**NLP ASSIGNMENT\_6**

**1.What are Vanilla autoencoders**

Autoencoders are a type of neural network architecture where the goal is to learn a compact representation (also known as encoding) of the input data, and then use this encoding to recreate the original input data (reconstruction). Vanilla autoencoders are the simplest form of autoencoders and do not use any additional techniques such as convolution or recurrent layers. They consist of an encoder, a bottleneck (compressed representation) layer, and a decoder. The encoder maps the input data to the bottleneck representation, and the decoder maps the bottleneck representation back to the original input data. The objective of a vanilla autoencoder is to minimize the reconstruction error between the original input and its reconstructed version.

**2. What are Sparse autoencoders**

Sparse autoencoders are a variant of vanilla autoencoders that enforce sparsity on the bottleneck representation, i.e. encourage the model to use a small number of activations to represent the input data. The idea behind this is that by forcing the model to learn a more compact and interpretable representation, it will be better suited for tasks such as dimensionality reduction or feature extraction. To enforce sparsity, an additional sparsity constraint is added to the loss function during training, which penalizes the model for producing dense representations. This constraint can be implemented in several ways, such as adding a regularization term based on the Kullback-Leibler divergence between the desired sparsity level and the actual sparsity of the representation. By training the autoencoder with this added constraint, it will learn to produce sparse, compact representations of the input data.

**3. What are Denoising autoencoders**

Denoising autoencoders are a variant of autoencoders that are trained to reconstruct the original input data from a corrupted version of it. The idea behind denoising autoencoders is to train the model to learn robust representations of the data that are invariant to small amounts of noise or corruption. To achieve this, during training, random noise is added to the input data before it is passed through the encoder and decoder. The objective is to reconstruct the original, uncorrupted input from the noisy version, effectively "denoising" the data. By doing so, the model is forced to learn a more robust and meaningful representation of the data that is not easily affected by small amounts of noise or corruption. This can lead to improved performance on tasks such as image classification or data compression, as the model is able to generalize better to new, unseen data.

**4. What are Convolutional autoencoders**

Convolutional autoencoders are a type of autoencoder that make use of convolutional layers, which are a fundamental building block of convolutional neural networks (CNNs). Convolutional autoencoders are specifically designed to handle data with grid-like structure, such as images, and use convolutional layers in the encoder and decoder instead of fully connected layers. The idea behind using convolutional layers is to capture local spatial correlations in the input data, which are important for tasks such as image denoising or compression. Convolutional autoencoders can also be extended to include pooling layers, which help reduce the spatial size of the representation and reduce the number of parameters in the model. By using convolutional and pooling layers, convolutional autoencoders can learn more compact and meaningful representations of image data while requiring fewer parameters than fully connected autoencoders.

**5. What are Stacked autoencoders**

Stacked autoencoders are a type of deep neural network that are constructed by stacking multiple autoencoder layers on top of each other. The idea behind stacked autoencoders is to use a sequence of autoencoders to learn hierarchical representations of the input data, where each layer learns a more abstract representation of the data compared to the previous layer. The first layer of the network learns a basic representation of the input data, while the deeper layers learn increasingly complex and abstract representations.

Stacked autoencoders can be trained in an unsupervised manner, where each layer is trained to reconstruct the output of the previous layer. The final layer of the stacked autoencoder is typically a classifier that uses the learned hierarchical representations for a specific task, such as image classification. Stacked autoencoders can also be fine-tuned using supervised learning, where the final layer is trained to minimize the prediction error on a labeled dataset. By using multiple layers, stacked autoencoders can capture more complex and abstract features in the input data, leading to improved performance on many tasks.

**6. Explain how to generate sentences using LSTM autoencoders**

Generating sentences using LSTM autoencoders involves training an LSTM-based autoencoder to reconstruct sequences of words, where the input and output sequences are the same. The goal is to train the autoencoder to capture the underlying patterns and dependencies in the input data, and use this information to generate new, novel sentences.

The general steps for generating sentences using LSTM autoencoders are as follows:

Prepare the training data: This involves transforming a corpus of text into a sequence of numerical values, where each word is represented by a unique integer.

