CS 6787 Final Project Report: Optimal Optimizers for Learning Discrete Choice Models

Kiran Tomlinson

Abstract

Discrete choice models have recently attracted increased attention in the machine learning community; however, little attention has been paid to the optimization methods used to train these models. Unlike large neural networks, discrete choice models typically have convex objectives, and many choice datasets are small enough that minibatching is not necessary—as a result, the state-of-the-art optimizers for deep learning may not translate to discrete choice. Here, we perform a thorough empirical evaluation of twelve PyTorch optimizers on four discrete choice models and six choice datasets, with hyperparamers selected by random search. While Adam has been widely used in the literature, it is significantly outperformed by L-BFGS and Rprop, the two best methods in this study. L-BFGS typically converges within only five epochs and 2-10x faster than Adam in wall-clock time. For one model, the CDM, only L-BFGS and Rprop converge within 500 epochs. L-BFGS and Rprop also require far less hyperparameter tuning than competing methods to achieve optimal performance. These insights will allow faster and more accurate training of discrete choice models in future research.

1. Introduction

Many of the most important actions people take are *choices*: they determine which products are successful, what universities people attend, how public resources (e.g., roads and parks) are used, and what clicks people make online. A *discrete choice* setting occurs whenever the choice is made from a finite set of alternatives (for instance, selecting which movie to watch on Netflix). Due to the ubiquity of this setting, discrete choice has received significant attention in economics (seminal works include (McFadden, 1974; Tversky, 1972; Tversky & Simonson, 1993)) and, more recently, computer science (Maystre & Grossglauser, 2015; Benson et al., 2016; Seshadri et al., 2019; Rosenfeld et al., 2020; Tomlinson & Benson, 2020a).

The fundamental problem in discrete choice is to predict future choices given prior observations. The standard approach to prediction is to postulate a probabilistic model for choice, fit that model to data, and use the learned model to make predictions in new instances. For many models, the estimation process involves minimizing a convex loss function, which has typically been done using the Adam (Kingma & Ba, 2015) optimizer (for instance, in (Seshadri et al., 2019; Tomlinson & Benson, 2020a;b)), likely due to its popularity for training deep-learning models. Training these models, especially on large datasets, can be quite time-intensive, so identifying the best optimizers is of significant practical value.

This report begins with a brief introduction to discrete choice models. We then provide empirical results evaluating the performance of every[1] optimizer built into the PyTorch library (Paszke et al., 2019). We find that L-BFGS (Byrd et al., 1995) and Rprop (Riedmiller & Braun, 1993) perform much better than Adam (and other related methods, such as Adadelta (Zeiler, 2012), Adagrad (Duchi et al., 2011), and vanilla gradient descent with momentum). In addition to converging faster and across more datasets, these two optimizers also require almost no hyperparameter tuning to achieve excellent performance, in stark constrast with other methods. This surprising result indicates that significant time-savings are possible in future discrete choice research by switching optimizers.

2. Discrete choice models

We begin with some notation and then describe the four discrete choice models used in this study. Let $\mathcal{U}$ denote the universe of all items. A discrete choice instance consists of a *choice set* $C \subseteq \mathcal{U}$ from which someone selects one item $i \in C$. A *discrete choice model* specifies choice probabilities $\Pr(i \mid C)$ for each item $i \in C$. A *discrete choice dataset* $\mathcal{D}$ is a collection of tuples (i, C) where i was selected from C.

2.1. Logit

The prototypical discrete choice model is the *multinomial logit* (McFadden, 1974) (for simplicity, we will refer to is as

[1]Except two that are equivalent to Adam for our purposes: AdamW and SparseAdam.

the *logit* model). In a logit, each item i has a *utility* $u_i \in \mathbb{R}$ and the choice probabilities are a softmax of the utilities:

$$\Pr(i \mid C) = \frac{\exp(u_i)}{\sum_{j \in C} \exp(u_j)}. \tag{1}$$

The logit is especially appealing to economists because it is compatible with rational utility-maximization. If individuals observe random utilities $u_i + \epsilon_i$ for each item $i \in C$ (where each ϵ_i is i.i.d. standard Gumbel distributed) and choose the item $\arg\max_i u_i + \epsilon_i$, then the resulting choice probabilities are exactly eq. (1).

