OPT2021: 13th Annual Workshop on Optimization for Machine Learning

Farkas' Theorem of the Alternative for Prior Knowledge in Deep Networks

Suhaas Bhat SUHAASBHAT@COLLEGE.HARVARD.EDU
Harvard University
Massachusetts Hall
Cambridge, MA 02138

Jeffery Kline JKLIN1@AMFAM.COM
Glenn Fung GFUNG@AMFAM.COM
6000 American Parkway
Madison, WI 53783

Abstract

In this paper, prior knowledge expressed in the form of polyhedral sets is introduced into the training of a deep neural network. This approach to using prior knowledge extends earlier work that applies Farkas' Theorem of the Alternative to linear support vector machine classifiers. Through this extension, we gain the ability to sculpt the decision boundary of a neural network by training on a set of discrete data while simultaneously fitting an uncountable number of points that live within a polytope that is defined by prior knowledge. We demonstrate viability of this approach on both synthetic and benchmark data sets.

1. Introduction

Prior knowledge allows one to regularize solutions of network models using information that is either implicit in observations, or known from experience. In this work, we incorporate prior knowledge that is represented as polyhedral sets into deep neural networks. Our approach builds on the technique introduced in [4] that applies a consequence of Farkas' Lemma, which concerns solvability of linear systems, to linear support vector machines (SVM). Prior knowledge expressed as polyhedral sets is quite general: one asserts that entire *regions* are contained within one of two linear half-spaces.

When training networks using prior knowledge, one uses a hybrid of "virtual" and instance-based information [14]. Since prior knowledge exists in a wide variety of forms, significant effort has been spent developing techniques that incorporate this extra information into existing machine learning models [3, 10]. Our paper is another contribution to this effort. In addition to polyhedral sets, other forms of prior knowledge include logical rules [15], invariance under group transformations [13], and radial symmetry [6].

Contributions In this paper, we extend *polyhedral* knowledge to deep learning architectures. Polyhedral knowledge is a parsimonious description of entire *regions* containing an uncountable number of points. These regions supplement training data. To our knowledge, this is the first attempt to use knowledge in this way for deep networks. Our approach can be used for data augmentation, model robustness, applications of fairness, and regularization.

Notation All vectors are column vectors. If $x \in \mathbb{R}^n$, the ith entry of x is denoted x_i. The notation $A \in \mathbb{R}^{m \times n}$ represents a matrix, and A_i represents row i of the matrix A. e is a vector of ones. A hyperplane in $\mathbb{R}^n$ is specified by a vector $w \in \mathbb{R}^n$ and a scalar $\gamma \in \mathbb{R}$. The hyperplane defined by the pair (w, γ) is the set of $x \in \mathbb{R}^n$ that satisfy $w'x = \gamma$. A convex polyhedral region in $\mathbb{R}^n$ is described by all $x \in \mathbb{R}^n$ that satisfy a linear inequality $Bx \leq b$, where $B \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. Unless otherwise specified, we will assume such sets are nonempty and convex.

2. Linear Support Vector Machines and Polyhedral Prior Knowledge

Our formulation of polyhedral knowledge for neural networks adapts the technique introduced for use with linear support vector machines in [4]. For reasons of space, we refer reader to that reference for details. The following proposition is the fundamental result of polyhedral knowledge for linear support vector machines.

Proposition 1 *Let $K := \{x : Bx \leq b\}$ be nonempty. Then K lies in the halfspace $\{x : x'w \geq \gamma + 1\}$ if and only if there exists $u \in \mathbb{R}^m$ that satisfies $B'u + w = 0$, $b'u + \gamma + 1 \leq 0$, $u \geq 0$.*

The proof of this proposition combines Farkas' Lemma with the observation that the statement in the proposition is equivalent to the statement, "$Bx \leq b$, $x'w < \gamma+1$ has no solution." Equivalently, for a two class SVM, $Bx \leq b$ implies that the label of x is $+1$. A similar kind of statement holds for the points with label -1. The set $\{x : Bx \leq b\}$ is a polytope, and in the context of training a network, we call it a *knowledge set*. Figure 1 illustrates the effect that polyhedral prior knowledge

