Proceedings of Machine Learning Research 1–7

Adversarial Contrastive Pre-training for Protein Sequences

Matthew B.A. McDermott MMD@MIT.EDU
Brendan Yap YAP@MIT.EDU
Tzu Ming Harry Hsu STMHARRY@MIT.EDU
Di Jin JINDI15@MIT.EDU
Peter Szolovits PSZ@MIT.EDU
CSAIL, MIT

Abstract

Recent developments in Natural Language Processing (NLP) demonstrate that large-scale, self-supervised pre-training can be extremely beneficial for downstream tasks. These ideas have been adapted to other domains, including the analysis of the amino acid sequences of proteins. However, to date most attempts on protein sequences rely on direct masked language model style pre-training. In this work, we design a new, adversarial pre-training method for proteins, extending and specializing similar advances in NLP. We show compelling results in comparison to traditional MLM pre-training, though further development is needed to ensure the gains are worth the significant computational cost.

Keywords: Protein pre-training, adversarial methods, pre-training, contrastive estimation, transformers

1. Introduction

Pre-training, particularly using a self-supervised masked language model (MLM) task over a large corpus, has recently emerged as a powerful tool to improve various prediction and generation tasks, first in natural language processing (NLP) via systems like BERT (Devlin et al., 2019), and later in other domains, including biomedical domains such as protein sequences[1] (Rao et al., 2019; Alley et al., 2019; Conneau et al., 2019; Lu et al., 2020). However, unlike NLP, where newer methods have brought improved performance, the state of the art for pre-training on protein sequences remains simple MLM style pre-training. For many tasks of interest, this offers only small benefits over prior methods, or even fails to match methods based on human constructed, alignment based features.

In this work, we design a new pre-training method for protein sequences, adapting the traditional MLM task by replacing the random masking scheme with a fully differentiable, adversarial masking model trained to choose which tokens to mask and how in order to make the pre-training model's recovery task *most* difficult subject to a budget constraint. This method is a form of adversarial contrastive estimation (Bose et al., 2018), building on the ideas explored in ELECTRA within NLP (Clark et al., 2019), but fully differentiable and trained adversarially, a task that is made more feasible given the protein domain's smaller vocabulary. Using an adversarial mask proposal distribution has also been recently explored in connection with reducing gradient variance (Chen et al., 2020).

[1] Proteins are biological macromolecules responsible for the majority of functions within living cells, represented by, linear chains of "amino acids," over which we perform MLM style pre-training

We test our system on the TAPE protein pre-training benchmark system (Rao et al., 2019), achieving modest improvements over comparably trained random pre-training benchmarks, though further development will be needed to ensure this method is worth the increased computational cost. All our code will also be made public after review.

2. Methods

Traditionally, to learn a masked language model $\mathcal{M}_{\text{PT}}$, we begin with a large, unlabeled corpus of sequences and form training examples by *randomly* choosing a fraction (e.g., 15% in the case of BERT (Devlin et al., 2019)) of tokens to "mask." The "masked" tokens are traditionally noised according to three masking strategies in an 80-10-10 ratio: `[MASK]` Masking, in which the masked tokens are replaced with a sentinel, out-of-vocabulary token `[MASK]`; Keep-original masking, in which the masked tokens are kept as the original token, but the model is still scored on its ability to form a correct language model prediction for these tokens, and Replace Masking, in which the masked tokens are replaced with another random valid token from the vocabulary. Given a sentence from the input corpus masked according to this process, the pre-training model $\mathcal{M}_{\text{PT}}$ is then tasked to recover the original sentence.

In this work, we generalize this paradigm by introducing a model $\mathcal{M}_{\text{noiser}}$ to decide which tokens to mask and how. $\mathcal{M}_{\text{noiser}}$ is broken down into two parts: a sequence-to-sequence model yielding masking probabilities, and a budgeted differentiable sampling module to actually choose the mask. The whole model is trained end to end via an adversarial approach, such that it learns to mask tokens in a way that makes it most difficult for $\mathcal{M}_{\text{PT}}$ to correct. This setup is shown pictorially in Figure 1. Our full adversarial masker $\mathcal{M}_{\text{noiser}}$ algorithm is given in pseudocode in the Appendix, Algorithms 2,1, Section A, but we will detail several points further here.

