SC-LSTM: Learning Task-Specific Representations in Multi-Task Learning for Sequence Labeling

Peng Lu
RALI-DIRO
Université de Montréal
Montréal, Canada
peng.lu@umontreal.ca

Ting Bai[*]
School of Information
Renmin University of China
Beijing, China
baiting@ruc.edu.cn

Philippe Langlais
RALI-DIRO
Université de Montréal
Montréal, Canada
felipe@iro.umontreal.ca

Abstract

Multi-task learning (MTL) has been studied recently for sequence labeling. Typically, auxiliary tasks are selected specifically in order to improve the performance of a target task. Jointly learning multiple tasks in a way that benefit all of them simultaneously can increase the utility of MTL. In order to do so, we propose a new LSTM cell which contains both shared parameters that can learn from all tasks, and task-specific parameters that can learn task specific information. We name it a Shared-Cell Long-Short Term Memory (SC-LSTM). Experimental results on three sequence labeling benchmarks (named-entity recognition, text chunking, and part-of-speech tagging) demonstrate the effectiveness of our SC-LSTM cell.

1 Introduction

As one of the fundamental tasks in NLP, sequence labeling has been studied for years. Before the blooming of neural network methods, handcrafted features were widely used in traditional approaches like CRFs, HMMs, and maximum entropy classifiers (Lafferty et al., 2001; McCallum et al., 2000; McCallum and Li, 2003; Florian et al., 2003). However, applying them to different tasks or domains is hard. Recently, instead of using handcrafted features, end-to-end neural network based systems have been developed for sequence labeling tasks, such as LSTM-CNN (Chiu and Nichols, 2015), LSTM-CRF (Huang et al., 2015; Lample et al., 2016), and LSTM-CNN-CRF (Ma and Hovy, 2016). These models utilize LSTM to encode the global information of a sentence into a word-level representation of its tokens, which avoids manual feature engineering. Moreover, by incorporating a character-level representation of tokens, these models further improve.

In many such studies, though, neural network models are trained toward a single task in a supervised way by making use of relatively small annotated training material. Jointly learning multiple tasks can reduce the risk of over-fitting to one task, and many attempts have been made at doing so for sequence labeling tasks (Caruana, 1997; Collobert and Weston, 2008; Collobert et al., 2011). Results so far are not conclusive.

Some works have reported negative results overall. For instance in their pioneering work, Collobert et al. (2011) observed that training their model on NER, POS tagging and chunking altogether led to slight decrease in performance compared to a similar model trained on each task separately. Søgaard and Goldberg (2016) study chunking and CCG super tagging, coupled with an additional POS tagging task. They do report gains on both target tasks over single task models, but results varied depending where the additional task was taken care of in their architecture. The authors actually reported a failure to leverage other labelling tasks, and concluded that combined tasks should be sufficiently similar to the target one, for significant gains to be observed. Similarly, Alonso and Plank (2017) achieved significant improvements for only 1 out of 5 tasks considered. Also of interest is the work of (Changpinyo et al., 2018) where the authors investigate the classical shared encoder-based MTL framework (Collobert et al., 2011; Collobert and Weston, 2008) on 11 sequence labeling datasets including POS, NER, and chunking. They report that chunking is beneficial to NER, while POS tagging can be harmful.

We present in Section 2 the two major approaches proposed for multi-task learning and discuss their limitations. We describe our approach in Section 3, and present our experimental settings and results in Section 4 and 5 respectively. We further analyze our approach in Section 6, discuss

[*] Co-first author.

related works in Section 7 and conclude in Section 8.

2 Multi-task Learning

2.1 Problem Statement

We are given a set of K tasks (in our case named-entity recognition, text chunking and part of speech tagging) that we want to train jointly in an end-to-end fashion. Each task k has an associated training set $S_k = \{(x_i^k, y_i^k)_{i \in [1,n_k]}\}$ of n_k examples, where x_i^k and y_i^k are sequences of size m_i of tokens and tags respectively. We wish to learn a single function $\mathcal{F}$ which maps any token input sequence x_i to its task-specific labels, where the mapping defines a probabilistic distribution for each involved task: $p(y_i^1, \dots, y_i^K) = \mathcal{F}(x_i)$.

2.2 Existing Shared Encoder Methods

There are two kinds of neural-based MTL methods. The first one — LSTM-s hereafter — uses an identical representation for all tasks, as proposed in (Collobert et al., 2011). This is illustrated in Figure 1 (left), where 3 layers of LSTMs are being stacked.[1] While different tasks directly interact with all parameters of the model, this increases the risk of optimization conflicts when gold-standard labels from different tasks have no significant correlation.

