Sobolev Training for Neural Networks

Wojciech Marian Czarnecki, Simon Osindero, Max Jaderberg
Grzegorz Swirszcz, and Razvan Pascanu
DeepMind, London, UK
{lejlot,osindero,jaderberg,swirszcz,razp}@google.com

Abstract

At the heart of deep learning we aim to use neural networks as function approximators – training them to produce outputs from inputs in emulation of a ground truth function or data creation process. In many cases we only have access to input-output pairs from the ground truth, however it is becoming more common to have access to derivatives of the target output with respect to the input – for example when the ground truth function is itself a neural network such as in network compression or distillation. Generally these target derivatives are not computed, or are ignored. This paper introduces Sobolev Training for neural networks, which is a method for incorporating these target derivatives in addition the to target values while training. By optimising neural networks to not only approximate the function's outputs but also the function's derivatives we encode additional information about the target function within the parameters of the neural network. Thereby we can improve the quality of our predictors, as well as the data-efficiency and generalization capabilities of our learned function approximation. We provide theoretical justifications for such an approach as well as examples of empirical evidence on three distinct domains: regression on classical optimisation datasets, distilling policies of an agent playing Atari, and on large-scale applications of synthetic gradients. In all three domains the use of Sobolev Training, employing target derivatives in addition to target values, results in models with higher accuracy and stronger generalisation.

1 Introduction

Deep Neural Networks (DNNs) are one of the main tools of modern machine learning. They are consistently proven to be powerful function approximators, able to model a wide variety of functional forms – from image recognition [6, 23], through audio synthesis [24], to human-beating policies in the ancient game of GO [21]. In many applications the process of training a neural network consists of receiving a dataset of input-output pairs from a ground truth function, and minimising some loss with respect to the network's parameters. This loss is usually designed to encourage the network to produce the same output, for a given input, as that from the target ground truth function. Many of the ground truth functions we care about in practice have an unknown analytic form, *e.g.* because they are the result of a natural physical process, and therefore we only have the observed input-output pairs for supervision. However, there are scenarios where we do know the analytic form and so are able to compute the ground truth gradients (or higher order derivatives), alternatively sometimes these quantities may be simply observable. A common example is when the ground truth function is itself a neural network; for instance this is the case for distillation [7, 19], compressing neural networks [5], and the prediction of synthetic gradients [10]. Additionally, if we are dealing with an environment/data-generation process (vs. a pre-determined set of data points), then even though we may be dealing with a black box we can still approximate derivatives using finite differences. In this work, we consider how this additional information can be incorporated in the learning process, and what advantages it can provide in terms of data efficiency and performance. We propose Sobolev Training (ST) for neural networks as a simple and efficient technique for leveraging

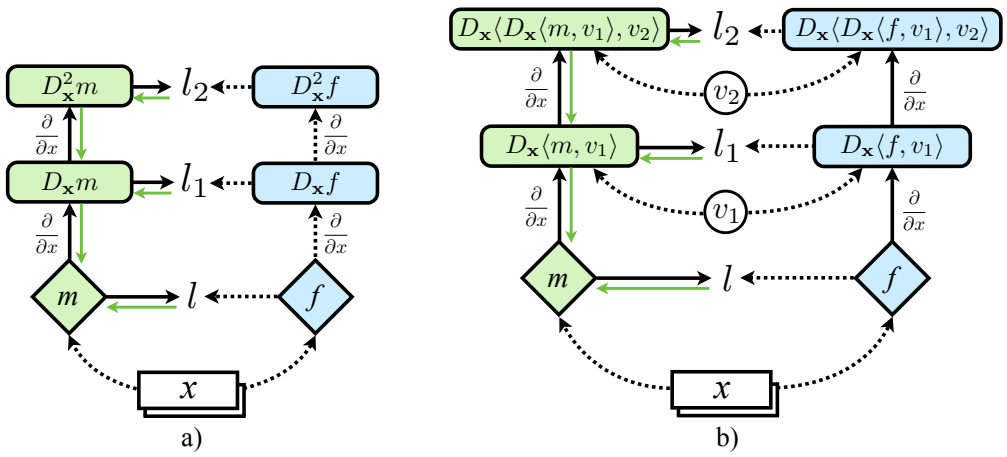


Figure 1: a) Sobolev Training of order 2. Diamond nodes m and f indicate parameterised functions, where m is trained to approximate f. Green nodes receive supervision. Solid lines indicate connections through which error signal from loss l, l_1, and l_2 are backpropagated through to train m. b) Stochastic Sobolev Training of order 2. If f and m are multivariate functions, the gradients are Jacobian matrices. To avoid computing these high dimensional objects, we can efficiently compute and fit their projections on a random vector v_j sampled from the unit sphere.

derivative information about the desired function in a way that can easily be incorporated into any training pipeline using modern machine learning libraries.

The approach is inspired by the work of Hornik [8] which proved the universal approximation theorems for neural networks in Sobolev spaces – metric spaces where distances between functions are defined both in terms of their differences in values and differences in values of their derivatives.

In particular, it was shown that a sigmoid network can not only approximate a function's value arbitrarily well, but that the network's derivatives with respect to its inputs can approximate the corresponding derivatives of the ground truth function arbitrarily well too. Sobolev Training exploits this property, and tries to match not only the output of the function being trained but also its derivatives.

There are several related works which have also exploited derivative information for function approximation. For instance Wu et al. [27] and antecedents propose a technique for Bayesian optimisation with Gaussian Processess (GP), where it was demonstrated that the use of information about gradients and Hessians can improve the predictive power of GPs. In previous work on neural networks, derivatives of predictors have usually been used either to penalise model complexity (e.g. by pushing Jacobian norm to 0 [18]), or to encode additional, hand crafted invariances to some transformations (for instance, as in Tangentprop [22]), or estimated derivatives for dynamical systems [4]. Similar techniques have also been used in critic based Reinforcement Learning (RL), where a critic's derivatives are trained to match its target's derivatives [26, 13] using shallow, sigmoid based models. Finally, Hyvärinen proposed Score Matching Networks [9], which are based on the somewhat surprising observation that one can model unknown derivatives of the function without actual access to its values – all that is needed is a sampling based strategy and specific penalty. However, such an estimator has a high variance [25], thus it is not really useful when true derivatives are given.

To the best of our knowledge and despite its simplicity, the proposal to directly match network derivatives to the true derivatives of the target function has been minimally explored for deep networks. In our method, we show that by using the additional knowledge of derivatives with Sobolev Training we are able to train better models – models which achieve lower approximation errors and generalise to test data better – and reduce the sample complexity of learning. The contributions of our paper are therefore threefold: (**1**): We introduce Sobolev Training – a new paradigm for training neural networks. (**2**): We look formally at the implications of matching derivatives, extending previous results of Hornik [8] and showing that modern architectures are well suited for such training regimes. (**3**): Empirical evidence demonstrating that Sobolev Training leads to improved performance and generalisation, particularly in low data regimes. Example domains are: regression on classical

optimisation problems; policy distillation from RL agents trained on the Atari domain; and training deep, complex models using synthetic gradients – we report the first successful attempt to train a large-scale ImageNet model using synthetic gradients.

2 Sobolev Training

We begin by introducing the idea of training using Sobolev spaces. When learning a function f, we may have access to not only the output values $f(x_i)$ for training points x_i, but also the values of its j-th order derivatives with respect to the input, $D^j_{\mathbf{x}} f(x_i)$. In other words, instead of the typical training set consisting of pairs $\{(x_i, f(x_i))\}_{i=1}^N$ we have access to $(K+2)$-tuples $\{(x_i, f(x_i), D^1_{\mathbf{x}} f(x_i), ..., D^K_{\mathbf{x}} f(x_i))\}_{i=1}^N$. In this situation, the derivative information can easily be incorporated into training a neural network model of f by making derivatives of the neural network match the ones given by f.

Considering a neural network model m parameterised with θ, one typically seeks to minimise the empirical error in relation to f according to some loss function ℓ

$$\sum_{i=1}^{N} \ell(m(x_i|\theta), f(x_i)).$$

When learning in Sobolev spaces, this is replaced with:

$$\sum_{i=1}^{N} \left[\ell(m(x_i|\theta), f(x_i)) + \sum_{j=1}^{K} \ell_j \left(D^j_{\mathbf{x}} m(x_i|\theta), D^j_{\mathbf{x}} f(x_i) \right) \right], \tag{1}$$

where ℓ_j are loss functions measuring error on j-th order derivatives. This causes the neural network to encode derivatives of the target function in its own derivatives. Such a model can still be trained using backpropagation and off-the-shelf optimisers.

