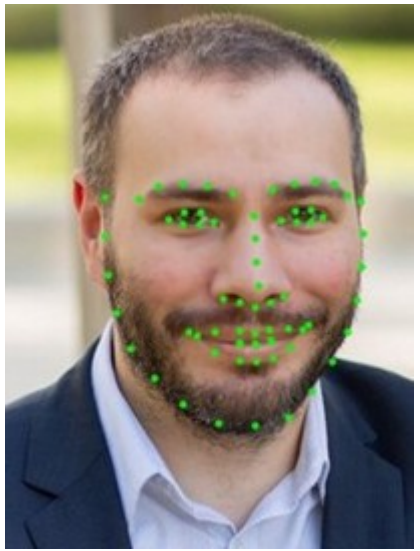
Sample Detecting Facial Features

https://towardsdatascience.com/detecting-face-features-with-python-30385aee4a8e



Point map
(67 points)

Jaw Points = 0–16

Right Brow Points = 17–21

Left Brow Points = 22–26

Nose Points = 27–35

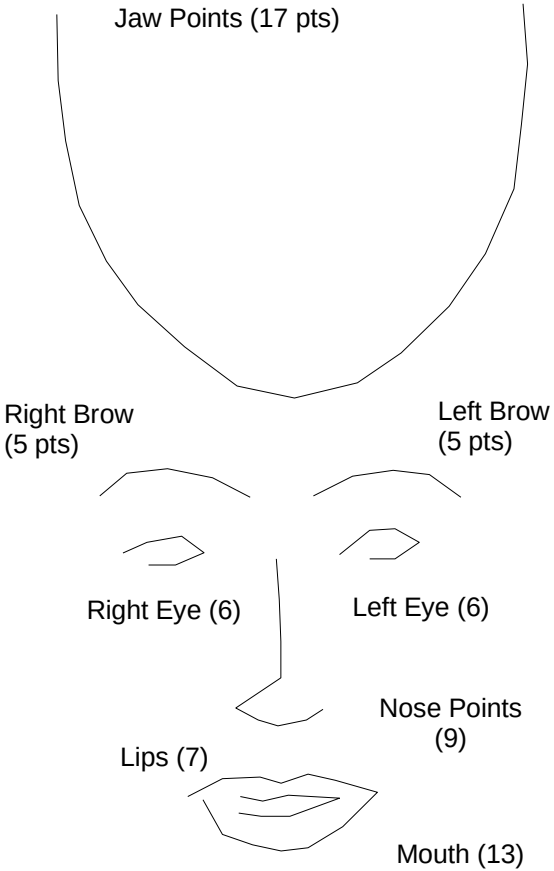
Right Eye Points = 36–41

Left Eye Points = 42–47

Mouth Points = 48–60

Lips Points = 61–67

Feature Vectors



$pip -V (to check your pip version)

Note this model is simplified version with no upper face

Harry Li, Ph.D.

Using http://dlib.net/ For Feature Extraction

Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high performance computing environments. Dlib's open source licensing allows you to use it in any application, free of charge.

1. Machine learning
2. Numerical algorithms
3. Graphical model inference algorithms
4. Image processing
5. Threading
6. Networking
7. GUI
8. Testing (Unit testing framework)
9. XML Parser etc.

Sample code:

```
import cv2
import numpy as np
import dlib   #for facial feature extraction
```

```
# Load the predictor
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
```

From dlib, additional features are possible

Pycharm IDE for Python

PyCharm is probably the only Python dedicated IDE that supports the vast expanse of features Python has. Sublime, Atom and Sypder do exist, but they only exist! The integrated development environment experience that PyCharm provides is way better than the others.

