An Investigation of Input Convex Neural Networks

Valerie Chen

December 2019

1 Introduction

1.1 Motivation

Deep learning has taken the AI research community by storm. This technique is being used in various decision-making models from those powering self-driving cars to others found in robotic systems. In most cases, we care about learning a model that maximizes accuracy of the result, by minimizing the loss to training examples. When training a model, it is difficult to tell in the typically non-convex landscape how to navigate and escape local minima solutions. Particularly in deep neural networks, it is not clear if the training will be able to overcome the local solution and find a better one.

On the other hand, convex optimization techniques are able to provide guarantees for global minimums. A recent paper from Amos et al. [1] proposes the formulation of an input convex neural network (ICNN) which bridges deep networks and optimization by incorporating simple constraints into existing network architectures. The goal of ICNNs is to allow for optimization in the inference phase, rather than a pure feed forward prediction. The paper demonstrates improvement over existing state-of-the-art results on various tasks including prediction, face-image completion, and reinforcement learning.

1.2 Paper Outline

In this project, I conduct a deep-dive into ICNNs. The paper is divided into two sections: a more theoretical focused section and an experimental section. Section 2 covers a walk-through which provides more details on how ICNNs work. Note that I omit some of the proofs in the paper that are more involved, but highlight the key points necessary to understand the implementation in the next section. Section 3 presents experimental results of applying the ICNN to synthetic data and comparing the results to feedforward networks. My goal of this project is to get a thorough understanding of how convex optimization can be combined with deep neural networks and what benefits and trade-offs can be achieved when optimization is used.

2 How do ICNNs work?

2.1 Background

A large portion of the work on ICNNs is based on energy based learning. A good reference the ICNN paper cited is LeCun et al. [4]. I present a summary overview that is helpful for the remainder of the ICNN paper. Energy based models capture dependencies by associating a scalar energy $f(x, y; \theta)$ to each configuration of the variables. The model is characterized by x, the observed variables, y, the predicted variables, parameterized by the weights θ. Compared to feedforward networks, energy based models directly parameterize the energy function which gives practitioners more opportunities to incorporate domain knowledge. This type of set-up could lead to a more parsimonious model that is still generalizable. However, a draw-back to energy based models is that learning and prediction can be more complicated to implement and calculate in energy based models [2].

Training an energy based model consists of finding the energy function that associates lower energies to the correct values of the prediction variables and higher energies to incorrect values of the prediction variables. So the input variables x are passed through the network with weights θ and then backpropagated through a loss against y to get the output $f(x, y; \theta)$. The training essentially aims to solve this optimization problem:

$$\theta^* = \arg\min_{\theta \in \Theta} \mathcal{L}(x, y, \theta)$$

where $\mathcal{L}$ is the loss functional and Θ the set of all parameters. In most cases, according to [4], the loss functional is defined as:

$$\mathcal{L}(x, y, \theta) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i, \hat{y}, \theta)) + \mathcal{R}(\theta)$$

where $\mathcal{R}$ is the regularizer, L the loss function, and $\hat{y}$ is the minimizing set of non-fixed variables. The loss functional is computed on a per-sample loss basis.

Inference consists of fixing a set of variables and minimizing the energy by finding the values of the non-fixed, prediction variables. Given a fixed set of learned parameters θ and input variables x, the inference optimization problem can be formulated as such:

$$\arg\min_{y} f(x, y; \theta)$$

Instead of a typical feedforward prediction, inference is instead framed as an optimization problem on the network. Feedforward networks have been widely demonstrated to do well on a variety of tasks, so it is interesting to see a different approach used. To give a little insight on where optimizing over the set of

outputs is useful, the authors [1] give some potential useful examples:

1. **Structured Prediction:** This is an example also provided by [4] where the problem is framed as finding the y that is most compatible with a given x in any prediction, classification, or decision-making problem.

2. **Data Imputation:** In this case, the entire input space is captured by y, without x. A subset of the missing input y_m can be imputed by defining the energy function as $f(y, y_m; \theta)$.

