Neural Utility Functions

Porter Jenkins [1], Ahmad Farag [2], J. Stockton Jenkins [3],
Huaxiu Yao [1], Suhang Wang [1], Zhenhui Li [1]
[1]Pennsylvania State University
[2]Georgia Tech University
[3]Brigham Young University

Abstract

Current neural network architectures have no mechanism for explicitly reasoning about item trade-offs. Such trade-offs are important for popular tasks such as recommendation. The main idea of this work is to give neural networks inductive biases that are inspired by economic theories. To this end, we propose Neural Utility Functions, which directly optimize the gradients of a neural network so that they are more consistent with utility theory, a mathematical framework for modeling choice among items. We demonstrate that Neural Utility Functions can recover theoretical item relationships better than vanilla neural networks, analytically show existing neural networks are not quasi-concave and do not inherently reason about trade-offs, and that augmenting existing models with a utility loss function improves recommendation results. The Neural Utility Functions we propose are theoretically motivated, and yield strong empirical results.

Introduction

Recommendation systems are ubiquitous in daily life and have been built into applications for products, music, and movies. The explicit goal of such systems is to connect users with the items they like. Implicitly, recommendation systems seek to discover user preference relationships among items. In recent years, many methods have been developed to tackle this problem at scale, such as computing empirical item-to-item similarities (Linden, Smith, and York 2003), matrix factorization (Koren and Bell 2009; Zhang et al. 2019), and most recently deep neural networks (He et al. 2017; Chen et al. 2019; Guo et al. 2017; Zhang et al. 2019; Hidasi et al. 2016). While many of these methods have produced strong results, they are mostly atheoretical and are unable to reason about the economic relationships between items.

Theoretical economics provides a useful framework for reasoning about choice among items. In particular, utility theory postulates that consumers have rational preferences and that a function exists by which consumers rank items. A consumer assigns more utility to desirable items and less utility to undesirable ones. These utility functions have useful analytical properties that shed light on the trade-offs people make when choosing between alternatives. The theory also is seen in the real world. When a consumer is choosing items at a grocery store he or she is likely to choose Coke **or** Pepsi, but not both. The consumer reasons about the item relationship and decides that the two drinks are substitutes. Additionally, the consumer is likely reason about the complementary relationship between hamburgers and buns and purchase them together.

Typical recommender systems minimize an objective function of either ratings or choice prediction error. This can be viewed as a utility function since the recommendation agent learns a function that maps item alternatives to an ordinal, real-value from a user's history. Our goal in this work is to learn a function that better mirrors a user's internal utility function and to increase his or her overall utility. However, current deep learning based recommender systems cannot reason about the trade-offs between items (e.g. substitutes and complements) because they are not quasi-concave in their inputs. One key assumption of microeconomic theory is that utility functions are quasi-concave. This property ensures the slope of indifference curves are negative and that proper item trade-offs occur (Nicholson and Snyder 2012). We analytically show that a two-layer neural network is not guaranteed to be quasi-concave, and therefore common networks trained to minimize error will not have the desirable properties of theoretical utility functions.

Therefore, in this paper, we seek to integrate ideas from microeconomics and deep learning and imbue neural networks with economic inductive biases, which we call Neural Utility Functions (NUFs). NUFs are capable of reasoning about supplementary and complementary relations among sets of items. Our hypothesis is that a network capable of learning economic relationships is better suited for common tasks such as recommendation. We empirically demonstrate that Neural Utility Functions can recover theoretical substitution effects better than networks trained to only minimize errors. Additionally, we show that augmenting common recommendation architectures with our proposed NUF loss function can improve performance on the Movielens 25M and Amazon 18 datasets using both explicit and implicit feedback.

In summary, the main contributions of the paper are: First, we analytically show that neural networks are not guaranteed to be quasi-concave, and therefore are not guaranteed to discover proper item relationships in data. Second, we propose Neural Utility Functions, a novel framework for training neural network based recommender systems by constraining the parameter search space to economically desirable optima. Third, we demonstrate that Neural Utility Functions can recover theoretical item substitution effects better than vanilla neural networks. Fourth, we conduct extensive experiments on the Movielens 25M and Amazon 18 datasets and show that a variety of state-of-the-art architectures can be improved using the proposed Neural Utility framework.

Related Work

Recommendation: A great body of literature studies the recommendation problem. Many approaches have been proposed, including item-based collaborative filtering (Linden, Smith, and York 2003), matrix factorization (Koren and Bell 2009), deep learning (He et al. 2017; Chen et al. 2019; Guo et al. 2017; Zhang et al. 2019; Ying et al. 2018; Wang et al. 2018b,a), and reinforcement learning (Chen et al. 2019; Zheng et al. 2018; Zhao et al. 2018). Many current state-of-the-art methods are based on deep neural networks in part due to their ability to capture complex, non-linear user-item relationships, and learn hierarchical representations of users and items (Zhang et al. 2019). For example, (Hidasi et al. 2016) use recurrent networks to learn item representations over time. Other work seeks to identify relationships by building item graphs from raw text reviews (McAuley, Pandey, and Leskovec 2015) and images (McAuley et al. 2015). Recent work seeks to integrate economic knowledge for recommendation (Zhang et al. 2016; Zhao, Zhang, and Freidman 2017). Finally, (Rendle et al. 2009) propose Bayesian Personalized Ranking (BPR) to rank items in an implicit feedback setting. Our work differs from these approaches in that we seek to learn economic item relationships from data by constraining the gradients during the training process and is agnostic to the choice of model architecture. Our proposed Utility Prior can also be combined with other loss terms such as (Rendle et al. 2009).

