Shape Constraints for Set Functions

Andrew Cotter [1] **Maya R. Gupta** [1] **Heinrich Jiang** [1] **Erez Louidor** [1] **James Muller** [1] **Taman Narayan** [1] **Serena Wang** [1] **Tao Zhu** [1]

Abstract

Set functions predict a label from a permutation-invariant variable-size collection of feature vectors. We propose making set functions more understandable and regularized by capturing domain knowledge through shape constraints. We show how prior work in monotonic constraints can be adapted to set functions, and then propose two new shape constraints designed to generalize the conditioning role of weights in a weighted mean. We show how one can train standard functions and set functions that satisfy these shape constraints with a deep lattice network. We propose a non-linear estimation strategy we call the semantic feature engine that uses set functions with the proposed shape constraints to estimate labels for compound sparse categorical features. Experiments on real-world data show the achieved accuracy is similar to deep sets or deep neural networks, but provides guarantees on the model behavior, which makes it easier to explain and debug.

1. Introduction

Common set functions such as *mean*, *median*, *min* and *max* share the satisfying property that they are monotonically increasing functions: if one of the elements in the set is increased, the output can only increase. In this paper, we show that one can machine learn flexible set functions with this monotonicity guarantee.

Another common and useful set function is the *weighted mean*, in which a conditioning feature (the weight input) specifies how much to trust a primary input. We propose and investigate new shape constraints that act on pairs of features that capture this relationship of one feature conditioning the importance of a second feature.

For example, suppose one wants to predict how good a restaurant is based on its customer reviews, where each restaurant x is described by a set of $M(x)$ reviews, and each review is described by D features, such as the star rating and the reviewer's credibility. We will show how to train a flexible function that maps the set of reviews to a prediction for the restaurant, and is guaranteed to be a monotonically increasing function of each review's star rating, and is guaranteed to be more sensitive to a rating if its reviewer's credibility is higher.

In general, our proposed new shape constrained set function learning is applicable whenever one is estimating a measurement from a set of noisy measurements, and has side information about how noisy each measurement is. The monotonicity shape constraints can be applied to multiple *primary* features, and the new conditioning shape constraints can be applied to multiple pairs of features. These shape constraints provide guarantees on how the model behaves, which improves the ability to summarize the model (improved interpretability) and makes the model behave more predictably (improved debuggability).

While we focus here on set functions, our proposed conditioning shape constraints can also be used to capture complementary relationships, and submodularity and supermodularity relationships, for standard machine learned functions that map $\mathbb{R}^D \to \mathbb{R}$.

We will show that shape-constrained set functions are especially helpful for problems that arise in feature engineering from *sparse compound categorical inputs* such as queries. A common heuristic for such compound inputs is to split the compound input up into a set of elements (e.g., split a query into its ngrams), estimate a label for each element, then combine the set of estimates, usually using a classic set function like mean or max. We propose such a *split-estimate-combine* method that we call *semantic feature engine* (SFE), where the *combine* step uses a *learned* set function with the proposed shape constraints. SFE is easy to debug and understand, can be re-trained with less churn than deep models, and we will show it works well in practice.

[1]Equal Contribution. Google AI, Mountain View, CA. Correspondence to: Tao Zhu <tzhu@google.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

2. Related Work on Set Functions

We restrict our focus to set functions that output a label $y \in \mathbb{R}$ and take an input x which is specified as a set of unordered feature vectors $\{x_m \in \mathbb{R}^D\}$ for $m = 1, \ldots, M(x)$ for some finite $M(x)$. The special case of $D = 1$ includes classic set functions like mean, min, max, geometric mean, and harmonic mean, and has been studied under the name *aggregation functions* (Torra & Naurkawa, 2010). Another special case is if every element of the set is a one-hot Boolean feature vector, often used to represent a set of categorical elements (see e.g. Bach (2013)).

Zaheer et al. (2017) showed that set functions of this form can be expressed as some transform ρ of the average of a per-element transform ϕ:

$$f(x) = \rho\left(\frac{1}{M(x)}\sum_{m=1}^{M(x)}\phi(x_m)\right) \tag{1}$$

where $\phi : \mathbb{R}^D \to \mathbb{R}^K$, and $\rho : \mathbb{R}^K \to \mathbb{R}$. Equation (1) can be read as a function ρ acting on a K-dimensional mean embedding defined by ϕ. See Fig. 1 for a block diagram.

Zaheer et al. (2017) proposed using a DNN for ϕ and ρ, and jointly optimizing the training of (1); they called this *deep sets*. Earlier work includes *support distribution machines* (Muandet et al., 2012; Poczos et al., 2012), which we express in the form of (1) in Appendix A.

Distribution regression handles the same set-up, with the additional assumption that each element x_m in the set x was drawn independently and identically (IID) from a probability distribution P_X (Póczos et al., 2013; Szabo et al., 2016). This assumption enables handling the learning in two phases: first learn the underlying probability distribution $\hat{P}_X$ for each x (e.g. with a kernel density estimation), and then learn a mapping from a probability distribution to the label. That work is best-suited to sets large enough to make distribution estimation reasonable, though Bayesian approaches can help (Law et al., 2018). Other work has also defined kernels for distributions derived from sets of inputs (Kondor & Jebara, 2003).

