Input Convex Neural Networks

Brandon Amos [1]   Lei Xu [2 *]   J. Zico Kolter [1]

Abstract

This paper presents the input convex neural network architecture. These are scalar-valued (potentially deep) neural networks with constraints on the network parameters such that the output of the network is a convex function of (some of) the inputs. The networks allow for efficient inference via optimization over some inputs to the network given others, and can be applied to settings including structured prediction, data imputation, reinforcement learning, and others. In this paper we lay the basic groundwork for these models, proposing methods for inference, optimization and learning, and analyze their representational power. We show that many existing neural network architectures can be made input-convex with a minor modification, and develop specialized optimization algorithms tailored to this setting. Finally, we highlight the performance of the methods on multi-label prediction, image completion, and reinforcement learning problems, where we show improvement over the existing state of the art in many cases.

1. Introduction

In this paper, we propose a new neural network architecture that we call the *input convex neural network* (ICNN). These are *scalar-valued* neural networks $f(x, y; \theta)$ where x and y denotes inputs to the function and θ denotes the parameters, built in such a way that the network is convex in (a subset of) *inputs* y.[3] The fundamental benefit to these ICNNs is that we can *optimize* over the convex inputs to the network given some fixed value for other inputs. That is, given some fixed x (and possibly some fixed elements of

Because the problem is convex, it is easier to optimize.

thor was at Carnegie Mellon University. ter Science, Carnegie Mellon University. [2]Department of Computer Science and Technology, Tsinghua University. Beijing, China. Correspondence to: Brandon Amos <bamos@cs.cmu.edu>, J. Zico Kolter <zkolter@cs.cmu.edu>.

y) we can globally and efficiently (because the problem is convex) solve the optimization problem

$$\underset{y}{\operatorname{argmin}} f(x, y; \theta). \tag{1}$$

Fundamentally, this formalism lets us perform inference in the network via *optimization*. That is, instead of making predictions in a neural network via a purely feedforward process, we can make predictions by optimizing a scalar function (which effectively plays the role of an energy function) over some inputs to the function given others. There are a number of potential use cases for these networks.

Structured prediction   As is perhaps apparent from our notation above, a key application of this work is in structured prediction. Given (typically high-dimensional) structured input and output spaces $\mathcal{X} \times \mathcal{Y}$, we can build a network over (x, y) pairs that encodes the energy function for this pair, following typical energy-based learning formalisms (LeCun et al., 2006). Prediction involves finding the $y \in \mathcal{Y}$ that minimizes the energy for a given x, which is exactly the argmin problem in (1). In our setting, assuming that $\mathcal{Y}$ is a convex space (a common assumption in structured prediction), this optimization problem is convex. This is similar in nature to the structured prediction energy networks (SPENs) (Belanger & McCallum, 2016), which also use deep networks over the input and output spaces, with the difference being that in our setting f is convex in y, so the optimization can be performed globally.

Data imputation   Similar to structured prediction but slightly more generic, if we are given some space $\mathcal{Y}$ we can learn a network $f(y; \theta)$ (removing the additional x inputs, though these can be added as well) that, given an example with some subset $\mathcal{I}$ missing, imputes the likely values of these variables by solving the optimization problem as above $\hat{y}_{\mathcal{I}} = \operatorname{argmin}_{y_{\mathcal{I}}} f(y_{\mathcal{I}}, y_{\bar{\mathcal{I}}}; \theta)$ This could be used

---

[3]We emphasize the term "input convex" since convexity in machine learning typically refers to convexity (of the loss minimization learning problem) in the *parameters*, which is not the case here. Note that in our notation, f needs only be a convex function in y, and may still be non-convex in the remaining inputs x. Training these neural networks remains a nonconvex problem, and the convexity is only being exploited at inference time.

e.g., in image inpainting where the goal is to fill in some arbitrary set of missing pixels given observed ones.

Continuous action reinforcement learning Given a reinforcement learning problem with potentially continuous state and action spaces $\mathcal{S} \times \mathcal{A}$, we can model the (negative) Q function, $-Q(s, a; \theta)$ as an input convex neural network. In this case the action selection procedure can be formulated as a convex optimization problem $a^\star(s) = \operatorname{argmin}_a -Q(s, a; \theta)$.

This paper lays the foundation for optimization, inference, and learning in these input convex models, and explores their performance in the applications above. Our main contributions are: we propose the ICNN architecture and a partially convex variant; we develop efficient optimization and inference procedures that are well-suited to the complexity of these specific models; we propose techniques for training these models, based upon either max-margin structured prediction or direct differentiation of the argmin operation; and we evaluate the system on multi-label prediction, image completion, and reinforcement learning domains; in many of these settings we show performance that improves upon the state of the art.

2. Background and related work

Energy-based learning The interplay between inference, optimization, and structured prediction has a long history in neural networks. Several early incarnations of neural networks were explicitly trained to produce structured sequences (e.g. (Simard & LeCun, 1991)), and there was an early appreciation that structured models like hidden Markov models could be combined with the outputs of neural networks (Bengio et al., 1994). Much of this earlier work is surveyed and synthesized by (LeCun et al., 2006), who give a tutorial on these energy based learning methods. In recent years, there has been a strong push to further incorporate structured prediction methods like conditional random fields as the "last layer" of a deep network architecture (Peng et al., 2009; Zheng et al., 2015; Chen et al., 2015). Several methods have proposed to build general neural networks over joint input and output spaces, and perform inference over outputs using generic optimization techniques such as Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) and Structured Prediction Energy Networks (SPENs) (Belanger & McCallum, 2016). SPENs provide a deep structure over input and output spaces that performs the inference in (1) as a non-convex optimization problem.

The current work is highly related to these past approaches, but also differs in a very particular way. To the best of our knowledge, each of these structured prediction methods based upon energy-based models operates in one of two ways, either: 1) the architecture is built in a very particular way such that optimization over the output is guaranteed to be "easy" (e.g. convex, or the result of running some inference procedure), usually by introducing a structured linear objective at the last layer of the network; or 2) no attempt is made to make the architecture "easy" to run inference over, and instead a general model is built over the output space. In contrast, our approach lies somewhere in between: by ensuring convexity of the resulting decision space, we are constraining the inference problem to be easy in some respect, but we specify very little about the architecture other than the constraints required to make it convex. In particular, as we will show, the network architecture over the variables to be optimized over can be deep and involve multiple non-linearities. The goal of the proposed work is to allow for complex functions over the output without needing to specify them manually (exactly analogous to how current deep neural networks treat their input space).

Structured prediction and MAP inference Our work also draws some connection to MAP-inference-based learning and approximate inference. There are two broad classes of learning approaches in structured prediction: method that use probabilistic inference techniques (typically exploiting the fact that the gradient of log likelihood is given by the actual feature expectations minus their expectation under the learned model (Koller & Friedman, 2009, Ch 20)), and methods that rely solely upon MAP inference (such as max-margin structured prediction (Taskar et al., 2005; Tsochantaridis et al., 2005)). MAP inference in particular also has close connections to optimization, as various convex relaxations of the general MAP inference problem often perform well in theory and practice. The proposed methods can be viewed as an extreme case of this second class of algorithm, where inference is based *solely* upon a convex optimization problem that may not have any probabilistic semantics at all. Finally, although it is more abstract, we feel there is a philosophical similarity between our proposed approach and sum-product networks (Poon & Domingos, 2011); both settings define networks where inference is accomplished "easily" either by a sum-product message passing algorithm (by construction) or via convex optimization.

