

Exploring Shared Structures and Hierarchies for Multiple NLP Tasks

Junkun Chen*, Kaiyu Chen*, Xinchi Chen, Xipeng Qiu[†], Xuanjing Huang

Shanghai Key Laboratory of Intelligent Information Processing, Fudan University

School of Computer Science, Fudan University

{jkchen16, kychen15, xinchichen13, xpqiu, xjhuang}@fudan.edu.cn

Abstract

Designing shared neural architecture plays an important role in multi-task learning. The challenge is that finding an optimal sharing scheme relies heavily on the expert knowledge and is not scalable to a large number of diverse tasks. Inspired by the promising work of neural architecture search (NAS), we apply reinforcement learning to automatically find possible shared architecture for multi-task learning. Specifically, we use a controller to select from a set of shareable modules and assemble a task-specific architecture, and repeat the same procedure for other tasks. The controller is trained with reinforcement learning to maximize the expected accuracies for all tasks. We conduct extensive experiments on two types of tasks, text classification and sequence labeling, which demonstrate the benefits of our approach.

Introduction

Multi-task Learning (MTL) is an essential technique to improve the performance of a single task by sharing the experiences of other related tasks. A fundamental aspect of MTL is designing the shared schemes. This is a challenging “meta-task” of itself, often requires expert knowledge of the various NLP tasks at hand.

Figure 1 shows, broadly, the categories of existing MTL schemes:

1. **Hard-sharing:** All tasks share a common set of modules before fanning out to their private “heads”. This is a basic shared strategy and an efficient approach to fight overfitting (Baxter 1997).
2. **Soft-sharing:** In this scheme, modules of each task are *shareable* and can contribute to different tasks. Consequently, the input to a task’s certain module is a weighted sum of the outputs from depending modules of all tasks; the weights can differ across tasks (Misra et al. 2016). This scheme is much more flexible.
3. **Hierarchical-sharing** When the tasks are related but belong to different stages in a pipeline or general data flow graph, the sharing scheme should have a hierarchical structure, where lower level tasks branching off earlier to feed to their private heads (Søgaard and Goldberg 2016). This scheme is very suitable for NLP tasks since most of NLP tasks are related and there is an implicit hierarchy

* Equal contribution

[†] Corresponding Author

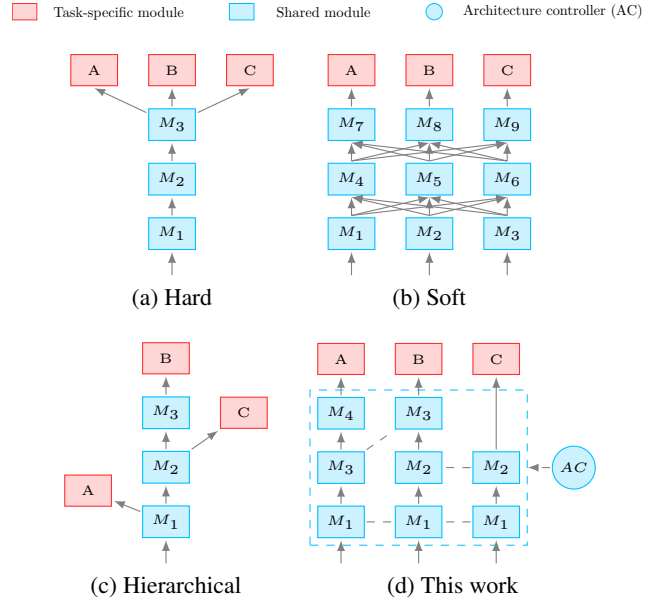


Figure 1: Sharing schemes of multi-task learning. Modules connected by dashed lines in (1d) are identical, selected from a shared module pool.

among their representations. A well-designed hierarchical sharing scheme can boost the performance of all tasks significantly (Hashimoto et al. 2017).

Despite their success, there are still two challenging issues.

The first issue is that judging relatedness of two tasks is not easy, and it is even more difficult of when number of tasks scale. For instance, consider three classification tasks for movie reviews, product reviews, and spam detection. A preferred option is sharing information between movie reviews and product reviews classification, but not spam detection. However, it is hard to explore all configurations when number of tasks become large. An improper sharing scheme design may cause “negative transfer”, a severe problem in MTL and transfer learning. Besides, it suffers from a problem of “architecture engineering”. Once datasets alter, we need to redesign and explore a new sharing scheme. This is taxing even for experts.

