LEARNING LANGUAGE REPRESENTATIONS WITH LOGICAL INDUCTIVE BIAS

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformer architectures have achieved great success in solving natural language tasks, which learn strong language representations from large-scale unlabeled texts. In this paper, we seek to go further beyond and explore a new *logical inductive bias* for better language representation learning. Logic reasoning is known as a formal methodology to reach answers from given knowledge and facts. Inspired by such a view, we develop a novel neural architecture named FOLNet (**F**irst-**O**rder **L**ogic **Net**work), to encode this new inductive bias. We devise and compose several neural logic operators into a set of learnable Horn clauses, which are further *forward-chained* into a fully differentiable neural architecture (FOLNet). Interestingly, we find that the self-attention module in transformers can be composed by two of our neural logic operators, which probably explains their strong reasoning performance. Our proposed FOLNet has the same input and output interfaces as other pretrained models (e.g., BERT) and thus could be pretrained/finetuned by using similar losses. It also allows FOLNet to be used in a plug-and-play manner when replacing other pretrained models. With our logical inductive bias, the *same* set of "logic deduction skills" learned through pretraining are expected to be equally capable of solving diverse downstream tasks. For this reason, FOLNet learns language representations that have much stronger transfer capabilities. Experimental results on several language understanding tasks show that our pretrained FOLNet model outperforms the existing strong transformer-based approaches.[1]

1 INTRODUCTION

Pretrained transformer models have achieved great success in solving natural language tasks, which learn strong language representations from large-scale unlabeled texts. The learned representations can be easily transferred to different downstream tasks by tuning over limited amount of labeled data (Radford et al., 2018; Devlin et al., 2018; Lan et al., 2019; Liu et al., 2019; Yang et al., 2019). They even exhibit strong zero-shot or few-shot generalization capability without finetuning when further scaling up the model size (Radford et al., 2019; Brown et al., 2020; Chowdhery et al., 2022). Besides large-scale models and training data, one important reason for the success is the strong relational inductive bias encoded in the transformer architecture (Vaswani et al., 2017); it effectively models the pairwise relations between tokens and use it to compute the language representations.

In this paper, we seek to go beyond the inductive bias in transformer models and explore a new *logical inductive bias* for better language representation learning. The main idea is to view the computation of language representations as a logic reasoning process; that is, the language representations are deduced via logic reasoning step-by-step from the original discrete token sequences. Specifically, we treat the tokens in the input sequence as the terms in logic programming, and treat their properties and relations as the predicates of different arities. Then, the final language representations are derived as the *advanced* properties and relations from the *basic* input properties and relations (e.g., token ids and relative distances). Most importantly, we require the construction of such deduction process to follow the principles of first-order logic, in order to encode such logical inductive bias.

Following the above logical inductive bias, we derive a principled neural architecture, named FOLNet (**F**irst-**O**rder **L**ogic **Net**work), for learning language representations. Specifically, we construct a set

[1]The code along with the pretrained model checkpoints will be released publicly.

of neural logic operators as learnable Horn clauses, which are further *forward-chained* into a fully differentiable neural architecture. In particular, the FOLNet architecture consists of two interacting branches responsible for unary and binary relational reasoning, respectively. Interestingly, we find that the self-attention mechanism can be constructed by two of our developed neural logic operators, and the entire transformer architecture can be understood as a single-branch version of FOLNet. This newly discovered connection might partially explain the surprisingly strong reasoning performance of the transformer architecture (Wei et al., 2022; Lewkowycz et al., 2022). As we will demonstrate in our experiments, such dual-branch architecture has several significant advantages that are essential for learning better language representations. Furthermore, we also establish a new unified understanding of different positional encoding strategies with our logical inductive bias. For instance, we find that the existing popular relative positional encoding can be constructed by the degenerated version of our two neural logic operators. More importantly, it also allows us to develop a new principled relative positional encoding that is simple yet quite effective in practice. Notably, our proposed FOLNet has the same input and output interfaces as other pretrained transformer models (e.g., BERT) and thus could be trained by using similar losses. It also allows FOLNet to be used in a plug-and-play manner when replacing other pretrained models in solving downstream tasks. Our logical inductive bias assumes that the "logic deduction skills" are shared across all natural language tasks; that is, these skills learned during pretraining should be equally applicable to solving diverse downstream tasks. For this reason, FOLNet learns language representations that have much stronger transfer generalization capabilities. Experimental results on several language understanding tasks (GLUE, SQuAD 2.0 and FOLIO) show that our FOLNet model outperforms the transformer architecture by a large-margin when they are pretrained using similar losses. The results clearly show that advantage of using the logical inductive bias for learning language representations.

2 Logical Inductive Bias for Language Representations

Natural language text can be viewed as a sequence of discrete symbols, and language representations learning considers the problem of mapping the discrete symbols into certain computable forms. One widely used approach is distributed representation, which maps the discrete token ids into dense vectors (Mikolov et al., 2013; Pennington et al., 2014; Peters et al., 2018). Many different functional forms, such as BiLSTM (Hochreiter & Schmidhuber, 1997), and more recently, transformer models (Vaswani et al., 2017), have been used to implement such mapping. They generally encode different kinds of inductive bias for modeling natural languages. For example, RNNs use the same set of model parameters to update the hidden states over time, which encodes translation-invariance over time (Battaglia et al., 2018). These forms of inductive bias continuously push the state-of-the-arts in solving natural language tasks. In this section, we introduce a new form of inductive bias, named *logical inductive bias*, which will work together with distributed representations to design more effective representation mappings. Our main idea is to view the language representation mapping as a logic reasoning process; that is, the language representations are deduced step-by-step from the original discrete token sequences. Specifically, we treat the tokens in the input sequence as terms (or objects) in logic programming, and treat their properties and relations as predicates of different arities. In light of logical inductive bias, the language representations that we seek to compute are the (advanced) properties and relations that can be deduced from these input (basic) properties and relations. Most importantly, we require the construction of such deduction process to follow the principles of first-order logic, in order to encode the logical inductive bias into the representation learning process. We now formulate the language representation learning as a logic programming problem by using similar notations as (Evans & Grefenstette, 2018).

- **Terms:** We consider a first-order logic system without function symbols, so that terms can only be variables or constant. They are used to represent general objects or a particular object of interest, respectively. In the context of language representation learning, we model each instance of text sequence x (of length T) as a collection of constants $x = \{x_1, \ldots, x_T\}$, where each token x_t is a constant ($t = 1, \ldots, T$). We use lower-case letters to denote constants and upper case for variables as in logic programming. For example, X is a variable to represent a general object (e.g., token).
- **Atoms:** For each term, we will define its properties and relations as an r-ary predicate $p(X_1, \ldots, X_r)$, which takes the value of T (True) or F (False) depending on whether a certain property/relation regarding $(X_1, \ldots, X_r)$ holds or not. For example, whether the a token a takes the v-th id in the vocabulary is a unary predicate $\texttt{TokenID}_v(a)$ for $v = 1, \ldots, V$, where V

Language representations	**Logic programming**	**FOLNet**
Tokens x_t in the text sequence	Constant x_t	Tensor index t
Token ids, relative distance, etc	Input (basic) atoms $\mathcal{C}_0 = \mathcal{B}$	Input tensors $\{\mathbf{u}_0(x), \mathbf{u}_0(x,y)\}$
Final representation	Deduced (advanced) atoms $\mathcal{C}_L$	Output tensors $\{\mathbf{u}_L(x), \mathbf{u}_L(x,y)\}$
Representation mapping (partial)	Apply one clause via Modus Ponens (5)	Logic operator
Representation mapping (partial)	Apply all clauses via $\mathcal{C}_l = \texttt{con}_{\mathcal{R}}(\mathcal{C}_{l-1})$	Forward pass (1-layer)
Representation mapping (full)	Forward-chaining of clauses via (3)	Forward pass (full model)

Table 1: Identification of language representation learning as a logic programming problem.

is the vocabulary size, and whether the distance between two tokens a and b is equal to d is a binary predicate $\texttt{Dist}_d(a,b)$, for $|d| < T$. An atom is *ground* if it has no variables, e.g., the above $\texttt{TokenID}_v(a)$ and $\texttt{Dist}_d(a,b)$ are all ground atoms.