The second class of multi-task architectures is depicted in the middle part of Figure 1, and is named LSTM-d hereafter. In this configuration, each LSTM layer feeds a task-specific classifier and serves as input to the next stacked LSTM layer (Søgaard and Goldberg, 2016). The underlying assumption is that tasks may be ordered in such a way that easier tasks are learned first, the target tasks being the latest one considered, thus benefiting the hidden state of the lower layers. One drawback, however, is that one must decide which task to consider first, a decision which may impact the overall performance. Furthermore, using the hidden state of lower layers increases the limitation of learning representation for that task.

We believe that one reason for the lack of consistent benefits of MTL in the labelling literature is that the proposed models share all or part of parameters for extracting hidden states, which leads to optimization conflicts when different tasks require different features. We believe it would be helpful if we make the model have ability to learn a task-specific representation (Ammar et al., 2016; Östling and Tiedemann, 2016; Kiperwasser and Ballesteros, 2018) at the same time. This observation led us to design a new LSTM cell which allows at almost no additional computation cost to efficiently train a single RNN-based model, where task-specific labelers clearly outperform their singly-tasked counterparts. Actually, by training our model on NER, chunking and POS tagging, we report state-of-the-art (or highly competitive) results on each task, without using external knowledge (such as gazetteers that has been shown to be important for NER), or hand-picking tasks to combine.

[1]This typically delivers better performance than having just one. In practice also, LSTM layers are replaced by bi-LSTM ones.

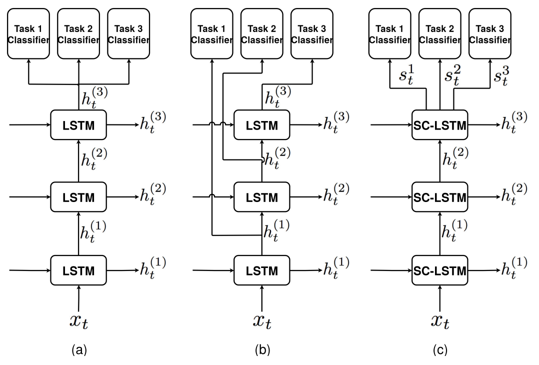

Figure 1: Overview of three shared encoder MTL tagging system: (a) LSTM-s (b) LSTM-d (c) our SC-LSTM based system.

3 Proposed Method

Our solution is depicted in the right part of Figure 1 and detailed in the next section. It is actually very similar to the LSTM-s one, except that each LSTM layer passes on task-specific hidden representations that learn the peculiarities of individual tasks. In the last layer, each classifier is fed with a concatenation of a global representation (as in LSTM-s) and the task-specific one. By doing so, we keep the advantages of both aforementioned approaches, where one task can have its task-specific representation as in LSTM-d, while not enforcing any task order, further giving the freedom to the model to learn specificities of each task. For this architecture to work, we need to modify the classical LSTM cell, which is described in the next section.

3.1 LSTM Cell

An LSTM cell (Hochreiter and Schmidhuber, 1997) is made up of four functional gates which control the input and output of the memory state $\mathbf{c_t}$: a forget gate $\mathbf{f_t}$ controls what information to remove from the memory state of the last time step, an input gate $\mathbf{i_t}$ controls the information to add to the current memory, and an output gate $\mathbf{o_t}$ controls what information to release from the current memory state. This mechanism is formalized in Equation 1 where $\mathbf{x}_t$, $\mathbf{h}_t$ is the input vector and hidden vector at time step t, σ is the sigmoid function, $\hat{\mathbf{c}}$ is the new candidate state, $\mathbf{c}_t$ is the memory state, which encodes information of the current input and history information, and $*$ indicates the element-wise product. $\mathbf{W}_f, \mathbf{U}_f, \mathbf{W}_i, \mathbf{U}_i, \mathbf{W}_o, \mathbf{U}_o, \mathbf{W}_c, \mathbf{U}_c$ are weight matrices that are being learned.

$$
\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{h}_{t-1} + \mathbf{U}_f\mathbf{x}_t), \qquad (1)\\
\mathbf{i}_t &= \sigma(\mathbf{W}_i\mathbf{h}_{t-1} + \mathbf{U}_i\mathbf{x}_t),\\
\mathbf{o}_t &= \sigma(\mathbf{W}_o\mathbf{h}_{t-1} + \mathbf{U}_o\mathbf{x}_t),\\
\hat{\mathbf{c}} &= \tanh(\mathbf{W}_c\mathbf{h}_{t-1} + \mathbf{U}_c\mathbf{x}_t),\\
\mathbf{c}_t &= \mathbf{i}_t * \hat{\mathbf{c}} + \mathbf{f}_t * \mathbf{c}_{t-1},\\
\mathbf{h}_t &= \mathbf{o}_t * \tanh(\mathbf{c}_t),
\end{aligned}
$$

