Understanding Synthetic Gradients and Decoupled Neural Interfaces

Wojciech Marian Czarnecki [1] **Grzegorz Swirszcz** [1] **Max Jaderberg** [1] **Simon Osindero** [1] **Oriol Vinyals** [1] **Koray Kavukcuoglu** [1]

Abstract

When training neural networks, the use of Synthetic Gradients (SG) allows layers or modules to be trained without update locking – without waiting for a true error gradient to be backpropagated – resulting in Decoupled Neural Interfaces (DNIs). This unlocked ability of being able to update parts of a neural network asynchronously and with only local information was demonstrated to work empirically in Jaderberg et al. (2016). However, there has been very little demonstration of what changes DNIs and SGs impose from a functional, representational, and learning dynamics point of view. In this paper, we study DNIs through the use of synthetic gradients on feed-forward networks to better understand their behaviour and elucidate their effect on optimisation. We show that the incorporation of SGs does not affect the representational strength of the learning system for a neural network, and prove the convergence of the learning system for linear and deep linear models. On practical problems we investigate the mechanism by which synthetic gradient estimators approximate the true loss, and, surprisingly, how that leads to drastically different layer-wise representations. Finally, we also expose the relationship of using synthetic gradients to other error approximation techniques and find a unifying language for discussion and comparison.

1. Introduction

Neural networks can be represented as a graph of computational modules, and training these networks amounts to optimising the weights associated with the modules of this graph to minimise a loss. At present, training is usually performed with first-order gradient descent style algorithms,

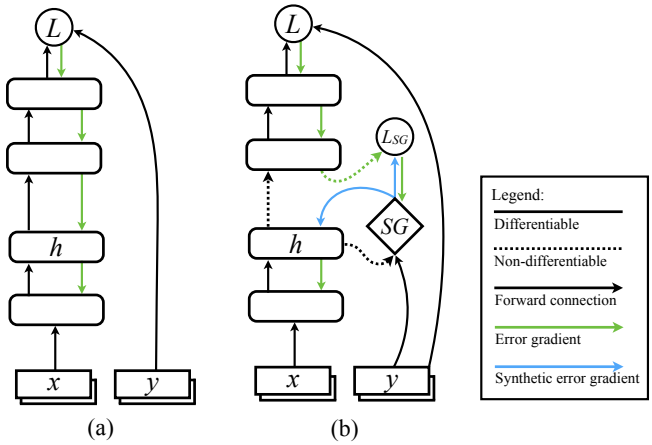


Figure 1. Visualisation of SG-based learning (b) vs. regular backpropagation (a).

where the weights are adjusted along the direction of the negative gradient of the loss. In order to compute the gradient of the loss with respect to the weights of a module, one performs backpropagation (Williams & Hinton, 1986) – sequentially applying the chain rule to compute the exact gradient of the loss with respect to a module. However, this scheme has many potential drawbacks, as well as lacking biological plausibility (Marblestone et al., 2016; Bengio et al., 2015). In particular, backpropagation results in locking – the weights of a network module can only be updated after a full forwards propagation of the data through the network, followed by loss evaluation, then finally after waiting for the backpropagation of error gradients. This locking constrains us to updating neural network modules in a sequential, synchronous manner.

One way of overcoming this issue is to apply Synthetic Gradients (SGs) to build Decoupled Neural Interfaces (DNIs) (Jaderberg et al., 2016). In this approach, models of error gradients are used to approximate the true error gradient. These models of error gradients are local to the network modules they are predicting the error gradient for, so that an update to the module can be computed by using the predicted, synthetic gradients, thus bypassing the need for subsequent forward execution, loss evaluation, and backpropagation. The gradient models themselves are trained

[1]DeepMind, London, United Kingdom. Correspondence to: WM Czarnecki <lejlot@google.com>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

at the same time as the modules they are feeding synthetic gradients to are trained. The result is effectively a complex dynamical system composed of multiple sub-networks cooperating to minimise the loss.

