Sample-Specific Output Constraints for Neural Networks

Mathis Brosowsky [1,2], **Florian Keck** [2], **Olaf Dünkel** [2], **Marius Zöllner** [1,2]
[1]FZI Research Center for Information Technology,
[2]Karlsruhe Institute of Technology,
{brosowsky, zoellner}@fzi.de, {florian.keck, olaf.duenkel}@student.kit.edu

Abstract

It is common practice to constrain the output space of a neural network with the final layer to a problem-specific value range. However, for many tasks it is desired to restrict the output space for each input independently to a different subdomain with a non-trivial geometry, *e.g.* in safety-critical applications, to exclude hazardous outputs sample-wise. We propose ConstraintNet—a scalable neural network architecture which constrains the output space in each forward pass independently. Contrary to prior approaches, which perform a projection in the final layer, ConstraintNet applies an input-dependent parametrization of the constrained output space. Thereby, the complete interior of the constrained region is covered and computational costs are reduced significantly. For constraints in form of convex polytopes, we leverage the vertex representation to specify the parametrization. The second modification consists of adding an auxiliary input in form of a tensor description of the constraint to enable the handling of multiple constraints for the same sample. Finally, ConstraintNet is end-to-end trainable with almost no overhead in the forward and backward pass. We demonstrate ConstraintNet on two regression tasks: First, we modify a CNN and construct several constraints for facial landmark detection tasks. Second, we demonstrate the application to a follow object controller for vehicles and accomplish safe reinforcement learning in this case. In both experiments, ConstraintNet improves performance and we conclude that our approach is promising for applying neural networks in safety-critical environments.

Introduction

Deep neural networks (NNs) have become state-of-the-art in many competitive learning challenges. Crucial for this success is the learning of complex non-linear relationships implicitly from data. However, frequently additional domain knowledge exists in form of explicit relationships, *e.g.* symmetry conditions for human pose estimation (Márquez-Neila, Salzmann, and Fua 2017) or identified sets of safe outputs (Gros, Zanon, and Bemporad 2020). Imposing such relations as constraints on NNs is promising to achieve safety (Gros, Zanon, and Bemporad 2020), to reduce the black-box character (Cui et al. 2020), to stabilize the training (Lezcano-Casado and Martínez-Rubio 2019), to upgrade the performance, or to improve the data efficiency (Li and Srikumar 2019).

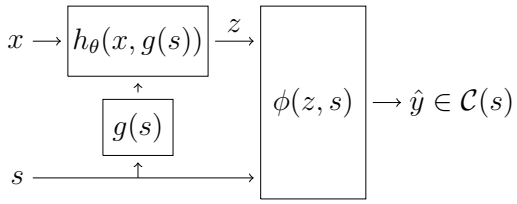

Figure 1: Approach to construct ConstraintNet for a class of constraints $\mathfrak{C} = \{\mathcal{C}(s) \subset \mathcal{Y} | s \in \mathcal{S}\}$. A final layer ϕ without learnable parameters maps the output of previous layers $z = h_\theta(x, g(s))$ on the constrained output space $\mathcal{C}(s)$ depending on the constraint parameter s. The previous layers h_θ get a representation $g(s)$ of s as an additional input to the data point x. This enables ConstraintNet to deal with different constraints for the same x.

We focus on embedding sample-specific output constraints in the NN architecture. Prior research is known from deep reinforcement learning with safe action sets (Dalal et al. 2018). These approaches perform a projection on the constrained region in the final layer by solving an optimization problem. However, the projection layer maps output violations only on the boundary of the constrained region and has a significant computational overhead. With ConstraintNet, we propose an NN architecture which overcomes this limitation and requires almost no additional computational costs.

Instead of performing a projection, ConstraintNet applies an input-dependent parametrization of the constrained output space in the final layer, the so-called constraint guard layer (see Figure 1). Thereby, less computational costs are required and the complete interior of the constrained region is covered. The second crucial design choice of ConstraintNet consists of an auxiliary input in form of a tensor description of the chosen constraint. Thereby, we are to the best of our knowledge the first proposing a method capable to handle multiple constraints for the same input. This is beneficial for many applications. *E.g.* constraints in form of bounding boxes for a facial landmark detection do not need to be centered perfectly and may vary in size and shape. Altogether, the constraint guard layer encodes a precise description of a class of constraints in the NN architecture, *e.g.* convex poly-

tope with five vertices, and a specific constraint from this class can be chosen via the additional input in each forward pass independently.

ConstraintNet addresses safety-critical applications in particular. Instead of monitoring the output of the NN with a second algorithm and intervening when safety-critical behavior is detected, the constraints restrict the output to safe solutions in the first place. *E.g.* in an NN-based trajectory planner constraints can be leveraged for collision avoidance. Apart from safety-critical applications, output constraints are applicable wherever a partition into valid and invalid output domains is given by an external source, *e.g.* by a human expert, map data, a rule-based model, or even a second NN. For the NN-based trajectory planner, further output constraints can be applied to ensure consideration of navigation instructions. In medical image processing, the constrained region can be annotated by a human expert to restrict the localization of an anatomical landmark.

We evaluate ConstraintNet on a facial landmark detection task with constraints in form of bounding boxes covering the face. Further, we analyze relative constraints, *e.g.* that the positions of the eyes are above the nose-landmark for anatomical reasons. Furthermore, we apply ConstraintNet on a follow object controller for vehicles and leverage output constraints for safe deep reinforcement learning. Our main contributions are:

- leveraging an input-dependent parametrization of the constrained output space for imposing hard output constraints,
- proposing a compact parametrization for convex polytopes based on the vertex representation,
- setting the output constraint in each forward pass independently via specifying a tensor description of the constraint (multiple constraints for the same sample applicable),
- clear improvements in runtime over performing a projection while upgrading the performance.

Related Work

An ongoing challenge in machine learning is to combine the implicit and data-driven learning of patterns with explicit and a priori known relations. However, in certain learning tasks the incorporation of domain knowledge is crucial for the consistency, interpretability, stability, and performance of the application as well as the trust into the output. *E.g.* a traditional facial landmark detector aims to predict landmarks consistently to a global shape pattern for robustness against occlusion, illumination, hairstyle, and accessoires. The frequently used constrained local models refine independent local appearance information with a global shape pattern (Zadeh, Baltrušaitis, and Morency 2017). The first step can be performed *e.g.* with an NN and the second step can be considered as an advanced post-processing and optimization step. However, it would be advantageous to solve this end-to-end in a single forward pass.

In recent years, we observe increasing attention in research regarding imposing constraints on NNs. This allows to incorporate a priori known and intended relations. The contributions are spread over a variety of applications and deal with different types of constraints. We identify three main categories of methods.

Methods of the first category add a loss term to penalize constraint violations. This is also known as soft constraints, since the NN is only encouraged to satisfy constraints. However, constraint violations might still occur. In Karpatne et al. (2017), physical relationships are incorporated by such a loss term to improve a lake temperature model. A second group consists of approaches that modify the optimizer in training. In Márquez-Neila, Salzmann, and Fua (2017), constraints are included by solving in each training step a linearized version of the Lagrangian dual problem. In their paper, the approach is evaluated on a human pose estimation task with symmetry conditions. Methods of the third category ensure constraint satisfaction by construction of the NN architecture. In Li and Srikumar (2019), an approach is proposed to impose logical statements on neurons with assigned semantics. A manually designed distance function is added to the pre-activation score to realize logical expressions. In the following, we show further methods of the third category and relate them to ConstraintNet.