A potential concern is that this optimisation might be expensive when either the output dimensionality of f or the order K are high, however one can reduce this cost through stochastic approximations. Specifically, if f is a multivariate function, instead of a vector gradient, one ends up with a full Jacobian matrix which can be large. To avoid adding computational complexity to the training process, one can use an efficient, stochastic version of Sobolev Training: instead of computing a full Jacobian/Hessian, one just computes its projection onto a random vector (a direct application of a known estimation trick [18]). In practice, this means that during training we have a random variable v sampled uniformly from the unit sphere, and we match these random projections instead:

$$\sum_{i=1}^{N} \left[\ell(m(x_i|\theta), f(x_i)) + \sum_{j=1}^{K} \mathbb{E}_{v^j} \left[\ell_j \left(\left\langle D^j_{\mathbf{x}} m(x_i|\theta), v^j \right\rangle, \left\langle D^j_{\mathbf{x}} f(x_i), v^j \right\rangle \right) \right] \right]. \tag{2}$$

Figure 1 illustrates compute graphs for non-stochastic and stochastic Sobolev Training of order 2.

3 Theory and motivation

While in the previous section we defined Sobolev Training, it is not obvious that modeling the derivatives of the target function f is beneficial to function approximation, or that optimising such an objective is even feasible. In this section we motivate and explore these questions theoretically, showing that the Sobolev Training objective is a well posed one, and that incorporating derivative information has the potential to drastically reduce the sample complexity of learning.

Hornik showed [8] that neural networks with non-constant, bounded, continuous activation functions, with continuous derivatives up to order K are universal approximators in the Sobolev spaces of order K, thus showing that sigmoid-networks are indeed capable of approximating elements of these spaces arbitrarily well. However, nowadays we often use activation functions such as ReLU which are neither bounded nor have continuous derivatives. The following theorem shows that for $K = 1$ we can use ReLU function (or a similar one, like leaky ReLU) to create neural networks that are universal approximators in Sobolev spaces. We will use a standard symbol $\mathcal{C}^1(S)$ (or simply $\mathcal{C}^1$) to denote a space of functions which are continuous, differentiable, and have a continuous derivative on a space S [12]. All proofs are given in the Supplementary Materials (SM).

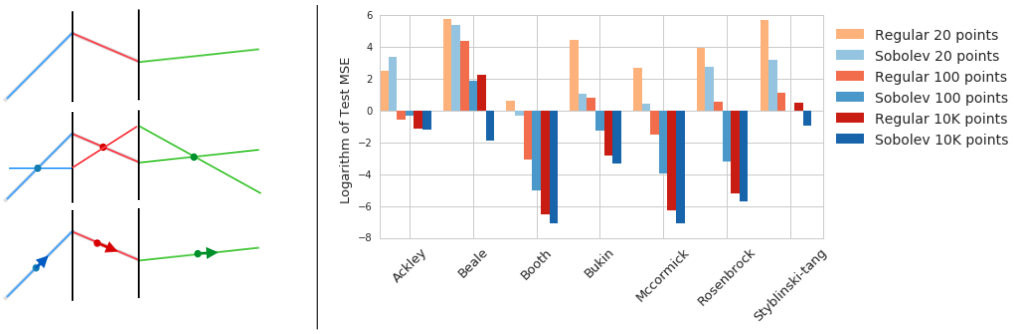


Figure 2: *Left:* From top: Example of the piece-wise linear function; Two (out of a continuum of) hypotheses consistent with 3 training points, showing that one needs two points to identify each linear segment; The only hypothesis consistent with 3 training points enriched with derivative information. *Right:* Logarithm of test error (MSE) for various optimisation benchmarks with varied training set size (20, 100 and 10000 points) sampled uniformly from the problem's domain.

Theorem 1. *Let f be a $\mathcal{C}^1$ function on a compact set. Then, for every positive ε there exists a single hidden layer neural network with a ReLU (or a leaky ReLU) activation which approximates f in Sobolev space $\mathcal{S}_1$ up to ϵ error.*

This suggests that the Sobolev Training objective is achievable, and that we can seek to encode the values and derivatives of the target function in the values and derivatives of a ReLU neural network model. Interestingly, we can show that if we seek to encode an arbitrary function in the derivatives of the model then this is impossible not only for neural networks but also for any arbitrary differentiable predictor on compact sets.

Theorem 2. *Let f be a $\mathcal{C}^1$ function. Let g be a continuous function satisfying $\|g - \frac{\partial f}{\partial x}\|_\infty > 0$. Then, there exists an $\eta > 0$ such that for any $\mathcal{C}^1$ function h either $\|f - h\|_\infty \geq \eta$ or $\left\|g - \frac{\partial h}{\partial x}\right\|_\infty \geq \eta$.*

However, when we move to the regime of finite training data, we can encode any arbitrary function in the derivatives (as well as higher order signals if the resulting Sobolev spaces are not degenerate), as shown in the following Proposition.

Proposition 1. *Given any two functions $f : S \to \mathbb{R}$ and $g : S \to \mathbb{R}^d$ on $S \subseteq \mathbb{R}^d$ and a finite set $\Sigma \subset S$, there exists neural network h with a ReLU (or a leaky ReLU) activation such that $\forall x \in \Sigma : f(x) = h(x)$ and $g(x) = \frac{\partial h}{\partial x}(x)$ (it has 0 training loss).*

Having shown that it is possible to train neural networks to encode both the values and derivatives of a target function, we now formalise one possible way of showing that Sobolev Training has lower sample complexity than regular training.

Let $\mathcal{F}$ denote the family of functions parametrised by ω. We define $K_{reg} = K_{reg}(\mathcal{F})$ to be a measure of the amount of data needed to learn some target function f. That is K_{reg} is the smallest number for which there holds: for every $f_\omega \in \mathcal{F}$ and every set of distinct K_{reg} points $(x_1, ..., x_{K_{reg}})$ such that $\forall_{i=1,...,K_{reg}} f(x_i) = f_\omega(x_i) \Rightarrow f = f_\omega$. K_{sob} is defined analogously, but the final implication is of form $f(x_i) = f_\omega(x_i) \wedge \frac{\partial f}{\partial x}(x_i) = \frac{\partial f_\omega}{\partial x}(x_i) \Rightarrow f = f_\omega$. Straight from the definition there follows:

Proposition 2. *For any $\mathcal{F}$, there holds $K_{sob}(\mathcal{F}) \leq K_{reg}(\mathcal{F})$.*

For many families, the above inequality becomes sharp. For example, to determine the coefficients of a polynomial of degree n one needs to compute its values in at least $n + 1$ distinct points. If we know values and the derivatives at k points, it is a well-known fact that only $\lceil \frac{n}{2} \rceil$ points suffice to determine all the coefficients. We present two more examples in a slightly more formal way. Let $\mathcal{F}_{\mathrm{G}}$ denote a family of Gaussian PDF-s (parametrised by μ, σ). Let $\mathbb{R}^d \supset D = D_1 \cup \ldots \cup D_n$ and let $\mathcal{F}_{\mathrm{PL}}$ be family of continuous piecewise-linear functions from D to $\mathbb{R}$ which are linear on each the elements D_i of the (fixed) decomposition of the domain D (Figure 2 Left).

Proposition 3. *There holds $K_{sob}(\mathcal{F}_{\mathrm{G}}) < K_{reg}(\mathcal{F}_{\mathrm{G}})$ and $K_{sob}(\mathcal{F}_{\mathrm{PL}}) < K_{reg}(\mathcal{F}_{\mathrm{PL}})$.*

This result relates to Deep ReLU networks as they build a hyperplanes-based model of the target function. If those were parametrised independently one could expect a reduction of sample complexity by $d+1$ times, where d is the dimension of the function domain. In practice parameters of hyperplanes in such networks are not independent, furthermore the hinges positions change so the Proposition

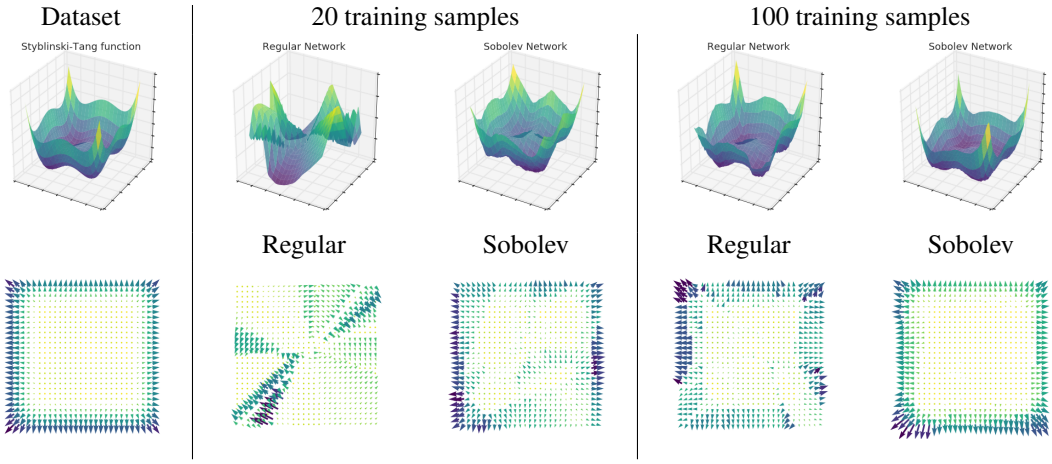


Figure 3: Styblinski-Tang function (on the left) and its models using regular neural network training (left part of each plot) and Sobolev Training (right part). We also plot the vector field of the gradients of each predictor underneath the function plot.

cannot be directly applied, but it can be seen as an intuitive way to see why the sample complexity drops significantly for Deep ReLU networks too.

