**Unit 4**

**Q.1) What is sequential or time series data. Explain various Application which involves time series data?**

* **Sequential or time series** data refers to a type of data that is ordered and indexed based on time or sequence. It represents information collected at different time points or instances, where the order of the data points is significant and meaningful. In other words, each data point in a time series dataset is associated with a specific timestamp or index, indicating its temporal ordering.
* **Application which involves time series data:**

1. **Financial Forecasting:** Time series data is extensively used in finance for predicting stock prices, market trends, asset prices, and risk assessment. It helps investors and financial institutions make informed decisions and develop trading strategies.
2. **Weather Forecasting:** Time series analysis is crucial in weather forecasting to predict variables such as temperature, humidity, wind speed, and precipitation. These predictions help in planning agricultural activities, disaster preparedness, and energy management.
3. **Health Monitoring:** Time series data collected from medical sensors, wearable devices, or electronic health records are used for patient monitoring, disease progression analysis, and predicting health outcomes. It enables personalized medicine and early intervention.
4. **Internet of Things (IoT):** IoT devices generate time series data from various sensors and devices. Analyzing this data helps in smart home automation, environmental monitoring, traffic management, and predictive maintenance of IoT infrastructure.
5. **Natural Language Processing (NLP):** Time series analysis is used in NLP for tasks like sentiment analysis, text classification, and machine translation. The temporal order of words or sentences is essential to capture meaning and context.
6. **Signal Processing:** Time series analysis is fundamental in signal processing applications, such as audio and speech recognition, image and video processing, and communication systems. It enables noise removal, pattern recognition, and signal enhancement.

**Q.2) What is sequential model. What are different sequential models in deep learning?**

* **The sequential model** is typically built using recurrent neural networks (RNNs) or their variants, such as long short-term memory (LSTM) or gated recurrent units (GRUs). These models are designed to capture dependencies and patterns in sequential data by maintaining an internal state that is updated at each step based on the current input and the previous state. The sequential model operates in a sequential manner, taking one input element at a time and passing it through the network.
* **Sequential models in deep learning:**
* **Recurrent Neural Networks (RNNs):** RNNs are a class of neural networks that can process sequential data by maintaining a hidden state that carries information from previous time steps. They are designed to handle variable-length input sequences and capture long-term dependencies. However, traditional RNNs suffer from the vanishing gradient problem, which can limit their ability to capture long-term dependencies.
* **Long Short-Term Memory (LSTM) Networks:** LSTMs are a type of RNN that addresses the vanishing gradient problem by using memory cells and gating mechanisms. LSTMs are capable of learning long-term dependencies and are widely used for tasks like language modeling, speech recognition, and sentiment analysis.
* **Gated Recurrent Units (GRUs**): GRUs are another variant of RNNs that address the vanishing gradient problem. They have a simplified architecture compared to LSTMs, combining the forget and input gates into a single "update" gate. GRUs are known for their efficiency and have been successful in various sequence modeling tasks.
* **Convolutional Neural Networks (CNNs) with 1D Convolutions:** While CNNs are commonly used for image recognition, they can also be adapted for sequential data processing. By applying 1D convolutions along the sequence, CNNs can capture local patterns and extract relevant features. They are often used for tasks like text classification, sentiment analysis, and time series forecasting.

