Efficient Contextualized Representation: Language Model Pruning for Sequence Labeling

Liyuan Liu † **Xiang Ren** ♯ **Jingbo Shang** † **Jian Peng** † **Jiawei Han** †

† University of Illinois at Urbana-Champaign, Urbana, IL, USA

♯ University of Southern California, Los Angeles, CA, USA

†{ll2, shang7, jianpeng, hanj}@illinois.edu ♯{xiangren}@usc.edu

Abstract

Many efforts have been made to facilitate natural language processing tasks with pre-trained language models (PTLMs), and brought significant improvements to various applications. To fully leverage the nearly unlimited corpora and capture linguistic information of multifarious levels, large-size LMs are required; but for a specific task, only parts of these information are useful. Such large models, even in the inference stage, lead to overwhelming computation workloads, thus making them too time-consuming for real-world applications. For a specific task, we aim to keep useful information while compressing bulky PTLMs. Since layers of different depths keep different information, we can conduct the compression via layer selection. By introducing the dense connectivity, we can detach any layers without eliminating others, and stretch shallow and wide LMs to be deep and narrow. Moreover, PTLMs are trained with layer-wise dropouts for better robustness, and are pruned by a sparse regularization which is customized for our goal. Experiments on benchmarks demonstrate the effectiveness of our proposed method.

1 Introduction

Language modeling is a fundamental task in natural language processing (NLP) and understanding (NLU). Benefited from advances in neural networks (NNs) and the nearly unlimited corpora, recent neural language models are able to capture abundant linguistic information of multifarious levels. Therefore, it is desirable to facilitate other NLP and NLU tasks with these models.

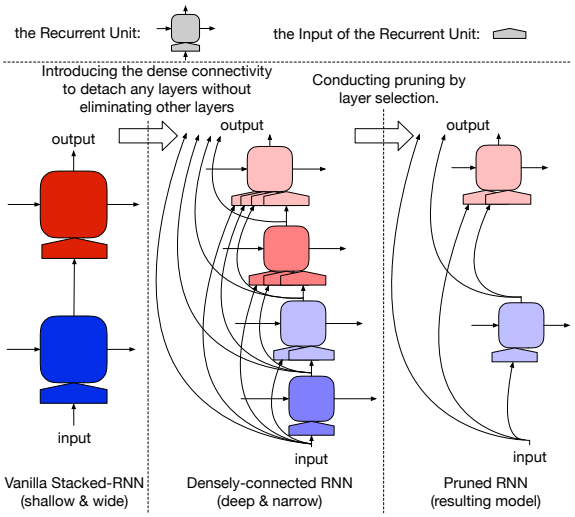


Figure 1: Leverage the dense connectivity to compress models via layer selection, and replace wide and shallow RNNs with deep and narrow ones.

There have been some initial attempts on employing pre-trained language models (PTLMs) as the contextualized representations (Peters et al., 2018, 2017). Such representations brought significant improvements to various NLP benchmarks, including up to 30% relative error reductions. However, for an informative PTLMs, gigantic NNs (e.g., LSTM with 8,192 hidden states) are required to extract multifarious linguistic information. Although those models can be directly integrated without retraining and only require the forward pass, they still introduce significant resource overheads which can be prohibitive for real-world applications.

Meanwhile, PTLMs are trained in a task-agnostic manner, thus providing useful but redundant information for a specific task. Therefore, there is a great potential to compress PTLMs for the target task and obtain efficient models without much loss of effectiveness.

Typical neural network compression methods require the retraining of the whole model (Mellempudi et al., 2017). However, neural language models are usually composed of RNNs, and their backpropagations require significantly more memories than their inference stages. It would become even more troublesome when the target task equips the coupled PTLMs to capture information in both directions. Therefore, these compression methods do not fit our scenario very well. In this paper, we aim to distill efficient and helpful models from such bulky PTLMs, while *avoiding costly retraining*.

Fortunately, in neural language models, layers of different depths naturally capture linguistic information of different levels (Peters et al., 2018). Hence, we propose to conduct layer selection to compress models, which retains useful layers for the target task and prunes irrelevant ones. However, for the widely-used stacked-LSTM, directly pruning any layers will eliminate all subsequent ones. To address this challenge, we introduce the dense connectivity. As shown in Fig. 1, it allows us to detach any layers while keeping all remaining ones, thus creating the basis to avoid retraining. Moreover, such connectivity can stretch shallow and wide LMs to be deep and narrow (Huang et al., 2016), and enables a more fine-grained layer selection.

To further avoid the costly retraining, we need to ensure the effectiveness of the pruned model. Specifically, we modify the L_1 regularization to encourage the selection weights to be not only sparse but binary, which protects the retained layer connections from shrinkage. Besides, we design a layer-wise dropout to make PTLMs more robust and better prepare them for the layer selection.