The second issue is how to design a hierarchical-sharing

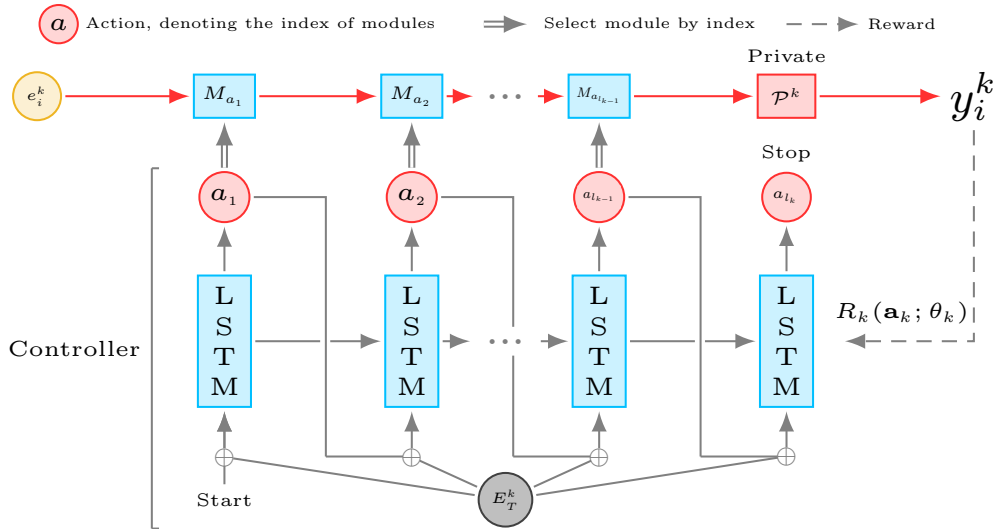


Figure 2: The architecture of proposed model. The blue modules are shared between different tasks. The RNN-based controller consecutively selects $M_{a_1}, M_{a_2}, \dots, M_{a_{l_k}}$ for task k . a_t is the action of step t generated by controller, indicating the index of selected modules. Finally, the controller generates a *Stop* action, terminating the selection process. \mathcal{P}^k denotes to the private module of task k . \oplus is a concatenation operation. The reward of the controller is the performance of the task k under the selected architecture of the shared layers.

scheme for multi-level tasks. For instance, many tasks in NLP are related but not belonging to the same level, such as part-of-speech (POS) tagging, named entity recognition (NER), semantic role labeling (SRL), etc. Thus, the hierarchical-sharing scheme should be adopted. However, it is non-trivial to choose the proper sharing structure manually especially facing a large set of tasks according to the expert knowledge. It also suffers from the problems of “negative transfer” and “architecture engineering”.

To address the above two challenging issues for MTL, in this paper, we learn a controller by reinforcement learning to automatically design the shared neural architecture for a group of tasks, inspired by a recent promising work, neural architecture search (NAS) (Zoph and Le 2016). Specifically, there is a pool of sharable modules, where one or multiple modules can be integrated into an end-to-end task-specific network. As shown in Figure 1d, for each task, the controller chooses one or multiple modules from sharable module pool, stacks them in proper order, and then integrates them into a task-specific network. The controller is trained to maximize the expected accuracies of its designed architectures for all the tasks in MTL. Additionally, we also propose a novel training strategy, namely *simultaneous on-line update strategy*, which accelerates the training procedure.

The benefits of our approach could be summarized as follows.

- Since not all the sharable modules are used in a task-specific network, the related tasks can share overlapping modules, whereas the unrelated tasks keep their private modules.
- The controller can stack the chosen modules in various orders. Therefore its designed architecture can handle the

multi-level tasks. One shared module can posit at low-layer for one high-level task, but at high-layer for another low-level task, therefore a hierarchical-sharing structure is implicitly constructed.

- Overall, our approach provides a flexible solution to the problem of designing shared architecture for MTL and can handle complicated correlation and dependency relationship among multiple tasks.

Model

Usually, a multi-task framework contains shared layers and task-specific layers. The proposed model gives a mechanism for automatically learning the structure of the shared layers for each task. The model contains L shared modules: $\mathcal{M} = \{M_1, M_2, \dots, M_L\}$, and a RNN-based controller will select modules for each task. Figure 2 gives an illustration.

Training Controller with REINFORCE

The structure of the shared layers for task k is selected by a RNN-based controller (a list of actions $\mathbf{a}_k = \{a_1, a_2, \dots, a_{l_k}\}$), which takes the task embedding of $E_T(k)$ of task k as the input. The chosen layers stacked one by one form the shared part of the model for the specific task k . And performance under the chosen structure gives the reward feedback R_k for the controller. Thus, we can use reinforcement learning to train the controller. Concretely, our objective is to find the optimal with policy $\pi_\phi(E_T(k), \mathbf{a}_k)$ by maximizing the expected reward $\mathcal{J}_k(\phi)$:

$$\mathcal{J}_k(\phi) = \mathbb{E}_{\pi_\phi(E_T(k), \mathbf{a}_k)}[R_k(\mathbf{a}_k; \theta_k)]. \quad (1)$$

