1. **What are Vanilla autoencoders**

**Vanilla Autoencoder**

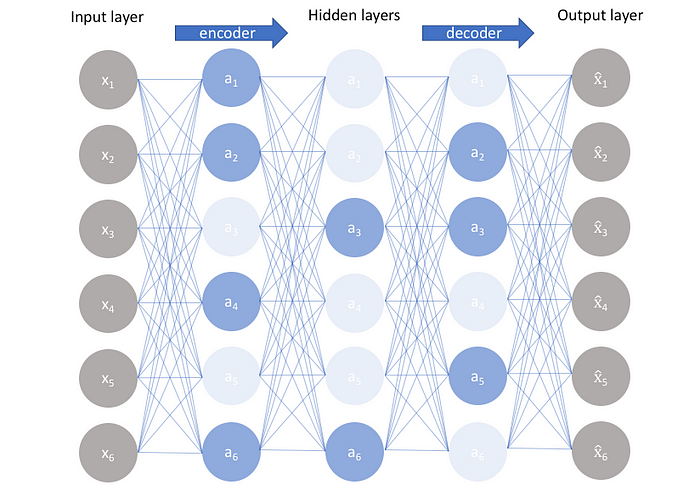
**The most basic type of [autoencoder,](https://deeplearningofpython.blogspot.com/2023/05/Autoencoder-Architecture-Keras.html" \t "_blank) which consists of a decoder network that reconstructs the input from the compressed representation after the input has been compressed by an encoder network.**

* **Purpose: The Vanilla Autoencoder is a straightforward**[**neural network**](https://deeplearningofpython.blogspot.com/2023/03/Neural%20Network-Deep%20Learning-Working-types%20of%20neural%20networks.html)**design with the aim of learning to compress input data into a low-dimensional representation and then recover the original input from this representation. This particular autoencoder’s objective is to preserve the most crucial aspects of the input data while minimizing the amount of data that is lost during compression.**
* **Inventor: In 1985, D. H. Ackley and G. E. Hinton made the initial suggestion for the Vanilla Autoencoder.**
* **Architecture: Encoder and decoder networks make up the two portions of the Vanilla Autoencoder’s architecture. The decoder network converts the lower-dimensional representation back to the original input data once the encoder network has converted the input data to it.**
* **Working: The Vanilla Autoencoder reduces the reconstruction error between the input’s original value and the decoder network’s output. Usually, a loss function like mean squared error (MSE) or**[**binary cross-entropy (BCE)**](https://deeplearningofpython.blogspot.com/2023/04/Deeplearning-implementation-example-python.html)**is used for this.**
* **Application: Image and video compression, anomaly detection, and dimensionality reduction are just a few of the many uses for vanilla autoencoders.**

1. **What are Sparse autoencoders**

**A sparse autoencoder is simply an autoencoder whose training criterion involves a sparsity penalty. In most cases, we would construct our loss function by penalizing activations of hidden layers so that only a few nodes are encouraged to activate when a single sample is fed into the network.**

**The intuition behind this method is that, for example, if a man claims to be an expert in mathematics, computer science, psychology, and classical music, he might be just learning some quite shallow knowledge in these subjects. However, if he only claims to be devoted to mathematics, we would like to anticipate some useful insights from him. And it’s the same for autoencoders we’re training — fewer nodes activating while still keeping its performance would guarantee that the autoencoder is actually learning latent representations instead of redundant information in our input data.**



Sparse Autoencoder

1. **What are Denoisingautoencoders**

# Denoising AutoEncoders In Machine Learning

**Autoencoders are types of neural network architecture used for unsupervised learning. The architecture consists of an encoder and a decoder. The encoder encodes the input data into a lower dimensional space while the decode decodes the encoded data back to the original input. The network is trained to minimize the difference between decoded data and input. Autoencoders have the risk of becoming an Identify function meaning the output equals the input which makes the whole neural network of autoecoders useless. This generally happens when there are more nodes in the hidden layer than there are inputs.**

## Denoising Autoencoder (DAE)

**Now, a denoising autoencoder is a modification of the original autoencoder in which instead of giving the original input we give a corrupted or noisy version of input to the encoder while decoder loss is calculated concerning original input only. This results in efficient learning of autoencoders and the risk of autoencoder becoming an identity function is significantly reduced.**

