Linear Inequality Constraints for Neural Network Activations

Thomas Frerix ¹  Matthias Nießner ¹  Daniel Cremers ¹

Abstract

We propose a method to impose linear inequality constraints on neural network activations. The proposed method allows a data-driven training approach to be combined with modeling prior knowledge about the task. Our algorithm computes a suitable parameterization of the feasible set at initialization and uses standard variants of stochastic gradient descent to find solutions to the constrained network. Thus, the modeling constraints are always satisfied during training. Crucially, our approach avoids to solve a sub-optimization problem at each training step or to manually trade-off data and constraint fidelity with additional hyperparameters. We consider constrained generative modeling as an important application domain and experimentally demonstrate the proposed method by constraining a variational autoencoder.

1. Introduction

Deep learning models (LeCun et al., 2015) have demonstrated remarkable success in tasks that require exploitation of subtle correlations, such as computer vision (Krizhevsky et al., 2012) and sequence learning (Sutskever et al., 2014). Typically, humans have strong prior knowledge about a task, e.g., based on symmetry, geometry, or physics. Learning such a priori assumptions in a purely data-driven manner is inefficient and, in some situations, may not be feasible at all. While certain prior knowledge was successfully imposed – for example translational symmetry through convolutional architectures (LeCun et al., 1998) – incorporating more general modeling assumptions in the training of deep networks remains an open challenge. Recently, generative neural networks have advanced significantly (Goodfellow et al., 2014; Kingma & Welling, 2014). With such models, controlling the generative process beyond a data-driven, black-box approach is particularly important.

In this paper, we present a method to impose prior knowledge in form of linear inequality constraints on the activations of deep learning models. We directly impose these constraints through a suitable parameterization of the feasible set. This has several advantages:

- The constraints are hard-constraints in the sense that they are satisfied at any point during training.
- There is no manual trade-off between constraint satisfaction and data representation.
- The proposed method can easily be applied to constrain not only the network output, but also any intermediate activations.

In summary, the main contribution of our method is a reparameterization that incorporates linear inequality hard-constraints on neural network activations. The model can be optimized by standard variants of stochastic gradient descent. As an application in generative modeling, we demonstrate that our method is able to produce authentic samples from a variational autoencoder while satisfying the imposed constraints.

2. Related Work

Various works have introduced methods to impose some type of hard constraint on neural network activations. This differs from a classical constrained optimization problem (Nocedal & Wright, 2006) in that the constraints are on the image of a parameterized function rather than optimization variables, i.e., neural network parameters.
Márquez-Neila et al. (2017) formulated generic differentiable equality constraints as soft constraints and employed a Lagrangian approach to train their model. While this is a principled approach to constrained optimization, it does not scale well to practical deep neural network models with their vast number of parameters. To make their method computationally tractable, a subset of the constraints is selected at each training step. In addition, these constraints are locally linearized; thus, there is no guarantee that this subset will be satisfied after a parameter update.

Pathak et al. (2015) proposed an optimization scheme that alternates between optimizing the deep learning model and fitting a constrained distribution to these intermediate models. They deal with a classification task and the fitting step is in the Kullback-Leibler sense. However, this method involves solving a (convex) sub optimization problem at each training step. Furthermore, the overall convergence path depends on how the alternating optimization steps are combined, which introduces an additional hyperparameter that must be tuned.

OptNet, an approach to solve a generic quadratic program as a differentiable network layer, was proposed by Amos & Kolter (2017). OptNet backpropagates through the first-order optimality conditions of the quadratic program, and linear inequality constraints can be enforced as a special case. The formulation is flexible; however, it scales cubically with the number of variables and constraints. Thus, it becomes prohibitively expensive to train large-scale deep learning models.

Finally, several works have proposed handcrafted solutions for specific applications, such as skeleton prediction (Zhou et al., 2016) and prediction of rigid body motion (Byravan & Fox, 2017). In contrast, to avoid laborious architecture design, we argue for the value of generically modeling constraint classes. In practice, this makes constraint methods more accessible for a broader class of problems.

Contribution In this work, we tackle the problem of imposing linear inequality constraints on neural network activations. Rather than solving an optimization problem during training, we split this task into a feasibility step at initialization and an optimality step during training. At initialization, we compute a suitable parameterization of the constraint set and use the neural network training algorithm to find a good solution within this feasible set. Conceptually, compared to an unconstrained model, we are trading-off computational cost during initialization to obtain a model that can be trained with nearly no overhead. The proposed method is implemented as a neural network layer that is specified by a set of linear inequalities and whose output parameterizes the feasible set.