The logit model can be learned from data using *maximum-likelihood estimation* (MLE). The likelihood of the parameters u given a dataset $\mathcal{D}$ is the probability of observing $\mathcal{D}$ if the parameters are u:

$$\mathcal{L}(u; \mathcal{D}) = \prod_{(i,C) \in \mathcal{D}} \frac{\exp(u_i)}{\sum_{j \in C} \exp(u_j)}. \tag{2}$$

The maximum-likelihood parameters are given by $u^* = \arg\max_u \mathcal{L}(u; \mathcal{D})$. We can find u^* by noticing that maximizing $\mathcal{L}(u; \mathcal{D})$ is equivalent to minimizing the negative log-likelihood (NLL) $-\ell(u; \mathcal{D}) = -\log \mathcal{L}(u; \mathcal{D})$. Moreover, the NLL of the logit model is convex (by the convexity of log-sum-exp):

$$\begin{aligned} -\ell(u; \mathcal{D}) &= -\log \prod_{(i,C) \in \mathcal{D}} \frac{\exp(u_i)}{\sum_{j \in C} \exp(u_j)} && (3) \\ &= \sum_{(i,C) \in \mathcal{D}} -\log \frac{\exp(u_i)}{\sum_{j \in C} \exp(u_j)} && (4) \\ &= \sum_{(i,C) \in \mathcal{D}} \left(-u_i + \log \sum_{j \in C} \exp(u_j) \right). && (5) \end{aligned}$$

In fact, $-\ell(u; \mathcal{D})$ has a Lipschitz-continuous gradient: the gradient of log-sum-exp is the softmax function, which is bounded above by 1, so every partial derivative of $-\ell(u; \mathcal{D})$ is bounded by a constant. We can therefore be confident that gradient descent (and related methods) will converge to a global optimum, given the correct hyperparameters.

2.2. Context effects and the CDM

While the logit is the logical first step in choice modeling, it is also quite restrictive. In particular, it adheres to the *independence of irrelevant alternatives* (IIA) (Luce, 1959), the assumption that relative choice probabilities are conserved across choice sets:

$$\frac{\Pr(i \mid C)}{\Pr(j \mid C)} = \frac{\Pr(i \mid C')}{\Pr(j \mid C')} \tag{6}$$

for all C, $i \in C$, $j \in C$, and $C' \supseteq C$. While this assumption is intuitively pleasing, a long line of experimental work shows that IIA is very often violated in practice (Huber et al., 1982; Simonson, 1989; Simonson & Tversky, 1992). Violations of IIA are termed *context effects*, and recent work in machine learning for discrete choice has focused on learnable models accounting for context (Seshadri et al., 2019; Rosenfeld et al., 2020; Bower & Balzano, 2020; Tomlinson & Benson, 2020a). One of these models is the *context-dependent random utility model* (CDM) (Seshadri et al., 2019). The CDM can be viewed as a fist-order approximation to a model accounting for arbitrary choice distributions.

Under the CDM, each item $j \in C$ exerts a pull p_{ij} on the utility of item i. The total utility of each item is the sum of pulls over the choice set: $\sum_{j \in C \setminus i} p_{ij}$. This results in the following model:

$$\Pr(i \mid C) = \frac{\exp(\sum_{k \in C \setminus i} p_{ik})}{\sum_{j \in C} \exp(\sum_{k \in C \setminus j} p_{jk})}. \tag{7}$$

Like the logit model, the CDM enjoys a convex NLL, making it easy to learn.

2.3. Conditional logit

In some datasets, items are described by a set of features rather than a unique identifier (or there may be too many unique items to learn per-item parameters). Let $y_i \in \mathbb{R}^d$ denote the vector of features of item i. To adapt the logit model to this setting, we assume each item's utility u_i takes the linear form $\theta^T y_i$ for some parameter $\theta \in \mathbb{R}^d$. This gives the *conditional logit* (McFadden, 1974), which looks almost exactly like multinomial logistic regression (with the exception that the denominator only sums over items in the choice set):

$$\Pr(i \mid C) = \frac{\exp(\theta^T y_i)}{\sum_{j \in C} \exp(\theta^T y_j)}. \tag{8}$$

Again, we can learn θ by minimizing $-\ell(\theta; \mathcal{D})$.

