Reinforcement Calibration SimCSE

WANG Yu

YE Jingyi

ZHAO Qianyi

Table of Contents

1. Introduction

Natural language processing (NLP) studies have experienced a boom and witnessed many breakthroughs over the past few years. Large-scale pre-trained language models (e.g., BERT, Devlin et al., 2018) and self-supervised learning started the revolutions across many NLP tasks, from machine translation to sentiment analysis.

One of the most important NLP tasks that supports many downstream tasks is Semantic Textual Similarity (STS). STS task focuses on measuring and evaluating the degree of textual relatedness between natural language texts based on their semantic meanings. STS plays a crucial role in machine translation, dialogue system, social media analysis, and many other NLP tasks. However, measuring STS is a challenging task because of the complexity and ambiguity of natural languages. Common obstacles include semantic ambiguity, syntactic variations, and lexical differences in natural languages, making it difficult for NLP scholars to develop an effective method to measure semantic similarity with a high accuracy.

As NLP systems like ChatGPT are becoming increasingly integrated into our working pipeline and everyday applications, there is an urgent need to build novel NLP models to represent textual data and understand semantic meaning in a self-supervised or unsupervised way. Self-supervised learning, including contrastive learning, reduces the heavy burden of class labelling efforts and makes models more generalizable to many NLP tasks, including STS.

One of the notable challenges in self-supervised representation learning is learning highly accurate sentence embeddings that can capture the semantic meanings of natural languages. SimCSE model (Gao et al., 2021) adopts the contrastive learning method to learn sentence embeddings and uses dropouts as the data augmentation strategy. We notice that SimCSE model also has some limitations, especially in unsupervised SimCSE. First, as unsupervised SimCSE augments the data with dropout, this method will lead to the imbalance of numbers of positive pairs versus and negative pairs. Second, their approach ignores semantically similar samples in negative pairs. Third, data augmentation via dropouts will result in the sentence “length bias”. In another word, the model will regard two sentences as more similar when they are closer in length. These limitations can lead to less accurate embeddings and negatively affect the model's performance on downstream tasks.

To address these limitations of SimCSE and further enhance the performance of self-supervised representation learning, our project presents a novel model named "Reinforcement Calibration SimCSE". This model builds upon the idea of SimCSE, and borrows the ideas of Artificial Potential Fields (Khatib, 1986), Perceptual Loss (Johnson et al., 2016), Reinforcement Learning from Human Feedback (RLHF) (e.g., Christiano et a., 2017), and Deep Deterministic Policy Gradient (DDPG) (Silver et al., 2014) from various fields to improve the learning of sentence embeddings.

We then conduct a series of experiments including the stages of training, fine-tuning and evaluation. To evaluate the effectiveness of the proposed model and its implementation, the model is tested on several benchmark datasets for STS tasks with the SentEval NLP evaluation toolkit (Conneau and Kiela, 2018). The experiment results suggest the potential of our model as a starting point in addressing the challenges in STS tasks.

2. Related Work

Our work falls in the field of self-supervised and unsupervised sentence representation learning in NLP. Socher, et al. (2013), Le and Mikolov (2014) and Hill, et al. (2016) suggest learning sentence representation based on each phrase's intrinsic structure. Based on the distribution hypothesis, Logeswaran and Lee (2018) can forecast the sentences that come before and after a given sentence. Pagliardini, et al (2018) propose Sent2Vec which enables the creation of sentence embeddings using word vectors and n-gram embeddings.

In particular, our work is closely related to contrastive learning application in sentence representation learning. These contrastive learning-based techniques for embedding sentences are often predicated on the idea that a good semantic representation ought to be able to draw similar phrases together while pushing away sentences that are different. SimCSE, as proposed by Gao, et al (2021), followed by ESimCSE (Wu et al., 2021), uses contrastive learning for embedding sentences. Their model is based on this idea.

Our work is also related to reinforcement learning in NLP tasks, especially reinforcement learning from human feedback (RLHF). Bohm, et al. (2019) build a reward function and train a strategy whose summaries are preferable to a policy maximizing ROUGE. Ziegler, et al. train Transformer models to optimize human feedback on a variety of tasks.

More generally, our work falls in the huge stream of literature on Semantic textual similarity (STS). There are many seminal papers that address the challenges in STS task and achieve high performance, such as Kiros, et al. (2015), Conneau, et al., (2017), Cer, et al. (2018), and Yang, et al. (2018).

