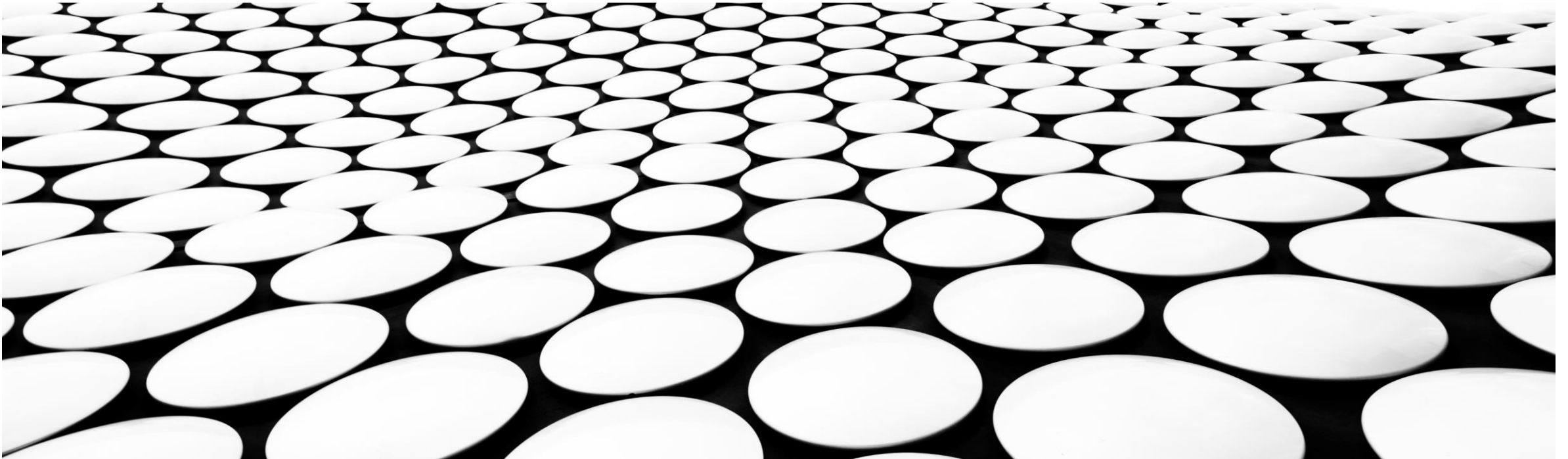


NEURAL NETWORKS

CLASSIFICATION AND REGRESSION





NEURAL NETWORKS

- Part 1 – Introduction
- Part 2 – Supervised Machine Learning
- Part 3 – Error minimisation
- Part 4 – Linear regression
- Part 5 – Logistic regression (Perceptron)
- Part 6 – Multinomial Perceptron
- Part 7 – XOR affair

Part 1

Introduction

Artificial Intelligence

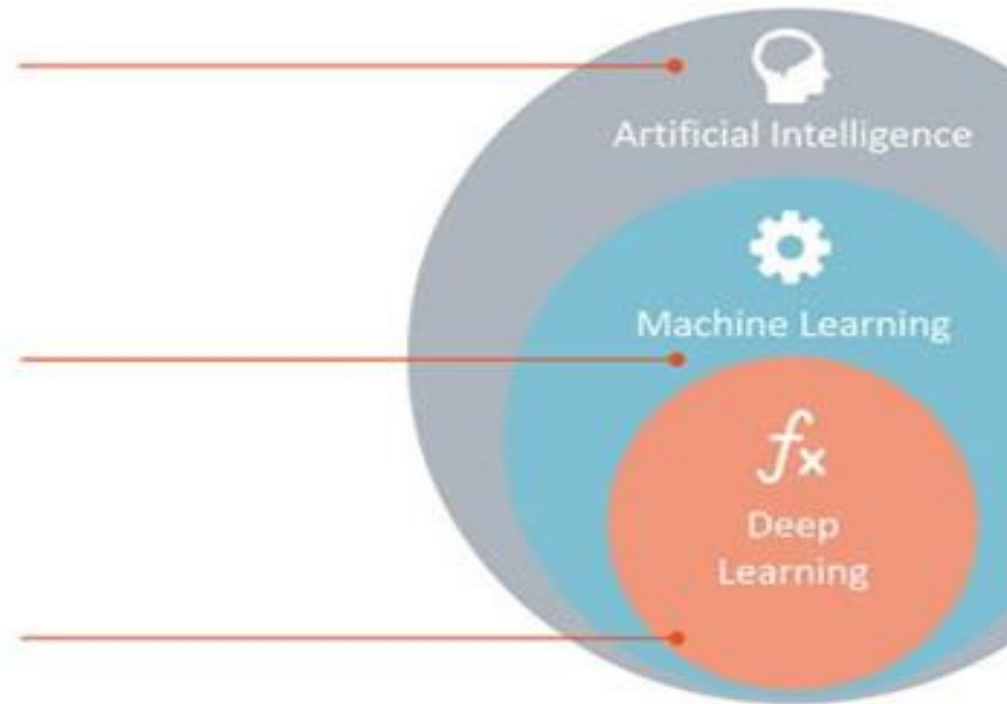
Any technique which enables computers to mimic human behavior.

Machine Learning

Subset of AI techniques which use statistical methods to enable machines to improve with experiences.

Deep Learning

Subset of ML which make the computation of multi-layer neural networks feasible.



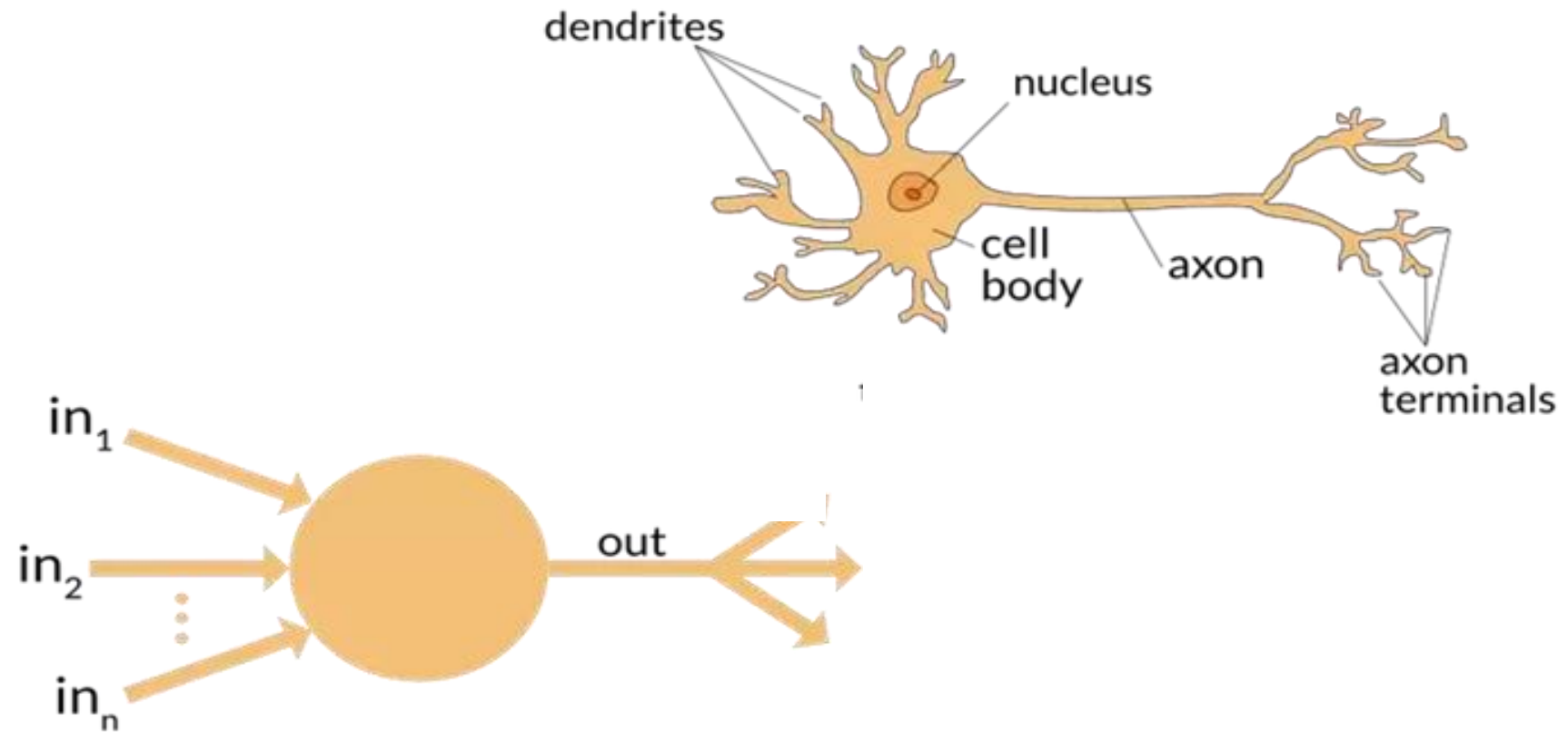
What Is A Neural Network?

The simplest definition of a neural network, more properly referred to as an 'artificial' neural network (ANN), is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as:

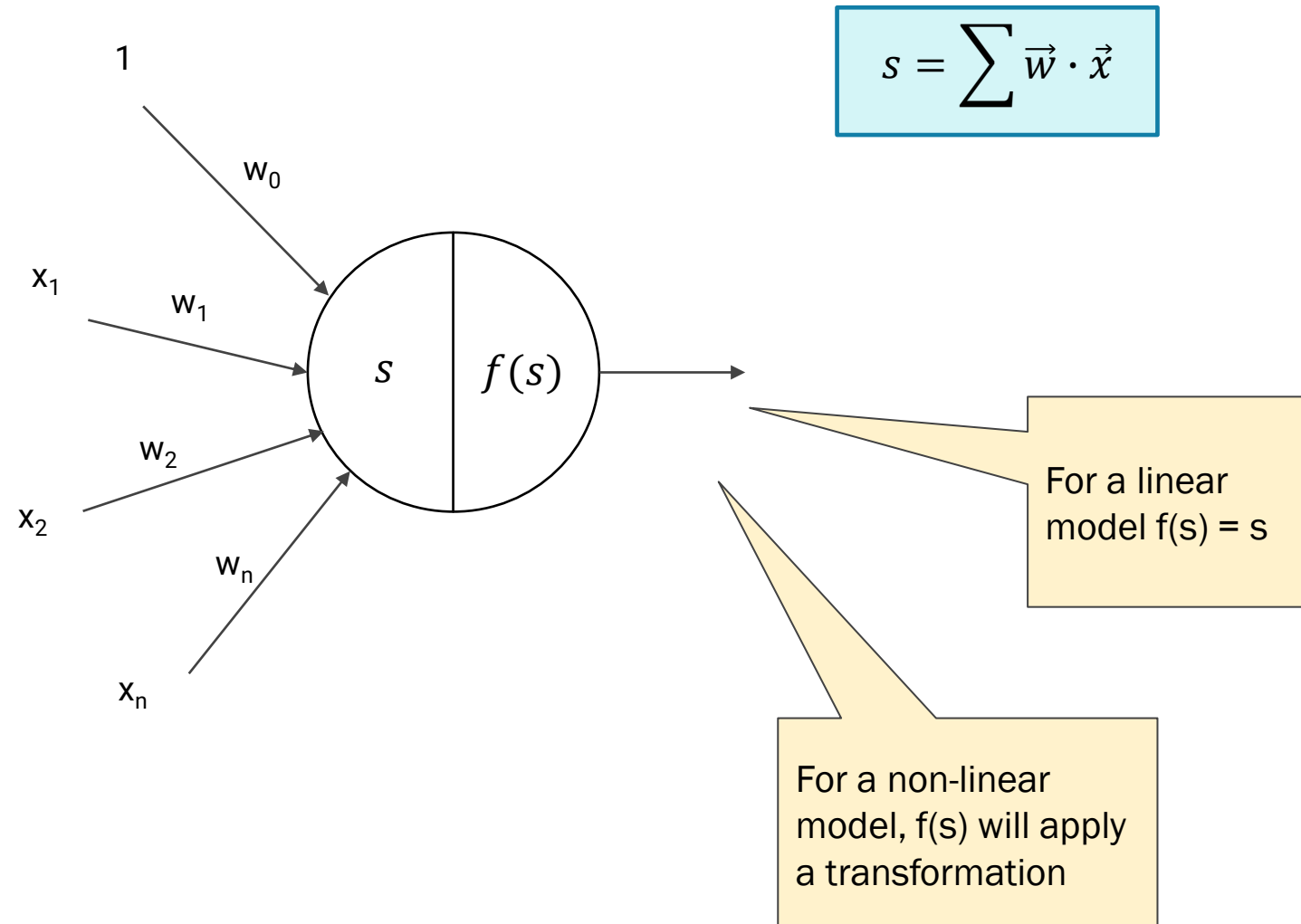
"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

In "Neural Network Primer: Part I" by Maureen Caudill, AI Expert, Feb. 1989

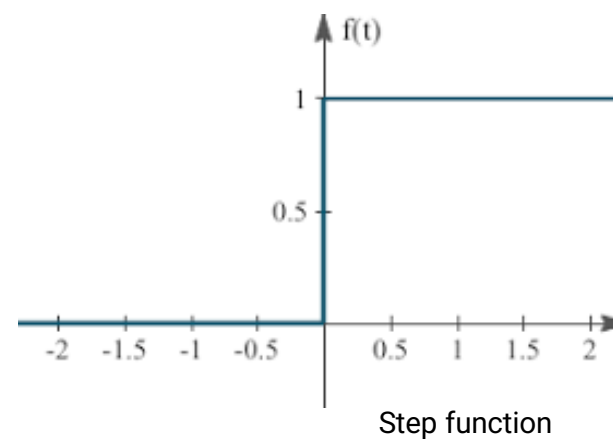
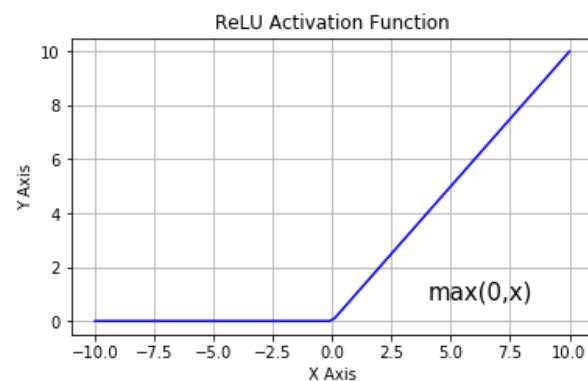
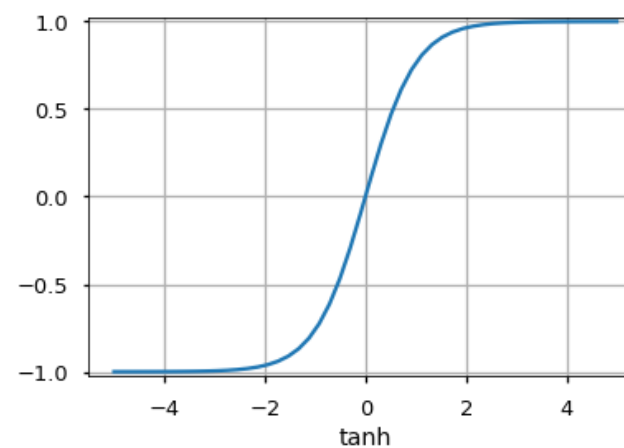
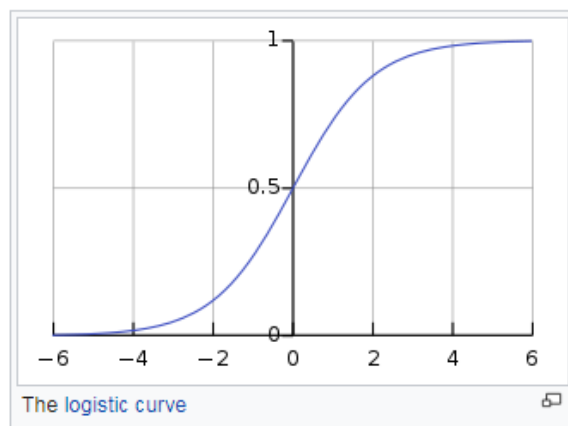
SIMPLE PROCESSING ELEMENT



DYNAMIC RESPONSE

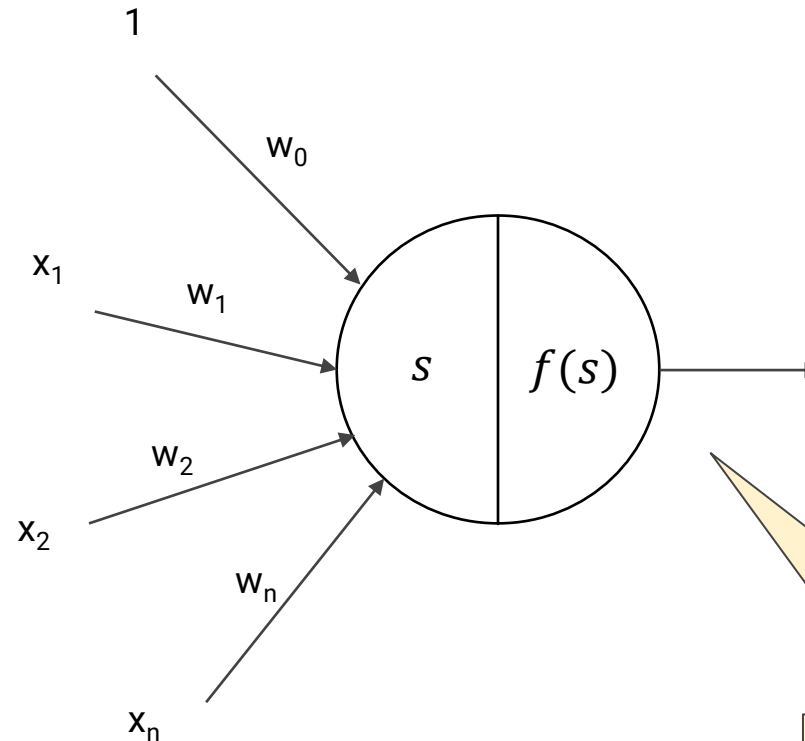


EXAMPLES OF NON-LINEAR TRANSFORMATIONS



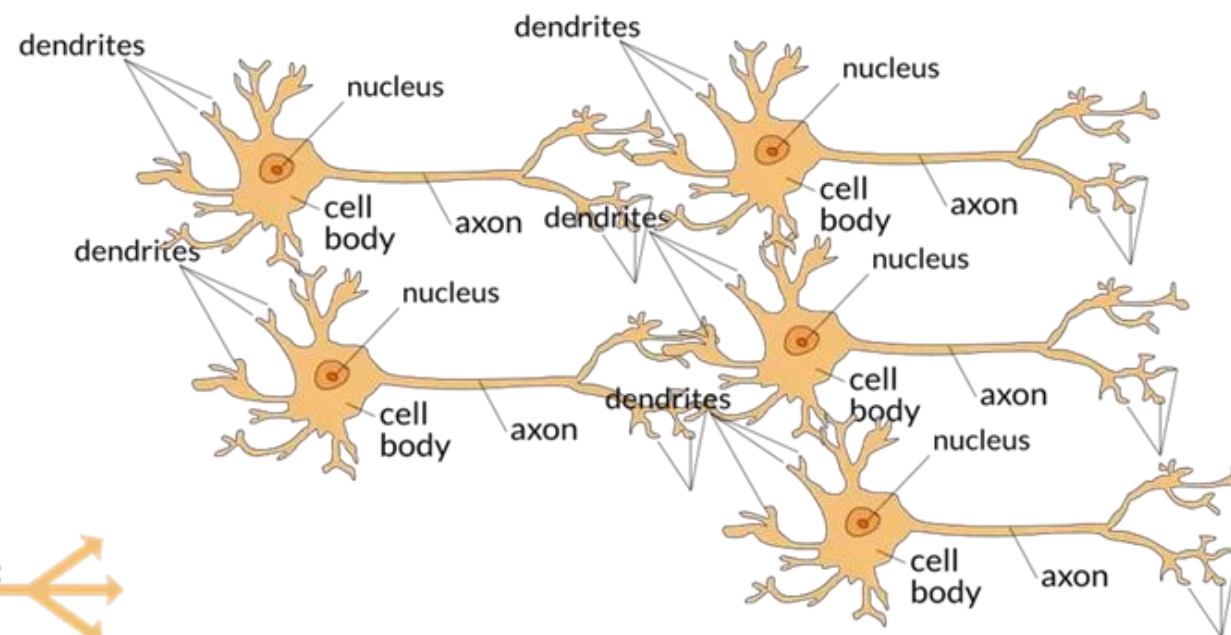
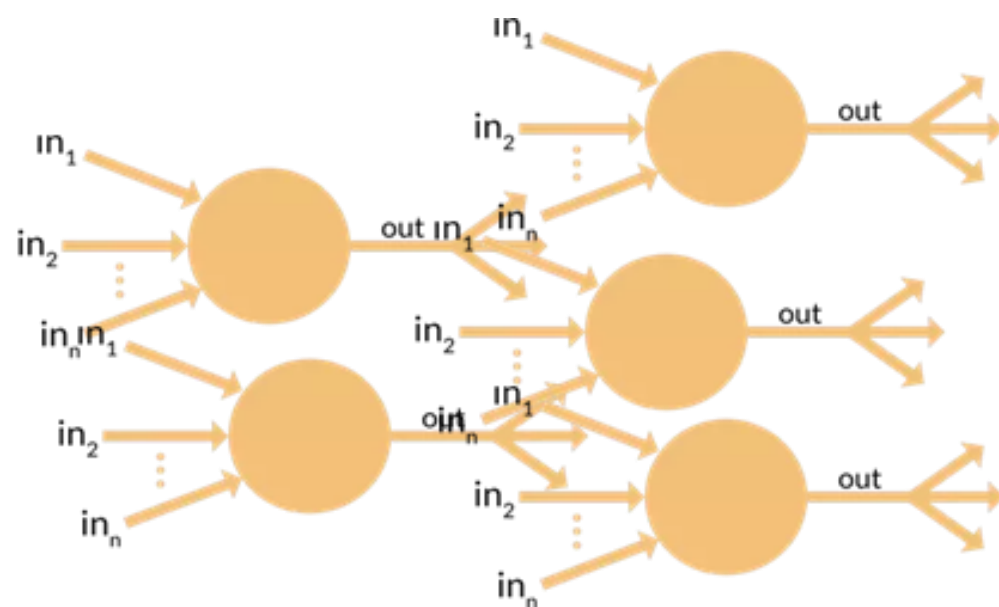
DYNAMIC RESPONSE