To install pycharm:

https://medium.com/@singh.shreya8/how-to-install-pycharm-in-ubuntu-16-04-ubuntu-14-04-ubuntu-18-04-linux-easiest-way-5ae19d052693

To start pycharm:
$sh pycharm.sh

https://www.youtube.com/watch?v=BPC-bGdBSM8&list=PLQ176FUIyIUZ1mwB-uImQE-gmkwzjNLjP

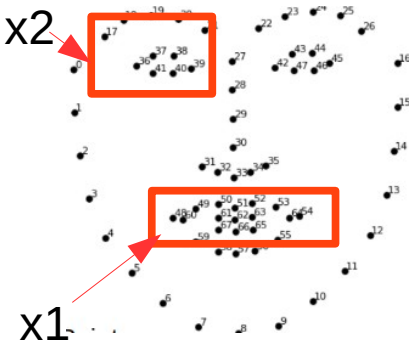
Pycharm hello the world

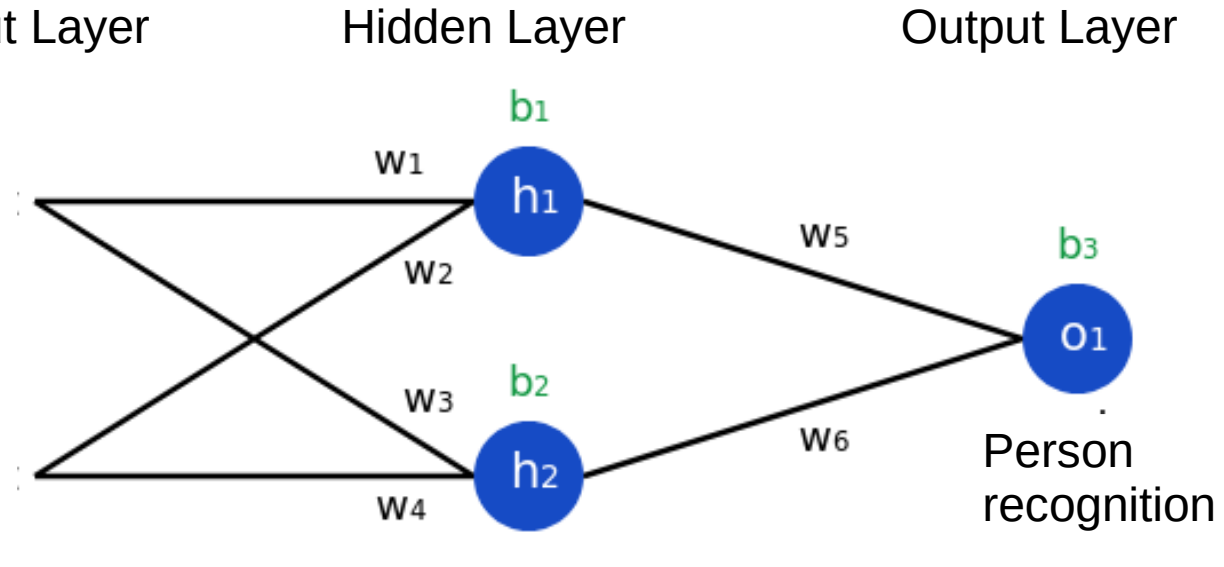
https://www.jetbrains.com/help/pycharm/creating-and-running-your-first-python-project.html#summary

use Cmake to build and install OpenCV and Extra Modules from source and configure your Pycharm IDE

https://towardsdatascience.com/how-to-install-opencv-and-extra-modules-from-source-using-cmake-and-then-set-it-up-in-your-pycharm-7e6ae25dbac5

Simple Feed Forward Neural Networks





To do: (1) add h3 hidden layer, (2) add o2 at output layer, modify the code

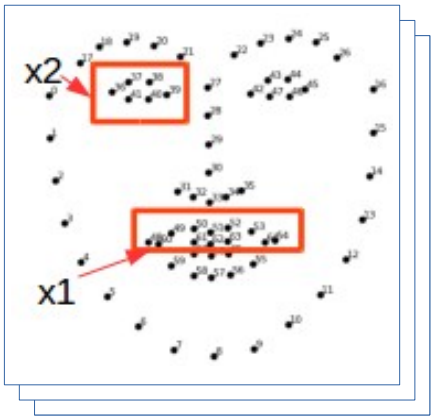
```
def feedforward(self, x):
  # x is a numpy array with 2 elements.
  h1 = sigmoid(self.w1 * x[0] + self.w2 * x[1] + self.b1)
  h2 = sigmoid(self.w3 * x[0] + self.w4 * x[1] + self.b2)
  o1 = sigmoid(self.w5 * h1 + self.w6 * h2 + self.b3)
  return o1
```

