Efficient Learning of Discrete-Continuous Computation Graphs

Anonymous Author(s)
Affiliation
Address
email

Abstract

Numerous models for supervised, unsupervised, and reinforcement learning benefit from combining discrete and continuous components. The resulting discrete-continuous models are end-to-end learnable and are compositional, tend to generalize better, and are more interpretable. A popular approach to build discrete-continuous computation graphs is that of integrating discrete probability distributions into neural networks using stochastic softmax tricks. Prior work has focused on computation graphs with a single discrete component on each of the graph's execution paths. We analyze the behavior of more complex stochastic computations graphs with multiple sequential discrete components. We show that it is challenging to optimize the parameters of these models, mainly due to vanishing gradients. We then propose two new strategies to overcome these challenges. First, we propose dropout residual connections specifically tailored to discrete-continuous computation graphs. Second, we show that matching the temperature parameters of the Gumbel-Softmax and increasing the scale parameter of the Gumbel noise perturbations drastically improves the learning behavior. With an extensive set of experiments, we show that we can train complex discrete-continuous models which one cannot train with standard stochastic softmax tricks. We also show that using complex discrete-stochastic models allows us to generalize better than their continuous counterparts on several benchmark datasets.

1 Introduction

Neuro-symbolic learning systems aim to combine discrete and continuous operations. The majority of recent research has focused on integrating neural network components into probabilistic logics [21, 15], that is, making logic-based reasoning approaches more amenable to noisy and high-dimensional input data. On the other end of the spectrum are the data-driven approaches which aim to learn a modular and discrete program structure in an end-to-end neural network based system. The broader vision followed by proponents of these approaches are systems capable of assembling the required modular operations to solve a variety of tasks with heterogeneous input data. Reinforcement learning, program induction, and neural module networks [1] are particular instances of such data-driven discrete-continuous learning systems.

We take a more focused look at learning systems comprised of both discrete and continuous (differentiable) operations where the discrete operations are modeled as discrete probability distributions. The resulting systems can be described by their *stochastic computation graphs* a formalism recently introduced to unify several related approaches [23]. Figure 1 illustrates two instances of such stochastic computation graphs. For a given high-dimensional input such as a set of images or a list of symbols, a computation graph, consisting of stochastic (discrete) and continuous (neural) nodes, is created to solve a particular task. Both, the mechanism to assemble the graphs (if not already provided) and

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

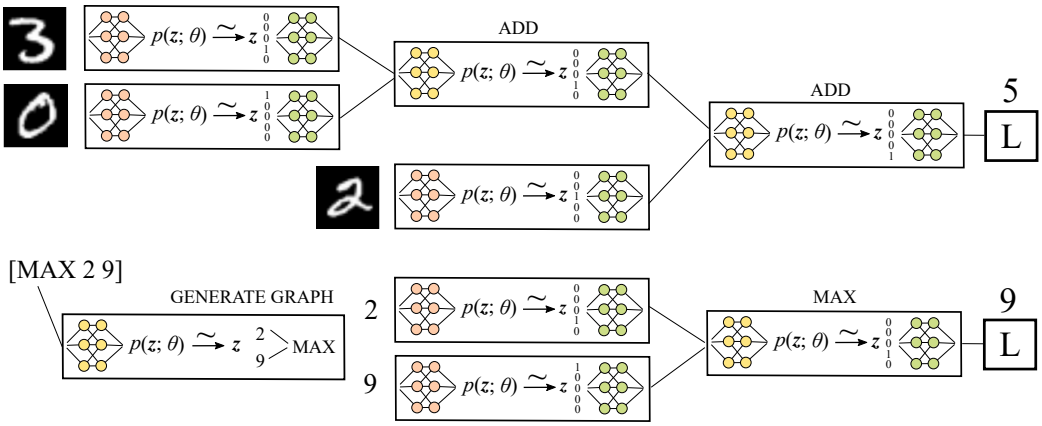

Figure 1: Two typical instances of stochastic computation graphs we consider in this paper.

the various operations are learned end-to-end. Due to their modularity and discrete behavior, these models often exhibit improved generalization behavior and are more interpretable, an observation which we will also make in the experiments.

With this paper we address the problem of learning such discrete-continuous computation graphs. The majority of prior work has focused on graphs with a single discrete probability distribution along each execution path. Examples are the discrete variational autoencoder [11], learning to explain [2], and other applications of stochastic softmax tricks [20]. We aim to analyze and improve the training behavior of *complex* computation graphs, that is, graphs with more than one discrete probability distribution in its execution paths. More concretely, we focus on computation graphs where the stochastic nodes are categorical random variables modeled with the Gumbel-softmax trick [11, 14]. We show both analytically and empirically that it is challenging to optimize the parameters of these models, mainly due to vanishing gradients. We then propose two new methods for mitigating the vanishing gradient problem. First, we propose dropout residual connections specifically tailored to discrete-continuous computation graphs. Second, we show that matching the temperature parameter τ of the Gumbel-Softmax and increasing the scale parameter of the Gumbel noise perturbations *relative to* τ improves the models' learning behavior. We show empirically for several complex discrete-continuous models that they can only be properly trained when using the proposed methods. We also show that the resulting discrete-continuous models generalize better and significantly outperform state of the art approaches on benchmark datasets.

