DSelect-k: Differentiable Selection in the Mixture of Experts with Applications to Multi-Task Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

The Mixture-of-experts (MoE) architecture is showing promising results in multi-task learning (MTL) and in scaling high-capacity neural networks. State-of-the-art MoE models use a trainable "sparse gate" to select a subset of the experts for each input example. While conceptually appealing, existing sparse gates, such as Top-k, are not smooth. The lack of smoothness can lead to convergence and statistical performance issues when training with gradient-based methods. In this paper, we develop DSelect-k: the first, continuously differentiable and sparse gate for MoE, based on a novel binary encoding formulation. Our gate can be trained using first-order methods, such as stochastic gradient descent, and offers explicit control over the number of experts to select. We demonstrate the effectiveness of DSelect-k in the context of MTL, on both synthetic and real datasets with up to 128 tasks. Our experiments indicate that MoE models based on DSelect-k can achieve statistically significant improvements in predictive and expert selection performance. Notably, on a real-world large-scale recommender system, DSelect-k achieves over 22% average improvement in predictive performance compared to the Top-k gate. We provide an open-source TensorFlow implementation of our gate[1].

1 Introduction

The Mixture of Experts (MoE) [14] is the basis of many state-of-the-art deep learning models. For example, MoE-based layers are being used to perform efficient computation in high-capacity neural networks and to improve parameter sharing in multi-task learning (MTL) [32, 22, 21]. While different MoE architectures are considered in the literature, two major components are needed in any MoE model: a set of experts (neural networks) and one or more trainable gates. A gate selects a combination of the experts that is specific to each input example. This allows experts to specialize in different partitions of the input space, which has the potential to improve the predictive performance and interpretability. In Figure 1(a), we show an example of a simple MoE architecture that can be used as a standalone learner or as a layer in a neural network.

The literature on the MoE has traditionally focused on softmax-based gates, in which all experts are assigned nonzero weights [17]. More recent works propose *sparse gates* that assign nonzero weights to only a small subset of the experts [1, 32, 28, 21]. Sparsity in gating can have various advantages, including better computational efficiency, interpretability, and improved statistical performance in certain settings. Existing sparse gates are not differentiable, and reinforcement learning algorithms are commonly used to train these gates [1, 28]. In an exciting work, Shazeer et al. [32] introduced a new sparse gate (Top-k gate) and proposed training it using stochastic gradient descent (SGD). The ability to train the gate using SGD is appealing because it allows for training the neural network in an

[1]The code is in the supplementary material. It will be made public if the paper is accepted.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

end-to-end fashion. However, the Top-k gate is not continuous, which can lead to convergence issues in SGD that in turn affect the statistical performance (as we demonstrate in our experiments).

In this paper, we introduce DSelect-k: the first, continuously differentiable and sparse gate for MoE. Given a user-specified parameter k, the gate selects at most k out of the n experts. This explicit control over sparsity leads to a cardinality-constrained optimization problem, which is computationally challenging. To circumvent this challenge, we propose a novel, unconstrained reformulation that is equivalent to the original problem. The reformulated problem uses a binary encoding scheme to implicitly enforce the cardinality constraint. We demonstrate that by carefully smoothing the binary encoding variables, the reformulated problem can be effectively optimized using first-order methods such as SGD.

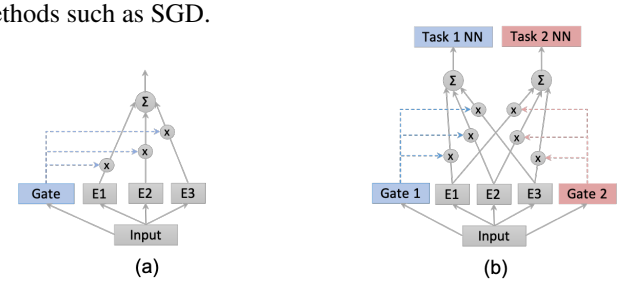

Figure 1: **(a)**: An example of a MoE that can be used as a standalone learner or layer in a neural network. "Ei" is the i-th expert. **(b):** A multi-gate MoE for learning two tasks simultaneously. "Task i NN" is a neural network which generates the output of Task i.

Our gate supports two mechanisms: *per-example gating* and *static gating*. The per-example gating is the typical gating technique used in MoE models, in which the weights assigned to the experts are a function of the input example [14, 32]. In static gating, a subset of experts is selected and the corresponding weights do not depend on the input [28]. Based on our experiments, each gating mechanism can outperform the other in certain settings. Thus, we study both mechanisms and advocate for experimenting with each.

MTL is an important area where MoE models in general, and our gate in particular, can be useful. The goal of MTL is to learn multiple tasks simultaneously by using a shared model. Compared to the usual single task learning, MTL can achieve better generalization performance through exploiting the relationships between tasks [4]. One of the key problems in MTL is how to share model parameters between tasks [30]. For instance, sharing parameters between unrelated tasks can potentially degrade performance. The multi-gate MoE [22] is a flexible architecture which allows for learning what to share between tasks. Figure 1(b) shows an example of a multi-gate MoE (in the simple case of two tasks). Here, each task has its own gate which adaptively decides whether to share experts with the other task. In our experiments, we study the effectiveness of our proposed gate in the context of the multi-gate MoE.

