Differentiable Top-k: From One-Hot to k-Hot

Klas Wijk[1,2] **Ricardo Vinuesa**[2,3] **Hossein Azizpour**[1,2,4]
[1]KTH Royal Institute of Technology [2]Swedish e-Science Research Centre
[3]University of Michigan [4]Science for Life Laboratory
kwijk@kth.se rvinuesa@umich.edu azizpour@kth.se

Abstract

The one-hot representation, argmax operator, and its differentiable relaxation, softmax, are ubiquitous in machine learning. These building blocks lie at the heart of everything from the cross-entropy loss and attention mechanism to differentiable sampling. However, their k-hot counterparts are not as universal. In this paper, we consolidate the literature on differentiable top-k, showing how the k-capped simplex connects relaxed top-k operators and πps sampling to form an intuitive generalization of one-hot sampling. In addition, we propose sigmoid top-k, a scalable relaxation of the top-k operator that is fully differentiable and defined for continuous k. We validate our approach empirically and demonstrate its computational efficiency.

1 Introduction

A cornerstone of machine learning is the **one-hot vector**. In this work, we consider the more general set of ***k*-hot vectors**, where k "hot" indices are allowed instead of just one,

$$\{0,1\}_1^n \coloneqq \{\boldsymbol{x} \in \{0,1\}^n \mid \textstyle\sum_{i=1}^n x_i = 1\} \quad \xrightarrow{k\text{-hot}} \quad \{0,1\}_k^n \coloneqq \{\boldsymbol{x} \in \{0,1\}^n \mid \textstyle\sum_{i=1}^n x_i = k\}.$$

A fundamental operator related to one-hot vectors is the one-hot **argmax**, which outputs the one-hot vector corresponding to the index of the input vector's largest value[1]. The k-hot counterpart to argmax is the **top-*k*** operator,

$$\text{argmax} : \mathbb{R}^n \to \{0,1\}_1^n \quad \xrightarrow{k\text{-hot}} \quad \text{top-}k : \mathbb{R}^n \to \{0,1\}_k^n.$$

Unfortunately, these operators are non-differentiable. In the one-hot case, **softmax** is an established differentiable relaxation of argmax. Softmax is widely used in classification networks, attention mechanisms, reinforcement learning, and probabilistic models to parameterize categorical distributions. In this work, we cope with the non-differentiability of top-k by proposing an efficient and fully differentiable relaxation. Then, we turn to the stochastic case, and combine the relaxation with ***π*ps sampling** (Tillé, 2006) for differentiable k-hot sampling.

2 Method

Generalized softmax. The codomain of softmax is the **simplex**. This set can be generalized to the k-hot case as the ***k*-cappped simplex** (Wang and Lu, 2015; Ang et al., 2021),

$$\Delta^{n-1} \coloneqq \{\boldsymbol{\pi} \in [0,1]^n \mid \textstyle\sum_{i=1}^n \pi_i = 1\} \quad \xrightarrow{k\text{-hot}} \quad \Delta_k^{n-1} \coloneqq \{\boldsymbol{\pi} \in [0,1]^n \mid \textstyle\sum_{i=1}^n \pi_i = k\}.$$

Figure 1 shows the set in three dimensions for different values of k. We want a k-hot counterpart to softmax with the k-capped simplex as its codomain,

[1]This argmax should not be confused with the argmax operator in optimization.

1st Workshop on Differentiable Systems and Scientific Machine Learning @ EurIPS 2025

$$\text{softmax} : \mathbb{R}^n \to \Delta^{n-1} \quad \xrightarrow{k\text{-hot}} \quad \sigma_k : \mathbb{R}^n \to \Delta_k^{n-1}.$$

We propose the **sigmoid top-k** function as a differentiable relaxation of the top-k operator. For $\boldsymbol{x} \in \mathbb{R}^n$ and $k \in (0, n)$ we define:

$$\sigma_k(\boldsymbol{x}) \coloneqq \sigma(\boldsymbol{x} + c\mathbf{1}), \text{ where } c \in \mathbb{R} \text{ solves } \sum_{i=1}^n \sigma(x_i + c) = k.$$

We prove the existence and uniqueness of c in §A. The root is found numerically by scalar root-finding (Kong et al., 2020). The time complexity is $\mathcal{O}(n)$ per root-finding iteration, of which only a handful are typically required. Empirically, the number of root-finding iterations does not appear to grow with n. Intuitively, c is a scalar regardless of n. The number of iterations is also independent of k, which simply shifts the location of the root. This means σ_k can be computed efficiently in high dimensions, see Figure 3. The sum-constraint implicitly defines c as a function of $\boldsymbol{x}$ and k, i.e., $c(\boldsymbol{x}, k)$. Using the chain rule and implicit differentiation, we get:

$$\frac{\mathrm{d}\sigma_k(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \sigma_k(\boldsymbol{x})}{\partial \boldsymbol{x}} + \frac{\partial \sigma_k(\boldsymbol{x})}{\partial c}\frac{\partial c}{\partial \boldsymbol{x}} = \operatorname{diag}(\sigma'(\boldsymbol{x} + c)) - \frac{\sigma'(\boldsymbol{x} + c)\sigma'(\boldsymbol{x} + c)^\top}{\sum_{i=1}^n \sigma'(x_i + c)},$$
$$\frac{\mathrm{d}\sigma_k(\boldsymbol{x})}{\mathrm{d}k} = \frac{\partial \sigma_k(\boldsymbol{x})}{\partial c}\frac{\partial c}{\partial k} = \frac{\sigma'(\boldsymbol{x} + c)}{\sum_{i=1}^n \sigma'(x_i + c)},$$

where $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. See §A for the derivation. By using implicit differentiation to find derivatives of c, we avoid unrolling the steps of the root-finding algorithm and differentiating through them. This uses less memory and computation than unrolling (Blondel et al., 2022). Note that sigmoid top-k is also differentiable with respect to k, which can optionally be treated as learnable.

