Australian National University

Deep Learning Tutorial

Ibrahim Radwan

Agenda

- Why deep learning?
- What is deep learning?
- Introduction to Neural Networks
- Activation Functions
- Cost Functions
- Back propagation
- Building Neural Networks in Python

What you will learn

- Getting familiar with the deep learning techniques and methods.
- Hand on the essentials of understanding deep learning.
- Being able to code and run scripts for applying deep learning tools on different kind of data, i.e. texts, images, etc.

Why Deep Learning?

- Beats state-of-the-art in many applications:
  - Computer Vision
  - Speech Recognition
  - Natural Language Processing

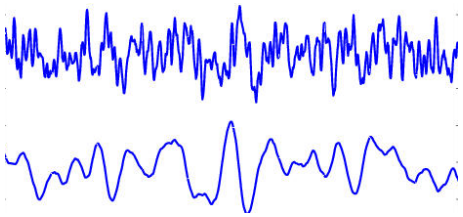
- Can be used for classification, regression, prediction, etc.
- No needs for feature engineering such as in regular machine learning techniques.

What is Deep Learning?

Algorithms which deal with data to build machines that see, learn and interact.

Input

Text Text Text Text
Text Text Text Text
……



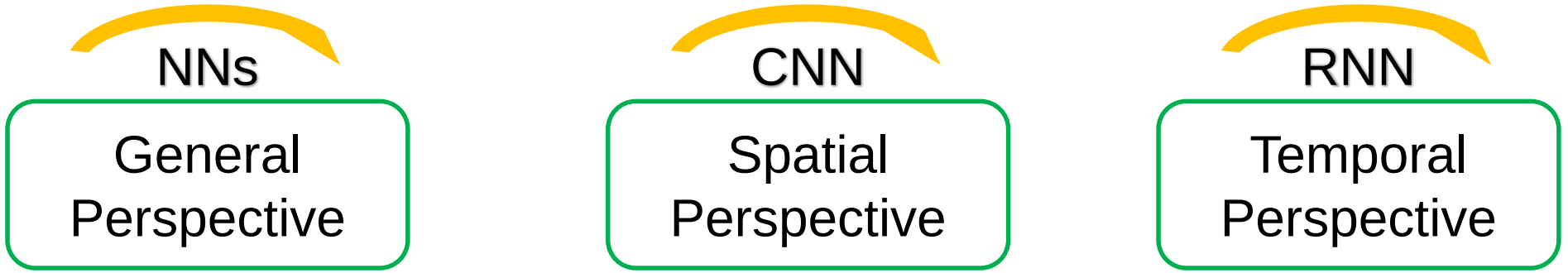
Derive relationships

Deep Learning

Output

- Groups
- Categories
- Classes
- Prediction
- Decision Making
- Features

What to learn to know about Deep Learning?