A central design choice for imposing constraints is to apply a parametrization. In this way, the output or an intermediate variable of the NN can be constrained to a problem-specific value range. A simple example is the softmax function which generates only valid parameters of the categorical distribution. However, more complex parametrizations can be constructed. In Lezcano-Casado and Martínez-Rubio (2019), the Lie-algebra and the exponential map is leveraged to parametrize the special orthogonal group. Thereby, the kernel matrices of recurrent NNs are constrained and the vanishing and exploding gradient problem can be reduced. In Cui et al. (2020), a vehicle model is leveraged to generate a parametrization of only kinematically feasible trajectories. However, all these approaches parametrize a globally fixed subspace $\mathcal{C} = \{\phi(z) | z \in \mathcal{Z}\}$. To the best of our knowledge, we are the first leveraging input-dependent parametrizations for adjustable output constraints $\mathcal{C}(s)$ in NNs:

$$\mathcal{C}(s) = \{\phi(z, s) | z \in \mathcal{Z}\}, \tag{1}$$

with s being a vector description of the constraint.

As we already mentioned in the introduction, approaches from deep reinforcement learning (Dalal et al. 2018; Gros, Zanon, and Bemporad 2020) propose to realize output constraints by projecting the output of the NN n_θ on a safe set $\mathcal{C}(s)$, *i.e.* the element with the minimal distance:

$$n_{\perp,\theta}(x) = \underset{y \in \mathcal{C}(s)}{\arg\min} \frac{1}{2} \|y - n_\theta(x)\|^2. \tag{2}$$

The projection can be considered as an optimization problem and the more generally studied differentiable optimization layers (Agrawal et al. 2019; Amos and Kolter 2017) are applicable. However, solving the optimization problem requires additional computational overhead. In the experiments section, we compare ConstraintNet with this projection-based approach.

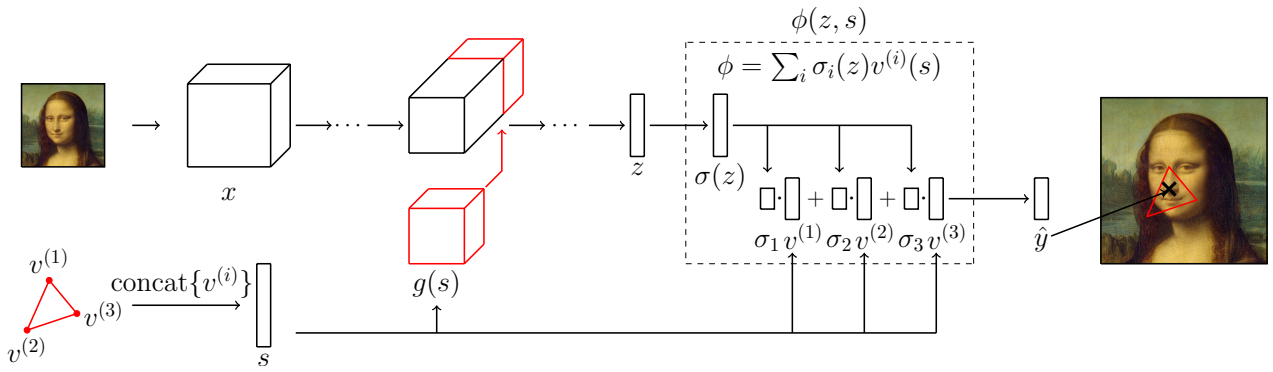

Figure 2: Construction of ConstraintNet by extending a CNN. For illustration purposes, we show a nose landmark detection on an image x with an output constraint in form of a triangle, *i.e.* a convex polytope with three vertices $\{v^{(i)}(s)\}_{i=1}^{3}$. The constraint parameter s specifies the chosen constraint and in this case consists of concatenated vertex coordinates. A tensor representation $g(s)$ of s is concatenated to the output of an intermediate convolutional layer and extends the input of the next layer. Instead of creating the final output for the nose landmark with a 2-dimensional dense layer, a 3-dimensional intermediate representation z is generated. The constraint guard layer ϕ applies a softmax function σ on z and weights the three vertices of the triangle with the softmax outputs. This guarantees a detection $\hat{y}$ within the specified triangle.

Neural Networks with Sample-Specific Output Constraints

This section is structured as follows: (1) First of all, we define sample-specific output constraints and ConstraintNet formally. (2) Next, we propose our approach to create the architecture of ConstraintNet. This approach requires a specific layer without learnable parameters for the considered class of constraints, the constraint guard layer. (3) We model this constraint guard layer for constraints in the form of convex polytopes and sectors of a circle. Furthermore, we derive the layer for constraints on different output parts. (4) For training ConstraintNet, we propose to sample specific constraints from valid sets, then standard stochastic gradient descent algorithms are applicable. Finally, (5) we present the supported constraint types and possible generalizations to give an idea about the broad applicability.

Sample-Specific Output Constraints

Consider an NN $n_\theta : \mathcal{X} \to \mathcal{Y}$ with learnable parameters $\theta \in \Theta$, input space $\mathcal{X}$ and output space $\mathcal{Y}$. We introduce an output constraint as a subset of the output space $\mathcal{C} \subset \mathcal{Y}$ and a class of output constraints as a parametrized set of them $\mathfrak{C} = \{\mathcal{C}(s) \subset \mathcal{Y} : s \in \mathcal{S}\}$. $\mathcal{S}$ is a set of parameters and we call an element $s \in \mathcal{S}$ constraint parameter. We define ConstraintNet as an NN $f_\theta : \mathcal{X} \times \mathcal{S} \to \mathcal{Y}$ with the constraint parameter $s \in \mathcal{S}$ as an additional input and the guarantee to predict within $\mathcal{C}(s)$ by design of the NN architecture, *i.e.* independently of the learned weights θ:

$$\forall \theta \in \Theta \; \forall s \in \mathcal{S} \; \forall x \in \mathcal{X} : f_\theta(x, s) \in \mathcal{C}(s). \tag{3}$$

Furthermore, we require that f_θ is (piecewise) differentiable w.r.t. θ so that backpropagation and gradient-based optimization algorithms are amenable.