2 Efficient Learning of Discrete-Continuous Computation Graphs

Any deterministic neural network can be represented by a composition of basic differentiable functions. These networks, therefore, can be fully described by a directed acyclic graph (the computation graph) that determines the operations executed during the forward and backward passes. Schulman et al. [23] proposed *stochastic computation graphs* as an extension of neural networks that combine deterministic and stochastic operations – a node in the computation graph can be either a differentiable function or a probability distribution. A *stochastic node* X is typically a random variable with a parameterized probability distribution $p_{\boldsymbol{\theta}(x)}$ with parameters $\boldsymbol{\theta}$. Suppose that f is a smooth function (such as the loss function of a learning problem), then the gradient of f at the stochastic node is $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(x)}[f(x)]$. Kingma and Welling [12] proposed the *reparameterization trick* to overcome the problem of computing gradients with respect to the parameters of a distribution. The idea is to find a function g and distribution ρ such that one can replace $x \sim p_{\boldsymbol{\theta}}(x)$ by $x = g(z, \boldsymbol{\theta})$ with $z \sim \rho(z)$. If this is possible, then one can write

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{x \sim p_{\boldsymbol{\theta}}(x)}[f(x)] = \mathbb{E}_{z \sim \rho(z)}[\nabla_{\boldsymbol{\theta}} f(g(z, \boldsymbol{\theta}))] \approx \frac{1}{S} \sum_{i=1}^{S} \nabla_{\boldsymbol{\theta}} f(g(z_i, \boldsymbol{\theta})) \text{ with } z_i \sim \rho(z). \quad (1)$$

With this paper we address the problem of training stochastic computation graphs where the stochastic nodes are based on categorical (discrete) distributions approximated using stochastic softmax

tricks [11, 20]. More specifically, we consider discrete-continuous components modeling a categorical variable X with k possible values $\boldsymbol{z}_1, ..., \boldsymbol{z}_k$. Each of the $\boldsymbol{z}_i$ is the one-hot encoding of the category i. We consider discrete-continuous functions[1] $f : \mathbb{R}^n \to \mathbb{R}^n$ with $\boldsymbol{v} = f(\boldsymbol{u})$ defined as follows

$$\boldsymbol{\theta} = g_{\boldsymbol{w}}(\boldsymbol{u}) \tag{2a}$$

$$p(\boldsymbol{z}_i; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta}_i)}{\sum_{j=1}^{k} \exp(\boldsymbol{\theta}_j)} \tag{2b}$$

$$\boldsymbol{z} \sim p(\boldsymbol{z}; \boldsymbol{\theta}) \tag{2c}$$

$$\boldsymbol{v} = h_{\boldsymbol{w}'}(\boldsymbol{z}) \tag{2d}$$

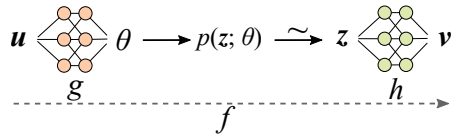

Figure 2: Illustration of generic discrete-continuous component (function f).

We assume that the functions g and h (parameterized by $\boldsymbol{w}$ and $\boldsymbol{w}'$) are expressed using a differentiable neural network component.[2] Figure 2 illustrates the generic discrete-continuous component and Figure 1 depicts two complex computation graphs comprised of such components.

Let $\mathrm{Gumbel}(0, \beta)$ be the Gumbel distribution with location 0 and scale β. Using the Gumbel-max trick, we can sample $\boldsymbol{z} \sim p(\boldsymbol{z}; \boldsymbol{\theta})$ as follows:

$$\boldsymbol{z} := \boldsymbol{z}_i \quad \text{with} \quad i = \underset{j \in \{1,\ldots,k\}}{\arg\max} \, [\boldsymbol{\theta}_j + \epsilon_j] \text{ where } \epsilon_j \sim \mathrm{Gumbel}(0, 1). \tag{3}$$

The Gumbel-softmax trick is a relaxation of the Gumbel-max trick (relaxing the argmax into a softmax with a scaling parameter λ) that allows one to perform standard backpropagation:

$$\boldsymbol{z}_i = \frac{\exp\left((\boldsymbol{\theta}_i + \epsilon_i)/\tau\right)}{\sum_{j=1}^{k} \exp\left((\boldsymbol{\theta}_j + \epsilon_j)/\tau\right)} \text{ where } \epsilon_i \sim \mathrm{Gumbel}(0, 1). \tag{4}$$

Hence, instead of sampling a discrete $\boldsymbol{z}$, the Gumbel-Softmax trick computes a relaxed $\boldsymbol{z}$ in its place.

We are concerned with the analysis of the behavior of the Gumbel-softmax trick in more complex stochastic computation graphs. In these computation graphs, multiple sequential Gumbel-softmax components occur on a single execution path. Throughout the remainder of the paper, we assume that during training we use the Softmax-trick as outlined above while at test time we sample discretely using the Gumbel-max trick.

Let us first take a look at $\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{u}}$, that is, the derivative of the output of the discrete-continuous function f with respect to its input. Using the chain rule (and with a slight abuse of notation to simplify the presentation), we can write

$$\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{u}} = \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{u}}.$$

By assumption $\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{z}}$ and $\frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{u}}$ exist and are efficiently computable. Using the derivative of the softmax function and setting $Z := \sum_{\ell=1}^{k} \exp\left((\boldsymbol{\theta}_\ell + \epsilon_\ell)/\tau\right)$ (the partition function) we have

$$\frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{\theta}_j} = \begin{cases} \frac{1}{\tau} \left(\frac{\exp((\boldsymbol{\theta}_i + \epsilon_i)/\tau)}{Z}\right) \left(1 - \frac{\exp((\boldsymbol{\theta}_j + \epsilon_j)/\tau)}{Z}\right) & \text{if } i = j \\ -\frac{1}{\tau} \left(\frac{\exp((\boldsymbol{\theta}_i + \epsilon_i)/\tau)}{Z}\right) \left(\frac{\exp((\boldsymbol{\theta}_j + \epsilon_j)/\tau)}{Z}\right) & \text{if } i \neq j. \end{cases} \tag{5}$$