Sequence-to-sequence model The sequence to sequence model $\mathcal{M}^{(\text{seq})}_{\text{noiser}}$ (in this work implemented with a gated reucrrent unit (GRU) (Cho et al., 2014) architecture), ingests a sequence of (unmasked) amino acids $\boldsymbol{x}$ and returns a sequence of unnormalized, multi-dimensional masking scores, $\boldsymbol{s} = \mathcal{M}^{(\text{seq})}_{\text{noiser}}$. $\boldsymbol{s}$ contains, per-token, a score for (1) whether or not to mask that token at all (we will denote this the *any-mask* score), and (2) a conditional score for each masking option, including `[MASK]` masking, keep-original masking, and replace masking, with $\boldsymbol{s}$ taking on a score for each possible token that could be used as the replacement mask within the vocabulary (we will refer to these collectively as the *mask-options* scores). These are then passed into our differentiable sampler to obtain a hard, masked sample.

Differentiable Sampling Options We use a slight adaption on the relaxed subset selection algorithm of Sand and Ermon (Xie and Ermon, 2019) to transform the any-mask scores of $\boldsymbol{s}$ into a vector of normalized probabilities that have two important properties: first, they will average to our masking budget constraint, μ, and, second, they will, in expectation, converge to the their sampled, one-hot approximations (i.e., these probabilities will be a member of a Concrete distribution). This algorithm frames the sampling problem as choosing a fixed-size subset of items from the valid tokens in each sentence, guided by the provided unnormalized scores, through repeated application of the Gumbel-Softmax (GS) trick (Maddison et al., 2016; Jang et al., 2016). This process is outlined in pseudo-code in Appendix Algorithm 1.

Next, to decide *how* to mask these tokens, conditioned on the token being masked, we use GS normalization directly on the mask-options scores. Finally, to obtain differen-

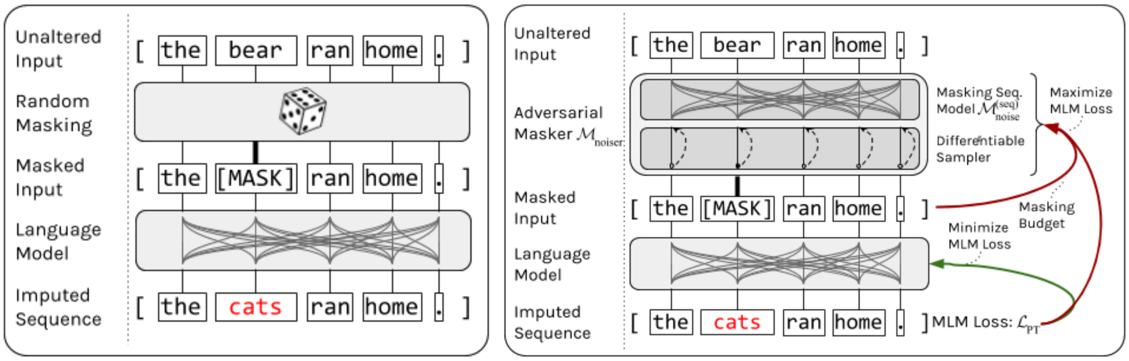

Figure 1: *Left* Traditional random masked language model pre-training. *Right* Our architecture, adversarial contrastive language model pre-training. Note one can use various options for the *Language Model*, *Masking Sequence Model*, or *Differentiable Sampler* components.

tiable one-hot outputs, we use the straight-through estimator (Bengio et al., 2013), which simply sets the gradient of the output hard-sample to be equal to the gradient of its source probability, directly, and is especially well suited for probabilities from the Concrete distribution which converge in expectation to their sampled values.

