arXiv:2412.19048v1 [cs.IR] 26 Dec 2024

Jasper and Stella: distillation of SOTA embedding models

Dun Zhang[1], **Fulong Wang**[2]
[1]Independent Researcher [2]Independent Researcher
[1]infgrad@163.com [2]wangfl1989@163.com

Abstract

A crucial component of many deep learning applications (such as FAQ and RAG) is dense retrieval, in which embedding models are used to convert raw text to numerical vectors and then get the most similar text by MIPS (Maximum Inner Product Search). Some text embedding benchmarks (e.g. MTEB (Muennighoff et al., 2022), BEIR (Thakur et al., 2021), and AIR-Bench (Chen et al., 2024)) have been established to evaluate embedding models accurately. Thanks to these benchmarks, we can use SOTA models; however, the deployment and application of these models in industry were hampered by their large vector dimensions and numerous parameters. To alleviate this problem, 1) we present a distillation technique that can enable a smaller student model to achieve good performance. 2) Inspired by MRL(Kusupati et al., 2024) we present a training approach of reducing the vector dimensions based on its own vectors or its teacher vectors. 3) We do simple yet effective alignment training between images and text to make our model a multimodal encoder. We trained Stella and Jasper models using the technologies above and achieved high scores on the MTEB leaderboard. We release the model and data at Hugging Face Hub[1] [2], the training codes will be in this project [3] and the training logs are at Weights & Biases[4].

1 Introduction

With the rapid development of natural language processing technologies, text embedding models play a crucial role in text representation, information retrieval, and generation tasks. By mapping words, sentences, or documents into a high-dimensional continuous space, these models enable similar texts to have closer vector representations, thus not only enhancing the manipulability of textual data but also significantly improving the performance of various downstream tasks. Especially in retrieval-enhanced generation (RAG) techniques, the ability of the embedding model directly affects the quality of the generated results.

Retrieval systems that index passages by embedded models can efficiently deploy and utilize them to achieve fast retrieval of relevant passages, as shown by Maximum Inner Product Search (MIPS). The growing interest in text embedding models in academia and industry has led to the recent release of many new models, such as E5 (Wang et al., 2022)(Wang et al., 2023)(Wang et al., 2024), GTE (Li et al., 2023), and Jina (Günther et al., 2023). To provide a consistent method for comparing the accuracy of existing text embedding models several benchmarks have been created, and public leaderboards have been made available for different tasks on the HuggingFace platform, such as MTEB (Muennighoff et al., 2022), BEIR (Thakur et al., 2021), and AIR-Bench (Chen et al., 2024). These efforts have further advanced the use of text embedding models in Natural Language Processing (NLP) tasks, contributing to the rapid development of a wide range of applications.

However, models that have excellent performance usually contain a large number of parameters and high dimensions; for example, NV-Embed-v2 (Lee et al., 2024) (Moreira et al., 2024), bge-en-icl (Xiao et al., 2023)(Li et al., 2024), and e5-mistral-7b-instruct (Wang et al., 2022)(Wang et al., 2023)(Wang et al., 2024) have 7B parameters, and their vectors are 4096d. These features lead to slow inference and retrieval speeds. Actually, this is a trade-off between accuracy and speed.

In this paper, we use knowledge distillation (Hinton et al., 2015) to let a student model learn a teacher model's vectors by several well-designed

[1] https://huggingface.co/infgrad/jasper_en_vision_language_v1

[2] https://huggingface.co/datasets/infgrad/jasper_text_distill_dataset

[3] https://github.com/NLPJCL/RAG-Retrieval

[4] https://api.wandb.ai/links/dunnzhang0/z8jqoqpb

losses. In order to further improve the student model's performance, we use serval excellent embedding models as teacher models (i.e. concatenate multiple teacher vectors at dimension -1). This method does not need any supervised data, and the MTEB leaderboard results and in-house test dataset results show that our student models are far better than those in the same parameter. Because of the distillation of serval teacher models, our student models have large vector dimensions (i.e., the summary of teacher models's vector dimensions). To solve this problem, we introduce a training method that can reduce the model's dimensions. Just like distillation, this method also does not need any supervised data. This method does not even need the teacher's vectors, just doing a self-distillation can still achieve a good performance.