Contributions: On a high-level, our main contribution is DSelect-k: a new continuously differentiable and sparse gate for MoE, which can be directly trained using first-order methods. Our technical contributions can be summarized as follows. **(i)** The gate selects (at most) k out of the n experts, where k is a user-specified parameter. This leads to a challenging, cardinality-constrained optimization problem. To deal with this challenge, we develop a novel, unconstrained reformulation and we prove that it is equivalent to the original problem. The reformulation uses a binary encoding scheme that implicitly imposes the cardinality constraint using learnable binary codes. **(ii)** To make the unconstrained reformulation smooth, we relax and smooth the binary variables. We demonstrate that, with careful initialization and regularization, the resulting problem can be optimized with first-order methods such as SGD. **(iii)** We carry out a series of experiments on synthetic and real MTL datasets, which show that our gate outperforms state-of-the-art gates in terms of parameter sharing and predictive performance. **(iv)** We provide an open-source implementation of DSelect-k.

1.1 Related Work

MoE and Conditional Computation: Since MoE was introduced by Jacobs et al. [14], an exciting body of work has extended and studied this model, e.g., see Jordan and Jacobs [17], Jacobs [13], Jiang and Tanner [16]. Recently, MoE-based models are showing success in deep learning. For example, Shazeer et al. [32] introduced the sparse Top-k gate for MoE and showed significant computational improvements on machine translation tasks; we discuss exact connections to this gate in Section 2.

The Top-k gate has also been utilized in several state-of-the-art deep learning models that considered MTL tasks, e.g., Lepikhin et al. [21], Ramachandran and Le [27], Fedus et al. [9]. Our work is also related to the conditional computation models that activate parts of the neural network based on the input [1]. These models use a trainable gate that decides which part of the neural network to activate for each input example, with the goal of speeding up training or inference. Works in this area include [2, 1, 32, 12, 33]. Unlike our work, these works are based on non-differentiable models, or heuristics where the training and inference models are different.

Stochastic Subset Selection and Sparse Transformations: A related line of work develops mechanisms for stochastic subset selection in neural networks, e.g., see [25, 5, 36] and the references therein. Specifically, these works allow for sampling k-subsets from a categorical distribution, based on extensions or generalizations of the Gumbel-softmax trick [23, 15]. However, in the MoE we consider *deterministic* subset selection—determinism is a common assumption in MoE models (for example, in [14, 17, 32]) that can improve interpretability. In contrast, the approaches described above assume stochastic selection and are suitable in applications where there is an underlying sampling distribution, such as in variational inference [19]. In Appendix A, we provide more context, and we discuss important differences between our proposal and these approaches. In addition, a related body of work has developed sparse transformations as alternatives to the softmax function, e.g., the sparsemax [24] and its generalizations [26, 6, 3]. However, there is an important distinction: our formulation imposes a cardinality constraint that controls the number of nonzeros precisely, whereas the latter approaches do not impose a cardinality constraint (e.g., sparsemax cannot control sparsity). The cardinality constraint is important for interpretability and computational efficiency in MoE because all examples or tasks will respect the desired sparsity level (in contrast, sparse transformations that do not precisely control sparsity, may assign some examples or tasks sparse combinations and others dense combinations).

MTL: In Appendix A, we review related literature on MTL.

2 Gating in the Mixture of Experts

In this section, we first review the MoE architecture and popular gates, and then discuss how these gates compare to our proposal. We will assume that the inputs to the MoE belong to a space $\mathcal{X} \subset \mathbb{R}^p$. In its simplest form, the MoE consists of a set of n experts (neural networks) $f_i : \mathcal{X} \to \mathbb{R}^u$, $i \in \{1, 2, \ldots, n\}$, and a gate $g : \mathcal{X} \to \mathbb{R}^n$ that assigns weights to the experts. The gate's output is assumed to be a probability vector, i.e., $g(x) \geq 0$ and $\sum_{i=1}^n g(x)_i = 1$, for any $x \in \mathcal{X}$. Given an example $x \in \mathcal{X}$, the corresponding output of the MoE is a weighted combination of the experts:

$$\sum_{i=1}^{n} f_i(x) g(x)_i. \tag{1}$$

Next, we discuss two popular choices for the gate $g(.)$ that can be directly optimized using SGD.

Softmax Gate: A classical model for $g(x)$ is the softmax gate: $\sigma(Ax+b)$, where $\sigma(.)$ is the softmax function, $A \in \mathbb{R}^{n \times p}$ is a trainable weight matrix, and $b \in \mathbb{R}^n$ is a bias vector [17]. This gate is dense, in the sense that all experts are assigned nonzero probabilities. Note that static gating (i.e., gating which does not depend on the input example) can be obtained by setting $A = 0$.

Top-k Gate: This is a sparse variant of the softmax gate that returns a probability vector with only k nonzero entries [32]. The Top-k gate is defined by $\sigma(\mathit{KeepTopK}(Ax + b))$, where for any vector v, $\mathit{KeepTopK}(v)_i := v_i$ if v_i is in the top k elements of v, and $\mathit{KeepTopK}(v)_i := -\infty$ otherwise[2]. This gate is conceptually appealing since it allows for direct control over the number of experts to select and is trained using SGD. Moreover, the Top-k gate supports *conditional training*: in backpropagation, for each input example, only the gradients of the loss w.r.t. top k elements need to be computed. With a careful implementation, conditional training can lead to computational savings. However, the Top-k gate is not continuous, which implies that the gradient does not exist at certain inputs. This can be problematic when training is done using gradient-based methods. To gain more insight, in Appendix C.3, we plot the expert weights chosen by the Top-k gate during training with SGD. The results indicate an oscillatory behavior in the output of the Top-k gate, which can be attributed to its discontinuous nature: a small change in the input can lead to "jumps" in the output.

