



# FastNER: Speeding up Inferences for Named Entity Recognition Tasks

Yuming Zhang<sup>1</sup>, Xiangxiang Gao<sup>2</sup>, Wei Zhu<sup>3</sup>, and Xiaoling Wang<sup>3</sup>(✉)

<sup>1</sup> College of Computer Science and Software Engineering, Shenzhen University,  
Shenzhen, China

<sup>2</sup> Shanghai Jiaotong University, Shanghai, China

<sup>3</sup> East China Normal University, Shanghai, China

xlwang@cs.ecnu.edu.cn

**Abstract.** BERT and its variants are the most performing models for named entity recognition (NER), a fundamental information extraction task. We must apply inference speedup methods for BERT-based NER models to be deployed in the industrial setting. Early exiting allows the model to use only the shallow layers to process easy samples, thus reducing the average latency. In this work, we introduce FastNER, a novel framework for early exiting with a BERT biaffine NER model, which supports both flat NER tasks and nested NER tasks. First, we introduce a convolutional bypass module to provide suitable features for the current layer's biaffine prediction head. This way, an intermediate layer can focus more on delivering high-quality semantic representations for the next layer. Second, we introduce a series of early exiting mechanisms for BERT biaffine model, which is the first in the literature. We conduct extensive experiments on 6 benchmark NER datasets, 3 of which are nested NER tasks. The experiments show that: (a) Our proposed convolutional bypass method can significantly improve the overall performances of the multi-exit BERT biaffine NER model. (b) our proposed early exiting mechanisms can effectively speed up the inference of BERT biaffine model. Comprehensive ablation studies are conducted and demonstrate the validity of our design for our FastNER framework.

**Keywords:** Early Exiting · Pre-trained language models · Inference speed-up

## 1 Introduction

Since the rise of BERT [3], the pre-trained language models (PLMs) are the state-of-the-art (SOTA) models for natural language processing (NLP) [13, 31]. Many PLMs are developed by the academia and industry, such as GPT [15], XLNet [25], and ALBERT [10], and so forth. These BERT-style models achieved

---

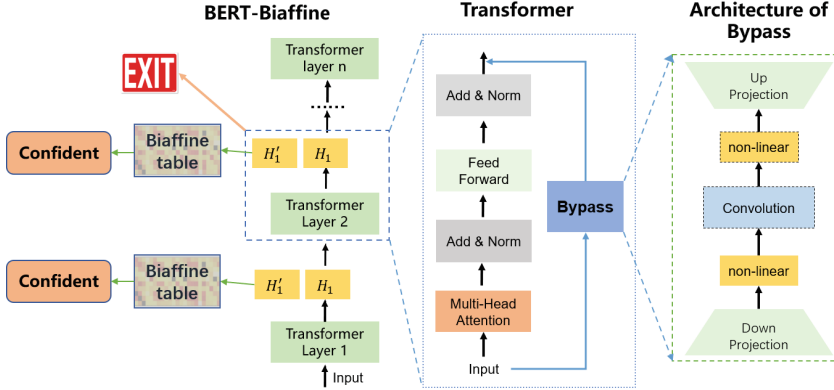
Y. Zhang, X. Gao and W. Zhu—Equal contributions.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

X. Yang et al. (Eds.): ADMA 2023, LNAI 14176, pp. 185–199, 2023.

[https://doi.org/10.1007/978-3-031-46661-8\\_13](https://doi.org/10.1007/978-3-031-46661-8_13)

considerable improvements in many NLP tasks by self-supervised pre-training and transfer learning on labeled tasks, such as classification, text pair matching, named entity recognition (NER), etc. Despite their outstanding performances, their industrial usage is still limited by the high latency during inference.



**Fig. 1.** The overall architecture of our FastNER framework.

NER and other sequence labeling tasks play a central role in many application scenarios, such as question answering, document search, document-level information extraction, etc. However, these applications require low latency. For example, an online search engine needs to respond to the user’s query in less than 100 milo-seconds. Thus, a NER model should be accurate and efficient. In addition, at certain time intervals, consumer query traffic is very concentrated. For example, during dinner hours, food search engines will be used much often than usual. Thus, it is important for deployed NER models to flexibly adjust their latency.

Literature has focused on making PLMs’ inference more efficient via adaptive inference [14, 23, 28, 30]. The idea of adaptive inference is to process simple queries with lower layers of BERT and more difficult queries with deeper layers, thus speeding up inference on average without loss of accuracy. The speedup ratio can be flexibly controlled with certain hyper-parameters without re-deploying the model services. Early exiting is the representative adaptive inference methods [1]. As depicted in Fig. 1, it implements adaptive inference by installing an early exit, i.e., an intermediate prediction head, at each layer of PLMs (multi-exit PLMs) and early exiting “easy” samples to speed up inference. All the exits are jointly optimized at the training stage with BERT’s parameters. At the inference stage, certain early exiting strategies are designed to decide which layer to exit [4, 8, 18, 23, 27, 28]. In this mode, different samples can exit at different depths.

For our framework to be generally applicable, we mainly adopt the biaffine model [26] for NER. The biaffine model converts the NER task into a 2-dimensional table filling task, thus providing a solution to both the flat and nested NER problem. [26] shows that the biaffine model can achieve state-of-the-art (SOTA) performances on both nested NER tasks and flat NER tasks.