We also train a siglip model to align its visual embedding with the text encoder's token embeddings; as a result, our model can both encode images and text.

2 Methods

In this section, we describe our model architecture designs and four-stage training method.

2.1 Definitions

To be able to better introduce our model and training method, we make the following definitions:

- **student model**: the text embedding model to be trained
- **teacher model**: the SOTA embedding model used to teach student models to generate its vectors; this model will not be trained
- **vectors**: also called text embedding or text representation, is a pooling of last hidden states. Its shape is (batch_size, vector_dimensions)
- **student vectors**: student model's vectors
- **teacher vectors**: teacher model's vectors
- **sv**: abbreviation for student vectors
- **tv**: abbreviation for teacher vectors
- **mse**: abbreviation for Mean Square Error
- **matmul**: matrix product of two tensors
- **vectors.T**: View of the transposed vectors

2.2 Model Architecture

Our model architecture follows the simple and standard design of combining a language model with a vision encoder. As shown in Figure 1, it consists of four components:

1. A ViT image encoder that independently maps images into vision token embedding;
2. A pool that projects the vision tokens to the language model's input dimension and reduces their count;
3. A transform encoder or decoder (e.g. BERT, GPT2, XLMRoberta, generally called a text embedding model);
4. Several fully connected layers (FC) that project the vectors to the specific dimension;

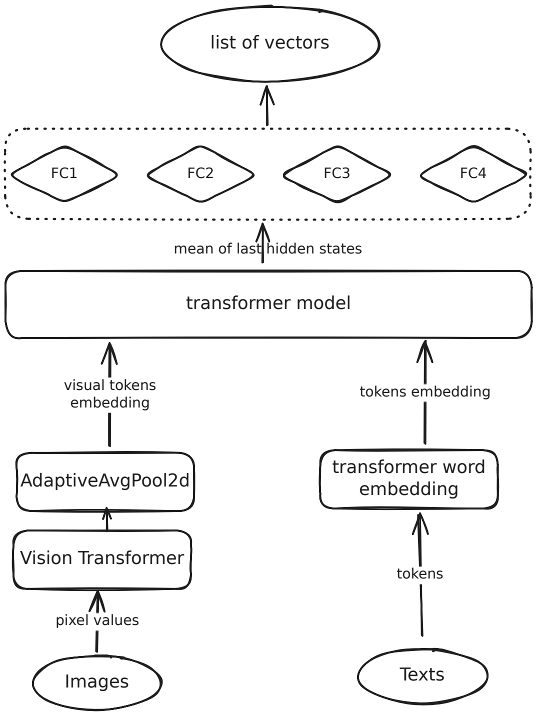

Figure 1: The model architecture of Jasper model

2.3 Stage1&2: Distillation from Teacher Vectors

The two stages's target is to let student model to learn serval teacher models's vectors. To achieve this goal, we designed three loss functions.

The first loss function is cosine loss which is formulated as follows:

$$\text{cosine\_loss}(\text{sv}, \text{tv}) = 1 - \frac{\text{sv} \cdot \text{tv}}{|\text{sv}||\text{tv}|} \tag{1}$$

The cosine_loss was designed with the simple intention of making the angle between the student vectors and teacher vectors in the higher-dimensional space as small as possible.

The second loss function is a supplement for cosine_loss. Apparently, the cosine loss value will not go down to zero, so there will always be an angle between student vectors and teacher vectors, we accept this angle but hope the similarity values for any text pair computed by student vectors and teacher vectors are the same. We call this loss function similarity_loss which is formulated as follows:

$$\text{similarity\_loss} = mse(matmul(sv, sv.T), matmul(tv, tv.T)) \tag{2}$$

The third loss function is a supplement for cosine_loss and similarity_loss, it further reduces the requirement for student vectors. This loss function is a standard triplet_loss which is formulated as follows:

$$\text{triplet\_loss} = \frac{1}{N} \sum_{\text{a, p, n} \in \text{triplet\_corpus}} \max(0, \text{score(a, p)} - \text{s(a, n)} + \text{margin}) \tag{3}$$

In this equation, a, p and n separately represent anchor text, positive text and negative text; $score(a, p)$ represents the cosine similarity between anchor text and positive text computed by student vectors. We use teacher vector to compute cosine similarity and then get anchor text, positive text and negative text according to the size of the cosine similarity. We generate triplet_corpus from a batch data, if the batch size is bsz, then the number of triplet_corpus (i.e. N) is:

$$\text{N} = C^2_{C^2_{bsz}} \tag{4}$$

Note that, N is very large and proportional to the fourth power of bsz, so if bsz is too large, you may encounter OOM errors. In a short word, we use teacher vectors to generate many triplet data and then train student model by using triplet loss.