Fitting convex functions Finally, the proposed work relates to a topic less considered in the machine learning literature, that of fitting convex functions to data (Boyd & Vandenberghe, 2004, pg. 338). Indeed our learning problem can be viewed as parameter estimation under a model that is guaranteed to be convex by its construction. The most similar work of which we are aware specifically fits sums of rectified half-planes to data (Magnani & Boyd, 2009), which is similar to one layer of our rectified linear units. However, the actual training scheme is much differ-
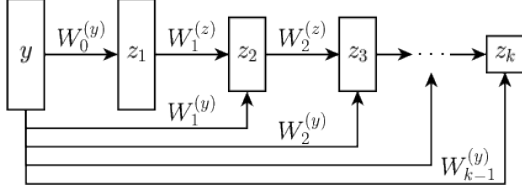
Figure 1. A fully input convex neural network (FICNN).

Figure 2. A partially input convex neural network (PICNN).

ent, and our deep network architecture allows for a much richer class of representations, while still maintaining convexity.

3. Convex neural network architectures

Here we more formally present different ICNN architectures and prove their convexity properties given certain constraints on the parameter space. Our chief claim is that the class of (full and partial) input convex models is rich and lets us capture complex joint models over the input to a network.

3.1. Fully input convex neural networks

To begin, we consider a fully convex, k-layer, fully connected ICNN that we call a FICNN and is shown in Figure 1. This model defines a neural network over the input y (i.e., omitting any x term in this function) using the architecture for $i = 0, \ldots, k-1$

$$z_{i+1} = g_i\left(W_i^{(z)} z_i + W_i^{(y)} y + b_i\right), \quad f(y;\theta) = z_k \quad (2)$$

where z_i denotes the layer activations (with $z_0, W_0^{(z)} \equiv 0$), $\theta = \{W_{0:k-1}^{(y)}, W_{1:k-1}^{(z)}, b_{0:k-1}\}$ are the parameters, and g_i are non-linear activation functions. The central result on convexity of the network is the following:

Proposition 1. *The function f is convex in y provided that all $W_{1:k-1}^{(z)}$ are non-negative, and all functions g_i are convex and non-decreasing.*

The proof is simple and follows from the fact that non-negative sums of convex functions are also convex and that the composition of a convex and convex non-decreasing function is also convex (see e.g. Boyd & Vandenberghe (2004, 3.2.4)). The constraint that the g_i be convex non-decreasing is not particularly restrictive, as current non-linear activation units like the rectified linear unit or max-pooling unit already satisfy this constraint. The constraint that the $W^{(z)}$ terms be non-negative is somewhat restrictive, but because the bias terms and $W^{(y)}$ terms can be negative, the network still has substantial representation power, as we will shortly demonstrate empirically.

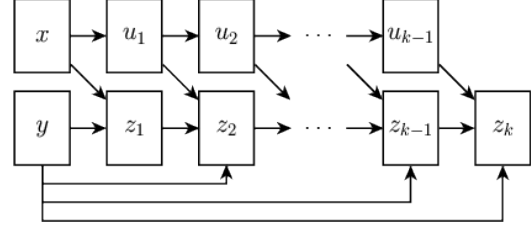One notable addition in the ICNN are the "passthrough" layers that directly connect the input y to hidden units in deeper layers. Such layers are unnecessary in traditional feedforward networks because previous hidden units can always be mapped to subsequent hidden units with the identity mapping; however, for ICNNs, the non-negativity constraint subsequent $W^{(z)}$ weights restricts the allowable use of hidden units that mirror the identity mapping, and so we explicitly include this additional passthrough. Some passthrough layers have been recently explored in the deep residual networks (He et al., 2015) and densely connected convolutional networks (Huang et al., 2016), though these differ from those of an ICNN as they pass through hidden layers deeper in the network, whereas to maintain convexity our passthrough layers can only apply to the input directly.

Other linear operators like convolutions can be included in ICNNs without changing the convexity properties. Indeed, modern feedforward architectures such as AlexNet (Krizhevsky et al., 2012), VGG (Simonyan & Zisserman, 2014), and GoogLeNet (Szegedy et al., 2015) with Re-LUs (Nair & Hinton, 2010) can be made input convex with Proposition 1. In the experiment that follow, we will explore ICNNs with both fully connected and convolutional layers, and we provide more detail about these additional architectures in Section A of the supplement.

3.2. Partially input convex architectures

The FICNN provides joint convexity over the entire input to the function, which indeed may be a restriction on the allowable class of models. Furthermore, this full joint convexity is unnecessary in settings like structured prediction where the neural network is used to build a joint model over an input and output example space and only convexity over the outputs is necessary.

In this section we propose an extension to the pure FICNN, the partially input convex neural network (PICNN), that is convex over only some inputs to the network (in general ICNNs will refer to this new class). As we will show, these networks generalize both traditional feedforward networks and FICNNs, and thus provide substantial representational benefits. We define a PICNN to be a network over (x, y) pairs $f(x, y; \theta)$ where f is convex in y but not convex in x. Figure 2 illustrates one potential k-layer PICNN architec-

ture defined by the recurrences

$$u_{i+1} = \tilde{g}_i(\tilde{W}_i u_i + \tilde{b}_i)$$
$$z_{i+1} = g_i \left(W_i^{(z)} \left(z_i \circ [W_i^{(zu)} u_i + b_i^{(z)}]_+ \right) + \right.$$
$$\left. W_i^{(y)} \left(y \circ (W_i^{(yu)} u_i + b_i^{(y)}) \right) + W_i^{(u)} u_i + b_i \right)$$
$$f(x, y; \theta) = z_k, \ u_0 = x$$

$$(3)$$

where $u_i \in \mathbb{R}^{n_i}$ and $z_i \in \mathbb{R}^{m_i}$ denote the hidden units for the "x-path" and "y-path", where $y \in \mathbb{R}^p$, and where $\circ$ denotes the Hadamard product, the elementwise product between two vectors. The crucial element here is that unlike the FICNN, we only need the $W^{(z)}$ terms to be nonnegative, and we can introduce arbitrary products *between* the u_i hidden units and the z_i hidden units. The following proposition highlights the representational power of the PICNN.

Proposition 2. *A PICNN network with k layers can represent any FICNN with k layers and any purely feedforward network with k layers.*

Proof. To recover a FICNN we simply set the weights over the entire x path to be zero and set $b^{(z)} = b^{(y)} = 1$. We can recover a feedforward network by noting that a traditional feedforward network $\hat{f}(x; \theta)$ where $f : \mathcal{X} \to \mathcal{Y}$, can be viewed as a network with an inner product $f(x; \theta)^T y$ in its last layer (see e.g. (LeCun et al., 2006) for more details). Thus, a feedforward network can be represented as a PICNN by setting the x path to be exactly the feedforward component, then having the y path be all zero except $W_{k-1}^{(yu)} = I$ and $W_{k-1}^{(y)} = 1^T$. $\square$

4. Inference in ICNNs

Prediction in ICNNs (which we also refer to as inference), requires solving the convex optimization problem

$$\underset{y \in \mathcal{Y}}{\text{minimize}} \ f(x, y; \theta) \qquad (4)$$

While the resulting tasks are convex optimization problems (and thus "easy" to solve in some sense), in practice this still involves the solution of a potentially very complex optimization problem. We discuss here several approaches for approximately solving these optimization problems. We can usually obtain reasonably accurate solutions in many settings using a procedure that only involves a small number of forward and backward passes through the network, and which thus has a complexity that is at most a constant factor worse than that for feedforward networks. The same consideration will apply to training such networks, which we will discuss in Section 5.

Exact inference in ICNNs Although it is not a practical approach for solving the optimization tasks, the inference problem for the networks presented above (where the non-linear are either ReLU or linear units) can be posed as as linear program. We show how to do this in Section B.

4.1. Approximate inference in ICNNs

Because of the impracticality of exact inference, we focus on approximate approaches to optimizing over the inputs to these networks, but ideally ones that still exploit the convexity of the resulting problem. We specifically focus on gradient-based approaches, which use the fact that we can easily compute the gradient of an ICNN with respect to its inputs, $\nabla_y f(x, y; \theta)$, using backpropagation.