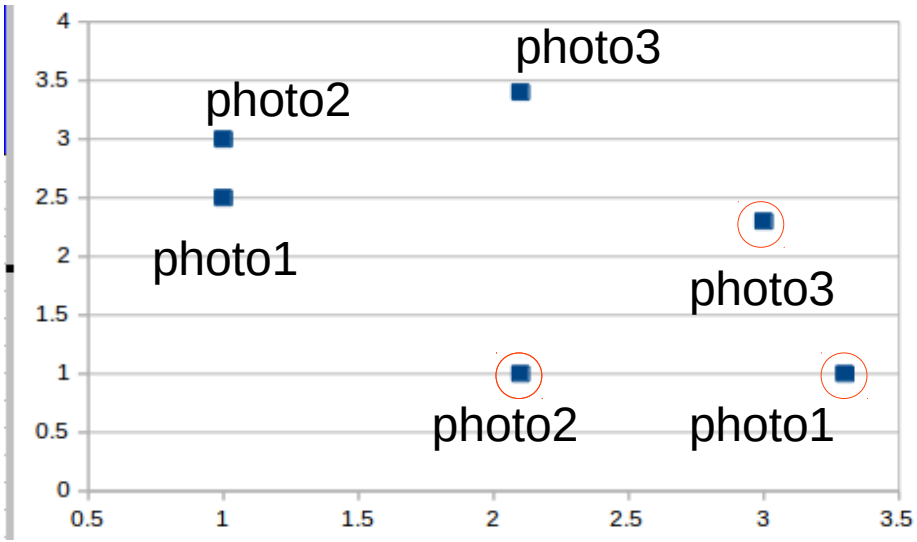
Prepare the Data Set

For 2 persons

```
#---------------------------------
# Define dataset and all_y_trues
#---------------------------------
data = np.array([
  [1, 2.5],     # person A
  [1, 3],       # person A
  [2.1, 3.4],   # Person A
  [2.1, 1], # person B
  [3.3, 1], # person B
  [3, 2.3], # person B
])
all_y_trues = np.array([
  1, # person A
  1, # person A
  1, # person A
  0, # person B
  0, # person B
  0, # person B
])
```

For 2 persons





Code Sample 10-2020F-103-2introNN.py

https://github.com/hualili/opencv/blob/master/deep-learning-2020S/10-2020F-103-2introNN.py

From base line code → (1) saving the trained NN → (2) load trained weights, add new feature vectors to the existing data set

10-2020F-103-2introNN.py

(3) load saved weights back to the same NN code but use it as initial condition, perform training with updated data set.

10-2020F-103-6introNN-Adaptive.py

Training Without Prior	With Prior
680 EPOCHS	~150 EPOCHS

Determine the Number of Neurons and Layers

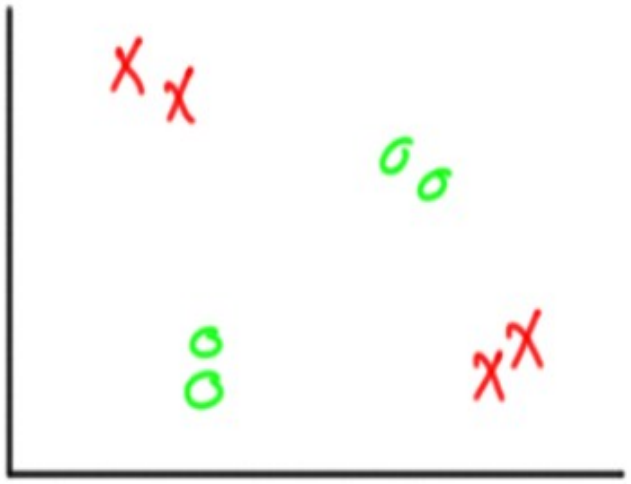
https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-51466afa0d3e