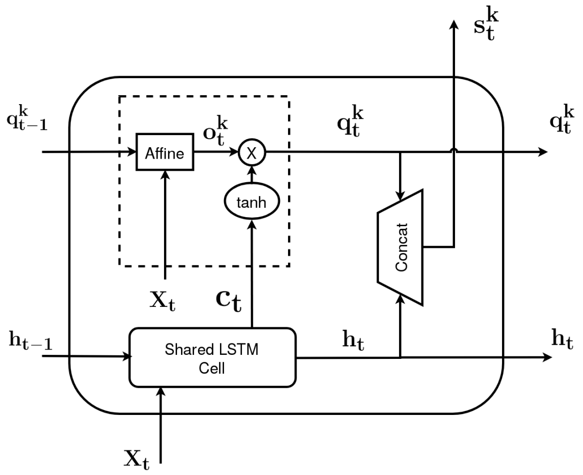

Figure 2: Structure of an SC-LSTM cell. The dashed delimited box depicts the task-specific cell. For clarity reasons, we only show one such cell, while in practice there is one task-specific cell for each task.

3.2 SC-LSTM Cell

The overall structure of our cell is depicted in Figure 2. On top of a standard LSTM cell, we add one cell per task with its own parameters. The standard LSTM cell is thus shared among the K task-specific cells, therefore the name we choose for this new cell, which stands for Shared-Cell LSTM. Task-specific cells are each parametrized by an output gate $\mathbf{o}_t^k$ which learns to select the useful information from the shared memory cell $\mathbf{c}_t$ and outputs $\mathbf{q}_t^k$. This is formally described in Equation 2, where $\mathbf{W}_k$ and $\mathbf{U}_k$ are two extra weight matrices that parametrize the kth task, and $\mathbf{q}_t^k$ has to be understood as a task-specific hidden representation since parameters of kth task-specific cell are only updated by supervision from task k.

$$
\begin{aligned}
\mathbf{o}_t^k &= \sigma(\mathbf{W}_k\mathbf{q}_{t-1}^k + \mathbf{U}_k\mathbf{x}_t^k) \qquad (2)\\
\mathbf{q}_t^k &= \mathbf{o}_t^k * \tanh(\mathbf{c}_t)
\end{aligned}
$$

In order to make use of both shared and task-specific information (Kim et al., 2016; Peng et al., 2017; Hershcovich et al., 2018), for the kth task, we concatenate the output of the shared cell $\mathbf{h}_t$ and of the task-specific one $\mathbf{q}_t^k$ to generate the final latent representation, as noted in Equation 3, where $\oplus$ is the concatenation operation. In practice, we stack SC-LSTM layers. The top-most layer uses $\mathbf{s}_t^k$ as a representation of the current input, while cells in lower layers pass the current shared hidden state $\mathbf{h}_t$ to the upper SC-LSTM cell. The use of s in the topmost layer only is arbitrary and should be investigated.

$$\mathbf{s}_t^k = \mathbf{q}_t^k \oplus \mathbf{h}_t \qquad (3)$$

3.3 Training procedure

The training material available may be gathered from different datasets S_k which means that input sequences differ from one task to another. Therefore in practice, we build K dataloaders, and the training is achieved in a stochastic manner by looping over the tasks at each epoch, as detailed in Algorithm 1.

The loss function ϵ we minimize is a linear combination of task-specific loss functions, where the weighting coefficients (λ^k in Equation 4) are hyper-parameters.

$$\epsilon = \sum_{k=1}^{K} \lambda^k L(\mathbf{y}^k, \hat{\mathbf{y}}^k) \qquad (4)$$

Algorithm 1 Stochastic training procedure

```
1: procedure TRAINING
2:     for each epoch do
3:         Randomly choose a task $k$.
4:         Randomly choose a training example not yet considered in the dataset $S_k$
5:         Update the parameters for this task by taking a gradient step based on this example.
6:         Go to step 3 until using all examples.
```

We seek to minimize cross-entropy of the predicted and true distributions, therefore task-specific loss functions are defined according to Equation 5.

$$L(\mathbf{y}^k, \hat{\mathbf{y}}^k) = -\sum_{i=1}^{n_k}\sum_{j=1}^{m_i} \mathbf{y}_{i,j}^k \log \hat{\mathbf{y}}_{i,j}^k \qquad (5)$$

where $\hat{\mathbf{y}}_{i,j}^k$ is the prediction of the kth softmax classifier parametrized by a projection matrix $\mathbf{W}^k$ and a bias vector $\mathbf{b}^k$:

$$\hat{\mathbf{y}}_{i,j}^k = softmax(\mathbf{W}^k \mathbf{s}_{\mathbf{i,j}}^{\mathbf{k}} + \mathbf{b}^k) \qquad (6)$$

where $\mathbf{s}_{\mathbf{i,j}}^{\mathbf{k}}$ stands for jth token of the ith sequence for task k. In an SC-LSTM cell with k tasks, we will add k matrices and k bias vectors, compared with a vanilla LSTM cell, which increases the capacity of the resulting model. This extra calculation is conducted in parallel with the original LSTM computations.