Network Architecture

Construction Approach. We propose the approach visualized in Figure 1 to create the architecture of ConstraintNet for a specific class of constraints $\mathfrak{C}$. The key idea is a final layer $\phi : \mathcal{Z} \times \mathcal{S} \to \mathcal{Y}$ without learnable parameters which maps the output of the previous layers $z \in \mathcal{Z}$ on the constrained output space $\mathcal{C}(s)$ depending on the constraint parameter s. Given a class of constraints $\mathfrak{C} = \{\mathcal{C}(s) \subset \mathcal{Y} : s \in \mathcal{S}\}$, we require that ϕ fulfills:

$$\forall s \in \mathcal{S} \; \forall z \in \mathcal{Z} : \phi(z, s) \in \mathcal{C}(s). \tag{4}$$

When ϕ is furthermore (piecewise) differentiable w.r.t. z we call ϕ constraint guard layer for $\mathfrak{C}$. The constraint guard layer ϕ has no adjustable parameters and therefore the logic is learned by the previous layers h_θ with parameters θ. In the ideal case, ConstraintNet predicts the same true output y for a data point x under different but valid constraints. This behavior requires that h_θ depends on s in addition to x. Without this requirement, $z = h_\theta(\cdot)$ would have the same value for fixed x, and ϕ would project this z for different but valid constraint parameters s to different outputs in general. We transform s into an appropriate representation $g(s)$ and consider it as an additional input of h_θ, *i.e.* $h_\theta : \mathcal{X} \times g(\mathcal{S}) \to \mathcal{Z}$. For the construction of h_θ, we propose to start with a common NN architecture with input domain $\mathcal{X}$ and output domain $\mathcal{Z}$. In a next step, this NN is extended by adding an input for $g(s)$. We propose to concatenate $g(s)$ to the output of an intermediate layer since it provides information with a high level of abstraction. Finally, we construct ConstraintNet for the considered class of constraints $\mathfrak{C}$ by applying the layers h_θ and the corresponding constraint guard layer ϕ subsequently:

$$f_\theta(x, s) = \phi\big(h_\theta(x, g(s)), s\big). \tag{5}$$

The required property for ϕ in Eq. (4) implies that ConstraintNet predicts within the constrained output space $\mathcal{C}(s)$ according to Eq. (3). Furthermore, the constraint guard layer propagates gradients and backpropagation is applicable.

Construction by Modifying a CNN. Figure 2 illustrates the construction of ConstraintNet by using a convolutional NN (CNN) for the generation of the intermediate variable $z = h_\theta(x, g(s))$, where h_θ is a CNN. As an example, a nose landmark detection task on face images is shown. The output constraints are triangles randomly located around the nose, *i.e.* convex polytopes with three vertices. Such constraints can be specified by a constraint parameter s consisting of the concatenated vertex coordinates. The constraint guard layer ϕ for convex polytopes is modeled in the next section and requires a 3-dimensional intermediate variable $z \in \mathbb{R}^3$ for triangles. The previous layers h_θ map the image data $x \in \mathcal{X}$ on the 3-dimensional intermediate variable $z \in \mathbb{R}^3$. A CNN with output domain $\mathcal{Z} = \mathbb{R}^N$ can be realized by adding a dense layer with N output neurons and linear activations. To incorporate the dependency of h_θ on s, we suggest to concatenate the output of an intermediate convolutional layer by a tensor representation $g(s)$ of s. Thereby, we extend the input of the next layer in a natural way.

Constraint Guard Layer for Different Classes of Constraints

Convex Polytopes. We consider convex polytopes $\mathcal{P}$ in $\mathbb{R}^N$ which can be described by the convex hull of M vertices $\{v^{(i)}\}_{i=1}^M$ of dimension N:

$$\mathcal{P}\big(\{v^{(i)}\}_{i=1}^M\big) = \big\{\sum_i p_i v^{(i)} : p_i \geq 0,\ \sum_i p_i = 1\big\}. \quad (6)$$

We assume that the vertices $v^{(i)}(s)$ are functions of the constraint parameter s and define output constraints via $\mathcal{C}(s) = \mathcal{P}(\{v^{(i)}(s)\}_{i=1}^M)$. The constraint guard layer for a class of these constraints $\mathfrak{C} = \{\mathcal{C}(s) : s \in \mathcal{S}\}$ can be constructed with $z \in \mathbb{R}^M$:

$$\phi(z, s) = \sum_i \sigma_i(z) v^{(i)}(s). \quad (7)$$

$\sigma_i(\cdot)$ denotes the ith component of the M-dimensional softmax function $\sigma : \mathbb{R}^M \to \mathbb{R}^M$. The required property of ϕ in Eq. (4) follows directly from the properties $0 < \sigma_i(\cdot) < 1$ and $\sum_i \sigma_i(\cdot) = 1$ of the softmax function. However, the boundary of the convex polytope is not reachable exactly but up to arbitrary accuracy because $\sigma_i(\cdot) \neq 1$. Note that ϕ is differentiable w.r.t. z.

Sectors of a Circle. Consider a sector of a circle $\mathcal{O}$ with center position $(x_\mathrm{c}, y_\mathrm{c})$ and radius R. We assume that the sector is symmetric w.r.t. the vertical line $x = x_\mathrm{c}$ and covers Ψ radian. Then, the sector of a circle can be described by the following set of points:

$$\mathcal{O}(x_\mathrm{c}, y_\mathrm{c}, R, \Psi) = \big\{r \cdot (\sin\varphi, \cos\varphi) + (x_\mathrm{c}, y_\mathrm{c}) \in \mathbb{R}^2 : r \in [0, R], \varphi \in [-\Psi/2, +\Psi/2]\big\}. \quad (8)$$

With $s = (x_\mathrm{c}, y_\mathrm{c}, R, \Psi)$, the output constraint can be written as $\mathcal{C}(s) = \mathcal{O}(x_\mathrm{c}, y_\mathrm{c}, R, \Psi)$. The following constraint guard layer with an intermediate variable $z \in \mathbb{R}^2$ fulfills Eq. (4) for a class of these constraints $\mathfrak{C} = \{\mathcal{C}(s) : s \in \mathcal{S}\}$:

Algorithm 1 Training algorithm for ConstraintNet. The constraint parameter s_i for a data point (x_i, y_i) is sampled from a set of valid parameters $\mathcal{S}_{y_i} = \{s \in \mathcal{S} : y_i \in \mathcal{C}(s)\}$.

procedure TRAIN($\{x_i, y_i\}_{i=1}^N$)
  $\theta \leftarrow$ random initialization
  for batch **do**
    $I_\text{batch} \leftarrow$ get_indices(batch)
    for $i \in I_\text{batch}$ **do**
      $s_i \leftarrow$ sample($\mathcal{S}_{y_i}$)
      $\hat{y}_i \leftarrow f_\theta(x_i, s_i)$ ▷ ConstraintNet
    end for
    $L(\theta) \leftarrow \frac{1}{|I_\text{batch}|} \sum_{i \in I_\text{batch}} l(y_i, \hat{y}_i) + \lambda R(\theta)$
    $\theta \leftarrow$ update($\theta, \nabla_\theta L$)
  end for
  return θ
end procedure

$$\phi(z, s) = r(z_1) \cdot \big(\sin\varphi(z_2), \cos\varphi(z_2)\big) + (x_\mathrm{c}, y_\mathrm{c}), \quad (9)$$

with $r(z_1) = R \cdot \mathrm{sig}(z_1)$ and $\varphi(z_2) = \Psi \cdot (\mathrm{sig}(z_2) - 0.5)$. Note that we use the sigmoid function $\mathrm{sig}(t) = 1/(1 + \exp(-t))$ to map a real number to the interval $(0, 1)$.