3. Linear Inequality Constraints for Deep Learning Models

We consider a generic $L$ layer neural network $F_{\theta}$ with model parameters $\theta$ for inputs $x$ as follows:

$$F_{\theta}(x) = f_{\theta_L}^{(L)}(\sigma(f_{\theta_{L-1}}^{(L-1)}(\ldots f_{\theta_1}^{(1)}(x)\ldots))), \quad (1)$$

where $f_{\theta_i}^{(l)}$ are affine functions, e.g., a fully-connected or convolutional layer, and $\sigma$ is an elementwise non-linearity, e.g., a sigmoid or rectified linear unit (ReLU). In supervised learning, training targets $y$ are known and a loss $L_{\theta}(F_{\theta}(x))$ is minimized as a function of the network parameters $\theta$. A typical loss for a classification task is the cross entropy in the Kullback-Leibler sense. However, this method does not scale well to practical deep neural network models with their vast number of parameters. To make their method a principled approach to constrained optimization, it does not consider equality constraints as soft constraints and employed a Lagrangian approach to train their model. While this is formally a different element-wise non-linearity for each layer. We omit such details in favor of notational simplicity.
by the convex hull of its vertices. If it is unbounded (not shown), then it has a conic contribution. Each polyhedral set of the form (2) can be expressed as follows
\[
\mathcal{H} := \{ z | \tilde{A} z \leq 0, \tilde{A} = [A, -b] \in \mathbb{R}^{m \times d+1}, z_{d+1} \leq 1 \},
\]
in the sense that if \((z_1, \ldots, z_{d+1}) \in \mathcal{H}\), then \((z_1, \ldots, z_d) \in \mathcal{C}\). We refer to this form as the \textit{homogeneous} formulation of the problem. In other words, with an additional constraint, every convex polyhedron can be lifted by one dimension. For this description, the vertices can be considered as endpoints of rays. The homogeneous problem is often formulated with an equality constraint \(z_{d+1} = 1\); however, the relaxed formulation is numerically advantageous (Section 3.2).

The theorem states that an intersection of half-spaces (half-space or H-representation) can be written as the Minkowski sum of a convex combination of the polyhedron’s vertices and a conical combination of some rays (vertex or V-representation). One can switch algorithmically between these two viewpoints via the double description method (Motzkin et al., 1953; Fukuda & Prodon, 1996), which we discuss in the following. Thus, the H-representation, which is natural when modeling inequality constraints, can be transformed into the V-representation, which can be incorporated into gradient-based neural network training.

### 3.1. Double Description Method

The double description method converts between the half-space and vertex representation of a system of linear inequalities. It was originally proposed by Motzkin et al. (1953) and further refined by Fukuda & Prodon (1996)\(^2\). Here, we are only interested in the conversion from H-representation to V-representation in homogeneous form (5).

\[
\mathcal{H} \rightarrow \text{cone}(r_1, \ldots, r_s).
\]

The core algorithm proceeds as follows. Let the rows of \(A\) define a set of homogeneous inequalities and let \(R = [r_1, \ldots, r_s]\) be the matrix whose columns are the rays of the corresponding cone. Here, \((A, R)\) form a double description pair. The algorithm iteratively builds a double description pair \((A^{k+1}, R^{k+1})\) from \((A^k, R^k)\) in the following manner. The rows in \(A^k\) represent a \(k\)-subset of the rows of \(A\) and thus define a convex polyhedron associated with \(R^k\). Adding a single row to \(A^k\) introduces an additional half-space constraint, which corresponds to a hyperplane. If the vector \(r = r_i - r_j\) for two columns \(r_i\), \(r_j\) of \(R^k\) intersects with this hyperplane and \(\text{cone}(r_i, r_j)\) is a face of \(R^k\), then this intersection point is added to \(R^k\). Existing rays that are cut-off by the additional hyperplane are removed from \(R^k\). The result is the double description pair \((A^{k+1}, R^{k+1})\). This procedure is shown in Figure 2.