4 Experimental Results

We consider three domains where information about derivatives is available during training[1].

4.1 Artificial Data

First, we consider the task of regression on a set of well known low-dimensional functions used for benchmarking optimisation methods.

We train two hidden layer neural networks with 256 hidden units per layer with ReLU activations to regress towards function values, and verify generalisation capabilities by evaluating the mean squared error on a hold-out test set. Since the task is standard regression, we choose all the losses of Sobolev Training to be L2 errors, and use a first order Sobolev method (second order derivatives of ReLU networks with a linear output layer are constant, zero). The optimisation is therefore:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \|f(x_i) - m(x_i|\theta)\|_2^2 + \|\nabla_x f(x_i) - \nabla_x m(x_i|\theta)\|_2^2.$$

Figure 2 right shows the results for the optimisation benchmarks. As expected, Sobolev trained networks perform extremely well – for six out of seven benchmark problems they significantly reduce the testing error with the obtained errors orders of magnitude smaller than the corresponding errors of the regularly trained networks. The stark difference in approximation error is highlighted in Figure 3, where we show the Styblinski-Tang function and its approximations with both regular and Sobolev Training. It is clear that even in very low data regimes, the Sobolev trained networks can capture the functional shape.

Looking at the results, we make two important observations. First, the effect of Sobolev Training is stronger in low-data regimes, however it does not disappear even in the high data regime, when one has 10,000 training examples for training a two-dimensional function. Second, the only case where regular regression performed better is the regression towards Ackley's function. This particular example was chosen to show that one possible weak point of our approach might be approximating functions with a very high frequency signal component in the relatively low data regime. Ackley's function is composed of exponents of high frequency cosine waves, thus creating an extremely bumpy surface, consequently a method that tries to match the derivatives can behave badly during testing if one does not have enough data to capture this complexity. However, once we have enough training data points, Sobolev trained networks are able to approximate this function better.

[1] All experiments were performed using TensorFlow [1] and the Sonnet neural network library [2].

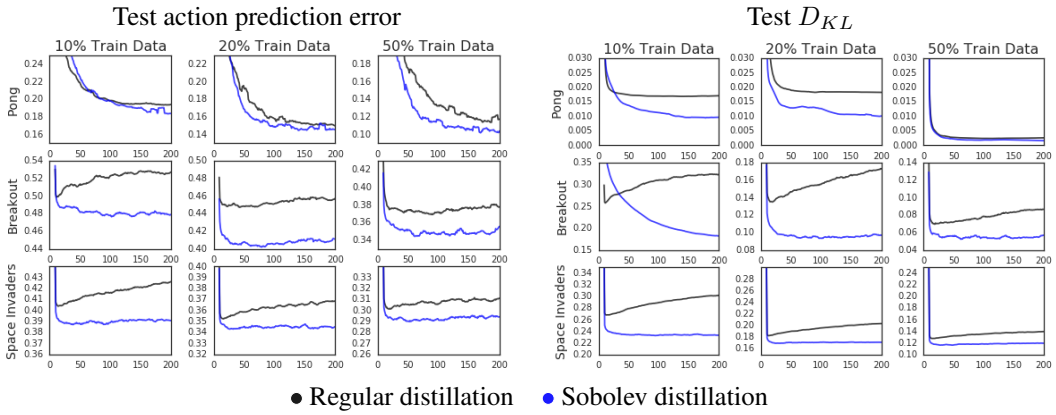


Figure 4: Test results of distillation of RL agents on three Atari games. Reported test action prediction error (left) is the error of the most probable action predicted between the distilled policy and target policy, and test D_{KL} (right) is the Kulblack-Leibler divergence between policies. Numbers in the column title represents the percentage of the 100K recorded states used for training (the remaining are used for testing). In all scenarios the Sobolev distilled networks are significantly more similar to the target policy.

4.2 Distillation

Another possible application of Sobolev Training is to perform model distillation. This technique has many applications, such as network compression [20], ensemble merging [7], or more recently policy distillation in reinforcement learning [19].

We focus here on a task of distilling a policy. We aim to distill a target policy $\pi^*(s)$ – a trained neural network which outputs a probability distribution over actions – into a smaller neural network $\pi(s|\theta)$, such that the two policies π^* and π have the same behaviour. In practice this is often done by minimising an expected divergence measure between π^* and π, for example, the Kullback–Leibler divergence $D_{KL}(\pi(s)\|\pi^*(s))$, over states gathered while following π^*. Since policies are multivariate functions, direct application of Sobolev Training would mean producing full Jacobian matrices with respect to the s, which for large actions spaces is computationally expensive. To avoid this issue we employ a stochastic approximation described in Section 2, thus resulting in the objective

$$\min_{\theta} D_{KL}(\pi(s|\theta)\|\pi^*(s)) + \alpha\mathbb{E}_v\left[\|\nabla_s\langle\log\pi^*(s), v\rangle - \nabla_s\langle\log\pi(s|\theta), v\rangle\|\right],$$

where the expectation is taken with respect to v coming from a uniform distribution over the unit sphere, and Monte Carlo sampling is used to approximate it.

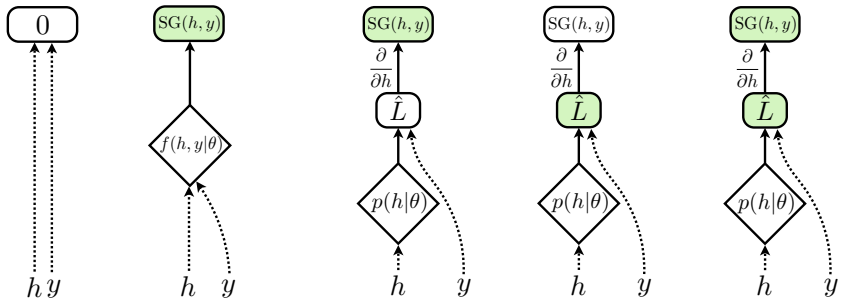
As target policies π^*, we use agents playing Atari games [16] that have been trained with A3C [15] on three well known games: Pong, Breakout and Space Invaders. The agent's policy is a neural network consisting of 3 layers of convolutions followed by two fully-connected layers, which we distill to a smaller network with 2 convolutional layers and a single smaller fully-connected layer (see SM for details). Distillation is treated here as a purely supervised learning problem, as our aim is not to re-evaluate known distillation techniques, but rather to show that if the aim is to minimise a given divergence measure, we can improve distillation using Sobolev Training. Figure 4 shows test error during training with and without Sobolev Training[2]. The introduction of Sobolev Training leads to similar effects as in the previous section – the network generalises much more effectively, and this is especially true in low data regimes. Note the performance gap on Pong is small due to the fact that optimal policy is quite degenerate for this game[3]. In all remaining games one can see a significant performance increase from using our proposed method, and as well as minor to no overfitting.

Despite looking like a regularisation effect, we stress that Sobolev Training is not trying to find the simplest models for data or suppress the expressivity of the model. This training method aims at

[2]Testing is performed on a held out set of episodes, thus there are no temporal nor causal relations between training and testing

[3]For majority of the time the policy in Pong is uniform, since actions taken when the ball is far away from the player do not matter at all. Only in crucial situations it peaks so the ball hits the paddle.

Table 1: Various techniques for producing synthetic gradients. Green shaded nodes denote nodes that get supervision from the corresponding object from the main network (gradient or loss value). We report accuracy on the test set $\pm$ standard deviation. Backpropagation results are given in parenthesis.