The final loss is a weighted sum of those three losses. Algorithm 1 provides the pseudo-code of this distillation training and loss's weight. The biggest advantage of distillation vectors is that we do not need any supervised data. No consideration of resource constraints, we can use trillions of unsupervised texts to do distillation training to get extreme performance for a given model size.

Algorithm 1 Pseudocode of Stage1&2 in a PyTorch-like style.

```
def get_score_diff(vectors):
   scores = torch.matmul(vectors, vectors.T)
   scores = scores[torch.triu(torch.ones_like(
        scores), diagonal=1).bool()]

   score_diff = scores.reshape((1, -1)) - scores.
        reshape((-1, 1))
   score_diff = score_diff[torch.triu(torch.
        ones_like(score_diff), diagonal=1).bool()]

   return score_diff

# start train
for batch in active_dataloader:
   # teacher_vectors = torch.cat( stella_vectors,
        nv_v2_vectors )
   teacher_vectors = batch.pop("teacher_vectors")

   model_output = model(**batch)

   # in stage1 and stage2, we only train the fc
        that convert the last hidden states to
        12288d
   ## this fc is for learning teacher vectors, and
         other fc is for reducing dimensions
   student_vectors = model_output["vectors_12288"].
        float()
   student_vectors = F.normalize(student_vectors,
        p=2, dim=-1)

   # cosine loss, cosine_loss_scale is 10.0 in our
         codes
   cosine_loss = (1 - (student_vectors *
        teacher_vectors).sum(axis=1).mean()) *
        cosine_loss_scale

   # similarity loss, similarity_loss_scale is
        200.0 in our codes
   similarity_loss = F.mse_loss(
      input=torch.matmul(student_vectors,
          student_vectors.T),
      target=torch.matmul(teacher_vectors,
          teacher_vectors.T),
   ) * similarity_loss_scale

   # triplet_loss, triplet_loss_scale is 20.0 and
        triplet_margin is 0.015 in our codes
   triplet_label = torch.where(get_score_diff(
        teacher_vectors) < 0, 1, -1)
   triplet_loss = F.relu(get_score_diff(
        student_vectors) * triplet_label +
        triplet_margin).mean() * triplet_loss_scale

   loss = cosine_loss + similarity_loss +
        triplet_loss
   # do backward, step and zero_grad...
```

The difference between stage1 and stage2 is the trained parameters. The stage1 only train the fully connected layer and the stage2 only train the fully connected layer and last three layers of student model.

2.4 Stage3: Dimension Reduction

In the distillation of stage1 and stage2, we use a fully connected layer to project the student vectors to the teacher vectors' dimension. Specifically, in Jasper model, we use *stella_en_1.5B_v5* and *NV-Embed-v2* (Lee et al., 2024) as teacher model, their dimensions is 4096 and 8192, so the student vector dimensions is 12288 (4096+8192) which is too large. According to the Johnson–Lindenstrauss lemma (Ghojogh et al., 2021), we can make the dimensions lower without compromising performance. Our method is simple, we add several fully connected layers to reduce student vector dimension. For example, if we add a fully connected layer with a shape of (hidden_size, 512), we can get vectors of 512d.

In stage3, we use *similarity_loss* and *triplet_loss* as loss functions. The reduced vector dimensions are not the same with teacher vectors, so we skip the *cosine_loss*. In this stage, we train all parameters. Note that, to ensure the accuracy of the last 2 stage vectors (i.e., vectors of 12288d), this vector still be trained with all three loss functions.

Besides the above approach of dimension reduction, we can also consider vectors of the student model as teacher vectors (a sort of self-distillation). By using this approach, we might get a little bit of performance degradation, but we can reduce the dimensionality of any embedding model just by using unsupervised data and itself. Because this paper mainly introduces the training method of the Stella and Jasper model, so we didn't do experiments to argue the merits of the method.

