Embedding Skills

Many industries hope to have the capabilities of LLMs and hope that LLMs can solve their own internal problems. This includes employee-related content such as onboarding instructions, leave and reimbursement processes, and benefit inquiries. Enterprise business flow-related content includes relevant documents, regulations, execution processes, etc., as well as some customer-oriented inquiries. Although LLMs have strong knowledge capabilities, industry-based data and knowledge cannot be obtained. So how to inject this industry-based knowledge content? This is also an important step for LLMs to enter the corporate world. In this chapter, we will talk to you about how to inject industry data and knowledge to make LLMs more professional. This is the basis on which we create RAG applications.

Let’s start with vectors in natural language

In the field of natural language, we know that the finest granularity is words, words constitute sentences, and sentences constitute paragraphs, chapters and final documents. Computers don't know words, so we need to convert words into mathematical representations. This representation is a vector, that is, Vector. Vector is a mathematical concept. It is a directional quantity with magnitude and direction. With vectors, we can effectively vectorize text, which is also the basis of the field of computer natural language. In the field of natural language processing, we have many vectorization methods, such as One-hot, TF-IDF, Word2Vec, Glove, ELMO, GPT, BERT, etc. These vectorization methods have their own advantages and disadvantages, but they are all based on word vectorization, that is, word vectors. Word vectors are the basis of natural language processing and the basis of all OpenAI models. Let’s look at several common methods in word vectors respectively.

One-hot

One-hot encoding uses 0 and 1 encoding to represent words. For example, we have 4 words, namely: I, love, Beijing, and Tiananmen. Then we can use 4 vectors to represent these 4 words, which are:

```
I = [1, 0, 0, 0]
love = [0, 1, 0, 0]
Beijing = [0, 0, 1, 0]
Tiananmen = [0, 0, 0, 1]
```

In traditional natural language application scenarios, we regard each word as represented by a One-Hot vector as a unique discrete symbol. The number of words in our vocabulary is the dimension of the vector. For example, the above example contains a total of four words, so we can use a four-dimensional vector to represent it. In this vector, each word is unique, which means that each word is independent and has no relationship. Such vectors are called One-Hot vectors. The advantage of One-Hot vector is that it is simple, easy to understand, and each word is unique without any relationship. However, the disadvantage of One-Hot vector is also obvious, that is, the dimension of the vector will increase as the number of words increases. For example, if we have 1000 words, then our vector will be 1000 dimensions. Such a vector is very sparse, that is, most of its values are 0. Such vectors will cause a very large amount of computer

calculations and are not conducive to computer calculations. Therefore, the disadvantage of One-Hot encoding is that the vector dimension is large, the calculation amount is large, and the calculation efficiency is low.

TF-IDF

TF-IDF is a statistic that evaluates the importance of a word to a corpus. TF-IDF is the abbreviation of Term Frequency - Inverse Document Frequency, which is called Term Frequency-Inverse Document Frequency in Chinese. The main idea of TF-IDF is: if a word appears frequently in an article and rarely appears in other articles, then this word is the keyword of this article. Generally, we are used to splitting this concept into two parts: TF and IDF.

TF - Term Frequency

Term Frequency refers to the frequency with which a word appears in an article. The formula for calculating word frequency is as follows:

```
TF = The number of times a word appears in the article / the total number
of words in the article
```

One problem with TF is that if a word appears many times in an article, the TF value of the word will be very large. In this case, we will consider this word to be the keyword of this article. But in this case, we will find that many words are keywords in this article. In this case, we will not be able to distinguish which words are keywords in this article. So we need to make some adjustments to TF, and this adjustment is IDF.

IDF - Inverse document frequency

Inverse Document Frequency refers to the frequency of a certain word appearing in all articles. The formula for calculating inverse document frequency is as follows:

```
IDF = log(Total number of documents in the corpus / (number of documents
containing the word + 1))
```

IDF 的计算公式中，分母加 1 是为了避免分母为 0 的情况。IDF 的计算公式中，语料库的文档总数是固定的，所以我们只需要计算包含该词的文档数就可以了。如果一个词在很多文章中都出现，那么这个词的 IDF 值就会很小。如果一个词在很少的文章中出现，那么这个词的 IDF 值就会很大。这样的话，我们就可以通过 TF 和 IDF 的乘积来计算一个词的 TF-IDF 值。TF-IDF 的计算公式如下：

```
TF-IDF = TF * IDF
```