2.4. LCL

Just as the CDM extends the logit model to account for context effects, the *linear context logit* (LCL) (Tomlinson & Benson, 2020a) extends conditional logit. In the LCL, the parameter θ is replaced by $\theta + Ay_C$, where $y_C = 1/|C| \sum_{i \in C} y_i$ is the mean feature vector over C and the context effect matrix $A \in \mathbb{R}^{d \times d}$ describes the effect of each feature on the importance of every other feature (for instance, in a choice set of hotels with high mean price, star rating might be a more important consideration). The LCL has choice probabilities:

$$\Pr(i \mid C) = \frac{\exp([\theta + Ay_C]^T y_i)}{\sum_{j \in C} \exp([\theta + Ay_C]^T y_j)}. \tag{9}$$

Optimizer	**Hyperparameters Tested**
`Adadelta` (Zeiler, 2012)	`rho` $\in [0.8, 1]$, `lr` $\in^* [10^{-2}, 10^2]$
`Adagrad` (Duchi et al., 2011)	`lr` $\in^* [10^{-4}, 10^0]$
`Adam` (Kingma & Ba, 2015)	`lr` $\in^* [10^{-5}, 10^{-1}]$
`Adam` w/ `amsgrad` (Reddi et al., 2018)	`lr` $\in^* [10^{-5}, 10^{-1}]$
`AdamW` (Loshchilov & Hutter, 2018)	*not tested*
`SparseAdam`	*not tested*
`Adamax` (Kingma & Ba, 2015)	`lr` $\in^* [10^{-5}, 10^{-1}]$
`ASGD` (Polyak & Juditsky, 1992)	`lr` $\in^* [10^{-4}, 10^0]$, `lambd` $\in^* [10^{-6}, 10^{-2}]$, `alpha` $\in [0.5, 1]$, `t0` $\in^* [10^4, 10^8]$
`LBFGS` (Byrd et al., 1995)	`lr` $\in^* [10^{-2}, 10^2]$
`RMSprop` (Hinton, 2012)	`lr` $\in^* [10^{-4}, 10^{-0}]$, `alpha` $\in [10^{0.8}, 10^1]$
`Rprop` (Riedmiller & Braun, 1993)	`lr` $\in^* [10^{-4}, 10^0]$
`SGD`	`lr` $\in^* [10^{-4}, 10^0]$
`SGD` w/ `momentum`	`lr` $\in^* [10^{-4}, 10^0]$, `momentum` $\in^* [10^{-5}, 10^{-1}]$
`SGD` w/ `nesterov` (Nesterov, 1983)	`lr` $\in^* [10^{-4}, 10^0]$, `momentum` $\in^* [10^{-5}, 10^{-1}]$

Table 1. Every optimizer in `torch.optim` (PyTorch 1.7.0), along with some notable variants. The second column indicates which hyperparameters we trained using random search and the range of the search ($\in$ indicates uniform sampling, while $\in^*$ indicates log-uniform sampling). Hyperparameters not listed here were left at their default values.

Just like the other models we consider, the NLL of the LCL is also convex.

3. Experimental results

3.1. Optimizers

PyTorch 1.7.0 comes with 11 optimizers, some of which have important variants (Table 1). We tested all of them, with the exception of `AdamW`, which is equivalent to `Adam` since we don't use weight decay, and `SparseAdam`, since we don't have sparse tensors. To select hyperparameters for these methods, we performed random search (Bergstra & Bengio, 2012) with either uniform or log-uniform sampling. For parameters that can span several orders of magnitude, we picked ranges of search based on the default values in PyTorch, plus or minus two orders of magnitude. For instance, if a learning rate has a default value of 10^{-2}, we sampled log-uniformly from $[10^{-4}, 10^0]$. For parameters constrained in $[0, 1]$ (such as `alpha` in RMSprop), we sampled uniformly in a range determined by inspecting the values tested in the original papers. We left less consequential hyperparameters at their default values (such as `eps` in Adadelta and the `betas` in Adam).

