Learning Sparse Sharing Architectures for Multiple Tasks

Tianxiang Sun,[*] **Yunfan Shao,**[*] **Xiaonan Li, Pengfei Liu, Hang Yan, Xipeng Qiu,**[†] **Xuanjing Huang**
Shanghai Key Laboratory of Intelligent Information Processing, Fudan University
School of Computer Science, Fudan University
{txsun19, yfshao19, pfliu14, hyan19, xpqiu, xjhuang}@fudan.edu.cn, lixiaonan@stu.xidian.edu.cn

Abstract

Most existing deep multi-task learning models are based on parameter sharing, such as hard sharing, hierarchical sharing, and soft sharing. How choosing a suitable sharing mechanism depends on the relations among the tasks, which is not easy since it is difficult to understand the underlying shared factors among these tasks. In this paper, we propose a novel parameter sharing mechanism, named *Sparse Sharing*. Given multiple tasks, our approach automatically finds a sparse sharing structure. We start with an over-parameterized base network, from which each task extracts a subnetwork. The subnetworks of multiple tasks are partially overlapped and trained in parallel. We show that both hard sharing and hierarchical sharing can be formulated as particular instances of the sparse sharing framework. We conduct extensive experiments on three sequence labeling tasks. Compared with single-task models and three typical multi-task learning baselines, our proposed approach achieves consistent improvement while requiring fewer parameters.

Introduction

Deep multi-task learning models have achieved great success in computer vision (**?**; **?**) and natural language processing (**?**; **?**; **?**). Multi-task learning utilizes task-specific knowledge contained in the training signals of related tasks to improve performance and generalization for each task (**?**).

Existing work often focuses on knowledge sharing across tasks, which is typically achieved by parameter sharing. Figure 1-(a)(b)(c) illustrate three common parameter sharing mechanisms: (a) *hard sharing* approaches stack the task-specific layers on the top of the shared layers (**?**; **?**; **?**); (b) *soft sharing* approaches allow each task has separate model and parameters, but each model can access the information inside other models (**?**; **?**; **?**); (c) *hierarchical sharing* approaches put different tasks at different network layers (**?**; **?**).

Despite their success, these approaches have some limitations. Hard sharing architectures force all tasks to share

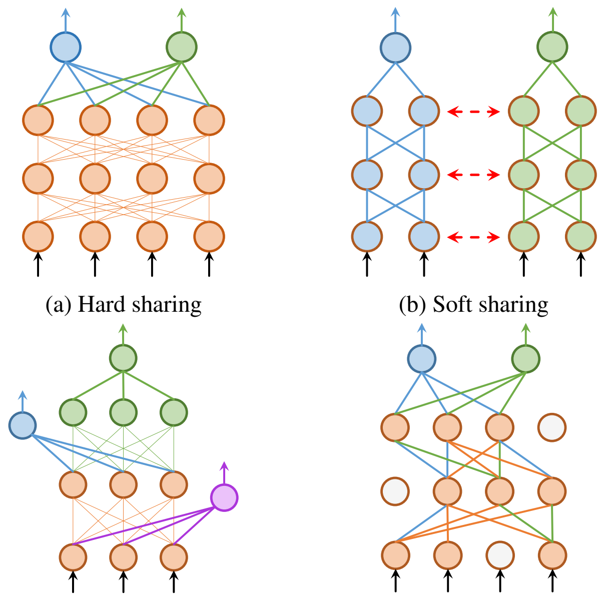

Figure 1: Parameter sharing mechanisms. 🟠 and — represent shared neurons and weights respectively. 🔵, 🟢, 🟣 represent task-specific neurons. —, —, — denote task-specific weights for three different tasks.

the same hidden space, which limits its expressivity and makes it difficult to deal with loosely related tasks. Hierarchical sharing architectures only force part of the model to be shared. Therefore task-specific modules are left with wiggle room to handle heterogeneous tasks. However, designing an effective hierarchy is usually time-consuming. Soft sharing approaches make no assumptions about task relatedness but need to train a model for each task, so are not parameter efficient.

The above limitations motivate us to ask the following question: Does there exist a multi-task sharing architecture that meets the following requirements?

1. It is compatible with a wide range of tasks, regardless of whether the tasks are related or not.

[*]Equal contribution.
[†]Corresponding author: Xipeng Qiu (xpqiu@fudan.edu.cn).

2. It does not depend on manually designing the sharing structure based on characteristic of tasks.
3. It is parameter efficient.