	Noprop	Direct SG [10]	VFBN [14]	Critic	Sobolev
CIFAR-10 with 3 synthetic gradient modules					
Top 1 (94.3%)	54.5% $\pm$1.15	79.2% $\pm$0.01	88.5% $\pm$2.70	93.2% $\pm$0.02	93.5% $\pm$0.01
ImageNet with 1 synthetic gradient module					
Top 1 (75.0%)	54.0% $\pm$0.29	-	57.9% $\pm$2.03	71.7% $\pm$0.23	72.0% $\pm$0.05
Top 5 (92.3%)	77.3% $\pm$0.06	-	81.5% $\pm$1.20	90.5% $\pm$0.15	90.8% $\pm$0.01
ImageNet with 3 synthetic gradient modules					
Top 1 (75.0%)	18.7% $\pm$0.18	-	28.3% $\pm$5.24	65.7% $\pm$0.56	66.5% $\pm$0.22
Top 5 (92.3%)	38.0% $\pm$0.34	-	52.9% $\pm$6.62	86.9% $\pm$0.33	87.4% $\pm$0.11

matching the original function's smoothness/complexity and so reduces overfitting by effectively extending the information content of the training set, rather than by imposing a data-independent prior as with regularisation.

4.3 Synthetic Gradients

The previous experiments have shown how information about the derivatives can boost approximating function values. However, the core idea of Sobolev Training is broader than that, and can be employed in both directions. Namely, if one ultimately cares about approximating derivatives, then additionally approximating values can help this process too. One recent technique, which requires a model of gradients is Synthetic Gradients (SG) [10] – a method for training complex neural networks in a decoupled, asynchronous fashion. In this section we show how we can use Sobolev Training for SG.

The principle behind SG is that instead of doing full backpropagation using the chain-rule, one splits a network into two (or more) parts, and approximates partial derivatives of the loss L with respect to some hidden layer activations h with a trainable function $SG(h, y|\theta)$. In other words, given that network parameters up to h are denoted by Θ

$$\frac{\partial L}{\partial \Theta} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial \Theta} \approx SG(h, y|\theta)\frac{\partial h}{\partial \Theta}.$$

In the original SG paper, this module is trained to minimise $L_{SG}(\theta) = \left\| SG(h, y|\theta) - \frac{\partial L(p_h, y)}{\partial h} \right\|_2^2$, where p_h is the final prediction of the main network for hidden activations h. For the case of learning a classifier, in order to apply Sobolev Training in this context we construct a loss predictor, composed of a class predictor $p(\cdot|\theta)$ followed by the log loss, which gets supervision from the true loss, and the gradient of the prediction gets supervision from the true gradient:

$$m(h, y|\theta) := L(p(h|\theta), y), \quad SG(h, y|\theta) := \partial m(h, y|\theta)/\partial h,$$

$$L_{SG}^{sob}(\theta) = \ell(m(h, y|\theta), L(p_h, y))) + \ell_1\left(\frac{\partial m(h, y|\theta)}{\partial h}, \frac{\partial L(p_h, y)}{\partial h}\right).$$

In the Sobolev Training framework, the target function is the loss of the main network $L(p_h, y)$ for which we train a model $m(h, y|\theta)$ to approximate, and in addition ensure that the model's

derivatives $\partial m(h, y|\theta)/\partial h$ are matched to the true derivatives $\partial L(p_h, y)/\partial h$. The model's derivatives $\partial m(h, y|\theta)/\partial h$ are used as the synthetic gradient to decouple the main network.

This setting closely resembles what is known in reinforcement learning as critic methods [11]. In particular, if we do not provide supervision on the gradient part, we end up with a loss critic. Similarly if we do not provide supervision at the loss level, but only on the gradient component, we end up in a method that resembles VFBN [14]. In light of these connections, our approach in this application setting can be seen as a generalisation and unification of several existing ones (see Table 1 for illustrations of these approaches).

We perform experiments on decoupling deep convolutional neural network image classifiers using synthetic gradients produced by loss critics that are trained with Sobolev Training, and compare to regular loss critic training, and regular synthetic gradient training. We report results on CIFAR-10 for three network splits (and therefore three synthetic gradient modules) and on ImageNet with one and three network splits [4].

The results are shown in Table 1. With a naive SG model, we obtain 79.2% test accuracy on CIFAR-10. Using an SG architecture which resembles a small version of the rest of the model makes learning much easier and led to 88.5% accuracy, while Sobolev Training achieves 93.5% final performance. The regular critic also trains well, achieving 93.2%, as the critic forces the lower part of the network to provide a representation which it can use to reduce the classification (and not just prediction) error. Consequently it provides a learning signal which is well aligned with the main optimisation. However, this can lead to building representations which are suboptimal for the rest of the network. Adding additional gradient supervision by constructing our Sobolev SG module avoids this issue by making sure that synthetic gradients are truly aligned and gives an additional boost to the final accuracy.

For ImageNet [3] experiments based on ResNet50 [6], we obtain qualitatively similar results. Due to the complexity of the model and an almost 40% gap between no backpropagation and full backpropagation results, the difference between methods with vs without loss supervision grows significantly. This suggests that at least for ResNet-like architectures, loss supervision is a crucial component of a SG module. After splitting ResNet50 into four parts the Sobolev SG achieves 87.4% top 5 accuracy, while the regular critic SG achieves 86.9%, confirming our claim about suboptimal representation being enforced by gradients from a regular critic. Sobolev Training results were also much more reliable in all experiments (significantly smaller standard deviation of the results).

5 Discussion and Conclusion

In this paper we have introduced Sobolev Training for neural networks – a simple and effective way of incorporating knowledge about derivatives of a target function into the training of a neural network function approximator. We provided theoretical justification that encoding both a target function's value as well as its derivatives within a ReLU neural network is possible, and that this results in more data efficient learning. Additionally, we show that our proposal can be efficiently trained using stochastic approximations if computationally expensive Jacobians or Hessians are encountered.

In addition to toy experiments which validate our theoretical claims, we performed experiments to highlight two very promising areas of applications for such models: one being distillation/compression of models; the other being the application to various meta-optimisation techniques that build models of other models dynamics (such as synthetic gradients, learning-to-learn, etc.). In both cases we obtain significant improvement over classical techniques, and we believe there are many other application domains in which our proposal should give a solid performance boost.

In this work we focused on encoding true derivatives in the corresponding ones of the neural network. Another possibility for future work is to encode information which one believes to be highly correlated with derivatives. For example curvature [17] is believed to be connected to uncertainty. Therefore, given a problem with known uncertainty at training points, one could use Sobolev Training to match the second order signal to the provided uncertainty signal. Finite differences can also be used to approximate gradients for black box target functions, which could help when, for example, learning a generative temporal model. Another unexplored path would be to apply Sobolev Training to *internal derivatives* rather than just derivatives with respect to the inputs.

[4] N.b. the experiments presented use learning rates, annealing schedule, etc. optimised to maximise the backpropagation baseline, rather than the synthetic gradient decoupled result (details in the SM).

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] DeepMind. Sonnet. https://github.com/deepmind/sonnet. 2017.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[4] A Ronald Gallant and Halbert White. On learning the derivatives of an unknown mapping with multilayer feedforward networks. *Neural Networks*, 5(1):129–138, 1992.

[5] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[7] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[8] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[9] Aapo Hyvärinen. Estimation of non-normalized statistical models using score matching. *Journal of Machine Learning Research*, pages 695–709, 2005.

[10] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. *arXiv preprint arXiv:1608.05343*, 2016.

[11] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *NIPS*, volume 13, pages 1008–1014, 1999.

[12] Steven G Krantz. *Handbook of complex variables*. Springer Science & Business Media, 2012.

[13] W Thomas Miller, Paul J Werbos, and Richard S Sutton. *Neural networks for control*. MIT press, 1995.

[14] Takeru Miyato, Daisuke Okanohara, Shin-ichi Maeda, and Koyama Masanori. Synthetic gradient methods with virtual forward-backward networks. *ICLR workshop proceedings*, 2017.

[15] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[17] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

[18] Salah Rifai, Grégoire Mesnil, Pascal Vincent, Xavier Muller, Yoshua Bengio, Yann Dauphin, and Xavier Glorot. Higher order contractive auto-encoder. *Machine Learning and Knowledge Discovery in Databases*, pages 645–660, 2011.

[19] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[20] Bharat Bhusan Sau and Vineeth N Balasubramanian. Deep model compression: Distilling knowledge from noisy teachers. *arXiv preprint arXiv:1610.09650*, 2016.

[21] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[22] Patrice Simard, Bernard Victorri, Yann LeCun, and John S Denker. Tangent prop-a formalism for specifying selected invariances in an adaptive network. In *NIPS*, volume 91, pages 895–903, 1991.

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

[24] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

[25] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011.

[26] Paul J Werbos. Approximate dynamic programming for real-time control and neural modeling. *Handbook of intelligent control*, 1992.

[27] Anqi Wu, Mikio C Aoi, and Jonathan W Pillow. Exploiting gradients and hessians in bayesian optimization and bayesian quadrature. *arXiv preprint arXiv:1704.00060*, 2017.

