Experiment 5

1. Constructing and Plotting Dependency Graphs for semantic connections

The following points explain the work done in this experiment.

The code that implements the following is present in the "depend.ipynb" jupyter notebook.

1. Dependency Parsing and Visualization - The code uses the spaCy library to perform dependency parsing and visualize the results. We loaded a pre-trained English language model (en_core_web_lg) and processes the input sentence, "that would be very easy." We then print out a table displaying token information, including the token text, dependency relation, head token, and children tokens. Finally, we visualize the dependency parsing tree using displacy.render, which displays a graphical representation of the sentence's syntactic structure.

2. Dependency Tree Visualization using NetworkX - The code also utilizes the NetworkX library to create a directed graph representing the dependency relationships in the input sentence. We add nodes for each token in the sentence and edges to represent the dependency relationships between tokens. The graph is then visualized using matplotlib. This provides a visual representation of the sentence's dependency parsing, showing how tokens are connected through dependency relationships.

3. Co-Occurrence Analysis of Informal and Formal Phrases - We analyze co-occurrences of informal and formal phrases in a set of sample sentences. We also defined lists of informal and formal phrases. and created a graph using NetworkX with nodes labeled by phrase formality. We can process sample sentences, identify phrase co-occurrences, and add edges to the graph. We also visualize the graph with different colors for informal and formal phrases.

4. Graphical Visualization - We utilize graphical visualizations throughout the code to represent linguistic dependencies, co-occurrence patterns, and formality distinctions effectively.

Conlusion

We attempted to use dependency parsing to create a connection between different parts of speech typically found in idioms. We also tried to establish semantic connections between different parts of a sentence. However idioms and many colloquial expressions commonly found in informal expression of language do not follow a syntactic or semantic pattern. Therefore we cannot generalize this for a large set of idioms as they do not follow any such rules.

2. Prediction and Completion of idiomatic sentences using generative models

Experimented with generative GPT models to calculate the percentages associated with words when attempting to complete idiomatic sentences.

Associated code present in "gen.ipynb" file.

1. Experiment with GPT-2 for Language Generation - This code is designed to experiment with the GPT-2 language model. GPT-2 (Generative Pre-trained Transformer 2) is a state-of-the-art generative language model developed by OpenAI.

2. Library Imports and Model Loading - The code imports essential libraries: TensorFlow and the Transformers library. Transformers is a library developed by Hugging Face for working with various pre-trained models. It loads a pre-trained GPT-2 model and tokenizer. The model is a large neural network that has been trained on a massive amount of text data, enabling it to generate coherent and contextually relevant text.

3. Word Prediction in a Sentence - The code demonstrates how to use the GPT-2 model for word prediction within a sentence. It starts with an input sentence, "fine if you are so clever why don't you tell me what to." The sentence is tokenized using the GPT-2 tokenizer, and the tokens are converted into input IDs. The GPT-2 model predicts the next word in the sentence by calculating a probability distribution over all possible words and selecting the word with the highest probability. The predicted word is then decoded back into text and printed.

4. Word Probability Calculation - This part of the code calculates the probability of a specific word occurring next in a given context. It uses the sentence "its raining" as the context and aims to find the probability of the word "down" appearing next. The GPT-2 model generates probabilities for all possible tokens, and these probabilities are sorted in descending order. The code identifies and prints the probability associated with the target word "down."

5. Sentence Continuation with a Specific Word - This section illustrates how to generate a sentence continuation by specifying a particular word. It takes the input sentence "its raining cats and" and intends to complete it with the word "dogs" to form the common phrase "its raining cats and dogs." The code tokenizes both the input sentence and the target word "dogs." It concatenates the tokens for the input sentence and the target word and generates probabilities for the next token. The probability associated with the input word "dogs" in the context of the sentence is calculated and printed.

6. Language Modeling Experimentation - Overall, this code serves as an introductory exploration of how to interact with a pre-trained GPT-2 language model. It showcases the model's capabilities, such as word prediction, probability calculation, and sentence generation with specific words. This experimentation demonstrates the power of large pre-trained language models in understanding and generating human-like text based on context..

Conlusion

Our hypothesis was that extremely advanced large language models like GPT 2 and GPT 3 would be able to predict the next word in a sentence very accurately.
1. When passed the starting parts of common idiomatic phrases, the model is able to successfully determine the next word that would correctly complete the idiom when given the correct context.
2. This proves that the model is able to understand context of certain common phrases and recognise the fact that they are idioms.