In this work, we propose a framework for the early exiting of BERT biaffine NER models, inspired by BADGE [29]. First, we add a convolutional bypass to the current transformer layer to provide different representations for the current layer’s biaffine exit and the next transformer layer of the BERT backbone. In this way, the BERT backbone will not be distracted from different tasks, thus improving the cross-layer average performance of the multi-exit biaffine model. Second, we extend the commonly used early exiting mechanisms in sentence classification tasks, entropy-based early exiting and max-probability based early exiting, to the biaffine NER model, we can perform adaptive inferences for NER tasks. Intuitively, the decision of early exit is made when the intermediate biaffine exit is confident in its predictions.

Extensive experiments are conducted on the six benchmark NER tasks. Three of the tasks are nested NER tasks, ACE2004<sup>1</sup>, ACE2005<sup>2</sup>, GENIA [9]. We also experiment on three flat NER tasks, CONLL2003 [19], OntoNotes 4.0 Chinese<sup>3</sup> and the Chinese MSRA task [11]. We show that: (1) our FastNER training method consistently performs better than the baseline multi-exit model training methods. (2) We show that we can achieve 2–3x speedup with limited performance losses. In addition, we show that with the better multi-exit model trained with our FastNER, better efficiency-performance tradeoffs can be made. Ablation studies validate the architectural design of our FastNER methods.

The rest of the paper is organized as follows. First, we introduce the preliminaries for the Biaffine NER model and early exiting. Second, we elaborate on our FastNER method. Third, we conduct experiments on 6 NER tasks and conduct a series of ablations studies. Finally, we conclude with possible future works.

## 2 Preliminaries

This section introduces the background for PLMs and early exiting. Throughout this work, we consider a NER task with samples  $\{(x, y), x \in \mathcal{X}, y \in \mathcal{Y}, i = 1, 2, \dots, N\}$ , e.g., sentences and their NER span information, and the number of entity categories is  $K$  (including the non-entity type label). The input sequence length after BERT’s subword tokenization is  $L$ .

### 2.1 PLM Models

We use BERT as the backbone model. BERT is a Transformer [20] model pre-trained in a self-supervised manner on a large corpus. In the ablation studies, we also use ALBERT [10] as backbones. ALBERT is more lightweight than BERT since it shares parameters across different layers, and the embedding matrix is factorized. The number of layers of our PLM backbone is denoted as  $M$ , and the hidden dimension is  $d$ .

<sup>1</sup> <https://catalog ldc.upenn.edu/LDC2005T09>.

<sup>2</sup> <https://catalog ldc.upenn.edu/LDC2006T06>.

<sup>3</sup> <https://catalog ldc.upenn.edu/LDC2011T03>.

## 2.2 The Biaffine Model for NER

The BERT-Biaffine model [26] transforms the NER task into a two-dimensional table filling task. It asks the model to identify whether the slot in the table with coordinate  $(s, e)$  corresponds to an entity with category  $k$ , that is, whether a pair of tokens  $(x_s, x_e)$  in the input sequence  $x = (x_1, x_2, \dots, x_L)$  is the start and end tokens for an entity with category  $k$ . Formally, after BERT encoding, the contextualized embedding of tokens  $s$  and  $e$  are  $h_s$  and  $h_e$  ( $h_s, h_e \in \mathcal{R}^d$ ). Then  $h_s$  and  $h_e$  will go through two multi-layer perceptrons with Tanh activation function (denoted as MLP-start and MLP-end),

$$h_s = \text{Tanh}(h_s W_1^{start}) W_2^{start}, \quad (1)$$

$$h_e = \text{Tanh}(h_e W_1^{end}) W_2^{end}. \quad (2)$$

MLP-start and MLP-end transform the BERT's representations to adapt to the table-filling NER task. Then in a biaffine layer  $f$ , the score of span  $(s, e)$  is calculated by

$$f(s, e) = h_s^T U h_e + W(h_s \oplus h_e) + b. \quad (3)$$

Since we need to calculate the scores for  $K$  entity categories,  $U$  is a  $d \times K \times d$  tensor, and  $W$  is a  $2d \times K$  tensor.  $f(s, e) \in \mathcal{R}^K$  is the scores (or logits). A softmax operation will transform  $f(s, e)$  into a probability distribution  $p(s, e)$ , which represents how likely the span  $(s, e)$  is a category  $k$  entity.

The learning objective of the biaffine model is to assign a correct category (including the non-entity) to each valid span. Hence it is a multi-class classification problem at each slot of the two-dimensional table and can be optimized with cross-entropy loss:

$$\mathcal{L} = - \sum_{s=1}^L \sum_{e=s}^L \sum_{k=1}^K \mathcal{I}(y(s, e) = k) \log p_k(s, e), \quad (4)$$

where  $y(s, e)$  is the ground-truth label of span  $(s, e)$ ,  $p_k(s, e)$  is the predicted probability mass of  $(s, e)$  having label  $k$ , and  $\mathcal{I}(\cdot)$  is the indicator function. After fine-tuning the BERT biaffine model, the inference procedure of the BERT biaffine model follows [26], which involves determining the final named entity spans. Since there may be conflicting spans, [26] rank the spans via their probability masses, and the span with a higher probability mass will be kept when it conflicts with other predicted spans.