Train the autoencoder: The autoencoder is trained on the numerical sequences using an LSTM-based architecture. The encoder takes the input sequence and maps it to a hidden representation, while the decoder takes the hidden representation and maps it back to the original input sequence.

Generate sentences: Once the autoencoder is trained, the decoder can be used to generate new sentences. This is typically done by passing a "seed" sequence, such as a single word or a short phrase, through the encoder to obtain a hidden representation. Then, the decoder is used to generate a new sequence one word at a time, where each word is selected based on the probabilities predicted by the decoder for the next word given the current hidden representation and the previously generated words.

Refine the generated sentences: The generated sentences can be further refined by adding constraints, such as controlling the length of the sentences or enforcing grammatical rules. Additionally, the autoencoder can be fine-tuned on a specific task, such as machine translation or text summarization, to generate more specific and relevant sentences.

It is important to note that LSTM autoencoders are a relatively simple and flexible approach to generating text, but they can suffer from the issue of repetitive or boring generated sentences, and might not always generate coherent or meaningful sentences without additional constraints or fine-tuning.

**7. Explain Extractive summarization**

Extractive summarization is a type of text summarization that involves selecting the most important sentences or phrases from an input document and combining them to create a concise and informative summary of the original text. In extractive summarization, the summary is typically composed of contiguous pieces of the original text, rather than being generated as a new, synthetic text.

The process of extractive summarization can be performed in several ways, including:

Keyword-based summarization: This involves identifying the most frequent or important keywords in the input document and selecting the sentences that contain those keywords to form the summary.

Sentence scoring-based summarization: This involves scoring each sentence in the input document based on features such as sentence length, position, or centrality in the text, and selecting the top-scoring sentences to form the summary.

Latent semantic analysis (LSA): This involves using mathematical techniques, such as singular value decomposition, to identify the most important concepts in the input document and selecting the sentences that best represent those concepts to form the summary.

The goal of extractive summarization is to create a concise and informative summary of the original text that retains its most important content and meaning. Extractive summarization is often used for information retrieval and natural language processing applications, such as news aggregation, email summarization, and document retrieval. It is a simple and straightforward approach that can be easily implemented, but it can sometimes result in summaries that are not coherent or that lack context, especially when the selected sentences are taken out of their original context.

**8. Explain Abstractive summarization**

Abstractive summarization is a type of text summarization that involves generating new, synthetic text that summarizes the content of an input document. Unlike extractive summarization, which selects the most important sentences or phrases from the original text to form the summary, abstractive summarization creates a new and more compact representation of the original text that summarizes its most important content and meaning.

The process of abstractive summarization typically involves several steps, including:

Understanding the input text: This involves using natural language processing techniques to analyze the input text and identify its main ideas, concepts, and relationships.

Generating a summary: This involves using the information obtained from the input text to generate a new and concise summary that retains its most important content and meaning. This is typically done by using machine learning models, such as deep learning models, to generate a new text that summarizes the input text.

Evaluating the summary: This involves evaluating the quality of the generated summary based on metrics such as coherence, accuracy, and relevance to the original text.

The goal of abstractive summarization is to create a new and concise summary that accurately represents the most important content and meaning of the original text. Abstractive summarization is more challenging than extractive summarization, as it requires the ability to understand and generate natural language text. However, it can result in more coherent and informative summaries, especially when the original text is long or complex.

**9. Explain Beam search**

Beam search is a heuristic search algorithm that is commonly used in natural language processing and other domains to generate sequences, such as text or speech. The main idea behind beam search is to keep track of a small number of the most promising candidate sequences, rather than exploring the entire search space, in order to reduce the computational complexity of the search process.

The basic steps of beam search are as follows:

Initialize the search with a set of candidate sequences, typically consisting of the start symbol of the desired sequence.

Evaluate the probability of each candidate sequence using a scoring function, such as a language model, and keep only the K best-scoring candidates.

Generate the next set of candidate sequences by extending each of the K best-scoring candidates with the next symbol in the desired sequence.

Repeat steps 2 and 3 until a candidate sequence reaches the end of the desired sequence, or a maximum number of iterations has been reached.

