Multi-Task Recurrent Modular Networks

Dongkuan Xu[1], Wei Cheng[2], Xin Dong[3], Bo Zong[2], Wenchao Yu[2], Jingchao Ni[2], Dongjin Song[2], Xuchao Zhang[2], Haifeng Chen[2], Xiang Zhang[1]

[1]The Pennsylvania State University
[2]NEC Laboratories America, Inc.
[3]Rutgers University
{dux19, xzz89}@psu.edu, {weicheng, bzong, wyu, jni, dsong, xuczhang, haifeng}@nec-labs.com, {xd48}@rutgers.edu

Abstract

We consider the models of deep multi-task learning with recurrent architectures that exploit regularities across tasks to improve the performance of multiple sequence processing tasks jointly. Most existing architectures are painstakingly customized to learn task relationships for different problems, which is not flexible enough to model the dynamic task relationships and lacks generalization abilities to novel test-time scenarios. We propose multi-task recurrent modular networks (MT-RMN) that can be incorporated in any multi-task recurrent models to address the above drawbacks. MT-RMN consists of a shared encoder and multiple task-specific decoders, and recurrently operates over time. For better flexibility, it modularizes the encoder into multiple layers of sub-networks and dynamically controls the connection between these sub-networks and the decoders at different time steps, which provides the recurrent networks with varying degrees of parameter sharing for tasks with dynamic relatedness. For the generalization ability, MT-RMN aims to discover a set of generalizable sub-networks in the encoder that are assembled in different ways for different tasks. The policy networks augmented with the differentiable routers are utilized to make the binary connection decisions between the sub-networks. The experimental results on three multi-task sequence processing datasets consistently demonstrate the effectiveness of MT-RMN.

Introduction

Deep learning models with recurrent architectures have been wildly studied for sequence processing tasks, such as sequence labeling (Lin et al. 2018; Cui et al. 2019), time series analysis (Dennis et al. 2019; Guo, Lin, and Antulov-Fantulin 2019), spatio-temporal prediction (Wang et al. 2018; Zhou et al. 2018; Sun et al. 2019) and question answering tasks (Ke et al. 2018). These models utilize a chain of repeating cells to encode the global information of a sequence into a word-level representation of its elements. Recent progresses have been made on jointly learning multiple tasks to improve overall performance by exploiting regularities across tasks (Chen et al. 2018; Gupta, Chakraborty, and Chakrabarti 2019; Cui et al. 2019; Lu, Bai, and Langlais 2019). Multi-task learning (MTL) not only helps reduce the risk of over-fitting to individual tasks, but also saves computation cost by sharing model architectures and low-level representations (Ruder 2017; Zhang and Yang 2017; Ma et al. 2019; Stickland and Murray 2019; Meyerson and Miikkulainen 2019).

The multi-task architectures applicable to recurrent models, however, are underexplored in literature. Existing approaches typically adopt the general architectures of multi-task learning to address sequence processing tasks. Most of them use a set of task-specific decoders to extract task-specific knowledge (Luong et al. 2016; Subramanian et al. 2018) and an ad-hoc structure to store the knowledge shared by related tasks (Chen et al. 2018), however, the task relationships are not considered explicitly and the structure is usually predefined, restricting the model capacity. It is also infeasible when users have limited knowledge of the task relatedness. Moreover, we highlight the dynamics of task relationships, which indicates the strength of relatedness between tasks is not static but subject to change, depending on the input data at hand (Cui et al. 2019; Liu et al. 2019), such as the relationships between the POS tagging tasks of code-switched sentences at different token positions (Gupta, Chakraborty, and Chakrabarti 2019; Lu, Bai, and Langlais 2019). Existing approaches are not flexible enough to learn the dynamic relationships, which calls for effective multi-task architecture that is elaborately developed for recurrent models. In addition, for more complex tasks of sequence processing, we conjecture that the generalization ability is essential (Lake 2019; Goyal et al. 2019), which helps the models handle the unseen tasks that are common in the real-world applications (Chang et al. 2019; Huang et al. 2019; Purushwalkam et al. 2019). Previous studies show that recurrent neural network models generalize well when the training and test data is similar, but fail spectacularly when generalization requires systematic compositional skills (Lake and Baroni 2018; Loula, Baroni, and Lake 2018). How to build recurrent neural architectures with the compositional generalization abilities would constitute a daunting challenge.

To address these challenges, we propose multi-task recurrent modular networks (MT-RMN) that dynamically learn task relationships and accordingly learn to assemble composable modules into complex layouts to jointly solve multiple sequence processing tasks. MT-RMN consists of a shared encoder to store the knowledge shared by related tasks and multiple decoders to extract task-specific knowledge. It recurrently operates over time. MT-RMN is general and with good compositional generalization ability. It is easy to be

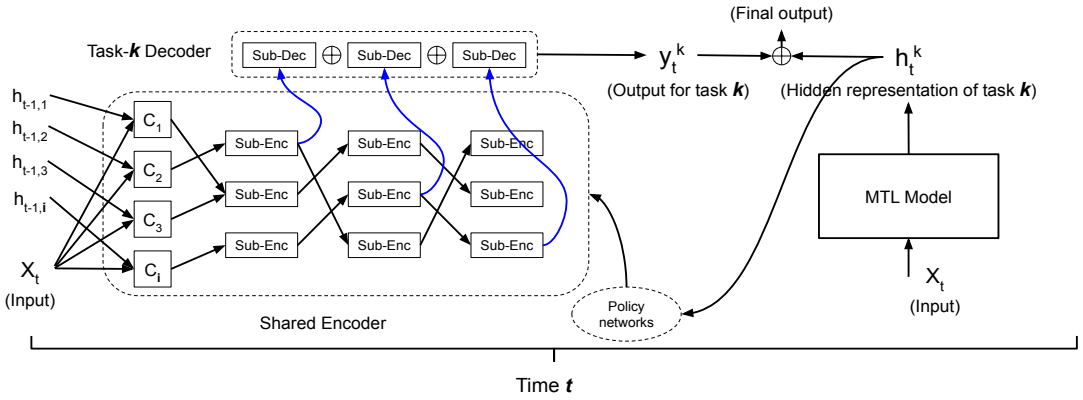


Figure 1: Architecture of MT-RMN.

incorporated in any multi-task recurrent model by combining the extracted task-specific knowledge with the task-specific output of the recurrent model.