There is a very appealing potential of using DNIs *e.g.* the potential to distribute and parallelise training of networks across multiple GPUs and machines, the ability to asynchronously train multi-network systems, and the ability to extend the temporal modelling capabilities of recurrent networks. However, it is not clear that introducing DNIs and SGs into a learning system will not negatively impact the learning dynamics and solutions found. While the empirical evidence in Jaderberg et al. (2016) suggests that SGs do not have a negative impact and that this potential is attainable, this paper will dig deeper and analyse the result of using SGs to accurately answer the question of the impact of synthetic gradients on learning systems.

In particular, we address the following questions, using feed-forward networks as our probe network architecture: **Does introducing SGs change the critical points of the neural network learning system?** In Section 3 we show that the critical points of the original optimisation problem are maintained when using SGs. **Can we characterise the convergence and learning dynamics for systems that use synthetic gradients in place of true gradients?** Section 4 gives first convergence proofs when using synthetic gradients and empirical expositions of the impact of SGs on learning. **What is the difference in the representations and functional decomposition of networks learnt with synthetic gradients compared to backpropagation?** Through experiments on deep neural networks in Section 5, we find that while functionally the networks perform identically trained with backpropagation or synthetic gradients, the layer-wise functional decomposition is markedly different due to SGs.

In addition, in Section 6 we look at formalising the connection between SGs and other forms of approximate error propagation such as Feedback Alignment (Lillicrap et al., 2016), Direct Feedback Alignment (Nøkland, 2016; Baldi et al., 2016), and Kickback (Balduzzi et al., 2014), and show that all these error approximation schemes can be captured in a unified framework, but crucially only using synthetic gradients can one achieve unlocked training.

2. DNI using Synthetic Gradients

The key idea of synthetic gradients and DNI is to approximate the true gradient of the loss with a learnt model which predicts gradients without performing full backpropagation.

Consider a feed-forward network consisting of N layers $f_n, n \in \{1, \ldots, N\}$, each taking an input h_i^{n-1} and producing an output $h_i^n = f_n(h_i^{n-1})$, where $h_i^0 = x_i$ is the input data point x_i. A loss is defined on the output of the network $L_i = L(h_i^N, y_i)$ where y_i is the given label or supervision for x_i (which comes from some unknown $P(y|x)$). Each layer f_n has parameters θ_n that can be trained jointly to minimise L_i with the gradient-based update rule

$$\theta_n \leftarrow \theta_n - \alpha \frac{\partial L(h_i^N, y_i)}{\partial h_i^n} \frac{\partial h_i^n}{\partial \theta_n}$$

where α is the learning rate and $\partial L_i / \partial h_i^n$ is computed with backpropagation.

The reliance on $\partial L_i / \partial h_i^N$ means that an update to layer i can only occur after every subsequent layer $f_j, j \in \{i+1, \ldots, N\}$ has been computed, the loss L_i has been computed, and the error gradient $\partial L / \partial h_i^N$ backpropgated to get $\partial L_i / \partial h_i^N$. An update rule such as this is *update locked* as it depends on computing L_i, and also *backwards locked* as it depends on backpropagation to form $\partial L_i / \partial h_i^n$.

Jaderberg et al. (2016) introduces a learnt prediction of the error gradient, the *synthetic gradient* $\mathrm{SG}(h_i^n, y_i) = \widehat{\partial L_i / \partial h_i^n} \simeq \partial L_i / \partial h_i^n$ resulting in the update

$$\theta_k \leftarrow \theta_k - \alpha \, \mathrm{SG}(h_i^n, y_i) \frac{\partial h_i^n}{\partial \theta_k} \quad \forall k \leq n$$

This approximation to the true loss gradient allows us to have both update and backwards unlocking – the update to layer n can be applied without any other network computation as soon as h_i^n has been computed, since the SG module is not a function of the rest of the network (unlike $\partial L_i / \partial h_i$). Furthermore, note that since the true $\partial L_i / \partial h_i^n$ can be described completely as a function of just h_i^n and y_i, from a mathematical perspective this approximation is sufficiently parameterised.

The synthetic gradient module $\mathrm{SG}(h_i^n, y_i)$ has parameters θ_{SG} which must themselves be trained to accurately predict the true gradient by minimising the L_2 loss $L_{\mathrm{SG}_i} = \|\mathrm{SG}(h_i^n, y_i) - \partial L_i / \partial h_i^n\|^2$.