Random Utility Models: In empirical microeconomics, linear or hierarchical Bayesian models are used as a random utility models for multi-class classification (Train 2003; Howell et al. 2017; Dotson et al. 2018; McFadden 1974). (Bentz and Merunka 2000) were among the first to use a feedforward neural network with softmax outputs to predict consumer choices. We aim to build on these works by proposing a neural network that explicitly reasons about item trade-offs and can scale to very large datasets.

Gradient-constrained Optimization: Recent work has studied directly optimizing gradients to enforce theoretical properties while training neural nets (Greydanus, Dzamba, and Yosinski 2019). Most existing work focuses on learning physical laws with gradient constraints.

Theory

Recommender systems try to maximize users' utility by learning a function that suggests products that a user will like. Thus, classical demand theory in economics could be a good choice to guide the design of recommender systems. Central to classical demand theory are the axioms of rational choice. Preferences are assumed to be rational if they are *complete*, *transitive*, and *continuous* (Mas-Colell and Whinston 1995).

If a preference relation is rational then a utility function exists that provides a transitive ranking of choices (Nicholson and Snyder 2012; Mas-Colell and Whinston 1995). A utility function is a mapping from the item bundle to a real-value, and is used for ranking choices.

The classical formulation of utility is defined in terms of the Utility Maximization Problem (Mas-Colell and Whinston 1995). We assume that a person has rational, continuous, and locally non-satiated preferences. For a set of item quantities, $\mathbf{x} = \{x_1, x_2, ...x_n\}$, the consumer faces the following UMP:

$$\arg\max_{\mathbf{x}} \quad U(\mathbf{x}) \quad \text{s.t.} \quad \mathbf{p}^\top \mathbf{x} \leq w \tag{1}$$

where $U(\mathbf{x})$ is the consumer's utility function, $\mathbf{p}$ is a vector of prices, and the scalar w denotes the total wealth of the consumer. Equation (1) shows that the consumer will choose the set of items that maximize utility and are also in the Walrasian budget set, $B_{p,w} = \{x \in \mathbb{R}_+^L : px \leq w\}$ (Mas-Colell and Whinston 1995). Economists are primarily concerned with using mathematical programming techniques to solve the UMP.

However, the UMP has many interesting corollaries. One such property that is useful in many contexts is known as the Marginal Rate of Substitution (MRS).

$$s(x_1, x_2) = \frac{\delta U/\delta x_1}{\delta U/\delta x_2} \tag{2}$$

Equation (2) is the ratio of the derivatives of the utility function with respect to the items x_1, and x_2. This quantity is a statement about the relative rates of change in utility between the two items, given a fixed level of utility. Mathematically, $s(x_1, x_2)$ is the slope of the indifference curve of two items (Nicholson and Snyder 2012). It reveals the amount of x_2 the consumer must gain in order to compensate for a one unit loss in x_1. More intuitively, $s(x_1, x_2)$ sheds light on the degree of substitutability between items. When $s(x_1, x_2)$ is large, a person is willing to substitute x_1 for x_2. Conversely, as $s(x_1, x_2)$ approaches 0, a person is less willing to give up x_1 for x_2.

Issues with Existing Training Techniques

In general, a fundamental assumption of consumer demand and utility theory is that a utility function, U, is continuous, differentiable, and quasi-concave in its inputs (Nicholson and Snyder 2012; Mas-Colell and Whinston 1995). In particular, quasi-concavity of U has two important corollaries: 1) an increase in inputs increases the level of utility, but at a diminishing rate; 2) the indifference curve of the utility function is quasi-convex, and therefore a decrease in one item is compensated for by an increase in some other item. Mathematically, this is represented by $-\frac{\delta U}{\delta x_1} / \frac{\delta U}{\delta x_2}$ and suggests that people reason about trade-offs between items.

It is well understood that training neural networks represents a non-convex optimization problem, often with highly irregular loss surfaces (Li et al. 2018). However, less obvious is how a network behaves as function of its inputs. We expect that most neural networks are not concave due to use of non-linear (or piece-wise linear) activation functions. However, to the best of our knowledge no work explicitly shows this fact.

If a utility function, U is not quasi-concave there is no guarantee that proper substitution effects between items will hold. It can be shown that the probability, p, that a two-layer neural network is **not** quasi-concave if $p \geq 1 - (\frac{1}{2})^n$, where n is the dimension of the input feature vector. Consequently, when this probability is high, standard training techniques will not learn proper item relationships.