- **Clauses:** The reasoning process is constructed by a set of "if-then" clauses in the form of:

$$q \leftarrow p_1 \wedge \cdots \wedge p_m, \tag{1}$$

where $p_1, \ldots, p_m$ and q are the body atoms and head atoms, respectively, and $\wedge$ denotes conjunction (i.e., logical AND). They play the roles of premises and conclusions: if $p_1, \ldots, p_m$ are true, then q is also true. Clauses of the above form are known as *definite Horn clauses* (Horn, 1951). We call a clause *a ground rule* if all its variables are substituted by constants. For example, when applying a substitution $\theta = \{a/X, b/Y\}$ to a clause $q(X,Y) \leftarrow p(X,Y)$, we get a ground rule: $q(a,b) \leftarrow p(a,b)$. It can be viewed as applying a general clauses to a particular instantiation.

Our objective is to learn a collection of clauses and compose them into a mapping from input predicates (e.g., $\texttt{TokenID}_v(x_t)$ and $\texttt{RelDist}_d(x_t, x_\tau)$) to language representations. Specifically, let $\mathcal{R}$ be a set of clauses and $\texttt{ground}(\mathcal{R})$ be the corresponding set of ground rules. We define the *immediate consequences* of applying the ground rules in $\texttt{ground}(\mathcal{R})$ to a set of ground atoms $\mathcal{X}$ as

$$\texttt{con}_{\mathcal{R}}(\mathcal{X}) = \mathcal{X} \cup \Big\{ q \Big| q \leftarrow p_1 \wedge \cdots \wedge p_m \in \texttt{ground}(\mathcal{R}), \bigwedge_{i=1}^{m} p_i \in \mathcal{X} \Big\}. \tag{2}$$

It can be understood as a set of ground atoms that can be deduced from $\mathcal{X}$ together with $\mathcal{X}$ itself. Given a set of input ground atoms $\mathcal{B}$, we can repeatedly apply the ground rules in $\mathcal{R}$ for L steps:

$$\mathcal{C}_l = \texttt{con}_{\mathcal{R}}(\mathcal{C}_{l-1}), \qquad \mathcal{C}_0 = \mathcal{B} \text{ and } l = 1, \ldots, L. \tag{3}$$

Then, $\mathcal{C}_L$ is all the possible ground atoms (predicates) that can be deduced from $\mathcal{B}$ (including $\mathcal{B}$ itself) in L steps. The above procedure is known as **forward-chaining**: it deduces all the possible conclusions from the input premises (i.e., $\mathcal{B}$) by repeatedly applying (i.e., chaining) clauses. If we want to verify whether a predicate q' holds (i.e., can be *entailed*), it suffices to check if q' is in $\mathcal{C}_L$. In language representation learning, we start, for example, from the following input (basic) atoms

$$\mathcal{B} = \{\texttt{TokenID}_v(x_t), \texttt{RelDist}_d(x_t, x_\tau), \ldots | t, \tau = 1, \ldots, T\}, \tag{4}$$

and deduce $\mathcal{C}_L$ as the final representations by forward-chaining our (learned) clauses. For example, in solving a question answering problem, whether a certain token is the beginning (or end) of the answer span is modeled as an advanced deduced property of this token. Table (1) summarizes the above identifications between language representations and logic programming. Next, we will develop a neural architecture to encode the above logical inductive bias. Throughout the paper, we will use bold letters to denote vectors (lowercase) and matrices/tensors (uppercase).

3 FOLNet: A Neural Architecture with Logical Inductive Bias

In this section, we develop a novel neural architecture, named **F**irst-**O**rder **L**ogic **Net**work (FOLNet), which encodes the logical inductive bias presented in Section 2. Specifically, we focus on: (i) how to represent the atoms in a continuous vector form (Section 3.1), (ii) how to devise a neural inference step that approximates (2) (Section 3.2), and (iii) how to forward-chain them into a fully-differentiable architecture based on (3) (Section 3.3). The overall neural architecture and its correspondence to the logical inductive bias are shown in Figure 1. Next, we will discuss step-by-step how we devise the architecture, its advantages, and also some interesting connections to existing approaches.

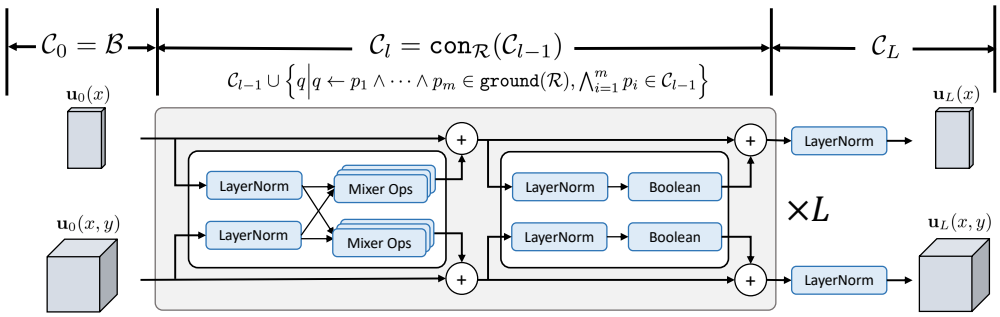

Figure 1: Overview of the FOLNet architecture and how it encodes the logical inductive bias. The neural logic operators model the clauses, which are forward-chained into a differentiable architecture. We call operators c, j, m, and t in Table 2 mixer operators as they reduce over the object dimension.

3.1 Vector Representations of Atoms

Recall that a ground atom is an r-ary predicate $p_d(x_1, \ldots, x_r)$ taking the values of T or F, where $d = 1, \ldots, D_r$. In our language representation context, it characterizes whether the d-th property or relation holds for a particular tuple of tokens $(x_1, \ldots, x_r)$. To overcome the difficulty of learning discrete-valued $p_d(x_1, \ldots, x_r)$, we introduce $u_d(x_1, \ldots, x_r) \in \mathbb{R}$ as its continuous representation and characterizes the extent to which the atom p_d is true. For example, in ProbLog (De Raedt et al., 2007; De Raedt & Kimmig, 2015; Fierens et al., 2012; Manhaeve et al., 2018), $u_d(\cdot)$ gives the probability of the atom p_d being true. Let $\mathbf{u}(x_1, \ldots, x_r) \in \mathcal{U} \subset \mathbb{R}^{D_r}$ be a D_r-dimensional vector that collects $u_d(x_1, \ldots, x_r)$ for $d = 1, \ldots, D_r$ as its r-th component. Then, we will have $\mathbf{u}(x_1, \ldots, x_r)$ as a continuous vector representation of all the r-ary atoms, which, in our context, means all the r-ary properties/relations for a particular tuple of tokens $(x_1, \ldots, x_r)$. For example, for an input text sequence $x = (x_1, \ldots, x_T)$, we can use $\mathbf{u}(x_t)$ to represent all the (unary) properties for each token x_t, and use $\mathbf{u}(x_t, x_\tau)$ to characterize all the (binary) pairwise relations between any pair of tokens. Note that both the input (basic) properties/relations and the deduced (advanced) ones are represented in the same continuous-vectors forms, where the deduction process will be carried out. For convenience, we may also refer to a set of atoms by their vector representation $\mathbf{u}(\cdot)$.

