UNIT – V

5.1 Introduction to Artificial Neural Networks with Keras.

a) Artificial Neural Networks:

"Artificial Neural Networks or ANN is an information processing paradigm that is inspired by the way the biological nervous system such as brain process information. It is composed of large number of highly interconnected processing elements(neurons) working in unison to solve a specific problem."

b) Keras

Deep learning is one of the major subfield of machine learning framework. Machine learning is the study of design of algorithms, inspired from the model of human brain. Deep learning is becoming more popular in data science fields like robotics, artificial intelligence(AI), audio & video recognition and image recognition. Artificial neural network is the core of deep learning methodologies. Deep learning is supported by various libraries such as Theano, TensorFlow, Caffe, Mxnet etc., Keras is one of the most powerful and easy to use python library, which is built on top of popular deep learning libraries like TensorFlow, Theano, etc., for creating deep learning models.

c) Key Concepts

Neurons: The basic unit of an ANN, similar to a biological neuron. Each neuron receives inputs, processes them using a weighted sum, and applies an activation function to produce an output.

Layers: Neurons are organized into layers. A typical ANN has:

Input Layer: Receives the input data.

Hidden Layers: Intermediate layers where computations occur.

Output Layer: Produces the final prediction.

Activation Functions: Functions that determine the output of neurons. Common examples include:

ReLU (Rectified Linear Unit): f(x) = max(0, x)

Sigmoid: f(x) = 1 / (1 + e^(-x))

Softmax: Used in the output layer for multi-class classification.

Loss Function: A measure of how well the model's predictions match the actual labels. Common loss functions include mean squared error for regression and categorical cross-entropy for classification.

Optimizer: An algorithm that adjusts the weights of the network to minimize the loss function. Popular optimizers include SGD (Stochastic Gradient Descent) and Adam.

d) Conclusion

Keras simplifies the process of building and training ANNs. As you progress, you can explore more advanced topics like convolutional neural networks (CNNs), recurrent neural networks (RNNs), and hyperparameter tuning. Keras also supports callbacks, regularization techniques, and integration with other libraries for enhanced functionality.

5.2 Implementing Multi-Layer Perceptrons (MLPs) using Keras:

1. Multi-layer perception is the basic type of algorithm used in deep learning it is also known as an artificial neural network and they are the most useful type of neural network.
2. They are connected to multiple layers in a directed graph a perceptron is a single neuron model that was a precursor to large neural Nets it is a field of study that investigates how simple models of the biological brain can be used to solve different computational tasks.
3. The goal is to create robust algorithms and data structures that can be used to model difficult problems MLP utilizes a supervised learning technique called backpropagation.
4. MLP is used to solve regression and classification problems we use MLP in speech and image processing computer vision time series prediction and machine translation.
5. In MLP neurons are arranged into networks a row of neurons is called a layer and one network can have multiple layers any network model starts with an input layer that takes input from the data set layers after the input layer are called hidden layers