3. **Q-Learning:** The energy based model can be used to model the Q-function. However, in this case we would $f(s, a; \theta) = -Q(s, a; \theta)$ so f is convex.

2.2 The ICNN Model

Building on top of these preliminaries from energy based models, I present the formulation of the input convex neural network. The authors emphasize the term "input convex" since convexity often refers to convexity of parameters in optimization problems, whereas only a subset of input are convex in this model. Enforcing convexity in a subset of inputs allows for energy based models to leverage the use of optimization solvers in the inference phase.

First, a formulation of a fully-input convex network presented in the paper, with k-layers:

$$z_{i+1} = g_i(W_i^{(z)} z_i + W_i^{(y)} y + b_i)$$

where $z_0 = 0$, $W_0^{(z)} = 0$, the final output of the energy function $f(y; \theta) = z_k$, the final output of the k-th layer and g_i a non-linear activation function. With regards to convexity, we want to enforce that this above ICNN formulation is convex with respect to y, the prediction space. Below is a proposition the authors of ICNN mention in their paper but do not prove.

Theorem 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.*

Proof. A given constraint is that all g_i are convex and non-decreasing. In deep neural network training, there are many commonly used activation functions with the desired features of convexity and non-decreasing behavior. One such example is the ReLU function which takes the maximum between the value and 0: $ReLU(x) = max(0, x)$. The main restriction is that $W_{1:k-1}^{(z)}$ are non-negative. Let $z_{i+1}(y) = z_{i+1}$ for a given y. The following proof follows Section 3.2.4 of Boyd and Vandenberghe [3]. For any layer in the network, let

$y_1, y_2 = y$.

$$z_{i+1}(y) = g_i(W_i^{(z)} z_i + W_i^{(y)}(\alpha y_1 + (1-\alpha)y_1) + b_i)$$

By convexity and non-decreasing nature of g_i, we have that:

$$g_i(W_i^{(z)} z_i + W_i^{(y)}(\alpha y_1 + (1-\alpha)y_2) + b_i) \leq g_i(W_i^{(z)} z_i + W_i^{(y)}(\alpha y_1) + b_i) + g_i(W_i^{(z)} z_i + W_i^{(y)}((1-\alpha)y_2) + b_i)$$

Since $W_i^{(z)}$ non-negative, enforces the inequality:

$$z_{i+1}(y) \leq \alpha g_i(W_i^{(z)} z_i + W_i^{(y)}(y_1) + b_i) + (1-\alpha) g_i(W_i^{(z)} z_i + W_i^{(y)}(y_2) + b_i)$$

$$= \alpha z_{i+1}(y_1) + (1-\alpha) z_{i+1}(y_2)$$

□

To reduce the fully-input convex network to a partially-input convex network, one such formulation presented in the paper is a k-layer network:

$$u_{i+1} = \tilde{g}_i(\tilde{W}_i u_i + \tilde{b}_i)$$

$$z_{i+1} = g_i(W_i^{(z)}(z_i \cdot [W_i^{(zu)} u_i + b_i^{(z)}]) + W_i^{(y)}(y \cdot (W_i^{(yu)} u_i + b_i^{(y)})) + W_i^{(u} u_i + b_i)$$

$$f(x, y; \theta) = z_k, u_0 = x$$

where u_i and z_i are hidden units. The final output for the energy function is the final hidden unit z_k of the k-th layer. Note that with this formulation, a purely feedforward network can be represented by setting the y path to be all zero except $W_{k-1}^{(yu)} = I$ and $W_{k-1}^{(y)} = 1^T$. In summary, the fully-input convex network and partially input convex network are presented in Figure 1. Later, in the experiment section, I evaluate the expressiveness and power of both.

