SNR: Sub-Network Routing for Flexible Parameter Sharing in Multi-task Learning

Jiaqi Ma[1*] **Zhe Zhao**[2] **Jilin Chen**[2] **Ang Li**[3] **Lichan Hong**[2] **Ed H. Chi**[2]
[1]School of Information, University of Michigan, Ann Arbor [2]Google AI [3]DeepMind
[1]jiaqima@umich.edu [2,3]{zhezhao, jilinc, anglili, lichan, edchi}@google.com

Abstract

Machine learning applications, such as object detection and content recommendation, often require training a single model to predict multiple targets at the same time. Multi-task learning through neural networks became popular recently, because it not only helps improve the accuracy of many prediction tasks when they are related, but also saves computation cost by sharing model architectures and low-level representations. The latter is critical for real-time large-scale machine learning systems.

However, classic multi-task neural networks may degenerate significantly in accuracy when tasks are less related. Previous works (Misra et al. 2016; Yang and Hospedales 2016; Ma et al. 2018) showed that having more flexible architectures in multi-task models, either manually-tuned or soft-parameter-sharing structures like gating networks, helps improve the prediction accuracy. However, manual tuning is not scalable, and the previous soft-parameter sharing models are either not flexible enough or computationally expensive.

In this work, we propose a novel framework called Sub-Network Routing (SNR) to achieve more flexible parameter sharing while maintaining the computational advantage of the classic multi-task neural-network model. SNR modularizes the shared low-level hidden layers into multiple layers of sub-networks, and controls the connection of sub-networks with learnable latent variables to achieve flexible parameter sharing. We demonstrate the effectiveness of our approach on a large-scale dataset YouTube8M. We show that the proposed method improves the accuracy of multi-task models while maintaining their computation efficiency.

Introduction

In recent years, neural network based multi-task learning (Caruna 1993; Caruana 1998) has been successfully applied to a variety of real-world applications such as real-time object detection (Girshick 2015; Ren et al. 2015) and online recommender systems (Bansal, Belanger, and McCallum 2016; Ma et al. 2018). Given a single input, these systems usually predict multiple targets (or categories) at the same time. They often have low-latency requirement at the serving time. For example, a movie recommender system may need to predict both the probability of a user clicking a movie and that of a user liking watching a movie, so as to decide whether or not to recommend the movie in milliseconds. The many concurrent tasks, the low latency requirement, and the large exploration space of user and movie combination make efficient multi-task learning highly desirable.

Figure 1(a) provides an illustration of a classic yet widely-used multi-task learning model (Caruana 1998), which we call Shared-Bottom (SB) model. This model consists of several large low-level layers (Shared-Bottom) that are shared by all tasks, and several small task-specific high-level layers built on top of the shared layers. Compared to having a separate model for each task, the SB model learns a joint representation of many related tasks which can not only improve its model accuracy, but also save computation cost at serving time by sharing low-level network layers.

Despite many successful use cases in multi-task learning, the classic SB model is known to suffer from significant degeneration in accuracy when tasks are unrelated to each other (Ma et al. 2018). Compared to single-task models, the SB model introduces inductive bias into the shared low-level layers. When tasks are unrelated, the inductive biases in different tasks will have conflicts and hurt the model accuracy. A straightforward solution to this problem is to try multi-task models as well as single-task models, *i.e.*, use multi-task models for related tasks and use single-task models for unrelated tasks. Another solution is to manually tune the network architecture to allow flexible parameter sharing, *i.e.*, sharing more layers for highly related tasks and less layers for less related ones. Indeed, Misra et al. (2016) showed that different multi-task architectures are required to work well for two different related pairs of tasks. Note that both solutions rely on the knowledge of task relatedness. Despite the effort of several prior works (Baxter 2000; Ben-David, Gehrke, and Schuller 2002; Ben-David and Schuller 2003), efficiently measuring task relatedness in real-world data still remains an open problem. So, for both aforementioned solutions, we need to manually tune model structures by training and optimizing directly on the task accuracies. This usually does not scale for large-scale multi-task problems.

A scalable way to address the conflict problem is to design a multi-task model with a flexible parameter sharing architecture, e.g. gating structures, that can adapt to different representation needs and level of task relatedness (Misra et

[*]Work done while interning at Google.

