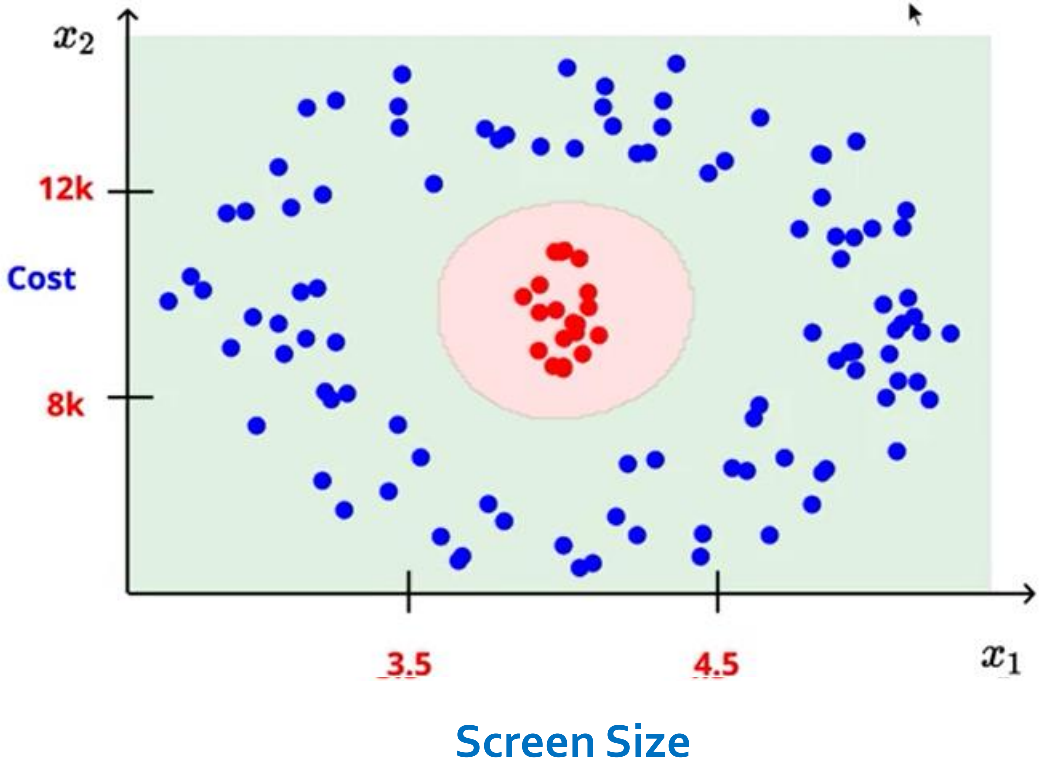
Comparison of MP Neuron, Perceptron & Sigmoid Neuron

	Data	Task	Model	Loss	Learning	Evaluation
MP neuron	$\{0, 1\}$	Binary Classification	$\hat{y} = 1$ if $\sum_{i=1}^{n} x_i \geq b$ $\hat{y} = 0$ otherwise	$Loss = \sum_{i=1}^{n} \mathbf{1}_{(y != \hat{y})}$		$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$
Perceptron	Real inputs	Binary Classification	$\hat{y} = 1$ if $\sum_{i=1}^{n} w_i x_i \geq b$ $\hat{y} = 0$ otherwise	$Loss = \sum_{i=1}^{n} (y - \hat{y})^2$	Perceptron Learning Algorithm	$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$
Sigmoid	Real inputs	Classification/Regression	$y = \frac{1}{1+e^{(-w^T x + b)}}$	$Loss = \sum_{i=1}^{n} (y - \hat{y})^2$	Gradient Descent	$Accuracy/RMSE$