Since the $R_k(\cdot)$ is non-differentiable, we need to use a policy gradient method to optimize ϕ , thus the derivative of $\mathcal{J}_k(\phi)$ is:

Algorithm 1 Training with Simultaneous On-line Update Strategy

```
1: repeat
2:   for each batch do
3:     for each task  $k$  do
4:       Drawn  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_N \sim \pi_\phi(E_T(k))$  // Sample  $N$  structures with policy  $\pi_\phi(E_T(k))$ 
5:       for each  $\mathbf{a} \in \{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_N\}$  do
6:         Update  $\theta \leftarrow \theta + \alpha \nabla_\theta R_k(\mathbf{a}; \theta)$  // Update the parameters of shared modules
7:       end for
8:       Calculate  $N$  rewards  $\mathcal{R}_k = \{R_k(\mathbf{a}_1), R_k(\mathbf{a}_2), \dots, R_k(\mathbf{a}_N)\}$  //  $R_k$  is calculated by Eq (4)
9:       Normalize rewards  $\bar{R}_k = \text{softmax}(\mathcal{R}_k, \tau)$  // Eq (5)
10:      Update  $\phi \leftarrow \phi + \alpha \nabla_\phi \bar{R}_k$  // Update the parameters of controller
11:    end for
12:  end for
13: until  $\theta$  and  $\phi$  convergence
```

$$\nabla_\phi \mathcal{J}_k(\phi) = \sum_{t=1}^{l_k} \mathbb{E}[\nabla_\phi \log \pi_\phi(E_T(k), a_t) R_k(\mathbf{a}_k; \theta_k)]. \quad (2)$$

In this paper, we use LSTM to model $\pi_\phi(E_T(k), a_t)$, where a_t indicates a selection among L shared modules \mathcal{M} .

Reward Function

The reward is the performance of the predicted architecture on task k . Specifically, given training set $\{x_i^k, y_i^k\}_{i=1}^{N_k}$ of task k , we firstly map the input sentence x_i^k to \mathbf{e}_i^k with word embeddings. Assuming that the predicted actions are $\mathbf{a}_k = \{a_1, a_2, \dots, a_{l_k}\}$ and the selected modules are $\mathcal{M}_{\mathbf{a}_k} = \{M_{a_1}, M_{a_2}, \dots, M_{a_{l_k}}\}$, the reward $R_k(\mathbf{a}_k; \theta)$ is:

$$\mathbf{e}_i^k \xrightarrow{\mathcal{M}_{\mathbf{a}_k}} \mathbf{s}_i^k \xrightarrow{\mathcal{P}^k} p(y_i^k | x_i^k) \quad (3)$$

$$R_k(\mathbf{a}_k; \theta) = \frac{1}{N_k} \sum_{i=1}^{N_k} \log p(y_i^k | x_i^k). \quad (4)$$

In this paper, modules are Bi-LSTM layers.

Training with Simultaneous On-line Update Strategy

Our model is composed of two parts. The learning modules focus on processing textual data for each task given the architecture. The controller aims to find a better structure for learning modules to achieve higher performance.

These two parts are trained together with our on-line update strategy demonstrated in Algorithm 1, which means we simultaneously train these two parts at each batch. It is different from previous work, where the controller updates itself only when the multi-task learning procedure completed. A significant advantage of simultaneous training strategy is its high efficiency compared with previous training strategy.

In implementing on-line update strategy, we have three major components as follows.

Multi-sampling of Architecture Controller As the amplitudes and distributions of rewards of each task are probably distinct, rewards can only be privately used for individual tasks, causing a problem of insufficient rewards. To

solve the problem and help the controller distinguish the performance of different architectures, we sample N times of rewards for each single training instance. These rewards are subsequently used for training the controller after reward normalization.

Reward Normalization Given N rewards $\mathcal{R}_k = \{R_k^{(1)}, R_k^{(2)}, \dots, R_k^{(N)}\}$ for task k , \bar{R}_k could be finally derived by normalizing these rewards with a parameterized softmax function (Norouzi et al. 2016):

$$\bar{R}_k = \frac{\exp(R_k^{(i)} / \tau)}{\sum_{j=1}^N \exp(R_k^{(j)} / \tau)}, \quad (5)$$

where τ is a hyper-parameter that adjusts the variance of \bar{R}_k . This normalization method maps rewards to $[0, 1]$, will stabilize the training process.

Experiment

In this section, we evaluate our model on two types of tasks, text classification, and sequence labeling.

Exp-I: Text Classification

Firstly, we apply our model to a text classification dataset whose tasks are separated by domains. With this dataset, we can check if our model can effectively use shared information across domains.

