Advanced Deep Learning Topics: The Rise (And Fall) Of Transformers - Final Project

Gabriele Pagano*, Lorenzo Manoni†
M.Sc. Computer Science and Engineering, Politecnico di Milano
*gabriele1.pagano@mail.polimi.it, †lorenzo.manoni@mail.polimi.it

Abstract

This paper aims to show the application of a CLIP (Contrastive Language-Image Pretraining) model within the context of medical applications, particularly focusing on tasks such as image and text retrieval and zero-shot performance analysis on a new unseen dataset. The model was trained using a dataset comprised of medical images paired with their corresponding captions.

1. Introduction

Artificial Intelligence (AI) and Deep Learning have become increasingly intertwined with biomedical sciences, healthcare, and, more broadly, medical applications. In recent times, there has been a burgeoning need to classify, analyze, and interpret medical images, paralleled by a corresponding increase in available data. Deep Learning, as a versatile tool, has seamlessly aligned with these objectives, facilitating the automation and simplification of these tasks. And it's in this scenario that the aim of our project [1] is collocated: we would like to find out if this type of model performs well in a medical scenario and, given the fact that it has been trained with a set of images and associated captions belonging to the restricted domain of radiological images, if it preserves the exceptional zero-shot performance capabilities that it has demonstrated in the original CLIP implementation.

2. Related work

Our study is centered on adapting and training a CLIP model, originally developed by OpenAI in 2021. Our primary references were the seminal OpenAI CLIP paper [2] and its accompanying GitHub repository [3]. These resources provided foundational insights into the CLIP model's logic, particularly regarding the loss function and the design of key architectural components, such as the Projection Head's structure.

Additionally, a pivotal source of inspiration was a CLIP implementation example on the official Keras website [4]. This example was instrumental in guiding our approach to adapting the CLIP model within the TensorFlow/Keras framework. Key takeaways included the utilization of TFRecords for efficient data retrieval and a high-level framework for reimplementing CLIP. This example not only facilitated the efficient handling of the original, non-resized dataset but also provided a benchmark for comparing the performance of our implementation with existing models. Furthermore, it offered insights into the effective implementation of a text/image retrieval task.

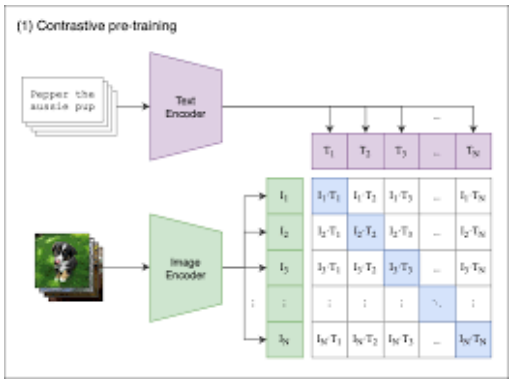

Figure 1: Contrastive Pretraining - the logic behind CLIP

However, these CLIP implementations were not entirely suited to our specific requirements. Therefore, we adopted a selective approach, integrating certain architectural and workflow elements from these sources while allowing room for creative experimentation and customization in our CLIP implementation.

3. Proposed approach

In our approach to implementing CLIP, we chose to train five distinct models (called v1, v2, v3, v4 and v4+). Each model was trained with varying configurations, including the use or omission of data augmentation, different datasets,

and diverse image encoders. This strategic diversity in training setups was designed to provide a broad spectrum of insights. The culmination of our project was dedicated to a comparative analysis phase. In this stage, we evaluated each model against a set of performance metrics. This thorough comparison allowed us to illuminate the distinct differences between the models, while also shedding light on their respective advantages and disadvantages. Our aim was to not only assess each model's performance but to also understand the impact of different training variables on their effectiveness.

3.1. Datasets

In our research, we trained the models using two distinct datasets, referred to in this paper as the 'non-resized' and 'resized' datasets:

- **Non-Resized Dataset:** This dataset comprises the ROCO (Radiology Objects in COntext) collection [5], featuring 81,771 high-resolution radiology images, each accompanied by a textual caption. These images are provided in their original high resolution and vary in size.
- **Resized Dataset:** As an extension of the first, this dataset includes a slightly larger collection of 83,275 image-caption pairs. It is important to note that the resized dataset is more comprehensive than the non-resized one. This discrepancy arose because some images from the original ROCO GitHub repository [6] were no longer available at the time of our data retrieval.

