Introduction to Neural Networks and Deep Learning

(Practical Sec-02)

---

1. Introduction to Neural Networks

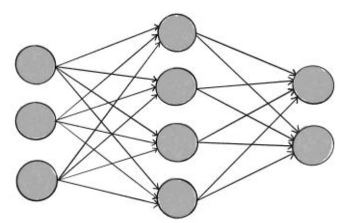
Basic concepts of neural networks.

What is a Neural Network?

- Inspired by the human brain.
- Composed of layers (input, hidden, and output).
- Learn patterns from data using weights and activation functions.

Structure of a Neural Network

- **Input Layer:** Receives input data.
- **Hidden Layers (Dense):** Extracts patterns & features.
- **Output Layer (Dense):** Produces final predictions.



Where are Neural Networks Used?

- Image recognition (e.g., face detection, medical imaging).
- Natural language processing (e.g., chatbots, language translation).
- Self-driving cars, fraud detection, etc.

Basic Structure of a Neural Network

- **Neurons:** Mathematical functions that process inputs.
- **Weights & Biases:** Control the importance of inputs.
- **Activation Functions:** Transform input values (ReLU, Sigmoid, etc.).
- **Forward Propagation:** Computes output.
- **Backpropagation:** Adjusts weights to improve learning.

---

2. Setting Up the Practical Environment

Install Required Libraries

pip install jupyterlab numpy pandas matplotlib tensorflow keras torch torchvision

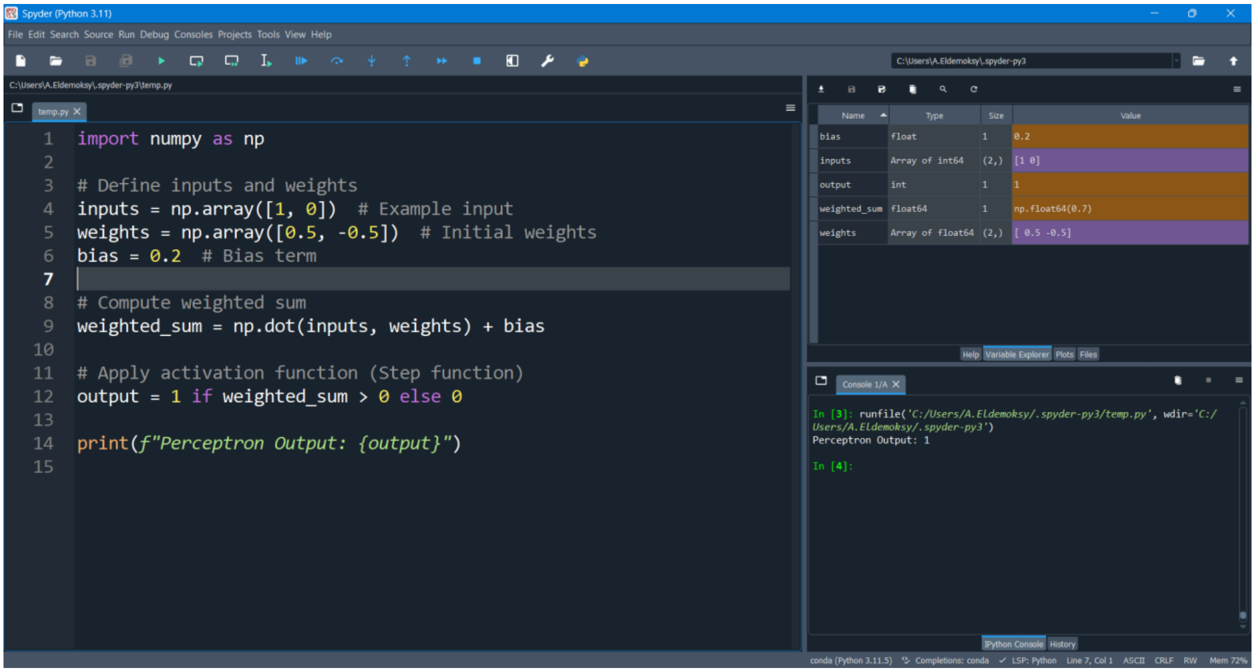
```
Anaconda Prompt
(base) C:\Users\A.Eldemoksy>pip install jupyterlab numpy pandas matplotlib tensorflow keras torch torchvision
```

```
Anaconda Prompt - pip insta
59f93d312077a9a2efbe04d59c393bc2b8802248c908d4/webcolors-24.11.1-py3-none-any.whl.metadata
  Downloading webcolors-24.11.1-py3-none-any.whl.metadata (2.2 kB)
Requirement already satisfied: arrow>=0.15.0 in c:\users\a.eldemoksy\anaconda3\lib\site-packages (from isoduration->json
schema>=4.17.3->jupyterlab-server~=2.19->jupyterlab) (1.2.3)