A closer inspection of these formulas points to the problem of vanishing gradients especially when components are connected sequentially. Similar to the problem of sigmoid activation units in standard deep neural networks, which have been replaced by ReLUs and similar modern activation functions but which, unfortunately, are not applicable to our setting, the Gumbel-Softmax-based components lead to vanishing gradients whenever the probability saturates. We hypothesize that it is crucial for the neural networks operating on the continuous (and often high-dimensional) input features to receive more direct gradient information through the discrete-continuous components. Indeed, we observe in several cases that the Gumbel-softmax distributions saturate and, as a result, that sufficient gradient information cannot reach the parameters of upstream neural network components.

We propose two strategies to mitigate the vanishing gradient behavior in complex discrete-continuous computation graphs. First, we analyze the interplay between the temperature parameter τ and the scale

[1] For the sake of simplicity we assume the same input and output dimensions.

[2] Both g and h can also be the identity.

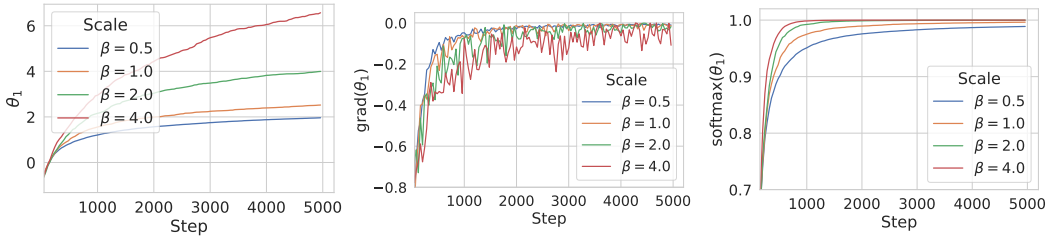

Figure 3: The values of the parameters, their gradients, and the softmax probabilities for various values of β (the scale parameter) for a constant Gumbel-Softmax parameter τ. Early during training, a lower scale parameter β (relative to τ) works well and has less variance. Once the probabilities saturate, however, we can only continue to obtain a sufficient gradient signal for larger values of β.

parameter β of the Gumbel distribution. We make the subtle difference between these parameters explicit which may also be of interest for improving stochastic softmax tricks [20]. Second, we introduce DROPRES connections allowing us to lower-bound the gradients in expectation and leading to more direct gradient flow to parameters of upstream model components.

2.1 TEMPMATCH: Temperature Matching and Inverse Scale Annealing

In this section we explore the behavior of two interdependent parameters: the Gumbel-softmax temperature τ and the Gumbel scale parameter β. First, we have the temperature parameter τ from the Gumbel-softmax trick (see Equation 4). The purpose of this parameter is to make the output of the softmax function more or less discrete, that is, more or less concentrated on one of the categories. Second, and this is a new insight, we can change the scale parameter β of the Gumbel distribution. For scale parameter β, we sample the noise perturbation iid as $\epsilon_i \sim \mathrm{Gumbel}(0, \beta)$. If we use the Gumbel-max trick of Equation 3 we implicitly generate samples from the distribution $p(\boldsymbol{z}_i; \boldsymbol{\theta}) = \exp(\boldsymbol{\theta}_i/\beta)/\sum_{j=1}^{k} \exp(\boldsymbol{\theta}_j/\beta)$. Increasing the scale parameter β, therefore, makes the Gumbel-max samples more uniform and less concentrated. When using the Gumbel-softmax trick instead of the Gumbel-max trick we obtain $\boldsymbol{z}_i$ that are more uniform and discrete. Indeed, in the limit of $\beta \to \infty$ we obtain discrete samples uniformly distributed over the categories, independent of the logits. For $\beta \to 0$, we obtain the standard softmax distribution with temperature 1. Now, the problem of vanishing gradients caused by Gumbel-softmax components can be mitigated by increasing the scale parameter (decreasing or annealing the inverse scale parameter) β *relative to* the temperature parameter τ. Intuitively, increasing β increases the probability of generating more uniform (relative to the current parameters $\boldsymbol{\theta}$) samples and, therefore, allowing the model to avoid saturation.

To illustrate the effect of matching the two parameters, we conducted the toy experiment depicted in Figure 3. The model here computes $\mathrm{Softmax}(\boldsymbol{\theta} + \boldsymbol{\epsilon})$ with parameters $\boldsymbol{\theta}^\intercal = (\theta_1, \theta_2)$ and $\boldsymbol{\epsilon} \sim \mathrm{Gumbel}(0, \beta)$. The learning problem is defined through a cross-entropy loss between the output probabilities of the Gumbel-Softmax and a constant target vector $(1.0, 0.0)^\intercal$. We observe that early during training (before the probabilities saturate) lower values for β work well and exhibit less variance than higher values. However, once the probabilities and, therefore, the gradients start to saturate, a higher value for β enables the continued training of the model. Indeed, even though the softmax probability is close to 1, we observe a continuous increase of the parameter θ_1 and a high enough gradient signal for $\beta \geq 2$. It also shows that annealing the *inverse* scale parameter, that is, increasing the scale parameter relative to τ over time, should be beneficial.

In summary, the two parameters τ and β should be *matched*, that is, optimized relative to each other, to obtain the desired behavior during training. We show in later experiments that for more realistic learning problems choosing a higher value for β (and/or increasing β over time) is highly beneficial.