[2] To help in load balancing across the experts, Shazeer et al. [32] add Gaussian noise and additional regularizers to the model.

Comparison with DSelect-k: We develop DSelect-k in Section 3. Here we present a high-level comparison between DSelect-k and Top-k. Similar to Top-k, DSelect-k can select k out of the n experts and can be trained using gradient-based optimization methods. A major advantage of DSelect-k over Top-k is that it is continuously differentiable, which leads to more stable selection of experts during training—see Appendix C.3 for visualizations of the expert weights during training. During inference, DSelect-k only needs to evaluate a subset of the experts, which can lead to computational savings. However, DSelect-k supports conditional training only partially. At the start of training, it uses all the available experts, so conditional training is not possible. As we discuss in Section 3, after a certain point during training, DSelect-k converges to a small subset of the experts, and then conditional training becomes possible. Our experiments indicate that DSelect-k has a significant edge over Top-k in terms of prediction and expert selection performance, so the full support for conditional training in Top-k comes at the expense of statistical performance.

3 Differentiable and Sparse Gating

In this section, we develop the DSelect-k gate, for both the static and per-example gating settings. First, we introduce the problem setup and notation. To simplify the presentation, we will develop the gate for a single supervised learning task, and we note that the same gate can be used in MTL models. We assume that the task has an input space $\mathcal{X} \subset \mathbb{R}^p$, an output space $\mathcal{Y}$, and an associated loss function $\ell : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}$. We denote the set of N training examples by $\mathcal{D} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^N$. We consider a learning model defined by the MoE in Equation (1). For simplicity, we assume that the experts are scalar-valued and belong to a class of continuous functions $\mathcal{H}$. We assume that the number of experts $n = 2^m$ for some integer m—in Appendix B.2, we discuss how the gate can be extended to arbitrary n. For convenience, given a non-negative integer i, we denote the set $\{1, 2, \ldots, i\}$ by $[i]$.

In Section 3.1, we develop DSelect-k for the static gating setting. Then, in Section 3.2, we generalize it to the per-example setting.

3.1 DSelect-k for Static Gating

Our goal here is to develop a static gate that selects a convex combination of at most k out of the n experts. The output of the gate can be thought of as a probability vector w with at most k nonzero entries, where w_i is the weight assigned to the expert f_i. A natural way to minimize the empirical risk of the MoE model is by solving the following problem:

$$\min_{f_1,\ldots f_n, w} \quad \frac{1}{N} \sum_{(x,y)\in\mathcal{D}} \ell\Big(y, \sum_{i=1}^{n} f_i(x) w_i\Big) \tag{2a}$$

$$\text{s.t.} \quad \|w\|_0 \leq k \tag{2b}$$

$$\sum_{i=1}^{n} w_i = 1, \; w \geq 0. \tag{2c}$$

In the above, the L_0 norm of w, $\|w\|_0$, is equal to the number of nonzero entries in w. Thus, the cardinality constraint (2b) ensures that the gate selects at most k experts. Problem (2) is a combinatorial optimization problem that is not amenable to SGD due to the cardinality constraint (2b) and the simplex constraints in (2c). In what follows of this section, we first transform Problem (2) into an equivalent unconstrained optimization problem, based on a binary encoding scheme. However, the unconstrained problem cannot be directly handled using SGD due to the presence of binary variables. Thus, in a second transformation, we smooth the binary variables, which leads to an optimization problem that is amenable to SGD.

Roadmap: In Section 3.1.1, we introduce the *single expert selector*: a construct for choosing 1 out of n experts by using binary encoding. In Section 3.1.2, we leverage the single expert selector to transform Problem (2) into an unconstrained one. Then, in Section 3.1.3, we smooth the unconstrained problem and discuss how SGD can be applied.

3.1.1 Single Expert Selection: Binary Encoding

The single expert selector (selector, for short) is a fundamental construct that we will later use to convert Problem (2) to an unconstrained optimization problem. At a high-level, the single expert selector chooses the index of 1 out of the n experts and returns a one-hot encoding of the choice. For example, in the case of 4 experts, the selector can choose the first expert by returning the binary

vector $[1\ 0\ 0\ 0]^T$. Generally, the selector can choose any of the experts, and its choice is determined by a set of binary encoding variables, as we will describe next.

The selector is parameterized by m (recall that $m = \log_2 n$) binary variables, $z_1, z_2, \ldots, z_m$, where we view these variables collectively as a binary number: $z_m z_{m-1} \ldots z_1$. The integer represented by the latter binary number determines which expert to select. More formally, let l be the integer represented by the binary number $z_m z_{m-1} \ldots z_1$. The selector is a function $r : \mathbb{R}^m \to \{0,1\}^n$ which maps $z := [z_1, z_2, \ldots, z_m]^T$ to a one-hot encoding of the integer $(l+1)$. For example, if all the z_i's are 0, then the selector returns a one-hot encoding of the integer 1. Next, we define the selector $r(z)$. For easier exposition, we start with the special case of 4 experts and then generalize to n experts.

