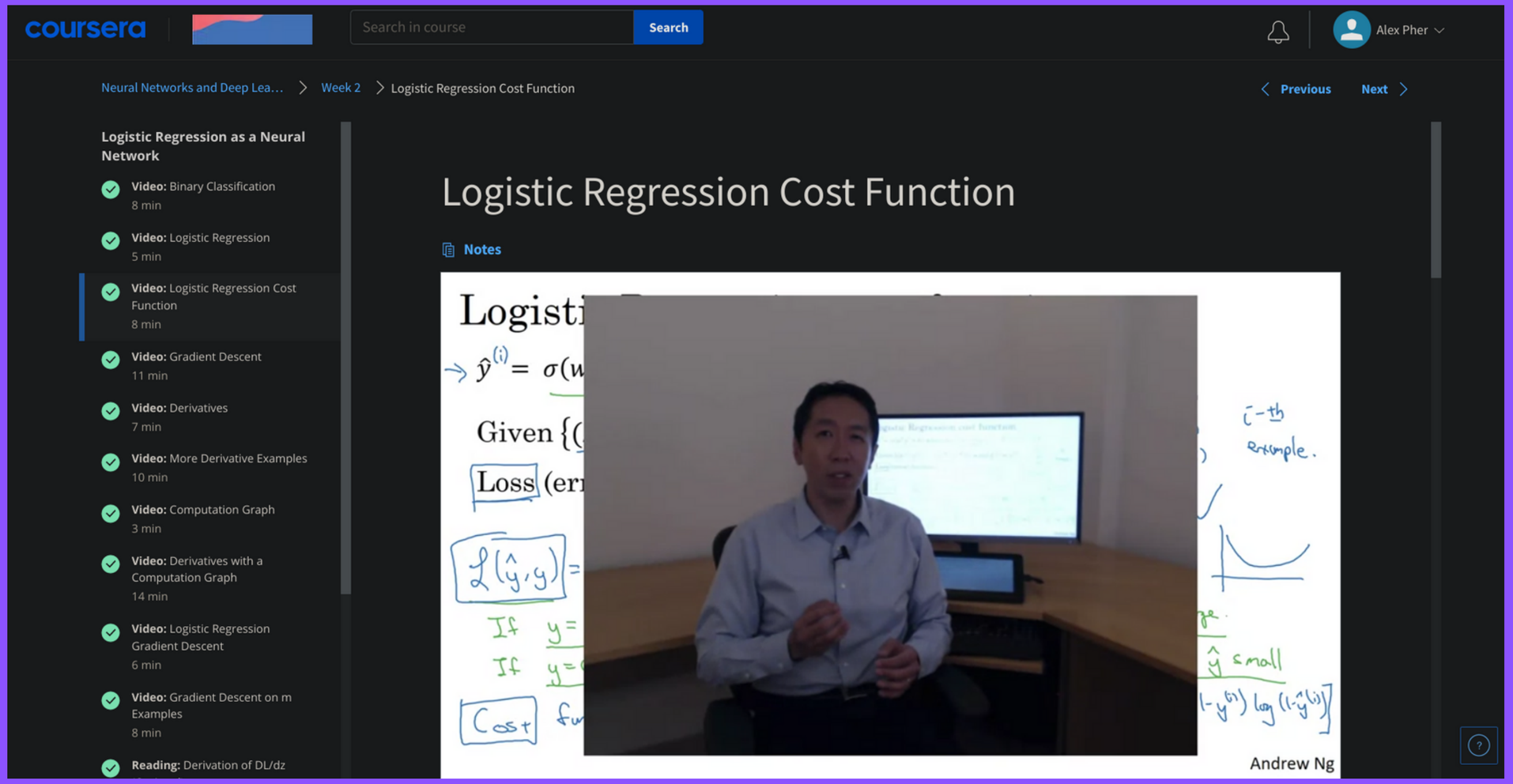
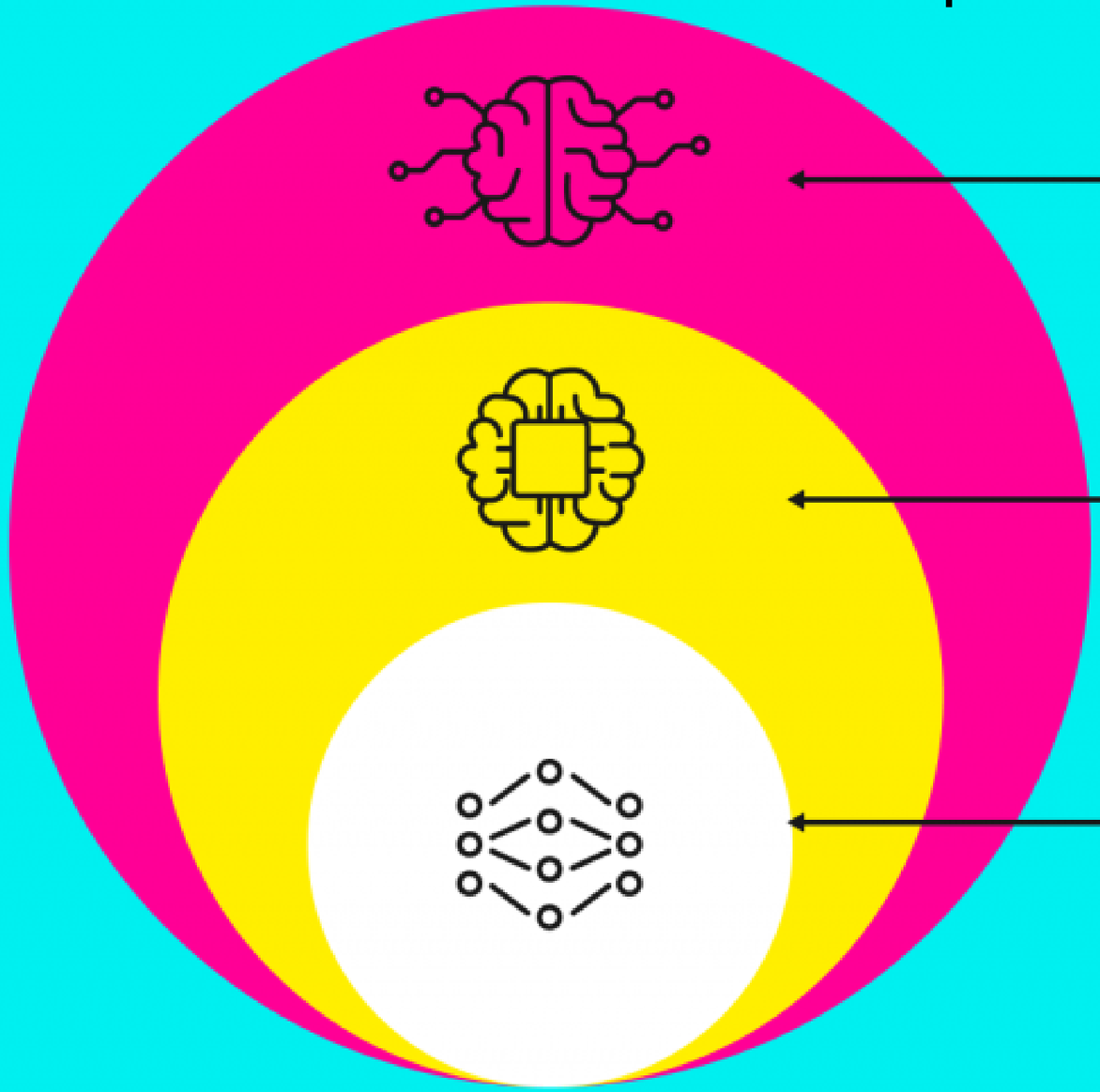
AI: LOGISTIC REGRESSION ON MOLECULAR DYNAMICS

Alessandro Michele Ferrara

DISCLAIMER: some images used in this presentation were taken from Andrew Ng's course on Deep Learning and Neural Networks on Coursera, but are free to use in a divulgative and non-commercial way.



What's the difference between AI in general, Machine Learning and Deep Learning?



Artificial Intelligence (AI)

Any process where we use computers to mimic the cognitive abilities and functions of humans.

Machine Learning (ML)

A subset of AI. It focuses on the ability of machines to receive data and improve at a task with experience.

