[![Logo](../../../_static/logo.png)](../../../index.html)

Getting Started

* [Installation](../../installation.html)
  * [Install with pip](../../installation.html#install-with-pip)
  * [Install with Conda](../../installation.html#install-with-conda)
  * [Install from Source](../../installation.html#install-from-source)
  * [Editable Install](../../installation.html#editable-install)
  * [Install PyTorch with CUDA support](../../installation.html#install-pytorch-with-cuda-support)
* [Quickstart](../../quickstart.html)
  * [Sentence Transformer](../../quickstart.html#sentence-transformer)
  * [Cross Encoder](../../quickstart.html#cross-encoder)
  * [Next Steps](../../quickstart.html#next-steps)

Sentence Transformer

* [Usage](../../sentence_transformer/usage/usage.html)
  * [Computing Embeddings](../../../examples/applications/computing-embeddings/README.html)
    * [Initializing a Sentence Transformer Model](../../../examples/applications/computing-embeddings/README.html#initializing-a-sentence-transformer-model)
    * [Calculating Embeddings](../../../examples/applications/computing-embeddings/README.html#calculating-embeddings)
    * [Prompt Templates](../../../examples/applications/computing-embeddings/README.html#prompt-templates)

* [Input Sequence Length](../../../examples/applications/computing-embeddings/README.html#id1)
* [Multi-Process / Multi-GPU Encoding](../../../examples/applications/computing-embeddings/README.html#multi-process-multi-gpu-encoding)
* [Semantic Textual Similarity](../../sentence_transformer/usage/semantic_textual_similarity.html)
* [Similarity Calculation](../../sentence_transformer/usage/semantic_textual_similarity.html#similarity-calculation)
* [Semantic Search](../../../examples/applications/semantic-search/README.html)
* [Background](../../../examples/applications/semantic-search/README.html#background)
* [Symmetric vs. Asymmetric Semantic Search](../../../examples/applications/semantic-search/README.html#symmetric-vs-asymmetric-semantic-search)
* [Manual Implementation](../../../examples/applications/semantic-search/README.html#manual-implementation)
* [Optimized Implementation](../../../examples/applications/semantic-search/README.html#optimized-implementation)
* [Speed Optimization](../../../examples/applications/semantic-search/README.html#speed-optimization)
* [Elasticsearch](../../../examples/applications/semantic-search/README.html#elasticsearch)
* [Approximate Nearest Neighbor](../../../examples/applications/semantic-search/README.html#approximate-nearest-neighbor)
* [Retrieve & Re-Rank](../../../examples/applications/semantic-search/README.html#retrieve-re-rank)

* [Examples](../../../examples/applications/semantic-search/README.html#examples)
* [Retrieve & Re-Rank](../../../examples/applications/retrieve_rerank/README.html)
* [Retrieve & Re-Rank Pipeline](../../../examples/applications/retrieve_rerank/README.html#retrieve-re-rank-pipeline)
* [Retrieval: Bi-Encoder](../../../examples/applications/retrieve_rerank/README.html#retrieval-bi-encoder)
* [Re-Ranker: Cross-Encoder](../../../examples/applications/retrieve_rerank/README.html#re-ranker-cross-encoder)
* [Example Scripts](../../../examples/applications/retrieve_rerank/README.html#example-scripts)
* [Pre-trained Bi-Encoders (Retrieval)](../../../examples/applications/retrieve_rerank/README.html#pre-trained-bi-encoders-retrieval)
* [Pre-trained Cross-Encoders (Re-Ranker)](../../../examples/applications/retrieve_rerank/README.html#pre-trained-cross-encoders-re-ranker)
* [Clustering](../../../examples/applications/clustering/README.html)
* [k-Means](../../../examples/applications/clustering/README.html#k-means)
* [Agglomerative Clustering](../../../examples/applications/clustering/README.html#agglomerative-clustering)
* [Fast Clustering](../../../examples/applications/clustering/README.html#fast-clustering)
* [Topic Modeling](../../../examples/applications/clustering/README.html#topic-modeling)
* [Paraphrase Mining](../../../examples/applications/paraphrase-mining/README.html)
* [`paraphrase_mining()`](../../../examples/applications/paraphrase-mining/README.html#sentence_transformers.util.paraphrase_mining)
* [Translated Sentence

Mining](../../../examples/applications/parallel-sentence-mining/README.html)