Gradient descent. The simplest gradient-based methods for solving (4) is just (projected sub-) gradient descent, or modifications such as those that use a momentum term (Polyak, 1964; Rumelhart et al., 1988), or spectral step size modifications (Barzilai & Borwein, 1988; Birgin et al., 2000). That is, we start with some initial $\hat{y}$ and repeat the update

$$\hat{y} \leftarrow \mathcal{P}_{\mathcal{Y}} \left(\hat{y} - \alpha \nabla_y f(x, \hat{y}; \theta) \right) \qquad (5)$$

This method is appealing in its simplicity, but suffers from the typical problems of gradient descent on non-smooth objectives: we need to pick a step size and possibly use a sequence of decreasing step sizes, and don't have an obvious method to assess how accurate of a current solution we have obtained (since an ICNN with ReLUs is piecewise linear, it will not have zero gradient at the solution). The method is also more challenging to integrate with some learning procedures, as we often need to differentiate through an entire chain of the gradient descent algorithm (Domke, 2012). Thus, while the method can sometimes work in practice, we have found that other approaches typically far outperform this method, and we will focus on alternative approximate approaches for the remainder of this section.

4.2. Approximate inference via the bundle entropy method

An alternative approach to gradient descent is the bundle method (Smola et al., 2008), also known as the epigraph cutting plane approach, which iteratively optimizes a piecewise lower bound on the function given by the maximum over a set of first-order approximations. However, as, the traditional bundle method is not well suited to our setting (we need to evaluate a number of gradients equal to the dimension of x, and solve a complex optimization problem at each step) we have developed a new optimization algorithm for this domain that we term the *bundle entropy method*. This algorithm specifically applies to the (common) case where $\mathcal{Y}$ is bounded, which we assume to be $\mathcal{Y} = [0, 1]^n$

(other upper or lower bounds can be attained through scaling). The method is also easily extensible to the setting where elements of $\mathcal{Y}$ belong to a higher-dimensional probability simplex as well.

For this approach, we consider adding an additional "barrier" function to the optimization in the form of the negative entropy $-H(y)$, where

$$H(y) = -\sum_{i=1}^{n}(y_i \log y_i + (1 - y_i) \log(1 - y_i)). \quad (6)$$

In other words, we instead want to solve the optimization problem $\operatorname{argmin}_y \ f(x, y; \theta) - H(y)$ (with a possible additional scaling term). The negative entropy is a convex function, with the limits of $\lim_{y \to 0} H(y) = \lim_{y \to 1} H(y) = 0$, and negative values in the interior of this range. The function acts as a barrier because, although it does not approach infinity as it reaches the barrier of the feasible set, its gradient *does* approach infinity as it reaches the barrier, and thus the optimal solution will always lie in the interior of the unit hypercube $\mathcal{Y}$.

An appealing feature of the entropy regularization comes from its close connection with sigmoid units in typical neural networks. It follows easily from first-order optimality conditions that the optimization problem

$$\underset{y}{\text{minimize}} \ c^T y - H(y) \quad (7)$$

is given by $y^\star = 1/(1 + \exp(c))$. Thus if we consider the "trivial" PICNN mentioned in Section 3.2, which simply consists of the function $f(x, y; \theta) = y^T \tilde{f}(x; \theta)$ for some purely feedforward network $\tilde{f}(x; \theta)$, then the entropy-regularized minimization problem gives a solution that is equivalent to simply taking the sigmoid of the neural network outputs. Thus, the move to ICNNs can be interpreted as providing a more structured joint energy functional over the linear function implicitly used by sigmoid layers.

At each iteration of the bundle entropy method, we solve the optimization problem

$$y^{k+1}, t^{k+1} := \underset{y,t}{\operatorname{argmin}} \ \{t - H(y) \mid Gy + h \leq t1\} \quad (8)$$

where $G \in \mathbb{R}^{k \times n}$ has rows equal to

$$g_i^T = \nabla_y f(x, y^i; \theta)^T \quad (9)$$

and $h \in \mathbb{R}^k$ has entries equal to

$$h_i = f(x, y^i; \theta) - \nabla_y f(x, y^i; \theta)^T y^i. \quad (10)$$

The Lagrangian of the optimization problem is

$$\mathcal{L}(y, t, \lambda) = t - H(y) + \lambda^T (Gy + h - t1) \quad (11)$$

and differentiating with respect to y and t gives the optimality conditions

$$\nabla_y \mathcal{L}(y, t, \lambda) = 0 \implies y = \frac{1}{1 + \exp(G^T \lambda)}$$
$$\nabla_t \mathcal{L}(y, t, \lambda) = 0 \implies 1^T \lambda = 1 \quad (12)$$

which in turn leads to the dual problem

$$\underset{\lambda}{\text{maximize}} \ (G1 + h)^T \lambda - 1^T \log(1 + \exp(G^T \lambda))$$
$$\text{subject to} \ \lambda \geq 0, 1^T \lambda = 1. \quad (13)$$

This is a smooth optimization problem over the unit simplex, and can be solved using a method like the Projected Newton method of (Bertsekas, 1982, pg. 241, eq. 97). A complete description of the bundle entropy method is given in Section D. For lower dimensional problems, the bundle entropy method often attains an exact solution after a relatively small number of iterations. And even for larger problems, we find that the approximate solutions generated by a very small number of iterations (we typically use 5 iterations), still substantially outperform gradient descent approaches. Further, because we maintain an explicit lower bound on the function, we can compute an optimality gap of our solution, though in practice just using a fixed number of iterations performs well.

5. Learning ICNNs

Generally speaking, ICNN learning shapes the objective's energy function to produce the desired values when optimizing over the relevant inputs. That is, for a given input output pair $(x, y^\star)$, our goal is to find ICNN parameters θ such that

$$y^\star \approx \underset{y}{\operatorname{argmin}} \tilde{f}(x, y; \theta) \quad (14)$$

where for the entirely of this section, we use the notation $\tilde{f}$ to denote the combination of the neural network function *plus* the regularization term such as $-H(y)$, if it is included, i.e.

$$\tilde{f}(x, y; \theta) = f(x, y; \theta) - H(y). \quad (15)$$

Although we only discuss the entropy regularization in this work, we emphasize that other regularizers are also possible. Depending on the setting, there are several different approaches we can use to ensure that the ICNN achieves the desired targets, and we consider three approaches below: direct functional fitting, max-margin structured prediction, and argmin differentiation.

Direct functional fitting. We first note that in some domains, we do not need a specialized procedure for fitting ICNNs, but can use existing approaches that directly fit the ICNN. An example of this is the Q-learning setting. Given

some observed tuple (s, a, r, s'), Q learning updates the parameters θ with the gradient

$$\left(Q(s,a) - r - \gamma \max_{a'} Q(s',a')\right) \nabla_\theta Q(s,a), \quad (16)$$

where the maximization step is carried out with gradient descent or the bundle entropy method. These updates can be applied to ICNNs with the only additional requirement that we project the weights onto their feasible sets after this update (i.e., clip or project any W terms that are required to be positive). Section E gives a complete description of deep Q-learning with ICNNs.

Max-margin structured prediction. Although max-margin structured prediction is a simple and well-studied approach (Tsochantaridis et al., 2005; Taskar et al., 2005), in our experiences using these methods within an ICNN, we had substantial difficulty choosing the proper margin scaling term (especially for domains with continuous-valued outputs), or allowing for losses other than the hinge loss. For this reason, Section F discusses max-margin structured prediction in more detail, but the majority of our experiments here focus on the next approach, which more directly encodes the loss suffered by the full structured-prediction pipeline.

5.1. Argmin differentiation

In our final proposed approach, that of argmin differentiation, we propose to directly minimize a loss function between true outputs and the outputs predicted by our model, where these predictions themselves are the result of an optimization problem. We explicitly consider the case where the approximate solution to the inference problem is attained via the previously-described bundle entropy method, typically run for some fixed (usually small) number of iterations. To simplify notation, in the following we will let

$$\hat{y}(x;\theta) = \underset{y}{\arg\min} \min_t \ \{t - H(y) \mid Gy + h \leq t1\}$$
$$\approx \underset{y}{\arg\min} \ \tilde{f}(x,y;\theta) \quad (17)$$

refer to the *approximate* minimization over y that results from running the bundle entropy method, specifically at the last iteration of the method.