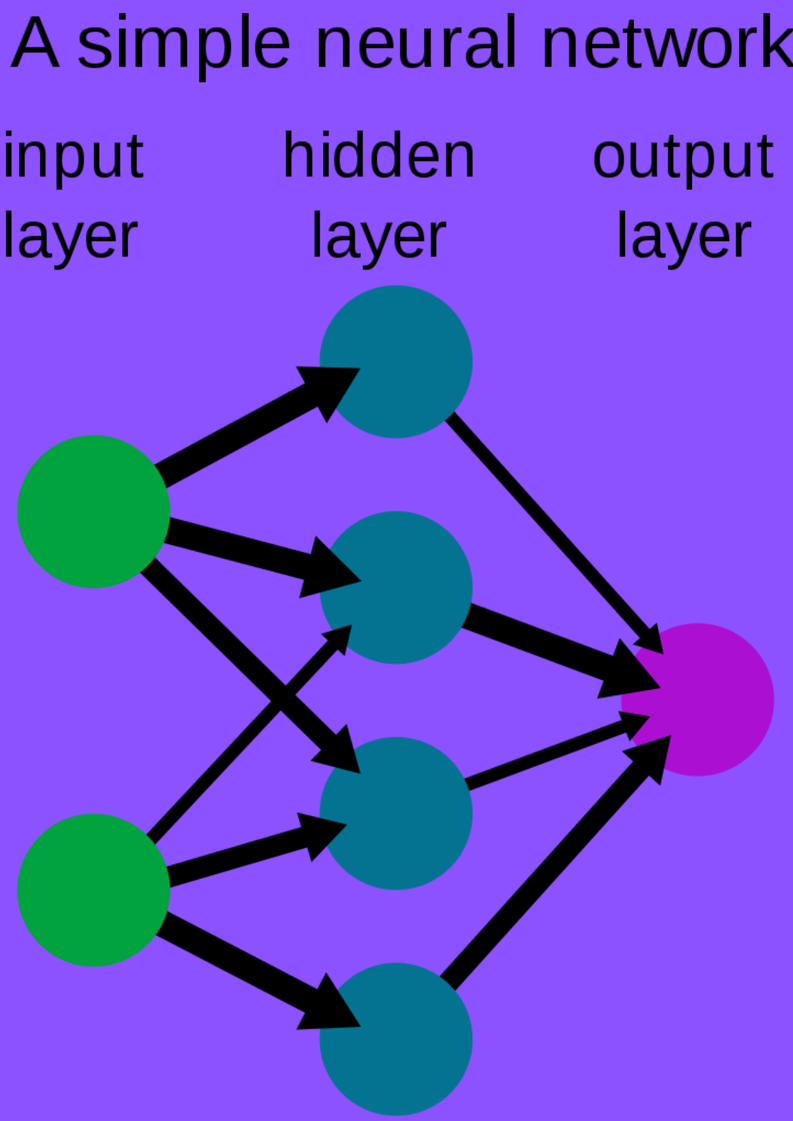
Deep Learning (DL)

A subset of ML. It works with multi-layered neural networks that can learn by themselves from vast amounts of data.

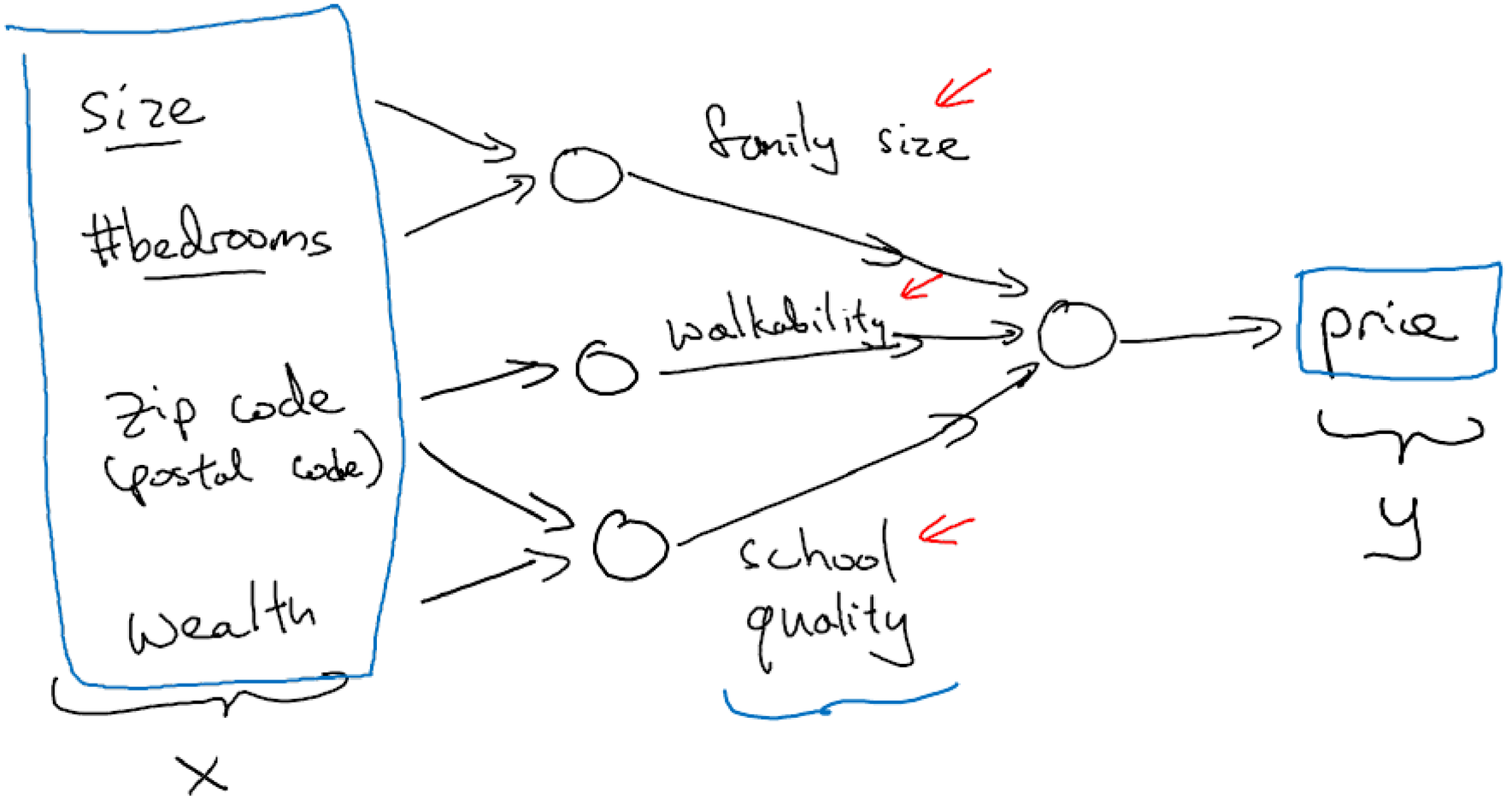
Deep learning ⟷ Neural Network

But what's a Neural Network?

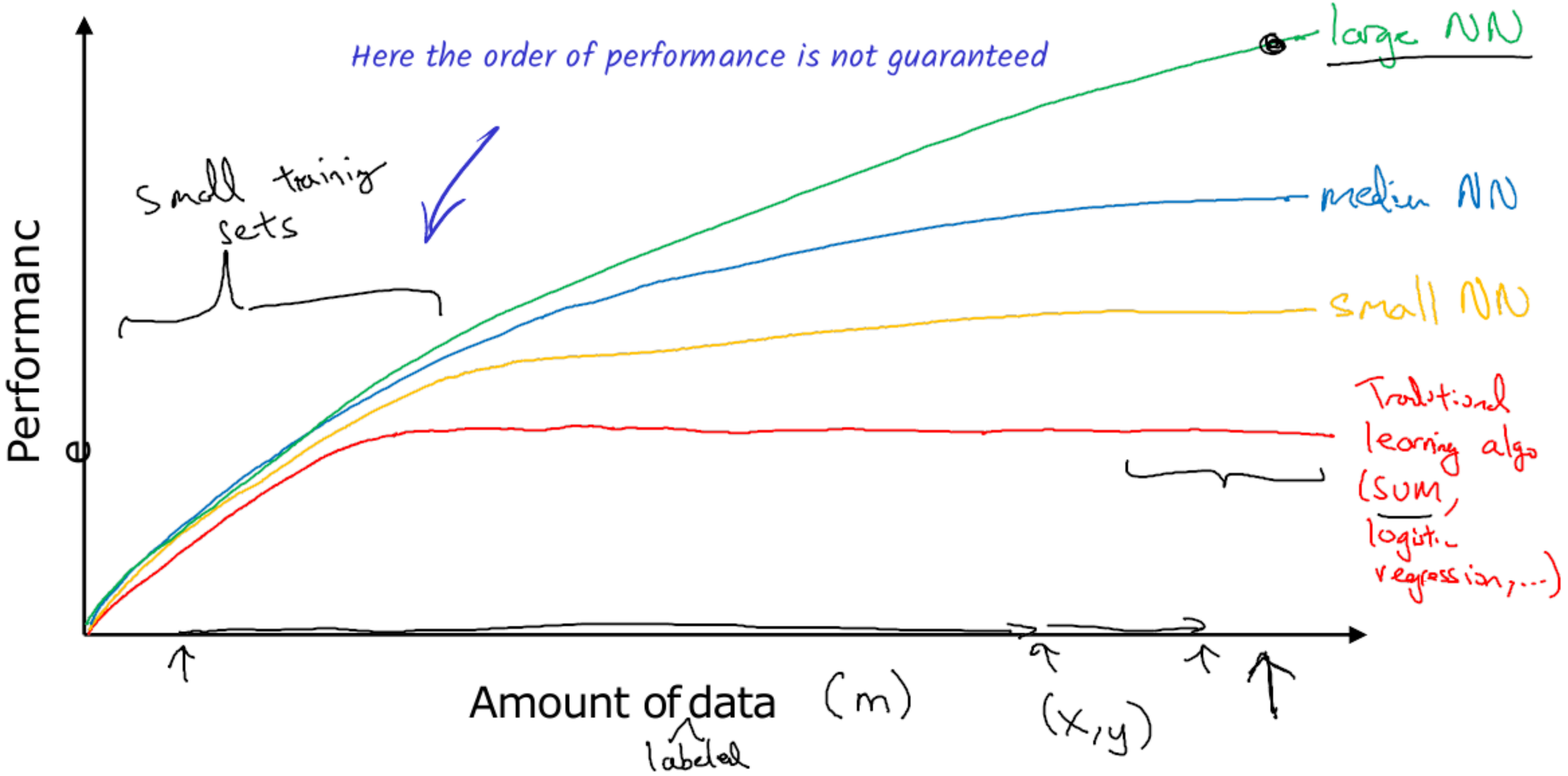
A Neural Network works mimicking the process that our brains use to process informations