Some machine-learning algorithms have also been proposed for *fixed-size* permutation-invariant sets, e.g. (Shivaswamy & Jebara, 2006). The multiple instance learning set-up (Dietterich et al., 1997) differs in that *each element in the input set has a label*, that is each example has the form $\{x_m, y_m\}$ (and the label for the set x is usually taken to be the maximum of the $\{y_m\}$).

3. Monotonicity Guarantees

The unlimited flexibility of *deep sets* makes it difficult to know what the model has learned, predict how it will behave for a given example, and be confident it hasn't overfit. For many applications one does have domain knowledge that certain features should only have a positive impact on the output. For standard machine learning, such domain knowledge can be captured by constraining the model to be monotonically increasing with respect to selected inputs. Such monotonicity constraints provide regularization and make the model's behavior easy to describe with respect to the constrained features (see e.g. Groeneboom & Jongbloed (2014); Chetverikov et al. (2018); Gupta et al. (2016)).

Here, we expand the definition of monotonicity to handle set functions as defined in (1). Without loss of generality, we only address increasing monotonicity, analogous definitions can be made for decreasing monotonicity.

Definition 1 (Monotonically Increasing Set Function)**.** *A set function $f(x)$ as defined by (1) is* monotonically increasing *with respect to feature d if $f(x^+) \geq f(x^-)$ for any pair of inputs x^+ and x^- that are the same except that $x_m^+[d] > x_m^-[d]$ for some m, and* strictly monotonically increasing *if $f(x^+) > f(x^-)$.*

An analogous definition was given for the special case of $D = 1$ in Torra & Naurkawa (2010). A different notion is a *monotone* set function, which requires $f(A) \leq f(B)$ if $A \subseteq B$ (Rosenthal, 1948).

3.1. Learning Monotonic Set Functions

One approach to producing monotonic set functions would be to use deep sets, but constrain all the DNN parameters in both the ϕ and ρ models to be non-negative. However, such monotonic neural nets are known to be very restricted in their expressability (Daniels & Velikova, 2010), in part because all the weights must be constrained to be non-negative even if one only wants to constrain one feature to be monotonic. And if the activation function is a ReLU, monotonic neural nets also become convex (Gupta et al., 2018).

Instead, we propose using a *deep lattice network* (DLN) (You et al., 2017) for each of ϕ and ρ, as shown in Fig. 1 and described in Section 5. DLNs are state-of-the-art for learning monotonic functions (You et al., 2017). A DLN can be made-up of three types of layers: (i) *linear embedding* layers that linearly mix inputs (ii) *calibration layers* of one-dimensional piece-wise linear functions that maps D inputs to D outputs without mixing the D inputs, and (iii) *lattice layers* that nonlinearly mix inputs using an ensemble of multi-dimensional interpolated look-up tables (lattices). Fig 2 shows an example of just a simple lattice layer with $D = 2$ inputs and one ouput. The lattice layer is parameterized by a 2×2 look-up table, such that the four look-up table parameters are the values of the function at the four corners of the $D = 2$ input space, and the look-up table is bilinearly interpolated to produce the output.

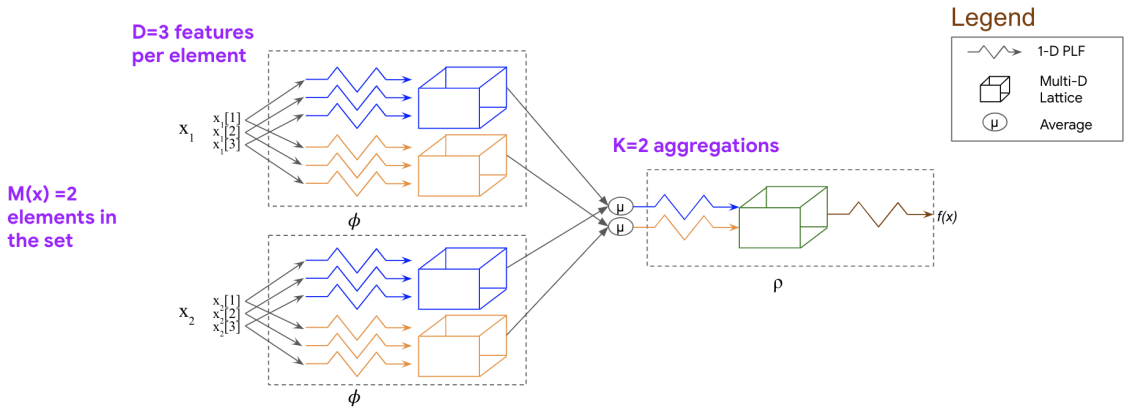

Figure 1. Figure shows an example of the set function expressed in (1) using calibrator and lattice layers for ϕ and ρ, as proposed. The $D = 3$ features for each element x_m are fused together using the same ϕ function. The $K = 2$ outputs of ϕ can be interpreted as $K = 2$ separate fusions, one shown in blue and orange. Each of the $K = 2$ fusions is averaged over the $M(x) = 2$ elements, then the K averaged values are inputs to ρ. Each squiggly line denotes a one-dimensional piece-wise linear function (PLF) that calibrates its input, and each box denotes a multi-dimensional lattice function that nonlinearly mixes its inputs (see Fig. 2 for an example lattice function). The output $f(x)$ can be constrained to be a monotonic function of any inputs by constraining each of the functions along the path between those inputs and $f(x)$ to be monotonically increasing. Further, any of the D inputs can be constrained to condition the importance of another of the D inputs as defined by the proposed *trapezoid* shape constraint by constraining each ϕ_k to have a *trapezoid* shape constraint, and constraining the ρ to be monotonic and continuous w.r.t each ϕ_k.