3.2 Neural Modus Ponens Inference

We now develop a set of neural operators for implementing the deduction step characterized in (2). We begin with the introduction of the *Modus Ponens* (MP) rule from first-order logic Andrews (2013), which states that if clause $B \leftarrow A$ and statement A hold, then B is true:

$$B \Leftarrow \{B \leftarrow A \text{ and } A\}. \tag{5}$$

In the context of (2), when choosing $A = p_1 \wedge \cdots \wedge p_m$ and $B = q$, the deduction in (2) can be viewed as the applications of MP inference using all the ground clauses in $\mathtt{ground}(\mathcal{R})$. In this paper, we restrict our focus to the setting where all the atoms have arity of either one or two. That is, we will only consider all the atoms of the form $\mathbf{u}(x_t)$ and $\mathbf{u}(x_t, x_\tau)$ (represented in their vector forms), respectively.[2] Then, we will need to develop the MP inference from a set of atoms $\{\mathbf{v}(x_t), \mathbf{v}(x_t, x_\tau)\}$ to another set of atoms $\{\mathbf{u}(x_t), \mathbf{u}(x_t, x_\tau)\}$. Specifically, we need the following four groups of MPs:

$$\mathbf{u}(x) \Leftarrow \mathcal{R}_{\mathtt{UU}}, \mathbf{v}(a); \quad \mathbf{u}(x) \Leftarrow \mathcal{R}_{\mathtt{UB}}, \mathbf{v}(a,b); \quad \mathbf{u}(x,y) \Leftarrow \mathcal{R}_{\mathtt{BU}}, \mathbf{v}(a); \quad \mathbf{u}(x,y) \Leftarrow \mathcal{R}_{\mathtt{BB}}, \mathbf{v}(a,b);$$

where $\mathcal{R}_{\mathtt{UU}}$, $\mathcal{R}_{\mathtt{UB}}$, $\mathcal{R}_{\mathtt{BU}}$ and $\mathcal{R}_{\mathtt{BB}}$ denote the sets of rules that maps between atoms of different arities, and the subscripts U and B stand for unary and binary, respectively. For example, $\mathcal{R}_{\mathtt{UU}}$ defines a collection of clauses that can deduce the set of atoms represented by $\mathbf{u}(x)$ from another set of atoms represented by $\mathbf{v}(a)$. Notably, we approximate the above MP inferences by the following expressions:

$$\mathbf{u}(x) = \sum_a \mathbf{K}_{\mathtt{UU}}(x,a)\mathbf{v}(a) \tag{6}$$

[2]Generalizing our work to higher arities is relatively straighforward in principle but will lead to high computation complexities in pracitce. We leave such an extension as a future work.

Sym.	Ops.	Typing	B-dim.	R-dim.	Definition of the operator	Kernel act.	Remarks
b	bool	$\mathtt{U} \leftarrow \mathtt{U} \times \mathtt{U}$	x	w	$\mathbf{u}_{hs}(x) = \sum_w \mathbf{K}_{hw}(x)\mathbf{v}_{ws}(x)$	GeLU/Identity	
c	cjoin	$\mathtt{U} \leftarrow \mathtt{U} \times \mathtt{B}$	h	a	$\mathbf{u}_{hs}(x) = \sum_a \mathbf{K}_{hs}(a)\mathbf{v}_h(x,a)$	Softmax_a	
j	join	$\mathtt{U} \leftarrow \mathtt{B} \times \mathtt{U}$	h	a	$\mathbf{u}_{hs}(x) = \sum_a \mathbf{K}_h(x,a)\mathbf{v}_{hs}(a)$	Softmax_a	self-attention
m	mu	$\mathtt{U} \leftarrow \mathtt{B} \times \mathtt{B}$	x	a	$\mathbf{u}_{hs}(x) = \sum_a \mathbf{K}_h(x,a)\mathbf{v}_s(x,a)$	Softmax_a	general RPE
a	assoc	$\mathtt{B} \leftarrow \mathtt{U} \times \mathtt{U}$	h	w	$\mathbf{u}_h(x,y) = \sum_w \mathbf{K}_{hw}(x)\mathbf{v}_{hw}(y)$	GeLU/Identity	self-attention
p	prod	$\mathtt{B} \leftarrow \mathtt{U} \times \mathtt{B}$	x	w	$\mathbf{u}_h(x,y) = \sum_w \mathbf{K}_{hw}(x)\mathbf{v}_w(x,y)$	GeLU/Identity	general RPE
t	trans	$\mathtt{B} \leftarrow \mathtt{B} \times \mathtt{B}$	h	a	$\mathbf{u}_h(x,y) = \sum_a \mathbf{K}_h(x,a)\mathbf{v}_h(a,y)$	Softmax_a	

Table 2: List of all our neural logic operators with restricted kernels (see Appendix A.1 for our naming protocols). "B-dim." stands for the broadcast dimension and "R-dim." means the reduction dimension. Note that each operator will have a unique broadcast dimension and a unique reduction dimension. The typing of the operator defines type of the input kernel and atoms along with the output atoms. For example, $\mathtt{U} \leftarrow \mathtt{B} \times \mathtt{U}$ means the arities of the kernel, the input atom, and the output atom are 2, 1 and 1, respectively. We also list the activation functions that are used to compute the corresponding kernels, where the normalization dimension of Softmax is listed in its subscript.

$$\mathbf{u}(x) = \sum_{a,b} \mathbf{K}_{\mathtt{UB}}(x,a,b)\mathbf{v}(a,b) \tag{7}$$

$$\mathbf{u}(x,y) = \sum_{a} \mathbf{K}_{\mathtt{BU}}(x,y,a)\mathbf{v}(a) \tag{8}$$

$$\mathbf{u}(x,y) = \sum_{a,b} \mathbf{K}_{\mathtt{BB}}(x,y,a,b)\mathbf{v}(a,b) \tag{9}$$

where $\mathbf{K}_{\mathtt{UU}}(x,a)$, $\mathbf{K}_{\mathtt{UB}}(x,a,b)$, $\mathbf{K}_{\mathtt{BU}}(x,y,a)$ and $\mathbf{K}_{\mathtt{BB}}(x,y,a,b)$ are $D_1 \times D_1$, $D_1 \times D_2$, $D_2 \times D_1$ and $D_2 \times D_2$ matrices that correspond to $\mathcal{R}_{\mathtt{UU}}$, $\mathcal{R}_{\mathtt{UB}}$, $\mathcal{R}_{\mathtt{BU}}$ and $\mathcal{R}_{\mathtt{BB}}$, respectively. In fact, these matrices are the tensor representations of their corresponding set of clauses, similar to the vector representations for atoms in Section 3.1. Therefore, $\mathbf{K}_{\mathtt{UU}}(x,a)$, $\mathbf{K}_{\mathtt{UB}}(x,a,b)$, $\mathbf{K}_{\mathtt{BU}}(x,y,a)$ and $\mathbf{K}_{\mathtt{BB}}(x,y,a,b)$ can be interpreted as the kernels that implement the clauses in $\mathcal{R}_{\mathtt{UU}}$, $\mathcal{R}_{\mathtt{UB}}$, $\mathcal{R}_{\mathtt{BU}}$ and $\mathcal{R}_{\mathtt{BB}}$, respectively. We will call them the *kernel-of-clauses* (or *kernels* for short) in the remainder of the paper.

One major limitation of directly implementing the above MP inference operations (6)–(9) is the high computation and memory costs. For example, the kernel $\mathbf{K}_{\mathtt{BB}}(x,y,a,b)$ needs $O(D_2^2T^4)$ to store its value. When properly reshaping $\mathbf{K}$ and $\mathbf{v}$ on the right-hand-side of (6)–(9) and jointly considering the summation and matrix multiplication, each expression in (6)–(9) can be equivalent to one huge matrix multiplication. For example, expression (9) is equivalent to a matrix multiplication between two $T^2D_2 \times T^2D_2$ matrices, which has a computation complexity of $O(D_2^2T^4)$. Note that the reduction dimension of such a matrix multiplicaiton is T^2D_2, which is also huge in computation. Therefore, we need to reduce the overall complexity to make them practically feasible, and there could be different ways to achieve this, such as restricting the size of $\mathbf{K}_{\mathtt{BB}}(x,y,a,b)$ by sharing the values across different dimensions. Here we provide a systematic approach to achieve this objective, which are based on the following assumptions and principles:

1. We restrict all the kernels to be in the form of $\{\mathbf{K}(\omega), \mathbf{K}(\omega,\nu)\}$, i.e., the arity $r = 1, 2$.
2. We pick one dimension for reduction and another for broadcasting in the matrix multiplication.