Housing Price Prediction



Scale drives deep learning progress



Supervised Learning

Input(x)	Output (y)	Application
Home features	Price	Real Estate
Ad, user info	Click on ad? (0/1)	Online Advertising
Image	Object (1,…,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image, Radar info	Position of other cars	Autonomous driving

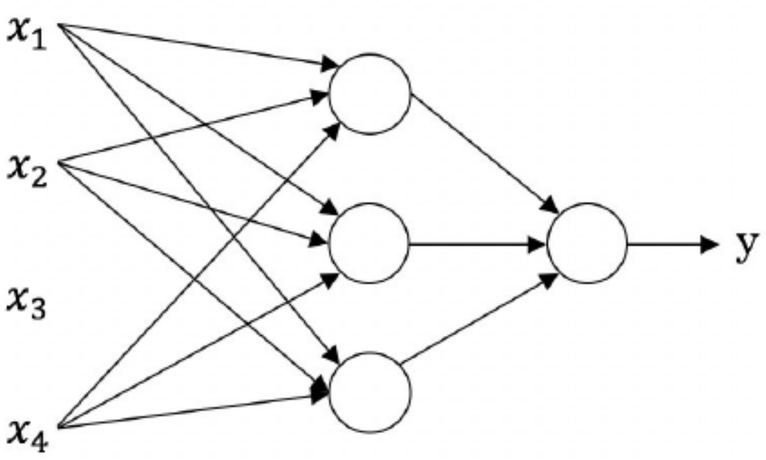
Standard NN

CNN

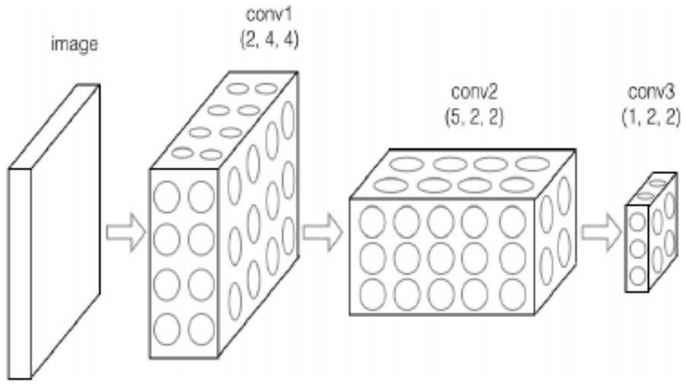
RNN

Custom/ Hybrid

What kind of Neural Network have I built?

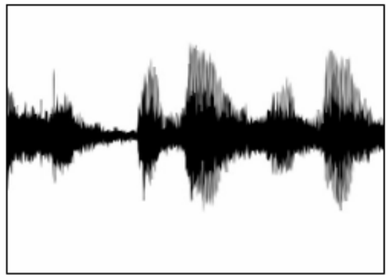


Standard NN



Convolutional NN

Unstructured Data



Audio

Image

Four scores and seven years ago…

Text

Binary Classification

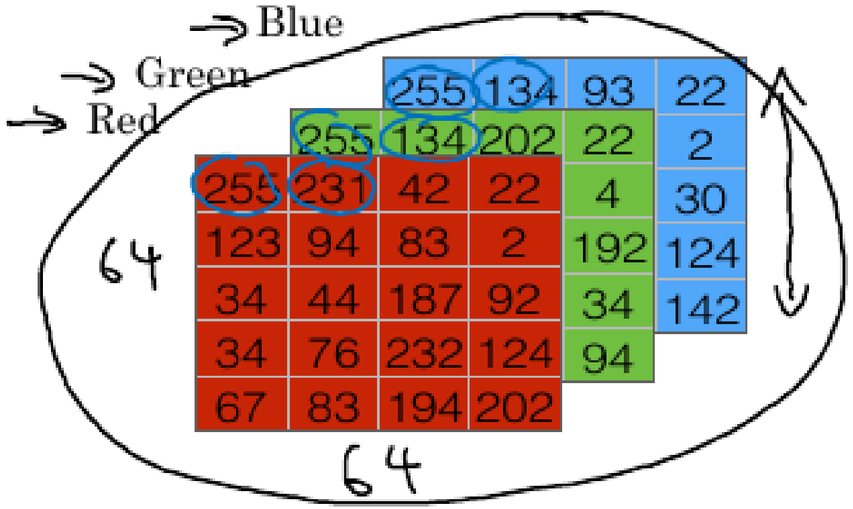
64

64

1 (cat) vs 0 (non cat)

y

we make a single vector!



$$x = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ \vdots \\ 255 \\ 134 \\ \vdots \end{bmatrix}$$

$64 \times 64 \times 3 = 12288$

$n = n_x = 12288$

$x \longrightarrow y$

A good knowledge and practice with tensors is essential in building and understanding neural networks.

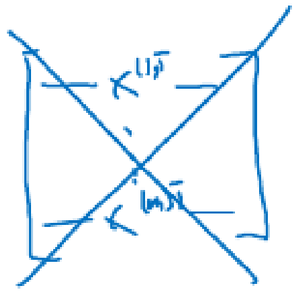
Notation

(x, y) $\quad x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$

m training examples: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

$m = m_{train}$ $\qquad$ m_{test} = #test examples.

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad n_x \text{ (rows)}, \quad m \text{ (columns)}$$



$X \in \mathbb{R}^{n_x \times m}$ $\qquad$ $X.shape = (n_x, m)$

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

$Y \in \mathbb{R}^{1 \times m}$

$Y.shape = (1, m)$

How does this work?

Logistic Regression

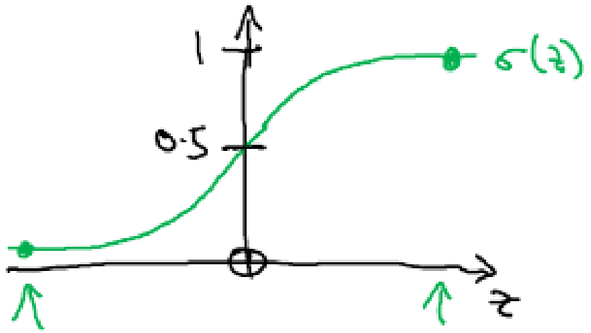
Given x, want $\hat{y} = P(y=1|x)$

$0 \le \hat{y} \le 1$

$x \in \mathbb{R}^{n_x}$

Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$.

Output $\hat{y} = \sigma(\underbrace{w^T x + b}_{z})$



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

If z large $\sigma(z) \approx \frac{1}{1+0} = 1$

If z large negative number

$$\sigma(z) = \frac{1}{1+e^{-z}} \approx \frac{1}{1+\text{Bignum}} \approx 0$$

We want a metric on how well our network learn:

Logistic Regression cost function

$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$, where $\sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$

$z^{(i)} = w^T x^{(i)} + b$

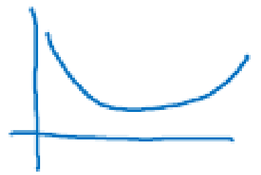
$x^{(i)}$, $y^{(i)}$, $z^{(i)}$ — i-th example.

Given $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$.

Loss (error) function:

$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$ not good

$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$ ← good



If $y=1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y}$ ← want $\log \hat{y}$ large, want $\hat{y}$ large.

If $y=0$: $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$ ← want $\log 1-\hat{y}$ large .... want $\hat{y}$ small

Cost function: $J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \right]$

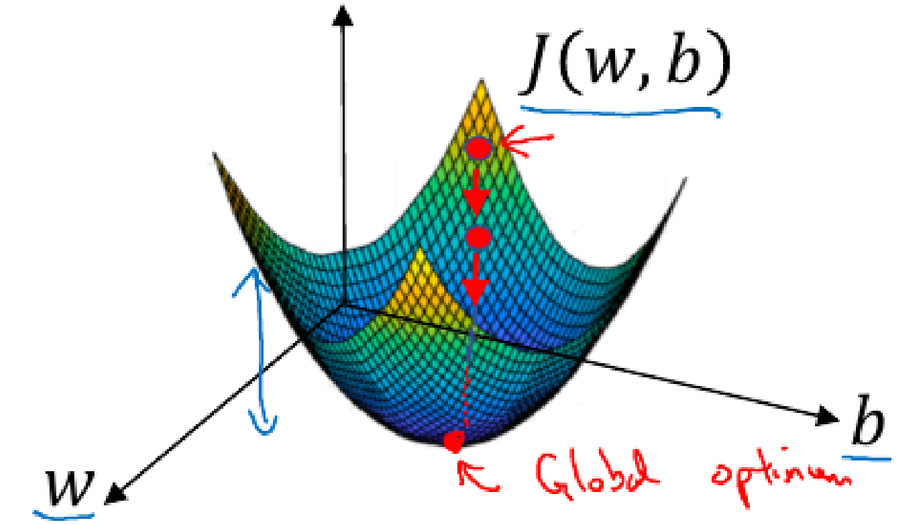
How do we minimize a n-parameter function?