We refer our model as LD-Net, since the *layer selection* and the *dense connectivity* form the basis of our pruning methods. For evaluation, we apply LD-Net on two sequence labeling benchmark datasets, and demonstrated the effectiveness of the proposed method. In the CoNLL03 Named Entity Recognition (NER) task, the F_1 score increases from 90.78±0.24% to 91.86±0.15% by integrating the unpruned PTLMs. Meanwhile, after pruning over 90% calculation workloads from the best performing model[1] (92.03%), the resulting model still yields 91.84±0.14%. All codes are released[2].

[1]Based on their performance on the development sets

[2]github.com/LiyuanLucasLiu/LD-Net

2 LD-Net

Given a input sequence of T tokens, $\{x_1, x_2, \cdots, x_T\}$, we use $\mathbf{x}_t$ to denote the embedding for the t^{th} token, which is the input of the recurrent neural networks (RNNs) for language modeling or contextualized representation generation. The network is assumed to contain L layers. For the t^{th} time stamp, we mark the input and output of the l^{th} layer as $\mathbf{x}_{l,t}$ and $\mathbf{h}_{l,t}$, where $l \in [1, L], t \in [1, T]$.

2.1 RNN and Dense Connectivity

Despite various types of RNNs, we represent one RNN layer as a function:

$$\mathbf{h}_{l,t} = F_l(\mathbf{x}_{l,t}, \mathbf{h}_{l,t-1}) \quad (1)$$

where F_l is the recurrent unit of l^{th} layer, and it could be a Gated Recurrent Unit (GRU), a Long-short Term Memory (LSTM) or any other variants. We choose the vanilla LSTM in our experiments.

As deeper NNs usually have more representation powers, RNN layers are often stacked together to form the final model by setting $\mathbf{x}_{l,t} = \mathbf{h}_{l-1,t}$. These vanilla stacked-RNN models, however, suffer from problems like the vanishing gradient, and it's hard to train very deep models.

Recently, the dense connectivity and residual connectivity have been proposed to handle these problems (He et al., 2016; Huang et al., 2016). Specifically, dense connectivity refers to adding direct connections from any layer to all its subsequent layers. As illustrated in Fig. 1, the input of l^{th} layer is composed of the original input and the output of all preceding layers as follows.

$$\mathbf{x}_{l,t} = [\mathbf{x}_t, \mathbf{h}_{1,t}, \cdots, \mathbf{h}_{l-1,t}]$$

Similarly, the final output of the L-layer RNN is $\mathbf{h}_t = [\mathbf{x}_t, \mathbf{h}_{1,t}, \cdots, \mathbf{h}_{L,t}]$. With dense connectivity, we can detach any single layer without eliminating its subsequent layers (as in Fig. 1). Also, existing practices in computer vision demonstrate that such connectivities can lead to deep and narrow NNs and distribute parameters into different layers. Moreover, different layers in PTLMs usually capture linguistic information of different levels. Hence, we can compress PTLMs for a specific task by pruning unrelated or unimportant layers.

2.2 Language Modeling

Language modeling aims to describe the sequence generation. Normally, the generation probability

of the sequence $\{x_1, \cdots, x_T\}$ is defined in a "forward" manner:

$$p(x_1, \cdots, x_T) = \prod_{t=1}^{T} p(x_t | x_1, \cdots, x_{t-1}) \quad (2)$$

As to $p(x_t|x_1, \cdots, x_{t-1})$, it is computed based on the output of RNN, $\mathbf{h}_{t-1}$. Due to the dense connectivity, $\mathbf{h}_{t-1}$ is composed of outputs from different layers, which are designed to capture linguistic information of different levels. Similar to the bottleneck layers employed in the DenseNet (Huang et al., 2016), we may need additional layers to unify such information. Accordingly, we add an projection layer with the ReLU activation function:

$$\mathbf{h}^*_{t-1} = \mathrm{ReLU}(W_{proj} \cdot \mathbf{h}_{t-1} + \mathbf{b}_{proj}) \quad (3)$$

Based on $\mathbf{h}^*_{t-1}$, it's intuitive to calculate $p(x_t|x_1, \cdots, x_{t-1})$ by the softmax function, i.e., $\mathrm{softmax}(W_{out} \cdot \mathbf{h}^*_{t-1} + \mathbf{b})$.

Since the training of language models needs nothing but the raw text, it has almost unlimited corpora. However, conducting training on extensive corpora results in a huge dictionary, and makes calculating the vanilla softmax intractable. Several techniques have been proposed to handle this problem, including adaptive softmax (Grave et al., 2016), slim word embedding (Li et al., 2017), the sampled softmax and the noise contrastive estimation (Jozefowicz et al., 2016). Since the major focus of our paper does not lie in the language modeling task, we choose the adaptive softmax because of its practical efficiency when accelerated with GPUs.