To answer the above question, we propose a novel parameter sharing mechanism, named *Sparse Sharing*. To obtain a sparse sharing architecture, we start with an over-parameterized neural network, named *Base Network*. Then we extract subnetworks from the base network for each task. Closely related tasks tend to extract similar subnets, thus they can use similar parts of weights, while loosely related or unrelated tasks tend to extract subnets that are different in a wide range. During training, each task only updates the weights of its corresponding subnet. In fact, both hard sharing and hierarchical sharing can be formulated into the framework of sparse sharing, as shown in the latter section. Sparse sharing mechanism is depicted in Figure 1(d).

Particularly, we utilize Iterative Magnitude Pruning (IMP) (**?**) to induce sparsity and obtain subnet for each task. Then, these subnets are merged to train all tasks in parallel. Our experiments on sequence labeling tasks demonstrate that sparse sharing meets the requirements mentioned above, and outperforms our single- and multi-task baselines while requiring fewer parameters.

We summarize the our contributions as follows:

- We propose a novel parameter-efficient sharing mechanism, Sparse Sharing, for multi-task learning, in which different tasks are partially shared in parameter-level.
- To induce such a sparse sharing architecture, we propose a simple yet efficient approach based on the Iterative Magnitude Pruning (IMP) (**?**). Given data of multiple tasks, our approach can automatically learn their sparse sharing architecture, without prior knowledge of task relatedness.
- Experiments on three sequence labeling tasks demonstrate that sparse sharing architectures achieve consistent improvement compared with single-task learning and existing sharing mechanisms. Moreover, experimental results show that the sparse sharing mechanism helps alleviate the negative transfer, which is a common phenomenon in multi-task learning.

Deep Multi-Task Learning

Multi-task learning (MTL) utilizes the correlation among tasks to improve performance by training tasks in parallel. In this section, we formulate three widely used neural based multi-task learning mechanisms.

Consider T tasks to be learned, with each t associated with a dataset $\mathcal{D}_t = \{x_n^t, y_n^t\}_{n=1}^{N_t}$ containing N_t samples. Suppose that all tasks have shared layers $\mathcal{E}$ parameterized by $\theta_{\mathcal{E}} = \{\theta_{\mathcal{E},1}, \ldots, \theta_{\mathcal{E},L}\}$, where $\theta_{\mathcal{E},l}$ denotes the parameters in the l-th network layer. Each task t has its own task-specific layers $\mathcal{F}^t$ parameterized by $\theta_{\mathcal{F}}^t$. Given parameters $\theta = (\theta_{\mathcal{E}}, \theta_{\mathcal{F}}^1, \ldots, \theta_{\mathcal{F}}^T)$ and sample x_n^t of task t as input, the multi-task network predicts

$$\hat{y}_n^t = \mathcal{F}^t(\mathcal{E}(x_n^t; \theta_{\mathcal{E}}); \theta_{\mathcal{F}}^t). \tag{1}$$

The parameters of the multi-task model are optimized during joint training with a loss

$$\mathcal{L}(\theta) = \sum_{t=1}^{T} \lambda_t \sum_{n=1}^{N_t} \mathcal{L}_t(\hat{y}_n^t, y_n^t), \tag{2}$$

where $\mathcal{L}_t(\cdot)$ and λ_t are the loss and its weight of task t respectively. In practice, λ_t is often considered as a hyper-parameter to be tuned, but can also be treated as a learnable parameter (**?**).

Let $\theta_{\mathcal{E}(1:l)} = \{\theta_{\mathcal{E},1}, \ldots, \theta_{\mathcal{E},l}\}$, then the prediction of task t in hierarchical sharing architecture is

$$\hat{y}_n^t = \mathcal{F}^t(\mathcal{E}(x_n^t; \theta_{\mathcal{E}(1:l)}); \theta_{\mathcal{F}}^t), \tag{3}$$

if the supervision of task t is put on the l-th network layer.

As shown in Eq. (1), task-specific layers can only extract information from the output of the shared layers $\mathcal{E}$, which forces representation of all tasks to be embedded into the same hidden space. In this way, it is hard to learn shared feature for heterogeneous tasks. Hierarchical sharing, as shown in Eq. (3), leaves task-specific modules more space to model heterogeneous tasks by putting supervision at the different layers. However, this does not fundamentally solve the problem. In addition, hierarchical sharing structures are usually hand-crafted, which heavily depends on skills and insights of experts.

For soft sharing models, each task has its own hidden layers as the same as single-task learning. However, during training, each model can access the representations (or other information) produced by the models of other tasks. Obviously, soft sharing is not parameter-efficient because it assigns each task a model.