3.2. Datasets

We used three datasets with item features for conditional logit and LCL (SFWork, SFShop, and Sushi) and three datasets without item features for logit and CDM (Expedia, Allstate, and FeatureSushi). See Table 2 for summary statistics of these datasets. SFWork and SFShop (Koppelman & Bhat, 2006) are survey datasets of transportation choices for San Fransisco residents (travelling to work/shopping, respectively). Sushi is a survey dataset of sushi preferences (Kamishima, 2003). Expedia (Kaggle, 2013) and Allstate (Kaggle, 2014) are Kaggle datasets of online hotel and insurance shopping choices, respectively. The FeatureSushi dataset is a version of the Sushi data that uses features included in the data. We train models on the whole dataset rather than performing a train-test split, since we are only concerned with how well each optimizer can find an optimum.

Name	**Samples**	**Unique Items**	**Features**
SFWork	5029	6	–
SFShop	2150	8	–
Sushi	5000	100	–
Expedia	276593	–	5
Allstate	97009	–	16
FeatureSushi	5000	–	6

Table 2. Dataset summary.

3.3. Experimental setup

For each dataset/model/optimizer combination, we ran 32 random search trials. In each trial, we ran the optimizer for 500 epochs with no mini-batching, or until convergence (defined as gradient magnitude $< 10^{-8}$). We selected the trial that converged fastest (in wall-clock time), or the one with smallest NLL after 500 epochs if no trial converged. We measured the running time, iterations to convergence, and loss.

We were interested in three questions: (1) which optimizers consistently converge, (2) which optimizers are fastest, and

(3) which optimizers are least sensitive to hyperparameters.

3.4. Results

Table 3 shows the wall-clock time to convergence (or 500 epoch timeout), Table 4 shows the number of iterations to convergence, and Table 5 shows the final loss.

ASGD and the three SGD variants did not converge for any of the logit or CDM runs. Only L-BFGS and Rprop converged on any dataset for the CDM. Adam, AMSGrad Adam, L-BFGS, and Rprop converged on all item-feature model/dataset pairs except LCL Expedia, for which no optimizer converged. L-BFGS and Rprop are the clear winners in terms of convergence.

L-BFGS converged fastest in all but three cases, logit Sushi (1.8s), CDM Sushi (52s), and conditional logit Expedia (15s), where it was outpaced by Rprop (0.88s, 14s) and Adagrad (7.6s), respectively. However, when it did not converge (on LCL Expedia), it took much longer to complete all 500 iterations: 59.5 minutes compared to 12.3 minutes for Rprop and 9.2 minutes for Adam. Rprop had very competitive run time in all cases except LCL Allstate, where it was almost 5 times slower to converge than L-BFGS. In terms of running time, L-BFGS is the clear winner.

L-BFGS really shines in the number of iterations to convergence. In six cases, in converged in a single step, and in another five cases in converges in less than 5 steps. The only exception was CDM Sushi, where it took 54 iterations to converge. Rprop tends to converge in fewer iterations than the other first-order methods.

In addition to these numerical results, we can also take a closer look at the behavior of each optimizer. Figure 1 displays the loss trajectories in four selected cases. Adam is far more reliant on good hyperparameter selection than L-BFGS, which converges very rapidly for a large range of learning rates (with the exception of two trials that diverge). In fact, notice that *every* L-BFGS trial that didn't immediately diverge convergd within 40 epochs for CDM SFWork. The same rapid convergence pattern also holds in the other datasets and models (with the exceptions noted above). Similarly, Rprop converges to a good solution on LCL Expedia extremely quickly regardless of the learning rate. In contrast, SGD with Nesterov momentum oscillates on this dataset for many values of the hyperparamers. In summary, L-BFGS and Rprop are far less sensitive to hyperparameters than competing methods.

4. Discussion

While Adam has been the dominant optimizer used in machine learning papers on discrete choice, it is significantly outperformed by other PyTorch optimizers, especially L-