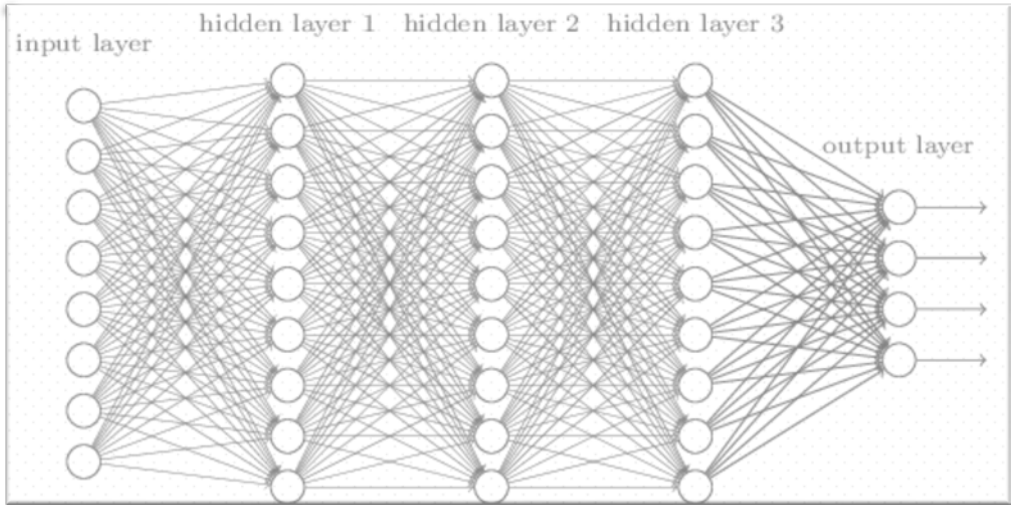


- ANN
- Gradient Descent
- Backpropagation
- Activation functions

- Convolutions
- Pooling
- Deep Networks
- FCN

- Embedding
- Recurrent NNs

Artificial Neural Network



After 2014

2009

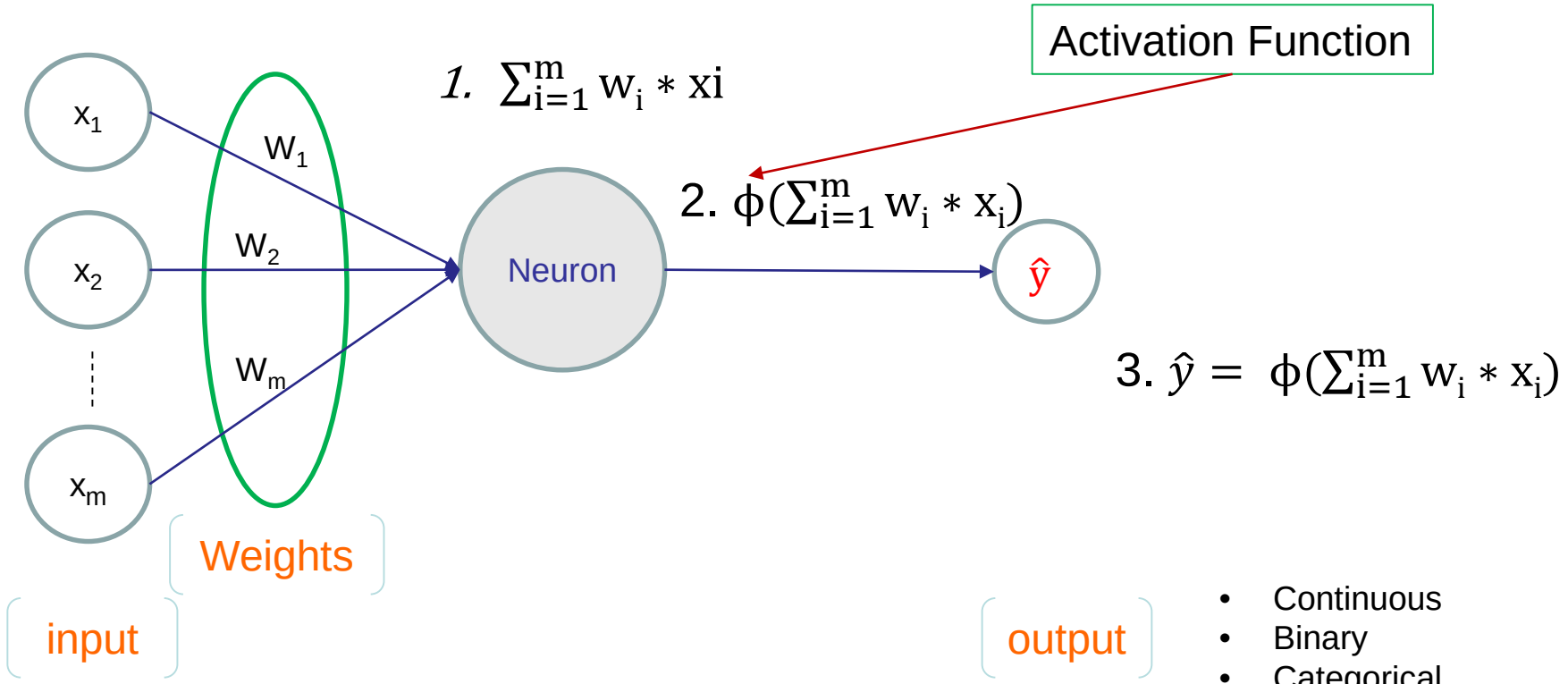
1980s

Before 1980s

1990s

2000s

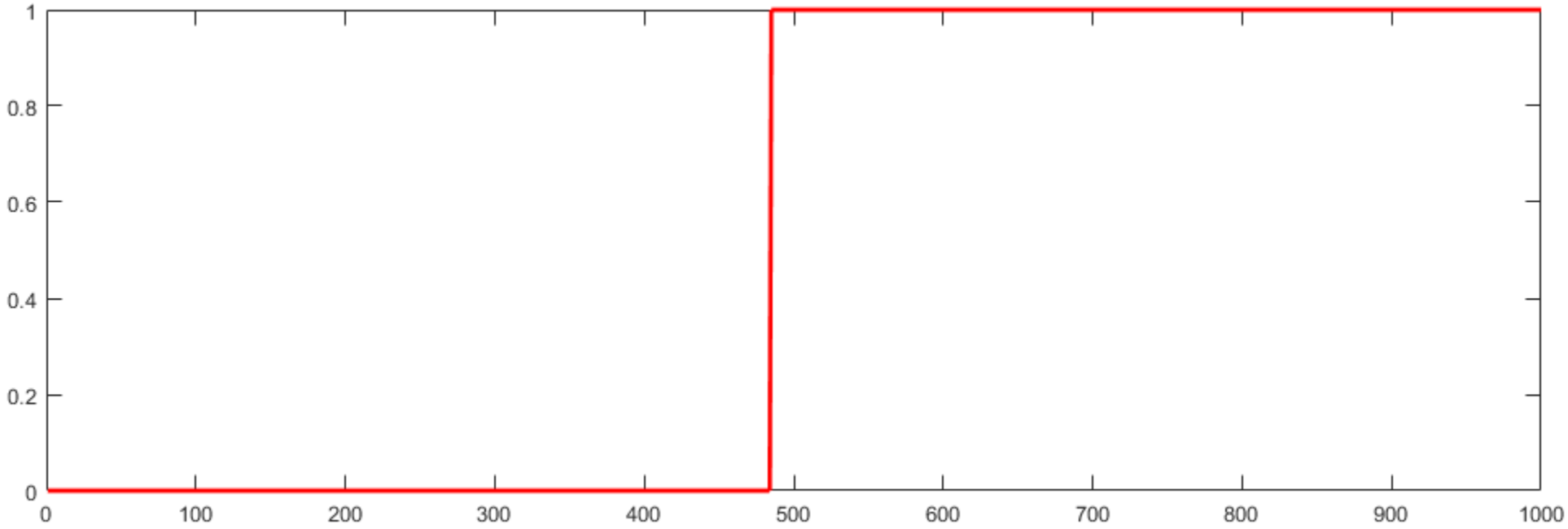
The Neuron


Activation Function
1. $\sum_{i=1}^{m} w_i * xi$
x_1
x_2
x_m
W_1
W_2
W_m
Neuron
2. $\phi(\sum_{i=1}^{m} w_i * x_i)$
$\hat{y}$
3. $\hat{y} = \phi(\sum_{i=1}^{m} w_i * x_i)$
Weights
input
output
Continuous
Binary
Categorical

Activation Functions

1. Threshold

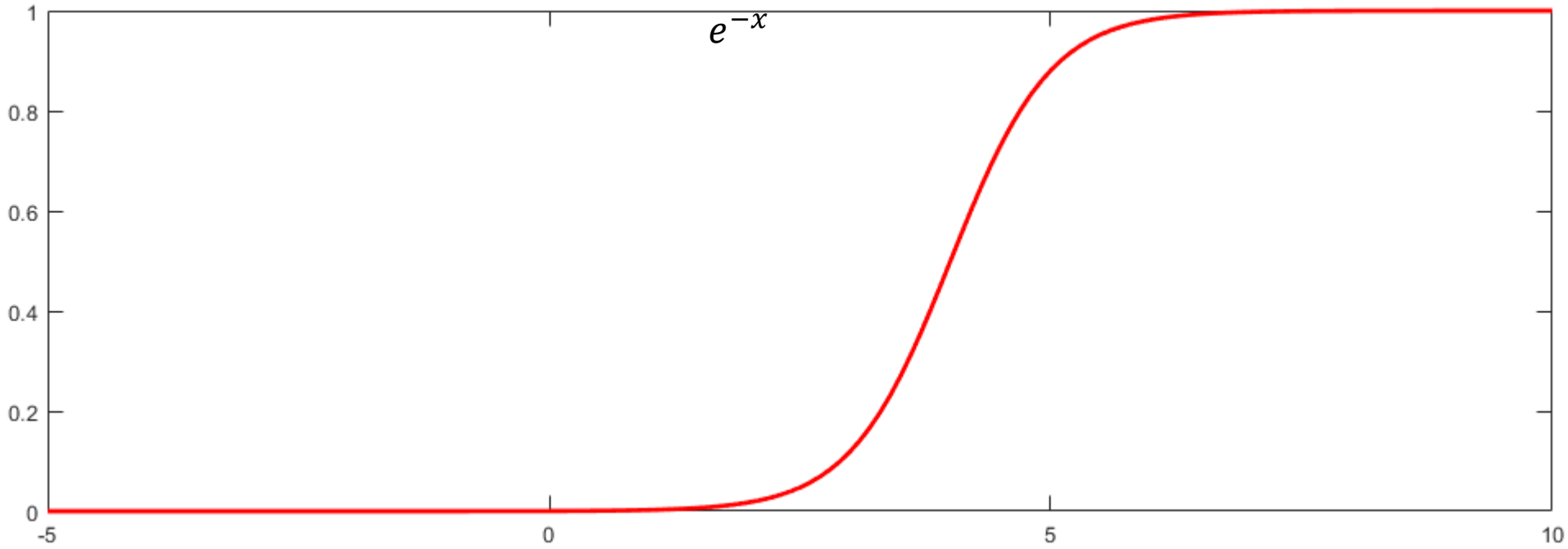
$$\phi(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$