$$s = \sum \vec{w} \cdot \vec{x}$$



The DYNAMIC response is a response that changes as the weights are learned

HIGHLY INTERCONNECTED



DEEP LEARNING

Jeff Dean, lead at Google AI

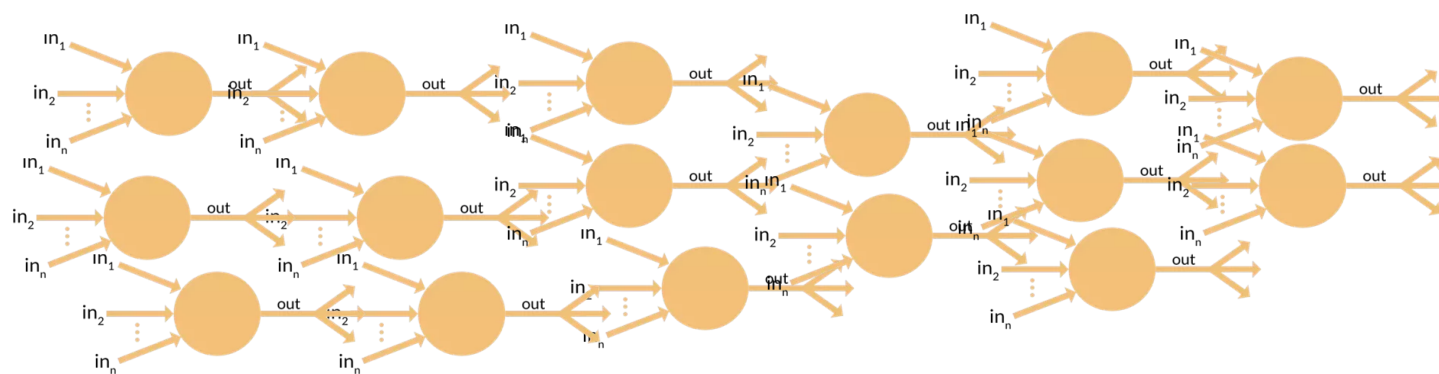
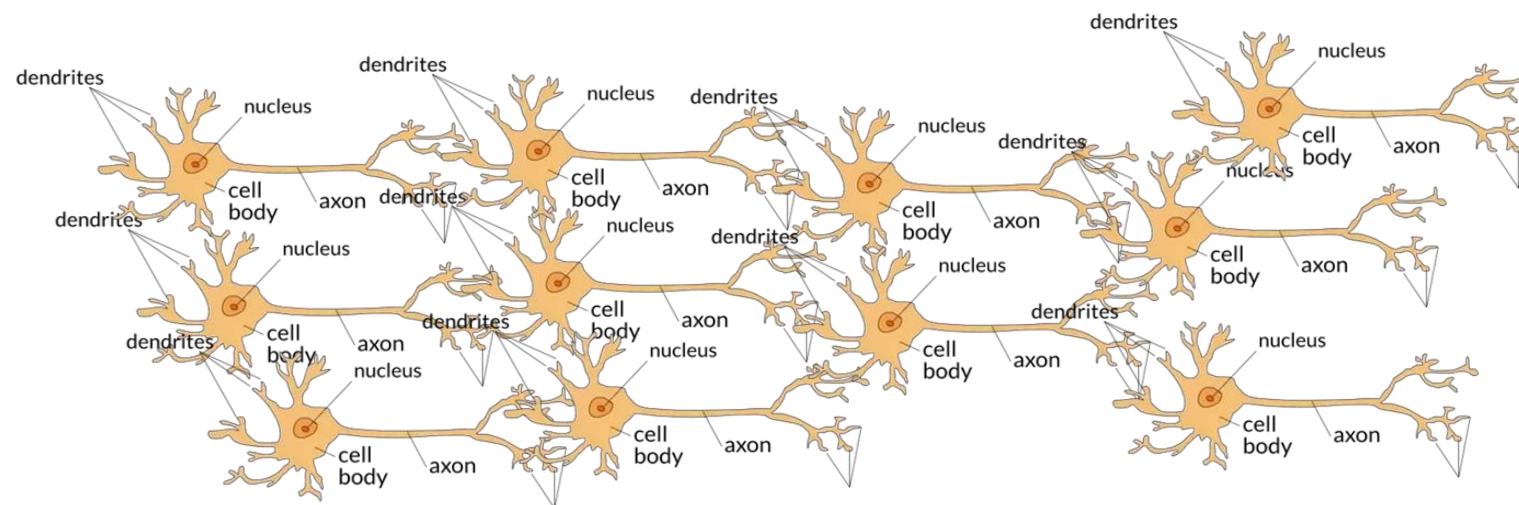


In a 2016 talk titled “Deep Learning for Building Intelligent Computer Systems” he made a comment in the similar vein, that deep learning is really all about large neural networks.



When you hear the term deep learning, just think of a large deep neural net. Deep refers to the number of layers typically and so this kind of the popular term that's been adopted in the press. I think of them as deep neural networks generally.

MULTIPLE LAYERS BETWEEN INPUT AND OUTPUT



HISTORICAL CONTEXT AND THE EFFERVESCENCE OF DEEP LEARNING

Explained: Neural networks

Ballyhooed artificial-intelligence technique known as “deep learning” revives 70-year-old idea.

Larry Hardesty | MIT News Office
April 14, 2017

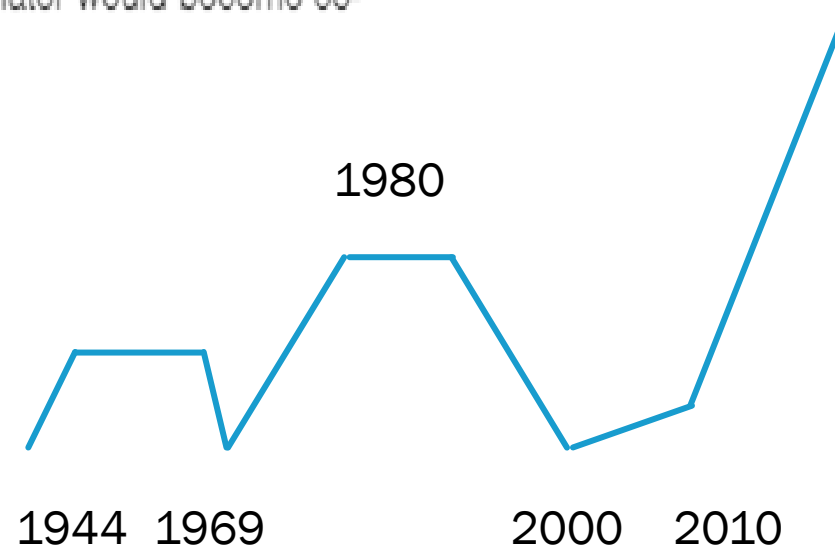
In the past 10 years, the best-performing artificial-intelligence systems — such as the speech recognizers on smartphones or Google's latest automatic translator — have resulted from a technique called “deep learning.”

Deep learning is in fact a new name for an approach to artificial intelligence called neural networks, which have been going in and out of fashion for more than 70 years. Neural networks were first proposed in 1944 by Warren McCulloch and Walter Pitts, two University of Chicago researchers who moved to MIT in 1952 as founding members of what's **sometimes called** the first cognitive science department.

Neural nets were a major area of research in both neuroscience and computer science until 1969, when, according to computer science lore, they were killed off by the MIT mathematicians Marvin Minsky and Seymour Papert, who a year later would become co-directors of the new MIT Artificial Intelligence Laboratory.

The technique then enjoyed a resurgence in the 1980s, fell into eclipse again in the first decade of the new century, and has returned like gangbusters in the second, fueled largely by the increased processing power of graphics chips.

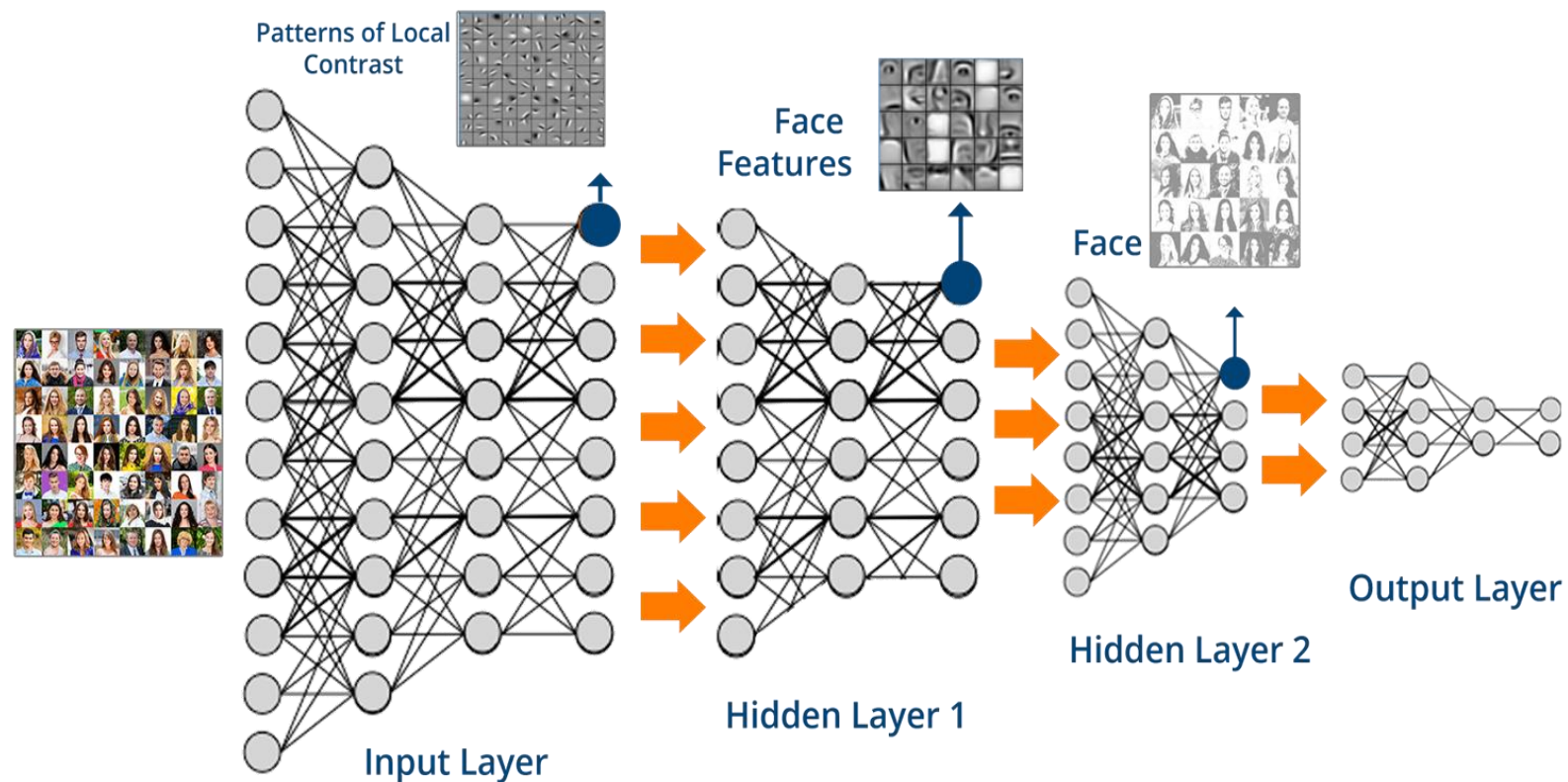
Neural nets were a major area of research in both neuroscience and computer science until 1969, when, according to computer science lore, they were killed off by the MIT mathematicians Marvin Minsky and Seymour Papert, who a year later would become co-directors of the new MIT Artificial Intelligence Laboratory.



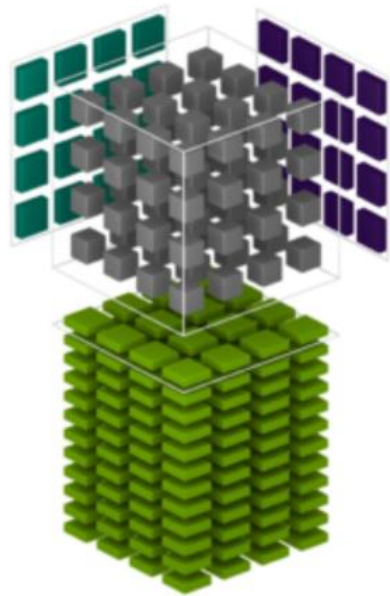
And LOTS of data!

The technique then enjoyed a resurgence in the 1980s, fell into eclipse again in the first decade of the new century, and has returned like gangbusters in the second, fueled largely by the increased processing power of graphics chips.

DEEP LEARNING FOR IMAGE PROCESSING



DEEP LEARNING FOR IMAGE PROCESSING

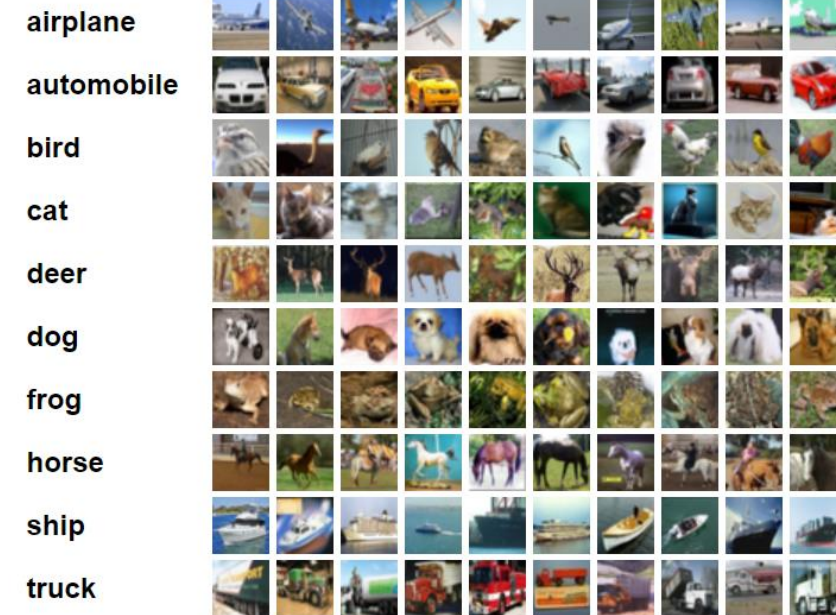


NVIDIA Tensor Cores

For AI researchers and application developers, NVIDIA Volta and Turing GPUs powered by **tensor cores** give you an immediate path to faster training and greater deep learning performance. With Tensor Cores enabled, FP32 and FP16 mixed precision matrix multiply dramatically accelerates your throughput and reduces AI training times.

DEEP LEARNING FOR IMAGE PROCESSING

Here are the classes in the dataset, as well as 10 random images from each:



The CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.



