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

Getting Started

Sentence Transformer

* [Re-Ranker: Cross-Encoder](../../applications/retrieve_rerank/README.html#re-ranker-cross-encoder)

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

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

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

* [Clustering](../../applications/clustering/README.html)

* [k-Means](../../applications/clustering/README.html#k-means)

* [Agglomerative Clustering](../../applications/clustering/README.html#agglomerative-clustering)

* [Fast Clustering](../../applications/clustering/README.html#fast-clustering)

* [Topic Modeling](../../applications/clustering/README.html#topic-modeling)

* [Paraphrase Mining](../../applications/paraphrase-mining/README.html)

* [`paraphrase_mining()`](../../applications/paraphrase-mining/README.html#sentence_transformers.util.paraphrase_mining)

* [Translated Sentence Mining](../../applications/parallel-sentence-mining/README.html)

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

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

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

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

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

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

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

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

* [Scalar (int8) Quantization](../../applications/embedding-quantization/README.html#scalar-int8-quantization)
* [Additional extensions](../../applications/embedding-quantization/README.html#additional-extensions)
* [Demo](../../applications/embedding-quantization/README.html#demo)
* [Try it yourself](../../applications/embedding-quantization/README.html#try-it-yourself)
* [Speeding up Inference](../../../docs/sentence_transformer/usage/efficiency.html)
* [PyTorch](../../../docs/sentence_transformer/usage/efficiency.html#pytorch)
* [ONNX](../../../docs/sentence_transformer/usage/efficiency.html#onnx)
* [OpenVINO](../../../docs/sentence_transformer/usage/efficiency.html#openvino)
* [Benchmarks](../../../docs/sentence_transformer/usage/efficiency.html#benchmarks)
* [Creating Custom Models](../../../docs/sentence_transformer/usage/custom_models.html)
* [Structure of Sentence Transformer Models](../../../docs/sentence_transformer/usage/custom_models.html#structure-of-sentence-transformer-models)
* [Sentence Transformer Model from a Transformers Model](../../../docs/sentence_transformer/usage/custom_models.html#sentence-transformer-model-from-a-transformers-model)
* [Pretrained Models](../../../docs/sentence_transformer/pretrained_models.html)
* [Original Models](../../../docs/sentence_transformer/pretrained_models.html#original-models)
* [Semantic Search Models](../../../docs/sentence_transformer/pretrained_models.html#semantic-search-models)
* [Multi-QA Models](../../../docs/sentence_transformer/pretrained_models.html#multi-qa-models)
* [MSMARCO Passage Models](../../../docs/sentence_transformer/pretrained_models.html#msmarco-passage-models)
* [Multilingual Models](../../../docs/sentence_transformer/pretrained_models.html#multilingual-models)

* [Semantic Similarity Models](../../../docs/sentence_transformer/pretrained_models.html#semantic-similarity-models)

* [Bitext Mining](../../../docs/sentence_transformer/pretrained_models.html#bitext-mining)

* [Image & Text-Models](../../../docs/sentence_transformer/pretrained_models.html#image-text-models)

* [INSTRUCTOR models](../../../docs/sentence_transformer/pretrained_models.html#instructor-models)

* [Scientific Similarity Models](../../../docs/sentence_transformer/pretrained_models.html#scientific-similarity-models)

* [Training Overview](../../../docs/sentence_transformer/training_overview.html)

* [Why Finetune?](../../../docs/sentence_transformer/training_overview.html#why-finetune)

* [Training Components](../../../docs/sentence_transformer/training_overview.html#training-components)

* [Dataset](../../../docs/sentence_transformer/training_overview.html#dataset)

* [Dataset Format](../../../docs/sentence_transformer/training_overview.html#dataset-format)

* [Loss Function](../../../docs/sentence_transformer/training_overview.html#loss-function)

* [Training Arguments](../../../docs/sentence_transformer/training_overview.html#training-arguments)

* [Evaluator](../../../docs/sentence_transformer/training_overview.html#evaluator)

* [Trainer](../../../docs/sentence_transformer/training_overview.html#trainer)

* [Callbacks](../../../docs/sentence_transformer/training_overview.html#callbacks)

