**Building LLM Apps**

**Data Preparation**

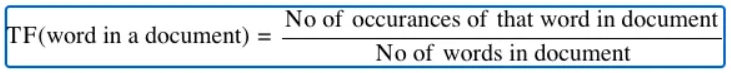
**Embedding Models**

* Embeddings are a type of word representation (with numerical vectors) that allows words with similar meanings to have a similar representation.
* These vectors can be learned by a variety of machine-learning algorithms and large datasets of texts. One of the main roles of word embeddings is to provide input features for downstream task like text classification and information retrieval
* **A diagram of audio embeddings

  Description automatically generated**
* Several word embedding methods have been proposed in the past decade, here are some of them.
* **A diagram of a computer

  Description automatically generated**
* **Context-independent Embeddings:**
  + Context-independent embeddings redefine word representations, assigning unique vectors regardless of contextual variations. This concise exploration focusses on the implication for homonym disambiguation.
    - Context-independent models allocate distinct vectors to each word, irrespective of context.
    - Homonyms like “duck” receive a single vector, blending diverse meanings without contextual cues.
    - The approach yields a comprehensive map of word vectors, capturing multiple meanings in a fixed representation.
    - **A diagram of a word embedding

      Description automatically generated**
  + Context-independent embeddings offer efficiency but pose challenges in nuanced language understanding, especially with homonyms. This paradigm shift prompts a closer look at the trade-offs in natural language processing.
  + Some common **frequency-based** context-independent embeddings:
    - **Bag of Words**
      * Bag of words will create a dictionary of the most common words in all the sentences and then encode the sentences as shown below
  + **A diagram of a number of words

    Description automatically generated**
    - **TF-IDF**
      * TF-IDF is a simple technique to find features from sentences. While in Count features we take count of all the words/ngrams present in a document, with TFIDF we take features only for the significant words. How do we do that? If you think of a document in a corpus, we will consider two things about any words in that document:
      * **Term Frequency:** How important is the word in the document
    - ****
      * **Inverse Document Frequency:** How important the term is in the whole corpus?
    - ****
  + **A diagram of a diagram

    Description automatically generated**
  + Some common **prediction-based** context-independent embeddings:
  + **Word2Vec:**
    - Word embeddings in Word2Vec are learned through a two-layer neural network, which inadvertently captures linguistic contexts during the training process. The embeddings serve as a byproduct of the algorithm’s primary object, showcasing the efficiency of this approach. Word2Vec provides flexibility through two distinct model architectures: CBOW and continuous skip-gram.
  + **Continuous Skip-Gram:**
    - Uses the current word to predict the surrounding window of context words.
    - Focuses on the predictive power of the target word in generating the context words.
  + **A diagram of a diagram

    Description automatically generated**
  + Word2Vec’s dual model architectures offer versatility in capturing linguistic nuances, allowing practitioners to choose between CBOW and continuous skip-gram based on the specific requirements of their natural language processing tasks. Understanding the interplay between these architectures enhances the application of Word2Vec in diverse contexts.
  + **GloVe (Global Vectors for Word Representation):** GloVe’s strength lies in its utilization of aggregated global word-word co-occurrence statistics from a corpus during training. The resulting representation not only encapsulates semantic relationships but also unveil intriguing linear substructures within the word vector space, adding depth to the understanding of word embeddings.
  + **FastText:** In contrast to GloVe, FastText takes a novel approach by treating each word as composed of character n-grams. This distinctive features allows FastText to not only learn rare words but also handle out-of-vocabulary words with finesse. The emphasis on character-level embeddings empowers FastText to capture morphological nuances, offering a more comprehensive representation of the vocabulary.
* **Context-Dependent Embeddings:**
  + Context-Dependent methods learn different embeddings for the same word based on its context.
  + **A diagram of a word