## 2.3 Early Exiting

As depicted in Fig. 1, early exiting architectures, or multi-exit architectures, are networks with exits<sup>4</sup> at each transformer layer. Since the previous literature usually considers sentence-level classification tasks, the exits are sentence classifiers. However, since we are dealing with sequence labeling tasks formulated as two-dimensional table filling, with  $M$  exits,  $M$  separate biaffine modules  $f^{(m)}$  are

<sup>4</sup> Some literature (e.g., DeeBERT [23]) also refers to exits as off-ramps.

installed right after each layer of BERT ( $m = 1, 2, \dots, M$ ), and the scores for span  $(s, e)$  at layer  $m$  is given by:

$$f^{(m)}(s, e) = h_s^T U^{(m)} h_e + W^{(m)}(h_s \oplus h_e) + b^{(m)}. \quad (5)$$

And the loss function at each layer becomes

$$\mathcal{L}^{(m)} = - \sum_{s=1}^L \sum_{e=s}^L \sum_{k=1}^K \mathcal{I}(y(s, e) = k) \log p_k^{(m)}(s, e), \quad (6)$$

where  $p^{(m)}(s, e) = \text{Softmax}(f^{(m)}(s, e))$  is the predicted probability distribution at exit  $m$ .

**Training.** The most commonly used training method for early exiting architectures is joint training (JT). All the exits are jointly optimized at the training stage with a summed loss function. Following [6] and [28], the overall loss function is:

$$\mathcal{L}^{WA} = \frac{\sum_{m=1}^M m * \mathcal{L}^{(m)}}{\sum_{m=1}^M m}. \quad (7)$$

Note that the weight  $m$  corresponds to the relative inference cost of exit  $m$ .

Two other commonly used training methods are two-stage training [14, 23] (2ST) and alternating training [24] (ALT). 2ST first fine-tunes the PLM backbone and the last exit till convergence in the first stage and then fine-tunes the intermediate exits in the second stage. ALT trains the backbone and the last exit at the even optimization steps, and the intermediate exits at the odd optimization steps.

**Inference.** At inference, the multi-exit PLMs can operate in two different modes, depending on whether or not the computational budget to classify an example is known.

**Static Early Exiting.** We can directly appoint a fixed exit  $m^*$  of PLM,  $f^{(m^*)}$ , to predict all the queries.

**Dynamic Early Exiting.** Under this mode, upon receiving a query input  $x$ , the model starts to predict on the exits  $f^{(1)}$ ,  $f^{(2)}$ , ..., in turn. It will continue until it receives a signal to stop early at an exit  $m^* < M$ , or arrives at the last exit  $M$ . At this point, it will output a predictions by combining the current and previous predictions in a certain way. Different samples might exit at different layers under this early exit setting.

**Speedup Ratio.** Following PABEE [28], we mainly report the speedup ratio as the efficiency metric. For each test sample  $x_i$ , the inference time cost is  $t_i$  under early exiting, and is  $T_i$  without early exiting, then the average speedup ratio on the test set is calculated by

$$\text{Speedup} = 1 - \frac{\sum_{i=1}^N t_i}{\sum_{i=1}^N T_i}. \quad (8)$$

### 3 FastNER

#### 3.1 A Lite Biaffine Module

Note that the original BERT biaffine NER model does not consider the early exiting scenarios. Each biaffine module (Eq. 5) introduces 5–7 million parameters. If we add this biaffine module at each layer, the added parameters will amount to 60 million or above. Introducing too many randomly initialized parameters would result in low efficiency, difficulty in optimization, and overfitting for shallow layers. Thus, we propose a modified version of biaffine module called the lite biaffine module.

In the lite biaffine module, MLP-start and MLP-end are substituted by a simple linear projection layer that project  $h_s$  and  $h_e$  from dimension  $d$  to  $d_1 = d/4$ ,<sup>5</sup> and the down-projected  $h-s$  and  $h_e$  are fed into Eq. 5 for logit calculations. This way, the parameters in a biaffine layer will be reduced to less than 0.5 million. In the experiments, we will show that our lite biaffine module performs better than the original one in the early exiting scenarios.

#### 3.2 Motivation

Similar to the analysis in [29], training a multi-exit BERT biaffine model requires training multiple prediction heads of different depths simultaneously. Thus, under this setting, an intermediate layer has to fulfill two tasks at once: (a) providing semantic representations to the next layer and (b) providing proper token features to the biaffine module of the current layer. One may wonder whether these two tasks conflict with each other and result in poor optimizations. [12] investigate this problem in the sentence classification tasks and find that each layer’s optimizations are often in conflict and can cause gradient instability. They provide a solution called gradient equilibrium, which is to adjust the gradients from each exit. However, in our experiments, we will show that this method does not provide significant improvements.

Another solution is to use different sub-networks for these two tasks, following the literature on sparse multi-task learning [17]. However, this method provides two different representations with two different, forward passes with different sub-networks, significantly slowing down the inference speed. Thus, this approach does not meet our purpose.

To summarize, we need a new method that can provide two different representations, one for the next layer and the other for the current layer’s prediction, within a single forward pass.