* [Margin Based Mining](../../../examples/applications/parallel-sentence-mining/README.html#margin-based-mining)

* [Examples](../../../examples/applications/parallel-sentence-mining/README.html#examples)

* [Image Search](../../../examples/applications/image-search/README.html)

* [Installation](../../../examples/applications/image-search/README.html#installation)

* [Usage](../../../examples/applications/image-search/README.html#usage)

* [Examples](../../../examples/applications/image-search/README.html#examples)

* [Embedding Quantization](../../../examples/applications/embedding-quantization/README.html)

* [Binary Quantization](../../../examples/applications/embedding-quantization/README.html#binary-quantization)

* [Scalar (int8) Quantization](../../../examples/applications/embedding-quantization/README.html#scalar-int8-quantization)

* [Additional extensions](../../../examples/applications/embedding-quantization/README.html#additional-extensions)

* [Demo](../../../examples/applications/embedding-quantization/README.html#demo)

* [Try it yourself](../../../examples/applications/embedding-quantization/README.html#try-it-yourself)

* [Speeding up Inference](../../sentence_transformer/usage/efficiency.html)

* [PyTorch](../../sentence_transformer/usage/efficiency.html#pytorch)

* [ONNX](../../sentence_transformer/usage/efficiency.html#onnx)

* [OpenVINO](../../sentence_transformer/usage/efficiency.html#openvino)

* [Benchmarks](../../sentence_transformer/usage/efficiency.html#benchmarks)

* [Creating Custom Models](../../sentence_transformer/usage/custom_models.html)

gloss)

* [Paraphrase Data](../../../examples/training/paraphrases/README.html)

* [Pre-Trained Models](../../../examples/training/paraphrases/README.html#pre-trained-models)

* [Quora Duplicate Questions](../../../examples/training/quora_duplicate_questions/README.html)

* [Training](../../../examples/training/quora_duplicate_questions/README.html#training)

* [MultipleNegativesRankingLoss](../../../examples/training/quora_duplicate_questions/README.html#multiplenegativesrankingloss)

* [Pretrained Models](../../../examples/training/quora_duplicate_questions/README.html#pretrained-models)

* [MS MARCO](../../../examples/training/ms_marco/README.html)

* [Bi-Encoder](../../../examples/training/ms_marco/README.html#bi-encoder)

* [Matryoshka Embeddings](../../../examples/training/matryoshka/README.html)

* [Use Cases](../../../examples/training/matryoshka/README.html#use-cases)

* [Results](../../../examples/training/matryoshka/README.html#results)

* [Training](../../../examples/training/matryoshka/README.html#training)

* [Inference](../../../examples/training/matryoshka/README.html#inference)

* [Code Examples](../../../examples/training/matryoshka/README.html#code-examples)

* [Adaptive Layers](../../../examples/training/adaptive_layer/README.html)

* [Use Cases](../../../examples/training/adaptive_layer/README.html#use-cases)

* [Results](../../../examples/training/adaptive_layer/README.html#results)

* [Training](../../../examples/training/adaptive_layer/README.html#training)

* [Inference](../../../examples/training/adaptive_layer/README.html#inference)

* [Code Examples](../../../examples/training/adaptive_layer/README.html#code-examples)

* [Multilingual Models](../../../examples/training/multilingual/README.html)

* [Extend your own models](../../../examples/training/multilingual/README.html#extend-your-own-models)

sentence-pairs)](../../../examples/training/data_augmentation/README.html#scenario-2-no-annotated-datasets-only-unlabeled-sentence-pairs)

* [Training](../../../examples/training/data_augmentation/README.html#training)

* [Citation](../../../examples/training/data_augmentation/README.html#citation)

* [Training with Prompts](../../../examples/training/prompts/README.html)

* [What are Prompts?](../../../examples/training/prompts/README.html#what-are-prompts)

* [Why would we train with Prompts?](../../../examples/training/prompts/README.html#why-would-we-train-with-prompts)

* [How do we train with Prompts?](../../../examples/training/prompts/README.html#how-do-we-train-with-prompts)

* [Training with PEFT Adapters](../../../examples/training/peft/README.html)

* [Compatibility Methods](../../../examples/training/peft/README.html#compatibility-methods)

* [Adding a New Adapter](../../../examples/training/peft/README.html#adding-a-new-adapter)

* [Loading a Pretrained Adapter](../../../examples/training/peft/README.html#loading-a-pretrained-adapter)

