Constraining neural networks output by an interpolating loss function with region priors

Hannes Bergkvist[*]
Sony, R&D Center Europe
Lund, Sweden
hannes.bergkvist@sony.com

Peter Exner
Sony, R&D Center Europe
Lund, Sweden
peter.exner@sony.com

Paul Davidsson
Malmö University
Malmö, Sweden
paul.davidsson@mau.se

Abstract

Deep neural networks have the ability to generalize beyond observed training data. However, for some applications they may produce output that apriori is known to be invalid. If prior knowledge of valid output regions is available, one way of imposing constraints on deep neural networks is by introducing these priors in a loss function. In this paper, we introduce a novel way of constraining neural network output by using encoded regions with a loss function based on gradient interpolation. We evaluate our method in a positioning task where a region map is used in order to reduce invalid position estimates. Results show that our approach is effective in decreasing invalid outputs for several geometrically complex environments.

1 Introduction

Two common approaches to improve generalization of deep models involve introducing more diverse training data and introducing inductive bias through prior knowledge. Lutter et al. [4] show how Lagrangian mechanics can be encoded as physics priors into a network topology to impose physical constraints and thereby improving generalization of deep models. Zambaldi et al. [10] demonstrate how raw pixel data can be transformed to a spatial feature map to introduce relational inductive bias to a reinforcement learning (RL) agent. One way of encoding prior knowledge as constraints on neural networks is to introduce a constraining loss function based on these priors. Xu et al. [9] present a semantic loss function based on symbolic knowledge for semi-supervised classification. An additional constraining loss can also be seen as learning another task, which can provide inductive bias and cause the model to generalize better [7][2].

In this work we introduce a novel method of constraining neural network output by using prior knowledge of valid output regions with a loss function based on gradient interpolation. The region maps are easy to create even for complex regions, for example by using a standard drawing application, or simply generating them from existing formats such as images or drawings if available. By encoding region maps into a loss function, we demonstrate an interpretable approach to include prior knowledge into deep neural networks (DNN).

We evaluate our method on a static positioning task where the objective is to compute a single position estimate from several simultaneously taken distance measurements from known positions, independent of previous or future measurements or estimates. Examples of approaches for static positioning are iterative least square (ILS)[1], or machine learning methods such as support vector machines (SVM) and DNN. For example Xiao et al. [8] achieves better results with DNN than with a SVM. Félix et al. [3] investigates DNN for positioning with supervised and unsupervised training. In this work we demonstrate our approach based on a DNN for positioning, but the approach is applicable for any regression task where invalid ouput regions can be represented as a binary matrix.

[*]Also with Malmö University

1st NeurIPS workshop on Interpretable Inductive Biases and Physically Structured Learning (2020), virtual.

2 Constraining loss function

The constraining loss function uses the output of the model, $\hat{y}$, and the region map encoded in the form of a binary matrix $\mathbf{Z}$. The loss is based on where $\hat{y}$ is located on $\mathbf{Z}$. As a first step the region map is created as a binary matrix with pixel value zero for valid regions and one for invalid regions $\mathbf{Z} \in \{0, 1\}^{r \times c}$. $\mathbf{Z}$ is then used to generate a topographic matrix $\mathbf{Z}_{top}$ where invalid region pixel values are increased as a function of the distance to the closest allowed region. This conversion is only done once, before training. The loss function should return a loss from $\mathbf{Z}_{top}$ corresponding to $\hat{y}$. Further, we need to consider that the resolution of the $\mathbf{Z}_{top}$ is limited to the size of the matrix, which might be less than the resolution of $\hat{y}$. Additionally, the loss needs to have derivatives with respect to $\hat{y}$. We achieve all these aspects by applying bilinear interpolation for $\hat{y}$ on $\mathbf{Z}_{top}$. Bilinear interpolation uses the four points with known values (1), closest to the point with an unknown value (x, y). Starting with interpolating in the x-direction, then use this result and interpolate in the y-direction to get an approximate topographic value at (x, y) as (2), with partial derivatives as (3).