For more details on lattice layers or calibration layers, see Gupta et al. (2016). All three types of DLN layers can be constrained for monotonicity, resulting in end-to-end monotonicity guarantees by composition (You et al., 2017).

For set functions, one can interpret the K outputs of ϕ in (1) as K different fusions $\{\phi_k(x_m)\}$ of the D features in x_m. Then if we constrain the K fusions to each be monotonic w.r.t the selected monotonic features, and also constrain ρ to be monotonic with respect to each $\{\phi_k\}$ then the overall set function will be monotonic w.r.t the selected features.

Proposition 1. *For functions of the form (1), if ϕ_k is monotonically increasing w.r.t d and ρ is monotonically increasing w.r.t ϕ_k for any ϕ_k that acts on feature d, then f is monotonically increasing in d.*

(All proofs are in the appendix.)

4. New Two-Input Shape Constraints

Another type of domain knowledge that is common when dealing with set functions is that some inputs are known to play a conditioning role for other inputs, like the weights in a weighted mean. For example, we may have a set of ratings and a confidence in each rater as depicted in Fig. 2, or more generally, a set of measurements and knowledge about how precise each measurement is. We propose and investigate two different new shape constraints that can capture this type of domain knowledge, discuss how to guarantee them for multi-layer functions, show that we can guarantee these new shape constraints for lattice models for standard functions that map $\mathbb{R}^D \to \mathbb{R}$, and show how to guarantee the new *trapezoid* shape constraints for set functions of form (1).

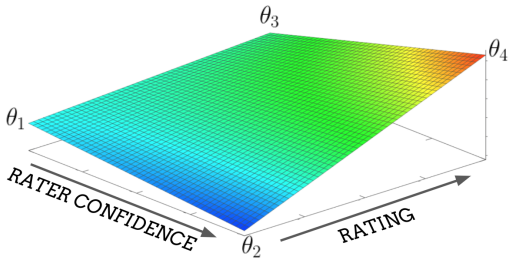

Figure 2. An example lattice function $\phi : \mathbb{R}^2 \to \mathbb{R}$ that acts on a rating and the rater's confidence. The lattice parameters $\theta_1, \theta_2, \theta_3, \theta_4$ are the function values at the corners, the rest of the function is bilinearly interpolated from these parameters. Here, ϕ is monotonically increasing in the rating, and ϕ satisfies the proposed *Edgeworth constraint*: the slope of ϕ w.r.t rating gets steeper as the rater confidence increases. It also satisfies the proposed *trapezoid constraint*: as rater confidence grows, the set of possible outputs grows.

4.1. Edgeworth Shape Constraint

We call our first new shape constraint the *Edgeworth* shape constraint, named for the 19th century economist Francis Ysidro Edgeworth, who defined complementary goods as inputs that increase the marginal value of each other's impact on an output (Milgrom & Roberts, 1995). Suppose f is monotonically increasing w.r.t feature d, then an Edgeworth shape constraint guarantees the model is more sensitive to

feature d if the conditioning feature w is higher, see Fig. 2 for an example.

Definition 2 (Edgeworth Shape Constraint)**.** *A set function $f(x)$ as per (1) satisfies the* Edgeworth shape constraint *on input d and input w if the change in f w.r.t. d is monotonically increasing in w, that is,*

$$f(x^{++}) - f(x^{-+}) \geq f(x^{+-}) - f(x^{--}) \tag{2}$$

for any choice of x^{++}, x^{+-}, x^{-+}, x^{--} that are identical except there is an m and real numbers $a^{-*} \leq a^{+*}$ and $a^{*-} \leq a^{*+}$ where:

$$\begin{aligned} x_m^{--}[d] &= x_m^{-+}[d] = a^{-*} \\ x_m^{+-}[d] &= x_m^{++}[d] = a^{+*} \\ x_m^{--}[w] &= x_m^{+-}[w] = a^{*-} \\ x_m^{-+}[w] &= x_m^{++}[w] = a^{*+}. \end{aligned}$$

We extend this definition to standard (non-set) functions $f : \mathbb{R}^D \to \mathbb{R}$, by regarding them as set functions with fixed set size $M(x) = 1$.

One can apply the Edgeworth constraint to non-monotonic inputs (see Appendix D for an example). For the special case that each element of a set is a one-hot Boolean feature vector, then the Edgeworth constraint (2) is equivalent to constraining f to be supermodular (Bach, 2013).

To guarantee that a multi-layer set function satisfies an Edgeworth constraint, it is sufficient to guarantee it for each aggregation ϕ_k and constrain ρ to be increasing and convex in each ϕ_k, and constrain f to be monotonically increasing in w (details in Appendix C), which is not a reasonable constraint if w is a conditioning feature like measurement precision. This difficulty of satisfying Edgeworth constraints in multi-layer functions motivates our next shape constraint.