Learning Sparse Sharing Architectures

Based on the discussion above, we explore a new multi-task mechanism named *Sparse Sharing*. The architecture of sparse sharing network can be the same as hard sharing, but the parameters in sparse sharing are partially shared.

Sparse sharing starts with an over-parameterized network $\mathcal{E}$, which we call *Base Network*. To handle heterogeneous tasks, we assign each task a different subnet $\mathcal{E}^t$. Rather than training a model for each task as soft sharing does, we employ a binary mask matrix to select subnet from the base network for each task. Let $M_t \in \{0,1\}^{|\theta_{\mathcal{E}}|}$ be the mask of task t. Then the parameters of subnetwork associated with task t become $M_t \odot \theta_{\mathcal{E}}$, where $\odot$ denotes element-wise multiplication. In this way, each task can obtain its unique representation $\mathcal{E}^t(x) = \mathcal{E}(x; M_t \odot \theta_{\mathcal{E}})$. Like other sharing mechanisms, each task has its own task-specific layers $\mathcal{F}^t$.

Note that when M_t takes certain values, sparse sharing can lead to hard sharing and hierarchical sharing. On the one hand, sparse sharing is equivalent to hard sharing if $M_t = \mathbf{1}$. On the other hand, consider a 2-layer base network with parameters $\theta_{\mathcal{E}} = \{\theta_{\mathcal{E},1}, \theta_{\mathcal{E},2}\}$ and two tasks with $M_1 = \{\mathbf{1}, \mathbf{0}\}$ and $M_2 = \{\mathbf{1}, \mathbf{1}\}$. In this case, the two tasks form a hierarchical sharing architecture.

In particular, our approach can be split into two steps (as shown in Figure 2), that is: (1) generating subnets for each task and (2) training subnets in parallel.

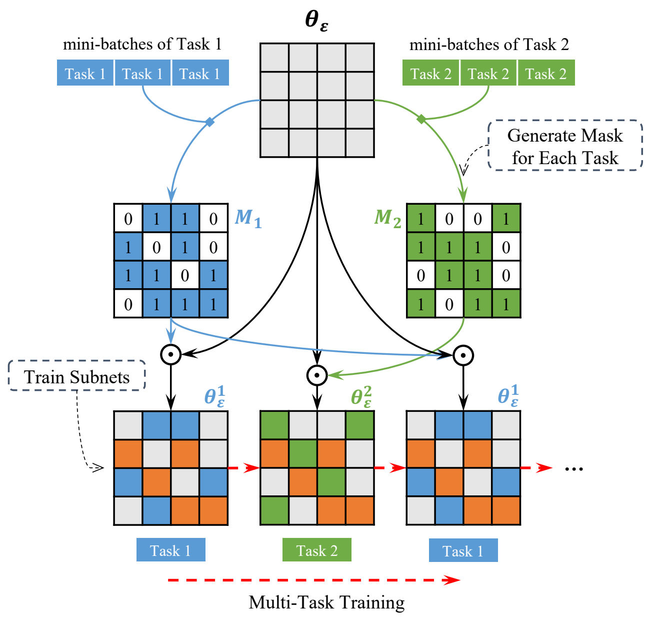

Figure 2: Illustration of our approach to learn sparse sharing architectures. Gray squares are the parameters of a base network. Orange squares represent the shared parameters, while blue and green ones represent private parameters of task 1 and task 2 respectively.

Generating Subnets for Each Task

The base model should be over-parameterized so that its hypothesis space is large enough to contain solutions for multiple tasks simultaneously. From the over-parameterized base network, we extract a subnet for each task, whose structure is associated with a hypothesis subspace that suits the given task. In other words, the inductive bias customized to the task is to some extent embedded into the subnet structure. Ideally, tasks with similar inductive bias should be assigned similar parts of parameters. Therefore, subnets corresponding to closely related tasks should share a large number of parameters while subnets corresponding to loosely related tasks should share few parameters.

Recently, some methods are proposed to automatically find network structures for certain tasks, called *Neural Architecture Search* (**?**; **?**). Instead of these methods, we use a simple yet efficient pruning method to find a structure of subnet for each task.

Our method – Iterative Magnitude Pruning (IMP) – are inspired by the recently proposed Lottery Ticket Hypothesis (**?**). The Lottery Ticket Hypothesis states that dense, randomly-initialized, neural networks contain subnetworks (winning tickets) that – when trained in isolation – can match test accuracy as the original network. The success of winning tickets also demonstrates the importance of network structures for specific tasks.