Select the best-scoring candidate sequence as the final output of the beam search.

The parameter K, known as the beam width, determines the number of candidate sequences that are kept and explored at each iteration of the beam search. A larger beam width results in a more exhaustive search, but also increases the computational complexity of the algorithm. A smaller beam width results in a faster and less exhaustive search, but also increases the risk of missing the best candidate sequence.

Beam search is a powerful and flexible technique that can be applied to a wide range of problems, including machine translation, speech recognition, and text generation. It is especially useful when the search space is very large and the optimal solution is not easily predictable.

**10. Explain Length normalization**

Length normalization is a technique used to adjust the scores of sequences generated by machine learning models, such as language models, in order to take into account the length of the sequences. The main idea behind length normalization is to penalize longer sequences, as they are typically less likely to be correct, while encouraging shorter sequences, which are more concise and to the point.

Length normalization can be performed in several ways, including:

Division-based normalization: This involves dividing the score of each sequence by its length to obtain a normalized score that takes into account the length of the sequence.

Exponential normalization: This involves raising the score of each sequence to a power less than 1, to penalize longer sequences and encourage shorter sequences.

Additive normalization: This involves adding a term to the score of each sequence that depends on its length, to penalize longer sequences and encourage shorter sequences.

Length normalization is a critical step in many natural language processing applications, such as machine translation, text generation, and summarization, as it helps to ensure that the generated sequences are concise, to the point, and of high quality. Without length normalization, machine learning models can generate very long and verbose sequences that are unlikely to be correct, and that do not reflect the desired characteristics of the target language.

**11. Explain Coverage normalization**

Coverage normalization is a technique used in machine translation and other natural language processing tasks to address the problem of repeated content in generated sequences. The main idea behind coverage normalization is to keep track of the source words or phrases that have already been translated and to penalize the model for generating repetitions of previously translated content.

The basic steps of coverage normalization are as follows:

Initialize a coverage vector that keeps track of the source words or phrases that have already been translated.

At each step of the sequence generation process, update the coverage vector to reflect the source words or phrases that have been translated in the current step.

Compute a coverage penalty for the current step based on the current state of the coverage vector. This penalty can be based on the percentage of source words or phrases that have been covered, the number of repeated words or phrases, or any other measure of the quality of the coverage.

Add the coverage penalty to the score of the current step to obtain a modified score that takes into account the quality of the coverage.

Repeat steps 2 to 4 until the end of the sequence is reached or a maximum number of iterations has been reached.

Coverage normalization is an important technique for avoiding repeated content and improving the quality of generated sequences in machine translation and other natural language processing tasks. By penalizing repeated content, coverage normalization encourages the model to generate more diverse and informative sequences that accurately reflect the meaning and content of the source text.

**12. Explain ROUGE metric evaluation**

The ROUGE (Recall-Oriented Understudy for Gisting Evaluation) metric is a commonly used evaluation metric for summarization tasks, particularly in the field of automatic summarization. ROUGE measures the similarity between an automatically generated summary and a reference summary, based on overlapping n-grams (sequences of words).

ROUGE has several variants, including ROUGE-N, ROUGE-L, and ROUGE-W, which differ in the way that they measure the overlap between the generated summary and the reference summary.

ROUGE-N: This metric measures the overlap between the generated summary and the reference summary based on the number of common n-grams, where n is a specified parameter.

ROUGE-L: This metric is similar to ROUGE-N, but it takes into account the longest common contiguous subsequence between the generated summary and the reference summary.

ROUGE-W: This metric is similar to ROUGE-N, but it takes into account the weights assigned to the n-grams, typically based on their frequency in the reference summary.

The ROUGE score for a given generated summary is calculated as a recall-style metric, meaning that it measures the amount of information in the reference summary that is present in the generated summary. The ROUGE score is expressed as a value between 0 and 1, where 1 indicates that the generated summary is a perfect match to the reference summary.

ROUGE is widely used in the evaluation of summarization systems, as it provides a simple and effective way to measure the quality of generated summaries. However, it is important to note that ROUGE has some limitations, such as its dependence on the quality of the reference summary and its lack of consideration for the fluency and coherence of the generated summary.