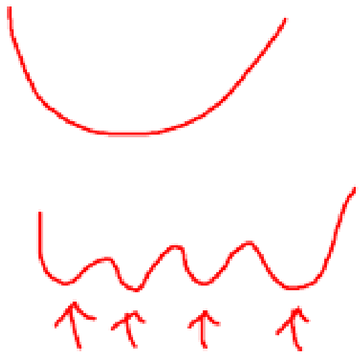
Gradient Descent

Recap: $\hat{y} = \sigma(w^T x + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$

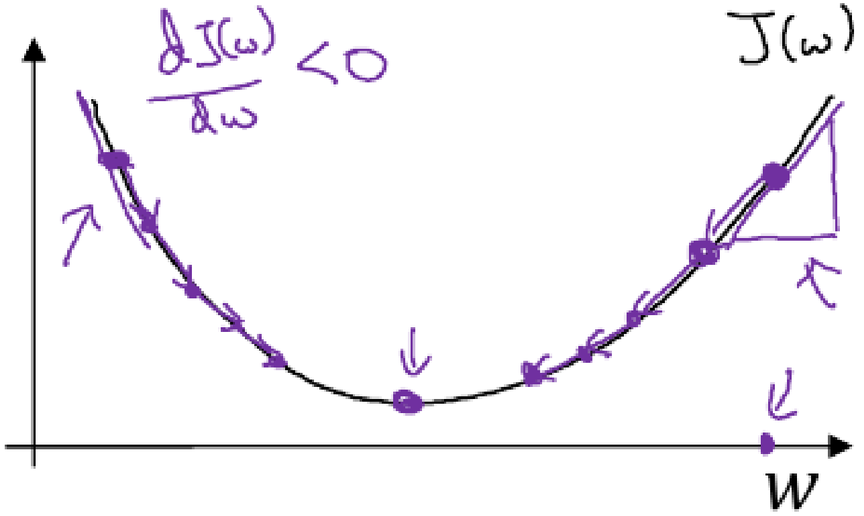
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

Want to find w, b that minimize $J(w,b)$





Gradient Descent



Repeat {

$w := w - \alpha \frac{dJ(w)}{dw}$

learning rate

"dw"

}

$w := w - \alpha dw$

$\frac{dJ(w)}{dw} = ?$

$J(w,b)$

$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$

$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$

$\frac{\partial J(w,b)}{\partial w}$ → dw

$\frac{\partial J(w,b)}{\partial b}$ → db

∂ "partial derivative"

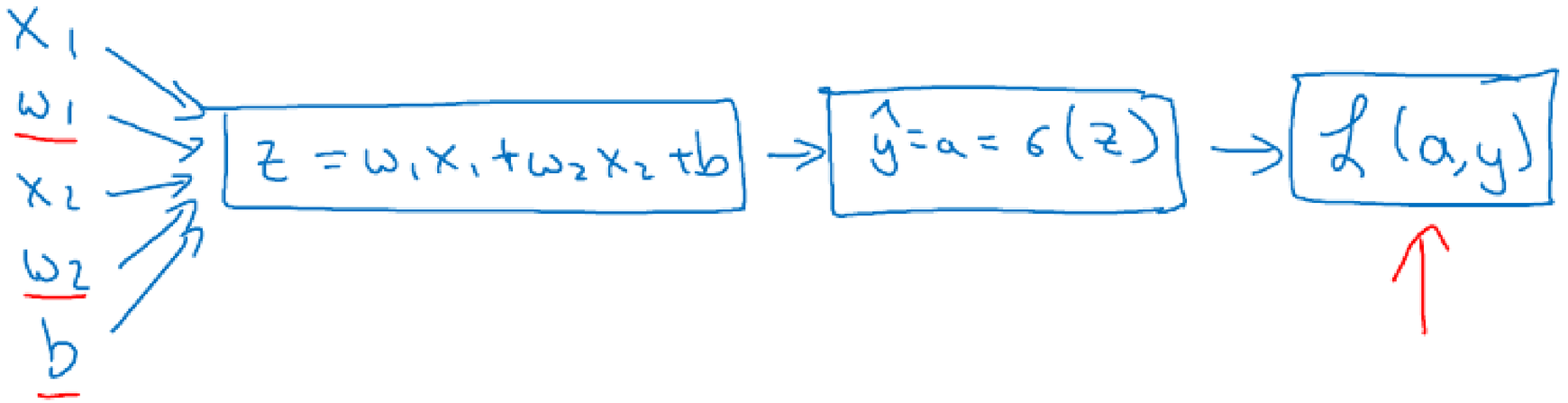
J

Logistic regression recap

$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z)$$

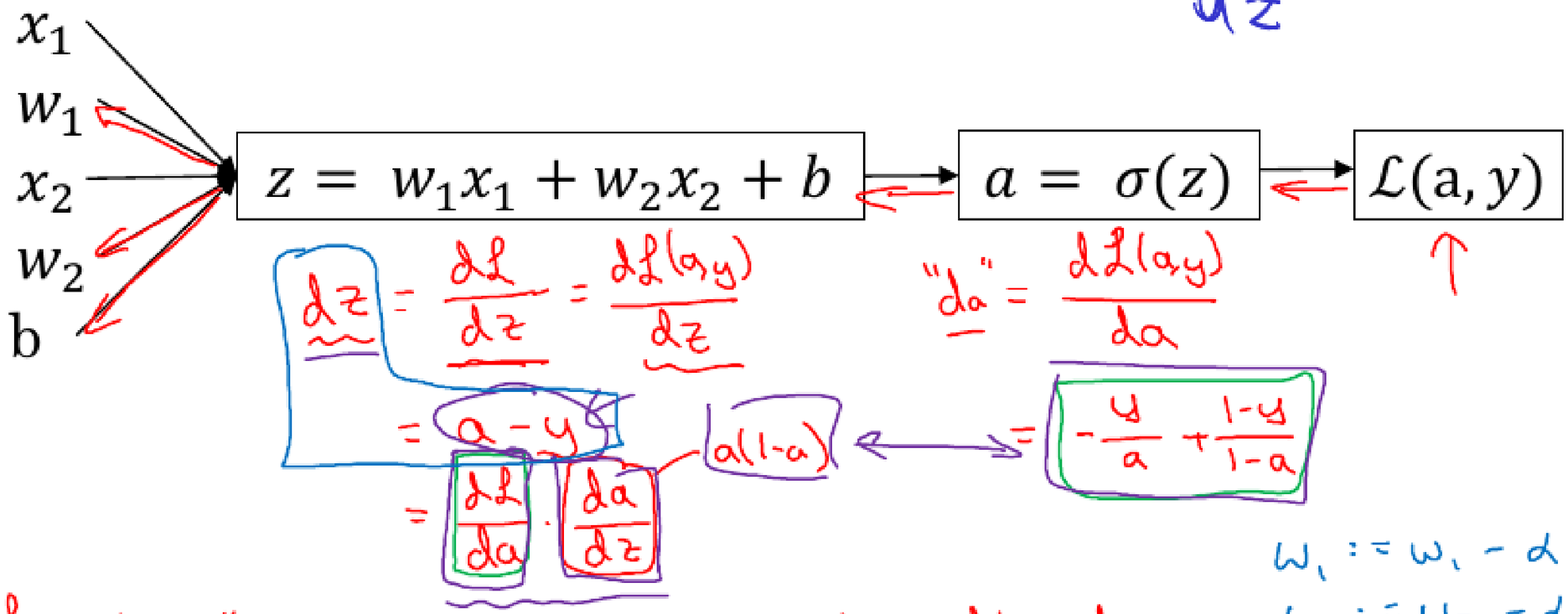
$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$