In the calculation formula of IDF, 1 is added to the denominator to avoid the situation where the denominator is 0. In the calculation formula of IDF, the total number of documents in the corpus is fixed, so we only need to calculate the number of documents containing the word. If a word appears in many articles, the IDF value of this word will be small. If a word appears in few articles, the IDF value of this word will be large. In this case, we can calculate the TF-IDF value of a word by multiplying TF and IDF. The calculation formula of TF-IDF is as follows:

Word2Vec

We also call Word2Vec Word Embeddings, which is called word embedding in Chinese. The main idea of Word2Vec is that the semantics of a word can be determined by its context. Word2Vec has two models, CBOW and Skip-Gram. CBOW is the abbreviation of Continuous Bag-of-Words, which is called the continuous bag-of-words model in Chinese. Skip-Gram is the abbreviation of Skip-Gram Model, which is called skip model in Chinese. The idea of the CBOW model is to predict a word through its context. The idea of the Skip-Gram model is to predict the context of a word from a word. The advantage of Word2Vec is that it can get the semantics of words and the relationship between words. Compared with One-Hot encoding and TF-IDF encoding, the advantage of Word2Vec encoding is that it can obtain the semantics of words and the relationship between words. The disadvantage of Word2Vec encoding is that it is computationally intensive and requires a large corpus.

We mentioned before that the dimension of One-Hot encoding is the number of words, while the dimension of Word2Vec encoding can be specified. Generally we will specify 100 dimensions or 300 dimensions. The higher the dimensionality of Word2Vec encoding, the richer the relationships between words, but the greater the computational complexity. The lower the dimensionality of Word2Vec encoding, the simpler the relationship between words, but the smaller the calculation amount. The dimensionality of Word2Vec encoding is generally 100 or 300 dimensions, which can meet most application scenarios.

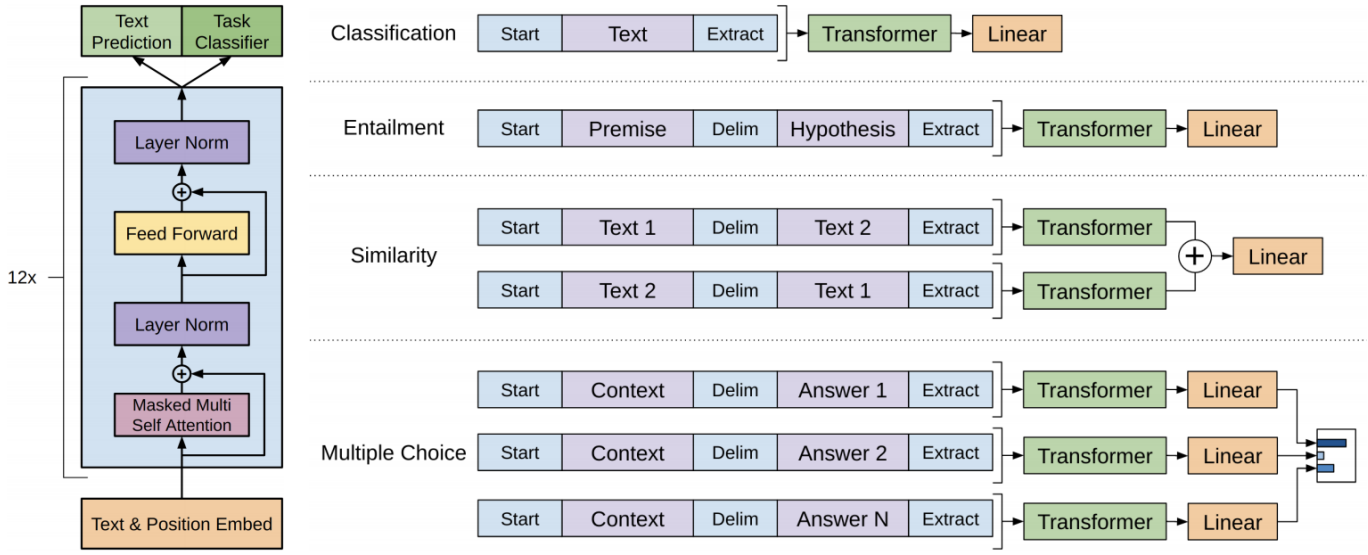
The calculation formula of Word2Vec encoding is very simple, which is Word Embeddings. Word Embeddings is a word vector whose dimensions can be specified. The dimensions of Word Embeddings are generally 100 or 300 dimensions, which can meet most application scenarios. The calculation formula for Word Embeddings is as follows:

```
Word Embeddings = Semantics of words + relationships between words
```

Think of Word2Vec as a simplified neural network.