4.2. Trapezoid Shape Constraint

A different property we expect a conditioning feature w to have is that the range of possible outputs should grow as w grows. For example, as illustrated in Fig. 2, if we trust a rater more, then we should be more negative if they give a 1 star review, and more positive if they give a 5 star review (on a scale of 1-5 stars). The proposed Edgeworth constraint does not in itself guarantee this property because it is defined on *differences*. We capture this property in a second new shape constraint we refer to as a *trapezoid* constraint, as it forces the output ranges to have an *acute trapezoidal* shape as w is increased.

Definition 3 (Trapezoid Shape Constraint)**.** *Let f be a set function as per (1), and assume the domain of input d is a bounded interval $[a^{\min *}, a^{\max *}]$ for some real-values $a^{\min *} < a^{\max *}$. Then f satisfies the* trapezoid shape constraint *for input d conditioned on input w if f is continuous, monotonically increasing w.r.t. d, and the range of f never shrinks as w grows, that is:*

$$[f(x^{\min -}), f(x^{\max -})] \subseteq [f(x^{\min +}), f(x^{\max +})] \tag{3}$$

for any choice of $x^{\min -}$, $x^{\max -}$, $x^{\min +}$, $x^{\max +}$ that are identical except there is an m and real numbers $a^{*-} \leq a^{*+}$ where:

$$\begin{aligned} x_m^{\min -}[d] &= x_m^{\min +}[d] = a^{\min *} \\ x_m^{\max -}[d] &= x_m^{\max +}[d] = a^{\max *} \\ x_m^{\min -}[w] &= x_m^{\max -}[w] = a^{*-} \\ x_m^{\min +}[w] &= x_m^{\max +}[w] = a^{*+}. \end{aligned}$$

Equivalently, (3) can be written,

$$f(x^{\min -}) \geq f(x^{\min +}) \tag{4}$$

$$f(x^{\max +}) \geq f(x^{\max -}). \tag{5}$$

We extend this definition to standard (non-set) functions $f : \mathbb{R}^D \to \mathbb{R}$, by regarding them as set functions with fixed set size $M(x) = 1$.

Lemma 1. *For (1) to satisfy the trapezoid constraint, it is sufficient that ϕ_k satisfies the constraint for all k, and that ρ is continuous and monotonically increasing w.r.t each of its K inputs.*

4.3. Two-Input Shape Constraints with Lattice Models

Next we show how to constrain a calibrated lattice model (and an ensemble of them) to make sure that a trained model satisfies either the Edgeworth or trapezoid constraint. The example shown in Figure 1 uses a calibrated lattice model for each ϕ_k. Recall that a calibrated lattice model (Gupta et al., 2016) is a two-layer deep lattice network where each of the D inputs is individually calibrated, then fused together with a multi-dimensional lattice function. That is,

$$g(x_m) = \theta^T \psi(c_\alpha(x_m)), \tag{6}$$

with definitions as follows. The first layer calibrates each of the D inputs with a different 1-D calibrator trained per input: $c_\alpha(x_m) = (c_{\alpha_1}[1](x_m[1]), \ldots, c_{\alpha_D}[D](x_m[D]))$ and each calibrator $c[d] : \mathbb{R} \to [0, 1]$ is a 1-D piecewise linear function parameterized by V fixed knots defined by the V quantiles of the training data and V corresponding free parameters $\alpha_d \in \mathbb{R}^V$. Thus the first layer c_α outputs a calibrated feature vector $[0, 1]^D$, and unless otherwise noted, we constrain each calibrator to produce outputs over its full output range $[0, 1]$ (i.e. to be a surjection). Thus the input to the second layer is a point in the D-dimensional hypercube, which has has 2^D vertices. Then $\psi : [0, 1]^D \to \mathbb{R}^{2^D}$ is a fixed nonlinear interpolation kernel that maps the D-dimensional point to 2^D interpolation weights on the 2^D

hypercube vertices in a way that satisfies the linear interpolation equations (Gupta et al., 2006), such as the standard multilinear interpolation kernel or the faster (but non-differentiable) simplex kernel which produces the Lovasz extension (Gupta et al., 2016). Then the linear interpolation weight vector $\psi(\cdot)$ weights the 2^D look-up table parameters $\theta \in \mathbb{R}^{2^D}$ to form the output $\theta^T\psi(\cdot)$.

The following lemmas show sufficient and necessary conditions for the calibrated lattice function to satisfy the Edgeworth and trapezoid constraints. Consult Fig. 2 to help visualize these constraints.

We index the lattice layer's parameters $\theta \in \mathbb{R}^{2^D}$ by $\mathbf{p} \in \{0,1\}^D$, where $\theta_\mathbf{p}$ is the value of the function at the corner $\mathbf{p}$ of the 2^D unit hypercube; that is $\theta_\mathbf{p} = \theta^T\psi(\mathbf{p})$. We denote by $\mathbf{e}_i \in \{0,1\}^D$ the vector containing 1 in the ith entry and 0 everywhere else.