3. Methodology

3.1 Artificial Potential Field

3.1.1 Basic concepts

We introduce the artificial potential field to make improvements to address the problem of severe imbalance in the positive and negative pairs and the neglect of similar samples in the negative pairs in the unsupervised SimCSE model. By using the gravitational and repulsive potential fields, the target is subject to the combined force of gravity and repulsion at this point in the construction of the positive and negative pairs, thus achieving the objective of more accurately pulling in the more similar samples and pushing out the less similar ones.

3.1.2 Integrating ideas

To integrate the idea of artificial potential fields into SimCSE, we set a similarity threshold T_h based on a pre-trained similarity matrix of sentences, with the parts above this threshold being drawn closer and the parts below this threshold being pushed further away, thus constructing the potential field function as follows:

$$G(x) = \begin{cases} (x - T_h)^2 \\ -\dfrac{T_h^2}{(T_h - 1)^2}(x - 1)^2 \end{cases}$$

A negative $G(x)$ indicates that the output vector is drawn close, and a positive $G(x)$ indicates that the output vector is pushed far away. From this, the corresponding potential field matrix after input similarity can be obtained as follows:

$$G_k = \begin{pmatrix} G_{11} & \cdots & G_{1k} \\ \vdots & \ddots & \vdots \\ G_{k1} & \cdots & G_{kk} \end{pmatrix}$$

In order to avoid the disadvantage that the potential field matrix constructed from the input alone cannot be adjusted by the output, the distance matrix of the output vector needs to be calculated:

$$D_k = \begin{pmatrix} d_{11} & \cdots & d_{1k} \\ \vdots & \ddots & \vdots \\ d_{k1} & \cdots & d_{kk} \end{pmatrix}$$

Then, the potential field matrix (that determines the velocity vector of the output vector's motion) and the distance matrix (that determines the distance of the output vector) are combined to construct the potential field force matrix:

$$T_k = \frac{G_k}{D_k}$$

The potential field force matrix is then replaced with the SimCSE positive and negative example matrix to avoid the problem of $\frac{0}{0}$ appearing on the diagonal of the potential field force matrix by setting all the diagonal elements to 0, indicating that this output vector does not need to be shifted and does not contribute to the final gradient.

3.2 Perceptual Loss

3.2.1 Basic concepts

Perceptual loss is mainly used for image transformation problems. The deeper network layer extracts the more abstract and advanced features. Perceptual loss enhances the detailed information of the output features. The *L2* loss is constructed by a fixed network with the input ground truth and the prediction, and then the corresponding output features *feature_gt* and *feature_pre* are obtained to approximate the perceptual information between the ground truth and the prediction.

3.2.2 Integrating ideas

SimCSE augments data via dropout, which is straightforward and efficient, but it ignores the length information and may result in "length bias" issues. This is mainly because the model is built on the Transformer, which uses a position vector to represent the length information of a sentence. This implies that two sentences from the same source will typically have the same length information in a positive pair, but two phrases from separate sources will often have different length information in a negative pair. As a result, the model is more likely to deviate from objective scores when the given pairs of phrases are of the same or similar length.

To incorporate the idea of perceptual loss into SimCSE, we use cosine similarity to measure the semantic similarity between estimated sentence embeddings and use this measure to guide the optimization of force field loss in artificial potential fields by substituting the sum of potential field force matrices T_k as the loss function into the network for gradient descent:

3.3 Deep Deterministic Policy Gradient

3.3.1 Basic concepts

The Deep Deterministic Policy Gradient (DDPG) algorithm builds on the deterministic policy gradients (DPG) algorithm's ability to handle tasks in continuous action spaces but cannot learn policies directly from high-dimensional inputs. It combines the success of the Deep Q-Network (DQN) algorithm which can perform direct end-to-end learning but can only handle discrete action space problems. It can achieve a model-free deep reinforcement learning algorithm that performs end-to-end learning directly from raw data.

3.3.2 Integrating ideas

To make our model work better, we used the Reinforcement Learning for Human Feedback (RLHF) algorithm for fine-tuning. The DDPG loss function is used to update the parameters of the critic network (i.e., the PerceptualBERT model) based on the feedback received from the user. The DDPG loss function is defined as follows:

$$loss = -k \times pi \times rewardi$$

where $rewardi$ is reward from reward function, pi is the probability of selecting the given sentence pair, and k is a positive constant (set to 50 in this implementation).