Activation Functions

2. Sigmoid

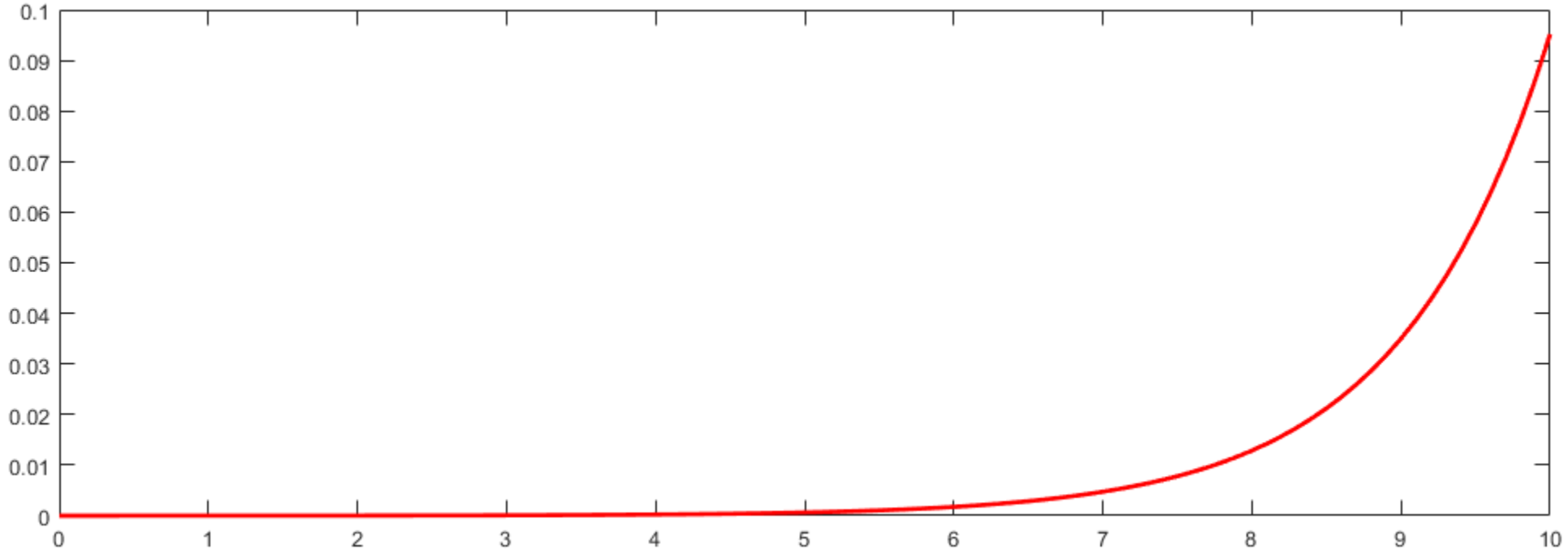
$$\phi(x) = \frac{1}{1+e^{-x}}$$



Activation Functions

3. Softmax (probabilities)

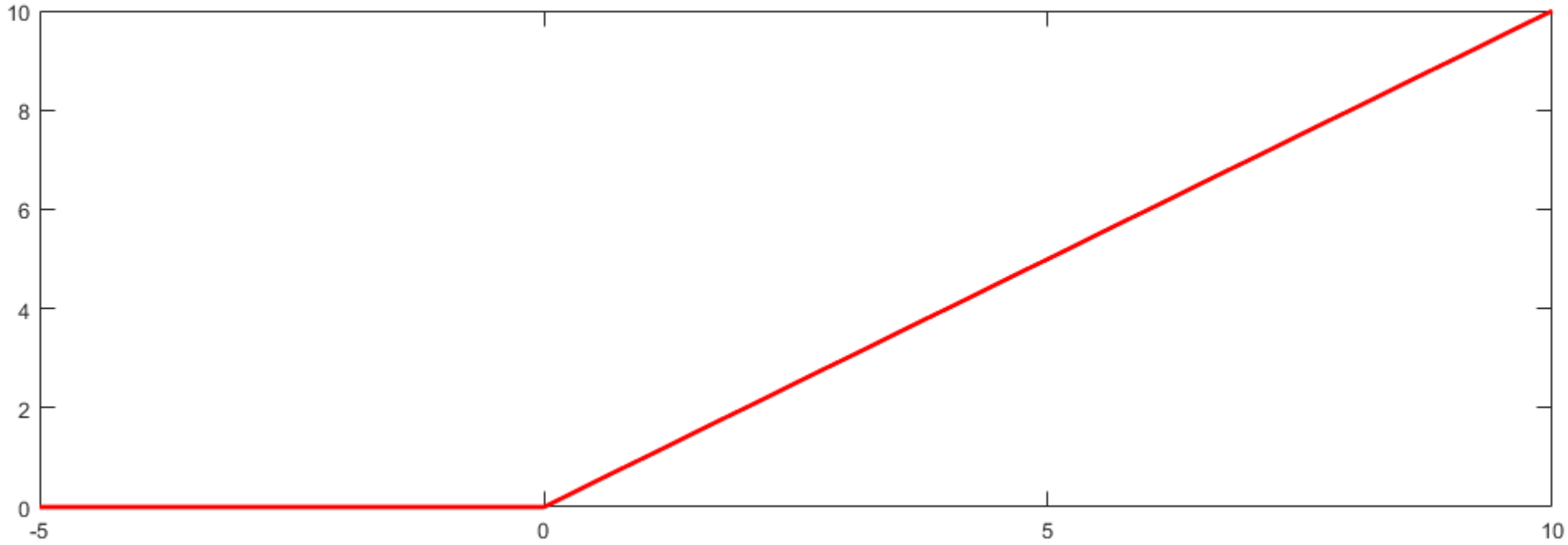
$$\phi(x) = \frac{e^x}{\sum_{i=1}^{m} e^{x_i}}$$