2.3 Contextualized Representation

As pre-trained LMs can describe the text generation accurately, they can be utilized to extract information and construct features for other tasks. These features, referred as contextualized representations, have been demonstrated to be essentially useful (Peters et al., 2018). To capture information from both directions, we utilized not only forward LMs, but also backward LMs. Backward LMs are based on Eqn. 4 instead of Eqn. 2. Similar to forward LMs, backward LMs model $p(x_t|x_{t+1}, \cdots, x_T)$ using NNs. For reference, the output of the backward LMs' RNNs for predicting x_t is recorded as $\mathbf{h}^r_{t+1}$.

$$p(x_1, \cdots, x_n) = \prod_{t=1}^{T} p(x_t | x_{t+1}, \cdots, x_T) \quad (4)$$

We refer the forward and backward LMs as the bi-directional LMs. Actually, the outputs of such LMs can be viewed as the skip-thought vectors (Kiros et al., 2015), which tries to embed sentence and conducts training by predicting the previous and afterwards sentences. Specifically, the skip-thought model contains an encoder and two decoders. The encoder would embed a sentence as a vector, and the decoder would try to predict the previous or afterwards sentence in a conditional language modeling manner.

At the same time, feeding a sentence ($\{x_1, \cdots, x_{t-1}\}$) to the forward LMs, we would get $\mathbf{h}_{t-1}$, while the forward LMs would try to decode the afterwards sentence based on $\mathbf{h}_{t-1}$. Also, the backward LMs would generate $\mathbf{h}^r_{h+1}$, which is used to decode the previous sentence. Accordingly, $\mathbf{h}$ and $\mathbf{h}^r$ can be viewed as the sentence embedding, thus capturing the context information.

Thus, we choose to use the output of the densely connected RNN ($\mathbf{h}_t$ and $\mathbf{h}^r_t$) for constructing the contextualized representation. Since the dimensions of $\mathbf{h}_t$ could be too large for the end task, we add a non-linear transformation to calculate the contextualized representation ($\mathbf{r}_t$):

$$\mathbf{r}_t = \mathrm{ReLU}(W_{cr} \cdot [\mathbf{h}_t, \mathbf{h}^r_t] + \mathbf{b}_{cr}) \quad (5)$$

Our proposed method bears the same intuition as the ELMo (Peters et al., 2018). ELMo is designed for the vanilla stacked-RNN, and tries to calculate a weighted average of different layers' outputs as the contextualized representation. Our method, benefited from the dense connectivity and its narrow structure, can directly combine the outputs of different layers by concatenation. It does not assume the outputs of different layers to be in the same vector space, thus having more potentials.

2.4 Layer Selection

Typical model compression methods require retraining or gradient calculation. For the coupled PTLMs, these methods require even more computation resources compared to the training of PTLMs, thus not fitting our scenario very well.

Benefited from the dense connectivity, we are able to train deep and narrow networks. Moreover, we can detach one of its layer without eliminating all subsequent layers (as in Fig. 1). Since different layers in NNs could capture different linguistic information, only a few of them would be relevant

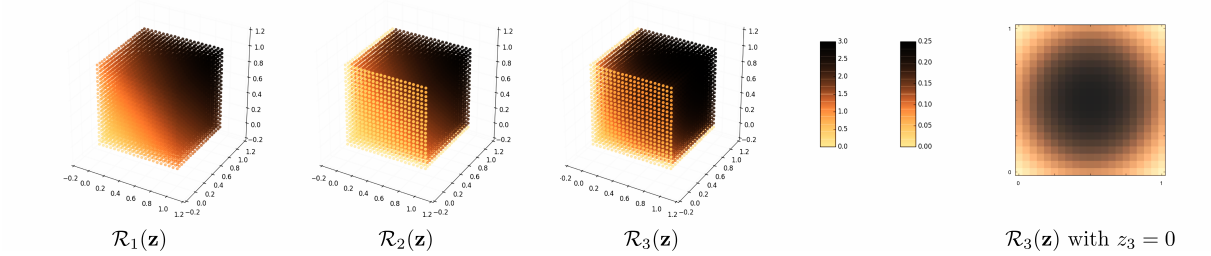


Figure 2: Penalty values of various $\mathcal{R}$ for $\mathbf{z}$ with three dimensions. λ_1 has been set to 2 for $\mathcal{R}_2$ and $\mathcal{R}_3$.

or useful for a specific task. As a result, we try to compress these models by the task-guided layer selection. For i-th layer, we introduce a binary mask $z_i \in \{0, 1\}$ and calculate $\mathbf{h}_{l,t}$ with Eqn. 6 instead of Eqn. 1.

$$\mathbf{h}_{l,t} = z_i \cdot F_l(\mathbf{x}_{l,t}, \mathbf{h}_{l,t-1}) \tag{6}$$

With this setting, we can conduct a layer selection by optimizing the regularized empirical risk:

$$\min \mathcal{L} + \lambda_0 \cdot \mathcal{R} \tag{7}$$

where $\mathcal{L}$ is the empirical risk for the sequence labeling task and $\mathcal{R}$ is the sparse regularization.