Proposition 1. Let $\boldsymbol{x} \in \mathbb{R}^n$, $k \in (0, n)$, and $\tau \in \mathbb{R}_+$, then the following properties hold:

a) Order-preservation. $\sigma_k(\boldsymbol{x})_i > \sigma_k(\boldsymbol{x})_j$ if and only if $x_i > x_j$, for all $i \neq j$.

b) Shift-invariance. $\sigma_k(\boldsymbol{x} + \alpha) = \sigma_k(\boldsymbol{x})$, for all $\alpha \in \mathbb{R}$.

c) Invertible up to an additive constant. $\sigma^{-1}(\sigma_k(\boldsymbol{x})) = \boldsymbol{x} + c$.

d) Infinite temperature limit. $\lim_{\tau \to \infty} \sigma_k\left(\frac{\boldsymbol{x}}{\tau}\right)_i = \frac{k}{n}$.

e) Zero temperature limit. $\lim_{\tau \to 0} \sigma_k\left(\frac{\boldsymbol{x}}{\tau}\right) = \text{top-}k(\boldsymbol{x})$ for distinct x_i and $k \in \mathbb{N}$.

Proof. The proposition follows by definition, see §A. □

We list some important properties of sigmoid top-k in Proposition 1. All of the properties are shared with softmax, or have softmax counterparts. Figure 2 shows a concrete example of tempering sigmoid top-k. The temperature plays an important role in relaxed sampling, controlling the bias–variance trade-off (Maddison et al., 2017; Jang et al., 2017). It can be fixed, annealed, or learned.

Proposition 2. Let $H(p) = -p \log p - (1 - p) \log(1 - p)$ be the binary entropy function and $\boldsymbol{x} \in \mathbb{R}^n$. Sigmoid top-$k$ solves the optimization problem:

$$\sigma_k(\boldsymbol{x}) = \underset{\boldsymbol{\pi} \in \Delta_k^{n-1}}{\arg\max}\ \boldsymbol{x}^\top \boldsymbol{\pi} + \sum_{i=1}^n H(\pi_i)$$

Proof. The proposition is derived from the Lagrangian, see §A. □

The optimization problem in Proposition 2 is an instance of a regularized prediction function (Blondel et al., 2019). Intuitively, the optimization problem maximizes the similarity between the input $\boldsymbol{x}$ and output probabilities $\boldsymbol{\pi}$ as measured by the inner product. The projection is regularized by the elementwise binary entropy, which is maximized at the center of Δ_k^{n-1} for any k.

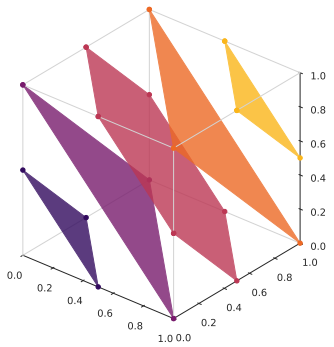

Figure 1: **The k-capped simplex**. The set of points in the k-capped simplex Δ_k^{n-1} is the intersection of the unit hypercube and a hyperplane, shown here in $n = 3$ dimensions for different values of k.

Algorithm 1 Sample k-hot

Require: $\boldsymbol{\theta} \in \mathbb{R}^n, k \in \mathbb{N}, 1 < k < n$
1: $\boldsymbol{\pi} \leftarrow \sigma_k(\boldsymbol{\theta})$
2: Sample $\boldsymbol{x} \in \{0,1\}_k^n$ such that $p(x_i) = \pi_i$
3: **return** $\boldsymbol{x}$

Algorithm 2 Sample relaxed k-hot

Require: $\boldsymbol{\theta} \in \mathbb{R}^n, k \in \mathbb{N}, 1 < k < n$
1: Sample $g_i \sim \text{Gumbel}(0,1)$ for $i = 1, \ldots, n$
2: $\boldsymbol{x} \leftarrow \sigma_k(\boldsymbol{\theta} + \boldsymbol{g})$
3: **return** $\boldsymbol{x}$

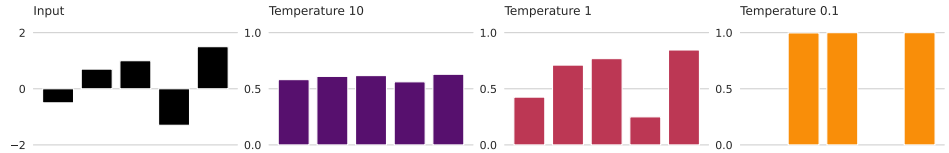

Figure 2: **Temperature scaling.** Just like the standard sigmoid and softmax functions, sigmoid top-k can be tempered by dividing its input by a temperature $\tau \in \mathbb{R}_+$. Its output approaches uniform ($\pi_i = \frac{k}{n}$) as $\tau \to \infty$, and its output approaches top-k as $\tau \to 0$.

Generalized categorical. The categorical distribution draws samples $\boldsymbol{x} \in \{0,1\}_1^n$ with parameters $\boldsymbol{\pi} \in \Delta^{n-1}$. These parameters define the **inclusion probabilities**,

$$p(x_i) = \textstyle\sum_{\boldsymbol{x} \in \{0,1\}_1^n} p(\boldsymbol{x}) \mathbf{1}_{x_i = 1}.$$

In other words, $p(x_i) = \pi_i$ is the marginal probability that x_i equals one. We want to generalize this to the k-hot case. A natural choice is πps sampling (Tillé, 2006), which draws samples $\boldsymbol{x} \in \{0,1\}_k^n$ with parameters $\boldsymbol{\pi} \in \Delta_k^{n-1}$.

$$\text{Categorical} \xrightarrow{k\text{-hot}} \pi\text{ps sampling}$$

However, there are many possible **sampling designs** that define different distributions given the same parameters. Furthermore, the same sampling design can be implemented using different algorithms, or **sampling procedures**. Importantly, some designs, like the one implemented by the Gumbel top-k procedure (Kool et al., 2020), do not produce exact inclusion probabilities $p(x_i) = \pi_i$ (Tillé, 2023). We propose using designs parameterized, at least approximately, by their inclusion probabilities. See §B for more details.

Gradient estimation. **Straight-through** estimation (Bengio et al., 2013) replaces the sample $\boldsymbol{x}$ by its parameters $\boldsymbol{\pi}$ in the backward pass. Intuitively, this works because $\mathbb{E}[\boldsymbol{x}] = \boldsymbol{\pi}$ if $p(x_i) = \pi_i$. In fact, the expected gradient is a first-order approximation for categorical samples (Liu et al., 2023). By using πps sampling such that $p(x_i) = \pi_i$, we extend this proof to the k-hot case in Proposition 3.