Activation Functions

4. Rectifier

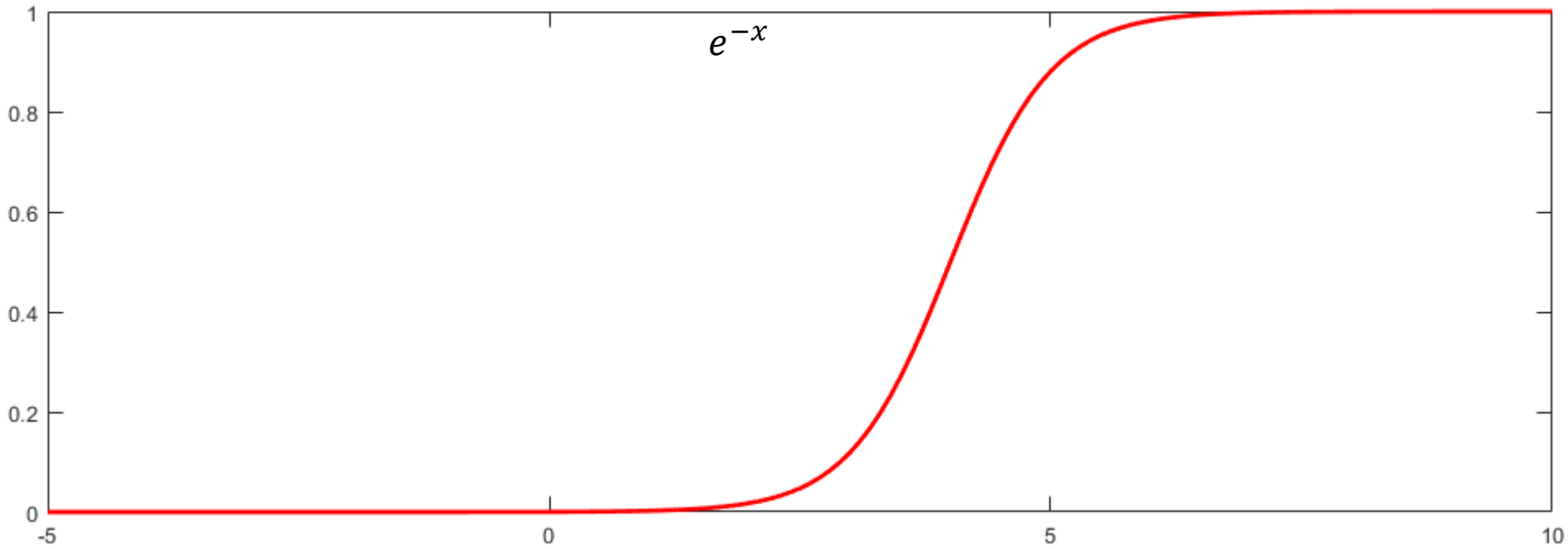
$$\phi(x) = \max(x, 0)$$



Activation Functions

5. Tangent (tanh)

$$\phi(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$



More about activation functions: *"Deep sparse rectifier neural networks (Xavier Clorot et al. 2011)"*

How do NNs work?

- Real Example: Property Evaluation

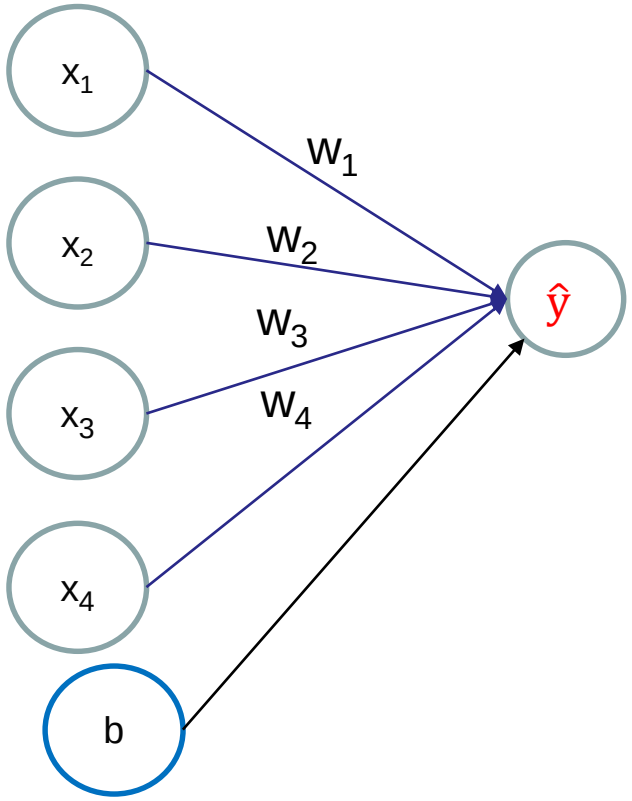
Independent Variables

Area (feet^2)

Bedroom

Distance to city centre (miles)

Age



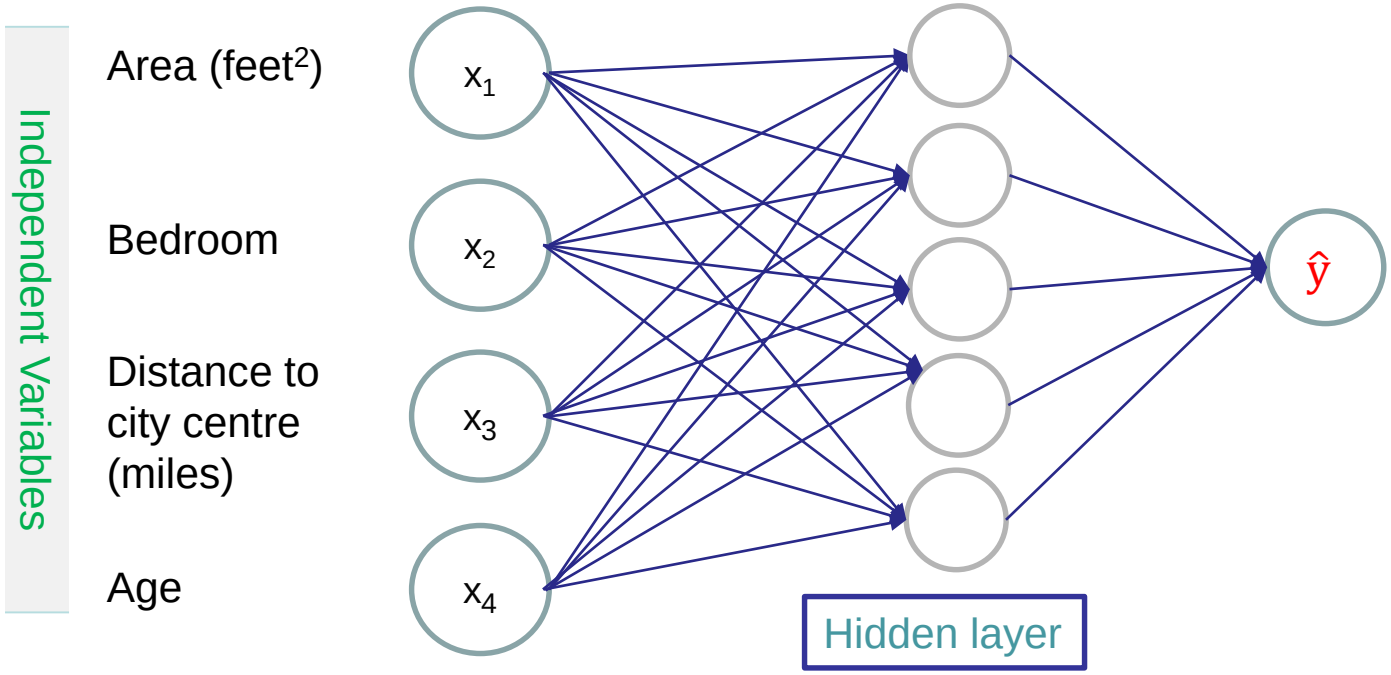
$$\text{Price} = w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + w_4 * x_4 + b$$

$$= \left(\sum_{i=1}^{4} w_i * xi\right) + b$$

Linear Regression

How do NNs work?

