BERTAC: Enhancing Transformer-based Language Models with Adversarially Pretrained Convolutional Neural Networks

Jong-Hoon Oh[§] **Ryu Iida**[§¶] **Julien Kloetzer**[§] **Kentaro Torisawa**[§¶]

Data-driven Intelligent System Research Center (DIRECT),
National Institute of Information and Communications Technology (NICT)[§]
Graduate School of Science and Technology, NAIST[¶]
{rovellia, ryu.iida, julien, torisawa}@nict.go.jp

Abstract

Transformer-based language models (TLMs), such as BERT, ALBERT and GPT-3, have shown strong performance in a wide range of NLP tasks and currently dominate the field of NLP. However, many researchers wonder whether these models can maintain their dominance forever. Of course, we do not have answers now, but, as an attempt to find better neural architectures and training schemes, we pretrain a simple CNN using a GAN-style learning scheme and Wikipedia data, and then integrate it with standard TLMs. We show that on the GLUE tasks, the combination of our pretrained CNN with ALBERT outperforms the original ALBERT and achieves a similar performance to that of SOTA. Furthermore, on open-domain QA (Quasar-T and SearchQA), the combination of the CNN with ALBERT or RoBERTa achieved stronger performance than SOTA and the original TLMs. We hope that this work provides a hint for developing a novel strong network architecture along with its training scheme. Our source code and models are available at https://github.com/nict-wisdom/bertac.

1 Introduction

Transformer-based language models (TLMs) such as BERT (Devlin et al., 2019), ALBERT (Lan et al., 2020), and GPT-3 (Brown et al., 2020) have shown that large-scale self-supervised pretraining leads to strong performance on various NLP tasks. Many researchers have used TLMs for various downstream tasks, possibly as subcomponents of their methods, and/or they have focused on scaling up TLMs or improving their pretraining schemes. As a result, other architectures like Recurrent Neural Networks (RNN) (Hochreiter and Schmidhuber, 1997; Cho et al., 2014) and Convolutional Neural Networks (CNN) (LeCun et al., 1999) are fading away. In this work, we propose a method for improving TLMs by integrating a simple conventional CNN to them. We pretrained this CNN on Wikipedia using a Generative Adversarial Network (GAN) style training scheme (Goodfellow et al., 2014), and then combined it with TLMs. Oh et al. (2019) similarly used GAN-style training to improve a QA model using a CNN, but their training scheme was applicable only to QA-specific datasets. On the other hand, similarly to TLM, our proposed method for training the CNN is independent of specific tasks. We show that the combination of this CNN with TLMs can achieve higher performance than that of the original TLMs on publicly available datasets for several distinct tasks. We hope that this gives an insight into how to develop novel strong network architectures and training schemes.

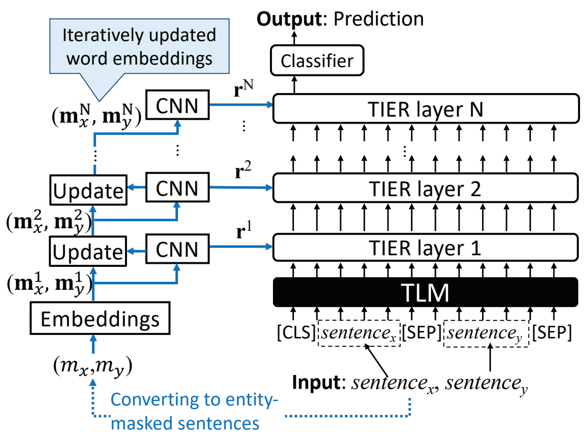
Iteratively updated word embeddings
Output: Prediction
Classifier
$(\mathbf{m}_x^N, \mathbf{m}_y^N)$
CNN
$\mathbf{r}^N$
TIER layer N
Update
CNN
$\mathbf{r}^2$
TIER layer 2
$(\mathbf{m}_x^2, \mathbf{m}_y^2)$
Update
CNN
$\mathbf{r}^1$
TIER layer 1
$(\mathbf{m}_x^1, \mathbf{m}_y^1)$
TLM
Embeddings
[CLS] $sentence_x$ [SEP] $sentence_y$ [SEP]
(m_x, m_y)
Input: $sentence_x$, $sentence_y$
Converting to entity-masked sentences

Figure 1: Overall architecture of BERTAC under the setting of classification for a two-sentence input ($sentence_x$ and $sentence_y$).

We call our combination of a TLM and a CNN *BERTAC* (BERT-style TLM with an Adversarially pretrained Convolutional neural network). Its architecture is illustrated in Fig. 1. We do not impose any particular restriction on the TLM in BERTAC, so any TLM, ALBERT (Lan et al.,

Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, pages 2103–2115
August 1–6, 2021. ©2021 Association for Computational Linguistics

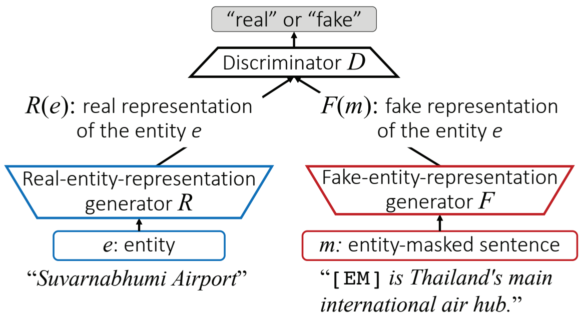
"real" or "fake"
Discriminator D
$R(e)$: real representation of the entity e
$F(m)$: fake representation of the entity e
Real-entity-representation generator R
Fake-entity-representation generator F
e: entity
m: entity-masked sentence
"Suvarnabhumi Airport"
"[EM] is Thailand's main international air hub."

Figure 2: GAN-style pretraining of CNNs. The discriminator D takes either a real representation generated by R or a fake representation generated by F as its input and then it predicts whether the input is a real or fake representation.

s_1	Suvarnabhumi Airport$_{e_1}$ is Thailand's main international air hub.
m_1	[EM] is Thailand's main international air hub.
e_1	Suvarnabhumi Airport

Table 1: Example of an entity-masked sentence (m_1) and the original sentence (s_1)

2020) or RoBERTa (Liu et al., 2019) for example, can be used as a subcomponent of BERTAC.

We used the CNN to compute representations of a slightly modified version of the input given to a TLM. To integrate these representations with those of the TLM, we stacked on top of the TLM several layers of *Transformers for Integrating External Representation (TIERs)*, which are our modified version of normal transformers (Vaswani et al., 2017). A TIER has the same architecture as that of a normal transformer encoder except for its attention: we replace the transformer's self-attention with an attention based on the representation provided by the CNN. We expect that, by keeping the basic architecture of transformer encoders, the CNN's representations can be integrated more effectively with the TLM's original representations.