Lemma 2. *Assume that $c[d]$ and $c[w]$ are monotonically increasing. Then g as defined in (6) satisfies the Edgeworth constraint if and only if for all $\mathbf{p} \in \{0,1\}^D$ that has 0's in its dth and wth entries,*

$$\theta_{\mathbf{p}+\mathbf{e}_w+\mathbf{e}_d} - \theta_{\mathbf{p}+\mathbf{e}_w} \geq \theta_{\mathbf{p}+\mathbf{e}_d} - \theta_\mathbf{p}. \tag{7}$$

Lemma 3. *Assume that $c[w]$ and $c[d]$ are monotonically increasing and that g is monotonically increasing w.r.t d. Then g as defined in (6) satisfies the trapezoid constraint if and only if for all $\mathbf{p} \in \{0,1\}^D$ that has 0's in its dth and wth entries,*

$$\theta_{\mathbf{p}+\mathbf{e}_d+\mathbf{e}_w} \geq \theta_{\mathbf{p}+\mathbf{e}_d} \text{ and } \theta_\mathbf{p} \geq \theta_{\mathbf{p}+\mathbf{e}_w}. \tag{8}$$

Corollary 1. *If a calibrated lattice defined by (6) satisfies the trapezoid constraint, then it also satisfies the Edgeworth constraint.*

Corollary 2. *Given an ensemble of calibrated lattices of the form $g(x) = \sum_t \theta_t^T\psi(c_{\alpha_t}(x_m))$ (Canini et al., 2016) where all lattices satisfy equation (7) (or equation (8)), $g(x)$ satisfies equation (7) (or equation (8)).*

5. Training Set Functions with Shape Constraints

Given N training example pairs $\{x_i, y_i\}$, where x_i is a set of $M(x_i)$ feature vectors $\{x_{im} \in \mathbb{R}^D\}$ for $m = 1, \ldots, M(x_i)$ and $y_i \in \mathbb{R}$, we propose training a 5-layer deep lattice network *(DLN)* set function of the form (1) where the kth component of ϕ is denoted ϕ_k and is a calibrated lattice function as per (6) and ρ is a calibrated lattice with its output also calibrated (as shown in Fig. 1), using a constrained empirical risk minimization:

$$\arg\min_{\alpha,\beta,\gamma,\theta,\zeta} \quad \sum_{i=1}^{N} L\left(\rho\left(\sum_{m=1}^{M(x_{im})} \phi(x_{im})\right), y_i\right) \tag{9}$$

$$\text{s.t.} \quad A^T[\alpha\ \beta\ \gamma\ \theta\ \zeta] \geq 0, \tag{10}$$

$$\text{where} \quad \phi_k = \theta_k^T\psi(c_{\alpha_k}(x_{im})), \text{ for } k = 1, \ldots, K,$$
$$\rho = c_\gamma\left(\zeta^T\psi(c_\beta(\cdot))\right),$$

and where the matrix A specifies the sparse linear inequality constraints needed for any specified monotonicity or trapezoid constraints. One can also enforce Edgeworth constraints, but that requires simplifying the model to just a calibrated lattice by setting $K = 1$ and removing ρ, or adding additional constraints on ρ (see Sec 4.1 for details).

All parameters of (9) have gradients so that (9) can be trained using SGD. To handle the sparse linear inequality constraints in (10) we use the Light-Touch algorithm (Cotter et al., 2016) on top of Adagrad (Duchi et al., 2011). It is straightforward to extend (9) to use an ensemble of lattices for the ρ or ϕ_k.

6. Semantic Feature Engine

We propose a general strategy we term *Semantic Feature Engine* (SFE) that uses set functions with shape constraints to estimate a label Y for a compound sparse categorical input z, such as a query, or the set of actors in a movie, or ingredients in a recipe. Such inputs are *sparse* in that any given z may have never occurred before, and are *compound* in the sense that they can be split up into meaningful tokens. For example, a query can be split into its ngrams, a recipe split into common subsets of ingredients, etc.

The *bag of words* strategy is to treat the compound input as a Boolean feature vector signifying which tokens are present, and learn an embedding to reduce the dimensionality, often followed by a DNN. This strategy can be difficult to debug because the embedding vectors are not inherently meaningful, and can have problematically high *churn* because the embedding may radically change when re-trained (Cormier et al., 2016). Lastly, we get no guarantees on what the model learns.

Alternatively, the classic *split-estimate-combine* heuristic splits the compound input into tokens, computes an estimate of the label Y for each token, and then combines the per-token estimates with a classic set function like the mean or max into one estimate for z. This approach is easy to debug, and the per-token estimates are relatively stable when refreshed with new training data, and is a positive function of each of the per-token estimates. The proposed SFE builds on this strategy, but uses a more flexible learned set function with shape constraints to combine D features

Figure 3. Example SFE steps to predict how much you will enjoy "green tea cake" (show by acronym "G T C" in the figure). First, tokenize "green tea cake" into all its ngrams. Then retrieve pre-computed estimates of your enjoyment for each ngram. In this example, we do not have a pre-computed estimate for the ngrams "green tea cake" or "tea cake", but we do for "green tea", "green", "tea", and "cake." Next, because there is an estimate for "green tea," we filter out its child-token estimates "green" and "tea" because they are likely to be less useful and possibly misleading. Then other useful features about the remaining two tokens "green tea" and "cake" are retrieved, say the order of each ngram (a bigram and a unigram), and how confident we are in each token's estimate. Now we have a set with $M(x) = 2$ elements, and each element has $D = 3$ features (the estimate, the ngram order, and the confidence). Fuse this set into a final estimate with a learned set function, where the set function has been trained to predict the overall estimate, and has been constrained to be a positive function of each of the estimates, and with a trapezoid constraint that makes the output trust a token's estimate more if it is a higher-order ngram or its confidence scores is higher.

per-token. SFE preserves delivers an easier to debug model with guaranteed behavior and much less churn than embeddings; see Appendix M for more details on debuggability and Appendix N for more details on churn.