    Description automatically generated**
  + **RNN based:**
    - **ELMO (Embeddings from Language Model):** Learns contextualized word representations based on a neural language model with a character-based encoding layer and two BiLSTM layers
    - **CoVe (Contextualized Word Vectors):** Uses a deep LSTM encoder from an attentional sequence-to-sequence model trained for machine translation to contextualized word vectors.
  + **Transformer-based:**
    - **BERT (Bidirectional Encoder Representations from Transformers):** Transformer-based language representation model trained on a large cross-domain corpus. Applies a masked language model to predict words that are randomly masked in a sequence, and this is followed by a next-sentence-prediction task for learning the associations between sentences.
    - **XLM (Cross-lingual Language Model):** It’s a transformer pre-trained using next token prediction, a BERT-like masked language modeling objective, and a translation objective.
    - **RoBERTa (Robustly Optimized BERT Pretraining Approach):** It builds on BERT and modifies key hyperparameters, removing the next-sentence pretraining objective and training with much larger mini-batches and learning rates.
    - **ALBERT (A Lite BERT for Self-Supervised Learning of Language Representations):** It presents parameter-reduction techniques to lower memory consumption and increase the training speed of BERT.

**BERT**

* BERT (Bidirectional Encoder Representations from Transformers), a powerhouse in natural language processing developed by Google AI, has reshaped the landscape of language models. This exploration delves into the pre-training methodology and the intricacies of its bi-directional architecture.
* **Pre-training:** Bert is pre-trained for two unsupervised tasks – Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). The MLM randomly masks some (15%) of the tokens from the input at random, and the objective is to predict the original vocabulary ID of the masked word based only on its context.
* In addition to the masked language model, BERT uses a NSP task that jointly pre-trains text-pair representations. Many important downstream tasks such as Question Answering (QA) and Natural Language Inference (NLI) are based on understanding the relationship between two sentences, which is not directly captured by language modeling.
* **Pre-training data:**  The pre-training procedure largely follows the existing literature on language model pre-training. For the pre-training corpus we use the Books Corpus (800M words) and English Wikipedia (2,500M words).
* **Bi-directional**: Unlike left-right language model pre-training, the MLM objective enables the representation to fuse the left and the right context, which allows us to pre-train a deep bidirectional Transformer.
* The architecture of BERT is structured with multiple encoder layers, each applying self-attention to the input and passing it to the subsequent layer. Even the smallest variant, BERT BASE, boasts 12 encoder layers, a feed-forward neural network block with 68 hidden units, and 12 attention heads.
* A diagram of a basic structure

  Description automatically generated with medium confidence

**Input Representations**

* BERT takes as input sequences that are composed of sentences or pairs of sentences (e.g., <Question, Answer>) in one token sequence for question-answering tasks.
* Input sequences are prepared before being fed to the model using WordPiece tokenizer with a 30k vocabulary size token. It works by splitting a word into several subwords (Tokens).
* Special tokens are:
  + **[CLS]** used as the first token of each sequence. The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks.
  + **[SEP]** Sentence pairs are packed together into a single sequence. We differentiate the sentences in two ways. First, we separate them with a special token([SEP]). Second, we add a learned embedding to every token indicating whether it belongs to sentence A or sentence B.
  + **[PAD]** used to represent paddings in the input sentences (empty tokens). The model expects fixed-length sentences as input. A maximum length is thus fixed depending on the dataset. Shorter sentences are padded, whereas longer sentences are truncated. To explicitly differentiate between real tokens and [PAD] tokens, we use an attention mask.
* Segmentation embeddings is introduced to indicate if a given token belongs to the first or second sentence. Positional embedding indicates the position of tokens in a sentence. By contrast to the original Transformer, BERT learns positional embeddings from absolute ordinal position, instead of using trigonometric functions.
* For a given token, its input representation is constructed by summing the corresponding token, segment, and position embeddings.
* **A screenshot of a computer

  Description automatically generated**
* To get the token embeddings, an embedding lookup table is used at the embedding layer (seen above), where rows represent all possible token IDs in the vocabulary (30k rows for instance) and columns represent the token embedding size.

**Why Sentence BERT (S-BERT) Over BERT?**

* So far, so good, but these transformer models had one issue when building sentence vectors: Transformers work using word or token-level embeddings, not sentence-level embeddings.
* Before sentence transformers, the approach to calculating accurate sentence similarity with BERT was to use a **cross-encoder structure.** This meant that we would pass two sentences to BERT, add a classification head to the top of BERT – and use this to output a similarity score.
* The BERT cross-encoder architecture consists of a BERT model which consume sentences A and B. Both are processed in the same sequence, separated by a **[SEP]** token. All of this is followed by a feedforward NN classifier that outputs a similarity score.
* A screenshot of a computer