Given some example $(x, y^\star)$, our goal is to compute the gradient, with respect to the ICNN parameters, of the loss between $y^\star$ and $\hat{y}(x;\theta)$: $\ell(\hat{y}(x;\theta), y^\star)$. This is in some sense the most direct analogue to traditional neural network learning, since we typically optimize networks by minimizing some loss between the network's (feedforward) predictions and the true desired labels. Doing this in the predictions-via-optimization setting requires that we differentiate "through" the argmin operator, which can be ac-

complished via implicit differentiation of the KKT optimality conditions. Although the derivation is somewhat involved, the final result is fairly compact, and is given by the following proposition (for simplicity, we will write $\hat{y}$ below instead of $\hat{y}(x;\theta)$ when the notation should be clear):

Proposition 3. *The gradient of the neural network loss for predictions generated through the minimization process is*

$$\nabla_\theta \ell(\hat{y}(x;\theta), y^\star) = \sum_{i=1}^{k} (c_i^\lambda \nabla_\theta f(x, y^i; \theta) + \quad (18)$$
$$\nabla_\theta \left(\nabla_y f(x, y^i; \theta)^T \left(\lambda_i c^y + c_i^\lambda \left(\hat{y}(x;\theta) - y^i\right)\right)\right))$$

where y^i denotes the solution returned by the ith iteration of the entropy bundle method, λ denotes the dual variable solution of the entropy bundle method, and where the c variables are determined by the solution to the linear system

$$\begin{bmatrix} H & G^T & 0 \\ G & 0 & -1 \\ 0 & -1^T & 0 \end{bmatrix} \begin{bmatrix} c^y \\ c^\lambda \\ c^t \end{bmatrix} = \begin{bmatrix} -\nabla_{\hat{y}} \ell(\hat{y}, y^\star) \\ 0 \\ 0 \end{bmatrix}. \quad (19)$$

where $H = \text{diag}\left(\frac{1}{\hat{y}} + \frac{1}{1-\hat{y}}\right)$.

The proof of this proposition is given in Section G, but we highlight a few key points here. The complexity of computing this gradient will be linear in k, which is the number of *active* constraints at the solution of the bundle entropy method. The inverse of this matrix can also be computed efficiently by just inverting the $k \times k$ matrix $GH^{-1}G^T$ via a variable elimination procedure, instead of by inverting the full matrix. The gradients $\nabla_\theta f(x, y_i; \theta)$ are standard neural network gradients, and further, can be computed in the same forward/backward pass as we use to compute the gradients for the bundle entropy method. The main challenge of the method is to compute the terms of the form $\nabla_\theta(\nabla_y f(x, y_i; \theta)^T v)$ for some vector v. This quantity can be computed by most autodifferentiation tools (the gradient inner product $\nabla_y f(x, y_i; \theta)^T v$ itself just becomes a graph computation than can be differentiated itself), or it can be computed by a finite difference approximation. The complexity of computing this entire gradient is a small constant multiple of computing k gradients with respect to θ.

Given this ability to compute gradients with respect to an arbitrary loss function, we can fit the parameter using traditional stochastic gradient methods examples. Specifically, given an example (or a minibatch of examples) x_i, y_i, we compute gradients $\nabla_\theta \ell(\hat{y}(x_i;\theta), y_i)$ and update the parameters using e.g. the ADAM optimizer (Kingma & Ba, 2014).

Method	Test Macro-F1
Feedforward net	0.396
ICNN	0.415
SPEN (Belanger & McCallum, 2016)	**0.422**

Table 1. Comparison of approaches on BibTeX multi-label classification task. (Higher is better.)

6. Experiments

Our experiments study the representational power of ICNNs to better understand the interplay between the model's restrictiveness and accuracy. Specifically, we evaluate the method on multi-label classification on the BibTeX dataset (Katakis et al., 2008), image completion using the Olivetti face dataset (Samaria & Harter, 1994), and continuous action reinforcement learning in the OpenAI Gym (Brockman et al., 2016). We show that the methods compare favorably to the state of the art in many situations. The full source code for all experiments is available in the `icml2017` branch at `https://github.com/locuslab/icnn` and our implementation is built using Python (Van Rossum & Drake Jr, 1995) with the numpy (Oliphant, 2006) and TensorFlow (Abadi et al., 2016) packages.

6.1. Synthetic 2D example

Though we do not discuss it here, Section I presents a simple synthetic classification experiment comparing FICNN and PICNN decision boundaries.

6.2. Multi-Label Classification

We first study how ICNNs perform on multi-label classification with the BibTeX dataset and benchmark presented in (Katakis et al., 2008). This benchmark maps text classification from an input space $\mathcal{X}$ of 1836 bag-of-works indicator (binary) features to an output space $\mathcal{Y}$ of 159 binary labels. We use the train/test split of 4880/2515 from (Katakis et al., 2008) and evaluate with the macro-F1 score (higher is better). We use the ARFF version of this dataset from Mulan (Tsoumakas et al., 2011). Our PICNN architecture for multi-label classification uses fully-connected layers with ReLU activation functions and batch normalization (Ioffe & Szegedy, 2015) along the input path. As a baseline, we use a fully-connected neural network with batch normalization and ReLU activation functions. Both architectures have the same structure (600 fully connected, 159 (#labels) fully connected). We optimize our PICNN with 30 iterations of gradient descent with a learning rate of 0.1 and a momentum of 0.3.

Table 1 compares several different methods for this problem. Our PICNN's final macro-F1 score of 0.415 outper-

Figure 3. Example test set image completions of the ICNN with bundle entropy.

forms our baseline feedforward network's score of 0.396, which indicates PICNNs have the power to learn a robust structure over the output space. SPENs obtain a macro-F1 score of 0.422 on this task (Belanger & McCallum, 2016) and pose an interesting comparison point to ICNNs as they have a similar (but not identical) deep structure that is non-convex over the input space. The difference of 0.007 between ICNNs and SPENs could be due to differences in our experimental setups, architectures, and random experimental noise. More details are included in Section J.

6.3. Image completion on the Olivetti faces

As a test of the system on a structured prediction task over a much more complex output space $\mathcal{Y}$, we apply a convolutional PICNN to face completion on the sklearn version (Pedregosa et al., 2011) of the Olivetti data set (Samaria & Harter, 1994), which contains 400 64x64 grayscale images. ICNNs for face completion should be invariant to translations and other transformations in the input space. To achieve this invariance, our PICNN is inspired by the DQN architecture in Mnih et al. (2015), which preserves this invariance in the different context of reinforcement learning. Specifically, our network is over (x, y) pairs where x (32x64) is the left half and y (32x64) is the right half of the image. The input and output paths are: 32x8x8 conv (stride 4x2), 64x4x4 conv (stride 2x2), 64x3x3 conv, 512 fully connected.

This experiment uses the same training/test splits and minimizes the mean squared error (MSE) as in Poon & Domingos (2011) so that our results can be directly compared to (a non-exhaustive list of) other techniques. We also explore the tradeoffs between the bundle entropy method and gradient descent and use a non-convex baseline to better understand the impacts of convexity. We use a learning rate of 0.01 and momentum of 0.9 with gradient descent for the inner optimization in the ICNN.

Method	MSE
ICNN - Bundle Entropy	**833.0**
ICNN - Gradient Decent	872.0
ICNN - Nonconvex	850.9
Sum-product (Poon & Domingos, 2011)	942

Table 2. Comparisons of reconstruction error on image completion.