4. Experiments

4.1 Load the Based BERT Weight From Pretrained

BertModel.from_pretrained() loads a pre-trained BERT model from HuggingFace. It loads a BERT model that Google trained on uncased English text with 6 layers, a hidden size of 256, and 4 attention heads. The *output_hidden_states =True* option instructs the model to also deliver the output of the final layer together with the hidden states for all other levels. *BertTokenizer.from_ pretrained()* initializes a tokenizer object for the pre-trained BERT model. The tokenizer parses the input text, which then assigns a unique identifier to each token so that the model can process it.

4.2 Construct Our Model

4.2.1 BertModel.from_pretrained()

A class named PerceptualBERT is first defined, using a common practice when performing sentence classification tasks with the BERT model for the purpose of extracting features from a given sentence, in particular by extracting the hidden representation of the sentence from the encoder layer. The process first initializes the embedding layer and extract the weights of the encoder layer from the pre-trained BERT model (*basemodel*) as our encoder weights. Then a *forward()* method is defined to first calculate the embedding of each word in the sentence as *(batch_size, seq_len, embedding_dim)*, where *embedding_dim* is the dimension of the word embedding used in the pre-trained model. The following embedded sentences are passed through the three encoder layers individually. Then a tensor of the form *(batch_size, hidden_size)*, representing the size of the hidden representations is obtained by passing the output tensor from the final encoder layer through the output layer. Finally, the result of the *forward()* function is the mean of this tensor along the second dimension or the sequence length dimension.

4.2.2 ForceBasedInfoNCE

4.2.2.1 force_field()

The function of *force_field()* is used to calculate the force field between sentence embeddings based on cosine similarity. The idea of artificial potential field is mainly reflected in this method. First, we take the absolute value of the cosine similarity matrix and set the threshold T_h to 0.95. For values below the threshold, the force is calculated as the square of the difference between the similarity and the threshold, thus attracting the corresponding sentence pairs closer to each other; for values above the threshold, the force is calculated as the square of the difference between the similarity and 1 multiplied by a negative constant, thus repelling the corresponding sentence pairs from each other. The *force_field()* method is then called in *forward()* to calculate the *trend_move* tensor, which represents the force applied to each sentence pair and is then used to calculate the loss. This loss encourages the output embeddings to move towards or away from each other depending on whether the corresponding input sentences are similar or dissimilar.

4.2.2.2 forward()

The function of *forward()* receives input_embeddings (sentence embeddings generated by the model for a given input text) and output_embeddings (sentence embeddings generated by the model for a different text that should be semantically similar to the input text). The idea of perceptual loss is mainly reflected in this method (the calculation of the estimated sentence embeddings, and the way used to calculate the cosine similarity). The process is as follows:

The input sentence embeddings are averaged along the sequence dimension to obtain an estimated sentence embedding. Next, the estimated sentence embedding is used to calculate the cosine similarity between all estimated sentence embeddings. Then the *force_field()* is called to calculate the force field based on the cosine similarity matrix. The force values are modified by being divided by the *real_distance matrix* (the pairwise distances between all output embeddings). This ensures that the force values are scaled according to the actual distance between the output embeddings. Finally, the loss is changed to the sum of the modified force values. The cosine similarity between the estimated sentence embeddings can be considered as a measure of how similar the input sentences are in terms of semantic content. The model is encouraged to generate output embeddings with similar semantic content by optimizing the force field loss so that the output embeddings are close to each other.

4.3 Train with the Corpus and Test the Model

We read the *wikisent2.txt* corpus, shuffle it randomly, and then select 1 million sentences as the training corpus. Each sentence is then tokenized using a tokenizer object created by the HuggingFace Transformers library. The [CLS] token was added to the beginning of the token list to indicate the start of the sentence, while the [SEP] token was added to the end to indicate the finish of the sentence. The length of the tokenized sentence is set to 128 and is padded if it is less than this value or cut off if it is greater than this value. Finally, the ID of each token in the sentence is obtained and appended to the list.