Adding a hyperplane might drastically increase the number of rays in intermediate representations, which, in turn, contribute combinatorically in the subsequent iteration. In fact, there exist worst case polyhedra for which the algorithm has exponential run time as a function of the number of inequalities and the input dimension, as well as the number of rays (Dyer, 1983; Bremner, 1999). Under certain assumptions more efficient bounds are known. A convex polyhedron \(\mathcal{C} \subseteq \mathbb{R}^d\) is \textit{degenerate} if there exists \(x \in \mathcal{C}\) such that \(x\) fulfills more than \(d\) inequalities with equality; otherwise, \(\mathcal{C}\) is \textit{nondegenerate}. For nondegenerate polyhedra, the problem can be solved in \(\mathcal{O}(mdn_r)\) time complexity, where \(n_r\) is the number of rays in the final V-representation and \(m\) the number of constraints (Avis & Fukuda, 1992). However, \(m\) and \(n_r\) may depend unfavorably on the dimension \(d\). An extreme example is the unit box \(B = \{ x \in \mathbb{R}^d | -1 \leq x_i \leq 1 \}\) where \(m = 2d\) and \(n_r = 2^d\); thus, the algorithm has exponential run time in the dimension \(d\). Overall, one can expect the algorithm to be efficient only for problems with a reasonably small number \(m\) of inequalities and dimension \(d\).

### 3.2. Integration in Neural Network Architectures

We parameterize the homogeneous form (5) of the problem via a neural network layer. This layer takes as input some (latent) representation of the data, which is mapped to activations satisfying the desired hard constraints. The algorithm is provided with the H-representation of linear inequality constraints, i.e., \(A\) and \(b\) to specify the feasible set (5). At initialization, we convert this to the V-representation via the

---

\(^2\)In our experiments we use \texttt{pycddlib}, which is a Python wrapper of Fukuda’s \texttt{cddlib}.

\(^3\)\(F \subseteq \mathcal{C}\) is a face of the convex set \(\mathcal{C}\) if it holds for all \(x, y \in \mathcal{C}\) that \((\forall \lambda \in (0, 1), (1 - \lambda)x + \lambda y \in F \Rightarrow x, y \in F)\).
Linear Inequality Constraints for Neural Network Activations

double description method (Section 3.1). This corresponds to computing the set of rays \{r_1, \ldots, r_s\} to represent the polyhedral cone. During training, the neural network training algorithm is used to optimize within the feasible set. There are two critical aspects in this procedure. First, as outlined in Section 3.1, the run-time complexity of the double description method may be prohibitive. Conceptually, the proposed approach allows for significant compute time at initialization to obtain an algorithm that is very efficient at training time. Second, we must ensure that the mapping from the latent representation to the parameters integrates well with the training algorithm. We assume that the model is trained with gradient-based backpropagation, as is common for current deep learning applications. The constraint layer comprises an affine mapping (fully-connected layer with biases) followed by the element-wise absolute value function that ensures the non-negativity required by the conical combination parameters. To ensure the constraint \(z_{d+1} \leq 1\), we scale the entire vector \(z \in \mathbb{R}^{d+1}\) by \(z_{d+1}\) if \(z_{d+1} > 1\). This is a valid operation on a cone, i.e., if \(z \in \mathbf{C}\), then \(\alpha z \in \mathbf{C}\) \(\forall \alpha \geq 0\). Therefore, our original constraints are not violated.

Several choices must be made to obtain a working backpropagation algorithm. Since adding a constraint to the network maps Euclidean space to a smaller subset, such a formulation is intuitively susceptible to a vanishing gradient problem. There must be points that are distant in Euclidean space, but are mapped to relatively close points in the feasible set. Thus, moving in Euclidean space induces a small change in the feasible set. A one-dimensional example of such a mapping is the sigmoid \(\sigma : \mathbb{R} \rightarrow (0, 1), x \mapsto 1/(1 + \exp(-x))\), which exhibits vanishing gradients for large positive and negative values. We build on this intuition to design our constraint layer. We do not work with the non-homogeneous formulation (2) directly. Instead, we lift the problem to the homogeneous formulation (5). In fact, a natural way to enforce that the convex combination parameters of the homogeneous formulation are on the probability simplex \(\{\lambda | \lambda_i \geq 0, \sum_i \lambda_i = 1\}\) is via a softmax mapping, as follows:

\[
\text{softmax} : \mathbb{R}^n \rightarrow \mathbb{R}^n, x \mapsto \left[ \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \right]_{i=1}^n.
\]

