GiBERT: Enhancing BERT with Linguistic Information using a Lightweight Gated Injection Method

Nicole Peinelt$^{1,2}$ and Marek Rei$^3$ and Maria Liakata$^{1,2,4}$

$^1$The Alan Turing Institute, UK
$^2$University of Warwick, UK
$^3$Imperial College London, UK
$^4$Queen Mary University of London, UK
n.peinelt@warwick.ac.uk, marek.rei@imperial.ac.uk, m.liakata@qmul.ac.uk

Abstract

Large pre-trained language models such as BERT have been the driving force behind recent improvements across many NLP tasks. However, BERT is only trained to predict missing words – either through masking or next sentence prediction – and has no knowledge of lexical, syntactic or semantic information beyond what it picks up through unsupervised pre-training. We propose a novel method to explicitly inject linguistic information in the form of word embeddings into any layer of a pre-trained BERT. When injecting counter-fitted and dependency-based embeddings, the performance improvements on multiple semantic similarity datasets indicate that such information is beneficial and currently missing from the original model. Our qualitative analysis shows that counter-fitted embedding injection is particularly beneficial, with notable improvements on examples that require synonym resolution.

1 Introduction

Detecting the semantic similarity between a given text pair is at the core of many NLP tasks. It is a challenging problem due to the inherent variability of language and the limitations of surface form similarity. Recent pre-trained language models such as ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019) have led to noticeable improvements in semantic similarity detection and subsequent work has explored how these architectures can be further improved. One line of work aims at model compression, making BERT smaller and accessible while mostly preserving its performance (Xu et al., 2020; Goyal et al., 2020; Sanh et al., 2019; Aguilar et al., 2020; Lan et al., 2020; Chen et al., 2020). Other studies seek to further improve model performance by enhancing BERT with external information from knowledge bases (Peters et al., 2019; Wang et al., 2020) or additional modalities (Lu et al., 2019; Lin et al., 2020).

Before the rise of contextualised models, transfer of pre-trained information between datasets and tasks in NLP was based on word embeddings. Over many years, substantial effort was placed into the creation of such embeddings. While originally capturing mainly collocation patterns (Mikolov et al., 2013; Pennington et al., 2014), subsequent work enriched these embeddings with additional information, such as dependencies (Levy and Goldberg, 2014), subword information (Luong et al., 2013; Bojanowski et al., 2017) and semantic lexicons (Faruqui et al., 2015). As a result, there exists a wealth of pre-trained embedding resources for many languages in a unified format which could provide complementary information for contemporary pre-trained contextual models. Moreover, aligning contextual embeddings with static embeddings has shown to increase the performance of the former (Liu et al., 2020).

We propose a new method for injecting pre-trained linguistically-enriched embeddings into any layer of BERT. The model maps any word embeddings into the same space as BERT’s hidden representations, then combines them using learned gating parameters. Evaluation of this method on five semantic similarity tasks shows that injecting pre-trained dependency-based and counter-fitted embeddings can further enhance BERT’s performance. More specifically, we make the following contributions:

1. We propose GiBERT - a lightweight gated method for injecting externally pre-trained embeddings into BERT (section 3.1).

2. We provide an ablation study and a detailed analysis of the components in the injection architecture (section 5).

3. We demonstrate that the proposed model improves BERT’s performance on multiple se-

$^{1}$Code available at https://github.com/wuningxi/GiBERT.
mantic similarity detection datasets. In comparison to multi-head attention injection, our gated injection method uses fewer parameters while achieving comparable performance for dependency embeddings and improved results for counter-fitted embeddings (section 5).

4. Our qualitative analysis provides insights into GiBERT’s improved performance, such as in cases of sentence pairs involving synonyms. (section 5).

2 Related work

BERT modifications Due to BERT’s widespread success in NLP, recent studies have focused on further improving BERT by introducing external information. Such work covers a variety of application areas and technical approaches. We broadly categorise such approaches into input-related, external and internal. Input modifications (Zhao et al., 2020; Singh et al., 2020; Lai et al., 2020; Ruan et al., 2020) adapt the information that is fed to BERT – e.g. feeding text triples separated by [SEP] tokens instead of sentence pairs as in Lai et al. (2020) – while leaving the architecture unchanged. Output modifications (Xuan et al., 2020; Zhang et al., 2020) build on BERT’s pre-trained representation by adding external information after the encoding step – e.g. combining it with additional semantic information as in Zhang et al. (2020) – without changing BERT itself. By contrast, internal modifications introduce new information directly into BERT by adapting its internal architecture. Fewer studies have taken this approach as this is technically more difficult and might increase the risk of so-called catastrophic forgetting – completely forgetting previous knowledge when learning new tasks (French, 1999; Wen et al., 2018). However, such modifications also offer the opportunity to directly harness BERT’s powerful architecture to process the external information alongside the pretrained one. Most existing work on internal modifications has attempted to combine BERT’s internal representation with visual and knowledge base information: Lu et al. (2019) modified BERT’s transformer block with co-attention to integrate visual and textual information, while Lin et al. (2020) introduced a multimodal model which uses multi-head attention to integrate encoded image and text information between each transformer block. Peters et al. (2019) suggested a word-to-entity attention mechanism to incorporate external knowledge into BERT and Wang et al. (2020) proposed to inject factual and linguistic knowledge through separate adapter modules. Our method introduces external information with an addition-based mechanism which uses fewer parameters than existing attention-based techniques (Lu et al., 2019; Lin et al., 2020; Peters et al., 2019). We further incorporate a gating mechanism to scale injected information so as to reduce the risk of catastrophic forgetting. Moreover, our work investigates the injection of pretrained word embeddings, rather than multimodal or knowledge base information as in previous studies.