The ideal choice for $\mathcal{R}$ would be the L_0 regularization of $\mathbf{z}$, i.e., $\mathcal{R}_0(\mathbf{z}) = |\mathbf{z}|_0$. However, it is not continuous and cannot be efficiently optimized. Hence, we relax z_i from binary to a real value (i.e., $0 \leq z_i \leq 1$) and replace $\mathcal{R}_0$ by:

$$\mathcal{R}_1 = |\mathbf{z}|_1$$

Despite the sparsity achieved by $\mathcal{R}_1$, it could hurt the performance by shifting all z_i far away from 1. Such shrinkage introduces additional noise in $\mathbf{h}_{l,t}$ and $\mathbf{x}_{l,t}$, which may result in ineffective pruned PTLMs. Since our goal is to conduct pruning without retraining, we further modifies the L_1 regularization to achieve sparsity while alleviating its shrinkage effect. As the target of $\mathcal{R}$ is to make $\mathbf{z}$ sparse, it can be "turned-off" after achieving a satisfying sparsity. Therefore, we extend $\mathcal{R}_1$ to a margin-based regularization:

$$\mathcal{R}_2 = \delta(|\mathbf{z}|_0 > \lambda_1)|\mathbf{z}|_1$$

In addition, we also want to make up the relaxation made on $\mathbf{z}$, i.e., relaxing its values from binary to $[0, 1]$. Accordingly, we add the penalty $|\mathbf{z}(1 - \mathbf{z})|_1$ to encourage $\mathbf{z}$ to be binary (Murray and Ng, 2010) and modify $\mathcal{R}_2$ into $\mathcal{R}_3$:

$$\mathcal{R}_3 = \delta(|\mathbf{z}|_0 > \lambda_1)|\mathbf{z}|_1 + |\mathbf{z}(1 - \mathbf{z})|_1$$

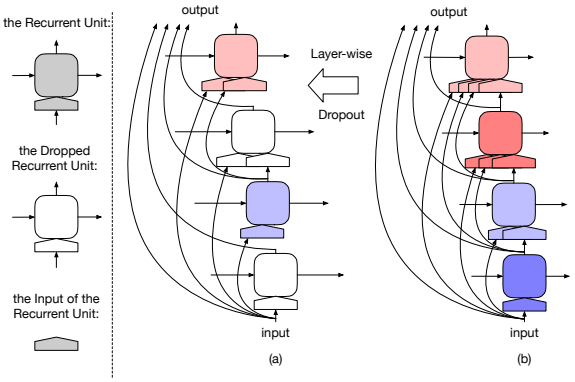


Figure 3: Layer-wise dropout conducted on a 4-layer densely connected RNN. (a) is the remained RNN. (b) is the original densely connected RNN.

To compare $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$, we visualize their penalty values in Fig. 2. The visualization is generated for a 3-dimensional $\mathbf{z}$ while the targeted sparsity, λ_1, is set to 2. Comparing to $\mathcal{R}_1$, we can observe that $\mathcal{R}_2$ enlarges the optimal point set from $\mathbf{0}$ to all $\mathbf{z}$ with a satisfying sparsity, thus avoid the over-shrinkage. To better demonstrate the effect of $\mathcal{R}_3$, we further visualize its penalties after achieving a satisfying sparsity (w.l.o.g., assuming $z_3 = 0$). One can observe that it penalizes non-binary $\mathbf{z}$ and favors binary values.

2.5 Layer-wise Dropout

So far, we've customized the regularization term for the layer-wise pruning, which protects the retained connections among layers from shrinking. Still, to avoid retraining, we need to prevent the pruned PTLMs from crashing, i.e., cannot work properly with a pruned input. One possible solution is to prepare the PTLMs for the pruned inputs, thus making them more robust to pruning.

Accordingly, we conduct the training of PTLMs with a layer-wise dropout. As in Fig. 3, a random part of layers in the PTLMs are randomly dropped during the each batch. The outputs of the dropped layers will not be passed to their subsequent recurrent layers, but will be sent to the projection layer

(Eqn. 3) for predicting the next word. In other words, this dropout is only applied to the input of recurrent layers, which aims to imitate the pruned input without totally removing any layers.

3 Sequence Labeling

In this section, we will introduce our sequence labeling architecture, which is augmented with the contextualized representations.

3.1 Neural Architecture

Following the recent studies (Liu et al., 2017; Kuru et al., 2016), we construct the neural architecture as in Fig. 4. Given the input sequence $\{x_1, x_2, \cdots, x_T\}$, for t^{th} token (x_t), we assume its word embedding is $\mathbf{w}_t$, its label is y_t, and its character-level input is $\{c_{i,1}, c_{i,2}, \cdots, c_{i,\_}\}$, where $c_{i,\_}$ is the space character following x_t.

The character-level representations have become the required components for most of the state-of-the-art. Following the recent study (Liu et al., 2017), we employ LSTMs to take the character-level input in a context-aware manner, and mark its output for x_t as $\mathbf{c}_t$. Similar to the contextualized representation, $\mathbf{c}_t$ usually has more dimensions than $\mathbf{w}_t$. To integrate them together, we set the output dimension of Eqn. 5 as the dimension of $\mathbf{w}_t$, and project $\mathbf{c}_t$ to a new space with the same dimension number. We mark the projected character-level representation as $\mathbf{c}_t^*$.