2.2 DROPRES: Residual Dropout Connections

Residual (or shortcut) connections are crucial when training very deep neural networks [9]. A standard residual connection for the discrete-continuous components we consider here would be achieved by replacing Equation (2d) with $\boldsymbol{v} = (1 - \alpha)\boldsymbol{u} + \alpha h_{\boldsymbol{w}'}(\boldsymbol{z})$. At training time, one could use several strategies to balance the residual and model connections by, for instance, setting $\alpha > 0$ to be

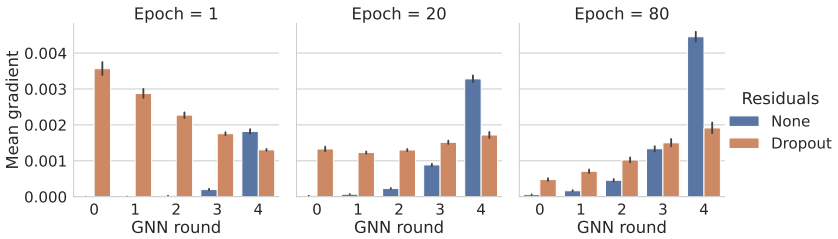

Figure 4: The influence of dropout residuals on the mean absolute gradients for ListOps. The values on the x-axis are the GNN message-passing iterations: the higher the number the closer to the loss function is the corresponding discrete-continuous component. Adding dropout residual connections mitigates the vanishing gradient problem for components with greater distance to the loss.

a constant or increasing it towards 1. What sets our problem apart from other end-to-end learnable neural networks is that we want to completely remove the residual connections at test time. We show empirically that models with residual connections tend to strongly overfit even if the parameter α is close to 1, and completely fail to perform well at test time. To mitigate the problem of overfitting to the existence of residual connections while at the same time reducing vanishing gradients in expectation, we propose DROPRES connections. These are residual connections sampled from a Bernoulli distribution with parameter α. We replace Equation (2d) by

$$\boldsymbol{v} = \begin{cases} \boldsymbol{u} + h_{\boldsymbol{w}'}(\boldsymbol{z}) & \text{with probability } 1-\alpha \\ h_{\boldsymbol{w}'}(\boldsymbol{z}) & \text{with probability } \alpha, \end{cases} \tag{6}$$

that is, we drop out the residual connection with probability α. This way, the model is exposed to both purely discrete and discrete-continuous computation paths. Hence, with probability $(1-\alpha)$ we obtain a *shortcut connection* between the neural components, effectively bypassing the Gumbel-softmax. With probability α the training path goes exclusively through the Gumbel-softmax distribution. The expectation of the gradients, taken over a Bernoulli random variable with parameter α, is now

$$\mathbb{E}\left[\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{u}}\right] = (1-\alpha)\mathbf{1} + \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\theta}}\frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{u}}. \tag{7}$$

Hence, the model obtains a stronger gradient signal for upstream models *in expectation* while not being able to overfit using the residual connections.

To illustrate the impact of dropout residual connections, we analyzed the gradients of message passing steps for the ListOps problem (see experiments) in a discrete-continuous computation graph of the type depicted in Figure 1 (bottom). As we can observe in Figure 4, if we do not use dropout residual connections, the greater the distance of a discrete-continuous operation to the loss function (the lower the value on the x-axis), the smaller is the mean absolute value of the gradients reaching it. This illustrates the vanishing gradient problem. When using dropout residual connections, on the other hand, the mean absolute values of the gradients do not vanish proportional to their distance to the loss and are more evenly distributed.

3 Related Work

Various stochastic gradient estimators have been proposed to tackle the problem of backpropagating through stochastic nodes. The REINFORCE algorithm [29] utilizes the log derivative trick to estimate score functions. More recent approaches are based on reparameterization tricks [12] that enable the gradient computation by removing the dependence of the density on the input parameters. Maddison et al. [14] and Jang et al. [11] proposed the Gumbel-Softmax distribution, a continuous (but biased) reparameterization trick for categorical distributions. Tucker et al. [27] and Grathwohl et al. [7] introduce parameterized control variates to lower the variance and obtain unbiasedness for these gradient estimators. Recently, Paulus et al. [20] propose a framework that generalizes the Gumbel-Softmax trick for various combinatorial distributions. In contrast to the above, we do not focus on further improving gradient estimators for a single stochastic node but explore the capabilities and difficulties of backpropagating through compositional chains of stochastic nodes. SparseMAP [19] is an approach to structured prediction and latent variables, replacing the exponential distribution

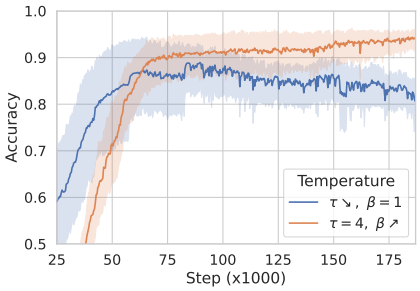

Figure 5: Comparison of annealing strategies on the MNIST Addition task for 3 MNIST images. The initial values are $\tau = 4$ and $\beta = 1$. Annealing the inverse scale parameter β results in a significantly higher test accuracy compared to the standard approach of annealing τ.