**Q.3) What is natural language processing/ text processing. Explain various application of NLP?**

* **Natural Language Processing (NLP)** is a field of study that combines linguistics, computer science, and artificial intelligence to enable computers to understand, interpret, and generate human language in a meaningful way. NLP focuses on developing algorithms and models that facilitate communication between computers and humans using natural language. NLP involves various techniques and methodologies for processing and analyzing text data. Text processing refers to the tasks involved in manipulating and extracting useful information from raw text.
* **Application of NLP:**
* **Sentiment Analysis:** Analyzing text to determine the sentiment expressed, such as positive, negative, or neutral. It is used in social media monitoring, customer feedback analysis, brand monitoring, and market research.
* **Machine Translation:** Automatically translating text from one language to another. NLP-based translation systems enable communication across language barriers and have applications in international business, travel, and content localization.
* **Chatbots and Virtual Assistants:** Creating conversational agents that can interact with users in natural language. Chatbots and virtual assistants are used in customer support, personal assistants, and information retrieval systems.
* **Information Extraction:** Automatically extracting structured information from unstructured text, such as named entities, relationships between entities, and events. It is used in applications like knowledge graph construction, question answering, and data mining.
* **Text Summarization**: Generating concise summaries of larger text documents or articles. Text summarization finds applications in news aggregation, document analysis, and information retrieval systems.
* **Speech Recognition:** Converting spoken language into written text. Speech recognition technologies are used in voice assistants, transcription services, and accessibility tools.
* **Natural Language Understanding:** Enabling computers to comprehend and interpret the meaning of text. This includes tasks like semantic role labeling, coreference resolution, and understanding context.
* **Text Generation:** Creating human-like text based on a given prompt or context. Text generation is used in creative writing assistance, content generation, and chatbot responses.

**Q.4) What is mean by feature extraction from text. Explain bag of words and TF idf technique for text representation with suitable example?**

* **Feature extraction from text** involves transforming raw text data into numerical representations, known as features, that can be used as input for machine learning algorithms. Let's explore two commonly used techniques: Bag-of-Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF).

1. **Bag-of-Words (BoW):** The Bag-of-Words approach represents text as a collection of individual words, ignoring grammar and word order. It creates a vocabulary of unique words in the corpus and represents each document as a vector indicating the presence or absence of these words.

* **Example:** Consider the following three documents:
* **Document 1:** "I love cats" Document 2: "I love dogs" Document 3: "I hate spiders" To represent these documents using BoW, we create a vocabulary consisting of unique words:

Vocabulary: ["I", "love", "cats", "dogs", "hate", "spiders"] Next, we represent each document as a vector indicating the presence or absence of these words:

* Document 1: [1, 1, 1, 0, 0, 0]
* Document 2: [1, 1, 0, 1, 0, 0]
* Document 3: [1, 0, 0, 0, 1, 1]
* In this representation, each document is transformed into a fixed-length vector, where each element of the vector corresponds to a word in the vocabulary. A value of 1 indicates that the word is present in the document, while 0 indicates its absence.

1. **Term Frequency-Inverse Document Frequency (TF-IDF):** TF-IDF represents text by assigning a weight to each word based on its frequency in a document and its rarity across the entire corpus. It emphasizes words that are important within a document but relatively rare across the corpus.

* **Example:** Using the same documents as above, let's calculate the TF-IDF values for the words. TF (Term Frequency) measures how frequently a word appears in a document. IDF (Inverse Document Frequency) measures the rarity of a word across the entire corpus. Assuming a logarithmic IDF calculation, the TF-IDF values for the words in the documents would be:
* Document 1: [0.176, 0.176, 0.176, 0, 0, 0]
* Document 2: [0.176, 0.176, 0, 0.176, 0, 0]
* Document 3: [0.176, 0, 0, 0, 0.176, 0.176]
* In this representation, each document is still transformed into a fixed-length vector, but the values are now real numbers representing the TF-IDF weights for the corresponding words. Words that are more important within a document and rare across the corpus receive higher weights. These text representation techniques (BoW and TF-IDF) provide numerical representations of text that can be used as features for machine learning algorithms.