Data We use a dataset of 16 tasks. The dataset is a combination of two categories. The first category that makes up 14 tasks is product reviews, extracted from the dataset¹ constructed by (Blitzer et al. 2007). Reviews of each task come from different Amazon products domain, such as Books, DVDs, and so on. The second category making up the remaining 2 tasks is movie reviews, specifically IMDB² conducted by (Maas et al. 2011), and MR³ from rotten tomato website, conducted by (Pang and Lee 2005).

¹<https://www.cs.jhu.edu/~mdredze/datasets/sentiment/>

²<https://www.cs.jhu.edu/~mdredze/datasets/sentiment/unprocessed.tar.gz>

³<https://www.cs.cornell.edu/people/pabo/movie-review-data/>

Tasks	Single-Task		Multi-Task				
	LSTM	Bi-LSTM	FS	SSP	PSP	CS	Our
Books	80.3	82.0	84.5	87.0	84.7	84.0	87.3
Electronics	78.5	80.0	86.7	86.5	85.5	85.5	88.3
DVD	78.3	84.0	86.5	86.3	86.0	86.5	86.0
Kitchen	81.7	83.5	86.0	88.7	88.0	87.3	87.5
Apparel	83.0	85.7	85.7	86.3	87.5	87.0	88.7
Camera	87.3	88.3	86.7	88.0	87.7	86.3	90.0
Health	84.5	85.3	89.3	90.0	89.0	90.0	90.3
Music	77.5	78.7	82.5	81.0	81.3	84.0	84.7
Toys	84.0	85.7	88.5	87.7	88.5	87.5	89.7
Video	80.3	83.7	86.7	85.5	84.5	85.7	86.0
Baby	85.7	85.7	88.0	87.7	87.3	88.3	89.0
Magazines	92.3	92.5	91.5	93.0	93.3	91.0	90.3
Software	84.5	86.7	87.0	88.3	89.7	87.5	91.0
Sports	79.3	81.0	85.5	86.5	86.7	87.7	88.7
IMDB	79.5	83.7	82.3	85.3	84.5	82.5	84.7
MR	73.5	73.7	71.0	72.7	72.5	73.3	76.3
Avg	81.9	83.0	85.5	86.3	86.1	85.9	87.4

Table 1: The results of text classification experiment.

The dataset for each task contains 2000 samples, which partitioned randomly into training data, development data and testing data with the proportion of 70%, 10%, and 20% respectively. Each sample has two class labels, either positive or negative.

Baseline Model We compare our proposed model with four multi-task models and one single-task model. These baseline models have the same Bi-LSTM modules with proposed model. They differ only in how we organize these modules.

- **FS-MTL**: All tasks use a fully-shared module and a task-specific classifier.
- **SSP-MTL**: This model is followed stack share-private scheme. There is a shared module that receives embed inputs of all tasks and takes its outputs as inputs for task-specific modules. The outputs of task-specific modules are then fed into the task-specific classifier.
- **PSP-MTL**: This model is followed a parallel share-private scheme. Concatenate the outputs of a fully-shared module with the feature extracted from task-specific modules as inputs to the task-specific classifier.
- **CS-MTL**: Cross Stitch multi-task model proposed by (Misra et al. 2016). It’s a soft-sharing approach, using units to model shared representations as a linear combination.
- **Single Task**: Standard Bi-LSTM classifier that trains separately on each task.

The classifier mentioned above is a linear layer taking the average of the text sequence as inputs.

Hyperparameters The networks are trained with back-propagation, and the gradient-based optimization is performed using the Adam update rule (Kingma and Ba 2014), with an early-stop parameter of 12 epochs.

Domains	Train	Dev	Test
Broadcast Conversation (bc)	173,289	29,957	35,947
Broadcast News (bn)	206,902	25,271	26,424
Magazines (mz)	164,217	15,421	17,874
Newswire (nw)	878,223	147,955	60,756
Pivot Corpus (pc)	297,049	25,206	25,883
Telephone Conversation (tc)	90,403	11,200	10,916
Weblogs (wb)	388,851	49,393	52,225

Table 2: Statistics of tokens for each domain in OntoNote 5.0 dataset.

As for the controller, the size of the RNN’s hidden state is 50. Additionally, an exploration probability of sampling randomly is set to 0.2, and the scaling factor in reward normalization τ is $\frac{1}{30}$. The $N = 20$ modules are available, but the model tends to use less than $N = 20$ modules. The size of task embeddings S is 15.

The word embeddings for all of the models are initialized with the 100d GloVe vectors (6B token version) (Pennington, Socher, and Manning 2014), while the size of hidden state \mathbf{h} in Bi-directional LSTMs is 50. Per-task Fine-tuning during training with shared parameters fixed is used to improve the performance. The mini-batch size is set to 64.

