Sobolev Training for Neural Networks

Authored by:

Simon Osindero
Grzegorz Swirszcz
Razvan Pascanu
Max Jaderberg
Wojciech M. Czarnecki

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   Paper Body

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 [8, 24], through audio synthesis [27], to human-beating policies in the ancient game of GO [22]. In many applications the process of training a neural network consists

1

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 [9, 20], compressing neural networks [7], and the prediction of synthetic gradients [12]. 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 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

D_{mathbf{x}} f
Dx hDx hm, v1 i, v2 i Dx2 m
l2
@ @x
Dx m
@ @x
l1
Dx f
l
@ @x
@ @x
l1
Dx hf, v1 i @ @x
v1 m
f
Dx hDx hf, v1 i, v2 i
v2
mv1 = Dx hm, v1 i
@ @x
@ @x
m
@ @x
Dx2 f
l2
l
x
x

a)

b)

f

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, l1 , and l2 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 vj sampled from the unit sphere. $D_{\mathbf{x}} \langle D_{\mathbf{x}} \langle m, v_1 \rangle, v_2 \rangle$

propose Sobolev Training (ST) for neural networks as a simple and efficient technique for leveraging 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 [10] 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. [30] 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 [19]), or to encode additional, hand crafted invariances to some transformations (for instance, as in Tangentprop [23]), or estimated derivatives for dynamical systems [6] and very recently to provide additional learning signal during attention distillation [31]1 . 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 [29, 15, 5, 4, 26] using small, sigmoid based models. Finally, Hyv?rinen proposed Score Matching Networks [11], 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 [28], 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, especially modern ReLU based models. 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 1

Please relate to Supplementary Materials, section 5 for details

2

Sobolev Training ? a new paradigm for training neural networks. (2): We look formally at the implications of matching derivatives, extending previous results of Hornik [10] 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 (xi) for training points xi , but also the values of its j-th order derivatives with respect to the input, Dxj f (xi). In other words, instead of the typical training set consisting of pairs {(xi , f (xi))}N i=1 we have access to (K + 2)-tuples {(xi , f (xi), Dx1 f (xi), ..., DxK f (xi))}N i=1 . 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 ?, one typically seeks to minimise the empirical error in relation to f according to some loss function ' N X i=1

'(m(xi —?), f (xi)).

When learning in Sobolev spaces, this is replaced with: ? ? N K X X

¿(m(xi —?), f (xi)) + 'j Dxj m(xi —?), Dxj f (xi) ? , i=1

(1)

j=1

where '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 [19]). 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: ? ? N K X X

j

¿(m(xi —?), f (xi)) + Evj 'j Dx m(xi —?), v j , Dxj f (xi), v j ? . (2) i=1

j=1

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 [10] 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 3

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. 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 C 1 (S) (or simply C 1) to denote a space of functions which are continuous, differentiable, and have a continuous derivative on a space S [14]. All proofs are given in the Supplementary Materials (SM). Theorem 1. Let f be a C 1 function on a compact set. Then, for every positive ? there exists a single hidden layer neural network with a ReLU (or a leaky ReLU) activation which approximates f in Sobolev space S1 up to 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 C 1 function. Let g be a continuous function satisfying kg ? ?f ?x k?

¿ 0. Then,

there exists an ? ¿ 0 such that for any C 1 function h either kf ? hk? ? ? or g ? ?h ?x ? ? ?. However, when we move to the regime of finite training data, we can encode any arbitrary function in the derivatives (as well as higher

5

order signals if the resulting Sobolev spaces are not degenerate), as shown in the following Proposition. Proposition 1. Given any two functions f : S ? R and g : S ? Rd on S ? Rd and a finite set ? ? S, there exists neural network h with a ReLU (or a leaky ReLU) activation such that ?x ? ? : f (x) = h(x) and g(x) = ?h ?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 F denote the family of functions parametrised by ?. We define Kreg = Kreg (F) to be a measure of the amount of data needed to learn some target function f . That is Kreg is the smallest number for which there holds: for every f? ? F and every set of distinct Kreg points (x1 , ..., xKreg) such that ?i=1,...,Kreg f (xi) = f? (xi) ? f = f? . Ksob is defined analogously, but the final implication is of ?f? form f (xi) = f? (xi) ? ?f ?x (xi) = ?x (xi) ? f = f? . Straight from the definition there follows: Proposition 2. For any F, there holds Ksob (F) ? Kreg (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 d n2 e points suffice to determine all the coefficients. We present two more examples in a slightly more formal way. Let FG denote a family of Gaussian PDF-s (parametrised by ?, ?). Let Rd ? D = D1 ? . . . ? Dn and let FPL be a family of functions from D1 ? ... ? Dn (Cartesian product of sets Di) to Rn of form f (x) = [A1 x1 + b1 , . . . , An xn + bn] (linear element-wise) (Figure 2 Left). 4

Dataset
20 training samples
Regular
Sobolev
100 training samples
Regular
Sobolev

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. Proposition 3. There holds Ksob (FG) ¡ Kreg (FG) and Ksob (FPL) ¡ Kreg (FPL). 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 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 training2 . 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 N1 ?