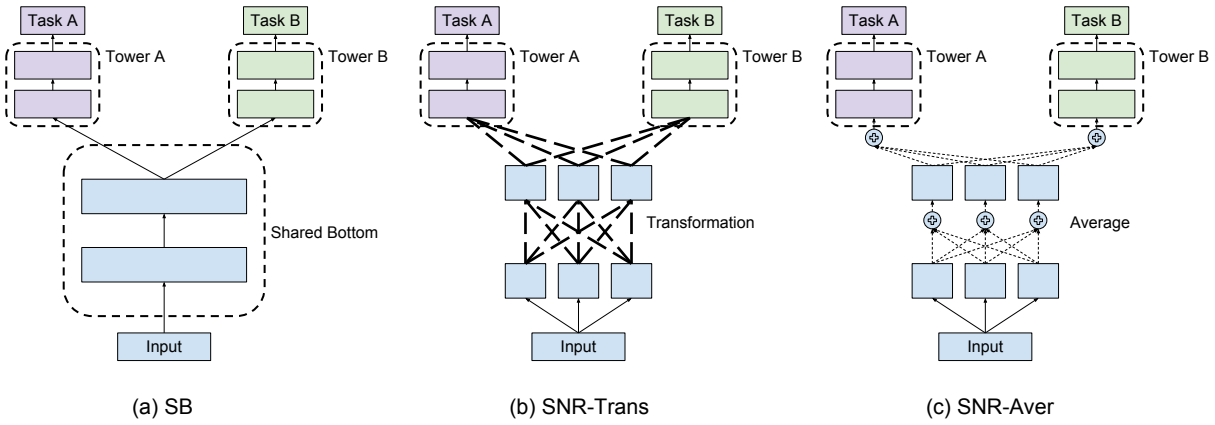

Figure 1: Multi-task Models. (a) Shared-Bottom (SB) model: the conventional multi-task neural network model. (b) Sub-Network Routing with Transformation (SNR-Trans) model: the shared layers are split into sub-networks and the connection (dashed line) between the sub-networks is a transformation matrix multiplied by a scalar latent variable. (c) Sub-Network Routing with Average (SNR-Aver) model: the shared layers are split into sub-networks and the connection (dashed line) between the sub-networks is a weighted average with scalar latent variables as weights.

al. 2016; Yang and Hospedales 2016; Ma et al. 2018). However, prior approaches often are either of limited flexibility, or flexible but computationally expensive due to incorporating too many substructures. For example, on the limited flexibility side, Ma et al. (2018) split the shared layers into one layer of sub-networks, and allowed only limited combinations of sharing structures. On the other hand, on the flexible but expensive side, Misra et al. (2016) built a multi-task model by connecting internal layers of single-task models, making the model as costly as multiple single-task models; Yang and Hospedales (2016) used a computationally expensive tensor factorization technique.

We propose a novel framework called Sub-Network Routing (SNR) to achieve more flexible parameter sharing while maintaining the computational advantage of the SB model. SNR framework modularizes the shared low-level layers into parallel sub-networks and learns their connections. Modularization with sub-networks is known to improve the trainability of multi-task models (Ma et al. 2018). The connectivity between two sub-networks is controlled by a binary variable, which we call a coding variable. With several layers of sub-networks and different coding variables, we can model a large set of sharing architectures in multi-task models. The computational advantage of the SB model is also maintained because, the related tasks can utilize the same sub-networks. Depending on how we connect sub-networks, we designed two types of connections in the SNR framework, namely SNR-Trans and SNR-Aver. SNR-Trans (shown in Figure 1(b)) uses matrix transformations to transform embeddings from lower-level sub-networks to higher-level sub-network; SNR-Aver (shown in Figure 1(c)) takes a weighted average of embeddings from lower-level sub-networks to high-level sub-networks. Our framework is also related to the area of Neural Architecture Search (NAS) (Zoph and Le 2016) in the sense we are searching for a neural architecture that works best for our tasks at hand. Our purpose of architecture search is for flexible parameter sharing in multi-task learning. Therefore we can encode the architecture space simply by the coding variables in the aforementioned SNR frameworks. With modularization, we also have a tradeoff between the flexibility of architecture space and the difficulty of architecture search.

The next question is how to efficiently learn the connections between sub-networks. We model the coding variables as latent random variables from parameterized distributions. The distribution parameters can be trained by gradient-based optimization together with the multi-task model parameters using the reparameterization trick (Kingma and Welling 2013; Rezende, Mohamed, and Wierstra 2014; Louizos, Welling, and Kingma 2017). At serving time, we use a deterministic estimator derived from the learned distribution to get the serving coding variables. This method shares insights with recent works (Pham et al. 2018; Liu, Simonyan, and Yang 2018) of accelerating NAS, where they showed that reusing model parameters across different network architecture samples, and joint learning architectures and model parameters can significantly speed up the process of NAS. The latent variable method also provides additional benefits with sparsity priors or penalties. We incorporate a technique of L0 regularization for neural networks (Louizos, Welling, and Kingma 2017) to further improve the accuracy of SNR-Trans under limited serving model size.

We evaluate our method on a large public video dataset, YouTube8M (Abu-El-Haija et al. 2016). This dataset consists of 6.1 million of YouTube video IDs, each with (multiple) labels from a vocabulary of more than $3,000$ entities. The input features of each video ID are pre-computed visual and audio features from the corresponding video. The $3,000+$ label classes are organized into 24 top categories. We construct a multi-task learning dataset by treating the top categories as tasks so each task is a multi-label classification problem. Our experiment indicates that both SNR-Trans and

SNR-Aver significantly outperform several baseline multi-task models. With L0-regularization, we further reduce the serving model size of SNR-Trans by up to 11%.

The rest of this paper is organized as follows: Section 2 reviews related work. Section 3 introduces the proposed approach in details. Section 4 evaluates the proposed models on YouTube8M dataset. Finally, Section 5 concludes the paper.