Specifically, we employ modularity as a way of improving adaptability and generalization to novel test-time scenarios (Chang et al. 2019; Lake 2019; Pathak et al. 2019). The encoder is modularized into layers of sub-networks and a task-specific decoder consists of several sub-decoders. The first layer of the encoder is a group of recurrent cells and other layers consist of sub-encoders, representing a set of generalizable sub-networks that are assembled in different ways for different tasks. More importantly, modularity provides flexibility for the recurrent model to capture the dynamics of task relationships. The recurrent cells in MT-RMN operate independently and are used to capture the dynamics originated from different feature subspaces, similar to (Goyal et al. 2019). By dynamically learning the connections between sub-encoders and sub-decoders at different time steps, MT-RMN selectively activates different parts of the encoder for a task over time, which provides the recurrent networks with varying degrees of parameter sharing for tasks with dynamic relatedness. The connections are learned by decision policies (Guo et al. 2019; Ahmed and Torresani 2019), which are sampled from discrete distributions indirectly parameterized by the output of lightweight neural networks called policy networks. These policies decide to connect or disconnect the route between two sub-networks, on a per-instance basis.

Intuitively, similar tasks are routed through similar paths and dissimilar tasks are routed through different paths. This enables the model to express tasks as a combination of subtasks and to generalize to unseen categories by dynamically routing a set of sub-encoders. As such, apart from reusability and good generalization, modularity also provides interpretability. Users can inspect how the network operates to understand which tasks are deemed more related, and at which time steps the task relatedness dramatically change, etc. As the decisions are non-differentiable, we propose the differentiable binary routers based upon the Gumbel-Softmax reparameterization (Jang, Gu, and Poole 2017; Maddison, Mnih, and Teh 2017) to train the policy networks, which are jointly trained with other parts of the networks. MT-RMN is first validated on a task-fMRI dataset, where we analyze its ability to model task relationships dynamically and its generalization ability, then applied to a part-of-speech tagging dataset and a clinical time series dataset. Experimental results verify the effectiveness of MT-RMN.

The MT-RMN Architecture

The architecture of MT-RMN is shown in Figure 1. The connection decisions between modules are made by policy networks augmented with straight-through routers.

Shared Encoder

The shared encoder is motivated by modular networks (Kirsch, Kunze, and Barber 2018; Pahuja et al. 2019). We aim to discover a set of generalizable sub-networks that are assembled in different ways for different tasks. The encoder has m layers and each layer $\ell \in \{1, 2, ..., m\}$ contains n_ℓ sub-networks. The first layer is a group of independent LSTM cells. Other layers consist of MLPs that are used as sub-encoders.

Recurrent Independent Cells. Let $\boldsymbol{x}_t$ be the input data at time step t. We assume that the overall dynamical system of interest is divided into k small subsystems (Goyal et al. 2019). We use k recurrent cells to model these subsystems and each cell has its own independent dynamics operating. The cell denoted by C_i is flexible to use different RNN modules and we use LSTM (Hochreiter and Schmidhuber 1997) in this paper. The hidden state $\boldsymbol{h}_{t,i}$ after cell C_i is applied is

$$\boldsymbol{h}_{t,i} = LSTM_i(\boldsymbol{h}_{t-1,i}, \boldsymbol{x}_t; \theta_i). \quad (1)$$

Policy Networks for Sub-Encoders. We build a policy network for each sub-encoder to make decision connections between itself and the sub-networks of the previous layer. Specifically, for sub-encoder j in layer ℓ, policy network $\mathcal{N}_j$ estimates a decision vector $\boldsymbol{\alpha}_{t,ij} \in \mathbb{R}^2$ for every sub-network i in layer ℓ-1 at time step t, given the output $\boldsymbol{u}_{t,i}$ of sub-network i at time step t.

$$\{\mathcal{N}_j : \boldsymbol{u}_{t,i} \rightarrow \boldsymbol{\alpha}_{t,ij} | i \in \{1, ..., n_{\ell-1}\}, j \in \{1, ..., n_\ell\}\} \quad (2)$$

Routers. Given $\boldsymbol{\alpha}_{t,ij}$, a straight-through router is used to learn the decision policy $\mathcal{P}_{ij}$, which estimates a binary

decision value $\zeta_{t,ij} \in \{0, 1\}$, indicating whether to connect ($\zeta_{t,ij}$=1) or disconnect ($\zeta_{t,ij}$=0) the route between sub-networks i and sub-encoder j at time step t.

$$\{\mathcal{P}_{ij} : \boldsymbol{\alpha}_{t,ij} \rightarrow \zeta_{t,ij} | i \in \{1, ..., n_{\ell-1}\}, j \in \{1, ..., n_{\ell}\}\} \tag{3}$$

Calculation of Input and Output. Each sub-encoder j in layer ℓ receives a list of $n_{\ell-1}$ tuples of features from the sub-networks in layer ℓ-1, where tuple ($\boldsymbol{u}_{t,i}$, $\zeta_{t,ij}$) corresponds to sub-network i in layer ℓ-1. The input $\boldsymbol{v}_{t,j}$ and output $\boldsymbol{u}_{t,j}$ of sub-network j in layer ℓ are

$$\boldsymbol{v}_{t,j} = \sum_{i=1}^{n_{\ell-1}} \zeta_{t,ij} / (\sum_{i'=1}^{n_{\ell-1}} \zeta_{t,i'j}) \boldsymbol{u}_{t,i}, \tag{4}$$

$$\boldsymbol{u}_{t,j} = MLP_j(\boldsymbol{v}_{t,j}). \tag{5}$$

Task-Specific Decoders

Task-specific decoders share a similar idea with the encoder and they selectively activate only parts of the encoder. Every task-specific decoder $\mathcal{D}^k$ has m-1 independent sub-decoder $\mathcal{D}^k_\ell$, which are used to extract task-related knowledge from different layers of the encoder. We use MLPs as the sub-decoders in this paper. In particular, task-specific decoder $\mathcal{D}^k$ is used for task k and sub-decoder $\mathcal{D}^k_\ell$ is used to extract the knowledge from the ℓ-th layer of encoder. Because of the domain specialization of different tasks (Cui et al. 2019), we use different policy networks for different tasks.