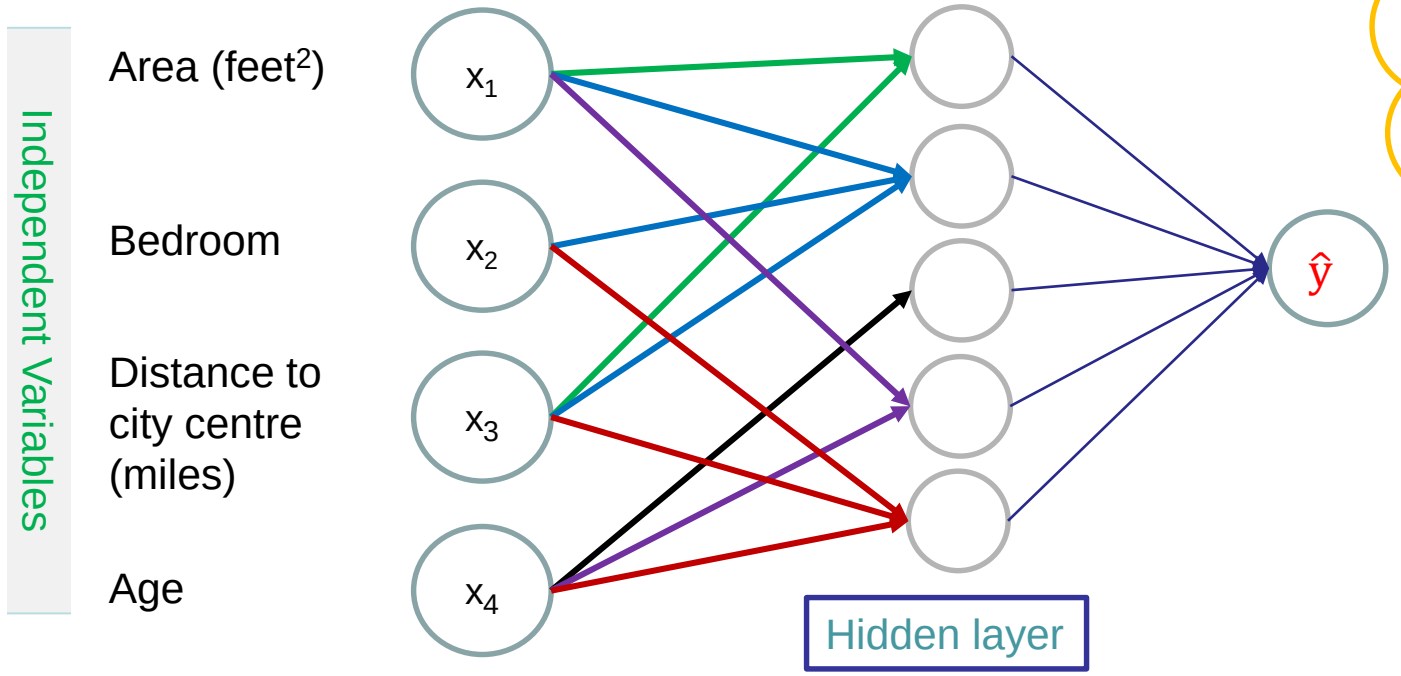
- Real Example: Property Evaluation



All inputs have the same impact

How do NNs work?

- Real Example: Property Evaluation



How many combinations do you need to encode the input – output relationships?

How do NNs learn?

- Real Example: Property Evaluation

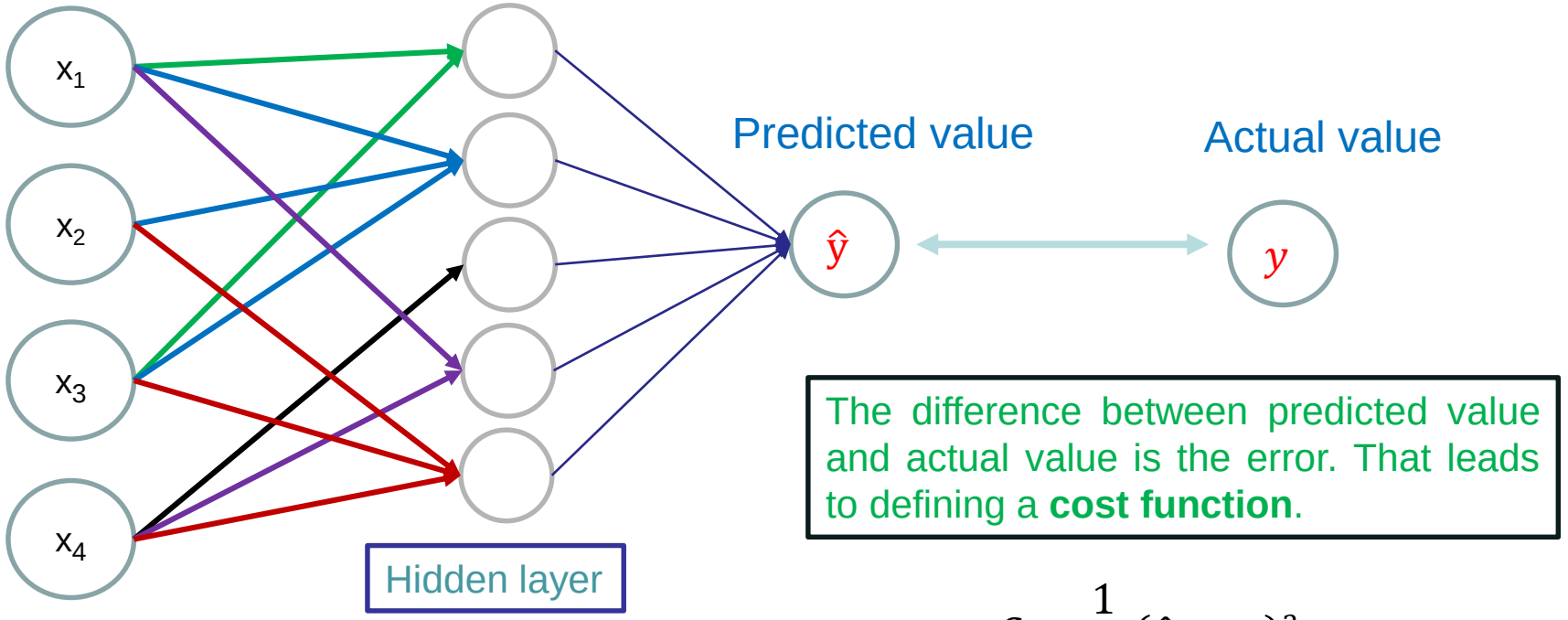
Independent Variables

Area (feet^2)

Bedroom

Distance to city centre (miles)

Age



The difference between predicted value and actual value is the error. That leads to defining a **cost function**.