Related Work

Flexible Parameter Sharing in Multi-task Learning

Several related works have been proposed to improve multi-task learning when tasks are less related. Duong et al. (2015) split the multi-task model into two single-task models and added an L2 constraint between the difference of their model parameters. The Cross-Stitch model (Misra et al. 2016) also split the multi-task model into single-task models and concatenated the low-level layers of different single-task models weighted by learnable parameters. Yang and Hospedales (2016) used a tensor factorization model to generate hidden-layer parameters for each task. All of the above methods are not designed to maintain the computational advantage in the classic SB model. The model in Duong et al. (2015) is specifically designed for two-task scenario and cannot directly generalize to many-task scenario.

The closest work along this line is the MMoE model Ma et al. (2018), which split the shared low-level layers into sub-networks (called experts) and used different gating networks for different tasks to utilize different sub-networks. Our approach generalizes this idea into a more flexible form. Our approach also allows sparse connections among sub-networks, which further improves the serving computational efficiency.

Neural Architecture Search

Neural Architecture Search (NAS) (Zoph and Le 2016; Zoph et al. 2017; Real et al. 2018; Pham et al. 2018; Liu, Simonyan, and Yang 2018) is an emerging area of methods that automatically design neural architectures for a given task by reinforcement learning or evolution strategy. Our approach searches for a multi-task model architecture that can both alleviate the task conflicts and maintain the computation efficiency at serving time, which can be viewed as a special case of NAS. We specifically care about the parameter sharing problem and have a simpler architecture space within the proposed SNR framework.

The very first NAS method (Zoph and Le 2016) used a double-loop approach to search model architectures: in the outer-loop, an RNN-based controller generates model architectures and is trained with reinforcement learning rewarded by the accuracy of the generated architectures; in the inner-loop, the generated architecture is trained on the target task. As can be imagined, this process is very computational expensive. Several recent works (Pham et al. 2018; Liu, Simonyan, and Yang 2018) have been proposed for efficient NAS by merging the double-loop process and learning the architecture and model parameters simultaneously. Our approach shares similar insights with these efficient NAS methods in the sense that we learn the architecture and model parameters together. However, as we have a relatively simpler architecture space, we can treat the coding variables as latent random variables and use a simple parametrized distribution (a continuous relaxation of Bernoulli distribution) as the policy to generate the architectures, which is more efficient to train.

Approach

In this section, we introduce the proposed approach in details.

Sub-Network Modularization and Routing

We aim to improve multi-task learning models by flexible parameter sharing, where we want to make more related tasks share more model parameters and less related tasks share fewer model parameters. Several previous works (Misra et al. 2016; Ma et al. 2018) approached this goal by splitting the whole neural network model into some forms of sub-networks, allowing different tasks to utilize different sub-networks. Ma et al. (2018) showed that such modularization benefits the trainability of multi-task models. In this work, we further extend this idea by modularizing each of the shared low-level layers in the classic SB model into parallel sub-networks (see Figure 1 (b) and (c)).

Based on such modularization, we can have different extents of parameter sharing in the multi-task model by controlling the connection routing among the sub-networks of different layers. We call this framework Sub-Network Routing (SNR). By exploring a large set of connection patterns, we hope to find a good architecture that shares the sub-networks as much as possible so that it can both alleviate the task conflicts and maintain the computation efficiency at serving time.

There are many choices of the routing between sub-networks. We implement two natural types: the first type, which we call SNR-Trans (shown in Figure 1(b)), is to use a transformation matrix multiplied by a scalar coding variable; the second type, which we call SNR-Aver (shown in Figure 1(c)), is to have a weighted average with scalar coding variables as weights.

Suppose there are two subsequent layers of sub-networks, and the lower-level layer has 3 sub-networks and the higher-level layer has 2 sub-networks. Let $\boldsymbol{u_1}, \boldsymbol{u_2}, \boldsymbol{u_3}$ be the outputs of the lower-level sub-networks and let $\boldsymbol{v_1}, \boldsymbol{v_2}$ be the inputs of the higher-level sub-networks. Then SNR-Trans can be formulated as

$$\begin{bmatrix} \boldsymbol{v_1} \\ \boldsymbol{v_2} \end{bmatrix} = \begin{bmatrix} z_{11}\boldsymbol{W_{11}} & z_{12}\boldsymbol{W_{12}} & z_{13}\boldsymbol{W_{13}} \\ z_{21}\boldsymbol{W_{21}} & z_{22}\boldsymbol{W_{22}} & z_{23}\boldsymbol{W_{23}} \end{bmatrix} \begin{bmatrix} \boldsymbol{u_1} \\ \boldsymbol{u_2} \\ \boldsymbol{u_3} \end{bmatrix}$$

where $\boldsymbol{W_{ij}}$ is a transformation matrix from the jth lower-level sub-network to the ith higher-level sub-network and $\boldsymbol{z}$ represents the coding variables (a group of binary variables controlling the connection).

Similarly, SNR-Aver can be formulated as

$$\begin{bmatrix} \boldsymbol{v_1} \\ \boldsymbol{v_2} \end{bmatrix} = \begin{bmatrix} z_{11}\boldsymbol{I_{11}} & z_{12}\boldsymbol{I_{12}} & z_{13}\boldsymbol{I_{13}} \\ z_{21}\boldsymbol{I_{21}} & z_{22}\boldsymbol{I_{22}} & z_{23}\boldsymbol{I_{23}} \end{bmatrix} \begin{bmatrix} \boldsymbol{u_1} \\ \boldsymbol{u_2} \\ \boldsymbol{u_3} \end{bmatrix}$$

where $\boldsymbol{I_{ij}}$ is an identity matrix for all i, j.