With the above assumption, we factor the predicate dimension of the unary kernel and the unary atom so that $\mathbf{u}_d(x) = \mathbf{u}_{hs}(x)$ and $\mathbf{K}_d(\omega) = \mathbf{K}_{hs}(\omega)$, where $d = (h-1)S + s$ with $h = 1, \ldots, H$ and $s = 1, \ldots, S$. This is inspired by the multi-head attention mechanism (Vaswani et al., 2017), where h is akin to the head index, H is the number of heads and S is the size of the head. Then, we can enumerate all possible operators that can be compatibly multiplied between the kernel set $\{\mathbf{K}(\omega), \mathbf{K}(\omega,\nu)\}$ and the premise set $\{\mathbf{v}(a), \mathbf{v}(a,b)\}$ by properly choosing the reduction dimension and the broadcasting dimension. With this, we list the resulting logic operators in Table 2.

Interestingly, we find that the j-operator and the a-operator share the same form as the self-attention mechanism in transformer architectures, where the a-operator computes the self-attention scores while the j-operator performs the self-attention operation. Furthermore, the m-operator and the p-operator indeed generalize the existing relative positional encodings developed (Shaw et al., 2018), which are widely used in different transformer architectures, such as in T5 models (Raffel et al., 2020).

Specifically, with some simple derivation, we can show that by making $\mathbf{v}_w(x, y)$ instance-independent (i.e., does not have the batch-dimension), the p-operator computes the second term in equation (5) of Shaw et al. (2018), where $\mathbf{v}_w(x, y)$ play the role of a_{ij}^K. And by setting $\mathbf{v}_s(x, a)$ instance-independent, the m-operator computes the second term in equation (3) of Shaw et al. (2018), where $\mathbf{v}_s(x, a)$ plays the role of a_{ij}^V. That is, under such *degenerated* settings, these two operators will compose the relative positional encoding developed therein. Note that their a_{ij}^K and a_{ij}^V are static learnable embeddings while our $\mathbf{v}_w(x, y)$ and $\mathbf{v}_s(x, a)$ can indeed be dynamic and instance-dependent. As we will show in Section (3.3), they are dynamically computed for each instance in FOLNet. Therefore, our m-op and p-op can be viewed as a more adaptive relative positional encoding, which is also layer-dependent in the network. Our experiments in Section 4 will show the advantage of such "adaptive" RPEs.

3.3 Forward-chaining and differentiable learning

Note that, in general, a logic operator takes a kernel $\mathbf{K}$ and a premise $\mathbf{v}$ to computes a conclusion atom $\mathbf{u}$, as shown in Table 2. Specifically, it "neuralize" the logic deduction in (2) for a particular typing (e.g., $\mathtt{U} \leftarrow \mathtt{B} \times \mathtt{U}$) shown in Table 2. Applying all the neural logic operators amounts to have a full execution of (2), which is one recursion step in (3) that maps a set of $\{\mathbf{u}_{l-1}(x), \mathbf{u}_{l-1}(x, y)\}$ into $\{\mathbf{u}_l(x), \mathbf{u}_l(x, y)\}$. Therefore, we can naturally forward-chain them together by L stages to create a fully-differentiable architecture (i.e., FOLNet) that models the reasoning chain in (3). Figure 1 depicts such forward-chaining process and also how it encodes the logical inductive bias described in Section 2. One remaining problem is how to obtain the kernels $\mathbf{K}(\cdot)$ and the premises $\mathbf{v}(\cdot)$ from our FOLNet architecture in Figure 1. We do this simply by applying two linear projections (one for the premise and one for the kernel) to the previous layer's output $\{\mathbf{u}_l(x), \mathbf{u}_l(x, y)\}$. For the kernel, we may further apply an activation function after the linear projection to compute the kernel (see Table 2 for the list of kernel activation functions for each operator). In other words, we parameterize the kernels $\mathbf{K}(\cdot)$ and the premises $\mathbf{v}(\cdot)$ by (the intermediate deduction results of) FOLNet itself. This is based on the observation that clauses are themselves atoms since $A \leftarrow B$ is defined as $A \vee \neg B$ in first-order logic (Andrews, 2013). We now describe the input and the output of FOLNet in Figure 1. At the input, we can encode the discrete token ids for a token x_t into vectors of the form $\mathbf{u}_0(x_t)$ by standard embedding lookup. Likewise, we also convert the (discrete) relative distance between two tokens x_t and x_τ into a vector of the form $\mathbf{u}_0(x_t, x_\tau)$. The $\{\mathbf{u}_0(x_t), \mathbf{u}_0(x_t, x_\tau)\}_{t,\tau}$ will be used as vector representation of the base atoms $\mathcal{B}$ and fed into the FOLNet model (Figure 1). After L layers (i.e., L steps of deduction), the output $\{\mathbf{u}_L(x_t), \mathbf{u}_L(x_t, x_\tau)\}_{t,\tau}$ becomes the vector representation of $\mathcal{C}_L$ in (3), which is used as the final language representations. Therefore, our FOLNet model has the same input-output interface as other transformer models and can be used in a plug-and-play manner for solving downstream tasks. Because of this, our model can also be pretrained over large-scale texts using the same losses (e.g., MLM, NSP, SOP, etc) as the other models. We will conclude this section by discussing several important properties of FOLNet architecture.

The dual-branch architecture Note that the FOLNet model in Figure 1 has two branches: (i) a unary predicate branch for reasoning over $\mathbf{u}_l(x)$, and (ii) a binary predicate branch for reasoning over $\mathbf{u}_l(x, y)$. This is in sharp contrast to the single-branch architecture of the transformer models (Figure 2 in Appendix A.2). Note that when FOLNet is only loaded with j-operator and a-operator, it degenerated into a dual-branch variant of the transformer architecture. In our experiments, we will show that, even in such degenerated setting, FOLNet still outperforms transformer. This is probably because the binary predicate branch explicitly maintains the binary pairwise relations $\mathbf{u}_l(x, y)$ throughout the reasoning process. In addition, the explicit binary predicate branch also allows us to directly input the relative distance into the model without performing less-intuitive operations as in existing RPEs (Shaw et al., 2018). In our experiments in Section 4, we will demonstrate the advantage of such a simple yet effective approach for consuming the relative positional information. In Section 4, we will further demonstrate other advantages of the dual-branch architecture of FOLNet.