Special case of 4 experts: In this case, the selector uses two binary variables z_1 and z_2. Let l be the integer represented by the binary number $z_2 z_1$. Then, the selector should return a one-hot encoding of the integer $(l+1)$. To achieve this, we define the selector $r(z)$ as follows:

$$r(z) = [\bar{z}_1 \bar{z}_2,\ \ z_1 \bar{z}_2,\ \ \bar{z}_1 z_2,\ \ z_1 z_2]^T \tag{3}$$

where $\bar{z}_i := 1 - z_i$. By construction, exactly one entry in $r(z)$ is 1 (specifically, $r(z)_{l+1} = 1$) and the rest of the entries are zero. For example, if $z_1 = z_2 = 0$, then $r(z)_1 = 1$ and $r(z)_i = 0$ for $i \in \{2, 3, 4\}$.

General case of n experts: Here we generalize the selector $r(z)$ to the case of n experts. To aid in the presentation, we make the following definition. For any non-negative integer l, we define $\mathcal{B}(l)$ as the set of indices of the nonzero entries in the binary representation of l (where we assume that the least significant bit is indexed by 1). For example, $\mathcal{B}(0) = \emptyset$, $\mathcal{B}(1) = \{1\}$, $\mathcal{B}(2) = \{2\}$, and $\mathcal{B}(3) = \{1, 2\}$. For every $i \in [n]$, we define the i-th entry of $r(z)$ as follows:

$$r(z)_i = \prod_{j \in \mathcal{B}(i-1)} (z_j) \prod_{j \in [m] \setminus \mathcal{B}(i-1)} (1 - z_j) \tag{4}$$

In the above, $r(z)_i$ is a product of m binary variables, which is equal to 1 iff the integer $(i-1)$ is represented by the binary number $z_m z_{m-1} \ldots z_1$. Therefore, $r(z)$ returns a one-hot encoding of the index of the selected expert. Note that when $n = 4$, definitions (3) and (4) are equivalent.

3.1.2 Unconstrained Minimization

In this section, we develop a combinatorial gate which allows for transforming Problem (2) into a unconstrained optimization problem. We design this gate by creating k instances of the single expert selector $r(.)$, and then taking a convex combination of these k instances. More formally, for every $i \in [k]$, let $z^{(i)} \in \{0,1\}^m$ be a (learnable) binary vector, so that the output of the i-th instance of the selector is $r(z^{(i)})$. Let Z be a $k \times m$ matrix whose i-th row is $z^{(i)}$. Moreover, let $\alpha \in \mathbb{R}^k$ be a vector of learnable parameters. We define the *combinatorial gate* q as follows:

$$q(\alpha, Z) = \sum_{i=1}^{k} \sigma(\alpha)_i r(z^{(i)}),$$

where we recall that $\sigma(.)$ is the softmax function. Since for every $i \in [k]$, $r(z^{(i)})$ is a one-hot vector, we have $\|q(\alpha, Z)\|_0 \leq k$. Moreover, since the weights of the selectors are obtained using a softmax, we have $q(\alpha, Z) \geq 0$ and $\sum_{i=1}^{n} q(\alpha, Z)_i = 1$. Thus, $q(\alpha, Z)$ has the same interpretation of w in Problem (2), without requiring any constraints. Therefore, we propose replacing w in the objective of Problem (2) with $q(\alpha, Z)$ and removing all the constraints. This replacement leads to an equivalent unconstrained optimization problem, as we state in the next proposition.

Proposition 1. *Problem* (2) *is equivalent*[3] *to:*

$$\begin{aligned} \min_{f_1, \ldots f_n, \alpha, Z} \quad & \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \ell\Big(y, \sum_{i=1}^{n} f_i(x) q(\alpha, Z)_i\Big) \\ & z^{(i)} \in \{0,1\}^m,\ i \in [k] \end{aligned} \tag{5}$$

The proof of Proposition 1 is in Appendix B.1. Unlike Problem (2), Problem (5) does not involve any constraints, aside from requiring binary variables. However, these binary variables cannot be directly handled using first-order methods. Next, we discuss how to smooth the binary variables in order to obtain a continuous relaxation of Problem (5).

[3] By equivalent we mean that the two problems have the same optimal objective, and given an optimal solution for one problem, we can construct an optimal solution for the other.

3.1.3 Smooth Gating

In this section, we present a procedure to smooth the binary variables in Problem (5) and discuss how the resulting problem can be optimized using first-order methods. The procedure relies on the *smooth-step* function, which we define next.

Smooth-step Function: This is a continuously differentiable and S-shaped function, similar in shape to the logistic function. However, unlike the logistic function, the smooth-step function can output 0 and 1 exactly for sufficiently large magnitudes of the input. The smooth-step and logistic functions are depicted in Appendix B.3. More formally, given a non-negative scaling parameter γ, the smooth-step function, $S : \mathbb{R} \to \mathbb{R}$, is a cubic piecewise polynomial defined as follows:

$$S(t) = \begin{cases} 0 & \text{if } t \leq -\gamma/2 \\ -\frac{2}{\gamma^3}t^3 + \frac{3}{2\gamma}t + \frac{1}{2} & \text{if } -\gamma/2 \leq t \leq \gamma/2 \\ 1 & \text{if } t \geq \gamma/2 \end{cases}$$

The parameter γ controls the width of the fractional region (i.e., the region where the function is strictly between 0 and 1). Note that $S(t)$ is continuously differentiable at all points—this follows since at the boundary points $\pm\gamma/2$, we have: $S'(-\gamma/2) = S'(\gamma/2) = 0$. This function has been recently used for conditional computation in soft trees [11] and is popular in the literature on computer graphics [8, 29].