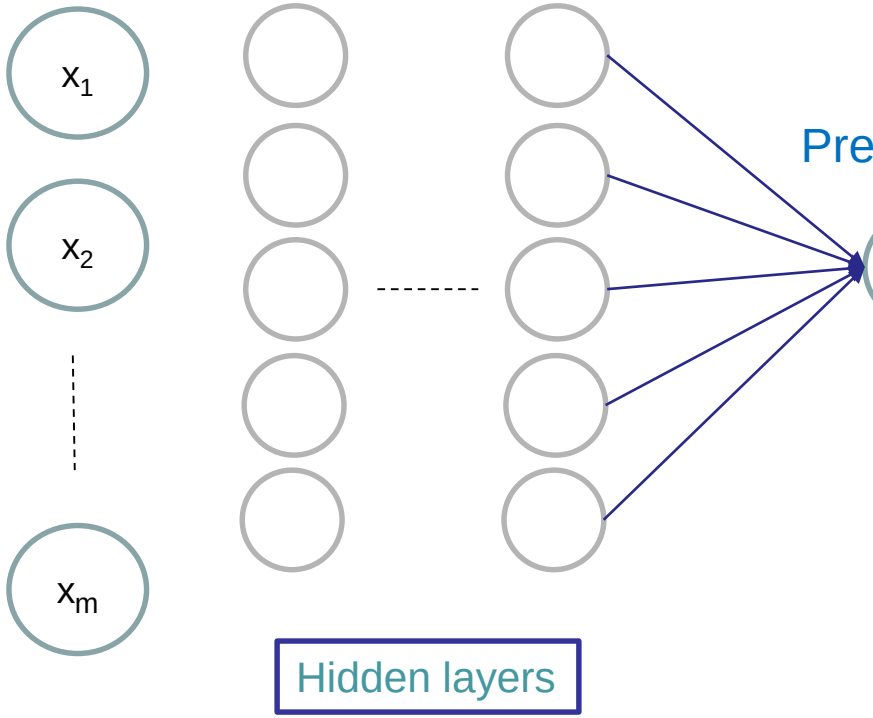
$$C = \frac{1}{2}(\hat{y} - y)^2$$

Another Example

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	RowNum	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
2	1	15634602	Hargrave	619	France	Female	42	2	0	1	1	1	101348.88	1
3	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
4	3	15619304	Onio	502	France	Female	42	8	159660.8	3	1	0	113931.57	1
5	4	15701354	Boni	699	France	Female	39	1	0	2	0	0	93826.63	0
6	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.1	0
7	6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	1	0	149756.71	1
8	7	15592531	Bartlett	822	France	Male	50	7	0	2	1	1	10062.8	0
9	8	15656148	Obinna	376	Germany	Female	29	4	115046.74	4	1	0	119346.88	1
10	9	15792365	He	501	France	Male	44	4	142051.07	2	0	1	74940.5	0
11	10	15592389	H?	684	France	Male	27	2	134603.88	1	1	1	71725.73	0
12	11	15767821	Bearce	528	France	Male	31	6	102016.72	2	0	0	80181.12	0
13	12	15737173	Andrews	497	Spain	Male	24	3	0	2	1	0	76390.01	0
14	13	15632264	Kay	476	France	Female	34	10	0	2	1	0	26260.98	0
15	14	15691483	Chin	549	France	Female	25	5	0	2	0	0	190857.79	0
16	15	15600882	Scott	635	Spain	Female	35	7	0	2	1	1	65951.65	0
17	16	15643966	Goforth	616	Germany	Male	45	3	143129.41	2	0	1	64327.26	0
18	17	15737452	Romeo	653	Germany	Male	58	1	132602.88	1	1	0	5097.67	1
19	18	15788218	Henderson	549	Spain	Female	24	9	0	2	1	1	14406.41	0
20	19	15661507	Muldrow	587	Spain	Male	45	6	0	1	0	0	158684.81	0
21	20	15568982	Hao	726	France	Female	24	6	0	2	1	1	54724.03	0
22	21	15577657	McDonald	732	France	Male	41	8	0	2	1	1	170886.17	0
23	22	15597945	Dellucci	636	Spain	Female	32	8	0	2	1	0	138555.46	0
24	23	15699309	Gerasimov	510	Spain	Female	38	4	0	1	1	0	118913.53	1
25	24	15725737	Mosman	669	France	Male	46	3	0	2	0	1	8487.75	0

Cost Function

- Real Example: Predicting bank exit



Predicted value

Actual value

$\hat{y}$

y

row#	$\hat{y}$	y
1		
2		
…		
n		

$$C = \frac{1}{n}\sum(\hat{y}_i - yi)^2$$

Mean square error (MSE)

Cost Function

More reading:

"*A list of cost functions used in neural networks*", article in "*cross-validated*"

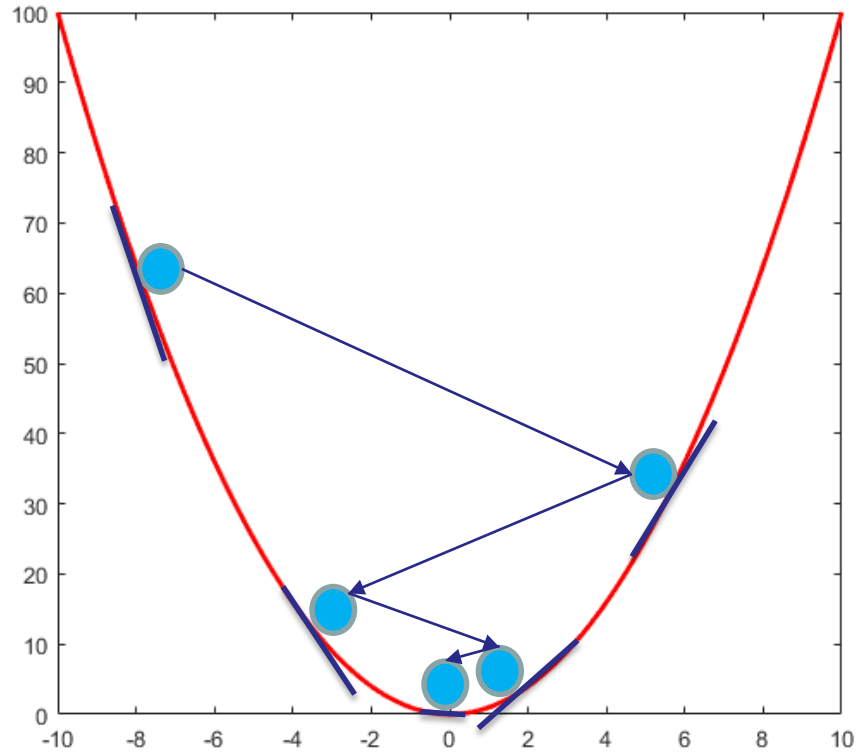
Gradient Descent

- How we minimise the cost functions, so we reach the optimum values for the weights.
  - Brute force: try millions of weights values and choose the one that gives the best results. (Impossible; takes ages)
  - Gradient Descent:
    1. Take the derivative of the cost function
    2. Compute the slope direction