$$Q_{11} = (x_1, y_1), Q_{12} = (x_1, y_2), Q_{21} = (x_2, y_1), Q_{22} = (x_2, y_2) \tag{1}$$

$$\begin{aligned} f(x, y) &\approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2) \\ &= \frac{y_2 - y}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \right) \\ &+ \frac{y - y_1}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \right) \end{aligned} \tag{2}$$

$$\begin{aligned} \frac{\partial f}{\partial x} &= \frac{(y - y_2) f(Q_{11}) + (y_2 - y) f(Q_{21}) + (y_1 - y) f(Q_{12}) + (y - y_1) f(Q_{22})}{(y_2 - y_1)(x_2 - x_1)} \\ \frac{\partial f}{\partial y} &= \frac{(x - x_2) f(Q_{11}) + (x_1 - x) f(Q_{21}) + (x_2 - x) f(Q_{12}) + (x - x_1) f(Q_{22})}{(y_2 - y_1)(x_2 - x_1)} \end{aligned} \tag{3}$$

For bilinear interpolation with a topographic matrix $\mathbf{Z}_{top} \in \mathbb{R}^{r \times c}$, the coordinates x and y need to be normalized according to the size of the matrix. The points for interpolation are then (4). Resulting in a constraining loss function that outputs a low or zero loss for valid positions and a higher loss for invalid positions with derivatives negative towards valid positions (5).

$$\begin{aligned} [r_1, c_1] &= [\lfloor y' \rfloor, \lfloor x' \rfloor], & [r_1, c_2] &= [\lfloor y' \rfloor, \lfloor x' \rfloor + 1] \\ [r_2, c_1] &= [\lfloor y' \rfloor + 1, \lfloor x' \rfloor], & [r_2, c_2] &= [\lfloor y' \rfloor + 1, \lfloor x' \rfloor + 1] \end{aligned} \tag{4}$$

$$\begin{aligned} \mathcal{L}_c((x, y), \mathbf{Z}_{top}) &= \frac{r_2 - y'}{r_2 - r_1} \left(\frac{c_2 - x'}{c_2 - c_1} \mathbf{Z}_{top}[r_1, c_1] + \frac{x' - c_1}{c_2 - c_1} \mathbf{Z}_{top}[r_2, c_1] \right) \\ &+ \frac{y' - r_1}{r_2 - r_1} \left(\frac{c_2 - x'}{c_2 - c_1} \mathbf{Z}_{top}[r_1, c_2] + \frac{x' - c_1}{c_2 - c_1} \mathbf{Z}_{top}[r_2, c_2] \right) \end{aligned} \tag{5}$$

The default loss and constraining loss functions are combined to form a total loss (6). Here p represents the weighting between the losses. The most straight forward approach is to use static weighting between the losses. A problem is to find the optimal value for p that avoids overfitting one loss.

$$\mathcal{L}_{tot}(\hat{y}, y) = \mathcal{L}_d(\hat{y}, y) p + \mathcal{L}_c(\hat{y}, \mathbf{Z}_{top})(1 - p) \tag{6}$$

In this work, we apply an adaptive weighting, where $\mathcal{L}_d$ acts as the primary loss while $\mathcal{L}_c$ is introduced over time. Initially $p = 1$ resulting in $\mathcal{L}_{tot} = \mathcal{L}_d$. For every epoch, p is decreased or increased with step size s, dependent on if the training error is below or above a threshold t. The threshold t decides how much the model is allowed to optimize for $\mathcal{L}_d$ or $\mathcal{L}_c$, while s decides how fast the weight between the losses change. Both s and t are hyperparameters that need to be tuned for optimal results.

3 Experiments

To validate our method we train a DNN for positioning with and without the constraining loss in three different environments. We are working on this method for real world positioning applications, where invalid regions often exist in the form of sealed off rooms or buildings. The environments aim to represent such scenarios but also test the general ability of constraining outputs to geometrically different regions. We use a DNN as proposed by Félix et al. [3] with seven layers and five hidden layers of size $(n_h^1, n_h^2, n_h^3, n_h^4, n_h^5,) = (1000, 1000, 500, 100, 10)$. The input and output layer sizes are $n_x = 30$ and $n_y = 2$, according to the number of features in the training data and the output position coordinates. We use Rectified Linear Unit (ReLU) as activation functions [5] and the Mean Squared Error (MSE) loss function as default loss. Training is done for 5000 epochs with batch size 1024 and learning rate 10^{-3}. The loss balance weighting has step size $s = 0.0005$ and threshold $t = 5$. The code for all experiments are implemented in Python with models, loss functions and training using PyTorch [6] .