4 Experimental Setting

4.1 Benchmarks

We test several baseline systems and our SC-LSTM model on three well-established sequence labeling benchmarks: CoNLL2003 (Tjong Kim Sang and De Meulder, 2003) for named-entity recognition, CoNLL2000 (Tjong Kim Sang and Buchholz, 2000) for chunking, and the more recent Universal Dependency dataset (Nivre et al., 2016) for part-of-speech tagging,[2] and on which recent MTL investigations have been conducted (Alonso and Plank, 2017; Changpinyo et al., 2018).

We conformed to the pre-defined splits into train/dev/test except for chunking that does not contain a validation set. For this dataset, following recent works (Peters et al., 2017; Liu et al., 2017), sections 15-18 of the Wall Street Journal are used for training, and we randomly sampled 1000 sentences in the training set as the development set. Section 20 is used for tests. Table 1 presents the main characteristics of the training, development and test sets we used.

[2]We used UD English POS v1.3.

	Train	Dev	Test
NER			**#tags** = 4
#sentences	14,987	3,644	3,486
#tokens	205k	52k	47k
#entities	23,523	5,943	5,654
Chunking			**#tags** = 10
#sentencies	8,936	-	2,012
#tokens	212k	-	47k
#chunks	107k	-	24k
POS			**#tags** = 17
#sentences	12,543	2,002	2,077
#tokens	204k	25k	25

Table 1: Main characteristics of the datasets used.

4.2 Architectural Choices

We used bidirectional LSTM or SC-LSTM as our encoders for the vector representation of words. Bidirectional LSTM can capture global information of the whole sentence, thanks to the encoding of a sequence in a recurrent way. The vector representation of words consists of three parts: word embedding, character-level representation and contextual representation:

- previous works have proven that pre-trained word embeddings like Word2vec (Mikolov et al., 2013), SENNA (Collobert et al., 2011), or Glove (Pennington et al., 2014) have a positive impact on sequence labeling tasks. We used Glove embeddings[3] of dimension 100, that are fine-tuned during training;

- character-level information has been proven useful for three sequence labeling tasks (Santos and Zadrozny, 2014; Ma and Hovy, 2016; Lample et al., 2016) and Some works further show its effectiveness (Reimers and Gurevych, 2017; Yang et al., 2018). In order to encode character sequences, we used a CNN. The character embedding look-up table is initialized by randomly sampling from

[3]The embedding file `glove6B` are available `https://nlp.stanford.edu/projects/glove/`

the uniform distribution in the range [-0.1, 0.1];

- the third part of the input vector is contextual embedding. Most of the recent works found that contextualized features such as ELMo or BERT (Peters et al., 2018; Devlin et al., 2018) can greatly boost performance. We incorporate ELMo into our input vector by using the ELMo implementation of Gardner et al. (2018).

Conditional random field (CRF) classifiers can consider the dependency of output tags and has been proven useful for tasks like NER or chunking. We consider CRF layers in our models.

To compare the effectiveness of three MTL models, we first test our SC-LSTM and other vanilla LSTM based models without CNN based character-level information extractor and contextual embeddings. In this case, the input will be a concatenation of word embedding and capitalization features[4]. To compare with the state-of-art models, we further implemented three more variants: SC-LSTM-CNN-CRF which makes use of CNN-based character level features with a CRF layer on top, very similar to (Ma and Hovy, 2016), and SC-LSTM-LM-CNN, a variant which considers contextualized word embeddings as in (Peters et al., 2018).

4.3 Implementation

For SC-LSTM-LM-CNN, we used mainly the configuration advocated in (Peters et al., 2018) except for the hidden size of SC-LSTM, which we report in Table 2. We further weighted the NER task in the objective function of Equation 4 to 3 (λ_{NER}), the weights of the other tasks where set to 1.[5] We trained this model SC-LSTM-LM-CNN using the Adam optimizer (Kingma and Ba, 2014) with default setting. Such a model spends typically less than 30 epochs to converge. We choose the mini-batch size of 10 and used the gradient clipping threshold 5. All models are implemented using the Pytorch library (Paszke et al., 2017).

Several tagging schemas have been proposed for conducting chunking and NER tasks. We used the most common BIO one in this work.

[4]Eight features encode capitalization patterns, such as AllUpper, InitialUpper, etc.

[5]We found the loss of the NER task to be around 1/3 of the loss of the chunking and POS tasks.

Layers	Parameters	
Char-level Embedding	dimension	16
Character-CNN	kernel size	3
	padding	1
	stride	1
	channel	128
Word Embedding	dimension	100
SC-LSTM	hidden size	256∗3
	layer	3
ELMo	dimension	1024
Dropout	rate	0.5

Table 2: Hyper-parameters used for training the SC-LSTM-LM-CNN model.

5 Experiments

In this section, we start by comparing MTL approaches based on LSTM and SC-LSTM cells. We then report the performance of variants of our approach we implemented and compare it to state-of-the-art models.