This function has vanishing gradients when one \(x_i\) is significantly greater than the other vector entries. In this case, the softmax maps close to a vertex on the probability simplex. This is undesirable behavior for our task and leads to slow convergence, which we also observed numerically. In the homogeneous formulation, we must enforce an additional constraint. We enforce an inequality constraint \(z_{d+1} \leq 1\) rather than an equality constraint \(z_{d+1} = 1\), which would be an equally valid description of the problem. However, with the inequality, the optimization algorithm is less constrained and we only need to interfere by scaling when \(z_{d+1} > 1\). Finally, we selected the absolute value function to enforce non-negativity of the conical combination parameters. In theory, any function \(f : \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}\) would fulfill this requirement; however, care must be taken to not interfere with backpropagated gradients. For example, the ReLU function \(\sigma(x) = \max(0, x)\), which is commonly used as a non-linearity in neural networks, has zero gradient for \(x < 0\). This implies that certain conical combination parameters are zero and cannot become non-zero during optimization. The absolute value function interferes least with the backpropagated gradient in the sense that it preserves the magnitude of the backpropagated signal and at most changes its sign.

### 3.3. Combining Modeling and Domain Constraints

Domain constraints are often formulated as box constraints, \(\{x | a \leq x \leq b\}\), such as a pixel domain in computer vision applications. As indicated in Section 3.1, box constraints are particularly unfit to be converted using the double description method because the number of vertices is exponential in the dimension. Therefore, in this paper, we distinguish modeling constraints and domain constraints and only convert the former into V-representation while enforcing the latter through a joint projection step at test time. These two types of constraints differ conceptually. While the modeling constraints may conflict with the data-driven task, the domain constraints are in line with fitting the data. Consequently, the joint projection after successful training can be expected to result in only a small correction.

### 4. Numerical Results

We demonstrate the proposed constraint method in two different settings. In an initial experiment, we project the input data onto a constraint set. Here, the result can be compared to the optimal solution of the convex optimization problem. The purpose of this simple setup is to show that, despite being non-convex, the proposed method does not interfere with the optimizability of the problem. In a second experiment, consistent with our motivation to modify the output of generative models, we constrain the output of a variational autoencoder, and show samples drawn from this constrained model.

We used the MNIST dataset (LeCun et al.) for both experiments (50000 training samples, 1000 validation and test samples).

We have used PyTorch (Paszke et al., 2017) for our implementation and all experiments were performed on a single Nvidia Titan X GPU.

\*Our implementation will be publicly available.
4.1. Orthogonal Projection onto a Constraint Set

Using a simple toy problem, we demonstrate that the proposed algorithm can find good solutions to a constrained learning problem. For given linear inequalities specified in H-representation, we solve the following problem

$$\min_{z \in \mathbb{R}^n} \|z - y\|_2$$

subject to

$$Az \leq b,$$

where $y$ is an MNIST image. Here, the problem is convex; therefore, the global optimum can be readily computed and can be compared to the performance on a held-out validation set. We impose a checkerboard constraint with 16 tiles, where neighboring tiles are constrained to be either below or above the median intensity of the image domain on average. The pixel intensity domain in our experiments is $[-1, 1]$ and thus the tiles’ average intensity is positive or negative, respectively. In this setting, we can expect that training an unconstrained network with subsequent projection onto the constraint set at test time will yield good results. Here, let $P_C(y) = \arg\min_{z \in C} \|z - y\|_2$ be the orthogonal projection onto the constraint set $C$ and denote the mean-squared error as $L_y(x) = \|x - y\|_2$. Both mappings are Lipschitz continuous with Lipschitz constant $L = 1$. Consequently, for output $\hat{y}$ of an unconstrained model,

$$|L_y(P_C(\hat{y})) - L_y(P_C(y))| \leq \|P_C(\hat{y}) - P_C(y)\|_2$$

$$\leq \|\hat{y} - y\|_2,$$

where, by definition, the term $L_y(P_C(y))$ is the optimal value of problem (8). The training algorithm fits $\hat{y}$ to $y$; therefore, it follows that projecting the unconstrained output $\hat{y}$ onto the constraint set will yield an objective value that is close to the optimal value of the constrained optimization problem.

