Ben Glaser

"LLMs with Python: A Word Embeddings and Transformer Sandbox"

Learning goals:

- Learn about word embeddings, a fundamental part of LLMs
- Assess word embeddings as semantic fields and contrast with other ways of determining the meaning of words in literary texts
- Interact with and familiarize ourselves with the python ("Jupyter") notebook coding environments by using Google's "colab" (more advanced courses could include active coding with AI support)
- Access open-source language models (BERT, via huggingface)

Three Classroom Workshops:

Class Session #1: work through a pre-built colab notebook loaded with word2vec embeddings of a google news dataset:

1) As a class, try out some vector math: similar words and analogies within these embeddings (search for "cosine similarity" in the notebook)
2) Students breakout into pairs and run the similarity functions on their own
3) Interpret results and brainstorm other language processing tasks using this tool

Class Session #2: create our own word embeddings the word embeddings model "word2vec." The notebook uses a .txt file of Shakespeare's works, but you can choose similarly sized data sets.

1) Optional: Outside of class, students run the notebook with their own data.
2) With an AI coding agent, re-use the cosine similarity functions from our original pre-built notebook on this new corpus
3) Bring observations of word embeddings back to the class. Compare embeddings. Possible writing assignment.

Class Session #3: move on to Transformers, and specifically the bidirectional transformer model BERT.

1) Using a pre-built colab notebook, Load and run the “fill mask” function using a transformer model (BERT):
2) Students break out again to input different lines of poetry or prose.
3) Qualitative discussion of the results; interpret the original poem using counterfactuals produced by BERT

Calibrating the Coding Level:

Depending on the level of coding literacy for the course, this sequence could have an increasing level of complexity:

1) Students run a completely preconfigured notebook with a set corpus (Shakespeare, for example)
2) Students upload their own corpus into an otherwise prebuilt notebook
3) Students merge the two notebooks above—the tensorflow word2vec model and the embeddings analysis notebook—into a new colab notebook to do vector math.
4) Students compare vector results across two corpora (i.e. google news and Shakespeare)
5) Students rebuild a word embedding project from scratch using gemini, claude code (if using vscode / CLI), or another AI coding assistant
6) Students fine-tune a transformer model using their dataset so that they can generate text as well as studying embeddings.

Glossary:

- **Word Embeddings**: A type of word representation that allows words to be represented as vectors in a continuous vector space. This representation captures semantic relationships between words.
- **word2vec**: A popular word embedding model that uses neural networks to learn word associations from a large corpus of text. It produces word vectors that can be used for various natural language processing tasks.
- **Cosine Similarity**: A measure of similarity between two non-zero vectors. It calculates the cosine of the angle between the vectors, which indicates how similar they are.
- **BERT (Bidirectional Encoder Representations from Transformers)**: A transformer-based model designed to understand the context of a word in search queries. It uses a bidirectional approach to consider the context from both directions.
- **Fill Mask Function**: A function used in transformer models like BERT to predict the missing word in a sentence. It helps in understanding the context and generating meaningful text.

- **Vector Math**: Mathematical operations performed on vectors, such as addition, subtraction, and finding similarities. In the context of word embeddings, it is used to find relationships between words.
- **Colab Notebook**: An interactive Python notebook provided by Google Colab that allows users to write and execute code in a web-based environment. It is commonly used for machine learning and data analysis tasks.
- **Corpus**: A large collection of texts used for training language models. Examples include datasets like Shakespeare's works or Google News articles.
- **Transformer Model**: A type of deep learning model that uses self-attention mechanisms to process and generate sequences of text. It is the foundation for models like BERT and GPT.
- **Fine-Tuning**: The process of taking a pre-trained model and further training it on a specific dataset to improve its performance on a particular task.

Instructor Reflection:

This is a challenging set of activities to run without moderate coding literacy. AI-coding agents help, but I recommend asking for support from a CS student, Student AI Liaison, or DH Lab staff member.

The first and third sessions are very adaptable and it can be really fun to have students drop different word pairs or analogies and see how language models understand them. One senses a lot of bias, especially around gender and raced language! The fill-mask activity is wonderful for getting students to think about *counterfactuals*, i.e. what poets didn't write, and then why they wrote what they did. Students then start to see just how strange a "model" of language poets use. And also how strange LLMs can be. The final learning goal here is both estrangement within language and familiarization with tools for its analysis.

For the second class session, I had to run the training in advance of discussion; in the future, I would work with YCRC to create a Jupyter notebook environment in their computing infrastructure so that students could log in in real time and drop datasets into the notebook, then run in real time. Training sometimes fails, or takes a lot of time, so this is better as an asynchronous task.