* [Training Script](../../../examples/training/peft/README.html#training-script)

* [Unsupervised Learning](../../../examples/unsupervised_learning/README.html)

* [TSDAE](../../../examples/unsupervised_learning/README.html#tsdae)

* [SimCSE](../../../examples/unsupervised_learning/README.html#simcse)

* [CT](../../../examples/unsupervised_learning/README.html#ct)

* [CT (In-Batch Negative Sampling)](../../../examples/unsupervised_learning/README.html#ct-in-batch-negative-sampling)

* [Masked Language Model (MLM)](../../../examples/unsupervised_learning/README.html#masked-language-model-mlm)

* [GenQ](../../../examples/unsupervised_learning/README.html#genq)

* [GPL](../../../examples/unsupervised_learning/README.html#gpl)

* [Performance

Comparison](../../../examples/unsupervised_learning/README.html#performance-comparison)

* [Domain Adaptation](../../../examples/domain_adaptation/README.html)

* [Domain Adaptation vs. Unsupervised Learning](../../../examples/domain_adaptation/README.html#domain-adaptation-vs-unsupervised-learning)

* [Adaptive Pre-Training](../../../examples/domain_adaptation/README.html#adaptive-pre-training)

* [GPL: Generative Pseudo-Labeling](../../../examples/domain_adaptation/README.html#gpl-generative-pseudo-labeling)

* [Hyperparameter Optimization](../../../examples/training/hpo/README.html)

* [HPO Components](../../../examples/training/hpo/README.html#hpo-components)

* [Putting It All Together](../../../examples/training/hpo/README.html#putting-it-all-together)

* [Example Scripts](../../../examples/training/hpo/README.html#example-scripts)

* [Distributed Training](../../sentence_transformer/training/distributed.html)

* [Comparison](../../sentence_transformer/training/distributed.html#comparison)

* [FSDP](../../sentence_transformer/training/distributed.html#fsdp)

Cross Encoder

* [Usage](../../cross_encoder/usage/usage.html)

* [Retrieve & Re-Rank](../../../examples/applications/retrieve_rerank/README.html)

* [Retrieve & Re-Rank Pipeline](../../../examples/applications/retrieve_rerank/README.html#retrieve-re-rank-pipeline)

* [Retrieval: Bi-Encoder](../../../examples/applications/retrieve_rerank/README.html#retrieval-bi-encoder)

* [Re-Ranker:

Cross-Encoder](../../../examples/applications/retrieve_rerank/README.html#re-ranker-cross-encoder)

* [Example Scripts](../../../examples/applications/retrieve_rerank/README.html#example-scripts)

* [Pre-trained Bi-Encoders (Retrieval)](../../../examples/applications/retrieve_rerank/README.html#pre-trained-bi-encoders-retrieval)

* [Pre-trained Cross-Encoders (Re-Ranker)](../../../examples/applications/retrieve_rerank/README.html#pre-trained-cross-encoders-re-ranker)

* [Pretrained Models](../../cross_encoder/pretrained_models.html)

* [MS MARCO](../../cross_encoder/pretrained_models.html#ms-marco)

* [SQuAD (QNLI)](../../cross_encoder/pretrained_models.html#squad-qnli)

* [STSbenchmark](../../cross_encoder/pretrained_models.html#stsbenchmark)

* [Quora Duplicate Questions](../../cross_encoder/pretrained_models.html#quora-duplicate-questions)

* [NLI](../../cross_encoder/pretrained_models.html#nli)

* [Community Models](../../cross_encoder/pretrained_models.html#community-models)

* [Training Overview](../../cross_encoder/training_overview.html)

* [Training Examples](../../cross_encoder/training/examples.html)

* [MS MARCO](../../../examples/training/ms_marco/cross_encoder_README.html)

* [Cross-Encoder](../../../examples/training/ms_marco/cross_encoder_README.html#cross-encoder)

* [Cross-Encoder Knowledge Distillation](../../../examples/training/ms_marco/cross_encoder_README.html#cross-encoder-knowledge-distillation)

