Multi-task Learning with Task, Group, and Universe Feature Learning

Shiva Pentyala[*]
Texas A&M University
pk123@tamu.edu

Mengwen Liu
Amazon Alexa
mengwliu@amazon.com

Markus Dreyer
Amazon Alexa
mddreyer@amazon.com

Abstract

We present methods for multi-task learning that take advantage of natural groupings of related tasks. Task groups may be defined along known properties of the tasks, such as task domain or language. Such task groups represent supervised information at the inter-task level and can be encoded into the model. We investigate two variants of neural network architectures that accomplish this, learning different feature spaces at the levels of individual tasks, task groups, as well as the universe of all tasks: (1) parallel architectures encode each input *simultaneously* into feature spaces at different levels; (2) serial architectures encode each input *successively* into feature spaces at different levels in the task hierarchy. We demonstrate the methods on natural language understanding (NLU) tasks, where a grouping of tasks into different task domains leads to improved performance on ATIS, Snips, and a large in-house dataset.

1 Introduction

In multi-task learning (Caruana, 1993), multiple related tasks are learned together. Rather than learning one task at a time, multi-task learning uses information sharing between multiple tasks. This technique has been shown to be effective in multiple different areas, e.g., vision (Zhang et al., 2014), medicine (Bickel et al., 2008), and natural language processing (Collobert and Weston, 2008; Luong et al., 2016; Fan et al., 2017).

The selection of tasks to be trained together in multi-task learning can be seen as a form of supervision: The modeler picks tasks that are known *a priori* to share some commonalities and decides to train them together. In this paper, we consider the case when information about the *relationships* of these tasks is available as well, in the form of natural *groups* of these tasks. Such task groups can be available in various multi-task learning scenarios: In *multi-language* modeling, when learning to parse or translate multiple languages jointly, information on language families would be available; in *multimodal* modeling, e.g., when learning text tasks and image tasks jointly, clustering the tasks into these two groups would be natural. In *multi-domain* modeling, which is the focus of this paper, different tasks naturally group into different domains.

We hypothesize that adding such inter-task supervision can encourage a model to generalize along the desired task dimensions. We introduce neural network architectures that can encode task groups, in two variants:

- Parallel network architectures encode each input *simultaneously* into feature spaces at different levels;
- serial network architectures encode each input *successively* into feature spaces at different levels in the task hierarchy.

These neural network architectures are general and can be applied to any multi-task learning problem in which the tasks can be grouped into different task groups.

Application Example. To illustrate our method, we now introduce the specific scenario that we use to evaluate our method empirically: multi-domain natural language understanding (NLU) for virtual assistants. Such assistants, e.g., Alexa, Cortana, or Google Assistant, perform a range of tasks in different domains (or, categories), such as *Music*, *Traffic*, *Calendar*, etc. With the advent of frameworks like Alexa Skills Kit, Cortana Skills Kit and Actions on Google, third-party developers can extend the capabilities of these virtual assistants by

[*] This work was done while Shiva Pentyala was interning at Amazon Alexa.

developing new tasks, which we call *skills*, e.g., *Uber*, *Lyft*, *Fitbit*, in any given domain. Each skill is defined by a set of intents that represents different functions to handle a user's request, e.g., a `play_artist` or `play_station` for a skill in the *Music* domain. Each intent can be instantiated with particular slots, e.g., `artist` or `song`. An utterance like "play madonna" may be parsed into intent and slots, resulting in a structure like `PlayArtist(artist="madonna")`. Skill developers provide their own labeled sample utterances in a grammar-like format (Kumar et al., 2017), individually choosing a label space that is suitable for their problem.[1] We learn intent classification (IC) and slot filling (SF) models for these skills, in order to recognize user utterances spoken to these skills that are similar but not necessarily identical to the sample utterances given by their skill developers.

In this paper, we apply multi-task learning to this problem, learning the models for all skills jointly, as the individual training data for any given skill may be small and utterances across multiple skills have similar characteristics, e.g., they are often short commands in spoken language. In addition, we wish to add information on task groups (here: skill domains) into these models, as utterances in different skills of the same domain may be especially similar. For example, although the *Uber* and *Lyft* skills are built by independent developers, end users may speak similar utterances to them. For example, users may say "get me a ride" to *Uber* and "I'm requesting a ride" to *Lyft*.

Contributions. The main contributions of this paper are as follows:

- We introduce unified models for multi-task learning that learn three sets of features: *task*, *task group*, and *task universe* features;
- we introduce two architecture variants of such multi-task models: *parallel* and *serial* architectures;
- we evaluate the proposed models to perform multi-domain joint learning of slot filling and intent classification tasks on both public datasets and a real-world dataset from the Alexa virtual assistant;
- we demonstrate experimentally the superiority of introducing group-level features and learning features in both parallel and serial ways.

[1] They may choose to pick from a set of predefined labels with prepopulated content, e.g., *cities* or *first names*.

2 Proposed Architectures

The goal of multi-task learning (MTL) is to utilize shared information across related tasks. The features learned in one task could be transferred to reinforce the feature learning of other tasks, thereby boosting the performance of all tasks via mutual feedback within a unified MTL architecture. We consider the problem of multi-domain natural language understanding (NLU) for virtual assistants. Recent progress has been made to build NLU models to identify and extract structured information from user's request by jointly learning two tasks, intent classification (IC) and slot filling (SF) (Tur et al., 2010). However, in practice, a common issue when building NLU models for every skill is that the amount of annotated training data varies across skills and is small for many individual skills. Motivated by the idea of learning multiple tasks jointly, the paucity of data can be resolved by transferring knowledge between different tasks that can reinforce one another.