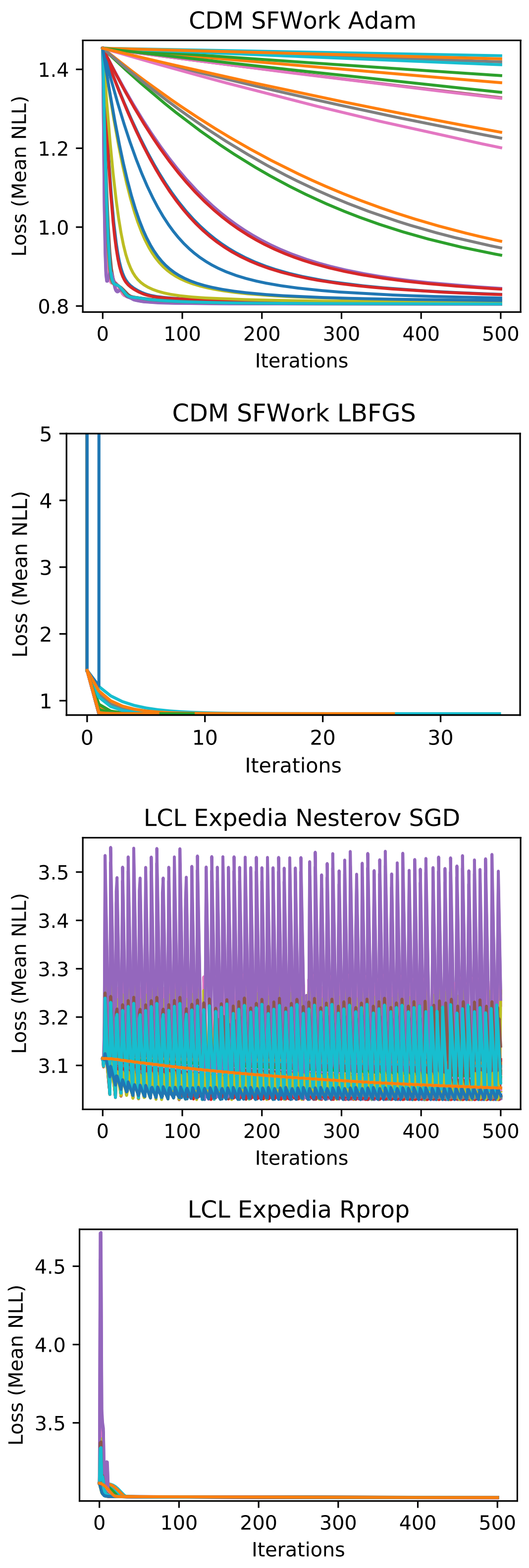

Figure 1. Loss trajectories for all 32 random search trials in selected model/dataset/optimizer combinations. L-BFGS and Rprop are much less sensitive to hyperparameters than other methods.

		Adadelta	Adagrad	Adam	AMSGrad Adam	Adamax	ASGD	LBFGS	RMSprop	Rprop	SGD	Nesterov SGD	Momentum SGD
Logit	SFWork	**0.13**	**0.12**	**0.38**	**0.3**	**0.58**	1.3	**0.053**	**0.14**	**0.06**	1.2	0.98	1.2
Logit	SFShop	**0.25**	**0.078**	**0.18**	**0.19**	**0.41**	0.71	**0.03**	**0.54**	**0.043**	0.6	0.62	0.61
Logit	Sushi	18	**3.1**	**6.5**	**5.7**	17	18	**1.8**	18	**0.88**	18	17	16
CDM	SFWork	1.4	1.4	1.4	1.5	1.5	1.4	**0.18**	1.5	**0.48**	1.4	1.1	1.5
CDM	SFShop	0.82	0.66	0.96	0.76	0.68	0.66	**0.18**	0.8	**0.68**	0.73	0.81	0.74
CDM	Sushi	19	17	19	18	19	18	**52**	16	**14**	19	19	15
CLogit	Expedia	**17**	**7.6**	**111**	**114**	**132**	484	**15**	**10**	**46**	**529**	**511**	**540**
CLogit	Allstate	**67**	**34**	**46**	**48**	**43**	184	**7.5**	188	**21**	**150**	**151**	**149**
CLogit	FeatureSushi	**2.5**	**0.36**	**0.9**	**0.9**	**1.1**	**0.71**	**0.082**	**0.58**	**0.25**	**0.29**	**0.31**	**0.3**
LCL	Expedia	495	495	490	492	498	494	3569	731	738	724	736	724
LCL	Allstate	165	167	**49**	**54**	178	167	**23**	175	**111**	152	148	162
LCL	FeatureSushi	3.9	**1.1**	**1.1**	**1.1**	**1.9**	3.8	**0.31**	3.8	**0.48**	3.8	3.8	4

Table 3. Run time (to convergence or 500 iterations). Bolded entries converged.