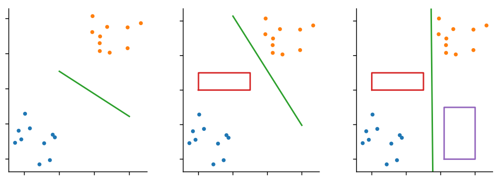

Figure 1: Example of a soft margin hyperplane found by solving a standard SVM (*left*), an SVM with one polyhedral knowledge set (*middle*) and an SVM with two polyhedral knowledge sets (*right*).

has on the SVM separating hyperplane. In the examples shown, two linearly separable sets live in $\mathbb{R}^2$, and they are used to train a linear classifier. A segment of the decision boundary is shown. The middle and right axes show the data as well as the knowledge sets, along with a segment of the decision boundary found by solving a linear program [4]. In contrast to a standard linear classifier, *regions* and discrete points serve as part of the training set. Note that the program does not use the vertices of the regions, which are left implicitly defined.

3. Polyhedral Knowledge for Neural Networks

We now show how to encode polyhedral knowledge in a way that is compatible with a neural network. We adapt the linear SVM theory of Section 2 to the training of the network. Our data consist of points living in two disjoint classes, A^+ and A^-, with corresponding labels $y \in \{1, -1\}$, both of which live in the input space, $\mathbb{R}^n$.

Embedding Let $\Psi : \mathbb{R}^n \to \mathbb{R}^h$ denote the mapping from the input space to the penultimate layer of a feed-forward network, and let $\Phi : \mathbb{R}^n \to \mathbb{R}$ denote the mapping from the input space to the final layer of the network. For binary classification, one trains by minimizing $\sum_i \|\Phi(A_i) - y_i\|_p$. The function Ψ is an embedding function, for it maps points in the input space to $\mathbb{R}^h$. Assume the trained network is an ideal classifier. Since the last layer of the network is *linear*, we have that $\Psi(A^+)$ and $\Psi(A^-)$ are linearly separable in $\mathbb{R}^h$. That is, the final layer of the trained network describes $w \in \mathbb{R}^h$ and $\gamma \in \mathbb{R}$ that define a separating hyperplane for the sets $\Psi(A^+)$ and $\Psi(A^-)$.

Polyhedral knowledge Polyhedral knowledge is expressed as a collection of inequalities that apply to points in $\mathbb{R}^n$ (the input space). We need to create a collection of new inequalities that holds for the embedding of our points in $\mathbb{R}^h$. Let B and b characterize one of the knowledge sets in $\mathbb{R}^n$ through the inequality $Bx \leq b$. There is equivalence between the set of x that satisfy $Bx \leq b$, and the set of extremal vertices of this set. Let V denote the extremal vertices of our knowledge set. Now consider the embedding of these points, namely $\Psi(V)$. The convex hull of $\Psi(V)$ is the set of $\xi \in \mathbb{R}^h$ that satisfy $F\xi \leq f$ for some matrix F and column vector f. We will let (F, f) represent the embeddings of the knowledge set (B, b).

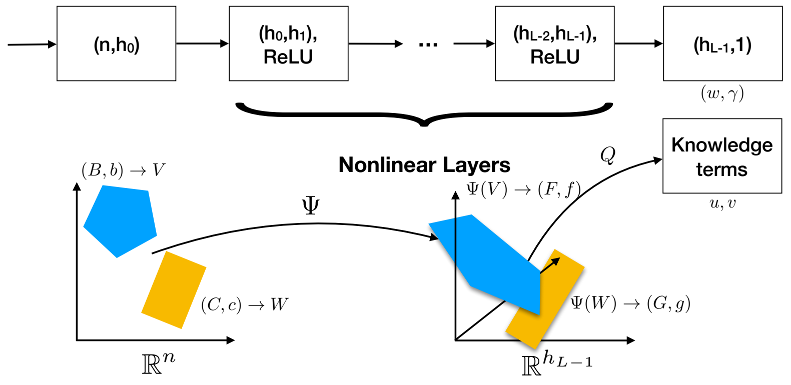

Figure 2: Incorporating prior knowledge into a neural network relies on the function Ψ that maps the extremal vertices of the knowledge sets, expressed in the input space of dimension n, into the embedding space of dimension h. Qhull is used to construct embedded knowledge sets.

Network loss A linear SVM solves a constrained program, but network training optimizes an unconstrained program. To adapt the constrained program to our network, wherever a constraint is desired, we add to the network loss function a penalty term, unless the constraint is variable nonnegativity, in which case we replace the variable with its square. The network loss function has three types of terms: one for model fit, one each for knowledge sets of the $+1$ and the -1 classes.