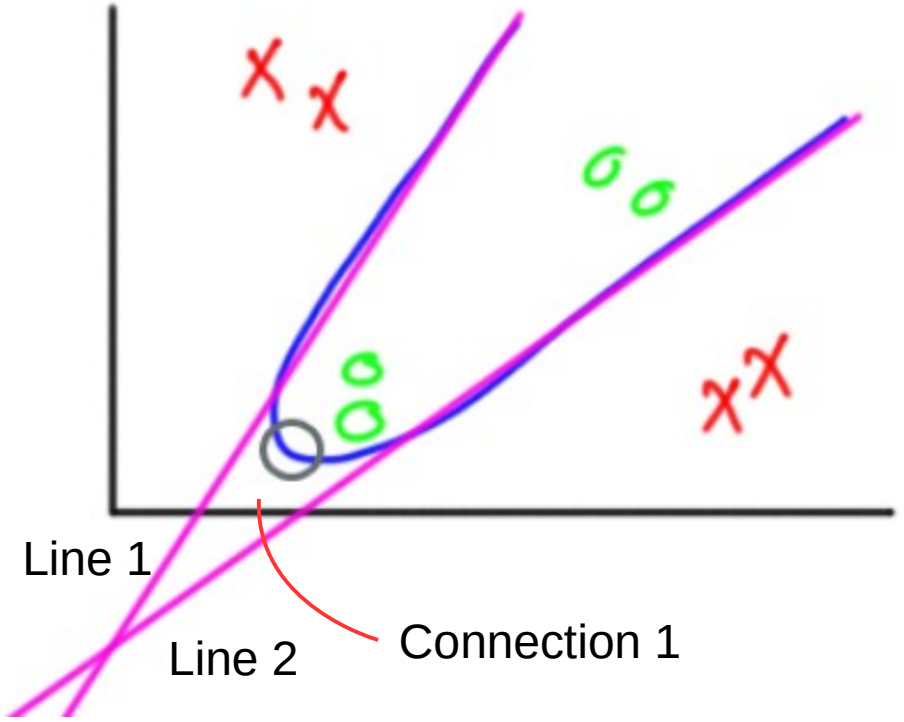
In artificial neural networks, hidden layers are required if and only if the data must be separated non-linearly.

Step 1. Get the data set first;
Step 2. Draw an expected decision boundary to separate the classes.
Step 3. Count the number N of these decision boundary as a set of lines.
Step 4. The number N of selected lines equal to the number of hidden neurons in the first hidden layer.
Step 5. To connect the drawn N lines, a new hidden layer is added. Note that a new hidden layer is added each time you need to create connections among the lines in the previous hidden layer. The number of hidden neurons in each new hidden layer equals the number of connections to be made.

Step 1 Data Set and Step 2 Draw Decision Lines

https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-51466afa0d3e





Decision making function is composed of lines, each line is a linear function from a single neuron:

$y = x1 * w1 + x2 * w2 + b$ … (1)