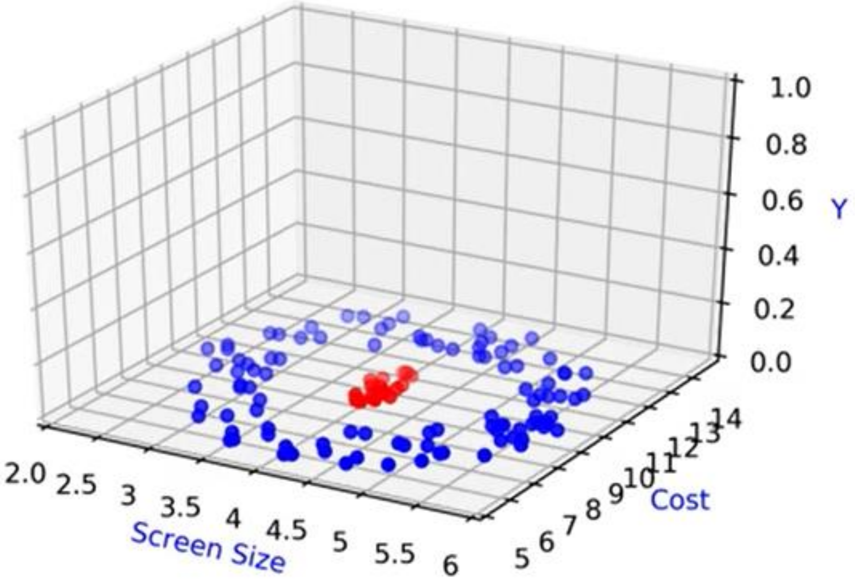
Why do we need complex functions?

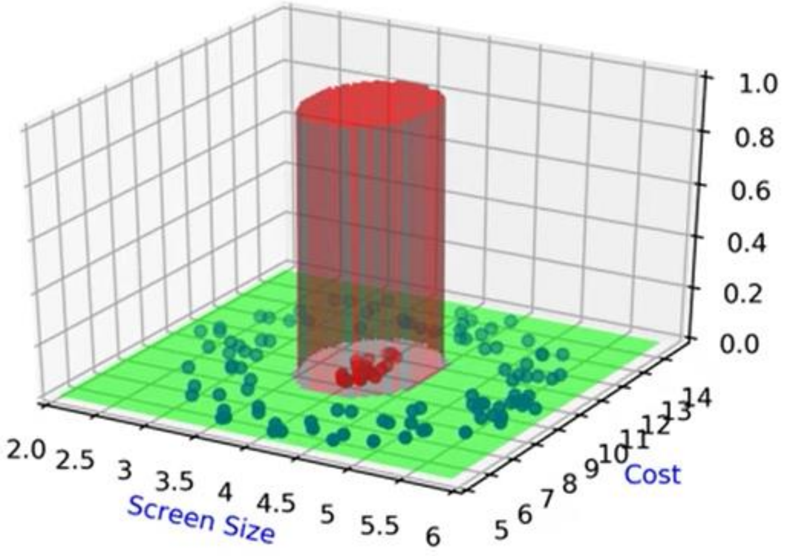


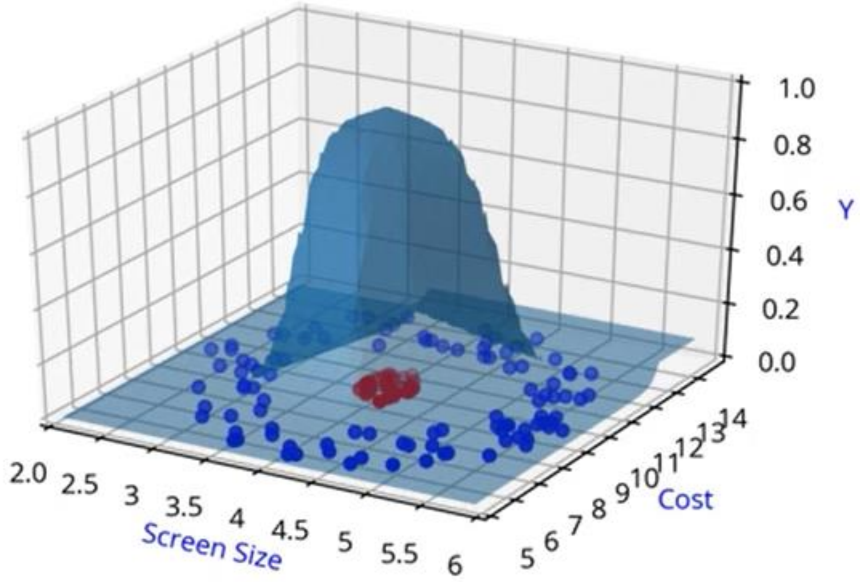
Why do we need complex functions?



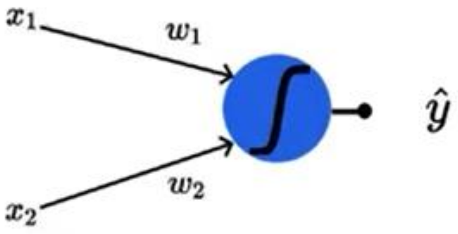
Why do we need complex functions?