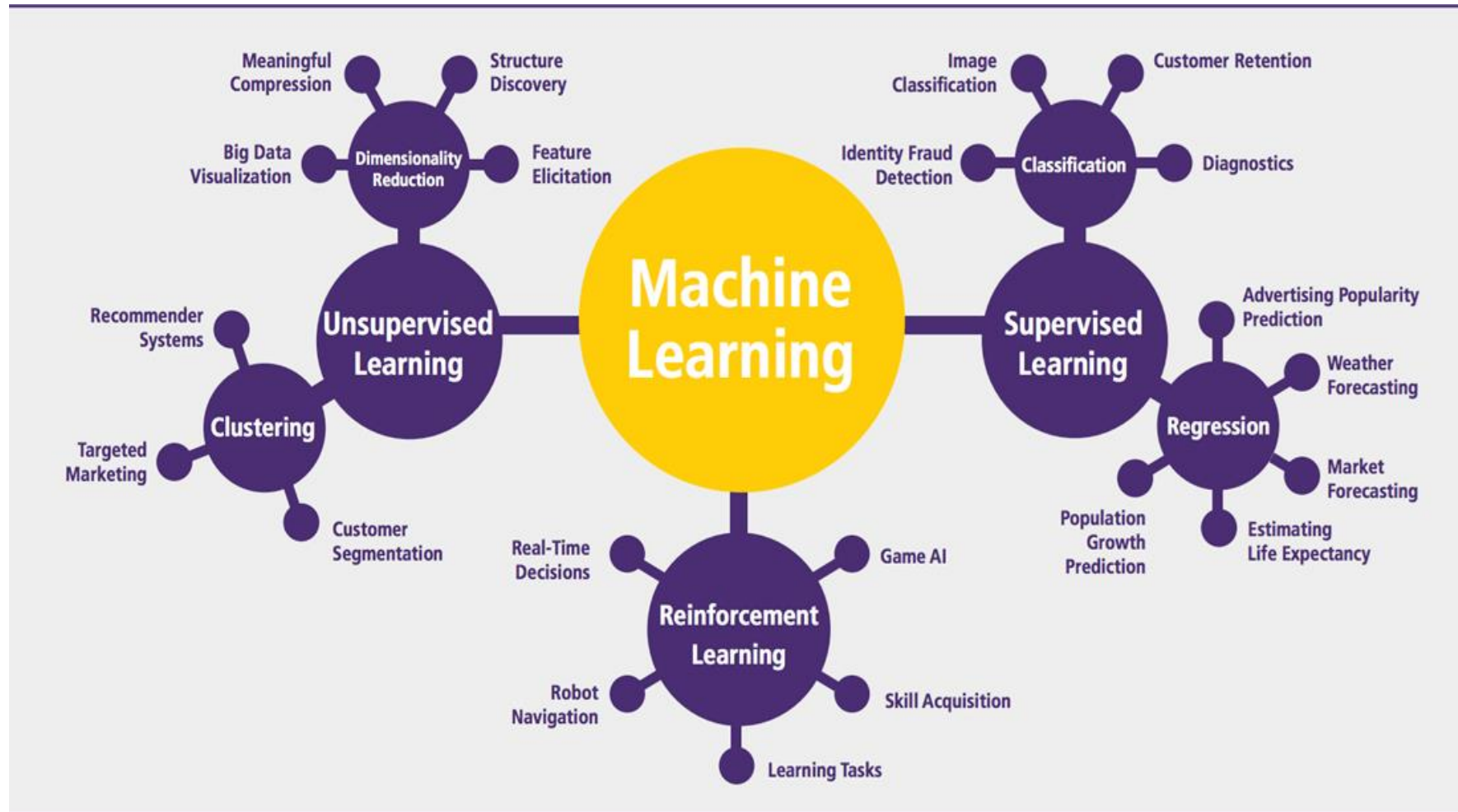
IN SUMMARY

- Definitions and underlying principles of neural networks
- Evolution toward deep learning

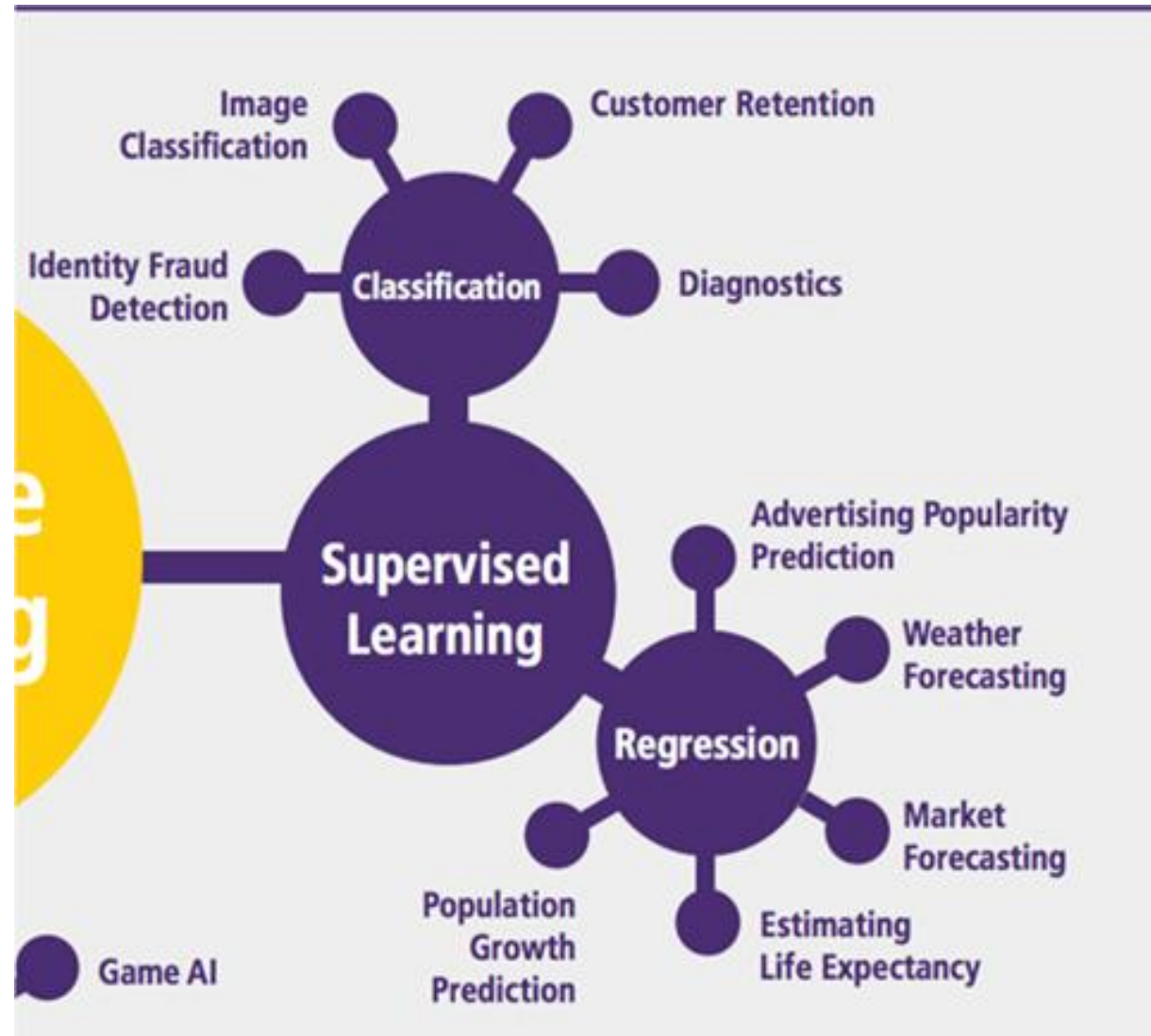
Part 2

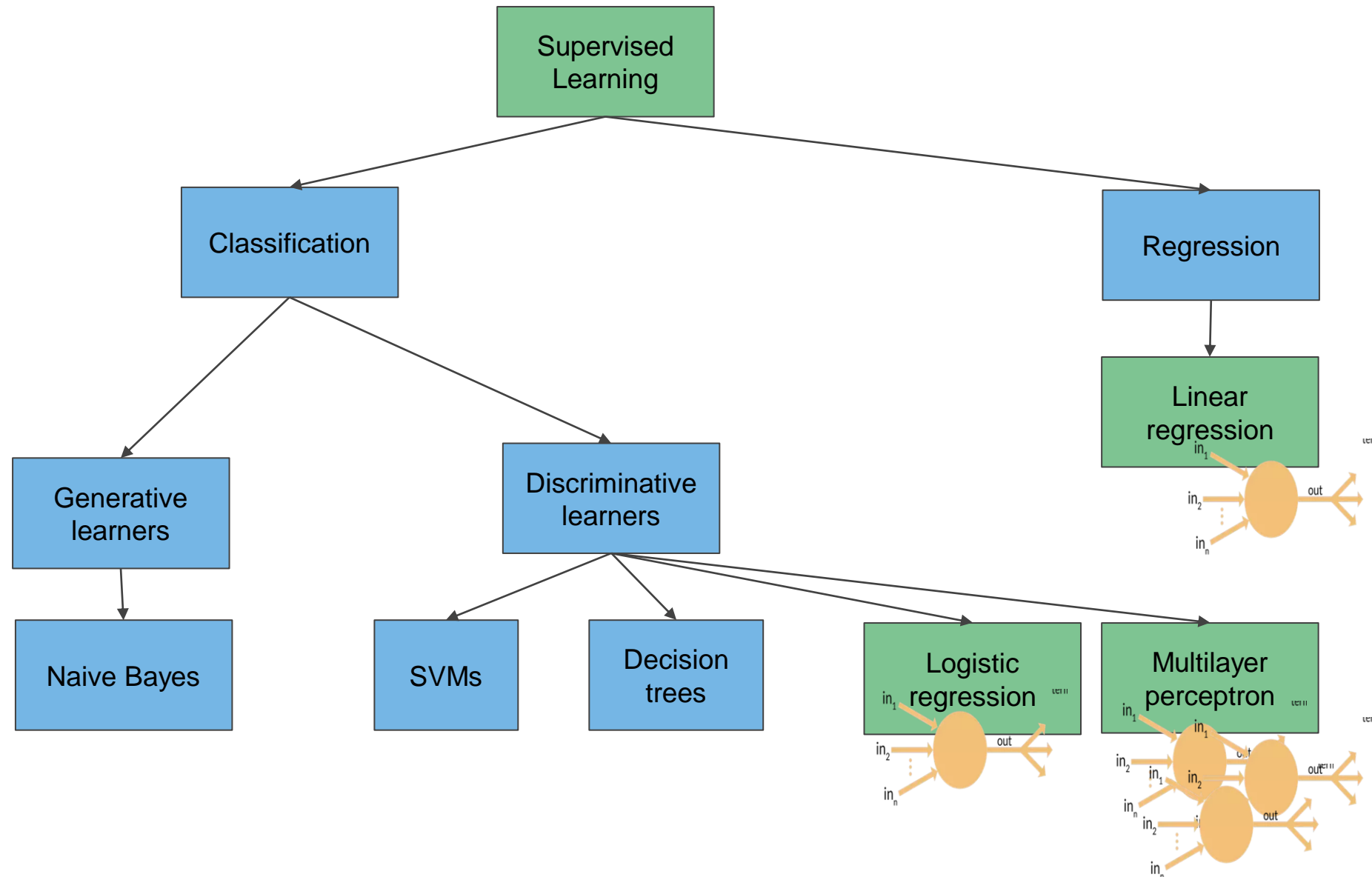
Supervised Machine Learning

WHERE ARE NN?

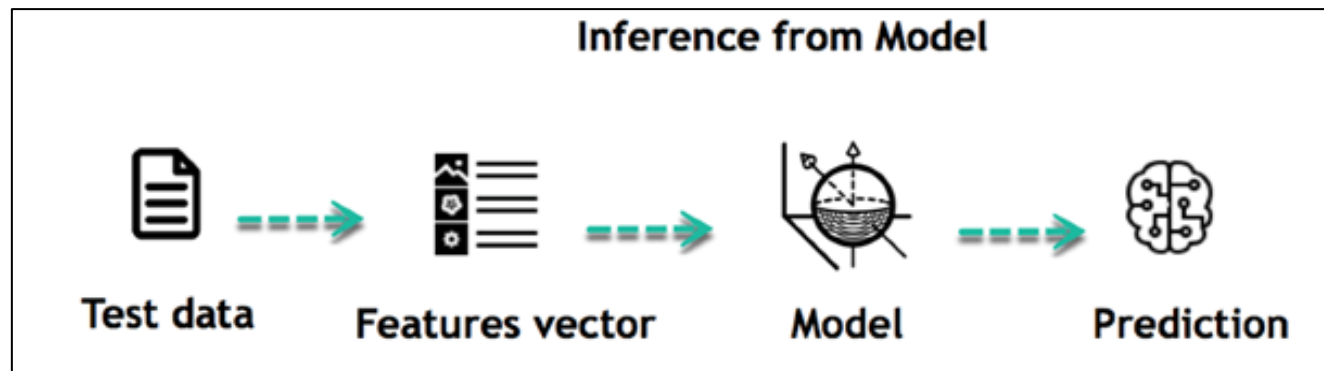
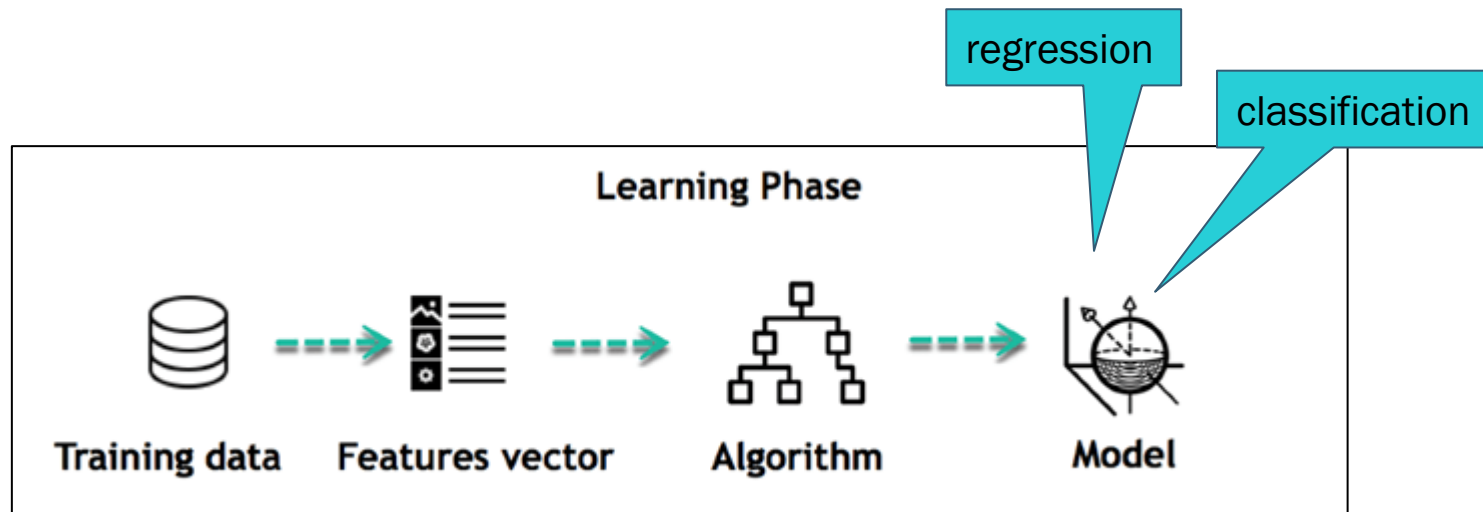


WHERE ARE NN?





MODELS



Sample	Temperature	Rain/Snow	Km biking
s1	-22	10	2
s2	-24	12	0
s3	-15	20	3
s4	-8	13	4
s5	-5	40	3
s6	8	45	8
s7	-2	5	10
s8	5	5	12
s9	12	2	12
s10	8	5	14
s11	12	20	20
s12	13	15	20
s13	15	10	25
s14	18	5	20
s15	19	20	40

REGRESSION EXAMPLE

Training data

15 samples, \mathbf{s}_i for $i = 1..15$

Feature vectors

$\vec{\mathcal{X}}$ has 2 dimensions

\mathcal{X}_1 Temperature

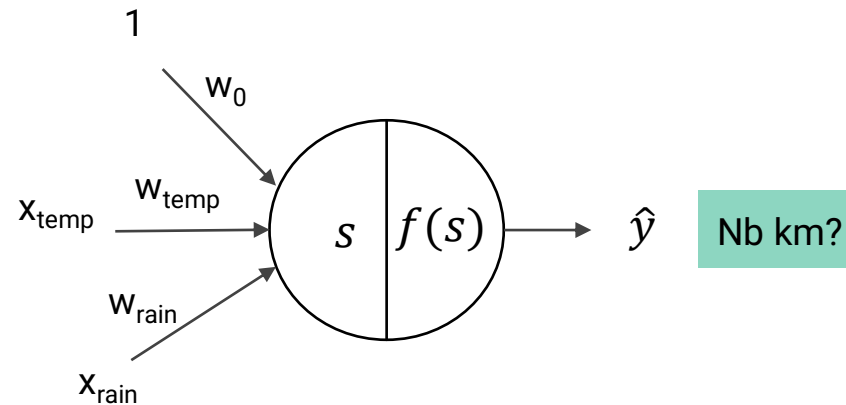
\mathcal{X}_2 Rain/snow

MODEL BUILDING

Sample	Temperature	Rain/Snow	Km biking
s1	-22	10	2
s2	-24	12	0
s3	-15	20	3
s4	-8	13	4
s5	-5	40	3
s6	8	45	8
s7	-2	5	10
s8	5	5	12
s9	12	2	12
s10	8	5	14
s11	12	20	20
s12	13	15	20
s13	15	10	25
s14	18	5	20
s15	19	20	40

$$s = \sum \vec{w} \cdot \vec{x}$$

$$\hat{y} = f(s)$$



$$\Rightarrow f(s) = s$$

Model:

$$(w_0, w_{\text{temp}}, w_{\text{rain}})$$

« FITTING » TRAINING DATA

Sample	Temperature	Rain/Snow	Km biking
s1	-22	10	2
s2	-24	12	0
s3	-15	20	3
s4	-8	13	4
s5	-5	40	3
s6	8	45	8
s7	-2	5	10
s8	5	5	12
s9	12	2	12
s10	8	5	14
s11	12	20	20
s12	13	15	20
s13	15	10	25
s14	18	5	20
s15	19	20	40

Model:

$$(w_0, w_{\text{temp}}, w_{\text{rain}})$$

The training data is used to learn the model.

$$S1: f(1 \cdot w_0 + -22 \cdot w_{\text{temp}} + 10 \cdot w_{\text{rain}}) = 2$$

$$S2: f(1 \cdot w_0 + -24 \cdot w_{\text{temp}} + 12 \cdot w_{\text{rain}}) = 0$$

$$S3: f(1 \cdot w_0 + -15 \cdot w_{\text{temp}} + 20 \cdot w_{\text{rain}}) = 3$$

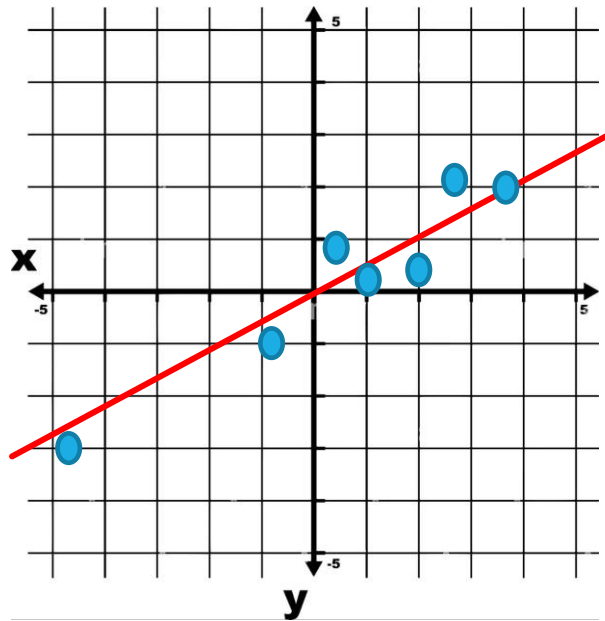
...