Task	DDPG	NAF	ICNN
Ant	1000.00	999.03	**1056.29**
HalfCheetah	2909.77	2575.16	**3822.99**
Hopper	**1501.33**	1100.43	831.00
Humanoid	524.09	**5000.68**	433.38
HumanoidStandup	134265.96	116399.05	**141217.38**
InvDoubPend	**9358.81**	9359.59	9359.41
InvPend	**1000.00**	**1000.00**	**1000.00**
Reacher	-6.10	-6.31	**-5.08**
Swimmer	49.79	**69.71**	64.89
Walker2d	**1604.18**	1007.25	298.21

Table 3. Maximum test reward for ICNN algorithm versus alternatives on several OpenAI Gym tasks. (All tasks are v1.)

Table 2 shows the test MSEs for the different approaches. Example image completions are shown in Figure 3. These results show that the bundle entropy method can leverage more information from these five iterations than gradient descent, even when the convexity constraint is relaxed. The PICNN trained with back-optimization with the relaxed convexity constraint slightly outperforms the network with the convexity constraint, but not the network trained with the bundle-entropy method. This shows that for image completion with PICNNs, convexity does not seem to inhibit the representational power. Furthermore, this experiment suggests that a small number of inner optimization iterations (five in this case) is sufficient for good performance.

6.4. Continuous Action Reinforcement Learning

Finally, we present standard benchmarks in continuous action reinforcement learning from the OpenAI Gym (Brockman et al., 2016) that use the MuJoCo physics simulator (Todorov et al., 2012). We model the (negative) Q function, $-Q(s, a; \theta)$ as an ICNN and select actions with the convex optimization problem $a^{\star}(s) = \mathrm{argmin}_a -Q(s, a; \theta)$. We use Q-learning to optimize the ICNN as described in Section 5 and Section E. At test time, the policy is selected by optimizing $Q(s, a; \theta)$. All of our experiments use a PICNN with two fully-connected layers that each have 200 hidden units. We compare to Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) and Normalized Advantage Functions (NAF) (Gu et al., 2016) as state-of-the-art off-policy learning baselines.[4]

Table 3 shows the maximum test reward achieved by the different algorithms on these tasks. Although no method strictly dominates the others, the ICNN approach has some clear advantages on tasks like HalfCheetah, Reacher, and HumanoidStandup, and performs comparably on many other tasks, though with also a few notable poor performances in Hopper and Walker2D. Nonetheless, given the strong baseline, and the fact that the method is literally just a drop-in replacement for a function approximator in

Q-learning, these results are overall positive. NAF poses a particularly interesting comparison point to ICNNs. In particular, NAF decomposes the Q function in terms of the value function an an advantage function $Q(s, a) = V(s) + A(s, a)$ where the advantage function is restricted to be *concave quadratic* in the actions, and thus always has a closed-form solution. In a sense, this closely mirrors the setup of the PICNN architecture: like NAF, we have a separate non-convex path for the s variables, and an overall function that is convex in a; however, the distinction is that while NAF requires that the convex portion be quadratic, the ICNN architecture allows any convex functional form. As our experiments show, this representational power does allow for better performance of the resulting system, though the trade-off, of course, is that determining the optimal action in an ICNN is substantially more computationally complex than for a quadratic.

7. Conclusion and future work

This paper laid the groundwork for the input convex neural network model. By incorporating relatively simple constraints into existing network architectures, we can fit very general convex functions and the apply optimization as an inference procedure. Since many existing models already fit into this overall framework (e.g., CRF models perform an optimization over an output space where parameters are given by the output of a neural network), the proposed method presents an extension where the entire inference procedure is "learned" along with the network itself, without the need for explicitly building typical structured prediction architectures. This work explored only a small subset of the possible applications of these network, and the networks offer promising directions for many additional domains.

---

[4]Because there are not official DDPG or NAF implementations or results on the OpenAI gym tasks, we use the Simon Ramstedt's DDPG implementation from `https://github.com/SimonRamstedt/ddpg` and have re-implemented NAF.

Acknowledgments

References

Abadi, Martın, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Barzilai, Jonathan and Borwein, Jonathan M. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.

Belanger, David and McCallum, Andrew. Structured prediction energy networks. In *Proceedings of the International Conference on Machine Learning*, 2016.

Bengio, Yoshua, LeCun, Yann, and Henderson, Donnie. Globally trained handwritten word recognizer using spatial representation, convolutional neural networks, and hidden markov models. *Advances in neural information processing systems*, pp. 937–937, 1994.

Bertsekas, Dimitri P. Projected newton methods for optimization problems with simple constraints. *SIAM Journal on control and Optimization*, 20(2):221–246, 1982.

Birgin, Ernesto G, Martínez, José Mario, and Raydan, Marcos. Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization*, 10(4): 1196–1211, 2000.

Boyd, Stephen and Vandenberghe, Lieven. *Convex optimization*. Cambridge university press, 2004.

Boyd, Stephen, Parikh, Neal, Chu, Eric, Peleato, Borja, and Eckstein, Jonathan. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, and Zaremba, Wojciech. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Chen, Liang-Chieh, Schwing, Alexander G, Yuille, Alan L, and Urtasun, Raquel. Learning deep structured models. In *Proceedings of the International Conference on Machine Learning*, 2015.

Domke, Justin. Generic methods for optimization-based modeling. In *Proceedings of the Conference on AI and Statistics*, pp. 318–326, 2012.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

Gu, Shixiang, Lillicrap, Timothy, Sutskever, Ilya, and Levine, Sergey. Continuous deep q-learning with model-based acceleration. In *Proceedings of the International Conference on Machine Learning*, 2016.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Huang, Gao, Liu, Zhuang, and Weinberger, Kilian Q. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pp. 448–456, 2015.

Katakis, Ioannis, Tsoumakas, Grigorios, and Vlahavas, Ioannis. Multilabel text classification for automated tag suggestion. *ECML PKDD discovery challenge*, 75, 2008.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Koller, Daphne and Friedman, Nir. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

LeCun, Yann, Chopra, Sumit, Hadsell, Raia, Ranzato, M, and Huang, F. A tutorial on energy-based learning. *Predicting structured data*, 1:0, 2006.

Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Magnani, Alessandro and Boyd, Stephen P. Convex piecewise-linear fitting. *Optimization and Engineering*, 10(1):1–17, 2009.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814, 2010.

Oliphant, Travis E. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

Pedregosa, Fabian, Varoquaux, Gaël, Gramfort, Alexandre, Michel, Vincent, Thirion, Bertrand, Grisel, Olivier, Blondel, Mathieu, Prettenhofer, Peter, Weiss, Ron, Dubourg, Vincent, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

Peng, Jian, Bo, Liefeng, and Xu, Jinbo. Conditional neural fields. In *Advances in neural information processing systems*, pp. 1419–1427, 2009.

Polyak, Boris T. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Poon, Hoifung and Domingos, Pedro. Sum-product networks: A new deep architecture. In *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, pp. 337–346, 2011.

Ratliff, Nathan D, Bagnell, J Andrew, and Zinkevich, Martin. (Approximate) subgradient methods for structured prediction. In *International Conference on Artificial Intelligence and Statistics*, pp. 380–387, 2007.

Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Samaria, Ferdinando S and Harter, Andy C. Parameterisation of a stochastic model for human face identification. In *Applications of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on*, pp. 138–142. IEEE, 1994.

Simard, Patrice and LeCun, Yann. Reverse tdnn: an architecture for trajectory generation. In *Advances in Neural Information Processing Systems*, pp. 579–588. Citeseer, 1991.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Smola, Alex J., Vishwanathan, S.v.n., and Le, Quoc V. Bundle methods for machine learning. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T. (eds.), *Advances in Neural Information Processing Systems 20*, pp. 1377–1384. Curran Associates, Inc., 2008.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

Taskar, Ben, Chatalbashev, Vassil, Koller, Daphne, and Guestrin, Carlos. Learning structured prediction models: A large margin approach. In *Proceedings of the 22nd International Conference on Machine Learning*, pp. 896–903. ACM, 2005.

Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Tsochantaridis, Ioannis, Joachims, Thorsten, Hofmann, Thomas, and Altun, Yasemin. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

Tsoumakas, Grigorios, Spyromitros-Xioufis, Eleftherios, Vilcek, Jozef, and Vlahavas, Ioannis. Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, 12(Jul):2411–2414, 2011.

Van Rossum, Guido and Drake Jr, Fred L. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

Wright, Stephen J. *Primal-dual interior-point methods*. Siam, 1997.

Zheng, Shuai, Jayasumana, Sadeep, Romera-Paredes, Bernardino, Vineet, Vibhav, Su, Zhizhong, Du, Dalong, Huang, Chang, and Torr, Philip HS. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1529–1537, 2015.

Input Convex Neural Networks: Supplementary Material

Brandon Amos   Lei Xu   J. Zico Kolter

A. Additional architectures

A.1. Convolutional architectures

Convolutions are important to many visual structured tasks. We have left convolutions out to keep the prior ICNN notation light by using matrix-vector operations. ICNNs can be similarly created with convolutions because the convolution is a linear operator.

The construction of convolutional layers in ICNNs depends on the type of input and output space. If the input and output space are similarly structured (e.g. both spatial), the jth feature map of a convolutional PICNN layer i can be defined by

$$z_{i+1}^j = g_i \left(z_i * W_{i,j}^{(z)} + (Sx) * W_{i,j}^{(x)} + (Sy) * W_{i,j}^{(y)} + b_{i,j} \right) \tag{20}$$

where the convolution kernels W are the same size and S scales the input and output to be the same size as the previous feature map, and were we omit some of the Hadamard product terms that can appear above for simplicity of presentation.

If the input space is spatial, but the output space has another structure (e.g. the simplex), the convolution over the output space can be replaced by a matrix-vector operation, such as

$$z_{i+1}^j = g_i \left(z_i * W_{i,j}^{(z)} + (Sx) * W_{i,j}^{(x)} + B_{i,j}^{(y)} y + b_{i,j} \right) \tag{21}$$

where the product $B_{i,j}^{(y)} y$ is a scalar.

B. Exact inference in ICNNs

Although it is not a practical approach for solving the optimization tasks, we first highlight the fact that the inference problem for the networks presented above (where the nonlinear are either ReLU or linear units) can be posed as as linear program. Specifically, considering the FICNN network in (2) can be written as the optimization problem

$$\begin{aligned}
\underset{y, z_1, \dots, z_k}{\text{minimize}} \quad & z_k \\
\text{subject to} \quad & z_{i+1} \geq W_i^{(z)} z_i + W_i^{(y)} y + b_i, \quad i = 0, \dots, k-1 \\
& z_i \geq 0, \quad i = 1, \dots, k-1.
\end{aligned} \tag{22}$$

This problem exactly replicates the equations of the FICNN, with the exception that we have replaced ReLU and the equality constraint between layers with a positivity constraint on the z_i terms and an inequality. However, because we are minimizing the final z_k term, and because each inequality constraint is convex, at the solution one of these constraints must be tight, i.e., $(z_i)_j = (W_i^{(z)} z_i + W_i^{(y)} y + b_i)_j$ or $(z_i)_j = 0$, which recovers the ReLU non-linearity exactly. The exact same procedure can be used to write to create an exact inference procedure for the PICNN.

Although the LP formulation is appealing in its simplicity, in practice these optimization problems will have a number of variables equal to the *total* number of activations in the entire network. Furthermore, most LP solution methods to solve such problems require that we form *and invert* structured matrices with blocks such as $W_i^T W_i$ — the case for most interior-point methods (Wright, 1997) or even approximate algorithms such as the alternating direction method of multipliers (Boyd et al., 2011) — which are large dense matrices or have structured forms such as non-cyclic convolutions that are expensive to invert. Even incremental approaches like the Simplex method require that we form inverses of subsets of columns of these matrices, which are additionally different for structured operations like convolutions, and which overall still involve substantially more computation than a single forward pass. Furthermore, such solvers typically do not exploit the substantial effort that has gone in to accelerating the forward and backward computation passes for neural networks using hardware such as GPUs. Thus, as a whole, these do not present a viable option for optimizing the networks.

Algorithm 1 A typical bundle method to optimize $f : \mathbb{R}^{m \times n} \to \mathbb{R}$ over $\mathbb{R}^n$ for K iterations with a fixed x and initial starting point y^1.

---

function BUNDLEMETHOD(f, x, y^1, K)
  $G \leftarrow 0 \in \mathbb{R}^{K \times n}$
  $h \leftarrow 0 \in \mathbb{R}^K$
  for $k = 1, K$ **do**
    $G_k^T \leftarrow \nabla_y f(x, y^k; \theta)^T$ $\quad \triangleright k$th row of G
    $h_k \leftarrow f(x, y^k; \theta) - \nabla_y f(x, y^k; \theta)^T y^k$
    $y^{k+1}, t^{k+1} \leftarrow \operatorname{argmin}_{y \in \mathcal{Y}, t} \{t \mid G_{1:k} y + h_{1:k} \leq t1\}$
  end for
  return y^{K+1}
end function

C. The bundle method for approximate inference in ICNNs

We here review the basic bundle method (Smola et al., 2008) that we build upon in our bundle entropy method. The bundle method takes advantage of the fact that for a convex objective, the first-order approximation at any point is a global *underestimator* of the function; this lets us maintain a piecewise linear lower bound on the function by adding cutting planes formed by this first order approximation, and then repeatedly optimizing this lower bound. Specifically, the process follows the procedure shown in Algorithm 1. Denoting the iterates of the algorithm as y^k, at each iteration of the algorithm, we compute the first order approximation to the function

$$f(x, y^k; \theta) + \nabla_y f(x, y^k; \theta)^T (y - y^k) \quad (23)$$

and update the next iteration by solving the optimization problem

$$y^{k+1} := \operatorname*{argmin}_{y \in \mathcal{Y}} \max_{1 \leq i \leq k} \{f(x, y^i; \theta) + \nabla_y f(x, y^i; \theta)^T (y - y^i)\}. \quad (24)$$

A bit more concretely, the optimization problem can be written via a set of linear inequality constraints

$$y^{k+1}, t^{k+1} := \operatorname*{argmin}_{y \in \mathcal{Y}, t} \{t \mid Gy + h \leq t1\} \quad (25)$$

where $G \in \mathbb{R}^{k \times n}$ has rows equal to

$$g_i^T = \nabla_y f(x, y^i; \theta)^T \quad (26)$$

and $h \in \mathbb{R}^k$ has entries equal to

$$h_i = f(x, y^i; \theta) - \nabla_y f(x, y^i; \theta)^T y^i. \quad (27)$$

D. Bundle Entropy Algorithm

In Algorithm 2.

Algorithm 2 Our bundle entropy method to optimize $f : \mathbb{R}^m \times [0, 1]^n \to \mathbb{R}$ over $[0, 1]^n$ for K iterations with a fixed x and initial starting point y^1.

---

function BUNDLEENTROPYMETHOD(f, x, y^1, K)
  $G_\ell \leftarrow [\,]$
  $h_\ell \leftarrow [\,]$
  for $k = 1, K$ **do**
    APPEND($G_\ell, \nabla_y f(x, y^k; \theta)^T$)
    APPEND($h_\ell, f(x, y^k; \theta) - \nabla_y f(x, y^k; \theta)^T y^k$)
    $a_k \leftarrow$ LENGTH(G_ℓ) $\quad \triangleright$ The number of active constraints.
    $G_k \leftarrow$ CONCAT(G_ℓ)$\in \mathbb{R}^{a_k \times n}$
    $h_k \leftarrow$ CONCAT(h_ℓ)$\in \mathbb{R}^{a_k}$
    if $a_k = 1$ **then**
      $\lambda_k \leftarrow 1$
    else
      $\lambda_k \leftarrow$ PROJNEWTONLOGISTIC(G_k, h_k)
    end if
    $y^{k+1} \leftarrow (1 + \exp(G_k^T \lambda_k))^{-1}$
    DELETE($G_\ell[i]$ and $h_\ell[i]$ where $\lambda_i \leq 0$) $\quad \triangleright$ Prune inactive constraints.
  end for
  return y^{K+1}
end function

E. Deep Q-learning with ICNNs

In Algorithm 3.