* [Multi-Dataset Training](../../../docs/sentence_transformer/training_overview.html#multi-dataset-training)

* [Deprecated Training](../../../docs/sentence_transformer/training_overview.html#deprecated-training)

* [Best Base Embedding Models](../../../docs/sentence_transformer/training_overview.html#best-base-embedding-models)

* [Dataset Overview](../../../docs/sentence_transformer/dataset_overview.html)

* [Datasets on the Hugging Face Hub](../../../docs/sentence_transformer/dataset_overview.html#datasets-on-the-hugging-face-hub)

* [Pre-existing Datasets](../../../docs/sentence_transformer/dataset_overview.html#pre-existing-datasets)

* [Loss Overview](../../../docs/sentence_transformer/loss_overview.html)

  * [Loss modifiers](../../../docs/sentence_transformer/loss_overview.html#loss-modifiers)

  * [Distillation](../../../docs/sentence_transformer/loss_overview.html#distillation)

  * [Commonly used Loss Functions](../../../docs/sentence_transformer/loss_overview.html#commonly-used-loss-functions)

  * [Custom Loss Functions](../../../docs/sentence_transformer/loss_overview.html#custom-loss-functions)

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

  * [Semantic Textual Similarity](../sts/README.html)

    * [Training data](../sts/README.html#training-data)

    * [Loss Function](../sts/README.html#loss-function)

  * [Natural Language Inference](../nli/README.html)

    * [Data](../nli/README.html#data)

    * [SoftmaxLoss](../nli/README.html#softmaxloss)

    * [MultipleNegativesRankingLoss](../nli/README.html#multiplenegativesrankingloss)

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

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

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

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

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

Cross Encoder

* [Usage](../../../docs/cross_encoder/usage/usage.html)
  * [Retrieve & Re-Rank](../../applications/retrieve_rerank/README.html)
    * [Retrieve & Re-Rank Pipeline](../../applications/retrieve_rerank/README.html#retrieve-re-rank-pipeline)
    * [Retrieval: Bi-Encoder](../../applications/retrieve_rerank/README.html#retrieval-bi-encoder)
    * [Re-Ranker: Cross-Encoder](../../applications/retrieve_rerank/README.html#re-ranker-cross-encoder)
    * [Example Scripts](../../applications/retrieve_rerank/README.html#example-scripts)
    * [Pre-trained Bi-Encoders (Retrieval)](../../applications/retrieve_rerank/README.html#pre-trained-bi-encoders-retrieval)
    * [Pre-trained Cross-Encoders (Re-Ranker)](../../applications/retrieve_rerank/README.html#pre-trained-cross-encoders-re-ranker)
* [Pretrained Models](../../../docs/cross_encoder/pretrained_models.html)
  * [MS MARCO](../../../docs/cross_encoder/pretrained_models.html#ms-marco)
  * [SQuAD (QNLI)](../../../docs/cross_encoder/pretrained_models.html#squad-qnli)
  * [STSbenchmark](../../../docs/cross_encoder/pretrained_models.html#stsbenchmark)
  * [Quora Duplicate Questions](../../../docs/cross_encoder/pretrained_models.html#quora-duplicate-questions)
  * [NLI](../../../docs/cross_encoder/pretrained_models.html#nli)
  * [Community Models](../../../docs/cross_encoder/pretrained_models.html#community-models)
* [Training Overview](../../../docs/cross_encoder/training_overview.html)
* [Training Examples](../../../docs/cross_encoder/training/examples.html)
  * [MS MARCO](../ms_marco/cross_encoder_README.html)
    * [Cross-Encoder](../ms_marco/cross_encoder_README.html#cross-encoder)
    * [Cross-Encoder Knowledge Distillation](../ms_marco/cross_encoder_README.html#cross-encoder-knowledge-distillation)

Package Reference

  * [Sentence Transformer](../../../docs/package_reference/sentence_transformer/index.html)

*

[SentenceTransformer](../../../docs/package_reference/sentence_transformer/SentenceTransformer.html)

*

[SentenceTransformer](../../../docs/package_reference/sentence_transformer/SentenceTransformer.html#id1)

*

[SentenceTransformerModelCardData](../../../docs/package_reference/sentence_transformer/SentenceTransformer.html#sentencetransformermodelcarddata)

