Embeddings

Concept

Embeddings are used in LlamaIndex to represent your documents using a sophisticated numerical representation. Embedding models take text as input, and return a long list of numbers used to capture the semantics of the text. These embedding models have been trained to represent text this way, and help enable many applications, including search!

At a high level, if a user asks a question about dogs, then the embedding for that question will be highly similar to text that talks about dogs.

When calculating the similarity between embeddings, there are many methods to use (dot product, cosine similarity, etc.). By default, LlamaIndex uses cosine similarity when comparing embeddings.

There are many embedding models to pick from. By default, LlamaIndex uses `text-embedding-ada-002` from OpenAI. We also support any embedding model offered by Langchain here, as well as providing an easy to extend base class for implementing your own embeddings.

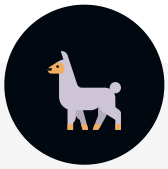
Usage Pattern

Most commonly in LlamaIndex, embedding models will be specified in the `Settings` object, and then used in a vector index. The embedding model will be used to embed the documents used during index construction, as well as embedding any queries you make using the query engine later on. You can also specify embedding models per-index.

If you don't already have your embeddings installed:

```
pip install llama-index-embeddings-openai
```

Then:



```
from llama_index.embeddings.openai import OpenAIEmbedding
from llama_index.core import VectorStoreIndex
from llama_index.core import Settings

# global
Settings.embed_model = OpenAIEmbedding()
```

```
# per-index
index = VectorStoreIndex.from_documents(documents, embed_model=embed_model)
```

To save costs, you may want to use a local model.

```
pip install llama-index-embeddings-huggingface
```

```
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from llama_index.core import Settings

Settings.embed_model = HuggingFaceEmbedding(
    model_name="BAAI/bge-small-en-v1.5"
)
```

This will use a well-performing and fast default from Hugging Face.

You can find more usage details and available customization options below.

Getting Started

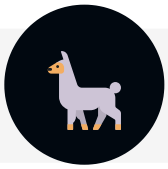
The most common usage for an embedding model will be setting it in the global `Settings` object, and then using it to construct an index and query. The input documents will be broken into nodes, and the embedding model will generate an embedding for each node.

By default, LlamaIndex will use `text-embedding-ada-002`, which is what the example below manually sets up for you.

```
from llama_index.core import VectorStoreIndex, SimpleDirectoryReader
from llama_index.embeddings.openai import OpenAIEmbedding
from llama_index.core import Settings

# global default
Settings.embed_model = OpenAIEmbedding()

documents = SimpleDirectoryReader("./data").load_data()

index = VectorStoreIndex.from_documents(documents)
```

Then, at query time, the embedding model will be used again to embed the query text.

```
query_engine = index.as_query_engine()

response = query_engine.query("query string")
```



Customization

Batch Size

By default, embeddings requests are sent to OpenAI in batches of 10. For some users, this may (rarely) incur a rate limit. For other users embedding many documents, this batch size may be too small.

```
# set the batch size to 42
embed_model = OpenAIEmbedding(embed_batch_size=42)
```

Local Embedding Models

The easiest way to use a local model is:

```
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from llama_index.core import Settings

Settings.embed_model = HuggingFaceEmbedding(
    model_name="BAAI/bge-small-en-v1.5"
)
```

HuggingFace Optimum ONNX Embeddings

LlamaIndex also supports creating and using ONNX embeddings using the Optimum library from HuggingFace. Simple create and save the ONNX embeddings, and use them.

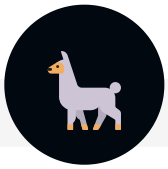
Some prerequisites:

```
pip install transformers optimum[exporters]
pip install llama-index-embeddings-huggingface-optimum
```

Creation with specifying the model and output path:

```
from llama_index.embeddings.huggingface_optimum import OptimumEmbedding

OptimumEmbedding.create_and_save_optimum_model(
    "BAAI/bge-small-en-v1.5", "./bge_onnx"
)
```

And then usage:



```
Settings.embed_model = OptimumEmbedding(folder_name="./bge_onnx")
```

LangChain Integrations

We also support any embeddings offered by Langchain here.

The example below loads a model from Hugging Face, using Langchain's embedding class.

```
pip install llama-index-embeddings-langchain
```

```
from langchain.embeddings.huggingface import HuggingFaceBgeEmbeddings
from llama_index.core import Settings

Settings.embed_model = HuggingFaceBgeEmbeddings(model_name="BAAI/bge-base-en")
```

Custom Embedding Model

If you wanted to use embeddings not offered by LlamaIndex or Langchain, you can also extend our base embeddings class and implement your own!

The example below uses Instructor Embeddings (install/setup details here), and implements a custom embeddings class. Instructor embeddings work by providing text, as well as "instructions" on the domain of the text to embed. This is helpful when embedding text from a very specific and specialized topic.

```
from typing import Any, List
from InstructorEmbedding import INSTRUCTOR
from llama_index.core.embeddings import BaseEmbedding


class InstructorEmbeddings(BaseEmbedding):
    def __init__(
        self,
        instructor_model_name: str = "hkunlp/instructor-large",
        instruction: str = "Represent the Computer Science documentation or
question:",
        **kwargs: Any,
    ) -> None:
        super().__init__(**kwargs)
        self._model = INSTRUCTOR(instructor_model_name)
        self._instruction = instruction

        def _get_query_embedding(self, query: str) -> List[float]:
            embeddings = self._model.encode([[self._instruction, query]])
            return embeddings[0]

        def _get_text_embedding(self, text: str) -> List[float]:
            embeddings = self._model.encode([[self._instruction, te
            return embeddings[0]

        def _get_text_embeddings(self, texts: List[str]) -> List[L
            embeddings = self._model.encode(
                [[self._instruction, text] for text in texts]
            )
            return embeddings

        async def _get_query_embedding(self, query: str) -> List[float]:
            return self._get_query_embedding(query)
```

```
    async def _get_text_embedding(self, text: str) -> List[float]:
        return self._get_text_embedding(text)
```

Standalone Usage

You can also use embeddings as a standalone module for your project, existing application, or general testing and exploration.

```
embeddings = embed_model.get_text_embedding(
    "It is raining cats and dogs here!"
)
```

List of supported embeddings

We support integrations with OpenAI, Azure, and anything LangChain offers.

- Azure OpenAI
- CalrifAI
- Cohere
- Custom
- Dashscope
- ElasticSearch
- FastEmbed
- Google Palm
- Gradient
- Anyscale
- Huggingface
- JinaAI
- Langchain
- LLM Rails
- MistralAI
- OpenAI
- Sagemaker
- Text Embedding Inference
- TogetherAI

- Upstage
- VoyageAI
- Nomic
- Fireworks AI