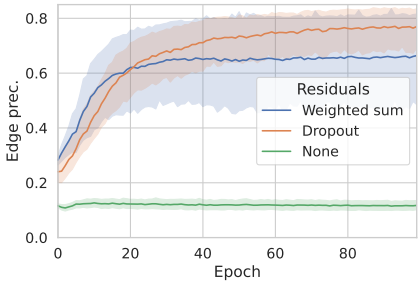

Figure 6: Comparison of two types of residual connections for the graph structure recovery task of ListOps. Increasing the dropout rate of the residuals linearly, see Equation 6, achieves a higher edge precision than increasing α in the weighted sum $(1-\alpha)\boldsymbol{u} + \alpha h_{\boldsymbol{w}'}(\boldsymbol{z})$.

(specifically, the softmax) with a sparser distribution. LP-SparseMAP [18] is a continuation of SparseMAP using a relaxation of the optimization problem rather than a MAP oracle. In addition, the idea to enforce more sparsity can also be exploited for efficient marginal inference in latent variable models [3] which can then be used to compute stochastic gradients.

The respective advantages of neural and symbolic methods have led to various attempts to combine these two. Early work like Towell and Shavlik [25] and Hölldobler et al. [10] has explored the possibilities of translating propositional logic programs into neural networks. Evans and Grefenstette [6] and Dong et al. [5] build upon these ideas by utilizing the framework of inductive logic programming to learn rules from examples. A different direction of work focuses on knowledge base reasoning. Rocktäschel and Riedel [21] combine Prolog's backward chaining algorithm with a differentiable unification whereas Yang et al. [30] learn to compose the inference task into differentiable operations. Serafini and Garcez [24] propose a differentiable first-order logic language to differentiate deductive reasoning and Manhaeve et al. [15] extend the probabilistic logical language ProbLog to interact with neural networks. Also related is prior work on automatic program induction. Andreas et al. [1] combine neural networks with the compositional linguistic structure of questions, Mao et al. [16] additionally learn visual concepts and Tsamoura and Michael [26] introduce the concept of abduction to learn the inputs of a symbolic program. All prior work integrate neural modules into concepts of classical logic or fuzzy logic or separate the neural and the symbolic parts in other ways. In contrast, we propose a method that allows to train complex stochastic computation graphs with multiple discrete probability distributions along their execution paths.

4 Experiments

The aim of the experiments is threefold. First, we want to analyze the behavior of complex stochastic computations graphs arising in typical application domains such as multi-hop reasoning in knowledge graphs and unsupervised parsing of lists of operations. Second, we want to evaluate the behavior of the stochastic computation graphs when incorporating the proposed methods (dropout residual connections and temperature matching) to improve the vanishing gradient problem. Third, we want to compare the resulting discrete-continuous models with state of the art models which do *not* have stochastic components. Here, we are especially interested in analyzing the generalization (extrapolation) behavior of the models under consideration. The implementations are in PyTorch and included in the supplementary material. All experiments were run on a GeForce RTX 2080 Ti GPU.

Unsupervised Parsing on ListOps The Listops dataset contains sequences in the style of prefix arithmetic such as $\max[\ 2\ 9\ \min[\ 4\ 7\ ]\ 0\ ]$ and its unique solutions (here: 9) [17]. Prior work adapted and used this dataset to evaluate the performance of stochastic softmax tricks [20]. Since the resulting dataset was not published, we generated a dataset following their description. More precisely, we used the three operators $\min$, $\max$ and med and capped the maximum length of a sequence at 50. For each depth $d \in \{1, 2, 3, 4, 5\}$ we generated $20,000$ samples for the training set and $2,000$ samples

Table 1: The results for unsupervised parsing on ListOps. Results taken from Paulus et al. [20] are marked with an asterisk (*) and are based on a different (unpublished) dataset.

				Task acc. (extrapolation)	
Model	Task acc.	Edge prec.	Inter. acc.	$d = 8$	$d = 10$
Und.*	$91.2 \pm 1.8^*$	$33.1 \pm 2.9^*$	-	n.a.	n.a.
Arb.*	$95.0 \pm 3.0^*$	$75.0 \pm 7.0^*$	-	n.a.	n.a.
LSTM	91.5 ± 0.3	-	-	83.7 ± 2.0	76.9 ± 3.7
Arb., $\tau = 2$	$\mathbf{96.8} \pm 0.3$	77.4 ± 1.8	-	84.3 ± 1.4	75.4 ± 1.7
Ours, $\tau = 1$	96.1 ± 0.4	$\mathbf{82.3} \pm 1.1$	$\mathbf{70.9} \pm 1.2$	$\mathbf{92.6} \pm 1.1$	$\mathbf{86.9} \pm 4.9$
Ours, $\tau = 2$	$\mathbf{96.3} \pm 0.5$	76.8 ± 2.2	62.9 ± 2.4	$\mathbf{92.7} \pm 1.3$	$\mathbf{88.7} \pm 3.3$
Arb. (GT)	98.7 ± 0.1	100.0	-	86.4 ± 1.1	71.0 ± 2.1
Ours (GT)	99.8 ± 0.1	100.0	99.9 ± 0.0	99.8 ± 0.1	99.9 ± 0.1

Table 2: The ablation study for the ListOps task. Temperature matching improves the training behavior of the model. The addition of dropout residual connections is crucial for efficient learning. Using the standard settings for softmax tricks (bottom row) leads to poor results.