### Architecture of DAE

**The denoising autoencoder (DAE) architecture resembles a standard [autoencoder](https://www.geeksforgeeks.org/auto-encoders/)and consists of two main components:**

#### Encoder:

* **The encoder is a neural network with one or more hidden layers.**
* **It receives noisy input data instead of the original input and generates an encoding in a low-dimensional space.**
* **There are several ways to generate a corrupted input. The most common being adding a Gaussian noise or randomly masking some of the inputs.**

#### Decoder:

* **Similar to encoders, decoders are implemented as neural networks with one or more hidden layers.**
* **It takes the encoding generated by the encoder as input and reconstructs the original data.**
* **When calculating the Loss function it compares the output values with the original input, not with the corrupted input.**

### What DAE Learns?

**The above architecture of using a corrupted input helps decrease the risk of overfitting and prevents the DAE from becoming an identity function.**

* **If DAEs are trained with partially corrupted inputs (e.g., with masking values), they learn to impute or fill in missing information during the reconstruction process. This makes them useful for tasks involving incomplete datasets.**
* **If DAEs are trained with partially noisy inputs (gaussian noise) DAEs tend to generalize well to unseen, real-world data with different levels of noise or corruption as they learn to extract robust features. This is beneficial in various applications where data quality is compromised, such as image denoising or signal processing.**

### Objective Function of DAE

**The objective of DAE is to minimize the difference between the original input (clean input without the notice) and the reconstructed output. This is quantified using a reconstruction loss function. Two types of loss function are generally used depending on the type of input data.**

#### Mean Squared Error (MSE):

**If we have input image data in the form of floating pixel values i.e. values between (0 to 1) or (0 to 255) we use mse**

**Here,**

* **each of xi is the pixel value of input data**
* **yi is the pixel value of reconstructed data**
  + **yi = D(E(xi\*noise) )**
  + **Where E represents encoder and D represents decoder**
* **this is summed over all the training set**

#### Binary Cross-Entropy (log-loss):

**If we have input image data in the form of bits pixel values i.e. values will be either 0 or 1 only then we can use binary cross entrop loss for each pixel value**

**Here**

* **each of xi is the pixel value of input data with value being only 0 or 1**
* **yi is the pixel value of reconstructed data.**
  + **yi = D(E(xi\*noise))**
* **Where E represents encoder and D represents decoder**
* **this is summed over all the training set**

### Training Process of DAE

**The training of DAE consists of below steps:**

* **Initialze ender and decoer with random weights**
* **Noise is intentionally added to the input data.**
* **Feedforward the input data through encoder and decoder to get the reconstructed image**
* **Calculate the reconstruction loss as defined in our objective function w**
* **Do backprogoagation and update weights.The goal during training is to minimize the reconstruction loss.**

**The training is typically done through optimization algorithms like stochastic gradient descent (SGD) or its variants.**

### Applications of DAE

* **Image Denoising: DAEs are widely employed for cleaning and enhancing images by removing noise.**
* **Audio Denoising: DAEs can be applied to denoise audio signals, making them valuable in speech-enhancement tasks.**
* **Sensor Data Processing: DAEs are valuable in processing sensor data, removing noise, and extracting relevant information from sensor readings.**
* **Data Compression: Autoencoders, including DAEs, can be utilized for data compression by learning compact representations of input data.**
* **Feature Learning: DAEs are effective in unsupervised feature learning, capturing relevant features in the data without explicit labels.**

### Implementation of DAE

**Let us implement DAE in PyTorch for MNIST dataset.**

#### 1. Import Libraries

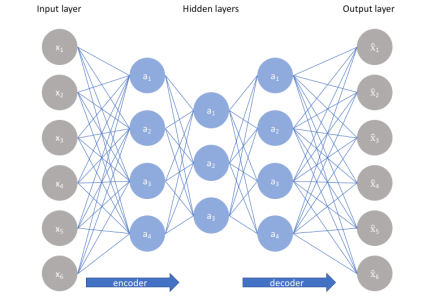
* **torch. utils.data provides tools for working with datasets and data loaders in [PyTorch](https://www.geeksforgeeks.org/getting-started-with-pytorch/).**
* **torch-vision is a PyTorch library specifically designed for**[**computer vision**](https://www.geeksforgeeks.org/computer-vision/)**tasks. datasets contain popular datasets (like MNIST, CIFAR-10, etc.), and transforms provide image transformations and preprocessing functions.**
* **nn provides building blocks for constructing**[**neural network**](https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/)**architectures, and optim includes optimization algorithms (like**[**SGD**](https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/)**,**[**Adam**](https://www.geeksforgeeks.org/intuition-of-adam-optimizer/)**, etc.) for training neural networks.**
* **If a GPU is available, it sets the device variable to ‘cuda’; otherwise, it sets it to ‘CPU’**