4 Experiments

4.1 Experimental settings

We now evaluate our FOLNet models under different settings and seek to answer the following question: *Can the neural architecture (FOLNet) that encodes the logical inductive bias learns better*

	Model	PE	D_2	Loss	MNLI-m/mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg
1	BERT	APE	-	MLM.NSP	84.5/-	91.3	91.7	93.2	58.9	89.5	87.3	68.6	83.1
2	FOLNet: j.a	APE	12	MLM.NSP	84.9/84.5	91.1	91.6	92.2	61.1	89.6	89.5	72.2	84.0
3	FOLNet: j.a	RPE	12	MLM.NSP	85.0/84.9	91.4	91.4	93.5	63.8	90.1	89.9	72.9	84.7
4	FOLNet: j.a	RPE*	12	MLM.NSP	84.7/84.9	91.4	91.5	93.8	63.4	89.8	90.6	72.6	84.7
5	FOLNet: jm.ap	RPE	12	MLM.NSP	**85.7/85.3**	**91.6**	**91.8**	**93.8**	**65.7**	**90.4**	**91.0**	**73.5**	**85.4**
6	FOLNet: j.a	RPE	16	MLM.NSP	85.2/85.2	91.3	91.8	93.7	64.1	89.9	89.3	71.8	84.6
7	FOLNet: j.a	RPE	32	MLM.NSP	**85.8/85.7**	91.4	**92.0**	**93.7**	**64.2**	90.0	**90.5**	71.8	84.9
8	FOLNet: j.a	RPE	64	MLM.NSP	85.7/85.5	**91.4**	91.8	93.2	63.5	**90.1**	90.2	**74.7**	**85.1**
9	FOLNet: j.at	RPE	64	MLM.NSP	86.7/86.6	91.6	92.8	93.1	63.6	91.1	89.9	80.9	86.2
10	FOLNet: j.atp	RPE	64	MLM.NSP	87.4/87.4	91.9	93.3	94.0	62.9	91.3	91.4	81.6	86.7
11	FOLNet: jm.atp	RPE	64	MLM.NSP	88.1/87.6	91.7	93.9	94.2	64.7	91.2	91.4	83.2	87.3
12	FOLNet: jmc.atp	RPE	64	MLM.NSP	**88.2/87.9**	**91.9**	**94.1**	**94.5**	**66.9**	**91.6**	**91.5**	**83.5**	**87.7**
13	FOLNet: jmc.atp	RPE	12	MLM.NSP	85.9/86.3	91.6	92.7	93.6	63.4	90.5	91.0	75.8	85.6
14	FOLNet: jmc.atp	RPE	64	MLM.SOP	88.3/87.9	91.8	94.2	94.7	65.6	91.1	91.1	83.2	87.5

Table 3: Analysis of FOLNet on the development sets of GLUE benchmark. All the results are medians of five random seeds. From top to bottom, the first block shows the advantage of dual-branch architecture and compares our new positional encoding to others, the second block analyzes the influence of the binary predicate dimension, the third block performs ablation study of all the logic operators, and the last block shows the results of other pretraining losses for FOLNet. APE stands for absolute positional encoding, RPE* denotes the relative positional encoding used by T5 (Raffel et al., 2020), and RPE means our proposed relative positional encoding.

language representations than the transformer models? To this end, we need to eliminate other confouding factors and make sure the only difference lies in the model architecture itself. First, we choose to pretrain our FOLNet model using the widely used masked language modeling (MLM) loss (Devlin et al., 2018), and add an extra loss of either the next sentence prediction (NSP) (Devlin et al., 2018) or sentence-order prediction (SOP) (Lan et al., 2019). Many different variants of widely encoder-only transformer models such as BERT, RoBERTa, ALBERT, DeBERTa and Megatron-LM are pretrained with these losses. Therefore, we will also use these models as our primary baselines. Although there could be other more efficient pretraining losses such as the ones in (Bao et al., 2020) (Clark et al., 2020; Yang et al., 2019; Meng et al., 2021), we believe that developing a new model architecture with a better inductive bias is an orthogonal line of research. Therefore, we leave the exploration of other pretraining loss for FOLNet as a future work. In addition, we consider two settings of pretraining dataset: (i) Wikipedia + BookCorpus (Zhu et al., 2015) (16GB in texts) and (ii) a larger set of 160GB texts consisting of Wikipedia, BookCorpus2, OpenWebText2, and Pile-CC (extracted from the Pile dataset (Gao et al., 2020)). We use the BERT tokenizer with 32,768 uncased BPE vocabulary (Sennrich et al., 2016) throughout our experiments.[3] We consider FOLNet models of two different sizes: $\text{FOLNet}_{\text{Base}}$ and $\text{FOLNet}_{\text{Large}}$, which are comparable in size to the *base* (e.g., $\text{BERT}_{\text{Base}}$) and *large* models (e.g., $\text{BERT}_{\text{Large}}$) in literature. Finally, the FOLNet model will always be pretrained with a sequence length of 128 tokens, although it will be evaluated on different downstream tasks with longer sequence lengths (e.g., 384 or 512). For evaluation, we consider three benchmarks: GLUE Wang et al. (2019), SQuAD 2.0 (Rajpurkar et al., 2016b), and FOLIO (Han et al., 2022). More details about these downstream tasks and hyper-parameters can be found in Appendix B.

4.2 Analysis of the FOLNet architecture

We begin with in-depth analysis of FOLNet to demonstrates its advantage over the transformer architecture. To this end, we first pretraining our $\text{FOLNet}_{\text{Base}}$ model on the dataset of Wikipedia and BookCorpus, which is the same as the one used by BERT. And we further use MLM and NSP as our pretraining losses to make it consistent with BERT. We analyze FOLNet from different aspects on GLUE benchmark and report the results in Table 4. We now proceed to discuss the results below.

[3]Although it was pointed out that (Liu et al., 2019) it would be more ideal to use the byte-level BPE for preprocessing the much larger 160GB texts, we use the same BERT tokenizer (based on character-level BPE) to process the 160GB text to simplify our logistics, and it already demonstrates the strong performance of our models. Using the byte-level BPE tokenizer with a larger 50K vocabulary as in (Liu et al., 2019; Radford et al., 2019; Brown et al., 2020) may further improve our FOLNet models that are pretrained on the 160GB texts.

Model	Params	GLUE									SQuAD 2.0		FOLIO
		MNLI-m/mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg	EM	F1	Acc
$\text{BERT}_{\text{Base}}$	110M	84.5/-	91.3	91.7	93.2	58.9	89.5	87.3	68.6	83.1	73.7	76.3	57.8
$\text{RoBERTa}_{\text{Base}}$	110M	85.8/85.5	91.3	92.0	93.7	60.1	88.5	87.3	68.2	83.3	77.7	80.5	-
$\text{DeBERTa}_{\text{Base}}$	134M	86.3/86.2	-	-	-	-	-	-	-	-	79.3	82.5	-
$\text{FOLNet}_{\text{Base}}$	138M	**88.2/87.9**	**91.9**	**94.1**	**94.5**	**66.9**	**91.6**	**91.5**	**83.5**	**87.7**	**84.7**	**87.9**	**64.2**
$\text{BERT}_{\text{Large}}$	340M	86.6/-	91.3	92.3	93.2	60.6	90.0	88.0	70.4	84.1	79.0	81.8	62.3
$\text{RoBERTa}_{\text{Base}}$	125M	87.6/-	91.9	92.8	94.8	63.6	91.2	90.2	78.7	86.4	80.5	83.7	**64.7**
$\text{DeBERTa}_{\text{Base}}$	134M	88.8/88.5	-	-	-	-	-	-	-	-	83.1	86.2	-
$\text{FOLNet}_{\text{Base}}$	138M	**89.4/89.7**	**92.2**	**94.4**	**95.6**	**69.9**	**92.5**	**92.0**	**87.0**	**89.2**	**85.5**	**88.6**	64.2
$\text{RoBERTa}_{\text{Large}}$	356M	90.2/90.2	92.2	94.7	96.4	68.0	92.4	90.9	86.6	88.9	86.5	89.4	67.7
$\text{DeBERTa}_{\text{Large}}$	384M	91.1/91.1	92.3	95.3	96.8	70.5	-	-	-	-	88.0	90.7	-
$\text{ALBERT}_{\text{XXL}}$	235M	90.4/-	92.0	95.2	96.8	68.7	92.7	90.2	88.1	89.3	87.2	89.9	-
$\text{ALBERT}_{\text{XXL+}}$	235M	90.8/-	92.2	95.3	96.9	71.4	**93.0**	90.9	89.2	89.9	87.4	90.2	-
$\text{FOLNet}_{\text{Large}}$	437M	**91.2/91.3**	**92.5**	**95.8**	**96.8**	**71.5**	92.2	**93.5**	**91.1**	**90.6**	**88.5**	**91.5**	**70.6**
$\text{Megatron}_{1.3\text{B}}$	1.3B	90.9/91.0	92.6	-	-	-	-	-	-	-	87.1	90.2	-
$\text{Megatron}_{3.9\text{B}}$	3.9B	91.4/91.4	92.7	-	-	-	-	-	-	-	88.5	91.2	-

Table 4: Overall results on the development sets of GLUE, SQuAD 2.0 and FOLIO. The upper block (separated by the solid line) of the table shows the results of the models pretrained on Wikipedia + BookCorpus (16GB), and the lower block are the models pretrained on extended data (160GB). We use dashed lines to separate models of different sizes within each block. All the results are medians of five random seeds. The baseline results of FOLIO are provided by the authors of Han et al. (2022). Here we use $\text{ALBERT}_{\text{XXL}}$ to refer to the ALBERT model pretrained by 1M steps and use $\text{ALBERT}_{\text{XXL+}}$ to refer to the $\text{ALBERT}_{\text{XXL}}$ modeled pretrained by 1.5M steps.

