**NLP and Transformers**

# **Environment Setup:**

**Step 1:**

Click on the GitHub repository below and download the environment.yaml file, then store it in a folder created on the desktop. (Assume I have created a folder on the desktop named NLP and stored the environment.yaml file there.)

<https://github.com/jamescalam/transformers>

**Step 2:**

**I**n Anaconda Prompt, navigate to the directory containing the environment.yaml file and write the command below:

conda env create -f environment.yaml.

(In Anaconda Prompt, I need to navigate to my NLP directory and write the above command.)

**Step-3:**

Activate the new environment with conda activate ml.

(After all the libraries have been installed write the above command )

**Step-4(Installing pytorch):**

Open the PyTorch installation page.(<https://pytorch.org/>)



Select your specifications.

Copy the given command and run it in the Anaconda prompt.

**Step-5:**

After installation of pytorch library if your environment is not activated activate it using the command conda activate ml and then run the following command.

python -m ipykernel install --user --name ml --display-name "ML"

**Step 6:**

The kernel has been installed, switch back to base with

conda activate base command and then open the jupyter notebook using the jupyter notebook command.

**After completing the task, to reopen the Jupyter Notebook, follow these steps:**

Open Anaconda Prompt and navigate to your directory (in my case, NLP), then write the command:

jupyter notebook

# 

# 

# **Natural Language processing:**

Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) and linguistics that focuses on the interaction between computers and humans through natural language. It involves the development of algorithms and models to enable computers to understand, interpret, and generate human language in a way that is both meaningful and contextually appropriate.

Key tasks in natural language processing include:

Text Classification and Categorization: Assigning predefined categories or labels to text documents based on their content. This can include tasks like sentiment analysis, spam detection, topic modeling, and more.

Named Entity Recognition (NER): Identifying and classifying named entities mentioned in text into predefined categories such as names of persons, organizations, locations, dates, etc.

Semantic Analysis: Understanding the meaning of words, phrases, and sentences in context. This includes tasks like word sense disambiguation, semantic role labeling, and semantic similarity computation.

Machine Translation: Translating text from one language to another while preserving the meaning and context of the original content.

Question Answering: Building systems that can understand questions posed in natural language and provide accurate answers by extracting relevant information from text sources.

Text Generation: Creating coherent and contextually relevant text based on input prompts or predefined patterns. This includes tasks like language modeling, text summarization, and dialogue generation.

NLP techniques rely heavily on machine learning and deep learning algorithms, including neural networks and transformer models, to process and analyze large volumes of text data. These models are trained on a vast amount of annotated text to learn patterns, structures, and semantic representations of language, enabling them to perform a wide range of language understanding and generation tasks. NLP has applications in various fields, including information retrieval, healthcare, finance, customer service, and more.

## **Neural AI:**

"Neural AI" typically refers to artificial intelligence systems that are based on neural networks, a type of computational model inspired by the structure and function of biological neural networks in the human brain. In the context of NLP, neural AI often refers to deep learning models, particularly transformer-based architectures like BERT, GPT, and their variants, which have revolutionized the field of natural language processing in recent years.

**Pros:**

1. Improved Performance

2. End-to-End Learning

3. Flexibility

4. Scalability

5. Contextual Understanding

6. Transfer Learning

**Cons:**

1. Data Dependency

2. Computational Resources

3. Black Box Nature

4. Robustness and Bias

5. Domain Specificity

6. Resource Intensive Inference

## **Word Vectors:**

Word vectors, also known as word embeddings, are numerical representations of words in a high-dimensional vector space. These representations capture semantic and syntactic similarities between words based on their distributional properties in a corpus of text. Word vectors are essential in natural language processing (NLP) tasks as they enable machines to understand and process human language more effectively.

**Word2Vec:**

Word2vec is a technique used in natural language processing (NLP) to turn words into numerical representations. These representations, called word embeddings, capture the meaning of a word based on the words around it. Imagine a word embedding as a unique address for each word in a high dimensional space, where similar words are clustered together.