Downloading tensorflow-2.18.0-cp311-cp311-win_amd64.whl (7.5 kB)
Downloading tensorflow_intel-2.18.0-cp311-cp311-win_amd64.whl (390.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 390.2/390.2 MB 633.8 kB/s eta 0:00:00
Downloading numpy-2.0.2-cp311-cp311-win_amd64.whl (15.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15.9/15.9 MB 1.4 MB/s eta 0:00:00
Downloading keras-3.8.0-py3-none-any.whl (1.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 1.4 MB/s eta 0:00:00
Downloading torch-2.6.0-cp311-cp311-win_amd64.whl (204.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 204.2/204.2 MB 987.5 kB/s eta 0:00:00
Downloading sympy-1.13.1-py3-none-any.whl (6.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.2/6.2 MB 541.0 kB/s eta 0:00:00
Downloading torchvision-0.21.0-cp311-cp311-win_amd64.whl (1.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 608.7 kB/s eta 0:00:00
Downloading absl_py-2.1.0-py3-none-any.whl (133 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 133.7/133.7 kB 563.8 kB/s eta 0:00:00
Downloading h5py-3.12.1-cp311-cp311-win_amd64.whl (3.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.0/3.0 MB 743.0 kB/s eta 0:00:00
Downloading ml_dtypes-0.4.1-cp311-cp311-win_amd64.whl (126 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 126.7/126.7 kB 497.8 kB/s eta 0:00:00
Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Downloading namex-0.0.8-py3-none-any.whl (5.8 kB)
Downloading optree-0.14.0-cp311-cp311-win_amd64.whl (300 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 300.4/300.4 kB 976.9 kB/s eta 0:00:00
Downloading rich-13.9.4-py3-none-any.whl (242 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 242.4/242.4 kB 1.3 MB/s eta 0:00:00
Downloading astunparse-1.6.3-py2.py3-none-any.whl (12 kB)
```