Step 3 and Step 4 for Number of Neurons

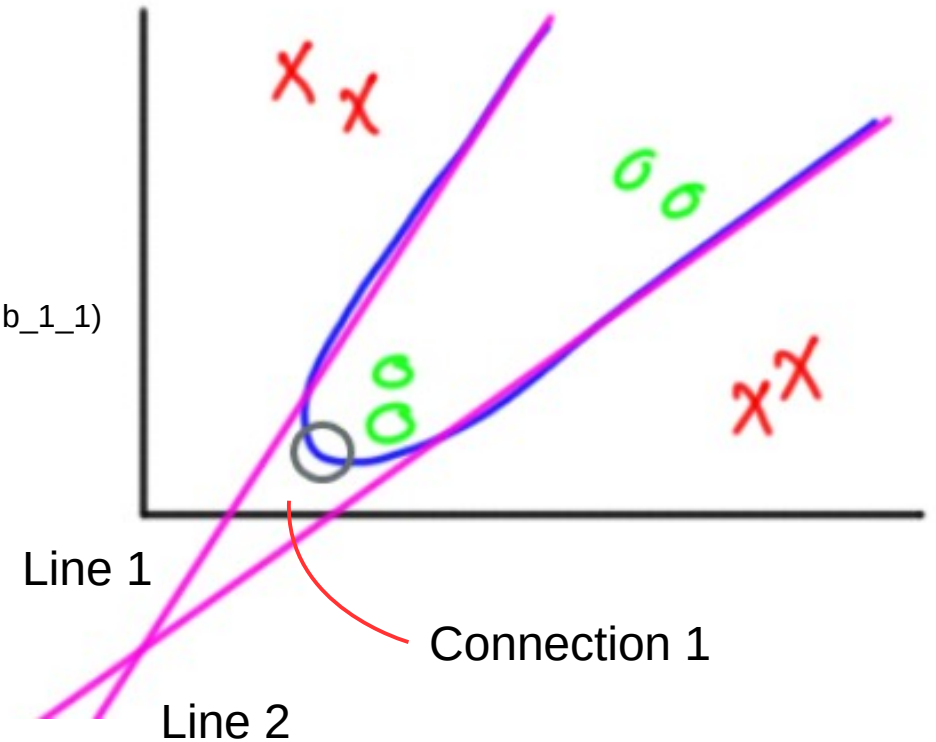
https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-51466afa0d3e

Step 3. Count the number N of these decision boundary as a set of lines.
Step 4. The number N of selected lines equal to the number of hidden neurons in the first hidden layer.

2 lines leads to 2 neurons in the first hidden layer

```
self.w_1_1=s[0]   # weight

h_1_1 = sigmoid(self.w_1_1 * x[0] + self.w_1_2 * x[1] + self.b_1_1)
# Neuron 1 at hidden layer 1
```



Step 5 Connection Lines for Hidden Neurons

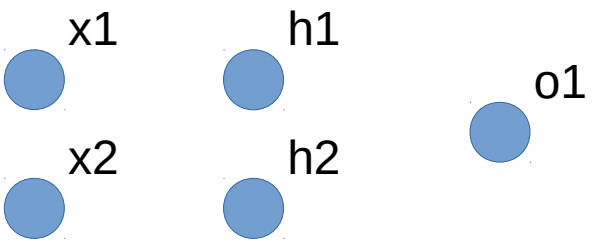
https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-51466afa0d3e

Step 5. To connect the lines created by the previous layer, a new hidden layer is added. Note that a new hidden layer is added each time you need to create connections among the lines in the previous hidden layer. The number of hidden neurons in each new hidden layer equals the number of connections to be made.