...

$$S14: f(1 \cdot w_0 + 18 \cdot w_{\text{temp}} + 5 \cdot w_{\text{rain}}) = 20$$

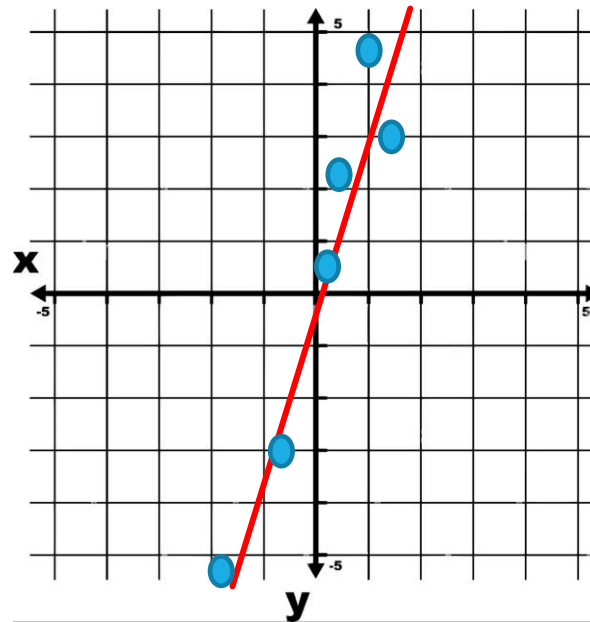
$$S15: f(1 \cdot w_0 + 19 \cdot w_{\text{temp}} + 20 \cdot w_{\text{rain}}) = 40$$

W0 ROLE



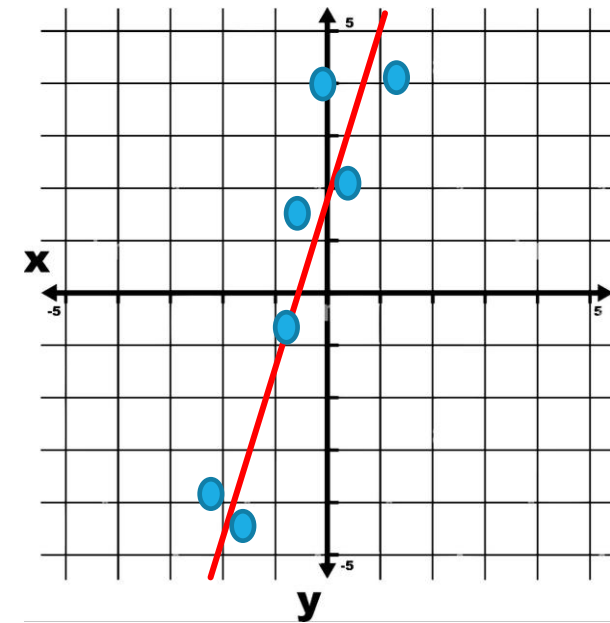
$$w_1 * x_1 = y$$

$$1/2 * x_1 = y$$



$$w_1 * x_1 = y$$

$$3 * x_1 = y$$



$$w_1 * x_1 + w_0 = y$$

$$3 * x_1 + 2 = y$$

Sample	Temperature	Rain/Snow	Class
s1	-22	10	Drive
s2	-24	12	Drive
s3	-15	20	Drive
s4	-8	13	Drive
s5	-5	40	Drive
s6	8	45	Drive
s7	-2	5	Bike
s8	5	5	Bike
s9	12	2	Bike
s10	8	5	Bike
s11	12	20	Bike
s12	13	15	Bike
s13	15	10	Bike
s14	18	5	Bike
s15	19	20	Bike

CLASSIFICATION EXAMPLE

Training data

15 samples, \mathbf{s}_i for $i = 1..15$

Feature vectors

$\vec{\mathcal{X}}$ has 2 dimensions

\mathcal{X}_1 Temperature

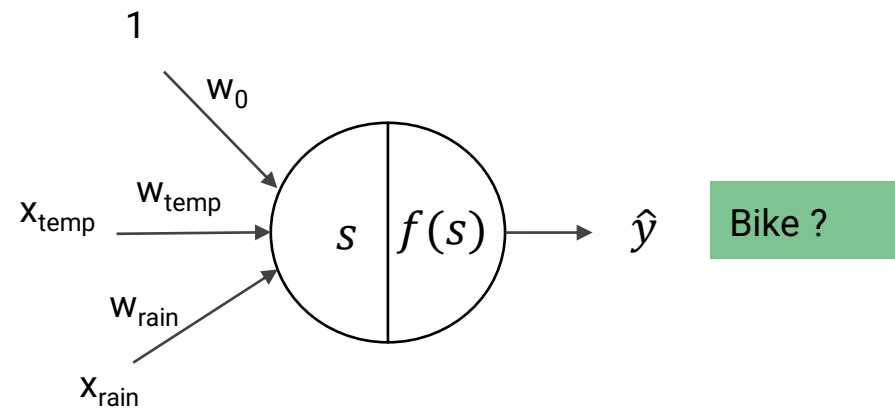
\mathcal{X}_2 Rain/snow

MODEL BUILDING

Sample	Temperature	Rain/Snow	Class
s1	-22	10	Drive
s2	-24	12	Drive
s3	-15	20	Drive
s4	-8	13	Drive
s5	-5	40	Drive
s6	8	45	Drive
s7	-2	5	Bike
s8	5	5	Bike
s9	12	2	Bike
s10	8	5	Bike
s11	12	20	Bike
s12	13	15	Bike
s13	15	10	Bike
s14	18	5	Bike
s15	19	20	Bike

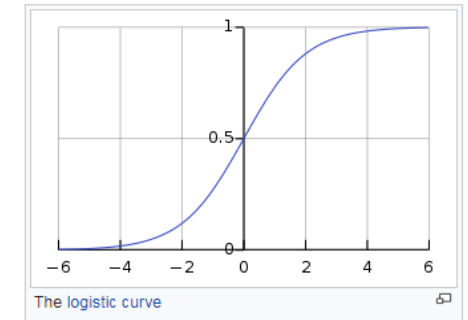
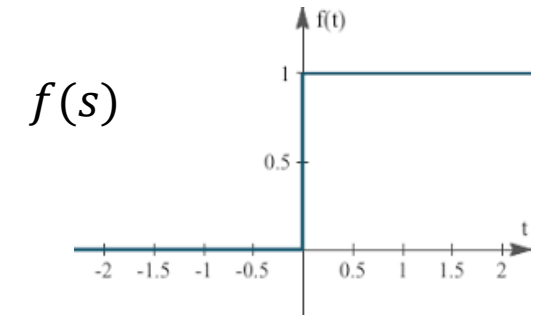
$$s = \sum \vec{w} \cdot \vec{x}$$

$$\hat{y} = f(s)$$

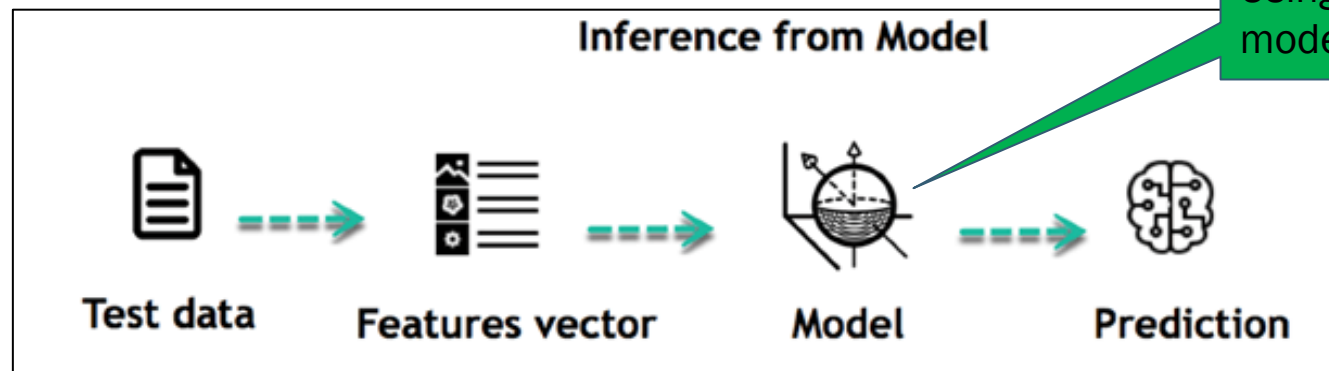
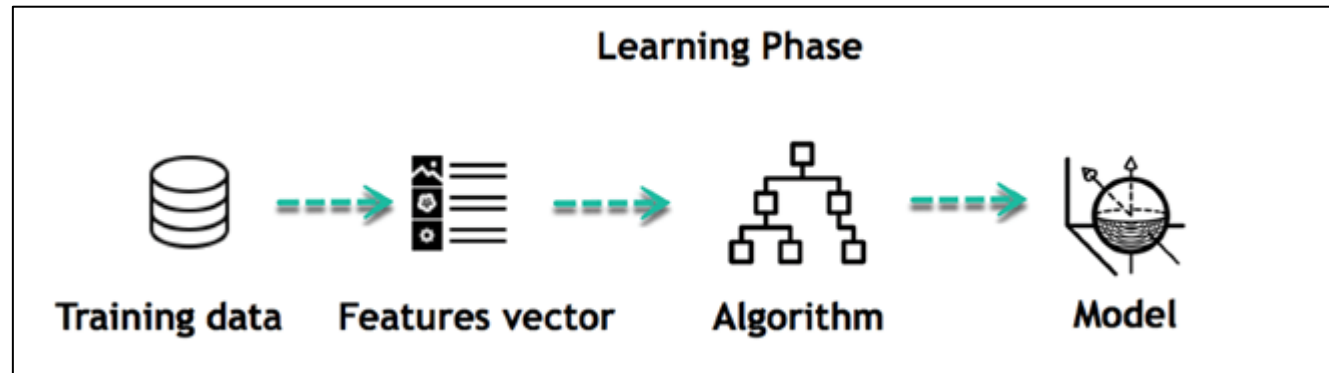


Model:

$(w_0, w_{\text{temp}}, w_{\text{rain}})$



ALL THAT WAS MENTIONED FOR SML STILL HOLDS

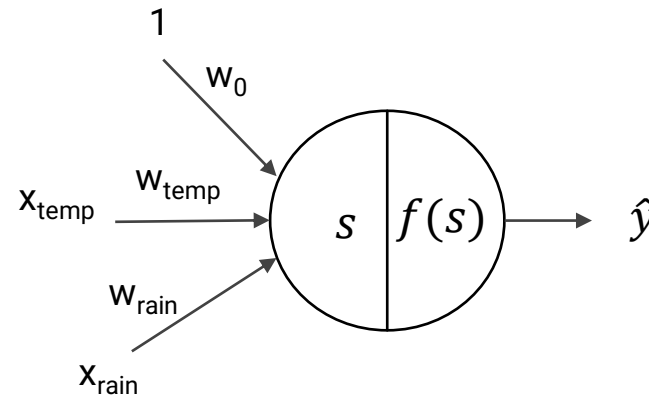


MODEL PREDICTION

$$s = \sum \vec{w} \cdot \vec{x}$$

$$\hat{y} = f(s)$$

Sample	Temperature	Rain/snow	
s16	-15	2	??
s17	+10	20	??



Prediction:

$$\text{S16 : } f(1 \cdot w_0 + -15 \cdot w_{\text{temp}} + 2 \cdot w_{\text{rain}}) = \hat{y}$$

$$\text{S17 : } f(1 \cdot w_0 + 10 \cdot w_{\text{temp}} + 20 \cdot w_{\text{rain}}) = \hat{y}$$



IN SUMMARY

- NN as Supervised Learners
 - Possible tasks: Regression and Classification
 - Model building: learning weights
 - Prediction

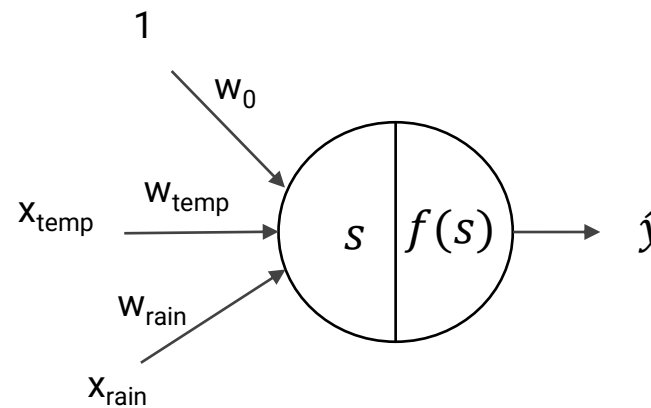
Part 3

Error Minimization

Sample	Temperature	Rain/Snow	Km Biking (y)
s1	-22	10	2
s2	-24	12	0
s3	-15	20	3
s4	-8	13	4
s5	-5	40	3
s6	8	45	8
s7	-2	5	10
s8	5	5	12
s9	12	2	12
s10	8	5	14
s11	12	20	20
s12	13	15	20
s13	15	10	25
s14	18	5	20
s15	19	20	40

$$s = \sum \vec{w} \cdot \vec{x}$$

$$\hat{y} = f(s)$$



Model:

$(w_0, w_{\text{temp}}, w_{\text{rain}})$

How to learn these weights?

Sample	Temperature	Rain/Snow	Km Biking (y)
s1	-22	10	2
s2	-24	12	0
s3	-15	20	3
s4	-8	13	4
s5	-5	40	3
s6	8	45	8
s7	-2	5	10
s8	5	5	12
s9	12	2	12
s10	8	5	14
s11	12	20	20
s12	13	15	20
s13	15	10	25
s14	18	5	20
s15	19	20	40

Modèle:

$$(w_0, w_{\text{temp}}, w_{\text{pluie}})$$

We need the training set to learn the weights (model).

$$S1: f(1 \cdot w_0 + -22 \cdot w_{\text{temp}} + 10 \cdot w_{\text{pluie}}) = 2$$

$$S2: f(1 \cdot w_0 + -24 \cdot w_{\text{temp}} + 12 \cdot w_{\text{pluie}}) = 0$$

$$S3: f(1 \cdot w_0 + -15 \cdot w_{\text{temp}} + 20 \cdot w_{\text{pluie}}) = 3$$

...

...

$$S14: f(1 \cdot w_0 + 18 \cdot w_{\text{temp}} + 5 \cdot w_{\text{pluie}}) = 20$$

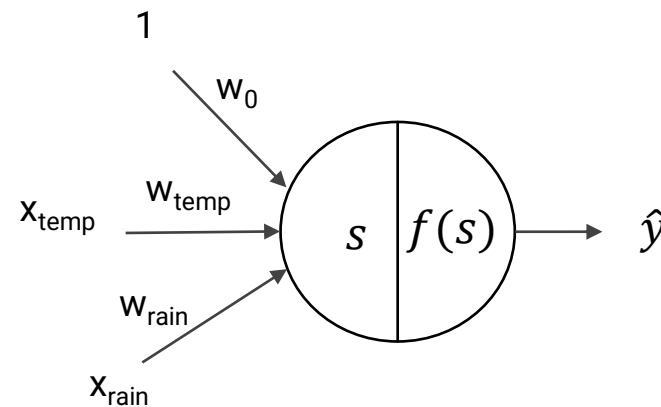
$$S15: f(1 \cdot w_0 + 19 \cdot w_{\text{temp}} + 20 \cdot w_{\text{pluie}}) = 40$$

Ideally, the model would make all these equations true

Sample	Temperature	Rain/Snow	Km Biking (y)	Km Biking (\hat{y})
s1	-22	10	2	
s2	-24	12	0	
s3	-15	20	3	
s4	-8	13	4	
s5	-5	40	3	
s6	8	45	8	
s7	-2	5	10	
s8	5	5	12	
s9	12	2	12	
s10	8	5	14	
s11	12	20	20	
s12	13	15	20	
s13	15	10	25	
s14	18	5	20	
s15	19	20	40	

$$s = \sum \vec{w} \cdot \vec{x}$$

$$\hat{y} = f(s)$$



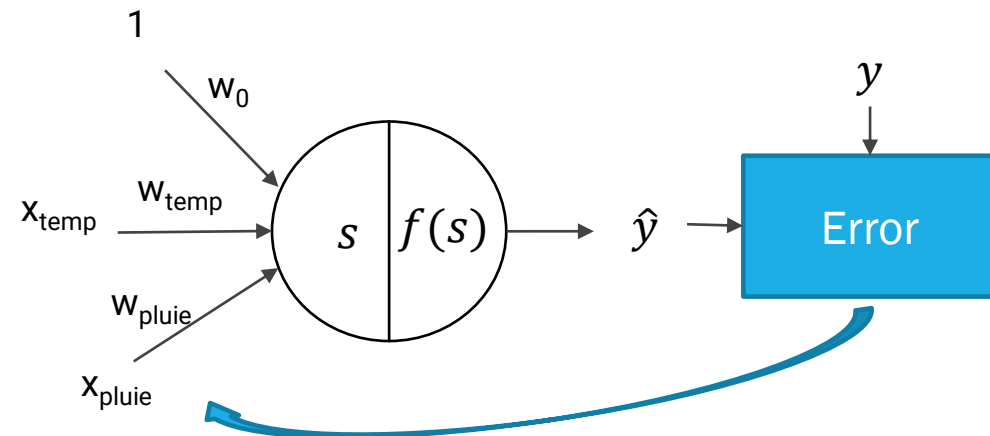
Model:

$(w_0, w_{\text{temp}}, w_{\text{rain}})$

Iterative approximation

Exemple	Température	Pluie/neige	Km Vélo (y)	Km Vélo (\hat{y})
s1	-22	10	2	
s2	-24	12	0	
s3	-15	20	3	
s4	-8	13	4	
s5	-5	40	3	
s6	8	45	8	
s7	-2	5	10	
s8	5	5	12	
s9	12	2	12	
s10	8	5	14	
s11	12	20	20	
s12	13	15	20	
s13	15	10	25	
s14	18	5	20	
s15	19	20	40	

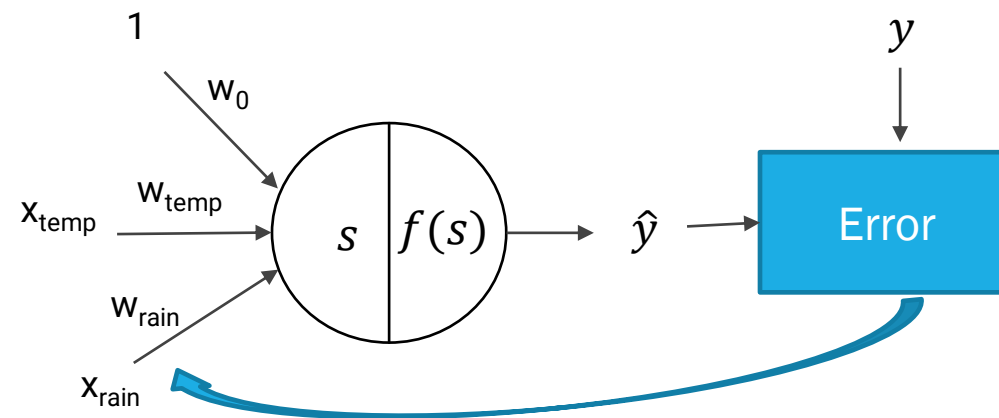
The model learned should **MINIMIZE** the error on the overall training set.



Modèle:

$(w_0, w_{\text{temp}}, w_{\text{pluie}})$

Sample	Temperature	Rain/Snow	Km Biking (y)	Km Biking (\hat{y})
s1	-22	10	2	
s2	-24	12	0	
s3	-15	20	3	
s4	-8	13	4	
s5	-5	40	3	
s6	8	45	8	
s7	-2	5	10	
s8	5	5	12	
s9	12	2	12	
s10	8	5	14	
s11	12	20	20	
s12	13	15	20	
s13	15	10	25	
s14	18	5	20	
s15	19	20	40	



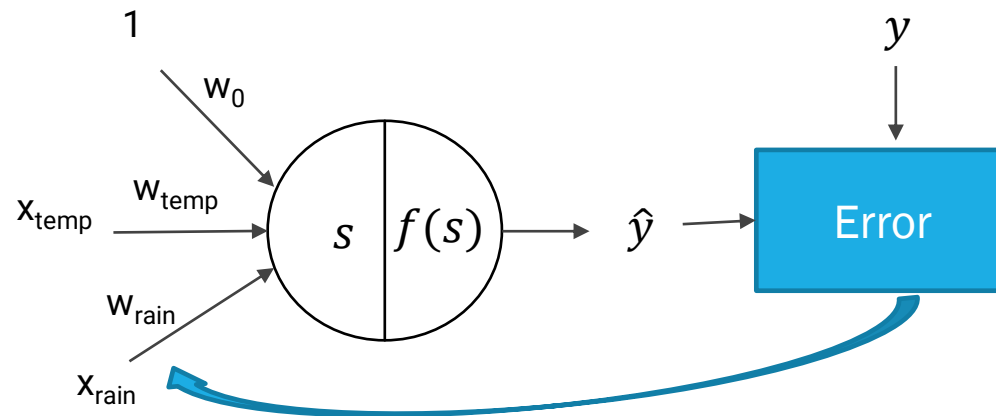
Model:

$(w_0, w_{\text{temp}}, w_{\text{rain}})$

L1 ERROR

$$L_1(E) = \sum_{e \in E} |y(e) - \hat{y}(e)|$$

Sample	Temperature	Rain/Snow	Km Biking (y)	Km Biking (ŷ)
s1	-22	10	2	
s2	-24	12	0	
s3	-15	20	3	
s4	-8	13	4	
s5	-5	40	3	
s6	8	45	8	
s7	-2	5	10	
s8	5	5	12	
s9	12	2	12	
s10	8	5	14	
s11	12	20	20	
s12	13	15	20	
s13	15	10	25	
s14	18	5	20	
s15	19	20	40	



Model:

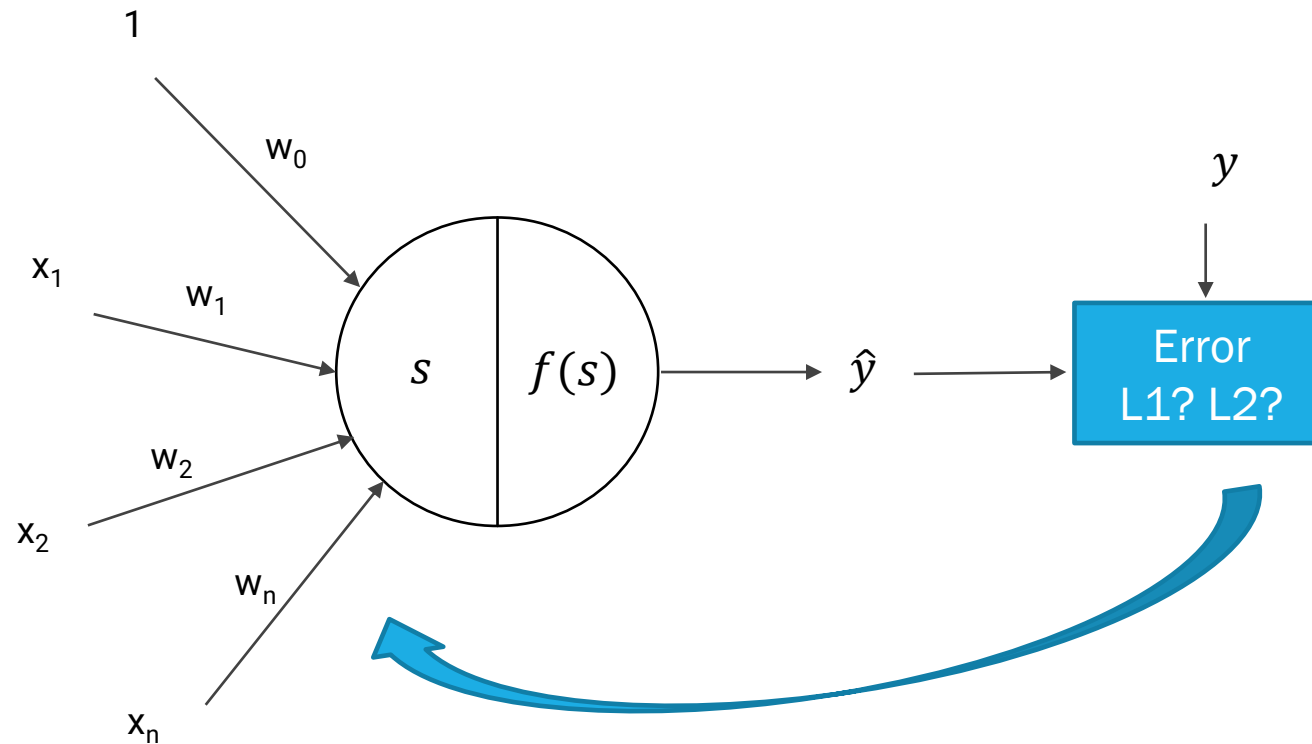
$(w_0, w_{\text{temp}}, w_{\text{rain}})$

L2 ERROR

$$L_2(E) = \frac{1}{2} \sum_{e \in E} (y(e) - \hat{y}(e))^2$$

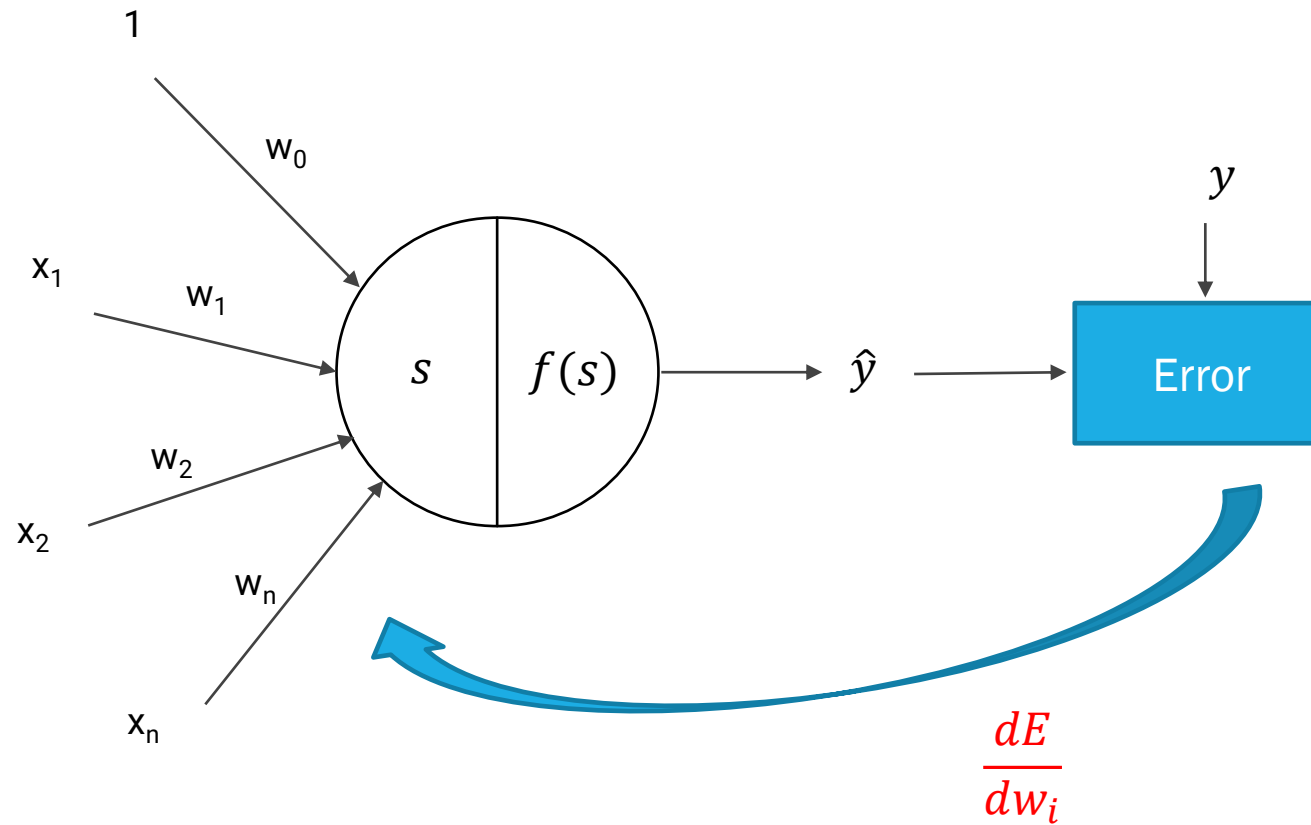
HOW TO MINIMIZE THE ERROR?

We will use optimization
techniques --- SEARCH



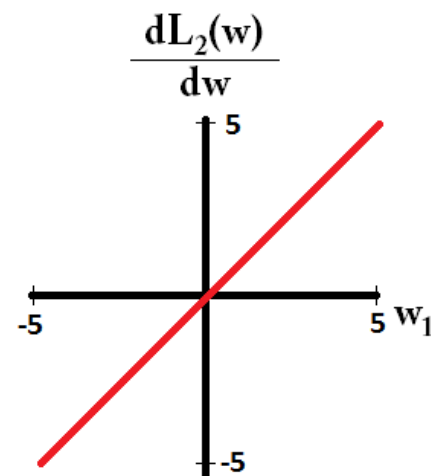
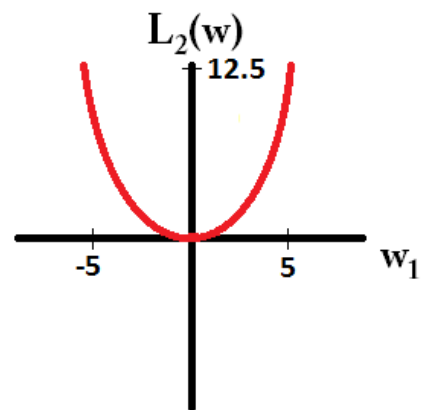
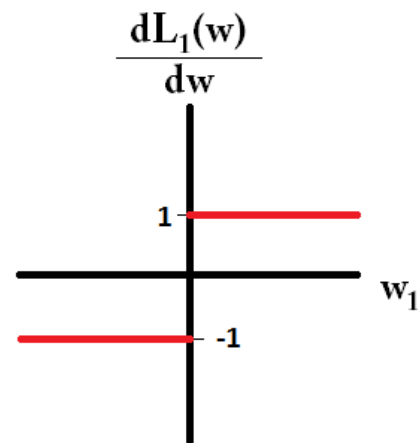
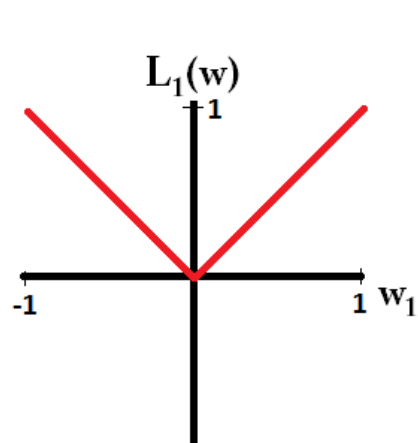
Greedy Strategy : gradient descent

Let's evaluate the contribution of a weight on the error, and adjust that weight (up or down) to reduce the error.



Each weight contributes to the error

L1 VERSUS L2



What does the derivative tell us about the weight?

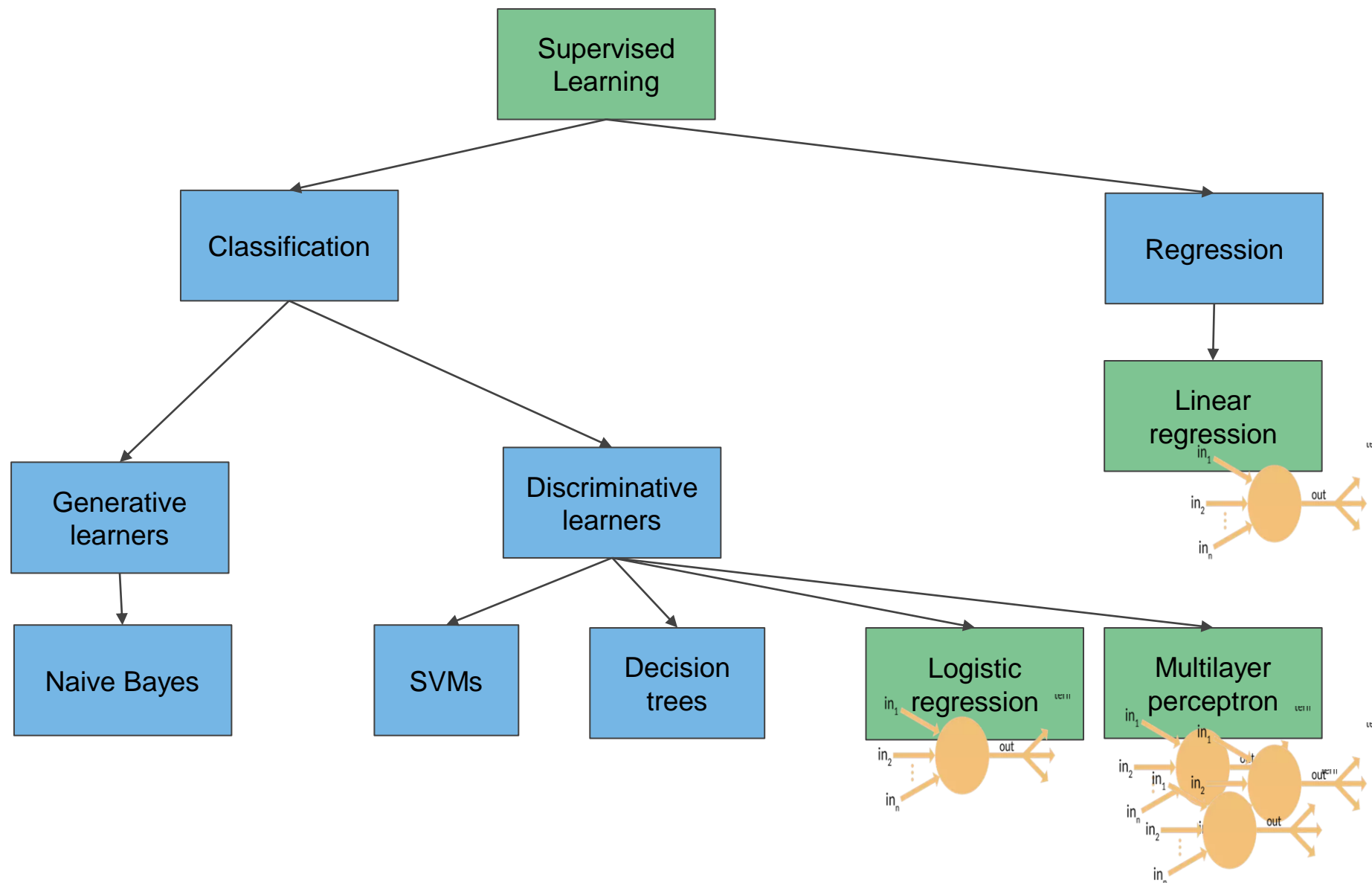


IN SUMMARY

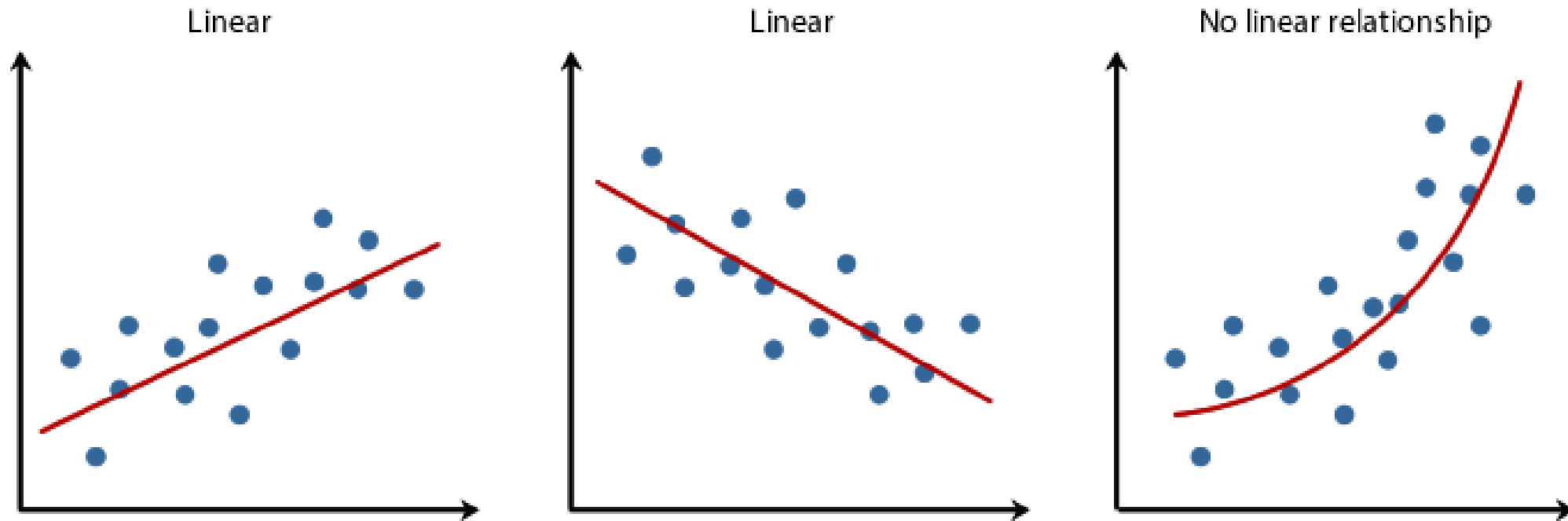
- Learning as error minimization
- Types of errors: L1 and L2

Part 4

Linear Regression

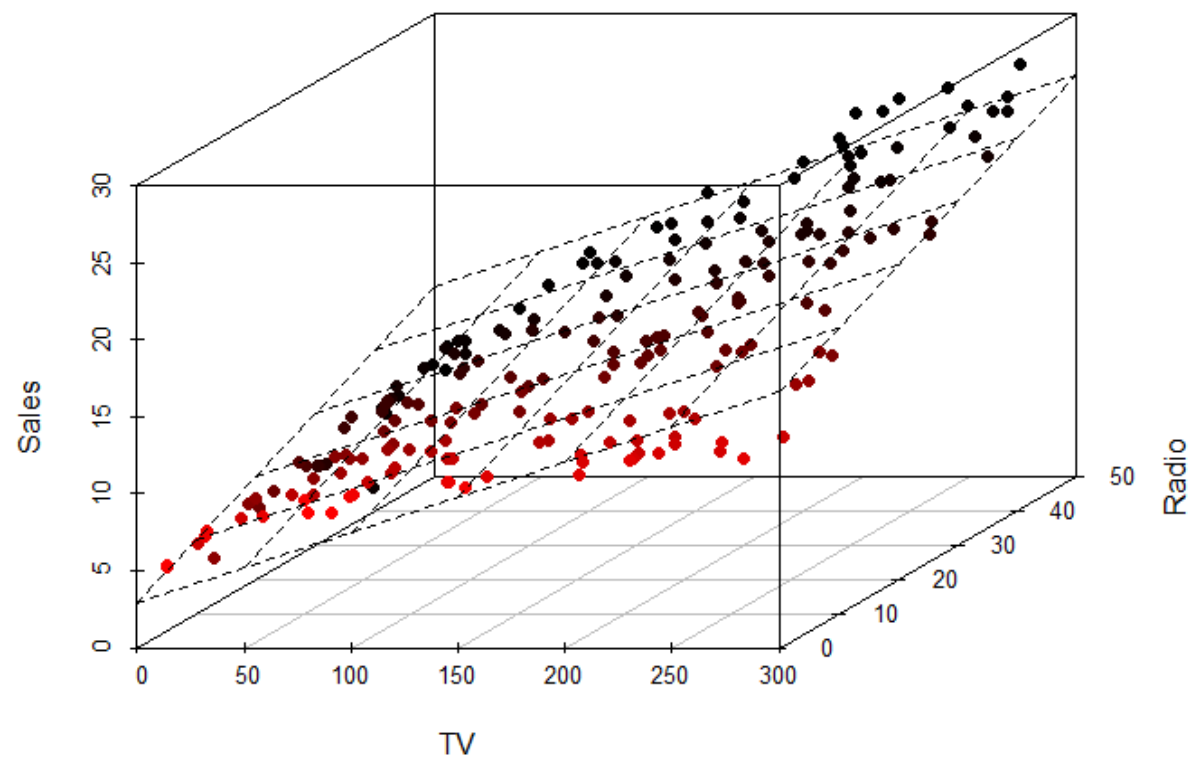


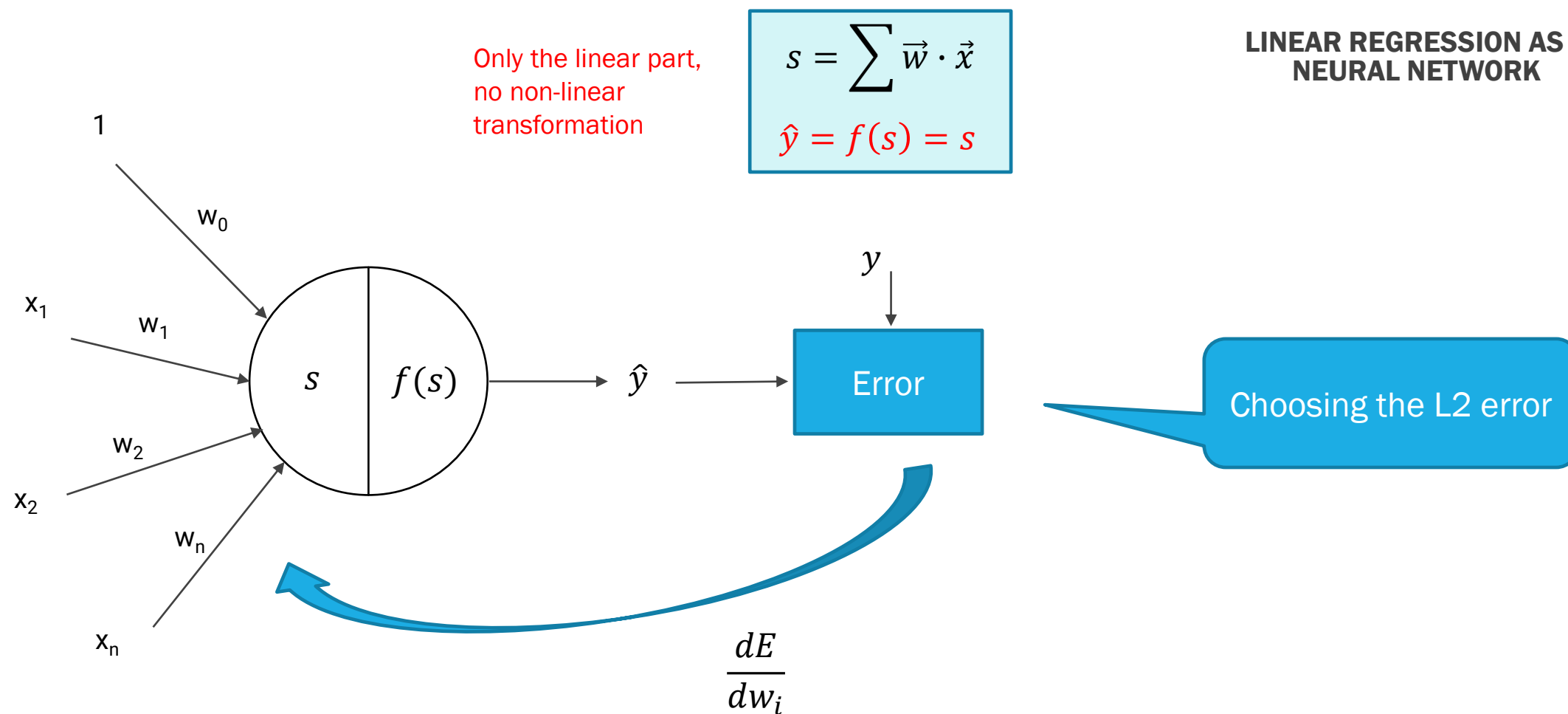
LINEAR REGRESSION



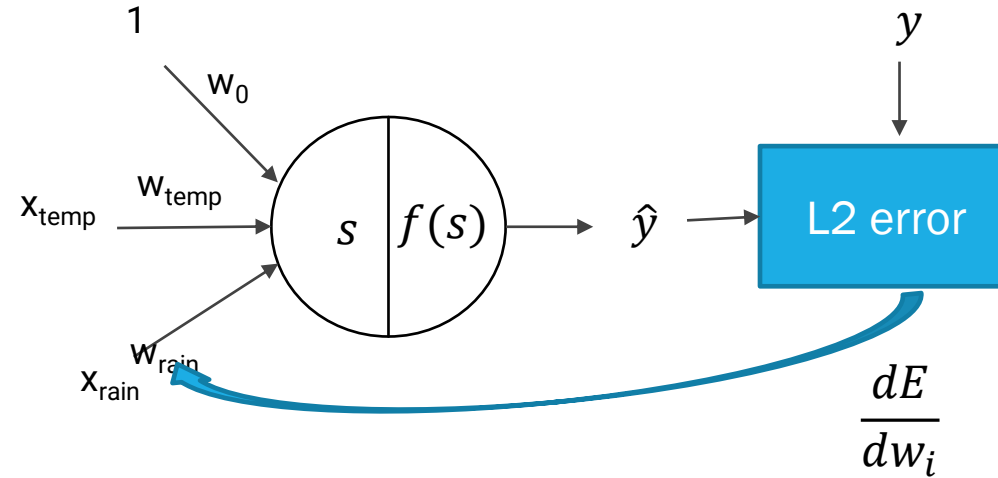
Copyright 2014. Laerd Statistics.