*

[SimilarityFunction](../../../docs/package_reference/sentence_transformer/SentenceTransformer.html#similarityfunction)

    * [Trainer](../../../docs/package_reference/sentence_transformer/trainer.html)

*

[SentenceTransformerTrainer](../../../docs/package_reference/sentence_transformer/trainer.html#sentencetransformertrainer)

    * [Training Arguments](../../../docs/package_reference/sentence_transformer/training_args.html)

*

[SentenceTransformerTrainingArguments](../../../docs/package_reference/sentence_transformer/training_args.html#sentencetransformertrainingarguments)

    * [Losses](../../../docs/package_reference/sentence_transformer/losses.html)

*

[BatchAllTripletLoss](../../../docs/package_reference/sentence_transformer/losses.html#batchalltripletloss)

*

[BatchHardSoftMarginTripletLoss](../../../docs/package_reference/sentence_transformer/losses.html#batchhardsoftmargintripletloss)

*

[BatchHardTripletLoss](../../../docs/package_reference/sentence_transformer/losses.html#batchhardtripletloss)

*

[BatchSemiHardTripletLoss](../../../docs/package_reference/sentence_transformer/losses.html#batchsemihardtripletloss)

*

[ContrastiveLoss](../../../docs/package_reference/sentence_transformer/losses.html#contrastiveloss)

*

[OnlineContrastiveLoss](../../../docs/package_reference/sentence_transformer/losses.html#onlinecontrastiveloss)

*

[ContrastiveTensionLoss](../../../docs/package_reference/sentence_transformer/losses.html#contrastivetensionloss)

*

[ContrastiveTensionLossInBatchNegatives](../../../docs/package_reference/sentence_transformer/losses.html#contrastivetensionlossinbatchnegatives)

* [CoSENTLoss](../../../docs/package_reference/sentence_transformer/losses.html#cosentloss)

* [AnglELoss](../../../docs/package_reference/sentence_transformer/losses.html#angleloss)

*

[CosineSimilarityLoss](../../../docs/package_reference/sentence_transformer/losses.html#cosinesimilarityloss)

*

[DenoisingAutoEncoderLoss](../../../docs/package_reference/sentence_transformer/losses.html#den oisingautoencoderloss)

*

[GISTEmbedLoss](../../../docs/package_reference/sentence_transformer/losses.html#gistembedloss

)

*

[CachedGISTEmbedLoss](../../../docs/package_reference/sentence_transformer/losses.html#cache

dgistembedloss)

* [MSELoss](../../../docs/package_reference/sentence_transformer/losses.html#mseloss)

*

[MarginMSELoss](../../../docs/package_reference/sentence_transformer/losses.html#marginmseloss

)

*

[MatryoshkaLoss](../../../docs/package_reference/sentence_transformer/losses.html#matryoshkaloss

)

*

[Matryoshka2dLoss](../../../docs/package_reference/sentence_transformer/losses.html#matryoshka2

dloss)

*

[AdaptiveLayerLoss](../../../docs/package_reference/sentence_transformer/losses.html#adaptivelaye

rloss)

*

[MegaBatchMarginLoss](../../../docs/package_reference/sentence_transformer/losses.html#megabat

chmarginloss)

*

[MultipleNegativesRankingLoss](../../../docs/package_reference/sentence_transformer/losses.html#

multiplenegativesrankingloss)

*

*

[CachedMultipleNegativesRankingLoss](../../../docs/package_reference/sentence_transformer/losses.html#cachedmultiplenegativesrankingloss)

*

[MultipleNegativesSymmetricRankingLoss](../../../docs/package_reference/sentence_transformer/losses.html#multiplenegativessymmetricrankingloss)

*

[CachedMultipleNegativesSymmetricRankingLoss](../../../docs/package_reference/sentence_transformer/losses.html#cachedmultiplenegativessymmetricrankingloss)

    * [SoftmaxLoss](../../../docs/package_reference/sentence_transformer/losses.html#softmaxloss)

    * [TripletLoss](../../../docs/package_reference/sentence_transformer/losses.html#tripletloss)

  * [Samplers](../../../docs/package_reference/sentence_transformer/sampler.html)