Gradient Descent

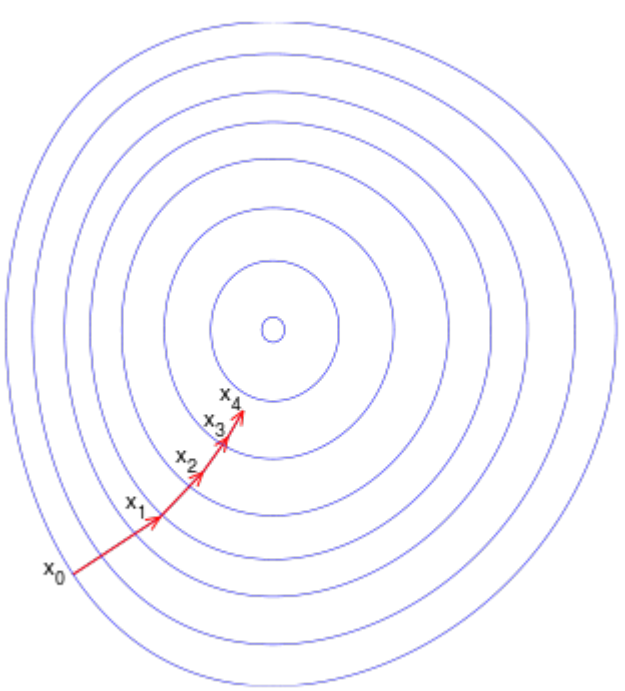
1. Take Derivative
2. Compute the slope direction.
   - Positive: Right is downhill and left is uphill -> Move right
   - Negative: Right is uphill and left is downhill -> Move left

We just watch which direction it is slopping-



1-D view of the Gradient Descent

Gradient Descent



2-D view of the Gradient Descent

Picture is from Wikipedia

Gradient Descent

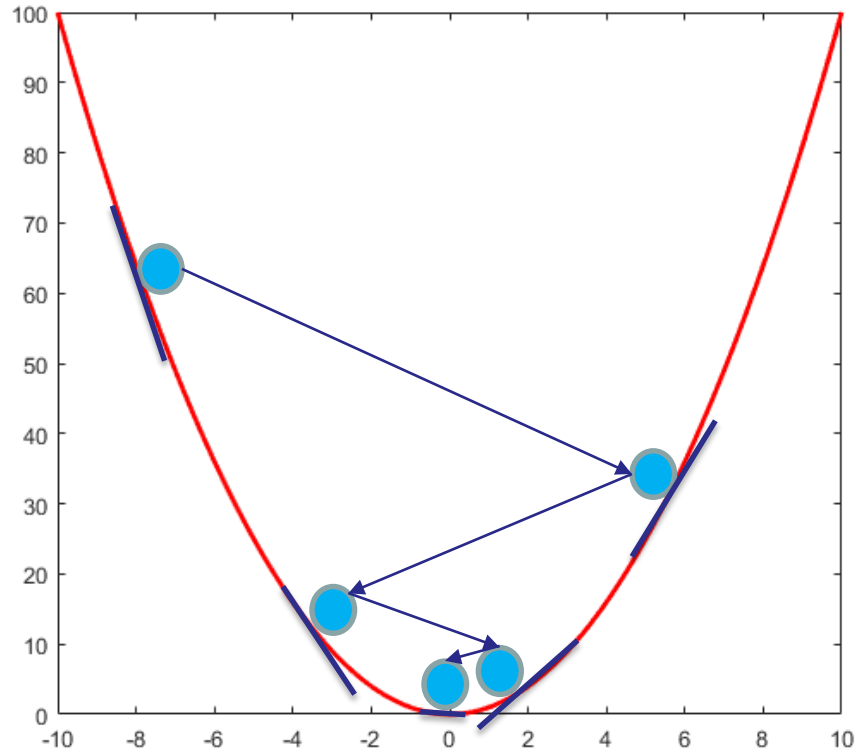
$$w_{n+1} = w_n - \gamma \nabla F(w_n)$$

The weights are updated after a complete epoch.

Gradient Descent

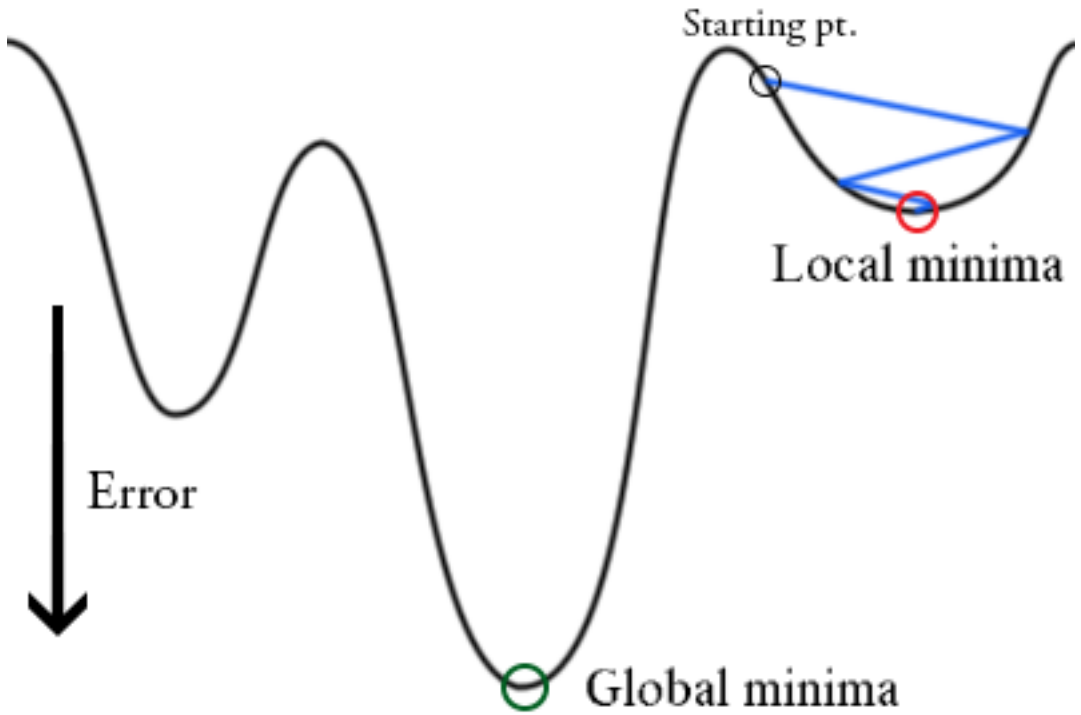
1. Take Derivative
2. Compute the slope direction.
   - Positive: Right is downhill and left is uphill -> Move right
   - Negative: Right is uphill and left is downhill -> Move left