The input and output calculation of the sub-decoder is similar to the one of the sub-encoder. For sub-decoder $\mathcal{D}^k_\ell$, we use policy network $\widehat{\mathcal{N}}^k$ to estimate a decision $\boldsymbol{\beta}^k_{t,j} \in \mathbb{R}^2$ for sub-encoder j in layer ℓ of encoder, *i.e.,* $\{\widehat{\mathcal{N}}^k : \boldsymbol{u}_{t,j} \rightarrow \boldsymbol{\beta}^k_{t,j}\}$, where $\boldsymbol{u}_{t,j}$ is the output of sub-encoder j. A router is further used to learn a policy $\widehat{\mathcal{P}}^k_j$ to estimates a binary decision value, *i.e.,* $\{\widehat{\mathcal{P}}^k_j : \boldsymbol{\beta}^k_{t,j} \rightarrow \widehat{\zeta}^k_{t,j} \in \{0, 1\}\}$. Thus, sub-decoder $\mathcal{D}^k_\ell$ receives a list of tuples ($\boldsymbol{u}_{t,j}$, $\widehat{\zeta}^k_{t,j}$), each of which corresponds to a sub-encoder of layer ℓ. The input and output of sub-decoder $\mathcal{D}^k_\ell$ are $\widehat{\boldsymbol{v}}^k_{t,\ell} = \sum_{j=1}^{n_\ell} \widehat{\zeta}^k_{t,j} / (\sum_{j'} \widehat{\zeta}^k_{t,j'}) \boldsymbol{u}_{t,j}$ and $\widehat{\boldsymbol{u}}^k_{t,\ell} = MLP_\ell(\widehat{\boldsymbol{v}}^k_{t,\ell})$ respectively. To consider the hierarchical structure information of the encoder, we concatenate the output of every sub-decoder $\mathcal{D}^k_l$ to construct the output of $\mathcal{D}^k$ as $[\widehat{\boldsymbol{u}}^k_{t,1} \oplus ... \oplus \widehat{\boldsymbol{u}}^k_{t,m-1}]$.

Policy Networks

The role of policy networks is to estimate the decision vector $\boldsymbol{\alpha}_{t,ij}$ (or $\boldsymbol{\beta}^k_{t,j}$), which is further fed into the routers to make the binary connection decisions. The form of policy networks is flexible. We opt to use MLP as the policy network for the encoder, which is defined as $\boldsymbol{\alpha}_{t,ij} = \mathcal{N}_j(\boldsymbol{u}_{t,i}) = MLP(\boldsymbol{u}_{t,i})$. Note that MT-RMN is used to be incorporated in an existing multi-task recurrent model. There are two types of multi-task recurrent models in general and they have different ways to incorporate MT-RMN. Thus, we propose two types of policy networks for the decoders. First, when the multi-task model itself generates task-specific representation $\boldsymbol{h}^k_t$ for task k, the policy networks are defined as

$$\boldsymbol{\beta}^k_{t,j} = \widehat{\mathcal{N}}^k(\boldsymbol{u}_{t,j}) = MLP(\boldsymbol{u}_{t,j} \oplus \boldsymbol{W}_k \boldsymbol{h}^k_t), \tag{6}$$

where $\boldsymbol{W}_k$ is the transformation matrix for task k. Second, when the multi-task model does not generate task-specific representations, the policy networks are defined as

$$\boldsymbol{\beta}^k_{t,j} = \widehat{\mathcal{N}}^k(\boldsymbol{u}_{t,j}) = MLP(\boldsymbol{u}_{t,j}). \tag{7}$$

The policy networks are jointly trained with other parts of MT-RMN. Their simple architecture makes the estimation of the decision vector fast and efficient, which is similar to the design of the policy network in (Guo et al. 2019) and the design of decision-learner in (Ahmed and Torresani 2019).

Straight-Through Router

Given the output, $\boldsymbol{\alpha}$ (or $\boldsymbol{\beta}$), of a policy network, a router is applied to learn the decision policy which estimates a binary decision value $\zeta \in \{0, 1\}$. Conceptually, the policy $\mathcal{P}$ can be seen as a binarization function of the decision scores $\boldsymbol{\alpha}$=$\{\alpha_0, \alpha_1\}$ and each value in the pair of the binary outcomes is the complement of the other. A simple way to implement the binarization function is to select the position with maximum value of $\{\alpha_0, \alpha_1\}$, however, this approach is non-differentiable. There are several ways that allow us to propagate gradients through the discrete nodes (Bengio, Léonard, and Courville 2013). In this work, we adopt the Gumbel-Softmax sampling approach (Jang, Gu, and Poole 2017; Maddison, Mnih, and Teh 2017).

We use Gumbel-Max trick (Maddison, Mnih, and Teh 2017). Specifically, we have two classes (disconnect and connect). $\{\alpha_0, \alpha_1\}$ represent the log probabilities $\{\log(p_0), \log(p_1)\}$ of the two classes. Thus, we can draw samples from a Bernoulli distribution parameterized by class probabilities $\{p_0, p_1\}$ in the following way: we first draw i.i.d samples $\{g_0, g_1\}$ from a Gumbel distribution, *i.e.,* $g = -\log(-\log(x)) \sim Gumbel$, where $x \sim$ Uniform(0, 1). Then produce the discrete sample by adding g to introduce stochasticity:

$$\zeta = \arg\max_i [\alpha_i + g_i], i \in \{0, 1\}. \tag{8}$$

The $\arg\max$ operation is non-differentiable, but we can use the softmax as a continuous differentiable approximation to it (Eq. (9)). τ is the temperature to control the discreteness. Thus, we use the $\arg\max$ to make the binary connection decision on the forward pass, while approximate it with softmax on the backward pass, which is called the straight-through estimator (Jang, Gu, and Poole 2017).

$$\mu_i = \exp\left((\alpha_i + g_i)/\tau\right) / (\sum_{\hat{i}=0}^{1} \exp\left((\alpha_{\hat{i}} + g_{\hat{i}})/\tau\right)), i \in \{0, 1\} \tag{9}$$

Sparsity Constrains

We add the sparsity constraint on the connection decision values $\zeta_{t,ij}$. We want the proposed module to avoid connecting every pair of sub-networks in the encoder, which can be viewed as the coarse-grained neural network pruning (Frankle and Carbin 2019) and benefits the specialization of sub-networks (Goyal et al. 2019). We also add the sparsity constraint on the connection decision values $\widehat{\zeta}^k_{t,i}$, which

helps the task-specific decoders avoid focusing on every sub-network of the encoder.

Inspired by (Ke et al. 2018), we add two penalty terms $\lambda_1 R_1(\boldsymbol{\zeta})$ and $\lambda_2 R_2(\widehat{\boldsymbol{\zeta}})$. They represent the penalties for connecting sub-networks in the encoder and the penalty for connections between the decoders and these sub-networks. Their definitions are

$$\lambda_1 R_1(\boldsymbol{\zeta}) = \lambda_1 ReLU((\sum \zeta_{t,ij}) - \gamma_1 C_1), \quad (10)$$

$$\lambda_2 R_2(\widehat{\boldsymbol{\zeta}}) = \lambda_2 ReLU((\sum \widehat{\zeta}^k_{t,i}) - \gamma_2 C_2), \quad (11)$$

λ and γ are hyper-parameters. C_1 is the number of all possible connections between sub-networks in the encoder and C_2 is the number of all possible connections between the sub-networks and sub-decoders. γ represents the proportion of connections that are without being penalized. Intuitively, each connection above the threshold $\gamma_1 C_1$ or $\gamma_2 C_2$ is penalized.