If we hold the two models having the same number of model parameters (and thus similar model size), SNR-Trans has more model parameters in the connection while SNR-Aver has more budget of model parameters in the sub-networks. Although it is hard to tell the pros and cons in terms of model representation for the two routing schemes, we argue that when applying sparse connections among the sub-networks, it is easier to reduce the model parameters in SNR-Trans than in SNR-Aver, which could benefit model serving.

Connecting to Manual Tuning and NAS

Manually tuning the network architecture is equivalent to manually setting the coding variables $\boldsymbol{z}$, where $z_{ij} \in \{0, 1\}$. For example, let's suppose $\boldsymbol{v_1}, \boldsymbol{v_2}$ represent the outputs of sub-networks to two tasks and there is only one layer of hidden sub-networks $\boldsymbol{u_1}, \boldsymbol{u_2}, \boldsymbol{u_3}$. If we set all elements of $\boldsymbol{z}$ as 1, then the corresponding model degenerates to the classic shared-bottom model. If we set $z_{11} = z_{22} = 1$ and all other elements of $\boldsymbol{z}$ as 0, then the model degenerates to two small single-task models.

If we have infinite computation resource, manual tuning perhaps allows us to find the best architectures that are pareto optimal in terms of prediction accuracy and computation efficiency. However, when there are many tasks and many hidden-layers in a multi-task model, the search space for coding variable $\boldsymbol{z}$ becomes exponentially large: $2^{|\boldsymbol{z}|}$, where $|\boldsymbol{z}|$ denotes the number of elements in $\boldsymbol{z}$. As a result, manual tuning could be very inefficient when we lack the knowledge of task relatedness.

Inspired by the efficient NAS methods mentioned in the related work, we turn to automatically learn the connection routing within the fairly flexible SNR framework. In our scenario, our goal is to efficiently explore different multi-task architectures to achieve flexible parameter sharing across tasks rather than general neural architecture search. And we have a relative constrained architecture space encoded by $\boldsymbol{z}$. We propose to model the coding variables $\boldsymbol{z}$ as latent random variables from parameterized distributions, and learn the distribution parameters and model parameters simultaneously.

Learning the Architecture with Latent Variables

In this section, we formulate the problem of learning the connection routing in the SNR framework with latent variables.

Let $f(\cdot; \boldsymbol{W}, \boldsymbol{z})$ be a neural network model parameterized by weights $\boldsymbol{W}$ and coding variables $\boldsymbol{z}$ where the coding variables is supposed to be drawn from a latent policy distribution $p(\boldsymbol{z}; \boldsymbol{\pi})$ parametrized by $\boldsymbol{\pi}$. Given a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$, where $\boldsymbol{x}_i$ is a feature vector of the sample i and $\boldsymbol{y}_i$ is the corresponding label vector containing the labels of multiple tasks, the problem of learning coding variables and model parameters can be formulated as an optimization problem as follows:

$$\min_{\boldsymbol{W}, \boldsymbol{\pi}} \boldsymbol{E}_{\boldsymbol{z} \sim p(\boldsymbol{z}; \boldsymbol{\pi})} \frac{1}{N} \sum_{i=1}^{N} L(f(\boldsymbol{x}_i; \boldsymbol{W}, \boldsymbol{z}), \boldsymbol{y}_i), \tag{1}$$

where L is a loss function.

We simply use Bernoulli distributions as our policy for the coding variables $\boldsymbol{z}$. That is $z_i \sim Bern(\pi_i)$ for all elements z_i in $\boldsymbol{z}$. The coding variables can also be viewed as latent variables in a graphical model perspective. This latent variable method can be applied to both SNR-Trans and SNR-Aver.

The objective function in Eq. 1 is non-differentiable *w.r.t.* the distribution parameters $\boldsymbol{\pi}$, but the gradients can be estimated by gradient estimator REINFORCE (Williams 1992). Louizos, Welling, and Kingma (2017) further proposed a relaxation to smooth such objective functions so that we can directly calculate the gradients of the distribution parameters. We adopt this relaxation method in our model.

The main idea of the method in (Louizos, Welling, and Kingma 2017) is to first find a continuous random variable $s \sim q(s; \phi)$ and compute the coding variable z as a hard-sigmoid of s, *i.e.*,

$$z = g(s) = \min(1, \max(0, s)).$$

Then, Eq. 1 becomes

$$\min_{\boldsymbol{W}, \boldsymbol{\pi}} \boldsymbol{E}_{\boldsymbol{s} \sim q(\boldsymbol{s}; \boldsymbol{\phi})} \frac{1}{N} \sum_{i=1}^{N} L(f(\boldsymbol{x}_i; \boldsymbol{W}, g(\boldsymbol{s})), \boldsymbol{y}_i). \tag{2}$$