N X i=1

kf (xi) ? m(xi —?)k22 + k?x f (xi) ? ?x m(xi —?)k22 .

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 2

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

5

Test DKL

Test action prediction error

Regular distillation

Sobolev distillation

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 DKL (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. 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. 4.2

Distillation

Another possible application of Sobolev Training is to perform model distillation. This technique has many applications, such as network compression [21], ensemble merging [9], or more recently policy distillation in reinforcement learning [20]. We focus here on a task of distilling a policy. We aim to distill a target policy ? ? (s) ? a trained neural network which outputs a probability distribution over actions ? into a smaller neural network ?(s—?), such that the two policies ? ? and ? have the same behaviour. In practice this is often done by minimising an expected divergence measure between ? ? and ?, for example, the Kullback?Leibler divergence DKL (?(s)k? ? (s)), over states gathered while following ? ? . 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 DKL (?(s—?)k? ? (s)) + ?Ev [k?s hlog ? ? (s), vi ? ?s hlog ?(s—?), vik] , ?

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 ? ? , we use agents playing Atari games [17] that have been trained with A3C [16] 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 Training3 . The introduction of Sobolev Training leads to similar effects as in the previous section ? the network generalises much more effectively, and this 3

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

6

xx

x x yxy

y y y xx

(a)(a) (a)(a)(a)

x x yxy

Synthetic error gradient error gradient Synthetic error gradient Synthetic error gradient Synthetic error gradient y y ySynthetic

(b) (b)(b)(b) (b)

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 ? standard deviation. Backpropagation results are given in parenthesis.

00

SG(h, SG(h, y) y) SG(h, SG(h, y) SG(h, y)SG(h, SG(h, y)SG(h, y) y)y)y) SG(h, y)SG(h, y) SG(h, y) SG(h, SG(h, y) y) y) y) SG(h, y) SG(h, 0 y) SG(h, 0SG(h, 0SG(h,

@ @ @h @h

@ @ @ @ @h @h @h @h

? ?L L

?L ? L ? ? L L

@ @ @@ @ @h@h @h @h @h

? L ? ? L ?L L

@ @ @ @h @h @h

? ? L ? L L

f (h, y—?) (h, y—?) f (h, f (h, y—?)y—?) f (h,fy—?)

p(h—?) p(h—?) p(h—?) p(h—?) p(h—?) p(h—?) p(h—?) p(h—?) p(h—?) p(h—?) p(h—?) p(h—?) p(h—?)

h yh y Noprop

y hy y hy h hy hy y hy h hy y y y h h h yhhy h y y h hy hhyh y y h Direct SG [12]

VFBN [25]

Critic

Sobolev

CIFAR-10 with 3 synthetic gradient modules Top 1 (94.3%) 54.5% ?1.15 79.2% ?0.01

88.5% ?2.70

93.2% ?0.02

93.5% ?0.01

ImageNet with 1 synthetic gradient module Top 1 (75.0%) 54.0% ?0.29 Top 5 (92.3%) 77.3% ?0.06 -

57.9% ?2.03 81.5% ?1.20

71.7% ?0.23 90.5% ?0.15

72.0% ?0.05 90.8% ?0.01

ImageNet with 3 synthetic gradient modules Top 1 (75.0%) 18.7% ?0.18 Top 5 (92.3%) 38.0% ?0.34 -

28.3% ?5.24 52.9% ?6.62

65.7% ?0.56 86.9% ?0.33

66.5% ?0.22 87.4% ?0.11

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 game4 . 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 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) [12] ? 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—?). In other words, given that network parameters up to h are denoted by ? ?L ?h ?h ?L = ? SG(h, y—?) . ?? ?h ?? ??

2

h ,y) In the original SG paper, this module is trained to minimise LSG (?) = SG(h, y—?) ? ?L(p

, ?h 2 where ph 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 4

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.