**Q.5) What is word Embedding. Explain in detail?**

* **Word embedding** is a technique used in natural language processing (NLP) to represent words or phrases as dense vectors in a high-dimensional space. It aims to capture the semantic relationships and contextual meanings of words based on their usage in a large corpus of text data. Word embeddings have become a powerful tool in various NLP tasks, such as language modeling, sentiment analysis, and machine translation.
* **Here's an explanation of word embedding in more detail:**
* **Representation of Words:** Traditionally, words in NLP were represented using one-hot encoding, where each word is represented as a sparse binary vector with a 1 at the index corresponding to the word's position in a predefined vocabulary and 0s elsewhere. However, one-hot encoding is high-dimensional and lacks semantic information.
* **Dense Vector Representation:** Word embedding provides a more compact and meaningful representation by mapping words to dense vectors in a continuous vector space. Each dimension of the vector represents a specific aspect or feature of the word's meaning.
* **Learning Word Embeddings:** Word embeddings are typically learned using neural network models, such as Word2Vec, GloVe (Global Vectors for Word Representation), or FastText. These models are trained on large amounts of text data to learn the vector representations of words.
* **Semantic Relationships and Contextual Meaning:** One of the key advantages of word embeddings is that they capture semantic relationships between words. Words with similar meanings or in similar contexts tend to have similar vector representations, and their vectors are closer in the vector space. For example, the vectors for "king" and "queen" are expected to be closer than those for "king" and "banana."
* **Arithmetic Operations on Word Vectors:** Another interesting property of word embeddings is that vector arithmetic operations can be performed on them to yield meaningful results. For instance, by subtracting the vector for "man" from the vector for "king" and adding the vector for "woman," the resulting vector is closest to the vector for "queen."
* **Pre-trained Word Embeddings:** Pre-trained word embeddings are word embeddings that are trained on large-scale corpora and made publicly available. These embeddings can be readily used in NLP tasks, even with limited amounts of data, and can provide a good starting point for various applications.

**Q.6) What is Recurrent Neural Network. Explain some application of RNN?**

* **A Recurrent Neural Network (RNN)** is a type of neural network architecture designed to process sequential data by introducing connections that allow information to persist from previous steps. Unlike traditional feedforward neural networks, which process inputs independently, RNNs have a form of memory that enables them to retain information from previous inputs and use it in the current step. This memory allows RNNs to handle sequential and time-series data efficiently.
* **Applications of RNN:**
* **Language Modeling:** RNNs are widely used for language modeling tasks, such as predicting the next word in a sentence or generating coherent text. The ability of RNNs to capture contextual dependencies makes them effective for modeling language patterns.
* **Machine Translation:** RNNs, particularly a variant called the Sequence-to-Sequence (Seq2Seq) model, have been successful in machine translation tasks. Seq2Seq models use an encoder RNN to encode the source sentence and a decoder RNN to generate the target translation.
* **Speech Recognition:** RNNs, specifically the Long Short-Term Memory (LSTM) variant, have been extensively used in speech recognition systems. LSTM-RNNs can effectively model temporal dependencies in audio data and convert spoken language into written text.
* **Sentiment Analysis:** RNNs are utilized in sentiment analysis to classify the sentiment of a text, whether it is positive, negative, or neutral. By considering the context and sequence of words, RNNs can capture the sentiment expressed throughout a sentence or document.
* **Time Series Analysis:** RNNs are well-suited for time series analysis tasks, such as stock market prediction, weather forecasting, and energy load forecasting. RNNs can capture temporal dependencies in the data and make predictions based on historical information.
* **Handwriting Recognition:** RNNs, particularly the Connectionist Temporal Classification (CTC) model, have been employed in handwriting recognition systems. RNNs can process sequential pen strokes and decode them into meaningful text.

**Q.7) Explain with forward and back propagation pass in neural network.**

* **A Recurrent Neural Network** (RNN) is a type of artificial neural network that is designed to process sequential data. Unlike traditional feedforward neural networks, which process inputs in a one-directional flow, RNNs have feedback connections that allow them to maintain and update an internal state, or memory, as they process each input in a sequence. **Now let's understand how the forward and backpropagation passes work in an RNN:**