6.1. SFE Evaluation

First, we explain how SFE evaluation works, given pre-computed per-token estimates and a trained set function. See Fig. 3 for an example where the input z is short text, and see Appendix H for an example where the input z is a set of attributes. Given a compound sparse categorical example z, first tokenize z into a set of tokens $\{z_m\}$. For example, a natural tokenization for short text inputs is into its set of ngrams up to some order, e.g. up to quadgrams. For each token, SFE retrieves an estimate of the label Y for that token from a table built as part of training; if a token was not seen enough times during training to provide a useful estimate, then no estimate exists for that token. Additional features are retrieved for the remaining tokens, such as the token order (i.e. unigram vs bigram), other measures of confidence in each token estimate, and how many tokens there were originally, how many tokens had estimates, etc., producing a D dimensional feature vector for each retrieved token. A set function is then run on the set of D feature vectors, and the output is taken to be the estimate $\hat{E}[Y|z]$. One can apply the set function on all the retrieved tokens. However, to improve accuracy and debuggability, we recommend filtering less-precise tokens if more-precise tokens have estimates. For example, for ngrams treat all sub-ngrams of a ngram as its children. We filter out a child token if any of its parent tokens have estimates. So if we have found an estimate for the bigram *ice cream*, we throw away the estimates for the unigrams *ice* and *cream* (see Fig. 3 for another example). More generally, define a partial ordering of the tokens that can be described as a directed acyclic graph on the tokens, and remove a child token if its parent token is retrieved. See Appendix H for another example.

6.2. Computing the SFE Per-Token Estimates

Next we describe how to compute the estimates of Y for each token. Start with a training set $\{z_j, y_j\}$ for $j = 1, \ldots, J$, where each z_j is a categorical input, and tokenize each z_j into a set of tokens $\{z_{jm}\}$ using the same tokenization that will be used at runtime.

Per-token estimates can be computed as the maximum likelihood estimate (MLE), that is, just take the average Y over all examples in which a token appears (as done in our experiments). In addition, we drop any estimates computed from *too few examples*, where one can specify *too few* by a required minimum count (done in the experiments), or by a required maximum confidence interval for a token estimate. Other estimation strategies than MLE can of course be used, such as MAP estimates, or one can take advantage of the simultaneous multiple estimates to do Stein estimation or multi-task averaging (Feldman et al., 2014).

However, if at runtime one filters out child tokens if their parent token is retrieved (as we recommend, and done in all our examples and experiments), then one should also do this filtering when computing the per-token estimates to make training and run-time consistent. Specifically, partition all seen training tokens $\{z_{jm}\}$ into Q disjoint subsets $\{\mathcal{T}_q\}$ for $q = 1, \ldots, Q$ following the topological ordering of the partial ordering DAG used to filter at evaluation time, so that the set $\mathcal{T}_Q$ has the highest-order tokens. That is, if $t_1 \in \mathcal{T}_r$, $t_2 \in \mathcal{T}_s$, and $r < s$, then t_1 cannot be an ancestor of t_2 in the DAG. For example, if the tokens are ngrams, partition by ngram order so that $\mathcal{T}_q$ is the set of all q-order ngrams, that is, the tokens *ice*,*cream* $\in \mathcal{T}_1$ have ancestor *ice cream truck* $\in \mathcal{T}_3$. Then we compute the maximum likelihood estimate $\hat{E}[Y|t]$ for the token t as the empirical mean of

Y over the training examples that contain t but were not already used to form estimates for ancestors of t. That is, for $q = Q, Q-1, \ldots, 1$ in turn, for each token $t \in \mathcal{T}_q$,

$$\hat{E}[Y|t] = \frac{\sum_{j=1}^{J} y_j I_{t \text{ not covered } z_j}}{\sum_{j=1}^{J} I_{t \text{ not covered } z_j}}$$

where $I_{t \text{ not covered} z_j} = I_{t \in \{z_{jm}\}} I_{\nexists \hat{E}[Y|s] \text{ for any } s \in \{z_{jm}\}, t \in s}$

and $t \in s$ means s is an ancestor of t. (11)

Before proceeding on to $\mathcal{T}_{q-1}$, remove away any estimates computed from *too few examples*, whether that is controlled by a minimum count or a maximum confidence interval. See Appendix I for a worked example.

6.3. Training the SFE Set Function

Training the SFE set function requires a training set $\{z_i, y_i\}$, where each z_i is converted into a set x_i as described in Sec. 6.1, and the set function is then trained on $\{x_i, y_i\}$. We recommend constraining the set function to be monotonically increasing in each token's estimate. We also investigate imposing a trapezoid constraint on the token estimate d conditioned on the token order w (e.g. unigram vs bigram), and possibly on other features that specify the precision of the token estimate. To reduce overfitting, use a different set of training data for the set function than is used to form the token-wise estimates.

7. Experiments

We compare the DLN set functions with shape constraints to Deep Sets with four experiments, summarized in Table 1.