*

[BatchSamplers](../../../docs/package_reference/sentence_transformer/sampler.html#batchsamplers)

*

[MultiDatasetBatchSamplers](../../../docs/package_reference/sentence_transformer/sampler.html#multidatasetbatchsamplers)

  * [Evaluation](../../../docs/package_reference/sentence_transformer/evaluation.html)

*

[BinaryClassificationEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#binaryclassificationevaluator)

*

[EmbeddingSimilarityEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#embeddingsimilarityevaluator)

*

[InformationRetrievalEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#informationretrievalevaluator)

*

[NanoBEIREvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#nanobeirevaluator)

*

[MSEEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#mseevaluator)

*

[ParaphraseMiningEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#paraphraseminingevaluator)

*

[RerankingEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#rerankingevaluator)

*

[SentenceEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#sentenceevaluator)

*

[SequentialEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#sequentialevaluator)

*

[TranslationEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#translationevaluator)

*

[TripletEvaluator](../../../docs/package_reference/sentence_transformer/evaluation.html#tripletevaluator)

* [Datasets](../../../docs/package_reference/sentence_transformer/datasets.html)

*

[ParallelSentencesDataset](../../../docs/package_reference/sentence_transformer/datasets.html#par allelsentencesdataset)

*

[SentenceLabelDataset](../../../docs/package_reference/sentence_transformer/datasets.html#sentencelabeldataset)

*

[DenoisingAutoEncoderDataset](../../../docs/package_reference/sentence_transformer/datasets.html#denoisingautoencoderdataset)

*

[NoDuplicatesDataLoader](../../../docs/package_reference/sentence_transformer/datasets.html#noduplicatesdataloader)

* [Models](../../../docs/package_reference/sentence_transformer/models.html)

* [Main Classes](../../../docs/package_reference/sentence_transformer/models.html#main-classes)

* [Further Classes](../../../docs/package_reference/sentence_transformer/models.html#further-classes)

* [quantization](../../../docs/package_reference/sentence_transformer/quantization.html)

*

[`quantize_embeddings()`](../../../docs/package_reference/sentence_transformer/quantization.html#sentence_transformers.quantization.quantize_embeddings)

*

[`semantic_search_faiss()`](../../../docs/package_reference/sentence_transformer/quantization.html#sentence_transformers.quantization.semantic_search_faiss)

*

[`semantic_search_usearch()`](../../../docs/package_reference/sentence_transformer/quantization.html#sentence_transformers.quantization.semantic_search_usearch)

* [Cross Encoder](../../../docs/package_reference/cross_encoder/index.html)

* [CrossEncoder](../../../docs/package_reference/cross_encoder/cross_encoder.html)

* [CrossEncoder](../../../docs/package_reference/cross_encoder/cross_encoder.html#id1)

* [Training Inputs](../../../docs/package_reference/cross_encoder/cross_encoder.html#training-inputs)

* [Evaluation](../../../docs/package_reference/cross_encoder/evaluation.html)

* [CEBinaryAccuracyEvaluator](../../../docs/package_reference/cross_encoder/evaluation.html#cebinaryaccuracyevaluator)

* [CEBinaryClassificationEvaluator](../../../docs/package_reference/cross_encoder/evaluation.html#cebinaryclassificationevaluator)

* [CECorrelationEvaluator](../../../docs/package_reference/cross_encoder/evaluation.html#cecorrelationevaluator)

* [CEF1Evaluator](../../../docs/package_reference/cross_encoder/evaluation.html#cef1evaluator)

* [CESoftmaxAccuracyEvaluator](../../../docs/package_reference/cross_encoder/evaluation.html#cesoftmaxaccuracyevaluator)

* [CERerankingEvaluator](../../../docs/package_reference/cross_encoder/evaluation.html#cererankingevaluator)

* [util](../../../docs/package_reference/util.html)

* [Helper Functions](../../../docs/package_reference/util.html#module-sentence_transformers.util)

* [`community_detection()`](../../../docs/package_reference/util.html#sentence_transformers.util.community_detection)

* [`http_get()`](../../../docs/package_reference/util.html#sentence_transformers.util.http_get)