Theorem 1. *A two-layer neural network $U(\boldsymbol{x})$ is **not** quasi-concave with probability, $p \geq 1 - (\frac{1}{2})^n$ for $x \geq 0$*

Proof. Let $U(\mathbf{x})$ for $x \in \mathbb{R}^n$ be a two-layer neural network, with an $m-$dimensional hidden state, $h \in \mathbb{R}^m$, and activation function, $\sigma(\cdot)$:

$$U(\mathbf{x}) = g(f(x)), \quad \mathbf{h} = f(x) = \sigma(\mathbf{W}_1\mathbf{x}), \quad y = \mathbf{z}^\top\mathbf{h} \quad (3)$$

Assumption 1. *Assume that the weights, w_{ii} and z_i all follow Gaussian distributions with mean 0 and scale parameter σ.* [1] *(Hanson and Burr 1990)*

Lemma 1. *A function f is quasi-concave if its Hessian, $\boldsymbol{H}$, is negative semi-definite*

Lemma 2. *A matrix, $\boldsymbol{X}$, is negative-semidefinite if the scalar $\boldsymbol{u}^\top\boldsymbol{X}\boldsymbol{u} \leq 0, \forall \boldsymbol{u} \neq \boldsymbol{0}$*

From the chain rule we can get the gradient and the Hessian of U (refer to the online appendix for more details):

$$\mathbf{H} = \begin{bmatrix} z_1 w_{11}^2 \sigma''(w_{11}x_1) & \cdots & z_n w_{n1}^2 \sigma''(w_{n1}x_1) \\ \vdots & \ddots & \vdots \\ z_1 w_{1n}^2 \sigma''(w_{1n}x_n) & \cdots & z_n w_{nn}^2 \sigma'(w_{nn}x_n) \end{bmatrix} \quad (4)$$

From the Hessian matrix, $\mathbf{H}$, and lemma 2, we can show that Theorem 1 holds.

Suppose $\mathbf{u} = [1, 0, ..., 0]^\top$. It follows that

$$\mathbf{u}^\top\mathbf{H}\mathbf{u} = z_1 w_{11}^2 \sigma''(w_{11}x_1) \quad (5)$$

Note that $\sigma''(x) > 0$ for $x < 0$, and $\sigma''(x) < 0$ for $x > 0$. Assume that $x_1 \geq 0$, then the sign of $z_1 w_{11}^2 \sigma''(w_{11}x_1)$ is strictly dependent on z_1 and w_{11}. This results holds when the sigmoid and hyperbolic tangent functions are chosen for $\sigma(\cdot)$ because $\sigma''(x)$ is always non-negative for $x > 0$ (see online appendix for more details).

Lemma 3. *The necessary conditions for $\boldsymbol{H}$ to be negative semi-definite are $z_i w_{ii}^2 \sigma''(w_{ii}x_i) < 0 \iff z_i$ and w_{ii} share the same sign.*

Due to Assumption 1, the probability that w_{ii} and z_i share the same sign is:

$$\Big(P(z_i > 0) \cap P(w_{ii} > 0)\Big) \cup \Big(P(z_i < 0) \cap P(w_{ii} < 0)\Big) = \frac{1}{2} \quad (6)$$

for z_i, $w_{ii} \sim N(0, \sigma^2)$. The probability in (6) can be seen from the cdf of the Gaussian distribution.

Lemma 2 should hold for all $u \neq \mathbf{0}$. We can repeat this operation for all $\mathbf{u}^{(i)} \in \mathcal{U} = \{\mathbf{u}^{(i)} = [1, 0, ..., 0]^\top, ...\mathbf{u}^{(n)} = [0, 0, ..., 1]^\top\}$ and the result should be non-positive:

$$\bigcap_{i=1}^{n} = P(z_i w_{ii}^2 \sigma''(w_{ii}x_i) \leq 0) = (\frac{1}{2})^n \quad (7)$$

Because (7) must hold for all $\mathbf{u}$, this implies that the probability that U is quasi-concave is less than or equal to $(\frac{1}{2})^n$. Thus, it follows that the probability, p, that U is not quasi-concave is greater than or equal to the complement: $p \geq 1 - (\frac{1}{2})^n$. $\square$

It is interesting to note the effect of the dimensionality of U. As the number of input features increases, the probability that the network is quasi-concave approaches 0, $\lim_{n\to\infty} \frac{1}{2}^n = 0$, and p approaches 1. In high-dimensional feature spaces we are almost guaranteed to have a function that is not quasi-concave in its inputs.

The result in theorem 1 suggests that a typical two-layer neural network is very unlikely to be quasi-concave. If the network is not quasi-concave, it is not guaranteed to capture item trade-offs. This result motivates our hypothesis that we can improve recommendation performance through inductive biases that better mimic human reasoning.

Neural Utility Functions

In empirical microeconomics and econometrics, a great deal of previous work exists on fitting random utility functions to data (McFadden 1974; Train 2003; Dotson et al. 2018). However, the class of functions explored in these works is typically restricted to linear functions. Utility functions of this nature are not flexible enough to handle complex non-linearities found in real-world data. In other cases, the parametric assumptions these models make do not scale to modern tasks such as recommendation that consist of millions of users and items.

While deep learning architectures are usually developed irrespective to theories about rational choice, training a recommender system can be viewed as learning each user's utility function from data. In general, the primary goal of the recommendation agent is to minimize prediction error over users and items. Assuming rational preferences, error is minimized when expected utility is maximized. Additionally, in ratings-based recommendation the rating is an ordinal measurement of item quality akin to traditional models of utility. Users are assumed to choose items with higher ratings, or higher estimated utility. While classical approaches typically assume linear models to learn part-worth item utilities (Train 2003), a neural network maximizes expected utility by learning abstract representations of items and users.