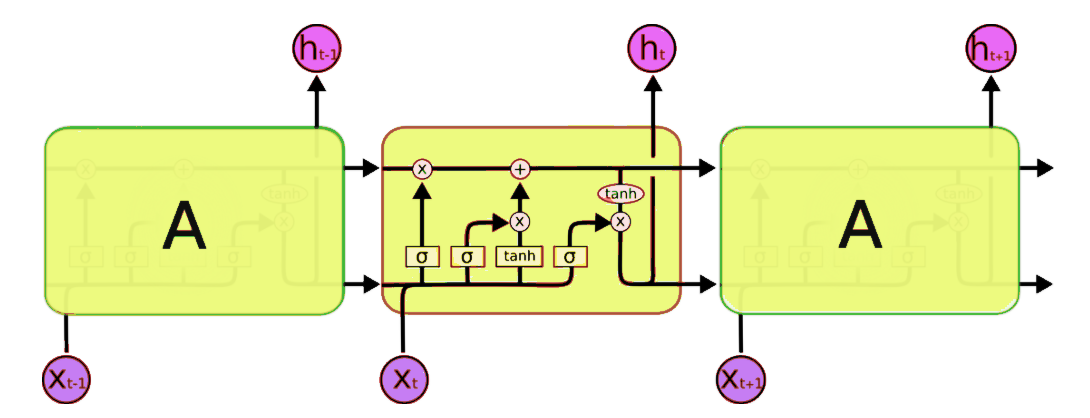
1. **Forward Propagation:** During forward propagation, the RNN processes each input in the sequence one by one. Let's assume we have a sequence of inputs x = [x₁, x₂, ..., xₙ], where each xₜ represents the input at time step t.
2. **Initialize the hidden state:** At time step t = 0, the RNN initializes its hidden state h₀ to a fixed value or zeros.
3. **Calculate the hidden state:** For each time step t, the RNN calculates the hidden state hₜ based on the current input xₜ and the previous hidden state hₜ₋₁. It uses a set of weights W and biases b to perform the calculations. The specific calculation can vary depending on the type of RNN architecture (e.g., simple RNN, LSTM, GRU).
4. **Calculate the output:** The RNN can generate an output at each time step t based on the current hidden state hₜ. It can use the same set of weights and biases as the calculation of the hidden state, or it can have separate weights and biases for the output.
5. **Repeat steps** b) and c) for each time step t in the sequence.
6. **Backpropagation:** After forward propagation, the RNN compares the generated outputs with the desired outputs to calculate the loss/error. The loss represents the discrepancy between the predicted and target outputs.
7. **Calculate the loss:** Using a suitable loss function (e.g., mean squared error, cross-entropy), the RNN computes the loss by comparing the predicted outputs with the target outputs.
8. **Backpropagate the error:** The error is then backpropagated through time, starting from the last time step t = n and moving backward. The RNN calculates the gradient of the loss with respect to the parameters (weights and biases) at each time step.
9. **Update the parameters:** The gradients obtained from backpropagation are used to update the parameters of the RNN (weights and biases) using an optimization algorithm such as gradient descent or its variants.
10. Repeat steps a) to c) for a certain number of iterations or until convergence. By iteratively going through the forward and backpropagation passes, the RNN adjusts its parameters to minimize the loss and improve its ability to capture sequential patterns in the data.