We then create an instance of the PerceptualBERT class and move the model to the CUDA device defined earlier. Then we load the trained model parameters from the *model.pt* file into the model instance, constructe the loss function and create an instance of the Adam optimizer for updating the model parameters during training. Finally, we set the batch size and the number of epochs, respectively.

Next, we create a training loop for the PerceptualBERT model using the force-based InfoNCE loss function and the Adam optimizer for a chosen training corpus. We also develop code to calculate the loss and cosine similarity between several embedding pairings to test and assess the effectiveness of the model and loss function.

4.4 Fine-Tune the Model Using Reinforcement Learning from Human Feedback (RLHF)

After developing the basic model, we can fine-tune it. In this part, we design a window-shaped GUI interface to collect human feedback on the similarity of two sentences randomly selected in the test corpus. Users can choose the options below the sentences to give their own human perceptions on the sentence similarity. **Figure 1** in Appendix shows a screenshot of the GUI interface.

In more details, for each iteration of DDPG, two sentences are randomly sampled from the test set. They are passed through PerceptualBERT to obtain the embedding vector of the sentences, and the similarity of the network output is calculated through cosine similarity. Meanwhile, these two sentences are manually scored to obtain the similarity of the human feedback. For these two similarities, *reward* can be calculated by the reward function, followed by probability calculation and update through probability estimator (the number of corresponding similarities in the corresponding probability estimator is added by 1). Subsequently, the loss of DDPG is calculated by – *reward* × *probability*, and back-propagate to PerceptualBERT. Then, it goes to the next iteration. **Figure 2** in Appendix shows the process of fine-tuning for one iteration.

5. Evaluation

We use SentEval (Conneau and Kiela, 2018) to evaluate the effectiveness of our proposed model. SentEval is an evaluation toolkit specialized for assessing the performance of a target model on a wide range of downstream tasks and probing tasks, including the Semantic Textual Similarity tasks. By using SentEval, we can quantitatively measure the performance of our model, and if possible, compare our results to other state-of-the-art methods.

Specifically, our evaluation process is conducted on several classical STS datasets, including STS12, STS13, STS14, STS15, STS16, and SICK-R. These datasets are frequently used because they can measure the relatedness between two sentences according to the cosine similarity of their sentence embeddings generated from the target model. The evaluation criteria used in STS tasks are Pearson correlation and Spearman correlation. They are the popular metrics to quantize the linear relationship between two variables. In the case of assessing model performance on STS tasks, we want to measure the association between the predicted textual similarity scores generated by the target model and the ground-truth scores annotated by humans.

As we have limited computing resources, we can only roughly adopt similar evaluation procedures and settings used in previous works like SimCSE. We train the model with 1 million randomly selected Wikipedia sentences (i.e., WikiSent2 dataset). We cannot conduct grid-search of the optimal set of hyperparameters, so we ad hoc adopt the batch size of 64 and learning rate of 1e-7.

The pre-trained BERT is a smaller version with 6 layers, 256 hidden dimensions, and 4 attention heads. Following unsupervised SimCSE, we train the model for only one epoch. As we do not have enough users or much time to fine-tune with RLHF method, therefore, we evaluate the trained model using the model checkpoint without fine-tuning.

The evaluation results are reported in **Table 1** in Appendix. Overall, our model has a fair performance on SentEval STS tasks. The model gets a very high performance on SICK-R dataset. While it appears badly on STS12-STS16 datasets, we hypothesize that this low performance might result from an error or bug in SentEval due to Python versions. As suggested by the literature, the model performance on SICK-R dataset and STS12-STS16 datasets should be comparable. We would like to solve this problem in future work. The results from this evaluation on SICK-R dataset showcase the potential of our model as a starting point of finding a novel solution for measuring the semantic relationships between texts.

6. Discussion

6.1 Limitations

In this project, we have developed a novel model for STS tasks. Although our approach has shown promising applications, there are some limitations and drawbacks that need attention:

(1) Shallow model architecture: We use a 6-layer model instead of a deeper model due to our computing resource constraint. Deeper and larger models require more computation time but may lead to better sentence embedding quality and high performance in STS tasks.

(2) Limited RLHF in fine-tuning: The users have to manually input for the RLHF, which is time-consuming and unscalable. As a result, we could only have several rounds of fine-tuning as tests and demonstrations.

(3) Sub-optimal set of hyperparameters: The choice of hyperparameter values is based on ad hoc decisions, including the learning rate of 1e-7 in Adam optimizer, and batch size of 64 in training. A grid-search method could be used to find a better set of hyperparameters.