Result We evaluate all these models, and the classifying accuracy is demonstrated in Table 1. Compared with the best baseline model, our model still achieves higher performance by 1.5 percent. With more observation of the final architecture selected by the controller, we find out that the controller groups similar tasks together, showing the ability of clustering, which will be further discussed in Section 5.

Domain	Task	Bi-LSTM	PSP	SSP	CS	Our	Task	Bi-LSTM	PSP	SSP	CS	Our
bc	POS	89.56	94.21	94.67	94.17	94.61	NER	92.15	94.26	94.50	93.03	94.50
	CHUNK	85.60	89.20	89.81	86.24	91.19	SRL	97.58	98.07	98.11	97.80	98.21
bn	POS	92.26	94.67	95.17	92.49	95.28	NER	88.46	93.19	93.11	89.70	93.68
	CHUNK	87.23	90.80	91.06	86.02	91.34	SRL	95.56	97.94	97.71	97.66	97.35
mz	POS	79.77	91.22	92.80	84.61	92.88	NER	89.69	92.09	93.22	91.23	93.37
	CHUNK	83.79	89.85	90.36	81.81	91.04	SRL	97.75	97.52	97.86	97.96	98.16
nw	POS	94.35	95.11	94.80	96.23	95.02	NER	93.27	94.30	93.62	95.48	93.94
	CHUNK	90.51	92.19	91.00	92.25	92.29	SRL	96.45	97.05	96.73	96.83	97.81
wb	POS	89.40	92.69	93.24	90.08	92.71	NER	96.20	96.48	96.51	98.17	96.58
	CHUNK	87.62	91.39	91.56	88.00	92.52	SRL	95.13	95.17	96.17	95.84	96.36
tc	POS	92.75	93.88	94.63	94.24	94.99	NER	95.25	95.32	95.37	97.21	95.95
	CHUNK	88.05	90.41	90.23	87.35	91.38	SRL	98.32	98.80	98.84	98.81	99.05
pt	POS	97.26	97.73	97.54	98.81	97.68	NER	/	/	/	/	/
	CHUNK	93.86	95.33	94.45	95.47	95.91	SRL	96.38	96.35	96.36	96.20	97.42
Avg	POS	90.76	94.22	94.69	92.95	94.74	NER	92.50	94.27	94.39	94.14	94.67
	CHUNK	88.09	91.31	91.21	88.16	92.24	SRL	96.74	97.27	97.40	97.30	97.77

Table 3: Resultant accuracy for the sequence labeling dataset, regarding every combination of (domain, problem) as tasks, accumulating to $7 * 4 - 1 = 27$ separate tasks (accurately 27 tasks, since pt has no NER annotation).

	Unite Domains	Separated Domains
POS	94.23	94.74
CHUNK	91.86	92.24
NER	94.58	94.67
SRL	97.12	97.77
Avg.	94.45	94.85

Table 4: Comparison of results in separated domains and united domains, respectively.

Exp-II: Sequence labeling

Secondly, we evaluate our model on sequence labeling task. It challenges the model to discover a shared architecture with hidden hierarchies to complete tasks that are different in semantic levels and significantly differ from each other. Different from Exp-I, to make the model more suitable for sequence labeling, we use a conditional random field (CRF)(Lafferty, McCallum, and Pereira 2001) as the output layer.

Data We use OntoNotes 5.0 dataset⁴ (Weischedel et al. 2013) for our sequence labeling experiment.

OntoNotes contains several tasks across different domains, which helps us validate our model’s ability to learn shared

⁴<https://catalog.ldc.upenn.edu/ldc2013t19>.

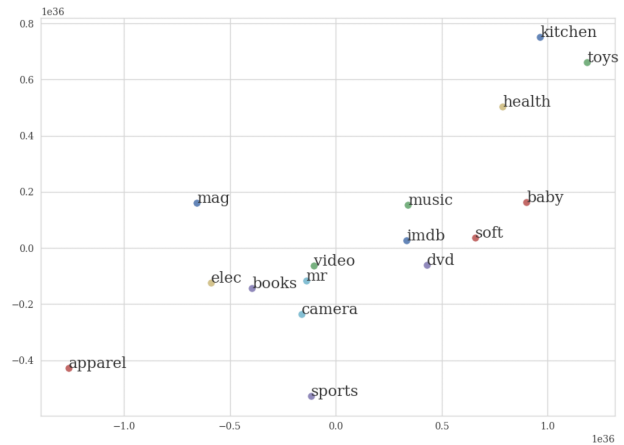
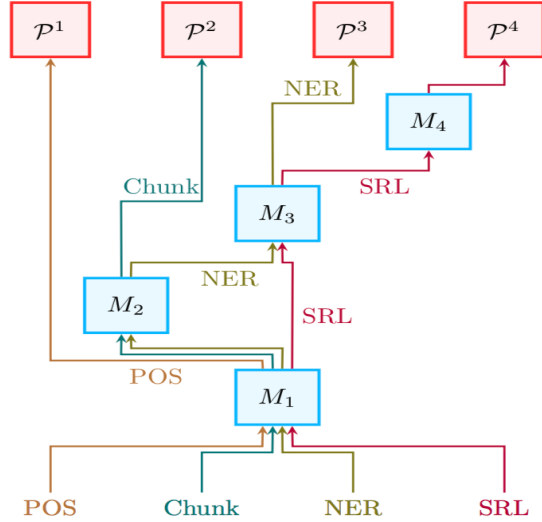