2.3 Inference Techniques

Following the inference methods for energy based models, prediction in ICNN involves finding the y that minimizes $f(x, y; \theta)$ for input x and trained weights θ. In practice this optimization could be difficult to do, but given that it is a neural network, gradients can be easily computed using backpropogation. Thus, this

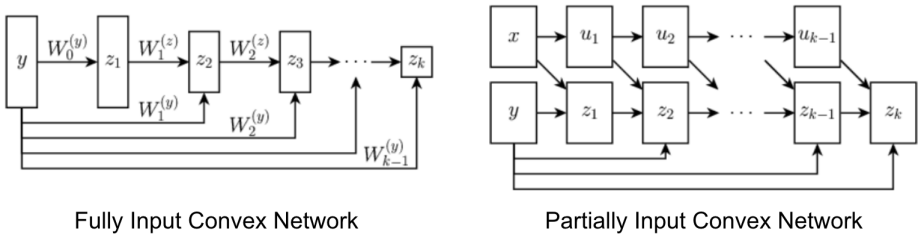

Figure 1: In summary, a pictorial representation of each model is presented. A main point to highlight is the inclusion of pass-through layers that connect y to hidden units which are not present in feedforward networks. The authors explicitly do this to prevent the hidden units from learning the identity function.

can be approximated using projected gradient descent.

$$y_+ \leftarrow \mathcal{P}_y(y - \alpha \nabla_y f(x, y; \theta))$$

In [1], the authors found varying effectiveness of this gradient descent method, so they developed a new approach called the bundle entropy method assuming that the convex space is bounded. An additional "barrier" function is added to the optimization in the form of negative entropy:

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

Negative entropy acts as a barrier because its gradient approaches infinity as it reaches the barrier and so the optimal solution must lie in the interior of the bounded space. So the final energy function includes a regularization term: $\tilde{f}(x, y; \theta) = f(x, y; \theta) - H(y)$ and the problem becomes to solve:

$$\arg\min_y \tilde{f}(x, y; \theta)$$

2.4 Learning Techniques

The paper presents the argmin differentiation method to directly minimize a loss function between true outputs and outputs predicted by the model. The authors draw a parallel to traditional feedforward neural network training of minimizing loss between predicted and true labels. From to the optimization set-up, this is done through implicit differentiation of the KKT optimality conditions. The authors include an in-depth proof to calculate the gradient of the network loss:

$$\nabla_\theta l(\hat{y}(x; \theta), y^*) = \sum_{i=1}^{k} (c_i^\lambda \nabla_\theta f(x, y^i; \theta) + \nabla_\theta (\nabla_y f(x, y^i; \theta)^T (\lambda_i c^y + c_i^\lambda (\hat{y}(x; \theta) - y^i))$$

where y^i is the solution returned at the ith bundle entropy method, λ the dual solution, and c the variables from the KKT conditions:

$$\begin{bmatrix} H & G^T & 0 \\ G & 0 & -1 \\ 0 & -1^T & 0 \end{bmatrix} \begin{bmatrix} c^y \\ c^\lambda \\ c^t \end{bmatrix} = \begin{bmatrix} -\nabla_{\hat{y}} l(\hat{y}, y^*) \\ 0 \\ 0 \end{bmatrix}$$

The main takeaway from this formulation is that each gradient component can be computed using standard forward and backward passes so it can be differentiated without being explicitly computed. The authors implement this in Tensorflow and use the ADAM optimizer to update parameters in training by minimizing the mean squared error of the predicted to true value.

3 How do ICNNs perform?

In my experiments, I aim to understand how ICNNs perform in classification and decision-making tasks. One aspect of ICNNs that I am interested in is the expressiveness of network, given that is it is restricted in some capacity due to the enforcement of convexity on a subset of inputs. After watching the presentation (see reference video) given by the main author at ICML 2017, a few people asked questions at the end about exactly how powerful ICNNs are. This is because one might expect that the convexity restriction on the outputs y might affect whether or not ICNNs are able to capture non-convex decision boundaries. The answer that was given in the video was not quite satisfactory as he mainly said that he thought ICNNs were quite expressive but did not have specific theories or experiments to support this claim. Particularly for non-convex output decision boundaries, one might expect a difference in accuracy compared to a standard feedforward network if ICNNs were not sufficiently powerful.