2.5 Stage4: Visualized Jasper Model

In this stage, we use image caption data as training data and only train the visual encoder. The essence of this training is still distillation: the vector of caption is teacher vector, and the vector of image is student vector. The loss function is the same with stage1 (i.e. *cosine_loss*, *similarity_loss* and *triplet_loss*).

As in stage3, we have multiple fully connected layers, we have multiple student vectors. During training, we compute the loss for each student vector and teacher vector, and then the average of multiple losses is the final loss. The pseudo-code can be found in Algorithm 2. We think there is a lot of room for improvement in this approach, which will be explained in the Discussion section.

Algorithm 2 Pseudocode of Stage4 in a PyTorch-like style.

```
def get_score_diff(vectors):
    scores = torch.matmul(vectors, vectors.T)
    scores = scores[torch.triu(torch.ones_like(
         scores), diagonal=1).bool()]

    score_diff = scores.reshape((1, -1)) - scores.
         reshape((-1, 1))
    score_diff = score_diff[torch.triu(torch.
         ones_like(score_diff), diagonal=1).bool()]

    return score_diff

# start train
for batch in active_dataloader:

    # get teacher_vectors
    with torch.no_grad():
        teacher_vectors = model.encode(batch.pop("
             image_captions"))

    # get all_student_vectors
    all_student_vectors = model(**batch)["
         all_vectors"]

    # for each vector we compute loss
    loss = 0.0
    for student_vectors in all_student_vectors:

        # cosine loss, cosine_loss_scale is 10.0 in
             our codes
        cosine_loss = (1 - (student_vectors *
             teacher_vectors).sum(axis=1).mean()) *
             cosine_loss_scale

        # similarity loss, similarity_loss_scale is
             200.0 in our codes
        similarity_loss = F.mse_loss(
            input=torch.matmul(student_vectors,
                student_vectors.T),
            target=torch.matmul(teacher_vectors,
                teacher_vectors.T),
        ) * similarity_loss_scale

        # triplet_loss, triplet_loss_scale is 20.0
             and triplet_margin is 0.015 in our codes
        triplet_label = torch.where(get_score_diff(
             teacher_vectors) < 0, 1, -1)
        triplet_loss = F.relu(get_score_diff(
             student_vectors) * triplet_label +
             triplet_margin).mean() *
             triplet_loss_scale

        loss += (cosine_loss + similarity_loss +
             triplet_loss)
    loss /= len(all_student_vectors)
    # do backward, step and zero_grad...
```

3 Experiments

3.1 Implementation details

Our model (*jasper_en_vision_language_v1*) is initialized from *stella_en_1.5B_v5* and *google/siglip-so400m-patch14-384*(Alabdulmohsin et al., 2024)(Zhai et al., 2023). *stella_en_1.5B_v5* and *NV-Embed-v2* is our teacher models. The total number of parameters in the model is 1.9B (stella 1.5B parameters and siglip 400M parameters).

In all four stages, the model is trained with a maximum input length of 512 tokens, mixed precision training (BF16), DeepSpeed ZERO-stage-2 and AdamW optimizer.

In stage1 training (distillation training), the batch size is 128, the learning rate is 1e-04, trained with 8 RTX A6000, the checkpoint of step-4000 is the final model.

In stage2 training (distillation training), the batch size is 128, the learning rate is 8e-05, trained with 8 RTX A6000, the checkpoint of step-7000 is the final model.

In stage3 training (dimension reduction training), the batch size is 128, the learning rate is 7e-05, trained with 8 RTX A6000, the checkpoint of step-2200 is the final model.

In stage4 training (multimodal training), the batch size is 90, the learning rate is 1e-04, trained with 8 RTX A6000, the checkpoint of step-3500 is the final model.

3.2 Datasets

In stage1, stage2 and stage3, we use *fineweb-edu*(Lozhkov et al., 2024) as our main training dataset, which accounts for 80% of the full data. The remaining 20% of the data comes from *sentence-transformers/embedding-training-data*[5]. The reason we choose *sentence-transformers/embedding-training-data* is that most fineweb-edu data is about passage; except for passage, we also need questions to enhance data diversity.

For the documents of data, we also do the following actions:

1. We randomly select 30% of the documents and cut them into short text (consisting of 1-10 sentences).
2. We randomly select 0.08% of the text and shuffle their words.

