Artificial Neural Networks

Dr. Muhammad Aqib

University Institute of Information Technology

PMAS-Arid Agriculture University Rawalpindi

Layers of Artificial Neural Network

Contents

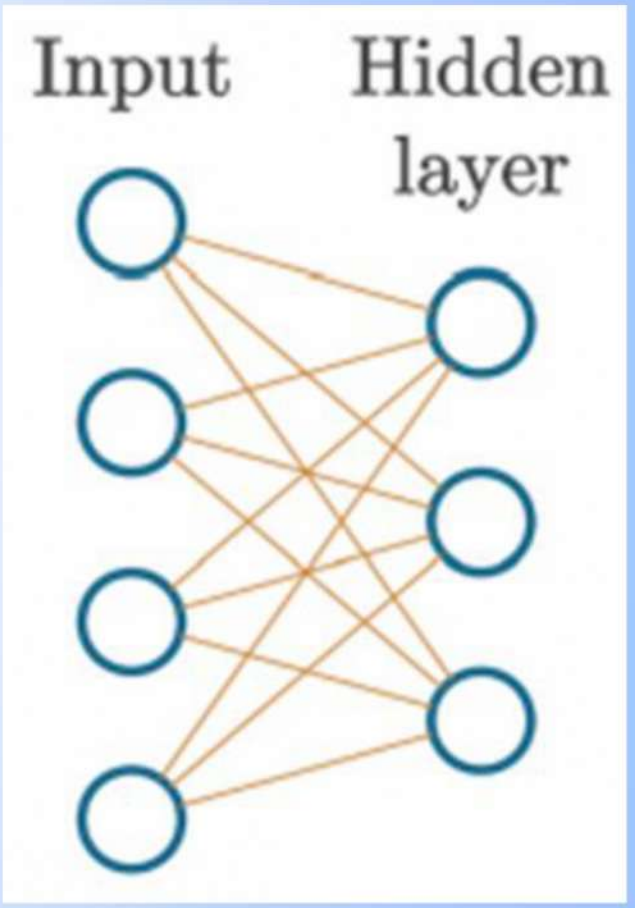
- Adding Layers to Network
- Training Data
- Data Visualization
- Data Generation
- Working with Dense Layer
- Dead Neurons

Adding Layers to Network

- Neural networks having two or more hidden layers are known as Deep Neural Networks.
- Hidden layers are sandwiched between Input and Output layers.
- Most of the time, main concern is:
  - What kind of input is required/expected?
  - What would be the output?
  - Hidden layers are typically not involved in high level perspective - Hence named "Hidden".
- Are hidden layers not interactable? Can't we tinker with hidden layers?
  - Hidden layers can also be tuned and tinkered with. "Hidden" doesn't mean you can't tune it.