		Adadelta	Adagrad	Adam	AMSGrad Adam	Adamax	ASGD	LBFGS	RMSprop	Rprop	SGD	Nesterov SGD	Momentum SGD
Logit	SFWork	48	44	150	151	289	–	1	53	30	–	–	–
Logit	SFShop	195	59	144	144	366	–	1	406	31	–	–	–
Logit	Sushi	–	89	182	201	–	–	3	–	31	–	–	–
CDM	SFWork	–	–	–	–	–	–	3	–	205	–	–	–
CDM	SFShop	–	–	–	–	–	–	4	–	472	–	–	–
CDM	Sushi	–	–	–	–	–	–	54	–	460	–	–	–
CLogit	Expedia	14	7	116	117	135	–	1	6	31	370	369	369
CLogit	Allstate	176	98	123	124	119	–	1	–	59	422	421	421
CLogit	FeatureSushi	388	58	147	165	177	107	1	96	44	51	51	51
LCL	Expedia	–	–	–	–	–	–	–	–	–	–	–	–
LCL	Allstate	–	–	156	159	–	–	4	–	372	–	–	–
LCL	FeatureSushi	–	143	140	141	236	–	2	–	63	–	–	–

Table 4. Iterations to convergence.

		Adadelta	Adagrad	Adam	AMSGrad Adam	Adamax	ASGD	LBFGS	RMSprop	Rprop	SGD	Nesterov SGD	Momentum SGD
Logit	SFWork	**0.82**	**0.82**	**0.82**	**0.82**	**0.82**	1.34	**0.82**	**0.82**	**0.82**	1.37	0.82	1.25
Logit	SFShop	**1.54**	**1.54**	**1.54**	**1.54**	**1.54**	1.88	**1.54**	**1.54**	**1.54**	1.54	1.54	1.54
Logit	Sushi	1.99	**1.93**	**1.93**	**1.93**	2.29	2.29	**1.93**	2.16	**1.93**	2.23	2.26	2.20
CDM	SFWork	0.81	0.81	1.37	1.43	0.85	1.20	**0.80**	0.81	**0.80**	1.25	0.81	0.83
CDM	SFShop	1.50	1.50	1.53	1.92	1.50	1.90	**1.49**	1.50	**1.49**	1.51	1.52	1.52
CDM	Sushi	1.41	1.71	2.12	2.10	2.09	2.30	**1.13**	1.17	**1.13**	2.24	2.30	2.29
CLogit	Expedia	**3.04**	**3.04**	**3.04**	**3.04**	**3.04**	3.04	**3.04**	**3.04**	**3.04**	**3.04**	**3.04**	**3.04**
CLogit	Allstate	**1.80**	**1.80**	**1.80**	**1.80**	**1.80**	1.87	**1.80**	2.19	**1.80**	**1.80**	**1.80**	**1.80**
CLogit	FeatureSushi	**1.96**	**1.96**	**1.96**	**1.96**	**1.96**	**1.96**	**1.96**	**1.96**	**1.96**	**1.96**	**1.96**	**1.96**
LCL	Expedia	3.03	3.09	3.02	3.09	3.03	3.10	3.02	3.03	3.02	3.08	3.04	3.05
LCL	Allstate	1.83	1.85	**1.73**	**1.73**	1.73	1.73	**1.73**	1.74	**1.73**	1.75	1.75	1.84
LCL	FeatureSushi	1.95	**1.95**	**1.95**	**1.95**	**1.95**	1.95	**1.95**	2.17	**1.95**	2.07	1.96	1.95

Table 5. Final loss at convergence (bold) or 500 iterations.

BFGS and Rprop. L-BFGS and Rprop both demonstrate better and faster convergence than alternative methods. Using either of these methods can significantly improve training speed, which is the primary bottleneck in running discrete choice experiments. Additionally, these two methods are very robust to different hyperparameter values and converge rapidly without extensive tuning.

4.1. Limitations

We did not perform any minibatching, which may explain the worse performance of modern methods like Adam designed with minibatching in mind. Vanilla L-BFGS does not perform well with minibatching (Berahas et al., 2016), so it is possible that the results would reverse if the datasets were large enough that minibatching was necessary. Additionally, it is possible that some of our hyperparameter ranges were too restrictive if the PyTorch defaults are poorly calibrated. It is also possible that random search got unlucky in some cases (although the consistent performance of the top methods means this probably has a negligible effect on our overall results). Further, all of our testing was performed with no regularization for simplicity. In practice, people do tend to use regularization when training these models. It is possible that adding regularization would affect different optimizers in different ways. Finally, there may have been hyperparameters we deemed unimportant that actually can improve performance.