Proposition 3. Let $\boldsymbol{\pi} \in \Delta_k^{n-1}$ be inclusion probabilities and $\boldsymbol{x} \in \{0,1\}_k^n$ be samples drawn such that $p(x_i) = \pi_i$. Then, the expected gradient of the straight-through estimator is a first-order approximation of the true gradient.

Proof. The proposition is shown by extending the proof in Liu et al. (2023), see §C. □

For **score function estimators**, we need to use a πps sampling design with a tractable score function. Clearly, this is possible if we can compute $p(\boldsymbol{x})$ and differentiate, as was done in Wijk et al. (2025). Another approach is **relaxed sampling**. Xie and Ermon (2019) extended Gumbel–softmax (Maddison et al., 2017; Jang et al., 2017) to the k-hot case. Here, the top-k relaxation used is critical. Using sigmoid top-k, as shown in Algorithm 2, reduces the time complexity from $\mathcal{O}(nk)$ to $\mathcal{O}(n)$ and improves experimental results.

3 Related work

Relaxed top-k. The relaxation in Xie and Ermon (2019) is based on applying softmax sequentially k times. This is $\mathcal{O}(nk)$ and does not guarantee an output in Δ_k^{n-1}. In Pervez et al. (2023), the sigmoid output is rescaled directly. This relaxation is not shift-invariant, and non-differentiable along the boundary of its piecewise definition. Multiple works on top-k relaxations are posed as

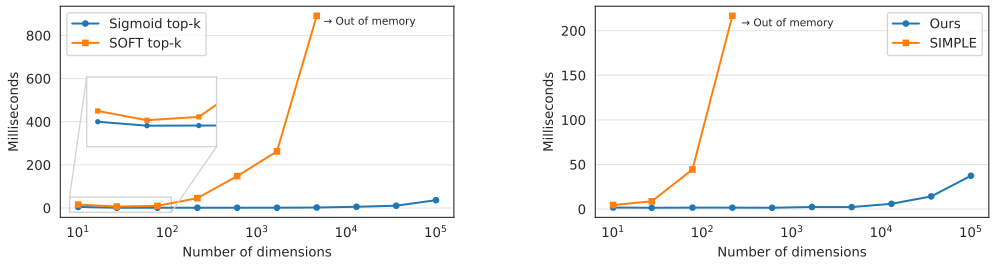

Figure 3: **Runtime. Left:** Relaxed top-k wall-clock time comparison between sigmoid top-k and SOFT top-k. The experimental setup is the same as for the root-finding benchmark in §A. **Right:** Differentiable k-hot sampling wall-clock time comparison between Algorithm 1 and SIMPLE. SIMPLE is using warmed-up `torch.compile` and loading precomputed values from disk. Both comparisons use the official implementations and measure both forward and backward times.

optimization problems (Xie et al., 2020; Sander et al., 2023). Compared to these, the root-finding problem in sigmoid top-k can be solved more efficiently Figure 3.

Differentiable k-hot sampling. Xie and Ermon (2019) proposed the top-k generalization of Gumbel–softmax in Algorithm 2. As discussed earlier in this section, sigmoid top-k improves on the top-k relaxation used in the original work. NCPSS (Pervez et al., 2023), on the other hand, is similar to Algorithm 1 with straight-through estimation. As mentioned in the previous section, the rescaled sigmoid is a top-k relaxation (although it is not presented as such), and iterative Poisson sampling[2] is an approximate approach to πps sampling that does not produce exactly k-hot samples. Our proposed sigmoid top-k and exact sampling address improve these two components, respectively. SIMPLE (Ahmed et al., 2023), in effect, computes the inclusion probabilities from their recursive definition[3] and uses them as the gradient estimates. In other words, straight-through estimation. We find that computing the inclusion probabilities using dynamic programming is less efficient than sampling according to Algorithm 1 with straight-through estimation, see Figure 3. SFESS (Wijk et al., 2025), like SIMPLE, considers conditional Poisson sampling and computes a score function estimator.

4 Experiments

We validate our approach to differentiable k-hot sampling in two settings: feature selection (Huijben et al., 2019) and sparse representation learning (Kingma and Welling, 2014). We report results for the MNIST and Fashion-MNIST datasets (LeCun et al., 1998; Xiao et al., 2017). See §D for more details and results. As expected, Algorithm 1 with straight-through gradients performs similarly to SIMPLE (Ahmed et al., 2023), since they essentially compute the same gradient estimate, but Algorithm 1 is able to scale to larger instances. Algorithm 2 significantly improves on the original relaxed top-k sampling of Xie and Ermon (2019) by using sigmoid top-k, achieving both better results and scalability. We also evaluate the wall-clock time of our proposed methods in Figure 3 and different root-finding methods for sigmoid top-k in §A.

5 Conclusion

In this work, we proposed a framework for differentiable top-k by generalizing from one-hot to k-hot. By broadening the perspective to both relaxations and sampling, we identify top-k relaxations and πps sampling as key components of multiple algorithms. These components, along with a principled straight-through estimator, pave the way for future work on improved estimators. Finally, we proposed sigmoid top-k, an efficient and fully differentiable generalization of softmax that scales to larger problems. In total, our work establishes a foundation for more powerful and scalable differentiable top-k.

[2]We note that the similar method of Grafström (2009) iterates until the sample is exactly k-hot instead.
[3]This was proposed earlier in the sampling literature, see Chen et al. (1994).

Author contributions

Klas Wijk conceived and designed the study, developed the methodology, performed the experiments, and wrote the paper. Ricardo Vinuesa and Hossein Azizpour supervised the work and offered feedback.

Acknowledgements

This work was supported by the Swedish e-Science Research Centre (SeRC). The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS). Klas Wijk thanks Anton Grafström for helpful discussions.

References

Kareem Ahmed, Zhe Zeng, Mathias Niepert, and Guy Van den Broeck. SIMPLE: A Gradient Estimator for k-Subset Sampling. In *International Conference on Learning Representations*, 2023.