Here's an example:

Imagine you have a sentence like "The king is happy today". Word2vec can analyze this sentence and create embeddings for each word. The embedding for "king" might be close to the embeddings for "queen", "ruler", or "monarch" because they share similar meanings. Likewise, the embedding for "happy" might be close to "joyful" or "glad".

## 

## **Recurrent Neural Networks:**

Recurrent neural networks (RNNs) are a kind of artificial neural network designed specifically to handle sequential data. Unlike regular neural networks where each input is independent, RNNs can process information that unfolds over time.

Here's a breakdown of RNNs:

**Sequential Data:** RNNs excel at dealing with data that has a specific order, like text, speech, or even financial time series. In a sentence I Am happy today , understanding the meaning of "today" depends on the previous word "happy". RNNs can account for these dependencies.

**Memory:** A key feature of RNNs is their hidden state. This hidden layer acts like a memory, allowing the network to store information about past elements in the sequence and use it to influence the processing of current elements.

**Applications**: RNNs power a variety of tasks like machine translation, speech recognition,etc.

Imagine you're feeding the sentence "The quick brown fox jumps" into an RNN one word at a time. The RNN can consider the previous word "quick" when processing "brown", allowing it to learn the relationship between adjectives and nouns. This way, the RNN can predict the upcoming word "fox" more accurately.

**Disadvantages:**

**Vanishing Gradient Problem:**

During backpropagation in RNNs, gradients are used to update the weights of the network. In vanishing gradients, the gradients become infinitesimally small as they travel backward through the network. This makes it difficult to update the weights in the earlier layers, hindering the network's ability to learn long-term dependencies in the sequence.

**Exploding Gradients:** The opposite of vanishing gradients, exploding gradients become very large as they travel backward. This can lead to weights becoming extremely large, causing the training process to diverge and the network to become unstable.

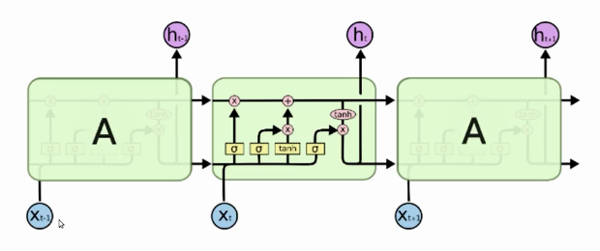
To address these issues LSTM was introduced

## **LSTM Neural Network:**

LSTMs, or Long Short-Term Memory networks, are a type of recurrent neural network (RNN) specifically designed to address the vanishing gradient problem that plagues traditional RNNs.

LSTM stands for Long Short-Term Memory, a specific type of recurrent neural network (RNN) architecture. RNNs are designed to handle sequential data, where the order of information matters. However, traditional RNNs struggle with long-term dependencies, meaning they can have difficulty remembering information from very far back in a sequence.

It excels at capturing long-term dependencies, making it ideal for sequence prediction tasks.



Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem and capture long-term dependencies in sequential data.

Key features of LSTM include:

**1. Memory Cells**: LSTMs have memory cells that allow them to store information over long periods of time. These memory cells maintain a constant state over time and can learn to retain or forget information based on the input and the current context.

**2. Gating Mechanisms:** LSTMs use gating mechanisms to control the flow of information through the network, helping to mitigate the vanishing gradient problem and facilitate the learning of long-range dependencies. The three main gates in an LSTM cell are:

**Forget Gate:** Determines which information from the previous cell state to forget.

**Input Gate:** Determines which new information to add to the cell state.

**Output Gate**: Determines which information to output from the current cell state.

**3. Cell State:** LSTMs maintain a cell state that runs through the entire sequence, allowing information to persist over time. The cell state is updated dynamically based on the input, previous cell state, and gate outputs, enabling the network to capture dependencies across long sequences.

By effectively managing information flow and memory retention, LSTM networks can learn complex patterns and dependencies in sequential data, making them well-suited for tasks that require understanding and processing of time series or sequential information. They have become a fundamental building block in modern deep learning architectures for sequence modeling and have contributed to significant advancements in various fields, including natural language understanding, speech recognition, and machine translation.

To improve long range dependencies transformers were introduced.