Constraints on Output Parts. We consider an output $y = (y^{(1)}, \ldots, y^{(K)}) \in \mathcal{Y} = \mathcal{Y}^{(1)} \times \cdots \times \mathcal{Y}^{(K)}$ with K parts $y^{(k)}$ ($k \in \{1, \ldots, K\}$). Each output part $y^{(k)}$ should be constrained independently to an output constraint $\mathcal{C}^{(k)}(s^{(k)})$ of a part-specific class of constraints $\mathfrak{C}^{(k)} = \{\mathcal{C}^{(k)}(s^{(k)}) \subset \mathcal{Y}^{(k)} : s^{(k)} \in \mathcal{S}^{(k)}\}$. This is equivalent to constrain the overall output y to $\mathcal{C}(s) = \mathcal{C}^{(1)}(s^{(1)}) \times \cdots \times \mathcal{C}^{(K)}(s^{(K)})$ with $s = (s^{(1)}, \ldots, s^{(K)})$. The class of constraints for the overall output is then given by $\mathfrak{C} = \{\mathcal{C}(s) \subset \mathcal{Y} : s \in \mathcal{S}^{(1)} \times \cdots \times \mathcal{S}^{(K)}\}$. Assume that the constraint guard layers for the parts $\phi^{(k)}$ are given, *i.e.* for $\mathfrak{C}^{(k)}$. Then, an overall constraint guard layer ϕ, *i.e.* for $\mathfrak{C}$, can be constructed by concatenating the constraint guard layers of the parts:

$$\phi(z, s) = \big(\phi^{(1)}(z^{(1)}, s^{(1)}), \ldots, \phi^{(K)}(z^{(K)}, s^{(K)})\big), \quad (10)$$

with $z = (z^{(1)}, \ldots, z^{(K)})$. The validity of the property in Eq. (4) for ϕ w.r.t. $\mathfrak{C}$ follows immediately from the validity of this property for $\phi^{(k)}$ w.r.t. $\mathfrak{C}^{(k)}$.

Training

In supervised learning, the parameters θ of an NN are learned from data by utilizing a set of input-output pairs $\{(x_i, y_i)\}_{i=1}^N$. However, ConstraintNet has an additional input $s \in \mathcal{S}$ which is not unique for a sample. The constraint parameter s provides information in form of a region restricting the true output. Therefore, the constraint parameter s_i for a sample (x_i, y_i) could be any element of a set of valid constraint parameters $\mathcal{S}_{y_i} = \{s \in \mathcal{S} : y_i \in \mathcal{C}(s)\}$. We propose to sample s_i from this set $\mathcal{S}_{y_i}$ to create representative input-output pairs (x_i, s_i, y_i). This sampling procedure enables ConstraintNet to be trained with standard gradient descent

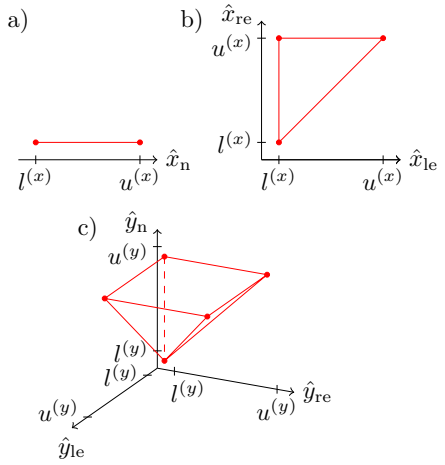

Figure 3: Combining the bounding box constraints with relative relations between landmarks is equivalent to constraining the output parts $\hat{y}^{(1)} = \hat{x}_{\text{n}}$ to the line segment in a), $\hat{y}^{(2)} = (\hat{x}_{\text{le}}, \hat{x}_{\text{re}})$ to the triangle in b), and $\hat{y}^{(3)} = (\hat{y}_{\text{n}}, \hat{y}_{\text{le}}, \hat{y}_{\text{re}})$ to the pyramid in c).

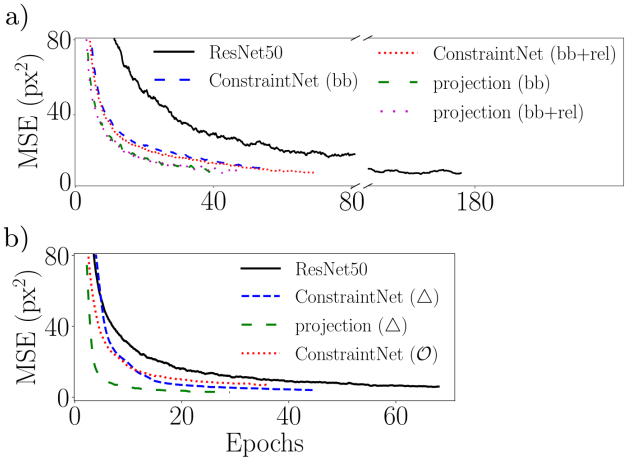

Figure 4: Learning curves for ConstraintNet, the projection-based approach and ResNet50. The plots show the mean squared error as a function of the training progress up to the point when overfitting begins. a) For bounding box (bb) constraints and additional enforcing of relations between landmarks (bb+rel). b) For triangle (△) and sector of a circle constraints ($\mathcal{O}$).

algorithms for NNs as shown in Algorithm 1. Note that many input-output pairs can be generated for the same data point (x_i, y_i) by sampling different constraint parameters s_i. Therefore, ConstraintNet is forced to learn an invariant prediction for the same sample under different constraint parameters.

Supported Constraints and Generalizations

In general, an output constraint $\mathcal{C}$ is applicable if a differentiable parametric equation ϕ exists for $\mathcal{C}$. Consequently, the presented constraints are exemplary and our goal is to encourage the construction of own constraint guard layers depending on the problem at hand. The following ideas should give a glimpse of ConstraintNet's broad applicability.

- ConstraintNet can be generalized to classification tasks by constraining the logit space.
- To ensure the satisfaction of each constraint of a set $\{\mathcal{C}_i\}_{i \in I}$, we propose to parametrize the intersection $\mathcal{C} = \bigcap_{i \in I} \mathcal{C}_i$. *E.g.* in the following of the paper, in the facial landmark detection task we combine bounding box constraints with relative constraints in this way (compare Eqs. (11, 12, 13)).
- To ensure the satisfaction of at least one constraint of a set $\{\mathcal{C}_i\}_{i \in I}$, we propose to predict within each $\mathcal{C}_i$. For the decision whether $\mathcal{C}_i$ covers the ground truth, associated probabilities can be predicted with an additional softmax layer. *E.g.* then, non-convex polytopes are feasible by partitioning the polytope into a triangle (convex polytope) mesh.
- ConstraintNet is also capable to deal with unbounded regions. *E.g.* for predicting $\hat{y} \in \mathbb{R}$ above a threshold b, we propose the constraint guard layer $\phi(z, s) = \exp(z) + b$ with the constraint parameter $s = b$ given by the threshold.

Experiments

In this section, ConstraintNet is applied to facial landmark detection tasks and a follow object controller for vehicles. The output constraints for the facial landmark detection tasks restrict the solution space to consistent outputs, whereas the constraints for the follow object controller are modeled to improve safety.

Consistent Facial Landmark Detection[1]

In this experiment, we consider a facial landmark detection on images of the *Large-scale CelebFaces Attributes (CelebA) dataset* (Liu et al. 2015) and evaluate ConstraintNet for different classes of constraints. Each image of the *CelebA* dataset pictures one face.