We pretrained the CNN using a GAN-style training scheme in order to generate representations of sentences rather freely without the constraint of token embedding prediction in the masked language modeling used for TLMs, as we explain later. For the training, we used masked sentences autogenerated from Wikipedia. As in the masked language modeling, neither human intervention nor downstream task-specific hacking is required. As illustrated in Fig. 2, the GAN-style training requires three networks, namely, a discriminator D and two CNN-based generators R and F. Once the training is done, we use the generator F as CNN in BERTAC. The training data consists of pairs of an entity mention and a sentence in which the entity mention is masked with a special token [EM]. For example, the entity-masked sentence m_1 in Table 1 is obtained by masking the entity mention e_1, "Suvarnabhumi Airport," in the original text s_1. The network F generates a vector representation of the masked sentence (m_1), while R produces a representation of the masked entity (e_1). The discriminator D takes representations generated by either R or F as the input, and it predicts which generator actually gave the representation.

In the original GAN, a generator learns to generate an artificial image from random noise so that the resulting artificial image is indistinguishable from given real images. By analogy, we used an *entity-masked sentence* as "random noise" and a *masked entity* as a "real image." In our GAN-style training, we regard the vector representation of a masked entity given by generator R as a *real representation* of the entity (or the representation of the "real image" in the above analogy). On the other hand, we regard the representation of the masked sentence, generated by F, as a *fake representation* of the entity (or the representation of the "artificial image" generated from the "random noise" in the above analogy). This representation is deemed *fake* because the entity is masked in the masked sentence, and F does not know what the entity is exactly. During the training, F should try to deceive the discriminator D by mimicking the real representation and generating a fake representation that is indistinguishable from the real representation of the entity generated by R. On the other hand, R and D, as a team, try to avoid being mimicked by F and also to make the mimic problem harder for F. If everything goes well, once the training is over, F should be able to generate a fake representation of the entity that is similar to its real representation.

An interesting point is that F's output can be interpreted in two ways: it is a representation of a *masked sentence* because it is computed from the sentence, and at the same time it is a representation of the *masked entity* because it is *indistinguishable* from R's representation of the entity. This duality suggests that F's output can be seen as a representation of the *entire sentence*.

We exploit F as a CNN in BERTAC as follows: first, we use F to compute a representation of a masked version of the sentence originally given as input to a TLM. The entity mention to be masked is chosen by simple rules and, if the input consists of multiple sentences, we generate a representation of each (masked) input sentence and concatenate these together into a single one. Then, this representation is integrated to the output of the TLM through multiple TIER layers.

Our GAN-style pretraining is conceptually similar to TLM pretraining with masked language modeling (predicting what a masked word in a sentence should be). However, it was designed to pretrain a model that is able to rather *freely generate entity representations* without strongly sticking to the prediction of token embeddings. Our hypothesis is that such freely generated representations may be useful for improving the performance of downstream tasks. Moreover, we assumed that using multiple text representations computed from different perspectives (i.e., predicting token embeddings and freely generating entity representations) would help to improve the performance of downstream tasks.

In our experiments, we show that for the GLUE tasks (Wang et al., 2018), BERTAC's average performance on the development set was 0.7% higher than that of ALBERT, which was used as a subcomponent of BERTAC, leading to a performance on the test set comparable to that of SOTA (90.3% vs 90.8% (SOTA)). It also outperformed the SOTA method of open-domain QA (Chen et al., 2017) on Quasar-T (Dhingra et al., 2017) and SearchQA (Dunn et al., 2017) using either ALBERT or RoBERTa. We also compared our method with alternative models using a CNN pretrained in a self-supervised (non GAN-style) manner to directly predict embeddings of the entity mentions. Consequently, we confirmed that our method worked better: only the CNN trained by our GAN-style pretraining gave significant performance improvement over base TLMs.

Note that the computational overhead of BERTAC is reasonably small. It took 20 hours with 16 GPUs to pretrain a single CNN model and 180 hours for the nine models tested with different parameter settings in this work (cf., 480 hours with 96 GPUs for pretraining DeBERTa (He et al., 2021), for example). Moreover, once pretrained, the CNN models can be re-used for various downstream tasks and combined with various TLMs, including potentially future ones. As for the parameter number, BERTAC had just a 14% increase in parameters when ALBERT-xxlarge was used as its base TLM (268 M parameters for BERTAC vs. 235 M for ALBERT-xxlarge). We confirmed from these results that BERTAC could improve pretrained TLMs with reasonably small computational overhead.

The code and models of BERTAC are available at https://github.com/nict-wisdom/bertac.

2 Related Work

Pretraining TLMs with entity information: There have been attempts to explicitly learn entity representation from text corpora using TLMs (He et al., 2020; Peters et al., 2019; Sun et al., 2020; Wang et al., 2020a; Xiong et al., 2020; Zhang et al., 2019). Our proposed method is a complementary alternative to these existing methods in the sense that entity representations are integrated into TLMs via CNNs and not directly produced by the TLMs.

Fine-tuning TLMs with external resources or other NNs: Yang et al. (2019a) and Liu et al. (2020) have used knowledge graphs for augmenting TLMs with entity representations during fine-tuning. Unlike these approaches, BERTAC uses unstructured texts rather than clean structured knowledge, such as knowledge graphs, to adversarially train a CNN. Other previous works have proposed combining CNNs or RNNs with BERT for NLP tasks (Lu et al., 2020; Safaya et al., 2020; Shao et al., 2019; Zhang et al., 2020), but their use of CNNs/RNNs was task-specific, so their models were not directly applicable to other tasks.

Adversarial learning for improving TLMs: Oh et al. (2019) proposed a CNN-based *answer representation generator* for QA that can *guess* the vector representation of answers from given why-type questions and answer passages. The generator was trained in a GAN-style manner using QA datasets. We took inspiration from their adversarial training scheme to train task-independent representation generators from unsupervised texts (i.e., Wikipedia sentences in which an entity was masked in a cloze-test style).

ELECTRA (Clark et al., 2020) also employed an adversarial technique (not a GAN) to pretrain two TLMs: A generator was trained to perform masked language modeling and a discriminator was trained to distinguish tokens in the training data from tokens replaced by the generator. On downstream tasks, only the discriminator was fine-tuned. In BERTAC, the GAN-style pretraining was applied only to the CNN, thus reducing the training cost. Furthermore, the CNN can be combined easily with any available TLM, even potentially future ones, without having to re-do the pretraining. In this work, we show that BERTAC outperformed ELECTRA on the GLUE task.