5.1 biLSTM versus biSC-LSTM

We compare bidirectional LSTM (STL), SC-LSTM, and baseline MTL models. The results are shown in Table 3. Training Bi-LSTM models on each task separately (STL) was first conducted as a point of comparison (line 1).

For LSTM-s and LSTM-d, we regard one task as the main task and the others as auxiliary tasks; a setting consistent with Søgaard and Goldberg (2016). Because LSTM-s and LSTM-d always fail to achieve stable and competitive results on the three tasks we considered at the same time, we report the best performance we could obtain for each task (line 2 and 3) specifically. On the contrary, our SC-LSTM model is trained once jointly on all tasks, and only one model is being tested in the end, which is much easier and more realistic of a real deployment (line 4).

The results show that our SC-LSTM model improves the performance of the three tasks simultaneously compared with LSTM (STL), and outperforms the other two MTL methods. By joint learning three tasks, both LSTM-s and LSTM-d can boost the chunking task significantly, but both fail to improve NER and POS tasks. This is consistent with observations made in (Collobert et al., 2011; Søgaard and Goldberg, 2016). We also observe that our SC-LSTM model also benefits the chunking task the most. We will analyze this further in Section 6.

	POS	chunking	NER
LSTM (STL)	95.46	94.44	89.39
LSTM-s	95.45	95.12	89.35
LSTM-d	95.44	95.24	89.37
SC-LSTM	**95.51**	**96.04**	**89.96**

Table 3: Results of models being trained in STL or MTL mode. For all MTL models, we report the best performance via a small grid search over combinations of the hidden size [100, 200, 300, 400] and the number of layers [1, 2, 3]. The best performance of each MTL model was obtained with hidden size 300 and 3 layers.

5.2 SC-LSTM versus state-of-the-art

To further demonstrate the effectiveness of our SC-LSTM model, we compared different variants with state-of-the-art approaches, that we classify into three broad categories:

- **Single sequence labeling** where models are trained without the supervision of other tasks. Specifically, we compare our results to the LSTM-CRF model of Lample et al. (2016) and the LSTM-CNN-CRF of Ma and Hovy (2016), since those are state-of-the-art singly tasked sequence labelers.
- **Multi-tasked sequence labelers** where models leverage the supervision of other tasks. We compare our model with the representative approaches of Luo et al. (2015); Søgaard and Goldberg (2016); Collobert and Weston (2008); Collobert et al. (2011).
- **Models with language model.** Recently, several studies in using contextualized word embeddings achieved great success in a number of tasks. Some recent studies (Peters et al., 2017; Rei, 2017; Peters et al., 2018; Devlin et al., 2018) are particularly considered.

It is worth noting that we did not engineer tasks specific features or integrate external ressources such as gazetteers in our variants.

5.2.1 NER

Results for the CoNLL 2003 dataset are reported in Table 4. We observe that our SC-LSTM-LM-CNN model outperforms all approaches but Devlin et al. (2018) and Akbik et al. (2018). The latter work is using the development set as training

NER	F1-score
Collobert et al. (2011)	89.59
Chiu and Nichols (2015) ♣	91.62
Huang et al. (2015)	88.83
Luo et al. (2015)	91.20
Ma and Hovy (2016)	91.21
Lample et al. (2016)	90.94
Shen et al. (2017)	90.89
Yang et al. (2017)	91.20
Rei (2017)	86.26
Liu et al. (2017)	91.71
Peters et al. (2017)	91.93
Zhang et al. (2018)	91.2
Liu et al. (2018)	91.95
Peters et al. (2018)	92.22
Clark et al. (2018)	92.60
Akbik et al. (2018) ♣	**93.09**
Devlin et al. (2018)	92.80
SC-LSTM	89.96
SC-LSTM-CNN-CRF	91.37
SC-LSTM-LM-CNN	92.60

Table 4: F1-score on the CoNLL03 NER dataset. Models with ♣ use both train and dev splits for training.

material, which avoids a direct comparison. The former model (BERT) is achieving great success by leveraging a huge amount of unannotated data as well as a lot of computation resources we could not afford in this study. We are however pleased that our model is leveraging contextual embeddings with 0.38 absolute F1 improvement over the results of Peters et al. (2018). We leave as future work to investigate whether our MTL model can leverage BERT embeddings.

5.2.2 Chunking

We compared a number of models on the CoNLL2000 chunking dataset. A few of them (Ma and Hovy, 2016; Lample et al., 2016; Peters et al., 2018) where not tested on this benchmark, and we reimplemented them. We also trained the companion toolkits of those models, but (as detailed in the next section) got slightly lower results for some reasons. Table 5 reports the performance of the many approaches we tested.

We observe that our SC-LSTM-LM-CNN architecture achieves a new state-of-the-art F1 score, with over 1 absolute point over the competitive approach of Peters et al. (2017), and an improvement of 0.4% over the current state-of-the-art method of Clark et al. (2018).