3.1 Training Set-up

I started building the experiments off of the initial open-source code provided by the authors. Given that this code is a few years old and has not been maintained, I needed to make quite a few changes to it to even make it run since the Tensorflow framework has changed within the years. The updated code can be found in my github repository.

3.2 Synthetic Data

In my experiments, I consider a set of synthetically generated data. The benefit of synthetic data is that they are easy to generate and easy to ensure that they capture some underlying structure. I use functions

from sklearn to generate data in a variety of shapes with varying levels of noise. The workflow for this set of experiments is as follows:

1. Generate data using sklearn or a predefined function.
2. Split data into train and test sets. (Note: the author's original implementation fails to do this so they could very well be overfitting in their original experiments.)
3. Train ICNN model for 200 steps.
4. Run the model in a grid of valid input spaces in (x,y) and plot decision boundary.

3.3 FICNN vs. ICNN

Firstly, I was interested in comparing the difference between ICNN and fully input convex networks. In Figure 2, we see that fully input convex networks are extremely restrictive in terms of the decision boundaries that they are able to learn. In this example, the canonical lima bean shaped decision boundary is clearly non-convex. We can see that the fully input convex network is not able to capture this type of shape. However, the input convex model is able to clearly capture the lima bean shape. The authors present a few more examples in their paper where the FICNN decision boundary does not fully match the true decision boundary. This evaluation thus far demonstrates that the ICNN is much more flexible than the fully input convex networks. Non-convex decision boundaries are desirable because many real world problems cannot be captured with convex decision boundaries.

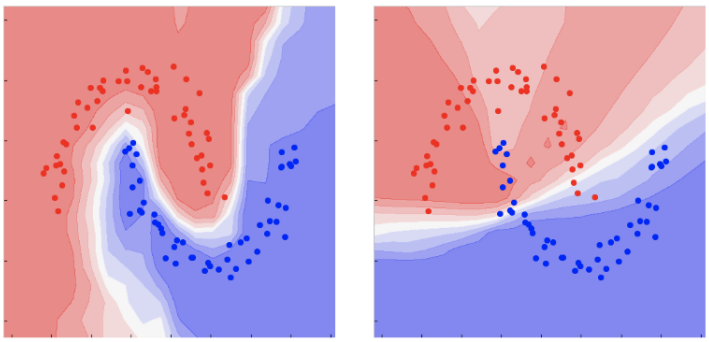

Figure 2: (Left) partially input convex networks (ICNN), (Right) fully input convex networks (FICNN). The colored dots on the graph denote the true label and the background denotes the prediction for that area. If the colored dot matches the background then the network would have classified that point correctly. The same interpretation can be used for the later figures.

3.4 Feedforward Network Comparison

For this experiment, I looked at a saddlepoint shaped function:

$$f(x, y) = x^3 - 3xy^2$$

To convert this into a discrete prediction problem, I define the function in this way:

$$\hat{f}(x, y) = \begin{cases} 0 & f(x, y) \leq 0 \\ 1 & else \end{cases}$$

This is an interesting function to approximate because if you slice the function at $f(x, y) = 0$, the saddle-point nature means that it cannot be divided by a separating plane into a half that lies on one half of the function and one that lies in the other half. Then I trained a neural network with 5 hidden layers (sizes 1280, 320, 64, 32, 16) and sigmoid output with binary crossentropy loss since the output is 0 or 1. I also use the Adam optimizer with learning rate 0.01.