Accompanying the resized dataset, we also utilized a set of 8,374 concepts. Each concept is linked to a varying number of images, ranging from as few as 2 to over 10,000. We chose to use this diverse set of concepts as our test set, challenging the models to identify the most relevant images for a given concept name it has never interacted with before.

3.2. Data Preprocessing

Faced with the challenge of managing a substantial volume of data on the limited resources offered by Google Colab, it became necessary for us to properly prepare our input data to overcome hardware constraints.

Initially, we randomly divided both the resized and non-resized datasets into training and validation sets, adhering to an 80%/20% split. This resulted in:

- **Training Sets:** 66,620 images and captions for the resized dataset, and 65,415 pairs for the non-resized dataset.
- **Validation Sets:** 16,655 images and captions for the resized dataset, and 16,353 pairs for the non-resized dataset.

Subsequently, to address the significant loading time required for the non-resized images, we opted to store each image and its corresponding caption in separate TFRecords, carefully categorizing them as either training or validation data.
This method involved encoding images and captions into a binary file format that retains the original resolution, offering a compact and efficient storage solution. This approach not only conserved disk space but also significantly reduced loading times thanks to the use of contiguous storage cells in the binary format further that enhanced the efficiency of data retrieval.

The next phase involved the preprocessing of data to ensure readiness for the encoders:

- **Images:** We decoded, resized (to 224x224 and to 128x128 for the image encoder of model v1), and normalized each image according to the requirements of each vision encoder. Additionally, for model v4+, we implemented a data augmentation pipeline.
- **Captions:** We tokenized each caption and converted it into two lists: 'input_ids', containing the ID of each token, and 'attention_mask', a sequence of '1's corresponding to the indices in 'input_ids' and zeros elsewhere, for efficient processing.

3.2.1 Data Augmentation

To assess the influence of data augmentation techniques on the model's generalization capabilities, we incorporated a comprehensive data augmentation pipeline in model v4+. This pipeline included a variety of transformations to enhance the robustness and versatility of the model:

- **Random Rotations:** Introducing variability in the orientation of the images to simulate different viewing angles.
- **Random Specular Reflections:** Mimicking the effect of varying lighting conditions and perspectives.
- **Random Contrast Adjustments:** Altering the contrast levels to test the model's ability to recognize features under diverse lighting conditions.
- **Random Gaussian Noise:** Adding a level of complexity to the images to simulate real-world imperfections and noise.

The integration of these augmentation techniques into the final model notably enhanced its performance, thereby validating their effectiveness. Through this approach, we demonstrated that strategic data augmentation can significantly improve a model's ability to generalize across varied and challenging datasets.

3.3. Datasets creation

Upon completing the data preprocessing phase, we proceeded to construct the dataset. This process involved several key steps to optimize performance and efficiency. Firstly, the dataset was thoroughly shuffled to ensure a random distribution of data, which is crucial for robust model training. We then implemented prefetching to enhance the performance of the data retrieval phase, allowing for smoother and faster access to the data during training.
Finally, a critical aspect of this process was the batching of the data: since our objective was to strike a delicate balance between the limited memory availability of Google Colab and the parallel processing capabilities of the GPU, special attention was given to properly tune the batch size parameter. This careful optimization ensured that we maximized the efficiency of our resources without compromising the integrity and performance of our model training.

3.4. Models and Components

In accordance with the original CLIP model framework, our approach involved the simultaneous training of an image encoder and a text encoder. Our objective was to experiment with various architectural combinations to identify the most effective configuration, considering the limited data available to us. For this purpose, we made specific design choices for our encoders:

- **Image Encoders:** We utilized a ResNet-50 Convolutional Neural Network from Keras.applications, with 23.6M parameters, and a Visual Transformer (ViT-small) from the Hugging Face transformers library, with 21.8M parameters . Initially, we also experimented with an XCeption model and a ViT base. However, the XCeption model was eventually discarded due to poor performance during training with the resized dataset, and the ViT base was set aside owing to its excessive GPU RAM requirements.
- **Text Encoders:** Our choice was a BERT-small model from the Hugging Face transformers library, with 28.8M parameters. We initially considered using a BERT-base model, but concluded it was unnecessary given the limited complexity of our captions.