In what follows, we describe four end-to-end MTL architectures (Sections 2.1 to 2.4). These architectures are encoder-decoder architectures where the encoder extracts three different sets of features: *task*, *task-group*, and *task-universe* features, and the decoder produces desired outputs based on feature representations. In particular, the first one (Fig. 1) is a parallel MTL architecture where task, task-group, and task-universe features are encoded in parallel and then concatenated to produce a composite representation. The next three architectures (Fig. 2) are serial architectures in different variants: In the first serial MTL architecture, group and universe features are learned first and are then used as inputs to learn task-specific features. The next serial architecture is similar but introduces highway connections that feed representations from earlier stages in the series directly into later stages. In the last architecture, the order of serially learned features is changed, so that tasks-specific features are encoded first.

In Section 2.5, we introduce an encoder-decoder architecture to perform slot filling and intent classification jointly in a multi-domain sce-

nario for virtual assistants. Although we conduct experiments on multi-domain NLU systems of virtual assistants, the architectures can easily be applied to other tasks. Specifically, the encoder/decoder could be instantiated with any components or architectures, i.e., Bi-LSTM (Hochreiter and Schmidhuber, 1997) for the encoder, and classification or sequential labeling for the decoder.

2.1 A Parallel MTL Architecture

The first architecture, shown in Figure 1, is designed to learn the three sets of features at the same stage; therefore we call it a parallel MTL architecture, or PARALLEL[UNIV+GROUP+TASK]. This architecture uses three types of encoders: 1) A universe encoder to extract the common features across all tasks; 2) task-specific encoders to extract task-specific features; and 3) group-specific encoders to extract features within the same group. Finally, these three feature representations are concatenated and passed through the task-specific decoders to produce the output.

Assume we are given a MTL problem with m groups of tasks. Each task is associated with a dataset of training examples $D = \{(\boldsymbol{x}_1^i, \boldsymbol{y}_1^i), ..., (\boldsymbol{x}_{m_i}^i, \boldsymbol{y}_{m_i}^i)\}_{i=1}^m$, where $\boldsymbol{x}_k^i$ and $\boldsymbol{y}_k^i$ denote input data and corresponding labels for task k in group i. The parameters of the parallel MTL model (and also for the other MTL models) are trained to minimize the weighted sum of individual task-specific losses that can be computed as:

$$\mathcal{L}_{\text{tasks}} = \sum_{i=1}^{m} \sum_{j=1}^{m_i} \alpha_j^i * \mathcal{L}_j^i, \qquad (1)$$

where α_j^i is a static weight for task j in group i, which could be proportional to the size of training data of the task. The loss function $\mathcal{L}_j^i$ is defined based on the tasks performed by the decoder, which will be described in Section 2.5.

To eliminate redundancy in the features learned by three different types of encoders, we add adversarial loss and orthogonality constraints cost (Bousmalis et al., 2016; Liu et al., 2017). Adding adversarial loss aims to prevent task-specific features from creeping into the shared space. We apply adversarial training to our shared encoders, i.e., the universe and group encoders. To encourage task, group, and universe encoders to learn features from different aspects of the inputs, we add orthogonality constraints between task and

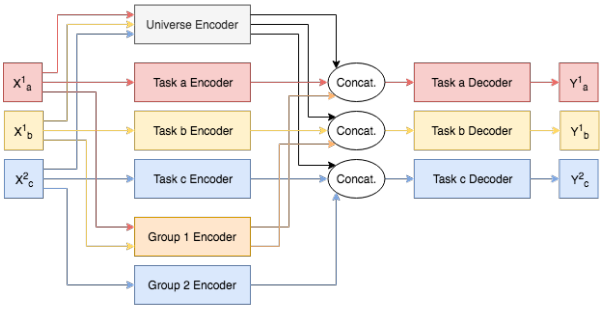


Figure 1: The PARALLEL[UNIV+GROUP+TASK] architecture, which learns universe, group, and task features. Three tasks a, b, and c are illustrated in the figure where $a, b \in \text{group}_1$ and $c \in \text{group}_2$.

task-universe/group representations of each domain. The loss function defined in Equation 1 becomes:

$$\mathcal{L}_{\text{all}} = \mathcal{L}_{\text{tasks}} + \lambda * \mathcal{L}_{\text{adv}} + \gamma * \mathcal{L}_{\text{ortho}} \qquad (2)$$

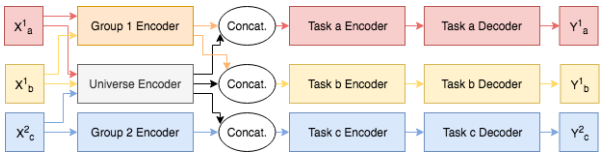
where $\mathcal{L}_{\text{adv}}$ and $\mathcal{L}_{\text{ortho}}$ denote the loss function for adversarial training and orthogonality constraints respectively, and λ and γ are hyperparameters.