Vernikos et al. (2020) proposed a method that used an adversarial objective and an adversarial classifier for regularizing the fine-tuning process of TLMs, inspired by adversarial learning for domain adaptation (Ganin et al., 2016). Our work uses a GAN-style training scheme only for pretraining CNNs, not for fine-tuning TLMs.

3 Pretraining of CNNs

This section describes the training data and training algorithm for our CNN.

3.1 Training data

We pretrained our CNN with an entity-masked version of Wikipedia sentences. WikiExtractor[1] was used to extract, from the English Wikipedia[2], sentences that have at least one entity mention, i.e., an entity with an internal Wikipedia link. Then we randomly selected one entity mention e_i in each sentence and generated an entity-masked sentence m_i by replacing the entire selected mention with `[EM]`. For example, we generated the masked sentence m_1, "`[EM]` is Thailand's main international air hub," (in Table 1) by replacing the entity mention e_1, *Suvarnabhumi Airport*, in the sentence s_1, "Suvarnabhumi Airport is Thailand's main international air hub," with `[EM]`. We obtained about 43.3 million pairs of an entity mention and a masked sentence ($\{(e_i, m_i)\}$) in this way and used 10% of them (randomly sampled) as the pretraining data for our CNN.

3.2 GAN-style pretraining

As illustrated in Fig. 2, the adversarial training is done using three subnetworks: R (*real-entity-representation generator*), F (*fake-entity-representation generator*), and D (*discriminator*). R and F are CNNs with average pooling and D is a feedforward neural network. Once the training is done, we use the generator F as CNN in BERTAC. In the training, we regard the representation of a masked entity output by generator R as a *real representation of the entity* that the fake-entity-representation generator F should mimic. F is trained so that, taking an entity-masked sentence as its input, it can generate a representation of the masked entity mention (called a *fake representation of the entity* in this work) that D cannot distinguish from the real representation. The representation generated by F is *fake* in the sense that the entity mention is masked in the input sentence and F cannot know what it is exactly.

As mentioned in the Introduction, our GAN-style pretraining was designed to train a model capable of *freely generating entity representations*. We assumed that using multiple text representations computed from different perspectives (i.e., prediction of token embeddings in TLMs and generation of entity representations in our CNN) would help to improve the performance of downstream tasks.

Algorithm 1: Adversarial Training Scheme

```
Input: Training examples $\{(e,m)\}$, training epochs $t$,
       mini-batch steps $b$, mini-batch size $n$
Output: Real representation generator $R$, fake
        representation generator $F$, discriminator $D$
1  $j \leftarrow 1$
2  Initialize $\theta_R$, $\theta_F$, and $\theta_D$ (parameters of $R$, $F$, and $D$)
     with random weights
3  while $j \leq t$ do
4      $k \leftarrow 1$
5      while $k \leq b$ do
6          Sample mini-batch of $n$ examples $\{(e_i,m_i)\}_{i=1}^{n}$
7          Generate word embeddings $\{(\mathbf{e}_i,\mathbf{m}_i)\}_{i=1}^{n}$ of the
             examples.
8          Update $D$ and $R$ by ascending their stochastic
             gradient:
             $\nabla_{\theta_D,\theta_R} \frac{1}{n}\sum_{i=1}^{n}[\log D(R(\mathbf{e}_i)) + \log(1 - D(F(\mathbf{m}_i)))]$
9          Update $F$ by descending its stochastic gradient:
             $\nabla_{\theta_F} \frac{1}{n}\sum_{i=1}^{n}\log(1 - D(F(\mathbf{m}_i)))$
10         $k \leftarrow k + 1$
11     end
12     $j \leftarrow j + 1$
13 end
```

For each pair of an entity mention (e_i) and an entity-masked sentence (m_i) in the training data, we first generate two matrices of word embeddings $\mathbf{e}_i$ and $\mathbf{m}_i$ using word embeddings pretrained on Wikipedia with fastText (Bojanowski et al., 2017). Then, R and F generate, respectively, a

[1]https://github.com/samuelbroscheit/wikiextractor-wikimentions

[2]We used the September 2020 version.

real entity representation from $\mathbf{e}_i$ and a *fake entity representation* from $\mathbf{m}_i$. Finally, they are given to D, which is a feed-forward network that judges whether F or R generated the representations, i.e., whether the representations are *real* or *fake*, using sigmoid outputs by the final logistic regression layer.

The pseudo code of the training scheme is given in **Algorithm 1**. The training proceeds as follows: R and D as a team try to avoid the possibility that D misjudges F's output (i.e., a fake entity representation) as a real entity representation. More precisely, R and D are trained so that D can correctly judge the representation $R(\mathbf{e}_i)$ given by generator R as *real* (i.e., $D(R(\mathbf{e}_i)) = 1$) and the representation $F(\mathbf{m}_i)$ given by generator F as *fake* (i.e., $D(F(\mathbf{m}_i)) = 0$). Therefore, the training is carried out with the objective of maximizing $\log D(R(\mathbf{e}_i)) + \log\big(1 - D(F(\mathbf{m}_i))\big)$ (line 8 in **Algorithm 1**). On the other hand, F tries to generate representation $F(\mathbf{m}_i)$ so that D judges it as *real* (i.e., $D(F(\mathbf{m}_i)) = 1$). Thus, F is trained to minimize $\log\big(1 - D(F(\mathbf{m}_i))\big)$ (line 9 in **Algorithm 1**). This minmax game is iterated for the pre-specified t training epochs.

3.3 Pretraining settings

We extracted 43.3 million pairs of an entity mention and a masked sentence from Wikipedia and randomly sampled 10% of them to use as training data (4.33 million pairs, around 700 MB in file size). We used word-embedding vectors in 300 dimensions (for 2.5 million words) pretrained on Wikipedia using fastText (Bojanowski et al., 2017). The embedding vectors were fixed during the training.

We set the training epochs to 200 ($t = 200$ in **Algorithm 1**) and did not use any early-stopping technique. We chose $t = 200$ from the results of our preliminary experiments in which we used 10% of the training data and set training epochs t to either of 100, 200, or 300; the loss robustly converged for $t = 200$ and $t = 300$, and thus the earliest point $t = 200$ was chosen. We used the RmsProp optimizer (Tieleman and Hinton, 2012) with a batch size of 4,000 ($n = 4,000$ and $b = 1,084$ in **Algorithm 1**) and a learning rate of 2e-4. We trained nine CNN models with all combinations of the filter's window sizes $\in$ {"1,2,3", "2,3,4", "1,2,3,4"} and number of filters $\in \{100, 200, 300\}$ for the generators F and R. All of the weights in the CNNs were initialized using He's method (He et al., 2015). We used a logistic regression layer with sigmoid outputs as discriminator D. The training of a single CNN model took around 20 hours using 16 Nvidia V100 GPUs with 32 GB of memory (180 hours in total for the nine models).