GPT

The full name of the GPT model is Generative Pre-Training, which is called pre-training generative model in Chinese. The GPT model was proposed by OpenAI in 2018. Its main idea is that the semantics of a word can be determined through its context. The advantage of the GPT model is that it can obtain the semantics of words and the relationship between words. The disadvantage of the GPT model is that it is computationally intensive and requires a large corpus. The structure of the GPT model is a multi-layered unidirectional Transformer structure. Its structure is shown in the figure below:



The training process is divided into two stages, the first stage is pre-training and the second stage is fine-tuning. The pre-training corpora are Wikipedia and BookCorpus, and the fine-tuning corpora are different natural language tasks. The goal of pre-training is to predict a word through its context, and the goal of fine-tuning is to fine-tune the semantic model to obtain different models based on different natural language tasks, such as text classification, text generation, question and answer systems, etc.

The GPT model has gone through four stages, the most famous of which are GPT-3.5 and GPT 4 used by ChatGPT. GPT opens a new era, and its emergence allows us to see the infinite possibilities of natural language processing. The advantage of the GPT model is that it can obtain the semantics of words and the relationship between words. The disadvantage of the GPT model is that it is computationally intensive and requires a large corpus. Many people hope to have a GPT that benchmarks their own industry. This is also a problem we need to solve in this chapter.

BERT

BERT is the abbreviation of Bidirectional Encoder Representations from Transformers, which is called Transformer of Bidirectional Encoder in Chinese. BERT is a pre-trained model, and its training corpus is Wikipedia and BookCorpus. The main idea of BERT is that the semantics of a word can be determined by its context. The advantage of BERT is that it can obtain the semantics of words and the relationship between words. Compared with One-Hot encoding, TF-IDF encoding and Word2Vec encoding, the advantage of BERT encoding is that it can obtain the semantics of words and the relationship between words. The disadvantage of BERT encoding is that it is computationally intensive and requires a large corpus.

Vector embedding

We mentioned One-Hot encoding, TF-IDF encoding, Word2Vec encoding, BERT encoding, and GPT models. These codes and models are all a type of Embeddings embedding technology. Embeddings The main idea of embedding technology is that the semantics of a word can be determined by its context. The advantage of Embeddings technology is that it can obtain the semantics of words and the relationship between words. Embeddings are the basis of natural language deep learning. Its emergence allows us to see the infinite possibilities of natural language processing.

For the Embeddings method of text content, let's combine it with the previous section. You will find that since the birth of word2vec technology, the Embeddings of text content have been continuously

strengthened. From word2vec to GPT to BERT, the effect of Embeddings technology is getting better and better. The essence of Embeddings technology is "compression", using fewer dimensions to represent more information. The advantage of this is that it can save storage space and improve computing efficiency.

In Azure OpenAI Service, Embeddings technology is widely used to convert text strings into floating-point vectors and measure the similarity between texts by the distance between vectors. If different industries want to add their own data, we can use these enterprise-level data to query the vectors through the OpenAI Embeddings - text-embedding-ada-002 model, and save them through mapping. When using them, we can also convert the questions into vectors, and use similar Compare the algorithms to find the closest TopN results, so as to find the enterprise content related to the problem.

We can vectorize the enterprise data through the vector database and save it, and then use the text-embedding-ada-002 model to query through the similarity of the vectors to find the enterprise content associated with the problem. Commonly used vector databases include Qdrant, Milvus, Faiss, Annoy, NMSLIB, etc.

Open AI 的 Embeddings Model

Correlation of text strings with text embedding vectors from OpenAI. Embedding is usually used in the following scenarios

- Search (results sorted by relevance to query string)
- Clustering (where text strings are grouped by similarity)
- Recommend (recommend items with relevant text strings)
- Anomaly detection (identify outliers with little correlation)
- Diversity measurement (analyzing similarity distribution)
- Classification (where text strings are classified by their most similar tags)

Embeddings are vectors (lists) of floating point numbers. The distance between two vectors measures their correlation. Small distances indicate high correlation, and large distances indicate low correlation. For example, if you have a string "dog" with an embedding of [0.1,0.2,0.3], that string is more similar to a string "cat" with an embedding of [0.2,0.3,0.4] than to a string "cat" with an embedding of [0.9,0.8 ,0.7] the string "car" is more relevant.