2.2 A Serial MTL Architecture

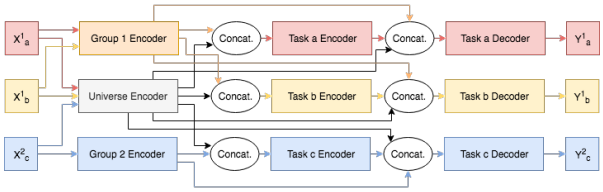
The second MTL architecture, called SERIAL, has the same set of encoders and decoders as the parallel MTL architecture. The differences are 1) the order of learning features and 2) the input for individual decoders. In this serial MTL architecture, three sets of features are learned in a sequential way in two stages. As shown in Figure 2a, group encoders and a universe encoder encode group-level and fully shared universe-level features, respectively, based on input data. Then, task encoders use that concatenated feature representation to learn task-specific features. Finally, in this serial architecture, the individual task decoders use their corresponding private encoder outputs only to perform tasks. This contrasts with the parallel MTL architecture, which uses combinations of three feature representations as input to their respective task decoders.

2.3 A Serial MTL Architecture with Highway Connections

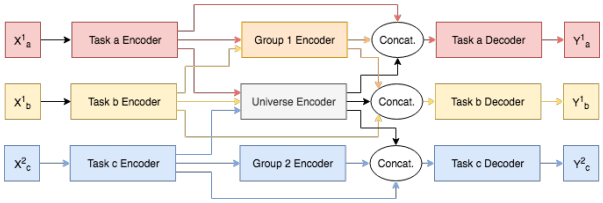
Decoders in the SERIAL architecture, introduced in the previous section, do not have direct access to group and universe feature representations. However, directly utilizing these shared features could be beneficial for some tasks. Therefore, we introduce additional highway connections to incor-



(a) SERIAL



(b) SERIAL+HIGHWAY



(c) SERIAL+HIGHWAY+SWAP

Figure 2: Three serial MTL architectures. In each of these architectures, individual decoders utilize all three sets of features (task, universe, and group features) to perform a task. Three tasks a, b, and c are illustrated in the figures where $a, b \in \text{group}_1$ and $c \in \text{group}_2$.

porate universe encoder output and corresponding group encoder outputs as inputs to the individual decoders in addition to task-specific encoder output; we call this model SERIAL+HIGHWAY.

As shown in Figure 2b, input to the task-specific encoders are the same as those in the serial MTL architecture, i.e., the concatenation of the group and universe features. The input to a task-specific decoder, however, is now the concatenation of the features from the group encoder, the universe encoder, and the task-specific encoder.

2.4 A Serial MTL Architecture with Highway Connections and Feature Swapping

In both serial MTL architectures introduced in the previous two sections, the input to the task encoders is the output of the more general group and universe encoders. That output potentially underrepresents some task-specific aspects of the input. Therefore, we introduce SERIAL+HIGHWAY+SWAP; a variant of SERIAL+HIGHWAY, in which the two stages of universe/group features and task-specific features are swapped. As shown in Figure 2c, the task-specific representations are now learned in the first stage, and group and universe feature representations based on the task features are learned in the second stage. In this model, the task encoder directly takes input data and learns task-specific features. Then, the universe encoder and group encoders take the task-specific representations as input and generate fully shared universe and group-level representations, respectively. Finally, task-specific decoders use the concatenation of all three features – universe, group and task features, to perform the final tasks.

2.5 An Example of Encoder-Decoder Architecture for a Single Task

All four MTL architectures introduced in the previous sections are general such that they could be applied to many applications. In this section, we use the task of joint slot filling (SF) and intent classification (IC) for natural language understanding (NLU) systems for virtual assistants as an example. We design an encoder-decoder architecture to perform SF and IC as a joint task, on top of which the four MTL architectures are built.

Given an input sequence $\boldsymbol{x} = (x_1, \ldots, x_T)$, the goal is to jointly learn an equal-length tag sequence of slots $\boldsymbol{y}^S = (y_1, \ldots, y_T)$ and the overall intent label y^I. By using a joint model, rather than two separate models, for SF and IC, we exploit the correlation of the two output spaces. For example, if the intent of a sentence is *BookRide*, then it is likely to contain the slot type *fromAddress* and *destinationAddress*, and vice versa. The JOINT-SF-IC model architecture is shown in Figure 2. It is a simplified version compared to the SLOTGATED model (Goo et al., 2018) which showed state-of-the-art results in jointly modeling SF and IC. By simplicity, we mean our architecture is neither using slot/intent attention nor a slot gate.

To overcome the challenges with small amounts of training data and out-of-vocabulary (OOV) words, we use pre-trained word embeddings along with character embeddings that are computed on the fly (Lample et al., 2016). These word and character representations are passed as input to the encoder which is a bidirectional long short-term memory (Bi-LSTM) (Hochreiter and Schmidhuber, 1997) layer that computes forward hidden state $\overrightarrow{\boldsymbol{h}_t}$ and backward hidden state $\overleftarrow{\boldsymbol{h}_t}$ per time step t in the input sequence. We then concatenate $\overrightarrow{\boldsymbol{h}_t}$ and $\overleftarrow{\boldsymbol{h}_t}$ to get final hidden state $\boldsymbol{h}_t = [\overrightarrow{\boldsymbol{h}_t}; \overleftarrow{\boldsymbol{h}_t}]$ at time step t.