Semantic similarity detection Semantic similarity detection is a framework for binary text pair classification tasks such as paraphrase detection, duplicate question identification and answer sentence selection which require detecting the semantic similarity between text pairs (Peinelt et al., 2020). Early semantic similarity methods used feature-engineering techniques, exploring various syntactic (Filice et al., 2017), semantic (Balchev et al., 2016) and lexical features (Tran et al., 2015; Almarwani and Diab, 2017). Subsequent work tried to model text pair relationships either based on increasingly complex neural architectures (Deriu and Cieliebak, 2017; Wang et al., 2017; Tan et al., 2018) or by combining both approaches through hybrid techniques (Wu et al., 2017a; Feng et al., 2017; Koreeda et al., 2017). Most recently, contextual models such as ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019) have reached state-of-the-art performance through pretraining large context-aware language models on vast amounts of textual data. Our study joins up earlier lines of work with current state-of-the-art contextual representations by investigating the combination of BERT with dependency-based and counter-fitted embeddings.

3 GiBERT

3.1 Architecture

We propose GiBERT - a Gated Injection Method for BERT. Our model (Figure 1) is designed with semantic similarity detection in mind and comprises the following: obtaining BERT’s intermediate representation from Transformer block i (step 1-2 in Figure 1), creating an alternative input representation based on linguistically-enriched word
embeddings (step 3-4), combining both representations (steps 5-7) and passing on the injected information to subsequent BERT layers to make a final prediction (steps 8-9).

**BERT representation** We encode a sentence pair with a pre-trained BERT model (Devlin et al. 2019) and obtain BERT’s internal representation at different layers (see section 5 for injection layer choices). Following standard practices, we process the two input sentences \( S_1 \) and \( S_2 \) with a word piece tokenizer (Wu et al., 2017b) and combine them using ‘[CLS]’ and ‘[SEP]’ tokens, which indicate sentence boundaries. The word pieces are then mapped to ids, resulting in a sequence of word piece ids \( \mathbf{E}^W = [w_1, \ldots, w_N] \) where \( N \) indicates the number of word pieces in the sequence (step 1 in Figure 1). In the case of embedding layer injection, we use BERT’s embedding layer output denoted with \( \mathbf{H}^0 \) which results from summing the word piece embeddings \( \mathbf{E}^W \), positional embeddings \( \mathbf{E}^P \) and segment embeddings \( \mathbf{E}^S \) (step 2):

\[
\mathbf{H}^0 = \text{LayerNorm}(\mathbf{E}^W + \mathbf{E}^P + \mathbf{E}^S)
\]

\[
\mathbf{E}^W, \mathbf{E}^P, \mathbf{E}^S, \mathbf{H}^0 \in \mathbb{R}^{N \times D} \quad (1)
\]

where \( D \) is the internal hidden size of BERT (\( D = 768 \) for BERT\_BASE). For injecting information at later layers, we obtain BERT’s internal representation \( \mathbf{H}^i \in \mathbb{R}^{N \times D} \) after transformer block \( i \) with \( 1 \leq i \leq L \) (step 2):

\[
\mathbf{M}^i = \text{LayerNorm}(\mathbf{H}^{i-1} + \text{MultiheadAtt}(\mathbf{H}^{i-1}))
\]

\[
\mathbf{H}^i = \text{LayerNorm}(\mathbf{M}^i + \text{FeedForward}(\mathbf{M}^i))
\]

(2)

where \( L \) is the number of Transformer blocks (\( L = 12 \) for BERT\_BASE) and MultiheadAtt denotes multihead attention.

**External embedding representation** To enrich this representation, we obtain alternative representations for the tokens in \( S_1 \) and \( S_2 \) by looking up word embeddings in a pre-trained embedding matrix \( \mathbf{E} \in \mathbb{R}^{|V| \times E} \), where \( |V| \) denotes vocabulary size and \( E \) the dimensionality of the pre-trained embeddings (step 3, section 3.2 presents details regarding our choice of pre-trained embeddings). In order to map word embedding representations to BERT’s word piece representations, an alignment function duplicates the word embedding for the corresponding number of subwords, then adds BERT’s special ‘[CLS]’ and ‘[SEP]’ tokens, resulting in an injection sequence \( \mathbf{I} \in \mathbb{R}^{F \times E} \) (step 4). For example, it assigns the pre-trained embedding of the word ‘prompt’ to both of the corresponding word pieces ‘pro’ and ‘##mpt’ (see Figure 1).