We tested all nine CNN models for BERTAC in our GLUE and open-domain QA experiments (Section 5). For each task, the parameters inside the CNNs (as well as the word-embedding vectors) were fixed during the fine-tuning of BERTAC.

4 BERTAC

As illustrated in Fig. 1, *BERTAC* (BERT-style TLM with an Adversarially pretrained Convolutional neural network) incorporates the representation provided by the adversarially pretrained CNN to the representation generated by a TLM. For the integration, we use several layers of TIERs (Transformers for Integrating External Representation) stacked on top of the TLM.

4.1 CNN in BERTAC

For simplicity, we describe how the CNN is integrated in BERTAC using the task of recognizing textual entailment (RTE) as an example. BERTAC for the RTE task takes two sentences s_x and s_y as input and predicts whether s_x entails s_y. First, we explain how the adversarially pretrained CNN (generator F in Section 3.2) generates the representation of the two input sentences. We regard the longest common noun phrase[3] of the two sentences as the entity mention to be masked and create entity-masked sentences m_x and m_y from s_x and s_y by masking the noun phrase with `[EM]` (we use $m_x = s_x$ and $m_y = s_y$ if no common noun phrase is found). Then each of the masked sentences m_x and m_y is given to the CNN. Our expectation here is that the CNN generates similar representations from the masked sentences if they have an entailment relation and that this helps to recognize the entailment relation.

Note that the CNN in BERTAC is connected to several TIER layers and that, as shown in Fig. 1, its input is iteratively updated so that it provides updated representations to the TIER layers. Let $\mathbf{m}_x^i \in \mathbb{R}^{|m_x| \times d_w}$ and $\mathbf{m}_y^i \in \mathbb{R}^{|m_y| \times d_w}$ be the matrices of word embeddings of m_x and m_y given

[3]For single-sentence tasks such as CoLA (Wang et al., 2018), we regard the longest noun phrase in a sentence as an entity.

to the CNN connected to the i-th TIER layer, where d_w is the dimension of a word embedding. We denote the representation generated by the CNN when the matrix of word embeddings $\mathbf{m}$ was used as the input by $CNN(\mathbf{m})$. The i-th TIER layer is given the concatenation of the two CNN representations of m_x and m_y, $\mathbf{r}^i = [\mathbf{r}^i_x, \mathbf{r}^i_y] \in \mathbb{R}^{2\times d_e}$, where $\mathbf{r}^i_x = CNN(\mathbf{m}^i_x) \in \mathbb{R}^{d_e}$, $\mathbf{r}^i_y = CNN(\mathbf{m}^i_y) \in \mathbb{R}^{d_e}$ and d_e is the dimension of the CNN representation. Note that, for single-sentence tasks, $\mathbf{r}^i = \mathbf{r}^i_x$, the CNN representation of m_x, is given to the TIER layers.

The initial matrices of word embeddings $\mathbf{m}^1_x$ and $\mathbf{m}^1_y$ are obtained using the fastText word embeddings (Bojanowski et al., 2017), the same as that used in our adversarial learning. Then, the updated input matrices $\mathbf{m}^{i+1}_x$ and $\mathbf{m}^{i+1}_y$ for the $(i+1)$-th CNN are obtained from the i-th input matrices $\mathbf{m}^i_x$ and $\mathbf{m}^i_y$ as described below. For the word embedding $\mathbf{m}^i_{x,j}$ of the j-th word in m_x, we compute its bilinear score to $\mathbf{r}^i_x$ (Sutskever et al., 2009):

$$\bar{\mathbf{m}}^i_{x,j} = \text{softmax}_j(\mathbf{m}^{i\mathsf{T}}_x \mathbf{B}^i_x \mathbf{r}^i_x)\mathbf{m}^i_{x,j},$$

where $\mathbf{B}^i_x \in \mathbb{R}^{d_w \times d_e}$ is a trainable matrix and $\text{softmax}_j(\mathbf{v})$ denotes the j-th element of the *softmaxed* vector of $\mathbf{v}$. The bilinear score indicates how much the corresponding token should be highlighted as one associated with the CNN representation $\mathbf{r}^i_x$ during the update process. We expect that this allows the CNN in the next TIER layer to generate further refined representations with the updated embeddings.

We then compute word embeddings $\mathbf{m}^{i+1}_x$ in a highway network manner (Srivastava et al., 2015) as follows:

$$\mathbf{m}^{i+1}_x = \mathrm{H}_x(\bar{\mathbf{m}}^i_x) \odot \mathrm{T}_x(\mathbf{m}^i_x) + \mathbf{m}^i_x \odot (1 - \mathrm{T}_x(\mathbf{m}^i_x)),$$

where $\mathrm{H}_x(\mathbf{m}^i_x) = \mathbf{W}^i_h \mathbf{m}^i_x + \mathbf{b}^i_h$, $\mathrm{T}_x(\mathbf{m}^i_x) = \sigma(\mathbf{W}^i_t \mathbf{m}^i_x + \mathbf{b}^i_t)$, σ is the sigmoid function, $\odot$ represents the element-wise product, and $\mathbf{W}^i_h$, $\mathbf{W}^i_t$, $\mathbf{b}^i_h$, and $\mathbf{b}^i_t$ are layer-specific trainable parameters. $\mathbf{m}^{i+1}_y$ is also computed from $\mathbf{m}^i_y$ and $\mathbf{r}^i_y$ in the same way. During the fine-tuning of BERTAC for downstream tasks, we fix the parameters of the pretrained CNN but train these parameters for updating CNN's input alongside those of TLMs and TIERs.

4.2 Transformers for integrating external representation (TIERs)

As explained in the Introduction, the main difference between a TIER and a normal transformer encoder (Vaswani et al., 2017) lies in the attention mechanism. In the TIER attention mechanism, the query representation, which is one of the three inputs of the transformer's self-attention, is replaced with the representation given by the CNN.