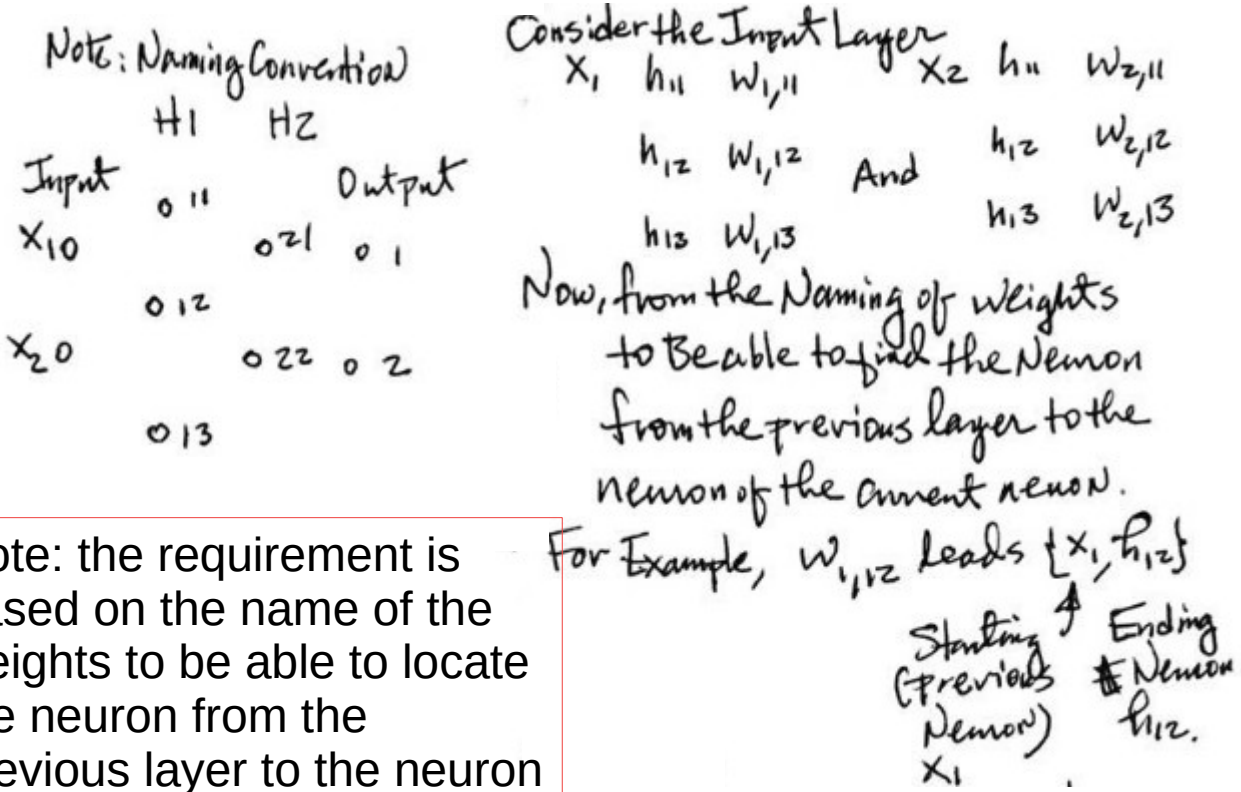
1. Need to connect Line 1 and Line 2 > add new hidden layer (or output layer);

2. One connection to join Line 1 and 2, so (a) add another layer, and (b) since it is one connection for both lines, so one neuron at that layer .

So the architecture of the NN is given in the right



Naming Convention for Coding



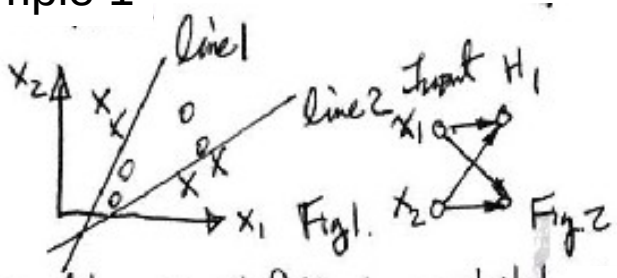
Note: the requirement is based on the name of the weights to be able to locate the neuron from the previous layer to the neuron of the current layer.