**Attention injection** Multihead attention was proposed by Vaswani et al. (2017):

\[
\text{MultiheadAtt}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\text{head}_1; \ldots; \text{head}_h] \mathbf{W}^O
\]

\[
\text{head}_j = \text{Attention}(\mathbf{QW}_j^Q, \mathbf{KW}_j^K, \mathbf{VW}_j^V)
\]

(3)

and is employed in Transformer networks in the form of self-attention (where queries \( \mathbf{Q} \), keys \( \mathbf{K} \) and values \( \mathbf{V} \) come from the previous layer) or encoder-decoder attention (where queries come from the decoder, keys and values from the encoder). Previous work has successfully employed multihead attention to combine BERT with external information (see section 2). In their multimodal ViLBERT model, Lu et al. (2019) combined textual and visual representations by passing the keys and values from each modality as input to the other modality’s multi-head attention block. Similarly, Peters et al. (2019) used multihead attention to combine BERT with external information.

For our case of combining BERT with the injection sequence, we can therefore experiment with the following multi-head attention injection method:

\[
\mathbf{H}^I = \mathbf{H}^i + \text{MultiHeadAtt}(\mathbf{H}^i, \mathbf{I}, \mathbf{I})
\]

(4)

where queries are provided by BERT’s internal representation, while keys and values come from the injected embeddings. The output of the attention mechanism is then combined with the previous layer through addition.

**Gated injection** We also propose an alternative method for combining external embeddings with BERT which requires only 14% of parameters used in multi-head attention (0.23M instead of 1.64M, see Appendix G). First, we add a feed-forward layer – consisting of a linear layer with weights \( \mathbf{W}_p \in \mathbb{R}^{D \times E} \) and bias \( \mathbf{b}_p \in \mathbb{R}^D \) with a tanh activation function – to project the aligned embedding sequence to BERT’s internal dimensions and squash the output values to a range between -1 and 1 (step 5):

\[
\mathbf{P} = \text{FeedForward}(\mathbf{I}) \in \mathbb{R}^{N \times D}
\]

(5)
Then, we use a residual connection to inject the projected external information into BERT’s representation from Transformer block $i$ (see section 5 for injection at different locations) and obtain a new enriched representation $H^i_\ell \in \mathbb{R}^{N \times D}$:

$$H^i_\ell = H^i + P \quad (6)$$

However, injection values in $P$ can range between $-1$ and $1$, whereas values in BERT’s internal representation $H^i$ usually range from $-0.1$ to $0.1$. When external information is directly injected using an additive operation, BERT’s pre-trained information can be easily overwritten by the injection, resulting in catastrophic forgetting. To address this potential pitfall, we further propose a gating mechanism which uses a gating vector $g \in \mathbb{R}^D$ to scale the injected information before combining it with BERT’s internal representation as follows:

$$H^i_\ell = H^i + g \odot P \quad (7)$$

where $\odot$ denotes element-wise multiplication using broadcasting (step 6 & 7). The gating parameters are initialised with zeros and updated during training. This has the benefit of starting fine-tuning from representations which are equivalent to vanilla BERT and gradually introducing the injected information during fine-tuning along certain dimensions. If specific features in the external representations are not beneficial for the task, it is easy for the model to ignore them by keeping the gating parameters at zero.

**Output layer** The combined representation $H^i_\ell$ is then fed to BERT’s next Transformer block $i + 1$ (step 8). At the final Transformer block $L$, we use the $c \in \mathbb{R}^D$ vector which corresponds to the ‘[CLS]’ token in the input and is typically used as the sentence pair representation (step 9). As proposed by Devlin et al. (2019), this is followed by a softmax classification layer (with weights $W^L \in \mathbb{R}^{C \times D}$ and $b^L \in \mathbb{R}^C$) to calculate class probabilities where $C$ indicates the number of classes. During finetuning, we train the entire model for 3 epochs with early stopping and cross-entropy loss. Learning rates are tuned for each seed and dataset based on development set performance (reported in Appendix D).
3.2 Injected Embeddings

While many different embedding resources exist, here we focus on experimenting with pre-trained word representations that are beneficial for the semantic similarity detection tasks and also contain information complementary to BERT. Embeddings such as word2vec and Glove leverage co-occurrence patterns which have been shown to be also captured by BERT (Gan et al., 2020). Recent contextualised embeddings risk redundancy with BERT due to the similarity of used approaches. We reason that linguistically-enriched embeddings are most likely to be complementary to BERT, as the model has not been explicitly trained on semantic or syntactic resources and has only partial knowledge of syntax and semantics (Rogers et al., 2020). We hence experiment with injecting dependency-based (Levy and Goldberg, 2014) and counter-fitted embeddings (Mrkšić et al., 2016) into BERT, which have been found useful for semantic similarity modelling and other related tasks (Filice et al., 2017; Feng et al., 2017; Alzantot et al., 2018; Jin et al., 2020).

The 300-dim dependency-based embeddings by Levy and Goldberg (2014) extend the SkipGram embedding algorithm proposed by Mikolov et al. (2013) by replacing linear bag-of-word contexts with dependency-based contexts which are extracted from parsed English Wikipedia sentences. As BERT has not been exposed to dependencies during pretraining and previous studies have found that BERT’s knowledge of syntax is only partial (Rogers et al., 2020), we reason that these embeddings could provide complementary information.