Chunking	F1-score
Collobert et al. (2011)	94.32
Huang et al. (2015)	94.13
Ma and Hovy (2016)	94.81†
Lample et al. (2016)	94.68†
Hashimoto et al. (2016)	95.77
Søgaard and Goldberg (2016)	95.56
Shen et al. (2017)	93.88
Yang et al. (2017)	94.66
Liu et al. (2017)	95.96
Peters et al. (2017)	96.37
Liu et al. (2018)	96.13
Peters et al. (2018)	96.92†
Clark et al. (2018)	97.00
Akbik et al. (2018)	96.72
SC-LSTM	96.04
SC-LSTM-CNN-CRF	96.41
SC-LSTM-LM-CNN	**97.40**

Table 5: F1-score on the CoNLL00 chunking dataset. Configurations with a † sign are approaches we reimplemented.

5.2.3 POS

We conducted experiments on the Universal Dependency POS English dataset and present the results in Table 6. The only study we found that reports results on the UD v1.3 benchmark we used here is (Bjerva et al., 2016), and we report the results they published. For Liu et al. (2017), Peters et al. (2018) we used the available companion toolkits[6] that we trained ourself with the default settings. We re-implemented the other approaches.

Again, we observe that SC-LSTM-LM-CNN outperforms all other approaches we tested. The absolute improvement in F1 score over the current state-of-the-art of Peters et al. (2018) is 0.21%.

In order to further validate our implementations, we also ran the toolkits of Ma and Hovy (2016) and Lample et al. (2016) and obtained slightly lower results[7].

[6]https://github.com/LiyuanLucasLiu/LM-LSTM-CRF and https://github.com/allenai/allennlp

[7]We obtained 95.7% for the toolkit we took from https://github.com/XuezheMax/NeuroNLP2 and 95.6% for the one at https://github.com/glample/tagger.

POS	Accuracy
Collobert et al. (2011)	95.41†
Huang et al. (2015)	95.63†
Ma and Hovy (2016)	95.80†
Lample et al. (2016)	95.78†
Bjerva et al. (2016)	95.67
Liu et al. (2017)	95.95†
Peters et al. (2018)	96.62 †
SC-LSTM	95.51
SC-LSTM-CNN-CRF	95.83
SC-LSTM-LM-CNN	**96.83**

Table 6: Accuracy on the Universal Dependency English POS dataset. Configurations with a † sign are approaches we reproduced.

6 Analysis

We conducted a number of investigations in order to understand better why our multi-task learning model is effective.

6.1 Convergence Analysis

We report in Figure 3 the convergence of different MTL models on the development set. To obtain those curves, we collected the F1-score on the NER and chunking tasks as well as the accuracy of the POS task, and averaged them after each epoch.

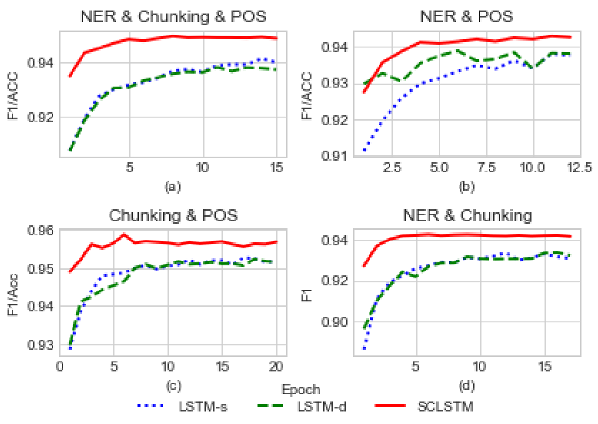

Figure 3: The change of F1-score/Accuracy on the development datasets at different epochs for SC-LSTM (red solid line), LSTM-s (blue dotted line) and LSTM-d (green dashed line) baselines. (a) reports results with NER, chunking and POS; (b) results with NER and POS; (c) results with chunking and POS; and (d) results with NER and chunking. All models were trained using the Adam optimizer with the same setting, and the mini-batch size was set to 10.

We clearly see that the SC-LSTM model converges faster than other ones. It achieves higher performance after the first epoch, and after about ten epochs, it shows a smooth performance curve, while LSTM-s and LSTM-d models still fluctuate. This indicates that our model can learn the hidden representation of multiple tasks in a faster and smoother way than the other two methods.

Besides, we observe in Figure 3c and 3d that combinations of tasks involving chunking typically show a smooth training curve, on the contrary to Figure 3b where NER and POS tasks are combined. The fact that the training regimen fluctuates in the latter case for both LSTM-s and LSTM-d suggests that conflicts with those two tasks happen during optimisation, which we do not observe for our model. Also, Figure 3a illustrates that combining the three tasks altogether leads to comparably better performance of our model over LSTM-s and LSTM-d.

6.2 Effect of Different Task Combinations

We analyzed which task is benefited or harmed by others under the three MTL settings we considered and present results in Figure 4.