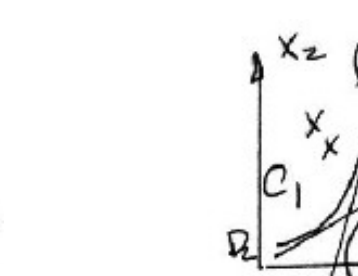
Example:

```
def __init__(self):
  # load prior weights & biases from a text file
  s=[]
  with open('trained_weights_bias.txt','r') as f:
      for line in f:
          for num in line.split(','):
              s.append(float(num))
  self.w_1_1=s[0]
  self.w_1_2=s[1]
  self.b_1_1=s[2]
```

Example 1 and 2

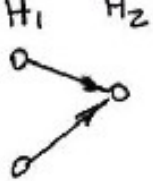
Example 1



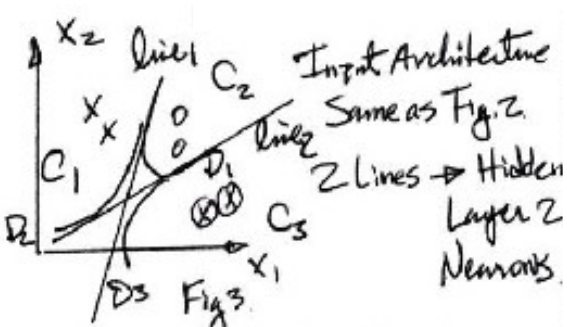


2 lines, so 1st hidden Layer Needed
And with 2 Neurons,
one neuron per one line

Then, one connection for Line1 & Line2
Next hidden Layer is needed
to join them, one connection
leads to one neuron.



Example 2



Input Architecture
Same as Fig.2.
2 Lines → Hidden
Layer 2
Neurons

Now, Decision-making Based on
"Directions" to join 2 lines.

C_2 : Join Line1 & 2 in One Direction

C_1 : Join Line1 & 2 in 2nd Direction

C_3 : Join Line1 & 2 in 3rd Direction

Thus, 2nd Hidden Layer is Needed
3 Directions, so 3 Neurons
to join

Example 3 and 4

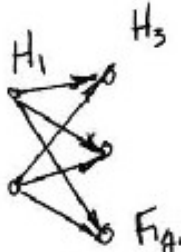


Fig.4

Now, Suppose we have 4 Classes.

Example 3

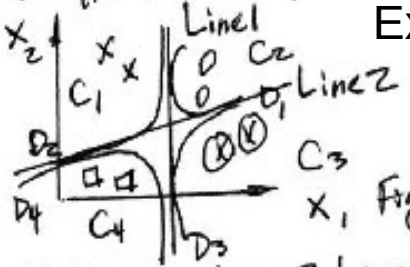


Fig.5

First, Hidden Layer1 for 2 Lines (Line1 & 2)
So 2 Nemons, So
its Structure is Same as in
Fig.2.

Then, Decision-making Based on Joining Line1 & 2 Directions.
4 Directions, So 4 Nemons for the 2nd Hidden Layer, e.g. Output layer.

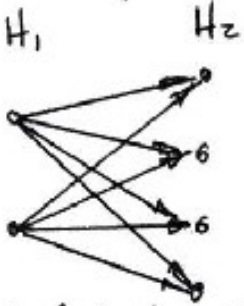


Fig.6

How About N Classes, N=10,
How About N= 50K Classes.
Mathematical Nature for the Decision Making part is How to Solve a set of Linear Equations.

Example 4

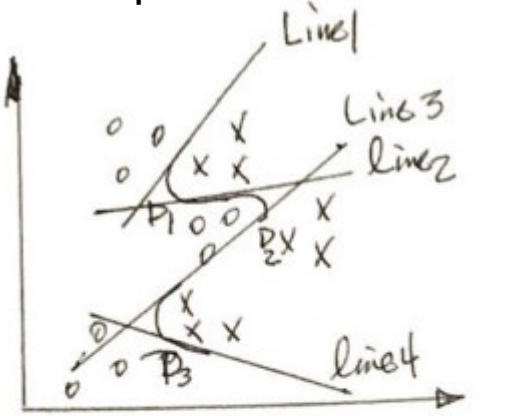


Fig.7

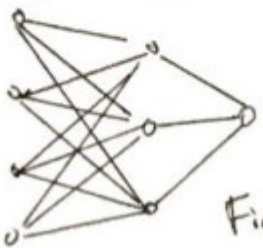


Fig.8

Additional Example

https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-51466afa0d3e