Smoothing: We obtain DSelect-k from the combinatorial gate $q(\alpha, Z)$ by (i) relaxing every binary variable in Z to be continuous in the range $(-\infty, +\infty)$, i.e., $Z \in \mathbb{R}^{k \times m}$, and (ii) applying the smooth-step function to Z element-wise. Formally, DSelect-k is a function $\tilde{q}$ defined as follows:

$$\tilde{q}(\alpha, Z) := q(\alpha, S(Z)) = \sum_{i=1}^{k} \sigma(\alpha)_i r\big(S(z^{(i)})\big), \tag{6}$$

where the matrix $S(Z)$ is obtained by applying $S(\cdot)$ to Z element-wise. Note that $\tilde{q}(\alpha, Z)$ is continuously differentiable so it is amenable to first-order methods. If $S(Z)$ is binary, then $\tilde{q}(\alpha, Z)$ selects at most k experts (this holds since $\tilde{q}(\alpha, Z) = q(\alpha, S(Z))$, and from Section 3.1.2, q selects at most k experts when its encoding matrix is binary). However, when $S(Z)$ has any non-binary entries, then more than k experts can be potentially selected, meaning that the cardinality constraint will not be respected. In what follows, we discuss how the gate can be optimized using first-order methods, while ensuring that $S(Z)$ converges to a binary matrix so that the cardinality constraint is enforced.

We propose using $\tilde{q}(\alpha, Z)$ in MoE, which leads to the following optimization problem:

$$\min_{f_1, \ldots f_n, \alpha, Z} \quad \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \ell\Big(y, \sum_{i=1}^{n} f_i(x) \tilde{q}(\alpha, Z)_i\Big) \tag{7}$$

Problem (7) can be viewed as a continuous relaxation of Problem (5). If the experts are differentiable, then the objective of Problem (7) is differentiable. Thus, we propose optimizing MoE end-to-end using first-order methods. We note that $\tilde{q}(\alpha, Z)$ uses $(k + k \log n)$ learnable parameters. In contrast, the Top-k and softmax gates (discussed in Section 2) use n parameters. Thus, for relatively small k, our proposal uses a smaller number of parameters. Next, we discuss how the DSelect-k gate's parameters should be initialized in order to ensure that it is trainable.

Initialization: By the definition of the smooth-step function, if $S(Z_{ij})$ is binary then $S'(Z_{ij}) = 0$, and consequently $\frac{\partial \ell}{\partial Z_{ij}} = 0$. This implies that, during optimization, if $S(Z_{ij})$ becomes binary, the variable Z_{ij} will not be updated in any subsequent iteration. Thus, we have to be careful about the initialization of Z. For example, if Z is initialized so that $S(Z)$ is a binary matrix then the gate will not be trained. To ensure that the gate is trainable, we initialize each Z_{ij} so that $0 < S(Z_{ij}) < 1$. This way, the Z_{ij}'s can have nonzero gradients at the start of optimization.

Accelerating Convergence to Binary Solutions: Recall that we need $S(Z)$ to converge to a binary matrix, in order for the gate $\tilde{q}$ to respect the cardinality constraint (i.e., to select at most k experts). Empirically, we observe that if the optimizer runs for a sufficiently large number of iterations, then $S(Z)$ typically converges to a binary matrix. However, early stopping of the optimizer can be desired in practice for computational and statistical considerations, and this can prevent $S(Z)$ from converging. To encourage faster convergence towards a binary $S(Z)$, we will add an entropy regularizer to Problem (7). The following proposition is needed before we introduce the regularizer.

Proposition 2. *For any $z \in \mathbb{R}^m$, $\alpha \in \mathbb{R}^k$, and $Z \in \mathbb{R}^{k \times m}$, $r(S(z))$ and $\tilde{q}(\alpha, Z)$ belong to the probability simplex.*

The proof of the proposition is in Appendix B.1. Proposition 2 implies that, during training, the output of each single expert selector used by $\tilde{q}(\alpha, Z)$, i.e., $r(S(z^{(i)}))$ for $i \in [k]$, belongs to the probability simplex. Note that the entropy of each $r(S(z^{(i)}))$ is minimized by any one-hot encoded vector. Thus, for each $r(S(z^{(i)}))$, we add an entropy regularization term that encourages convergence towards one-hot encoded vectors; equivalently, this encourages convergence towards a binary $S(Z)$. Specifically, we solve the following regularized variant of Problem (7):

$$\min_{f_1, \ldots f_n, \alpha, Z} \quad \sum_{(x,y) \in \mathcal{D}} \frac{1}{N} \ell\Big(y, \sum_{i=1}^{n} f_i(x) \tilde{q}(\alpha, Z)_i\Big) + \lambda \Omega(Z)$$

where $\Omega(Z) := \sum_{i=1}^{k} h\big(r(S(z^{(i)}))\big)$ and $h(.)$ is the entropy function. The hyperparameter λ is non-negative and controls how fast each selector converges to a one-hot encoding. In our experiments, we tune over a range of λ values. When selecting the best hyperparameters from tuning, we disregard any λ whose corresponding solution does not have a binary $S(Z)$. In Appendix C.4, we report the number of training steps required for $S(Z)$ to converge to a binary matrix, on several real datasets. Other alternatives to ensure that $S(Z)$ converges to a binary matrix are also possible. One alternative is to regularize the entropy of each entry in $S(Z)$ separately. Another alternative is to anneal the parameter γ of the smooth-step function towards zero.