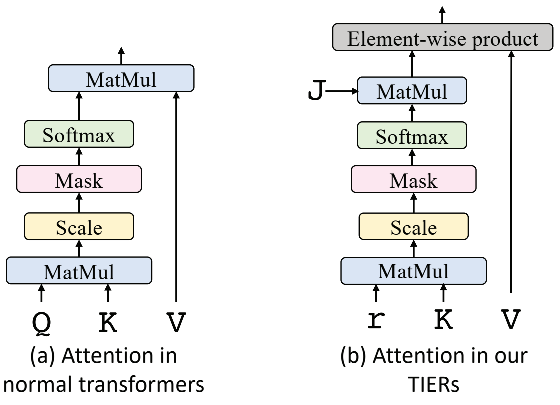

Figure 3: Attention in normal transformers and TIERs

Fig. 3 shows the difference between the TIERs' attention computation and that of normal transformers. Attention in normal transformers is computed in the following way:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\mathsf{T}}{\sqrt{d_k}})\mathbf{V}.$$

$\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are query, key, and value matrices in $\mathbb{R}^{l_k \times d_k}$, where l_k is the length of an input sequence and d_k is a dimension of keys. $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ all come from the same representation of the token sequence provided from the previous transformer layer. The attention should specify how much the corresponding tokens in $\mathbf{V}$ should be highlighted, so we designed ours in the same way.

In TIERs, we use the following attention. We basically replace the matrix $\mathbf{Q}$ with the CNN's representation $\mathbf{r} \in \mathbb{R}^{u \times d_k}$ while keeping the original $\mathbf{K}$ and $\mathbf{V}$, where u is the number of sentences in the input of the model ($u \in \{1, 2\}$ in this paper).

$$\text{Attention}(\mathbf{r}, \mathbf{K}, \mathbf{V}) = (\text{softmax}(\frac{\mathbf{r}\mathbf{K}^\mathsf{T}}{\sqrt{d_k}}))^\mathsf{T} \mathbf{J}_{u,d_k} \odot \mathbf{V}.$$

Since $\mathbf{r}$ is a matrix with a different size from $\mathbf{Q}$, we needed to adapt the attention computation. We first multiply $\mathbf{r}$ to $\mathbf{K}^\mathsf{T}$, and then its softmaxed results are converted into a $l_k \times d_k$ dimensional matrix using the all-one matrix $\mathbf{J}_{u,d_k} \in \mathbb{R}^{u \times d_k}$. Let the resulting matrix be $\mathbf{A} = (\text{softmax}(\frac{\mathbf{r}\mathbf{K}^\mathsf{T}}{\sqrt{d_k}}))^\mathsf{T} \mathbf{J}_{u,d_k} \in \mathbb{R}^{l_k \times d_k}$. We apply the attention score to $\mathbf{V}$ by using the element-wise product between matrices: $\mathbf{A} \odot \mathbf{V}$.

In addition, the actual CNN's representation $\mathbf{r}_{CNN} \in \mathbb{R}^{u \times d_e}$ given by our CNNs usually have a size that does not match the size requirement for $\mathbf{r}$. Thus, we convert it to $\mathbf{r} \in \mathbb{R}^{u \times d_k}$, a d_k-column matrix as follows: $\mathbf{r} = \mathbf{r}_{CNN}\mathbf{W} + \mathbf{b}$, where $\mathbf{W} \in \mathbb{R}^{d_e \times d_k}$ and $\mathbf{b}$ are trainable.

5 Experiments

We tested our model on GLUE and on open-domain QA. In this section, we report the results.

5.1 GLUE

GLUE (Wang et al., 2018) is a multi-task benchmark composed of nine tasks including two single-sentence tasks (CoLA and SST-2) and seven two-sentence tasks of similarity/paraphrase tasks (MRPC, QQP, and STS-B) and natural language inference tasks (MNLI, QNLI, RTE, and WNLI). Following the previous work of ALBERT (Lan et al., 2020), we performed single-task fine-tuning for each task under the following settings: single-model for the development set and ensemble for test set submissions. As in Liu et al. (2019) and Lan et al. (2020), we report the performance on the development set for each task by averaging over five runs with different random initialization seeds. As in Lan et al. (2020), for test set submissions, we fine-tuned the models for the RTE, STS-B, and MRPC tasks by initializing them with the fine-tuned MNLI single-task model, and we also used task-specific modification for CoLA and WNLI to improve scores (see Appendix A for details). We explored ensemble settings between 6 and 30 models per task for our test set submission.

5.1.1 Fine-tuning details of BERTAC for GLUE

We used ALBERT-xxlarge-v2 (Lan et al., 2020) as the pretrained TLM. As hyperparameters for BERTAC, for each task we tested learning rates $\in$ {8e-6, 9e-6, 1e-5, 2e-5, 3e-5}, a linear warmup for the first 6% of steps followed by a linear decay to 0, a maximum sequence length of 128, and all nine CNNs pretrained with different filter settings. We set the batch size to 128 for MNLI and QQP and 16 for the other tasks. Furthermore, we trained our model with the following set of training epochs: {1,2,3,4,5} for MNLI, QQP, and QNLI, {6,7,8,9,10} for CoLA, MRPC, RTE, SST-2, and STS-B, and {90,95,100,105,110} for WNLI. We set the number of TIER layers to 3 after preliminary experiments. See Table 9 in Appendix B for a summary of the hyperparameters tested in the GLUE experiments.

During the fine-tuning of BERTAC, the parameters inside the CNNs (as well as word embeddings of fastText) were fixed as explained in Section 3.3, while those used to update the input to the CNNs were optimized. For each task, we selected the pretrained CNN (out of nine) and the BERTAC hyperparameters that gave the best performance on the development data.

5.1.2 Results

Table 2 shows the results of eight tasks on the GLUE development set: all of them are single-model results. Our BERTAC consistently outperformed the previous TLM-based models over seven tasks, except for QQP, and, as a result, showed the best average performance on the development set. Crucially, our model improved the average performance around 0.7% over ALBERT, the base TLM in our model. This indicates the effectiveness of adversarially trained CNNs and TIERs in BERTAC. The test set results obtained from the GLUE leaderboard are summarized in Table 3. Our model showed comparable performance to SOTA, DeBERTa/TuringNLRv4, and achieved state-of-the-art results on 3 out of 9 task. It also showed better performance than ALBERT, our base TLM, in most tasks.

To investigate whether our GAN-style pretraining of CNNs contributed to the performance improvement, we also tested the following alternative training schemes for the CNN used in BERTAC.

Self-supervised CNN: We pretrained the CNN to generate representations of a masked sentence in a self-supervised way as follows: For an entity mention e and an entity-masked sentence m in the training data (Section 3.1), the CNN generates a representation $\mathbf{r}$ from the masked sentence trying to minimize MSE (mean squared error) between $\mathbf{r}$ and the entity mention's representation $\mathbf{e}$ (average word embedding of all tokens in e).

Randomly initialized CNN: We did not pretrained the CNNs, but trained them alongside the TLMs during the fine-tuning of BERTAC (the CNNs were randomly initialized).