We provide results obtained using the constraint parameterization algorithm and the projection of an unconstrained solution onto the constraint set. To have a comparable number of parameters for these models, we use a single fully-connection layer in both cases. For the unconstrained model, we employ an FC(784, 784) layer, and for the constrained model we employ an FC(784, $n_r$) layer with $n_r = 1552$ many rays to represent the constraint set in V-representation. Both models were optimized using the Adam optimizer (Kingma & Ba, 2015) and a learning rate of $10^{-4}$. Note that we are dealing with a mean-squared loss; thus, we expect the test time projection method to work well (Section 3). Figure 3 shows that a validation objective for both algorithms converges to the average optimum over the validation set. As expected for a more constrained optimization procedure, convergence with the constraint parameterization method is slower. Figure 4 shows a test set sample and the output of the constraint parameterization network.

![Figure 3](image1.png)

**Figure 3.** Convergence of the objective function (8) on a held-out validation set for the proposed constraint parameterization method and unconstrained optimization with subsequent projection. Both methods generalize close to the average optimum over the validation set, which is obtained as a solution to a convex optimization problem.

![Figure 4](image2.png)

**Figure 4.** MNIST sample from a test set (left) and output of a model using the proposed constraint parameterization layer (right). The model was trained to learn the orthogonal projection onto the constraint set as defined by (8).

4.2. Constrained Generative Modeling with Variational Autoencoders

Variational autoencoders (VAE) are a class of generative models that are jointly trained to encode observations into latent variables via an encoder or inference network and decode observations from latent variables using a decoder or generative network (Kingma & Welling, 2014). We model the joint distribution $p_\theta(x, z)$ of a generative latent variable model. In this model, $x \in D$ is the observed image data, $z$ is a latent variable, and $\theta$ are the generative network parameters. After optimizing these parameters, we can generate samples from the joint distribution via $p_\theta(x, z) = p_\theta(x|z)p_\theta(z)$ by assuming an isotropic Gaussian prior on the latent space, $z \sim \mathcal{N}(0, I)$, and a generative distribution $p_\theta(\cdot|z) \sim \mathcal{N}(f_\theta(z), I)$. The parameterized function $f_\theta(z)$ is expressed by a neural network (decoder or generative network). On the inference side, we approximate the intractable posterior $p_\theta(\cdot|x)$ by a variational distribution over the latent space. We assume a data conditional distribution $q_\lambda(\cdot|x) \sim \mathcal{N}(\mu_\lambda(x), \sigma_\lambda(x)I)$, which is parameterized by a neural network via parameters $\lambda$ (encoder or infer-
ence network). Here, we want to minimize the Kullback-Leibler divergence between this approximation and the true posterior, \( D_{KL}(q_{\lambda}(\cdot | x) \| p_{\theta}(\cdot | x)) \), which is generally intractable. Thus, we maximize a variational lower bound (ELBO) (Blei et al., 2017) as follows:

\[
\mathcal{L}_x(\theta, \lambda) := \mathbb{E}_{z \sim q_{\lambda}(\cdot | x)}[\log(p_{\theta}(x | z))] - D_{KL}(q_{\lambda}(\cdot | x) \| p_{\theta}(\cdot)) .
\]  

(10)

The first term maximizes the expected conditional likelihood under the variational distribution, and the second term regularizes this distribution to be close to the prior over the latent space. The objective can be optimized by sampling from the variational posterior as \( z = \sigma_{\lambda}(x) \varepsilon + \mu_{\lambda}(x), \varepsilon \sim N(0, I) \), which allows backpropagation through the sampling step to optimize parameters \( \lambda \). This is commonly referred to as the reparameterization trick (Kingma & Welling, 2014).

We base our implementation on (Baumgartner, 2018). The model has a fully-connected architecture:

encoder: \( FC(784, 256) - \text{ReLU} - FC(256, 2) \)

decoder: \( FC(2, 256) - \text{ReLU} - FC(256, 784) \)

- sigmoid - constraint

Here, \( \text{ReLU}(x) = \max(0, x) \) and the sigmoid non-linearity takes the form \( \sigma(x) = 1/(1 + \exp(-x)) \). In constrast to a standard VAE, we constrain the samples generated by the model to obey a checkerboard constraint as for the projection experiments in Section 4.1. To generate images, we sample the latent space prior \( z \sim N(0, I) \) and evaluate the decoding neural network (Figure 5). The model is able to sample authentic digits while obeying the checkerboard constraint.