The resulting learning system consists of three decoupled parts: first, the part of the network above the SG module which minimises L wrt. to its parameters $\{\theta_{n+1}, ..., \theta_N\}$, then the SG module that minimises the L_{SG} wrt. to θ_{SG}. Finally the part of the network below the SG module which uses $\mathrm{SG}(h, y)$ as the learning signal to train $\{\theta_1, ...\theta_n\}$, thus it is minimising the loss modeled internally by SG.

Assumptions and notation

Throughout the remainder of this paper, we consider the use of a single synthetic gradient module at a single layer k and for a generic data sample j and so refer to $h = h_j = h_j^k$; unless specified we drop the superscript k and subscript j. This model is shown in Figure 1 (b). We also focus on

SG modules which take the point's true label/value as conditioning $\mathrm{SG}(h, y)$ as opposed to $\mathrm{SG}(h)$. Note that without label conditioning, a SG module is trying to approximate not $\partial L/\partial h$ but rather $\mathbb{E}_{P(y|x)}\partial L/\partial h$ since L is a function of both input and label. In theory, the lack of label is a sufficient parametrisation but learning becomes harder, since the SG module has to additionally learn $P(y|x)$.

We also focus most of our attention on models that employ *linear* SG modules, $\mathrm{SG}(h, y) = hA + yB + C$. Such modules have been shown to work well in practice, and furthermore are more tractable to analyse.

As a shorthand, we denote $\theta_{<h}$ to denote the subset of the parameters contained in modules *up to* h (and symmetrically $\theta_{>h}$), *i.e.* if h is the kth layer then $\theta_{<h} = \{\theta_1 \ldots, \theta_k\}$.

Synthetic gradients in operation
Consider an N-layer feed-forward network with a single SG module at layer k. This network can be decomposed into two sub-networks: the first takes an input x and produces an output $h = F_h(x) = f_k(f_{k-1}(\ldots(f_1(x))))$, while the second network takes h as an input, produces an output $p = F_p(h) = f_N(\ldots(f_{k+1}(h)))$ and incurs a loss $L = L(p, y)$ based on a label y.

With regular backpropagation, the learning signal for the first network F_h is $\partial L/\partial h$, which is a signal that specifies how the input to F_p should be changed in order to reduce the loss. When we attach a linear SG between these two networks, the first sub-network F_h no longer receives the exact learning signal from F_p, but an approximation $\mathrm{SG}(h, y)$, which implies that F_h will be minimising an approximation of the loss, because it is using approximate error gradients. Since the SG module is a linear model of $\partial L/\partial h$, the approximation of the true loss that F_h is being optimised for will be a quadratic function of h and y. Note that this is *not* what a second order method does when a function is locally approximated with a quadratic and used for optimisation – here we are approximating the current loss, which is a function of parameters θ with a quadratic which is a function of h. Three appealing properties of an approximation based on h is that h already encapsulates a lot of non-linearities due to the processing of F_h, h is usually vastly lower dimensional than $\theta_{<h}$ which makes learning more tractable, and the error only depends on quantities (h) which are local to this part of the network rather than θ which requires knowledge of the entire network.

With the SG module in place, the learning system decomposes into two tasks: the second sub-network F_p tasked with minimising L given inputs h, while the first sub-network F_h is tasked with pre-processing x in such a way that the best fitted quadratic approximator of L (wrt. h) is minimised. In addition, the SG module is tasked with best approximating L.

The approximations and changing of learning objectives (described above) that are imposed by using synthetic gradients may appear to be extremely limiting. However, in both the theoretical and empirical sections of this paper we show that SG models can, and do, learn solutions to highly non-linear problems (such as memorising noise).

The crucial mechanism that allows such rich behaviour is to remember that the implicit quadratic approximation to the loss implied by the SG module is local (per data point) and non-stationary – it is continually trained itself. It is not a single quadratic fit to the true loss over the entire optimisation landscape, but a local quadratic approximation specific to each instantaneous moment in optimisation. In addition, because the quadratic approximation is a function only of h and not θ, the loss approximation is still highly non-linear w.r.t. θ.

