BAAI

/

bge-large-zh-v1.5

like224

Feature Extractionsentence-transformersPyTorchSafetensorsTransformersChinesebertsentence-similarityInference Endpoints5 papers

License:mit

Model card Files and versionsCommunity
15

Deploy
Use in libraries
Edit model card

FlagEmbedding

Model List | FAQ | Usage | Evaluation | Train | Contact | Citation | License

For more details please refer to our Github: FlagEmbedding.

If you are looking for a model that supports more languages, longer texts, and other retrieval methods, you can try using bge-m3.

English | 中文

FlagEmbedding focuses on retrieval-augmented LLMs, consisting of the following projects currently:

- **Long-Context LLM**: Activation Beacon
- **Fine-tuning of LM** : LM-Cocktail
- **Dense Retrieval**: BGE-M3, LLM Embedder, BGE Embedding
- **Reranker Model**: BGE Reranker
- **Benchmark**: C-MTEB

News

- 1/30/2024: Release **BGE-M3**, a new member to BGE model series! M3 stands for **M**ulti-linguality (100+ languages), **M**ulti-granularities (input length up to 8192), **M**ulti-Functionality (unification of dense, lexical, multi-vec/colbert retrieval). It is the first embedding model which supports all three retrieval methods, achieving new SOTA on multi-lingual (MIRACL) and cross-lingual (MKQA) benchmarks. Technical Report and Code. :fire:
- 1/9/2024: Release Activation-Beacon, an effective, efficient, compatible, and low-cost (training) method to extend the context length of LLM. Technical Report :fire:
- 12/24/2023: Release **LLaRA**, a LLaMA-7B based dense retriever, leading to state-of-the-art performances on MS MARCO and BEIR. Model and code will be open-sourced. Please stay tuned. Technical Report :fire:
- 11/23/2023: Release LM-Cocktail, a method to maintain general capabilities during fine-tuning by merging multiple language models. Technical Report :fire:
- 10/12/2023: Release LLM-Embedder, a unified embedding model to support diverse retrieval augmentation needs for LLMs. Technical Report
- 09/15/2023: The technical report and massive training data of BGE has been released
- 09/12/2023: New models:
    - **New reranker model**: release cross-encoder models `BAAI/bge-reranker-base` and `BAAI/bge-reranker-large`, which are more powerful than embedding model. We recommend to use/fine-tune them to re-rank top-k documents returned by embedding models.

- **update embedding model**: release `bge-*-v1.5` embedding model to alleviate the issue of the similarity distribution, and enhance its retrieval ability without instruction.

More

-
-
-
-
-

Model List

`bge` is short for `BAAI general embedding`.

Model	Language		Description	query instruction for retrieval [1]
BAAI/bge-m3	Multilingual	Inference Fine-tune	Multi-Functionality(dense retrieval, sparse retrieval, multi-vector(colbert)), Multi-Linguality, and Multi-Granularity(8192 tokens)	
BAAI/llm-embedder	English	Inference Fine-tune	a unified embedding model to support diverse retrieval augmentation needs for LLMs	See README
BAAI/bge-reranker-large	Chinese and English	Inference Fine-tune	a cross-encoder model which is more accurate but less efficient [2]	

Model	Language		Description	query instruction for retrieval [1]
BAAI/bge-reranker-base	Chinese and English	Inference Fine-tune	a cross-encoder model which is more accurate but less efficient [2]	
BAAI/bge-large-en-v1.5	English	Inference Fine-tune	version 1.5 with more reasonable similarity distribution	`Represent this sentence for searching relevant passages:`
BAAI/bge-base-en-v1.5	English	Inference Fine-tune	version 1.5 with more reasonable similarity distribution	`Represent this sentence for searching relevant passages:`
BAAI/bge-small-en-v1.5	English	Inference Fine-tune	version 1.5 with more reasonable similarity distribution	`Represent this sentence for searching relevant passages:`

Model	Language		Description	query instruction for retrieval [1]
BAAI/bge-large-zh-v1.5	Chinese	Inference Fine-tune	version 1.5 with more reasonable similarity distribution	`为这个句子生成表示以用于检索相关文章：`
BAAI/bge-base-zh-v1.5	Chinese	Inference Fine-tune	version 1.5 with more reasonable similarity distribution	`为这个句子生成表示以用于检索相关文章：`
BAAI/bge-small-zh-v1.5	Chinese	Inference Fine-tune	version 1.5 with more reasonable similarity distribution	`为这个句子生成表示以用于检索相关文章：`
BAAI/bge-large-en	English	Inference Fine-tune	:trophy: rank **1st** in MTEB leaderboard	`Represent this sentence for searching relevant passages:`
BAAI/bge-base-en	English	Inference Fine-tune	a base-scale model but with similar ability to `bge-large-en`	`Represent this sentence for`