After projections, these vectors are concatenated as $\mathbf{v}_t = [\mathbf{c}_t^*; \mathbf{r}_t; \mathbf{w}_t], \forall i \in [1, T]$ and further fed into the word-level LSTMs. We refer their output as $\mathbf{U} = \{\mathbf{u}_1, \cdots, \mathbf{u}_T\}$. To ensure the model to predict valid label sequences, we append a first-order conditional random field (CRF) layer to the model (Lample et al., 2016). Specifically, the model defines the generation probability of $\mathbf{y} = \{y_1, \cdots, y_T\}$ as

$$p(\mathbf{y}|\mathbf{U}) = \frac{\prod_{t=1}^{T} \phi(y_{t-1}, y_t, \mathbf{u}_t)}{\sum_{\hat{\mathbf{y}} \in \mathbf{Y}(\mathbf{U})} \prod_{t=1}^{T} \phi(\hat{y}_{t-1}, \hat{y}_t, \mathbf{u}_t)} \quad (8)$$

where $\hat{\mathbf{y}} = \{\hat{y}_1, \ldots, \hat{y}_T\}$ is a generic label sequence, $\mathbf{Y}(\mathbf{U})$ is the set of all generic label sequences for $\mathbf{U}$ and $\phi(y_{t-1}, y_t, \mathbf{u}_t)$ is the potential function. In our model, $\phi(y_{t-1}, y_t, \mathbf{u}_t)$ is defined as $\exp(W_{y_t}\mathbf{u}_t + b_{y_{t-1},y_t})$, where W_{y_t} and b_{y_{t-1},y_t} are the weight and bias parameters.

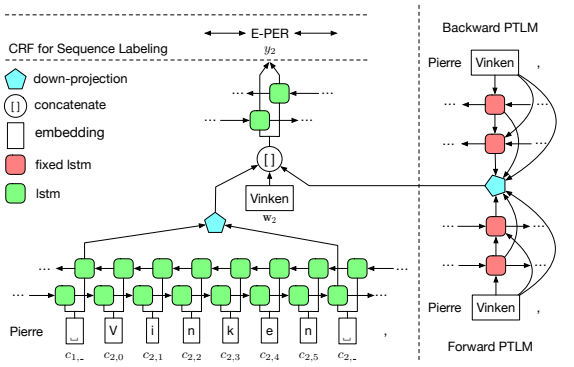


Figure 4: The proposed sequence labeling architecture with contextualized representations.

3.2 Model Training and Inference

We use the following negative log-likelihood as the empirical risk.

$$\mathcal{L} = -\sum_{\mathbf{U}} \log p(\mathbf{y}|\mathbf{U}) \quad (9)$$

For testing or decoding, we want to find the optimal sequence $\mathbf{y}^*$ that maximizes the likelihood.

$$\mathbf{y}^* = \underset{\mathbf{y} \in \mathbf{Y}(\mathbf{U})}{\operatorname{argmax}}\ p(\mathbf{y}|\mathbf{U}) \quad (10)$$

Although the denominator of Eq. 8 is complicated, we can calculate Eqs. 9 and 10 efficiently by the Viterbi algorithm.

For optimization, we decompose it into two steps, i.e., model training and model pruning.

Model training. We set λ_0 to 0 and optimize the empirical risk without any regularization, i.e., $\min \mathcal{L}$. In this step, we conduct optimization with the stochastic gradient descent with momentum. Following (Peters et al., 2018), dropout would be added to both the coupled PTLMs and the sequence labeling model.

Model pruning. We conduct the pruning based on the checkpoint which has the best performance on the development set during the model training. We set λ_0 to non-zero values and optimize $\min \mathcal{L} + \lambda_0 \mathcal{R}_3$ by the projected gradient descent with momentum. Any layer i with $z_i = 0$ would be deleted in the final model to complete the pruning. To get a better stability, dropout is only added to the sequence labeling model.

4 Experiments

We will first discuss the capability of the LD-Net as language models, then explore the effectiveness of its contextualized representations in sequence labeling tasks.

Network	Ind. #	Hid. #	Layer #	Param.# ($\cdot 10^7$)		PPL
				RNN	Others	
8192-1024 (Jozefowicz et al., 2016)	1	8192	2	180 (Total)		30.6
CNN-8192-1024 (Jozefowicz et al., 2016)	2	8192	2	104 (Total)		30.0
CNN-4096-512 (Peters et al., 2018)	3	4096	2	15.1†	40.6†	39.7
2048-Adaptive (Grave et al., 2016)	4	2048	2	5.2†	26.5†	39.8
2048-512 (Peters et al., 2017)	5	2048	1	2.1†	40.6†	47.50
vanilla LSTM	6	2048	2	5.3	25.6	40.27
	7	1600	2	3.2	24.2	48.85
LD-Net without Layer-wise Dropout	8	300	10	2.3	24.2	45.14
LD-Net with Layer-wise Dropout	9	300	10	2.3	24.2	50.06

Table 1: Performance comparison of language models. For models without official published parameter numbers, their values (marked with†) are estimated by assuming the vanilla LSTMs are adopted.