The advantage of the dual-branch architecture Recall that when FOLNet only has the join and assoc operators, it can be viewed as a dual-branch version of the transformer architecture. To have a fair comparison, we pretrain a FOLNet model with the same absolute positional encoding (APE) as BERT (line #2 of Table 4). Note that, even in such overly degenerated case, FOLNet still noticeably outperforms BERT on average. When equipping FOLNet with our new relative positional encoding (RPE) (line #3 of Table 4), we will outperform BERT by 2 points on average. Notably, it achieves on par (or slightly better) average performance compared to the one with T5 relative positional encoding ($\text{RPE}^\star$ in line #4). As we discussed in earlier section, the T5 RPE are degenerated version of our mu and prod operators. Line #5 of Table 4 show that adding mu and prod operators to the FOLNet would further improve the performance by a noticeable amount.

The benefits of a larger D_2 We can see (lines #6-7 of Table 4) that increasing the dimension D_2 will steadily improves the performance. As we will show soon, when FOLNet is fully loaded with all the operators, having a larger D_2 is essential to unleash their full power; that is, the performance improvement from increasing D_2 would be even larger in for fully-loaded FOLNet.

Contribution of the logic operators We now analyze the contributions of the logic operators by adding them one-by-one into FOLNet until fully loaded. We see from line #8 to line #12 in Table 4 that this drastically improves the average performance. In line #13, we evaluate a fully-loaded FOLNet with D_2 decreased from 16 to 12, which shows a significant performance drop. This confirms the importance of having a relatively large D_2 in order to store the deduced relations.

4.3 Overall performance

Having closely examined various aspects of FOLNet architecture, we now proceed to evaluate it comprehensively on three benchmarks (GLUE, SQuAD 2.0, and FOLIO). We will also examine its performance when we further scale up the pretraining data size and model size. To pretrain the model on a larger (160GB) dataset, we find it more efficient to generate the pretraining data with SOP losses. This is because NSP losses require us to sample negative sentences from another document. To begin with, we verify the performance by pretraining a $\text{FOLNet}_{\text{Base}}$ with SOP loss on Wikipedia and BookCorpus. The result (line #14 in Table 9) shows that it could slightly degrade the performance compared to the once with NSP (line #12). However, this is relatively tolerable given its convenience when pretraining on a large corpus. Therefore, we will replace NSP with SOP when pretraining

$\text{FOLNet}_{\text{Base}}$ and $\text{FOLNet}_{\text{Large}}$ on the 160GB dataset. We show our results on the GLUE benchmark in Table 2 and compare them with other baseline methods. Observe that our $\text{FOLNet}_{\text{Base}}$ models pretrained on both 16GB data and 160GB significantly outperform other transformer-based models on all three benchmarks. Our $\text{FOLNet}_{\text{Base}}$ pretrained on 16GB data outperforms $\text{BERT}_{\text{Large}}$ model that is $3\times$ larger in model size. In addition, it even outperforms $\text{RoBERTa}_{\text{Base}}$ that is pretrained on 160GB data. When pretraining our $\text{FOLNet}_{\text{Base}}$ model on 160G data, we even surpass the $\text{RoBERTa}_{\text{Large}}$ model (89.3 vs 88.9) on GLUE benchmark. Likewise, our $\text{FOLNet}_{\text{Large}}$ model also significantly outperforms all other baselines. It even outperforms $\text{ALBERT}_{\text{XXL+}}$ that is pretrained by 1.5M steps (i.e., 50% more tokens during pretraining). Notably, our $\text{FOLNet}_{\text{Large}}$ model achieves comparable performance as the $\text{Megatron}_{\text{3.9B}}$ model on both GLUE and SQuAD 2.0 benchmark, which is 10 times larger than our model. Furthermore, although our FOLNet model is not designed for solving reasoning problem (but incorporating reasoning as an inductive bias at the token-level), our model still consistently demonstrate stronger first-order logic reasoning capability on FOLIO task (e.g., $+3.9$ over $\text{RoBERTa}_{\text{Large}}$). Finally, we would like to highlight that all the $\text{FOLNet}_{\text{Base}}$ and $\text{FOLNet}_{\text{Large}}$ models are pretrained with sequence length of 128, in contrast to 512 as in other baselines. However, they are evaluated on GLUE, SQuAD 2.0 and FOLIO with sequence length of 128, 384 and 512, respectively. In particular, by finetuning with merely 1,004 training examples on FOLIO, it is able to generalize to much longer sequences (512) that have never been seen during pretraining.

5 Related Works

Transformer language models There have been a long line of research on neural language models since Bengio et al. (2000). Recently, it has achieved great success by exploring different variants of pretrained transformer models (Vaswani et al., 2017) for solving downstream language tasks, such as with finetuning (Radford et al., 2018; Devlin et al., 2018; Lan et al., 2019; Liu et al., 2019; Yang et al., 2019) or with zero/few-shot learning using large language models (Radford et al., 2019; Brown et al., 2020; Chowdhery et al., 2022). Another line of active research focuses on developing more effective pretraining losses (Yang et al., 2019; Clark et al., 2020; Bao et al., 2020; Tay et al., 2022) beyond the widely used autoregressive or masked language modeling objectives. There have been limited works on developing new neural architectures for learning better language representations. In this paper, we seek to move in this direction and develop a new neural architecture based on logical inductive bias.

Logic programming and neural reasoning Our logical inductive bias is inspired by logic programmings (Horn, 1951; De Raedt et al., 2007; De Raedt & Kimmig, 2015; Fierens et al., 2012; Manhaeve et al., 2018) and inductive logic programming (Evans & Grefenstette, 2018; Muggleton, 1991; 1996; Friedman et al., 1999). Different from these works, we do not directly work on reasoning problems. Instead, we seek to encode the logical inductive bias into the neural model to learn better language representations. Another line of related works is neural machine reasoning, which focuses on developing neural models that can perform reasoning in a broad sense Le et al. (2020); Santoro et al. (2018); Graves et al. (2014; 2016); Santoro et al. (2016); Le et al. (2019). Notably, different variants of memory augmented neural networks are developed, which augment control network (e.g., recurrent neural networks) with memory units and the associated neural read/write operations. They are effective in solving (relatively small-scale) reasoning tasks. However, it remains unclear whether they can be effectively pretrained to solve diverse downstream natural language tasks.

6 Conclusion

In this paper, we introduce a novel logical inductive bias which treats the language representation learning problem as logic programming. We then develop a fully-differentiable neural architecture (FOLNet) that forward-chains a set of neural logic operators, which effectively encodes this inductive bias. The proposed FOLNet architecture has the same input-output interface as the transformer models and can be pretrained over large-scale text data. Experimental results demonstrate that the FOLNet architecture significantly outperforms different variants of transformer models under the same pretraining losses. We further show that the inherent dual-branch architecture has many advantages over the single-branch version in transformers.