  Description automatically generated
* The cross-encoder network does produce very accurate similarity scores (better than SBERT), but it’s not scalable. If we wanted to perform a similarity search through a small 100K sentence dataset, we would need to complete the cross-encoder inference computation 100K times.
* To cluster sentences, we would need to compare all sentences in our 100K dataset, resulting in just under 500M comparisons – this is simply not realistic.
* **Ideally, we need to pre-compute sentence vectors that can be stored and then used whenever required.** If these vector representations are good, all we need to do is calculate the cosine similarity between each. With the original BERT (and other transformers), we can build a sentence embedding by averaging the values across all token embeddings output by BERT (if we input 512 tokens, we output 512 embeddings)
* Alternatively, we can use the output of the first [CLS] token (a BERT-specific token whose output embedding is used in classification tasks)
* Using one of these two approaches gives us our sentence embeddings that can be stored and compared much faster, shifting search times from 65 hours to around 5 seconds. However, the accuracy is not good, and is worse than using average GloVe embeddings
* A diagram of a computer program

  Description automatically generated
* Thus, finding the most similar sentence pair from 10K sentences took 65hrs with BERT. With SBERT, embeddings are created in ~5 seconds and compared with cosine similarity in ~0.01 seconds.
* Since the SBERT paper, many more sentence transformer models have been built using similar concepts that went into training the original SBERT. They’re all trained on many similar and dissimilar sentence pairs.
* Using a loss function such as **SoftMax loss, multiple negatives ranking loss, or MSE margin loss** these models are optimized to produce similar embeddings for similar sentences, and dissimilar embeddings otherwise.
* Deriving independent sentence embeddings is one of the main problems of BERT. To alleviate this aspect SBERT was developed.

**Sentence Transformers**

* We explained the cross-encoder architecture for sentence similarity with BERT. SBERT is similar but drops the final classification head, and processes one sentence at a time. SBERT then uses mean pooling on the final output layer to produce a sentence embedding.
* Unlike BERT, SBERT is fine-tuned on sentence pairs using a Siamese architecture. We can think of this as having two identical BERTs in parallel that share the exact same network weights.
* In reality, we are using a single BERT model. However, because we process sentence A followed by sentence B as pairs during training, it is easier to think of this as two models with tied weights.

**Siamese BERT Pre-Training**

**A diagram of a sentence

Description automatically generated**

* There are different approaches to training sentence transformers. We will describe the original process featured most prominently in the original SBERT paper that optimizes on SoftMax-loss.
* The SoftMax-loss approach used the ‘Siamese’ architecture fine-tuned on the Stanford Natural Language Inference (SNLI) and Multi-Genre NLI (MNLI) corpora.
* SNLI contains 570K sentence pairs, and MNLI contains 430K. The pairs in both corpora include a **premise** and a **hypothesis.** Each pair is assigned one of three labels:
  + **0** – entailment, e.g. the **premise** suggests the **hypothesis.**
  + **1** – neutral, the **premise** and **hypothesis** could both be true, but they are not necessarily related.
  + **2** – contradiction, the **premise** and **hypothesis** contradict each other.
* Given this data, we feed sentence A (let’s say the **premise**) into siamese BERT A and sentence V (**hypothesis**) into siamese BERT B.
* The siamese BERT outputs our pooled sentence embeddings. These were the results of three different pooling methods in the SBERT paper. Those **mean, max,** and **[CLS]-pooling**. The mean-pooling approach was best performing for both NLI and STSb datasets.
* There are now two sentence embeddings. We will call embeddings A as u and embeddings B as v. The next step is to concatenate u and v. Again, several concatentation approaches were tested, but the highest performing was a (u, v, |u-v|) operation:
* A diagram of a diagram

  Description automatically generated
* |u-v| is calculated to give us the element-wise difference between the two vectors. Alongside the original two embeddings (u and v), these are all fed into a feedforward neural net (FFNN) that has three outputs.
* These three outputs align to our NLI similarity labesl **0, 1,** and **2**. We need to calculate the SoftMax from our FFNN, which is done within the cross-entropy loss function. The SoftMax and labels are used to optimize on this ‘SoftMax-loss’
* A diagram of a function