#### 3.3 Bypass Architecture

We now present the core of our FastNER framework: the convolutional bypasses (depicted in Fig. 1). The notation follows [29]. Denote the hidden states of the

<sup>5</sup> The reason why we set  $d_1 = d/4$  is that smaller  $d_1$  would result in significant performance drops, according to our initial experiments.

input from the last layer as  $H_{m-1}$ .  $H_{m-1}$  will go through the transformer layer’s self-attention (MHSA) and positional feed-forward (FFN) modules, to become  $H_{m,0}$ , and then a LayerNorm operation to output  $H_m$ .

We want the efficient bypass  $B_m$  to adjust  $H_m$  to fit the task better.  $B_m$  is simple in architecture (On the right side of Fig. 1). After receiving the input  $H_{m-1}$ ,  $B_m$  reduce its dimension from  $d$  to  $r$ , and obtain  $H_{m,B}^{(1)}$  (where  $r \ll d$ , and  $r$  is called the bottleneck dimension) via a down-projection  $W_{down} \in \mathbb{R}^{d \times r}$ .  $H_{m,B}^{(1)}$  will go through a non-linear activation function  $g_1$ , a convolutional layer with kernel size 3 (denoted as *conv*) and another activation function  $g_2$ , and become  $H_{m,B}^{(2)}$ .  $H_{m,B}^{(2)}$  will then be up-projected to  $H_{m,B}^{(3)}$  to recover the dimension, by an up-projection matrix  $W_{up} \in \mathbb{R}^{r \times d}$ . The literature usually refer to  $r$  as the bottleneck dimension. Formally,  $B_m$  can be expressed as:

$$H_{m,B}^{(3)} \leftarrow g_2(\text{conv}(g_1(H_{m-1}W_{down})))W_{up}. \quad (9)$$

Finally, the current layer will output two representations,  $H_m$ , which is the original hidden states, and  $H'_m$ , which is modified by the bypass by:

$$\begin{aligned} H'_m &= \text{LayerNorm}(H_{m,B}^{(3)} + H_{m,0}), \\ H_m &= \text{LayerNorm}(H_{m,0}). \end{aligned} \quad (10)$$

$H_m$  is passed to the next transformer layers as input, and  $H_m^{by}$  will be the hidden states received by the intermediate biaffine exit. The bottleneck dimension  $r$  is very small, like 16, so that the extra parameters or flops introduced by the bypasses are less than 1% of the compared to the BERT backbone.

### 3.4 Early Exiting for Biaffine NER Model

Although the literature comprehensively studied the early exiting for sentence classification tasks, the early exiting mechanism of entity-level tasks like NER has been neglected. Based on the literature on early exiting on sentence classification tasks, this work proposes two plausible early exiting mechanisms for the biaffine NER model.

**Entropy-Based Early Exiting (Entropy).** This method is a directly extension of [18] and [23] from the sentence classification tasks to NER. We denote the table of distributions predicted by the biaffine exit  $m$  as  $\mathcal{T}^{(m)} = \{p^{(m)}(s, e) | s, e \in 1, \dots, L\}$ , which is a  $L \times L \times K$  tensor. On each slot  $p^{(m)}(s, e)$  of the biaffine table, we can calculate its entropy  $\text{Ent}^{(m)}(s, e)$  via

$$\text{Ent}^{(m)}(s, e) = \frac{-1}{\log(K)} \sum_{k=1}^K p_k^{(m)}(s, e) \log(p_k^{(m)}(s, e)). \quad (11)$$

Intuitively, if the biaffine exit is confident with its prediction, the average entropy  $\text{AvgEnt}^{(m)}$ , calculated by

$$\text{AvgEnt}^{(m)} = \frac{\sum_{s=1}^L \sum_{e=1}^L \text{Ent}^{(m)}(s, e)}{L * L}, \quad (12)$$

will be smaller. Thus  $\text{AvgEnt}^{(m)}$  can be treated as the early exiting criterion. A threshold  $\tau_e$  is predefined. If at layer  $m$ ,  $\text{AvgEnt}^{(m)}$  is smaller than  $\tau_e$ , the model will exit. Otherwise, the model will continue its forward pass.

**Maximum-Probability-Based Early Exiting (Maxprob).** This method is a direct extension of [16] from the sentence classification tasks to NER. Intuitively, if the biaffine exit is confident with its prediction, the table of predicted distributions  $\mathcal{T}^{(m)}$  will concentrate their probability masses on single specific labels. Denote the maximum probability mass at slot  $(s, e)$  as  $MP^{(m)}(s, e)$ , then the average maximum probability is given by

$$\text{AvgMP}^{(m)} = \frac{\sum_{s=1}^L \sum_{e=1}^L MP^{(m)}(s, e)}{L * L}. \quad (13)$$

A threshold  $\tau_{mp}$  is predefined. If at layer  $m$ ,  $\text{AvgMP}^{(m)}$  is larger than  $\tau_{mp}$ , the model will exit. Otherwise, the model will continue its forward pass.

## 4 Experiments

### 4.1 Datasets

We evaluate our FastNER on both nested and flat NER tasks. For the nested NER task, we use the ACE2004 task<sup>6</sup>, ACE2005 task<sup>7</sup>, and GENIA task [9]. For the flat NER task, we evaluate our method on the CONLL 2003 task [19] (CONLL03), the OntoNotes 4.0 corpus<sup>8</sup> (Onto-4), and the Chinese MSRA NER (MSRA) task [11].