Let (B^i, b^i) and (C^j, c^j) define knowledge sets for $+1$ and -1 classes, respectively. Let V_i denote the vertices of the polytope defined by (B^i, b^i), and let W^j denote the vertices of the polytope defined by (C^j, c^j). Let (F^i, f^i) and (G^j, g^j) denote the inequalities that define the convex hulls of $\Psi(V^i)$ and $\Psi(W^j)$, respectively. We represent the weight and bias parameters of the last layer of our network using w and γ. The complete form of the network's loss function is:

$$\mathcal{L}_{\{(F^i,f^i)\},\{(G^j,g^j)\}}(A,y) := \sum_m \|\Phi(A_m) - y_m\| + \nu \left(\sum_{0\le i<k} \left\| {F^i}' \left(u^i\right)^2 + w \right\| + \left({f^i}' \left(u^i\right)^2 + \gamma + 1\right)_+ + \sum_{0\le j<\ell} \left\| {G^j}' \left(v^j\right)^2 - w \right\| + \left({g^j}' \left(v^j\right)^2 - \gamma + 1\right)_+ \right),$$

where $\nu \ge 0$ is a parameter. The only structural difference between a standard network and the one described here is the addition of the parameters u^i and v^j, along with several terms that act on the final layer. The best choice of norm to use in the loss function remains an open question, and we were guided by empirical results. We experimented with a variety of alternatives, including a sum of squared Euclidean norms.

Algorithm 1 Forward evaluation of the network loss function, $\mathcal{L}$. The knowledge sets (B^i, b^i) and (C^j, c^j) are fixed attributes of the network, as are the extreme vertices of these sets, denoted V^i and W^j, respectively. The network contains parameters u^i and v^j that need to be trained with backpropagation. These weights do not interact with the data directly, but only with the knowledge sets and parameters of the last layer. Convex hulls are found by calling Qhull[2].

```
1: procedure $\mathcal{L}(A, y)$
2:     $l_0 \leftarrow \sum_m \|\Phi(A_m) - y_m\|$        ▷ Standard loss applied to data and label
3:     $(F^i, f^i) \leftarrow$ Qhull($\Psi(V^i)$)        ▷ Qhull constructs the convex hull of embedded vertices
4:     $(G^j, g^j) \leftarrow$ Qhull($\Psi\left(W^j\right)$)
5:     $l_1 \leftarrow \sum_{0\le i<k} \left\| {F^i}' \left(u^i\right)^2 + w \right\| + \left({f^i}' \left(u^i\right)^2 + \gamma + 1\right)_+$        ▷ Knowledge term for $\left\{(B^i, b^i)\right\}$
6:     $l_2 \leftarrow \sum_{0\le j<\ell} \left\| {G^j}' \left(v^j\right)^2 - w \right\| + \left({g^j}' \left(v^j\right)^2 - \gamma + 1\right)_+$        ▷ Knowledge term for $\left\{(C^j, c^j)\right\}$
7:     return $l_0 + \nu(l_1 + l_2)$
8: end procedure
```

Algorithm 1 provides details of the foreward evaluation of the network loss function. For the three terms l_0, l_1 and l_2, backpropagation in response to l_0 is standard. For l_1, the matrices F^i and the vectors f^i are treated as constants. This means the gradient with respect to u^i, w and γ is elementary to compute. A similar statement holds for l_2.

4. Numerical Experiments

In this section, we report results of several numerical experiments to demonstrate feasibility of the foregoing. Our implementation uses the software libraries PyTorch, NumPy, SciPy and Qhull [2, 8, 11, 16]. The compute resource for all examples was a 2.3 GHz 8-Core Intel Core i9. All models trained using the Adam optimizer, and the learning rate was held fixed at 0.001.

2D Examples In Figure 3 (first row), we consider four points in the plane arranged in an XOR pattern and four knowledge sets, also arranged in an XOR pattern. The XOR configuration is interesting because it is not linearly separable. Also note that three of the knowledge sets do not contain any data. The data and arrangement here approximates a similar example used to demonstrate polyhedral knowledge for nonlinear kernel SVMs in [5]. The top left and middle right axes show the values of the network's functional Φ evaluated on a subset of the plane, and the middle left and far right axes show $\operatorname{sign}\Phi$ on that same subset.