Algorithm 3 Deep Q-learning with ICNNs. `Opt-Alg` is a convex minimization algorithm such as gradient descent or the bundle entropy method. $\tilde{Q}_\theta$ is the objective the optimization algorithm solves. In gradient descent, $\tilde{Q}_\theta(s,a) = Q(s,a|\theta)$ and with the bundle entropy method, $\tilde{Q}_\theta(s,a) = Q(s,a|\theta) + H(a)$.

Select a discount factor $\gamma \in (0,1)$ and moving average factor $\tau \in (0,1)$
Initialize the ICNN $-Q(s,a|\theta)$ with target network parameters $\theta' \leftarrow \theta$ and a replay buffer $R \leftarrow \emptyset$
for each episode $e = 1, E$ **do**
 Initialize a random process $\mathcal{N}$ for action exploration
 Receive initial observation state s_1
 for $i = 1, I$ **do**
  $a_i \leftarrow$ Opt-Alg$(-Q_\theta, s_i, a_{i,0})+\mathcal{N}_i$ ▷ For some initial action $a_{i,0}$
  Execute a_i and observe r_{i+1} and s_{i+1}
  Insert$(R, (s_i, a_i, s_{i+1}, r_{i+1}))$
  Sample a random minibatch from the replay buffer: $R_M \subseteq R$
  for $(s_m, a_m, s_m^+, r_m^+) \in R_M$ **do**
   $a_m^+ \leftarrow$ Opt-Alg$(-Q_{\theta'}, s_m^+, a_{m,0}^+)$ ▷ Uses the target parameters θ'
   $y_m \leftarrow r_m^+ + \gamma Q(s_m^+, a_m^+|\theta')$
  end for
  Update θ with a gradient step to minimize $\mathcal{L} = \frac{1}{|R_M|}\sum_m \left(\tilde{Q}(s_m, a_m|\theta) - y_m\right)^2$
  $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ ▷ Update the target network.
 end for
end for

F. Max-margin structured prediction

In the more traditional structured prediction setting, where we do not aim to fit the energy function directly but fit the predictions made by the system to some target outputs, there are different possibilities for learning the ICNN parameters. One such method is based upon the max-margin structured prediction framework (Tsochantaridis et al., 2005; Taskar et al., 2005). Given some training example $(x, y^\star)$, we would like to require that this example has a joint energy that is lower than all other possible values for y. That is, we want the function $\tilde{f}$ to satisfy the constraint

$$\tilde{f}(x, y^\star; \theta) \le \min_y \tilde{f}(x, y; \theta) \qquad (28)$$

Unfortunately, these conditions can be trivially fit by choosing a constant $\tilde{f}$ (although the entropy term alleviates this problem slightly, we can still choose an approximately constant function), so instead the max-margin approach adds a margin-scaling term that requires this gap to be larger for y further from $y^\star$, as measured by some loss

function $\Delta(y, y^\star)$. Additionally adding slack variables to allow for potential violation of these constraints, we arrive at the typical max-margin structured prediction optimization problem

$$\underset{\theta, \xi \ge 0}{\text{minimize}} \ \ \frac{\lambda}{2}\|\theta\|_2^2 + \sum_{i=1}^m \xi_i$$

$$\text{subject to} \ \ \tilde{f}(x_i, y_i; \theta) \le \min_{y \in \mathcal{Y}}\left(\tilde{f}(x_i, y; \theta) - \Delta(y_i, y)\right) - \xi_i$$

$$(29)$$

As a simple example, for multiclass classification tasks where $y^\star$ denotes a "one-hot" encoding of examples, we can use a multi-variate entropy term and let $\Delta(y, y^\star) = y^{\star T}(1 - y)$. Training requires solving this "loss-augmented" inference problem, which is convex for suitable choices of the margin scaling term.

The optimization problem (29) is naturally still *not convex* in θ, but can be solved via the subgradient method for structured prediction (Ratliff et al., 2007). This algorithm iteratively selects a training example x_i, y_i, then 1) solves the optimization problem

$$y^\star = \underset{y \in \mathcal{Y}}{\text{argmin}} \ f(x_i, y; \theta) - \Delta(y_i, y) \qquad (30)$$

and 2) if the margin is violated, updates the network's parameters according to the subgradient

$$\theta := \mathcal{P}_+\left[\theta - \alpha\left(\lambda\theta + \nabla_\theta f(x_i, y_i, \theta) - \nabla_\theta f(x_i, y^\star; \theta)\right)\right]$$

$$(31)$$

where $\mathcal{P}_+$ denotes the projection of $W_{1:k-1}^{(z)}$ onto the nonnegative orthant. This method can be easily adapted to use mini-batches instead of a single example per subgradient step, and also adapted to alternative optimization methods like AdaGrad (Duchi et al., 2011) or ADAM (Kingma & Ba, 2014). Further, a fast approximate solution to $y^\star$ can be used instead of the exact solution.

G. Proof of Proposition 3

Proof (of Proposition 3). We have by the chain rule that

$$\frac{\partial\ell}{\partial\theta} = \frac{\partial\ell}{\partial\hat{y}}\left(\frac{\partial\hat{y}}{\partial G}\frac{\partial G}{\partial\theta} + \frac{\partial\hat{y}}{\partial h}\frac{\partial h}{\partial\theta}\right). \qquad (32)$$

The challenging terms to compute in this equation are the $\frac{\partial\hat{y}}{\partial G}$ and $\frac{\partial\hat{y}}{\partial h}$ terms. These can be computed (although we will ultimately not compute them explicitly, but just compute the product of these matrices and other terms in the Jacobian), by implicit differentiation of the KKT conditions. Specifically, the KKT conditions of the bundle entropy method (considering only the active constraints at the

solution) are given by

$$1 + \log \hat{y} - \log(1 - \hat{y}) + G^T \lambda = 0$$
$$G\hat{y} + h - t1 = 0 \tag{33}$$
$$1^T \lambda = 1.$$

For simplicity of presentation, we consider first the Jacobian with respect to h. Taking differentials of these equations with respect to h gives

$$\text{diag}\left(\frac{1}{\hat{y}} + \frac{1}{1-\hat{y}}\right) \mathsf{d}y + G^T \mathsf{d}\lambda = 0$$
$$G\mathsf{d}y + \mathsf{d}h - \mathsf{d}t1 = 0 \tag{34}$$
$$1^T \mathsf{d}\lambda = 0$$

or in matrix form

$$\begin{bmatrix} \text{diag}\left(\frac{1}{\hat{y}} + \frac{1}{1-\hat{y}}\right) & G^T & 0 \\ G & 0 & -1 \\ 0 & -1^T & 0 \end{bmatrix} \begin{bmatrix} \mathsf{d}y \\ \mathsf{d}\lambda \\ \mathsf{d}t \end{bmatrix} = \begin{bmatrix} 0 \\ -\mathsf{d}h \\ 0 \end{bmatrix}. \tag{35}$$