Modified ResNet50 Architecture. We construct ConstraintNet by modifying ResNet50 (He et al. 2016) according to Figure 2 and the approach described in the previous section. The modifications comprise adapting the output dimension of the final dense layer with linear activations to match the required dimension of z, adding the constraint guard layer ϕ for the considered class of constraints $\mathfrak{C}$, and concatenating the representation $g(s)$ of the constraint parameter s to the output of an intermediate layer. As intermediate layer, we choose the 22nd layer which is roughly in the middle of ConstraintNet and followed by further 27 convolutional layers. It is known that the first convolutional layers recognize image features with a low-level of abstraction, *e.g.* edges. The idea is to add $g(s)$ later when already high-level abstractions are extracted. However, it would be interesting to analyze if optimizing the layer for $g(s)$ would improve performance further. We define $g(s)$ as tensor and identify channels $c \in \{1, \dots, \dim(s)\}$

[1] https://github.com/mbroso/constraintnet_facial_detect

Method (Constraint)	MSE($\hat{x}_\mathrm{n}$)	MSE($\hat{y}_\mathrm{n}$)	MSE($\hat{x}_\mathrm{le}$)	MSE($\hat{y}_\mathrm{le}$)	MSE($\hat{x}_\mathrm{re}$)	MSE($\hat{y}_\mathrm{re}$)	MSE($\hat{y}$)	Runtime
ConstraintNet (bb)	2.17 ± 0.04	$\mathbf{1.63\pm0.05}$	1.88 ± 0.05	1.92 ± 0.06	2.06 ± 0.06	1.57 ± 0.04	11.23 ± 0.16	(33.2 ± 0.2) s
projection (bb)	1.84 ± 0.06	1.71 ± 0.10	2.00 ± 0.10	2.03 ± 0.08	2.93 ± 0.09	1.48 ± 0.06	11.99 ± 0.35	(64.8 ± 0.5) s
ConstraintNet (bb+rel)	$\mathbf{1.43\pm0.05}$	1.76 ± 0.05	$\mathbf{1.80\pm0.04}$	$\mathbf{1.62\pm0.06}$	$\mathbf{1.69\pm0.04}$	$\mathbf{1.39\pm0.03}$	$\mathbf{9.70\pm0.10}$	(33.7 ± 0.4) s
projection (bb+rel)	1.62 ± 0.06	1.95 ± 0.09	1.78 ± 0.06	1.74 ± 0.08	1.91 ± 0.05	1.43 ± 0.06	10.44 ± 0.25	(67.5 ± 0.8) s
ResNet50 (None)	3.28 ± 0.76	2.40 ± 0.34	3.78 ± 0.80	2.20 ± 0.22	3.82 ± 0.78	1.93 ± 0.27	17.42 ± 2.89	(32.9 ± 0.2) s
ConstraintNet ($\triangle$)	$\mathbf{1.44\pm0.03}$	$\mathbf{1.59\pm0.04}$	–	–	–	–	$\mathbf{3.03\pm0.05}$	(34.2 ± 0.7) s
projection ($\triangle$)	1.63 ± 0.04	1.66 ± 0.06	–	–	–	–	3.30 ± 0.08	(295.5 ± 0.8) s
ConstraintNet ($\mathcal{O}$)	2.17 ± 0.05	4.15 ± 0.14	–	–	–	–	6.33 ± 0.15	(33.4 ± 0.4) s
ResNet50 (None)	4.24 ± 0.53	3.78 ± 0.37	–	–	–	–	8.02 ± 0.67	(32.9 ± 0.2) s

Table 1: Mean squared error (MSE) for facial landmark detection with ConstraintNet, ResNet50 with projection layer, and pure ResNet50 on test set of *CelebA* dataset. Results are shown for constraints in form of bounding boxes (bb), additional constraints to enforce a relative arrangement (bb+rel), triangle ($\triangle$), and sector of a circle constraints ($\mathcal{O}$). The average covered area of the triangle constraints is $(2345 \pm 6)\,\mathrm{px}^2$ and the area of the sector of the circle constraints $(2040 \pm 8)\,\mathrm{px}^2$. Further, the runtimes for performing predictions on all instances of the test set of the CelebA dataset are shown. The times are measured on a Lambda Quad (CPU: 12x Intel Core i7-6850K 3.6 GHz, GPU: 4x Nvidia 12 GB Titan V, RAM: 128 GB) using one of the four dedicated graphics cards. All average values and the standard deviations are evaluated over 30 test runs.

with the components of the constraint parameter s, then we set all entries within a channel to a rescaled value $\lambda_c \cdot s_c$ of the corresponding constraint parameter component.

Constraints. We introduce constraints to confine the landmark detections for the nose $(\hat{x}_\mathrm{n}, \hat{y}_\mathrm{n})$, the left eye $(\hat{x}_\mathrm{le}, \hat{y}_\mathrm{le})$, and the right eye $(\hat{x}_\mathrm{re}, \hat{y}_\mathrm{re})$ to a bounding box which might be given by a face detector. The bounding box is specified by a left $l^{(x)}$, a right $u^{(x)}$, a top $l^{(y)}$, and a bottom $u^{(y)}$ boundary. Note that the y-axis starts at the top of the image and points downwards. Confining the landmark detections to a bounding box is equivalent to constrain $\hat{x}_\mathrm{n}, \hat{x}_\mathrm{le}, \hat{x}_\mathrm{re}$ to the interval $[l^{(x)}, u^{(x)}]$ and $\hat{y}_\mathrm{n}, \hat{y}_\mathrm{le}, \hat{y}_\mathrm{re}$ to the interval $[l^{(y)}, u^{(y)}]$ independently. These intervals are 1-dimensional convex polytopes with the interval boundaries as vertices. Thus, the constraint guard layers for the components are given by Eq. (7) and the overall constraint guard layer $\phi(z, s)$ can be constructed according to Eq. (10).

We extend the bounding box constraints to model relations between landmarks. As an example, we enforce that the left eye is in fact to the left of the right eye ($\hat{x}_\mathrm{le} \leq \hat{x}_\mathrm{re}$) and that the eyes are above the nose ($\hat{y}_\mathrm{le}, \hat{y}_\mathrm{re} \leq \hat{y}_\mathrm{n}$). To combine the bounding box constraints $\mathcal{C}_\mathrm{bb}$ with the relative constraints $\mathcal{C}_\mathrm{rel}$, we parametrize the intersection of both constraints $\mathcal{C}_\mathrm{bb} \cap \mathcal{C}_\mathrm{rel}$. This intersection can be written as three independent constraints for the output parts $\hat{y}^{(1)} = \hat{x}_\mathrm{n}$, $\hat{y}^{(2)} = (\hat{x}_\mathrm{le}, \hat{x}_\mathrm{re})$, $\hat{y}^{(3)} = (\hat{y}_\mathrm{n}, \hat{y}_\mathrm{le}, \hat{y}_\mathrm{re})$:

$$\mathcal{C}^{(1)}(s^{(1)}) = \{\hat{x}_\mathrm{n} \in \mathbb{R} : l^{(x)} \leq \hat{x}_\mathrm{n} \leq u^{(x)}\}, \tag{11}$$

$$\mathcal{C}^{(2)}(s^{(2)}) = \{(\hat{x}_\mathrm{le}, \hat{x}_\mathrm{re}) \in \mathbb{R}^2 : \hat{x}_\mathrm{le} \leq \hat{x}_\mathrm{re}, \; l^{(x)} \leq \hat{x}_\mathrm{le}, \hat{x}_\mathrm{re} \leq u^{(x)}\}, \tag{12}$$