Slot Filling (SF): For a given sentence $\boldsymbol{x}$ =

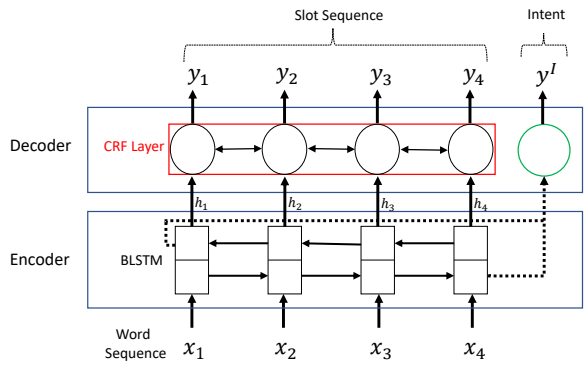


Figure 3: JOINT-SF-IC model.

$(x_1, \ldots, x_T)$ with T words, we use their respective hidden states $\boldsymbol{h} = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T)$ from the encoder (Bi-LSTM layer) to model tagging decisions $\boldsymbol{y}^S = (y_1, \ldots, y_T)$ jointly using a conditional random field (CRF) layer (Lample et al., 2016; Lafferty et al., 2001):

$$\boldsymbol{y}^S = \underset{\boldsymbol{y} \in \mathcal{Y}^S}{\operatorname{argmax}} f_S(\boldsymbol{h}, \boldsymbol{x}, \boldsymbol{y}), \tag{3}$$

where $\mathcal{Y}^S$ is the set of all possible slot sequences, and f_S is the CRF decoding function.

Intent Classification (IC): Based on the hidden states from the encoder (Bi-LSTM layer), we use the last forward hidden state $\overrightarrow{\boldsymbol{h}_T}$ and last backward hidden state $\overleftarrow{\boldsymbol{h}_1}$ to compute the moment $\boldsymbol{h}^I = [\overrightarrow{\boldsymbol{h}_T}; \overleftarrow{\boldsymbol{h}_1}]$ which can be regarded as the representation of the entire input sentence. Lastly, the intent y^I of the input sentence is predicted by feeding $\boldsymbol{h}^I$ into a fully-connected layer with softmax activation function to generate the prediction for each intent:

$$y^I = \text{softmax}(\boldsymbol{W}^I_{hy} \cdot \boldsymbol{h}^I + b), \tag{4}$$

where y^I is the prediction label, $\boldsymbol{W}^I_{hy}$ is a weight matrix and b is a bias term.

Joint Optimization: As our decoder models a joint task of SF and IC, we define the loss $\mathcal{L}$ as a weighted sum of individual losses which can be plugged into $\mathcal{L}^i_j$ in Equation 1:

$$\mathcal{L}_{\text{task}} = w_{SF} * \mathcal{L}_{\text{SF}} + w_{IC} * \mathcal{L}_{\text{IC}}, \tag{5}$$

where $\mathcal{L}_{\text{SF}}$ is the cross-entropy loss based on probability of the correct tag sequence (Lample et al., 2016), $\mathcal{L}_{\text{IC}}$ is the cross-entropy loss based on the predicted and true intent distributions (Liu et al., 2017) and w_{SF}, w_{IC} are hyperparameters to adjust the weights of the two loss components.

3 Experimental Setup

3.1 Dataset

We evaluate our proposed models for multi-domain joint slot filling and intent classification for spoken language understanding systems. We use the following benchmark dataset and large-scale Alexa dataset for evaluation, and we use classic intent accuracy and slot F1 as in Goo et al. (2018) as evaluation metrics.

Property	ATIS	Snips
Train set size	4,478	13,084
Dev set size	500	700
Test set size	893	700
#Slots	120	72
#Intents	21	7

Table 1: Statistics of the benchmark dataset.

Benchmark Dataset: We consider two widely used datasets ATIS (Tur et al., 2010) and Snips (Goo et al., 2018). The statistics of these datasets are shown in Table 1. For each dataset, we use the same train/dev/test set as Goo et al. (2018). ATIS is a single-domain (Airline Travel) dataset while Snips is a more complex multi-domain dataset due to the intent diversity and large vocabulary.

For initial experiments, we use ATIS and Snips as two tasks. For multi-domain experiments, we split Snips into three domains – *Music*, *Location*, and *Creative* based on its intents and treat each one as an individual task. Thus for this second set of experiments, we have four tasks (ATIS and Snips splits). Table 2 shows the new datasets obtained by splitting Snips. This new dataset allows us to introduce task groups (Section 4.1).

Alexa Dataset: We use live utterances spoken to the 90 Alexa skills with the highest traffic. These are categorized into 10 domains, based on assignments by the developers of the individual skills. Each skill is a task in the MTL setting, and each domain acts as a task group. Due to the limited annotated datasets for skills, we do not have val-

Dataset	Intent
Snips-creative	`search_creative_work` `rate_book`
Snips-music	`play_music` `add_to_playlist`

Snips-location	`get_weather`
	`book_restaurant`
	`search_screening_event`

Table 2: Snips after splitting based on intent.