We trained both the self-supervised and randomly initialized CNNs using the same hyperparameter settings as GAN-style CNNs (see Section 3.3). We confirm from the results in Table 4

Models	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	Avg.
RoBERTa$_{\text{LARGE}}$	90.2/90.2	94.7	92.2	86.6	96.4	90.9	68.0	92.4	88.9
XLNET$_{\text{LARGE}}$	90.8/90.8	94.9	92.3	85.9	97.0	90.8	69.0	92.5	89.2
ELECTRA$_{\text{LARGE}}$	90.9/-	95.0	**92.4**	88.0	96.9	90.8	69.1	92.6	89.5
ALBERT$_{\text{XXLARGE}}$	90.8/-	95.3	92.2	89.2	96.9	90.9	71.4	93.0	90.0
DeBERTa$_{\text{LARGE}}$	91.1/**91.1**	95.3	92.3	88.3	96.8	91.9	70.5	92.8	90.0
BERTAC$_{\text{XXLARGE}}$	**91.3/91.1**	**95.7**	92.3	**89.9**	**97.2**	**92.4**	**73.7**	**93.1**	**90.7**

Table 2: GLUE dev set results. The results of RoBERTa (Liu et al., 2019), XLNET (Yang et al., 2019b), ELECTRA (Clark et al., 2020), ALBERT (Lan et al., 2020), and DeBERTa (He et al., 2021) were taken from their papers. We omit the results of the WNLI task, since many previous works did not report the dev set results.

Models	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	WNLI	Score
Ensembles on test (from leaderboard as of Feb. 1, 2021)										
ALBERT	91.3/91.0	-	90.5	89.2	97.1	91.2	69.1	92.0	91.8	-
ELECTRA+Standard Tricks	91.3/90.8	95.8	90.8	89.8	97.1	90.7	71.7	92.5	91.8	89.4
ERNIE	91.4/91.0	96.6	90.9	90.9	**97.5**	91.4	74.4	92.6	**94.5**	90.4
StructBERT+TAPT	90.9/90.7	97.4	**91.0**	91.2	97.3	91.9	**75.3**	92.7	**94.5**	90.6
MacALBERT+DKM	91.3/91.1	97.8	90.6	92.0	97.0	**92.6**	74.8	92.6	**94.5**	90.7
DeBERTa/TuringNLRv4	**91.9/91.6**	**99.2**	90.8	**93.2**	**97.5**	92.0	71.5	92.6	**94.5**	**90.8**
BERTAC	91.1/**91.6**	97.9	90.6	90.4	**97.5**	91.7	72.3	**92.8**	**94.5**	90.3

Table 3: GLUE test set results. Our model for test set results incorporates task-specific modification for CoLA and WNLI to improve scores (see Appendix A for details). All results are from the GLUE leaderboard.

that only the proposed method with our GAN-style CNNs showed a higher average score than ALBERT. This suggests the effectiveness of our GAN-style pretraining scheme of CNNs.

5.2 Open-domain QA

We also tested BERTAC on open-domain QA (Chen et al., 2017) with the publicly available datasets Quasar-T (Dhingra et al., 2017) and SearchQA (Dunn et al., 2017). We used the pre-processed version[4] of the datasets provided by Lin et al. (2018), which contains passages retrieved for all questions, and followed their data split as described in Table 5.

5.2.1 BERTAC for open-domain QA

We implemented our QA model following the approach of Lin et al. (2018), which combines a *passage selector* to choose relevant passages from retrieved passages and an *answer span selector* to identify the answer span in the selected passages. For the given question q and the set of retrieved passages $P = \{p_i\}$, we computed the probability $Pr(a|q,P)$ of extracting answer span a to question q from P in the following way, and then we extracted the answer span $\hat{a}$ with the highest probability:

$$Pr(a|q,P) = \sum_i Pr(a|q,p_i)Pr(p_i|q,P),$$

[4]Available at https://github.com/thunlp/OpenQA

where $Pr(p_i|q,P)$ and $Pr(a|q,p_i)$ are computed by the passage selector and answer span selector, respectively.

We input "[CLS] question [SEP] passage [SEP]" to both the passage selector and answer span selector, where [CLS] and [SEP] are special tokens. In the passage selector, the representation of [CLS] in the top TIER layer is fed into a linear layer with a softmax, which computes the probability that the passage contains a correct answer to the question. Our BERTAC answer span selector identifies answer spans from passages by computing start and end probabilities of each token in passages, where we feed the representation of each token in the top layer of TIERs to two linear layers, each with a softmax for the probabilities (Devlin et al., 2019).

5.2.2 Training details for open-domain QA

We used all nine pretrained CNNs, as in the GLUE experiments. As pretrained TLMs, we used ALBERT-xxlarge-v2 (Lan et al., 2020) and RoBERTa-large (Liu et al., 2019). We set the learning rate to 1e-5, the number of epochs to 2, the maximum sequence length to 384, and the number of TIER layers to 3. We used a linear warmup for the first 6% of steps followed by a linear decay to 0 with a batch size of 48 for Quasar-T and 96 for SearchQA. We tested all of the pretrained CNNs and chose for each dataset the one that maximizes EM (the percentage of the predictions matching exactly one of the ground truth an-

CNNs used in BERTAC	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	Avg.
Proposed (GAN-style CNN)	**91.3/91.1**	**95.7**	**92.3**	**89.9**	**97.2**	**92.4**	**73.7**	**93.1**	**90.7**
Self-supervised CNN	91.0/90.8	95.3	91.5	88.4	96.6	90.9	71.1	93.0	89.8
Randomly initialized CNN	91.0/90.7	95.4	91.4	87.4	96.3	91.2	71.5	**93.1**	89.8
ALBERT$_{\text{XXLARGE}}$	90.8/-	95.3	92.2	89.2	96.9	90.9	71.4	93.0	90.0

Table 4: Comparison of BERTAC results in different CNN settings on GLUE dev set.

	Train	Dev	Test	#p
Quasar-T	37,012	3,000	3,000	100
SearchQA	99,811	13,893	27,247	50

Table 5: Number of questions in each dataset. #p is the number of retrieved passages for each question.

Non-TLM-based methods
OPENQA (Lin et al., 2018): An RNN-based method that jointly learns passage-selection and answer extraction.
OPENQA+ARG (Oh et al., 2019): An extension of OPENQA that additionally uses an answer representation generator (ARG) trained by adversarial learning.
TLM-based methods
WKLM (Xiong et al., 2020): This uses a TLM pretrained with a weakly supervised objective for learning Wikipedia entity information. BERT-base was used for the training.
MBERT (Wang et al., 2019): A BERT-based method that extracts answers using globally normalized answer scores across all the passages retrieved by the same question. BERT-large was used for the training.
CFORMER (Wang et al., 2020b): It uses a clustering-based sparse transformer for long-range dependency encoding. The method was trained using RoBERTa-large.