**Q.8) What are the limitation of RNN?**

* **Limitation of RNN:**
* **Difficulty in Capturing Long-Term Dependencies:** RNNs face challenges in capturing long-term dependencies, especially when the sequences are very long. The gradients may vanish or explode during backpropagation, making it difficult for the network to propagate information effectively over long distances. This limitation hinders the ability of RNNs to capture and utilize information from distant past steps.
* **Limited Memory Capacity:** The memory capacity of standard RNNs, such as basic RNN or LSTM, is limited. They struggle to retain and utilize information from the entire history of the input sequence, particularly when the sequences are very long. This limitation can hinder the performance of RNNs in tasks that require a deeper understanding of the context and longer dependencies.
* **Computational Inefficiency:** RNNs are inherently sequential in nature, as the current step's output depends on the previous step's output. This sequential computation limits parallelization, making RNNs slower compared to other network architectures, such as convolutional neural networks (CNNs) or transformers, which can process inputs in parallel. The computational inefficiency can be a challenge when training and deploying RNN models, especially for large-scale datasets.
* **Lack of Attention Mechanism:** Basic RNN and LSTM models do not inherently incorporate an attention mechanism. Attention mechanisms allow the model to focus on relevant parts of the input sequence while making predictions. Lack of attention can limit the network's ability to handle long sequences effectively and attend to important information selectively.
* **Difficulty in Handling Variable-Length Inputs:** RNNs typically expect fixed-length inputs. However, many real-world applications involve variable-length sequences, such as text of varying lengths or speech recordings of different durations. Handling variable-length inputs with RNNs requires additional techniques, such as padding or truncation, which can introduce challenges and potentially impact performance.
* **Gradient-Based Optimization Challenges:** Training deep RNNs can be challenging due to issues like vanishing and exploding gradients. These issues can make the optimization process unstable and hinder the convergence of the network. Additional techniques, such as gradient clipping or regularization, may be required to alleviate these problems.

**Q.9) Describe the architecture of lstm?**

* **Architecture of LSTMs:**



A typical LSTM network is comprised of different memory blocks called **cells**(the rectangles that we see in the image)**.** There are two states that are being transferred to the next cell; the **cell state** and the**hidden state**. The memory blocks are responsible for remembering things and manipulations to this memory is done through three major mechanisms, called **gates**. Whenever a new event occurs you take either of the three steps.

### 1.1 Forget Gate: Taking the example of a text prediction problem. Let’s assume an LSTM is fed in, the following sentence: Bob is a Nice person. Dan on the other hand is evil.

|  |
| --- |
|  |

### As soon as the first full stop after “person” is encountered, the forget gate realizes that there may be a change of context in the next sentence. As a result of this, the subject of the sentence is forgotten and the place for the subject is vacated. And when we start speaking about “Dan” this position of the subject is allocated to “Dan”. This process of forgetting the subject is brought about by the forget gate.

### The sigmoid function outputs a vector, with values ranging from 0 to 1, corresponding to each number in the cell state. Basically, the sigmoid function is responsible for deciding which values to keep and which to discard. If a ‘0’ is output for a particular value in the cell state, it means that the forget gate wants the cell state to forget that piece of information completely. Similarly, a ‘1’ means that the forget gate wants to remember that entire piece of information. This vector output from the sigmoid function is multiplied to the cell state.

### 1.2 Input Gate: let’s take another example where the LSTM is analyzing a sentence: Bob Knows Swimming. He told me over the Phone that he had Served the navy for 4 long years.

### Now the important information here is that “Bob” knows swimming and that he has served the Navy for four years. This can be added to the cell state, Here is its structure:

|  |
| --- |
|  |

1. Regulating what values need to be added to the cell state by involving a sigmoid function. This is basically very similar to the forget gate and acts as a filter for all the information from h\_t-1 and x\_t.
2. Creating a vector containing all possible values that can be added (as perceived from h\_t-1 and x\_t) to the cell state. This is done using the **tanh**function, which outputs values from -1 to +1.
3. Multiplying the value of the regulatory filter (the sigmoid gate) to the created vector (the tanh function) and then adding this useful information to the cell state via addition operation.

|  |
| --- |
| https://cdn.analyticsvidhya.com/wp-content/uploads/2017/12/10131340/18.png |

### 1.3 Output Gate: Not all information that runs along the cell state, is fit for being output at a certain time. We’ll visualize this with an example: Bob fought single handedly with the enemy and died for his country. For his contribution brave\_\_\_\_\_\_\_\_

* In this phrase, there could be a number of options for the empty space. But we know that the current input of ‘brave’, is an adjective that is used to describe a noun. Here is its structure: **The functioning of an output gate can again be broken down to three steps:**

1. Creating a vector after applying **tanh**function to the cell state, thereby scaling the values to the range -1 to +1.
2. Making a filter using the values of h\_t-1 and x\_t, such that it can regulate the values that need to be output from the vector created above. This filter again employs a sigmoid function.
3. Multiplying the value of this regulatory filter to the vector created in step 1, and sending it out as a output and also to the hidden state of the next cell. The filter in the above example will make sure that it diminishes all other values but ‘Bob’.