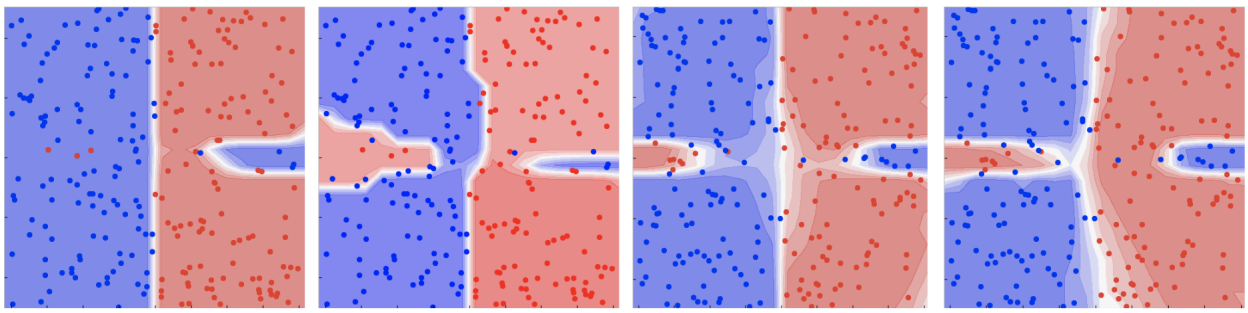

Figure 3: (1) and (2) are two different outcomes of training a feedforward network. (3) is the consistent outcome of an Input convex network (ICNN) and (4) is the consistent outcome of a fully input convex network (FICNN) .

I compare the learned decision boundary against that of the ICNN and fully convex neural network as shown in Figure 3. We can see that training feedforward networks to approximate saddle points is extremely unstable. This is but two widely differing learned decision boundaries. In (1) the network does not even learn the curve on the left side and in (2) the network seems to overfit heavily. In (3), the ICNN successfully learns the symmetric shape of the problem. In (4) the FICNN nearly seems to have some curvature in the separating space due to the convex formulation of the network. As shown in Table 1, the feedforward network is the fastest to run for the same amount of epochs. The FICNN is much faster than the ICNN due

to the larger set that it is being optimized over.

Architecture	Runtime (s)
Fully input convex	73.01
(Partially) Input convex network	149.45
Feed forward network	6.77

Table 1: Runtime results for 200 epochs of each of the three methods.

Finally I compare the training curves. The FICNN and ICNNs use the mean squared error loss compared to the feedforward neural network with a binary cross entropy loss. I am most interested in how smoothly the networks converge to the final decision boundary. As shown in Figure 4, it seems as if the more convexity is enforced in the training, the more smoothly the function converges to a minimum. This makes sense because the function in consideration is smooth and by enforcing convexity, the projected gradient descent algorithm is able to pick descent directions that are not explored previously. This is however not the case for feedforward networks which are clearly very noisy. In such cases, feedforward networks require more parameter tuning and special selection of stopping criterion, whereas convex optimization methods have a natural 'braking' system built in.

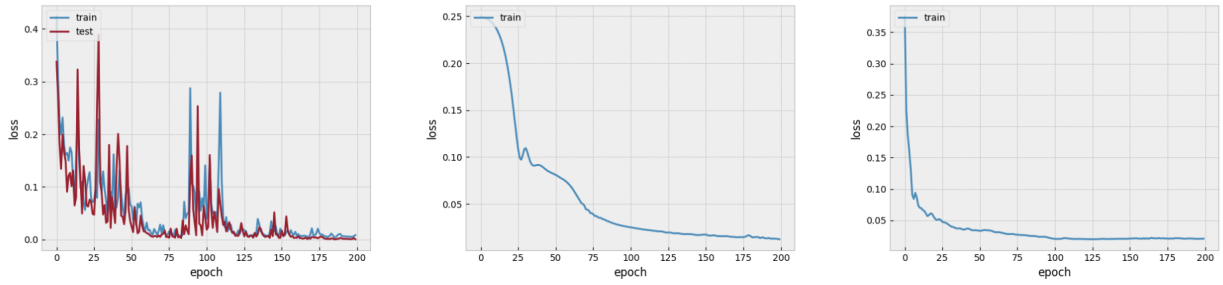

Figure 4: (Left) Loss curves for training a feedforward network. (Middle) Loss curve for training an ICNN. (Right) Loss curve for training an FICNN.