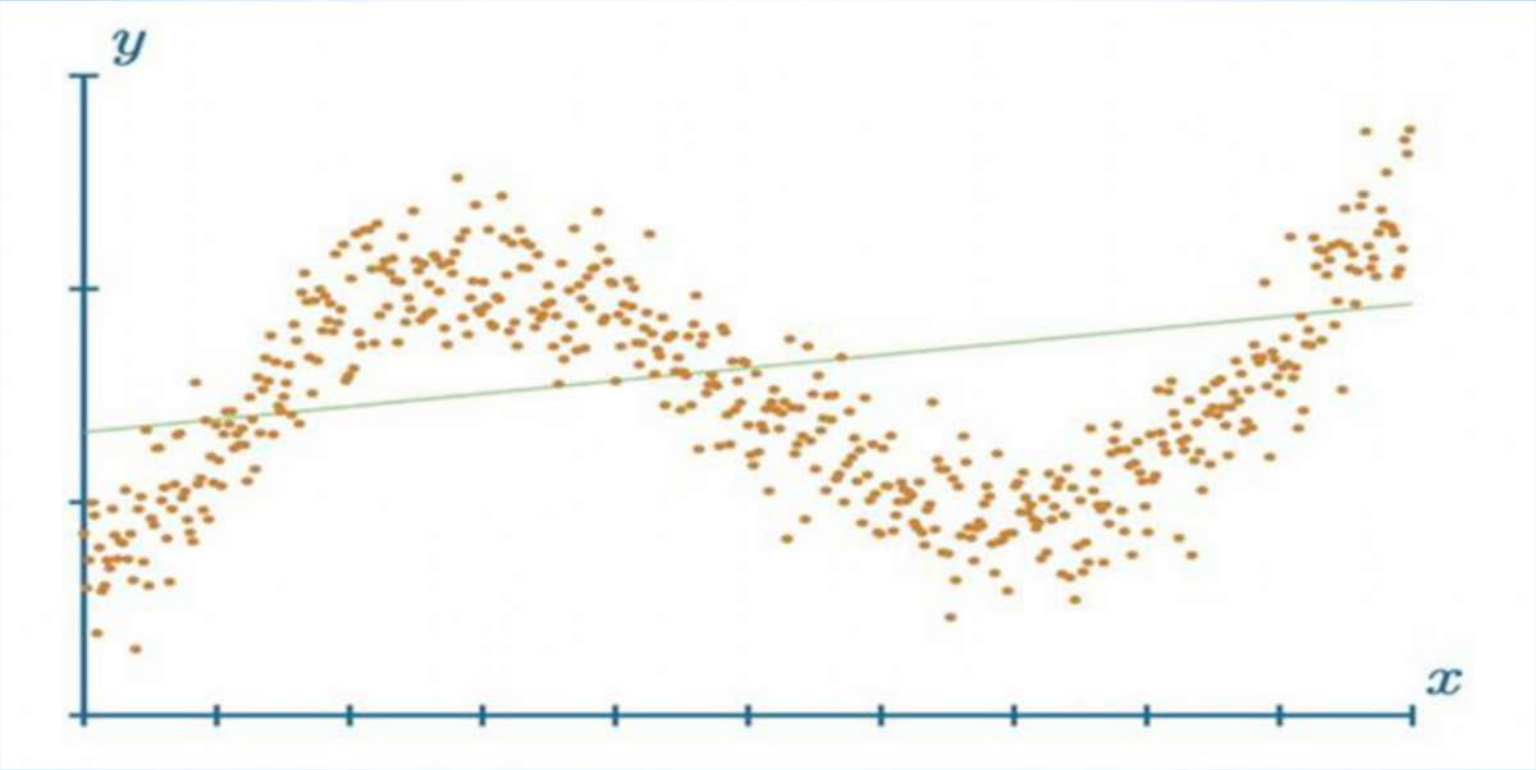
Adding Layers to Network (cont.)

- Example of Input layer with 4 features into a Hidden Layer with 3 Neurons.
- Samples are passed into network from Input layer.
- All 4 features are passed into hidden layer of 3 neurons with 3 sets of weights.
- Shape of initial weights is (3, 4) – 3 sets associated with 4 unique neurons.
- To add another hidden layer – Previous layers' output will become input to this new hidden layer.