LINEAR REGRESSION





Sample	Temperature	Rain/Snow	Km Biking (y)	Km Biking (\hat{y})
s1	-22	10	2	
s2	-24	12	0	
s3	-15	20	3	
s4	-8	13	4	
s5	-5	40	3	
s6	8	45	8	
s7	-2	5	10	
s8	5	5	12	
s9	12	2	12	
s10	8	5	14	
s11	12	20	20	
s12	13	15	20	
s13	15	10	25	
s14	18	5	20	
s15	19	20	40	



Model:

$(w_0, w_{temp}, w_{rain})$

GRADIENT DESCENT

Initialize weights at random

Repeat:

For each example in training set:

a. Predict \hat{y} (forward pass)

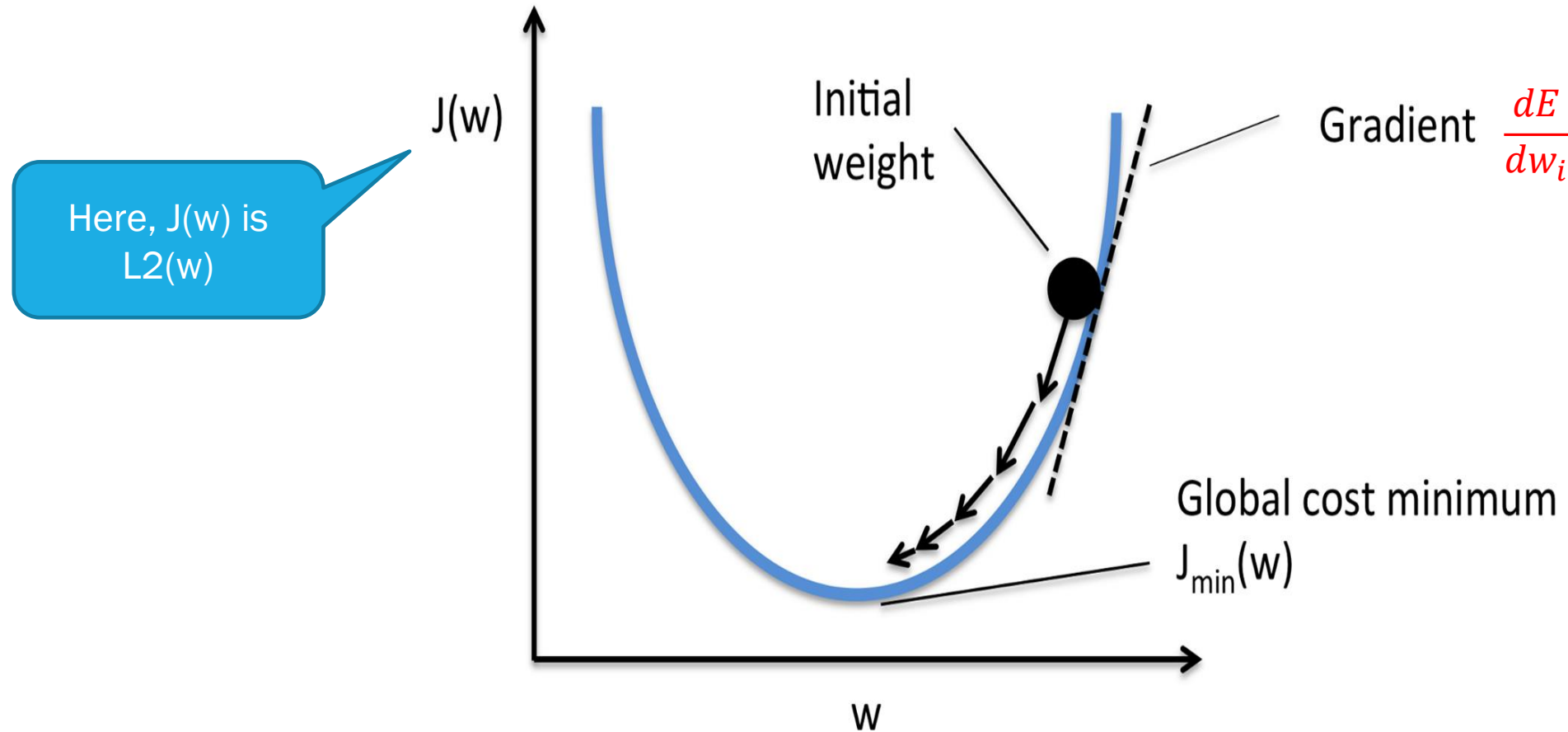
b. For each weight w_i

a. Calculate derivative of error $\frac{dE}{dw_i}$

b. Update weight $w_i^{(t)} = w_i^{(t-1)} - \alpha \frac{dE}{dw_i}$

Until "done"

UNDERSTANDING THE LEARNING RATE



$$\frac{\partial L_2}{\partial w_i} = \frac{\partial L_2}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_i}$$

$$L_2 = \frac{1}{2} (y - \hat{y})^2$$

$$\frac{\partial L_2}{\partial \hat{y}} = \frac{2}{2} (y - \hat{y}) \cdot -1 = -(y - \hat{y})$$

$$\hat{y} = \sum_{i=0}^n w_i x_i$$

$$\begin{aligned} \frac{\partial \hat{y}}{\partial w_i} &= \frac{\partial (w_0 x_0 + w_1 x_1 + \dots + w_i x_i + \dots + w_n x_n)}{\partial w_i} \\ &= x_i \end{aligned}$$

Calculating the derivative of the error

GRADIENT DESCENT

Initialize weights at random

Repeat:

For each example in training set:

- a. Predict \hat{y} (forward pass)
- b. For each weight w_i
 - a. Calculate derivative of error $\frac{dE}{dw_i} = -(y - \hat{y}) x_i$
 - b. Update weight $w_i^{(t)} = w_i^{(t-1)} - \alpha \frac{dE}{dw_i}$

Until "done"

GRADIENT DESCENT

Initialize weights at random

Repeat:

For each example in training set:

- a) Fixed number of iterations
- b) Error < error threshold
- c) $w_i(t+1) - w_i(t) < \text{change threshold}$

a. Predict \hat{y} (forward pass)

b. For each weight w_i

a. Calculate derivative of error $\frac{dE}{dw_i} = -(y - \hat{y}) x_i$

b. Update weight $w_i^{(t)} = w_i^{(t-1)} - \alpha \frac{dE}{dw_i}$

Until "done"

Initialize weights at random

Repeat:

For each example in training set:

a. Predict \hat{y} (forward pass)

b. Calculate $\delta = -(y - \hat{y})$

c. For each weight w_i

a. Calculate derivative of error $\frac{dE}{dw_i} = \delta x_i$

b. Update weight $w_i^{(t)} = w_i^{(t-1)} - \alpha \frac{dE}{dw_i}$

Until "done"

Move calculation outside the loop since it's repeated for all w_i

NORMALIZATION

Before starting the learning, we can normalize (between -1 and 1) or between (0 and 1) each of the attributes.

Sample	Temperature	Rain/snow	Km biking
s1	-22	10	2
s2	-24	12	0
s3	-15	20	3
s4	-8	13	4
s5	-5	40	3
s6	8	45	8
s7	-2	5	10
s8	5	5	12
s9	12	2	12
s10	8	5	14
s11	12	20	20
s12	13	15	20
s13	15	10	25
s14	18	5	20
s15	19	20	40

Multivariable regression

Let's say we are given [data](#) on TV, radio, and newspaper advertising spend for a list of companies, and our goal is to predict sales in terms of units sold.

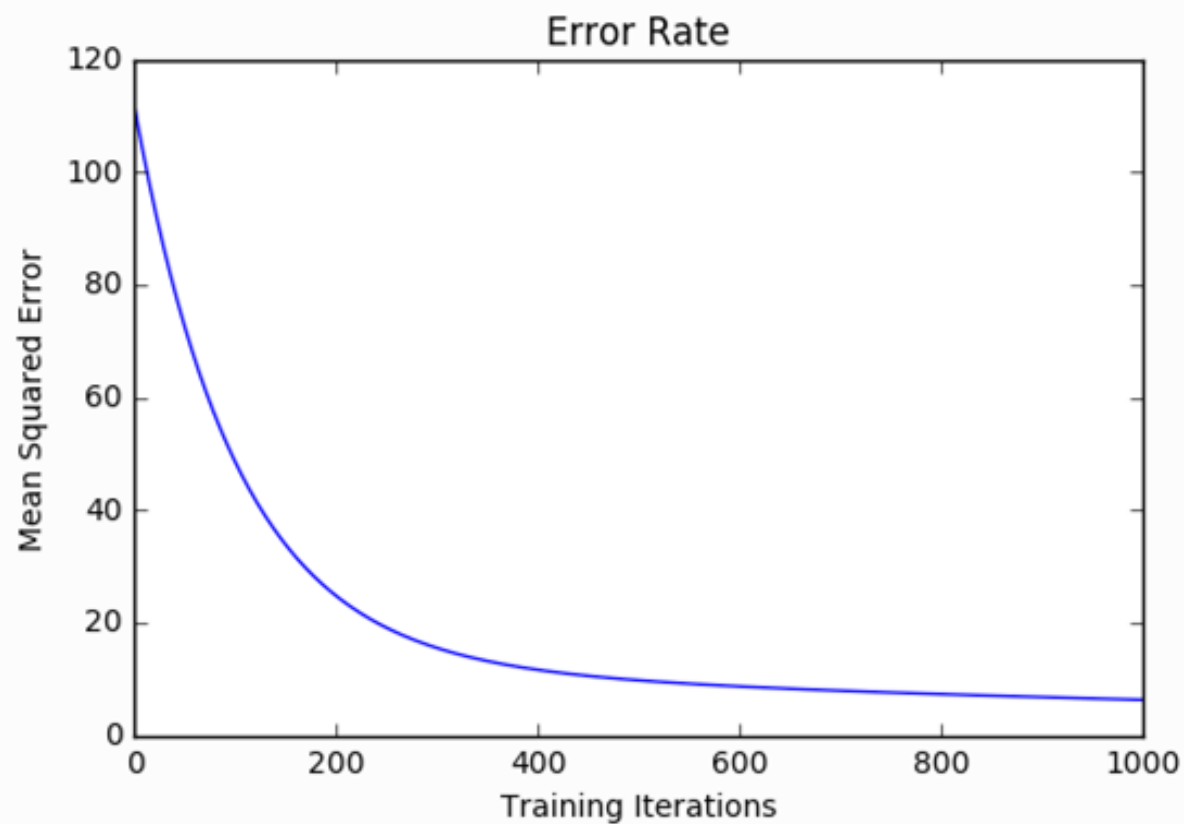
	TV	radio	newspaper	sales
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9
6	8.7	48.9	75	7.2
7	57.5	32.8	23.5	11.8
8	120.2	19.6	11.6	13.2
9	8.6	2.1	1	4.8
10	199.8	2.6	21.2	10.6
11	66.1	5.8	24.2	8.6
12	214.7	24	4	17.4
13	23.8	35.1	65.9	9.2
14	97.5	7.6	7.2	9.7
15	204.1	32.9	46	19
16	195.4	47.7	52.9	22.4
17	67.8	36.6	114	12.5
18	281.4	39.6	55.8	24.4
19	69.2	20.5	18.3	11.3
20	147.3	23.9	19.1	14.6
21	218.4	27.7	53.4	18
22	237.4	5.1	23.5	12.5
23	13.2	15.9	49.6	5.6
24	228.3	16.9	26.2	15.5

Model evaluation

After training our model through 1000 iterations with a learning rate of .0005, we finally arrive at a set of weights we can use to make predictions:

$$\text{Sales} = 4.7\text{TV} + 3.5\text{Radio} + .81\text{Newspaper} + 13.9$$

Our MSE cost dropped from 110.86 to 6.25.





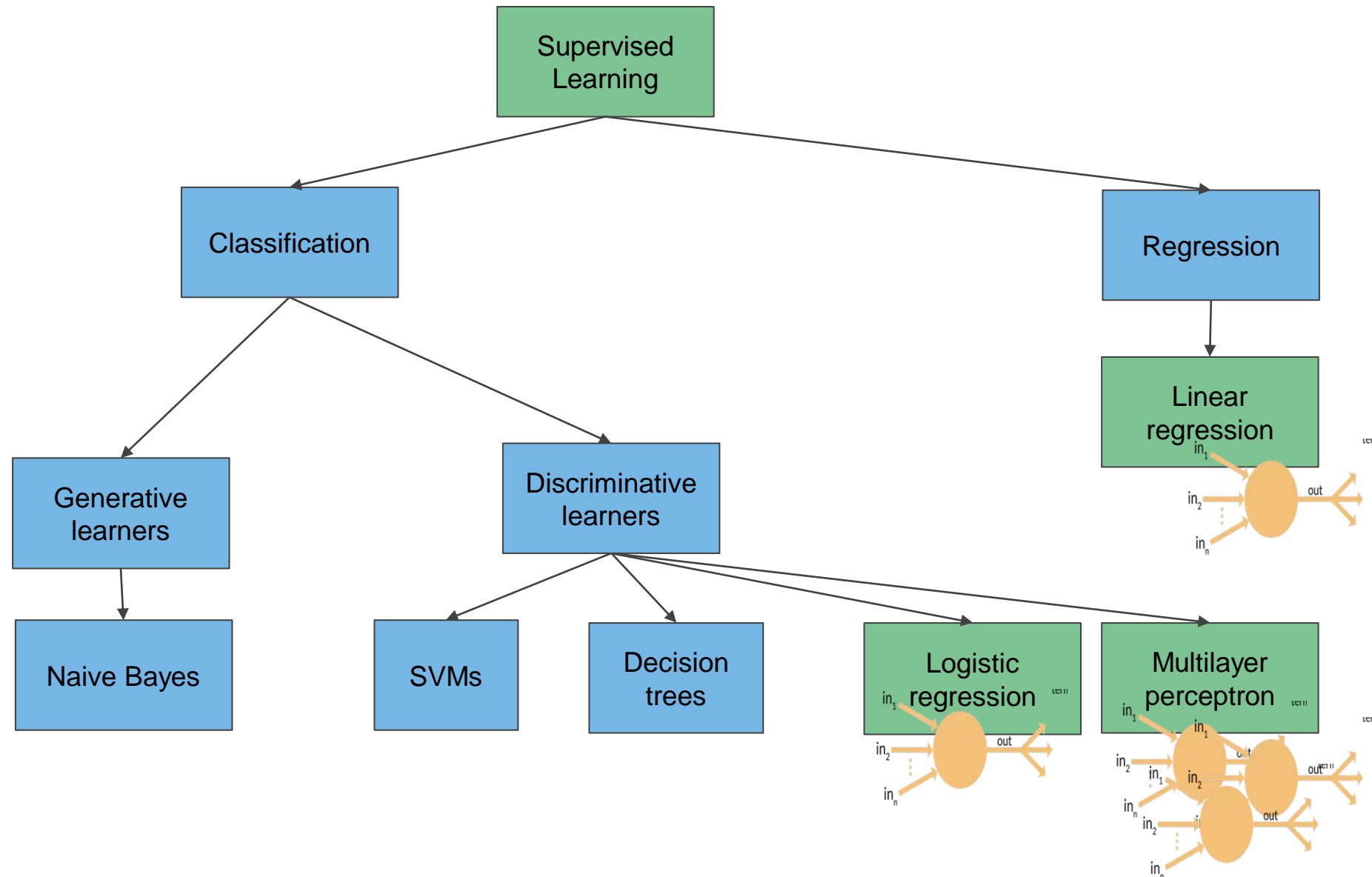
IN SUMMARY

- Gradient descent algorithm for linear regression

Part 5

Logistic Regression

(Perceptron)



**BIKING/DRIVING EXAMPLE
REGRESSION**Training data15 samples, \mathbf{s}_i for $i = 1..15$ Feature vectors \vec{x} has 2 dimensions x_1 Temperature x_2 Rain/snow

Sample	Temperature	Rain/snow	Km biking
s1	-22	10	2
s2	-24	12	0
s3	-15	20	3
s4	-8	13	4
s5	-5	40	3
s6	8	45	8
s7	-2	5	10
s8	5	5	12
s9	12	2	12
s10	8	5	14
s11	12	20	20
s12	13	15	20
s13	15	10	25
s14	18	5	20
s15	19	20	40