*

[`is_training_available()`](../../../docs/package_reference/util.html#sentence_transformers.util.is_training_available)

*

[`mine_hard_negatives()`](../../../docs/package_reference/util.html#sentence_transformers.util.mine_hard_negatives)

*

[`normalize_embeddings()`](../../../docs/package_reference/util.html#sentence_transformers.util.normalize_embeddings)

*

[`paraphrase_mining()`](../../../docs/package_reference/util.html#sentence_transformers.util.paraphrase_mining)

*

[`semantic_search()`](../../../docs/package_reference/util.html#sentence_transformers.util.semantic_search)

*

[`truncate_embeddings()`](../../../docs/package_reference/util.html#sentence_transformers.util.truncate_embeddings)

* [Model Optimization](../../../docs/package_reference/util.html#module-sentence_transformers.backend)

*

[`export_dynamic_quantized_onnx_model()`](../../../docs/package_reference/util.html#sentence_transformers.backend.export_dynamic_quantized_onnx_model)

*

[`export_optimized_onnx_model()`](../../../docs/package_reference/util.html#sentence_transformers.backend.export_optimized_onnx_model)

*

[`export_static_quantized_openvino_model()`](../../../docs/package_reference/util.html#sentence_tra nsformers.backend.export_static_quantized_openvino_model)

* [Similarity Metrics](../../../docs/package_reference/util.html#module-sentence_transformers.util)

* [`cos_sim()`](../../../docs/package_reference/util.html#sentence_transformers.util.cos_sim)

* [`dot_score()`](../../../docs/package_reference/util.html#sentence_transformers.util.dot_score)

*

[`euclidean_sim()`](../../../docs/package_reference/util.html#sentence_transformers.util.euclidean_si

m)

*

[`manhattan_sim()`](../../../docs/package_reference/util.html#sentence_transformers.util.manhattan_

sim)

*

[`pairwise_cos_sim()`](../../../docs/package_reference/util.html#sentence_transformers.util.pairwise_

cos_sim)

*

[`pairwise_dot_score()`](../../../docs/package_reference/util.html#sentence_transformers.util.pairwise

_dot_score)

*

[`pairwise_euclidean_sim()`](../../../docs/package_reference/util.html#sentence_transformers.util.pair

wise_euclidean_sim)

*

[`pairwise_manhattan_sim()`](../../../docs/package_reference/util.html#sentence_transformers.util.pai

rwise_manhattan_sim)

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

* [](../../../index.html)

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

* Adaptive Layers

* [Edit on GitHub](https://github.com/UKPLab/sentence-transformers/blob/master/examples/training/adaptive_layer/README.md)

* * *

Adaptive Layersïƒ•

Embedding models are often encoder models with numerous layers, such as 12 (e.g. [all-mpnet-base-v2](https://huggingface.co/sentence-transformers/all-mpnet-base-v2)) or 6 (e.g. [all-MiniLM-L6-v2](https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2)). To get embeddings, every single one of these layers must be traversed. The [2D Matryoshka Sentence Embeddings](https://arxiv.org/abs/2402.14776v1) (2DMSE) preprint revisits this concept by proposing an approach to train embedding models that will perform well when only using a selection of all layers. This results in faster inference speeds at relatively low performance costs.

Note

The 2DMSE preprint was later updated and renamed to [ESE: Espresso Sentence Embeddings](https://arxiv.org/abs/2402.14776). The Sentence Transformers implementation of Adaptive Layers and Matryoshka2d (Adaptive Layer + Matryoshka Embeddings) are based on the initial preprint, and we accept contributions that implement the updated ESE paper.

Use Casesïƒ•

The 2DMSE paper mentions that using a few layers of a larger model trained using Adaptive Layers and Matryoshka Representation Learning can outperform a smaller model that was trained like a standard embedding model.

Resultsïƒ•

Letâ€™s look at the performance that we may be able to expect from an Adaptive Layer embedding model versus a regular embedding model. For this experiment, I have trained two models:

* [tomaarsen/mpnet-base-nli-adaptive-layer](https://huggingface.co/tomaarsen/mpnet-base-nli-adaptive-layer): Trained by running [adaptive_layer_nli.py](https://github.com/UKPLab/sentence-transformers/tree/master/examples/training/adaptive_layer/adaptive_layer_nli.py) with [microsoft/mpnet-base](https://huggingface.co/microsoft/mpnet-base).