7

of a class predictor p(?—?) 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—?) := L(p(h—?), y),

SG(h, y—?) := ?m(h, y—?)/?h,

?m(h,y—?) ?L(ph ,y) Lsob (?) = '(m(h, y—?), L(p , y))) + ' . , h 1 SG ?h ?h In the Sobolev Training framework, the target function is the loss of the main network L(ph , y) for which we train a model m(h, y—?) to approximate, and in addition ensure that the model?s derivatives ?m(h, y—?)/?h are matched to the true derivatives ?L(ph , y)/?h. The model?s derivatives ?m(h, y—?)/?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 [13]. 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 [25]. 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). One could ask why we need these additional constraints, and what is gained over using a neural network based

approximator directly [12]. The answer lies in the fact that gradient vector fields are a tiny subset of all vector fields, and while each neural network produces a valid vector field, almost no (standard) neural network produces valid gradient vector fields. Using non-gradient vector fields as update directions for learning can have catastrophic consequences ? learning divergence, oscillations, chaotic behaviour, etc. The following proposition makes this observation more formal: Proposition 4. If an approximator SG(h, y—?) produces a valid gradient vector field of some scalar function L then the approximator?s Jacobian matrix must be symmetric. It is worth noting that having a symmetric Jacobian is an extremely rare property for a neural network model. For example, a linear model has a symmetric Jacobian if and only if its weight matrix is symmetric. If we sample weights iid from typical distribution (like Gaussian or uniform on an interval), the probability of sampling such a matrix is 0, but it could be easy to learn with strong, symmetric-enforcing updates. On the other hand, for highly non-linear neural networks, it is not only improbable to randomly find such a model, but enforcing this constraint during learning becomes much harder too. This might be one of the reasons why linear SG modules work well in Jaderberg et al. [12], but non-linear convolutional SG struggled to achieve state-of-the-art performance. When using Sobolev-like approach SG always produces a valid gradient vector field by construction, thus avoiding the problem described. 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 5 . 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 [8], 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 5

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).

8

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 [18] 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.

2   References

[1] Sonnet. https://github.com/deepmind/sonnet. 2017. [2] Mart?n 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. [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] Michael Fairbank and Eduardo Alonso.

Value-gradient learning. In Neural Networks (IJCNN), The 2012 International Joint Conference on, pages 1?8. IEEE, 2012. [5] Michael Fairbank, Eduardo Alonso, and Danil Prokhorov. Simple and fast calculation of the second-order gradients for globalized dual heuristic dynamic programming in neural networks. IEEE transactions on neural networks and learning systems, 23(10):1671?1676, 2012. [6] 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. [7] 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. [8] 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. [9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015. [10] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. Neural networks, 4(2):251? 257, 1991.

9

[11] Aapo Hyv?rinen. Estimation of non-normalized statistical models using score matching. Journal of Machine Learning Research, pages 695?709, 2005. [12] 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. [13] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In NIPS, volume 13, pages 1008?1014, 1999. [14] Steven G Krantz. Handbook of complex variables. Springer Science & Business Media, 2012. [15] W Thomas Miller, Paul J Werbos, and Richard S Sutton. Neural networks for control. MIT press, 1995. [16] 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. [17] 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. [18] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. arXiv preprint arXiv:1301.3584, 2013. [19] Salah Rifai, Gr?goire Mesnil, Pascal Vincent, Xavier Muller, Yoshua Bengio, Yann Dauphin, and Xavier Glorot. Higher order contractive autoencoder. Machine Learning and Knowledge Discovery in Databases, pages 645?660, 2011. [20] 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. [21] Bharat Bhusan Sau and Vineeth N Balasubramanian. Deep model compression: Distilling knowledge from noisy teachers. arXiv preprint arXiv:1610.09650, 2016. [22] 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. [23] 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. [24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014. [25] Shinichi Maeda Koyama Masanori Takeru Miyato, Daisuke Okanohara. Synthetic gradient methods with virtual forward-backward networks. ICLR workshop proceedings, 2017. [26] Yuval Tassa and Tom Erez. Least squares solutions of the hjb equation with neural network value-function approximators. IEEE transactions on neural networks, 18(4):1031?1041, 2007. [27] A?ron 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. [28] Pascal Vincent. A connection between score matching and denoising autoencoders. Neural computation, 23(7):1661?1674, 2011. [29] Paul J Werbos. Approximate dynamic programming for real-time control and neural modeling. Handbook of intelligent control, 1992. [30] 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. [31] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. arXiv preprint arXiv:1612.03928, 2016.

10