Table 6: Compared QA methods

Model	Quasar-T		SearchQA	
	EM	F1	EM	F1
OPENQA	42.2	49.3	58.8	64.5
OPENQA+ARG	43.2	49.7	59.6	65.3
WKLM$_{\text{BERT-base}}$	45.8	52.2	61.7	66.7
MBERT$_{\text{BERT-large}}$	51.1	59.1	65.1	70.7
CFORMER$_{\text{RoBERTa-large}}$	54.0	63.9	68.0	75.1
BERTAC$_{\text{RoBERTa-large}}$	55.8	63.7	71.9	77.1
BERTAC$_{\text{ALBERT-xxlarge}}$	**58.0**	**65.8**	**74.0**	**79.2**

Table 7: QA test set results. Figures of the previous works were taken from their original papers.

Model	Quasar-T		SearchQA	
	EM	F1	EM	F1
BERTAC$_{\text{ALBERT-xxlarge}}$	**58.0**	**65.8**	**74.0**	**79.2**
w/o CNN and TIER	55.6	63.5	72.7	78.0
w/o GAN-style CNN	56.1	63.9	73.1	78.4
w/o update	56.8	65.0	73.3	78.5

Table 8: Ablation test results.

swers) on the development set. See Table 10 in Appendix B for a summary of the hyperparameters tested for open-domain QA.

5.2.3 Results

We compared BERTAC with the previous works described in Table 6. Table 7 shows the performance of all of the methods. The subscripts of the TLM-based methods represent the type of pretrained TLM used by each method. All the methods were evaluated using EM and F1 score (average overlap between the prediction and gold answer). BERTAC$_{\text{ALBERT-xxlarge}}$ outperformed all of the baselines including the SOTA method (CFORMER) on both EM and F1. BERTAC$_{\text{RoBERTa-large}}$ in the same TLM setting as the SOTA method showed a better performance than SOTA except for F1 in Quasar-T. These results suggest that our framework is effective for QA tasks as well.

For ablation studies, we evaluated some variants of BERTAC$_{\text{ALBERT-xxlarge}}$: "w/o CNN and TIER," which uses ALBERT-xxlarge alone without using our CNN and TIER, "w/o GAN-style CNN," which does not use our CNN pretrained by the GAN-style training scheme but uses self-supervised CNNs (the same as used in the GLUE experiments, see Table 4), "w/o update," which does not perform layer-wise update of the CNN inputs. The results in Table 8 suggest that all of the following contributed to the performance improvement: the combination of TLMs and GAN-style CNNs, our GAN-style training of CNNs, and the layer-wise update of the CNN inputs.

6 Conclusion

We proposed *BERTAC* (BERT-style TLM with an Adversarially pretrained Convolutional neural network), a combination of a TLM and a CNN, where the CNN was pretrained using a novel GAN-style training scheme and masked sentences obtained automatically from Wikipedia. Using this CNN, we improved the performance of standard TLMs. We confirmed that BERTAC could achieve comparable performance with the SOTA and outperformed the base TLM used as a subcomponent of BERTAC in the GLUE task. We also show that BERTAC outperformed the SOTA method of open-domain QA on Quasar-T and SearchQA.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Siddhartha Brahma. 2018. Unsupervised learning of sentence representations using sequence consistency. *CoRR*, abs/1808.04217.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open–domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre–training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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.

Bhuwan Dhingra, Kathryn Mazaitis, and William W Cohen. 2017. Quasar: Datasets for question answering by search and reading. *arXiv preprint arXiv:1707.03904*.

Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Güney, Volkan Cirik, and Kyunghyun Cho. 2017. SearchQA: A new Q&A dataset augmented with context from a search engine. *CoRR*, abs/1704.05179.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario March, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 27, pages 2672–2680. Curran Associates, Inc.

Bin He, Di Zhou, Jinghui Xiao, Xin Jiang, Qun Liu, Nicholas Jing Yuan, and Tong Xu. 2020. BERT-MK: Integrating graph contextualized knowledge into pre-trained language models. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2281–2290.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1026–1034, Washington, DC, USA. IEEE Computer Society.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTa: Decoding-enhanced BERT with Disentangled Attention. In *International Conference on Learning Representations*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. 1999. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer.

Yankai Lin, Haozhe Ji, Zhiyuan Liu, and Maosong Sun. 2018. Denoising distantly supervised open-domain question answering. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018*, pages 1736–1745.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang. 2020. K-BERT: Enabling language representation with knowledge graph. In *Proceedings of AAAI 2020*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Zhibin Lu, Pan Du, and Jian-Yun Nie. 2020. VGCN-BERT: augmenting BERT with graph embedding for text classification. In *Advances in Information Retrieval - 42nd European Conference on IR Research, ECIR 2020, Lisbon, Portugal, April 14-17, 2020, Proceedings, Part I*, volume 12035 of *Lecture Notes in Computer Science*, pages 369–382. Springer.

Jong-Hoon Oh, Kazuma Kadowaki, Julien Kloetzer, Ryu Iida, and Kentaro Torisawa. 2019. Open–domain why-question answering with adversarial learning to encode answer texts. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4227–4237.

Matthew E. Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. 2019. Knowledge enhanced contextual word representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 43–54.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Ali Safaya, Moutasem Abdullatif, and Deniz Yuret. 2020. KUISAIL at SemEval-2020 task 12: BERT-CNN for offensive speech identification in social media. In *Proceedings of the Fourteenth Workshop on Semantic Evaluation*, pages 2054–2059.

Bo Shao, Yeyun Gong, Weizhen Qi, Nan Duan, and Xiaola Lin. 2019. Aggregating bidirectional encoder representations using MatchLSTM for sequence matching. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6059–6063.

Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Training very deep networks. In *Advances in Neural Information Processing Systems*, volume 28, pages 2377–2385.

Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. 2020. ERNIE 2.0: A continual pre-training framework for language understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 8968–8975.

Ilya Sutskever, Joshua B. Tenenbaum, and Ruslan R Salakhutdinov. 2009. Modelling relational data using bayesian clustered tensor factorization. In *Advances in Neural Information Processing Systems 22*, pages 1821–1828. Curran Associates, Inc.

T. Tieleman and G. Hinton. 2012. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Giorgos Vernikos, Katerina Margatina, Alexandra Chronopoulou, and Ion Androutsopoulos. 2020. Domain Adversarial Fine-Tuning as an Effective Regularizer. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3103–3112. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Jianshu Ji, Guihong Cao, Daxin Jiang, and Ming Zhou. 2020a. K-adapter: Infusing knowledge into pre-trained models with adapters. *CoRR*, abs/2002.01808.