Supplementary Materials for "Sobolev Training For Neural Networks"

1 Proofs

Theorem 1. *Let f be a $\mathcal{C}^1$ function on a compact set. Then, for every positive ε there exists a single hidden layer neural network with a ReLU (or a leaky ReLU) activation which approximates f in Sobolev space $\mathcal{S}_1$ up to ϵ error.*

We start with a definition. We will say that a function p on a set D is *piecewise-linear*, if there exist $D_1, \ldots, D_n$ such that $D = D_1 \cup \ldots \cup D_n = D$ and $p|_{D_i}$ is linear for every $i = 1, \ldots, n$ (note, that we assume finiteness in the definition).

Lemma 1. *Let D be a compact subset of $\mathbb{R}$ and let $\varphi \in \mathcal{C}^1(D)$. Then, for every $\varepsilon > 0$ there exists a piecewise-linear, continuous function $p : D \to \mathbb{R}$ such that $|\varphi(x) - p(x)| < \varepsilon$ for every $x \in D$ and $|\varphi'(x) - p'(x)| < \varepsilon$ for every $x \in D \setminus P$, where P is the set of points of non-differentiability of p.*

Proof. By assumption, the function φ' is continuous on D. Every continuous function on a compact set has to be uniformly continuous. Therefore, there exists δ_1 such that for every x_1, x_2, with $|x_1 - x_2| < \delta_1$ there holds $|\varphi'(x_1) - \varphi'(x_2)| < \varepsilon$. Moreover, φ' has to be bounded. Let M denote $\sup_x |\varphi'(x)|$. By Mean Value Theorem, if $|x_1 - x_2| < \frac{\varepsilon}{2M}$ then $|\varphi(x_1) - \varphi(x_2)| < \frac{\varepsilon}{2}$. Let $\delta = \min\left\{\delta_1, \frac{\varepsilon}{2M}\right\}$. Let ξ_i, $i = 0, \ldots, N$ be a sequence satisfying: $\xi_i < \xi_j$ for $i < j$, $|\xi_i - \xi_{i-1}| < \delta$ for $i = 1, \ldots, N$ and $\xi_0 < x < \xi_N$ for all $x \in D$. Such sequence obviously exists, because D is a compact (and thus bounded) subset of $\mathbb{R}$. We define

$$p(x) = \varphi(\xi_{i-1}) + \frac{\varphi(\xi_i) - \varphi(\xi_{i-1})}{\xi_i - \xi_{i-1}}(x - \xi_{i-1}) \quad \text{for} \quad \mathrm{x} \in [\xi_{\mathrm{i}-1}, \xi_{\mathrm{i}}] \cap \mathrm{D}.$$

It can be easily verified, that it has all the desired properties. Indeed, let $x \in D$. Let i be such that $\xi_{i-1} \leq x \leq \xi_i$. Then $|\varphi(x) - p(x)| = |\varphi(x) - \varphi(\xi_i) + p(\xi_i) - p(x)| \leq |\varphi(x) - \varphi(\xi_i)| + |p(\xi_i) - p(x)| \leq \varepsilon$, as $\varphi(\xi_i) = p(\xi_i)$ and $|\xi_i - x| \leq |\xi_i - \xi_{i-1}| < \delta$ by definitions. Moreover, applying Mean Value Theorem we get that there exists $\zeta \in [\xi_{i-1}, \xi_i]$ such that $\varphi'(\zeta) = \frac{\varphi(\xi_i) - \varphi(\xi_{i-1})}{\xi_i - \xi_{i-1}} = p'(\zeta)$. Thus, $|\varphi'(x) - p'(x)| = |\varphi'(x) - \varphi'(\zeta) + p'(\zeta) - p'(x)| \leq |\varphi'(x) - \varphi(\zeta)| + |p'(\zeta) - p'(x)| \leq \varepsilon$ as $p'(\zeta) = p'(x)$ and $|\zeta - x| < \delta$.

□

Lemma 2. *Let $\varphi \in \mathcal{C}^1(\mathbb{R})$ have finite limits $\lim_{x \to -\infty} \varphi(x) = \varphi_-$ and $\lim_{x \to \infty} \varphi(x) = \varphi_+$. Then, for every $\varepsilon > 0$ there exists a piecewise-linear, continuous function $p : \mathbb{R} \to \mathbb{R}$ such that $|\varphi(x) - p(x)| < \varepsilon$ for every $x \in \mathbb{R}$ and $|\varphi'(x) - p'(x)| < \varepsilon$ for every $x \in \mathbb{R} \setminus P$, where P is the set of points of non-differentiability of p.*

Proof. By definition of a limit there exist numbers $K_- < K_+$ such that $x < K_- \Rightarrow |\varphi(x) - \varphi_-| \leq \frac{\varepsilon}{2}$ and $x > K_+ \Rightarrow |\varphi(x) - \varphi_+| \leq \frac{\varepsilon}{2}$. We apply Lemma 1 to the function φ and the set $D = [K_-, K_+]$. We define $\tilde{p}$ on $[K_-, K_+]$ according to Lemma 1. We define p as

$$p(x) = \begin{cases} \varphi_- & \text{for} \quad x \in [-\infty, K_-] \\ \tilde{p}(x) & \text{for} \quad x \in [K_-, K_+] \\ \varphi_+ & \text{for} \quad x \in [K_+, \infty] \end{cases}.$$

It can be easily verified, that it has all the desired properties. □

Corollary 1. *For every $\varepsilon > 0$ there exists a combination of ReLU functions which approximates a sigmoid function with accurracy ε in the Sobolev space.*

Proof. It follows immediately from Lemma 2 and the fact, that any piecewise-continuous function on $\mathbb{R}$ can be expressed as a finite sum of ReLU activations. □

Remark 1. *The authors decided, for the sake of clarity and better readability of the paper, to not treat the issue of non-differentiabilities of the piecewise-linear function at the junction points. It can be approached in various ways, either by noticing they form a finite, and thus a zero-Lebesgue measure set and invoking the formal definition f Sobolev spaces, or by extending the definition of a derivative, but it leads only to non-interesting technical complications.*

Proof of Theorem 1. By Hornik's result (Hornik [8]) there exists a combination of N sigmoids approximating the function f in the Sobolev space with $\frac{\varepsilon}{2}$ accuracy. Each of those sigmoids can, in turn, be approximated up to $\frac{\varepsilon}{2N}$ accuracy by a finite combination of ReLU (or leaky ReLU) functions (Corollary 1), and the theorem follows. □

Theorem 2. *Let f be a $\mathcal{C}^1(S)$. Let g be a continuous function satisfying $\|g - \frac{\partial f}{\partial x}\| > 0$. Then, there exists an $\varepsilon = \varepsilon(f, g)$ such that for any $\mathcal{C}^1$ function h there holds either $\|f - h\| \geq \varepsilon$ or $\left\|g - \frac{\partial h}{\partial x}\right\| \geq \varepsilon$.*

Proof. Assume that the converse holds. This would imply, that there exists a sequence of functions h_n such that $\lim_{n\to\infty} \frac{\partial h_n}{\partial x} = g$ and $\lim_{n\to\infty} h_n = f$. But the convergence of h_n at any point implies that $\frac{\partial}{\partial x}\left(\lim_{n\to\infty} h_n\right) = \frac{\partial f}{\partial x}$. However, $\frac{\partial}{\partial x}\left(\lim_{n\to\infty} h_n\right) = \lim_{n\to\infty} \frac{\partial h_n}{\partial x} = g$, contradicting $\|g - \frac{\partial f}{\partial x}\| > 0$. □

Proposition 1. *Given any two functions $f : S \to \mathbb{R}$ and $g : S \to \mathbb{R}^d$ on $S \subseteq \mathbb{R}^d$ and a finite set $\Sigma \subset S$, there exists neural network h with a ReLU (or a leaky ReLU) activation such that $\forall x \in \Sigma : f(x) = h(x)$ and $g(x) = \frac{\partial h}{\partial x}(x)$ (it has 0 training loss).*

Proof. We first prove the theorem in a special, 1-dimensional case (when S is a subset of $\mathbb{R}$). Form now it will be assumed that S is a subset of $\mathbb{R}$ and $\Sigma = \{\sigma_1 < \ldots < \sigma_n\}$ is a finite subset of S. Let ε be smaller than $\frac{1}{5}\min(s_i - s_{i-1})$, $i = 2, \ldots, n$. We define a function p_i as follows

$$p_i(x) = \begin{cases} \frac{f(\sigma_i) - g(\sigma_i)\varepsilon}{\varepsilon}(x - \sigma_i + 2\varepsilon) & \text{for} \quad x \in [\sigma_i - 2\varepsilon, \sigma_i - \varepsilon] \\ f(\sigma_i) + g(\sigma_i)(x - \sigma_i) & \text{for} \quad x \in [\sigma_i - \varepsilon, \sigma_i + \varepsilon] \\ -\frac{f(\sigma_i) + g(\sigma_i)\varepsilon}{\varepsilon}(x - \sigma_i - 2\varepsilon) & \text{for} \quad x \in [\sigma_i + \varepsilon, \sigma_i + 2\varepsilon] \\ 0 & \text{otherwise} \end{cases}.$$

Note that the functions p_i have disjoint supports for $i \neq j$. We define $h(x) = \sum_{i=1}^n p_i(x)$. By construction, it has all the desired properties.