Given a user u_i and an item v_j, a typical recommender is trained to minimize ratings prediction error:

$$\min_\theta \mathcal{L}_r = \sum_{(u_i, v_j) \in \mathcal{R}} (r_{ij} - \hat{r}_{ij})^2 \quad (8)$$

[1]This assumption is grounded in empirical evidence. See citation for more discussion

While many network architectures have shown good empirical results, non-linear models trained to minimize (8) alone cannot explicitly reason about item trade-offs, which are central to how people make decisions and can impact tasks such as recommendation.

In this paper, we attempt to bridge this gap. We propose a cost function that accounts for item trade-offs by imposing novel priors (equation 9). In doing so, we endow our model with the capacity to reason about item relationships and substitution effects, and the ability to learn rich item representations. Our core hypothesis is that a model that can better understand item relationships, can provide better recommendations and make better decisions.

Constructing Complement and Supplement Sets

In order to learn these item relationships, We first sample a set of *complementary*, and *supplementary* items. Given a set of users $\mathcal{U} = \{u_i : 0 < i <= m\}$ we can sample a complement set, $x_c^{(i)} = \{c_j : 0 < j <= k\}$, and a supplement set, $x_s^{(i)} = \{s_j : 0 < j <= k\}$.

For each sample, x_i, (e.g., user/item pair) we construct $x_c^{(i)}$ and $x_s^{(i)}$. We assume that if a user, u_i, chooses two items they have some degree of complementary. This assumption is grounded in the economic notion that the presence of good A will increase the demand for good B if they are complements (Nicholson and Snyder 2012). For each x_i we construct $x_c^{(i)}$ by uniformly sampling k items that the user also chose[2]. Likewise, we construct $x_s^{(i)}$ by sampling k items the user did not choose. This assumption is valid from Hicks' second law of demand: all non-complementary goods can be considered substitutes from (Nicholson and Snyder 2012). The virtue of this straightforward procedure is that it is general and can be used with any recommendation data set. We acknowledge that other sampling schemes could be used if more information is present in the data. For example, if more information is available these sets could be produced following (McAuley, Pandey, and Leskovec 2015).

Neural Utility Loss

In order to constrain the parameter search of the neural network to points that better mirror the behavior of quasi-concave utility functions, we propose the *Utility Prior* in equation 9:

$$\mathcal{L}_{utility} = \left\| \frac{\delta \mathcal{L} / \delta x_i}{\nabla_{x_c^{(i)}} \mathcal{L}} \right\|_2 - \log \left\| \frac{\delta \mathcal{L} / \delta x_i}{\nabla_{x_s^{(i)}} \mathcal{L}} \right\|_2 \tag{9}$$

The equation in (9) is reflective of the theoretical result in (2) and induces the network to learn economic relationships (see online supplement). The ratio of the derivatives of two items expresses the degree of substitutability, or trade-off, between the two items. The first term minimizes the norm of the gradient ratio between a sample x_i and its complements. The second term has the effect of maximizing the norm of the gradient ratio of the sample and its supplement set. Because we are maximizing the norm of the second term, we find that in practice it is useful to take the log to ensure the optimization procedure does not push the second term towards infinity. A log on the first term is not necessary because it is bounded from below by zero. We also experimented with a max margin rectifer (Jean et al. 2019), which will maximize the second term up to a margin, μ, but found that the formulation with a log term had fewer hyperparameters and was easier to train.

[2]For the ratings prediction task, we sample items that the user also rated

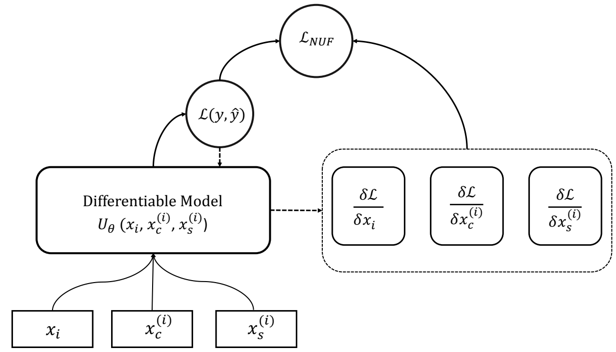

Figure 1: The training procedure for Neural Utility Functions has three main steps: 1) for each sample, x_i, a complement set, $x_c^{(i)}$, and a supplement set, $x_s^{(i)}$, are constructed and fed into the network, U; 2) we do a forward pass through the recommender model (e.g., neural network), compute the prediction loss, and the in-graph gradients with respect to the inputs; 3) we compute the Neural Utility Loss in (9) and backpropogate.

We combine the *Utility Prior* in (9) with the standard prediction loss to get the proposed NUF loss:

$$\mathcal{L}_{NUF} = \lambda \mathcal{L}_{utility} + \mathcal{L}(y, \hat{y}) \tag{10}$$

We optimize (10) to train our network. The term, $\mathcal{L}(y, \hat{y})$, represents the prediction loss between y and $\hat{y}$. Without loss of generality, any common prediction loss function can be used for $\mathcal{L}(y, \hat{y})$ (e.g., mean squared error, binary cross entropy, cross entropy). In our experiments we use both mean squared error and binary cross entropy. The strength of the contribution of the neural utility loss term is controlled by the hyperparameter lambda.

