Neural network inversion beyond gradient descent

Eric Wong ericwong@cs.cmu.edu
Carnegie Mellon University
J. Zico Kolter zkolter@cs.cmu.edu
Carnegie Mellon University

Abstract

In this paper, we tackle the problem of inverting deep networks, the task of finding an input that minimizes some criterion of the output of a deep network. We approach this problem by reformulating the network propagation as a constrained optimization problem and solve it using an operator splitting approach, the alternating direction method of multipliers. We show improvement in both iteration count and solution quality on a LeNet architecture in various settings, and we further apply this method to a large-scale network, VGG-11.

1 Introduction

This paper deals with the problem of "inverting" a neural network model: given some network function $f : \mathcal{X} \to Y$, for some $y \in \mathcal{Y}$ we want to find the $x \in \mathcal{X}$ such that $f(x) = y$. Or more generally, if $\ell : \mathcal{Y} \to \mathbb{R}$ is any function defined over the network output (or it's intermediate units), we may want to find an input x that minimizes $\ell(f(x))$.[1] Numerous applications have recently been proposed that rely on such methods: adversarial examples [1, 2, 3] optimize over inputs to to a nominal input to maximize the loss that the network will suffer; neural style algorithms [4] optimize over input to the network to maximize similarity of internal gram matrices of the network to match a some target; and GAN-based image completion algorithms optimize over a latent state (input to a generator network) to maximize similarity to some partially observed output [5].

Yet the workhorse for all these applications, the algorithm that actually performs the minimization of $\ell(f(x))$, is virtually always just some form of gradient descent (including adaptive gradient methods). While this is appealing in its simplicity, it also ignores a great deal of techniques that have become prevalent in modern optimization, such as operator splitting approaches (which can substantially outperform gradient-based methods on many classes of optimization problems).

In this paper, we consider an alternative approach based upon (nonconvex) operator splitting techniques for constrained optimization. In particular, instead of considering a neural network as a simple feedforward function, we consider it as set of non-convex constraints, namely 1) (convex) equality constraints of each layer with the activation of the previous layer and with the pre-activation of the next layer; and 2) non-convex constraints representing non-linearities such as the rectified linear unit or the maxpooling operator. We then derive an operator splitting approach (we use the alternating direction method of multipliers (ADMM) [6], as a simple example, though other splitting approaches could be similarly applies) that seeks to optimize the objective $\ell(f(x))$ by representing $f(x)$ via these constraints. A similar approach was used by Taylor et al. [7], though in the context of training the network rather than inverting it.

We show two crucial points: 1) the proposed method can improve (both in terms of iteration count and solution quality) over existing methods in actual use, such as the Adam optimizer, which is heavily used in practice for such tasks. And (perhaps most importantly) 2) we derive and demonstrate how to apply the method to *realistic large-scale networks actually used in modern deep learning*, specifically convolutional networks. Properly applying operator splitting to these convolutional architectures requires handling the closed-form solution of operator splitting updates involving convolutional architectures, which we achieve by using FFT representations of the convolutions as additional linear operators combined with carefully encoded zero-padding. In this aspect we notably build upon similar work such as that in Taylor et al. [7], which doesn't consider convolutional models of any type. We demonstrate the application to the VGG11 network, showing that each operator splitting iteration is only a few times more costly than a forward and backward pass for backprop.

[1] Although this problem may be viewed as just arbitrary optimization over the neural network input, we prefer the term "inversion" to describe the task because "optimization" in neural networks virtually always refers to optimization over the network parameters, not the input.

OPTML 2017: 10th NIPS Workshop on *Optimization for Machine Learning* (NIPS 2017).

2 Inversion using ADMM

2.1 Formulation as a constrained optimization problem

For a given neural network $f : \mathcal{X} \to Y$, any loss function $\ell : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, and a desired output $y \in \mathcal{Y}$, the model inversion problem is to find an input example $x \in \mathcal{X}$ which minimizes $\ell(f(x), x)$. While in general ℓ can be a function of any of the intermediary layer outputs, we consider the case where ℓ is dependent on the input and output space, which allows us to represent constraints or regularization on the input.

$$\min_x \ell(x, f(x)) \tag{1}$$

To turn this into a constrained optimization problem, we use k variables z_i at each of the k layers equal to the value of each layer:

$$\min_{z_1,\ldots,z_k} \ell(z_1, z_k) \text{ s.t.} \tag{2}$$

$$z_{i+1} = a_i(W_i z_i + b_i), 1 \leq i < k) \tag{3}$$

where W_i, b_i are the weights at each layer and a_i are the activation functions (the weights W_i, b_i are not restricted to be matrices, but can be any linear operation). We introduce two additional variables $(\tilde{v}_i, v_i)$ at each intermediary layer, equal to the layer's pre- and post-activation, and constrain them to the set of inputs and outputs of the corresponding activation, defined as $\mathcal{A}_i = \{(x, y) | a_i(x) = y\}$.