Complexity Analysis. MT-RMN is local in time. Its complexity per time step is proportional to the number of parameters. MT-RMN includes the encoder, decoders and policy networks, and they contain (ns+cr), $K(m$-$1)d$, $(K$+$n)p$ parameters respectively, where n, s, c, r, K, m, d, p are #sub-encoders, size of a sub-encoder, #cells, size of a cell, #tasks, #layers of a encoder, size of a sub-decoder, size of a policy network respectively. Thus, the complexity of MT-RMN is $\mathcal{O}(ns$+cr+$Kms)$.

Related Work

Multi-Task Learning. A number of MTL approaches (Luong et al. 2016; Subramanian et al. 2018) learned a shared low-level representation that is followed by unshared decoders to extract task-specific representations. However, the typical MTL methods might degenerate a lot when tasks are less related (Ruder 2017; Zhang and Yang 2017; Ma et al. 2019). Several approaches have been proposed to address this challenge. (Misra et al. 2016; Ruder et al. 2019) adopted trainable parameters to control the communications between different task-specific layers to share knowledge. Some used the routing methods to achieve flexible parameter sharing (Rosenbaum, Klinger, and Riemer 2018; Ma et al. 2018, 2019). However, the routing learned by (Ma et al. 2018, 2019) was applied to all tasks, instead of task-dependent. (Rosenbaum, Klinger, and Riemer 2018) employed the multi-agent reinforcement learning approach to jointly learn the routing and function blocks, but it did not focus on recurrent models. (Chen et al. 2018; Liu et al. 2019; Gupta, Chakraborty, and Chakrabarti 2019) applied MTL to sequence learning, however, these models are limited because they used a predefined structure to learn task relatedness. (Cases et al. 2019; Rosenbaum et al. 2019a) are closely related to ours. They extended (Rosenbaum, Klinger, and Riemer 2018) to natural language understanding. Our model differs from them in two crucial ways. First, our model considers layers of function modules instead of the recursive application of modules. The promise is that we can assemble the modules more flexibly to account for more complex task relationships, such as the hierarchical relationship (Huang et al. 2019; Lin et al. 2015). Second, our routing mechanism is more flexible, allowing multiple and dynamic connections between two layers of sub-encoders, which benefits the model capacity.

Modular Networks. We take inspiration from recent advances of modular works (Kirsch, Kunze, and Barber 2018; Pahuja et al. 2019). Modular networks are composed of modules and each module is a function with its own parameters. Learning an efficient set of such modules is akin to learning a set of functional primitives, which are evident in the natural world and can be combined to solve a given task (Meyerson and Miikkulainen 2018). The major advantage of modular networks is the compositional generalization ability, which has been studied in the context of modeling dynamical systems (Chang et al. 2019; Huang et al. 2019; Purushwalkam et al. 2019). Some recent progress has been made on the generalization of the complex sequence processing tasks (Lake and Baroni 2018; Loula, Baroni, and Lake 2018; Lake 2019). However, these methods are not designed for MTL. In addition, modular networks are related to routing networks (Haimerl, Savin, and Simoncelli 2019; Ramachandran and Le 2019; Rosenbaum, Klinger, and Riemer 2018), where each sample selectively activates only parts of the entire network. Some routing networks based on conditional computation are proposed to improve computational efficiency (Bengio et al. 2015; Ke et al. 2018; Guo et al. 2019). Several approaches routed the examples through a network in an MTL setting (Rosenbaum, Klinger, and Riemer 2018; Ramachandran and Le 2019). However, unlike these methods, we consider the recurrent models. (Cases et al. 2019; Rosenbaum et al. 2019a) considered the modular recurrent representation via routing. The differences between our model and them are discussed in the above paragraph. More introduction about modular and routing networks can be found in (Rosenbaum et al. 2019b).

Neural Architecture Search. Neural architecture search (NAS) aims to design the neural network architecture for a given task automatically (Zoph and Le 2017; Real et al. 2019). Our work can be considered as a special case of NAS, where we search for a dynamical multi-task model architecture that consists of a set of generalizable sub-networks over time. We particularly care about the parameter sharing problem dynamically. Previous NAS methods usually used an RNN-based controller to generate model architecture and then trained the target task based on this architecture (Zoph and Le 2017). Unfortunately, they are typically with high computation cost. Some efficient NSA methods have been proposed recently (Pham et al. 2018; Liu, Simonyan, and Yang 2019) and our module is of a similar spirit in the sense that the parameters of architecture and model are learned jointly, which is also shared by (Ma et al. 2019; Maziarz et al. 2019). However, all these methods focused on static models, while we focus on the recurrent models that differ per example and per time step.

Experiments

In our experiments, all the models are trained by Adam (Kingma and Ba 2014). The reported results were got by 5 times 5-fold cross validation. To make fair comparisons, we tuned the baselines as much as we could. We tried different parameter settings via grid-search, increased the number

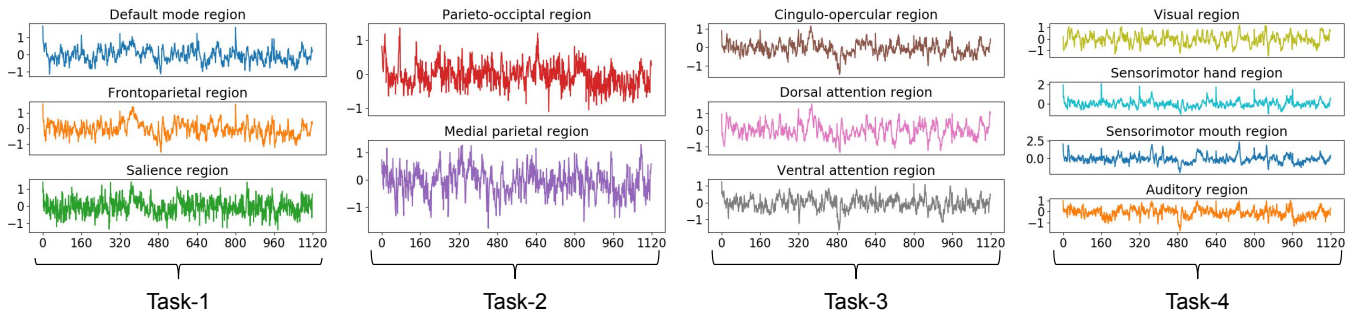
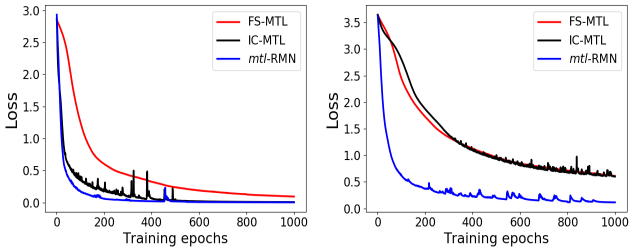