Following **?** (**?**), we employ IMP to generate subnets for tasks. The structures and parameters of subnets are controlled by mask matrices and base network respectively. With the mask M_t for task t and base network $\mathcal{E}$ at hand, the subnetwork for task t can be represented as $\mathcal{E}^t = \mathcal{E}(x; M_t \odot \theta_\mathcal{E})$.

In particular, we perform IMP for each task independently to obtain subnets. Since these subnets are generated from the same base network, the subnet structure of each task encodes how the base network parameters are used for that task. The process of searching for sparse multi-task architecture is detailed in Algorithm 1. We define the subnet sparsity as $\frac{\|M\|_0}{|\theta|}$.

Algorithm 1 Sparse Sharing Architecture Learning

```
Require: Base Network $\mathcal{E}$; Pruning rate $\alpha$; Minimal sparsity $S$; Datasets for $T$ tasks $\mathcal{D}_1, \cdots, \mathcal{D}_T$, where $\mathcal{D}_t = \{x^t_n, y^t_n\}_{n=1}^{N_t}$.
1: Randomly initialize $\theta_\mathcal{E}$ to $\theta_\mathcal{E}^{(0)}$.
2: for $t = 1 \cdots T$ do
3:     Initialize mask $M_t^z = \mathbf{1}^{|\theta_\mathcal{E}|}$, where $z = 1$.
4:     Train $\mathcal{E}(x; M_t^z \odot \theta_\mathcal{E})$ for $k$ steps with data sampled from $\mathcal{D}_t$, producing network $\mathcal{E}(x; M_t^z \odot \theta_\mathcal{E}^{(k)})$. Let $z \leftarrow z + 1$.
5:     Prune $\alpha$ percent of the remaining parameters with the lowest magnitudes from $\theta_\mathcal{E}^{(k)}$. That is, let $M_t^z[j] = 0$ if $\theta_\mathcal{E}^{(k)}[j]$ is pruned.
6:     If $\frac{\|M_t^z\|_0}{|\theta_\mathcal{E}|} \leq S$, the masks for task $t$ are $\{M_t^i\}_{i=1}^z$.
7:     Otherwise, reset $\theta_\mathcal{E}$ to $\theta_\mathcal{E}^{(0)}$ and repeat steps 4-6 iteratively to learn more sparse subnetwork.
8: end for
9: return $\{M_1^i\}_{i=1}^z, \{M_2^i\}_{i=1}^z, \cdots \{M_T^i\}_{i=1}^z$.
```

Note that since the pruning is iterative, IMP may generate multiple candidate subnets $\{M_t^i\}_{i=1}^z$ as the pruning proceeds. In practice, we only select one subnet for each task to form the sparse sharing architecture. To address this issue, we adopt a simple principle to select a subnet from multiple candidates. That is, picking the subnet that performs best on the development set. If there are multiple best-performing subnets, we take the subnet with the lowest sparsity.

Training Subnets in Parallel

Finally, we train these subnets with multiple tasks in parallel until convergence. Our multi-task training strategy is similar with **?** (**?**), which is in a stochastic manner by looping over the tasks:

1. Select the next task t.
2. Select a random mini-batch for task t.
3. Feed this batch of data into the subnetwork corresponding to task t, i.e. $\mathcal{E}(x; M_t \odot \theta_\mathcal{E})$.
4. Update the subnetwork parameters for this task by taking a gradient step with respect to this mini-batch.
5. Go to 1.

When selecting the next task (step 1), we use proportional sampling strategy (**?**). In proportional sampling, the probability of sampling a task is proportional to the relative size of each dataset compared to the cumulative size of all the datasets.

Although each task only uses its own subnet during training and inference, part of parameters in its subnet are updated by other tasks. In this way, almost every task has shared and private parameters.

Multi-Task Warmup

In practice, we find that randomly initialized base network is sensitive to pruning and data order. To stabilize our approach, we introduce the *Multi-Task Warmup* (MTW) before generating subnets.

More precisely, with randomly initialized base network $\mathcal{E}$ at hand, we first utilize training data of multiple tasks to warm up the base network. Let the initial parameters of $\mathcal{E}$ be $\theta_{\mathcal{E}}^{(0)}$, we train $\mathcal{E}$ for w steps and the warmuped parameters become $\theta_{\mathcal{E}}^{(w)}$. When generating subnets, at the end of each pruning iteration (the 7-th line in Algorithm 1), we reset $\theta_{\mathcal{E}}$ to $\theta_{\mathcal{E}}^{(w)}$ instead of $\theta_{\mathcal{E}}^{(0)}$.