These architectural choices, in conjunction with the use of either resized or non-resized images and corresponding captions, led to the development of four distinct models:

- **Model v1:** Resized images + ResNet-50, 54.2M parameters.
- **Model v2:** Resized images + ViT-small, 51.6M parameters.
- **Model v3:** Non-resized images + ResNet-50, same number of parameters of v1.
- **Model v4:** Non-resized images + ViT-small, same number of parameters of v2.

As previously mentioned, a fifth model, v4+, was developed by integrating a data augmentation pipeline into model v4.

3.4.1 Projection Head

In our pursuit to develop a multimodal model akin to CLIP, in the next phase we focused on creating a component that could map the features of each image and text batch onto the same embedding space. This crucial component is known as the Projection Head, and it comprises a sequential arrangement of the following layers:

- **Dense layer:** A standard Dense layer equipped with a ReLU activation function.
- **Normalization Layer:** Implemented with an epsilon parameter set to 10^{-5}.
- **Another Dense Layer:** To further process the embeddings.
- **Dropout Layer:** We opted for a dropout value of 0.1 to prevent overfitting.

The combination of the latter three layers forms what we refer to as the 'projection_layer'. Our experimental process also included a manual tuning of the optimal values for the number of projection layers and the dimension of the embeddings. For the projection layers, we tested values of 1 and 2, eventually deciding to incorporate 2 projection layers in our Projection Head. In terms of embedding dimensions, we experimented with sizes of 256, 512, and 1024, ultimately selecting 512 as the most effective choice.

3.4.2 CLIP

After delineating the essential components, our subsequent endeavor involved architecting our CLIP model. We achieved this by defining a class that, upon invocation during the training process, performs the following functions:

- **Initialization of Internal Parameters:** This includes setting up the encoders, Projection Head, and other necessary variables like a set of encoders' parameters, that will be used during fine-tuning stage.
- **Feature Retrieval:** The class retrieves image and text features from the current data batch.
- **Projection Head Processing:** These features are then fed into the Projection Head to generate the corresponding image and text embeddings.

```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

Figure 3. Numpy-like pseudocode for the core of an implementation of CLIP.

Figure 2: Numpy-like pseudocode of the loss function we used

- **Embeddings Normalization:** The aforementioned embeddings are normalized.
- **Loss Computation:** Utilizing these normalized embeddings, the model computes the loss.

Regarding the loss computation, we adhered to the procedure outlined in the official CLIP implementation, shown in Figure 2. We found this methodology as the most effective approach to mitigate training instabilities, despite it diverged significantly from the Keras example [3] and other CLIP implementations found online.

A key focus was placed on optimizing the temperature parameter: initially we experimented with higher values as recommended in the CLIP paper, but these values proved to be problematic for our specific implementation and after testing with values of 10, 15, 20, and 30, we ultimately determined that setting the temperature parameter to 0 was the most suitable choice for our model.

3.4.3 Optimizers

To train our CLIP model, we divided the training process into two distinct stages. In the initial phase, we focused solely on training the core components of the model, excluding the encoders. This necessitated a specific approach to optimization:

- For this initial phase, we utilized an Adam optimizer with an epsilon parameter of 10^{-8}, combined with a CosineDecay learning rate scheduler and a weight decay of 0.2. This setup was specifically designed to optimize the foundational aspects of the CLIP model, preparing it for the subsequent fine-tuning stage.
- During the fine-tuning phase, the encoders were then brought into play, requiring a more nuanced optimization strategy. To address this, in addition to the CLIP optimizer, we introduced an Adam optimizer for the encoders during the fine-tuning phase. This optimizer, featuring an epsilon parameter of 10^{-8}, was augmented with an ExponentialDecay learning rate scheduler, ensuring efficient and stable training of the encoders.

By adopting this two-phase optimization approach, we were able to carefully tailor the training process to the specific needs of each model component, thereby enhancing the overall performance and effectiveness of our CLIP model.

3.5. Training process

As previously mentioned, our training phase was structured into two distinct stages, both performed using a V100 GPU:

- In the initial stage, we set both encoders as non-trainable. This phase, spanning 8 epochs, was exclusively dedicated to training the Projection Head parameters. To streamline this process and enhance training efficiency, we implemented several useful callbacks:
  - **PrintLRCallback:** This callback monitors and displays the current learning rate on the screen.
  - **EarlyStopping Callback:** A standard feature from Keras.callbacks, this tool monitors the validation loss and halts the training prematurely if no decrease is observed within 5 epochs.
  - **ModelCheckpoint Callback:** Implemented to save the model checkpoints to Google Drive after each epoch.
  - **SaveLastEpoch Callback:** Designed to record the numerical value of the last completed epoch in a text file (subsequently saved to Google Drive). Essential for resuming training after any interruption on Colab.
- Following the completion of the first training phase, we initiated the second stage. This phase, lasting 17 epochs (thus bringing the total training duration to 25 epochs), involved setting both encoders of the model to trainable. During this stage we also updated the list of encoder parameters: this step was crucial for the fine-tuning process since applying different optimizers to separate parts of the model in Keras is not straightforward. We overcame this challenge by manually separating the encoder gradients from those of the Projection Head and applying the proper optimizer to each

list. This delicate and labor-intensive step was crucial for the success of our training strategy.
Upon executing these adjustments, training resumed, and the models were trained for the remaining 17 epochs.

3.6. Testing metrics

After training our model, the next objective was to find an effective method to measure its capability to generalize and perform well on previously unseen data. Our focus was on two key aspects:

- **Selecting an Appropriate Dataset:** This was particularly crucial because, although OpenAI's original CLIP model demonstrated exceptional zero-shot capabilities across diverse unseen datasets, it had been trained on an extensive set of 400 million general images. Given that our model was trained on a more limited and less varied dataset, it was vital for us to find an image dataset outside the radiological domain but still not overly specific to domains unfamiliar to our model. This consideration might seem counterintuitive in a zero-shot testing scenario but is logical given the specificity and limitations of our training dataset.

- **Establishing Robust Performance Metrics:** This was also crucial, considering the nature of our tasks: identifying a relevant set of images for a textual query and vice versa, retrieving captions that aptly represent a given query image.