### 4.2 Baselines

For multi-exiting BERT biaffine fine-tuning, we compare our FastNER framework with the following baselines:

**Two-Stage Training (2ST).** This method is adapted by [23] and [14]. It first fine-tunes the backbone and the last exit till convergence. Then all the intermediate layers' exits (except the last layer's exit) will be trained on top of the frozen backbone.

**Joint Training (JT).** This method trains the BERT backbone and all the biaffine exits jointly. Literature has different variations for JT. PABEE [28] and RightTool [16] adopts increasing loss weights for higher layers. We will denote their version of joint training as JT-PABEE. BranchyNet [18] adopts different and gradually increasing learning rates for different exits during training. We will denote their version as JT-BranchyNet.

<sup>6</sup> <https://catalog.ldc.upenn.edu/LDC2005T09>.

<sup>7</sup> <https://catalog.ldc.upenn.edu/LDC2006T06>.

<sup>8</sup> <https://catalog.ldc.upenn.edu/LDC2011T03>.



**Alternating Training.** BERxit proposes to combine the training of JT-PABEE and 2ST, that is, conduct back-propagation via the loss from the last exit at the odd optimization steps and conduct back-propagation via the average loss from all the intermediate exits.

**Gradient Equilibrium (GradEquil).** [12] proposes to adjust the gradient norm of each intermediate layer during optimization, so that the training process will be more stable.

**Sparse Multi-task (Sparse-MT).** [17] As analyzed in Sect. 3.2, this method is not suitable for model inference speedup methods since it requires multiple forward passes to generate different representations suitable for different tasks. We include this method as a sanity check and show that our FastNER method performs better even with a single forward pass.

**Early Exiting Mechanisms.** To show that our FastNER method can effectively improve the model’s early exiting performances, we will run dynamic early exiting with different early exiting mechanisms as described in Sect. 3.4: (a) Entropy-based method (entropy); (b) maximum probability-based method (maxprob). Early exiting will be run on different backbones to show that our FastNER framework can improve the efficiency-performance tradeoffs.

### 4.3 Experimental Settings

**Training.** English NER tasks use the open-sourced Google BERT [3]<sup>9</sup> as the backbone, and the Chinese tasks adopt the BERT-www-ext released from [2]<sup>10</sup> as the backbone model. We also use ALBERT-base and ALBERT base Chinese by [10] as the backbone models for ablation studies. We add a lite biaffine NER layer or an original biaffine layer after each intermediate layer of the pre-trained models as the intermediate classifiers. The convolutional bypasses’ activation function is set to be GELU [5]. We fine-tune models for at most 25 epochs; early stopping with patience eight is performed, and the best checkpoint is selected based on the dev set performances. We perform grid search over batch sizes of 16, 32, 128, learning rates of  $1e-5$ ,  $2e-5$ ,  $3e-5$ ,  $5e-5$  with an Adam optimizer, and the convolutional bypasses’ bottleneck dimension of 8, 16, 32. We implement FastNER and all the baselines on the base of Hugging Face’s Transformers [22]. Experiments are conducted on four Nvidia V100 16 GB GPUs.

**Inference.** Following prior work on input-adaptive inference [8, 18], inference is on a per-instance basis, i.e., the batch size for inference is set to 1. This is a common scenario in the industry where individual requests from different users [16] come at different time points. We report the median performance over five runs with different random seeds.

<sup>9</sup> <https://huggingface.co/bert-base-uncased>.

<sup>10</sup> <https://github.com/ymcui/Chinese-BERT-www>.

#### 4.4 Evaluation Metrics

Entity-level F1 is the most widely used metric for NER tasks [7]. For multi-exit PLMs, each exit has a performance score. Thus, to properly evaluate multi-exit PLMs, we propose the following three derived metrics: (a) F1-avg, which denotes the cross-layer average F1 score; (c) F1-best, which is the best F1 score among all the layers. We use F1-avg as our primary metric for experimental result reporting and checkpoint selection during training.

**Table 1.** Experimental results of models with BERT backbone on the six benchmark NER datasets.

	ACE2004	ACE2005	GENIA	CONLL2003	MSRA	OntoNotes4.0						
BERT biaffine model												
	F1	F1	F1	F1	F1	F1						
BERT + original Biaffine	83.5	83.9	77.4	90.7	95.3	81.7						
BERT + Lite Biaffine (ours)	83.6	83.8	77.6	91.0	95.2	82.2						
Multi-exit BERT biaffine model												
	F1-avg	F1-best	F1-avg	F1-best	F1-avg	F1-best	F1-avg	F1-best	F1-avg	F1-best	F1-avg	F1-best
2ST	69.6	83.6	73.0	83.8	70.2	77.9	82.8	91.1	88.1	94.3	71.3	75.9
JT-PABEE	77.5	82.5	78.7	82.8	70.3	77.8	86.6	90.3	89.5	94.5	69.3	79.8
JT-BranchyNet	77.6	82.8	78.6	82.9	70.5	77.4	86.9	90.5	89.1	94.0	70.8	80.6
GradEquil	77.8	83.2	78.4	83.8	70.7	77.9	87.1	90.7	89.5	94.3	71.2	81.1
ALT	78.1	83.3	78.4	83.3	70.5	77.8	86.8	90.6	88.7	93.7	71.9	81.8
Sparse-MT	77.9	83.0	78.1	82.9	70.8	78.2	86.3	90.2	88.4	93.6	71.6	80.5
FastNER + original biaffine	78.5	84.1	79.4	83.9	71.1	78.1	87.7	91.3	90.5	94.7	73.2	82.1
FastNER	<b>78.7</b>	84.0	<b>79.7</b>	84.1	<b>71.5</b>	78.8	<b>87.8</b>	91.5	<b>91.1</b>	95.1	<b>73.8</b>	82.3

