CacheBackedEmbeddings

sidebar_label: Caching

Embeddings can be stored or temporarily cached to avoid needing to recompute them.

Caching embeddings can be done using a `CacheBackedEmbeddings`. The cache backed embedder is a wrapper around an embedder that caches embeddings in a key-value store. The text is hashed and the hash is used as the key in the cache.

The main supported way to initialized a `CacheBackedEmbeddings` is `from_bytes_store`. This takes in the following parameters:

- underlying_embedder: The embedder to use for embedding.
- document_embedding_cache: Any `ByteStore` for caching document embeddings.

- namespace: (optional, defaults to `""`) The namespace to use for document cache. This namespace is used to avoid collisions with other caches. For example, set it to the name of the embedding model used.

Attention: Be sure to set the `namespace` parameter to avoid collisions of the same text embedded using different embeddings models.

```
from langchain.embeddings import
CacheBackedEmbeddings
```

Using with a Vector Store

First, let's see an example that uses the local file system for storing embeddings and uses FAISS vector store for retrieval.

```
%pip install --upgrade --quiet  langchain-
openai faiss-cpu
```

```
from langchain.storage import LocalFileStore
from langchain.text_splitter import
CharacterTextSplitter
from langchain_community.document_loaders
import TextLoader
```

```
from langchain_community.vectorstores import
FAISS
from langchain_openai import OpenAIEmbeddings

underlying_embeddings = OpenAIEmbeddings()

store = LocalFileStore("./cache/")

cached_embedder =
CacheBackedEmbeddings.from_bytes_store(
    underlying_embeddings, store,
namespace=underlying_embeddings.model
)
```

The cache is empty prior to embedding:

```
list(store.yield_keys())
```

```
[]
```

Load the document, split it into chunks, embed each chunk and load it into the vector store.

```
raw_documents =
TextLoader("../../state_of_the_union.txt").load
text_splitter =
CharacterTextSplitter(chunk_size=1000,
```

```
chunk_overlap=0)
documents =
text_splitter.split_documents(raw_documents)
```

Create the vector store:

```
%%time
db = FAISS.from_documents(documents,
cached_embedder)
```

```
CPU times: user 218 ms, sys: 29.7 ms, total:
248 ms
Wall time: 1.02 s
```

If we try to create the vector store again, it'll be much faster since it does not need to re-compute any embeddings.

```
%%time
db2 = FAISS.from_documents(documents,
cached_embedder)
```

```
CPU times: user 15.7 ms, sys: 2.22 ms, total:
18 ms
Wall time: 17.2 ms
```

And here are some of the embeddings that got created:

```
list(store.yield_keys())[:5]
```

```
['text-embedding-ada-00217a6727d-8916-54eb-
b196-ec9c9d6ca472',
 'text-embedding-ada-0025fc0d904-bd80-52da-
95c9-441015bfb438',
 'text-embedding-ada-002e4ad20ef-dfaa-5916-
9459-f90c6d8e8159',
 'text-embedding-ada-002ed199159-c1cd-5597-
9757-f80498e8f17b',
 'text-embedding-ada-0021297d37a-2bc1-5e19-
bf13-6c950f075062']
```

Swapping the `ByteStore`

In order to use a different `ByteStore`, just use it when creating your `CacheBackedEmbeddings`. Below, we create an equivalent cached embeddings object, except using the non-persistent `InMemoryByteStore` instead:

```
from langchain.embeddings import
CacheBackedEmbeddings
from langchain.storage import
InMemoryByteStore
```

```
store = InMemoryByteStore()

cached_embedder =
CacheBackedEmbeddings.from_bytes_store(
    underlying_embeddings, store,
namespace=underlying_embeddings.model
)
```