REFERENCES

Peter B Andrews. *An introduction to mathematical logic and type theory: to truth through proof*, volume 27. Springer Science & Business Media, 2013.

Hangbo Bao, Li Dong, Furu Wei, Wenhui Wang, Nan Yang, Xiaodong Liu, Yu Wang, Jianfeng Gao, Songhao Piao, Ming Zhou, et al. Unilmv2: Pseudo-masked language models for unified language model pre-training. In *International Conference on Machine Learning*, pp. 642–652. PMLR, 2020.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. In T. Leen, T. Dietterich, and V. Tresp (eds.), *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000. URL https://proceedings.neurips.cc/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. The fifth pascal recognizing textual entailment challenge. In *TAC*, 2009.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp. 1–14, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/S17-2001. URL https://aclanthology.org/S17-2001.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop*, pp. 177–190. Springer, 2005.

Luc De Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):5–47, 2015.

Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7, pp. 2462–2467. Hyderabad, 2007.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pp. 4171–4186, 2018.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005. URL https://aclanthology.org/I05-5002.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

Daan Fierens, Guy Van den Broeck, Maurice Bruynooghe, and Luc De Raedt. Constraints for probabilistic logic programming. In *Proceedings of the NIPS probabilistic programming workshop*, pp. 1–4, 2012.

Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *Proc. IJCAI*, 1999.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 3816–3830, 2021.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pp. 1–9, Prague, June 2007. Association for Computational Linguistics. URL https://aclanthology.org/W07-1401.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626): 471–476, 2016.

R Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. The second pascal recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, volume 7, 2006.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Luke Benson, Lucy Sun, Ekaterina Zubova, Yujie Qiao, Matthew Burtell, et al. Folio: Natural language reasoning with first-order logic. *arXiv preprint arXiv:2209.00840*, 2022.

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

Alfred Horn. On sentences which are true of direct unions of algebras1. *The Journal of Symbolic Logic*, 16(1):14–21, 1951.

Shankar Iyer, Nikhil Dandekar, and Kornl Csernai. First quora dataset release: Question pairs. January 2017. URL https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Hung Le, Truyen Tran, and Svetha Venkatesh. Neural stored-program memory. In *International Conference on Learning Representations*, 2019.

Hung Le, Truyen Tran, and Svetha Venkatesh. Self-attentive associative memory. In *International Conference on Machine Learning*, pp. 5682–5691. PMLR, 2020.

Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*, 2012.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models, 2022. URL https://arxiv.org/abs/2206.14858.

Percy Liang. Lambda dependency-based compositional semantics. *CoRR*, abs/1309.4408, 2013. URL http://arxiv.org/abs/1309.4408.

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

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in Neural Information Processing Systems*, 31, 2018.

Yu Meng, Chenyan Xiong, Payal Bajaj, Paul Bennett, Jiawei Han, Xia Song, et al. Coco-lm: Correcting and contrasting text sequences for language model pretraining. *Advances in Neural Information Processing Systems*, 34:23102–23114, 2021.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1gs9JgRZ.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

Stephen Muggleton. Inductive logic programming. *New Gener. Comput.*, 8(4):295–318, 1991.

Stephen Muggleton. Stochastic logic programs. advances in inductive logic programming. *Advances in Inductive Logic Programming*, 32:254–264, 1996.

Nvidia. A pytorch extension: Tools for easy mixed precision and distributed training in pytorch. *Available Online: https://https://github.com/NVIDIA/apex*, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL https://aclanthology.org/N18-1202.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016a. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, 2016b.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pp. 1842–1850. PMLR, 2016.

Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. *Advances in Neural Information Processing Systems*, 31:7299–7310, 2018.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *54th Annual Meeting of the Association for Computational Linguistics*, pp. 1715–1725. Association for Computational Linguistics (ACL), 2016.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *NAACL-HLT (2)*, pp. 464–468, 2018. URL https://aclanthology.info/papers/N18-2074/n18-2074.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Xavier Garcia, Dara Bahri, Tal Schuster, Huaixiu Steven Zheng, Neil Houlsby, and Donald Metzler. Unifying language learning paradigms. *arXiv preprint arXiv:2205.05131*, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1112–1122, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1101. URL https://aclanthology.org/N18-1101.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Syx4wnEtvH.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

Supplemental Materials

A More discussion about FOLNet architecture

A.1 Naming protocols for our neural logic operators

- bool: It operates over the predicate dimension and plays role of the Boolean function over a set of input truth values. Therefore we name it as bool-operator.
- join: We name this operator as join-operator because it is in a similar form as the join operation in λ-DCS (Liang, 2013).
- cjoin: Since it simply swap the roles of the kernel and the premise in the join-operator, we name it as the conjugate join operator.
- mu: We name it as mu-operator because it is can be viewed as a more general-form of the mu-abstraction operation in λ-DCS (Liang, 2013).
- assoc: We name it as assoc-operator because it can be viewed as computing the association between two vectors.
- prod: We name it as prod-operator because it is in resemblance of computing the product over the predicate dimension.
- trans: We name it as trans-operator because its form is in reminiscent of reasoning with transitive properties.

A.2 Comparing to the transformer architecture

Figure (2) show that the transformer architecture can be understood as a single-branch version of the FOLNet architecture with only `join`-operator and `assoc`-operator.

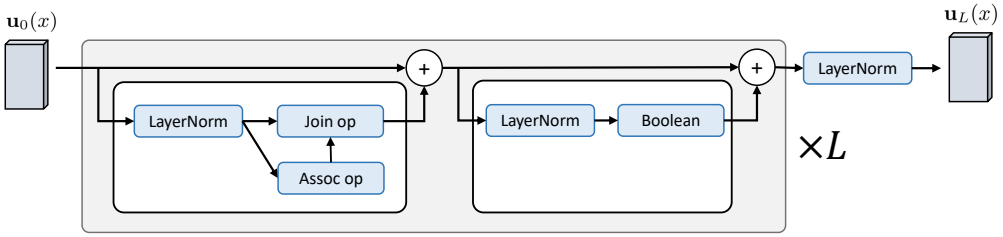

Figure 2: Overview of the Transformer architecture. It can be understood as a single-branch version of the FOLNet architecture with only `join`-operator and `assoc`-operator.

A.3 The Boolean operator

In FOLNet, we do not chain the Boolean operators in parallel with other operators but chain it with others in a cascade manner. This is akin to cascading the FFN with the self-attention module in transformers. Nevertheless, we may also place it in parallel to other operators just as making FFN in parallel to self-attention in transformer, which is done in Chowdhery et al. (2022).

B Experimental Details

B.1 Overview of the downstream tasks

GLUE The GLUE benchmark (Wang et al., 2019) consists of 9 tasks: MNLI (Williams et al., 2018), QQP (Iyer et al., 2017), QNLI (Rajpurkar et al., 2016a), SST-2 (Socher et al., 2013), CoLA (Warstadt et al., 2019), STS-B (Cer et al., 2017), MRPC (Dolan & Brockett, 2005), RTE (Dagan et al., 2005; Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009), WNLI (Levesque et al., 2012). They cover a wide range of natural language understanding tasks such as natural language

inference, paraphrasing, linguistic acceptability, and sentiment analysis. We finetune our FOLNet models by following the same procedures from BERT (Devlin et al., 2018). We do not evaluate our model on WNLI because it generally needs special procedures. The basic description of all the tasks in GLUE (including their evaluation metrics) can be found in Table 5.

SQuAD 2.0 SQuAD 2.0 (Rajpurkar et al., 2016b) is an extractive question answering dataset built from Wikipedia. The objective of the task is to predict an answer span from the context paragraphs. SQuAD 2.0 is an updated version that adds additional 50,000 unanswerable questions to the original SQuAD 1.1 version. The performance metrics are exact match (EM) and F1 scores.