Now let us move to the general case, when S is a subset of $\mathbb{R}^d$. We will denote by π_k a projection of a d-dimensional point σ onto the k-th coordinate. The obstacle to repeating the 1-dimensional proof in a straightforward matter (coordinate-by-coordinate) is that two or more of the points σ_i can have one or more coordinates equal. We will use a linear change of coordinates to get past this technical obstacle. Let $A \in GL(d, \mathbb{R})$ be matrix such that there holds $\pi_k(A\sigma_i) \neq \pi_k(A\sigma_j)$ for any $i \neq j$ and any $K = 1, \ldots, d$. Such A exists, as every condition $\pi_k(A\sigma_i) = \pi_k(A\sigma_j)$ defines a codimension-one submanifold in the space $GL(d, \mathbb{R})$, thus the complement of the union of all such submanifolds is a full dimension (and thus nonempty) subset of $GL(d, \mathbb{R})$. Using the one-dimensional construction we define functions $p^k(x)$, $k = 1, \ldots, d$, such that $p^k(\pi_k(A\sigma_i)) = \frac{1}{d} f(\sigma_i)$ and $(p^k)'(\pi_k(A\sigma_i)) = 0$. Similarly, we construct $q^k(x)$ in such manner $q^k(\pi_k(A\sigma_i)) = 0$ and $(q^k)'(\pi_k(A\sigma_i)) = A^{-1}g(\sigma_i)$. Note that those definitions a are valid because $\pi_k(A\sigma_i) \neq \pi_k(A\sigma_j)$ for $i \neq j$, so the right sides are well-defined unique numbers.

It remains to put all the elements together. This is done as follows. First we extend p^k, q^k to the whole space $\mathbb{R}$ "trivially", i.e. for any $\mathbf{x} \in \mathbb{R}$, $\mathbf{x} = (x^1, \ldots, x^d)$ we define $P^k(\mathbf{x}) := p^k(x^k)$. Similarly, $Q_i^k(\mathbf{x}) := q_i^k(x^k)$. Finally, $h(\mathbf{x}) := \sum_{k=1}^d P^k(A\mathbf{x}) + \sum_{k=1}^d Q^k(A\mathbf{x})$. This function has the desired properties. Indeed for every σ_i we have

$$h(\sigma_i) = \sum_{k=1}^d P^k(A\sigma_i) + \sum_{k=1}^d Q^k(A\sigma_i) = \sum_{k=1}^d p^k(\pi_k(A\sigma_i)) + \sum_{k=1}^d 0 = f(A\sigma_i)$$

and

$$\frac{\partial h}{\partial x}(\sigma_i) = \sum_{k=1}^d (P^k)'(A\sigma_i) + \sum_{k=1}^d (Q^k)'(A\sigma_i) = \sum_{k=1}^d 0 + \sum_{k=1}^d \frac{\partial Q^k}{\partial x}(\pi_k(A\sigma_i)) =$$

$$A\sum_{k=1}^d (0, \ldots, \underbrace{(q^k)'(\pi_k(A\sigma_i))}_{k}, \ldots, 0)^T = A \cdot A^{-1} g(\sigma_i) = g(\sigma_i).$$

□

This completes the proof.

Proposition 3. *There holds $K_{sob}(\mathcal{F}_{\mathrm{G}}) < K_{reg}(\mathcal{F}_{\mathrm{G}})$ and $K_{sob}(\mathcal{F}_{\mathrm{PL}}) < K_{reg}(\mathcal{F}_{\mathrm{PL}})$.*

Proof. Gaussian PDF functions form a 2-parameter family $\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. Therefore, determining f in that family is equivalent to determining the values of μ and σ^2. Given $\alpha = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, $\beta = -\frac{x-\mu}{\sigma^2\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, we get $\frac{\beta}{\alpha} = -\frac{x-\mu}{\sigma^2}$ and $2\ln(\sqrt{2\pi}\alpha) = -\ln(\sigma^2) - \frac{(x-\mu)^2}{\sigma^2}$. Thus $2\ln(\sqrt{2\pi}\alpha) =$

$-\ln(\sigma^2) - \frac{\beta^2}{\alpha^2}\sigma^2$. The right hand side is a strictly decreasing function of σ^2. Substituting its unique solution to $\frac{\beta}{\alpha} = -\frac{x-\mu}{\sigma^2}$ we determine μ. Thus K_{sob} is equal to 1 for the family of Gaussian PDF functions.

On the other hand, there holds $K_{reg} > 2$ for the family of Gaussian PDF functions. For example, $N(2,1)$ and $N(2.847..., 1.641...)$ have the same values at $x = 0$ and $x = 3$ (existence of a "real" solution near this approximate solution is an immediate consequence of the Implicit Function Theorem). This ends the proof for the $\mathcal{F}_{\mathrm{G}}$ family

We will discuss the family $\mathcal{F}_{\mathrm{PL}}$ now. Every linear function is uniquely determined by its value at a single point and its derivative. Thus, for any function $f \in \mathcal{F}_{\mathrm{PL}}$, as the partition $D = D_1 \cup \ldots \cup D_n$ is fixed, it is sufficient to know the values and the values of the derivative of f in $\sigma_1 \in D_n, \ldots, \sigma_1 \in D_n$ to determine it uniquely. On the other hand, we need at least $d + 1$ (recall that d is the dimension of the domain of f) in each of the domains D_i to determine f uniquely, if we are allowed to look only at the values.

□

2 Artificial Datasets

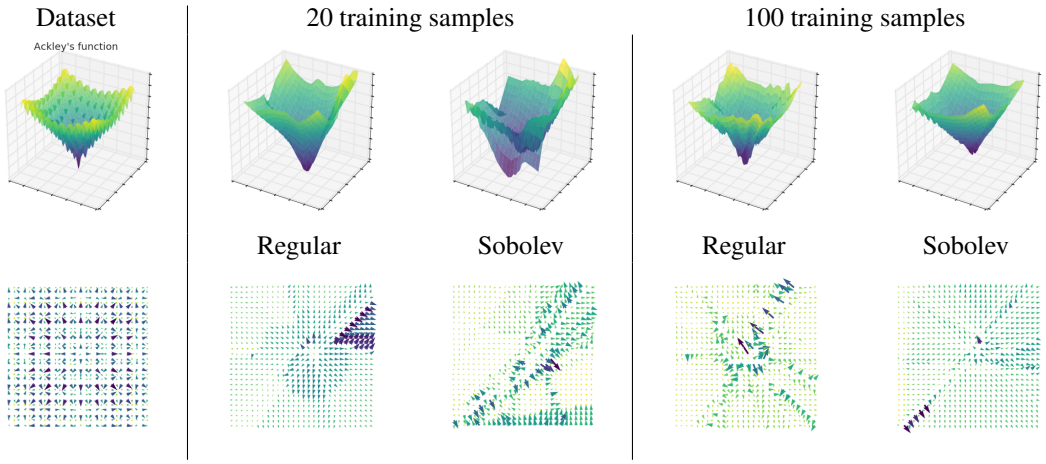


Figure 5: Ackley function (on the left) and its models using regular neural network training (left part of each plot) and Sobolev Training (right part). We also plot the vector field of the gradients of each predictor underneath the function plot.