(4) No data augmentation strategy: Unlike SimCSE, which uses dropouts as a data augmentation strategy, our approach does not involve any data augmentation techniques.

(5) No ablation studies: We did not conduct any ablation studies to assess the individual contributions of different components in our model. Ablation studies would help understand the importance of each component and identify issues for improvement.

6.2 Conclusions

In this project, building upon the idea of SimCSE, we develop a model "Reinforcement Calibration SimCSE" using both reinforcement learning and contrastive learning for the Semantic Textual Similarity (STS) task. Our methodology incorporates the ideas of Artificial Potential Fields, Perceptual Loss, Reinforcement Learning from Human Feedback (RLHF), and Deep Deterministic Policy Gradient (DDPG) from various fields to address the limitations of SimCSE and improve the learning of sentence embeddings. In a series of experiments and evaluations, our model performs well on the STS datasets provided by SentEval toolkit.

References

[1] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. arXiv preprint arXiv:1803.11175, 2018.

[2] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. Advances in neural information processing systems, 30, 2017.

[3] Alexis Conneau and Douwe Kiela. Senteval: An evaluation toolkit for universal sentence representations. arXiv preprint arXiv:1803.05449, 2018.

[4] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. arXiv preprint arXiv:1705.02364, 2017.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre- training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[6] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In Computer Vision–ECCV 2016: 14th European Confer- ence, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14, pages 694–711. Springer, 2016.

[7] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. The international journal of robotics research, 5(1):90–98, 1986.

[8] Ryan Kiros, Yukun Zhu, Russ R Salakhutdinov, Richard Zemel, Raquel Urtasun, Anto- nio Torralba, and Sanja Fidler. Skip-thought vectors. Advances in neural information processing systems, 28, 2015.

[9] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Ried- miller. Deterministic policy gradient algorithms. In International conference on machine learning, pages 387–395. Pmlr, 2014.

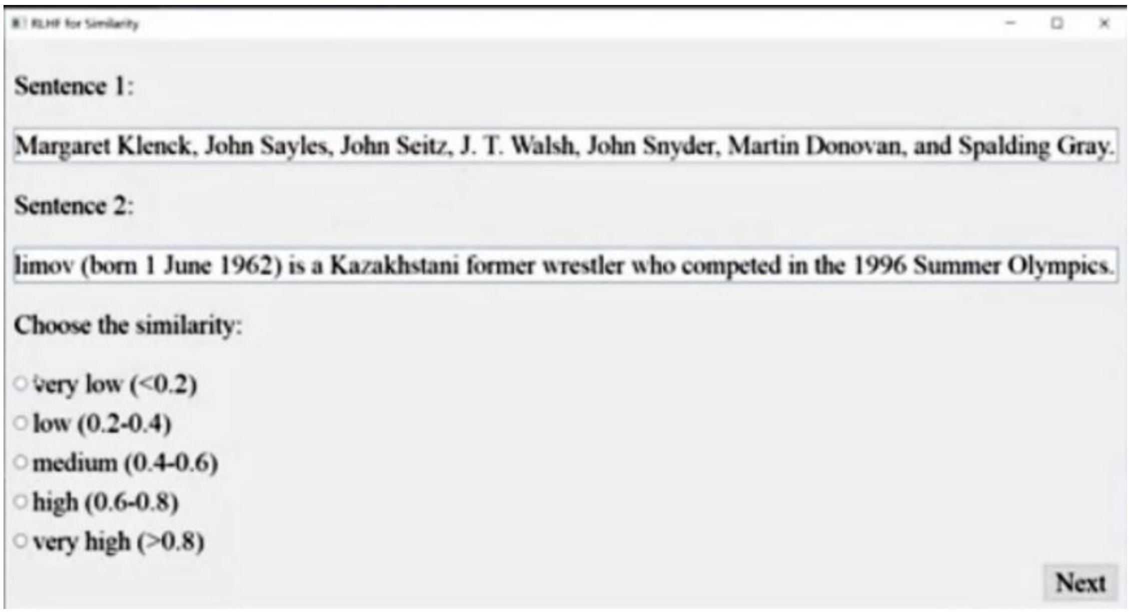
[10] Xing Wu, Chaochen Gao, Liangjun Zang, Jizhong Han, Zhongyuan Wang, and Songlin Hu. ESimCSE: Enhanced sample building method for contrastive learning of unsupervised sentence embedding. arXiv preprint arXiv:2109.04380, 2021.