In addressing the first point, our initial challenge was the absence of a dedicated test set. To address this, we leveraged a concepts file containing IDs of images related to 8374 concepts. From this, we constructed a test set and, after generating image embeddings from our dataset, we used these concepts as textual queries. Our objective was to assess how effectively our model could retrieve images closely associated with these concepts. While this approach appeared logical for testing our model's generalization, it posed three significant issues: the concepts were within the same domain as our training images, it solely tested image retrieval capabilities, and the concepts were more akin to labels than actual query captions.
To circumvent these limitations and after initially testing our models with the concepts, we sought a dataset within the medical field but with non-radiological images. This decision was made to avoid overwhelming the model with too many unfamiliar concepts. However, we soon realized that for a true zero-shot performance evaluation, it was essential to challenge the model with scenarios vastly different from its training environment. This led us to the decision of using neutral, general-purpose images. The Flickr-8k dataset [7] emerged as an ideal choice, fulfilling our criteria: it contains general images, each accompanied by well-crafted captions, and its size was comparable to our concept dataset.
For the latter point, we initially opted for common performance metrics: Precision@K, Recall@K, F1@K, and Accuracy@K (where K is the number of images retrieved in the text retrieval task and the number of captions retrieved for the image retrieval task). However, the concepts in our test set were linked to a highly variable number of images, ranging from 2 to over 10,000. This variation meant that the first three metrics did not accurately reflect our model's capabilities, as the count of false negatives or false positives heavily depended on the chosen value of K. For instance, a concept with only two relevant images would inevitably yield numerous false positives even at a small K, whereas a concept with numerous associated images would generate many false negatives.
Upon these considerations and recognizing that the aims of our tasks (i.e., providing the most appropriate images/texts) are akin to those of a Recommender System, we drew inspiration from that field. Eventually, we decided to retain only the accuracy metric and supplement it with:

- **MRR@K(Mean Reciprocal Rank at K):** This metric measures the average position of the most relevant image/caption among the model's results, offering a valuable insight into how swiftly the model delivers the most accurate answer.

- **nDCG@K(Normalized Discounted Cumulative Gain at K):** This is another critical metric as it assesses the order in which relevant answers are presented, considering their position in the ranking. It effectively evaluates the model's ability to identify and rank the most pertinent responses to a query.

4. Results

In this section, we delve into the performance analysis of our models, aiming to delineate the strengths and weaknesses of each. We will provide a comparison across various metrics to give a comprehensive view of their capabilities.

4.1. Testing on Concepts

As outlined earlier, the initial phase of our testing involved evaluating the models with new, unseen data. To achieve this, we utilized the available set of concepts, testing the image retrieval capabilities of our models when presented with novel query captions.
This testing procedure was specifically conducted for models v1 and v2, as they were the only ones trained with the resized dataset, which corresponded to the concepts available for testing.

Below are the results from this testing phase:

Model	nDCG@100	MRR@100	Accuracy@100
v1	0.0027	0.0090	0.0899
v2	0.0030	0.0092	0.1048

Analyzing these results, it's apparent that model v2, which employs a Visual Transformer (ViT-small) as its image encoder, slightly outperforms model v1, which uses a ResNet-50 encoder. Specifically:

- The nDCG@100 (Normalized Discounted Cumulative Gain at 100) score for model v2 is marginally higher than that of model v1. This suggests that model v2 is more effective in ranking relevant images higher in the retrieval results.
- Similarly, the MRR@100 (Mean Reciprocal Rank at 100) score, which measures the average of reciprocal ranks of the first relevant item, is slightly better in model v2. This indicates that, on average, the first relevant image is found at a slightly higher rank in the search results for model v2 compared to model v1.
- Most notably, the Accuracy@100 metric, which evaluates the presence of a correct image within the top 100 results, is significantly better in model v2 (0.1048) compared to model v1 (0.0899). This highlights model v2's superior ability to retrieve relevant images within the top 100 results, emphasizing the effectiveness of the ViT-small encoder in handling the task of image retrieval.

In conclusion, while both models show a relatively low performance overall, the ViT-small encoder in model v2 demonstrates a modest yet consistent improvement across all metrics compared to the ResNet-50 encoder in model v1. This suggests that the ViT-small architecture might be more adept at capturing the nuances necessary for the image retrieval task in this specific context.

4.2. Zero-shot performance

Another approach we employed to evaluate the performance of our models involved testing their zero-shot learning capabilities with a new, unseen dataset. As explained in previous chapters, it was crucial for us to select a dataset that was as general and neutral as possible. We achieved this by opting for the Flickr-8k Dataset, which comprises 8,091 neutral images, each accompanied by five captions.
To gauge the zero-shot performance on this dataset, we adhered to the following methodology:

- **Dataframe Creation:** Initially, we constructed a dataframe that included the paths to the images. For each image, we randomly selected one caption from the available five.