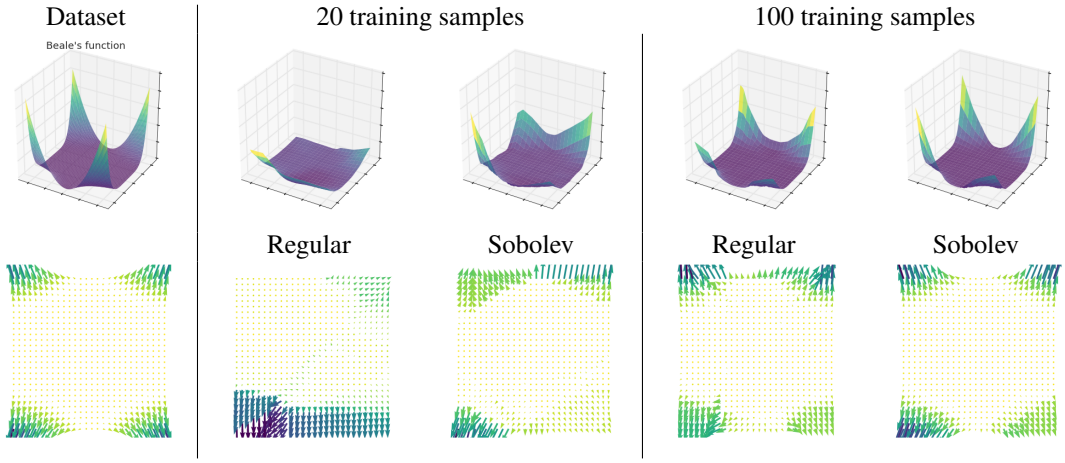


Figure 6: Beale function (on the left) and its models using regular neural network training (left part of each plot) and Sobolev Training (right part). We also plot the vector field of the gradients of each predictor underneath the function plot.

Functions used (visualised at Figures 5-11):

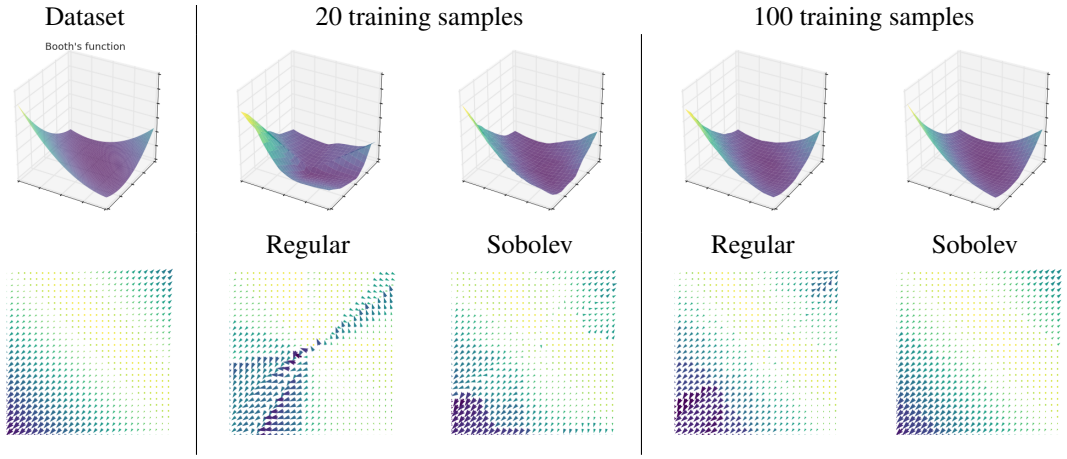


Figure 7: Booth function (on the left) and its models using regular neural network training (left part of each plot) and Sobolev Training (right part). We also plot the vector field of the gradients of each predictor underneath the function plot.

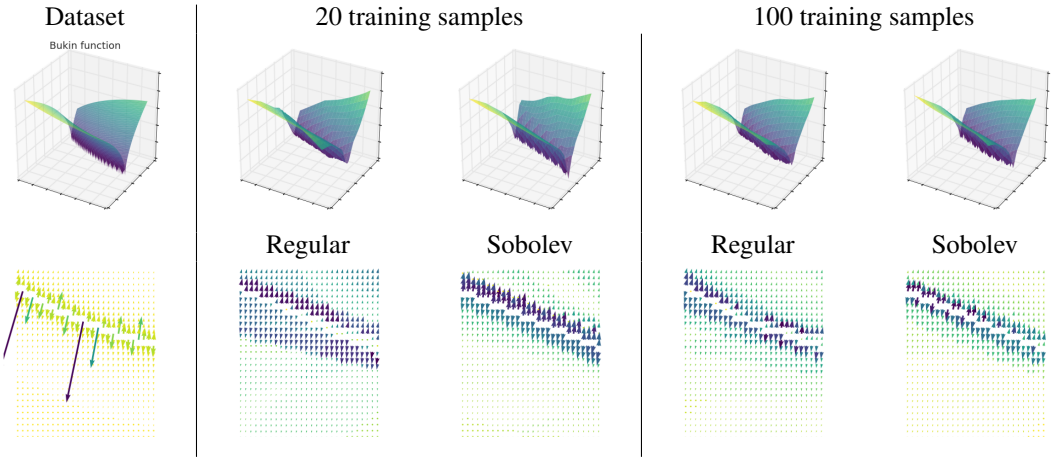


Figure 8: Bukin function (on the left) and its models using regular neural network training (left part of each plot) and Sobolev Training (right part). We also plot the vector field of the gradients of each predictor underneath the function plot.

- Ackley's

$$f(x,y) = -20\exp\left(-0.2\sqrt{0.5(x^2+y^2)}\right) - \exp\left(0.5(\cos(2\pi x)+\cos(2\pi y))\right) + e + 20,$$

  for $x, y \in [-5,5] \times [-5,5]$

- Beale's

$$f(x,y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2,$$

  for $x, y \in [-4.5,4.5] \times [-4.5,4.5]$

- Booth

$$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2,$$

  for $x, y \in [-10,10] \times [-10,10]$

- Bukin

$$f(x,y) = 100\sqrt{|y = 0.01x^2|} + 0.01|x + 10|,$$

  for $x, y \in [-15,-5] \times [-3,3]$

- McCormick

$$f(x,y) = \sin(x+y) + (x-y)^2 - 1.5x + 2.5y + 1,$$

  for $x, y \in [-1.5,4] \times [-3,4]$

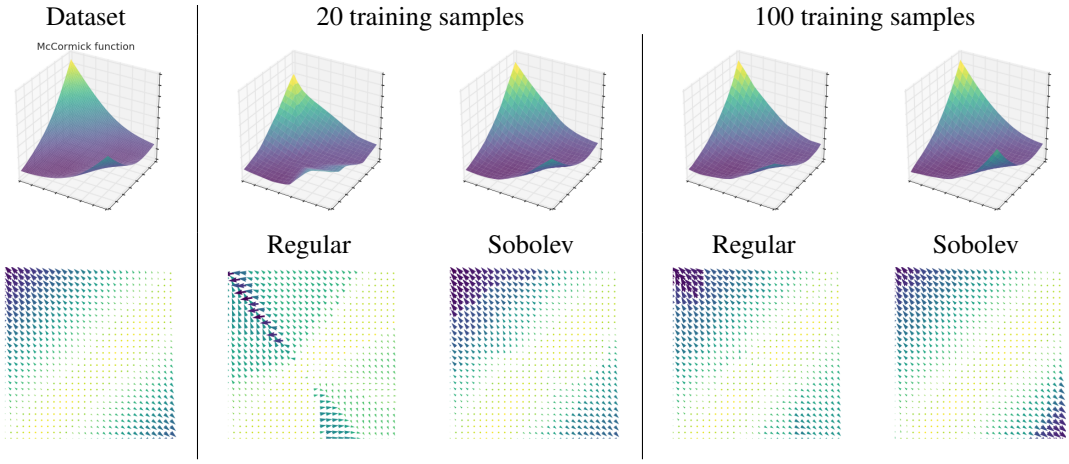


Figure 9: McCormick function (on the left) and its models using regular neural network training (left part of each plot) and Sobolev Training (right part). We also plot the vector field of the gradients of each predictor underneath the function plot.

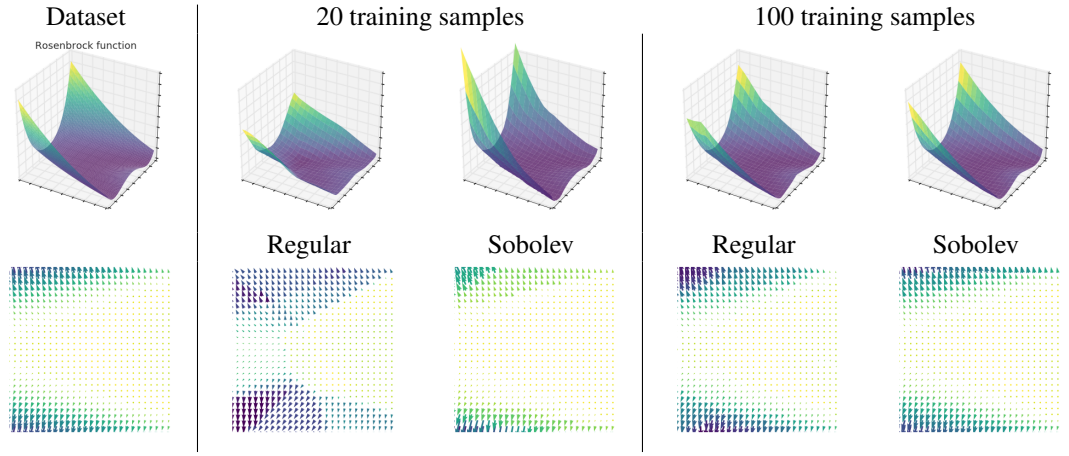


Figure 10: Rosenbrock function (on the left) and its models using regular neural network training (left part of each plot) and Sobolev Training (right part). We also plot the vector field of the gradients of each predictor underneath the function plot.