Data is generated with a simple procedure: A sample (x^i, y^i) is created by first generating a label position $y^i = (y_1^i, y_2^i)$ by drawing two samples from a uniform distribution $y_1, y_2 \sim \mathcal{U}(a, b)$, a and b are the maximum coordinates of the area. The the distances are then calculated $\{d_1, ..., d_{n_{kp}}\}$ to all known positions $\{(p1_j, p2_j); j = 1, ..., n_{kp})\}$ and Gaussian noise is added, $dg_j = d_j + Z_j$, $Z \sim \mathcal{N}(\mu, \sigma^2)$. This is combined to form the features $x^i = (dg_1, p1_1, p2_1, ...dg_{n_{kp}}, p1_{n_{kp}}, p2_{n_{kp}})$. As a last step, the n_{drop} largest distances are removed, $n_{drop} \sim \mathcal{U}(3, n_{kp})$. This is done to better generalize to real scenarios where known positions at large distances often are out of reach. The training data cover all valid and invalid positions, such as $\mathcal{A} = \{(Y_1, Y_2)|Y_1 \in \mathbb{R}[0, a], Y_2 \in \mathbb{R}[0, b]\}$. The validation and test data has positions only in valid areas, such as $\mathcal{B} = \{(Y_1, Y_2)|(Y_1, Y_2) \in \mathbf{Z}, \mathbf{Z} == 0\}$. All three data sets consists of 100k samples, with $n_{kp} = 10$, $\mu = 0$ and $\sigma = 5$.

We evaluate the models by running inference on the test data set. The positioning error is calculated as the L_2 norm between the model inference output and label of the data. Invalid output ratio is calculated as the percentage of the model inference output that are at invalid positions.

4 Results

Results for all experiments are visualised in Figure 1. The inference output on the test data set is plotted in white, the dark regions represent the invalid regions where we want to avoid outputs. The evaluation results are summarized in Table 1. Figure 2 shows training curves for experiment 1 and 2.

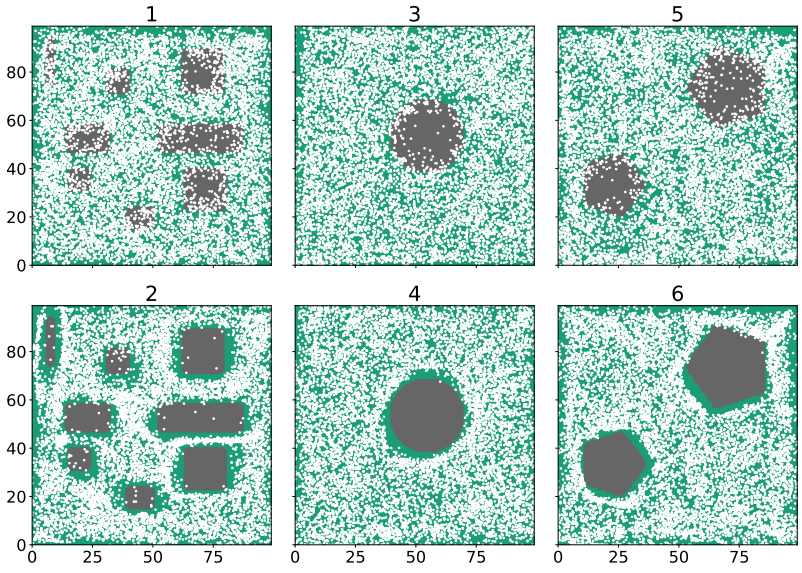

Figure 1: Result plots for experiments 1,3,5 (baseline) and 2,4,6 (constrained).

From plots 1,3 and 5 in Figure 1, we see that the baseline method produce outputs in invalid regions on the test data. In plots 2, 4 and 6, it's clear that the constraining loss effectively reduces the number of outputs in invalid regions. The different environments introduce varying challenges for the constraining task. The circle and dual pentagons prove to be the easiest with an almost perfect result, while the squares prove to be more challenging. One interesting observation is the aberrations in the uniform distribution of the constrained outputs. Especially the squares suffer from this side effect. We can also see that some invalid outputs still exists. These could be further reduced by weighting the constraint harder by adjusting the t and s parameters, but it would also result in more aberration.