The deep sets comparison used a TensorFlow implementation with a 3-layer fully-connected DNN for each of ϕ and ρ, as in Zaheer et al. (2017). The DLN set functions used the simplex interpolation kernel for ψ, and all DLN models were trained with monotonicity constraints on one or two features, some were also trained with trapezoidal constraints on conditioning features. The DNN comparisons used a standard feed-forward architecture. Input text strings are tokenized into ngrams, similarly to SFE, while input sets are tokenized into their individual elements. These elements are represented by a Boolean feature vector, passed through an embedding, then followed by a series of fully connected layers and a final softmax classification layer.

All hyperparameters were validated: see Appendix J for details. The training loss for all learned functions was squared error.

7.1. Sales Prediction From Reviews

For the Kaggle puzzles dataset, the goal is to predict the number of sales of each of 199 puzzles over a six month window given its set of customer reviews at the beginning of the six month period. The data is not IID: the test set is the most recent 6 month data, the validation data is from the previous 6 months and on only 168 puzzles, and the train set is from 6 months before that and only on 155 puzzles. We take $D = 3$ features for each review: its star rating, its word count, and $M(x)$. We constrain all the DLN models to be monotonic in star rating and $M(x)$ (as the number of reviews signals popularity). For DLN Trap., we also apply a trapezoidal constraint on star rating and word count, as we expect longer reviews to be more important. We also compare to simply averaging the feature vectors for each review, then training a $D = 3$ linear regression on the mean feature vectors. Results in Table 2 show that using the trapezoid constraint reduced test MSE, and it also makes it easier to explain what the model does and predict how it will behave on unseen data.

7.2. Predicting Kickstarter Success From Titles

The Kaggle Kickstarter dataset has $N = 331,034$ examples of Kickstarter campaign titles that are labeled as succeeded or failed. We removed capitalization. We split those examples randomly 70/10/20 to form a train/validation/test set, then split the train data 90/10 to train the SFE and the set function on separate train data; the DNN was trained on all 100% of the training data.

We trained SFE to estimate the probability of success given a title, which we tokenized into ngrams, and kept per-token estimates if a token appeared at least 10 times in the SFE train set. One can analyze the SFE token estimates to see which title ngrams were most and least effective. For example, the 871 titles with the ngram *a short film* had a success rate of 73%, but the 773 titles with the ngram *app* were only 10% successful (more per-ngram statistics in App. L).

For the SFE set function, we compared a simple average of the per-token estimates to a learned set function using deep sets or DLN's trained on the held-out 10% training examples and using $D = 5$ features per-token. Results in Table 3 show the SFE strategy works well compared to the DNN, and that the DLN test accuracy is consistently slightly better than the Deep Sets. The trapezoidal constraints improve our understanding of the model behavior and may have a slight positive effect on metrics.

7.3. Cuisine Classification from Recipe Ingredient List

The Kaggle recipes dataset consists of $N = 39,774$ recipes from 20 cuisines. We build a model that estimates $P\{\text{cuisine c is correct} \mid \text{recipe ingredients list, cuisine c}\}$ for 20 different cuisines c, one of which is correct for each recipe. We rank the 20 cuisine scores and report precision@3. The dataset was randomly split 70/10/20 into a train/validation/test set, and then the train set was

Table 1. Experiment Dataset Summary: $M(x)$ stats are for test set.

Dataset	# Train Set Function	# Train SFE	# Val	# Test	D	$M(x)$ Mean	$M(x)$ Max	# SFE Tokens
Reviews	155	-	168	199	3	9.3	55	-
Recipes	2,746	25,057	4,025	7,946	5	65.4	1121	3,394,220
Kickstarter Title	23,172	208,552	33,104	66,208	5	3.6	17	18,352
User Intent	551,696	$>$ 100 billion	78,814	157,627	8	1.5	29	$>$ 1 billion

Table 2. Sales Predicted From Reviews: Mean Squared Error

	Train	Val.	Test
Linear	3308	3771	8221
Deep Sets	562	2377	8323
DLN	3146	3320	8330
DLN Trap.	3084	3305	7587

randomly split 90/10 into SFE train set and set function train set. The DNN was given 100% of the train data. We did some standard pre-processing such as removing capitalization, which we have made available as a Kaggle kernel. We ran SFE on the ingredients crossed with each cuisine with a count threshold of 5 and a maximum subset size of 3, see Appendix K for a complete example. As Table 4 shows, the DNN performed best, with the DLN close behind, and then deep sets, and the classic mean token estimate. The trapezoid constraints did hurt accuracy a little - we hypothesize that and the DNN's good performance was due to the train/test split having some too-similar recipes that rewarded more memorization over generalization.

7.4. Predicting User Intent

For this Google binary classification problem, the goal is to predict if a given query is seeking a webpage or a place in the real world. For example, the query [coffee] seeks places, but [is coffee carcinogenic] seeks webpages. SFE was run on over 100 billion training examples, for ngrams up to quadgrams, where the label was a noisy binary label derived from user interactions. Only ngrams that met a very high count threshold were included, producing over 1 billion unique ngrams for which per-ngram statistics were kept.

Table 3. Kickstarter Accuracy (Prior is 59.6%).