Derivatives in this LR are easy and analitically avaiable

Logistic regression derivatives

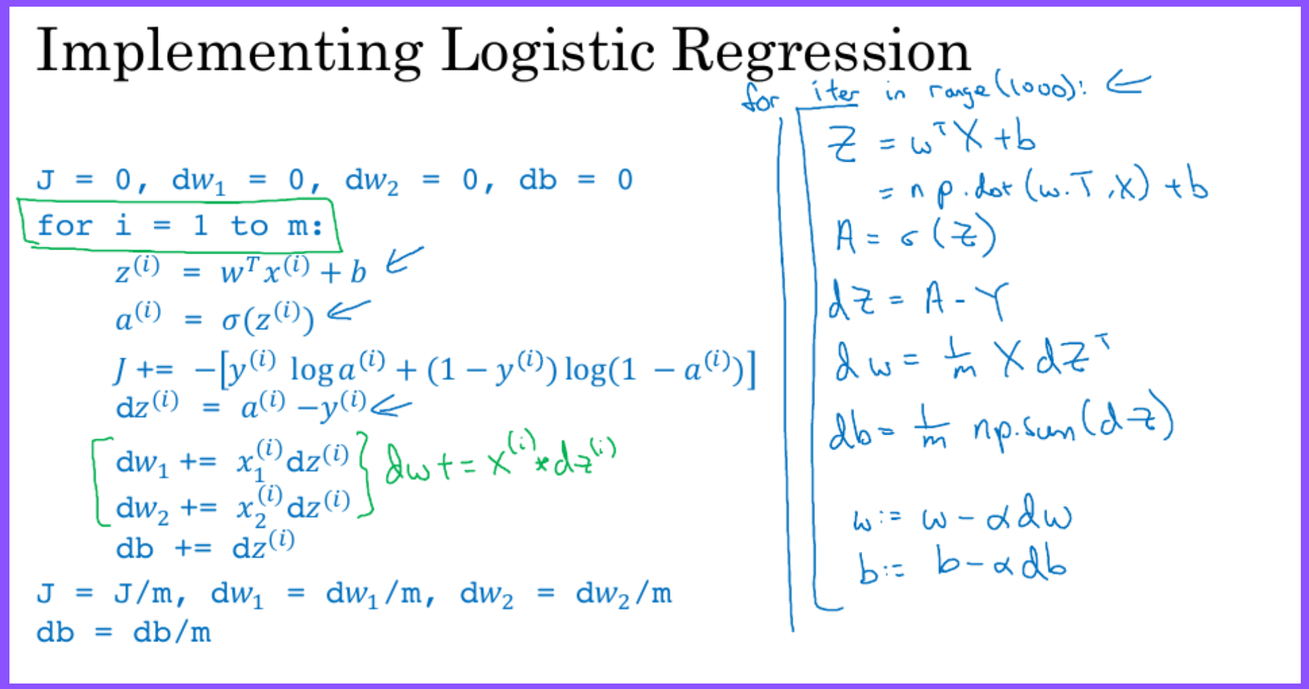
$$\frac{d\,\sigma(z)}{dz} = \sigma(z)\left(1-\sigma(z)\right)$$

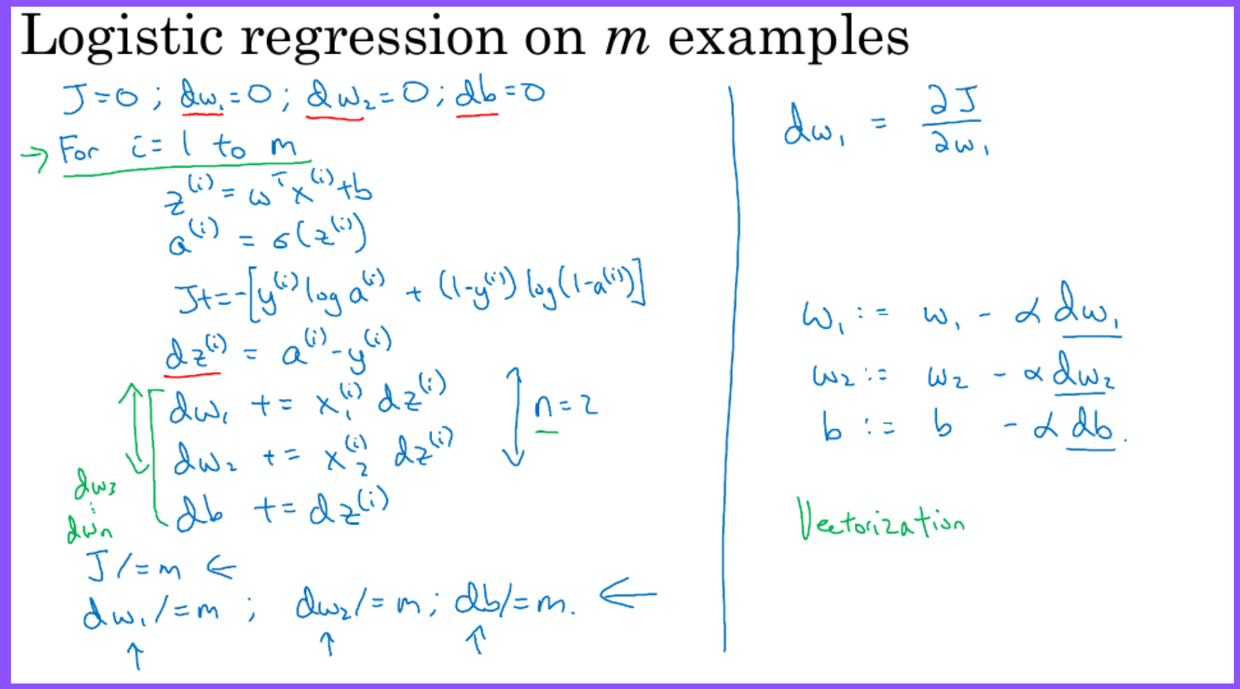


$$\frac{\partial \mathcal{L}}{\partial w_1} = \text{"}dw_1\text{"} = x_1 \cdot dz \qquad dw_2 = x_2 \cdot dz \qquad db = dz$$

$$w_1 := w_1 - \alpha\, dw_1$$
$$w_2 := w_2 - \alpha\, dw_2$$
$$b := b - \alpha\, db$$

When creating a Neural Network, the most challenging thing is to not get lost with tensor operations, while using vectorization over loops whenever possible.





Vectorization works so well because of python's C-written libraries, like numpy.

[1]:
```
import numpy as np
import time
```

[2]:
```
a = np.random.rand(10000000)
b = np.random.rand(10000000)
```

We want to calculate the scalar product between this two 10 million long vectors.

We first try the normal for loop version (it takes about 5 seconds in a ~2015 dual-core notebook)

[3]:
```
c = 0
tic = time.time()

for i in range(len(a)):
    c += a[i] * b[i]

toc = time.time()

t1= toc - tic

print("Normal for loop version: " + str(1000*t1) + "ms")
```

```
Normal for loop version: 4577.723503112793ms
```

Now we try a vectorized method

[4]:
```
tic = time.time()
c = np.dot(a,b)      #this makes a scalar product
toc = time.time()

t2= toc - tic

print("Vectorized version: " + str(1000*t2) + "ms")
```

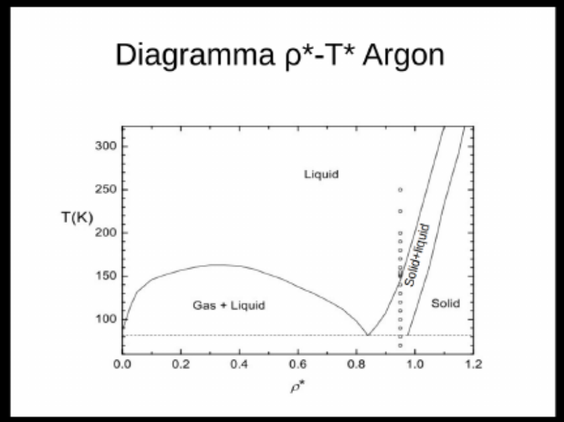
```
Vectorized version: 21.102190017700195ms
```

[5]:
```
print("The vectorized version was " +str(t1/t2) + " times faster, a " + str(int(100*t1/t2)) + "% increase in performance")
```

```
The vectorized version was 216.9312047362415 times faster, a 21693% increase in performance
```

As we can see, the vectorized method is much faster!

But what was this NN trained for? On molecular dynamics simulations of Argon!