Stabilizing the learning process To stabilize the learning process and prevent the masker and MLM model from getting stuck in a local regime of masking, we simply add in additionally a small fraction of random masking, to enable the MLM model to constantly make general progress, which, in turn, forces the masking model to constantly adapt its task to be more difficult than random masking. In order to ensure that some tokens were consistently masked from both varieties, we increased the general masking rate from 15% total to 20% total, distributed as 10% random masking and 10% adversarial masking in our system.

Overall Training Algorithm We then train the system in a traditional adversarial pattern, alternating between several iterations of training the masker to *maximize* the MLM loss, followed by several iterations of training the protein encoder to *minimize* said loss. We use via iterated stochastic gradient descent (using the AdamW optimizer (Loshchilov and Hutter, 2018)), with 10 iterations of noiser training followed by 10 iterations of encoder training; these values were determined after a very brief search over possible alternates, using MLM training curve metrics and apparent learning stability to motivate that choice. Additional details about our overall training algorithm are present in supplementary materials Section A.

Data & Tasks We use 4 tasks from the TAPE benchmarking datasets (Rao et al., 2019), profiled in Table 1. For full details of these datasets and tasks, we encourage readers to refer to Rao et al. (2019).

Experiments We compare our adversarial MLM model to a random MLM model (both at 20% total masking). For our adversarial MLM, $\mathcal{M}^{(seq)}_{\text{noiser}}$ is a 3-layer, GRU with a 1024 input embedding layer and 512 output layer, and our transformer sequence model is transformer architecture matching the size of that profiled in the TAPE system. Our random

Table 1: A numerical summary of the datasets & tasks used in this work (Rao et al., 2019).

Task	Train	Val.	Test
Language Modeling	32.2M	N/A	2.1M
Secondary Structure	8678	2170	513
Remote Homololgy	12312	736	718
Fluorescence	21446	5362	27217
Stability	53679	2447	12839

MLM system encoder is an identical transformer architecture.

Models were trained on 4 NVIDIA V100 GPUs, ranging in time per model but on the order of roughly 1M encoder iterations at a batch size of 128 for approximately 80% of training followed by specialization at a larger batch size of 256 using gradient accumulation. Neither memory saving gradients nor mixed-precision training were used. During pre-training, the system was optimized via the AdamW (Loshchilov and Hutter, 2018) optimizer with weight decay of of $1e-2$ and learning rate of $1e-4$ following the conventions of TAPE.

Hyperparameter tuning was performed in a limited, manual manner on train-set MLM results for the pre-training system, and using a grid search on validation set results over batch size, learning rate, weight decays, and early stopping parameters for fine-tuning (using fixed pre-trained models).

3. Results & Discussion

All of our results across all data settings are shown in Table 2. One can see we obtain improvements over random pre-training on 3 of 4 tasks, though in all cases changes are mild. Given the increased computational cost of this style of training (approximately 2x due to masker and encoder training), these gains are likely not currently worth these minor improvements. However, they do establish that this direction of research may be a viable vehicle to improve protein pre-training, with further improvements. Several directions stand out to offer such improvements. Firstly, we believe it is likely that higher-capacity noising models would work better, though early attempts with this architecture proved too unstable to train effectively. Second, we feel we could more effectively train this system with larger batch sizes (which has been shown to offer improvements in other pre-training contexts) through the use of mixed-precision training and gradient checkpointing. Finally, the incorporation of an importance sampling re-weighting penalty on the learning objective, rather than use of partial random masking, as in the style of (Chen et al., 2020) may offer further improvements here.

Table 2: Final results for a random MLM and our adversarial MLM, reported in accuracy / amino acid for secondary structure (SS), accuracy / sequence for remote homology (RH), and Spearman correlation coefficient for fluorescence and stability. Variance is sourced over repeated FT runs only, not PT runs, as the latter would be computationally intractable.