The resulting function learned by the neural network is still not guaranteed to be quasi-concave; however, due to the informative constraints imposed in the loss (equation 9), it does learn some of the same properties characteristic of quasi-concave utility functions (equation 2), namely, substitution effects and item trade-offs.

Training Procedure

One virtue of the proposed approach is that it is agnostic to network architecture. Any existing backbone model can be used in conjunction with (10). The training procedure is therefore straightforward and can be implemented with existing models. There are three main ingredients in our training procedure (see Figure 1). First, we collect samples using the sampling routine in the previous section. We feed each sample, complement set and supplement set into the parameterized neural network, $\hat{y}_i = U_\theta(x_i, x_c^{(i)}, x_s^{(i)})$. The output $\hat{y}_i$ can be any arbitrary regression or classification output (e.g., real number, binary class label, multi-class label). Second, we compute the loss of the prediction error, $\mathcal{L}(y, \hat{y})$, and backpropagate. This provides an in-graph gradient of $\mathcal{L}(y, \hat{y})$ with respect to the inputs, x_i, x_c, and x_s. Third, using the input gradients and the prediction loss, we compute the utility loss function (equation 10) and update the network weights. The procedure is depicted in Figure 1.

Experiments

In the following section, we discuss our empirical results using Neural Utility Functions [3]. First, we theoretically validate our proposed approach by recovering known, analytical substitution effects from data. Second, we perform recommendation tasks with both explicit and implicit feedback. Finally, we explore the learned item representations from a shallow Neural Utility Function.

Recovering Theoretical Substitution Effects

In this experiment our goal is to demonstrate that Neural Utility Functions can effectively recover theoretical item substitution effects better than networks trained to only minimize prediction loss. We can directly test this hypothesis by comparing the substitution effects learned from data to those from theoretical models of utility.

CES and Cobb-Douglas utility are two common and well-studied functions due their analytical tractability. Thus, we adopt these two functions to simulate ratings data and see how well different networks recover known, theoretical substitution effects. We first briefly introduce these two utility functions; more details can be found in the online supplement.

Cobb-Douglas Utility: The Cobb-Douglas utility function directly models trade-offs between m items, $x_1, x_2, ...x_m$, by means of the weights, $w_1, w_2, ...w_m$. The weight w_i denotes the marginal utility, or relative importance, of item i. Cobb-Douglas Utility and substitution effect are:

$$U(\mathbf{x}) = \prod_{i=1}^{m} x_i^{w_i}, \quad s_U(x_i, x_j) = \frac{w_i x_j}{w_j x_i}. \tag{11}$$

CES Utility: CES utility is also very common in consumer demand theory. The functional of form of CES utility and theoretical substitution effect are:

$$U(\mathbf{x}) = \Big[\sum_{i=1}^{m} w_i x_i^{\rho}\Big]^{\frac{1}{\rho}}, \quad s_U(x_i, x_j) = \frac{w_i}{w_i}\left(\frac{x_i}{x_j}\right)^{\rho-1} \tag{12}$$

See the online supplement for more details and derivations. For each of these theoretical functions, we fix the number of items, users, theoretical parameters, and simulate ratings data. The utility function weight vector is drawn from a uniform distribution and normalized, $\mathbf{w} \sim \text{unif}(0, 10)$, $\mathbf{w} = \mathbf{w}/\sum_{i=1}^{m} w_i$. These parameters are held out from the models during training. For each user, we randomly select 50% of the possible items for rating. Each item is input into $U(\mathbf{x})$ to get a scalar rating value. We perturb each rating with a small amount Gaussian noise to reflect more real-world settings. We fix the number of users at 100, and perform the experiment under three settings, where the number of items, $m = 64, m = 256$, and $m = 1024$.

We train two shallow networks ten times, each with one item embedding layer with a hidden dimension size of 256, and a single output weight layer. The first network (Baseline Neural Net) is trained to minimize the prediction error of ratings (i.e., mean-squared error loss). The second network (Neural Utility Function) is trained using the loss function in equation (10). We compute the learned pair-wise substitution values $\hat{s}(x_i, x_j)$ for each model (equation 2) and compare them to the theoretical quantity $s_U(x_i, x_j)$. Finally, we compute the mean-squared error between $\hat{s}(x_i, x_j)$ and $s_U(x_i, x_j)$ and perform a t-test on the difference of means to estimate statistical significance.

Results The results in Table 1 suggest that the Neural Utility Function is much more effective at capturing substitution effects and item trade-offs than the Baseline Neural Net. In both the Cobb-Douglas and CES cases, the proposed loss function reduces the error between learned and true substitution effect and is also statistically significant. Additionally, when the number of items, m, increases the substitution rates are generally more difficult to estimate and the errors increase. This likely occurs because the partial derivatives become more difficult to disentangle with a small dataset. However, the Neural Utility Function is more robust to additional items than the Baseline Neural Net. Overall, the results in Table 1 validate our hypothesis that NUFs can better learn economic relationships.

Recommendation Performance

We also evaluate recommendation performance on the Movielens 25M (Harper and Konstan 2015) and Amazon 18 (McAuley et al. 2015) datasets. Dataset summary statistics are reported in Table **??**. Both datasets provide timestamped user ratings of items. We filter the full Amazon dataset (233.1M samples) to three categories for our experiments: grocery and gourmet food, prime pantry, and home and kitchen.