The multi-task warmup (MTW) empirically improves performance in our experiments. Besides, we find that MTW reduces the difference between two subnets generated from the same base network on the same task. Similar phenomenon is obtained by **?** (**?**) in single-task settings.

Experiments

We conduct our experiments on three sequence labeling tasks: Part-of-Speech (POS), Named Entity Recognition (NER) and Chunking.

Datasets

Our experiments are carried out on several widely used sequence labeling datasets, including Penn Treebank(PTB) (**?**), CoNLL-2000 (**?**), CoNLL-2003 (**?**) and OntoNotes 5.0 English (**?**).The datasets are organized in 3 multi-task experiments:

- **Exp1 (CoNLL-2003).** POS, NER, Chunking annotations are simultaneously provided in CoNLL-2003.
- **Exp2 (OntoNotes 5.0).** Also, OntoNotes 5.0 provides annotations for the three tasks. Besides, it contains more data, which comes from multiple domains.
- **Exp3 (PTB + CoNLL-2000 + CoNLL-2003).** Although CoNLL-2003 and OntoNotes provide annotations for multiple tasks, few previous works have reported results for all the three tasks. To compare with the previous multi-task approaches, we follow the multi-task setting of **?** (**?**). The PTB POS, CoNLL-2000 Chunking, CoNLL-2003 NER are integrated to construct our multi-task dataset.

The statistics of the datasets are summarized in Table 1. We use the Wall Street Journal (WSJ) portion of PTB for POS. For OntoNotes, data in `pt` domain is excluded from our experiments due to its lack of NER annotations. The parse bits in OntoNotes are converted to chunking tags as the same as CoNLL-2003. We use the `BIOES` tagging scheme for NER and `BIO2` for Chunking.

Datasets	Train	Dev	Test
PTB	912,344	131,768	129,654
CoNLL-2000	211,727	-	47,377
CoNLL-2003	204,567	51,578	46,666
OntoNotes 5.0	1,903,815	279,495	204,235

Table 1: Number of tokens for each dataset.

Model Settings

In this subsection, we detail our single- and multi-task baselines. For the sake of comparison, we implement our models with the same architecture.

Base Model. We employ a classical CNN-BiLSTM architecture (**?**) as our base model. For each word, a character-level representation is encoded by CNN with character embedding as input. Then the character-level representation and the word embedding are concatenated to feed into the Bi-directional LSTM (**?**) network. We use 100-dimensional GloVe (**?**) trained on 6 billion words from Wikipedia and web text as our initial word embeddings. Dropout (**?**) is applied after embedding layer and before output layer. We use stochastic gradient descent (SGD) to optimize our models. Main hyper-parameters are summarized in Table 2.

Hyper-parameters	
Embedding dimension	100
Convolution width	3
CNN output size	30
LSTM hidden size	200
Learning rate	0.1
Dropout	0.5
Mini-batch size	10

Table 2: Hyper-parameters used in our experiments

Multi-Task Baselines. We compare the proposed sparse sharing mechanism with three existing mechanisms: hard sharing, soft sharing and hierarchical sharing. The base model described above is used as building block to construct our multi-task baselines.

In Exp1, we use 2-layer BiLSTM for hierarchical sharing and 1-layer BiLSTM for other settings. In Exp2, we use 2-layer BiLSTM in all experiments. For the sake of simplicity and focusing the comparison of different shared mechanisms, we just use a simple fully-connected output layer as decoder in Exp1 and Exp2.

In Exp3, for fair comparison with previous methods, we employ CRF (**?**) instead of fully-connected layer as task-specific layer. For hierarchical sharing, following the model setting of **?** (**?**), we put POS task supervision on the inner (1st) layer, put NER and Chunking task supervision on the outer (2nd) layer. For soft sharing, we implement the Cross-stitch network (**?**) as our baseline.

Systems	POS		NER		Chunking		# Params
	Test Acc.	Δ	Test F1	Δ	Test F1	Δ	
Exp1: CoNLL-2003							
Single task	95.09	-	89.36	-	89.92	-	1602k
Single task (subnet)	95.11	+0.02	89.39	+0.03	89.96	+0.04	811k
Hard sharing	95.34	+0.25	88.68	−0.68	90.92	+1.00	534k
Soft sharing	95.16	+0.07	89.35	−0.01	90.71	+0.79	1596k
Hierarchical sharing	95.09	+0.00	89.30	−0.06	90.89	+0.97	1497k
Sparse sharing (ours)	**95.56**	+0.47	**90.35**	+0.99	**91.55**	+1.63	**396k**
Exp2: OntoNotes 5.0							
Single task	97.40	-	82.72	-	95.21	-	4491k
Single task (subnet)	97.42	+0.02	82.94	+0.22	95.28	+0.07	1459k
Hard sharing	97.46	+0.06	82.95	+0.23	95.52	+0.31	1497k
Soft sharing	97.34	−0.06	81.93	−0.79	95.29	+0.08	4485k
Hierarchical sharing	97.22	−0.18	82.81	+0.09	95.53	+0.32	1497k
Sparse sharing (ours)	**97.54**	+0.14	**83.42**	+0.70	**95.56**	+0.35	**662k**