$$\mathcal{C}^{(3)}(s^{(3)}) = \{(\hat{y}_\mathrm{n}, \hat{y}_\mathrm{le}, \hat{y}_\mathrm{re}) \in \mathbb{R}^3 : \hat{y}_\mathrm{le}, \hat{y}_\mathrm{re} \leq \hat{y}_\mathrm{n}, \; l^{(y)} \leq \hat{y}_\mathrm{n}, \hat{y}_\mathrm{le}, \hat{y}_\mathrm{re} \leq u^{(y)}\}, \tag{13}$$

with constraint parameters $s^{(1)} = s^{(2)} = (l^{(x)}, u^{(x)})$ and $s^{(3)} = (l^{(y)}, u^{(y)})$. $\{\mathcal{C}^{(k)}\}_{k=1}^3$ are convex polytopes and visualized in Figure 3. Thus, the constraint guard layers for the parts and the total one are given by Eq. (7) and Eq. (10), respectively. In other words, for each output part a softmax layer is applied in parallel to generate the weights for the vertices of the convex polytopes in Figure 3. The first softmax is of dimension two, the second one of dimension three, and the third one of dimension five. Finally, each of the three output parts is generated by an average weighting of the vertices of the polytopes with the generated softmax probabilities.

Furthermore and for demonstration purposes, we only detect the nose landmark and constrain the detection to a triangle or a sector of a circle. The constraint guard layers for triangles and sectors of a circle are given by Eq. (7) and Eq. (9), respectively. Figure 2 visualizes ConstraintNet for triangle constraints.

Training and Results. For a fair comparison, we create a baseline by substituting the constraint guard layer of ConstraintNet with a differentiable optimization layer in form of a projection (Agrawal et al. 2019). The projection layer outputs an element within the constrained region with the shortest Euclidean distance to its input according to Eq. (2). Analogous to ConstraintNet, the projection-based approach gets $g(s)$ as an additional input and is trained end-to-end. Further, we compare to ResNet50 without constraints. Figure 4 shows the learning curves of ConstraintNet and the projection-based approach for the different output constraints. For further comparison, the learning curve of pure ResNet50 is depicted which is trained for the same task without constraints. ConstraintNet and the projection-based approach are trained according to Algorithm 1 with randomly generated bounding boxes, triangles, and sectors of a circle. We observe that ConstraintNet and the projection-based approach converge significantly faster than the ResNet50 baseline. This can be explained by the added domain knowledge in form of output constraints. The projection-based approach requires even fewer epochs than ConstraintNet. However, the total training time is higher due to longer runtimes per epoch. The

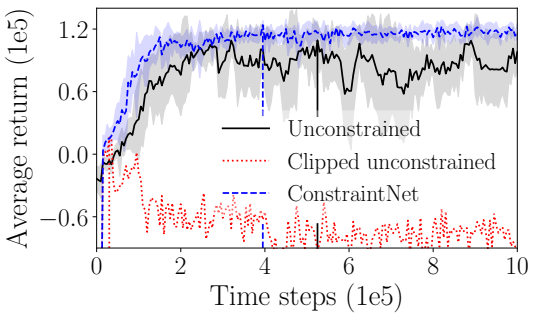

Figure 5: Learning curves for ConstraintNet and (clipped) unconstrained FOC. Bold lines show the average episode return over six trials, shaded regions the standard deviation. The maximum returns (vertical lines) are reached at time step 0.3e5, 3.95e5, and 5.25e5.

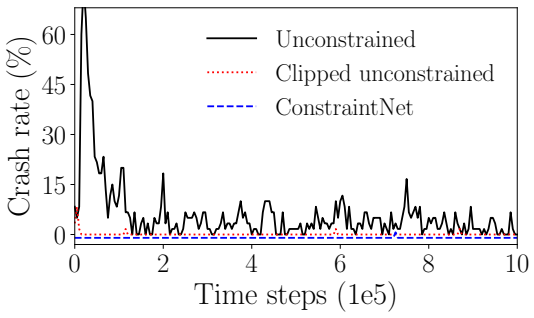

Figure 6: Average crash rate during training over six trials for ConstraintNet (slightly shifted for visualization) and (clipped) unconstrained FOC. The crash rate is defined as fraction of interrupted episodes with a maximum duration of five minutes due to a collision with the target vehicle.

exact runtimes are given in the supplementary material which is accessible on Github[2]. For an invariant prediction of the same input under varying constraints, ConstraintNet learns an intermediate representation z which depends on the constraint parameter s. Whereas this mechanism requires training the same input-output pairs under varying constraints, we have the following arguments for our design choice: (1) This technique enforces ConstraintNet to incorporate the constraint parameter. (2) Diverse decision paths for the same sample can support regularization (Opitz and Shavlik 1995). (3) Learning this additional relationship can be shared between all samples. (4) To reduce training time and to avoid permutations, in all experiments in the paper we defined an order on the vertices.

Besides training, we also apply the randomly generated constraints for evaluation on the test set. Table 1 shows the mean squared error on the test set for ConstraintNet and the projection-based approach with different output constraints. Further, results are shown for ResNet50 without constraints. ConstraintNet reaches for all considered constraints lower mean squared errors than the projection-based approach. The comparison to ResNet50 without constraints shows that both methods utilize the information encoded in the constraint. This is in accordance with the observation that the performance for constraints which enforce an additional relative arrangement improve over just bounding box constraints. In comparison to original ResNet50, ConstraintNet requires almost no additional time for predicting facial landmarks on the complete test set. Contrary, the projection-based approach solves an optimization problem for each instance and the runtimes are significantly higher.

Output Constraints on a Follow Object Controller for Safe Reinforcement Learning[3]

Adaptive cruise control (ACC) is a common driver assistance system for longitudinal control. The follow object controller (FOC) is part of the ACC and is activated when a vehicle appears ahead. We call the vehicle with active FOC ego vehicle and the vehicle ahead target vehicle. The output of the FOC is a demanded acceleration $a_{\text{ego,dem}}$ with the goal to keep a velocity-dependent distance to the target vehicle under consideration of comfort and safety aspects. We choose a desired distance corresponding to a time gap of 2 s with a modification to reach a distance of 3.2 m when stopping. The input x of the FOC consists of several sensor measurements: the distance to the target vehicle, the relative acceleration and velocity of the target vehicle and the acceleration and velocity of the ego vehicle. Improving the FOC explicitly based on expert knowledge and classical control theory leads to models with an increasing number of adjustable parameters. Adjusting these parameters is non-trivial. This motivates the idea to implement the FOC as an NN $a_{\text{ego,dem}} = n_\theta(x)$ and learn the parameters θ in a reinforcement learning setting (Desjardins and Chaib-draa 2011).