Figure 2: The task-fMRI data of twelve high-order brain regions of a participant. The regions are grouped into four groups based on their functionalities. Each group is used to construct a multi-class classification task of time series. The participant was asked to perform seven tasks successively (Barch et al. 2013).

of parameters to be similar and utilized different tricks. The learning rate was set to 10^{-3} initially and decreased during the training. We experimented with the values of k, λ_1, λ_2, γ_1 , γ_2 in the sets $\{2, 3, 4\}$, $\{2^{-2}, 2^{-1}, 2^0, 2^1, 2^2\}$, $\{2^{-2}, 2^{-1}, 2^0, 2^1, 2^2\}$, $\{0.25, 0.5, 0.75, 1\}$, $\{0.25, 0.5, 0.75, 1\}$ respectively. We obtained best results using k=3, λ_1=1, λ_1=1, γ_1=0.75, γ_2=0.75 in general. For the temperature τ used in the Gumbel-Max trick, we set its value as 100 initially and divided the value by 2 at each epoch. For all other parameters in the baseline methods, we adopted the default settings according to their original papers. The experimental results on the clinic time series dataset and the analysis of different model architectures are summarized in the appendix[1].



(a) Tasks are more related. (b) Tasks are less related.

Figure 3: Loss comparison.

Task-Evoked Functional Brain Activity

To test the flexibility and the generalization ability of MT-RMN in a controlled environment where we know the dynamic relationships between tasks, we create the *tfMRI* tasks based upon the task-evoked functional MRI dataset[2] (task-fMRI), which is widely used to analyze the relationship between brain connectivity and human behavior (Barch et al. 2013). Figure 2 shows the visualization of the task-fMRI data of a participant who performed seven tasks successively. There are twelve time series grouped into four groups and each time series corresponds to a high-order region of human brains. The x-axis represents time and the y-axis represents the normalized activation degree (see (Barch et al. 2013) for more details). The functional connectivity between regions varies over time, which indicates the relationships between the sequence processing tasks of brain regions are dynamic.

A *tfMRI* task is defined as a multi-class classification task of time series and we construct four tasks as shown in Figure 2. The brain regions involved in Task-1 and Task-4 are related to the cognition and sensory systems of humans respectively. The regions in Task-2 have mixed functions related to both cognition and sensory abilities. Task-3 is more related to the attention system. To extract meaningful features, each time series were further processed to be 28 time steps long with 10 feature dimensions (see supplementary material for pre-processing details). Every 4 time steps represent the time period when the participant conducted a task. The data was split into train (80%) and test (20%) sets randomly.

Flexibility To evaluate the flexibility, we consider two groups of *tfMRI* tasks. Group-1 includes Tasks-1,2,3 and Group-2 includes Tasks-1,3,4. The relationships between these tasks are dynamic but the tasks of Group-1 are more related in general compared to the tasks in Group-2. We create an MTL model (mtl-RMN) based on our networks. The network consists of three standard LSTMs that are connected to MT-RMN at each time step. Each LSTM is used for a *tfMRI* task to extract the task-specific hidden representation. The MT-RMN used here consists of a three-layer encoder and three task-specific decoders. The first layer of the encoder includes three LSTM cells and each of other layers contains three sub-encoders. Each decoder includes two sub-decoders. We choose this architecture because of its low enough capacity that contributes to the visible competition between tasks (Ramachandran and Le 2019).

We use five baselines that use different ways to utilize task relationships. FS-MTL (Caruana 1993) uses a fully shared neural layer for all tasks with separate task-specific output layers. SP-MTL (Liu, Qiu, and Huang 2017) uses the adversarial training to help the shared neural layer contains the task-invariant knowledge and uses the orthogonality constraint to eliminate knowledge redundancy. DC-MTL (Liu et al. 2019) has task-specific neural layers for different tasks

[1]Data and codes can be found in the authors' website.

[2]https://tinyurl.com/y4hhw8ro

Groups	Group-1			Group-2		
	Task-1	Task-2	Task-3	Task-1	Task-3	Task-4
FS-MTL	89.7±0.6	93.5±1.1	89.1±0.3	73.2±0.5	81.2±0.6	81.3±1.4
SP-MTL	88.7±1.4	94.0±1.9	90.7±2.3	73.7±1.5	78.0±1.5	81.2±2.1
DC-MTL	89.2±0.6	95.6±1.2	90.8±1.0	74.5±0.6	80.0±0.7	80.8±1.3
IC-MTL	89.6±0.4	95.7±1.1	91.3±1.7	74.5±0.4	81.4±1.1	81.8±2.6
RRNs	88.9±3.4	95.4±2.2	90.5±2.8	75.4±2.2	83.2±3.0	82.6±3.8
mtl-RMN	**90.8±2.1**	**96.7±1.6**	**92.9±2.4**	**76.1±1.8**	**84.4±2.4**	**83.5±2.2**

Table 1: Results (accuracy %) on two groups of *tfMRI* tasks.

and each task shares knowledge between any other directly. IC-MTL (Liu et al. 2019) uses an additional layer to share knowledge indirectly between tasks. RRNs (Cases et al. 2019) recursively chooses sub-functions from a single set of composable functions via a router for different input examples. We apply RRNs to route an LSTM input transformation.

We run these methods on Group-1 and Group-2 in the MTL setting, where the task weights are set to the same. We apply the softmax layer to the hidden representation at the last time step to get the predicted label. The results are reported in Table 1. It is observed that mtl-RMN achieved the best performance. When the tasks were more related (Group-1), all the methods showed high performance. However, the margin between the performance of mtl-RMN and the baselines was more clear when tasks were less related (Group-2). Both DC-MTL and IC-MTL outperformed FS-MTL on Group-1, but DC-MTL showed a little bit lower performance on Group-2. This is because the way of sharing knowledge in DC-MTL is more suitable for the case when tasks are more related. Compared to other baselines, RRNs showed higher performance on Group-2, because RRNs can limit task interference better for less related tasks. However, mtl-RMN outperformed RRNs on both groups, indicating the routing mechanism of mtl-RMN is more flexible. This might be because mtl-RMN can not only limit the shared weights for less related tasks, but also increase the shared ones for more related tasks via its dynamic/multiple connections between sub-encoder layers. We also report the loss (averaged over tasks) of FS-MTL, IC-MTL and mtl-RMN on the two groups over epochs, which are shown in Figure 3. We see that the average loss of mtl-RMN behaved in a similar way on the two groups, while FS-MTL and IC-MTL degenerated clearly when tasks were less related. This might be because there was task interference during training when tasks were less related, which resulted in negative transfer between tasks. mtl-RMN is more flexible and can reduce negative transfer more effectively. All the observations demonstrate that MT-RMN is able to learn task relationships flexibly.