Package Reference

    * [DenoisingAutoEncoderLoss](losses.html#denoisingautoencoderloss)
    * [GISTEmbedLoss](losses.html#gistembedloss)
    * [CachedGISTEmbedLoss](losses.html#cachedgistembedloss)
    * [MSELoss](losses.html#mseloss)
    * [MarginMSELoss](losses.html#marginmseloss)
    * [MatryoshkaLoss](losses.html#matryoshkaloss)
    * [Matryoshka2dLoss](losses.html#matryoshka2dloss)
    * [AdaptiveLayerLoss](losses.html#adaptivelayerloss)
    * [MegaBatchMarginLoss](losses.html#megabatchmarginloss)
    * [MultipleNegativesRankingLoss](losses.html#multiplenegativesrankingloss)
    * [CachedMultipleNegativesRankingLoss](losses.html#cachedmultiplenegativesrankingloss)
    * [MultipleNegativesSymmetricRankingLoss](losses.html#multiplenegativessymmetricrankingloss)
    * [CachedMultipleNegativesSymmetricRankingLoss](losses.html#cachedmultiplenegativessymmetricrankingloss)
    * [SoftmaxLoss](losses.html#softmaxloss)
    * [TripletLoss](losses.html#tripletloss)
  * [Samplers](sampler.html)
    * [BatchSamplers](sampler.html#batchsamplers)
    * [MultiDatasetBatchSamplers](sampler.html#multidatasetbatchsamplers)
  * [Evaluation](evaluation.html)
    * [BinaryClassificationEvaluator](evaluation.html#binaryclassificationevaluator)
    * [EmbeddingSimilarityEvaluator](evaluation.html#embeddingsimilarityevaluator)
    * [InformationRetrievalEvaluator](evaluation.html#informationretrievalevaluator)
    * [NanoBEIREvaluator](evaluation.html#nanobeirevaluator)
    * [MSEEvaluator](evaluation.html#mseevaluator)

*

[`export_static_quantized_openvino_model()`](../util.html#sentence_transformers.backend.export_static_quantized_openvino_model)

* [Similarity Metrics](../util.html#module-sentence_transformers.util)
* [`cos_sim()`](../util.html#sentence_transformers.util.cos_sim)
* [`dot_score()`](../util.html#sentence_transformers.util.dot_score)
* [`euclidean_sim()`](../util.html#sentence_transformers.util.euclidean_sim)
* [`manhattan_sim()`](../util.html#sentence_transformers.util.manhattan_sim)
* [`pairwise_cos_sim()`](../util.html#sentence_transformers.util.pairwise_cos_sim)
* [`pairwise_dot_score()`](../util.html#sentence_transformers.util.pairwise_dot_score)
* [`pairwise_euclidean_sim()`](../util.html#sentence_transformers.util.pairwise_euclidean_sim)
* [`pairwise_manhattan_sim()`](../util.html#sentence_transformers.util.pairwise_manhattan_sim)

__[Sentence Transformers](../../../index.html)

* [](../../../index.html)
* [Sentence Transformer](index.html)
* Models
* [Edit on GitHub](https://github.com/UKPLab/sentence-transformers/blob/master/docs/package_reference/sentence_transformer/models.md)

* * *

Modelsï*f*•

`sentence_transformers.models` defines different building blocks, that can be

used to create SentenceTransformer networks from scratch. For more details,

see [Training Overview](../../sentence_transformer/training_overview.html).

Main Classesïƒ•

_class _sentence_transformers.models.Transformer(_model_name_or_path : str_, _max_seq_length : int | None = None_, _model_args : dict[str, Any] | None = None_, _tokenizer_args : dict[str, Any] | None = None_, _config_args : dict[str, Any] | None = None_, _cache_dir : str | None = None_, _do_lower_case : bool = False_, _tokenizer_name_or_path : str | None = None_, _backend : str = 'torch'_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\models\\Transformer.py#L31-L551)ïƒ•

Hugging Face AutoModel to generate token embeddings. Loads the correct class,

e.g. BERT / RoBERTa etc.

Parameters:

* **model_name_or_path** â€“ Hugging Face models name (<https://huggingface.co/models>)

* **max_seq_length** â€“ Truncate any inputs longer than max_seq_length

* **model_args** â€“ Keyword arguments passed to the Hugging Face Transformers model

* **tokenizer_args** â€“ Keyword arguments passed to the Hugging Face Transformers tokenizer

* **config_args** â€“ Keyword arguments passed to the Hugging Face Transformers config

* **cache_dir** â€“ Cache dir for Hugging Face Transformers to store/load models

* **do_lower_case** â€“ If true, lowercases the input (independent if the model is cased or not)