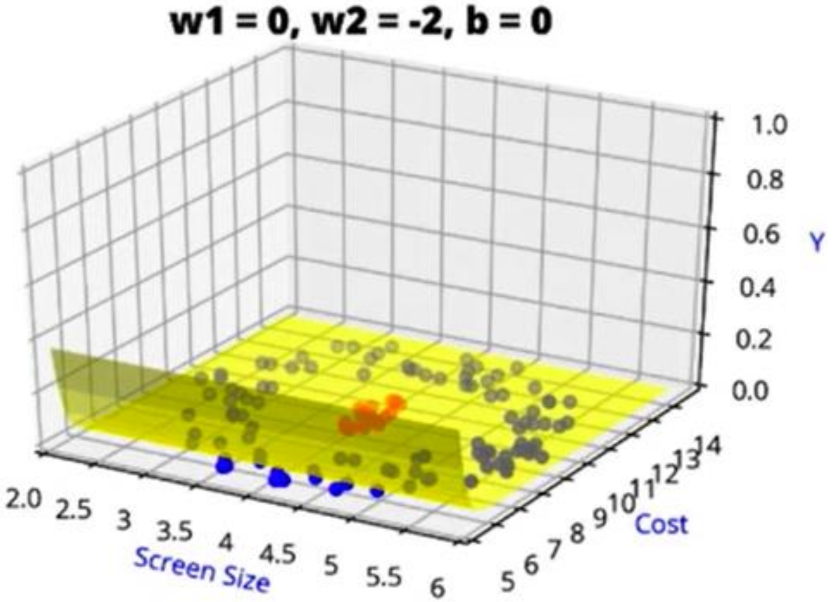




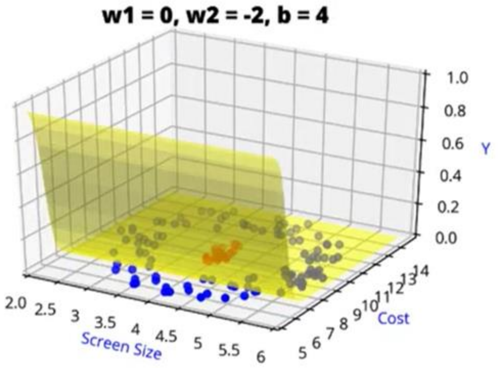


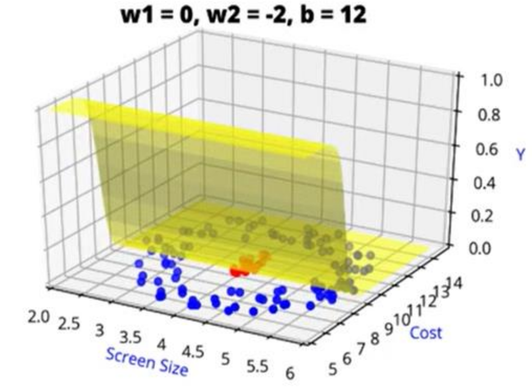
$$\hat{y} = \frac{1}{1+e^{-(w_1 * x_1 + w_2 * x_2 + b)}}$$

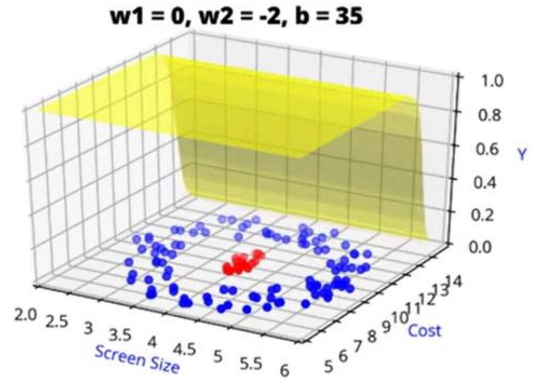




Can sigmoid function work for such data points?



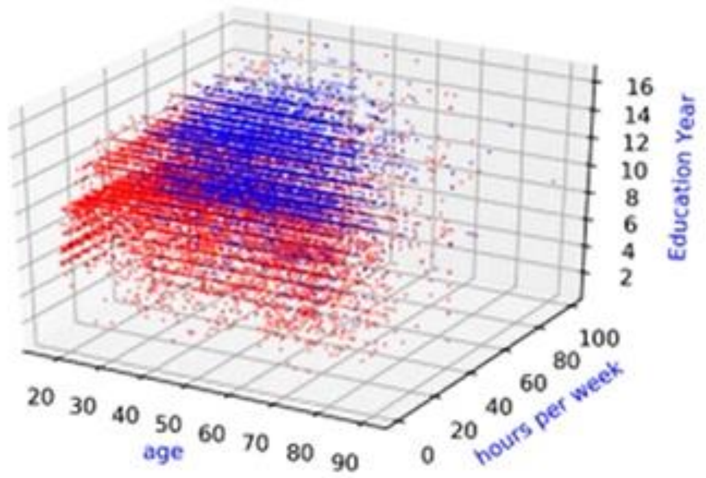




Adult Census Income*

Whether Annual Income of person $\geq$ 50k or $<$ 50k ?