Experimental Settings **Explicit Task:** We take 20% of each dataset for testing and the rest for training. In order to make inference for each user, we do a stratified random split. Meaning that for each user, 80% of the samples are allocated for training and 20% for testing. The target variable is the item rating, $y \in [1, 5]$. Here the baseline prediction loss is mean-squared error loss.

Implicit Task: In order to test how Neural Utility Functions perform on ratings-free data, we transform each rating to 0 or 1 denoting if the user has interacted with an item as is done in the literature (He et al. 2017). We split our data into train and test partitions with the following procedure: 1) For each user, we leave out the last item interaction and use

[3] For code see https://github.com/porterjenkins/neural-utility-functions

Predictive Model	Cobb-Douglas m=64	Cobb-Douglas m = 256	Cobb-Douglas m=1024	CES m=64	CES m=256	CES m=1024
Baseline NN	4.73	20.89	171.92	9.34	31.00	287.77
NUF	**2.91****	**7.08****	**132.40****	**4.76****	**6.94****	**124.81***

Table 1: Mean-squared error between the learned and theoretical substitution effects. The stars indicate the significance level (**: $\alpha = .05$, $*$: $\alpha = .10$). In both the Cobb-Douglas and CES case, the Neural Utility Function is more effective at recovering ground truth substitution effects and is statistically significant.

		EXPLICIT				IMPLICIT			
		Movielens		Amazon 18		Movielens		Amazon 18	
Model	Objective	RMSE	DCG@5	RMSE	DCG@5	HR@5	NDCG@5	HR@5	NDCG@5
MF	Prediction	2.694	9.145	3.050	19.348	0.424	0.318	0.516	0.354
	NUF	**1.817***	**9.953****	**3.147**	**19.403****	**0.698****	**0.562****	**0.554****	**0.395**
W&D	Prediction	0.953	10.816	1.109	19.514	0.891	0.729	0.700	0.526
	NUF	**0.945****	**10.930****	**1.078****	**19.608****	**0.916****	**0.734**	**0.786****	**0.610****
NCF	Prediction	1.059	9.236	1.123	19.409	0.888	0.728	0.710	0.529
	NUF	**0.978****	**10.655****	**1.094****	**19.573***	**0.914****	**0.743****	**0.772****	**0.603****
DFM	Prediction	1.013	10.429	1.166	19.415	0.402	0.289	0.676	0.443
	NUF	**1.010**	**10.732***	**1.130**	**19.471**	**0.854****	**0.804****	**0.855****	**0.711****

Table 2: Recommendation results for both the explicit and implicit tasks. We train four network architectures under both prediction loss and the NUF objective. Stars indicate the significance level (**: $\alpha = .05$, $*$: $\alpha = .10$)

Dataset	Interactions	Users	Items
Movielens 25M	25M	162,541	59,047
Amazon 18	8.18M	836,292	235,462

Table 3: Dataset Summary

the remaining samples as training data; 2) randomly choose k items that the user did not interact with; 3) combine the leave-one-out sample with the $negative$ samples to get the test set. We choose $k = 50$ in our experiments. In the implicit feedback case, the baseline prediction loss is binary cross-entropy loss.

Training: We train all models using the Adam optimizer (Kingma and Lei Ba 2014). We select $k = 5$ for the size of the complement and supplement sets. All models were implemented in Pytorch (Paszke et al. 2019) and trained on a Google Deep Learning VM with 60 GB of RAM and two Tesla K80 GPU's. We train each model multiple times to estimate the variance. See the supplementary materials for more training details.

Evaluation Measures We evaluate models trained in the explicit task using root mean-squared error (RMSE) and discounted cumulative gain (DCG@5). At test time we get a truncated, ranked list of item scores (top 5) and compute the RMSE and DCG. RMSE measures accuracy of ratings, while DCG is a measurement that judges the quality of their ranking. We do not normalize the DCG in the explicit feedback experiments because it requires us to compute the ideal DCG (IDCG), which is difficult to compute since each user does not rate each item.

In the implicit case we evaluate all models using the hit ratio (HR@5) and the normalized discounted cumulative gain (NDCG@5). We again get a truncated ranked list of all candidate items for each user and compute the HR@5 and NDCG@5. The hit ratio judges the frequency with which the ground truth test item was present in the top k items, while the NDCG accounts for its position in the list (He et al. 2017). We perform a permutation test (Good 2005) to estimate statistical significance (see supplement).