* **tokenizer_name_or_path** â€“ Name or path of the tokenizer. When None, then model_name_or_path is used

* **backend** â€“ Backend used for model inference. Can be torch, onnx, or openvino. Default is torch.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

_class _sentence_transformers.models.Pooling(_word_embedding_dimension : int_, _pooling_mode : str | None = None_, _pooling_mode_cls_token : bool = False_, _pooling_mode_max_tokens : bool = False_, _pooling_mode_mean_tokens : bool = True_, _pooling_mode_mean_sqrt_len_tokens : bool = False_, _pooling_mode_weightedmean_tokens : bool = False_, _pooling_mode_lasttoken : bool = False_, _include_prompt : bool = True_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\models\\Pooling.py#L11-L260)ïƒ•

Performs pooling (max or mean) on the token embeddings.

Using pooling, it generates from a variable sized sentence a fixed sized sentence embedding. This layer also allows to use the CLS token if it is returned by the underlying word embedding model. You can concatenate multiple poolings together.

Parameters:

* **word_embedding_dimension** â€“ Dimensions for the word embeddings

* **pooling_mode** â€“ Either â€œclsâ€•, â€œlasttokenâ€•, â€œmaxâ€•, â€œmeanâ€•, â€œmean_sqrt_len_tokensâ€•, or â€œweightedmeanâ€•. If set, overwrites the other pooling_mode_* settings

* **pooling_mode_cls_token** â€“ Use the first token (CLS token) as text representations

* **pooling_mode_max_tokens** â€“ Use max in each dimension over all tokens.

* **pooling_mode_mean_tokens** â€“ Perform mean-pooling

* **pooling_mode_mean_sqrt_len_tokens** â€“ Perform mean-pooling, but divide by sqrt(input_length).

* **pooling_mode_weightedmean_tokens** â€“ Perform (position) weighted mean pooling. See [SGPT: GPT Sentence Embeddings for Semantic Search](https://arxiv.org/abs/2202.08904).

* **pooling_mode_lasttoken** â€“

Perform last token pooling. See [SGPT: GPT Sentence Embeddings for Semantic Search](https://arxiv.org/abs/2202.08904) and [Text and Code Embeddings by Contrastive Pre-Training](https://arxiv.org/abs/2201.10005).

* **include_prompt** â€“ If set to false, the prompt tokens are not included in the pooling. This is useful for reproducing work that does not include the prompt tokens in the pooling like INSTRUCTOR, but otherwise not recommended.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

_class _sentence_transformers.models.Dense(_in_features : int_, _out_features : int_, _bias : bool = True_, _activation_function =Tanh()_, _init_weight : [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\") | None = None_, _init_bias : [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\") | None = None_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\models\\Dense.py#L14-L94)ïƒ•

Feed-forward function with activation function.

This layer takes a fixed-sized sentence embedding and passes it through a feed-forward layer. Can be used to generate deep averaging networks (DAN).

Parameters:

* **in_features** â€“ Size of the input dimension

* **out_features** â€“ Output size

* **bias** â€“ Add a bias vector

* **activation_function** â€“ Pytorch activation function applied on output

* **init_weight** â€“ Initial value for the matrix of the linear layer

* **init_bias** â€“ Initial value for the bias of the linear layer

Initializes internal Module state, shared by both nn.Module and ScriptModule.

Further Classesïƒ•

_class _sentence_transformers.models.Asym(_sub_modules : dict[str, list[[Module](https://pytorch.org/docs/stable/generated/torch.nn.Module.html#torch.nn.Module "\(in PyTorch v2.5\")]]_, _allow_empty_key : bool = True_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\models\\Asym.py#L12-L132)ïƒ•

This model allows to create asymmetric SentenceTransformer models, that apply different models depending on the specified input key.

In the below example, we create two different Dense models for â€˜queryâ€™ and â€˜docâ€™. Text that is passed as {â€˜queryâ€™: â€˜My queryâ€™} will be passed along along the first Dense model, and text that will be passed as {â€˜docâ€™: â€˜My documentâ€™} will use the other Dense model.

Note, that when you call encode(), that only inputs of the same type can be encoded. Mixed-Types cannot be encoded.