The total amount of data is 8 million.

In stage4, we use the caption data of *BAAI/Infinity-MM*(Gu et al., 2024) as our training data.

3.3 Results

We evaluate our model on the MTEB leaderboard and get an average score of 72.02 which is best in the models that have fewer than 2B parameters. The detailed results can be found in Table 1.

4 Discussion

In this section, we will discuss some interesting findings and possible improvements that have not yet been validated.

4.1 Robustness of instruction-based embedding models

The instruction-based embedding models mean that you should add an instruction to query or passage when encoding texts. Jasper model is also instruction-based. Currently, many excellent text embedding models use an instruction to prompt the model to get better embeddings. Just like usage of LLM, different tasks or scenarios use different instructions; this is both logical and intuitive. Hence, the ability to understand instructions is crucial to these text embedding models. In this subsection, we do a simple experiment on Jasper model to show the impact of different prompts. Specifically, we do a MTEB evaluation on some short evaluation-time tasks using similar instructions (generated by GPT-4o). Table 2 shows all original and modified instructions, and Table 3 shows the evaluation result. We think this evaluation result shows that Jasper model is robust and can correctly understand different instructions.

[5] https://huggingface.co/datasets/sentence-transformers/embedding-training-data

Model	Average(56 datasets)	Classification	Clustering	PairClassification	Reranking	Retrieval	STS	Summarization
NV-Embed-v2	72.31	90.37	58.46	88.67	60.65	62.65	84.31	30.7
jasper(our model)	72.02	88.49	58.04	88.07	60.91	63.12	84.67	31.42
bge-en-icl	71.67	88.95	57.89	88.14	59.86	62.16	84.24	30.77
stella_en_1.5B_v5	71.19	87.63	57.69	88.07	61.21	61.01	84.51	31.49

Table 1: MTEB Results

original instruction	synonym of original instruction
Classify the sentiment expressed in the given movie review text from the IMDB dataset	Determine the sentiment conveyed in the provided movie review text from the IMDB dataset.
Identify the topic or theme of StackExchange posts based on the titles	Determine the subject or theme of StackExchange posts based on the titles.
Given a news summary, retrieve other semantically similar summaries	Given a news summary, find other summaries with similar meanings.
Retrieve duplicate questions from StackOverflow forum	Find duplicate questions on the StackOverflow forum.
Given a title of a scientific paper, retrieve the titles of other relevant papers	Given the title of a scientific paper, find the titles of other related papers.
Classify the sentiment of a given tweet as either positive, negative, or neutral	Determine the sentiment of a given tweet as positive, negative, or neutral.
Given a claim, find documents that refute the claim	Given a claim, locate documents that contradict the claim.
Given a question, retrieve relevant documents that best answer the question	Given a question, find relevant documents that best answer it.
Retrieve tweets that are semantically similar to the given tweet	Find tweets that have similar meanings to the given tweet.
Retrieve semantically similar text.	Find text with similar meanings.
Identify the main category of Medrxiv papers based on the titles	Determine the primary category of Medrxiv papers based on the titles.
Retrieve duplicate questions from AskUbuntu forum	Find duplicate questions on the AskUbuntu forum.
Given a question, retrieve detailed question descriptions from Stackexchange that are duplicates to the given question	Given a question, find detailed question descriptions from Stackexchange that are duplicates.
Identify the main category of Biorxiv papers based on the titles and abstracts	Determine the primary category of Biorxiv papers based on the titles and abstracts.
Given a financial question, retrieve user replies that best answer the question	Given a financial question, find user replies that best answer it.
Given a online banking query, find the corresponding intents	Given an online banking query, identify the corresponding intents.
Identify the topic or theme of the given news articles	Determine the subject or theme of the given news articles.
Classify the emotion expressed in the given Twitter message into one of the six emotions: anger, fear, joy, love, sadness, and surprise	Determine the emotion expressed in the given Twitter message as one of six emotions: anger, fear, joy, love, sadness, and surprise.
Given a user utterance as query, find the user intents	Given a user utterance as a query, identify the user intents.
Identify the main category of Biorxiv papers based on the titles	Determine the primary category of Biorxiv papers based on the titles.
Classify the given Amazon review into its appropriate rating category	Classify the given Amazon review into its appropriate rating category.
Given a scientific claim, retrieve documents that support or refute the claim	Given a scientific claim, find documents that support or contradict the claim.
Identify the topic or theme of StackExchange posts based on the given paragraphs	Determine the subject or theme of StackExchange posts based on the given paragraphs.
Given a scientific paper title, retrieve paper abstracts that are cited by the given paper	Given a scientific paper title, find paper abstracts that are cited by the given paper.
Classify the given comments as either toxic or not toxic	Classify the given comments as toxic or non-toxic.
Classify the intent domain of the given utterance in task-oriented conversation	Determine the intent domain of the given utterance in task-oriented conversation.
Retrieve duplicate questions from Sprint forum	Find duplicate questions on the Sprint forum.
Given a user utterance as query, find the user scenarios	Given a user utterance as a query, identify the user scenarios.
Classify the intent of the given utterance in task-oriented conversation	Determine the intent of the given utterance in task-oriented conversation.
Classify a given Amazon customer review text as either counterfactual or not-counterfactual	Classify a given Amazon customer review text as counterfactual or non-counterfactual.
Identify the main category of Medrxiv papers based on the titles and abstracts	Determine the primary category of Medrxiv papers based on the titles and abstracts.
Given a query on COVID-19, retrieve documents that answer the query	Given a query on COVID-19, find documents that answer the query.