Baselines Our hypothesis is that training a network with the cost function described in (10) facilitates reasoning about item trade-offs and should therefore improve recommendation performance. We have designed our framework to be agnostic to the choice of the model "backbone". In our experiments, we train a variety of models under two settings: 1) we train each model with a standard prediction loss (e.g., MSE, binary cross-entropy); 2) we train each model with Neural Utility loss. We analyze the performance of the following models. **1) Matrix Factorization** A widely used recommendation method that factorizes the user-item matrix into two latent, low-rank matrices (Koren and Bell 2009). **2) Wide & Deep** A popular architecture that uses both wide and deep features and concatenates them at the output layer (Cheng et al. 2016). **3) Neural Collaborative Filtering (NCF)** A deep learning approach for modeling user-item interactions. We use the form that leverages a multi-layer perceptron to learn the interactions (He et al. 2017). **4) Deep Factorization Machines (DFM)** A powerful model that combines deep learning and factorization ma-

Model	Objective	Movielens HR@5	Movielens NDCG@5	Amazon 18 HR@5	Amazon 18 NDCG@5
MF	BCE	0.424	0.318	0.516	0.354
	BCE + Utility	**0.698****	**0.562****	**0.554****	**0.395**
	BPR	0.449	0.337	0.504	0.337
	BPR + Utility	**0.532****	**0.395****	**0.520****	**0.340***
W&D	BCE	0.891	0.729	0.700	0.526
	BCE + Utility	**0.916****	**0.734**	**0.786****	**0.610****
	BPR	0.901	0.736	0.728	0.563
	BPR + Utility	**0.915****	**0.746****	**0.771****	**0.599****
NCF	BCE	0.888	0.728	0.710	0.529
	BCE + Utility	**0.914****	**0.743****	**0.772****	**0.603****
	BPR	0.898	0.729	0.729	0.565
	BPR + Utility	**0.915****	**0.830****	**0.773****	**0.601****
DFM	BCE	0.402	0.289	0.676	0.443
	BCE + Utility	**0.854****	**0.804****	**0.855****	**0.711****
	BPR	0.986	0.955	0.975	0.926
	BPR + Utility	**0.998**	**0.973**	**0.991****	**0.955****

Table 4: A comparison of NUFs with different base loss functions for the implicit task. In all cases, the theoretical information in the Utility Prior increases performance.

chines to learn low- and high-order feature interactions (Guo et al. 2017).

Results The results from our recommendation experiments are reported in Table 2. We train each backbone architecture with and without Neural Utility loss. In all cases we see that the performance of a given model can be improved by training with the NUF loss function proposed in equation (10). In 81% of cases the performance boost in statistically significant.

Neural Utility Functions and Pairwise Loss

In Table 4 we demonstrate how NUFs can be combined with arbitrary loss functions. Bayesian Personalized Ranking (BPR) (Rendle et al. 2009) is a common approach to the recommendation problem and relies on a pairwise loss function that learns to rank by maximizing the similarity of items that co-occur within users. We perform the implicit task described in the previous section using Binary Cross Entropy (BCE) and BPR loss for $\mathcal{L}(y, \hat{y})$ in Equation (10). By comparing to BPR we can control for the added information available in the complement and supplement sets. Generally, BPR loss improves upon BCE. For both BPR and BCE as a base loss, the NUF extracts more information from the data via the *Utility Prior*, and boosts performance. This result confirms that the theoretical information in the NUF is causing an increase in performance.

Item Analogies

Finally, to explore what item relationships a NUF is capable of discovering, we perform an "item analogy" case study of the learned item embeddings. Word analogy exercises are common in the natural language processing literature (Mikolov et al. 2013) and are useful for diagnosing the semantics uncovered by a model. We train a shallow network on the Amazon 18 dataset using NUF loss. Our model has an item embedding layer and a single output layer. Each item in our dataset is represented as a 512 dimensional vector. We average over very similar items to deduplicate the item vectors (e.g., two peanut butter brands are aggregated to `Peanut Butter`). Results are reported in Table 5. We observe that the NUF is able to uncover complementary analogies such as "Coffee is to Creamer as Pizza is to Cheddar" "Biscuits are to Gravy as French Toast is to Pancakes." Additionally, the network encodes supplementary relationships such as "Sugar is to Sea Salt as Honey is to Jerky." These examples demonstrate that a Neural Utility Function is capable of learning meaningful item semantics.

Analogy	Result
Coffee - Creamer + Pizza ≈	Cheddar
Biscuits - Gravy + Coffee ≈	Almond Milk
Biscuits - Gravy + French Toast ≈	Pancakes
Sugar - Sea Salt + Honey ≈	Jerky
Sugar - Sea Salt + Egg ≈	Pastry

Table 5: Example item analogies using deduplicated item vectors and their corresponding euclidean distance. We also compare analogies between the NUF and base loss (see online supplement); the results indicate the NUF discovers better item semantics.

Conclusion

Motivated by theoretical economics, we proposed Neural Utility Functions, a framework for training neural network based recommender systems that can encourage the model to reason about item trade-offs. Specifically, the objective function we proposed constrains the ratio of item gradients to discover richer item relationships and economically favorable optima. We demonstrate that training a recommender system with NUFs can provide superior performance over multiple datasets and tasks.

References

Bentz, Y.; and Merunka, D. 2000. Neural Networks and the Multinomial Logit for Branch Choice Modelling: A Hybrid Approach. *Journal of Forecasting* .

Chen, X.; Li, S.; Li, H.; Shaohua, J.; Qi, Y.; and Song, L. 2019. Generative Adversarial User Model for Reinforcement Learning Based Recommendation System. In *International Conference of Machine Learning*.

Cheng, H.; Koc, L.; Harmsen, J.; Shaked, T.; Chandra, T.; Aradhye, H.; Anderson, G.; Corrado, G.; Chai, W.; Ispir, M.; Anil, R.; Haque, Z.; Hong, L.; Jain, V.; Liu, X.; and Shah, H. 2016. Wide & Deep Learning for Recommender Systems. *CoRR* abs/1606.07792. URL http://arxiv.org/abs/1606.07792.