Modified Fully Connected Neural Network. Implementing the FOC with a common NN comes at the expense of losing safety guarantees. However, ConstraintNet $a_{\text{ego,dem}} = f_\theta(x, s)$ allows to confine the demanded acceleration $a_{\text{ego,dem}}$ to a safe interval $[a_{\min}, a_{\max}]$ in each forward pass independently. We utilize an NN with three fully connected layers to generate the intermediate variable $z \in \mathbb{R}^2$. Each layer consists of 256 neurons. The interval $[a_{\min}, a_{\max}]$ is an 1-dimensional convex polytope and the constraint guard layer is given by Eq. (7). *I.e.* a 2-dimensional softmax layer generates the weights for interpolating between $a_{\min}$ and $a_{\max}$. For the representation $g(s)$, we choose a simple normalization procedure. Instead of inserting $g(s)$ at an intermediate layer, we add it to the input due to the small size of the NN.

Constraints. The upper bound $a_{\max}$ restricts the acceleration to avoid collisions. For deriving $a_{\max}$, we extrapolate the trajectory of the target vehicle, consider a response time for the demanded acceleration, and require that a breaking maneuver without violating maximum jerk and deceleration bounds does not lead to an undershooting of a minimum distance to the target vehicle. The derivation of this bound is related to the Responsibility-Sensitive Safety model (Shalev-

[2]https://github.com/mbroso/constraintnet_facial_detect
[3]https://github.com/mbroso/constraintnet_foc

NN for policy	Time steps (1e5)	$M_{\text{CrashRate}}$	M_{Safety}	$M_{\text{Discomfort}}$	M_{TrackErr}
ConstraintNet	10.0	$\mathbf{0.0 \pm 0.0\,\%}$	$\mathbf{0.88 \pm 0.01}$	$\mathbf{0.61 \pm 0.04}$	$\mathbf{1.1 \pm 0.3}$
Unconstrained (clipped)	10.0 (10.0)	$0.7 \pm 0.9\%$ ($\mathbf{0.0\%}$)	0.81 ± 0.04 (0.85)	0.63 ± 0.04 (1.52)	2.1 ± 1.2 (4.8)
ConstraintNet	3.95	$\mathbf{0.0 \pm 0.0\,\%}$	$\mathbf{0.88 \pm 0.01}$	0.69 ± 0.07	$\mathbf{1.2 \pm 0.3}$
Unconstrained (clipped)	5.25 (0.3)	$3.7 \pm 4.6\%$ ($\mathbf{0.0\%}$)	0.75 ± 0.04 (0.76)	$\mathbf{0.53 \pm 0.05}$ (1.04)	1.6 ± 0.2 (3.7)

Table 2: For ConstraintNet and a (clipped) unconstrained NN used as policy, the table shows the average value and the standard deviation of the crash rate $M_{\text{CrashRate}}$ (lower means safer), a safety metric M_{Safety} (higher means safer), a metric measuring the discomfort $M_{\text{Discomfort}}$ (lower means more comfortable), and a tracking metric M_{TrackErr} (lower is better). For each trial, a training is performed with the specified number of time steps (compare Figure 5), then the metrics are evaluated on 100 episodes with a maximum duration of five minutes.

Shwartz, Shammah, and Shashua 2017), however with relaxed assumptions. The minimum acceleration $a_{\min}$ is modeled to prevent driving in reverse direction. Furthermore, $a_{\min}$ is clipped to restrict the maximum deceleration to the operational limits of the ACC according to ISO15622 (The International Organization for Standardization 2018).

Training and Results. We train the FOC in a simulator with a variety of different traffic scenarios: target vehicle with constant and varying speed, stop-and-go traffic, randomly occurring lane changes of the target vehicle, and cutting-in vehicles. In terms of reinforcement learning, the ego vehicle represents the agent and the FOC the policy. For training, we choose the Twin Delayed DDPG (TD3) algorithm (Fujimoto, Van Hoof, and Meger 2018), a deep reinforcement learning algorithm for continuous control problems. We use ConstraintNet and a corresponding fully connected NN without constraints for the policy. In a third policy, we additionally clip the demanded acceleration of the unconstrained NN to the safe interval. Contrary to sampling the constraint parameter (Algorithm 1), one valid constraint parameter is computed per sample and given by $s = (a_{\min}, a_{\max})$. We design the reward function to take functional, safe, and comfortable driving into account. The modeling of functional and safe behavior is based on Desjardins and Chaib-draa (2011) and extended by a punishment of the demanded acceleration and jerk of the ego vehicle to increase driving comfort. For evaluation, we analyze several metrics: The crash rate $M_{\text{CrashRate}}$ indicates the percentage of interrupted episodes due to a collision with the target vehicle. Further, we define a safety metric (M_{Safety}), a discomfort metric ($M_{\text{Discomfort}}$), and the tracking error (M_{TrackErr}):

$$M_{\text{Safety}} = \langle \min(\frac{\min_{t\in[1,T]} \tau_t}{\tau_{\text{set}}}; 1) \rangle \in [0,1], \tag{14}$$

$$M_{\text{Discomfort}} = \langle m_1 \frac{1}{T} \sum_{t=1}^{T} a_{\text{ego},t}^2 + m_2 \frac{1}{T} \sum_{t=1}^{T} j_{\text{ego},t}^2 \rangle, \tag{15}$$

$$M_{\text{TrackErr}} = \langle m_0 \frac{1}{T} \sum_{t=1}^{T} (\tau_{\text{set}} - \tau_t)^2 \rangle. \tag{16}$$

Here, t is the index for the time step within an episode $[1, T]$, τ is the current time gap, $\tau_{\text{set}} = 2\,\text{s}$ is the set time gap, and $m_0 = 1\,\text{s}^{-2}$, $m_1 = 1\,\text{s}^4\,\text{m}^{-2}$, $m_2 = 0.5\,\text{s}^6\,\text{m}^{-2}$ are weighting factors for the tracking error, the squared acceleration a_{ego}, and the squared jerk j_{ego}. The metrics are averaged over 100 episodes and each episode has a maximum duration of five minutes corresponding to $T = 3000$ timestamps.

Figure 5 shows the average episode return during training. The ConstraintNet-based training converges fast and shows the most stable results. Figure 6 compares the crash rates of the three policies during training and shows that the chosen constraints prevent collisions efficiently. In rare cases, collisions might still occur due to a closely cutting in vehicle.

Table 2 summarizes several evaluation metrics measured after training. ConstraintNet performs with the lowest tracking error and reaches the best safety scores. Most comfortable driving is reached with an unconstrained NN at the expense of a high crash rate.

Conclusion

In this paper, we have presented ConstraintNet—a scalable neural network architecture with the capability to constrain the space of possible predictions in each forward pass independently. The validity of the output constraints has been proven and originates from the design of the architecture. Aiming to reach researchers of different domains, we focused on a general mathematical formalization which is applicable to output constraints of arbitrary parameterizable geometries. For output constraints in form of convex polytopes, we proposed a parametrization based on the vertex representation. Our experiments on facial landmark detection tasks have shown improved performance and clearly reduced runtimes over performing a projection on the constrained output region with a differentiable optimization layer. Furthermore, ConstraintNet is applicable to safety-critical applications by defining appropriate safe sets as output constraints. *e.g.* to include safety rules of the Responsibility-Sensitive Safety model (Shalev-Shwartz, Shammah, and Shashua 2017) for self-driving cars. As an example, we considered a follow object controller, applied ConstraintNet as policy, and trained it with a deep reinforcement learning algorithm. By defining safe sets with explicit rules, rear-end collisions have been efficiently avoided. Furthermore, ConstraintNet improved the stability of the learning behavior and performance. In future research, we plan to investigate more complex output constraints and to explore constraints between different outputs in multi-task learning.