The objective function 2 is reformulated using the reparameterization trick (Kingma and Welling 2013; Rezende, Mohamed, and Wierstra 2014; Louizos, Welling, and Kingma 2017) as

$$\min_{\boldsymbol{W}, \boldsymbol{\pi}} \boldsymbol{E}_{\boldsymbol{\epsilon} \sim r(\boldsymbol{\epsilon})} \frac{1}{N} \sum_{i=1}^{N} L(f(\boldsymbol{x}_i; \boldsymbol{W}, g(h(\boldsymbol{\phi}, \boldsymbol{\epsilon}))), \boldsymbol{y}_i), \tag{3}$$

where $\boldsymbol{\epsilon}$ is a noise random variable, $r(\boldsymbol{\epsilon})$ is a parameter-free noise distribution, and $h(\cdot, \cdot)$ is a deterministic and differentiable transformation of $\boldsymbol{\phi}$ and $\boldsymbol{\epsilon}$.

In practice, the hard concrete distribution (Louizos, Welling, and Kingma 2017) is used, which is defined (element-wise) as follows,

$$u \sim U(0, 1), s = \text{sigmoid}((\log(u) - \log(1 - u) + \log(\alpha))/\beta)$$
$$\bar{s} = s(\zeta - \gamma) + \gamma, z = \min(1, \max(\bar{s}, 0)),$$

where u is a uniform random variable, $\log(\alpha)$ is a learnable distribution parameter, and β, γ, ζ are all hyper-parameters.

More details about the hard concrete distribution can be found at (Louizos, Welling, and Kingma 2017).

Applying L0 Regularization on Latent Variables

Another benefit of this latent variable model is that we can add priors and regularizations on the latent variables. For example, Louizos, Welling, and Kingma (2017) provided a way to learn sparse latent variables through L0 regularization. The sparsity structure is also desirable in our scenario because this allows the multi-task model to be computationally efficient.

The L0 regularization on the latent variables $\boldsymbol{z}$ can be formulated as

$$\boldsymbol{E}_{\boldsymbol{z} \sim p(\boldsymbol{z}; \boldsymbol{\pi})} ||\boldsymbol{z}||_0 = \sum_{i=1}^{|\boldsymbol{z}|} p(z_i = 1; \pi_i).$$

With the relaxation from $\boldsymbol{z}$ to continuous random variables $\boldsymbol{s}$, we have

$$p(z_i = 1; \pi_i) = 1 - Q(s_i < 0; \phi_i),$$

where $Q(\cdot; \phi_i)$ is the cumulative distribution function of s_i. So the L0 regularization becomes

$$\boldsymbol{E}_{\boldsymbol{z} \sim p(\boldsymbol{z}; \boldsymbol{\pi})} ||\boldsymbol{z}||_0 = \sum_{i=1}^{|\boldsymbol{z}|} 1 - Q(s_i < 0; \phi_i).$$

The full objective function with L0 regularization is

$$\boldsymbol{E}_{\boldsymbol{\epsilon} \sim r(\boldsymbol{\epsilon})} \frac{1}{N} \sum_{i=1}^{N} L(f(\boldsymbol{x}_i; \boldsymbol{W}, g(h(\boldsymbol{\phi}, \boldsymbol{\epsilon}))), \boldsymbol{y}_i) + \lambda \sum_{j=1}^{|\boldsymbol{z}|} 1 - Q(s_j < 0; \phi_j),$$

where λ is a hyper-parameter (see the Experiment Setup section for more details).

Additional Details of Model Training and Serving

The whole model, including both model parameters $\boldsymbol{W}$ and latent variable distribution parameters $\log(\boldsymbol{\alpha})$, is trained by stochastic gradient based optimization. For each mini-batch in the forward pass, we first sample a group of uniform random variables $\boldsymbol{u}$, then calculate $\boldsymbol{z}$ to obtain the network architecture, and finally feed the input data into the model to compute the loss. The gradients *w.r.t.* $\boldsymbol{W}$ and $\log(\boldsymbol{\alpha})$ are calculated by back-propagation. Multiple samples of $\boldsymbol{u}$ can be drawn to reduce the variance of the gradient estimates but one sample of $\boldsymbol{u}$ per mini-batch works well in practice.

At serving time, the following estimator (Louizos, Welling, and Kingma 2017) is used for $\boldsymbol{z}$,

$$\hat{\boldsymbol{z}} = \min(1, \max(0, \text{sigmoid}(\log(\boldsymbol{\alpha}))(\zeta - \gamma) + \gamma)).$$

When $\text{sigmoid}(\log(\alpha_{ij}))(\zeta - \gamma) + \gamma < 0$, we will have $\hat{z}_{ij} = 0$ and the resulted model will be sparsely connected.

To reduce serving model parameter size in SNR-Aver model, we need to remove at least a whole sub-network from the model, which means we need to have $\hat{z}_{ij} = 0$ for all i to eliminate the jth sub-network. For SNR-Trans, however, any $\hat{z}_{ij} = 0$ will eliminate the corresponding W_{ij} from the model. So it's easier to reduce model parameters in SNR-Trans than in SNR-Aver.

Experiment

In this section, we conduct experiments on a public large-scale dataset, YouTube8M, to evaluate the effectiveness of the proposed models.