Dotson, J. P.; Howell, J. R.; Brazell, J. D.; Otter, T.; Lenk, P. J.; MacEachern, S.; and Allenby, G. 2018. A Probit Model with Structured Covariance for Similarity Effects and a Source of Volume Calculation. *Journal of Marketing Research* .

Good, P. 2005. *Permutation, Parametric and Bootstrap Tests of Hypotheses (3rd edition)*.

Greydanus, S.; Dzamba, M.; and Yosinski, J. 2019. Hamiltonian Neural Networks. In *Advances in Neural Information Processing Systems 32*, 15353–15363. Curran Associates, Inc.

Guo, H.; Tang, R.; Ye, Y.; Li, Z.; and He, X. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. *CoRR* abs/1703.04247. URL http://arxiv.org/abs/1703.04247.

Hanson, S. J.; and Burr, D. J. 1990. What connectionist models learn: Learning and representation in connectionist networks. *Behavioral and Brain Sciences* .

Harper, M. F.; and Konstan, J. A. 2015. The MovieLens Datasets: History and Context. In *ACM Transactions on Interactive Intelligent Systems (TiiS)*.

He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; and Chua, T. 2017. Neural Collaborative Filtering. *CoRR* abs/1708.05031. URL http://arxiv.org/abs/1708.05031.

Hidasi, B.; Karatzoglou, A.; Baltrunas, L.; and Tikk, D. 2016. Session-based Recommendations with Recurrent Neural Networks. In *Proceedings of ICLR*.

Howell, J.; Ebbes, P.; Liechty, J. C.; and Jenkins, P. 2017. Gremlins in the Data: Identifying the Information Content of Research Subjects. In *HEC Paris Research Paper No. MKG-2017-1223*.

Jean, N.; Wang, S.; Samara, A.; Azzari, G.; Lobell, D.; and Ermon, S. 2019. Tile2Vec: Unsupervised Representation Learning for Spatially Distributed ‘Data. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)*.

Kingma, D. P.; and Lei Ba, J. 2014. Adam: A method for stochastic optimization. In *International Conference of Learning Representations*.

Koren, Y.; and Bell, Robert an Volinksy, C. 2009. Matrix Factorization Techniques for Recommender Systems. In *IEEE Computer*.

Li, H.; Xu, Z.; Taylor, G.; Studer, C.; and Goldstein, T. 2018. Visualizing the Loss Landscape of Neural Nets. In *Advances in Neural Information Processing Systems 31*, 6389–6399.

Linden, G.; Smith, B.; and York, J. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. In *Internet Computing, IEEE*.

Mas-Colell, A.; and Whinston, M. D. 1995. *Microeconomic Theory*. Oxford University Press.

McAuley, J.; Pandey, R.; and Leskovec, J. 2015. Inferring Networks of Substitutable and Complementary Products. In *Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining, KDD'15*.

McAuley, J.; Targett, C.; Shi, Q.; and Van Den Hengel, A. 2015. Image-based Recommendation on Styles and Substitutes. In *SIGIR*.

McFadden, D. 1974. Conditional logit analysis of quantitative response models. In Zarembka, P., ed., *Frontiers in Econometrics*, 105–142. Academic Press.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient Estimation of Word Representations in Vector Space.

Nicholson, W.; and Snyder, C. 2012. *Microeconomic Theory: Basic Principles and Extensions, 11th edition*. South-Western.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32*, 8024–8035. Curran Associates, Inc.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *In Proceedings of UAI*.

Train, K. 2003. *Discrete Choice Methods with Simulation*. Cambridge University Press.

Wang, H.; Zhang, F.; Wang, J.; Zhao, M.; Li, W.; Xie, X.; and Guo, M. 2018a. Ripplenet: Propagating user preferences on the knowledge graph for recommender systems. In *CIKM*, 417–426.

Wang, H.; Zhang, F.; Xie, X.; and Guo, M. 2018b. DKN: Deep knowledge-aware network for news recommendation. In *Proceedings of the 2018 world wide web conference*, 1835–1844.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 974–983.

Zhang, S.; Yao, L.; Sun, A.; and Tayi, Y. 2019. Deep Learning Based Recommender System: A Survey and New Perspectives. In *ACM Computing Surveys*.

Zhang, Y.; Zhao, Q.; Zhang, Y.; Friedman, D.; Zhang, M.; Liu, Y.; and Ma, S. 2016. Economic Recommendation with Surplus Maximization. In *In Proceedings of the 25th International Conference on World Wide Web (WWW'16)*.

Zhao, Q.; Zhang, Y.; and Freidman, D. 2017. Multi-Product Utility Maximization for Economic Recommendation. In *In Proceedings of the 10th International Conference on Web Search and Data Mining (WSDM'17)*.

Zhao, X.; Xia, L.; Zhang, L.; Ding, Z.; Yin, D.; and Tang, J. 2018. Deep reinforcement learning for page-wise recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*, 95–103.

Zheng, G.; Zhang, F.; Zheng, Z.; Xiang, Y.; Yuan, N. J.; Xie, X.; and Li, Z. 2018. DRN: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*, 167–176.