- **Saving the Dataframe as a .csv File:** Subsequently, the dataframe was saved as a .csv file. This step was crucial as it allowed us to maintain consistency in the test set across different models. By using the same test set, we could ensure a fair and accurate comparison of the models' performance.
- **Embedding Generation:** Initially, the image and text encoders of our model were utilized to generate the respective image and text embeddings.
- **Similarity Metrics Calculation:** We then calculated the similarity metrics between each set of embeddings, employing cosine similarity to compare them against an identity matrix of the same dimensions.

In addition to using cosine similarity for comparing the matrices, we also assessed performance based on Accuracy@100 and MRR@100 metrics.

The outcomes of this analysis are detailed in the following table:

Model	MRR@100	Accuracy@100	Cosine sim.
v1	0.0004	0.0099	0.00255
v2	0.0009	0.0142	0.00440
v3	0.0007	0.0116	0.00505
v4	0.0008	0.0125	0.00509

These results can be analyzed from different points of view:

- **Performance Comparison:** Model v2 outperforms the others in terms of MRR@100 and Accuracy@100, but interestingly, it has a lower cosine similarity than Model v4. This is noteworthy as Model v4, trained with non-resized, theoretically less noisy images, was expected to perform better in a zero-shot context.
- **ViT-small vs. ResNet-50:** The superior performance of Model v2 (ViT-small) over Model v1 (ResNet-50) aligns with expectations about ViT architectures capturing global features effectively. However, the fact that Model v2 also surpasses Model v4 (also ViT-small but with non-resized images) suggests that resized images might have provided a form of regularization, aiding in the model's generalization.
- **Cosine Similarity Insights:** Although Models v3 and v4 have slightly higher cosine similarity, this does not translate into better performance in terms of MRR and Accuracy. This suggests that while they are better at aligning images and text in the embedding space, they may not be as effective in ranking relevant images in practical scenarios.

Concluding our discussion, we recall that model v4 underwent a retraining process where a data augmentation pipeline applied to the images. This enhanced version of the model has been designated as v4+.
Below is a comparative table that compares the performance metrics of v4, its augmented counterpart v4+, and the notably surprising model v2:

Model	MRR@100	Accuracy@100	Cosine sim.
v2	0.0009	0.0142	0.00440
v4	0.0008	0.0125	0.00509
v4+	0.0009	0.0138	0.00562

From these data we can retrieve two key observations:

- **Effect of Data Augmentation (v4 vs. v4+):** The most notable observation is the improvement in all performance metrics in v4+ compared to v4. This suggests that the data augmentation techniques used for v4+ have positively impacted its ability to generalize and perform better in zero-shot tasks and underscores the potential of data augmentation in enhancing model robustness.
- **Comparison with v2:** Interestingly, v4+ now matches the MRR@100 of v2 and closely approaches its accuracy, while maintaining a higher cosine similarity. This is significant because v2, which was trained on resized images, keeps its superior performance showed with the previous results.

5. Conclusions and future works

The obtained results suggest that for zero-shot learning tasks, using resized images and ViT-small as an image encoder may be advantageous over non-resized images, even with a potentially more powerful model configuration. This phenomenon is likely attributed to the innate robustness of the ViT-small architecture. We believe that the ViT-small perceives the noisy, resized images as a form of extensive augmentation of the non-resized dataset, enabling it to adapt and excel even in the presence of this 'noise.'
Moreover, the analysis demonstrates that data augmentation can be a powerful tool in improving the performance of CLIP models, especially in the context of zero-shot learning capabilities. Model v4+'s enhanced performance suggests that the augmentation strategies employed effectively compensated for the potential shortcomings of using non-resized images in training. This comparison highlights the importance of not only model architecture and training data but also the techniques used in data preprocessing and augmentation.
In conclusion, despite our results were way too far from the ones obtained by the original CLIP implementation, given the limited and restricted nature of our dataset we believe that from our findings future explorations could be done in the field, which include experimenting with a more wide and varied dataset, alternative encoder architectures, including those for text, and more nuanced adjustments in hyperparameters, especially the temperature value and other crucial parameters that can potentially enhance the model's performance.

References

[1] https://drive.google.com/drive/folders/157HbviuaoAqN-Y5Y6a5rSTRuRNoV6ZSG?usp=sharing

[2] https://arxiv.org/pdf/2103.00020.pdf

[3] https://github.com/openai/CLIP

[4] https://keras.io/examples/vision/nl_image_search/

[5] https://link.springer.com/chapter/10.1007/978-3-030-01364-6_20

[6] https://github.com/razorx89/roco-dataset/

[7] https://www.kaggle.com/datasets/adityajn105/flickr8k

A. Appendix

In conclusion, presented within this appendix is a comprehensive summary table that encapsulates all the final parameters employed in our project:

Parameter	Value
batch size	64
buffer size	5000
initial CLIP learning rate	10^{-4}
initial encoders learning rate	10^{-5}
weight decay	0.2
n° of patience epochs for Early Stopping	5
n° of total epochs	25
n° of first stage epochs	8
tokens max length	200
Adam epsilon	10^{-8}
target lr for CosineDecay	10^{-6}
warmup target	$5*10^{-4}$
warmup steps	200
exponential decay rate	0.9
temperature	0
n° of projection layers	2
dim of embedding space	512
dropout	0.1