Table 2: Original instructions and their synonyms

TaskType	**TaskName**	**original_score**	**score_with_modified_instructions**
Classification	**MTOPDomainClassification**	0.992	0.992
Classification	**AmazonCounterfactualClassification**	0.958	0.957
Classification	**TweetSentimentExtractionClassification**	0.773	0.776
Classification	**EmotionClassification**	0.877	0.859
Classification	**MassiveIntentClassification**	0.853	0.854
Classification	**AmazonReviewsClassification**	0.629	0.630
Classification	**MassiveScenarioClassification**	0.912	0.912
Classification	**Banking77Classification**	0.873	0.875
Classification	**ImdbClassification**	0.971	0.971
Classification	**ToxicConversationsClassification**	0.913	0.910
Classification	**MTOPIntentClassification**	0.915	0.912
Clustering	**MedrxivClusteringS2S**	0.448	0.448
Clustering	**StackExchangeClusteringP2P**	0.494	0.492
Clustering	**StackExchangeClustering**	0.800	0.795
Clustering	**TwentyNewsgroupsClustering**	0.630	0.625
Clustering	**MedrxivClusteringP2P**	0.470	0.468
Clustering	**BiorxivClusteringS2S**	0.476	0.475
Clustering	**BiorxivClusteringP2P**	0.520	0.518
PairClassification	**TwitterURLCorpus**	0.877	0.877
PairClassification	**SprintDuplicateQuestions**	0.964	0.964
PairClassification	**TwitterSemEval2015**	0.803	0.801
Reranking	**StackOverflowDupQuestions**	0.546	0.548
Reranking	**SciDocsRR**	0.891	0.890
Reranking	**AskUbuntuDupQuestions**	0.674	0.676
Retrieval	**CQADupstackMathematicaRetrieval**	0.369	0.370
Retrieval	**CQADupstackStatsRetrieval**	0.413	0.413
Retrieval	**CQADupstackTexRetrieval**	0.362	0.362
Retrieval	**SCIDOCS**	0.247	0.247
Retrieval	**CQADupstackEnglishRetrieval**	0.543	0.543
Retrieval	**ArguAna**	0.653	0.652
Retrieval	**TRECCOVID**	0.865	0.866
Retrieval	**CQADupstackUnixRetrieval**	0.482	0.482
Retrieval	**CQADupstackGamingRetrieval**	0.632	0.633
Retrieval	**CQADupstackGisRetrieval**	0.444	0.448
Retrieval	**CQADupstackWordpressRetrieval**	0.388	0.386
Retrieval	**FiQA2018**	0.601	0.601
Retrieval	**SciFact**	0.805	0.805
Retrieval	**CQADupstackPhysicsRetrieval**	0.549	0.548
Retrieval	**NFCorpus**	0.431	0.431
Retrieval	**CQADupstackProgrammersRetrieval**	0.505	0.505
Retrieval	**CQADupstackAndroidRetrieval**	0.571	0.571
Retrieval	**CQADupstackWebmastersRetrieval**	0.464	0.464
STS	**BIOSSES**	0.848	0.854
STS	**STS13**	0.897	0.888
STS	**STS12**	0.803	0.804
STS	**STSBenchmark**	0.888	0.886
STS	**STS15**	0.902	0.900
STS	**STS14**	0.853	0.851
STS	**STS16**	0.864	0.869
STS	**STS22**	0.672	0.748
STS	**SICK-R**	0.822	0.823
STS	**STS17**	0.911	0.908
Summarization	**SummEval**	0.313	0.314
Average Score		0.686	0.687