Table 3: Experimental results of Exp1 and Exp2. Δ denotes the improvement compared with single task baselines. We report accuracy(%) for POS and F1 score(%) for NER and Chunking. The best scores are in bold. The performance deterioration due to negative transfer is in gray. The embeddings and output layer are excluded when counting parameters. For the sake of simplicity and focusing the comparison of different shared mechanisms, we just use a simple fully-connected output layer as decoder in Exp1 and Exp2.

Details in Subnets Generation

Note that several hyper-parameters control the generation of subnets, such as the number of multi-task warmup (MTW) steps w, pruning rate α, etc.

	POS	NER	Chunk
Exp1	50.12%	56.23%	44.67%
Exp2	31.62%	25.12%	39.81%
Exp3	56.23%	56.23%	56.23%

Table 4: The sparsity (percent of remaining parameters) of our selected subnets.

In all of our experiments, we use global pruning with $\alpha = 0.1$. Word embeddings are excluded from pruning. Besides, we set the MTW steps $w = 20, 10, 10$ epochs in Exp1, Exp2, Exp3 respectively. As mentioned above, the IMP algorithm produces multiple candidate subnets (with different sparsity) for each task, as shown in Figure 3. It is time-consuming to test all the combinations of these subnets. When selecting subnets, we simply choose the one that performs best on the development set. In fact, we find that a wide range of combinations are effective. The final sparsity of our selected subnets are listed in Table 4.

Main Results

In this subsection, we present our main results on each experiment (Exp1, Exp2 and Exp3). Our proposed sparse sharing mechanism consistently outperforms our single-task and multi-task baselines while requires fewer parameters. Table 3 summarizes the results of Exp1 and Exp2. Table 5 shows experimental results on Exp3.

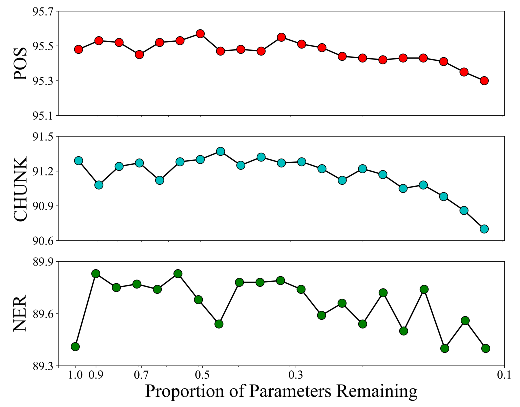

Figure 3: Performance on CoNLL-2003 development set with iteratively pruning. Each point represents a subnet.

In Exp1 and Exp2, though CoNLL-2003 and OntoNotes are well-suited for multi-task experiments, few previous works have reported the results on all the three tasks simultaneously. **?** (**?**) conduct their multi-task experiments on OntoNotes, but they only report the accuracy for each task instead of F1 score. Besides, the tasks they use are different from ours. To be able to compare with other parameter sharing mechanisms, we implement our single- and multi-task baselines using the same architecture (CNN-BiLSTM) and hyper-parameters.

To further compare our work with the various single- and multi-task models, we follow the experimental setting of **?** (**?**) and conduct Exp3. Baselines in Exp3 are implemented in the previous works. For fair comparison, we use CRFs as

Systems	POS	NER	Chunk.
Single Task Models:			
? (**?**)	97.29	89.59	**94.32**
? (**?**)	97.25	89.91	93.67
? (**?**)	**97.30**	**90.08**	93.71
Multi-Task Models:			
Hard sharing†	97.23	90.38	94.32
Meta-MTL-LSTM†	97.45	90.72	95.11
Sparse sharing (ours)	**97.57**	**90.87**	**95.26**

Table 5: Experimental results on Exp3. POS, NER and Chunking tasks come from PTB, CoNLL-2003, CoNLL-2000 respectively. † denotes the model is implemented by **?** (**?**). All listed models are equipped with CRF.

task-specific layers in Exp3.