1. **What are Convolutional autoencoders**

**Convolutional autoencoders are a powerful type of deep learning model that can extract features from images and reconstruct them with great accuracy. They have several applications in image processing, like image denoising, dimensionality reduction, image generation, and more. Understanding how CAEs work and how to train them is important for any computer vision practitioner looking to work with images. The field of deep learning has evolved a lot in recent years, and CAEs are just one example of the many exciting developments that are happening in this area.**

1. **What are Stacked autoencoders**

**A stacked autoencoder is a multi-layer neural network that consists of multiple autoencoders, where the output of each encoder gets fed into the next encoder until the last encoder feeds its output into a chain of decoders. This allows a step by step compression and decompression of the input data to happen with more control over each state of the process. Similar to autoencoders the output of an ideal stacked autoencoder is equal to its input. However, since this is a lossy process and in the real world, they would not be equal.**



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

# Generate Text Using Autoencoders

**Copy Code  Copy Command**

**This example shows how to generate text data using autoencoders.**

**An autoencoder is a type of deep learning network that is trained to replicate its input. An autoencoder consists of two smaller networks: and encoder and a decoder. The encoder maps the input data to a feature vector in some latent space. The decoder reconstructs data using vectors in this latent space.**

**The training process is unsupervised. In other words, the model does not require labeled data. To generate text, you can use the decoder to reconstruct text from arbitrary input.**

**This example trains an autoencoder to generate text. The encoder uses a word embedding and an LSTM operation to map the input text into latent vectors. The decoder uses an LSTM operation and the same embedding to reconstruct the text from the latent vectors.**

### Load Data

**The file sonnets.txt contains all of Shakespeare's sonnets in a single text file.**

**Read the Shakespeare's Sonnets data from the file "sonnets.txt".**

**Get**

**filename = "sonnets.txt";**

**textData = fileread(filename);**

**The sonnets are indented by two whitespace characters. Remove the indentations using replace and split the text into separate lines using the split function. Remove the header from the first nine elements and the short sonnet titles.**

**Get**

**textData = replace(textData," ","");**

**textData = split(textData,newline);**

**textData(1:9) = [];**

**textData(strlength(textData)<5) = [];**

### Prepare Data

**Create a function that tokenizes and preprocesses the text data. The function preprocessText, listed at the end of the example, performs these steps:**

1. **Prepends and appends each input string with the specified start and stop tokens, respectively.**
2. **Tokenize the text using tokenizedDocument.**

**Preprocess the text data and specify the start and stop tokens "<start>" and "<stop>", respectively.**

**Get**

**startToken = "<start>";**

**stopToken = "<stop>";**

**documents = preprocessText(textData,startToken,stopToken);**

**Create a word encoding object from the tokenized documents.**

**Get**

**enc = wordEncoding(documents);**

**When training a deep learning model, the input data must be a numeric array containing sequences of a fixed length. Because the documents have different lengths, you must pad the shorter sequences with a padding value.**