Generalization Ability To evaluate the generalization ability, we create four novel test-time scenarios, A, B, C, D, based on the four *tfMRI* tasks, as shown in the left part of Table 2. Each case consists of three training tasks and a test task that is not included in the training task set. We hope to test if MT-RMN could discover a set of generalizable sub-networks to capture the sub-task patterns, which is critical for the compositional skills. We create a new model called RMN$_{tr}$. Specifically, we first train mtl-RMN on the training tasks. After training, we fix the parameters of the encoder and its policy networks. Then, we connect the fixed encoder (including its policy networks) with a new LSTM and a new decoder with a policy network. The data of test task is fed into both the new LSTM and the fixed encoder. The output of the new decoder is concatenated with the hidden representation of the new LSTM as the final representation which is further fed into a softmax layer.

We use four baselines: LSTM, RMN$_{un}$, RRNs and IC-MTL. We run LSTM on the test task directly. RMN$_{un}$ is a model where an untrained encoder of MT-RMN is connected with an LSTM and a decoder. We run RMN$_{un}$ on the test task directly. For IC-MTL, we first train it on training tasks. After training, the shared LSTM in IC-MTL is fixed. Then we connect it with a new LSTM. The results are reported in the right part of Table 2. RMN$_{tr}$ outperformed LSTM, which indicates that MT-RMN can help the base model (LSTM) handle the unseen tasks. RMN$_{tr}$ outperformed RMN$_{un}$, which indicates that the regularities across tasks help improve the performance of a single task. Both RMN$_{tr}$ and RRNs outperformed IC-MTL, indicating the modularity benefits generalization. Task-2 performed similarly in Tables 1-2, because Task-2 might be an easy task and the task relatedness does not influence its performance much. Tasks-1,3 performed worse in Table 2 than that in Group-1 of Table 1 but better than that in Group-2, because training tasks are more/less related in Group-1/2 than the ones in both scenarios B and D. All these observations demonstrate that MT-RMN can improve the systematic compositional skills of the base models and further help them generalize well for unseen tasks.

POS Tagging of Code-Switched Sentences

To directly evaluate the ability of MT-RMN to improve the performance of multi-task models on sequence processing tasks, we test MT-RMN on the multi-task setup of part-of-speech (POS) tagging (Santos and Zadrozny 2014) of code-switched sentences, where the words are from two languages and the goal is to label the sequences from code-switched text (Gonen and Goldberg 2019). Code switching is common, especially in social media where the participants are multilingual users (Vyas et al. 2014). Given the input sentence denoted by $(w_1, w_2, w_3, ...)$, where words are from languages A, B, i.e., $w_i \in D_A \cup D_B$, the goal is to predict a label sequence $(y_1, y_2, y_3, ...)$. We pre-process the data and create our setup exactly following (Gupta, Chakraborty, and Chakrabarti 2019). Specifically, we use the Hindi-English code-switched dataset provided in (Patra, Das, and Das 2018)

Scenario settings					Results of different methods				
Scenarios	Training tasks			Test task	LSTM	RMN$_{un}$	RRNs	IC-MTL	RMN$_{tr}$
A	Task-1	Task-2	Task-3	Task-4	82.6±1.0	82.9±1.2	83.0±2.7	81.2±1.7	**83.3±2.3**
B	Task-1	Task-2	Task-4	Task-3	82.3±0.6	84.5±1.5	84.8±2.5	82.7±1.3	**85.6±1.7**
C	Task-1	Task-3	Task-4	Task-2	93.7±0.6	95.3±0.8	95.6±1.7	95.5±0.7	**97.0±1.8**
D	Task-2	Task-3	Task-4	Task-1	71.7±1.4	73.1±1.7	78.4±4.0	77.3±2.1	**79.3±2.6**

Table 2: Four test-time scenarios to evaluate generalization ability and the results (accuracy%).

Methods	Original		DC-MTL		IC-MTL		rrn-RMN		MT-RMN	
	Accuracy	F_1	Accuracy	F_1	Accuracy	F_1	Accuracy	F_1	Accuracy	F_1
NS-MTL	44.3±0.7	38.9±1.6	51.5±0.5	49.4±0.9	51.4±1.4	49.7±0.8	53.5±1.9	52.7±0.8	**55.2±1.2**	**53.4±2.4**
Cross-stitch.	46.0±1.9	12.7±0.8	48.5±0.9	16.1±1.3	47.7±1.7	14.3±0.7	**52.1±1.0**	19.5±0.6	51.7±1.9	**21.4±3.2**
Sluice	58.7±0.7	55.4±2.1	59.2±1.7	56.0±2.3	59.5±1.9	56.2±0.7	60.4±1.2	56.6±0.7	**61.5±1.7**	**57.2±1.8**
GIRNet	62.8±0.6	46.6±0.4	62.5±1.1	57.4±0.5	63.1±1.5	58.6±1.3	63.6±1.7	61.3±1.5	**64.5±2.1**	**62.6±2.4**

Table 3: Results of POS tagging.

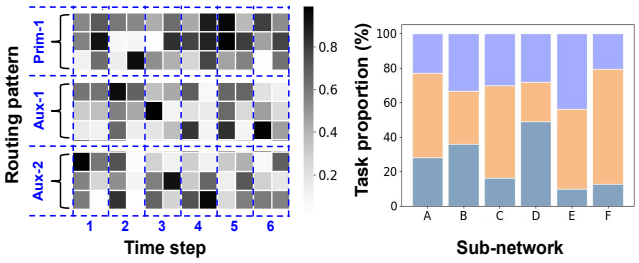


Figure 4: (*Left*) The routing patterns of examples of each task. The value represents the proportion of examples which activate the corresponding sub-network. (*Right*) The proportion of tasks assigned to each sub-network. Different colors distinguish different tasks and A-F represent the six sub-networks.

for the primary task. The dataset has 19 POS tags and contains 2102 and 528 instances for the training and test sets respectively. We use the Hindi POS tagging dataset provided in (Sachdeva et al. 2014) and the English one provided in (Sang and Buchholz 2000) for the auxiliary tasks of Hindi and English respectively. The Hindi dataset includes 14084 instances with 25 POS tags and the English dataset includes 8936 instances with 45 POS tags.