Nibia Aires. Algorithms to Find Exact Inclusion Probabilities for Conditional Poisson Sampling and Pareto ps Sampling Designs. *Methodology And Computing In Applied Probability*, 1(4):457–469, 1999.

Man Shun Ang, Jianzhu Ma, Nianjun Liu, Kun Huang, and Yijie Wang. Fast Projection onto the Capped Simplex with Applications to Sparse Regression in Bioinformatics. In *Advances in Neural Information Processing Systems*, 2021.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv preprint arXiv:1308.3432*, 2013.

Mathieu Blondel, André F. T. Martins, and Vlad Niculae. Learning Classifiers with Fenchel-Young Losses: Generalized Entropies, Margins, and Algorithms. In *International Conference on Artificial Intelligence and Statistics*, 2019.

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and Modular Implicit Differentiation. In *Advances in Neural Information Processing Systems*, 2022.

Kenneth E. W. Brewer. A Simple Procedure for Sampling πpswor. *Australian Journal of Statistics*, 17(3):166–172, 1975.

Xiang-Hui Chen, Arthur P. Dempster, and Jun S. Liu. Weighted Finite Population Sampling to Maximize Entropy. *Biometrika*, 81:457–469, 1994.

Anton Grafström. Repeated Poisson sampling. *Statistics & Probability Letters*, 79(6):760–764, 2009.

Anton Grafström. Entropy of unequal probability sampling designs. *Statistical Methodology*, 7(2): 84–97, 2010.

Iris A. M. Huijben, Bastiaan S. Veeling, and Ruud J. G. van Sloun. Deep Probabilistic Subsampling for Task-Adaptive Compressed Sensing. In *International Conference on Learning Representations*, 2019.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*, 2014.

Weiwei Kong, Walid Krichene, Nicolas Mayoraz, Steffen Rendle, and Li Zhang. Rankmax: An Adaptive Projection Alternative to the Softmax Function. In *Advances in Neural Information Processing Systems*, 2020.

Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic Beams and Where To Find Them: The Gumbel-Top-k Trick for Sampling Sequences Without Replacement. In *International Conference on Machine Learning*, 2019.

Wouter Kool, Herke van Hoof, and Max Welling. Ancestral Gumbel-Top-k Sampling for Sampling Without Replacement. *Journal of Machine Learning Research*, 21:1–36, 2020.

Yann LeCun, Corinna Cortes, and Chris Burges. The MNIST Database of Handwritten Digits, 1998. URL `http://yann.lecun.com/exdb/mnist/`.

Liyuan Liu, Chengyu Dong, Xiaodong Liu, Bin Yu, and Jianfeng Gao. Bridging Discrete and Backpropagation: Straight-Through and Beyond. In *Advances in Neural Information Processing Systems*, 2023.

Anders Lundquist. Contributions to the Theory of Unequal Probability Sampling. Master's thesis, Umeå University, 2009.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *International Conference on Learning Representations*, 2017.

Andre Martins and Ramon Astudillo. From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. In *International Conference on Machine Learning*, 2016.

Adeel Pervez, Phillip Lippe, and Efstratios Gavves. Scalable Subset Sampling with Neural Conditional Poisson Networks. In *International Conference on Learning Representations*, 2023.

Bengt Rosén. On sampling with probability proportional to size. *Journal of Statistical Planning and Inference*, 62(2):159–191, 1997.

M.R. Sampford. On sampling without replacement with unequal probabilities of selection. *Biometrika*, 54(3-4):499–513, 1967.

Michael Eli Sander, Joan Puigcerver, Josip Djolonga, Gabriel Peyré, and Mathieu Blondel. Fast, Differentiable and Sparse Top-k: a Convex Analysis Perspective. In *International Conference on Machine Learning*, 2023.

Yves Tillé. *Sampling Algorithms*. Springer New York, 2006.

Yves Tillé. Remarks on some misconceptions about unequal probability sampling without replacement. *Computer Science Review*, 47:100533, 2023.

Weiran Wang and Canyi Lu. Projection onto the capped simplex. *arXiv preprint arXiv:1503.01002*, 2015.

Klas Wijk, Ricardo Vinuesa, and Hossein Azizpour. SFESS: Score Function Estimators for k-Subset Sampling. In *International Conference on Learning Representations*, 2025.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Sang Michael Xie and Stefano Ermon. Reparameterizable Subset Sampling via Continuous Relaxations. In *International Joint Conference on Artificial Intelligence*, 2019.

Yujia Xie, Hanjun Dai, Minshuo Chen, Bo Dai, Tuo Zhao, Hongyuan Zha, Wei Wei, and Tomas Pfister. Differentiable Top-k with Optimal Transport. In *Advances in Neural Information Processing Systems*, 2020.

F. Yates and P. M. Grundy. Selection without replacement from within strata with probability proportional to size. *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, 15(2):253–261, 1953.

A Sigmoid top-k

Proof of existence and uniqueness of c.

Proof. The existence of $c \in \mathbb{R}$ such that

$$\sum_{i=1}^{n} \sigma(x_i + c) = k, \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ and $k \in (0, n)$ is easily seen using the intermediate value theorem. $f(c) = \sum_{i=1}^{n} \sigma(\boldsymbol{x} + c)$ is continuous, since it is a sum of continuous functions. As $c \to -\infty$, $f(c) \to 0$, and as $c \to \infty$, $f(c) \to n$. By the intermediate value theorem, there exists c such that $f(c) = k$, since $0 < k < n$. The uniqueness of this solution follows from the fact that $f(c)$ is strictly increasing, which in turn follows from it being a sum of strictly increasing functions. □

Derivation of derivatives. First, we use implicit differentiation to derive derivatives of c from the sum-constraint.

$$\sum_{i=1}^{n} \sigma(x_i + c) = 0 \tag{2}$$

We differentiate the sum-constraint with respect to x_j:

$$\frac{\partial}{\partial x_j} \textstyle\sum_{i=1}^{n} \sigma(x_i + c(x, k)) = 0 \tag{3}$$

$$\textstyle\sum_{i=1}^{n} \sigma'(x_i + c) \left(\frac{\partial x_i}{\partial x_j} + \frac{\partial c}{\partial x_j} \right) = 0. \tag{4}$$