Experimental Setup

YouTube8M Dataset

We use YouTube8M (Abu-El-Haija et al. 2016) as our benchmark dataset to evaluate the effectiveness of the proposed methods. This dataset consists of 6.1 million of YouTube videos, each with (multiple) labels from a vocabulary of more than $3,000$ topical entities. The topical entities can be further grouped into 24 top-level topic categories.

To create a multi-task learning problem from the dataset, we treat each top-level topic category as a separate prediction task, so that each task is a multi-label classification problem. To ensure data quantity per task, we used the top 16 categories in data volume.

We use the training set provided in the original dataset as our training set, and split the original validation set into our own validation set and test set, because this dataset comes from a Kaggle competition and the original test set labels are hidden to the public.

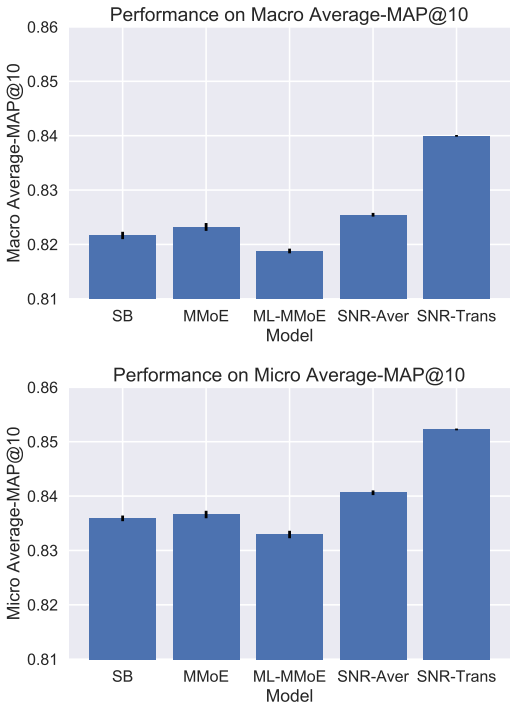

Figure 2: Accuracy of Best-Tuned Models. The bar chart shows the average test performance of the top 10 models selected by validation performance. The error bar indicates the standard error of the average test performance.

Methods

We compare five multi-task learning models in the experiments. As the main difference among the models lies in their shared low-level part, we fix the task-specific high-level part of all models as a one-layer fully-connected hidden layer with hidden size 16 for each task. ReLU activation is used whenever applicable. The implementation of the shared low-level part for each model is described as follows respectively,

SB : This model is the classic Shared-Bottom model where several low-level network layers are shared by all the tasks and each task has its own task-specific high-level layers built on top of the shared layers.

Shared part: Two fully-connected hidden layers with the hidden layer sizes as hyper-parameters to be tuned.

MMoE (Ma et al. 2018): This model splits the shared low-level layers into sub-networks and uses different gating networks for different tasks to utilize different sub-networks.

Shared part: One fully-connected hidden layer followed by a MMoE layer with 8 experts, each expert being a one-layer fully-connected sub-network. Both the hidden size of the first hidden layer and that of the sub-network are hyper-parameters to be tuned. The gating networks are linear transformations without hyper-parameters.

ML-MMoE : This model extends MMoE by adding multiple layers of sub-networks. The connections from the lower-level sub-networks to the higher-level sub-networks are also controlled by some gating networks. Each higher-level sub-network can be viewed as an internal task. All the gating networks share the same input, which is the input of the whole model.

Shared part: Two subsequent MMoE layers with each layer having 8 experts. Each expert is a one-layer fully-connected sub-network. The hidden size of the expert network is a tunable hyper-parameter.

SNR-Trans : This is the proposed Sub-Network Routing with Transformation model.

Shared part: Two subsequent transformation layers. The output size of each transformation matrix is tunable.

SNR-Aver : This is the proposed Sub-Network Routing with Averaging model.

Shared part: Two subsequent sub-network layers with each sub-network being a one-layer fully-connected network. The hidden size of each sub-network is tunable.

Evaluation Metrics

As each task in the YouTube8M dataset is a multi-label classification problem, we use Mean Average Precision (MAP) as the measurement of prediction accuracy for each task. Specifically, we use MAP@10 as our metric because most of the examples have fewer than 10 positive labels in each task. To evaluate the overall accuracy of the multi-task models, we compute an average of the MAP@10 of all tasks, which we will call it Average-MAP@10. As different tasks have different sample sizes, we calculate two types of Average-MAP@10 metrics: the first type directly calculates the mean of the MAP@10 scores of all the tasks, which we call Macro Average-MAP@10; the second type takes a weighted average of the MAP@10 scores weighted by the number of data examples in each task, which we call Micro Average-MAP@10.