3.5 Sensitivity Analysis

Overall, given the experiments thus far, ICNNs seem fairly robust. This was not a particular selling-point that the authors emphasized, perhaps because it is difficult to prove concretely and mathematically. However, I was interested to explore this direction further empirically. The remainder of the experiments focus specifically on the ICNNs. Specifically I looked at varying the learning rate on the loss curve and learned decision boundary. As shown in Figure 5, even with widely varying learning rates, the ICNN network converges to the minimum. This is unlike traditional feedforward networks which require very particular tuning of parameters, including learning rates.

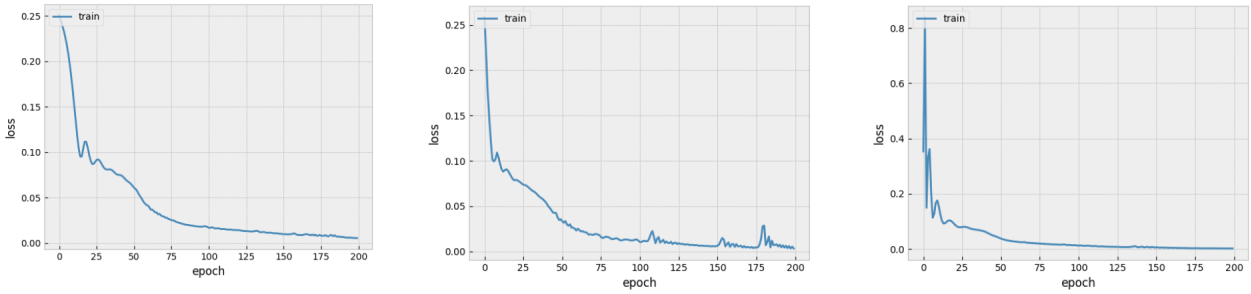

Figure 5: From left to right, adjusted learning rates of $\alpha = 0.1, 1, 5$.

4 Discussion

One limitation that I did not present is that the ICNN does not perform well on predicting continuous variables. In the saddle point problem, I tried to predict $f(x, y)$ instead of $\hat{f}(x, y)$ and the loss curve was unchanged throughout. This is perhaps due to the nature of energy based models. If the output space is large, as a continuous variable would be, then the network would require many more samples to learn the underlying distribution of the output variable whereas a categorical output is constrained to a fairly small, fixed amount. However, given the experiments that I looked at, it does seem like ICNN lies in the middle ground between optimization and deep learning. It would be interesting to look into proving this mathematically or doing more rigorous experiments in this direction.

After reading many online blogs, it seems like energy based models are a largely untapped potential in terms of directions for deep learning. Input neural networks are but one formulation of this type of approach. A recent ICML workshop paper [5] presents an idea of using energy based models for psychologically-inspired concept learning. I believe that energy based models are interesting because learning associations between variables and configurations of variables may well parallel how humans learn to associate concepts and make decisions.

5 Conclusion

In this paper I presented a walk-through of the ICNN formulation including a background on energy based models. Then I also presented some experiments that demonstrate the difference between fully and partially input convex networks, particularly how ICNNs are far more expressive than FICNNs. Then I extended the comparison to include feedforward neural networks. Overall, I found that by evaluating the learned decision boundary and the training curves that ICNNs are more stable to train than feedforward networks in particular settings. The code that I used to run the experiments can be found in github repository.

6 Acknowledgement

I would like to thank Sekhar and the TA's Timothy and Yifei for a great semester as I learned a lot in this optimization course that helped me to better understand the ICNN paper compared to when I initially chose the paper. Also I would like to thank the CMU authors for open-sourcing helpful code.

References

[1] Brandon Amos, Lei Xu, and J. Zico Kolter. Input convex neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 146–155. PMLR, 2017.

[2] David Belanger and Andrew McCallum. Structured prediction energy networks. In *International Conference on Machine Learning*, pages 983–992, 2016.

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

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

[5] Igor Mordatch. Concept learning with energy-based models. *arXiv preprint arXiv:1811.02486*, 2018.