Model	Language		Description	query instruction for retrieval [1]
				`searching relevant passages:`
[BAAI/bge-small-en](#)	English	[Inference](#) [Fine-tune](#)	a small-scale model but with competitive performance	`Represent this sentence for searching relevant passages:`
[BAAI/bge-large-zh](#)	Chinese	[Inference](#) [Fine-tune](#)	:trophy: rank **1st** in [C-MTEB](#) benchmark	`为这个句子生成表示以用于检索相关文章：`
[BAAI/bge-base-zh](#)	Chinese	[Inference](#) [Fine-tune](#)	a base-scale model but with similar ability to `bge-large-zh`	`为这个句子生成表示以用于检索相关文章：`
[BAAI/bge-small-zh](#)	Chinese	[Inference](#) [Fine-tune](#)	a small-scale model but with competitive performance	`为这个句子生成表示以用于检索相关文章：`

[1]: If you need to search the relevant passages to a query, we suggest to add the instruction to the query; in other cases, no instruction is needed, just use the original query directly. In all cases, **no instruction** needs to be added to passages.

[2]: Different from embedding model, reranker uses question and document as input and directly output similarity instead of embedding. To balance the accuracy and time cost, cross-encoder is widely used to re-rank top-k documents retrieved by other simple models. For examples, use bge embedding model to retrieve top 100 relevant documents, and then use bge reranker to re-rank the top 100 document to get the final top-3 results.

All models have been uploaded to Huggingface Hub, and you can see them at https://huggingface.co/BAAI. If you cannot open the Huggingface Hub, you also can download the models at https://model.baai.ac.cn/models .

Frequently asked questions

1. How to fine-tune bge embedding model?
   -
   -
   -
2. The similarity score between two dissimilar sentences is higher than 0.5
3. When does the query instruction need to be used

Usage

Usage for Embedding Model

Here are some examples for using `bge` models with FlagEmbedding, Sentence-Transformers, Langchain, or Huggingface Transformers.

Using FlagEmbedding

```
pip install -U FlagEmbedding
```

If it doesn't work for you, you can see FlagEmbedding for more methods to install FlagEmbedding.

```
from FlagEmbedding import FlagModel

sentences_1 = ["样例数据-1", "样例数据-2"]

sentences_2 = ["样例数据-3", "样例数据-4"]
```

```
model = FlagModel('BAAI/bge-large-zh-v1.5',

            query_instruction_for_retrieval="为这个句子生成表示以用于检索相关文章：",

            use_fp16=True) # Setting use_fp16 to True speeds up computation with a slight performance degradation

embeddings_1 = model.encode(sentences_1)

embeddings_2 = model.encode(sentences_2)

similarity = embeddings_1 @ embeddings_2.T

print(similarity)



# for s2p(short query to long passage) retrieval task, suggest to use encode_queries() which will automatically add the instruction to each query

# corpus in retrieval task can still use encode() or encode_corpus(), since they don't need instruction

queries = ['query_1', 'query_2']

passages = ["样例文档-1", "样例文档-2"]

q_embeddings = model.encode_queries(queries)

p_embeddings = model.encode(passages)

scores = q_embeddings @ p_embeddings.T
```

For the value of the argument `query_instruction_for_retrieval`, see Model List.

By default, FlagModel will use all available GPUs when encoding. Please set `os.environ["CUDA_VISIBLE_DEVICES"]` to select specific GPUs. You also can set `os.environ["CUDA_VISIBLE_DEVICES"]=""` to make all GPUs unavailable.