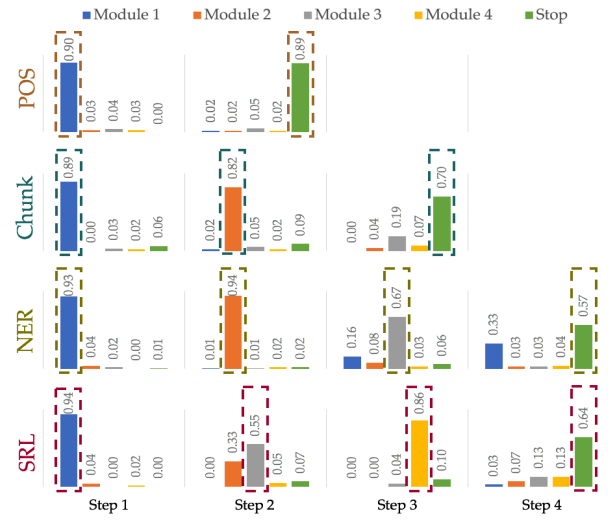
Figure 3: Task embedding for Exp-I (text classification), projected onto two-dimensional space by t-SNE. The full names of domains are displayed in Table 2.

hierarchies. We select four tasks: part-of-speech tagging (POS tagging), chunking (Chunk), named entity recognition (NER), and semantic role labeling(SRL). The statistical details of the sub-datasets for each domain are shown in Table 2. There are four annotations for each sentence, corresponding to four tasks.

In contrast to Exp-I, where the differences among tasks are simply domains, tasks in this experiment differ from each other more significantly, and the relationships of tasks



(a) The hierarchical architecture selected by the controller. Different colors represent different tasks. The lines and arrows show the chosen information flow of tasks. For example, given a sample of Chunking task, the controller sequentially chooses M_1 , M_2 , and $Stop$, forming the path in teal of the picture. The selective modules are originally disordered and are renamed $M_1 \dots M_4$ for more explicit representation.



(b) Probabilities of module selection at each step of the decision process. Probabilities are drawn from controller's module decision process after softmax function. The first four columns in every histogram represent the probability of choosing four different modules. The fifth column means the probability of deciding to $Stop$. For chunking, the decision sequence with the biggest probabilities is $M_1, M_2, Stop$, which corresponds with the path in 4a.

Figure 4: The selected architecture and the corresponding probabilities at every step of the decision process.

are more complicated. For example, solving NER problems can rarely help solving POS tagging task, while POS tagging may somehow provide low-level information for NER problems. There seems to be a hidden hierarchical structure of these tasks, and it challenges our model to find the hidden structure without any prior knowledge.

Result We regard every combination of (domain, problem) such as (bc, NER) as different tasks, accumulating to $7 * 4 - 1 = 27$ tasks. We evaluate our model together with all baseline models mentioned in Exp-I, with a slight change of removing the average pooling classifier to every step in the sequence. Table 3 shows the final accuracy of all these tasks. It is remarkable that our model achieves higher performance over all baseline models. It shows that our model successfully handles the situation of complicated task relationships.

Additionally, we carry out a small experiment of gathering data across domains together, reducing the number of tasks from 27 to 4. According to experiment results demonstrated in Table 4, the performance with united domains is lower than that with separated domains. It shows that our model is capable of leveraging domain-specific knowledge.

Analysis

The high performance of our model suggests that the controller selects proper structures in different situations. To get a more intuitive understanding of how our model works, we explore and visualize the chosen architecture. Three main

characteristics of our generated structure are as follows.

Task Embedding Clustering

In our text classification experiment, the structure generated by our model is capable of clustering tasks from similar domains together. To illustrate our clustering ability, we use t-SNE (Maaten and Hinton 2008) to project our task embedding onto two-dimensional space. The results are in Figure 3.