**Recreate the word encoding to also include a padding token and determine the index of that token.**

**Get**

**paddingToken = "<pad>";**

**newVocabulary = [enc.Vocabulary paddingToken];**

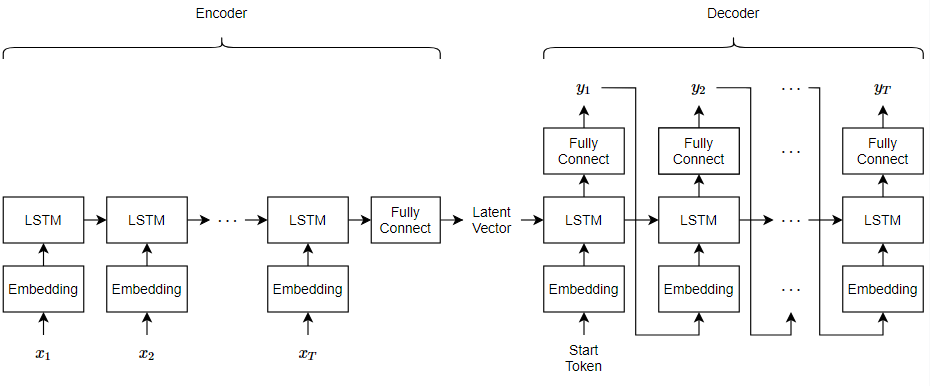
**enc = wordEncoding(newVocabulary);**

**paddingIdx = word2ind(enc,paddingToken)**

**paddingIdx = 3595**

### Initialize Model Parameters

**Initialize the parameters for the following model.**

****

**Here, *T* is the sequence length, *x*1,⋯,*xT* is the input sequence of word indices, and *y*1,⋯,*yT* is the reconstructed sequence.**

**The encoder maps sequences of word indices to a latent vector by converting the input to sequences of word vectors using an embedding, inputting the word vector sequences into an LSTM operation, and applying a fully connected operation to the last time step of the LSTM output. The decoder reconstructs the input using an LSTM initialized the encoder output. For each time step, the decoder predicts the next time step and uses the output for the next time-step predictions. Both the encoder and the decoder use the same embedding.**

**Specify the dimensions of the parameters.**

**Get**

**embeddingDimension = 100;**

**numHiddenUnits = 150;**

**latentDimension = 75;**

**vocabularySize = enc.NumWords;**

**Create a struct for the parameters.**

**Get**

**parameters = struct;**

**Initialize the weights of the embedding using the Gaussian using the initializeGaussian function which is attached to this example as a supporting file. Specify a mean of 0 and a standard deviation of 0.01. To learn more, see**[**Gaussian Initialization**](https://in.mathworks.com/help/deeplearning/ug/initialize-learnable-parameters-for-custom-training-loop.html#mw_df2a801c-4a6b-4d52-a070-1278ea628596)**.**

**Get**

**sz = [embeddingDimension vocabularySize];**

**mu = 0;**

**sigma = 0.01;**

**parameters.emb.Weights = initializeGaussian(sz,mu,sigma);**

**Initialize the learnable parameters for the encoder LSTM operation:**

* **Initialize the input weights with the Glorot initializer using the initializeGlorot function which is attached to this example as a supporting file. To learn more, see [Glorot Initialization](https://in.mathworks.com/help/deeplearning/ug/initialize-learnable-parameters-for-custom-training-loop.html" \l "mw_1bd0f2c3-c7df-4841-89ce-a7574d2db8d9).**
* **Initialize the recurrent weights with the orthogonal initializer using the initializeOrthogonal function which is attached to this example as a supporting file. To learn more, see**[**Orthogonal Initialization**](https://in.mathworks.com/help/deeplearning/ug/initialize-learnable-parameters-for-custom-training-loop.html#mw_2a30fd2b-1c69-45e3-a7ed-05f407a4da55)**.**
* **Initialize the bias with the unit forget gate initializer using the initializeUnitForgetGate function which is attached to this example as a supporting file. To learn more, see**[**Unit Forget Gate Initialization**](https://in.mathworks.com/help/deeplearning/ug/initialize-learnable-parameters-for-custom-training-loop.html#mw_975f9595-ce05-4163-a5d7-1719d1609e0a)**.**

**Get**

**sz = [4\*numHiddenUnits embeddingDimension];**

**numOut = 4\*numHiddenUnits;**

**numIn = embeddingDimension;**

**parameters.lstmEncoder.InputWeights = initializeGlorot(sz,numOut,numIn);**

**parameters.lstmEncoder.RecurrentWeights = initializeOrthogonal([4\*numHiddenUnits numHiddenUnits]);**

**parameters.lstmEncoder.Bias = initializeUnitForgetGate(numHiddenUnits);**

**Initialize the learnable parameters for the encoder fully connected operation:**

* **Initialize the weights with the Glorot initializer.**
* **Initialize the bias with zeros using the initializeZeros function which is attached to this example as a supporting file. To learn more, see**[**Zeros Initialization**](https://in.mathworks.com/help/deeplearning/ug/initialize-learnable-parameters-for-custom-training-loop.html#mw_f7c2db63-96b5-4a81-813e-ee621c9658ce)**.**