[11] Yinfei Yang, Steve Yuan, Daniel Cer, Sheng-yi Kong, Noah Constant, Petr Pilar, Heming Ge, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Learning semantic textual similarity from conversations. arXiv preprint arXiv:1804.07754, 2018.

[12] Florian B¨ohm, Yang Gao, Christian M Meyer, Ori Shapira, Ido Dagan, and Iryna Gurevych. Better rewards yield better summaries: Learning to summarise without references. arXiv preprint arXiv:1909.01214, 2019.

[13] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. arXiv preprint arXiv:1909.08593, 2019.

Appendix



Sentence 1:

Margaret Klenck, John Sayles, John Seitz, J. T. Walsh, John Snyder, Martin Donovan, and Spalding Gray.

Sentence 2:

limov (born 1 June 1962) is a Kazakhstani former wrestler who competed in the 1996 Summer Olympics.

Choose the similarity:

- very low (<0.2)
- low (0.2-0.4)
- medium (0.4-0.6)
- high (0.6-0.8)
- very high (>0.8)

Figure 1. GUI interface of RLHF

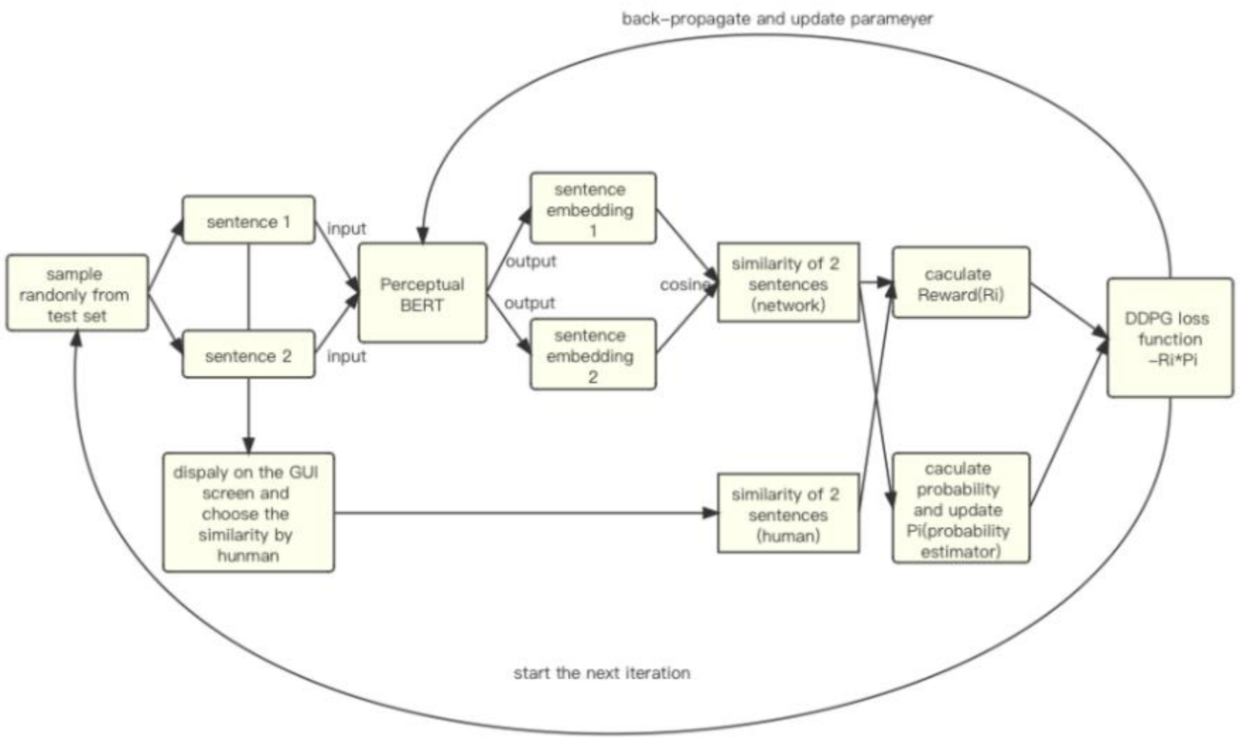

back-propagate and update parameyer
sample randonly from test set
sentence 1
sentence 2
input
input
Perceptual BERT
output
output
sentence embedding 1
sentence embedding 2
cosine
similarity of 2 sentences (network)
caculate Reward(Ri)
DDPG loss function -Ri*Pi
dispaly on the GUI screen and choose the similarity by hunman
similarity of 2 sentences (human)
caculate probability and update Pi(probability estimator)
start the next iteration