The 300-dim counter-fitted embeddings by Mrkšić et al. (2016) integrate antonymy and synonymy relations into word embeddings based on an objective function which combines three principles: repelling antonyms, attracting synonyms and preserving the vector space. For training, they obtain synonym and antonym pairs from the Paraphrase Database and WordNet, demonstrating an increased performance on SimLex-999 (Hill et al., 2015). We use their highest-scoring vectors which were obtained by applying the counter-fitting method to Paragram vectors from Wieting et al. (2015). Antonym and synonym relations are particularly important for paraphrase detection and injecting them into BERT gives the model access to this useful additional information.

4 Evaluation

4.1 Datasets and Tasks

We focus on semantic similarity detection which is a fundamental problem in NLP and involves modelling the semantic relationship between two sentences in a binary classification setup. We work with the following five widely used English language datasets which cover a range of sizes and tasks (including paraphrase detection, duplicate question identification and answer sentence selection, see Appendix A for details).

MSRP The Microsoft Research Paraphrase dataset (MSRP) contains 5K pairs of sentences from news websites which were collected based on heuristics and an SVM classifier. Gold labels are based on human binary annotations for sentential paraphrase detection (Dolan and Brockett, 2005).

SemEval The SemEval 2017 CQA dataset (Nakov et al., 2017) consists of three subtasks involving posts from the online forum Qatar Living and the original problem is a ranking setting, but since the gold labels are binary, we focus on a classification setup. In subtask A, the posts are questions and comments from the same thread, in an answer sentence selection setup (26K instances). Subtask B is question paraphrase detection (4K instances). Subtask C is similar to A but comments were retrieved from an external thread (47K). We use the 2016 test set as the dev set and the 2017 test set as the test set.

Quora The Quora duplicate questions dataset is the largest of the selected datasets, consisting of more than 400K question pairs with binary labels. The task is to predict whether two questions are duplicates. We use Wang et al. (2017)’s train/dev/test set partition.

All of the above datasets provide two short texts, each usually a single sentence but sometimes consisting of multiple sentences. For simplicity, we refer to each short text as ‘sentence’. We frame the
task as semantic similarity detection between two sentences through binary classification.

### 4.2 Metrics

Our main evaluation metric is the F1 score as this is more meaningful than accuracy for datasets with imbalanced label distributions (such as SemEval C, see Appendix A). We also report performance on difficult cases using the non-obvious F1 score (Peinelt et al., 2019). This metric distinguishes obvious from non-obvious instances in a dataset based on lexical overlap and gold labels, and calculates a separate F1 score for challenging cases. This value therefore tends to be lower than the regular F1 score. Dodge et al. (2020) recently showed that early stopping and random seeds can have considerable impact on the performance of finetuned BERT models, therefore we finetune all models for 3 epochs with early stopping (based on dev F1) and report average model performance across two different seeds. Hyperparameter settings of all BERT-based models are identical, except for learning rate and injection location which are tuned with grid search, see Appendix D.

### 4.3 Baselines

**SemEval systems** We compare against the best SemEval 2017 system for each subtask based on F1 score: KeLP (Filice et al., 2017), ECNU (Wu et al., 2017a) and Bunji (Koreeda et al., 2017).

**BERT** Following standard practice, we encode the sentence pair with BERT’s C vector from the final layer, followed by a softmax layer as proposed by Devlin et al. (2019). We use Tensorflow Hub’s distribution of BERT

**SemBERT** Additionally we compare with the semantics-aware BERT model (SemBERT, Zhang et al. 2020) which uses a semantic role labeler. As the original paper reports results on different dataset versions, we ran the official code on our datasets. The longer sentences in SemEval could not fit on a single GPU due to the larger model size.

**tBERT** We also combine embeddings with BERT using an averaging and concatenation method proposed in tBERT (Peinelt et al., 2020). Instead of the word topics in the original system, we use pretrained counter-fitted and dependency embeddings for direct comparison with our methods.

**AiBERT** We further provide an alternative Attention-based embedding Injection method for BERT based on the multihead attention injection mechanism described in equations 3 to 4. Following the same procedure as GiBERT, we tune the injection location (see Appendix E).

### 5 Results

**Full model** GiBERT with counter-fitted embeddings outperforms all other systems in both average F1 and average non-obvious F1 score (see Table 1). This shows that the model improves on challenging dataset instances, rather than merely leveraging shallow surface patterns. It is worth noting that