The Figure shows the results of training a network with two hidden layers, each of dimension 6. In the top left pair, no knowledge was used during training, and while the data are all correctly classified, the boundaries of the knowledge sets are violated. In the top right pair, we trained a network having the same structure, except that for this image knowledge was added to the network. The data are all still correctly classified, but the decision boundary has been shifted, and it has essentially no intersection with the knowledge sets. The top left axes converged after 1000 epochs. The top right axes trained for 5000 epochs. Both required less than 1 minute of CPU time to train.

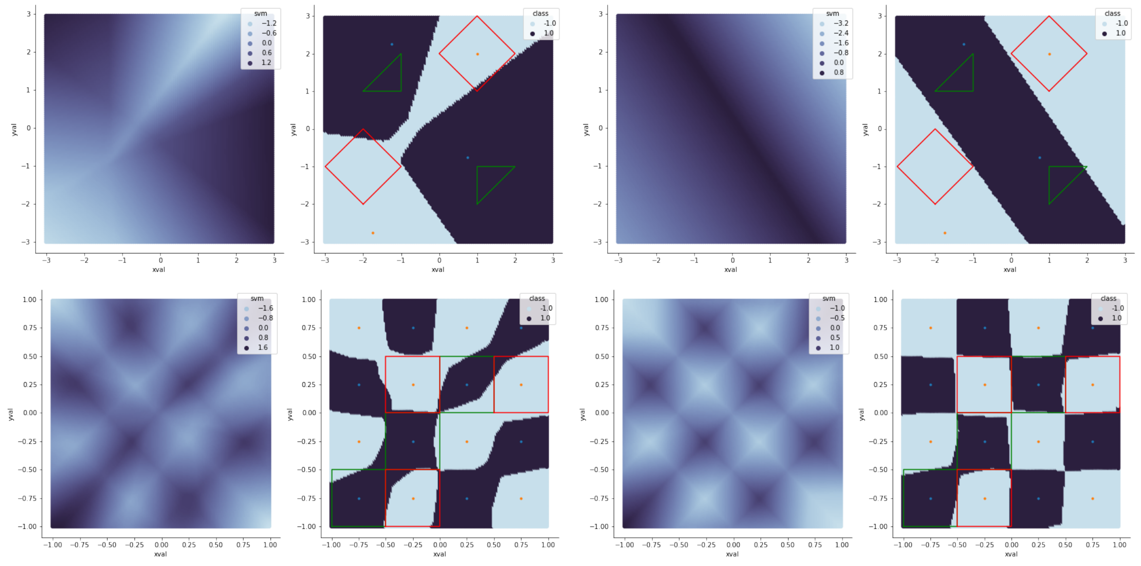

Figure 3: The top row shows results of training a 2-layer network with hidden dimension 6 on 4 points of data arranged in an XOR configuration without (left) and with (right) knowledge. The bottom row shows the same, but for a checkerboard pattern. The boundaries of the checkerboard are approximately learned, even though there are no additional observations added to the training.

Figure 3 (second row) shows the results of a similar experiment. In this figure, the network consists of two layers, with hidden dimension 160 and embedding dimension 20. The data consists of 16 points, arranged in a 4×4 checkerboard pattern. The knowledge consists of three essentially disjoint squares for each class. As before, neither the knowledge sets nor the data are linearly separable when represented in the plane.

In the left pair of axes, 16 data points were used to train a network, and no knowledge was present. The data are all correctly classified, but boundaries are unconstrained. In the right pair of axes, the six knowledge sets are used as well as 16 points to train the network. The network correctly classifies all the data, and it also approximates the desired boundaries that the knowledge sets describe. Note that the vertices of the knowledge sets are not included in the training set. The network of the left pair of axes converged after 1000 epochs, the network in the right pair ran for 20000 epochs, and it required 160 seconds to train.

MNIST One-Shot Learning Experiments We report here the results of one-shot learning applied to the MNIST dataset. For these experiments, we fed the model only 1 datapoint for each class, and tested discriminatory power of a binary classifier between digit pairs $(1, 7)$ and $(4, 9)$. For each digit, the associated knowledge provided to the neural net was generated by doing 10-means clustering on the entire MNIST dataset terms of that digit, and generating the convex hull of those points. On this task, the addition of this knowledge provides a significant performance improvement. As a result, we believe that knowledge-based neural net classifiers could be a very powerful tool for few-shot learning.

Table 1: An example of polyhedral knowledge applied to a sample of MNIST data. Only 1 data point is fed into the model, and the knowledge is the convex hull of 10 means from k-mean clustering the full MNIST dataset.