Model Training and Hyper-parameter Tuning

All the models are trained using Adam (Kingma and Welling 2013) with learning rate as a tunable hyper-parameter. The batch size is fixed as 128. Early stopping is used on the validation set. Model size related hyper-parameters are tuned with grid search for the purpose of comparing model accuracy at different model size, and all other hyper-parameters are tuned with random search. The hidden sizes of the two shared hidden layers in SB are grid-searched from $\{256, 512, 1024, 2048\}$ and the hidden size hyper-parameters in other models are grid-searched in a way to approximately match the model size of SB. The L0 regularization parameter λ will have an impact on the serving model size so we grid-search it from $\{0.001, 0.0001, 0.00001\}$. The learning rates of all models are random-searched within $[0.00001, 0.1]$ in log-scale. The hyper-parameters for the hard concrete distribution used in L-Act and L-Param models are random-searched from the following range: $\beta \sim [0.5, 0.9], \gamma \sim [-1, -0.1], \zeta \sim [1.1, 2]$.

For each model size setting, we run 500 independent trials of random search on hyper-parameters unrelated to model size and select the top 10 models with best validation accuracy. Then we report the average and the standard error of testing MAPs from these top 10 models.

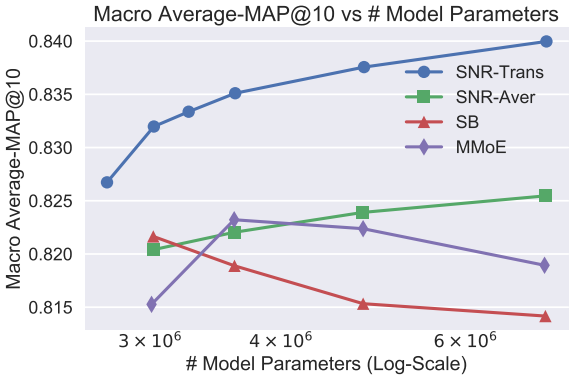

Figure 3: Macro Average-MAP@10 vs Model Size. The y-axis is the test performance on Macro Average-MAP@10. The x-axis is the number of total model parameters at training time. The total hidden size of first shared layer in each model is fixed as 2048 and the total hidden size of the second shared layer is varied to get different model sizes.

Results

Accuracy of Best-Tuned Models

We first show the accuracy of the best-tuned models using each method in Figure 2. We report the test accuracy on both Macro Average-MAP@10 and Micro Average-MAP@10. The differences between each pair of models on both metrics, except for MMoE vs SB on Micro Average-MAP@10, are significant with 0.05 significance level of two-sample t-test. The relative trends on both metrics are the same. So we will show the results of Macro Average-MAP@10 only in the remaining result analysis due to page limit.

As seen in Figure 2, SNR-Trans and SNR-Aver outperform all the baseline models. The ML-MMoE model performs surprisingly worse than all other models. One possible reason for this is that the stacked multi-gate structure may cause difficulty in optimization. There is also a design difficulty of selecting the input layer of gating networks that

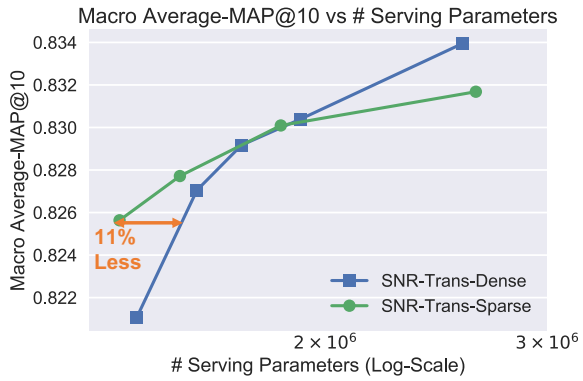

Figure 4: Macro Average-MAP@10 of SNR-Trans models with different model sizes and L0 regularization parameters. "SNR-Trans-Dense" indicates models that use a small L0 regularization parameter and the resulted models are dense; "SNR-Trans-Sparse" indicates models that use a large L0 regularization parameter and the resulted models are sparse. The sparse models can reduce the serving model size by up to 11% under certain serving constraint.

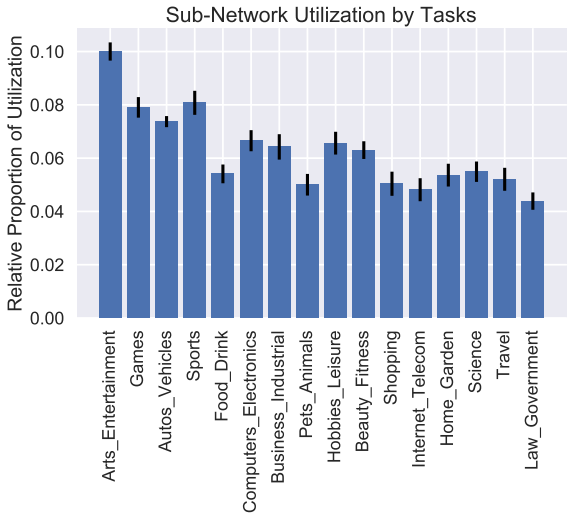

Figure 5: Average sub-network utilization by tasks in top performing SNR-Trans models. The tasks on x-axis are sorted by sample size in descending order from left to right. The y-axis is the average relative proportion of sub-networks utilized by the corresponding task.

lie between two layers of experts. As there are many internal outputs from the previous layer of experts, it is hard to decide which internal outputs should be used by the higher-level gating networks. In our implementation, we use the initial input features of the first layer as the input of all gating networks. However, the initial input features may not provide enough information about how to route in higher-level layers.