Where does the benefit come from? To figure out where the benefit of sparse sharing comes from, we evaluate the performance of generated subnets on their corresponding tasks in the single-task setting. As shown in the second row of Table 3, each tasks' subnet does not significantly improve the performance on that task. Thus the benefit comes from shared knowledge in other tasks, rather than pruning.

Avoiding negative transfer. Our experimental results show that multi-task learning does not always yield improvements. Sometimes it even hurts the performance on some tasks, as shown in Table 3, which is called *negative transfer*. Surprisingly, we find that sparse sharing mechanism is helpful to avoid negative transfer.

Analysis and Discussions

In this section, we further analyze the ability of sparse sharing on negative transfer, task interaction and sparsity. At last, an ablation study about multi-task warmup is conducted.

About Negative Transfer

Negative effects usually occurs when there are unrelated tasks. To further analyze the impact of negative transfer for our model, we construct two loosely related tasks. One task is the real CoNLL-2003 NER task. The other task, Position Prediction (PP), is synthetic. The PP task is to predict each token's position in the sentence. The PP task annotation is collected from CoNLL-2003. Thus, the PP task and CoNLL-2003 NER are two loosely related tasks and form a multi-task setting.

We employ a 1-layer BiLSTM with 50-dimensional hidden size as shared module and a fully-connected layer as task-specific layer. Our word embeddings are in 50 dimensions and randomly initialized. Our experimental results are shown in Table 6. Note that Δ is defined as the increase of performance compared with single-task models. The synthetic experiment shows that hard sharing suffers from negative transfer, while sparse sharing does not.

	NER	Δ	PP	Δ
Single task	71.05	-	99.21	-
Hard sharing	61.62	-9.43	99.50	$+0.29$
Sparse sharing	71.46	$+0.41$	99.45	$+0.24$

Table 6: Results of the synthetic experiment. The performance deterioration due to negative transfer is in gray.

About Task Relatedness

Furthermore, we quantitatively discuss the task relatedness and its relationship with multi-task benefit. We provide a novel perspective to analyze task relatedness. We define the Overlap Ratio (OR) among mask matrices as the zero-norm of their intersection divided by the zero-norm of their union:

$$\mathrm{OR}(M_1, M_2, \cdots, M_T) = \frac{\| \cap_{t=1}^{T} M_t \|_0}{\| \cup_{t=1}^{T} M_t \|_0}. \quad (4)$$

Each mask matrix is associated with a subnet. On the one hand, OR reflects the similarity among subnets. On the other hand, OR reflects the degree of sharing among tasks.

We group the three tasks on CoNLL-2003 into pairs and evaluate the performance of sparse and hard sharing on these task pairs. As shown in Table 7, we find that the larger the mask OR, which means the more correlated the two tasks are, the smaller the improvement of sparse sharing compared with hard sharing. This suggests that when tasks are closely related, hard sharing is still an efficient parameter sharing mechanism.

Task Pairs	Mask OR	$\Delta(S^2 - HS)$
POS & NER	0.18	0.4
NER & Chunking	0.20	0.34
POS & Chunking	0.50	0.05

Table 7: Mask Overlap Ratio (OR) and the improvement for sparse sharing (S^2) compared to hard sharing (HS) of tasks on CoNLL-2003. The improvement is calculated using the average performance on the test set.

About Sparsity

In addition, we evaluate various combinations of subnets with different sparsity. For the sake of simplicity, we select subnet for each task with the same sparsity to construct the sparse sharing architecture. Figure 4 plots the average test performance and mask OR with different sparsity combinations. Our evaluation is carried out on CoNLL-2003.

When $M_1 = M_2 = M_3 = \mathbf{1}$, the sparsity of subnets for the three tasks is 100%, and the mask OR takes 1. In this case, sparse sharing is equivalent to hard sharing. With the decrease of sparsity, the mask OR also decreases while the average performance fluctuates.

About Multi-Task Warmup

At last, we conduct an ablation study about multi-task warmup (MTW). The performance achieved by the model

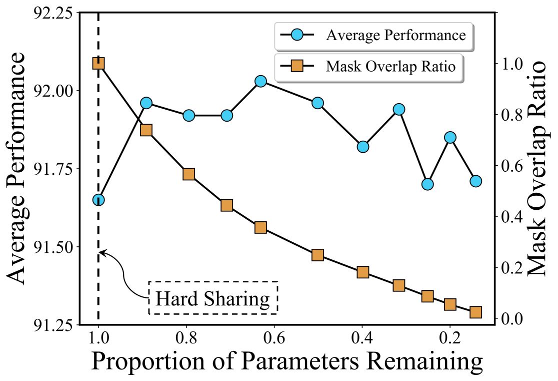

Figure 4: Average test performance and mask OR with different sparsity combinations.

with and without MTW are listed in Table 8. In most instances, the model with MTW achieves better performance.