To compute the Jacobian $\frac{\partial \hat{y}}{\partial h}$ we can solve the system above with the right hand side given by $\mathsf{d}h = I$, and the resulting $\mathsf{d}y$ term will be the corresponding Jacobian. However, in our ultimate objective we always left-multiply the proper terms in the above equation by $\frac{\partial \ell}{\partial \hat{y}}$. Thus, we instead define

$$\begin{bmatrix} c^y \\ c^\lambda \\ c^t \end{bmatrix} = \begin{bmatrix} \text{diag}\left(\frac{1}{\hat{y}} + \frac{1}{1-\hat{y}}\right) & G^T & 0 \\ G & 0 & -1 \\ 0 & -1^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} -(\frac{\partial \ell}{\partial \hat{y}})^T \\ 0 \\ 0 \end{bmatrix} \tag{36}$$

and we have the the simple formula for the Jacobian product

$$\frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} = (c^\lambda)^T. \tag{37}$$

A similar set of operations taking differentials with respect to G leads to the matrix equations

$$\begin{bmatrix} \text{diag}\left(\frac{1}{\hat{y}} + \frac{1}{1-\hat{y}}\right) & G^T & 0 \\ G & 0 & -1 \\ 0 & -1^T & 0 \end{bmatrix} \begin{bmatrix} \mathsf{d}y \\ \mathsf{d}\lambda \\ \mathsf{d}t \end{bmatrix} = \begin{bmatrix} -\mathsf{d}G^T \lambda \\ -\mathsf{d}Gy \\ 0 \end{bmatrix} \tag{38}$$

and the corresponding Jacobian products / gradients are given by

$$\frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial G} = c^y \lambda^T + \hat{y}(c^\lambda)^T. \tag{39}$$

Finally, using the definitions that

$$g_i^T = \nabla_y f(x, y^i; \theta)^T, \quad h_i = f(x, y^k; \theta) - \nabla_y f(x, y^i; \theta)^T y^i \tag{40}$$

we recover the formula presented in the proposition. $\quad\square$

H. State and action space sizes in the OpenAI gym MuJoCo benchmarks.

Environment	# State	# Action
InvertedPendulum-v1	4	1
InvertedDoublePendulum-v1	11	1
Reacher-v1	11	2
HalfCheetah-v1	17	6
Swimmer-v1	8	2
Hopper-v1	11	3
Walker2d-v1	17	6
Ant-v1	111	8
Humanoid-v1	376	17
HumanoidStandup-v1	376	17

Table 4. State and action space sizes in the OpenAI gym MuJoCo benchmarks.

I. Synthetic classification examples

We begin with a simple example to illustrate the classification performance of a two-hidden-layer FICNN and PICNN on two-dimensional binary classification tasks from the scikit-learn toolkit (Pedregosa et al., 2011). Figure 4 shows the classification performance on the dataset. The FICNN's energy function which is fully convex in $\mathcal{X} \times \mathcal{Y}$ jointly is able to capture complex, but sometimes restrictive decision boundaries. The PICNN, which is non-convex over $\mathcal{X}$ but convex over $\mathcal{Y}$ overcomes these restrictions and can capture more complex decision boundaries.

J. Multi-Label Classification Training Plots

In Figure 5.

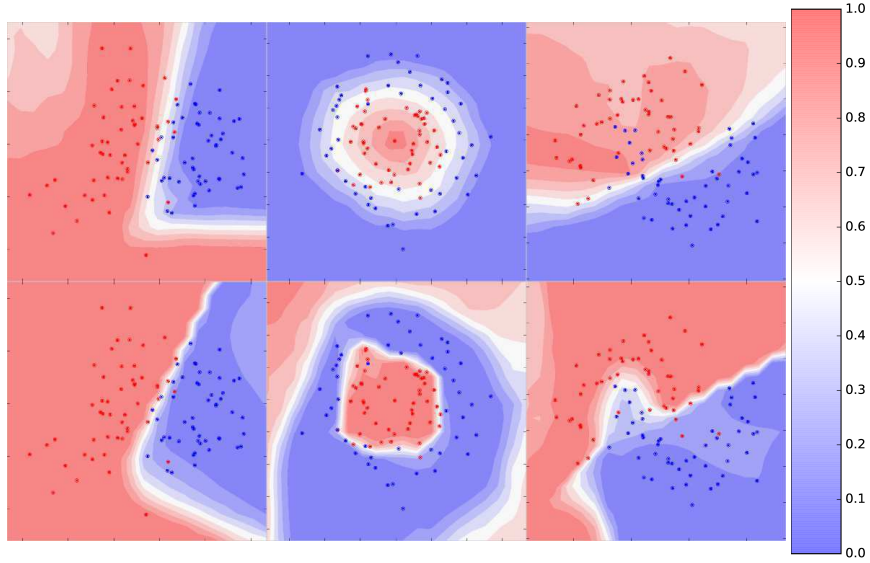## K. Image Completion

The losses are in Figure 6.

Figure 4. FICNN (top) and PICNN (bottom) classification of synthetic non-convex decision boundaries. Best viewed in color.
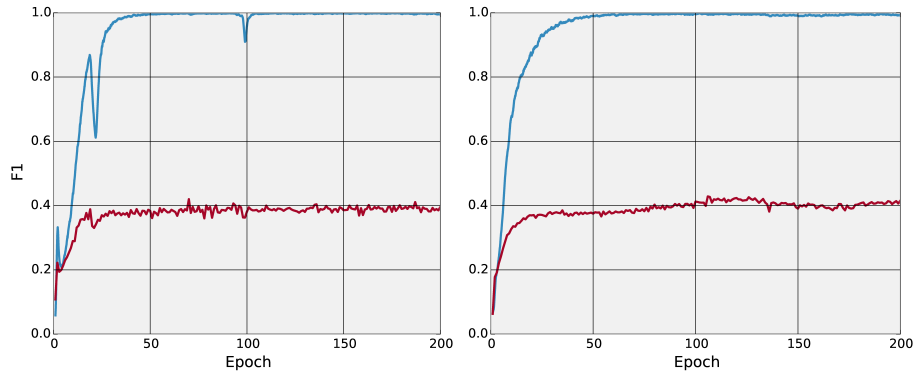

Figure 5. Training (blue) and test (red) macro-F1 score of a feedforward network (left) and PICNN (right) on the BibTeX multi-label classification dataset. The final test F1 scores are 0.396 and 0.415, respectively. (Higher is better.)
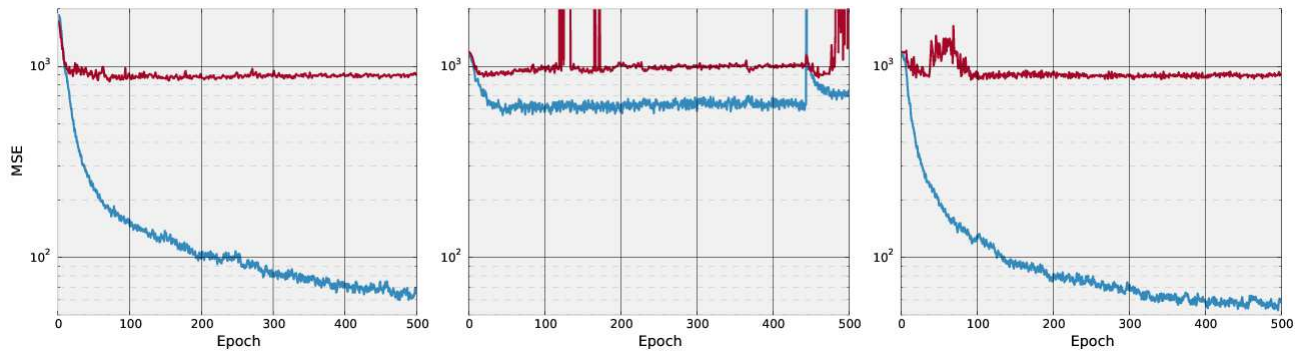

Figure 6. Mean Squared Error (MSE) on the train (blue, rolling over 1 epoch) and test (red) images from Olivetti faces for PICNNs trained with the bundle entropy method (left) and back optimization (center), and back optimization with the convexity constraint relaxed (right). The minimum test MSEs are 833.0, 872.0, and 850.9, respectively.