4.2. Future directions

The clearest next step suggested by the limitations above would be a second full run of the experiment with larger hyperparameter ranges, all hyperparameters active, and more random search trials. Experimenting on more datasets would also provide a fuller picture—perhaps one of L-BFGS and Rprop are better suited to large/small datasets or models with more/fewer parameters. Additionally, larger datsets would also support evaluation with minibatching. Adding in tests with regularization would also be very informative, although this would likely require performing train-validation-test splits to identify the correct amount of regularization to apply. Testing out-of-sample predictions would also allow us to evaluate whether some optimizers have useful regularizing effects for discrete choice models.

References

Benson, A. R., Kumar, R., and Tomkins, A. On the relevance of irrelevant alternatives. In *Proceedings of the 25th International Conference on World Wide Web*, pp. 963–973, 2016.

Berahas, A. S., Nocedal, J., and Takác, M. A multi-batch l-bfgs method for machine learning. In *Advances in Neural Information Processing Systems*, pp. 1055–1063, 2016.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

Bower, A. and Balzano, L. Preference modeling with context-dependent salient features. In *37th International Conference on Machine Learning*, 2020.

Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7), 2011.

Hinton, G. Neural networks for machine learning - lecture 6a - overview of mini-batch gradient descent, 2012.

Huber, J., Payne, J. W., and Puto, C. Adding asymmetrically dominated alternatives: Violations of regularity and the similarity hypothesis. *Journal of Consumer Research*, 9 (1):90–98, 1982.

Kaggle. Personalize expedia hotel searches - icdm 2013. https://www.kaggle.com/c/expedia-personalized-sort, 2013.

Kaggle. Allstate purchase prediction challenge. https://www.kaggle.com/c/allstate-purchase-prediction-challenge, 2014.

Kamishima, T. Nantonac collaborative filtering: recommendation based on order responses. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 583–588, 2003.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. *arXiv:1412.6980*.

Koppelman, F. S. and Bhat, C. A self instructing course in mode choice modeling: Multinomial and nested logit models, 2006.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

Luce, R. D. *Individual choice behavior: A theoretical analysis*. Courier Corporation, 1959.

Maystre, L. and Grossglauser, M. Fast and accurate inference of Plackett–Luce models. In *Advances in Neural Information Processing Systems*, pp. 172–180, 2015.

McFadden, D. Conditional logit analysis of qualitative choice behavior. *Frontiers in Econometrics*, pp. 105–142, 1974.

Nesterov, Y. E. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. *Soviet Math. Dokl.*, 27(2):372–376, 1983.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

Polyak, B. T. and Juditsky, A. B. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

Reddi, S. J., Kale, S., and Kumar, S. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

Riedmiller, M. and Braun, H. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE International Conference on Neural Networks*, pp. 586–591. IEEE, 1993.

Rosenfeld, N., Oshiba, K., and Singer, Y. Predicting choice with set-dependent aggregation. In *37th International Conference on Machine Learning*, 2020.

Seshadri, A., Peysakhovich, A., and Ugander, J. Discovering context effects from raw choice data. In *36th International Conference on Machine Learning*, 2019.

Simonson, I. Choice based on reasons: The case of attraction and compromise effects. *Journal of Consumer Research*, 16(2):158–174, 1989.

Simonson, I. and Tversky, A. Choice in context: Tradeoff contrast and extremeness aversion. *Journal of Marketing Research*, 29(3):281–295, 1992.

Tomlinson, K. and Benson, A. R. Learning interpretable feature context effects in discrete choice. *arXiv preprint arXiv:2009.03417*, 2020a.

Tomlinson, K. and Benson, A. R. Choice set optimization under discrete choice models of group decisions. In *37th International Conference on Machine Learning*, 2020b.

Tversky, A. Elimination by aspects: A theory of choice. *Psychological Review*, 79(4):281, 1972.

Tversky, A. and Simonson, I. Context-dependent preferences. *Management Science*, 39(10):1179–1189, 1993.

Zeiler, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.