<table>
<thead>
<tr>
<th></th>
<th>MSRP</th>
<th>Quora</th>
<th>SemEval A</th>
<th>B</th>
<th>C</th>
<th></th>
<th>MSRP</th>
<th>Quora</th>
<th>SemEval A</th>
<th>B</th>
<th>C</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Previous systems</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KeLP</td>
<td>-</td>
<td>-</td>
<td>.506</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECNU</td>
<td>-</td>
<td>-</td>
<td>.777</td>
<td></td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bunji</td>
<td>-</td>
<td>-</td>
<td>.197</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT*</td>
<td>.876</td>
<td>.902</td>
<td>.704</td>
<td>.473</td>
<td>.268</td>
<td>.645</td>
<td>.827</td>
<td>.860</td>
<td>.656</td>
<td>.243</td>
<td>.085</td>
<td>.534</td>
</tr>
<tr>
<td>SemBERT*</td>
<td>.876</td>
<td>.901</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>.820</td>
<td>.874</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>tBERT_{dependency}*</td>
<td>.882</td>
<td>.906</td>
<td>.780</td>
<td>.510</td>
<td>.242</td>
<td>.664</td>
<td>.827</td>
<td>.858</td>
<td>.728</td>
<td>.262</td>
<td>.090</td>
<td>.553</td>
</tr>
<tr>
<td>tBERT_{counter-fitted}*</td>
<td>.879</td>
<td>.906</td>
<td>.756</td>
<td>.500</td>
<td>.215</td>
<td>.651</td>
<td>.824</td>
<td>.857</td>
<td>.699</td>
<td>.238</td>
<td>.064</td>
<td>.540</td>
</tr>
<tr>
<td><strong>Our implementation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AiBERT_{dependency}</td>
<td>.863</td>
<td>.903</td>
<td>.738</td>
<td>.498</td>
<td><strong>282</strong></td>
<td>.657</td>
<td>.792</td>
<td>.866</td>
<td>.681</td>
<td><strong>268</strong></td>
<td>.090</td>
<td>.539</td>
</tr>
<tr>
<td>AiBERT_{counter-fitted}</td>
<td>.887</td>
<td>.904</td>
<td>.724</td>
<td>.496</td>
<td>.263</td>
<td>.653</td>
<td>.835</td>
<td>.867</td>
<td>.662</td>
<td>.264</td>
<td>.076</td>
<td>.541</td>
</tr>
<tr>
<td>GiBERT_{dependency}</td>
<td>.883</td>
<td>.904</td>
<td>.768</td>
<td>.474</td>
<td>.238</td>
<td>.653</td>
<td>.849</td>
<td>.864</td>
<td>.704</td>
<td>.231</td>
<td>.087</td>
<td>.547</td>
</tr>
<tr>
<td>GiBERT_{counter-fitted}</td>
<td><strong>884</strong></td>
<td>.907</td>
<td><strong>780</strong></td>
<td>.511</td>
<td>.256</td>
<td><strong>668</strong></td>
<td>.858</td>
<td>.862</td>
<td>.719</td>
<td>.248</td>
<td><strong>090</strong></td>
<td><strong>555</strong></td>
</tr>
</tbody>
</table>

Table 1: Model performance on test set. All BERT-based methods use BERT\_BASE. Bold font highlights the best result overall, our best systems are underlined. avg = average performance across all datasets, ◦ = results from publication, * = official code run on our data, X= run failed due to insufficient GPU memory, dep = dependency embeddings, counter = counter-fitted embeddings.
GiBERT has the fewest parameters of all BERT-enhancing models (see Appendix F) and doesn’t require any additional preprocessing tools (such as the neural SRL tagger required by SemBERT), making it more efficient than SemBERT, tBERT and AiBERT. The largest improvement of GiBERT over BERT is observed with counter-fitted embeddings, especially on SemEval A and B (the datasets with the highest proportion of examples involving synonym pairs, see Table 5). GiBERT with dependency embeddings still improves over vanilla BERT, but gains tend to be smaller and roughly similar to the more complex AiBERT injection method. tBERT always combines external information with BERT at the latest possible stage, which is beneficial for dependency embeddings but less effective for counter-fitted embeddings (compare Table 3). Compared to GiBERT, this makes it a less flexible method while also requiring more parameters. Our results indicate that semantic information is more important for the tasks at hand and syntactic information benefits from a late integration.

**Gating mechanism** Catastrophic forgetting is a potential problem when introducing external information into a pre-trained model as the injected information could disturb or completely overwrite existing knowledge (Wang et al., 2020). In our proposed model, a gating mechanism is used to scale injected embeddings before adding them to the pre-trained internal BERT representation (see section 3.1). To understand the importance of this mechanism, in Table 2 we contrast development set performance for injecting information after the embedding layer with gating – as defined in equation 7 – and without – as in equation 6. For dependency embedding injection without gating, performance only improves on 2 out of 5 datasets over the base-line and in some cases even drops below BERT’s performance, while it outperforms the baseline on all datasets when using the gating mechanism. Counter-fitted embedding injection without gating improves on 4 out of 5 datasets, with further improvements when adding gating, outperforming the vanilla BERT model across all datasets. In addition, gating makes model training more stable and reduces failed runs (where the model predicted only the majority class) on the particularly imbalanced SemEval C dataset. This highlights the importance of the gating mechanism in our proposed method.

**Injection location** In our proposed model, information can be injected between any of BERT’s pre-trained transformer blocks. We reason that different locations may be more appropriate for certain kinds of embeddings as previous research has found that different types of information tend to be encoded and processed at specific BERT layers (Rogers et al., 2020). We experiment with three locations: after the embedding layer (using $H^0$), after the middle layer (using $H^6$ in BERT BASE) and after the penultimate layer (using $H^{11}$ in BERT BASE). Table 3 shows that midlayer injection is ideal for counter-fitted embeddings, while late injection appears to work best for dependency embeddings (Table 3). This is in line with previous work which found that BERT tends to processes syntactic information at later layers than linear word-level information (Rogers et al., 2020). We consequently use these injection locations in our final model (see Appendix E for AiBERT’s tuned injection locations).