Training data

15 samples, \mathbf{s}_i for $i = 1..15$

Feature vectors

\vec{x} has 2 dimensions

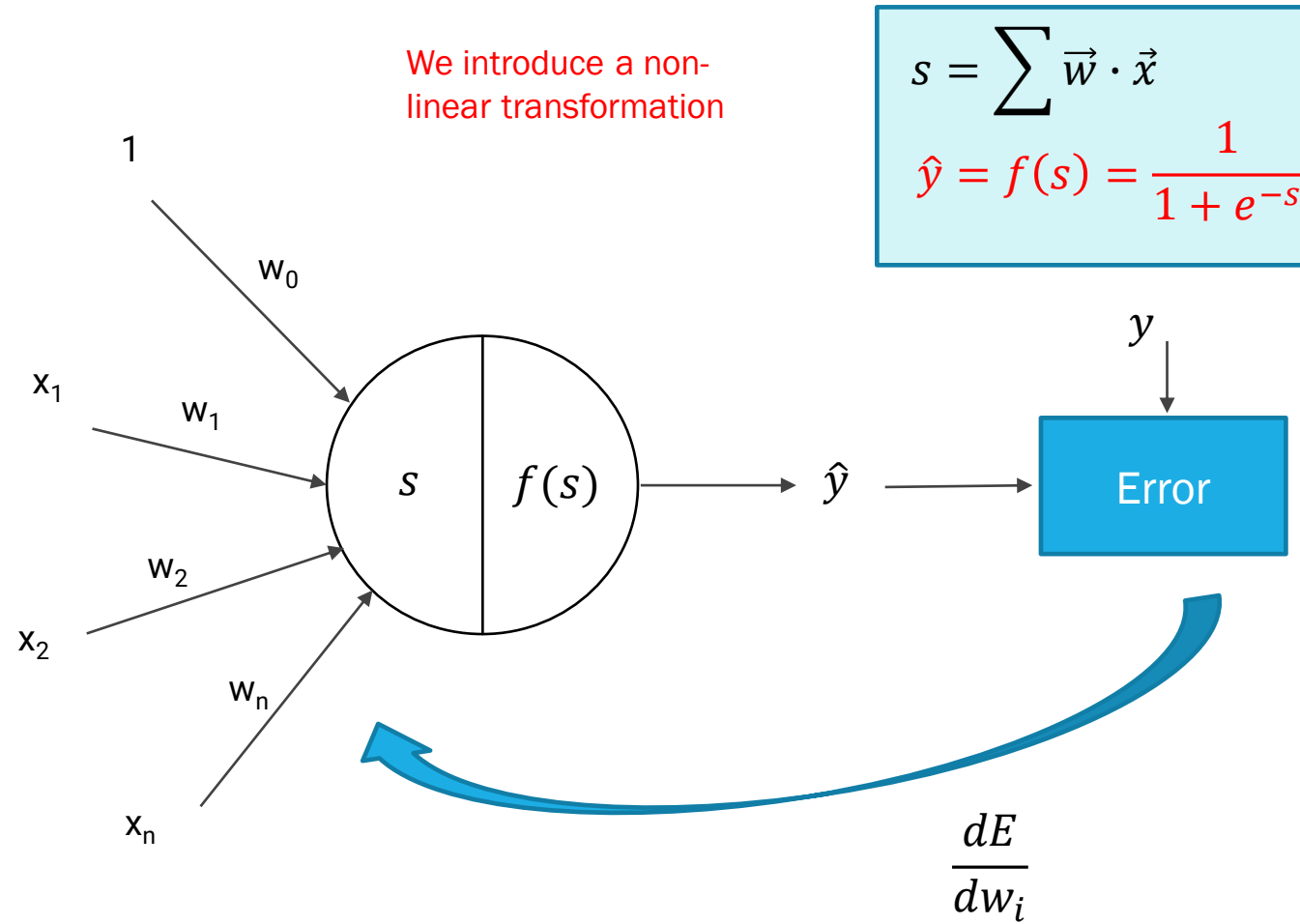
x_1 Temperature

x_2 Rain/snow

Sample	Temperature	Rain/snow	Bike / Drive
s1	-22	10	Drive
s2	-24	12	Drive
s3	-15	20	Drive
s4	-8	13	Drive
s5	-5	40	Drive
s6	8	45	Drive
s7	-2	5	Bike
s8	5	5	Bike
s9	12	2	Bike
s10	8	5	Bike
s11	12	20	Bike
s12	13	15	Bike
s13	15	10	Bike
s14	18	5	Bike
s15	19	20	Bike

BIKING/DRIVING EXAMPLE CLASSIFICATION

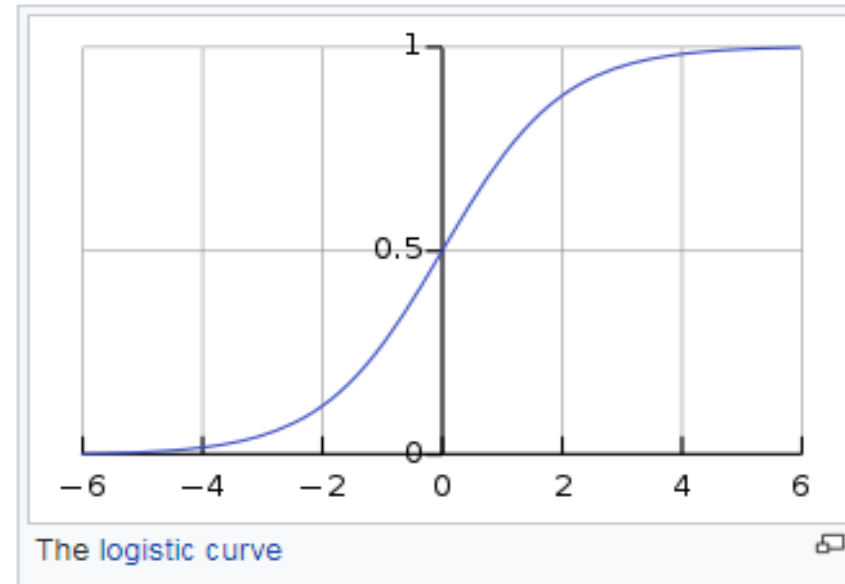
LOGISTIC REGRESSION – PERCEPTRON



SIGMOID FUNCTION

A **sigmoid function** is a mathematical function having a characteristic "S"-shaped curve or **sigmoid curve**. Often, *sigmoid function* refers to the special case of the **logistic function** shown in the first figure and defined by the formula

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}.$$



GRADIENT DESCENT

Initialize weights at random

Repeat:

For each example in training set:

- a. Predict \hat{y} (forward pass)
- b. For each weight w_i
 - a. Calculate derivative of error
 - b. Update weight

Which error ?

$$\frac{dE}{dw_i}$$

$$w_i^{(t)} = w_i^{(t-1)} - \alpha \frac{dE}{dw_i}$$

Until "done"

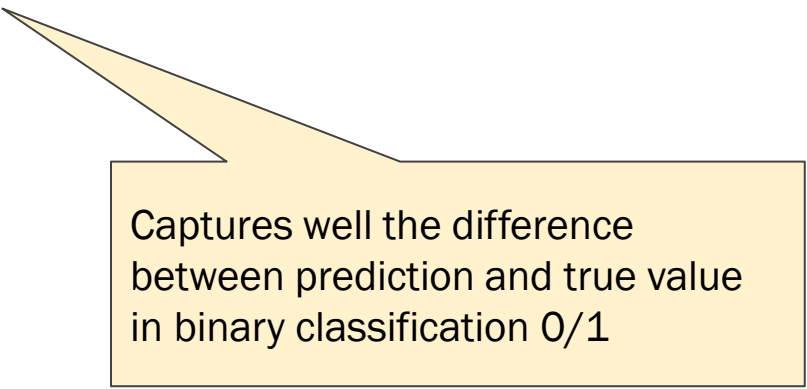
- a) Fixed number of iterations
- b) Error < error threshold
- c) $w_i(t+1) - w_i(t) < \text{change threshold}$

LOG LOSS ERROR

To optimize the log loss error for logistic regression, minimize the negative log-likelihood

$$LL(E, \bar{w}) = - \left(\sum_{e \in E} \left(Y(e) * \log \hat{Y}(e) + (1 - Y(e)) * \log(1 - \hat{Y}(e)) \right) \right)$$

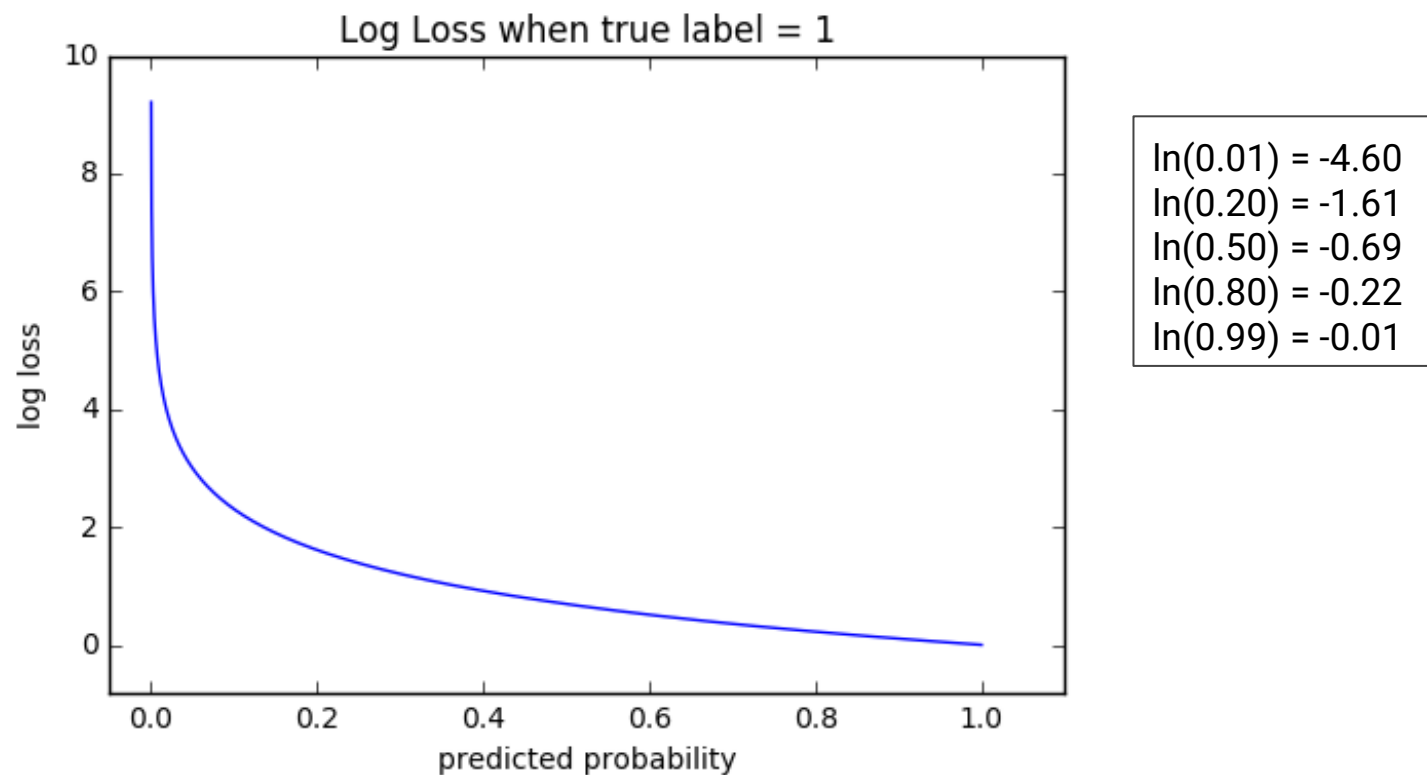
where $\hat{Y}(e) = \text{sigmoid}(\sum_{i=0}^n w_i * X_i(e))$.



Captures well the difference between prediction and true value in binary classification 0/1

LOG LOSS ERROR

$$LL(E, \bar{w}) = - \left(\sum_{e \in E} \left(Y(e) * \log \hat{Y}(e) + (1 - Y(e)) * \log(1 - \hat{Y}(e)) \right) \right)$$



Initialize weights at random

Repeat:

For each example in training set:

- a. Predict \hat{y} (forward pass)
- b. For each weight w_i
 - a. Calculate derivative of error $\frac{dE}{dw_i}$
 - b. Update weight

$$w_i^{(t)} = w_i^{(t-1)} - \alpha \frac{dE}{dw_i}$$

Until "done"

Now the error will be the log loss

Initialize weights at random

Repeat:

For each example in training set:

a. Predict \hat{y} (forward pass)

b. For each weight w_i

a. Calculate derivative of error $\frac{dE}{dw_i} = -(y - \hat{y}) x_i$

b. Update weight

$$w_i^{(t)} = w_i^{(t-1)} - \alpha \frac{dE}{dw_i}$$

Until "done"

You can do the derivative... the result turns out to be the same as with L2

$$\frac{\partial LL}{\partial w_i} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial s} \cdot \frac{\partial s}{\partial w_i}$$

EXAMPLE OF LOGISTIC REGRESSION

Can the model learn to predict if a person will like a holiday base on different feature values

<i>Culture</i>	<i>Fly</i>	<i>Hot</i>	<i>Music</i>	<i>Nature</i>	<i>Likes</i>
0	0	1	0	0	0
0	1	1	0	0	0
1	1	1	1	1	0
0	1	1	1	1	0
0	1	1	0	1	0
1	0	0	1	1	1
0	0	0	0	0	0
0	0	0	1	1	1
1	1	1	0	0	0
1	1	0	1	1	1
1	1	0	0	0	1
1	0	1	0	1	1
0	0	0	1	0	0
1	0	1	1	0	0
1	1	1	1	0	0
1	0	0	1	0	0
1	1	1	0	1	0
0	0	0	0	1	1
0	1	0	0	0	1

After 10,000 iterations of gradient descent with a learning rate of 0.05, the prediction found is (to one decimal point)

$$\begin{aligned} \text{lin}(e) = & 2.3 * \text{Culture}(e) + 0.01 * \text{Fly}(e) - 9.1 * \text{Hot}(e) \\ & - 4.5 * \text{Music}(e) + 6.8 * \text{Nature}(e) + 0.01 \end{aligned}$$

$$\widehat{\text{Likes}}(e) = \text{sigmoid}(\text{lin}(e)) \ .$$

In a Perceptron, the model learned is an equation.

<i>Culture</i>	<i>Fly</i>	<i>Hot</i>	<i>Music</i>	<i>Nature</i>	<i>Likes</i>	<i>lin</i>	\widehat{Likes}
0	0	1	0	0	0	-9.09	0.00011
0	1	1	0	0	0	-9.08	0.00011
1	1	1	1	1	0	-4.48	0.01121
0	1	1	1	1	0	-6.78	0.00113
0	1	1	0	1	0	-2.28	0.09279
1	0	0	1	1	1	4.61	0.99015
0	0	0	0	0	0	0.01	0.50250
0	0	0	1	1	1	2.31	0.90970
1	1	1	0	0	0	-6.78	0.00113
1	1	0	1	1	1	4.62	0.99024

Is the system making good predictions?

The purpose was LL minimisation, so we could evaluate the LL on the training set, but...

EVALUATION IN CLASSIFICATION

$$\begin{aligned}\text{Precision} &= \text{Tp} / (\text{Tp} + \text{Fp}) \\ &= 3 / (3 + 1) = 0.75\end{aligned}$$

$$\begin{aligned}\text{Recal} &= \text{Tp} / (\text{Tp} + \text{Fn}) \\ &= 3 / (3 + 3) = 0.5\end{aligned}$$

		Predicted		
		Like	Not Like	
Gold Standard	Like	Tp = 3	Fn = 3	6
	Not like	Fp = 1	Tn = 4	5
		4	7	11

Test	Gold Standard	Prediction
1	Like	Not like
2	Not like	Not like
3	Not like	Not like
4	Like	Not like
5	Like	Like
6	Not like	Not like
7	Like	Like
8	Not like	Not like
9	Like	Not like
10	Like	Like
11	Not like	Like



IN SUMMARY

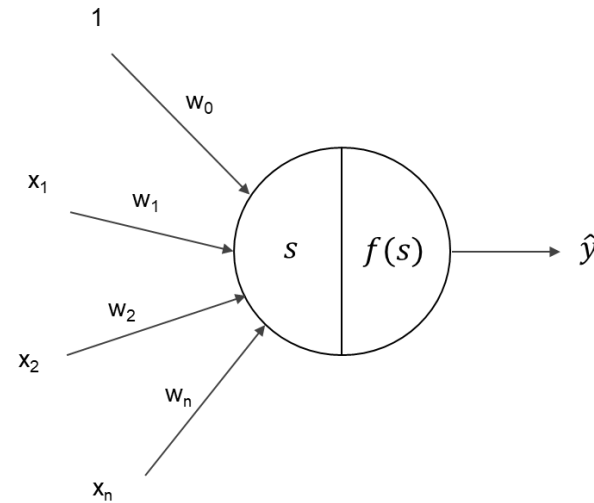
- Introduction to cross-entropy error
- Gradient descent algorithm for logistic regression
- Evaluation of logistic regression as classifier

Part 6

Multinomial Perceptron

Sample	Temperature	Rain/snow	Transport
s1	-22	10	Drive
s2	-24	12	Drive
s3	-15	20	Drive
s4	-8	13	Drive
s5	-5	40	Drive
s6	8	45	Drive
s7	-2	5	Bus
s8	5	5	Bus
s9	12	2	Walk
s10	8	5	Walk
s11	12	20	Walk
s12	13	15	Walk
s13	15	10	Bike
s14	18	5	Bike
s15	19	20	Bike

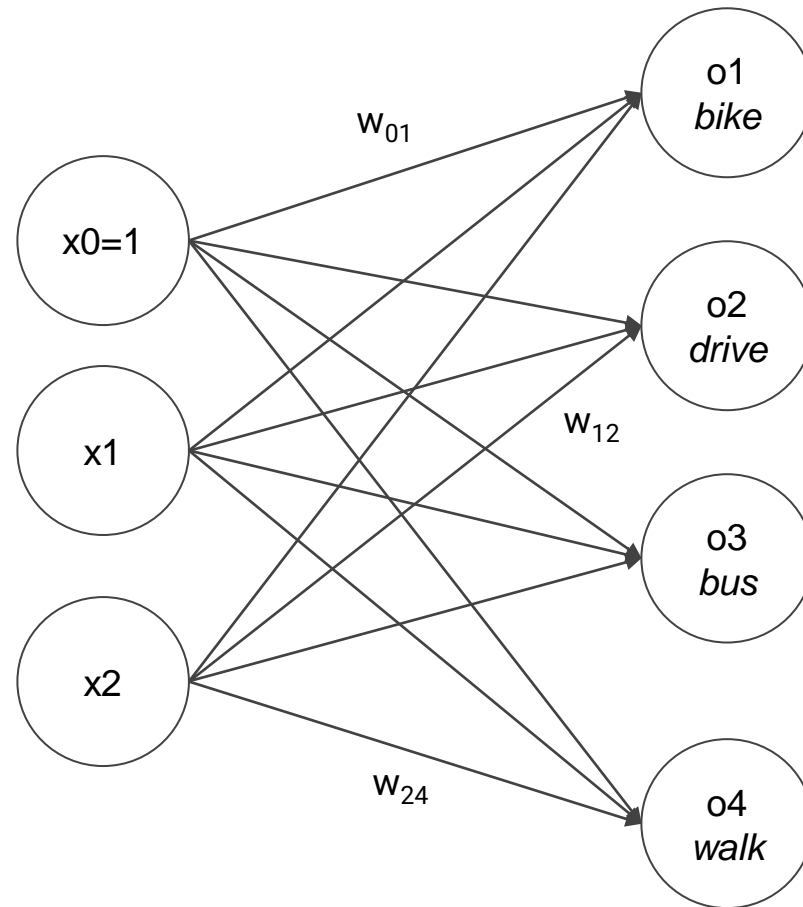
TRANSPORT CLASSIFICATION



Cannot predict 4 values

Sample	Temperature	Rain/snow	Bike	Drive	Bus	Walk
s1	-22	10		1		
s2	-24	12		1		
s3	-15	20		1		
s4	-8	13		1		
s5	-5	40		1		
s6	8	45		1		
s7	-2	5			1	
s8	5	5			1	
s9	12	2				1
s10	8	5				1
s11	12	20				1
s12	13	15				1
s13	15	10	1			
s14	18	5	1			
s15	19	20	1			

MULTINOMIAL PERCEPTRON



Input layer with n nodes
($n-1$ features)

Output layer with m nodes
(m classes)

w_{ij} -- weight from node i to j

Attention, some books
use a reverse indexing

t_j is the target (what we
called y before)

o_j is the prediction (what
we called \hat{y} before)