4.1 Language Modeling

For comparison, we conducted experiments with the one billion word benchmark dataset (Chelba et al., 2013) with both LD-Net (with 1,600 dimensional projection) and the vanilla stacked-LSTM. Both kinds of models use word embedding (random initialized) of 300 dimension as input and use the adaptive softmax (with default setting) as an approximation of the full softmax. Additionally, as preprocessing, we replace all tokens occurring equal or less than 3 times with as UNK, which shrinks the dictionary from 7.9M to 6.4M.

The optimization is performed by the Adam algorithm (Kingma and Ba, 2014), the gradient is clipped at 5.0 and the learning rate is set to start from 0.001. The layer-wise dropout ratio is set to 0.5, the RNNs are unrolled for 20 steps without resetting the LSTM states, and the batch size is set to 128. Their performances are summarized in Table 1, together with several LMs used in our sequence labeling baselines. For models without official reported parameter numbers, we estimate their values (marked with†) by assuming they adopted the vanilla LSTM. Note that, for models 3, 5, 6, 7, 8, and 9, PPL refers to the averaged perplexity of the forward and the backward LMs.

We can observe that, for those models taking word embedding as the input, embedding composes the vast majority of model parameters. However, embedding can be embodied as a "sparse" layer which is computationally efficient. Instead, the intense calculations are conducted in RNN layers and softmax layer for language modeling, or RNN layers for contextualized representations. At the same time, comparing the model 8192-1024 and CNN-8192-1024, their only difference is the input method. Instead of taking word embedding as the input, CNN-8192-1024 utilizes CNN to compose word representation from the character-level input. Despite the greatly reduced parameter number, the perplexity of the resulting models remains almost unchanged. Since replacing embedding layer with CNN would make the training slower, we only conduct experiments with models taking word embedding as the input.

Comparing LD-Net with other baselines, we think it achieves satisfactory performance with regard to the parameter number. This result demonstrates the LD-Net's capability of capturing the underlying structure of natural language. Meanwhile, we find that the layer-wise dropout makes it harder to train LD-Net and its resulting model achieves less competitive results. However, as would be discussed in the next section, layer-wise dropout allows the resulting model to generate better contextualized representations and be more robust to pruning, even with a higher perplexity.

4.2 Sequence Labeling

Following TagLM (Peters et al., 2017), we evaluate our methods in two benchmark datasets, the CoNLL03 NER task (Tjong Kim Sang and De Meulder, 2003) and the CoNLL00 Chunking task (Tjong Kim Sang and Buchholz, 2000).

CoNLL03 NER has four entity types: `PER`, `LOC`,

ORG, and MISC and includes the standard training, development and test sets.

CoNLL00 chunking defines eleven syntactic chunk types (e.g., NP and VP) in addition to Other. Since it only includes training and test sets, we sampled 1000 sentences from training set as a held-out development set (Peters et al., 2017).

In both cases, we use the BIOES labeling scheme for making predictions (Ratinov and Roth, 2009) and use the micro-averaged F_1 as the evaluation metric. Based on the analysis conducted in the development set, we set $\lambda_0 = 0.05$ for the NER task, and $\lambda_0 = 0.5$ for the Chunking task. As discussed before, we conduct optimization with the stochastic gradient descent with momentum. We set the batch size, the momentum, and the learning rate to 10, 0.9, and $\eta_t = \frac{\eta_0}{1+\rho t}$ respectively. Here, $\eta_0 = 0.015$ is the initial learning rate and $\rho = 0.05$ is the decay ratio. Dropout is applied in our model, and its ratio is set to 0.5. For a better stability, we use gradient clipping of 5.0. Furthermore, we employ the early stopping in the development set and report averaged score across five different runs.

Regarding the network structure, we use the character-level embedding of 30 dimensions. Both the character-level and word-level RNNs are set to one-layer LSTM with 150-dimension hidden states in each direction. The GloVe 100-dimension pre-trained word embedding[3] is used as the initialization of word embedding $\mathbf{w}_t$, and will be fine-tuned during the training. The layer selection variables z_i are initialized as 1, remained unchanged during the model training and only be updated during the model pruning. All other variables are randomly initialized (Glorot and Bengio, 2010).

Compared methods. The first baseline, referred as NoLM, is our sequence labeling model without the contextualized representations, i.e., calculating $\mathbf{v}_t$ as $[\mathbf{c}^*_t; \mathbf{w}_t]$ instead of $[\mathbf{c}^*_t; \mathbf{r}_t; \mathbf{w}_t]$. Besides, ELMo (Peters et al., 2018) is the major baseline. To make comparison more fair, we implemented the ELMo model and use it to calculate the $\mathbf{r}_t$ in Eqn. 5 instead of $[\mathbf{h}_t, \mathbf{h}^r_t]$. Results of re-implemented models are referred with R-ELMo (λ is set to the recommended value, 0.1) and the results reported in its original paper are referred with O-ELMo. Additionally, since TagLM (Peters et al., 2017) with one-layer NNs can be viewed as a special case of ELMo, we also include its results.