Table 3: MTEB Results on different instructions

4.2 The Inconsistency of MSMARCO Scores

After releasing the Jasper model, an enthusiastic user (user name is raghavlite, https://huggingface.co/raghavlite) points out that the NDCG/MRR score is perfect and the MAP score is very low. Jasper model is distilled from *stella_en_1.5B_v5* and *NV-Embed-v2* and their MSMARCO score do not have this appearance. As of now, we still haven't been able to figure out what happened. Anyway, we report this phenomenon in the hope that it will help some people. The details and updates can be found in https://huggingface.co/infgrad/jasper_en_vision_language_v1/discussions/3.

4.3 Possible Improvements for Vision Encoding

Because of time and resource constraints, we were only able to give the jasper model a basic image encoding capability. In our initial conception, Stage4 is a basic visual language alignment training, while Stage5 (i.e. final stage, but we do not have this training) is the process of contrastive learning by using dataset of VQA. Besides, we found our loss in stage4 is oscillatory. All in all, there is a lot of room for improvement in the multimodal training stages.

5 Conclusion

In this paper, we introduce training details of Stella and Jasper models. The approaches of distillation and reducing dimensions can enable small models to achieve better results.

References

Ibrahim Alabdulmohsin, Xiaohua Zhai, Alexander Kolesnikov, and Lucas Beyer. 2024. Getting vit in shape: Scaling laws for compute-optimal model design.

Jianlyu Chen, Nan Wang, Chaofan Li, Bo Wang, Shitao Xiao, Han Xiao, Hao Liao, Defu Lian, and Zheng Liu. 2024. Air-bench: Automated heterogeneous information retrieval benchmark.

Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, and Mark Crowley. 2021. Johnson-lindenstrauss lemma, linear and nonlinear random projections, random fourier features, and random kitchen sinks: Tutorial and survey.

Shuhao Gu, Jialing Zhang, Siyuan Zhou, Kevin Yu, Zhaohu Xing, Liangdong Wang, Zhou Cao, Jintao Jia, Zhuoyi Zhang, Yixuan Wang, Zhenchong Hu, Bo-Wen Zhang, Jijie Li, Dong Liang, Yingli Zhao, Yulong Ao, Yaoqi Liu, Fangxiang Feng, and Guang Liu. 2024. Infinity-mm: Scaling multimodal performance with large-scale and high-quality instruction data.

Michael Günther, Jackmin Ong, Isabelle Mohr, Alaeddine Abdessalem, Tanguy Abel, Mohammad Kalim Akram, Susana Guzman, Georgios Mastrapas, Saba Sturua, Bo Wang, Maximilian Werk, Nan Wang, and Han Xiao. 2023. Jina embeddings 2: 8192-token general-purpose text embeddings for long documents.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. 2024. Matryoshka representation learning.

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. 2024. Nv-embed: Improved techniques for training llms as generalist embedding models. *arXiv preprint arXiv:2405.17428*.

Chaofan Li, MingHao Qin, Shitao Xiao, Jianlyu Chen, Kun Luo, Yingxia Shao, Defu Lian, and Zheng Liu. 2024. Making text embedders few-shot learners.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. 2024. Fineweb-edu: the finest collection of educational content.

Gabriel de Souza P Moreira, Radek Osmulski, Mengyao Xu, Ronay Ak, Benedikt Schifferer, and Even Oldridge. 2024. Nv-retriever: Improving text embedding models with effective hard-negative mining. *arXiv preprint arXiv:2407.15831*.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023. Improving text embeddings with large language models. *arXiv preprint arXiv:2401.00368*.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Multilingual e5 text embeddings: A technical report. *arXiv preprint arXiv:2402.05672*.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. 2023. Sigmoid loss for language image pre-training.