Using Sentence-Transformers

You can also use the `bge` models with sentence-transformers:

```
pip install -U sentence-transformers

from sentence_transformers import SentenceTransformer

sentences_1 = ["样例数据-1", "样例数据-2"]

sentences_2 = ["样例数据-3", "样例数据-4"]

model = SentenceTransformer('BAAI/bge-large-zh-v1.5')

embeddings_1 = model.encode(sentences_1, normalize_embeddings=True)

embeddings_2 = model.encode(sentences_2, normalize_embeddings=True)

similarity = embeddings_1 @ embeddings_2.T

print(similarity)
```

For s2p(short query to long passage) retrieval task, each short query should start with an instruction (instructions see [Model List]). But the instruction is not needed for passages.

```
from sentence_transformers import SentenceTransformer

queries = ['query_1', 'query_2']

passages = ["样例文档-1", "样例文档-2"]

instruction = "为这个句子生成表示以用于检索相关文章："



model = SentenceTransformer('BAAI/bge-large-zh-v1.5')

q_embeddings = model.encode([instruction+q for q in queries], normalize_embeddings=True)

p_embeddings = model.encode(passages, normalize_embeddings=True)

scores = q_embeddings @ p_embeddings.T
```

Using Langchain

You can use `bge` in langchain like this:

```
from langchain.embeddings import HuggingFaceBgeEmbeddings

model_name = "BAAI/bge-large-en-v1.5"

model_kwargs = {'device': 'cuda'}

encode_kwargs = {'normalize_embeddings': True} # set True to compute cosine similarity

model = HuggingFaceBgeEmbeddings(

  model_name=model_name,

  model_kwargs=model_kwargs,

  encode_kwargs=encode_kwargs,

  query_instruction="为这个句子生成表示以用于检索相关文章："

)

model.query_instruction = "为这个句子生成表示以用于检索相关文章："
```

Using HuggingFace Transformers

With the transformers package, you can use the model like this: First, you pass your input through the transformer model, then you select the last hidden state of the first token (i.e., [CLS]) as the sentence embedding.

```
from transformers import AutoTokenizer, AutoModel

import torch

# Sentences we want sentence embeddings for

sentences = ["样例数据-1", "样例数据-2"]
```

```python
# Load model from HuggingFace Hub

tokenizer = AutoTokenizer.from_pretrained('BAAI/bge-large-zh-v1.5')

model = AutoModel.from_pretrained('BAAI/bge-large-zh-v1.5')

model.eval()



# Tokenize sentences

encoded_input = tokenizer(sentences, padding=True, truncation=True, return_tensors='pt')

# for s2p(short query to long passage) retrieval task, add an instruction to query (not add instruction for passages)

# encoded_input = tokenizer([instruction + q for q in queries], padding=True, truncation=True, return_tensors='pt')



# Compute token embeddings

with torch.no_grad():

  model_output = model(**encoded_input)

  # Perform pooling. In this case, cls pooling.

  sentence_embeddings = model_output[0][:, 0]

# normalize embeddings

sentence_embeddings = torch.nn.functional.normalize(sentence_embeddings, p=2, dim=1)

print("Sentence embeddings:", sentence_embeddings)
```

Usage for Reranker

Different from embedding model, reranker uses question and document as input and directly output similarity instead of embedding. You can get a relevance score by inputting query and passage to the reranker. The reranker is optimized based cross-entropy loss, so the relevance score is not bounded to a specific range.

Using FlagEmbedding

```
pip install -U FlagEmbedding
```

Get relevance scores (higher scores indicate more relevance):

```
from FlagEmbedding import FlagReranker

reranker = FlagReranker('BAAI/bge-reranker-large', use_fp16=True) # Setting use_fp16 to True speeds up computation with a slight performance degradation



score = reranker.compute_score(['query', 'passage'])

print(score)



scores = reranker.compute_score([['what is panda?', 'hi'], ['what is panda?', 'The giant panda (Ailuropoda melanoleuca), sometimes called a panda bear or simply panda, is a bear species endemic to China.']])

print(scores)
```

Using Huggingface transformers

```
import torch

from transformers import AutoModelForSequenceClassification, AutoTokenizer



tokenizer = AutoTokenizer.from_pretrained('BAAI/bge-reranker-large')

model = AutoModelForSequenceClassification.from_pretrained('BAAI/bge-reranker-large')

model.eval()
```

```python
pairs = [['what is panda?', 'hi'], ['what is panda?', 'The giant panda (Ailuropoda melanoleuca), sometimes called a panda bear or simply panda, is a bear species endemic to China.']]

with torch.no_grad():

    inputs = tokenizer(pairs, padding=True, truncation=True, return_tensors='pt', max_length=512)

    scores = model(**inputs, return_dict=True).logits.view(-1, ).float()

    print(scores)
```