[3] nlp.stanford.edu/projects/glove/

Network (PTLMs Ind.#)	Avg. ppl	#FLOPs ($\cdot 10^6$)	F_1 score (avg±std)
NoLM (/)	/	3	94.42±0.08
R-ELMo (6)	40.27	215	96.19±0.07
R-ELMo (7)	48.85	135	95.86±0.04
LD-Net (8)	45.14	98	96.01±0.07
LD-Net (9)	50.06	98	96.05±0.08
LD-Net (8*)	origin	98	96.13
	pruned	23	95.46±0.18
LD-Net (9*)	origin	98	96.15
	pruned	17	95.66±0.04

Table 2: Performance comparisons in the CoNLL00 Chunking task.

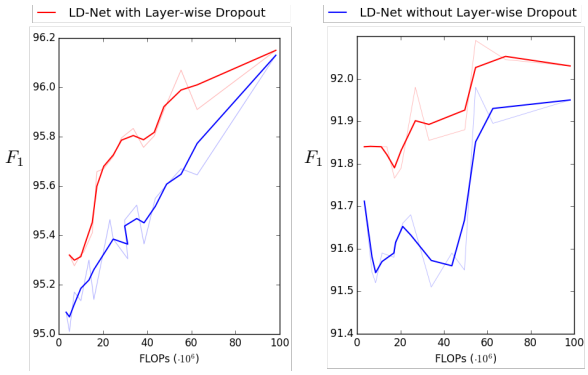
Sequence labeling results. Table 2 and 3 summarizes the results of LD-Net and baselines. Besides the F_1 score and averaged perplexity, we also estimate FLOPs (i.e., the number of floating-point multiplication-adds) for the efficiency evaluation. Since our model takes both word-level and character-level inputs, we estimated the FLOPs value for a word-level input with 4.39 character-level inputs, while 4.39 is the averaged length of words in the CoNLL03 dataset.

Before the model pruning, LD-Net achieves a 96.05±0.08 F_1 score in the CoNLL00 Chunking task, yielding nearly 30% error reductions over the NoLM baseline. Also, it scores 91.86±0.15 F_1 in the CoNLL03 NER task with over 10% error reductions. Similar to the language modeling, we observe that the most complicated models achieve the best perplexity and provide the most improvements in the target task. Still, considering the number of model parameters and the resulting perplexity, our model demonstrates its effectiveness in generating contextualized representations. For example, comparing to our methods, R-ELMo (7) leverages PTLMs with the similar perplexity and parameter number, but cannot get the same improvements with our method on both datasets. This phenomenon shows that LD-Net is more effective than the stacked-LSTM for contextualized representations.

Meanwhile, although the layer-wise dropout makes the model harder to train, their resulting PTLMs can generate better contextualized representations, even without the same perplexity. Also, as discussed in (Peters et al., 2018, 2017),

Network (PTLMs Ind.#)	Avg. ppl	#FLOPs ($\cdot 10^6$)	F_1 score (avg±std)
NoLM (/)	/	3	90.78±0.24
O-ELMo (3)	39.70	607†	92.22±0.10
R-ELMo (6)	40.27	215	91.99±0.24
R-ELMo (7)	48.85	135	91.54±0.10
TagLM (5)	47.50	87†	91.62±0.23
LD-Net (8)	45.14	98	91.76±0.18
LD-Net (9)	50.06	98	91.86±0.15
LD-Net (8*)	origin pruned	98 6	91.95 91.55±0.06
LD-Net (9*)	origin pruned	98 6	92.03 91.84±0.14

Table 3: Performance comparison in the CoNLL03 NER task. We estimate the #FLOPs for O-ELMo and TagLM by assuming they adopted the vanilla LSTMs / GRUs (marked with †).



(a) ConNLL00 Chunking (b) ConNLL03 NER

Figure 5: The performance of pruned models in two tasks w.r.t. their efficiency (FLOPs).

the performance of the contextualized representation can be further improved by adopting larger models, using the CNN input or fine-tuning on the target corpus.

For the pruning, we started from the model with the best performance on the development set (referred with "origin"), and refer the performances of pruned models with "pruned" in Table 2 and 3. Essentially, we can observe the pruned models get rid of the vast majority of calculation while still retaining a significant improvement. We will discuss more on the pruned models in Sec. 4.3.

4.3 Case Studies

To further explore the effect of pruning, we adjust the expected sparsity λ_1 and visualize the performance curve of pruned models w.r.t. their efficiency in Fig 5. We can observe that LD-Net obviously outperforms its variants and demonstrates the effectiveness of the layer-wise dropout.