|  |
| --- |
| architecture of Gated Recurrent Unit |

**Q.10) Describe the architecture of GRU.**

## The architecture of Gated Recurrent Unit: At each timestamp t, it takes an input Xt and the hidden state Ht-1 from the previous timestamp t-1. Later it outputs a new hidden state Ht which again passed to the next timestamp.

|  |
| --- |
| Gated recurrent unit - Reset Gate (Short term memory) |

## Reset Gate (Short term memory): The Reset Gate is responsible for the short-term memory of the network i.e the hidden state (Ht). Here is the equation of the Reset gate. If you remember from the LSTM gate equation it is very similar to that.

|  |
| --- |
| Gated recurrent unit - Update Gate (Long Term memory) |

## Update Gate (Long Term memory): Similarly, we have an Update gate for long-term memory and the equation of the gate is shown below. The only difference is of weight metrics i.e Uu and Wu.

## How GRU Works: The functioning of these gates. To find the Hidden state Ht in GRU, it follows a two-step process.

|  |
| --- |
| Candidate Hidden State 2 |

## Candidate Hidden State: It takes in the input and the hidden state from the previous timestamp t-1 which is multiplied by the reset gate output rt. Later passed this entire information to the tanh function, the resultant value is the candidate’s hidden state.

|  |
| --- |
| Hidden state |

## Hidden state: Once we have the candidate state, it is used to generate the current hidden state Ht. It is where the Update gate comes into the picture. Now, this is a very interesting equation, instead of using a separate gate like in LSTM in GRU we use a single update gate to control both the historical information which is Ht-1 as well as the new information which comes from the candidate state. Now assume the value of ut is around 0 then the first term in the equation will vanish which means the new hidden state will not have much information from the previous hidden state.

|  |
| --- |
| timestamp t-1 |

## On the other hand, the second part becomes almost one that essentially means the hidden state at the current timestamp will consist of the information from the candidate state only. Hence we can conclude that the value of ut is very critical in this equation and it can range from 0 to 1.

|  |
| --- |
| candidate state only |

**Q.11) Describe the various steps involved in text generation using lstm?**

* **Text generation using LSTM (Long Short-Term Memory) involves several steps, which include:**

1. **Data Preparation:** The first step is to gather and preprocess the text data that will be used for training the LSTM model. This involves tasks such as tokenization, creating a vocabulary, and encoding the text data into numerical representations that can be fed into the LSTM model.
2. **LSTM Model Architecture:** Designing the architecture of the LSTM model involves determining the number of LSTM layers, the size of the hidden state, and the number of neurons in each layer.
3. **Training the LSTM Model:** The LSTM model is trained using the prepared text data. The training process involves feeding sequences of input data to the LSTM model and adjusting the model's weights using backpropagation and gradient descent algorithms. The objective is to minimize a loss function that quantifies the difference between the predicted and actual next word or character.
4. **Generating Seed Text:** To initiate the text generation process, a seed text is provided as the initial input to the trained LSTM model. This seed text can be a prompt or a few words to guide the generation process.
5. **Predicting Next Word/Character:** Using the seed text as input, the LSTM model predicts the probability distribution over the vocabulary for the next word or character. This distribution represents the likelihood of each word or character being the next one in the sequence.
6. **Sampling the Next Word/Character:** Based on the probability distribution obtained from the LSTM model, a word or character is sampled as the next one in the generated text. The sampling process can be deterministic (choosing the word with the highest probability) or stochastic (sampling randomly based on the probabilities).
7. **Updating the Seed Text:** The generated word or character is appended to the seed text, creating an updated sequence. This updated sequence is used as the input for the next prediction step.
8. **Repeating the Prediction Process:** Steps 5 to 7 are repeated iteratively to generate the desired length of text. The LSTM model takes the updated sequence as input, predicts the next word/character, and the process continues until the desired length is reached or a termination condition is met.
9. **Post-processing the Generated Text:** Once the text generation is complete, post-processing steps can be applied, such as converting the numerical representation back into human-readable text, removing any unwanted artifacts, and improving the coherence and readability of the generated text.