				Task acc. (extrapolation)	
Model	Task acc.	Edge prec.	Inter. acc.	$d = 8$	$d = 10$
Ours, $\tau = 1$	96.1 ± 0.4	82.3 ± 1.1	70.9 ± 1.2	92.6 ± 1.1	86.9 ± 4.9
(-) TM	95.1 ± 1.7	76.9 ± 12.1	66.3 ± 9.2	90.5 ± 2.4	83.7 ± 4.0
(-) DR	74.8 ± 11.5	43.9 ± 30.3	29.4 ± 18.6	70.5 ± 9.3	66.7 ± 8.8
(-) DR, TM	56.2 ± 5.0	15.1 ± 7.7	12.7 ± 3.2	53.1 ± 6.6	49.2 ± 7.1

each for the validation set and test set. In order to evaluate the generalization behavior, we further generated $2,000$ test samples for each depth $d \in \{8, 10\}$. All other settings are those of the original code [17]. The arithmetic syntax of each training example induces a directed rooted in-tree, from now on called arborescence, which is the tree along which the message-passing steps of a GNN operate. We use the same bi-LSTM encoder as Paulus et al. [20] to compute the logits of all possible edges and the same directed GNN architecture. The original model uses a directed version of Kirchoff's matrix-tree theorem as introduced by Koo et al. [13] to induce the arborescence prior. We simplified this idea by taking, for each node, the Gumbel-softmax over all possible parent nodes. Here we make use of the fact that in an arborescece, each node (with the exception of the root) has exactly one parent. As in Paulus et al. [20], we exclude edges from the final closing bracket] since its edge assignment cannot be learned from the task objective.

In contrast to Paulus et al. [20], where only the edges of the latent graph are modeled with categorical variables, we also modelled the nodes of the latent graphs with discrete-continuous components (see Figure 1 (bottom)). Let $\mathrm{Num} \in \mathbb{R}^{10 \times \mathrm{dim}}$ be the embedding layer that maps the 10 numeral tokens to their respective embeddings and let $\mathrm{Pred} : \mathbb{R}^{\mathrm{dim}} \to \mathbb{R}^{10}$ be the classification layer that maps the embedding $\boldsymbol{x} \in \mathbb{R}^{\mathrm{dim}}$ of the final output to the logits of the 10 classes: $\mathrm{Pred}(\boldsymbol{x}) \coloneqq \mathrm{Lin}^{10 \times \mathrm{dim}}(\mathrm{ReLU}(\mathrm{Lin}_B^{\mathrm{dim} \times \mathrm{dim}}(\boldsymbol{x})))$. After each message-passing operation of the GNN, we obtain an embedding $\boldsymbol{u} \in \mathbb{R}^{\mathrm{dim}}$ for each node. By choosing $g_{\boldsymbol{w}} \coloneqq \mathrm{Pred}$, we obtain the 10 class logits $\boldsymbol{\theta} = g_{\boldsymbol{w}}(\boldsymbol{u}) = \mathrm{Pred}(\boldsymbol{u}) \in \mathbb{R}^{10}$ for the numerals $0, \ldots, 9$. Using the Gumbel-softmax trick with logits $\boldsymbol{\theta}$, scale β, and temperature τ, we obtain $\boldsymbol{z} \in \mathbb{R}^{10}$. We then compute $\boldsymbol{v} = h_{\boldsymbol{w}'}(\boldsymbol{z}) = \boldsymbol{z}^\intercal \mathrm{Num}$. We apply this on all intermediate node embeddings simultaneously and repeat this after each of the first 4 out of 5 message-passing rounds of the GNN. We use dropout residual connection with increasing dropout probability α. In total, we obtain a model with 5 sets of discrete operations: one set from the graph generation and 4 from the discrete-continuous message-passing steps. At test time, the model performs discrete and interpretable operations. Figure 1 depicts an example stochastic computation graph for the task. We used the same LSTM as in Paulus et al. [20] and a re-implemented version of their model with our arborescence prior. For ablation purposes, we run all our models with and without dropout residuals and temperature matching, respectively. All models are run for 100 epochs with a learning rate of 0.005 and we select $\tau \in \{1, 2, 4\}$. We keep all other hyperparameters as in Paulus et al. [20]. We evaluate the task accuracy, the edge precision

Table 3: Results for the path query benchmark [8].

	WordNet		Freebase	
Model	MQ	H@10	MQ	H@10
Bilinear-C	89.4	54.3	83.5	42.1
DistMult-C	90.4	31.1	84.8	38.6
TransE-C	93.3	43.5	88.0	50.5
Path-RNN	**98.9**	–	–	–
ROP	–	–	90.7	56.7
CoKE	94.2	67.9	95.0	77.7
Ours, $\tau = 4$	94.2	**68.8**	**97.9**	**89.6**

Table 4: Results for the extrapolation benchmark[28] for the path query dataset [8].

		WordNet		Freebase	
Paths	Model	MQ	H@10	MQ	H@10
≤ 4	CoKE	93.6	65.5	93.1	71.2
	Ours	93.6	**67.8**	**97.8**	**89.1**
≤ 3	CoKE	92.6	59.6	90.6	64.6
	Ours	92.4	**66.2**	**96.6**	**89.0**
≤ 2	CoKE	90.8	49.5	89.4	59.5
	Ours	90.6	**63.9**	**94.9**	**81.0**
≤ 1	CoKE	73.5	16.7	72.7	37.3
	Ours	**88.0**	**59.8**	**94.4**	**75.3**

(using the correct graphs), the task accuracy on intermediate operations, and two extrapolation tasks. For the evaluation of the extrapolation task with depth d, we run the GNN d rounds instead of 5.

Table 1 compares the proposed models with state of the art approaches. The best performing models are those with discrete-continuous components trained with dropout residuals and temperature matching. While our models do not improve the task accuracy itself, they exceed state of the art methods on all other metrics. Most noticeable is the improved generalization behavior: our models extrapolate much better to expressions with a depth not seen during training. This becomes even more visible when trained on the ground truth graph where the generalization ability is almost at 100%. Our model also achieves the best performance on the graph structure recovery with a precision of 82.3%. The mismatch between the task accuracy and the edge precision is mainly due to examples for which the correct latent graph is not required to solve the problem, e.g., for the task $\max[\ 2\ \max[\ 4\ 7\ ]]$ it does not matter whether the 7 points to the first or the second $\max[$. Note that our method performs discrete operations at test time which are therefore verifiable and more interpretable.