- Rosenbrock
$$f(x, y) = 100(y - x^2)^2 + (x - 1)^2,$$
for $x, y \in [-2, 2] \times [-2, 2]$
- Styblinski-Tang
$$f(x, y) = 0.5(x^4 - 16x^2 + 5x + y^4 - 16y^2 + 5y),$$
for $x, y \in [-5, 5] \times [-5, 5]$

Networks were trained using the Adam optimiser with learning rate $3e - 5$. Training set has been sampled uniformly from the domain provided. Test set consists always of 10,000 points sampled uniformly from the same domain.

3 Policy Distillation

Agents policies are feed forward networks consisting of:

- 32 8x8 kernels with stride 4
- ReLU nonlinearity

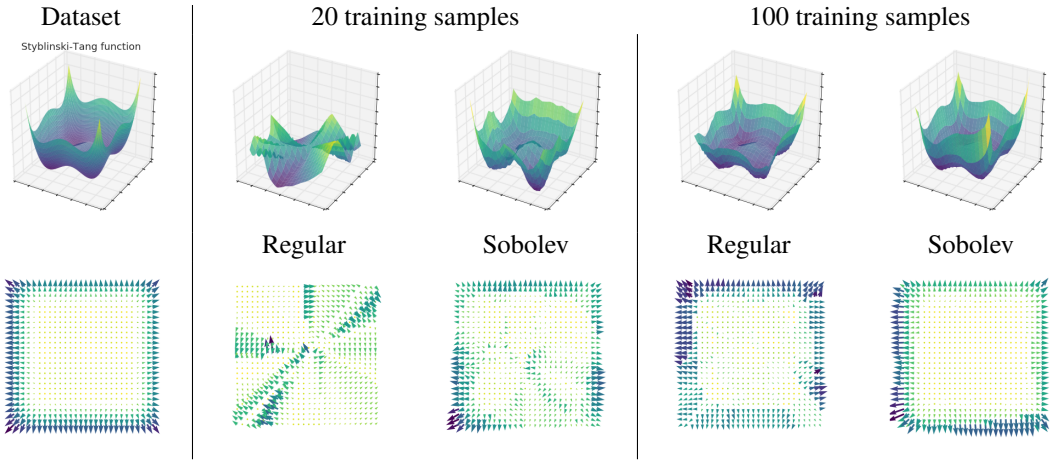


Figure 11: Styblinski-Tang function (on the left) and its models using regular neural network training (left part of each plot) and Sobolev Training (right part). We also plot the vector field of the gradients of each predictor underneath the function plot.

- 64 4x4 kernels with stride 2
- ReLU nonlinearity
- 64 3x3 kernels with stride 1
- ReLU nonlinearity
- Linear layer with 512 units
- ReLU nonlinearity
- Linear layer with 3 (Pong), 4 (Breakout) or 6 outputs (Space Invaders)
- Softmax

They were trained with A3C [15] over 80e6 steps, using history of length 4, greyscaled input, and action repeat 4. Observations were scaled down to 84x84 pixels.

Data has been gathered by running trained policy to gather 100K frames (thus for 400K actual steps). Split into train and test sets has been done time-wise, ensuring that test frames come from different episodes than the training ones.

Distillation network consists of:

- 16 8x8 kernels with stride 4
- ReLU nonlinearity
- 32 4x4 kernels with stride 2
- ReLU nonlinearity
- Linear layer with 256 units
- ReLU nonlinearity
- Linear layer with 3 (Pong), 4 (Breakout) or 6 outputs (Space Invaders)
- Softmax

and was trained using Adam optimiser with learning rate fitted independently per game and per approach between $1e-3$ and $1e-5$. Batch size is 200 frames, randomly selected from the training set.

4 Synthetic Gradients

All models were trained using multi-GPU optimisation, with Sync main network updates and Hogwild SG module updates.

4.1 Meaning of Sobolev losses for synthetic gradients

In the setting considered, the true label y is used only as a conditioning, however one could also provide supervision for $\partial m(h, y|\theta)/\partial y$. So what is the actual effect this Sobolev losses have on SG estimator? For L being log loss, it is easy to show, that they are additional penalties on matching $\log p(h, y)$ to $\log p_h$, namely:

$$\|\partial m(h, y|\theta)/\partial y - \partial L(h, y)/\partial y\|^2 = \|\log p(h|\theta) - \log p_h\|^2$$

$$\|m(h, y|\theta) - L(h, y)\|^2 = (\log p(h|\theta)_{\hat{y}} - \log p_{h\,\hat{y}})^2,$$

where $\hat{y}$ is the index of "1" in the one-hot encoded label vector y. Consequently loss supervision makes sure that the internal prediction $\log p(h|\theta)$ for the true label $\hat{y}$ is close to the current prediction of the whole model $\log p_h$. On the other hand matching partial derivatives wrt. to label makes sure that predictions for all the classes are close to each other. Finally if we use both – we get a weighted sum, where penalty for deviating from the prediction on the true label is more expensive, than on all remaining ones[5].

4.2 Cifar10

All Cifar10 experiments use a deep convolutional network of following structure:

- 64 3x3 kernels with stride 1
- BatchNorm and ReLU nonlinearity
- 64 3x3 kernels with stride 1
- BatchNorm and ReLU nonlinearity
- 128 3x3 kernels with stride 2
- BatchNorm and ReLU nonlinearity
- 128 3x3 kernels with stride 1
- BatchNorm and ReLU nonlinearity
- 128 3x3 kernels with stride 1
- BatchNorm and ReLU nonlinearity
- 256 3x3 kernels with stride 2
- BatchNorm and ReLU nonlinearity
- 256 3x3 kernels with stride 1
- BatchNorm and ReLU nonlinearity
- 256 3x3 kernels with stride 1
- BatchNorm and ReLU nonlinearity
- 512 3x3 kernels with stride 2
- BatchNorm and ReLU nonlinearity
- 512 3x3 kernels with stride 1
- BatchNorm and ReLU nonlinearity
- 512 3x3 kernels with stride 1
- BatchNorm and ReLU nonlinearity
- Linear layer with 10 outputs
- Softmax

with L2 regularisation of $1e-4$. The network is trained in an asynchronous manner, using 10 GPUs in parallel. Each worker uses batch size of 32. The main optimiser is Stochastic Gradient Descent with momentm of 0.9. The learning rate is initialised to 0.1 and then dropped by an order of magniture after 40K, 60K and finally after 80K updates.

Each of the three SG modules is a convolutional network consisting of:

- 128 3x3 kernels with stride 1

[5] Adding $\partial L/\partial y$ supervision on toy MNIST experiments increased convergence speed and stability, however due to TensorFlow currently not supporting differentiating cross entropy wrt. to labels, it was omitted in our large-scale experiments.

- ReLU nonlinearity
- Linear layer with 10 outputs
- Softmax

It is trained using the Adam optimiser with learning rate $1e-4$, no learning rate schedule is applied. Updates of the synthetic gradient module are performed in a Hogwild manner. Losses used for both loss prediction and gradient estimation are L1.

For direct SG model we used architecture described in the original paper – 3 resolution preserving layers of 128 kernels of 3x3 convolutions with ReLU activations in between. The only difference is that we use L1 penalty instead of L2 as empirically we found it working better for the tasks considered.

4.3 Imagenet

All ImageNet experiments use ResNet50 network with L2 regularisation of $1e-4$. The network is trained in an asynchronous manner, using 34 GPUs in parallel. Each worker uses batch size of 32. The main optimiser is Stochastic Gradient Descent with momentum of 0.9. The learning rate is initialised to 0.1 and then dropped by an order of magnitude after 100K, 150K and finally after 175K updates.

The SG module is a convolutional network, attached after second ResNet block, consisting of:

- 64 3x3 kernels with stride 1
- ReLU nonlinearity
- 64 3x3 kernels with stride 2
- ReLU nonlinearity
- Global averaging
- 1000 1x1 kernels
- Softmax

It is trained using the Adam optimiser with learning rate $1e-4$, no learning rate schedule is applied. Updates of the synthetic gradient module are performed in a Hogwild manner. Sobolev losses are set to L1.

Regular data augmentation has been applied during training, taken from the original Inception V1 paper.