  Description automatically generated

**SBERT Objective Functions**

* By using these two vectors **u** and **v**, three approaches for optimizing different objectives are discussed below.
* **Classification:**
  + The three vectors **u, v** and **|u-v|** are concatenated, multiplied by a trainable weight matrix W and the multiplication results is feed into the SoftMax classifier which outputs normalized probabilities of sentences corresponding to different classes. The cross-entropy loss function is used to update the weights of the model.
* **A diagram of a computer program

  Description automatically generated with medium confidence**
* **Regression:**
  + In this formulation, after getting vectors **u** and **v**, the similarity score between them is directly computed by a chosen similarity metric. The predicted similarity score is compared with the true value and the model is updated by using the MSE loss function.
* **A diagram of a diagram

  Description automatically generated with medium confidence**
* **Triplet Loss:**
  + The triplet objective introduces a triplet loss which is calculated on three sentences usually named *anchor, positive and negative*. It is assumed that *anchor and positive* sentences are very close to each other while *anchor and negative* are very different. During the training process, the model evaluates how closer the pair (*anchor, positive*) is, compared to the pair (*anchor, negative*)
* **A diagram of a structure

  Description automatically generated**
  + For now, let’s look at how we can initialize and use these sentence transformer models.

**Hands-On with Sentence Transformers**

* The fastest and easiest way to begin working with sentence transformers is through the **sentence-transformer** library created by the creators of SBERT.
* !pip install sentence-transformers
* from sentence\_transformers import SentenceTransformer  
    
  model = SentenceTransformer('bert-base-nli-mean-tokens')  
    
  model
  + OUTPUT:
* SentenceTransformer(  
   (0): Transformer({'max\_seq\_length': 128, 'do\_lower\_case': False}) with Transformer model: BertModel   
   (1): Pooling({'word\_embedding\_dimension': 768, 'pooling\_mode\_cls\_token': False, 'pooling\_mode\_mean\_tokens': True, 'pooling\_mode\_max\_tokens': False, 'pooling\_mode\_mean\_sqrt\_len\_tokens': False})  
  )
* The **output** we can see here is the **SentenceTransformer** object which contains three components:
  + The **transformer** itself, here we can see the max sequence length of **128** tokens and whether to lowercase any input (in this case, the model does not). We can also see the model class, **BertModel.**
  + The **pooling** operation, here we can see that we are producing a **768-dimensional** sentence embedding. We are doing this using the mean pooling method.
* Once we have the model, building sentence embeddings is quickly done using the **encode** method.
* sentences = [  
   "the fifty mannequin heads floating in the pool kind of freaked them out",  
   "she swore she just saw her sushi move",  
   "he embraced his new life as an eggplant",  
   "my dentist tells me that chewing bricks is very bad for your teeth",  
   "the dental specialist recommended an immediate stop to flossing with construction materials"  
  ]  
    
  embeddings = model.encode(sentences)  
    
  embeddings.shape
  + Output:
* (5, 768)

**Which Embedding Model to Choose?**

* However, as we will soon discover, the majority of the currently employed embedding models belong to the transformer category. These models are offered by various providers, with some being open source and others proprietary, each tailored to specific objectives:
  + Some are best suited for coding tasks.
  + Others are designed specifically for the English language
  + And there are also embedding models that excel in handling multilingual datasets
* The simplest approach is to leverage existing academic benchmarks. Nevertheless, it’s crucial to bear in mind that these benchmarks may not comprehensively mirror real-world usage of retrieval systems in AI applications.
* Alternatively, you can conduct existing with various embedding models and compile a final evaluation table to pinpoint the most suitable one for your specific use case. I highly recommend incorporating a re-ranker into this process as it can significantly enhance retriever performance, ultimately yielding the optimal results.
* To simplify your decision-making process, Hugging Face offers the remarkable “Massive Text Embedding Benchmark (MTEB) Leaderboard”. This resource provides comprehensive information about all available embedding models and their respective scores on various metrics:

**Wrap Up**

* The blog explores various embedding models for generating vector representations of text, including Bag of Words, TF-IDF, Word2Vec, GloVe, FastText, ELMO, BERT, and more. It delves into the architecture and pre-training of BERT, intorduces Sentence Bert (SBERT) for efficient sentence embeddings, and provides a hands-on example using the sentence transformers library. The conclusion emphasizes leveraging resources like the Hugging Face Massive Text Embedding Benchmark (MTEB) Leaderboard for evaluation.