Semantic Kernel's Embeddings

The support for Embeddings in Semantic Kernel is very good. In addition to supporting text-embedding-ada-002, it also supports vector databases. Semantic Kernel abstracts the vector database, and developers can use a consistent API to call the vector database. ***This case uses Qdrant as an example***. In order for you to run the example smoothly, please install Docker first, install the Qdrant container and run it. The running script is as follows:

```
docker pull qdrant/qdrant

docker run -p 6333:6333 qdrant/qdrant
```

.NET

Add Nuget library

```
#r "nuget: Microsoft.SemanticKernel.Connectors.Qdrant, *-*"
```

Reference library

```
using Microsoft.SemanticKernel.Memory;
using Microsoft.SemanticKernel.Connectors.Qdrant;
```

Create instances and Memory bindings

```
var textEmbedding = new AzureOpenAITextEmbeddingGenerationService("Your
Azure OpenAI Service Embedding Models Deployment Name" , "Your Azure
OpenAI Service Endpoint", "Your Azure OpenAI Service API Key");


var qdrantMemoryBuilder = new MemoryBuilder();
qdrantMemoryBuilder.WithTextEmbeddingGeneration(textEmbedding);
qdrantMemoryBuilder.WithQdrantMemoryStore("http://localhost:6333", 1536);

var qdrantBuilder = qdrantMemoryBuilder.Build();
```

Note: Semantic Kernel Memory component is still in the adjustment stage, so you need to pay attention to the risk of interface changes, and you also need to ignore the following information

```
#pragma warning disable SKEXP0003
#pragma warning disable SKEXP0011
#pragma warning disable SKEXP0026
```

Python

Reference library

```
from semantic_kernel.connectors.ai.open_ai import AzureChatCompletion,
AzureTextEmbedding
from semantic_kernel.connectors.memory.qdrant import QdrantMemoryStore
```

Add module support

```
kernel.add_text_embedding_generation_service(
        "embeddings_services", AzureTextEmbedding("EmbeddingModel",
endpoint,api_key=api_key,api_version = "2023-07-01-preview")
)
```

Add Memory

```
qdrant_store = QdrantMemoryStore(vector_size=1536,
url="http://localhost",port=6333)
await qdrant_store.create_collection_async('aboutMe')
kernel.register_memory_store(memory_store=qdrant_store)
```

Note: The Memory component is still in the adjustment stage, so you need to pay attention to the risk of interface changes.

Save and search your vectors in the Semantic Kernel

In Semantic Kernel, the methods of different vector data are unified through abstraction. You can easily save and search your vectors.

.NET

Save vector data

```
await qdrantBuilder.SaveInformationAsync(conceptCollectionName, id:
"info1", text: "Kinfey is Microsoft Cloud Advocate");
await qdrantBuilder.SaveInformationAsync(conceptCollectionName, id:
"info2", text: "Kinfey is ex-Microsoft MVP");
await qdrantBuilder.SaveInformationAsync(conceptCollectionName, id:
"info3", text: "Kinfey is AI Expert");
await qdrantBuilder.SaveInformationAsync(conceptCollectionName, id:
"info4", text: "OpenAI is a company that is developing artificial general
intelligence (AGI) with widely distributed economic benefits.");
```

Search vector data

```
string questionText = "Do you know kinfey ?";
var searchResults =  qdrantBuilder.SearchAsync(conceptCollectionName,
questionText, limit: 3, minRelevanceScore: 0.7);

await foreach (var item in searchResults)
{
    Console.WriteLine(item.Metadata.Text + " : " + item.Relevance);
}
```

Python

Save vector data

```
await kernel.memory.save_information_async(base_vectordb, id="info1",
text="Kinfey is Microsoft Cloud Advocate")
await kernel.memory.save_information_async(base_vectordb, id="info2",
text="Kinfey is ex-Microsoft MVP")
await kernel.memory.save_information_async(base_vectordb, id="info3",
text="Kinfei is AI Expert")
await kernel.memory.save_information_async(base_vectordb, id="info4",
text="OpenAI is a company that is developing artificial general
intelligence (AGI) with widely distributed economic benefits.")
```

Search vector data

```
ask = "who is kinfey?"

memories = await kernel.memory.search_async(
    base_vectordb, ask, limit=3, min_relevance_score=0.8
)

i = 0
for memory in memories:
    i = i + 1
    print(f"Top {i} Result: {memory.text} with score {memory.relevance}")
```

You can easily and conveniently access any vector database to complete related operations, which also means that you can build RAG applications very simply.

Sample

.NET Sample Please click here

Python Sample Please click here

Summary

Many enterprise data enter LLMs using Embeddings to build RAG applications. Semantic Kernel gives us a very simple way to complete related functions in both Python and .NET, so it is very helpful for those who want to add RAG applications to their projects.