Age	hour/week	Education year		Income
90	40	9	....	0
54	40	4		0
74	20	16		1
45	35	16		1
:	:	:	:	:



Are such complex functions seen in most real-world datasets?

$$\hat{y} = \hat{f}(x_1, x_2, ..., x_{14})$$

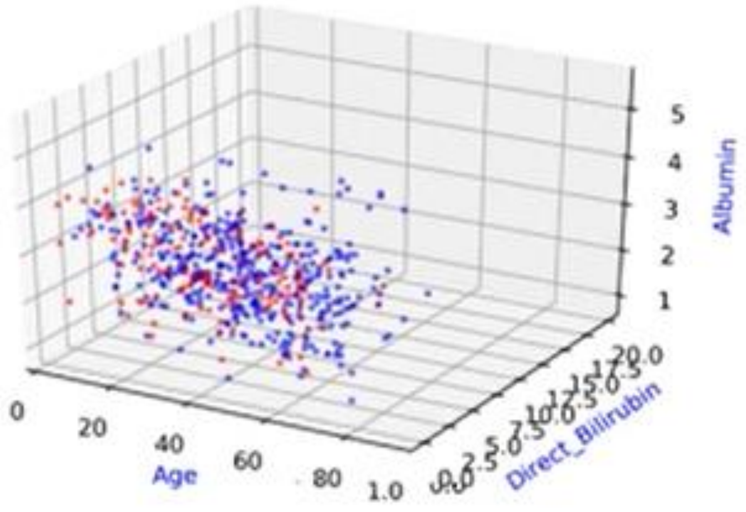
$$\hat{income} = \hat{f}(age, hour, ..., education)$$

Reference: https://www.kaggle.com/datasets/uciml/adult-census-income/

Indian Liver Patient Records*

whether person needs to be diagnosed or not ?

Age	Albumin	T_Bilirubin	....	D
65	3.3	0.7		0
62	3.2	10.9		0
20	4	1.1		1
84	3.2	0.7		1
⋮	⋮	⋮	⋮	⋮



Are such complex functions seen in most real-world datasets?

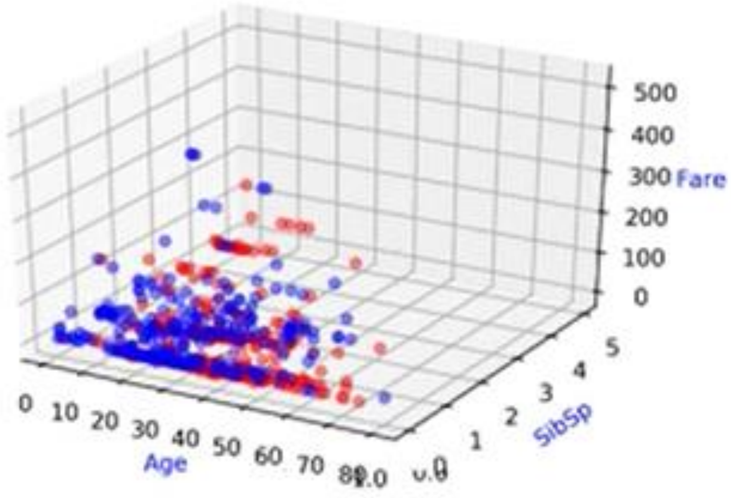
$$\hat{y} = \hat{f}(x_1, x_2, ..., x_{10})$$

Reference: https://www.kaggle.com/datasets/uciml/indian-liver-patient-records

Titanic: Machine Learning from Disaster*

Predict survival on the Titanic

Ticket class	# of siblings	Fare		Survived ?
93.85	83.81	20.1		0
-141.22	-81.79	-52.28	....	1
-65.2	-76.33	-76.23		0
142.4	137.03	93.65		1
⋮	⋮	⋮	⋮	⋮