The ablation study in Table 2 highlights the challenge of learning discrete-continuous computation graphs with multiple sequential Gumbel-softmax components. It is entirely impossible to train the model without dropout residuals and temperature matching. The analysis of Equation 5 and Figure 4 reveals the reason. Discrete nodes in the computation graph cause vanishing gradients that multiply when being chained. Temperature matching helps stabilizing the learning partially but to avoid vanishing gradients entirely, the use of dropout residuals is necessary. Figure 6 compares the dropout residual connections with residual connections implemented through a weighted sum and demonstrates their superior performance.

Multi-Hop Reasoning over Knowledge Graphs Here we consider the problem of answering multi-hop (path) queries in knowledge graphs (KGs) [8]. The objective is to find the correct object (subject) given the subject (object) and a sequence of relations (the path) without knowing the intermediate entities. We evaluate various approaches on the standard benchmarks for path queries [8]. Further experiments on a newly created dataset based on FB15K-237 can be found in the Appendix.

We model each application of a relation type to an (intermediate) entity as a discrete-continuous component that discretely maps to one of the KG entities. The discrete distribution $p(\boldsymbol{z}; \boldsymbol{\theta})$, therefore, has as many categories as there are entities in the KG. Without loss of generality, let $\mathrm{score}(\boldsymbol{s}, \boldsymbol{r}, \boldsymbol{o}) : \mathbb{R}^{3\times\mathrm{dim}} \rightarrow \mathbb{R}$ be a scoring function that maps the embeddings of a triple to the logit (unnormalized probability) of the triple being true. Given the current (intermediate) entity embedding $\boldsymbol{s}$ and relation type embedding $\boldsymbol{r}$ we compute the logits $\boldsymbol{\theta}$ for all possible entities. Hence, the function g computes the logits for all possible entities. Using the Gumbel-softmax trick with parameter $\boldsymbol{\theta}$, scale β, and temperature τ, we obtain the sample $\boldsymbol{z} \in \mathbb{R}^n$. The function h now computes $\boldsymbol{z}^\intercal \mathbf{E}$ where $\mathbf{E}$ be the matrix whose rows are the entity embeddings. We use dropout residual connection between the input and output vectors of the discrete-continuous component during training. Note that we do not increase the number of parameters compared to the base scoring models. We use the hyperparameters of ComplEx as proposed by Ruffinelli et al. [22], i.e., we choose a dimension and batchsize of 512, a learning rate of 0.001, and a L3 regularization of 0.01. We further train 1vsAll with the cross-entropy loss for 300 epochs and repeat the experiment for each temperature $\tau \in \{1, 2, 4\}$. For these

Table 5: A comparison of our proposed method with DeepProbLog and a CNN baseline.

		Properties		
Model	Task Acc.	Interpretable	End-to-End	Discrete (test time)
Baseline	89.14 ± 1.22	No	Yes	No
DeepProbLog	98.17 ± 0.20	Yes	No	No
Ours, $\tau = 4$	98.10 ± 0.15	Yes	Yes	Yes

experiments we use the Gumbel hard setting also during training. We compare our best performing model on the original dataset from Guu et al. [8] with their proposed evaluation protocol against RNN models, [4], [31] and a state-of-the-art transformer model [28] for the standard task as well as in an extrapolation setting.

Table 3 lists the results for the proposed model in comparison to several baselines. Our proposed model performs significantly better than other models: we obtain new state of the art results (up to 15% better) on all benchmarks except for the metric MQ on WordNet. Similar to the results on ListOps, the discrete-continuous models have the strongest generalization performance (see Table 4). Even when trained only on triples, that is, paths of length ≤ 1, our model performs better on the path query task than most baselines trained on all paths. Please note that these performance boosts are based solely on the use of discrete stochastic components while using the exact same standard scoring functions.

End-to-End Learning of MNIST Addition The MNIST addition task addresses the learning problem of simultaneously (i) recognizing digits from images and (ii) performing the addition operation on the digit's numerical values [15]. The dataset was introduced with DeepProbLog, a system that combines neural elements with a probabilistic logic programming language. In DeepProbLog the program that adds two numbers is provided. We adopt their approach of using a convolutional neural network (CNN) to encode the MNIST images. Contrary to their approach, we learn the operation without any prior knowledge about adding numbers in a data-driven manner.

The addition operation is modeled as $\mathrm{Add}(\boldsymbol{x}, \boldsymbol{y}) = \mathrm{Lin}_B^{\mathrm{dim} \times 2\,\mathrm{dim}}(\mathrm{ReLU}(\mathrm{Lin}_B^{2\,\mathrm{dim} \times 2\,\mathrm{dim}}([\boldsymbol{x}; \boldsymbol{y}])))$ and we use a single linear layer to compute the logits. Given $\boldsymbol{u} \in \mathbb{R}^{\mathrm{dim}}$, the encoding of the digit obained from the CNN, the function g computes $\boldsymbol{\theta} = \mathrm{Lin}(\boldsymbol{u})$, that is, the 19 logits for all possible numbers. Using the Gumbel-softmax trick with parameter $\boldsymbol{\theta}$, scale β, and temperature τ, we obtain $\boldsymbol{z} \in \mathbb{R}^{19}$. The function h now computes $\boldsymbol{v} = \boldsymbol{z}^{\intercal}\,\mathrm{P} \in \mathbb{R}^{\mathrm{dim}}$ where P is the weight matrix of Lin. We discretize the two outputs of the CNN, execute the addition layer and use the single linear layer P for the classification. This results into a model that has in total less parameters than the baseline in [15]. For the experiments with 3 summands, see Figure 1 and Figure 5, we use the same model with 2 separate addition layers and choose $\mathrm{P} \in \mathbb{R}^{28 \times \mathrm{dim}}$. Here, we discretize after each CNN encoding as well as after the first addition. We train this model with $\tau = 8$ and a learning rate of 0.0001.