We find that it leads to better performance for all MTL models by jointly learning chunking with NER or POS(see in Figure 4b). This is in particular the case of our SC-LSTM model which records the largest gain, especially when all tasks are being trained on.

Figure 4c shows results obtained on POS. Only our SC-LSTM model achieves a meaningful improvement. We however observe that the NER task tends to hurt the performance of POS, since in most cases, the performance of POS+NER is lower than the one obtained with POS+chunking.

For NER, Figure 4 shows that POS hurts LSTM-s and LSTM-d models, while the chunking task is beneficial for all MTL methods.

Clearly, the combination of different tasks has a different effect on the final performance of each task. The chunking task seems compatible with NER and POS tasks, and it boosts the other two tasks in all three MTL settings, which is consistent with the results of Changpinyo et al. (2018). Directly jointly training on POS and NER datasets tends to reduce the performance in LSTM-s and LSTM-d, which is also consistent with the conclusion in (Changpinyo et al., 2018). In conclusion, all of the results show that our SC-LSTM model is effective at capturing the mutual benefits of all combined tasks. Since it performs consistently better in various settings, we believe our model to be more robust.

7 Related Work

There are many works that use extra knowledge to improve the performance of sequence labeling tasks.

Many works have focussed on jointly learning two tasks, often with one being considered as the main task, the other being the auxiliary one (Søgaard and Goldberg, 2016; Bjerva et al., 2016; Alonso and Plank, 2017). For instance, chunking, combinatory categorical grammar supertagging, NER, super senses (SemCor), or multiword expression + supersense will be taken as the main task, while POS is the auxiliary task in (Søgaard and Goldberg, 2016). Exceptions to this line of work include (Collobert et al., 2011) that evaluates four tasks: POS, chunking, NER and semantic role labeling; (Kiperwasser and Ballesteros, 2018) that considers a machine translation task with POS and dependency parsing. And Niehues and Cho (2017) considers machine translation with POS and NER tasks; Zhang and Weiss (2016) show that jointly learning a POS tagger and a dependency parser is effective. Miwa and Bansal (2016) jointly trained models for entity detection and relation extraction in the field of relation extraction.

Other works are also trying to leverage language models to empower the performance of sequence labeling tasks. Notably, Liu et al. (2017) propose a model which uses a neural language model to learn character-level knowledge, and conducts sequence labeling to guide the language model towards specific tasks. Others (Peters et al., 2017, 2018; Devlin et al., 2018) use neural language models pre-trained on a large unlabeled corpus to learn context-sensitive representations of words, and leverage this representation into the sequence labeling model.

More related to the present work are studies that analyze the effectiveness of different combinations of sequence labeling tasks in a multi-task learning. In particular, Changpinyo et al. (2018) conduct an investigation on 11 sequence labeling tasks, while Alonso and Plank (2017) evaluate 5 tasks but report signifiant gains for only one task.

8 Conclusion

In this paper, we propose a simple yet powerful LSTM cell that leverage both shared and task-

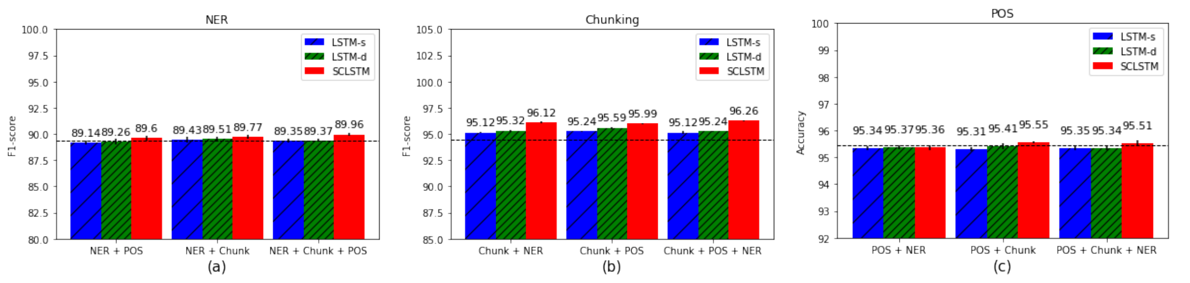

Figure 4: Results of different task groups on each test set: (a) NER, (b) chunking, and (c) POS. Horizontal lines show results for single-task models(NER: 89.39, Chunking: 94.44 and POS: 95.46).

specific parameters in a multi-task setting designed for sequence labelling. We conduct extensive experiments to compare both single-task learning and multi-task learning models. We analyzed the influence of grouping different tasks under various multi-task settings. Experiments demonstrate the effectiveness of our model for sequence labeling tasks. We report new state-of-the-art results on both POS and chunking tasks, and close to state-of-the-art performance on NER, without exploiting external ressources, neither diving into dedicated feature engineering.