MULTINOMIAL PERCEPTRON

Linearity

$$s_j = \sum_{i=0}^n w_{ij} \cdot x_i$$

Non-linearity

Assume m classes. Instead of the sigmoid, now the output on node \mathbf{k} (class k), among the m nodes, is given by the **softmax** equation:

$$o_k = \frac{e^{s_k}}{\sum_{j=1}^m e^{s_j}}$$

MULTINOMIAL PERCEPTRON

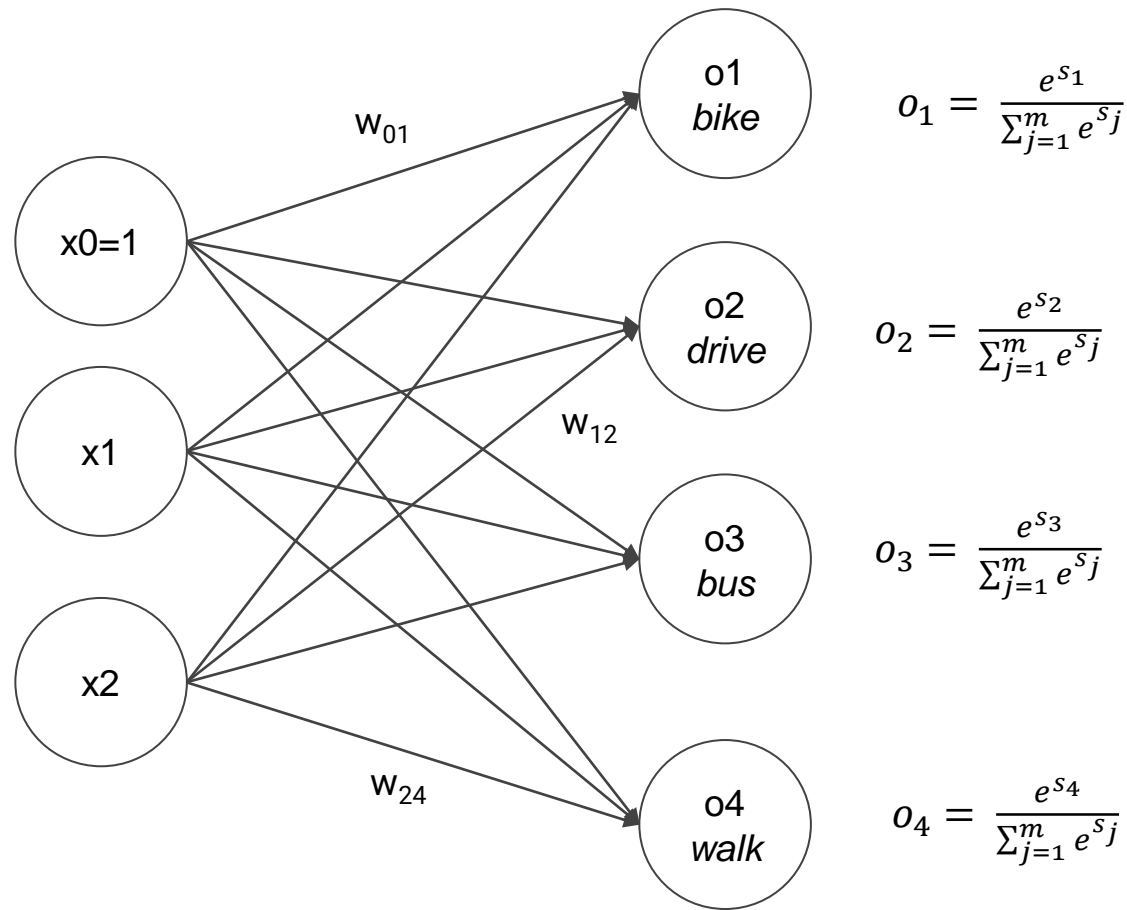
Error function

The error can still be the cross-entropy, generalized to multiple classes, given by:

$$E(t, o) = - \sum_{j=1}^m t_j \cdot \log(o_j)$$

where t_j is the target (0,1) and o_j is the output

MULTINOMIAL PERCEPTRON



$$E(t, o) = - \sum_{j=1}^m t_j \cdot \log(o_j)$$

Input layer with n nodes
($n-1$ features)

Output layer with m nodes
(m classes)

GRADIENT DESCENT

Initialize weights at random

Repeat:

For each example in training set:

a. Predict o_j (forward pass) for all m classes

b. For each weight w_{ij}

a. Calculate derivative of error

$$\frac{dE}{dw_{ij}}$$

b. Update weight

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} - \alpha \frac{dE}{dw_{ij}}$$

Until "done"

DERIVATIVE OF ERROR

Can still use gradient descent to slowly adapt the weights, need to calculate the derivative of the error with respect to each weight.

$$E(t, o) = - \sum_{j=1}^m t_j \cdot \log(o_j)$$

Try to do this one. You'll need the "quotient rule" for this part of the derivation.

$$\frac{\partial E(t, o)}{\partial w_{ij}} = \frac{\partial E(t, o)}{\partial o_j} \cdot \frac{\partial o_j}{\partial s_j} \cdot \frac{\partial s_j}{\partial w_{ij}}$$



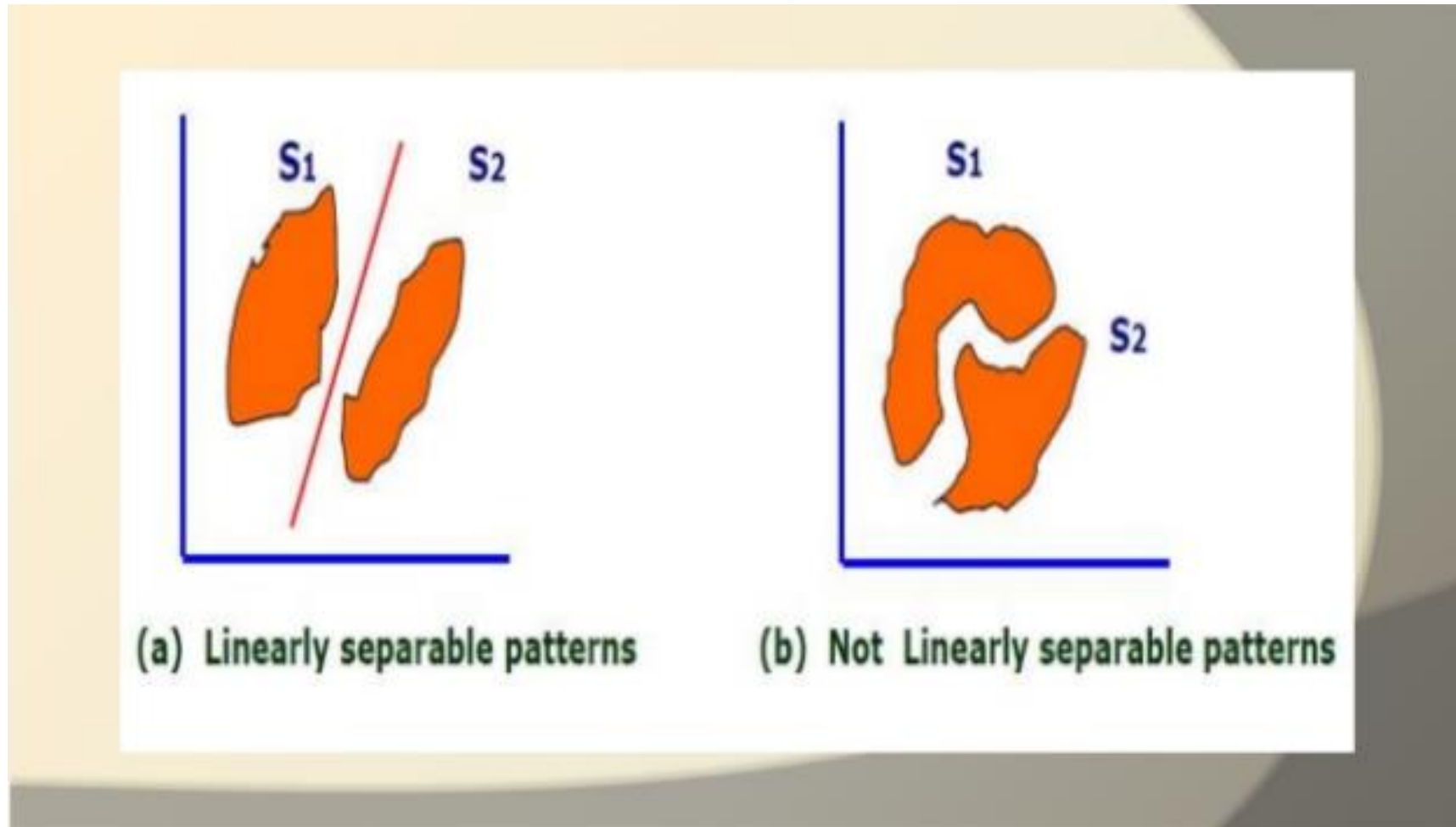
IN SUMMARY

- Expanding to multinomial perceptron

Part 7

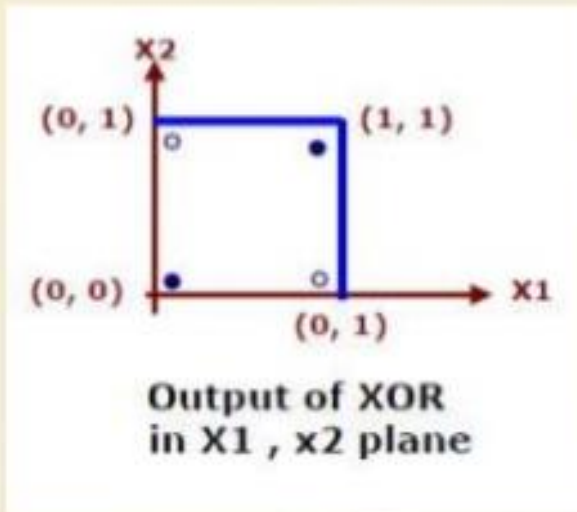
XOR Affair

CLASSIFICATION



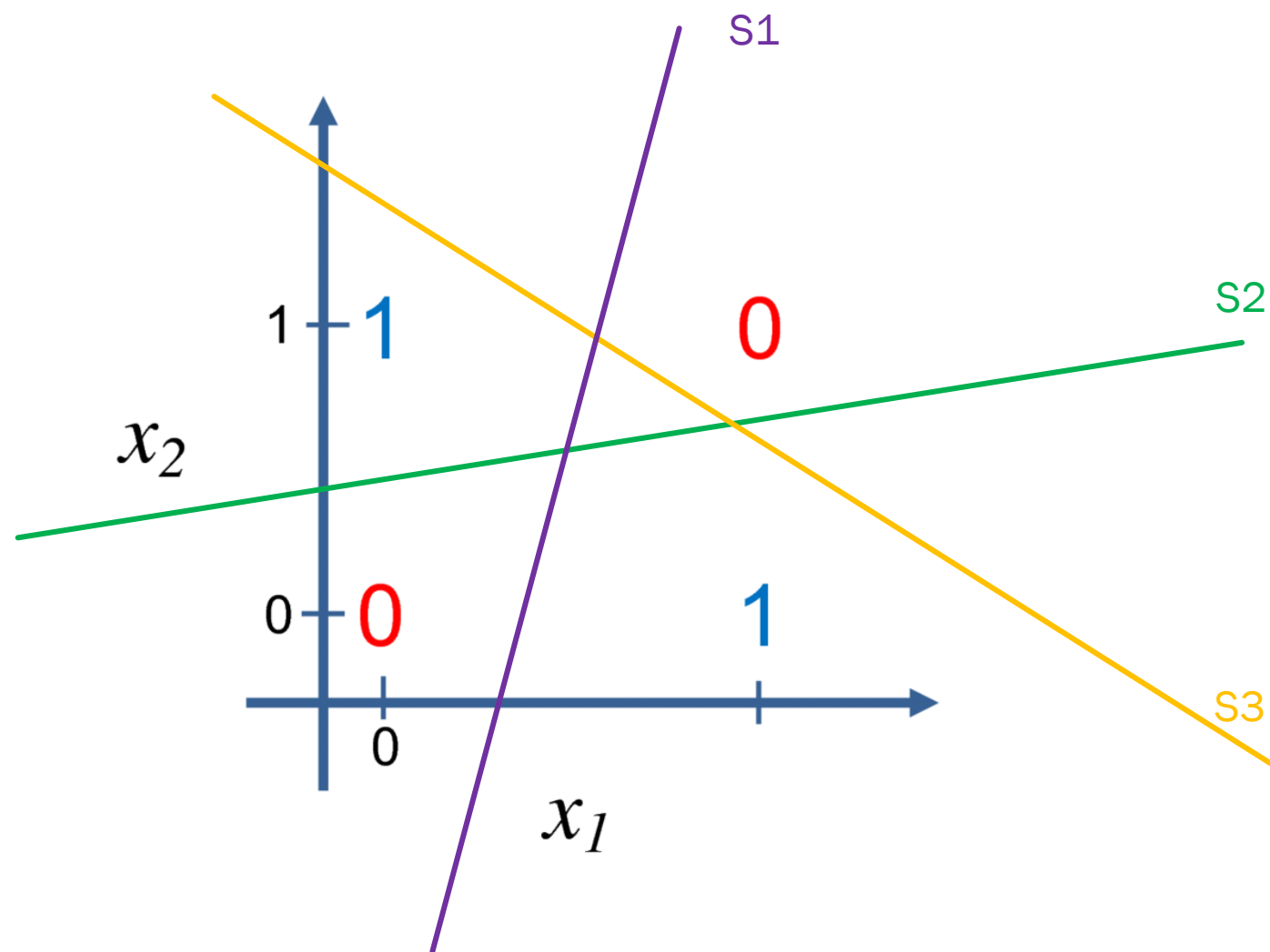
LIMIT TO LINEAR SEPARATORS

- There is no way to draw a single straight line so that the circles are on one side of the line and the dots on the other side.
- Perceptron is unable to find a line separating even parity input patterns from odd parity input patterns.

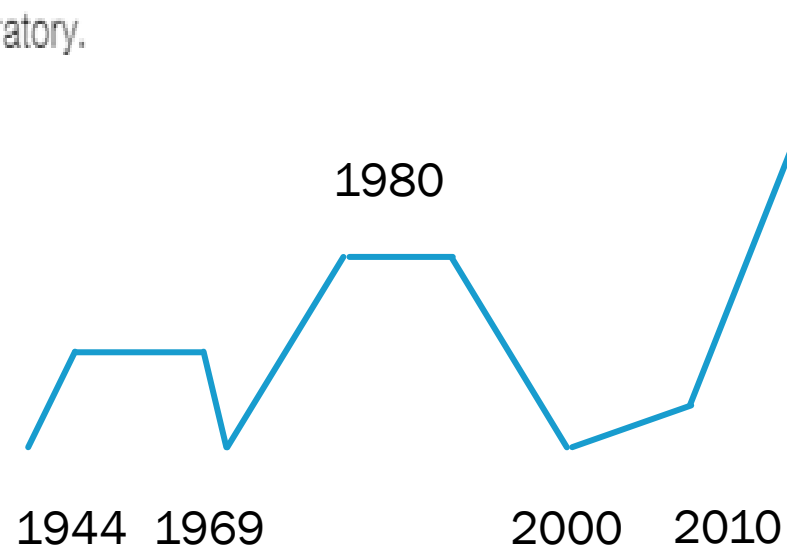


Also see the XOR affair: [https://en.wikipedia.org/wiki/Perceptrons_\(book\)#The_XOR_affair](https://en.wikipedia.org/wiki/Perceptrons_(book)#The_XOR_affair)

LIMIT TO LINEAR SEPARATORS



Neural nets were a major area of research in both neuroscience and computer science until 1969, when, according to computer science lore, they were killed off by the MIT mathematicians Marvin Minsky and Seymour Papert, who a year later would become co-directors of the new MIT Artificial Intelligence Laboratory.

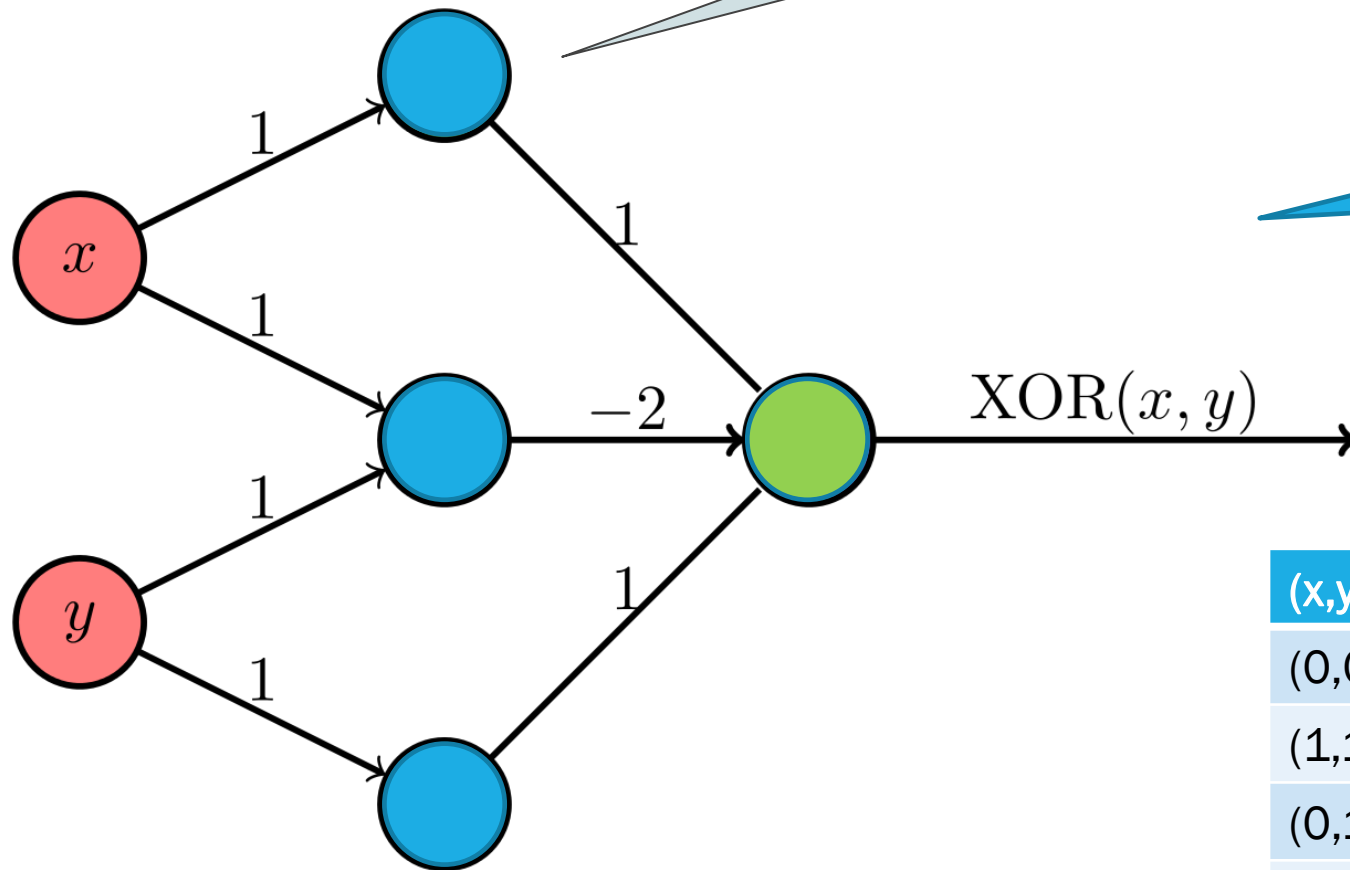


AI was interested in what after the fall of the 70s (between 70-90)

Between 2000-2010... SVMs!

The technique then enjoyed a resurgence in the 1980s, fell into eclipse again in the first decade of the new century, and has returned like gangbusters in the second, fueled largely by the increased processing power of graphics chips.

SOLUTION: ADDING A LAYER



Assume each node of this new "hidden" layer has a sigmoid output

BUT... How will these weights be learned?

(x,y)	z1	z2	z3	In
(0,0)	0:0	0:0	0:0	0
(1,1)	1:1	2:1	1:1	0
(0,1)	0:0	1:1	1:1	-1
(1,0)	1:1	1:1	0:0	-1



IN SUMMARY

- The XOR affair
- The need for multi-layer networks



NEURAL NETWORKS

- Part 1 – Introduction
- Part 2 – Supervised Machine Learning
- Part 3 – Error minimisation
- Part 4 – Linear regression
- Part 5 – Logistic regression (Perceptron)
- Part 6 – Multinomial Perceptron
- Part 7 – XOR affair