Task	Random	Adv. (Ours)
S. S.	72.1 ± 0.1%	**72.7 ± 0.2%**
R.H.	19.2 ± 0.8%	**20.0 ± 2.0%**
Fluorescence	**0.678 ± 0.002**	0.677 ± 0.002
Stability	0.626 ± 0.044	**0.630 ± 0.009**

4. Conclusion

In this work, we design a novel adversarial, contrastive contextual embedding system for protein sequences, attaining improvements over comparable random pre-training runs on three of four tasks in the TAPE benchmark, though these improvements are minor and further work will be needed to ensure realized gains are worth the increased computational cost.

References

Ethan C. Alley, Grigory Khimulya, Surojit Biswas, Mohammed AlQuraishi, and George M. Church. Unified rational protein engineering with sequence-based deep representation learning. *Nature Methods*, 16(12):1315–1322, December 2019. ISSN 1548-7091, 1548-7105. doi: 10.1038/s41592-019-0598-1. URL http://www.nature.com/articles/s41592-019-0598-1.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv:1308.3432 [cs]*, August 2013. URL http://arxiv.org/abs/1308.3432. arXiv: 1308.3432.

Avishek Joey Bose, Huan Ling, and Yanshuai Cao. Adversarial Contrastive Estimation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1021–1032, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1094. URL https://www.aclweb.org/anthology/P18-1094.

Liang Chen, Tianyuan Zhang, Di He, Guolin Ke, Liwei Wang, and Tie-Yan Liu. Variance-reduced language pretraining via a mask proposal network. *arXiv preprint arXiv:2008.05333*, 2020.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv:1406.1078 [cs, stat]*, September 2014. URL http://arxiv.org/abs/1406.1078. arXiv: 1406.1078.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. September 2019. URL https://openreview.net/forum?id=r1xMH1BtvB.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised Cross-lingual Representation Learning at Scale. *arXiv:1911.02116 [cs]*, November 2019. URL http://arxiv.org/abs/1911.02116. arXiv: 1911.02116.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparameterization with Gumbel-Softmax. November 2016. URL https://openreview.net/forum?id=rkE3y85ee.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. September 2018. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Amy X Lu, Haoran Zhang, Marzyeh Ghassemi, and Alan Moses. Self-supervised contrastive learning of protein representations by mutual information maximization. *bioRxiv*, 2020.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. November 2016. URL https://openreview.net/forum?id=S1jE5L5gl&noteId=S1jE5L5gl.

Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Peter Chen, John Canny, Pieter Abbeel, and Yun Song. Evaluating Protein Transfer Learning with TAPE. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alchè-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 9689–9701. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9163-evaluating-protein-transfer-learning-with-tape.pdf.

Sang Michael Xie and Stefano Ermon. Reparameterizable Subset Sampling via Continuous Relaxations. pages 3919–3925, 2019. URL https://www.ijcai.org/Proceedings/2019/544.

Appendix A. Full Methods

Pseudocode algorithms for the full masker, the straight through sampler, and the full adversarial system, including masker and protein encoder, are shown in the below Algorithms:

```
Procedure rss_sampler($\boldsymbol{s}$, valid_tokens_mask, $\rho, t$)
    Result: $\boldsymbol{p}$
    Input: $\boldsymbol{s}$: Per-element selection scores, valid_tokens: Binary mask highlighting valid
           inputs within the batch, $\rho$: Desired masking fraction, $t > 0$: sampling temperature
           value.

    Initialize $\varepsilon \leftarrow 10^{-18}$, $\boldsymbol{y}_{\text{soft}} \leftarrow \mathbf{0}$, $\boldsymbol{g} \leftarrow \boldsymbol{s} - \log(-\log($Uniform(0, 1)$))$
    seq_lens $\leftarrow$ valid_tokens_mask.sum(axis=1)
    subset_sizes $\leftarrow$ round(seq_lens * $\rho$)
    subset_sizes_left $\leftarrow$ subset_sizes.expand_as(valid_tokens_mask)

    while subset_sizes_left $> 0$ do
        khot_mask $\leftarrow \max((1 - \boldsymbol{y}_{\text{soft}})$ * valid_tokens_mask$, \varepsilon)$
        $\boldsymbol{g}$ += $\log($khot_mask$)$
        $\boldsymbol{y}_{\text{soft}}$ += softmax($\boldsymbol{g}/t$, dim$= -1$)

        subset_sizes_left -= 1
        valid_tokens_mask *= (subset_sizes_left > 0).float()

    end
    return $\boldsymbol{y}_{\text{soft}}$
```