* [tomaarsen/mpnet-base-nli](https://huggingface.co/tomaarsen/mpnet-base-nli): A near identical model as the former, but using only `MultipleNegativesRankingLoss` rather than `AdaptiveLayerLoss` on top of `MultipleNegativesRankingLoss`. I also use [microsoft/mpnet-base](https://huggingface.co/microsoft/mpnet-base) as the base model.

Both of these models were trained on the AllNLI dataset, which is a concatenation of the [SNLI](https://huggingface.co/datasets/snli) and [MultiNLI](https://huggingface.co/datasets/multi_nli) datasets. I have

evaluated these models on the
[STSBenchmark](https://huggingface.co/datasets/mteb/stsbenchmark-sts) test set
using multiple different embedding dimensions. The results are plotted in the
following figure:

`![adaptive_layer_results](https://huggingface.co/tomaarsen/mpnet-base-nli-adaptive-layer/resolve/main/adaptive_layer_results.png)`

The first figure shows that the Adaptive Layer model stays much more performant when reducing the number of layers in the model. This is also clearly shown in the second figure, which displays that 80% of the performance is preserved when the number of layers is reduced all the way to 1.

Lastly, the third figure shows the expected speedup ratio for GPU & CPU devices in my tests. As you can see, removing half of the layers results in roughly a 2x speedup, at a cost of ~15% performance on STSB (~86 -> ~75 Spearman correlation). When removing even more layers, the performance benefit gets larger for CPUs, and between 5x and 10x speedups are very feasible with a 20% loss in performance.

Trainingïƒ•

Training with Adaptive Layer support is quite elementary: rather than applying some loss function on only the last layer, we also apply that same loss function on the pooled embeddings from previous layers. Additionally, we employ a KL-divergence loss that aims to make the embeddings of the non-last layers match that of the last layer. This can be seen as a fascinating approach of [knowledge distillation](../distillation/README.html#knowledge-distillation), but with the last layer as the teacher model and the prior layers as the student models.

For example, with the 12-layer [microsoft/mpnet-base](https://huggingface.co/microsoft/mpnet-base), it will now be trained such that the model produces meaningful embeddings after each of the 12 layers.

```
from sentence_transformers import SentenceTransformer
from sentence_transformers.losses import CoSENTLoss, AdaptiveLayerLoss

model = SentenceTransformer("microsoft/mpnet-base")

base_loss = CoSENTLoss(model=model)
loss = AdaptiveLayerLoss(model=model, loss=base_loss)
```

* **Reference** : [`AdaptiveLayerLoss`](../../../docs/package_reference/sentence_transformer/losses.html#adaptivelayerloss)

Note that training with `AdaptiveLayerLoss` is not notably slower than without using it.

Additionally, this can be combined with the `MatryoshkaLoss` such that the resulting model can be reduced both in the number of layers, but also in the size of the output dimensions. See also the [Matryoshka Embeddings](../matryoshka/README.html) for more information on reducing output dimensions. In Sentence Transformers, the combination of these two losses is called `Matryoshka2dLoss`, and a shorthand is provided for simpler training.

```
from sentence_transformers import SentenceTransformer
from sentence_transformers.losses import CoSENTLoss, Matryoshka2dLoss

model = SentenceTransformer("microsoft/mpnet-base")

base_loss = CoSENTLoss(model=model)
loss = Matryoshka2dLoss(model=model, loss=base_loss, matryoshka_dims=[768, 512, 256, 128, 64])
```

* **Reference**: [`Matryoshka2dLoss`](../../../docs/package_reference/sentence_transformer/losses.html#matryoshka2dloss)

Inferenceïƒ•

After a model has been trained using the Adaptive Layer loss, you can then truncate the model layers to your desired layer count. Note that this requires doing a bit of surgery on the model itself, and each model is structured a bit differently, so the steps are slightly different depending on the model.