Domain/Group	Skill Count	
	Train	Dev
Games, Trivia & Accessories	37	37
Smart Home	12	4
Music & Audio	8	8
Lifestyle	7	7
Education & Reference	7	7
Novelty & Humor	6	6
Health & Fitness	5	5
Food & Drink	3	3
Movies & TV	3	3
News	2	0
Total	**90**	**80**

Table 3: Statistics of the Alexa dataset.

idation sets for these 90 skills. Instead, we use another 80 popular skills that fall into the same domain groups as the 90 skills as the validation set to tune model parameters. Table 3 shows the statistics of the Alexa dataset based on domains. For training and validation sets, we keep approximately the same number of skills per group to make sure hyperparameters of adversarial training to be unbiased. We use the validation datasets to choose the hyperparameters for the baselines as well as our proposed models.

3.2 Baselines

We compare our proposed model with the following three competitive architectures for single-task joint slot filling (SF) and intent classification (IC), which have been widely used in prior literature:

- JOINTSEQUENCE: Hakkani-Tür et al. (2016) proposed a Bi-LSTM joint model for slot filling, intent classification, and domain classification.

- ATTENTIONBASED: Liu and Lane (2016) showed that incorporating an attention mechanism into a Bi-LSTM joint model can reduce errors on intent detection and slot filling.

- SLOTGATED: Goo et al. (2018) added a slot-gated mechanism into the traditional attention-based joint architecture, aiming to explicitly model the relationship between intent and slots, rather than implicitly modeling it with a joint loss.

We also compare our proposed model with two most related multi-task learning (MTL) architectures which can be treated as simplified versions of our parallel MTL architecture:

- PARALLEL[UNIV]: This model, proposed by Liu et al. (2017), uses a universe encoder that is shared across all tasks, and decoders are task-specific.

- PARALLEL[UNIV+TASK]: This model, also proposed by Liu et al. (2017), uses task-specific encoders in addition to the shared encoder. To ensure non-redundancy in features learned across shared and task-specific encoders, adversarial training and orthogonality constraints are incorporated.

3.3 Training Setup

All our proposed models are trained with back-propagation, and gradient-based optimization is performed using Adam (Kingma and Ba, 2015). In all experiments, we set the character LSTM hidden size to 64 and word embedding LSTM hidden size to 128. We use 300-dimension GloVe vectors (Pennington et al., 2014) for the benchmark datasets and in-house embeddings for the Alexa dataset, which are trained with Wikipedia data and live utterances spoken to Alexa. Character embedding dimensions and dropout rate are set to 100 and 0.5 respectively. Minimax optimization in adversarial training was implemented via the use of a gradient reversal layer (Ganin and Lempitsky, 2015; Liu et al., 2017). The models are implemented with the TensorFlow library (Abadi et al., 2016).

For benchmark data, the models are trained using an early-stop strategy with maximum epoch set to 50 and patience (i.e., number of epochs with no improvement on the dev set for both SF and IC tasks) to 6. In addition, benchmark dataset has varied size vocabularies across its datasets. To give equal importance to each of them, α_i^j (mentioned in Equation 1) is proportional to $1/n$, where n is the training set size of task j in group i. We are able to train on CPUs, due to the low values of n.

For Alexa data, the models are first trained on dev set to determine optimal hyperparameters and later using those optimal hyperparameters, models are trained on the training set and tested on the test set. α_i^j is here set to 1 as all skills have $10,000$ training utterances sampled from the respective developer-defined skill grammars (Kumar et al., 2017). Here, training was done using GPU-enabled EC2 instances (p2.8xlarge).

Our detailed training algorithm is similar to the one used by Collobert and Weston (2008) and Liu

Model	ATIS		Snips	
	Intent Acc.	Slot F1	Intent Acc.	Slot F1
JointSequence	92.6	94.3	96.9	87.3
AttentionBased	91.1	94.2	96.7	87.9
SlotGated	93.6	94.8	97.0	88.8
Joint-SF-IC	96.1	95.4	**98.0**	**94.8**
Parallel[Univ]	95.9	95.1	98.1	94.3
Parallel[Univ+Task]	**96.6**	**95.8**	97.6	94.5

Table 4: Results on benchmark datasets (ATIS and original Snips).

Model	ATIS		Snips-location	
	Intent Acc.	Slot F1	Intent Acc.	Slot F1
Joint-SF-IC	96.1	95.4	99.7	96.3
Parallel[Univ]	96.4	95.4	99.7	95.8
Parallel[Univ+Task]	96.2	95.5	99.7	96.0
Parallel[Univ+Group+Task]	96.9	95.4	99.7	96.5
Serial	97.2	**95.8**	**100.0**	96.5
Serial+Highway	96.9	95.7	**100.0**	**97.2**
Serial+Highway+Swap	**97.5**	95.6	99.7	96.0

Model	Snips-music		Snips-creative	
	Intent Acc.	Slot F1	Intent Acc.	Slot F1
Joint-SF-IC	**100.0**	93.1	100.0	96.6
Parallel[Univ]	**100.0**	92.1	100.0	95.8
Parallel[Univ+Task]	**100.0**	93.4	100.0	97.2
Parallel[Univ+Group+Task]	99.5	94.4	100.0	97.3
Serial	**100.0**	93.8	100.0	97.2
Serial+Highway	99.5	**94.8**	100.0	97.2
Serial+Highway+Swap	**100.0**	93.9	100.0	**97.8**

Table 5: Results on benchmark dataset (ATIS and subsets of Snips).

et al. (2016, 2017), where training is achieved in a stochastic manner by looping over the tasks. For example, an epoch involves the below four steps: 1) select a random skill; 2) select a random batch from the list of available batches for this skill; 3) update the model parameters by taking a gradient step w.r.t this batch; 4) Update the list of available batches for this skill by removing the current batch.