Example::

word_embedding_model = models.Transformer(model_name) pooling_model = models.Pooling(word_embedding_model.get_word_embedding_dimension()) asym_model = models.Asym({â€˜queryâ€™: [models.Dense(word_embedding_model.get_word_embedding_dimension(), 128)], â€˜docâ€™: [models.Dense(word_embedding_model.get_word_embedding_dimension(), 128)]}) model = SentenceTransformer(modules=[word_embedding_model, pooling_model, asym_model])

model.encode([{â€˜queryâ€™: â€˜Q1â€™}, {â€˜queryâ€™: â€˜Q2â€™}]
model.encode([{â€˜docâ€™: â€˜Doc1â€™}, {â€˜docâ€™: â€˜Doc2â€™}]

#You can train it with InputExample like this. Note, that the order must

always be the same: train_example = InputExample(texts=[{â€˜queryâ€™: â€˜Train queryâ€™}, {â€˜docâ€™: â€˜Documentâ€™}], label=1)

Parameters:

* **sub_modules** â€“ Dict in the format str -> List[models]. The models in the specified list will be applied for input marked with the respective key.

* **allow_empty_key** â€“ If true, inputs without a key can be processed. If false, an exception will be thrown if no key is specified.

_class _sentence_transformers.models.BoW(_vocab : list[str]_, _word_weights : dict[str, float] = {}_, _unknown_word_weight : float = 1_, _cumulative_term_frequency : bool = True_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\models\\BoW.py#L16-L96)ïƒ•

Implements a Bag-of-Words (BoW) model to derive sentence embeddings.

A weighting can be added to allow the generation of tf-idf vectors. The output vector has the size of the vocab.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

_class _sentence_transformers.models.CNN(_in_word_embedding_dimension : int_, _out_channels : int = 256_, _kernel_sizes : list[int] = [1, 3, 5]_, _stride_sizes : list[int] | None = None_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\models\\CNN.py#L12-L88)ïƒ•

CNN-layer with multiple kernel-sizes over the word embeddings

Initializes internal Module state, shared by both nn.Module and ScriptModule.

_class _sentence_transformers.models.LSTM(_word_embedding_dimension : int_, _hidden_dim : int_, _num_layers : int = 1_, _dropout : float = 0_, _bidirectional : bool = True_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\models\\LSTM.py#L12-L90)ïƒ•

Bidirectional LSTM running over word embeddings.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

_class _sentence_transformers.models.Normalize[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\models\\Normalize.py#L7-L22)ïƒ•

This layer normalizes embeddings to unit length

Initializes internal Module state, shared by both nn.Module and ScriptModule.

_class _sentence_transformers.models.StaticEmbedding(_tokenizer : Tokenizer | PreTrainedTokenizerFast_, _embedding_weights : ndarray | [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)") | None = None_, _embedding_dim : int | None = None_, _**kwargs_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\models\\StaticEmbedding.py#L18-L220)ïƒ•

Initializes the StaticEmbedding model given a tokenizer. The model is a simple
embedding bag model that takes the mean of trained per-token embeddings to
compute text embeddings.

Parameters:

* **tokenizer** (_Tokenizer_ _|__PreTrainedTokenizerFast_) â€“ The tokenizer to be used. Must be a fast tokenizer from `transformers` or `tokenizers`.

* **embedding_weights** (_np.ndarray_ _|_[_torch.Tensor_](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)") _|__None_ _,__optional_) â€“ Pre-trained embedding weights. Defaults to None.

* **embedding_dim** (_int_ _|__None_ _,__optional_) â€“ Dimension of the embeddings. Required if embedding_weights is not provided. Defaults to None.

Example:

```
from sentence_transformers import SentenceTransformer
from sentence_transformers.models import StaticEmbedding
from tokenizers import Tokenizer

# Pre-distilled embeddings:
static_embedding = StaticEmbedding.from_model2vec("minishlab/potion-base-8M")
# or distill your own embeddings:
static_embedding = StaticEmbedding.from_distillation("BAAI/bge-base-en-v1.5", device="cuda")
# or start with randomized embeddings:
tokenizer = Tokenizer.from_pretrained("FacebookAI/xlm-roberta-base")
static_embedding = StaticEmbedding(tokenizer, embedding_dim=512)

model = SentenceTransformer(modules=[static_embedding])

embeddings = model.encode(["What are Pandas?", "The giant panda, also known as the panda bear or simply the panda, is a bear native to south central China."])
similarity = model.similarity(embeddings[0], embeddings[1])
# tensor([[0.8093]]) (If you use potion-base-8M)
# tensor([[0.6234]]) (If you use the distillation method)
```

tensor([[-0.0693]]) (For example, if you use randomized embeddings)

Raises:

* **ValueError** â€“ If the tokenizer is not a fast tokenizer.