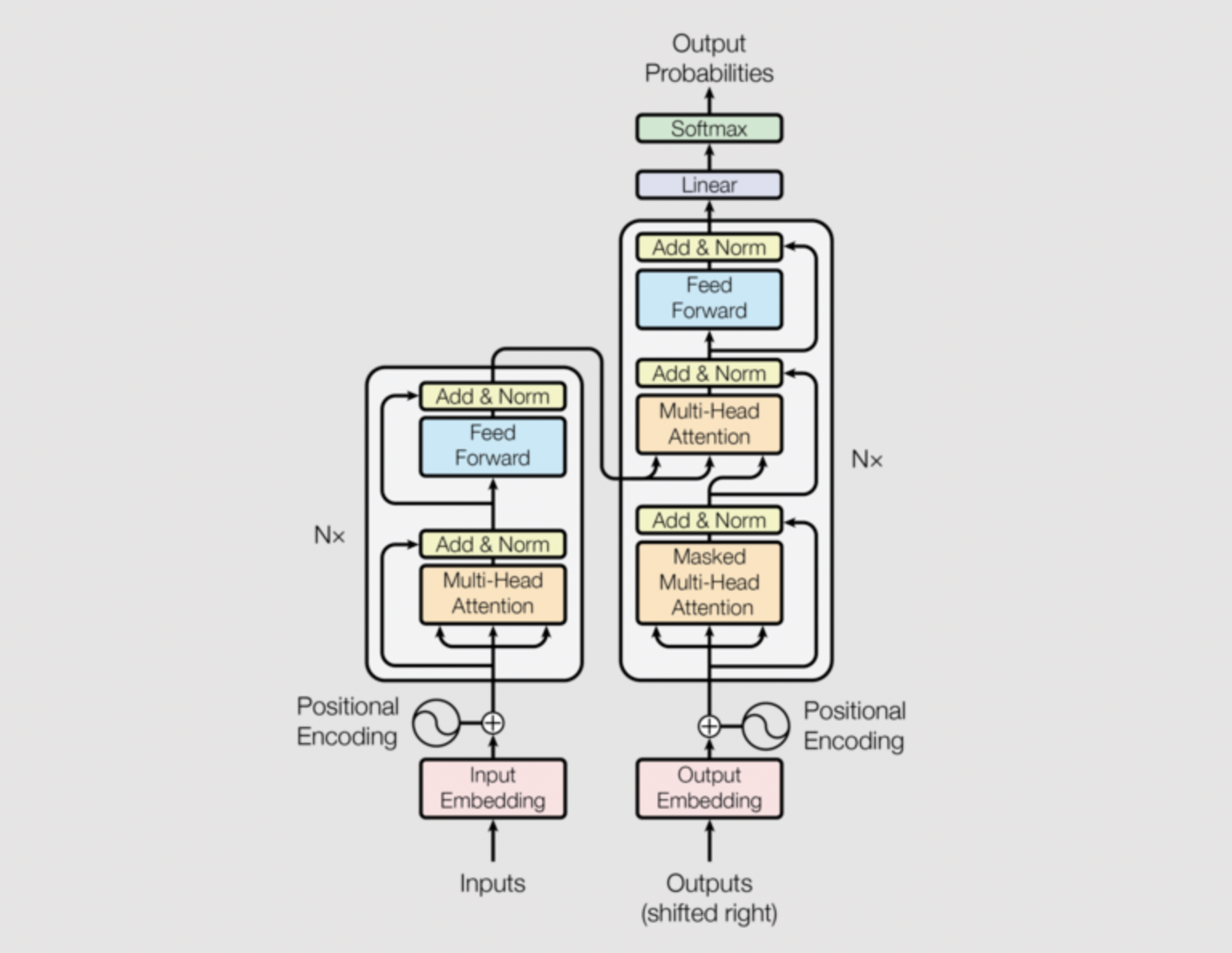
# **Transformer:**

A Transformer is a type of artificial intelligence model designed to understand and generate language. It's particularly good at tasks like translation, summarization, and answering questions.