Softmax-based Alternative to Binary Encoding: Recall that our proposed selectors in (6), i.e., $r\big(S(z^{(i)})\big)$, $i \in [k]$, learn one-hot vectors *exactly* (by using binary encoding). One "practical" alternative for learning a one-hot vector is by using a softmax function with temperature annealing (or entropy regularization). Theoretically, this alternative cannot return a one-hot vector, but after training, the softmax output can be transformed to a one-hot vector using a heuristic (e.g., by taking an argmax). In Appendix C.2, we perform an ablation study of our gate in which we replace the selectors with softmax functions (along with temperature annealing or entropy regularization). Although this alternative is heuristic, we consider it due to its popularity.

3.2 DSelect-k for Per-example Gating

In this section, we generalize the static version of DSelect-k, $\tilde{q}(\alpha, Z)$, to the per-example gating setting. The key idea is to make the gate's parameters α and Z functions of the input, so that the gate can make decisions on a per-example basis. Note that many functional forms are possible for these parameters. For simplicity and based on our experiments, we choose to make α and Z linear functions of the input example. More formally, let $G \in \mathbb{R}^{k \times p}$, $W^{(i)} \in \mathbb{R}^{m \times p}$, $i \in [k]$, be a set of learnable parameters. Given an input example $x \in \mathbb{R}^p$, we set $\alpha = Gx$ and $z^{(i)} = W^{(i)} x$ in $\tilde{q}(\alpha, Z)$ (to simplify the presentation, we do not include bias terms). Thus, the per-example version of DSelect-k is a function v defined as follows:

$$v(G, W, x) = \sum_{i=1}^{k} \sigma(Gx)_i r\big(S(W^{(i)} x)\big).$$

In the above, the term $r\big(S(W^{(i)} x)\big)$ represents the i-th single expert selector, whose output depends on the example x; thus different examples are free to select different experts. The term $\sigma(Gx)_i$ determines the input-dependent weight assigned to the i-th selector. The gate $v(G, W, x)$ is continuously differentiable in the parameters G and W, so we propose optimizing it using first-order methods. Similar to the case of static gating, if $S(W^{(i)} x)$ is binary for all $i \in [k]$, then each $r\big(S(W^{(i)} x)\big)$ will select exactly one expert, and the example x will be assigned to at most k experts.

To encourage $S(W^{(i)} x)$, $i \in [k]$ to become binary, we introduce an entropy regularizer (similar in essence to that in static gating). However, unlike static gating, the regularizer here should be on a per-example basis, so that each example respects the cardinality constraint. By Proposition 2, for any $i \in [k]$, $r\big(S(W^{(i)} x)\big)$ belongs to the probability simplex. Thus, for each example x in the training data, we introduce a regularization term of the form: $\Omega(W, x) := \sum_{i \in [k]} h\Big(r\big(S(W^{(i)} x)\big)\Big)$, and minimize the following objective function:

$$\sum_{(x,y)\in\mathcal{D}} \Big(\frac{1}{N}\ell\Big(y, \sum_{i=1}^{n} f_i(x)v(G,W,x)_i\Big) + \lambda\Omega(W,x)\Big),$$

where λ is a non-negative hyperparameter. Similar to the case of static gating, we tune over a range of λ values, and we only consider the choices of λ that force the average number of selected experts per example to be less than or equal to k. If the application requires that the cardinality constraint be satisfied strictly for every example (not only on average), then annealing γ in the smooth-step function towards zero enforces this.

4 Experiments

We study the performance of the DSelect-k gate in the context of MTL and compare with state-of-the-art gates and baselines. In the rest of this section, we present experiments on the following real MTL datasets: MovieLens, Multi-MNIST, and Multi-Fashion MNIST. Moreover, in Appendix C, we present two additional experiments: (i) an experiment on data from a large-scale, real-world recommender system and (ii) an experiment on synthetic data (with up to 128 tasks), in which we study the predictive and expert selection performance and perform ablation studies.

Competing Methods: We focus on a multi-gate MoE, and study the DSelect-k and Top-k gates in the both static and per-example gating settings. In addition, we consider two MTL baselines. The first baseline is a MoE with a softmax gate (which uses all the available experts). The second is a *shared bottom* model [4], where all tasks share the same bottom layers, which are in turn connected to task-specific neural networks.

Experimental Setup: All competing models were implemented in TensorFlow 2. We used Adam [18] and Adagrad [7] for optimization, and we tuned the key hyperparameters using random grid search (with an average of 5 trials per grid point). Full details on the setup are in Appendix D.

4.1 MovieLens

Dataset: MovieLens [10] is a movie recommendation dataset containing records for $4{,}000$ movies and $6{,}000$ users. Following Wang et al. [34], for every user-movie pair, we constructed two tasks. Task 1 is a binary classification task for predicting whether the user will watch a particular movie. Task 2 is a regression task to predict the user's rating (in $\{1, 2, \ldots, 5\}$) for a given movie. We use 1.6 million examples for training and $200,000$ for each of the validation and testing sets.