4 Experimental Results

4.1 Benchmark data

Table 4 shows the results on ATIS and the original version of the Snips dataset (as shown in Table 1). In the first four lines, ATIS and Snips are trained separately. In the last two lines (Parallel), they are treated as two tasks in the MTL setup. However, there are no task groups in this particular experiment, as each utterance belongs to either ATIS or Snips, and all utterances belong to the task universe. The Joint-SF-IC architecture with CRF layer performs better than all the three baseline models in terms of all evaluation metrics on both datasets, even after removing the slot-gate (Goo et al., 2018) and attention (Liu and Lane, 2016).

Model	Intent Acc.		Slot F1	
	Mean	Median	Mean	Median
Joint-SF-IC	93.36	95.90	79.97	85.23
Parallel[Univ]	**93.44**	95.50	**80.76**	86.18
Parallel[Univ+Task]	93.78	96.35	**80.49**	85.81
Parallel[Univ+Group+Task]	93.87	96.31	**80.84**	86.21
Serial	93.83	96.24	**80.84**	86.14
Serial+Highway	**93.81**	96.28	**80.73**	85.71
Serial+Highway+Swap	**94.02**	**96.42**	**80.80**	**86.44**

Table 6: Results on the Alexa dataset. Best results on mean intent accuracy and slot F1 values, and results that are not statistically different from the best model are marked in bold.

Learning universe features across both the datasets in addition to the task features help ATIS while performance on Snips degrades. This might be due to the fact that Snips is a multi-domain dataset, which in turn motivates us to split the Snips dataset (as shown in Table 2), so that the tasks in each domain (i.e., task group) may share features separately.

Table 5 shows results on ATIS and our split version of Snips. We now have four tasks: ATIS, Snips-location, Snips-music, and Snips-creative. Joint-SF-IC is our baseline that treats these four tasks independently. All other models process the four tasks together in the MTL setup. For the models introduced in this paper, we define two task groups: ATIS and Snips-location as one group, and Snips-music and Snips-creative as another. Our models, which use these groups, generally outperform the other MTL models (Parallel[Univ] and Parallel[Univ+Task]); especially the serial MTL architectures perform well.

4.2 Alexa data

Table 6 shows the results of the single-domain model and the MTL models on the Alexa dataset. The trend is clearly visible in these results compared to the results on the benchmark data. As Alexa data has more domains, there might not be many features that are common across all the domains. Capturing those features that are only common across a group became possible by incorporating task-group encoders. Serial+Highway+Swap yields the best mean intent accuracy. Parallel+Univ+Group+Task and Serial+Highway show statistically indistinguishable results. For slot filling, all MTL architecture achieve competitive results on mean Slot F1.

Model	Education	Food	Game	Health	Lifestyle	Movie	Music	News	Novelty	Smart Home
JOINT-SF-IC	95.89	90.60	96.29	92.80	93.84	67.50	93.51	90.05	95.00	89.58
PARALLEL[UNIV]	95.56	89.47	95.96	90.74	94.49	**74.40**	93.29	**90.90**	94.58	90.63
PARALLEL[UNIV+TASK]	95.99	91.10	96.50	93.60	94.46	68.40	94.45	88.60	94.93	90.66
PARALLEL[UNIV+GROUP+TASK]	96.17	91.23	**96.58**	93.92	94.33	68.10	94.56	87.15	95.00	91.06
SERIAL	96.11	90.77	96.44	94.04	**94.63**	69.07	94.59	87.30	94.92	90.89
SERIAL+HIGHWAY	96.04	91.70	96.45	94.10	92.71	68.67	94.86	87.90	95.03	91.41
SERIAL+HIGHWAY+SWAP	**96.20**	**91.80**	96.49	**94.16**	94.37	68.37	**94.94**	88.35	**95.08**	**91.64**

Table 7: Intent accuracy on different groups of the Alexa dataset.

Model	Education	Food	Game	Health	Lifestyle	Movie	Music	News	Novelty	Smart Home
JOINT-SF-IC	83.83	71.29	85.29	74.18	73.68	70.45	78.37	71.15	74.12	76.07
PARALLEL[UNIV]	84.75	**76.54**	**86.43**	**74.50**	71.85	72.32	78.46	70.53	**75.22**	76.48
PARALLEL[UNIV+TASK]	84.59	73.22	85.80	69.60	76.76	75.43	78.38	70.67	74.60	76.97
PARALLEL[UNIV+GROUP+TASK]	84.41	76.43	85.68	70.81	**78.24**	**76.74**	78.63	72.33	74.52	77.22
SERIAL	84.74	71.79	85.42	73.02	72.17	73.56	79.30	71.90	74.37	77.56
SERIAL+HIGHWAY	**85.20**	74.13	85.78	71.58	73.43	74.29	**80.12**	71.40	74.23	**77.75**
SERIAL+HIGHWAY+SWAP	84.93	74.87	**86.35**	72.38	72.02	72.09	78.86	**72.12**	74.49	77.69

Table 8: Slot F1 on different groups of the Alexa dataset.