**Error Analysis** Counter-fitted embeddings are designed to explicitly encode synonym and antonym relationships between words. To better understand how the injection of counter-fitted embeddings affects the ability of our model to deal with these relationships, we analyze the performance of GiBERT models with and without gating at different layers. Table 2 shows the development set F1 scores for injecting pretrained embeddings after the embedding layer with vs. without gating mechanism.

<table>
<thead>
<tr>
<th></th>
<th>MSRP</th>
<th>Quora</th>
<th>SemEval A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>.906</td>
<td>.906</td>
<td>.714</td>
<td>.754</td>
<td>.414</td>
</tr>
<tr>
<td>GiBERT with dependency embeddings - no gating</td>
<td>.906</td>
<td>.905</td>
<td>.732</td>
<td>.751</td>
<td>.424</td>
</tr>
<tr>
<td></td>
<td>.913</td>
<td>.908</td>
<td>.755</td>
<td>.778</td>
<td>.433</td>
</tr>
<tr>
<td></td>
<td>.914</td>
<td>.910</td>
<td>.760</td>
<td>.773</td>
<td>.444</td>
</tr>
<tr>
<td>GiBERT with counter-fitted embeddings - no gating</td>
<td>.907</td>
<td>.908</td>
<td>.751</td>
<td>.767</td>
<td>.451</td>
</tr>
<tr>
<td></td>
<td>.917</td>
<td>.909</td>
<td>.760</td>
<td>.771</td>
<td>.464</td>
</tr>
<tr>
<td></td>
<td>.910</td>
<td>.907</td>
<td>.755</td>
<td>.771</td>
<td>.450</td>
</tr>
</tbody>
</table>

Table 3: Development set F1 scores of embedding injection at different layers.
with instances involving such semantic relations, we use synonym and antonym pairs from the PPDB and Wordnet (provided by Mrkšić et al. 2016) and search the development partition of the datasets for sentence pairs where the first sentence contains one word of the synonym/antonym pair and the second sentence the other word. Table 5 reports F1 performance of our model on cases with synonym pairs, antonym pairs and neither one. We find that our model’s F1 performance particularly improves over BERT on instances containing synonym pairs, as illustrated in example (1) in Table 4. By contrast, the performance on cases with antonym pairs stays roughly the same, although slightly decreasing on Quora. As illustrated by example (2) in Table 4, word pairs can be antonyms in isolation (e.g. husband - wife), but not in the specific context of a given example. In rare cases, the injection of distant antonym pair embeddings can therefore deter the model from detecting related sentence pairs. We also observe a slight performance boost for cases without synonym or antonym pairs which could be due to improved representations for words which occurred in examples without their synonym or antonym counterpart.

### 6 Conclusion

In this paper, we introduced a new approach for injecting external information into BERT. Our proposed method adds linguistically enriched embeddings to BERT’s internal representation through a lightweight gating mechanism which requires fewer parameters than previous approaches. Evaluating our injection method on multiple semantic similarity detection datasets, we demonstrated that injecting counter-fitted embeddings clearly improved performance over vanilla BERT and on average outperformed all baselines on the task, while dependency embedding injection achieved slightly smaller gains. In comparison to the multi-head attention injection mechanism, we found the gated method at least as effective, with comparable performance for dependency embeddings and improved results for counter-fitted embeddings. In ablation studies, we showed that the choice of injection location and the use of the proposed gating mechanism are crucial for our architecture. Our qualitative analysis highlighted that counter-fitted injection was particularly helpful for instances involving synonym pairs. Future work could explore combining multiple embedding sources or injecting other types of information. Another direction is to investigate the usefulness of embedding injection for other tasks.

<table>
<thead>
<tr>
<th>Sentence 1</th>
<th>Sentence 2</th>
<th>Gold label</th>
<th>BERT prediction</th>
<th>GiBERT prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>it took me more than 10 people; over the course of the whole day to convince my point at Qatar airways... as to how my points needs to be redeemed... at long last my point was made... dont seem know what they are doing?? appalling to say the least</td>
<td>this isn’t the first time. so many rants by irate customers on so many diverse situations signals a very serious problem. so called first class airlines and no basic customer care. over confidence much?</td>
<td>is related</td>
<td>not related</td>
<td>is related</td>
</tr>
<tr>
<td>(1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hi; my wife was on a visit visa; today; her residency visa was issued; so i went to immigration and paid 500 so there is no need to leave the country and enter again on the residency visa. she has done her medical before for the visit visa extension; do we need to do the medical again for the residency visa? thanks</td>
<td>dear all; please let me know how many days taking for approve family visa nw; am last wednesday (12/09/2012) apply family visa for my husband and daughter; but still now showing in moi website itz under review; itz usual reply? why delayed like this? please help me regards divya</td>
<td>is related</td>
<td>is related</td>
<td>not related</td>
</tr>
<tr>
<td>(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Examples from the Semeval development set. Synonym and antonym pairs are highlighted in bold.