Experimental Details: We use the cross-entropy and squared error losses for tasks 1 and 2, respectively. We optimize a weighted average of the two losses, i.e., the final loss function is $\alpha(\text{Loss of Task 1}) + (1-\alpha)(\text{Loss of Task 2})$, and we report the results for $\alpha \in \{0.1, 0.5, 0.9\}$. The same loss function is also used for tuning and testing. The architecture consists of a multi-gate MoE with 8 experts, where each of the experts and the task-specific networks is composed of ReLU-activated dense layers. For each α, we tune over the optimization and gate-specific hyperparameters, including the number of experts to select (i.e., k in DSelect-k and Top-k). After tuning, we train each model for 100 repetitions (using random initialization) and report the averaged results. For full details, see Appendix D.1.

Results: In Table 1, we report the test loss and the average number of selected experts[4]. The results indicate that for all values of α, either one of our DSelect-k gates (static or per-example) outperforms the competing methods, in terms of both the test loss and the number of selected experts. In only one case ($\alpha = 0.1$), Top-k (static) performed marginally better than DSelect-k gate (static), but DSelect-k (per-example) achieves the best performance across all methods in this case. Notably, the softmax MoE is uniformly outperformed by the DSelect-k and Top-k gates, so sparsity in gating seems to be beneficial on this dataset. Moreover, there does not seem to be a clear winner between static and per-example gating: each performs better for certain α's.

4.2 Multi-MNIST and Multi-Fashion MNIST

Datasets: We consider two image classification datasets: Multi-MNIST and Multi-Fashion [31], which are multi-task variants of the MNIST [20] and Fashion MNIST [35] datasets. We construct the Multi-MNIST dataset similar to Sabour et al. [31]: (i) we start by uniformly sampling two images

[4]The number of experts is averaged over tasks and training repetitions, so it may take non-integer values.

from the MNIST dataset and overlaying them on top of each other, and (ii) we shift the first and second digits by 4 pixels (in each direction) towards the top-left and bottom-right corners, respectively. This procedure leads to 36×36 images with some overlap between the digits. The Multi-Fashion is constructed in a similar way by overlaying images from the Fashion MNIST dataset. For each dataset, we consider two classification tasks: Task 1 is to classify the top-left item and Task 2 is to classify the bottom-right item. We use $100{,}000$ examples for training, and $20{,}000$ examples for each of the validation and testing sets.

Experimental Details: We use cross-entropy loss for each of the tasks and optimize the sum of the two losses[5]. The model is a multi-gate MoE with 8 experts, where each expert is a convolutional neural network and each task-specific network is composed of a number of dense layers. We tune the optimization and gate-specific hyperparameters, including the number of experts to select (i.e., k in DSelect-k and Top-k), and use the average of the task accuracies as the tuning metric. After tuning, we train each model for 100 repetitions (using random initialization) and report the averaged results. For full details, see Appendix D.2.

Results: In Table 2, we report the test accuracy and the number of selected experts for the Multi-MNIST and Multi-Fashion MNIST datasets. On Multi-MNIST, DSelect-k (static) outperforms Top-k, in terms of both the task accuracies and number of selected experts. For example, it achieves over 1% improvement in Task 2's accuracy compared to Top-k (static). DSelect-k (static) comes close to the performance of the Softmax MoE, but uses less experts (1.7 versus 8 experts). Here the DSelect-k (per-example) does not offer improvement over the static variant (unlike the MovieLens dataset). On Multi-Fashion, we again see that the DSelect-k (static) performs best in terms of accuracy.

Table 1: Test loss (with standard error) and average number of selected experts on MovieLens. The parameter α determines the weight of Task 1's loss (see text for details). The test loss is multiplied by 10^4.

		$\alpha = 0.1$		$\alpha = 0.5$		$\alpha = 0.9$	
		Loss	Experts	Loss	Experts	Loss	Experts
Static	DSelect-k	4015 ± 5	2.7	$\mathbf{3804} \pm 3$	$\mathbf{1.5}$	$\mathbf{3690} \pm 2$	$\mathbf{1.3}$
	Top-k	$\mathbf{4012} \pm 4$	$\mathbf{2.0}$	3818 ± 2	2.0	3693 ± 6	2.0
Per-example	DSelect-k	$\mathbf{4006} \pm 6$	$\mathbf{1.5}$	$\mathbf{3823} \pm 3$	$\mathbf{1.2}$	$\mathbf{3679} \pm 2$	$\mathbf{1.1}$
	Top-k	4027 ± 8	2.0	3841 ± 4	2.0	3741 ± 3	2.0
Baselines	Softmax MoE	4090 ± 1	8.0	3960 ± 3	8.0	3847 ± 10	8.0
	Shared Bottom	4037 ± 2	-	3868 ± 2	-	3687 ± 1	-

Table 2: Test accuracy (with standard error) and number of selected experts on Multi-MNIST/Fashion.