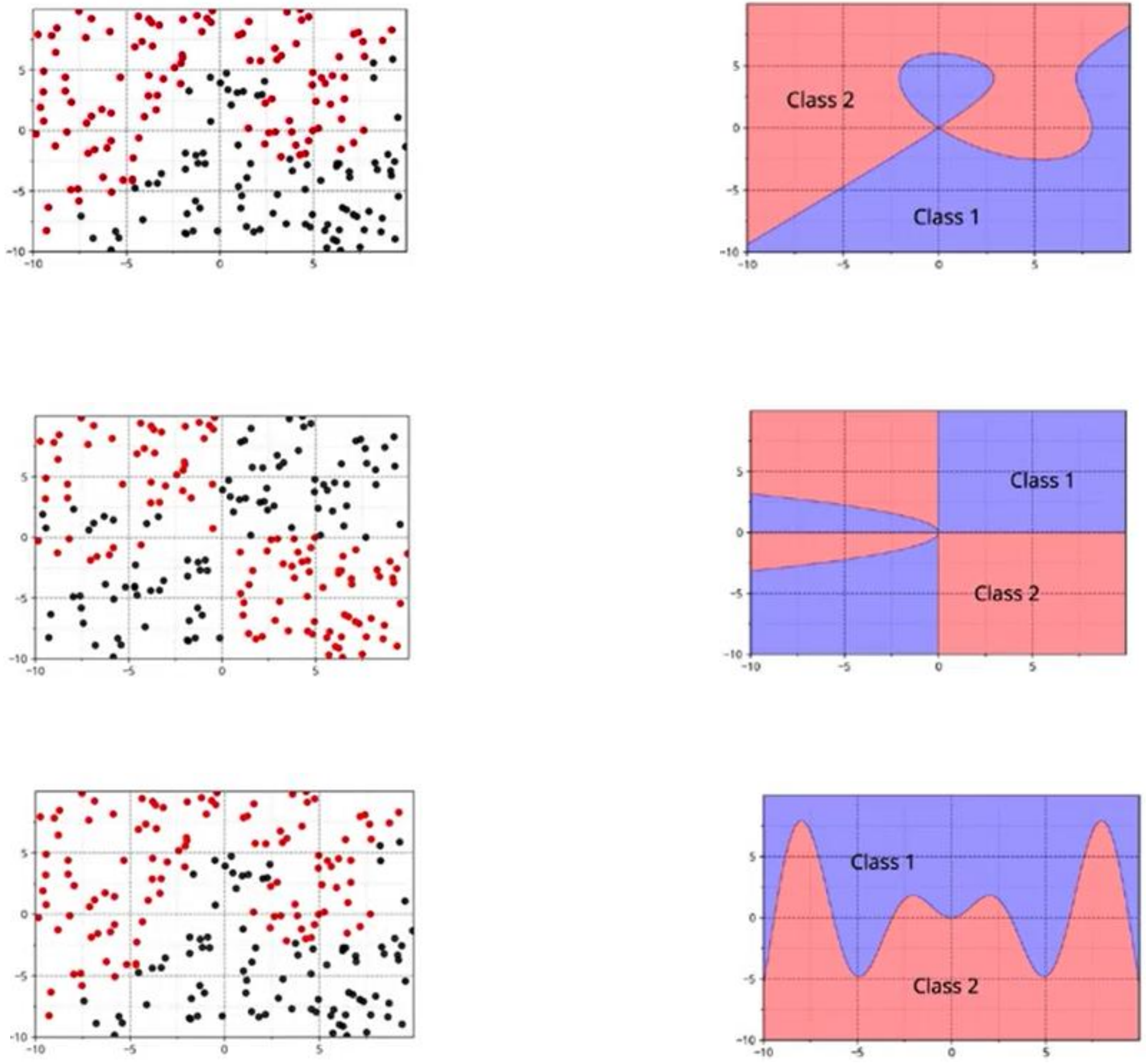


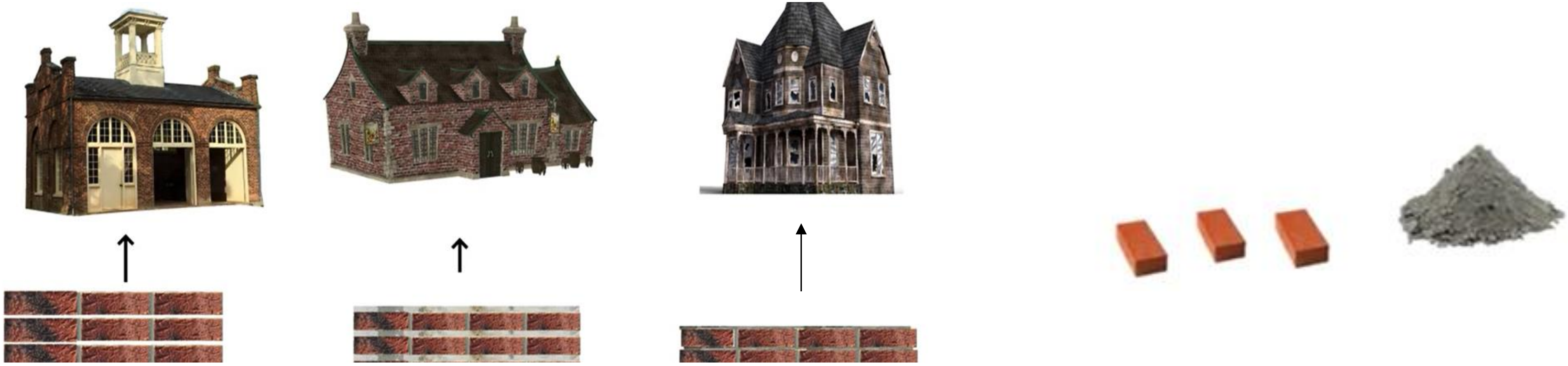
Are such complex functions seen in most real-world datasets?

$$\hat{y} = \hat{f}(x_1, x_2, ..., x_{10})$$

Reference: https://www.kaggle.com/c/titanic/data

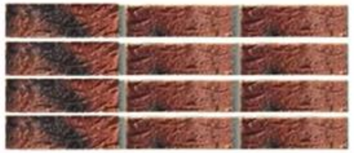
Complex Distributions



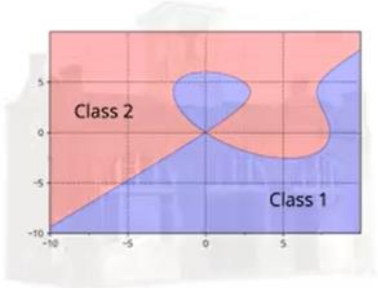


How do we come-up with such complex functions?
(Analogy of building complex functions)

Simple Concept for building complex Models

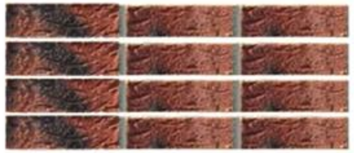


$\longrightarrow$