5. Conclusion

To combine a data-driven task with modeling constraints, we have developed a method to impose linear inequality constraints on neural network activations. At initialization, a suitable parameterization is computed and subsequently a standard variant of stochastic gradient descent is used to train the reparameterized network. In this way, we can efficiently guarantee that network activations – in the final or any intermediate layer – satisfy the constraints at any point during training. An important application of the proposed method is generative modeling with prior assumptions. Therefore, we demonstrated experimentally that the proposed method can be used successfully to constrain the output of a variational autoencoder. Our method is implemented as a layer, which is simple to combine with existing and novel neural network architectures in modern deep learning frameworks and is therefore readily available in practice.

Figure 5. Samples from a constrained variational autoencoder. The images represent authentic digits while satisfying the imposed checkerboard constraint. For a pixel intensity domain \([-1, 1]\), the checkerboard constraint forces the image tiles to be on average positive or negative, respectively.

Acknowledgements

The authors would like to thank Thomas Möllenhoff and Gideon Dresdner for fruitful discussions and valuable feedback on the manuscript. This work was supported by an Nvidia Professorship Award, the TUM-IAS Carl von Linde and Rudolf Mößbauer Fellowships, the ERC Starting Grant Scan2CAD (804724), and the Gottfried Wilhelm Leibniz Prize Award of the DFG.
References

Amos, B. and Kolter, J. Z. OptNet: Differentiable optimization as a layer in neural networks. In Proceedings of the 34th International Conference on Machine Learning (ICML 2017), pp. 136–145, 2017.

Avis, D. and Fukuda, K. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. Discrete & Computational Geometry, 8(3):295–313, 1992.

Baumgärtner, T. VAE-CVAE-MNIST. https://github.com/timbmg/VAE-CVAE-MNIST, 2018. commit: e4ba231.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. Variational Inference: A Review for Statisticians. Journal of the American Statistical Association, 112(518):859–877, 2017.

Bremner, D. Incremental convex hull algorithms are not output sensitive. Discrete & Computational Geometry, 21(1):57–68, 1999.

Byravan, A. and Fox, D. SE3-nets: Learning rigid body motion using deep neural networks. In 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017.

Dyer, M. E. The complexity of vertex enumeration methods. Mathematics of Operations Research, 8(3):381–402, 1983.

Fukuda, K. and Prodon, A. Double description method revisited, pp. 91–111. Combinatorics and Computer Science: 8th Franco-Japanese and 4th Franco-Chinese Conference Brest, France, July 3–5, 1995 Selected Papers. Springer Berlin Heidelberg, 1996.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In Advances in Neural Information Processing Systems 27 (NIPS 2014), pp. 2672–2680. 2014.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. International Conference on Learning Representations (ICLR 2015), 2015.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In International Conference on Learning Representations (ICLR 2014), 2014.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference of Neural Information Processing Systems (NIPS 2012), 2012.

LeCun, Y., Cortes, C., and Burges, C. The MNIST database of handwritten digits. URL http://yann.lecun.com/exdb/mnist/.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. Nature, 521(7553):436–444, 2015.

Márquez-Neila, P., Salzmann, M., and Fua, P. Imposing hard constraints on deep networks: Promises and limitations. First Workshop on Negative Results in Computer Vision, CVPR 2017, 2017.

Motzkin, T. S., Raiffa, H., Thompson, G. L., and Thrall, R. M. The double description method. In Contributions to the Theory of Games II, volume 8 of Ann. of Math. Stud., pp. 51–73. Princeton University Press, 1953.

Nocedal, J. and Wright, S. Numerical Optimization. Springer Series in Operations Research and Financial Engineering. Springer, 2006.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in PyTorch. Autodiff Workshop, NIPS 2017, 2017.

Pathak, D., Krähenbühl, P., and Darrell, T. Constrained Convolutional Neural Networks for Weakly Supervised Segmentation. In International Conference on Computer Vision (ICCV 2015), 2015.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to Sequence Learning with Neural Networks. In Proceedings of the 27th International Conference of Neural Information Processing Systems (NIPS 2014), 2014.

Zhou, X., Sun, X., Zhang, W., Liang, S., and Wei, Y. Deep Kinematic Pose Regression. Workshop on Geometry Meets Deep Learning. ECCV 2016, 2016.