#### 4.5 Overall Comparison

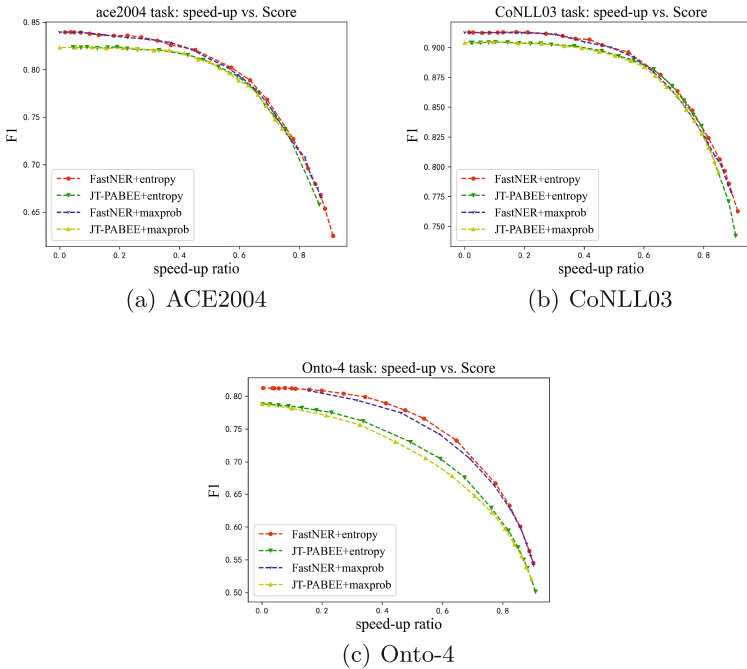
We compare our FastNER with the previous SOTA training methods of multi-exit BERT-biaffine models. Table 1 reports the performance on the six benchmark NER datasets when using BERT as the backbone model. The upper half of the table reports the performances of the last transformer layer’s biaffine exit or the 6-th layer’s exit. With fewer randomly initialized parameters, our lite biaffine layer can outperform comparably with the original biaffine layer.

The cross-layer average and best performances are reported in the lower half of Table 1. The following takeaways can be made:

- Our FastNER method consistently outperforms the existing multi-exit BERT biaffine model training methods in terms of F1-avg by a clear margin. Note that as modifications to the joint training methods, GradEquil and Sparse-MT perform comparably to JT-BranchyNet and JT-PABEE under our settings. Although ALT [24] and 2ST [23] perform well in sentence-level tasks like the GLUE benchmarks [21], it does not perform very well when training the BERT biaffine NER models.
- With the help of the bypasses, our FastNER method improves the average performances across all the intermediate layers and improves the F1-best scores compared with JT-PABEE or JT-BranchyNet by a large margin. This result is consistent with our motivation: introducing the convolutional bypasses can

help the intermediate transformer layer to concentrate on providing hidden representations. In contrast, the bypasses can provide a modified version of the current layer’s hidden states that are more suitable for the current layer’s biaffine exit. In this way, both the F1-best and F1-avg scores can improve. As a direct result, the best layer’s score under our FastNER framework is comparable to or performs better than vanilla fine-tuning.

- To show that our FastNER does not achieve such performance improvements merely by adding more parameters, we also run FastNER with the original biaffine module (the FastNER + original biaffine setting in Table 1). With much more additional parameters, FastNER + original biaffine still underperforms the FastNER setting in terms of F1-avg. We think this is because the original biaffine modules are too parameter cumbersome for the shallow layers to learn.



**Fig. 2.** The speedup-score curves with different dynamic early exiting methods, on the ACE, CoNLL03 and Onto-4 datasets. The multi-exit BERT biaffine models are trained with FastNER or JT-PABEE.

#### 4.6 Dynamic Early Exiting Performances

With the improved overall performances on each layer, intuitively, the model’s early exiting performances can be improved. We run early exiting with different

**Table 2.** Experimental results for the ablation study of whether to pass  $H'_i$  to the next layer. The F1-avg (cross-layer average F1) scores are reported. The performance differences between FastNER and FastNER-reduced are significant for both tasks.

	ACE2004		Onto-4	
	F1-avg	F1-best	F1-avg	F1-best
FastNER	78.7	84.0	73.8	82.3
FastNER-reduced	76.6	80.7	71.3	75.9

confidence thresholds on multi-exit BERT-biaffine models trained by FastNER or JT-PABEE. The early exiting mechanisms are the entropy-based method and the maxprob-based method. The resulting speedup-performance curves are plotted in Fig. 2, where the x-axis represents the speedup ratio, and the y-axis is the F1 score achieved on the test set.