**Get**

**sz = [latentDimension numHiddenUnits];**

**numOut = latentDimension;**

**numIn = numHiddenUnits;**

**parameters.fcEncoder.Weights = initializeGlorot(sz,numOut,numIn);**

**parameters.fcEncoder.Bias = initializeZeros([latentDimension 1]);**

**Initialize the learnable parameters for the decoder LSTM operation:**

* **Initialize the input weights with the Glorot initializer.**
* **Initialize the recurrent weights with the orthogonal initializer.**
* **Initialize the bias with the unit forget gate initializer.**

**Get**

**sz = [4\*latentDimension embeddingDimension];**

**numOut = 4\*latentDimension;**

**numIn = embeddingDimension;**

**parameters.lstmDecoder.InputWeights = initializeGlorot(sz,numOut,numIn);**

**parameters.lstmDecoder.RecurrentWeights = initializeOrthogonal([4\*latentDimension latentDimension]);**

**parameters.lstmDecoder.Bias = initializeZeros([4\*latentDimension 1]);**

**Initialize the learnable parameters for the decoder fully connected operation:**

* **Initialize the weights with the Glorot initializer.**
* **Initialize the bias with zeros.**

**Get**

**sz = [vocabularySize latentDimension];**

**numOut = vocabularySize;**

**numIn = latentDimension;**

**parameters.fcDecoder.Weights = initializeGlorot(sz,numOut,numIn);**

**parameters.fcDecoder.Bias = initializeZeros([vocabularySize 1]);**

**To learn more about weight initialization, see**[**Initialize Learnable Parameters for Model Function**](https://in.mathworks.com/help/deeplearning/ug/initialize-learnable-parameters-for-custom-training-loop.html)**.**

### Define Model Encoder Function

**Create the function modelEncoder, listed in the**[**Encoder Model Function**](https://in.mathworks.com/help/deeplearning/ug/generate-text-using-autoencoders.html#GenerateTextUsingAutoencodersExample-10)**section of the example, that computes the output of the encoder model. The modelEncoder function, takes as input sequences of word indices, the model parameters, and the sequence lengths, and returns the corresponding latent feature vector. To learn more about defining a model encoder function, see**[**Define Text Encoder Model Function**](https://in.mathworks.com/help/deeplearning/ug/define-text-encoder-model-function.html)**.**

### Define Model Decoder Function

**Create the function modelDecoder, listed in the**[**Decoder Model Function**](https://in.mathworks.com/help/deeplearning/ug/generate-text-using-autoencoders.html#GenerateTextUsingAutoencodersExample-11)**section of the example, that computes the output of the decoder model. The modelDecoder function, takes as input sequences of word indices, the model parameters, and the sequence lengths, and returns the corresponding latent feature vector. To learn more about defining a model decoder function, see**[**Define Text Decoder Model Function**](https://in.mathworks.com/help/deeplearning/ug/define-text-decoder-model-function.html)**.**

### Define Model Loss Function

**The modelLoss function, listed in the**[**Model Loss Function**](https://in.mathworks.com/help/deeplearning/ug/generate-text-using-autoencoders.html#GenerateTextUsingAutoencodersExample-12)**section of the example, takes as input the model learnable parameters, the input data and a vector of sequence lengths for masking, and returns the loss, and the gradients of the loss with respect to the learnable parameters. To learn more about defining a model loss function, see**[**Define Model Loss Function for Custom Training Loop**](https://in.mathworks.com/help/deeplearning/ug/define-model-gradients-function-for-custom-training-loop.html)**.**

### Specify Training Options

**Specify the options for training.**

**Train for 100 epochs with a mini-batch size of 128.**

**Get**

**miniBatchSize = 128;**

**numEpochs = 100;**

**Train with a learning rate of 0.01.**

**Get**

**learnRate = 0.01;**

### Train Network

**Train the network using a custom training loop.**

**Initialize the parameters for the Adam optimizer.**

**Get**

**trailingAvg = [];**

**trailingAvgSq = [];**

**Initialize the training progress plot. Create an animated line that plots the loss against the corresponding iteration.**

**Get**

**figure**

**C = colororder;**

**lineLossTrain = animatedline(Color=C(2,:));**

**xlabel("Iteration")**

**ylabel("Loss")**

**ylim([0 inf])**

**grid on**

**Train the model. For the first epoch, shuffle the data and loop over mini-batches of data.**

**For each mini-batch:**

* **Convert the text data to sequences of word indices.**
* **Convert the data to dlarray.**
* **For GPU training, convert the data to gpuArray objects.**
* **Compute loss and gradients.**
* **Update the learnable parameters using the adamupdate function.**
* **Update the training progress plot.**