Here, $\frac{\partial x_i}{\partial x_j} = 1$ for $i = j$ and 0 otherwise. The equation simplifies as

$$\sigma'(x_j + c) + (\textstyle\sum_{i=1}^{n} \sigma'(x_i + c)) \dfrac{\partial c}{\partial x_j} = 0 \tag{5}$$

$$\frac{\partial c}{\partial x_j} = -\frac{\sigma'(x_j + c)}{\sum_{i=1}^{n} \sigma'(x_i + c)} \tag{6}$$

We differentiate the sum-constraint with respect to k:

$$\frac{\partial}{\partial k} \textstyle\sum_{i=1}^{n} \sigma(x_i + c(x, k)) = 1 \tag{7}$$

$$\frac{\partial c}{\partial k} \textstyle\sum_{i=1}^{n} \sigma'(x_i + c) = 1 \tag{8}$$

$$\frac{\partial c}{\partial k} = \frac{1}{\sum_{i=1}^{n} \sigma'(x_i + c)}. \tag{9}$$

Now, we move on to differentiating the function

$$\sigma_k(\boldsymbol{x}) = \sigma(\boldsymbol{x} + c\mathbf{1}). \tag{10}$$

We differentiate with respect to x_i. Using the chain rule:

$$\frac{\mathrm{d}\sigma_k(\boldsymbol{x})_i}{\mathrm{d}x_j} = \sigma'(x_i + c) \left(\frac{\partial x_i}{\partial x_j} + \frac{\partial c}{\partial x_j} \right). \tag{11}$$

Again, $\frac{\partial x_i}{\partial x_j} = 1$ for $i = j$ and 0 otherwise. We substitute our previously derived $\frac{\partial c}{\partial x_j}$,

$$\frac{\mathrm{d}\sigma_k(\boldsymbol{x})_i}{\mathrm{d}x_j} = \begin{cases} \sigma'(x_i + c) - \sigma'(x_i + c) \dfrac{\sigma'(x_i + c)}{\sum_{l=1}^{n} \sigma'(x_l + c)} & \text{if } i = j, \\ -\sigma'(x_i + c) \dfrac{\sigma'(x_j + c)}{\sum_{l=1}^{n} \sigma'(x_l + c)} & \text{if } i \neq j. \end{cases} \tag{12}$$

Or, using vector notation,

$$\frac{\mathrm{d}\sigma_k(\boldsymbol{x})}{\mathrm{d}x} = \operatorname{diag}(\sigma'(\boldsymbol{x} + c)) - \frac{\sigma'(\boldsymbol{x} + c)\, \sigma'(\boldsymbol{x} + c)^\top}{\sum_{i=1}^{n} \sigma'(x_i + c)} \tag{13}$$

Finally, we differentiate with respect to k and substitute $\frac{\partial c}{\partial k}$,

$$\frac{\mathrm{d}\sigma_k(\boldsymbol{x})}{\mathrm{d}k} = \sigma'(\boldsymbol{x}+c)\,\frac{\partial c}{\partial k} \tag{14}$$

$$\frac{\mathrm{d}\sigma_k(\boldsymbol{x})}{\mathrm{d}k} = \frac{\sigma'(\boldsymbol{x}+c)}{\sum_{i=1}^{n}\sigma'(x_i+c)} \tag{15}$$

Proof of Proposition 1.

Proof. Properties a) to c) are easily seen:

a) Follows directly from $\sigma(x_i+c)$ being strictly increasing.

b) $\sigma_k(\boldsymbol{x}+\alpha) = \sigma(\boldsymbol{x}+\alpha+c) = \sigma_k(\boldsymbol{x})$.

c) $\sigma^{-1}(\sigma_k(\boldsymbol{x})) = \sigma^{-1}(\sigma(\boldsymbol{x}+c)) = \boldsymbol{x}+c$.

d) $\lim_{\tau\to\infty}\frac{x_i}{\tau} = 0$ so the sum-constraint requires $\lim_{\tau\to\infty}\frac{x_i+c}{\tau} = \sigma^{-1}(k/n)$.

Finally, for e) we know that

$$\lim_{\tau\to 0}\sigma\left(\frac{x}{\tau}\right) = \begin{cases}1, & x>0\\ 0, & x<0\end{cases} \tag{16}$$

Next, the sum-constraint $\sum_{i=1}^{n}\sigma\left(\frac{x_i+c}{\tau}\right) = k$, so $\lim_{\tau\to 0}\sigma\left(\frac{x_i+c}{\tau}\right) = 1$ for exactly k indices. Consider x_i in sorted order

$$x_1 > \cdots > x_k > x_{k+1} > \cdots > x_n,$$

Because the sigmoid function is increasing, the order is preserved for its outputs. It follows that all $\lim_{\tau\to 0}\sigma\left(\frac{x_i+c}{\tau}\right) = 1$ for $i \geq k$ and 0 otherwise.

$$\underbrace{\sigma\left(\frac{x_1+c}{\tau}\right) > \cdots > \sigma\left(\frac{x_k+c}{\tau}\right)}_{\to 1} > \underbrace{\sigma\left(\frac{x_{k+1}+c}{\tau}\right) > \cdots > \sigma\left(\frac{x_n+c}{\tau}\right)}_{\to 0},$$

This is exactly top-$k(\boldsymbol{x})$. □

Proof of Proposition 2.