We compare MT-RMN to three baselines, including DC-MTL, IC-MTL and a variant called rrn-RMN, where we replace the layers of sub-encoders with one set of sub-encoders in the recursive layout. We further incorporate these methods into five MTL approaches to test their ability to improve these MTL approaches. NS-MTL has an independent LSTM layer for each task and all the tasks are jointly optimized. Cross-stitch (Misra et al. 2016) has task-specific layers for different tasks. It uses parameters to control the amount of information shared between layers. Sluice (Ruder et al. 2019) splits the channels of layers and controls the information sharing by parameters. The outputs of intermediate layers are fed to the output layers. GIRNet (Gupta, Chakraborty, and Chakrabarti 2019) contains two LSTMs that are trained over auxiliary instances. The state sequences of the two LSTMs are merged into a composite state sequence for the primary task.

Results. We show the results in Table 3. MT-RMN improved these base models most compared to other baselines. NS-MTL performed much better when combined with any of the four methods, indicating the necessity of exploiting regularities across tasks to improve task performance. MT-RMN and rrn-RMN helped the base models perform better compared to others, which verifies the better ability of modular networks with routing to exploit regularities. Both Sluice and GIRNet outperformed Cross-stitch because they share knowledge at the granularity of word. All the observations verify the advantage of MT-RMN in improving the multi-task models on sequence processing tasks.

Sub-Encoder Specialization. To verify that the sub-encoders can capture primitives for different tasks, we visualize the routing patterns. We use a three-layer encoder that has 3 cells in the 1st layer and 3 sub-encoders in each of others. We incorporate this MT-RMN into GIRNet and test it on the POS tagging task. After training, examples from the test set are passed through MT-RMN. The results of the first 6 time steps are shown in Figure 4. The routing pattern of each time step is represented by a 3×2 grid, each of which corresponds to a sub-encoder. We aggregate the connection choices (between sub-encoders and sub-decoders) for every sample and concatenate the grids of different time steps. Figure 4 (left) shows that the routing patterns varied over time and the examples from different tasks tended to be routed through different sub-encoders. Figure 4 (right) shows how different sub-encoders handle different task distributions. All the observations demonstrate that sub-encoders focus on different tasks and can be assembled for different tasks.

Conclusion

We propose MT-RMN to improve the performance of multi-task recurrent models. MT-RMN includes a shared encoder and a set of task-specific decoders. It modularizes the encoder into layers of sub-encoders for flexibility and dynamically learns the connections via decision policies. MT-RMN recurrently operates over time and is easy to be incorporated in any multi-task recurrent models. Experiments demonstrate the effectiveness of MT-RMN.

Acknowledgements

This project was partially supported by NSF projects IIS-1707548 and CBET-1638320. This project was done when the first author was an intern at NEC Labs America. Wei Cheng is the corresponding author.

Ethics Statement

Our proposed networks are general and can be easily incorporated in any recurrent model to improve its performance via exploiting regularities across tasks. There are several impacts of using our networks. First, our networks enable models to generalize to unseen categories, which are common in real-world, via dynamically routing a set of sub-encoders. Second, our networks help models save computing resources via reusing the well-trained and generalizable sub-encoders for different tasks. Third, our networks help users inspect how the model operates to understand the dynamic relationships between tasks, such as the analysis of brain connectivity and human behavior based on the fMRI dataset. However, our networks also put deep recurrent models at a high risk of being attacked. Our networks seek a set of sub-networks that are assembled in different ways for different tasks. Disturbing these sub-networks leads to significant changes on the task performance. Besides, increasing the interpretability of deep recurrent models might bring about automation bias, such as an unwarranted trust on the models.

References

Ahmed, K.; and Torresani, L. 2019. STAR-Caps: Capsule networks with straight-through attentive routing. In *NeurIPS*, 9098–9107.

Barch, D. M.; Burgess, G. C.; Harms, M. P.; Petersen, S. E.; Schlaggar, B. L.; Corbetta, M.; Glasser, M. F.; Curtiss, S.; Dixit, S.; Feldt, C.; et al. 2013. Function in the human connectome: task-fMRI and individual differences in behavior. *Neuroimage* 80: 169–189.

Bengio, E.; Bacon, P.-L.; Pineau, J.; and Precup, D. 2015. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297* .

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* .

Caruana, R. 1993. Multitask learning: A knowledge-based source of inductive bias. In *ICML*, 41–48. Morgan Kaufmann.

Cases, I.; Rosenbaum, C.; Riemer, M.; Geiger, A.; Klinger, T.; Tamkin, A.; Li, O.; Agarwal, S.; Greene, J. D.; Jurafsky, D.; et al. 2019. Recursive routing networks: Learning to compose modules for language understanding. In *NAACL-HLT*, 3631–3648.

Chang, M.; Gupta, A.; Levine, S.; and Griffiths, T. L. 2019. Automatically composing representation transformations as a means for generalization. In *ICLR*.

Chen, J.; Qiu, X.; Liu, P.; and Huang, X. 2018. Meta multi-task learning for sequence modeling. In *AAAI*.

Cui, W.; Zheng, G.; Shen, Z.; Jiang, S.; and Wang, W. 2019. Transfer learning for sequences via learning to collocate. In *ICLR*.

Dennis, D.; Acar, D. A. E.; Mandikal, V.; Sadasivan, V. S.; Saligrama, V.; Simhadri, H. V.; and Jain, P. 2019. Shallow RNN: accurate time-series classification on resource constrained devices. In *NeurIPS*, 12896–12906.

Frankle, J.; and Carbin, M. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*.

Gonen, H.; and Goldberg, Y. 2019. Language modeling for code-switching: Evaluation, integration of monolingual data, and discriminative training. In *EMNLP-IJCNLP*.

Goyal, A.; Lamb, A.; Hoffmann, J.; Sodhani, S.; Levine, S.; Bengio, Y.; and Schölkopf, B. 2019. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893* .

Guo, T.; Lin, T.; and Antulov-Fantulin, N. 2019. Exploring interpretable LSTM neural networks over multi-variable data. In *ICML*.

Guo, Y.; Shi, H.; Kumar, A.; Grauman, K.; Rosing, T.; and Feris, R. 2019. SpotTune: transfer learning through adaptive fine-tuning. In *CVPR*, 4805–4814.

Gupta, D.; Chakraborty, T.; and Chakrabarti, S. 2019. Girnet: Interleaved multi-task recurrent state sequence models. In *AAAI*, volume 33, 6497–6504.

Haimerl, C.; Savin, C.; and Simoncelli, E. 2019. Flexible information routing in neural populations through stochastic comodulation. In *NeurIPS*, 14379–14388.

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

Huang, D.-A.; Nair, S.; Xu, D.; Zhu, Y.; Garg, A.; Fei-Fei, L.; Savarese, S.; and Niebles, J. C. 2019. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. In *CVPR*, 8565–8574.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical reparameterization with gumbel-softmax. In *ICLR*.