First of all, we will load the model & access the underlying `transformers` model like so:

```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("tomaarsen/mpnet-base-nli-adaptive-layer")

# We can access the underlying model with `model[0].auto_model`
print(model[0].auto_model)
```

```
MPNetModel(
  (embeddings): MPNetEmbeddings(
    (word_embeddings): Embedding(30527, 768, padding_idx=1)
    (position_embeddings): Embedding(514, 768, padding_idx=1)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): MPNetEncoder(
    (layer): ModuleList(
      (0-11): 12 x MPNetLayer(
```

```
      (attention): MPNetAttention(
        (attn): MPNetSelfAttention(
          (q): Linear(in_features=768, out_features=768, bias=True)
          (k): Linear(in_features=768, out_features=768, bias=True)
          (v): Linear(in_features=768, out_features=768, bias=True)
          (o): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (intermediate): MPNetIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
        (intermediate_act_fn): GELUActivation()
      )
      (output): MPNetOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (relative_attention_bias): Embedding(32, 12)
)
(pooler): MPNetPooler(
  (dense): Linear(in_features=768, out_features=768, bias=True)
  (activation): Tanh()
```

```
    )
)
```

This output will differ depending on the model. We will look for the repeated layers in the encoder. For this MPNet model, this is stored under `model[0].auto_model.encoder.layer`. Then we can slice the model to only keep the first few layers to speed up the model:

```
new_num_layers = 3
model[0].auto_model.encoder.layer = model[0].auto_model.encoder.layer[:new_num_layers]
```

Then we can run inference with it using [`SentenceTransformers.encode`](../../../docs/package_reference/sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer.encode).

```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("tomaarsen/mpnet-base-nli-adaptive-layer")
new_num_layers = 3
model[0].auto_model.encoder.layer = model[0].auto_model.encoder.layer[:new_num_layers]
```

```
embeddings = model.encode(
    [
        "The weather is so nice!",
        "It's so sunny outside!",
        "He drove to the stadium.",
    ]
)
# Similarity of the first sentence with the other two
similarities = model.similarity(embeddings[0], embeddings[1:])
# => tensor([[0.7761, 0.1655]])
# compared to tensor([[ 0.7547, -0.0162]]) for the full model
```

As you can see, the similarity between the related sentences is much higher than the unrelated sentence, despite only using 3 layers. Feel free to copy this script locally, modify the `new_num_layers`, and observe the difference in similarities.

Code Examplesïƒ•

See the following scripts as examples of how to apply the [`AdaptiveLayerLoss`](../../../docs/package_reference/sentence_transformer/losses.html#adaptivelayerloss)
in practice:

*

[adaptive_layer_nli.py](https://github.com/UKPLab/sentence-transformers/tree/master/examples/tr aining/adaptive_layer/adaptive_layer_nli.py) : This example uses the `MultipleNegativesRankingLoss` with `AdaptiveLayerLoss` to train a strong embedding model using Natural Language Inference (NLI) data. It is an adaptation of the [NLI](../nli/README.html) documentation.

* **[adaptive_layer_sts.py](https://github.com/UKPLab/sentence-transformers/tree/master/examples/training/adaptive_layer/adaptive_layer_sts.py)** : This example uses the CoSENTLoss with AdaptiveLayerLoss to train an embedding model on the training set of the STSBenchmark dataset. It is an adaptation of the [STS](../sts/README.html) documentation.

And the following scripts to see how to apply [`Matryoshka2dLoss`](../../../docs/package_reference/sentence_transformer/losses.html#matryoshka2dloss):

* **[2d_matryoshka_nli.py](https://github.com/UKPLab/sentence-transformers/tree/master/examples/training/adaptive_layer/../matryoshka/2d_matryoshka_nli.py)** : This example uses the `MultipleNegativesRankingLoss` with `Matryoshka2dLoss` to train a strong embedding model using Natural Language Inference (NLI) data. It is an adaptation of the [NLI](../nli/README.html) documentation.

* **[2d_matryoshka_sts.py](https://github.com/UKPLab/sentence-transformers/tree/master/examples/training/adaptive_layer/../matryoshka/2d_matryoshka_sts.py)** : This example uses the `CoSENTLoss` with `Matryoshka2dLoss` to train an embedding model on the training set of the STSBenchmark dataset. It is an adaptation of the [STS](../sts/README.html) documentation.

* * *