GD deals with convex function only



1-D view of the Gradient Descent

Stochastic Gradient Descent

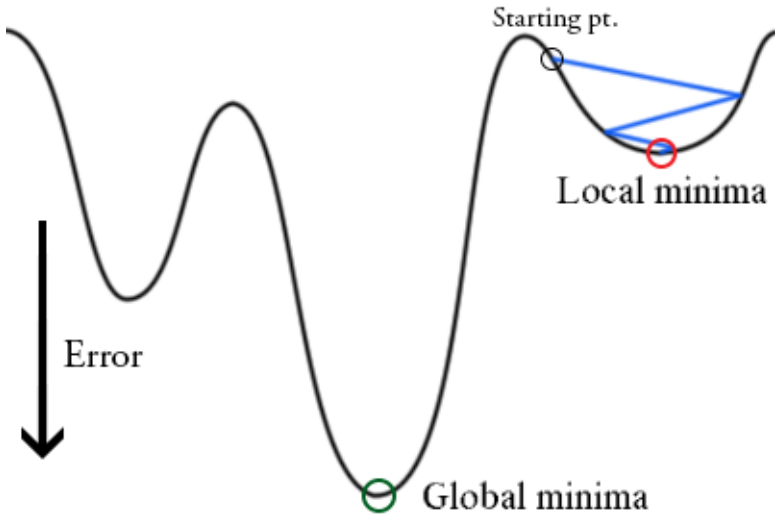


Picture is from "https://thinkingandcomputing.com/posts/genetic-algorithms-neural-networks.html"

Stochastic Gradient Descent

Steps of SGD:

- Choose an initial vector of parameters and learning rate
- Repeat until an approximate minimum is obtained:
  - Randomly shuffle examples in the training set.
  - For $i = 1, 2, \ldots, n$, do:
    - $w := w - \gamma \nabla F(w)$



The weights are updated after each iteration.

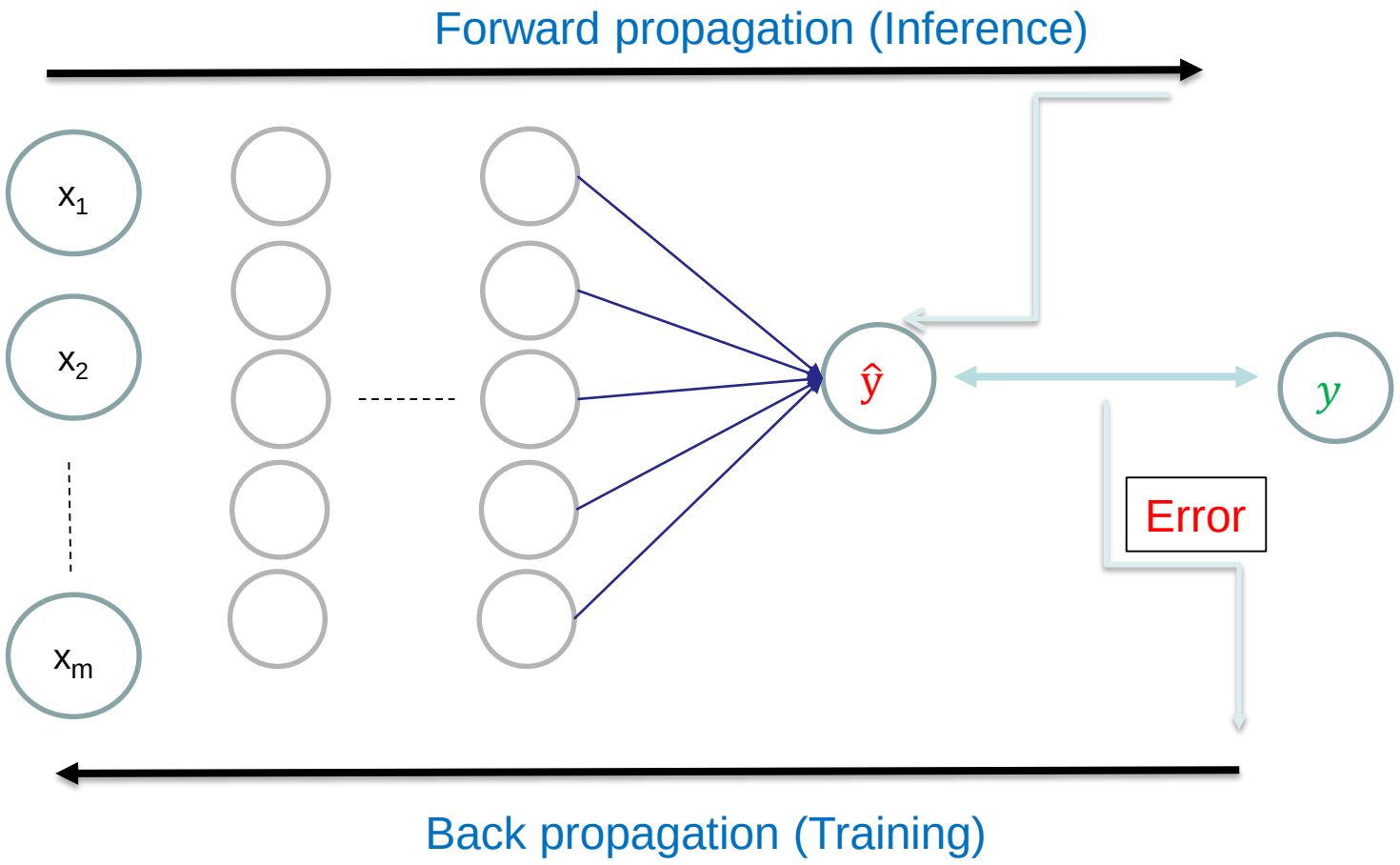
GD and SGD Differences

- In GD, the weights are updated after running on a full batch.
- In SGD, we go row-by-row and the weights are adjusted every iterations.
- SGD avoids falling into local minima while GD is not.
- GD is deterministic and SGD is stochastic.
- SGD is faster because we don’t need to load the whole training sequences into the memory.

GD and SGD – Additional Readings

- *"A neural network in 13 lines of python code – part 2 – Gradient Descent", Article, you can google it.*
- *"Neural Networks & Deep Learning" book, (Michael Nelsen, 2015). Read chapter 2.*

Back Propagation



ANN Training Steps

1. *Randomly, initialise the weights with small numbers (close to 0, but not 0).*
2. *Input the observations of your datasets in the input layer, each feature to one input node.*
3. *Do forward propagation until getting* $\hat{y}$.
4. Compare $\hat{y}$ and y and compute the cost function.
5. Do back propagation and update weights based on the propagated error.
6. Repeat steps 1 – 5 until convergence, normally after many epochs.

Let us Practice!