Overall, on both benchmark data and Alexa data, our architectures with group encoders show better results than others. Specifically, the serial architecture with highway connections achieves the best mean Slot F1 of 94.8 and 97.2 on Snips-music and Snips-location respectively and median Slot F1 of 81.99 on the Alexa dataset. Swapping its feature hierarchy enhances its intent accuracy to 97.5 on ATIS. It also achieves the best/competitive mean and median values on both SF and IC on the Alexa dataset. This supports our argument that when we try to learn common features across all the domains (Liu et al., 2017), we might miss crucial features that are only present across a group. Capturing those task-group features boosts the performance of our unified model on SF and IC. In addition, when we attempt to learn three sets of features – task, task-universe, and task-group features – the serial architecture for feature learning helps. Specifically, when we have datasets from many domains, learning task features in the first stage and common features, i.e., task-universe and task-group features, in the second stage yields the best results. This difference is more clearly visible in the results of the large-scale Alexa data than that of the small-scale benchmark dataset.

5 Result Analysis

To further investigate the performance of different architectures, we present the intent accuracy and slot F1 values on different groups of Alexa utterances in Tables 7 and 8. For intent classification, SERIAL+HIGHWAY+SWAP achieves the best results on six domains, and PARALLEL[UNIV] achieves the best results on movie and news domains. Such a finding helps explain the reason why PARALLEL[UNIV] is significantly indistinguishable from SERIAL+HIGHWAY+SWAP on the Alexa dataset, which is shown in Table 6. PARALLEL[UNIV] outperforms MTL with group encoders when there is more information shared across domains. Examples of similar training utterances in different domains are "go back eight hour" and "rewind for eighty five hour" in a *News* skill; "to rewind the Netflix" in a *Smart Home* skill; and "rewind nine minutes" in a *Music* skill. The diverse utterance context in different domains could be learned through the universe encoder, which helps to improve the intent accuracy for these skills.

For the slot filling, each domain favors one of the four MTL architectures including PARALLEL[UNIV], PARALLEL[UNIV+GROUP+TASK], SERIAL+HIGHWAY, and SERIAL+HIGHWAY+SWAP. Such a finding is consistent with the statistically indistinguishable performance between different MTL architectures shown in Table 6. Tables 9 and 10 show a few utterances from different datasets in the *Smart Home* category that are correctly predicted after learning task-group features. General words like *sixty*, *eight*, *alarm* can have different slot types across different datasets. Learning features of *Smart Home* category help overcome such conflicts. However, a word in different tasks un-

der same domain can still have different slot types. For example, the first two utterances in Table 11, which are picked from the *Smart Home* domain, have different slot types *Name* and *Channel* for the word *one*. In such cases, there is no guarantee that learning group features can overcome the conflicts. This might be due to the fact that the groups are predefined and they do not always represent the real task structure. To tackle this issue, learning task structures with features jointly (Zhang et al., 2017) rather than relying on predefined task groups, would be a future direction. In our experimental settings, all the universe, task, and task-group encoders are instantiated with Bi-LSTM. An interesting area for future experimentation is to streamlining the encoders, e.g., adding additional bits to the inputs to the task encoder to indicate the task and group information, which is similar to the idea of using a special token as a representation of the language in a multilingual machine translation system (Johnson et al., 2017).

Utterance
turn/Other to/Other channel/Other sixty/**Name** eight/**Name**
go/Other to/Other channel/Other sixty/**VolumeLevel** seven/**Name**
turn/Other the/article alarm/**device** away/**device**

Table 9: Predictions (incorrect predictions are marked in red) from *Smart Home* domain by the PARALLEL[UNIV+TASK] architecture.

Utterance
turn/Other to/Other channel/Other sixty/**Channel** eight/**Channel**
go/Other to/Other channel/Other sixty/**Channel** seven/**Channel**
turn/Other the/article alarm/**security_system** away/**type**

Table 10: Predictions (correct predictions are marked in green) from *Smart Home* domain by the SERIAL+HIGHWAY+SWAP architecture.

Utterance
tune/Other to/Other the/Other bbc/Name one/**Name** station/Other
change/Other to/Other channel/Other one/**Channel**
score/Other sixty/Number one/**Number**
four/Answer one/**Answer**

Table 11: Training samples from different domains with different slot types for the word *one* (highlighted in blue).

6 Related Work

Multi-task learning (MTL) aims to learn multiple related tasks from data simultaneously to improve the predictive performance compared with learning independent models. Various MTL models have been developed based on the assumption that all tasks are related (Argyriou et al., 2007; Negahban and Wainwright, 2008; Jalali et al., 2010). To tackle the problem that task structure is usually unclear, Evgeniou and Pontil (2004) extended support vector machine for single task learning in multi-task scenario by penalizing models if they are too far from a mean model. In (Xue et al., 2007) a Dirichlet process prior was introduced to automatically identify subgroups of related tasks. Passos et al. (2012) developed a nonparametric Bayesian model to learn task subspaces and features jointly.

On the other hand, with the advent of deep learning, MTL with deep neural networks have been successfully applied to different applications (Zhang et al., 2018; Masumura et al., 2018; Fares et al., 2018; Guo et al., 2018). Recent work on multi-task learning considers different sharing structures e.g. only sharing at lower layers (Søgaard and Goldberg, 2016) and introduces private and shared subspaces (Liu et al., 2016, 2017). Liu et al. (2017) incorporated adversarial loss and orthogonality constraints into the overall training object, which helps in learning task-specific and task-invariant features non-redundantly. However, they rarely explore task structures, which might contain crucial features that are only present across a group of tasks. Our work is motivated by utilizing task structure and appropriately encoding structure information in deep neural architectures.