3. Implementing a Simple Perceptron

Example1: Implement a Perceptron using NumPy

```
import numpy as np
# Define inputs and weights
inputs = np.array([1, 0])  # Example input
weights = np.array([0.5, -0.5])  # Initial weights
bias = 0.2  # Bias term
# Compute weighted sum
weighted_sum = np.dot(inputs, weights) + bias
# Apply activation function (Step function)
output = 1 if weighted_sum > 0 else 0
print(f"Perceptron Output: {output}")
```



Key Takeaways:

- Adjusting weights and bias impacts learning.
- Activation functions determine neuron output.

What is Bias in Neural Networks?

Bias is an additional parameter in a **neural network** that helps adjust the output along with the weighted sum of inputs. It ensures that the model can **shift** the activation function, allowing it to learn patterns more effectively.

Why Do We Need Bias?

- Without bias, a neural network might be too rigid and unable to **fit complex data**.
- It allows the model to **generalize better** by shifting activation functions.
- Helps prevent **underfitting**, especially when data doesn't pass through the origin (0,0).

Python code for activation functions

```
import numpy as np

def binary_step(x):

   return np.where(x >= 0, 1, 0)


def linear(x):

   return x

print('binary_step',binary_step(0.5))


def sigmoid(x):

   return 1 / (1 + np.exp(-x))

print('Sigmoid',sigmoid(0))
```

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

```
def relu(x):

   return np.maximum(0, x)

print('relu',relu([-1, 0, 2]))


def tanh(x):

   return np.tanh(x)

print('tanh',tanh(0.2))
```

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

```
def softmax(x):

   exp_x = np.exp(x - np.max(x))  # Stability improvement

   return exp_x / exp_x.sum(axis=0)

print('softmax',softmax([2,1,0]))
```

$$f(x_i) = \frac{e^{x_i}}{\sum e^{x_j}}$$

Function	Output Range	Used For
Binary Step	{0,1}	Simple classification
Linear	(-∞, ∞)	Regression
Sigmoid	(0,1)	Binary classification
ReLU	[0,∞)	Deep learning (hidden layers)
Tanh	(-1,1)	RNNs (saturates less than sigmoid)
Softmax	(0,1) (sums to 1)	Multi-class classification

Binary Classification on a Linearly Separable Dataset

Here, we are going to process building, training, and evaluating a **Perceptron model** for binary classification using a synthetic, linearly separable dataset. It covers data preprocessing, model training, and performance evaluation. We will follow these steps:

1. Import Libraries.
2. Generate Dataset using **make_blobs().**
3. Train-Test Split with **train_test_split().**
4. Scale Features using **StandardScaler().**
5. Initialize Perceptron with appropriate input size.
6. Train the Model with **fit()** over 100 epochs.
7. Predict test data and evaluate accuracy by comparing predictions with actual labels.
8. Visualize Results using a scatter plot.

Implementation

```
# Import the necessary library

import numpy as np

from sklearn.datasets import make_blobs

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import Perceptron
```

```
# Generate a linearly separable dataset with two classes
X, y = make_blobs(n_samples=1000,
   n_features=2,
   centers=2,
   cluster_std=3,
   random_state=23)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=23,
                                                    shuffle=True
                                                    )
# Scale the input features to have zero mean and unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Set the random seed legacy
np.random.seed(23)

# Initialize the Perceptron with the appropriate number of inputs
perceptron = Perceptron(max_iter=100, random_state=23)

# Train the Perceptron on the training data
perceptron.fit(X_train, y_train)

# Prediction
pred = perceptron.predict(X_test)
```

```
# Test the accuracy of the trained Perceptron on the testing data
accuracy = np.mean(pred == y_test)
print("Accuracy:", accuracy)


# Plot the dataset
plt.scatter(X_test[:, 0], X_test[:, 1], c=pred)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

- ⇨ **numpy (np)**: A library for numerical operations in Python.
- ⇨ **make_blobs**: A function from sklearn.datasets to generate synthetic datasets.
- ⇨ **matplotlib.pyplot (plt)**: A plotting library for visualizing data.
- ⇨ **train_test_split**: A function to split datasets into training and testing sets.
- ⇨ **StandardScaler**: A preprocessing tool to standardize features.
- ⇨ **Perceptron**: A linear model for binary classification from sklearn.linear_model.

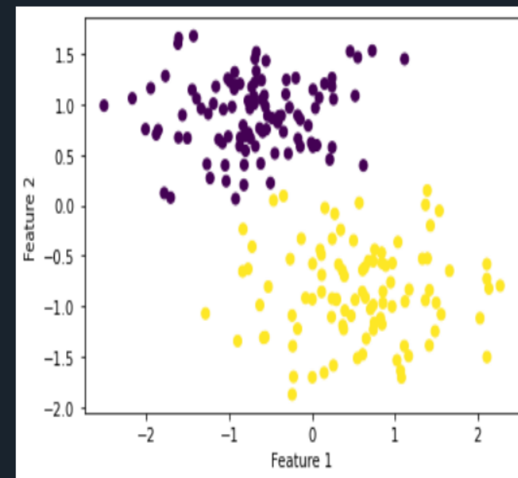
x, y = make_blobs(n_samples=1000, n_features=2, centers=2, cluster_std=3, random_state=23)

- **make_blobs**: Generates a dataset with 1000 samples, 2 features, and 2 centers classes).
- **n_samples=1000**: Number of data points.
- **n_features=2**: Number of features (dimensions) for each data point.
- **centers=2**: Number of classes (blobs).
- **cluster_std=3**: Standard deviation of the clusters, controlling the spread.
- **random_state=23**: Ensures reproducibility by setting a seed for random number generation.

\MET\2025\Term-2\NN&DL\Neural-Networks-and-Deep-Learning\Section02\Binary-Classification.py

Binary-Classification.py* | Implement-aPerceptron-using-NumPy.py | activation-function.py