# **Working:**

The transformer has two main components encoder and decoder they work

to generate text



## **Encoder**

The encoder takes the input sequence, like a sentence, and processes it to create a condensed representation or embedding of the meaning.

This embedding captures the important information from the entire sequence.

The encoder typically consists of multiple layers, each containing a self-attention mechanism and a feed-forward network.

Self-attention allows the model to understand how different words in the sequence relate to each other, not just their individual meaning.

### **Input Embeddings:**

Input embeddings convert words or tokens (subwords) in a sentence into numerical vectors. These vectors represent the semantic meaning of the word. Similar words will have similar vector representations, capturing semantic relationships between words.

### **Positional Encoding:**

In languages, the order of the words and their position in a sentence really matters. The meaning of the entire sentence can change if the words are re-ordered. When implementing NLP solutions, recurrent neural networks have an inbuilt mechanism that deals with the order of sequences. The transformer model, however, does not use recurrence or convolution and treats each data point as independent of the other. Hence, positional information is added to the model explicitly to retain the information regarding the order of words in a sentence. Positional encoding is the scheme through which the knowledge of the order of objects in a sequence is maintained.

### 

### **Multi head attention:**

Multi-head attention is a core building block of the Transformer architecture, a powerful deep learning model widely used in natural language processing (NLP) tasks. It allows the model to attend to relevant parts of an input sequence, considering not just the meaning of individual words but also their relationships within the context.

**Benefits of Multi-Head Attention:**

**Captures Relationships**: Goes beyond the meaning of individual words and considers their relationships within the context.

**Learns Different Aspects:** Multiple heads allow the model to learn different aspects of the relationships between elements.

**Parallelization:** The calculations for different heads can be parallelized, making it efficient for large models.

### **Adding and Normalization:**

In transformers, adding and normalization, often referred to as "Add & Norm", is a combined technique used within each layer of the encoder and decoder. It consists of two key components:

Residual Connection: This is an optimization technique inspired by ResNets. It adds the input to the layer (let's call it x) directly to the output of the layer (sublayer(x)). This ensures that the information from the previous layer is preserved as the network goes deeper. It also helps address the vanishing gradient problem during training, allowing the network to learn complex relationships.

Layer Normalization: This is a normalization technique applied to the output of the residual connection (x + sublayer(x)). Unlike batch normalization, which normalizes across a mini-batch of data, layer normalization normalizes across the features (dimensions) for each data point in the mini-batch. This helps the network learn faster and become more stable during training by keeping the activations within a specific range.

# **Decoder:**

The decoder takes the embedding created by the encoder as input.

It then generates the output sequence, word by word, based on the information from the encoder.

The decoder also uses multiple layers, with self-attention and feed-forward networks similar to the encoder.

But unlike the encoder, the decoder's self-attention mechanism is modified to only attend to previously generated words in the output sequence. This prevents the decoder from peeking ahead at future words it hasn't generated yet.

In some cases, the decoder also uses an additional attention mechanism called multi head attention , which allows it to focus on specific parts of the encoder's output while generating the target sequence.

## **Attention:**

In a transformer model, attention is a critical mechanism that allows the model to focus on specific parts of the input sequence when processing information. It's like having a spotlight that illuminates the most relevant sections of the text, instead of reading everything sequentially like a traditional RNN.

Here's a breakdown of attention in transformers:

Understanding Context: Attention helps the model understand the relationships between different parts of the sequence. It doesn't just process information linearly, but considers how each element relates to others.

Query, Key, Value: The attention mechanism works by using three elements:

Query: This represents the current element being processed.

Key: This represents all the elements in the sequence.

Value: This also represents all the elements in the sequence, containing the actual information.

Compatibility Scores: The model calculates compatibility scores between the query and each key. These scores indicate how relevant each element (represented by the value) is to the current element (query).

Weighted Sum: Based on the compatibility scores, the model creates a weighted sum of the values. Elements with higher scores (more relevant) contribute more to the final output.

## **Self Attention:**

Imagine you're reading a sentence and trying to understand the meaning of a specific word. Self-attention allows the transformer to consider all the other words in the sentence and determine which ones are most relevant to understanding the current word. For instance, to understand the meaning of "walk" in the sentence "The quick brown fox jumps over the lazy dog," self-attention would focus on words like "fox" and "jumps" more than "the" or "lazy" because they provide more context for the action of walking.