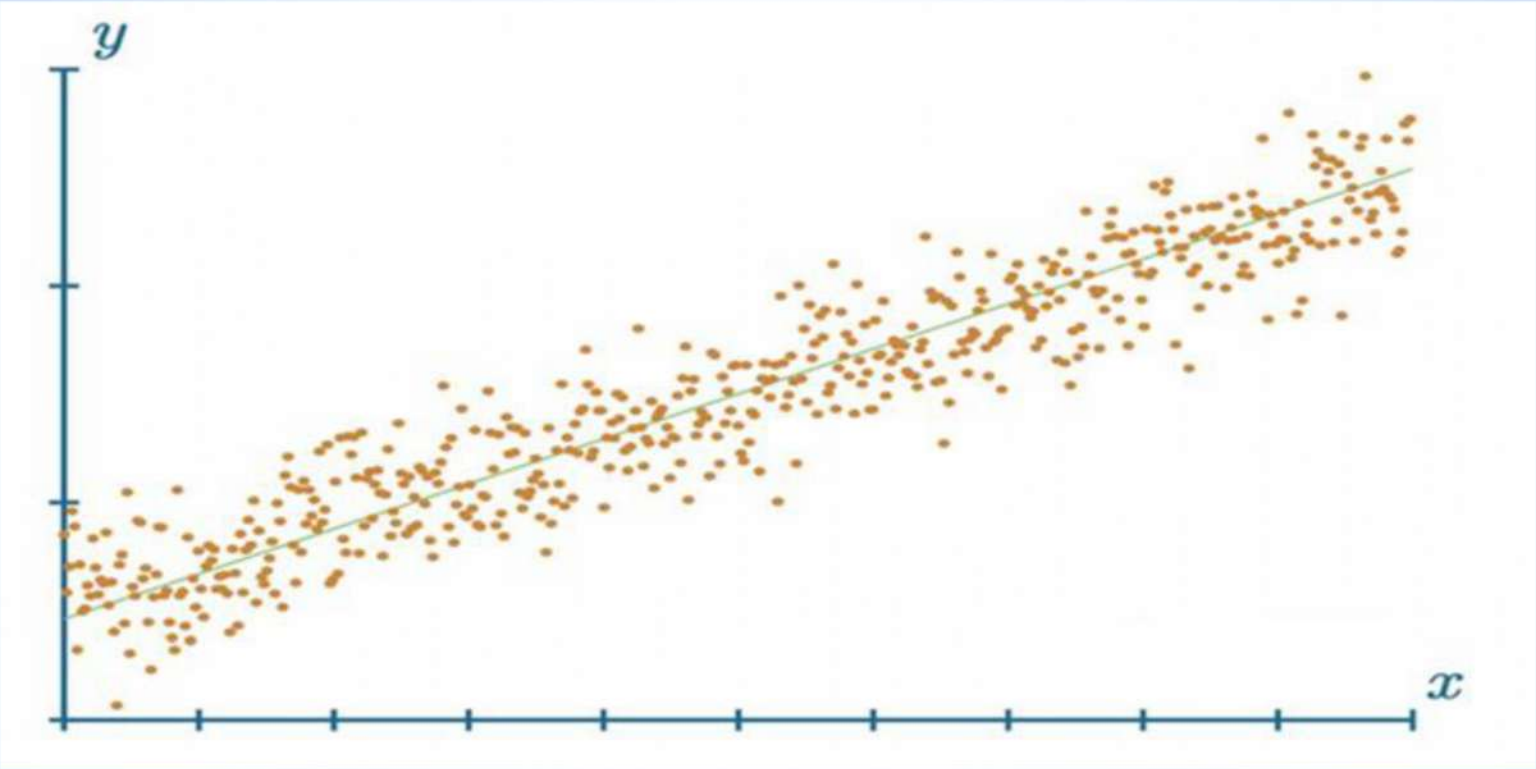
Training Data

- Non-linear data is data that cannot be represented well by a straight line.



Training Data (cont.)

- Linear data is data that can be represented by a straight line.



Data Visualization with Matplotlib

- Matplotlib is comprehensive library for creating static, animated, and interactive visualizations in Python.
- It can be installed using Python's package manager – pip.
- Command is `pip install matplotlib`
- For quick starting guide:

  https://matplotlib.org/stable/tutorials/introductory/quick_start.html

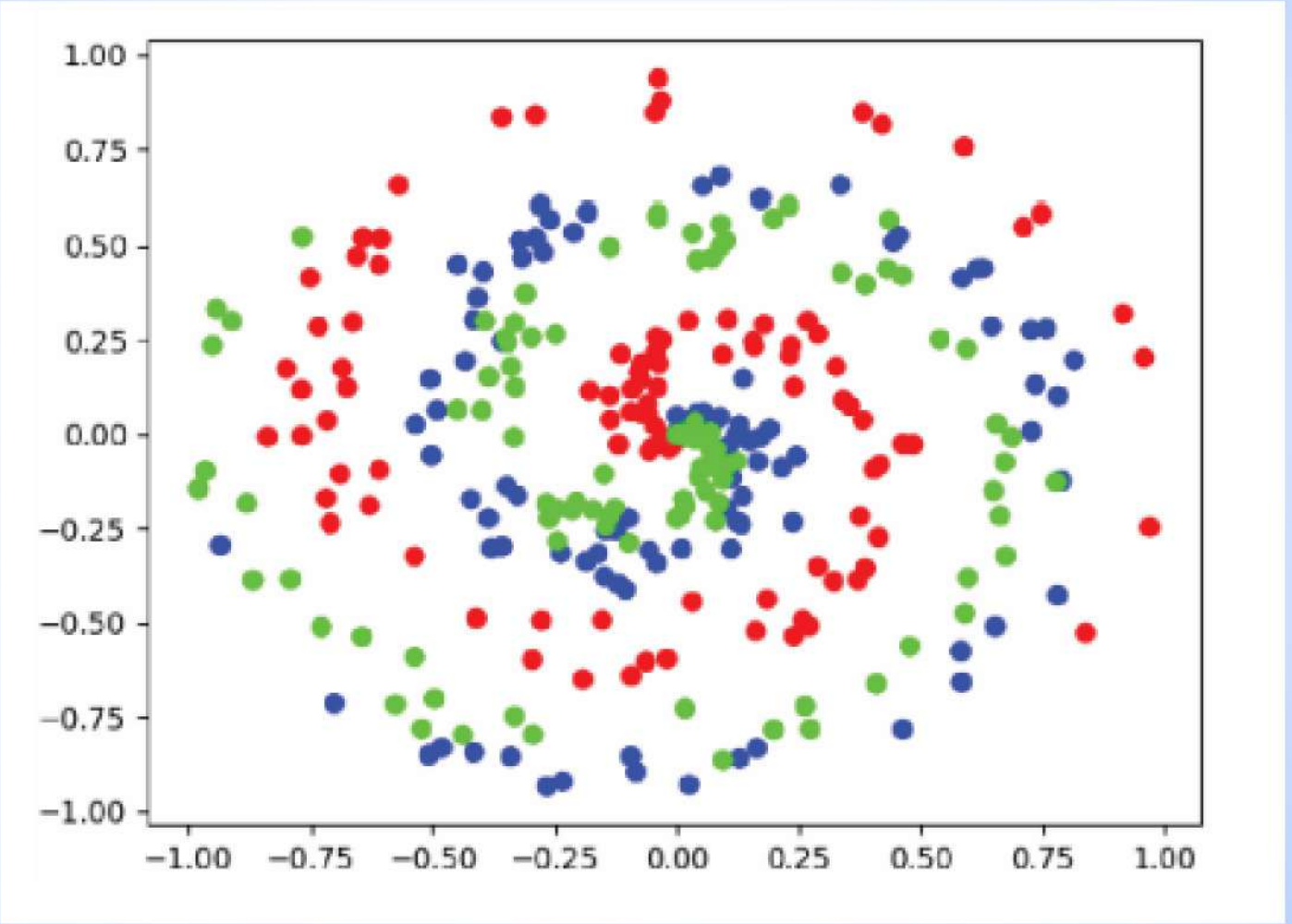
Data Generation for Neural Networks

- If you ~~were~~ want to graph data points of the form (x, y) where y = f(x), and it looks to be a line with a clear trend or slope, then chances are, they're linear data.
- Linear data is easily fitted/approximated by far simpler Machine Learning (ML) models
- Some of the ML models include:
  - Decision Trees
  - Random Forests
  - SVM…
- Non-linear datasets are not much well fitted/approximated by Machine Learning models.
- 'nnfs' package allows to generate random data consisting of random no. of classes.
- nnfs package can be installed with `pip install nnfs`

Data Generation for Neural Networks (cont.)


```
1  import numpy as np
2  import nnfs
3  from nnfs.datasets import spiral_data
4  import matplotlib.pyplot as plt
5
6  nnfs.init()
7  X, y = spiral_data(samples=100, classes=3)
8
9  plt.scatter(X[:, 0], X[:, 1], c=y, cmap='brg')
10 plt.show()
```

Data Generation for Neural Networks (cont.)



Working with Dense Layer

- Dense Layer / Fully Connected Layer (discussed in previous lectures – Recall!) are commonly referred as `dense` layer and sometimes `fc` layer – short for Fully connected layer.
- Dense layer Class will be consisting of two methods.
  - __init__(self, n_inputs, n_neurons)
  - forward(self, inputs)
- Weights are often initialized randomly for a model, but not always.
- In forward pass, all the inputs will be transformed from multiple neural networks layers till output is obtained at output layer.

Working with Dense (cont.)

- np.random.randn generates a number distribution with a mean of 0 and a variance of 1.
- It generates positives and negatives centered at 0 with mean close to 0.
- Multiplication with 0.01 is done just to make them a bit smaller.
- Otherwise, the model will take more time to fit the data during the training process
- np.zeros is used to create 1D array with specified neurons all set to 0. (Biases set to zero)

Working with Dense Layer (cont.)


```
def __init__(self, n_inputs, n_neurons):
    self.weights = 0.01 * np.random.randn(n_inputs, n_neurons)
    self.biases = np.zeros((1, n_neurons))
```

Thank You