Algorithm 1: Our variable masking rate per-batch variant on the relaxed subset selection algorithm of (Xie and Ermon, 2019)

```
Input: $\rho$: masking rate; $t$: mask option temperature; mask_id: [MASK] token ID in vocabu-
       lary
Procedure run_noiser($\boldsymbol{\theta}, \boldsymbol{x}$, valid_tokens_mask)
    $\boldsymbol{s} \leftarrow \mathcal{M}^{(\text{seq})}_{\text{noiser}}(\boldsymbol{x}$, valid_tokens_mask$; \boldsymbol{\theta})$
    $\boldsymbol{p}_{\text{mask overall}} \leftarrow$ rss_sampler($\boldsymbol{s}$[:,:,0], valid_tokens_mask, $\rho, t$)
    $\boldsymbol{g}_{\text{mask type}} \leftarrow \boldsymbol{s}$[:,:,1:] $- \log(-\log($Uniform(0, 1)$))$
    $\boldsymbol{p}_{\text{mask type}} \leftarrow$softmax($\boldsymbol{g}_{mask\ type}/t$)

    $\tilde{\boldsymbol{x}} \leftarrow$ straight_through($\boldsymbol{x}, \boldsymbol{p}_{mask\ overall}, \boldsymbol{p}_{mask\ type}$)
    return $\tilde{\boldsymbol{x}}$
```

Algorithm 2: The full noising process, taking in the raw, un-masked input vector $\boldsymbol{x}$ and producing a noised version of the input suitable for pre-training an MLM. rss_sampler is shown in Algorithm 1 and straight_through is shown in the supplementary materials, Algorithm 3

```
Result: $\tilde{\boldsymbol{x}}$
Input: $\boldsymbol{x}$: one-hot encoding of the input, which can be multiplied by an embedding layer to
       produce distributed embeddings of each token. $\boldsymbol{p}_{\text{mask overall}}$: probabilities of masking
       each token in any form; $\boldsymbol{p}_{\text{mask type}}$: probabilities of each kind of masking of each
       token, or None for simple masking; mask_id: [MASK] token ID in vocabulary; $v_{\text{idx}}$:
       the start index of the vocabulary, after skipping past all control tokens (e.g., [MASK]).
Procedure straight_through($\boldsymbol{x}$, $\boldsymbol{p}_{\textit{mask overall}}$, $\boldsymbol{p}_{\textit{mask type}}$)
    mask_ANY ← where($\boldsymbol{p}_{\textit{mask overall}} > 0.5$, $\mathbf{1}$, $\mathbf{0}$)
    $\boldsymbol{x}_{\text{masked}} \leftarrow \mathbf{0}$
    if $\boldsymbol{p}_{\textit{mask type}}$ = None then
        $\boldsymbol{x}_{\text{masked}}$ += one_hot(mask_id) $+p_{\text{mask overall}}$ − detach($p_{\textit{mask overall}}$)
    else
        $\boldsymbol{M} \leftarrow$ argmax_mask($\boldsymbol{p}_{\textit{mask type}}$) $+p_{\text{mask type}}$ − detach($p_{\textit{mask type}}$)

        $\boldsymbol{m}_{\texttt{[MASK]}} \leftarrow \boldsymbol{M}$[:,:,0]
        $\boldsymbol{m}_{\text{keep}} \leftarrow \boldsymbol{M}$[:,:,1]
        $\boldsymbol{m}_{\text{replace}} \leftarrow \boldsymbol{M}$[:,:,2:]

        $\boldsymbol{x}_{\text{masked}}$ += one_hot(mask_id) * $\boldsymbol{m}_{\texttt{[MASK]}}$
        $\boldsymbol{x}_{\text{masked}}$ += $\boldsymbol{x}$ * $\boldsymbol{m}_{\text{keep}}$
        $\boldsymbol{x}_{\text{masked}}$[:,:,$v_{\text{idx}}$:] += $\boldsymbol{m}_{\text{replace}}$
    end
    $\tilde{\boldsymbol{x}} \leftarrow (1 - \texttt{mask\_ANY})\boldsymbol{x} + (\texttt{mask\_ANY})\boldsymbol{x}_{\text{masked}}$
    return $\tilde{\boldsymbol{x}}$
```