**Q.12) Describe various steps involve in text classification in deep learning?**

* **Text classification in deep learning involves several steps, which include:**
* **Data Collection and Preprocessing:** The first step is to collect a dataset that consists of labeled text examples. This dataset should cover different classes or categories that you want the model to classify. Once the dataset is collected, preprocessing steps are performed, such as removing special characters, tokenizing the text into words or subwords, and converting the text into numerical representations.
* **Embedding Representation:** Text data needs to be transformed into a numerical representation that deep learning models can process. This is typically done using word embeddings, such as Word2Vec, GloVe, or FastText.
* **Model Architecture Design:** Choose an appropriate deep learning architecture for text classification, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), or Transformers. The architecture should be designed to take the word embeddings as input and output the class probabilities or predictions.
* **Model Training:** The next step is to train the text classification model using the labeled dataset. The training process involves feeding the preprocessed and embedded text data into the model, adjusting the model's weights using backpropagation and gradient descent, and optimizing a suitable loss function, such as categorical cross-entropy.
* **Validation and Hyperparameter Tuning:** During training, it is important to monitor the model's performance on a validation dataset. Hyperparameters, such as learning rate, batch size, or number of layers, may need to be tuned to improve the model's performance.
* **Evaluation:** Once the model is trained, it is evaluated on a separate test dataset to assess its generalization and performance. Common evaluation metrics for text classification include accuracy, precision, recall, and F1-score. These metrics provide insights into how well the model is classifying the text examples across different categories.
* **Prediction:** After the model has been trained and evaluated, it can be used for making predictions on new, unseen text data. The text is preprocessed, embedded, and fed into the trained model, which then assigns a predicted class label or outputs the probabilities of each class.
* **Model Deployment:** If the model performs satisfactorily, it can be deployed for real-world use cases. This involves integrating the model into a production system or application, where it can classify text inputs in real-time.

**Q.13) Explain in detail architecture of Transformer network?**

* The architecture of the Transformer network consists of two main components: the encoder and the decoder. The multi-head self-attention mechanism and the position-wise feed-forward neural network.
* **Encoder:** The encoder takes an input sequence, typically a series of word embeddings, and transforms it into a sequence of higher-level representations. Each encoder layer consists of two sub-layers:
  1. **Multi-Head Self-Attention:** This sub-layer allows the model to attend to different positions in the input sequence to capture dependencies between words. It computes attention weights for each word in the sequence by comparing it with all other words in the sequence.
  2. **Position-Wise Feed-Forward Neural Network:** This sub-layer applies a feed-forward neural network to each position separately and identically. It consists of two linear transformations with a ReLU activation function in between.

|  |
| --- |
| A schematic of the decoder block |

* **Decoder:** The decoder takes the encoded sequence and generates an output sequence, one word at a time. Similar to the encoder, the decoder also consists of multiple identical layers, each with two sub-layers:
  1. **Masked Multi-Head Self-Attention:** This sub-layer ensures that during training, each position can only attend to earlier positions, preventing the model from cheating by looking ahead.
  2. **Multi-Head Encoder-Decoder Attention:** This sub-layer attends to the encoded input sequence to incorporate information from the entire input.
  3. **Position-Wise Feed-Forward Neural Network:** Same as in the encoder, this sub-layer applies a feed-forward neural network to each position separately.
* **Additional Components:**
* **Positional Encoding:** Since Transformers do not inherently encode the order of the input sequence, positional encoding is used to inject position information into the word embeddings.
* **Residual Connections and Layer Normalization:** Each sub-layer in the encoder and decoder has residual connections, allowing the model to access both the original input and the output of the sub-layer.