Shuohang Wang, Luowei Zhou, Zhe Gan, Yen-Chun Chen, Yuwei Fang, Siqi Sun, Yu Cheng, and Jingjing Liu. 2020b. Cluster-former: Clustering-based sparse transformer for long-range dependency encoding. *arXiv preprint arXiv:2009.06097*.

Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. 2019. Multi-passage BERT: A globally normalized BERT model for open-domain question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5878–5882. Association for Computational Linguistics.

Wenhan Xiong, Jingfei Du, William Yang Wang, and Veselin Stoyanov. 2020. Pretrained encyclopedia: Weakly supervised knowledge-pretrained language model. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

An Yang, Quan Wang, Jing Liu, Kai Liu, Yajuan Lyu, Hua Wu, Qiaoqiao She, and Sujian Li. 2019a. Enhancing pre-trained language representations with rich knowledge for machine reading comprehension. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2346–2357.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019b. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32*, pages 5754–5764.

Shaohua Zhang, Haoran Huang, Jicong Liu, and Hang Li. 2020. Spelling error correction with soft-masked BERT. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 882–890. Association for Computational Linguistics.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. 2019. ERNIE: Enhanced language representation with informative entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1441–1451.

A Task-specific Modification for GLUE Test-set Submission

We applied task-specific modification to WNLI and CoLA in the GLUE tasks to achieve competitive GLUE leaderboard results, i.e., the test set submission results presented in Table 3. For WNLI, we followed Raffel et al. (2020), while, for CoLA, we propose our own modification. Note that we did not apply the tricks in obtaining the results on the development set results shown in Table 2. In the following, we describe the tricks.

A.1 WNLI

WNLI is a coreference resolution task with a two-sentence input. The first sentence has an ambiguous pronoun and the second sentence is generated from the first sentence by replacing the pronoun with one of the possible referents (noun phrases) in the first sentence (Wang et al., 2018). In this task, we must predict whether the candidate referent in the second sentence is the correct referent of the pronoun. Since the format of WNLI is known for being difficult to learn by a model, many previous works, including those using ALBERT, RoBERTa, or T5 (Liu et al., 2019; Lan et al., 2020; Raffel et al., 2020), converted the data to a simpler format before training their WNLI model for GLUE test-set submission.

Following these approaches, we also converted the data in the same way as Raffel et al. (2020). First, we extract candidate referents for an ambiguous pronoun as follows. Suppose that the following sentence pair of s_1 and s_2 is from the WNLI task's data and has the label *correct* (meaning that *Susan* in s_2 is the correct referent of the pronoun *she* in s_1).

s_1: Jane knocked on Susan's door but *she* $\underline{\text{did not get an answer}}$.

s_2: *Susan* $\underline{\text{did not get an answer}}$.

We first find all of the pronouns in the first sentence ("she" in s_1). For each pronoun, we find the longest sequence of words that precedes or follows the pronoun in the first sentence and that also appears in the second sentence ("did not get an answer" underlined in s_1 and s_2). We then choose the pronoun that precedes or follows the longest matching word sequence and obtain a candidate referent by deleting the matched sequence of words from the second sentence. In the example sentence pair (s_1, s_2), we choose the pronoun *she* from the first sentence (since there is a single pronoun) and obtain the candidate referent *Susan* from the second sentence through this process. Finally, we convert the original sentence pair into a pair of a masked sentence and a candidate referent by replacing the pronoun in the first sentence with `[MASK]` and replacing the second sentence with the extracted referent. The (s_1, s_2) pair is thus changed to the following (s'_1, s'_2):

s'_1: Jane knocked on Susan's door but `[MASK]` did not answer.

s'_2: *Susan*

Note that `[MASK]` in the sentence is different from the entity mask `[EM]` used in our GAN-style training for CNNs. For the input to our CNNs, we further replaced `[MASK]` with `[EM]`. Since the format of this converted data is similar to that of the training data for the GAN-style training scheme of our CNN, we expect that by using this data conversion, BERTAC can more effectively predict whether the candidate referent for the masked pronoun is correct.

A.2 CoLA

In the CoLA task, we need to predict whether a given sentence is grammatically acceptable. For

	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	WNLI
Learning rate	{8e-6, 9e-6,1e-5, 2e-5, 3e-5}								
Batch size	128			16					
Training epoch	{1,2,3,4,5}			{6,7,8,9,10}					{90,95,100, 105,110}
TIER layer	3								
Max sequence length	128								
Warmup step	linear warmup for the first 6% of steps								
CNN	9 models pretrained with different filter settings								

Table 9: Hyperparameters of BERTAC tested for GLUE experiments.

	Quasar-T	SearchQA
Learning rate	1e-5	
Batch size	48	96
Training epoch	2	
TIER layer	3	
Max sequence length	384	
Warmup step	linear warmup for the first 6% of steps	
CNN	9 models pretrained with different filter settings	

Table 10: Hyperparameters of BERTAC tested for open-domain QA experiments.

this task, we conducted a two-step fine-tuning. In the first step, we fine-tuned BERTAC with automatically generated pseudo-training data. This data was prepared as described below, and does not include the original CoLA training data. In the second step, we further fined-tuned the model obtained in the first step using the original CoLA training data. The BERTAC model obtained at this second step was used for the test-set submission.

To automatically generate pseudo-training data, we regarded all of the sentences in the training data of MNLI, QQP, and QNLI as grammatically acceptable and used them as positive examples in the pseudo-training data. After removing duplicate sentences, for each positive example, we generated one negative example by modifying the positive example under the assumption that the modification makes the generated example grammatically unacceptable. As a modification, we randomly applied one of the following three operations: permutation (of four words randomly selected), insertion (of two random words to random positions), and deletion (of two randomly selected words) (Brahma, 2018).

We obtained about 2.14 million examples in this way, half of them positives and the other half negatives. We used all of the training samples automatically generated in this way for the first-step fine-tuning of BERTAC, with a learning rate of 8e-6, a single training epoch, and a batch size of 128, while applying the same settings for the other hyperparameters as those used for the other tasks. The model obtained by the first-step fine-tuning is then used as a starting point for the second-step fine-tuning, using the original CoLA training data this time, of our final model for CoLA.

B Hyperparameters

Hyperparameters of BERTAC tested for GLUE and open-domain QA experiments are summarized in Tables 9 and 10, where CNN represents CNN models pretrained with different filter settings (filter's window sizes $\in$ {"1,2,3", "1,2,3,4", "2,3,4"} and number of filters $\in$ {100, 200, 300}) described in Section 3.3. We tested all combinations of these hyperparameters and chose the best one using the development set of each task.