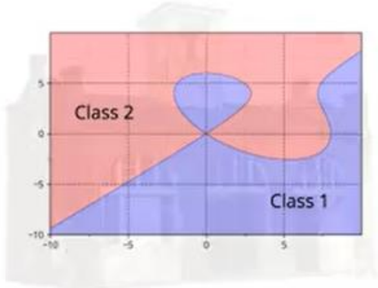


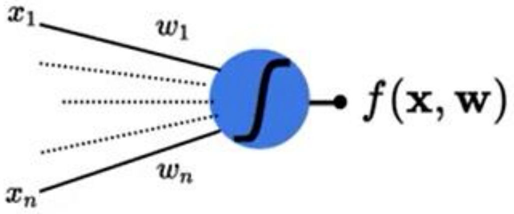
$$\boldsymbol{f}(\boldsymbol{x}_1, .., \boldsymbol{x}_n) = \frac{1}{1+e^{-(w_1 * x_1 + ... + w_n * x_n + b)}}$$

$$\boldsymbol{f}(\mathbf{x}, \mathbf{w}) = \frac{1}{1+e^{-(w * x + b)}}$$

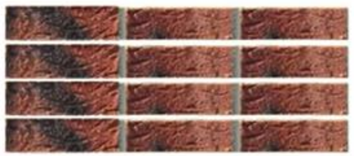


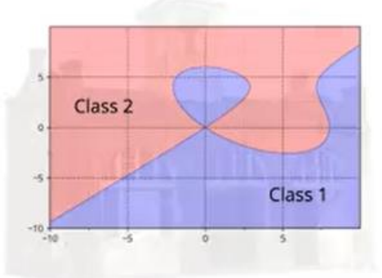
$\longrightarrow$





Simple Concept for building complex Models





$$f(x_1, .., x_n) = \frac{1}{1+e^{-(w_1 * x_1 + ... + w_n * x_n + b)}}$$

$$f(\mathbf{x}, \mathbf{w}) = \frac{1}{1+e^{-(w*x+b)}}$$