Table 5 lists the results on this task. Our proposed model, even though trained without a given logic program and in a data-driven manner, achieves the same accuracy as DeepProbLog. Since we use hard sampling during test time, we can discretely read out the outputs of the internal CNN layer. Furthermore, the learned operation is modular and likely able to extrapolate to multiple sequential addition operations.

5 Conclusion

We introduced two new strategies to overcome the challenges of learning complex discrete-continuous computation graphs. Such graphs can be effective links between neural and symbolic methods but suffer from vanishing gradients. We propose dropout residual connections and a new temperature matching schema for mitigating the vanishing gradient problem. Note that even with our proposed methods, learning models with discrete nodes is still considerably more challenging than models without. In experiments across a diverse set of domains, we demonstrate that our methods enable the training of complex discrete-continuous models, which could not be learned before. These models beat prior state of the art models without discrete components on several benchmarks and show a remarkable gain in generalization behaviour.

References

[1] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[2] J. Chen, L. Song, M. Wainwright, and M. Jordan. Learning to explain: An information-theoretic perspective on model interpretation. In *International Conference on Machine Learning*, pages 883–892. PMLR, 2018.

[3] G. M. Correia, V. Niculae, W. Aziz, and A. F. Martins. Efficient marginalization of discrete and structured latent variables via sparsity. *Advances in Neural Information Processing Systems*, 2020.

[4] R. Das, A. Neelakantan, D. Belanger, and A. McCallum. Chains of reasoning over entities, relations, and text using recurrent neural networks. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 132–141, Valencia, Spain, Apr. 2017.

[5] H. Dong, J. Mao, T. Lin, C. Wang, L. Li, and D. Zhou. Neural logic machines. *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9*, 2019.

[6] R. Evans and E. Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

[7] W. Grathwohl, D. Choi, Y. Wu, G. Roeder, and D. Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3*, 2018.

[8] K. Guu, J. Miller, and P. Liang. Traversing knowledge graphs in vector space. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] S. Hölldobler, Y. Kalinke, and H.-P. Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11(1):45–58, 1999.

[11] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26*, 2017.

[12] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16*, 2014.

[13] T. Koo, A. Globerson, X. Carreras, and M. Collins. Structured prediction models via the matrix-tree theorem. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 141–150, 2007.

[14] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26*, 2017.

[15] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, and L. De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, Montréal, Canada, December 3-8, 2018*, 31:3749–3759, 2018.

[16] J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, and J. Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9*, 2019.

[17] N. Nangia and S. Bowman. Listops: A diagnostic dataset for latent tree learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 92–99, 2018.

[18] V. Niculae and A. F. T. Martins. Lp-sparsemap: Differentiable relaxed optimization for sparse structured prediction. In *ICML*, 2020.

[19] V. Niculae, A. F. T. Martins, M. Blondel, and C. Cardie. Sparsemap: Differentiable sparse structured inference. In *ICML*, 2018.

[20] M. B. Paulus, D. Choi, D. Tarlow, A. Krause, and C. J. Maddison. Gradient estimation with stochastic softmax tricks. *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, December 6-12*, 2020.

[21] T. Rocktäschel and S. Riedel. End-to-end differentiable proving. *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, December 4-9*, 2017.

[22] D. Ruffinelli, S. Broscheit, and R. Gemulla. You {can} teach an old dog new tricks! on training knowledge graph embeddings. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30*, 2020.

[23] J. Schulman, N. Heess, T. Weber, and P. Abbeel. Gradient estimation using stochastic computation graphs. *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, Montreal, Quebec, Canada, December 7-12*, 2015.

[24] L. Serafini and A. d. Garcez. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *arXiv preprint arXiv:1606.04422*, 2016.

[25] G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial intelligence*, 70(1-2):119–165, 1994.

[26] E. Tsamoura and L. Michael. Neural-symbolic integration: A compositional perspective. *arXiv preprint arXiv:2010.11926*, 2020.

[27] G. Tucker, A. Mnih, C. J. Maddison, D. Lawson, and J. Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, December 4-9*, 2017.

[28] Q. Wang, P. Huang, H. Wang, S. Dai, W. Jiang, J. Liu, Y. Lyu, Y. Zhu, and H. Wu. Coke: Contextualized knowledge graph embedding. *arXiv preprint arXiv:1911.02168*, 2019.

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

[30] F. Yang, Z. Yang, and W. W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, December 4-9*, 2017.

[31] W. Yin, Y. Yaghoobzadeh, and H. Schütze. Recurrent one-hop predictions for reasoning over knowledge graphs. *arXiv preprint arXiv:1806.04523*, 2018.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Abstract and introduction are in line with the rest of the paper.
   (b) Did you describe the limitations of your work? [Yes] In the conclusion.
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (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? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
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] The code will be supplied and more details are found in the supplemental material.
   (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] The error bars can be found in the main paper or in the supplements.
   (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)? [Yes]
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]
   (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? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]
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]