Digits	No Knowledge		With Knowledge	
	Prec	AUC	Prec	AUC
1 vs 7	0.951 ± 0.071	0.895 ± 0.061	$\mathbf{0.980} \pm 0.008$	$\mathbf{0.952} \pm 0.002$
4 vs 9	0.628 ± 0.072	0.597 ± 0.044	$\mathbf{0.632} \pm 0.006$	$\mathbf{0.629} \pm 0.006$

5. Conclusion and Future Work

We have presented a knowledge-based formulation that allows one to train neural networks using a hybrid of data and polyhedral sets. The classifier is obtained by using the network to embed polytopes that live in the input space, and then to directly apply the theory of prior polyhedral knowledge for linear SVMs to the last layer. In doing this, we take advantage of the fact that the last layer of a network can be treated as a linear classifier, and so the previous layers act as a kernel, which we learn during the course of training. It is possible to modify existing networks to accommodate this type of knowledge by adding a small number of variables to the network and straightforward modifications to the network loss function.

Given the appeal to convexity theory in our framework, future work can explore the question of whether input convex neural networks [1] can benefit the embedding of polytopes. With our current method, there is no special structure imposed on the embedding function Ψ, which is implemented using Qhull. A practical alternative implementation could sample from each polytope's convex hull, and then use the samples to more directly control how the polytopes embed. Recent work [7, 9, 12] may facilitate using a black-box process for the construction of the convex hull of embedded vertices as an approximated differentiable process. In this way, the base network will synergize with the convex hull construction smoothly during the end-to-end gradient-based optimization process.

References

[1] Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. In *International Conference on Machine Learning*, pages 146–155. PMLR, 2017.

[2] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE*, 22(4):469–483, 1996.

[3] Mayukh Das, Devendra Dhami, Yang Yu, Gautam Kunapuli, and Sriraam Natarajan. Human-guided learning of column networks: Knowledge injection for relational deep learning. In *8th ACM IKDD CODS and 26th COMAD*, pages 110–118, 01 2021. doi: 10.1145/3430984.3431018.

[4] Glenn Fung, Olvi Mangasarian, and Jude Shavlik. Knowledge-based support vector machine classifiers. In *Advances in Neural Information Processing Systems*, pages 521–528, 01 2002.

[5] Glenn Fung, Olvi Mangasarian, and Jude Shavlik. Knowledge-based nonlinear kernel classifiers. *Lecture Notes in Computer Science*, 08 2003. doi: 10.1007/978-3-540-45167-9_9.

[6] Federico Girosi and Nicholas Tung Chan. Prior knowledge and the creation of "virtual" examples for rbf networks. In *In Neural networks for signal processing, Proceedings of the 1995 IEEE-SP Workshop*, pages 201–210, 1995.

[7] Will Grathwohl, Dami Choi, Yuhuai Wu, Geoffrey Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation, 2018.

[8] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

[9] Alon Jacovi, Guy Hadash, Einat Kermany, B. Carmeli, Ofer Lavi, George Kour, and Jonathan Berant. Neural network gradient-based learning of black-box function interfaces. *ArXiv*, abs/1901.03995, 2019.

[10] Fabien Lauer and Gérard Bloch. Incorporating prior knowledge in support vector machines for classification: A review. *Neurocomputing*, 71(7):1578–1594, 2008. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2007.04.010. URL https://www.sciencedirect.com/science/article/pii/S0925231207001439. Progress in Modeling, Theory, and Application of Computational Intelligenc.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[12] Elad Sarafian, Mor Sinay, Yoram Louzoun, Noa Agmon, and Sarit Kraus. Explicit gradient learning for black-box optimization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of*

Machine Learning Research, pages 8480–8490. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/sarafian20a.html.

[13] B Schölkopf, P. Y Simard, A. J Smola, and V. N Vapnik. Prior knowledge in support vector kernels. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural information processings systems*, volume 10, pages 640–646, Cambridge, MA, 1998. MIT Press. URL /papers/invar-final.ps.gz.

[14] Geoffrey G. Towell and Jude W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1):119–165, 1994. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(94)90105-8. URL https://www.sciencedirect.com/science/article/pii/0004370294901058.

[15] Geoffrey G. Towell, Jude W. Shavlik, and Michiel O. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *In Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 861–866, 1990.

[16] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.