$$\min_{z_1,\ldots,z_k} \ell(z_1, z_k) \text{ s.t.} \tag{4}$$

$$z_{i+1} = v_i, \ 1 \leq i < k \tag{5}$$

$$\tilde{v}_i = W_i z_i + b_i, \ 1 \leq i < k \tag{6}$$

$$(\tilde{v}_i, v_i) \in \mathcal{A}_i, \ 1 \leq i < k \tag{7}$$

2.2 ADMM update iterates

Following the standard ADMM methodology, we minimize the augmented Lagrangian with dual variables $(\tilde{u}_i, u_i)$ for $1 \leq i < k$, which results in the following ADMM z_i iterates.

$$z_k = \arg\min_{z_k} \ell(z_1, z_k) + \frac{\rho}{2} ||z_k - v_{k-1} - u_{k-1}||^2 \tag{8}$$

$$z_i = (I + W_i^T W_i)^{-1}(v_{i-1} + u_{i-1} + W_i^T(-b_i + \tilde{v}_i + \tilde{u}_i)), \ 1 < i < k \tag{9}$$

$$z_1 = \arg\min_{z_1} \ell(z_1, z_k) + \frac{\rho}{2} ||W_1 z_1 + b_1 - \tilde{v}_1 - \tilde{u}_1||^2 \tag{10}$$

Depending on the choice of ℓ, the updates for z_1 and z_k may have a closed form expression (e.g., this is the case for linear and mean squared error). In this case, the updates have the same complexity as a forward or backwards pass in backpropagation, and in practice is about 2-3 times slower per iteration. Otherwise, we can solve these two updates using a Newton based solver (e.g. for rectangular bounds and cross entropy loss), which converges rapidly due to strong convexity.

The $(\tilde{v}_i, v_i)$ update is a projection onto $\mathcal{A}_i$, denoted as $P_{\mathcal{A}_i}$, and the $(\tilde{u}_i, u)$ update is standard for ADMM.

$$(\tilde{v}_i, v_i) = P_{\mathcal{A}_i}(W_i z_i + b_i - \tilde{u}_i, z_{i+1} - u_i), \ 1 \leq i < k \tag{11}$$

$$(\tilde{u}_i, u_i) = (\tilde{u}_i + \tilde{v}_i - (W_i z_i + b_i), u_i + v_i - z_{i+1}), \ 1 \leq i < k \tag{12}$$

For our setting, we use projections onto linear activations, rectified linear units, max-pooling activations, and combinations of the aforementioned, however the approach is not restricted to these. The final algorithm is to simply repeat these updates until converegence.

Initialization If the objective ℓ contains a penalty or regularization around some starting example $x_0 \in \mathcal{X}$ (e.g. an L2 penalty), a natural initialization for the algorithm is to use the values of a forward pass on this example through the network. Since all constraints are satisfied by this initialization, changes in the variables propagate from the output layer in a fashion similar to backpropagation.

Figure 1: Plot of objective value of a neural network while inverting with ADMM when using different update orderings. The backwards ordering is most efficient.

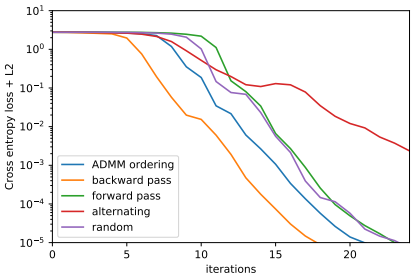
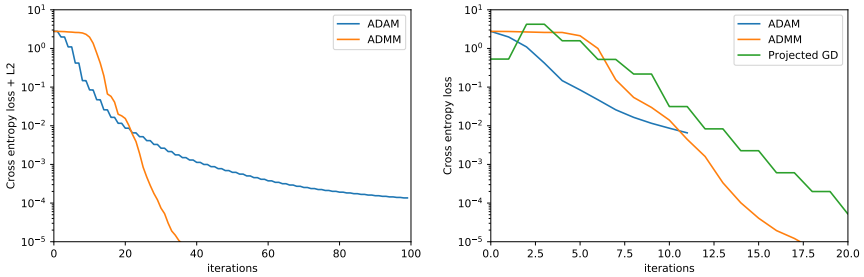


Figure 2: Comparison of ADMM-based inversion vs. ADAM and projected gradient descent on a LeNet architecture. Left: minimizing cross entropy loss with an L2 regularization. Right: minimizing cross entropy loss within a bounded rectangle. Here, ADAM is run until it reaches the boundary.



Convolutional layers The VGG architecture [8] presents a challenge in handling the convolutional layers. We exploit the fact that with proper padding, convolutional mappings are simply element-wise multiplication in the FFT space, an idea commonly utilized in convolutional networks [9, 10]. We can then implement a convolution as a linear map $\text{conv}_i(x) = W_i x = U^* D U x$, where U, U^* are the FFT and inverse FFT operations and D is a diagonal matrix performing the element-wise product in the FFT space. To achieve the proper padding, each projection operator $P_{\mathcal{A}_i}$ can be augmented to project from a given padding to a desired padding.