	Train	Val.	Test
DNN	68.9	65.0	64.9
Mean SFE Token Estimate	64.3	64.1	64.3
Deep Sets	65.8	65.5	65.7
DLN	65.9	65.7	65.8
DLN Trap.: Ngram Order	66.2	65.8	65.8
DLN Trap.: Ngram Freq.	66.1	65.7	65.9
DLN Trap.: Both	66.3	65.8	65.8

Table 4. Recipes Results

	Train Prec @3	Val. Prec @3	Test Prec @3
DNN	97.3	89.1	88.9
Mean SFE Token Estimate	87.3	87.5	87.5
Deep Sets	88.5	88.5	87.7
DLN	88.9	89.0	88.8
DLN Trap. on Freq.	88.6	88.8	88.3
DLN Trap. on Order	88.6	88.9	88.0
DLN Trap. on Both	88.7	88.5	88.0

Each set function was trained on a different, actively-sampled dataset of only relatively difficult examples and higher-quality classification labels. For the DLN, we constrained the set function to be positive in each ngram's SFE estimate. Six other features were used for each ngram: we imposed trapezoid constraints so that the ngram estimate was conditioned on the ngram frequency and ngram order. An eighth feature to the set function was the $M(x)$ for the query. Because of the large size of the token set and the fact that we used both a large noisy training set for the SFE and a smaller precise training set to learn the set function, we did not attempt to compare to a DNN or to the mean SFE token estimate. Results in Table 5 show the DLN's worked slightly better than the deep sets, as well as being easier to debug and understand.

Table 5. User Intent From Queries Accuracy

	Train	Val.	Test
Deep Sets	65.22	65.33	64.82
DLN	65.38	65.41	64.93
DLN Trap.	65.40	65.48	65.01

References

Bach, F. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends in Machine Learning*, 6(2), 2013.

Canini, K., Cotter, A., Milani Fard, M., Gupta, M. R., and Pfeifer, J. Fast and flexible monotonic functions with ensembles of lattices. *Advances in Neural Information Processing Systems (NIPS)*, 2016.

Chetverikov, D., Santos, A., and Shaikh, A. M. The econometrics of shape restrictions. *Annual Review of Economics*, 2018.

Cormier, Q., Milani Fard, M., and Gupta, M. R. Launch and iterate: Reducing prediction churn. *Advances in Neural Information Processing Systems (NIPS)*, 2016.

Cotter, A., Gupta, M. R., and Pfeifer, J. A Light Touch for heavily constrained SGD. In *29th Annual Conference on Learning Theory*, pp. 729–771, 2016.

Daniels, H. and Velikova, M. Monotone and partially monotone neural networks. *IEEE Trans. Neural Networks*, 21 (6):906–917, 2010.

Dietterich, T. G., Lathrop, R. H., and Lozano-Perez, T. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89:31–71, 1997.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal Machine Learning Research*, 12:2121–2159, 2011.

Feldman, S., Gupta, M. R., and Frigyik, B. A. Revisiting Steins paradox: Multi-task averaging. *Journal Machine Learning Research*, 2014.

Friedman, J. H. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

Groeneboom, P. and Jongbloed, G. *Nonparametric estimation under shape constraints*. Cambridge Press, New York, USA, 2014.

Gupta, M. R., Gray, R. M., and Olshen, R. A. Nonparametric supervised learning by linear interpolation with maximum entropy. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28(5):766–781, 2006.

Gupta, M. R., Cotter, A., Pfeifer, J., Voevodski, K., Canini, K., Mangylov, A., Moczydlowski, W., and Esbroeck, A. V. Monotonic calibrated interpolated look-up tables. *Journal of Machine Learning Research*, 17:1–47, 2016.

Gupta, M. R., Bahri, D., Cotter, A., and Canini, K. Diminishing returns shape constraints for interpretability and regularization. *Advances in Neural Information Processing Systems (NIPS)*, 2018.

Kondor, R. and Jebara, T. A kernel between sets of vectors. In *Proceedings of the 20th International Conference on Machine Learning (ICML)*, pp. 361–368, 2003.

Law, H., Sutherland, D. J., Sejdinovic, D., and Flaxman, S. Bayesian approaches to distribution regression. *AISTATS*, 2018.

Milgrom, P. and Roberts, J. Complementarities and fit: Strategy, structure, and organizational change in manufacturing. *Journal of Accounting and Economic*, 19: 179–208, 1995.

Muandet, K., Fukumizu, K., Dinuzzo, F., and Schoelkopf, B. Learning from distributions with support measure machines. *Advances in Neural Information Processing Systems (NIPS)*, 2012.

Poczos, B., Xiong, L., Sutherland, D., and Schenider, J. Nonparametric kernel estimators for image classification. *Computer Vision and Pattern Recognition*, 2012.

Póczos, B., Rinaldo, A., Singh, A., and Wasserman, L. Distribution-free distribution regression. *AISTATS*, 2013.

Rosenthal, A. What are set functions? *The American Mathematical Monthly*, 55:14–20, 1948.

Shivaswamy, P. K. and Jebara, T. Permutation invariant SVMs. In *Advances in Neural Information Processing Systems*, 2006.

Szabo, Z., Sriperumbudur, B., and Poczos, B. Learning theory for distribution regression. *Journal Machine Learning Research*, 2016.

Torra, V. and Naurkawa, Y. *Modeling Decisions: Information fusion and aggregation operators*. Springer, Berlin, Germany, 2010.

You, S., Ding, D., Canini, K., Pfeifer, J., and Gupta, M. R. Deep lattice networks and partial monotonic functions. *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R., and Smola, A. Deep sets. *Advances in Neural Information Processing Systems (NIPS)*, 2017.