From the illustration, we find that the task embedding shows the ability to cluster similar domains. For example, the task embedding of 'IMDB' (movie review) lies near 'video', 'music', and 'DVD'. It indicates that our model can distinguish characteristics of different tasks, and eventually finds a proper structure for the problem.

Learned Hierarchical Architecture

As proposed by (Hashimoto et al. 2017), tasks of POS Tagging, Chunking, NER and sentence-level problems form a hierarchical linguistic structure, to which multi-task models should conform. To validate if our model learns a hierarchical architecture, we investigate the selected architecture in the 4-task experiment of sequence labeling. For each task, we visualize the module selection sequence of the biggest probability. The results are shown in Figure 4a.

In the beginning, the controller randomly chooses modules, trying structures that may contain loop paths or even self-loops. Nevertheless, our model eventually comes up with a hierarchical structure, with decision probabilities

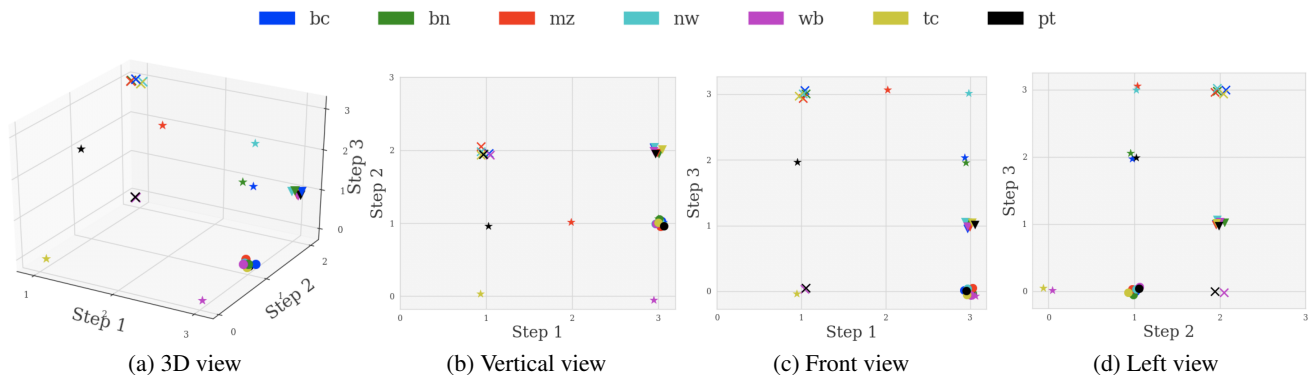


Figure 5: Module selection sequence projected into three-dimensional space. Each point in the space represents a task. The position of it is determined by its module selection sequence. POS Tagging, Chunking, NER and SRL are respectively denoted by ●, ▼, ★, and ×. Points align precisely on discrete positions. Slight shifts are imposed upon points for better visualization.

shown in Figure 4b. It convincingly indicates that our model has learned to generate a superior hierarchical structure.

Module Selection Clustering

Another characteristic of the selected architecture is that similar tasks are clustered to share similar module structures. It can be illustrated if we regard the sequence of chosen modules as coordinates and project it into high-dimensional space. Since only three steps are used in consequence, Figure 5 shows results in three-dimensional space.

There are some interesting patterns in the distributions of different tasks. Firstly, tasks of the same problem are likely to be neighboring. Secondly, high-level tasks like SRL are distributed more sparsely. Additionally, similar domains, for example, tasks in bc(broadcast conversation) have the same positions as tasks in bn(broadcast news). All these patterns found in the distributions show the ability of our model of clustering similar tasks and generating a reasonable structure.

Related Work

Multi-Task Learning has achieved superior performance in many applications (Collobert and Weston 2008; Glorot, Bordes, and Bengio 2011; Liu et al. 2015; Liu, Qiu, and Huang 2016; Duong et al. 2015). However, it is always done with hard or soft sharing of layers in the hope of obtaining a more generative representation shared between different tasks.

Because of restrictions on shared structure, so that only when facing the loosely relevant tasks or datasets, the effect will be good. Søgaard and Goldberg (2016) and Hashimoto et al. (2017) train their model considering linguistic hierarchies between related NLP tasks, by making lower level tasks affect the lower levels of the representation. They show that given a hierarchy of tasks, the effectiveness of multi-tasking learning increases, but they are pre-defined by human knowledge.

Recently, to break through the constraints of the shared structure, Lu et al. (2017) proposes a bottom-up approach that dynamically creates branches of networks during training that promotes grouping of similar tasks. Meyerson and

Miikkulainen (2017) introduced as a method for learning how to apply layers in different ways at different depths for different tasks, while simultaneously learning the layers themselves. However it can only change the order of shared layers, cannot learn the hierarchies.