<table>
<thead>
<tr>
<th>Instances with antonym pairs</th>
<th>MSRP</th>
<th>Quora</th>
<th>SemEval</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4%)</td>
<td>(4%)</td>
<td>(21%)</td>
<td>(28%)</td>
<td>(20%)</td>
<td><strong>BERT</strong></td>
<td>.81</td>
</tr>
<tr>
<td><strong>GiBERT</strong></td>
<td>.81</td>
<td>.86</td>
<td>.77</td>
<td>.75</td>
<td>.46</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instances with synonym pairs</th>
<th>MSRP</th>
<th>Quora</th>
<th>SemEval</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>(11%)</td>
<td>(9%)</td>
<td>(22%)</td>
<td>(31%)</td>
<td>(17%)</td>
<td><strong>BERT</strong></td>
<td>.87</td>
</tr>
<tr>
<td><strong>GiBERT</strong></td>
<td>.90</td>
<td>.91</td>
<td>.82</td>
<td>.83</td>
<td>.54</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instances without synonym/antonym pairs</th>
<th>MSRP</th>
<th>Quora</th>
<th>SemEval</th>
</tr>
</thead>
<tbody>
<tr>
<td>(85%)</td>
<td>(87%)</td>
<td>(64%)</td>
<td></td>
</tr>
<tr>
<td><strong>BERT</strong></td>
<td>.91</td>
<td>.91</td>
<td>.71</td>
</tr>
<tr>
<td><strong>GiBERT</strong></td>
<td>.92</td>
<td>.91</td>
<td>.73</td>
</tr>
</tbody>
</table>

Table 5: F1 score on instances containing synonymy pairs, antonymy pairs or no pairs across datasets.
Acknowledgments
This work was supported by Microsoft Azure and The Alan Turing Institute under the EPSRC grant EP/N510129/1. It was also supported by a UKRI/EPSRC Turing AI Fellowship to Maria Liakata (grant no. EP/V030302/1).

References


Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. **Efficient estimation of word representations in vector space.** In *1st International Conference on Learning Representations (ICLR 2013)*, Scottsdale, USA.


Tagging with Applications in PICO Annotation. arXiv:1910.09255 [cs, stat].


Appendix

A Dataset Characteristics

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Language</th>
<th>Source</th>
<th>Positive instances</th>
<th>Train</th>
<th>Dev</th>
<th>Test</th>
<th>Total size</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSRP</td>
<td>English</td>
<td>News websites</td>
<td>67%</td>
<td>3576</td>
<td>500</td>
<td>1725</td>
<td>5k</td>
</tr>
<tr>
<td>Quora</td>
<td>English</td>
<td>Q&amp;A website</td>
<td>37%</td>
<td>384347</td>
<td>9999</td>
<td>9999</td>
<td>404k</td>
</tr>
<tr>
<td>SemEval A</td>
<td>English</td>
<td>Online forum</td>
<td>39%</td>
<td>20340</td>
<td>3270</td>
<td>2930</td>
<td>26k</td>
</tr>
<tr>
<td>SemEval B</td>
<td>English</td>
<td>Online forum</td>
<td>36%</td>
<td>3169</td>
<td>700</td>
<td>880</td>
<td>4k</td>
</tr>
<tr>
<td>SemEval C</td>
<td>English</td>
<td>Online forum</td>
<td>9%</td>
<td>31690</td>
<td>7000</td>
<td>8800</td>
<td>47k</td>
</tr>
</tbody>
</table>

Table 6: Properties of selected semantic similarity detection data sets.

B Examples

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Task</th>
<th>Sentence 1</th>
<th>Example</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quora</td>
<td>Duplicate question identification</td>
<td>There are only 2,000 Roman Catholics living in Banja Luka now.</td>
<td>There are just a handful of Catholics left in Banja Luka.</td>
<td>is_paraphrase</td>
</tr>
<tr>
<td>MSRP</td>
<td>Paraphrase detection</td>
<td>Which is the best way to learn coding?</td>
<td>How do you learn to program?</td>
<td>is_paraphrase</td>
</tr>
<tr>
<td>SemEval</td>
<td>(A) Internal answer sentence selection</td>
<td>Anybody recommend a good dentist in Doha?</td>
<td>Dr Sarah Dental Clinic</td>
<td>is_related</td>
</tr>
<tr>
<td></td>
<td>(B) Question paraphrase detection</td>
<td>Where I can buy good oil for massage?</td>
<td>Blackheads - Any suggestions on how to get rid of them??</td>
<td>not_related</td>
</tr>
<tr>
<td></td>
<td>(C) External answer sentence selection</td>
<td>Can anybody tell me where is Doha clinic?</td>
<td>Dr. Rizwi - Al Ahli Hospital</td>
<td>not_related</td>
</tr>
</tbody>
</table>

Table 7: Text pair similarity data sets with examples.
C Preprocessing

We lowercase and tokenise all datasets, replacing images and URLs with placeholders. Sequences exceeding the maximum length are cut off. Each model is trained on a single NVIDIA Tesla K80 GPU.

D Hyper-Parameters

For datasets with long sentences, the batch size is reduced from 32 to 16 to fit on a single GPU. We tune learning rates (2e-5, 3e-5 and 5e-5) through grid search based on highest development set F1 scores. The best injection location (embd layer, layer 6 and layer 11) is selected based on best development set F1 scores across datasets (see Tables 2 and 9). Table 8 shows the final hyperparameters after tuning. In total, we train 6 BERT, SemBERT and tBERT models (2 different random seeds x 3 learning rates), as well as 18 AiBERT and GiBERT models (2 different random seeds x 3 learning rates x 3 injection locations) for each dataset.