The evaluation results in Table 1 show at least one order of magnitude decrease of invalid outputs for all three environments. The positioning error improves for the circle environment, while we observe no improvement and an increase in the variance for the pentagon and the squares. Our conclusion is that, while the reduction of invalid outputs lead to a decrease in positioning error, an increase in abberations has a negative impact.

Table 1: Positioning error and invalid outputs

Experiment	**Environment**	**Method**	**Position error** m (SD)	**Invalid output** % (SD)
1	squares	baseline	5.19 (0.06)	8.94 (0.16)
2	squares	constrained	5.29 (0.22)	0.79 (0.14)
3	circle	baseline	5.15 (0.08)	1.72 (0.05)
4	circle	constrained	4.57 (0.07)	0.01 (0.01)
5	pentagons	baseline	5.39 (0.25)	3.79 (0.10)
6	pentagons	constrained	5.37 (0.59)	0.07 (0.02)

To further analyze the workings of our loss function we look at training curves for experiment 1 and 2. From the p value graph we see how p starts decreasing with step s as the training position error reach t. At the same time, invalid output error start to decrease compared to the baseline. Based on these curves it is possible to examine and tune p with s and t to balance the constraining effect against the positioning task.

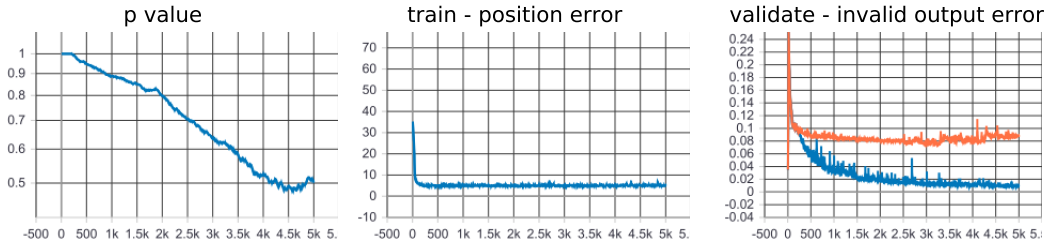

Figure 2: Training curves for experiments 1 (orange - baseline) and 2 (blue - constrained).

5 Conclusion and Future work

We introduced a novel way of constraining neural network output by using prior knowledge of valid output regions with a loss function based on gradient interpolation. We presented experiments validating our method in the positioning task. Results demonstrate that our method can be used to effectively reduce invalid outputs. The region maps are easily generated, the induced bias is interpretable and the loss can be tuned towards a stronger or weaker constrain. Future work include improved approaches for loss weighting as well as investigating the aberration side effect. Additionally, it would be interesting to apply our method on DNN models for tasks other than positioning.

References

[1] Simo Ali-löytty and Jussi Collin. MAT-45806 Mathematics for Positioning TKT-2546 Methods for Positioning. Technical report, 2008.

[2] Rich Caruana. Multitask Learning. *Machine Learning*, 28(1):41–75, 1997.

[3] Gibrán Félix, Mario Siller, and Ernesto Navarro Álvarez. A fingerprinting indoor localization algorithm based deep learning. In *International Conference on Ubiquitous and Future Networks, ICUFN*, volume 2016-Augus, pages 1006–1011. IEEE Computer Society, 8 2016.

[4] Michael Lutter, Christian Ritter, and Jan Peters. Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning. 7 2019.

[5] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve Restricted Boltzmann machines. In *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*, pages 807–814, 2010.

[6] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury Google, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf Xamla, Edward Yang, Zach Devito, Martin Raison Nabla, Alykhan Tejani, Sasank Chilamkurthy, Qure Ai, Benoit Steiner, Lu Fang Facebook, Junjie Bai Facebook, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[7] Sebastian Ruder. An Overview of Multi-Task Learning in Deep Neural Networks. 6 2017.

[8] Linchen Xiao, Arash Behboodi, and Rudolf Mathar. Learning the Localization Function: Machine Learning Approach to Fingerprinting Localization. 3 2018.

[9] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van Den Broeck. A Semantic Loss Function for Deep Learning with Symbolic Knowledge. Technical report, 2018.

[10] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. DEEP REINFORCEMENT LEARNING WITH RELATIONAL INDUCTIVE BIASES. Technical report.