Proof. We consider the optimization problem

$$\sigma_k(\boldsymbol{x}) = \operatorname*{arg\,max}_{\boldsymbol{\pi}\in\Delta_k^{n-1}} \boldsymbol{x}^T\boldsymbol{\pi} + \sum_{i=1}^{n} H(\pi_i) \tag{17}$$

$$= \operatorname*{arg\,min}_{\substack{\boldsymbol{\pi}\in(0,1)^n\\ \sum_{i=1}^{n}\pi_i = k}} \sum_{i=1}^{n} \log(1-\pi_i) + \pi_i \log\frac{\pi_i}{1-\pi_i} - x_i\pi_i \tag{18}$$

Here, we limited $\boldsymbol{\pi}$ to $(0,1)^n$ for simplicity. First, we note that the feasible set is convex as it is the intersection of two convex sets. For each i, the elementwise objective and its derivatives

$$f_i(\pi_i) = \log(1-\pi_i) + \pi_i\log\frac{\pi_i}{1-\pi_i} - x_i\pi_i \tag{19}$$

$$f_i'(\pi_i) = \log\frac{\pi_i}{1-\pi_i} - x_i \tag{20}$$

$$f_i''(\pi_i) = \frac{1}{\pi_i(1-\pi_i)} \tag{21}$$

$f_i''(\pi_i) > 0$ over the feasible set, so $f_i(\pi_i)$ is convex. The sum of convex functions is convex, so the original objective's derivative is convex. Since the feasible set and objective are convex, the

optimization problem is convex. This implies that it has a unique solution. Next, we have the Lagrangian for $\boldsymbol{\pi} \in (0,1)^n$

$$\mathcal{L}(\boldsymbol{\pi}, c) = \sum_{i=1}^{n} \left(\log(1-\pi_i) + \pi_i \log \frac{\pi_i}{1-\pi_i} - x_i \pi_i \right) + c \left(\sum_{i=1}^{n} \pi_i - k \right) \tag{22}$$

The elementwise stationarity condition $\frac{\partial \mathcal{L}}{\partial \pi_i} = 0$ is

$$\sigma^{-1}(\pi_i) = x_i + c \tag{23}$$

$$\pi_i = \sigma(x_i + c) \tag{24}$$

which retrieves the scalar shift in sigmoid top-k. Finally, at the stationarity condition, the sum constraint becomes

$$\sum_{i=1}^{n} \sigma(x_i + c) - k = 0, \tag{25}$$

exactly the equation that defines c in sigmoid top-k. It has a unique solution, which we have already proven in this appendix. □

Root-finding. The root-finding problem can be solved using the bisection method, which guarantees linear convergence. The root is bracketed by $\pm(\max_i |x_i| + \sigma^{-1}(1-\epsilon))$ for a small ϵ that saturates the sigmoid. Finding an acceptable c can be impossible with single-precision floats, especially for large n. We find that using double-precision just in the bisection resolves this issue at the cost of slightly increased memory use and computational overhead. The resulting $\sigma(\boldsymbol{x} + c)$ is cast back to single-precision.

Faster convergence can be achieved using a hybrid method. Since the sigmoid is an autonomous function, computing its derivatives is inexpensive. Newton's method can be much faster than bisection with quadratic convergence. However, Newton's method can diverge. We can keep the bisection method's linear worst-case performance by combining it with Newton's method. We found that simply evaluating both steps and picking the one that reduces the error the most works well. Halley's method uses the second derivative to achieve cubic convergence. We note that the first derivative is always positive, and the second is only zero if all x_i are equal, in which case the root is trivial. Higher-order Householder's methods are possible, but we don't see any improvement beyond Halley's method. Finally, Newton's method benefits from a good starting guess. We pick a starting guess based on two complementary heuristics. The logit-heuristic

$$c \approx \sigma^{-1}\left(\frac{k}{n}\right) - \frac{1}{n}\sum_{i=1}^{n} x_i,$$

is accruate when the x_i are approximately equal to their mean. The quantile-heurisitc

$$c \approx -x_{(\frac{n-k}{n})},$$

where the subscript denotes the $(\frac{n-k}{n})$-th quantile of $\boldsymbol{x}$, is accraute when the x_i are spread out. Picking the one with the lowest error results in a robust starting guess. Figure 4 shows a comparison of the root-finding approaches discussed above.

Comparison. Figure 5 shows a visual comparison of sigmoid top-k and some existing differentiable relaxations of top-k for $k = 1$. The methods are: the rescaled sigmoid used in Pervez et al. (2023), sparsemax (Martins and Astudillo, 2016), and SOFT top-k (Xie et al., 2020). Because $k = 1$, they can also be compared against softmax.

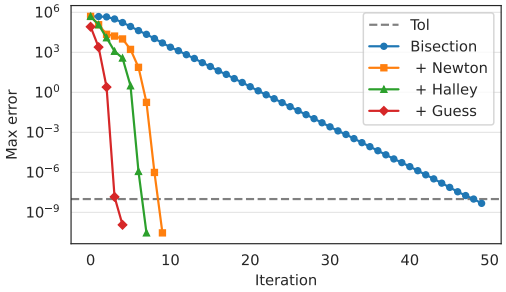

Figure 4: **Root-finding benchmark.** We solve a batch of 100 root-finding problems with $n = 10^6$ and tolerance 10^{-8}. The instances are random with $x_i \sim \mathcal{N}(0, \sigma^2)$ where σ is evenly spaced from 0.1 to 5 across the batches and $k \sim \mathcal{U}(1, n-1)$ for each batch. The sigmoid function σ and standard deviation σ should not be confused. The graph shows the maximum error across batches at each iteration.

Table 1: **Grid search results.** We evaluate different root-finding algorithms using the same experimental setup as in Figure 4, except with $n = 10^4$. The table shows the mean and standard deviation of 100 repetitions with different random batches, with the three fastest wall-clock times indicated on the right. Halley's method and choosing the best starting guess from both the logit and quantile heuristics gives both the fewest number of iterations and the fastest wall-clock time.

Step	Guess	Iterations	Time [ms]	
Bisection	–	43.0 ± 0.00	9.21 ± 1.58	
Newton	Zero	8.16 ± 0.44	5.62 ± 1.19	
	Quantile	7.27 ± 0.73	5.18 ± 2.54	
	Logit	7.34 ± 0.86	4.55 ± 1.04	
	Both	5.34 ± 0.47	3.90 ± 0.84	3rd
Halley	Zero	5.82 ± 0.41	5.52 ± 1.63	
	Quantile	4.50 ± 0.56	4.29 ± 0.98	
	Logit	4.32 ± 0.53	3.66 ± 0.87	2nd
	Both	3.15 ± 0.36	3.25 ± 0.79	1st

