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

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

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**
    $G_k^T \leftarrow \nabla_y f(x, y^k; \theta)^T$ ▷ kth 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

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

and update the next iteration by solving the optimization problem

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

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

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

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

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

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

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

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_ℓ, $\nabla_y f(x, y^k; \theta)^T$)
    APPEND(h_ℓ, $f(x, y^k; \theta) - \nabla_y f(x, y^k; \theta)^T y^k$)
    $a_k \leftarrow$ LENGTH(G_ℓ) ▷ 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 $\lambda_i \leq 0$) ▷ 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

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

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

$$\begin{aligned} &\underset{\theta,\xi\geq 0}{\text{minimize}} \;\; \frac{\lambda}{2}\|\theta\|_2^2 + \sum_{i=1}^m \xi_i \\ &\text{subject to} \;\; \tilde{f}(x_i, y_i; \theta) \leq \min_{y\in\mathcal{Y}} \left(\tilde{f}(x_i, y; \theta) - \Delta(y_i, y)\right) - \xi_i \end{aligned} \tag{29}$$

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

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

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

$$\theta := \mathcal{P}_+\left[\theta - \alpha\left(\lambda\theta + \nabla_\theta f(x_i, y_i, \theta) - \nabla_\theta f(x_i, y^\star; \theta)\right)\right] \tag{31}$$

where $\mathcal{P}_+$ denotes the projection of $W_{1:k-1}^{(z)}$ onto the non-negative 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

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

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

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

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

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

or in matrix form

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

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

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

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

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

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. □

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.

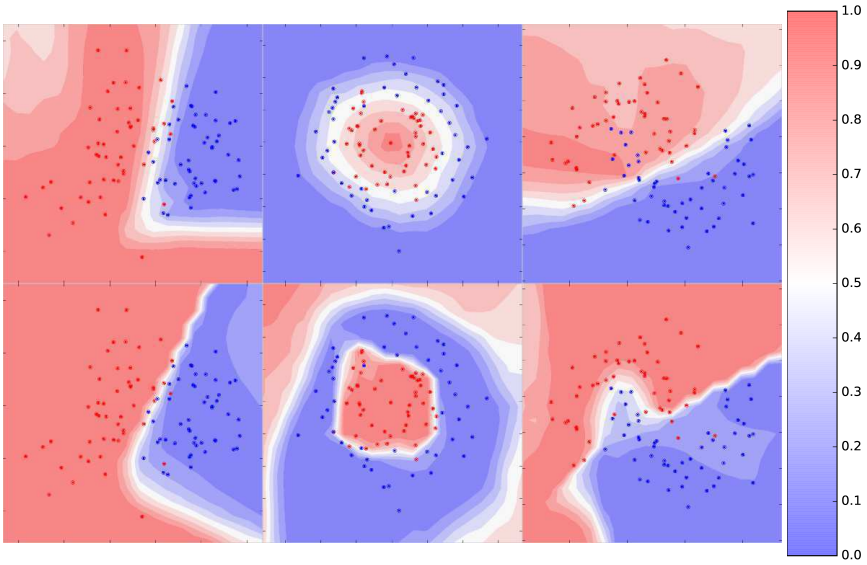

Figure 4. FICNN (top) and PICNN (bottom) classification of synthetic non-convex decision boundaries. Best viewed in color.

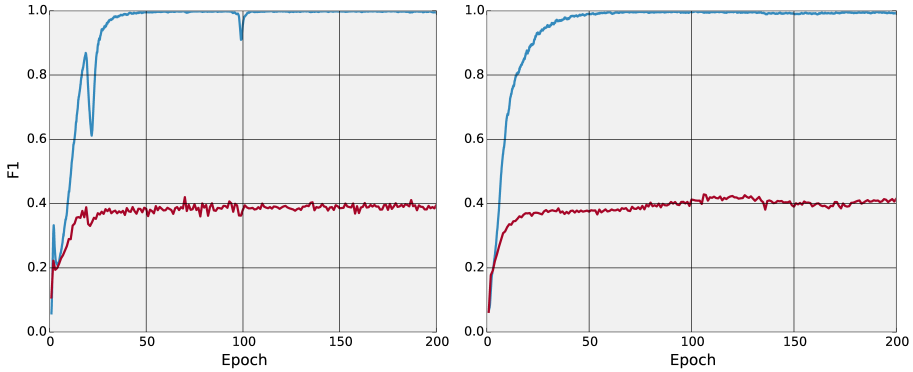

Figure 5. Training (blue) and test (red) macro-F1 score of a feedforward network (left) and PICNN (right) on the BibTeX multi-label classification dataset. The final test F1 scores are 0.396 and 0.415, respectively. (Higher is better.)

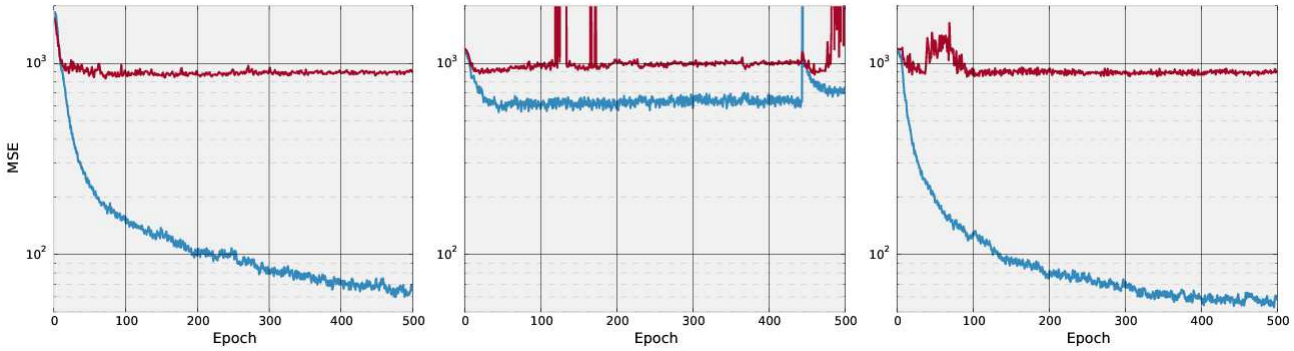

Figure 6. Mean Squared Error (MSE) on the train (blue, rolling over 1 epoch) and test (red) images from Olivetti faces for PICNNs trained with the bundle entropy method (left) and back optimization (center), and back optimization with the convexity constraint relaxed (right). The minimum test MSEs are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