		Multi-MNIST			Multi-Fashion MNIST		
		Accuracy 1	Accuracy 2	Experts	Accuracy 1	Accuracy 2	Experts
Static	DSelect-k	$\mathbf{92.56} \pm 0.03$	$\mathbf{90.98} \pm 0.04$	$\mathbf{1.7}$	$\mathbf{83.78} \pm 0.05$	$\mathbf{83.34} \pm 0.05$	1.8
	Top-k	91.93 ± 0.06	90.03 ± 0.08	4	83.44 ± 0.07	82.66 ± 0.08	4
Per-example	DSelect-k	$\mathbf{92.42} \pm 0.03$	$\mathbf{90.7} \pm 0.03$	$\mathbf{1.5}$	$\mathbf{83.69} \pm 0.04$	83.13 ± 0.04	$\mathbf{1.5}$
	Top-k	92.27 ± 0.03	90.45 ± 0.03	4	83.66 ± 0.04	$\mathbf{83.15} \pm 0.04$	4
Baselines	Softmax MoE	92.61 ± 0.03	91.0 ± 0.03	8	83.48 ± 0.04	82.81 ± 0.04	8
	Shared Bottom	91.3 ± 0.04	89.47 ± 0.04	-	82.05 ± 0.05	81.37 ± 0.06	-

5 Conclusion

We introduced DSelect-k: a continuously differentiable and sparse gate for MoE, which can be trained using first-order methods. Given a user-specified parameter k, the gate selects at most k of the n experts. Such direct control over the sparsity level is typically handled in the literature by adding a cardinality constraint to the optimization problem. One of the key ideas we introduced is a binary encoding scheme that allows for selecting k experts, without requiring any constraints in the optimization problem. We studied the performance of DSelect-k in MTL settings, on both synthetic and real datasets. Our experiments indicate that DSelect-k can achieve significant improvements in prediction and expert selection, compared to state-of-the-art MoE gates and MTL baselines.

Societal Impact: MoE models are used in various applications (as discussed in the related work). Our proposal has the potential to improve the interpretability and computational efficiency of MoE models and thus benefit the underlying applications. We do not see direct, negative societal impacts from our proposal.

[5]Due to the symmetry in the problem, assigning the two tasks equal weights is a reasonable choice.

References

[1] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *CoRR*, abs/1511.06297, 2015. URL `http://arxiv.org/abs/1511.06297`.

[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[3] Mathieu Blondel, Andre Martins, and Vlad Niculae. Learning classifiers with fenchel-young losses: Generalized entropies, margins, and algorithms. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 606–615. PMLR, 2019.

[4] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[5] Jianbo Chen, Le Song, Martin Wainwright, and Michael Jordan. Learning to explain: An information-theoretic perspective on model interpretation. In *International Conference on Machine Learning*, pages 883–892. PMLR, 2018.

[6] Gonçalo M Correia, Vlad Niculae, and André FT Martins. Adaptively sparse transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2174–2184, 2019.

[7] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[8] David S Ebert, F Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003.

[9] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

[10] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[11] Hussein Hazimeh, Natalia Ponomareva, Petros Mol, Zhenyu Tan, and Rahul Mazumder. The tree ensemble layer: Differentiability meets conditional computation. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4138–4148, Virtual, 13–18 Jul 2020. PMLR.

[12] Yani Ioannou, Duncan Robertson, Darko Zikic, Peter Kontschieder, Jamie Shotton, Matthew Brown, and Antonio Criminisi. Decision forests, convolutional networks and the models in-between. *arXiv preprint arXiv:1603.01250*, 2016.

[13] Robert A Jacobs. Bias/variance analyses of mixtures-of-experts architectures. *Neural computation*, 9(2):369–383, 1997.

[14] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[15] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[16] Wenxin Jiang and Martin A Tanner. On the identifiability of mixtures-of-experts. *Neural Networks*, 12(9):1253–1258, 1999.

[17] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[19] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[20] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[21] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[22] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. 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*, pages 1930–1939, 2018.

[23] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[24] Andre Martins and Ramon Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International Conference on Machine Learning*, pages 1614–1623. PMLR, 2016.

[25] Max Paulus, Dami Choi, Daniel Tarlow, Andreas Krause, and Chris J Maddison. Gradient estimation with stochastic softmax tricks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 5691–5704. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/3df80af53dce8435cf9ad6c3e7a403fd-Paper.pdf`.

[26] Ben Peters, Vlad Niculae, and André FT Martins. Sparse sequence-to-sequence models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1504–1519, 2019.

[27] Prajit Ramachandran and Quoc V Le. Diversity and depth in per-example routing models. In *International Conference on Learning Representations*, 2018.

[28] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *International Conference on Learning Representations*, 2018.

[29] Randi J Rost, Bill Licea-Kane, Dan Ginsburg, John Kessenich, Barthold Lichtenbelt, Hugh Malan, and Mike Weiblen. *OpenGL shading language*. Pearson Education, 2009.

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

[31] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866, 2017.

[32] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[33] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018.

[34] Yuyan Wang, Zhe Zhao, Bo Dai, Christopher Fifty, Dong Lin, Lichan Hong, and Ed H Chi. Small towers make big differences. *arXiv preprint arXiv:2008.05808*, 2020.

[35] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[36] Sang Michael Xie and Stefano Ermon. Reparameterizable subset sampling via continuous relaxations. In *International Joint Conference on Artificial Intelligence*, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In Section 2, we discuss a limitation of our proposal (specifically that it supports conditional training only partially).
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In the conclusion section, we discuss societal impact.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We included the code for the proposed method. However, we did not include the code for reproducing the experiments because it relies on proprietary libraries (e.g., for tuning).
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] We ran the experiments on a large cluster that uses various types of hardware (CPU, memory, etc.), so it is difficult to tell what exact hardware setup was used in the experiments.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] Our released code has an attached license.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]