Other approaches related to ours include meta-learning and neural architecture search(NAS), meta-learning tries to learn a meta-model that generates a model for specific tasks. Most of them are using the meta-network to controls the parameters of the task-specific networks (Ravi and Larochelle 2017; Chen et al. 2018). The NAS is introduced in (Zoph and Le 2016; Baker et al. 2016; Pham et al. 2018), where it’s applied to construct CNN for image classification. And Liu et al. (2017) has introduced a hierarchical genetic representation method into it. However, they are all dealing with single tasks, not the shared architecture search in Multi-Task Learning. Rosenbaum, Klinger, and Riemer (2017)’s work is similar with us, but they have made restrictions on the optional position of modules. And they only optimize on the same type of tasks, like image classification, do not consider the latent hierarchy between tasks that enhance representation.

Conclusion

In this paper, we introduce a novel framework to explore the shared structure for MTL with automatic neural architecture search. Different from the existing MTL models, we use a controller network to select modules from a shared pool for individual tasks. Through this way, we can learn an optimal shared structure for tasks to achieve better performance leverage the implicit relations between them. We evaluate our model on two types of tasks, have both achieved superior results. Besides, we have verified that our model can learn a reasonable hierarchy between tasks without human expertise.

References

- [2016] Baker, B.; Gupta, O.; Naik, N.; and Raskar, R. 2016. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*.

- [1997] Baxter, J. 1997. A bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine learning* 28(1):7–39.
- [2007] Blitzer, J.; Dredze, M.; Pereira, F.; et al. 2007. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL*, volume 7, 440–447.
- [2018] Chen, J.; Qiu, X.; Liu, P.; and Huang, X. 2018. Meta multi-task learning for sequence modeling. In *Proceeding of AAAI*.
- [2008] Collobert, R., and Weston, J. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of ICML*.
- [2015] Duong, L.; Cohn, T.; Bird, S.; and Cook, P. 2015. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, 845–850.
- [2011] Glorot, X.; Bordes, A.; and Bengio, Y. 2011. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *ICML-11*, 513–520.
- [2017] Hashimoto, K.; Tsuruoka, Y.; Socher, R.; et al. 2017. A joint many-task model: Growing a neural network for multiple nlp tasks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 1923–1933.
- [2014] Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- [2001] Lafferty, J. D.; McCallum, A.; and Pereira, F. C. N. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML-2001*.
- [2015] Liu, X.; Gao, J.; He, X.; Deng, L.; Duh, K.; and Wang, Y.-Y. 2015. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. In *NAACL*.
- [2017] Liu, H.; Simonyan, K.; Vinyals, O.; Fernando, C.; and Kavukcuoglu, K. 2017. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*.
- [2016] Liu, P.; Qiu, X.; and Huang, X. 2016. Recurrent neural network for text classification with multi-task learning. In *Proceedings of IJCAI*, 2873–2879.
- [2017] Lu, Y.; Kumar, A.; Zhai, S.; Cheng, Y.; Javidi, T.; and Feris, R. 2017. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5334–5343.
- [2011] Maas, A. L.; Daly, R. E.; Pham, P. T.; Huang, D.; Ng, A. Y.; and Potts, C. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the ACL*, 142–150.
- [2008] Maaten, L. v. d., and Hinton, G. 2008. Visualizing data using t-sne. *Journal of machine learning research* 9(Nov):2579–2605.
- [2017] Meyerson, E., and Miikkulainen, R. 2017. Beyond shared hierarchies: Deep multitask learning through soft layer ordering. *arXiv preprint arXiv:1711.00108*.
- [2016] Misra, I.; Shrivastava, A.; Gupta, A.; and Hebert, M. 2016. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3994–4003.
- [2016] Norouzi, M.; Bengio, S.; Jaitly, N.; Schuster, M.; Wu, Y.; Schuurmans, D.; et al. 2016. Reward augmented maximum likelihood for neural structured prediction. In *Advances In Neural Information Processing Systems*, 1723–1731.
- [2005] Pang, B., and Lee, L. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the ACL*, 115–124.
- [2014] Pennington, J.; Socher, R.; and Manning, C. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.
- [2018] Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*.
- [2017] Ravi, S., and Larochelle, H. 2017. Optimization as a model for few-shot learning.
- [2017] Rosenbaum, C.; Klinger, T.; and Riemer, M. 2017. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*.
- [2016] Søgaard, A., and Goldberg, Y. 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, 231–235.
- [2013] Weischedel, R.; Palmer, M.; Marcus, M.; Hovy, E.; Pradhan, S.; Ramshaw, L.; Xue, N.; Taylor, A.; Kaufman, J.; Franchini, M.; et al. 2013. Ontonotes release 5.0 ldc2013t19. *Linguistic Data Consortium, Philadelphia, PA*.
- [2016] Zoph, B., and Le, Q. V. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.