FOLIO FOLIO (Han et al., 2022) is a natural language reasoning dataset that contains first-order logic reasoning problems. It requires the models to decide whether a conclusion statement is correct or not given a world defined by a set of premises. It is formulated as a 3-class classification problem with the labels being "True", "False", "Unknown". We follow the same procedure as Han et al. (2022) by formulating the problem as a sequence pair classification problem. Specifically, we concatenate all the the premises into one sequence (i.e., sequence A), and then further concatenate it with the conclusion (i.e., sequence B), where the two sequences are separated by a [SEP]. In addition, we add a [CLS] token at the beginning and a [SEP] in the end before padding (in the end). By doing so, the task has the same format as a natural language inference task (e.g., MNLI). Table 6 (quoted from the original FOLIO paper (Han et al., 2022)) shows an example from the FOLIO dataset, which demonstrate that it requires strong first-order reasoning capabilities to solve the problem. The dataset has an official train/validation/test split with 1,004/204/227 examples, respectively. By the time of this submission, the test set is not available and thus we only report performance on the validation set.

	MNLI	**QQP**	**QNLI**	**SST-2**	**CoLA**	**RTE**	**MRPC**	**STS-B**
Size	393K	364K	108K	67K	8.5K	2.5K	3.7K	5.7K
Task	Inference	Similarity	QA/Inference	Sentiment	Acceptability	Inference	Paraphrase	Similarity
Metric(s)	Accuracy	Accuracy/F1	Accuracy	Accuracy	Matthews corr.	Accuracy	Accuracy/F1	Pearson/Spearman.
#Classes	3	2	2	2	2	2	2	1 (regression)

Table 5: Basic information about different tasks in GLUE benchmark.

A FOLIO example based on the Wild Turkey Wikipedia page	
NL premises	**NL Conclusion -> Labels**
1. There are six types of wild turkeys: Eastern wild turkey, Osceola wild turkey, Gould's wild turkey, Merriam's wild turkey, Rio Grande wild turkey, and the Ocellated wild turkey.	A. Tom is an Ocellated wild turkey. -> True
2. Tom is not an Eastern wild turkey.	B. Tom is an Eastern wild turkey. -> False
3. Tom is not an Osceola wild turkey.	C. Joey is a wild turkey. -> Unknown
4. Tom is also not a Gould's wild turkey, or a Merriam's wild turkey, or a Rio Grande wild turkey.	
5. Tom is a wild turkey.	

Table 6: We show an example from the FOLIO dataset (quoted directly from the original FOLIO paper (Han et al., 2022)), which demonstrate that it requires strong first-order reasoning capabilities to solve the problem. "NL" stands for "natural language".

B.2 Implementation details and hyper-parameters

We implement both our pretraining and finetuning pipelines using PyTorch (Paszke et al., 2019) and automatic mixed precision (AMP) learning (Micikevicius et al., 2018) based on the Apex library (Nvidia, 2019). For pretraining, we use large-batch training (with a batch-size 131K) using LAMB optimizer (You et al., 2020). The pretraining of $\text{FOLNet}_{\text{Base}}$ on Wikipedia + BookCorpus (16GB) for 8K steps takes about 12 hours using 512 V100 GPUs. The pretraining of $\text{FOLNet}_{\text{Base}}$ on 160GB data for 128K steps takes 7 days using 512 V100 GPUs. And the pretraining of $\text{FOLNet}_{\text{Large}}$ on 160GB data for 128K steps takes 19 days using 512 V100 GPUs. For the finetuning of all downstream tasks, we also use AMP learning based on Apex, and the optimizers are FusedAdam from Apex library.

We report the hyper-parameters of pretraining FOLNet in Table 7. The hypper-parameters for finetuning different downstream tasks are included in Table 8. The hyper-parameters for the finetuning tasks are searched per task, and the results are the median of five random seeds.

Hyperparams	**FOLNet$_{\text{Base}}$**	**FOLNet$_{\text{Base}}$**	**FOLNet$_{\text{Large}}$**
Pretraining data size	16G	160G	160G
Number of Layers	12	12	24
Unary Hidden size (D_1)	768	768	1024
Binary Hidden size (D_2)	64	64	64
Unary Boolean (FFN) intermediate size	3072	3072	4096
Binary Boolean (FFN) intermediate size	256	256	256
Number of Attention heads	12	12	16
Attention head size	64	64	64
Dropout rate	0.1	0.1	0.1
Attention Dropout rate	0.1	0.1	0.1
Warmup Ratio	1%	1%	1%
Peak Learning Rate	1e-2	2e-3	1.6e-3
Batch Size	131,072	131,072	131,072
Weight Decay	0.01	0.01	0.01
Max Steps	8K	128K	128K
Learning rate Decay	Linear	Linear	Linear
LAMB ϵ	1e-6	1e-6	1e-6
LAMB β_1	0.9	0.9	0.9
LAMB β_2	0.999	0.999	0.999
Gradient Clipping	0.0	0.0	0.0
Sequence length	128	128	128

Table 7: The hyper-parameters for pretraining FOLNet in different settings.

Hyperparams	**GLUE-small/big**	**SQuAD 2.0**	**FOLIO**
Max epochs	{5, 10, 20} / {2, 3, 5}	{2, 3}	{60, 80}
Peak lr for Base (16G):	{6e-5, 8e-5 1e-4, 2e-4}	{ 8e-5, 9e-5, 1e-4 }	{ 6e-5, 7e-5, 8e-5}
Peak lr for Base/Large (160G):	{1e-5, 2e-5, 3e-5, 4e-5}	{1e-5, 2e-5, 3e-5, 4e-5}	{ 1e-5, 2e-5, 3e-5}
Batch size	{16, 32} / {32}	{16, 32}	{16, 32}
Learning rate decay	Linear	Linear	Linear
Warmup ratio	{6%, 25%} / {6%}	6%	{6%, 25%}
Sequence length	128	384	512
Adam ϵ	1e-6	1e-6	1e-6
Adam β_1	0.9	0.9	0.9
Adam β_2	0.999	0.999	0.999
Gradient clipping	0.0	0.0	0.0
Dropout rate	0.1	0.1	0.1
Weight decay	0.01	0.01	0.01

Table 8: The hyperparameters for finetuning on GLUE, SQuAD 2.0, and FOLIO tasks. GLUE-small refers to CoLA, STS-B, MRPC and RTE. GLUE-big stands for MNLI, QQP, QNLI and SST-2.

C ADDITIONAL EXPERIMENTS

Zero-shot performance on GLUE We evaluate the zero-shot performance of our FOLNet model on GLUE benchmark. Specifically, we perform zeros-shot predictions by using the same method and prompt templates from Gao et al. (2021). We only consider the FOLNet$_{\text{Large}}$ model and compare it to RoBERTa$_{\text{Large}}$, which are similar in model-size, pretraining dataset and pretraining losses. In addition, the zero-shot prediction methods are also similar: they both predict the label words using their own MLM heads. The results are summarized in Table 9, which demonstrates that FOLNet$_{\text{Large}}$ outperforms RoBERTa$_{\text{Large}}$ on 7 out of 8 tasks with average gain of 7.2 points.

Model	Method	Data	MNLI-m/mm Acc	QQP F1	QNLI Acc	SST-2 Acc	CoLA mcc	STS-B Pear.	MRPC F1	RTE Acc	Avg
Majority guess	-	-	32.7/33.0	0.0	49.5	50.9	0.0	-	81.2	52.7	33.3
$\text{RoBERTa}_{\text{Large}}$	MLM	160G	50.8/51.7	49.7	50.8	**83.6**	2.0	-3.2	61.9	51.3	44.3
$\text{FOLNet}_{\text{Large}}$	MLM	160G	**50.8/52.6**	**55.4**	**59.7**	79.5	**6.4**	**23.6**	**77.2**	**58.2**	**51.5**

Table 9: The zero-shot performance on the GLUE benchmark. "Acc" stands for accuracy, "mcc" means Matthews's correlation coefficient, and "Pear." is short for Pearson's correlation coefficient.