* **ValueError** â€“ If neither embedding_weights nor embedding_dim is provided.

_classmethod _from_distillation(_model_name : str_, _vocabulary : list[str] | None = None_, _device : str | None = None_, _pca_dims : int | None = 256_, _apply_zipf : bool = True_, _use_subword : bool = True_) -> StaticEmbedding[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\models\\StaticEmbedding.py#L137-L189)ïƒ•

Creates a StaticEmbedding instance from a distillation process using the
model2vec package.

Parameters:

* **model_name** (_str_) â€“ The name of the model to distill.

* **vocabulary** (_list_ _[__str_ _]__|__None_ _,__optional_) â€“ A list of vocabulary words to use. Defaults to None.

* **device** (_str_) â€“ The device to run the distillation on (e.g., â€˜cpuâ€™, â€˜cudaâ€™). If not specified, the strongest device is automatically detected. Defaults to None.

* **pca_dims** (_int_ _|__None_ _,__optional_) â€“ The number of dimensions for PCA reduction. Defaults to 256.

* **apply_zipf** (_bool_) â€“ Whether to apply Zipfâ€™s law during distillation. Defaults to True.

* **use_subword** (_bool_) â€“ Whether to use subword tokenization. Defaults to True.

Returns:

An instance of StaticEmbedding initialized with the distilled modelâ€™s

tokenizer and embedding weights.

Return type:

StaticEmbedding

Raises:

ImportError â€“ If the model2vec package is not installed.

_classmethod _from_model2vec(_model_id_or_path : str_) ->

StaticEmbedding[[source]](https://github.com/UKPLab/sentence-

transformers/blob/master/sentence_transformers\\models\\StaticEmbedding.py#L191-L220)ïƒ•

Create a StaticEmbedding instance from a model2vec model. This method loads a

pre-trained model2vec model and extracts the embedding weights and tokenizer

to create a StaticEmbedding instance.

Parameters:

model_id_or_path (_str_) â€“ The identifier or path to the pre-trained

model2vec model.

Returns:

An instance of StaticEmbedding initialized with the tokenizer and embedding

weights

the model2vec model.

Return type:

StaticEmbedding

Raises:

ImportError â€“ If the model2vec package is not installed.

_class

_sentence_transformers.models.WeightedLayerPooling(_word_embedding_dimension_

, _num_hidden_layers : int = 12_, _layer_start : int = 4_, _layer_weights

=None_)[[source]](https://github.com/UKPLab/sentence-

transformers/blob/master/sentence_transformers\\models\\WeightedLayerPooling.py#L12-L70)ïƒ•

Token embeddings are weighted mean of their different hidden layer

representations

Initializes internal Module state, shared by both nn.Module and ScriptModule.

_class _sentence_transformers.models.WordEmbeddings(_tokenizer :

WordTokenizer_, _embedding_weights_ , _update_embeddings : bool = False_,

_max_seq_length : int =

1000000_)[[source]](https://github.com/UKPLab/sentence-

transformers/blob/master/sentence_transformers\\models\\WordEmbeddings.py#L22-L179)ïƒ•

Initializes internal Module state, shared by both nn.Module and ScriptModule.

_class _sentence_transformers.models.WordWeights(_vocab : list[str]_,

_word_weights : dict[str, float]_, _unknown_word_weight : float =

1_)[[source]](https://github.com/UKPLab/sentence-

transformers/blob/master/sentence_transformers\\models\\WordWeights.py#L13-L80)ïƒ•

This model can weight word embeddings, for example, with idf-values.

Initializes the WordWeights class.

Parameters:

* **vocab** (_List_ _[__str_ _]_) â€“ Vocabulary of the tokenizer.

* **word_weights** (_Dict_ _[__str_ _,__float_ _]_) â€“ Mapping of tokens to a float weight value. Word embeddings are multiplied by this float value. Tokens in word_weights must not be equal to the vocab (can contain more or less values).

* **unknown_word_weight** (_float_ _,__optional_) â€“ Weight for words in vocab that do not appear in the word_weights lookup. These can be, for example, rare words in the vocab where no weight exists. Defaults to 1.

[Previous](datasets.html "Datasets") [Next](quantization.html "quantization")

* * *