Algorithm 3: Straight through sampler, accounting for all three modes of masking.

```
Procedure run_encoder($\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}$, valid_tokens_mask)
    $\tilde{\boldsymbol{x}} \leftarrow$ run_noiser($\boldsymbol{\theta}, \boldsymbol{x}$, valid_tokens_mask)
    $\boldsymbol{p}_{\text{reconst.}} \leftarrow \mathcal{M}_{\text{PT}}(\tilde{\boldsymbol{x}}, \texttt{valid\_tokens\_mask}; \boldsymbol{\phi}_{\text{PT}})$
    return $\mathcal{L}(\boldsymbol{p}_{\text{reconst.}}, \boldsymbol{x}, \texttt{valid\_tokens\_mask})$

Procedure update_noiser($\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}$, valid_tokens_mask)
    return SGD(-run_encoder($\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}$, valid_tokens_mask))
Procedure update_encoder($\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}$, valid_tokens_mask)
    return SGD(run_encoder($\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}$, valid_tokens_mask))
```

Algorithm 4: Noiser & Encoder Update Steps. In practice we use more the more advanced AdamW SGD variant.

```
Result: $\boldsymbol{\theta}^*$, $\boldsymbol{\phi}_{\text{PT}}^*$
Input: $n_{\text{noiser}}$: # of noiser iterations per cycle
Input: $n_{\text{encoder}}$: # of encoder iterations per cycle
Initialize $\boldsymbol{\theta}^{(0)}, \boldsymbol{\phi}_{\text{PT}}^{(0)}$ randomly  Initialize $i \leftarrow 1$, mode $\leftarrow$ PRE_TRAINING_NOISING
 while Not Converged do
    $\boldsymbol{x}$, valid_tokens_mask $\leftarrow$get_batch($i$)
     if mode = PRE_TRAINING_NOISING then
        $\boldsymbol{\theta}^{(i+1)} \leftarrow$ update_noiser($\boldsymbol{\theta}^{(i)}, \boldsymbol{\phi}_{\text{PT}}^{(i)}, \boldsymbol{x}$, valid_tokens_mask)
         if $i$ mod $n_{noiser} = 0$ then
            mode $\leftarrow$ PRE_TRAINING_ENCODING
        end
    else
        $\boldsymbol{\phi}_{\text{PT}}^{(i+1)} \leftarrow$ update_encoder($\boldsymbol{\theta}^{(i+1)}, \boldsymbol{\phi}_{\text{PT}}^{(i)}, \boldsymbol{x}$, valid_tokens_mask)
         if $i$ mod $n_{encoder} = 0$ then
            mode $\leftarrow$ PRE_TRAINING_NOISING
        end
    end
    $i \leftarrow i + 1$

end
return $\boldsymbol{\theta}^{(i)}, \boldsymbol{\phi}_{\text{PT}}^{(i)}$
```

Algorithm 5: Our adversarial contrastive training algorithm