```python
# Import the necessary library
import numpy as np
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
# Generate a linearly separable dataset with two classes
X, y = make_blobs(n_samples=1000,
    n_features=2,
    centers=2,
    cluster_std=3,
    random_state=23)
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X,
    y,
    test_size=0.2,
    random_state=23,
    shuffle=True
    )
# Scale the input features to have zero mean and unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Set the random seed legacy
np.random.seed(23)
# Initialize the Perceptron with the appropriate number of inputs
perceptron = Perceptron(max_iter=100, random_state=23)
# Train the Perceptron on the training data
perceptron.fit(X_train, y_train)
# Prediction
pred = perceptron.predict(X_test)
# Test the accuracy of the trained Perceptron on the testing data
accuracy = np.mean(pred == y_test)
print("Accuracy:", accuracy)
# Plot the dataset
plt.scatter(X_test[:, 0], X_test[:, 1], c=pred)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



Help | Variable Explorer | Plots | Files

Console 1/A

```
   ...: print('softmax',softmax([2,1,0]))
softmax [0.66524096 0.24472847 0.09003057]

In [22]: runfile('D:/MET/2025/Term-2/NN&DL/Neural-Networks-and-Deep-Learning/Section02/Binary-Classification.py', wdir='D:/MET/2025/Term-2/NN&DL/Neural-Networks-and-Deep-Learning/Section02')
Accuracy: 0.98
```

Important

Figures are displayed in the Plots pane by default. To make them also appear inline in the console, you need to uncheck "Mute inline plotting" under the options menu of Plots.

```
In [23]:
```

IPython Console | History

1. **Importing Libraries**
   The necessary Python libraries are imported:
   - numpy for numerical operations.
   - make_blobs from sklearn.datasets to generate a dataset.
   - matplotlib.pyplot for visualization.
   - train_test_split from sklearn.model_selection to split the dataset.
   - StandardScaler from sklearn.preprocessing for feature scaling.
   - Perceptron from sklearn.linear_model to implement a Perceptron model.
2. **Generating the Dataset**
   - make_blobs generates a dataset with:
     - 1000 samples (n_samples=1000).
     - 2 features (n_features=2).
     - 2 clusters (centers=2).
     - A standard deviation of 3 for the clusters (cluster_std=3).
     - A fixed random state (random_state=23) to ensure reproducibility.
3. **Splitting the Data**
   - The dataset is split into training (X_train, y_train) and testing (X_test, y_test) sets.
   - test_size=0.2 means 20% of the data is used for testing, and 80% is used for training.
   - random_state=23 ensures the split remains consistent every time.
4. **Feature Scaling**
   - StandardScaler is used to standardize the dataset.
   - The training data is fitted and transformed (scaler.fit_transform(X_train)).
   - The testing data is only transformed (scaler.transform(X_test)) using the same scaler.
5. **Setting the Random Seed**
   - np.random.seed(23) ensures consistent random behavior.

6. **Initializing the Perceptron Model**
    - A Perceptron is initialized with:
        - A maximum of 100 iterations (max_iter=100).
        - A fixed random state (random_state=23).
7. **Training the Perceptron**
    - The model is trained using perceptron.fit(X_train, y_train).
8. **Making Predictions**
    - The trained Perceptron predicts labels for X_test.
9. **Calculating Accuracy**
    - The accuracy is computed as the percentage of correctly classified samples.
    - np.mean(pred == y_test) calculates the accuracy.
    - The accuracy is printed.
10. **Visualizing the Results**

- A scatter plot is created with:
    - X_test[:, 0] (Feature 1) on the x-axis.
    - X_test[:, 1] (Feature 2) on the y-axis.
    - The color (c=pred) represents the predicted class.