Ke, N. R.; Żołna, K.; Sordoni, A.; Lin, Z.; Trischler, A.; Bengio, Y.; Pineau, J.; Charlin, L.; and Pal, C. 2018. Focused hierarchical rnns for conditional sequence processing. In *ICML*, 2554–2563. PMLR.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Kirsch, L.; Kunze, J.; and Barber, D. 2018. Modular networks: Learning to decompose neural computation. In *NeurIPS*, 2408–2418.

Lake, B. M. 2019. Compositional generalization through meta sequence-to-sequence learning. In *NeurIPS*.

Lake, B. M.; and Baroni, M. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *ICML*.

Lin, R.; Liu, S.; Yang, M.; Li, M.; Zhou, M.; and Li, S. 2015. Hierarchical recurrent neural network for document modeling. In *EMNLP*, 899–907.

Lin, Y.; Yang, S.; Stoyanov, V.; and Ji, H. 2018. A multi-lingual multi-task architecture for low-resource sequence labeling. In *ACL*, 799–809.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. Darts: Differentiable architecture search. In *ICLR*.

Liu, P.; Fu, J.; Dong, Y.; Qiu, X.; and Cheung, J. C. K. 2019. Learning multi-task communication with message passing for sequence learning. In *AAAI*, volume 33, 4360–4367.

Liu, P.; Qiu, X.; and Huang, X. 2017. Adversarial multi-task learning for text classification. In *ACL*.

Loula, J.; Baroni, M.; and Lake, B. M. 2018. Rearranging the familiar: Testing compositional generalization in recurrent networks. In *EMNLP Workshop*.

Lu, P.; Bai, T.; and Langlais, P. 2019. SC-LSTM: Learning task-specific representations in multi-task learning for sequence labeling. In *NAACL-HLT*, 2396–2406.

Luong, M.-T.; Le, Q. V.; Sutskever, I.; Vinyals, O.; and Kaiser, L. 2016. Multi-task sequence to sequence learning. In *ICLR*.

Ma, J.; Zhao, Z.; Chen, J.; Li, A.; Hong, L.; and Chi, E. H. 2019. SNR: Sub-Network Routing for Flexible Parameter Sharing in Multi-task Learning. In *AAAI*, 216–223.

Ma, J.; Zhao, Z.; Yi, X.; Chen, J.; Hong, L.; and Chi, E. H. 2018. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *KDD*, 1930–1939.

Maddison, C. J.; Mnih, A.; and Teh, Y. W. 2017. The concrete distribution: A continuous relaxation of discrete random variables. *ICLR* .

Maziarz, K.; Kokiopoulou, E.; Gesmundo, A.; Sbaiz, L.; Bartok, G.; and Berent, J. 2019. Gumbel-Matrix Routing for Flexible Multi-task Learning. *arXiv:1910.04915* .

Meyerson, E.; and Miikkulainen, R. 2018. Beyond shared hierarchies: deep multitask learning through soft layer ordering. In *ICLR*.

Meyerson, E.; and Miikkulainen, R. 2019. Modular universal reparameterization: deep multi-task learning across diverse domains. In *NeurIPS*, 7901–7912.

Misra, I.; Shrivastava, A.; Gupta, A.; and Hebert, M. 2016. Cross-stitch networks for multi-task learning. In *CVPR*, 3994–4003.

Pahuja, V.; Fu, J.; Chandar, S.; and Pal, C. J. 2019. Structure learning for neural module networks. *arXiv preprint arXiv:1905.11532* .

Pathak, D.; Lu, C.; Darrell, T.; Isola, P.; and Efros, A. A. 2019. Learning to control self-assembling morphologies: a study of generalization via modularity. In *NeurIPS*.

Patra, B. G.; Das, D.; and Das, A. 2018. Sentiment Analysis of Code-Mixed Indian Languages: An Overview of SAIL_Code-Mixed Shared Task@ ICON-2017. *arXiv preprint arXiv:1803.06745* .

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. In *ICML*.

Purushwalkam, S.; Nickel, M.; Gupta, A.; and Ranzato, M. 2019. Task-Driven Modular Networks for Zero-Shot Compositional Learning. In *ICCV*, 3593–3602.

Ramachandran, P.; and Le, Q. V. 2019. Diversity and depth in per-example routing models. In *ICLR*.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized evolution for image classifier architecture search. In *AAA*, volume 33, 4780–4789.

Rosenbaum, C.; Cases, I.; Riemer, M.; Geiger, A.; Karttunen, L.; Greene, J. D.; Jurafsky, D.; and Potts, C. 2019a. Dispatched routing networks. *Technical Report Stanford AI Lab, NLP Group Tech Report 2019-1* .

Rosenbaum, C.; Cases, I.; Riemer, M.; and Klinger, T. 2019b. Routing networks and the challenges of modular and compositional computation. *arXiv preprint arXiv:1904.12774* .

Rosenbaum, C.; Klinger, T.; and Riemer, M. 2018. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *ICLR*.

Ruder, S. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* .

Ruder, S.; Bingel, J.; Augenstein, I.; and Søgaard, A. 2019. Latent multi-task architecture learning. In *AAAI*, 4822–4829.

Sachdeva, K.; Srivastava, R.; Jain, S.; and Sharma, D. M. 2014. Hindi to English Machine Translation: Using Effective Selection in Multi-Model SMT. In *LREC*, 1807–1811.

Sang, E. F.; and Buchholz, S. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *CONLL Workshop*.

Santos, C. D.; and Zadrozny, B. 2014. Learning character-level representations for part-of-speech tagging. In *ICML*, 1818–1826.

Stickland, A. C.; and Murray, I. 2019. BERT and PALs: Projected attention layers for efficient adaptation in multi-task learning. In *ICML*.

Subramanian, S.; Trischler, A.; Bengio, Y.; and Pal, C. J. 2018. Learning general purpose distributed sentence representations via large scale multi-task learning. In *ICLR*.

Sun, C.; Shrivastava, A.; Vondrick, C.; Sukthankar, R.; Murphy, K.; and Schmid, C. 2019. Relational action forecasting. In *CVPR*, 273–283.

Vyas, Y.; Gella, S.; Sharma, J.; Bali, K.; and Choudhury, M. 2014. Pos tagging of english-hindi code-mixed social media content. In *EMNLP*, 974–979.

Wang, Y.; Gao, Z.; Long, M.; Wang, J.; and Yu, P. S. 2018. Predrnn++: Towards a resolution of the deep-in-time dilemma in spatiotemporal predictive learning. In *ICML*.

Zhang, Y.; and Yang, Q. 2017. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114* .

Zhou, Y.; Li, Z.; Xiao, S.; He, C.; Huang, Z.; and Li, H. 2018. Auto-conditioned recurrent networks for extended complex human motion synthesis. In *ICLR*.

Zoph, B.; and Le, Q. V. 2017. Neural architecture search with reinforcement learning. In *ICLR*.