E Injection Location for AiBERT

Based on the development set results shown in Table 9, the final AiBERT model uses injection layer 6 for both dependency and counter-fitted embeddings.

Table 8: Tuned hyper-parameters for BERT-based models. LR = learning rate. Location = injection location, length = sequence length in tokens, PAS = predicate-argument structures.

<table>
<thead>
<tr>
<th>Model</th>
<th>Layer</th>
<th>LR (1st seed)</th>
<th>LR (2nd seed)</th>
<th>Max length</th>
<th>Batch size</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td></td>
<td>5e-5</td>
<td>5e-5</td>
<td>80</td>
<td>32</td>
</tr>
<tr>
<td>SemBERT</td>
<td></td>
<td>2e-5</td>
<td>2e-5</td>
<td>48</td>
<td>32</td>
</tr>
<tr>
<td>tBERT with dependency-based embeddings</td>
<td></td>
<td>2e-5</td>
<td>2e-5</td>
<td>300</td>
<td>16</td>
</tr>
<tr>
<td>BERT with counter-fitted embeddings</td>
<td></td>
<td>2e-5</td>
<td>2e-5</td>
<td>200</td>
<td>16</td>
</tr>
<tr>
<td>AiBERT with dependency-based embeddings</td>
<td></td>
<td>3e-5</td>
<td>3e-5</td>
<td>300</td>
<td>16</td>
</tr>
<tr>
<td>AiBERT with counter-fitted embeddings</td>
<td></td>
<td>3e-5</td>
<td>3e-5</td>
<td>200</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 9: Development F1 scores of attention-based embedding injection at different layers.

F Total Parameters

<table>
<thead>
<tr>
<th>Model</th>
<th>Total parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>110.1M</td>
</tr>
<tr>
<td>GIBERT</td>
<td>110.3M</td>
</tr>
<tr>
<td>tBERT</td>
<td>111.0M</td>
</tr>
<tr>
<td>AiBERT</td>
<td>111.7M</td>
</tr>
<tr>
<td>SemBERT</td>
<td>111.9M</td>
</tr>
</tbody>
</table>

Table 10: BERT-based models ordered by size (using BERT_BASE).
G Required Injection Parameters

This section compares the number of required parameters in the two alternative injection methods discussed in section 4.1: a multihead attention injection mechanism (AiBERT) and a novel lightweight gated injection mechanism (GiBERT).

**Attention injection** In multihead attention injection (equations 3 to 4), the keys are provided by BERT’s representation from the injection layer $H^i$ and the queries are the injected information $I$. Multimodal attention requires the following weight matrices $W$ and biases $b$ to transform queries, keys and values (indicated by $Q$, $K$ and $V$) and transform the attention output (indicated by $O$):

$$
\text{params}(\text{AttentionInjection}) = \text{params}(W^K, W^Q, W^V, b^K, b^Q, b^V, W^O, b^O) = D(2D + 2E) + 4D
$$

where $D$ indicates BERT’s hidden dimension and $E$ indicates the dimensionality of the injected embeddings. When injecting embeddings with $E = 300$ into BERT$_{BASE}$ with $D = 768$, this amounts to $\approx 1.64M$ new parameters.

**Gated injection** The proposed gated injection method (equations 6 to 7) only introduces the weights and biases from the projection layer, as well as the gating vector:

$$
\text{params}(\text{GatedInjection}) = \text{params}(W^P, b^P, g) = D(E + 1) + E.
$$

Gated injection of embeddings with $E = 300$ into BERT$_{BASE}$ requires $\approx 0.23M$ new parameters.

Therefore, the proposed gated injection mechanism only requires 14% of the parameters used in a multihead attention injection mechanism. Using fewer parameters results in a smaller model which is especially beneficial for injecting information during finetuning, where small learning rates and few epochs make it difficult to learn large amounts of new parameters.

H Gating Parameter Analysis

As described in section 4.1, the gating parameters $g$ in our proposed model are initialised as a vector of zeros. During training, the model can learn to gradually inject external information by adjusting gating parameters to $>0$ for adding, or $<0$ for subtracting injected information along certain dimensions. Alternatively, injection stays turned off if all parameters remain at zero. Figure 2 shows a histogram of learned gating vectors for our best GiBERT models with counter-fitted (left) and dependency embedding injection (right). On most datasets, the majority of parameters have been updated to small non-zero values, letting through controlled amounts of injected information without completely overwriting BERT’s internal representation. Only on Semeval B (with 4K instances the smallest of the datasets, compare section 3), more
than 500 of the 768 dimensions of the injected information stay blocked out for both model variants. The gating parameters also filter out many dimensions of the dependency-based embeddings on MSRP (the second smallest dataset). This suggests that models trained on smaller datasets may benefit from slightly longer finetuning or a different gating parameter initialisation to make full use of the injected information.\textsuperscript{5}

\textsuperscript{5}Note that we train models for the same number of epochs, but one epoch uses all training examples contained in the dataset. This gives models trained on larger datasets more opportunity to update their parameters.