**Train on a GPU if one is available. Using a GPU requires Parallel Computing Toolbox™ and a supported GPU device. For information on supported devices, see**[**GPU Computing Requirements**](https://in.mathworks.com/help/parallel-computing/gpu-computing-requirements.html)**(Parallel Computing Toolbox).**

**Training can take some time to run.**

1. **Explain Extractive summarization**

## Extractive Summarization

**So, what exactly happens in the extractive summarization method? It simply takes out the important sentences or phrases from the original text and joins them to form a summary.**

**Now, the question that comes is, exactly on what basis are those sentences termed as important? So, basically, a ranking algorithm is used, which assigns scores to each of the sentences in the text based on their relevance to the overall meaning of the document. The most relevant sentences are then chosen to be included in the summary.**

**There are various ways through which the ranking of sentences can be performed.  
TF-IDF (term frequency-inverse document frequency)  
Graph-based methods such as TextRank  
Machine learning-based methods such as**[**Support Vector Machines**](https://www.analyticsvidhya.com/blog/2022/06/one-class-classification-using-support-vector-machines/)**(SVM) and**[**Random Forests**](https://www.analyticsvidhya.com/blog/2020/12/lets-open-the-black-box-of-random-forests/)**.**

**The main motive of the extractive method is to maintain the original meaning of the text. Also, this method works well when the input text/content is already in a well-structured manner, both physically and logically, just like the content in newspapers.**

1. **Explain Abstractive summarization**

## Abstractive Summarization

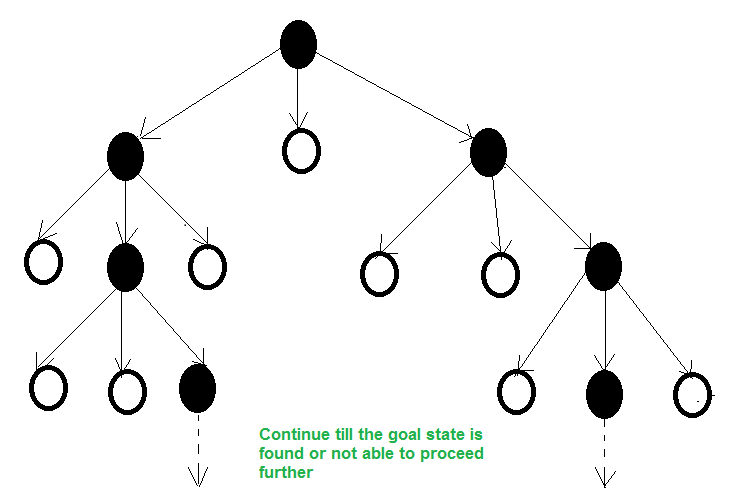
**Okay, now let’s come to the abstractive summarization method. The name itself implies that it has arrived from the root form of the word abstract, which means outline/summary or the basic idea of a voluminous thing(text). Now unlike the extractive method, it simply doesn’t pick out the important sentences, rather, it analyses the input text and generates new phrases or sentences that capture the essence of the original text and convey the same meaning as the original text but more concisely and coherently.**

**Again, how exactly is the summary generated in this method? So, in brief, the input text is analyzed by a neural network model that learns to generate new phrases and sentences that capture the essence of the original text. The model is trained on large amounts of text data and learns to understand the relationships between words and sentences, and generates new text that conveys the same meaning as the original text in a more understandable manner.**

**This method uses advanced NLP techniques such as natural language generation (NLG) and deep learning to understand the context and generate the summary. The resulting summaries are usually shorter and more readable than the ones generated by the extractive method, but they can sometimes contain errors or inaccuracies.**