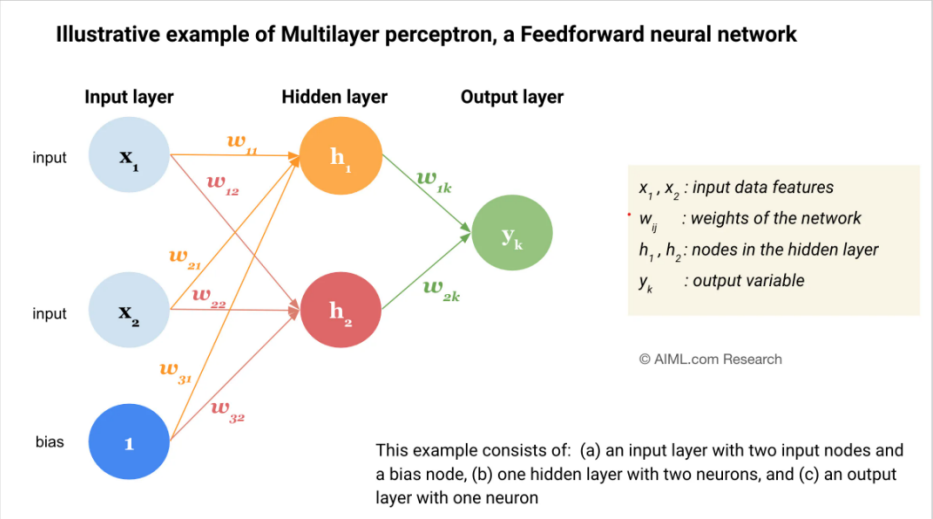

Figure-1

1. In the above example, we have two hidden layers that are not directly exposed to the input the planning can refer to have many hidden layers in our network model usually if we increase the number of hidden layers the model will work more efficiently.
2. The final layer is called the output layer and it is responsible for the final outcome the choice of activation function in the output layer strongly depends on the type of problem.
3. If we are modeling a regression problem may have a single output neuron and the neuron may have no activation function a binary classification problem may have a single output neuron and use a sigmoid activation

function to output a value between zero and one.

4. A multi-class classification problem may have multiple neurons in the final output layer one for each class in this scenario softmax activation function may be used to output our probability of the network.

Dataset

Data is comma-separated it has a total of 891 samples of five variables first one is survival column zero means dead one means survived this is our target variable next is passenger class so we have three classes 1 2 & 3 then 6 after that age and the final one is fair these four are our predictors using predictors we will predict the class survival which contains 0 & 1.

5.3 Installing TensorFlow

Introduction

TensorFlow is an open-source deep learning framework developed by Google that enables developers to build and deploy machine learning models easily. It provides a flexible architecture, allowing users to run computations on CPUs, GPUs, and even mobile devices.

Key Features of TensorFlow

1. **Versatility**: TensorFlow supports a wide range of machine learning and deep learning tasks, including neural networks, reinforcement learning, and natural language processing.
2. **Ecosystem**: TensorFlow has a rich ecosystem, including libraries and tools such as:
    - **Keras**: A high-level API for building and training neural networks.
    - **TensorBoard**: A visualization tool for monitoring and debugging machine learning models.
    - **TensorFlow Lite**: For deploying models on mobile and edge devices.
    - **TensorFlow Extended (TFX)**: For building production ML pipelines.
3. **Graph-Based Computation**: TensorFlow uses a computational graph model, where nodes represent operations and edges represent the flow of data. This allows for efficient computation, optimization, and parallel execution.

4. **Eager Execution**: TensorFlow supports eager execution, which allows for immediate computation and debugging without the need to build static graphs first. This is particularly useful for rapid prototyping.
5. **Scalability**: TensorFlow can scale across multiple CPUs and GPUs, making it suitable for training large models on large datasets.
6. **Community and Support**: Being open-source and widely adopted, TensorFlow has a large community, extensive documentation, and numerous tutorials available.

Installing TensorFlow

Installing TensorFlow is a straightforward process. Here's how you can do it, depending on your environment (e.g., local machine, virtual environment, or cloud service).

Installing TensorFlow

1. **Using pip (Recommended for most users)**

You can install TensorFlow via pip. Open your terminal (or command prompt) and run:

"pip install tensorflow"

2**. Installing a Specific Version**

If you need a specific version, you can specify it like this:

"pip install tensorflow==2.x.x # Replace 2.x.x with the desired version number"

3. Using a Virtual Environment (Recommended)

It's often a good idea to create a virtual environment to manage dependencies. Here's how to do it:

On macOS/Linux:

Install virtualenv if you haven't already

"pip install virtualenv"

Create a new virtual environment

"virtualenvmyenv"

Activate the virtual environment

"source myenv/bin/activate"

Install TensorFlow inside the virtual environment

"pip install tensorflow"

On Windows:

Install virtualenv if you haven't already

pip install virtualenv

Create a new virtual environment

virtualenvmyenv

Activate the virtual environment

myenv\Scripts\activate

Install TensorFlow inside the virtual environment

pip install tensorflow

4. Installing TensorFlow with GPU Support

If you have a compatible GPU and want to utilize it, install the GPU version of TensorFlow:

"pip install tensorflow-gpu"

5. Using Conda

If you prefer using Anaconda, you can install TensorFlow using conda:

"conda install tensorflow"

6. Verifying the Installation

To verify that TensorFlow has been installed correctly, run the following commands in a Python shell:

"import tensorflow as tf
print(tf.__version__)"

5.4 Loading and Pre-processing Data with TensorFlow:

Efficient data loading and preprocessing are critical steps in any machine learning pipeline. With TensorFlow's tf.data.Dataset API, data pipelines can be created to handle large datasets, ensuring that the data is properly shuffled, batched, normalized, and augmented. This not only improves the performance of the model but also reduces the overall training time by optimizing the way data is fed into the neural network.