Accuracy vs Model Size

One thing which we notice during the hyper-parameter tuning is that, while the accuracy of baseline models has saturated as we increase the model size, the accuracy of SNR-Trans and SNR-Aver still slowly increases at the boundary of our hyper-parameter range.

Figure 3 shows the test accuracy of a sampled set of models with different model sizes. In this figure, we fix the total hidden size of the first shared layer in each model as 2048, and we vary the total hidden size of the second shared layer to get models with different model sizes. Note that the x-axis is the number of total model parameters at training time, which means the model parameters eliminated by L0 regularization in SNR models are also counted.

This result shows that the SNR method can train larger models better than baseline methods. This phenomenon aligns well with our hypothesis that modularization in multi-task learning could improve model trainability.

Accuracy of Sparse Models

While being able to train large models well is critical, in many large-scale online systems we have strict low-latency requirements on the serving model. So we also care about models with limited serving model sizes. Here we show that with a proper L0 regularization parameter λ, we can effectively reduce the serving model size of the SNR-Trans model under certain serving constraints.

We first observe that the value of the L0 regularization parameter λ has a direct impact on the learned architecture of SNR-Trans models. When λ is set to 0.00001, the learned coding variables z are almost all 1s, which means the learned architecture is densely connected. When λ is set to 0.0001, the learned architecture is sparse and the sparse model has significantly smaller serving model size than the dense model. Given the same training model size, the sparse model usually performs worse than the dense model. This result is not unexpected as λ controls a tradeoff between higher model capacity and smaller serving model size, with large λ shrinking the effective capacity for the model. However, when the serving model size is limited, the sparse model outperforms the dense model that has similar serving model size.

As shown in Figure 4, "SNR-Trans-Dense" indicates SNR-Trans models with L0 regularization parameter 0.00001 while "SNR-Trans-Sparse" indicates SNR-Trans models with L0 regularization parameter 0.0001. We fix the total hidden size in the first layer as 1024 and vary the hidden size of the second layer to get different model sizes. The trend in several other settings where we set hidden size of the first layer as other values is similar and we omit the figures due to page limit. The accuracy of the dense model drops very fast as the serving model size decreases. And the accuracy of the sparse model outperforms the dense model by a large margin given limited serving model size. In other words, we can reduce the serving model size by up to 11% while maintaining the same accuracy.

Analysis of Sub-Network Utilization

To better understand how the sub-networks are utilized

by different tasks in the sparse models, we further summarize the average relative proportion of sub-networks used by each task in 10 top performing sparse models, shown in Figure 5. The figure shows that the utilization of sub-networks is positively related to the sample size of the tasks[1]. This implies that, when we add a stronger L0 regularization on the model, the model will learn to allocate more capacity to the tasks with more data.

Conclusion

In this work, we introduce a flexible parameter sharing framework in multi-task learning, Sub-Network Routing (SNR). SNR is able to encode various types of multi-task model architectures and allows us to add a wide range of priors on the model structure. We propose a scalable multi-task architecture search solution by using latent variables to model the architecture coding variables, and learning the latent variables and model parameters simultaneously. We empirically show that the proposed methods outperform baseline multi-task models on a large-scale dataset YouTube8M. We further reduce the serving model size by applying L0 regularization on the latent variables.

References

Abu-El-Haija, S.; Kothari, N.; Lee, J.; Natsev, P.; Toderici, G.; Varadarajan, B.; and Vijayanarasimhan, S. 2016. Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*.

Bansal, T.; Belanger, D.; and McCallum, A. 2016. Ask the gru: Multi-task learning for deep text recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, 107–114. ACM.

Baxter, J. 2000. A model of inductive bias learning. *Journal of Artificial Intelligence Research* 12:149–198.

Ben-David, S., and Schuller, R. 2003. Exploiting task relatedness for multiple task learning. In *Learning Theory and Kernel Machines*. Springer. 567–580.

Ben-David, S.; Gehrke, J.; and Schuller, R. 2002. A theoretical framework for learning from a pool of disparate data sources. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 443–449. ACM.

Caruana, R. 1998. Multitask learning. In *Learning to learn*. Springer. 95–133.

Caruna, R. 1993. Multitask learning: A knowledge-based source of inductive bias. In *Machine Learning: Proceedings of the Tenth International Conference*, 41–48.

Duong, L.; Cohn, T.; Bird, S.; and Cook, P. 2015. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *ACL (2)*, 845–850.

Girshick, R. 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, 1440–1448.

Kingma, D. P., and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[1]The original data distribution of YouTube8M can be found at https://research.google.com/youtube8m/

Liu, H.; Simonyan, K.; and Yang, Y. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Louizos, C.; Welling, M.; and Kingma, D. P. 2017. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*.

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 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1930–1939. ACM.

Misra, I.; Shrivastava, A.; Gupta, A.; and Hebert, M. 2016. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3994–4003.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2018. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, 91–99.

Rezende, D. J.; Mohamed, S.; and Wierstra, D. 2014. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

Yang, Y., and Hospedales, T. 2016. Deep multi-task representation learning: A tensor factorisation approach. *arXiv preprint arXiv:1605.06391*.

Zoph, B., and Le, Q. V. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2017. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012* 2(6).