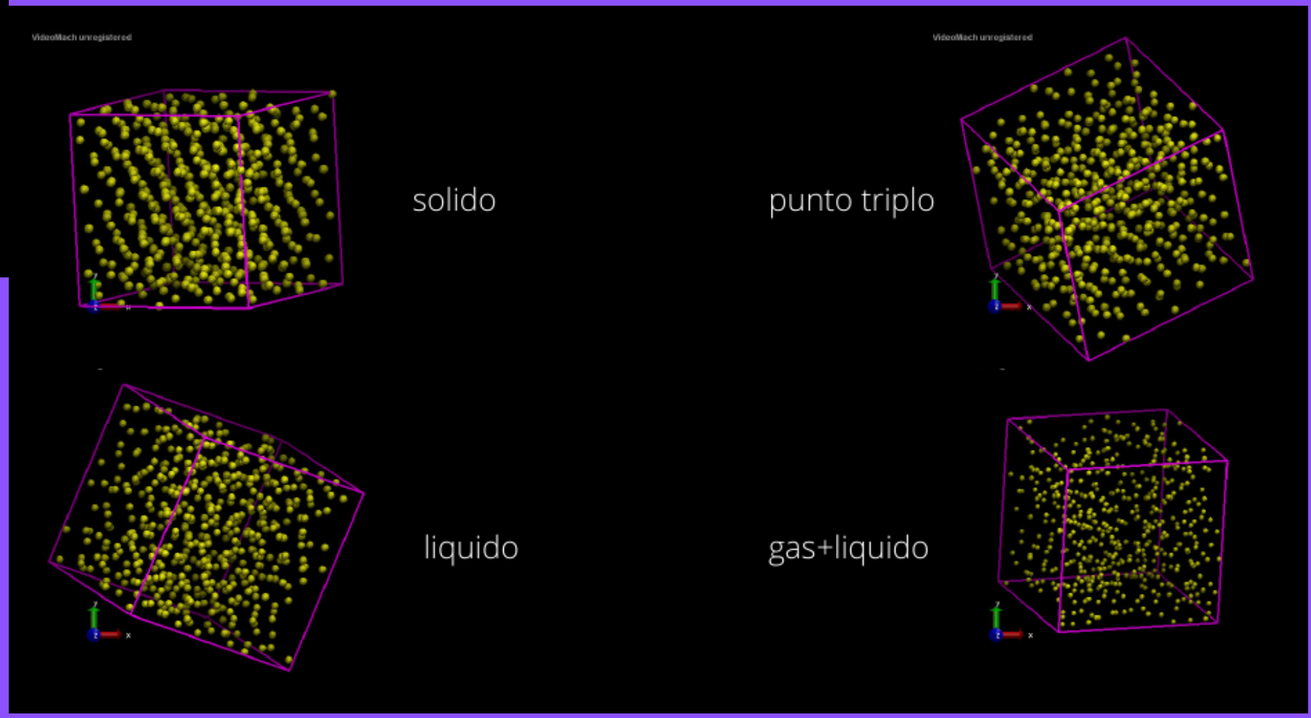


Ho selezionato 4 punti del diagramma delle fasi in modo da ottenere:
gas+liquido:T=1, rho=0.3
liquido:T=2, rho=0.7
solido:T=1, rho=1.1
punto triplo:T=0.76, rho=0.84
La temperatura del termometro è stata scelta uguale a quella iniziale

The cutoff radius was 3.5 this time

Tutte le simulazioni sono avvenute utilizzando il termometro di Andersen per 512 particelle, con raggio di cutoff pari a 2.5, sono sempre state utilizzate le unità ridotte:

Physical quantity	Unit	Value for Ar
length	σ	3.4×10^{-10} m
energy	ε	1.65×10^{-21} J
mass	m	6.69×10^{-26} kg
time	$\sigma(m/\varepsilon)^{1/2}$	2.17×10^{-12} s
velocity	$(\varepsilon/m)^{1/2}$	1.57×10^{2} m/s
force	ε/σ	4.85×10^{-12} N
pressure	ε/σ^3	4.20×10^{7} N·m^{-2}
temperature	ε/k_B	120 K



I didn't put the box or the axis, to have cleaner images

My work environment while performing MD simulations:

```
drwxr-xr-x 1 pher pher 8.8K Jan  8 23:18 ro0.3_t1/
drwxr-xr-x 1 pher pher 8.8K Jan  7 00:52 ro0.7_t2/
drwxr-xr-x 1 pher pher 8.8K Jan  7 01:05 ro0.84_t0.76/
drwxr-xr-x 1 pher pher 8.8K Jan  9 01:18 ro1.1_num2/
drwxr-xr-x 1 pher pher 8.8K Jan  9 01:22 ro1.1_num3/
drwxr-xr-x 1 pher pher 8.8K Jan  9 01:26 ro1.1_num4/
drwxr-xr-x 1 pher pher 8.8K Jan  9 01:31 ro1.1_num5/
drwxr-xr-x 1 pher pher 8.8K Jan  7 00:49 ro1.1_t1/
-rwxr-xr-x 1 pher pher  591 Jan  9 01:01 scriptino.sh*
-rwxr-xr-x 1 pher pher   66 Jan  8 23:31 traiettoria.sh*
pher@penguin:~/Documents/computational/dinamica_molecolare_work/2023$ bat scriptino.sh
```

File: **scriptino.sh**

```
1  cd ro0.3_num2 ; bash MDT.sh ; bash gdr.sh ; bash traiettoria.sh ; cd .. ;
2  cd ro0.3_num3 ; bash MDT.sh ; bash gdr.sh ; bash traiettoria.sh ; cd .. ;
3  cd ro0.3_num4 ; bash MDT.sh ; bash gdr.sh ; bash traiettoria.sh ; cd .. ;
4  cd ro0.3_num5 ; bash MDT.sh ; bash gdr.sh ; bash traiettoria.sh ; cd .. ;
5  cd ro1.1_num2 ; bash MDT.sh ; bash gdr.sh ; bash traiettoria.sh ; cd .. ;
6  cd ro1.1_num3 ; bash MDT.sh ; bash gdr.sh ; bash traiettoria.sh ; cd .. ;
7  cd ro1.1_num4 ; bash MDT.sh ; bash gdr.sh ; bash traiettoria.sh ; cd .. ;
8  cd ro1.1_num5 ; bash MDT.sh ; bash gdr.sh ; bash traiettoria.sh ; cd ..
```

```
pher@penguin:~/Documents/computational/dinamica_molecolare_work/2023$
```

```
drwxr-xr-x 1 pher pher 8.8K Jan  7 00:52 ro0.7_t2/
drwxr-xr-x 1 pher pher 8.8K Jan  7 01:05 ro0.84_t0.76/
drwxr-xr-x 1 pher pher 8.8K Jan  9 01:18 ro1.1_num2/
drwxr-xr-x 1 pher pher 8.8K Jan  9 01:22 ro1.1_num3/
drwxr-xr-x 1 pher pher 8.8K Jan  9 01:26 ro1.1_num4/
drwxr-xr-x 1 pher pher 8.8K Jan  9 01:31 ro1.1_num5/
drwxr-xr-x 1 pher pher 8.8K Jan  7 00:49 ro1.1_t1/
-rwxr-xr-x 1 pher pher  591 Jan  9 01:01 scriptino.sh*
-rwxr-xr-x 1 pher pher   66 Jan  8 23:31 traiettoria.sh*
pher@penguin:~/Documents/computational/dinamica_molecolare_work/2023$ bat istruzioni
```

File: **istruzioni**

```
1  ./MDT.exe -N 512 -rho 1.1 -dt 0.001 -rc 2.5 -ns 50000 -T0 1 -Tb 1 -nu 1 -seed
   299792458 -fs 100 > output &
2
3
4  ./gdr.exe -fnf %i.xyz -for 20000,49900,100 -N 512 -rc 2.5 -dr 0.02 > gdrfile
5
6
7  for i in {0..49900..100}.xyz ; do cat $i >> traiettoria.xyz; done
8
```

```
pher@penguin:~/Documents/computational/dinamica_molecolare_work/2023$
```

```
   (assumes the GNU Scientific Library is installed)

   You must have the GNU Scientific Library installed; see
   the coursenotes to learn how to do this.

   Drexel University, Department of Chemical Engineering
   Philadelphia
   (c) 2004
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>

/* Prints usage information */
void usage ( void ) {
  fprintf(stdout,"mdlj_and usage:\n");
  fprintf(stdout,"mdlj_and [options]\n\n");
  fprintf(stdout,"Options:\n");
  fprintf(stdout,"\t -N [integer]\t\tNumber of particles\n");
  fprintf(stdout,"\t -rho [real]\t\tNumber density\n");
  fprintf(stdout,"\t -dt [real]\t\tTime step\n");
:
```

```
pher@penguin:~/Documents/computational$ bat istruzioni_convertire_frames
```

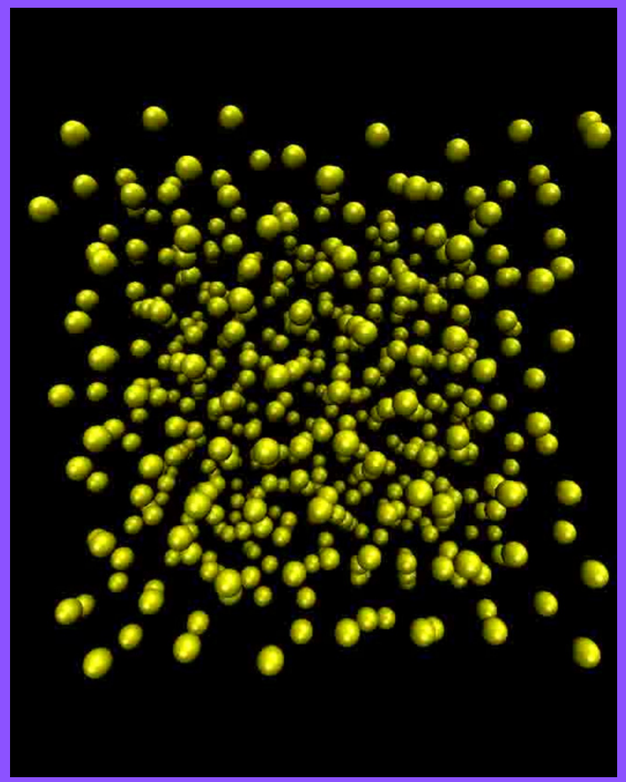
File: **istruzioni_convertire_frames**

```
1  ffmpeg -i input.mpg '%04d.jpg'
2
3  Chiaramente al posto di input.mpg si mette il nome del file video.
```