7 Conclusions

We proposed a series of end-to-end multi-task learning architectures, in which task, task-group and task-universe features are learned non-redundantly. We further explored learning these features in parallel and serial MTL architectures. Our MTL models obtain state-of-the-art performance on the ATIS and Snips datasets. Experimental results on a large-scale Alexa dataset show the effectiveness of adding task-group encoders into both parallel and serial MTL networks.

Acknowledgments

We thank Lambert Mathias for providing insightful feedback and Sandesh Swamy for preparing the Alexa test dataset. We also thank our team members as well as the anonymous reviewers for their valuable comments.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A system for large-scale machine learning. In *USENIX Symposium on OSDI*, pages 265–283.

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. 2007. Multi-task feature learning. In *NIPS*, pages 41–48.

Steffen Bickel, Jasmina Bogojeska, Thomas Lengauer, and Tobias Scheffer. 2008. Multi-task learning for HIV therapy screening. In *ICML*, pages 56–63.

Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. 2016. Domain separation networks. In *NIPS*, pages 343–351.

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

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, pages 160–167.

Theodoros Evgeniou and Massimiliano Pontil. 2004. Regularized multi–task learning. In *SIGKDD*, pages 109–117.

Xing Fan, Emilio Monti, Lambert Mathias, and Markus Dreyer. 2017. Transfer learning for neural semantic parsing. In *ACL-RepL4NLP*, pages 48–56.

Murhaf Fares, Stephan Oepen, and Erik Velldal. 2018. Transfer and multi-task learning for noun–noun compound interpretation. In *EMNLP*, pages 1488–1498.

Yaroslav Ganin and Victor Lempitsky. 2015. Unsupervised domain adaptation by backpropagation. In *ICML*, pages 1180–1189.

Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. 2018. Slot-gated modeling for joint slot filling and intent prediction. In *ACL(2)*, pages 753–757.

Han Guo, Ramakanth Pasunuru, and Mohit Bansal. 2018. Soft layer-specific multi-task summarization with entailment and question generation. In *ACL(1)*, pages 687–697.

Dilek Hakkani-Tür, Gökhan Tür, Asli Celikyilmaz, Yun-Nung Chen, Jianfeng Gao, Li Deng, and Ye-Yi Wang. 2016. Multi-domain joint semantic frame parsing using bi-directional RNN-LSTM. In *Interspeech*, pages 715–719.

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

Ali Jalali, Sujay Sanghavi, Chao Ruan, and Pradeep K Ravikumar. 2010. A dirty model for multi-task learning. In *NIPS*, pages 964–972.

Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. 2017. Googles multilingual neural machine translation system: Enabling zero-shot translation. *TACL*, 5:339–351.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICLR*.

Anjishnu Kumar, Arpit Gupta, Julian Chan, Sam Tucker, Bjorn Hoffmeister, Markus Dreyer, Stanislav Peshterliev, Ankur Gandhe, Denis Filiminov, Ariya Rastrow, Christian Monson, and Agnika Kumar. 2017. Just ASK: Building an Architecture for Extensible Self-Service Spoken Language Understanding. In *NIPS workshop on conversational AI*.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *NAACL-HLT*, pages 260–270.

Bing Liu and Ian Lane. 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. In *INTERSPEECH*, pages 685–689.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2016. Recurrent neural network for text classification with multi-task learning. In *IJCAI*, pages 2873–2879.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2017. Adversarial multi-task learning for text classification. In *ACL(1)*, pages 1–10.

Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2016. Multi-task sequence to sequence learning. In *ICLR*.

Ryo Masumura, Yusuke Shinohara, Ryuichiro Higashinaka, and Yushi Aono. 2018. Adversarial training for multi-task and multi-lingual joint modeling of utterance intent classification. In *EMNLP*, pages 633–639.

Sahand Negahban and Martin J Wainwright. 2008. Joint support recovery under high-dimensional scaling: Benefits and perils of $\ell_{1,\infty}$-regularization. In *NIPS*, pages 1161–1168.

Alexandre Passos, Piyush Rai, Jacques Wainer, and Hal Daumé III. 2012. Flexible modeling of latent task structures in multitask learning. In *ICML*, pages 1283–1290.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *EMNLP*, pages 1532–1543.

Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *ACL(2)*, pages 231–235.

Gokhan Tur, Dilek Hakkani-Tür, and Larry Heck. 2010. What is left to be understood in ATIS? In *SLT workshop*, pages 19–24.

Ya Xue, Xuejun Liao, Lawrence Carin, and Balaji Krishnapuram. 2007. Multi-task learning for classification with dirichlet process priors. *JMLR*, 8(Jan):35–63.

Honglun Zhang, Liqiang Xiao, Wenqing Chen, Yongkun Wang, and Yaohui Jin. 2018. Multi-task label embedding for text classification. In *EMNLP*, pages 4545–4553.

Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2017. Aspect-augmented adversarial networks for domain adaptation. *TACL*, 5:515–528.

Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. 2014. Facial landmark detection by deep multi-task learning. In *ECCV*.