	POS	NER	Chunking
CoNLL-2003			
Sparse sharing	95.56	90.35	91.55
– MTW	95.36	89.62	91.04
OntoNotes 5.0			
Sparse sharing	97.54	83.42	95.56
– MTW	97.53	81.15	95.48

Table 8: Test accuracy (for POS) and F1 (for NER and Chunking) on CoNLL-2003 and OntoNotes 5.0. MTW: Multi-Task Warmup.

Related Work

There are two lines of research related to our work – deep multi-task learning and sparse neural networks.

Neural based multi-task learning approaches can be roughly grouped into three folds: (1) hard sharing, (2) hierarchical sharing, and (3) soft sharing. Hard sharing finds for a representation that is preferred by as many tasks as possible (**?**). In spite of its simple and parameter-efficient, it is only guaranteed to work for closely related tasks (**?**). Hierarchical sharing approaches put different level of tasks on the different network layer (**?**; **?**), which to some extent, relax the constraint about task relatedness. However, the hierarchy of tasks is usually designed by hand through the skill and insights of experts. Besides, tasks can hurt each other when they are embedded into the same hidden space (**?**). To mitigate negative transfer, soft sharing is proposed and achieves success in computer vision (**?**) and natural language processing (**?**; **?**). In spite of its flexible, soft sharing allows each task to have its separate parameters, thus is parameter inefficient. Different from these work, sparse sharing is a fine-grained parameter sharing strategy that is flexible to handle heterogeneous tasks and parameter efficient.

Our approach is also inspired by the sparsity of networks, especially the Lottery Ticket Hypothesis (**?**). **?** (**?**) finds that a subnet (winning ticket) – that can reach test accuracy comparable to the original network – can be produced through Iterative Magnitude Pruning (IMP). Further, **?** (**?**) introduce late resetting to stabilize the lottery ticket hypothesis at scale. Besides, **?** (**?**) confirms that winning ticket initializations exist in LSTM and NLP tasks. Our experiments also demonstrate that winning tickets do exist in sequence labeling tasks. In addition, it is worth noticing that sparse sharing architecture is also possible to be learned using variational dropout (**?**), l_0 regularization (**?**) or other pruning techniques.

Conclusion

Most existing neural-based multi-task learning models are done with parameter sharing, e.g. hard sharing, soft sharing, hierarchical sharing etc. These sharing mechanisms have some inevitable limitations: (1) hard sharing struggles with heterogeneous tasks, (2) soft sharing is parameter-inefficient, (3) hierarchical sharing depends on manually design. To alleviate the limitations, we propose a novel parameter sharing mechanism, named Sparse Sharing. The parameters in sparse sharing architectures are partially shared across tasks, which makes it flexible to handle loosely related tasks.

To induce such architectures, we propose a simple yet efficient approach that can automatically extract subnets for each task. The obtained subnets are overlapped and trained in parallel. Our experimental results show that sparse sharing architectures achieve consistent improvement while requiring fewer parameters. Besides, our synthetic experiment shows that sparse sharing avoids negative transfer even when tasks are unrelated.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments. This work was supported by the National Key Research and Development Program of China (No. 2018YFC0831103), National Natural Science Foundation of China (No. 61672162), Shanghai Municipal Science and Technology Major Project (No. 2018SHZDZX01) and ZJLab.

Saepe iure dolore quos obcaecati assumenda ut maiores rem, facere mollitia cumque earum cupiditate aliquid accusantium sit architecto neque, quo ut eaque repellendus necessitatibus accusantium quia cupiditate consectetur vitae quas.Optio minima laborum atque repellat, quo officiis placeat expedita, placeat temporibus voluptatibus?Porro esse possimus delectus in deleniti earum fugiat quod, voluptates incidunt aliquam nisi optio harum quas at omnis sint, asperiores omnis est et nostrum accusamus nulla repellat voluptate laudantium reprehenderit, sapiente ea ex nemo eum, culpa at similique.Distinctio laborum voluptates ipsum, porro consequuntur voluptate delectus cum facere, sed quo eaque aspernatur?Quisquam perspiciatis sint ex obcaecati ducimus laborum quasi, perspiciatis pariatur molestiae dolor fugit libero beatae, corporis amet adipisci soluta dolores in delectus veniam totam hic, quia