Acknowledgments

We thank especially Daniel Slieter, Christian Hubschneider, and Eric Wahl for the valuable and inspiring discussions and the continuous feedback to previous versions of this manuscript. We also thank the team of the Innovation Campus from the Porsche AG for feedback, input, and profound discussions.

Ethical Impact

Major achievements in challenging learning tasks have resulted in a broad attention of neural networks and artificial intelligence in research, technology, politics and society. Contrary, the black-box behavior of neural networks (Gilpin et al. 2018; Rudin 2019) limits their further expansion and reduces the trust of the society in these algorithms (Toreini et al. 2020). With ConstraintNet, we propose a general methodology to mitigate the potential risk of the black-box by excluding hazardous outputs in the first place. However, the identification of reasonable output constraints needs to be carefully determined, biases the data-driven decision process and might be misused. In the following, we discuss the potential benefits and risks.

The output constraints of ConstraintNet are promising for the implementation and verification of requirements in neural networks. This is important for applying neural networks in safety-critical technology, *e.g*. for ensuring compliance with norms and standards (Nistér et al. 2019; Shalev-Shwartz, Shammah, and Shashua 2017; The International Organization for Standardization 2019). In this paper, ConstraintNet has been applied to a follow object controller for vehicles and appropriate constraints have been proven to prevent collisions efficiently. As a more complex application, a motion planner for an autonomous driving system may be considered. In this case, ConstraintNet would predict parameters of a trajectory (*e.g*. the curvature) for a given representation of the environment. However, only a subset of the trajectory parameter space (*e.g*. an interval of curvatures) would correspond to safe trajectories (*e.g*. in line with traffic regulations and collision free w.r.t. objects in the environment representation). The output constraints could be utilized to constrain the prediction of ConstraintNet only to trajectory parameters corresponding to safe trajectories. We conclude that ConstraintNet increases safety and promotes trust.

Verifiable output constraints might contribute to expand the range of applications for neural networks and increase the level of industrial automation further, *e.g*. when applied to driving functions in self-driving cars. On the one hand, this is beneficial and products or services become affordable for a broader class of the population. On the other hand, industrial automation often replaces human jobs.

Furthermore, there is a risk of misusing output constraints or applying them wrongly by accident with harmful implications, *e.g*. leading to a manipulation or discrimination (Barocas, Hardt, and Narayanan 2019; Hardt, Price, and Srebro 2016). This is conceivable in a data-driven assessment of individuals or an automated selection of applicants for admission. However, conventional neural networks might learn discriminatory behavior as well. In future research, it would be interesting to investigate if output constraints can even help to achieve fairness.

Note, modeling output constraints inherently requires balancing between positive and negative impacts. Finally, we strongly recommend a responsible and careful design of output constraints when applying ConstraintNet.

References

Agrawal, A.; Amos, B.; Barratt, S.; Boyd, S.; Diamond, S.; and Zico Kolter, J. 2019. Differentiable Convex Optimization Layers. In *Advances in Neural Information Processing Systems 32*, 9562–9574.

Amos, B.; and Kolter, J. Z. 2017. OptNet: Differentiable Optimization as a Layer in Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*, 136–145.

Barocas, S.; Hardt, M.; and Narayanan, A. 2019. *Fairness and Machine Learning*. fairmlbook.org, accessed 8/14/2020.

Cui, H.; Nguyen, T.; Chou, F.-C.; Lin, T.-H.; Schneider, J.; Bradley, D.; and Djuric, N. 2020. Deep Kinematic Models for Kinematically Feasible Vehicle Trajectory Predictions. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 10563–10569.

Dalal, G.; Dvijotham, K.; Vecerik, M.; Hester, T.; Paduraru, C.; and Tassa, Y. 2018. Safe Exploration in Continuous Action Spaces. In *arXiv preprint arXiv:1801.08757*.

Desjardins, C.; and Chaib-draa, B. 2011. Cooperative Adaptive Cruise Control: A Reinforcement Learning Approach. *IEEE Transactions on Intelligent Transportation Systems* 12: 1248–1260.

Fujimoto, S.; Van Hoof, H.; and Meger, D. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning*, 1587–1596.

Gilpin, L. H.; Bau, D.; Yuan, B. Z.; Bajwa, A.; Specter, M.; and Kagal, L. 2018. Explaining Explanations: An Overview of Interpretability of Machine Learning. In *Proceedings of the IEEE 5th International Conference on Data Science and Advanced Analytics*, 80–89.

Gros, S.; Zanon, M.; and Bemporad, A. 2020. Safe Reinforcement Learning via Projection on a Safe Set: How to Achieve Optimality? In *arXiv preprint arXiv:2004.00915*.

Hardt, M.; Price, E.; and Srebro, N. 2016. Equality of Opportunity in Supervised Learning. In *Advances in Neural Information Processing Systems 30*, 3323–3331.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Karpatne, A.; Watkins, W.; Read, J.; and Kumar, V. 2017. Physics-guided Neural Networks (PGNN): An Application in Lake Temperature Modeling. In *arXiv preprint arXiv:1710.11431*.

Lezcano-Casado, M.; and Martínez-Rubio, D. 2019. Cheap Orthogonal Constraints in Neural Networks: A Simple Parametrization of the Orthogonal and Unitary Group. In *Proceedings of the 36th International Conference on Machine Learning*, 3794–3803.

Li, T.; and Srikumar, V. 2019. Augmenting Neural Networks with First-order Logic. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* 292–302.

Liu, Z.; Luo, P.; Wang, X.; and Tang, X. 2015. Deep Learning Face Attributes in the Wild. In *Proceedings of the IEEE International Conference on Computer Vision*, 3730–3738.

Márquez-Neila, P.; Salzmann, M.; and Fua, P. 2017. Imposing Hard Constraints on Deep Networks: Promises and Limitations. In *arXiv preprint arXiv:1706.02025*.

Nistér, D.; Lee, H.-L.; Ng, J.; and Wang, Y. 2019. The Safety Force Field. In *NVIDIA White Paper*.

Opitz, D. W.; and Shavlik, J. W. 1995. Generating Accurate and Diverse Members of a Neural-Network Ensemble. In *Advances in Neural Information Processing Systems 8*, 535–541.

Rudin, C. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1: 206–215.

Shalev-Shwartz, S.; Shammah, S.; and Shashua, A. 2017. On a Formal Model of Safe and Scalable Self-driving Cars. In *arXiv preprint arXiv:1708.06374*.

The International Organization for Standardization. 2018. Adaptive Cruise Control. In *ISO 15622*.

The International Organization for Standardization. 2019. Road vehicles — Safety of the intended functionality. In *ISO/PAS 21448*.

Toreini, E.; Aitken, M.; Coopamootoo, K.; Elliott, K.; Zelaya, C. G.; and van Moorsel, A. 2020. The relationship between trust in AI and trustworthy machine learning technologies. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 272–283.

Zadeh, A.; Baltrušaitis, T.; and Morency, L.-P. 2017. Convolutional Experts Constrained Local Model for 3D Facial Landmark Detection. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2519–2528.