In the CoNLL03 NER task, we can observe that, PTLMs can be pruned to a relative small size without much loss of efficiency. As in Table 3, we can observe that, after pruning over 90% calculations, the resulting model only increases about 2% errors, yielding a very competitive performance. As for the CoNLL00 Chunking task, the performance of LD-Net decays in a faster rate than in the NER task. For example, after pruning over 80% calculations, the resulting model increases about 13% errors. Considering the fact that this corpus is only half the size of the CoNLL03 NER dataset, we can expect the resulting models have more dependent on the PTLMs. Still, the chunked model reduces about 25% of error over the NoLM baseline.

5 Related Work

Sequence labeling. Linguistic sequence labeling is one of the fundamental tasks in NLP, encompassing various applications including POS tagging, chunking, and NER. Many attempts have been made to conduct end-to-end learning and build reliable models without handcrafted features (Chiu and Nichols, 2016; Lample et al., 2016; Ma and Hovy, 2016). These models incorporate character-level layer and CRF layer, and report meaningful improvements over baselines.

Language modeling. Language modeling has been a core task in NLP. Many attempts have been paid to develop better neural language models (Zilly et al., 2016; Inan et al., 2016; Godin et al., 2017; Melis et al., 2017). Specifically, with extensive corpora, language models can be well trained to generate high-quality sentences from scratch (Jozefowicz et al., 2016; Grave et al., 2016; Li et al., 2017; Shazeer et al., 2017). Meanwhile, initial attempts have been made to improve the performance of other tasks with these methods. Some methods treat the language modeling as an additional supervision, and conduct co-training for knowledge transfer (Liu et al., 2017; Rei, 2017). Others, including this paper, aim to construct additional features (referred as contextualized representations) with the pre-trained language mod-

els (Peters et al., 2017, 2018).

Neural Network Acceleration. There are mainly three kinds of NN acceleration methods, i.e., prune network into smaller sizes (Han et al., 2015; Wen et al., 2016), converting float operation into customized low precision arithmetic (Hubara et al., 2016; Courbariaux et al., 2016), and using shallower networks to mimic the output of deeper ones (Hinton et al., 2015; Romero et al., 2014). However, most of them require costly retraining.

6 Conclusion

In this paper, we proposed LD-Net, a novel framework for efficient contextualized representation. As demonstrated on two benchmarks, it can conduct layer-wise pruning on PTLMs for a specific sequence labeling task. Moreover, it requires neither the gradient oracle of PTLMs nor the costly retraining. In the future, we plan to apply LD-Net to other NLP / NLU applications.

References

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. Technical report, Google. http://arxiv.org/abs/1312.3005.

Jason P. C. Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional lstm-cnns. *TACL* .

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* .

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*.

Fréderic Godin, Joni Dambre, and Wesley De Neve. 2017. Improving language modeling using densely connected recurrent neural networks. *arXiv preprint arXiv:1707.06130* .

Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. 2016. Efficient softmax approximation for gpus. *arXiv preprint arXiv:1609.04309* .

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. pages 1135–1143.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *European Conference on Computer Vision*. Springer, pages 630–645.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* .

Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. 2016. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993* .

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061* .

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462* .

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv:1602.02410* .

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Advances in neural information processing systems*. pages 3294–3302.

Onur Kuru, Ozan Arkan Can, and Deniz Yuret. 2016. Charner: Character-level named entity recognition. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. pages 911–921.

Guillaume Lample, Miguel Ballesteros, Kazuya Kawakami, Sandeep Subramanian, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *NAACL-HLT*.

Zhongliang Li, Raymond Kulhanek, Shaojun Wang, Yunxin Zhao, and Shuang Wu. 2017. Slim embedding layers for recurrent neural language models. *arXiv preprint arXiv:1711.09873* .

Liyuan Liu, Jingbo Shang, Frank Xu, Xiang Ren, Huan Gui, Jian Peng, and Jiawei Han. 2017. Empower sequence labeling with task-aware neural language model. *arXiv preprint arXiv:1709.04109* .

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *ACL*.

Gábor Melis, Chris Dyer, and Phil Blunsom. 2017. On the state of the art of evaluation in neural language models. *CoRR* abs/1707.05589.

Naveen Mellempudi, Abhisek Kundu, Dheevatsa Mudigere, Dipankar Das, Bharat Kaul, and Pradeep Dubey. 2017. Ternary neural networks with fine-grained quantization. *arXiv preprint arXiv:1705.01462* .

Walter Murray and Kien-Ming Ng. 2010. An algorithm for nonlinear optimization problems with binary variables. *Computational Optimization and Applications* 47(2):257–288.

Matthew E Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. *arXiv:1705.00108* .

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365* .

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *CoNLL*.

Marek Rei. 2017. Semi-supervised multitask learning for sequence labeling. In *ACL*.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* .

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* .

Erik F Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the conll-2000 shared task: Chunking. In *Learning language in logic and CoNLL*.

Erik F Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Natural language learning at NAACL-HLT*.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*. pages 2074–2082.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutnk, and Jrgen Schmidhuber. 2016. Recurrent Highway Networks. *arXiv preprint arXiv:1607.03474* .