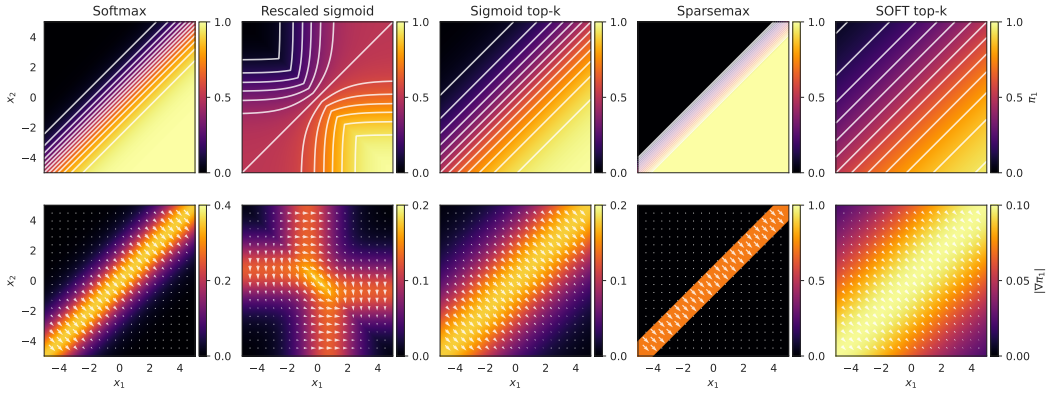

Figure 5: **Differentiable projections onto the 1D simplex.** The value (top row) and gradient (bottom row) of differentiable projections onto the simplex. In this special case of the k-capped simplex, when $k = 1$, we can compare against softmax. However, softmax is not applicable to the general problem. Although the functions are vector-valued, mapping x_1 and x_2 to π_1 and π_2, we only need to consider π_1, since $\pi_2 = 1 - \pi_1$.

B Sampling

Sampling designs. Perhaps the most common design used in machine learning is weighted random sampling (Yates and Grundy, 1953), which is often implemented as Gumbel top-k sampling (Kool et al., 2019). Both the papers' original authors and Tillé (2023) point out that its actual inclusion probabilities do not equal its parameters, i.e.,

$$p(x_i) = \pi_i,$$

does *not* hold. Worse yet, the actual inclusion probabilities are intractable, which limits both interpretability and modeling (you cannot, e.g., compute a KL-divergence with unknown probabilities). Another design used in machine learning is conditional Poisson sampling. It is the independent Bernoulli distribution[4] conditioned on $\sum_{i=1}^{n} x_i = k$. This design does not produce exact inclusion probabilities either. However, the actual inclusion probabilities can be computed. New parameters that produce the desired inclusion probabilities can be computed via numerical optimization or approximated analytically to correct the design (Chen et al., 1994; Aires, 1999; Tillé, 2006; Lundquist, 2009). There are many other sampling designs that have seen little to no use in machine learning thus far, some of which produce exact inclusion probabilities (Sampford, 1967; Brewer, 1975; Rosén, 1997).

Sampling procedures. A sampling procedure is an algorithm that implements a sampling design. As mentioned previously, the same design may be implemented by multiple sampling procedures. In a machine learning setting, we often require rapid sampling, making efficient procedures vital. We summarize some common types of procedures. **Draw-by-draw** procedures add one element to the sample at a time. They are often easy to implement, but have a time-complexity of $\mathcal{O}(nk)$ due to drawing k samples sequentially. **Rejection sampling** procedures make repeated attempts to accept samples based on certain criteria. For example, conditional Poisson sampling can be implemented by drawing independent Bernoulli samples until the sample's sum is k. The time complexity of these algorithms depends on the expected number of iterations until acceptance, which in turn depends on the parameter values. In a learning setting, these values change. We note that there may be a substantial risk of encountering cases with low acceptance rates as a result. **Order sampling** procedures sample ranking variables and pick the top-k of these. Assuming the ranking variables can be computed efficiently, the algorithm is as fast as top-k. Gumbel top-k is an example of such an algorithm. Not that there are many other procedures that do not fall into the categories above.

After considering these criteria, adjusted Pareto sampling (Rosén, 1997) appears to be a good option. It is a high-entropy design (Grafström, 2010) with exact inclusion probabilities (if adjusted).

[4]Independent Bernoulli sampling is known as Poisson sampling in the sampling design literature.

C Gradient estimation

Proof of Proposition 3.

Proof. We expand the proof in Liu et al. (2023) to the k-hot case. First, we write the true gradient as a sum over all k-hot vectors:

$$\nabla \coloneqq \frac{\partial}{\partial \boldsymbol{\theta}} \mathbb{E}[f(\boldsymbol{x})] = \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}} \sum_{\boldsymbol{x} \in \{0,1\}_k^n} f(\boldsymbol{x}) p(\boldsymbol{x}) = \sum_{\boldsymbol{x} \in \{0,1\}_k^n} f(\boldsymbol{x}) \frac{\mathrm{d}p(\boldsymbol{x})}{\mathrm{d}\boldsymbol{\theta}}.$$

We can rewrite this by adding and subtracting $\mathbb{E}[f(\boldsymbol{x})]$ and rearranging the sum

$$\begin{aligned}
\nabla &= \sum_{\boldsymbol{x} \in \{0,1\}_k^n} (f(\boldsymbol{x}) - \mathbb{E}[f(\boldsymbol{x})] + \mathbb{E}[f(\boldsymbol{x})]) \frac{\mathrm{d}p(\boldsymbol{x})}{\mathrm{d}\boldsymbol{\theta}} \\
&= \sum_{\boldsymbol{x} \in \{0,1\}_k^n} (f(\boldsymbol{x}) - \mathbb{E}[f(\boldsymbol{x})]) \frac{\mathrm{d}p(\boldsymbol{x})}{\mathrm{d}\boldsymbol{\theta}} + \underbrace{\sum_{\boldsymbol{x} \in \{0,1\}_k^n} \mathbb{E}[f(\boldsymbol{x})] \frac{\mathrm{d}p(\boldsymbol{x})}{\mathrm{d}\boldsymbol{\theta}}}_{=0}
\end{aligned}$$

Here, the second term is zero, since

$$\sum_{\boldsymbol{x} \in \{0,1\}_k^n} \mathbb{E}[f(\boldsymbol{x})] \frac{\mathrm{d}p(\boldsymbol{x})}{\mathrm{d}\boldsymbol{\theta}} = \mathbb{E}[f(\boldsymbol{x})] \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}} \sum_{\boldsymbol{x} \in \{0,1\}_k^n} p(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})] \frac{\mathrm{d}1}{\mathrm{d}\boldsymbol{\theta}} = 0.$$