Figure 2. The fine-tuning process

	STS12	*STS13*	*STS14*	*STS15*	*STS16*	*SICK-R*
Pearson	5.26	8.42	4.49	9.53	13.20	66.84
Spearman	6.94	7.97	4.21	9.44	13.12	76.12

Table 1. STS task evaluation results using SentEval

Concept Supplement

1. Artificial Potential Field

The gravitational potential field function is:

$$U_{att}(q) = \frac{1}{2}\eta\rho^2(q, q_g)$$

where η is the scale factor, and $\rho^2(q, q_g)$ denotes the Euclidean distance between the original sample q and the comparison data q_g. The further the distance, the greater the value of the potential energy q is subject to. From this, it is possible to calculate the negative gradient of the gravitational field over the distance, which is the corresponding gravitational force $F_{att}(X)$. The repulsive potential field function is:

$$U_{req}(q) = \begin{cases} \frac{1}{2}k(\frac{1}{\rho(q, q_o)} - \frac{1}{\rho_o})^2, 0 \leq \rho(q, q_o) \leq \rho_o \\ 0, \rho(q, q_o) > \rho_o \end{cases}$$

where ρ_o is a constant representing the radius of influence of the comparison data, i.e., the repulsive force loses its effect when the "distance is too far". The negative gradient of the repulsive field is then calculated to obtain the corresponding repulsive force $F_{req}(q)$.

2. Perceptual Loss

When the network extracts feature from the image data, the shallower layer typically extracts low-frequency information like edge, color, brightness, etc. In contrast, the deeper layer extracts some high-frequency information like detailed texture. The deepest network layer extracts some key features with discriminatory features. The perceptual loss is calculated as:

$$\ell_{\mathrm{feat}}^{\varphi,\mathrm{j}}(\hat{y},y)=\frac{1}{\mathrm{C_jH_jW_j}}||\varphi_\mathrm{j}(\hat{y})-\varphi_\mathrm{j}(y)||_2^2$$

where y denotes the original image, $\hat{\mathrm{y}}$ denotes the generated image, φ denotes the pre-trained neural network, and j represents the number of layers in the current network.

3. Deep Deterministic Policy Gradient

DDPG uses the Actor-Critic algorithm as its basic framework, using a dual neural network model architecture (i.e., online network and target network) for both the policy function and the value function, respectively, the Critic network Q and the Actor network μ. Creating two copies of the neural network Q' and $\mu^{'}$ makes the learning process of the algorithm more stable and the convergence speed faster. In online Critic Q, the loss function is the mean square error (MSE):

$$L=\frac{1}{N}\sum_i(y_i-Q(s_i,a_i|\theta_Q))^2$$

The gradient $\nabla_{\theta_Q}L$ of L is then derived from the backward propagation and further optimized to obtain θ_Q. Online Actor μ, according to the deterministic strategy, its loss gradient is as follows:

$$\nabla_{\theta_Q}\mu\backslash s_i=\frac{1}{N}\sum_i\nabla_aQ(s,a|\theta_Q)|_{s=s_i,a=\mu(s_i)}\nabla_{\theta_\mu}\mu(s|\theta_\mu)|_{s=s_t}$$

The two target networks, Q' and $\mu^{'}$, are then averaged on a sliding scale:

$$Q_{\theta'}\leftarrow\tau\theta_Q+(1-\tau)\theta_{Q'}$$

$$Q_{\mu'} \leftarrow \tau\theta_{\mu} + (1-\tau)\theta_{\mu'}$$

At the same time, the DDPG algorithm introduces an experience replay mechanism. The experience data samples produced by the interaction between the actor and the environment are stored in the experience pool and batch data samples are extracted for training. It is comparable to the experience replay mechanism of DQN, which removes the correlation and dependency of the samples and facilitates the algorithm's convergence.