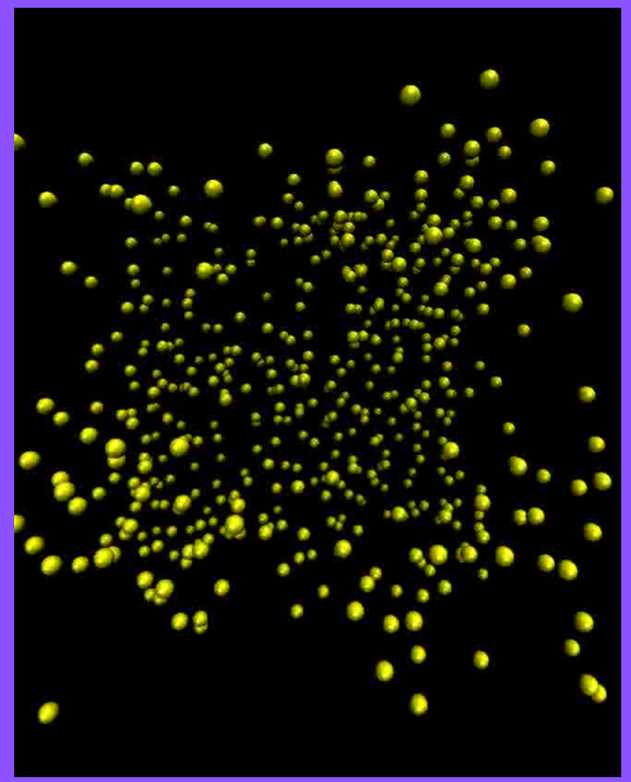
```
pher@penguin:~/Documents/computational$
```

```
[0] 0:bash*                                                    "pher@penguin: ~/Docum" 01:19 25-Jan-23
```

I simulated a solid state (rho 1.1, T=1) and a gas+liquid state (rho 0.3, T=1) multiple times, then extracted the frames from the last 80% of the simulation from VMD and trained the NN to differentiate between the two states.



Can you differentiate, visually, between this two states? If so, the NN can learn to do it as well!

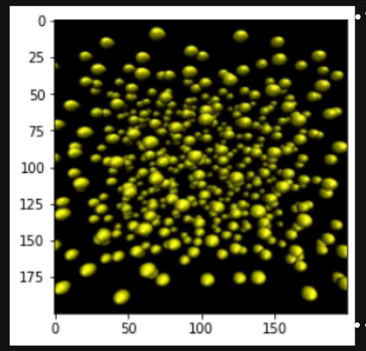


I loaded the images in my code using PIL (Python Image Library), and I was able to visualize them with matplotlib.pyplot.
I used a resolution of 200x200 pixel because it was enough to distinguish between solid and gas+liquid, considering in the RGB protocol every image has 3 layers, this accounts for a total of 120'000 values (between 0 and 1, from darker to brightest) per image.

[1]:
```
import random
import numpy as np
import matplotlib.pyplot as plt
import scipy
import math
import copy
from PIL import Image
```

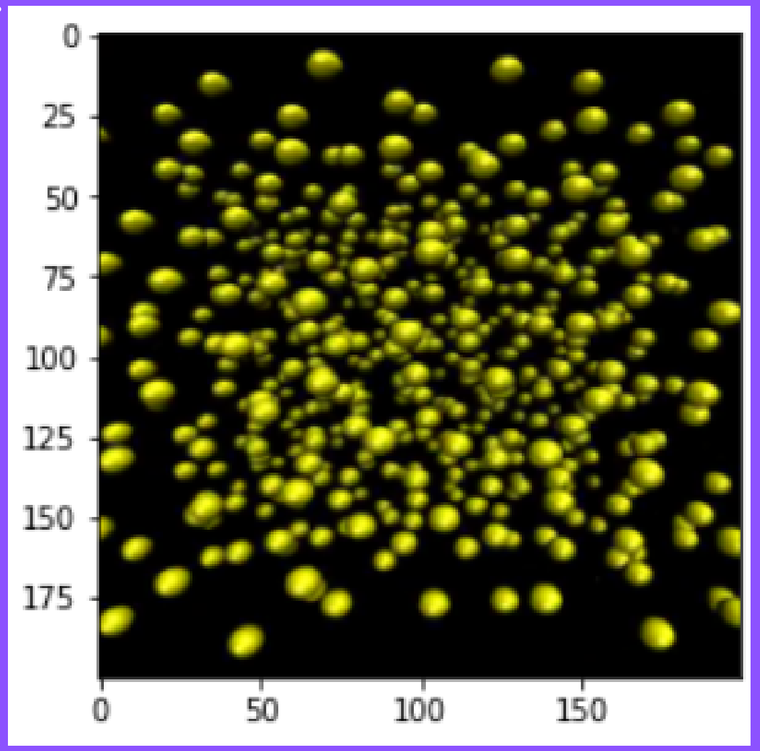
[2]:
```
num_px = 200 #this is very important! it's the dimension of the images!
```

[4]:
```
# change this to the name of the chosen image file
my_image = "0012.jpg"

# We preprocess the image to fit the algorithm.
fname = "/home/pher/Documents/computational/newmovies/frames/1.1_2/" + my_image
image = np.array(Image.open(fname).resize((num_px, num_px)))
plt.imshow(image)

image = image / 255.
image = image.reshape((1, num_px * num_px * 3)).T
```



[5]:
```
print(image.shape)
```

```
(120000, 1)
```



A solid state image

I obtained a total of 5000 frames from 10 different simulations of trajectories with both rho 0.3 and rho 1.1.
As the train set I choose to use 1 frame every 10, but just after the 101th so the system would already be evolved in a solid or gas+liquid state. The images were 50% split between solid and gas+liquid, for a total of 400 images.
As the test set, I used an image every 5 frames from the 51th to the 100th frame, for a total of 110 images.

```
start = 101
stop = 500
step = 10
number_movies = 10 # the number of movies in the simulation

m = (math.floor((stop-start)/step) + 1 ) * number_movies
print(m)
```

```
400
```

```
cartelle = ["1.1_3", "0.3", "1.1_4", "0.3_4", "1.1_5",  "0.3_2", "1.1", "0.3_3", "1.1_2", "0.3_5"]
solidi = ["1.1_3", "1.1_5", "1.1_4", "1.1", "1.1_2",]
randomici = [i for i in range(m)]

random.shuffle(randomici)
```

```
trainset_y = np.zeros([1,m])
```

```
print(trainset_y.shape)
```

```
(1, 400)
```

```
count = 0
x = 0
for cartella in cartelle:
    fnames = ['/home/pher/Documents/computational/newmovies/frames/' + cartella + '/{:04}.jpg'.format(i) for i in range(start, stop+1,
    if cartella in solidi:
        stato = 1
    else:
        stato = 0

    for i in fnames:
        j = randomici[x]
        x += 1
        fname = i
        image = np.array(Image.open(fname).resize((num_px, num_px)))
        image = image / 255.
        image = image.reshape((num_px * num_px * 3)).T
        trainset_x[:,j] = image
        trainset_y[0,j] = stato

    count += int(m/number_movies)
    print ("Test set acquired to {}/{}".format(count, m))
```

```
Test set acquired to 40/400
Test set acquired to 80/400
Test set acquired to 120/400
Test set acquired to 160/400
Test set acquired to 200/400
Test set acquired to 240/400
Test set acquired to 280/400
Test set acquired to 320/400
Test set acquired to 360/400
Test set acquired to 400/400
```

The code is easily scalable with only a few global variables to use more/less or different images. Also the images were loaded in a random order to avoid any bias from the NN.

```
print(trainset_x.shape)
```

```
(120000, 400)
```

My train set consisted of 48'000'000 million bit, if that seems much, it's only about 6 MB in storage.
The test set was smaller, just 1.65 MB.

```
0.0005
Cost after iteration 0: 0.693147
Cost after iteration 100: 0.477761
Cost after iteration 200: 0.379760
Cost after iteration 300: 0.312086
Cost after iteration 400: 0.263204
Cost after iteration 500: 0.226568
Cost after iteration 600: 0.198264
Cost after iteration 700: 0.175841
Cost after iteration 800: 0.157699
```