3 Experimental results

3.1 LeNet

This first set of experiments are run on a LeNet architecture trained on the Cifar-10 dataset. For ADAM and projected gradient descent we use a step size of 0.01 and for ADMM we use $\rho = 10^{-5}$.

Order of updates While traditionally the ADMM updates are z_i updates for all i, followed by $(\tilde{v}_i, v_i)$ and $(\tilde{u}_i, u_i)$, the structure of a neural network provides a natural ordering: a 'forward' or 'backwards' pass, which switches between z_i and $(\tilde{v}_i, v_i)$ updates. Alternating forwards and backwards passes, which most closely mimics the classical backpropagation algorithm, performed worst while the backward pass was most effective, and so we use the backward pass ordering for all remaining experiments.

Performance For the LeNet architecture, we minimize the cross entropy loss subject to L2 regularization and bounded rectangle constraints on the input, and compare them to using ADAM (and projected gradient descent in the case of bounded rectangular constraints) in figure 2. We see that ADMM is able to improve on both the iteration count and objective value in both settings,.

3.2 VGG

The next set of experiments are to demonstrate the qualitative ability of the method to work on deep networks, namely VGG-11 on Cifar-10. In this experiment, we invert a VGG network with softmax loss and

Figure 3: Convergence plots for ADMM on VGG-11. Left: Objective value over iterations. Right: ADMM primal and dual residuals for the ADMM constraints.

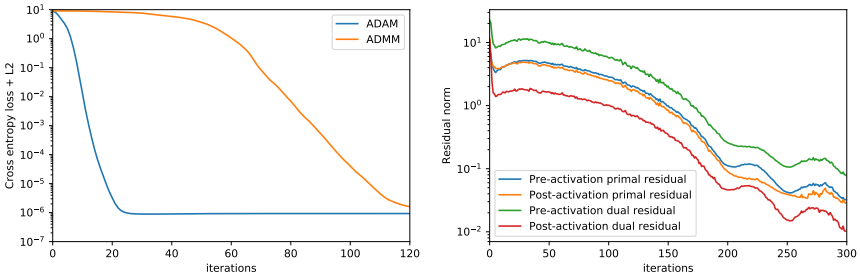
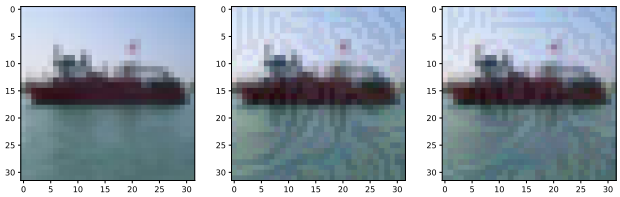


Figure 4: An adversarial example. From left to right: original image, projected gradient, ADMM. Both projected gradient and ADMM are able to fool the classifier, changing the label of the example below from ship to airplane with probability 1.



L2 regularization on the input. We see in figure 3 that ADMM is able to minimize the objective to the same value as ADAM, and have converging residuals despite the deep architecture and the non-convexity of the problem, showing that operator splitting methods can be used to optimize deep networks. However, some work remains to make these methods computationally competitive with current gradient based approaches in this setting.

In this second experiment, we use the ADMM method with bounded box constraints on the input to generate adversarial examples, by minimizing a linear function of the correct output minus the desired class, subject a bounding box constraint. We find that ADMM is able to achieve the same quality of adversarial example as projected gradient, as both are able to completely fool the VGG network with probability 1.

4 Discussion

We considered operator splitting techniques, namely ADMM, for inverting neural networks. The basic idea is to reformulate the neural network as a set of non-convex constraints and applies a wide range of architectures. We apply the method to modules found in typical deep learning architectures, and show empirical results on the LeNet architecture which outperforms gradient descent based approaches, but future work is required to bring this speed to deeper networks like VGG-11. However, despite the reduction in speed, the operator splitting technique is able to find solutions qualitatively as good as existing methods on deeper models.

References

[1] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[2] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[3] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[4] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[5] Raymond Yeh, Chen Chen, Teck Yian Lim, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with perceptual and contextual losses. *arXiv preprint arXiv:1607.07539*, 2016.

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

[7] Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training neural networks without gradients: A scalable admm approach. In *International Conference on Machine Learning*, pages 2722–2731, 2016.

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

[9] Tyler Highlander and Andres Rodriguez. Very efficient training of convolutional neural networks using fast fourier transform and overlap-and-add. *arXiv preprint arXiv:1601.06815*, 2016.

[10] Oren Rippel, Jasper Snoek, and Ryan P Adams. Spectral representations for convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 2449–2457, 2015.