Despite those positive outcomes, several issues with multi-task learning for sequence labeling remain open. In particular, we only considered 3 tasks here, and therefore plan to test our approach on more tasks, perhaps understanding better why some tasks are less useful to others. Also, there are several ways we could have used our SC-LSTM cell into our models which we would like to investigate further. In particular, in this work, we only used the specific-task hidden states in the last layer of the model, which can obviously be revisited.

References

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649. Association for Computational Linguistics.

Hector Martinez Alonso and Barbara Plank. 2017. When is multitask learning effective? semantic sequence prediction under varying data conditions. In *EACL 2017-15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 1–10.

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah A Smith. 2016. Many languages, one parser. *Transactions of the Association for Computational Linguistics*, 4:431–444.

Johannes Bjerva, Barbara Plank, and Johan Bos. 2016. Semantic tagging with deep residual networks. *arXiv preprint arXiv:1609.07053*.

Rich Caruana. 1997. Multitask learning. *Machine learning*, 28(1):41–75.

Soravit Changpinyo, Hexiang Hu, and Fei Sha. 2018. Multi-task learning for sequence tagging: An empirical study. *arXiv preprint arXiv:1808.04151*.

Jason PC Chiu and Eric Nichols. 2015. Named entity recognition with bidirectional lstm-cnns. *arXiv preprint arXiv:1511.08308*.

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc Le. 2018. Semi-supervised sequence modeling with cross-view training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1914–1925. Association for Computational Linguistics.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. 2003. Named entity recognition through classifier combination. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 168–171. Association for Computational Linguistics.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. In *ACL workshop for NLP Open Source Software*.

Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. 2016. A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587*.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2018. Multitask parsing across semantic representations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 373–385.

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

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Young-Bum Kim, Karl Stratos, and Ruhi Sarikaya. 2016. Frustratingly easy neural domain adaptation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 387–396.

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

Eliyahu Kiperwasser and Miguel Ballesteros. 2018. Scheduled multi-task learning: From syntax to translation. *arXiv preprint arXiv:1804.08915*.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Liyuan Liu, Xiang Ren, Jingbo Shang, Jian Peng, and Jiawei Han. 2018. Efficient contextualized representation: Language model pruning for sequence labeling. *arXiv preprint arXiv:1804.07827*.

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*.

Gang Luo, Xiaojiang Huang, Chin-Yew Lin, and Zaiqing Nie. 2015. Joint entity recognition and disambiguation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 879–888.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.

Andrew McCallum, Dayne Freitag, and Fernando CN Pereira. 2000. Maximum entropy markov models for information extraction and segmentation. In *Icml*, 2000, pages 591–598.

Andrew McCallum and Wei Li. 2003. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 188–191. Association for Computational Linguistics.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Makoto Miwa and Mohit Bansal. 2016. End-to-end relation extraction using lstms on sequences and tree structures. *arXiv preprint arXiv:1601.00770*.

Jan Niehues and Eunah Cho. 2017. Exploiting linguistic resources for neural machine translation using multi-task learning. *arXiv preprint arXiv:1708.00993*.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. 2016. Universal dependencies v1: A multilingual treebank collection. In *Tenth International Conference on Language Resources and Evaluation (LREC 2016)*.

Robert Östling and Jörg Tiedemann. 2016. Efficient word alignment with markov chain monte carlo. *The Prague Bulletin of Mathematical Linguistics*, 106(1):125–146.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Hao Peng, Sam Thomson, and Noah A Smith. 2017. Deep multitask learning for semantic dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 2037–2048.

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

Matthew Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1756–1765.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

Marek Rei. 2017. Semi-supervised multitask learning for sequence labeling. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 2121–2130.

Nils Reimers and Iryna Gurevych. 2017. Optimal hyperparameters for deep lstm-networks for sequence labeling tasks. *arXiv preprint arXiv:1707.06799*.

Cicero D Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826.

Yanyao Shen, Hyokun Yun, Zachary C Lipton, Yakov Kronrod, and Animashree Anandkumar. 2017. Deep active learning for named entity recognition. *arXiv preprint arXiv:1707.05928*.

Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 231–235.

Erik F Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the conll-2000 shared task: Chunking. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, pages 127–132. Association for Computational Linguistics.

Erik F Tjong Kim Sang and Fien De Meulder. 2003. 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-Volume 4*, pages 142–147. Association for Computational Linguistics.

Jie Yang, Shuailong Liang, and Yue Zhang. 2018. Design challenges and misconceptions in neural sequence labeling. *arXiv preprint arXiv:1806.04470*.

Zhilin Yang, Ruslan Salakhutdinov, and William W Cohen. 2017. Transfer learning for sequence tagging with hierarchical recurrent networks. *arXiv preprint arXiv:1703.06345*.

Yuan Zhang, Hongshen Chen, Yihong Zhao, Qun Liu, and Dawei Yin. 2018. Learning tag dependencies for sequence tagging. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4581–4587. International Joint Conferences on Artificial Intelligence Organization.

Yuan Zhang and David Weiss. 2016. Stack-propagation: Improved representation learning for syntax. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1557–1566.