We continue by expanding the expectation

$$\nabla = \sum_{\boldsymbol{x} \in \{0,1\}_k^n} \sum_{\boldsymbol{y} \in \{0,1\}_k^n} p(\boldsymbol{y}) (f(\boldsymbol{x}) - f(\boldsymbol{y})) \frac{\mathrm{d}p(\boldsymbol{x})}{\mathrm{d}\boldsymbol{\theta}}$$

Next, define a first-order estimator by finite difference approximation, $f(\boldsymbol{x}) - f(\boldsymbol{y}) \approx \frac{\mathrm{d}f(\boldsymbol{y})}{\mathrm{d}\boldsymbol{y}} (\boldsymbol{x} - \boldsymbol{y})$

$$\begin{aligned}
\nabla_{\text{1st-order}} &\coloneqq \sum_{\boldsymbol{x} \in \{0,1\}_k^n} \sum_{\boldsymbol{y} \in \{0,1\}_k^n} p(\boldsymbol{y}) \frac{\mathrm{d}f(\boldsymbol{y})}{\mathrm{d}\boldsymbol{y}} (\boldsymbol{x} - \boldsymbol{y}) \frac{\mathrm{d}p(\boldsymbol{x})}{\mathrm{d}\boldsymbol{\theta}} \\
&= \sum_{\boldsymbol{y} \in \{0,1\}_k^n} p(\boldsymbol{y}) \frac{\mathrm{d}f(\boldsymbol{y})}{\mathrm{d}\boldsymbol{y}} \underbrace{\sum_{\boldsymbol{x} \in \{0,1\}_k^n} \boldsymbol{x} \frac{\mathrm{d}p(\boldsymbol{x})}{\mathrm{d}\boldsymbol{\theta}}}_{= \frac{\mathrm{d}\mathbb{E}[\boldsymbol{x}]}{\mathrm{d}\boldsymbol{\theta}}} - \underbrace{\sum_{\boldsymbol{y} \in \{0,1\}_k^n} p(\boldsymbol{y}) \frac{\mathrm{d}f(\boldsymbol{y})}{\mathrm{d}\boldsymbol{y}} \boldsymbol{y} \sum_{\boldsymbol{x} \in \{0,1\}_k^n} \frac{\mathrm{d}p(\boldsymbol{x})}{\mathrm{d}\boldsymbol{\theta}}}_{=0} \\
&= \sum_{\boldsymbol{y} \in \{0,1\}_k^n} p(\boldsymbol{y}) \frac{\mathrm{d}f(\boldsymbol{y})}{\mathrm{d}\boldsymbol{y}} \frac{\mathrm{d}\mathbb{E}[\boldsymbol{x}]}{\mathrm{d}\boldsymbol{\theta}} = \mathbb{E}\left[\frac{\mathrm{d}f(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}} \frac{\mathrm{d}\mathbb{E}[\boldsymbol{x}]}{\mathrm{d}\boldsymbol{\theta}} \right]
\end{aligned}$$

Which proves

$$\mathbb{E}[\nabla_{\text{ST}}] = \nabla_{\text{1st-order}}$$

□

D Experiments

Hardware. Experiments were run using a single GPU, either an NVIDIA RTX 2080 Ti or an NVIDIA A40. The wall-clock times reported in Figure 3 and §A were recorded using an NVIDIA RTX 2080 Ti with 12 GB of VRAM.

Network architectures. For both the feature selection and VAE experiments, we use dense ReLU networks with two hidden layers of size 512 and 256 (in reversed order for the decoder). The decoder's output is passed through a sigmoid, and we use binary cross-entropy as the reconstruction loss.

Hyperparameters. We use the Adam optimizer (Kingma and Ba, 2015) with default parameters ($\beta_1 = 0.9$ and $\beta_2 = 0.999$) and no weight decay. We use a learning rate of 10^{-3} for feature selection and 10^{-4} for the VAE. We let all temperatures $\tau = 1$.

Results. In the results tables πST refers to πps sampling with a straight-through gradient estimate (Algorithm 1). Gumbel–σ_k refers to relaxed samplng with sigmoid top-k with hard samples and relaxed gradients (Algorithm 2).

Table 2: **Feature selection.** Test loss for feature selection with $n = 784$ features and $k = 50$ selections. Results are shown with one standard deviation computed from five different random seeds.

Method	MNIST	Fashion-MNIST
Xie and Ermon (2019)	0.113 ± 2.44e-03	0.300 ± 2.39e-03
NCPSS	0.134 ± 2.47e-03	0.317 ± 8.13e-03
SIMPLE	0.099 ± 9.03e-04	0.287 ± 4.08e-04
πST	0.102 ± 1.35e-03	0.291 ± 1.20e-03
Gumbel–σ_k	**0.096 ± 5.06e-04**	**0.286 ± 4.89e-04**

Table 3: **Variational autoencoders.** Test loss for a small VAE with a latent space of ten k-hot vectors ($n = 10$ and $k = 5$) and a large VAE with a single k-hot vector ($n = 1000$ and $k = 500$). Results are shown with one standard deviation computed from five different random seeds.

	$n = 10$ and $k = 5$ ($\times 10$)		$n = 1000$ and $k = 500$ ($\times 1$)	
Method	MNIST	Fashion-MNIST	MNIST	Fashion-MNIST
Xie and Ermon (2019)	98.26 ± 2.63e-00	234.94 ± 4.42e-01	–	–
NCPSS	82.88 ± 1.87e-01	226.37 ± 3.27e-01	66.25 ± 2.12e-01	223.94 ± 1.71e-00
SIMPLE	**81.73 ± 2.07e-01**	**225.16 ± 1.22e-01**	–	–
πST	82.78 ± 2.30e-01	226.29 ± 2.49e-01	66.04 ± 2.06e-01	**222.38 ± 7.96e-01**
Gumbel–σ_k	82.06 ± 2.12e-01	225.80 ± 2.06e-01	**63.46 ± 1.73e-01**	217.59 ± 2.74e-01