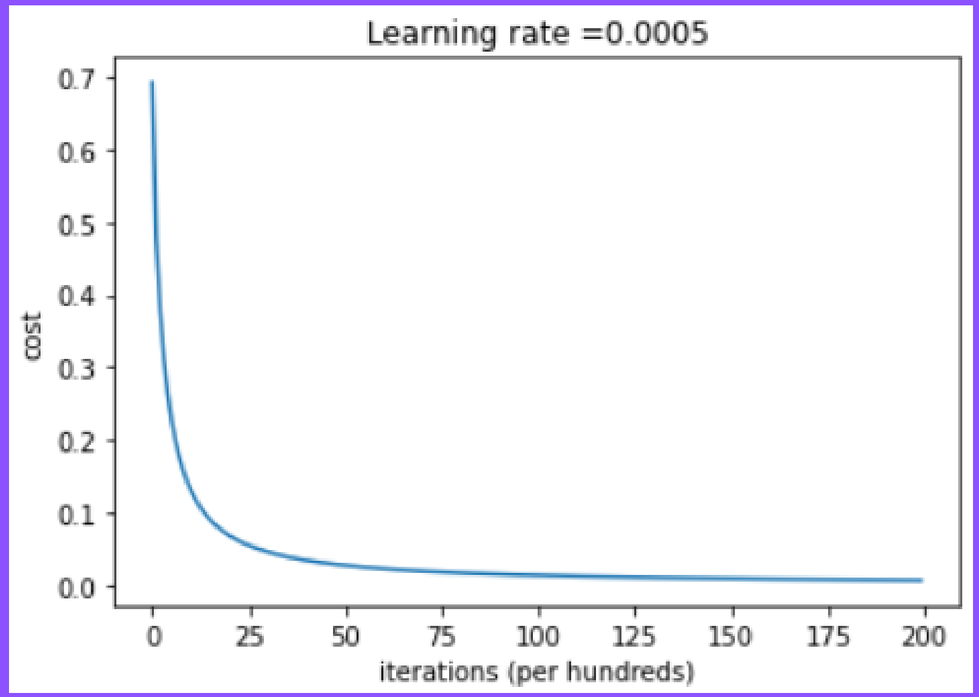
We can see that the model is working because the Cost function is consistently decreasing after every couple hundreds of iteration.

I trained the neural network with a 0.0005 learning step for 20000 iterations, so it processed about 9.6*10^11 bit of data.
The training process took about 40 minutes on a dual-core chromebook.

But how is this simple model performing?
Pretty well, actually, accuracy is close to 90%!

```
Cost after iteration 18700: 0.007068
Cost after iteration 18800: 0.007030
Cost after iteration 18900: 0.006993
Cost after iteration 19000: 0.006955
Cost after iteration 19100: 0.006918
Cost after iteration 19200: 0.006882
Cost after iteration 19300: 0.006845
Cost after iteration 19400: 0.006810
Cost after iteration 19500: 0.006774
Cost after iteration 19600: 0.006739
Cost after iteration 19700: 0.006704
Cost after iteration 19800: 0.006670
Cost after iteration 19900: 0.006636
train accuracy: 100.0 %
test accuracy: 88.18181818181819 %
```

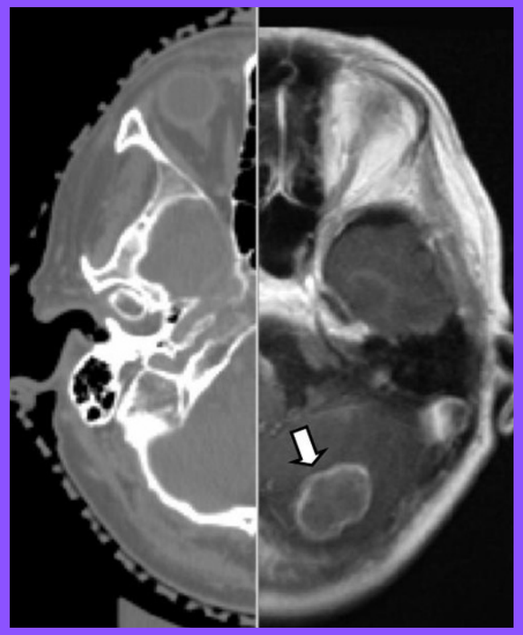
If training accuracy is close to 100% this is a good sanity check.

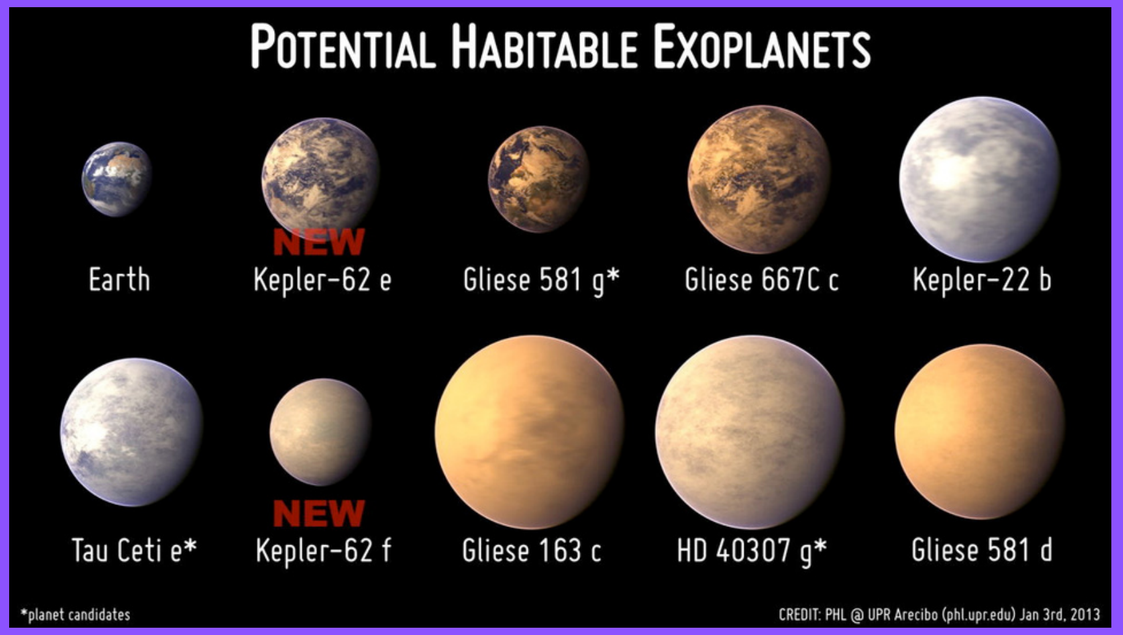


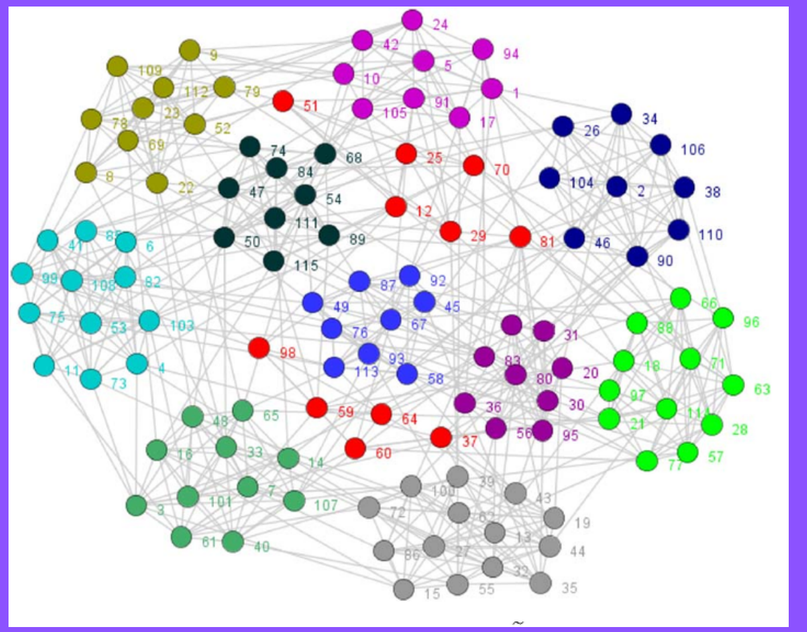
To further increase accuracy, it could be useful to increase the train set further (or the test set, to increase accuracy's... accuracy), or implementing something more advanced in the algorithm.

This Neural Network is easily adaptable to recognize any kind of pattern inside unprocessed data: images, audio, text, ecc.

It could be used to recognize disease patterns in medical imaging, to differentiate between data indicating the presence of an exoplanet of interest or not, to do community detection in complex networks, ecc.







BONUS: GitHub

Q: How can I adapt this NN to identify a pattern in any kind of images?

A: Follow these steps:

1) Clone this code depository: git clone https://github.com/Pherrara/AI-Logistic-Regression-on-Molecular-Dynamics.git

2) Acquire a set of train images, and a set of test images, es:0001.jpg.

3) Change num_px to the desired amount of resolution you want your image to have.

```
num_px = 200
```

4) Change the names of the folder you're using, instead of "solidi" you can just use the name "pattern" these are the folders with the images you want recognized.

```
cartelle = ["1.1_3", "0.3", "1.1_4", "0.3_4", "1.1_5",  "0.3_2", "1.1", "0.3_3", "1.1_2", "0.3_5"]
solidi = ["1.1_3", "1.1_5", "1.1_4", "1.1", "1.1_2",]
```

5) Change "start", "stop", "step", "m" to indicate how many images you want to use, and change the path in the for loops:

```
for cartella in cartelle:
    fnames = ['/home/pher/Documents/computational/newmovies/frames/' + cartella + '/{:04}.jpg'.format(i) for i in range(start, stop+1, step)]
```