Loading and Preprocessing Data with TensorFlow

In machine learning, the quality and structure of data directly impact the performance of a model. TensorFlow, a popular open-source machine learning framework, provides efficient tools and methods for loading and preprocessing data. This process ensures that data is in the right format and properly prepared before being fed into the model. Here's a step-by-step guide on how to perform these tasks in TensorFlow.

1. Loading Data

TensorFlow offers multiple APIs to load data depending on the type and format of the dataset:

- **tf.data.Dataset API**: This API is the backbone of data loading and input pipelines in TensorFlow. It helps create datasets from raw data, allowing for easy manipulation and processing.

Loading from Arrays or Tensors:

When working with small datasets stored in memory, tf.data.Dataset.from_tensor_slices() can be used to create a dataset from arrays or tensors.

```
dataset = tf.data.Dataset.from_tensor_slices((features, labels))
```

Loading from Files:

For large datasets, files such as images or text data can be loaded efficiently using Dataset.list_files() to create a list of file paths and Dataset.map() to process the files.

```
dataset = tf.data.Dataset.list_files('path_to_files/*.jpg')
dataset = dataset.map(process_image)
```

LoadingImages:

TensorFlow provides a high-level utility,tf.keras.preprocessing.image_dataset_from_directory(), to load image data directly from directories. This is useful when working with labeled images organized in folders.

```
image_dataset = tf.keras.preprocessing.image_dataset_from_directory('path_to_data/', image_size=(224, 224),
  batch_size=32)
```

Loading Text Data: For text datasets, tf.data.TextLineDataset() can be used to load data line by line from text files, making it ideal for handling large text-based datasets.

```
text_dataset = tf.data.TextLineDataset('file.txt')
```

2. Preprocessing Data

Once the data is loaded, preprocessing is necessary to ensure the data is in the correct format, scaled properly, and suitable for model training. Preprocessing typically includes normalization, augmentation, batching, shuffling, and other transformations.

Normalization: Normalization ensures that input data is within a specific range, usually [0, 1], making it easier for neural networks to process. For image data, this can be done using tf.image.per_image_standardization(), which standardizes each image.

```
def preprocess_image(image, label):
image = tf.image.resize(image, (224, 224))  # Resizing to a standard size
image = tf.cast(image, tf.float32) / 255.0  # Normalizing pixel values
return image, label
```

```
dataset = dataset.map(preprocess_image)
```

Shuffling and Batching: To ensure the model doesn't learn based on the order of the data, TensorFlow allows for shuffling the data with dataset.shuffle(buffer_size). Batching the data, done via dataset.batch(batch_size), groups the data into mini-batches for efficient training.

```
dataset = dataset.shuffle(1000).batch(32)
```

Data Augmentation: For image data, augmentation techniques like flipping, rotation, and cropping are often applied to artificially increase the size and variability of the dataset. This helps in making the model more robust.

```
def augment_image(image, label):
image = tf.image.random_flip_left_right(image)
image = tf.image.random_brightness(image, max_delta=0.2)
return image, label
dataset = dataset.map(augment_image)
```

Tokenization for Text Data: For text data, tokenization converts raw text into numerical form so it can be processed by the model. TensorFlow's Tokenizer class converts text to sequences of numbers and helps in handling word embeddings for NLP tasks.

```
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=10000)
tokenizer.fit_on_texts(text_data)
sequences = tokenizer.texts_to_sequences(text_data)
```

3. Creating Efficient Pipelines

TensorFlow allows the creation of efficient input pipelines to streamline the process of loading and preprocessing data. Two key features that improve efficiency are:

Prefetching: By using dataset.prefetch(buffer_size=tf.data.AUTOTUNE), TensorFlow can prepare the next batch of data while the current batch is being processed by the model, reducing latency and speeding up training.

```
dataset = dataset.prefetch(buffer_size=tf.data.AUTOTUNE)
```

Parallel Mapping: To speed up the preprocessing step, TensorFlow allows for parallel processing of data using num_parallel_calls. This helps in handling large datasets.

```
dataset = dataset.map(preprocess_image, num_parallel_calls=tf.data.AUTOTUNE)
```