By using self-attention multiple times throughout the transformer architecture, the model can learn complex relationships between words within a sequence, leading to a deeper understanding of the entire sentence. This is crucial for tasks like machine translation, sentiment analysis, and text summarization.

Here are some key takeaways about self-attention:

Focus within the Sequence: It allows the model to pinpoint the most relevant parts of the input sequence itself.

Building Context: By understanding relationships between elements, self-attention helps the model build a better context for each word or element in the sequence.

## **Multi head attention:**

Multi-head attention is a core building block of the Transformer architecture, a powerful deep learning model widely used in natural language processing (NLP) tasks. It allows the model to attend to relevant parts of an input sequence, considering not just the meaning of individual words but also their relationships within the context.

Here's a breakdown of how multi-head attention works:

**1. Linear Transformations:**

The input to the multi-head attention layer can be word embeddings (in the case of text) or other vector representations of the sequence elements.

These input vectors are first linearly transformed using three different weight matrices:

Query (Q) matrix: Transforms the input vectors to create "query" vectors. These represent what the model is looking for in the sequence.

Key (K) matrix: Transforms the input vectors to create "key" vectors. These represent the information available in the sequence.

Value (V) matrix: Transforms the input vectors to create "value" vectors. These contain the actual information from each element in the sequence.

**2. Scaled Dot-Product Attention:**

The core of multi-head attention is the scaled dot-product attention mechanism. It calculates a score for each element in the sequence, indicating how relevant it is to the current query position.

Here's how it works:

The query vector for a specific position is multiplied (dot product) with each key vector in the sequence.

These products are then scaled by a constant value (usually the square root of the key vector dimension) to prevent large values from dominating the attention scores.

The resulting scaled scores represent how well each element "matches" the current query.

**3. Applying Attention:**

The scaled scores are then used as weights to attend to the corresponding value vectors in the sequence.

Each value vector is multiplied by its corresponding score, and the weighted values are summed up. This creates a context vector that represents the most relevant information from the sequence based on the current query.

**4. Multi-Head :**

The power of multi-head attention comes from repeating the entire process described above multiple times (having multiple "heads"). Each head uses different randomly initialized weight matrices (Q, K, V) to learn different aspects of the relationships between elements in the sequence.

The outputs (context vectors) from all the heads are then concatenated and transformed by a final linear layer. This allows the model to capture a richer and more nuanced understanding of the context.

# 

# **Summary**

**Setting Up the Environment:**

Instructions are provided on how to set up a conda environment with the necessary libraries for NLP using Transformers, including installing PyTorch.

**Natural Language Processing (NLP):**

An overview of NLP is given, covering its goals and various tasks like text classification, named entity recognition, sentiment analysis, machine translation, and more.

It highlights the role of machine learning and deep learning, particularly transformers, in achieving these tasks.

Neural AI and Transformers:

Neural AI refers to deep learning models based on neural networks.

Transformer models are a type of neural network architecture that have revolutionized NLP.

**Word Embeddings:**

Word embeddings are numerical representations of words used in NLP tasks.

Word2vec is a technique for generating word embeddings based on the surrounding words.

**Recurrent Neural Networks (RNNs):**

RNNs are a type of neural network designed for sequential data like text.

They can struggle with long-term dependencies in sequences due to vanishing or exploding gradients.

**Long Short-Term Memory (LSTM):**

LSTMs are a specific RNN architecture that addresses the vanishing gradient problem and excels at capturing long-term dependencies.

Key features of LSTMs include memory cells and gating mechanisms to control information flow.

**Transformers:**

Transformers are a powerful architecture for various NLP tasks.

They consist of an encoder-decoder structure, where the encoder processes the input sequence and the decoder generates the output sequence.

Self-attention and multi-head attention mechanisms are core components that allow transformers to focus on relevant parts of the input sequence and understand relationships between elements.

Transformers offer advantages over LSTMs in terms of parallelization and handling long-range dependencies.