1. **Explain Beam search**

**Beam Search :**  
A heuristic search algorithm that examines a graph by extending the most promising node in a limited set is known as beam search.   
Beam search is a heuristic search technique that always expands the W number of the best nodes at each level. It progresses level by level and moves downwards only from the best W nodes at each level. Beam Search uses breadth-first search to build its search tree. It generates all the successors of the current level’s state at each level of the tree. However, at each level, it only evaluates a W number of states. Other nodes are not taken into account.   
The heuristic cost associated with the node is used to choose the best nodes. The width of the beam search is denoted by W. If B is the branching factor, at every depth, there will always be W × B nodes under consideration, but only W will be chosen. More states are trimmed when the beam width is reduced.   
When W = 1, the search becomes a hill-climbing search in which the best node is always chosen from the successor nodes. No states are pruned if the beam width is unlimited, and the beam search is identified as a breadth-first search.   
The beamwidth bounds the amount of memory needed to complete the search, but it comes at the cost of completeness and optimality (possibly that it will not find the best solution). The reason for this danger is that the desired state could have been pruned.   
Example: The search tree generated using this algorithm with *W = 2 & B = 3* is given below :



*Beam Search*

1. **Explain Length normalization**

The link you provide in the question already mentions one reason for using length-normalization: to avoid having high term-frequency counts in document vectors. This affects document ranking considerably. A direct application of this is, of course, query-based document retrieval.

There are other algorithm-specific applications as well. For example, if you want to cluster documents using cosine similarity between the vectors: simple clustering algorithms such as k-means may not converge unless the vectors are all on a sphere, i.e. all vectors have the same length.

1. **Explain Coverage normalization**

**Normalization is a scaling technique in Machine Learning applied during data preparation to change the values of numeric columns in the dataset to use a common scale. It is not necessary for all datasets in a model. It is required only when features of machine learning models have different ranges.**

**Mathematically, we can calculate normalization with the below formula:**

1. **Xn = (X - Xminimum) / ( Xmaximum - Xminimum)**

* **Xn = Value of Normalization**
* **Xmaximum = Maximum value of a feature**
* **Xminimum = Minimum value of a feature**

**Example: Let's assume we have a model dataset having maximum and minimum values of feature as mentioned above. To normalize the machine learning model, values are shifted and rescaled so their range can vary between 0 and 1. This technique is also known as Min-Max scaling. In this scaling technique, we will change the feature values as follows:**

**Case1- If the value of X is minimum, the value of Numerator will be 0; hence Normalization will also be 0.**

1. **Xn = (X - Xminimum) / ( Xmaximum - Xminimum)**

**Put X =Xminimum in above formula, we get;**

**Xn = Xminimum- Xminimum/ ( Xmaximum - Xminimum)**

**Xn = 0**

**Case2- If the value of X is maximum, then the value of the numerator is equal to the denominator; hence Normalization will be 1.**

1. **Xn = (X - Xminimum) / ( Xmaximum - Xminimum)**

**Put X =Xmaximum in above formula, we get;**

**Xn = Xmaximum - Xminimum/ ( Xmaximum - Xminimum)**

**Xn = 1**

**Case3- On the other hand, if the value of X is neither maximum nor minimum, then values of normalization will also be between 0 and 1.**

**Hence, Normalization can be defined as a scaling method where values are shifted and rescaled to maintain their ranges between 0 and 1, or in other words; it can be referred to as Min-Max scaling technique.**

## Normalization techniques in Machine Learning

**Although there are so many feature normalization techniques in Machine Learning, few of them are most frequently used. These are as follows:**

* **Min-Max Scaling: This technique is also referred to as scaling. As we have already discussed above, the Min-Max scaling method helps the dataset to shift and rescale the values of their attributes, so they end up ranging between 0 and 1.**
* **Standardization scaling:**

**Standardization scaling is also known as Z-score normalization, in which values are centered around the mean with a unit standard deviation, which means the attribute becomes zero and the resultant distribution has a unit standard deviation. Mathematically, we can calculate the standardization by subtracting the feature value from the mean and dividing it by standard deviation.**

1. **Explain ROUGE metric evaluation**

**ROUGE, or Recall-Oriented Understudy for Gisting Evaluation,**[**[1]**](https://en.wikipedia.org/wiki/ROUGE_(metric)#cite_note-1)**is a set of metrics and a software package used for evaluating**[**automatic summarization**](https://en.wikipedia.org/wiki/Automatic_summarization)**and**[**machine translation**](https://en.wikipedia.org/wiki/Machine_translation)**software in**[**natural language processing**](https://en.wikipedia.org/wiki/Natural_language_processing)**. The metrics compare an automatically produced summary or translation against a reference or a set of references (human-produced) summary or translation. ROUGE metrics range between 0 and 1, with higher scores indicating higher similarity between the automatically produced summary and the reference.**