From Fig. 2, we can see that with our FastNER training and early exiting methods, a BERT biaffine model can achieve 2x–3x speedups with limited performance degradations. The apparent gaps between our FastNER model and JT-PABEE shown in Fig. 2 can also be observed in Fig. 2, proving that improving the averaging performances across layers in better efficiency-performance tradeoffs during early exiting. In addition, the entropy-based and maxprob-based methods can perform comparably with each other for the early exiting of BERT biaffine models.

#### 4.7 Ablation on Whether to Pass $H'_i$ to the Next Layer

The core idea of FastNER is to provide different intermediate hidden states for different purposes via our convolutional bypasses. As a sanity check and to demonstrate that our design is necessary, we now consider the following setting: reducing our design of bypasses by passing  $H'_i$  to the next layer and using it for prediction. We will denote this setting as FastNER-reduced. We use FastNER-reduced for training on ACE2004 and Onto-4 datasets, and the results are reported in Table 2.

From Table 2, we can see that FastNER-reduced asks  $H'_i$  to complete two tasks at once, resulting in a significant drop in overall performances. Note that the performance difference between FastNER and FastNER-reduced is significant on ACE2004 and Onto-4. The results show that our design of providing different representations for different purposes via convolutional passes is the key to improving the overall performances of multi-exit BERT biaffine models.

#### 4.8 Comparisons of Different Bottleneck Dimensions $r$

The bottleneck dimension  $r$  is 16 for the main experiments. To investigate the effects of bottleneck dimensions on the model performances, We conducted ablation experiments on the ACE2004 and Onto-4 tasks. Table 3 reports the F1-avg

**Table 3.** Experimental results for comparing different bottleneck dimensions. The F1-avg (cross-layer average F1) scores are reported.

	ACE2004		Onto-4	
	F1-avg	F1-best	F1-avg	F1-best
r = 64	78.5	83.3	73.7	80.8
r = 32	78.7	83.9	<b>73.8</b>	82.3
r = 16	<b>78.8</b>	84.0	73.6	82.4
r = 8	78.7	83.8	73.3	81.9
r = 4	78.3	83.5	72.7	81.8

scores under different values of  $r$ . From Table 3, we can see that smaller bottleneck dimensions do not result in significant performance drops. An intriguing observation is that larger bottleneck dimensions do not provide performance improvements, demonstrating that the superior performances of FastNER do not come from introducing additional model parameters.

## 5 Conclusions

In this work, we first design the FastNER-bypass framework, consisting of convolutional bypasses, to enhance the overall performances of multi-exit BERT biaffine NER models. Second, the existing literature does not investigate the problem of early exiting for the NER tasks. Thus, we transfer the early exiting methods for sentence-level tasks to the biaffine NER model and propose two early exiting mechanisms: the entropy-based method and the maxprob-based method. Experiments are conducted on six benchmark NER datasets. The experimental results show that: (a) our FastNER framework can effectively improve the overall performances of multi-exit BERT biaffine models, thus providing stronger backbones for dynamic early exiting; (b) the early exiting mechanisms we designed for the BERT biaffine NER models can achieve 2–3 times inference speedups with quite limited performances drops.

**Acknowledgement.** This work was supported by NSFC grants (No. 61972155 and 62136002), National Key R&D Program of China (No. 2021YFC3340700), Shanghai Trusted Industry Internet Software Collaborative Innovation Center.

## References

1. Bolukbasi, T., Wang, J., Dekel, O., Saligrama, V.: Adaptive neural networks for efficient inference. In: ICML (2017)
2. Cui, Y., Che, W., Liu, T., Qin, B., Wang, S., Hu, G.: Revisiting pre-trained models for Chinese natural language processing. [arXiv:abs/2004.13922](https://arxiv.org/abs/2004.13922) (2020)

3. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. [arXiv:1810.04805](https://arxiv.org/abs/1810.04805) (2018)
4. Gao, X., Zhu, W., Gao, J., Yin, C.: F-PABEE: flexible-patience-based early exiting for single-label and multi-label text classification tasks. In: ICASSP 2023–2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1–5. IEEE (2023)
5. Hendrycks, D., Gimpel, K.: Bridging nonlinearities and stochastic regularizers with gaussian error linear units. [arXiv:abs/1606.08415](https://arxiv.org/abs/1606.08415) (2016)
6. Huang, G., Chen, D., Li, T., Wu, F., Maaten, L.V.D., Weinberger, K.Q.: Multi-scale dense convolutional networks for efficient prediction. [arXiv:abs/1703.09844](https://arxiv.org/abs/1703.09844) (2017)
7. Huang, Z., Xu, W., Yu, K.: Bidirectional LSTM-CRF models for sequence tagging. [arXiv:abs/1508.01991](https://arxiv.org/abs/1508.01991) (2015)
8. Kaya, Y., Hong, S., Dumitras, T.: Shallow-deep networks: understanding and mitigating network overthinking. In: ICML (2019)
9. Kim, J.D., Ohta, T., Tateisi, Y., Tsujii, J.: GENIA corpus—a semantically annotated corpus for bio-textmining. *Bioinformatics (Oxford, England)* **19**(Suppl. 1), i180-2 (2003). <https://doi.org/10.1093/bioinformatics/btg1023>
10. Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., Soricut, R.: ALBERT: a lite BERT for self-supervised learning of language representations. [arXiv:abs/1909.11942](https://arxiv.org/abs/1909.11942) (2020)
11. Levov, G.A.: The third international Chinese language processing bakeoff: word segmentation and named entity recognition. In: Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing, pp. 108–117. Association for Computational Linguistics, Sydney, Australia, July 2006. <https://aclanthology.org/W06-0115>
12. Li, H., Zhang, H., Qi, X., Yang, R., Huang, G.: Improved techniques for training adaptive deep networks. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 1891–1900 (2019)
13. Li, X., et al.: Pingan smart health and SJTU at COIN - shared task: utilizing pre-trained language models and common-sense knowledge in machine reading tasks. In: Proceedings of the First Workshop on Commonsense Inference in Natural Language Processing, pp. 93–98. Association for Computational Linguistics, Hong Kong, China, November 2019. <https://doi.org/10.18653/v1/D19-6011>, <https://aclanthology.org/D19-6011>
14. Liu, W., Zhou, P., Zhao, Z., Wang, Z., Deng, H., Ju, Q.: FastBERT: a self-distilling BERT with adaptive inference time. [arXiv:abs/2004.02178](https://arxiv.org/abs/2004.02178) (2020)
15. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al.: Language models are unsupervised multitask learners. *OpenAI Blog* **1**(8), 9 (2019)
16. Schwartz, R., Stanovsky, G., Swayamdipta, S., Dodge, J., Smith, N.A.: The right tool for the job: matching model and instance complexities. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 6640–6651. Association for Computational Linguistics, Online, July 2020. <https://doi.org/10.18653/v1/2020.acl-main.593>, <https://aclanthology.org/2020.acl-main.593>
17. Sun, T., et al.: Learning sparse sharing architectures for multiple tasks. In: AAAI (2020)
18. Teerapittayanon, S., McDanel, B., Kung, H.T.: BranchyNet: fast inference via early exiting from deep neural networks. In: 2016 23rd International Conference on Pattern Recognition (ICPR), pp. 2464–2469 (2016)

19. Tjong Kim Sang, E.F., De Meulder, F.: Introduction to the CoNLL-2003 shared task: language-independent named entity recognition. In: Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003, pp. 142–147 (2003). <https://aclanthology.org/W03-0419>
20. Vaswani, A., et al.: Attention is all you need. [arXiv:abs/1706.03762](https://arxiv.org/abs/1706.03762) (2017)
21. Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., Bowman, S.R.: GLUE: a multi-task benchmark and analysis platform for natural language understanding. arXiv preprint [arXiv:1804.07461](https://arxiv.org/abs/1804.07461) (2018)
22. Wolf, T., et al.: Transformers: state-of-the-art natural language processing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 38–45. Association for Computational Linguistics, Online, October 2020. [www.aclweb.org/anthology/2020.emnlp-demos.6](http://www.aclweb.org/anthology/2020.emnlp-demos.6)
23. Xin, J., Tang, R., Lee, J., Yu, Y., Lin, J.: DeeBERT: dynamic early exiting for accelerating BERT inference. arXiv preprint [arXiv:2004.12993](https://arxiv.org/abs/2004.12993) (2020)
24. Xin, J., Tang, R., Yu, Y., Lin, J.: BERxiT: early exiting for BERT with better fine-tuning and extension to regression. In: Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, pp. 91–104 (2021)
25. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., Le, Q.V.: XLNet: generalized autoregressive pretraining for language understanding. In: NeurIPS (2019)
26. Yu, J., Bohnet, B., Poesio, M.: Named entity recognition as dependency parsing. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 6470–6476. Association for Computational Linguistics, Online, July 2020. <https://doi.org/10.18653/v1/2020.acl-main.577>, <https://aclanthology.org/2020.acl-main.577>
27. Zhang, Z., Zhu, W., Zhang, J., Wang, P., Jin, R., Chung, T.S.: PCEE-BERT: accelerating BERT inference via patient and confident early exiting. In: Findings of the Association for Computational Linguistics: NAACL 2022, pp. 327–338. Association for Computational Linguistics, Seattle, United States, July 2022. <https://doi.org/10.18653/v1/2022.findings-naacl.25>, <https://aclanthology.org/2022.findings-naacl.25>
28. Zhou, W., Xu, C., Ge, T., McAuley, J., Xu, K., Wei, F.: BERT loses patience: fast and robust inference with early exit. [arXiv:abs/2006.04152](https://arxiv.org/abs/2006.04152) (2020)
29. Zhu, W., Wang, P., Ni, Y., Xie, G., Wang, X.: BADGE: speeding up BERT inference after deployment via block-wise bypasses and divergence-based early exiting. In: ACL Industry (2023)
30. Zhu, W., Wang, X., Ni, Y., Xie, G.: GAML-BERT: improving BERT early exiting by gradient aligned mutual learning. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 3033–3044. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, November 2021. <https://aclanthology.org/2021.emnlp-main.242>
31. Zhu, W., et al.: PANLP at MEDIQA 2019: pre-trained language models, transfer learning and knowledge distillation. In: Proceedings of the 18th BioNLP Workshop and Shared Task, pp. 380–388. Association for Computational Linguistics, Florence, Italy, August 2019. <https://doi.org/10.18653/v1/W19-5040>, <https://aclanthology.org/W19-5040>