If, instead of a linear SG module, one uses a more complex function approximator of gradients such as an MLP, the loss is effectively approximated by the integral of the MLP. More formally, the loss implied by the SG module in hypotheses space $\mathcal{H}$ is of class $\{l : \exists g \in \mathcal{H} : \partial l/\partial h = g\}$[1]. In particular, this shows an attractive mathematical benefit over predicting loss directly: by modelling gradients rather than losses, we get to implicitly model higher order loss functions.

3. Critical points

We now consider the effect SG has on critical points of the optimisation problem. Concretely, it seems natural to ask whether a model augmented with SG is capable of learning the same functions as the original model. We ask this question under the assumption of a locally converging training method, such that we always end up in a critical point. In the case of a SG-based model this implies a set of parameters θ such that $\partial L/\partial\theta_{>h} = 0, \mathrm{SG}(h, y)\partial h/\partial\theta_{<h} = 0$ and $\partial L_{\mathrm{SG}}/\partial\theta_{\mathrm{SG}} = 0$. In other words we are trying to establish whether SG introduces regularisation to the model class, which changes the critical points, or whether it merely introduces a modification to learning dynamics, but retains the same set of critical points.

In general, the answer is positive: SG does induce a regularisation effect. However, in the presence of additional assumptions, we can show families of models and losses for which the original critical points are not affected.

Proposition 1. *Every critical point of the original optimisation problem where SG can produce $\partial L/\partial h_i$ has a corresponding critical point of the SG-based model.*

Proof. Directly from the assumption we have that there exists a set of SG parameters such that the loss is minimal, thus $\partial L_{SG}/\partial\theta_{\mathrm{SG}} = 0$ and also $\mathrm{SG}(h, y) = \partial L/\partial h$ and

[1] We mean equality for all points where $\partial l/\partial h$ is defined.

$\mathrm{SG}(h, y)\partial h/\partial\theta_{<h} = 0$. □

The assumptions of this proposition are true for example when $L = 0$ (one attains global minimum), when $\partial L/\partial h_i = 0$ or a network is a deep linear model trained with MSE and SG is linear.

In particular, this shows that for a large enough SG module all the critical points of the original problem have a corresponding critical point in the SG-based model. Limiting the space of SG hypotheses leads to inevitable reduction of number of original critical points, thus acting as a regulariser. At first this might look like a somewhat negative result, since in practice we rarely use a SG module capable of exactly producing true gradients. However, there are three important observations to make: (1) Our previous observation reflects having an exact representation of the gradient at the critical point, not in the whole parameter space. (2) One does preserve all the critical points where the loss is zero, and given current neural network training paradigms these critical points are important. For such cases even if SG is linear the critical points are preserved. (3) In practice one rarely optimises to absolute convergence regardless of the approach taken; rather we obtain numerical convergence meaning that $\|\partial L/\partial\theta\|$ is small enough. Thus, all one needs from SG-based model is to have small enough $\|(\partial L/\partial h + e)\partial h/\partial\theta_{<h}\| \leq \|\partial L/\partial\theta_{<h}\| + \|e\|\|\partial h/\partial\theta_{<h}\|$, implying that the approximation error at a critical point just has to be small wrt to $\|\partial h/\partial\theta_{<h}\|$ and need not be 0.

To recap: so far we have shown that SG can preserve critical points of the optimisation problem. However, SG can also introduce *new* critical points, leading to premature convergence and spurious additional solutions. As with our previous observation, this does not effect SG modules which are able to represent gradients exactly. But if the SG hypothesis space does not include a good approximator[2] of the true gradient, then we can get new critical points which end up being an equilibrium state between SG modules and the original network. We provide an example of such an equilibrium in the Supplementary Materials Section B.

4. Learning dynamics

Having demonstrated that important critical points are preserved and also that new ones might get created, we need a better characterisation of the basins of attraction, and to understand when, in both theory and practice, one can expect convergence to a good solution.

Artificial Data

We conduct an empirical analysis of the learning dynamics on easily analysable artificial data. We create 2 and 100 dimensional versions of four basic datasets (details in the Supplementary Materials Section D) and train four simple models (a linear model and a deep linear one with 10 hidden layers, trained to minimise MSE and log loss) with regular backprop and with a SG-based alternative to see whether it (numerically) converges to the same solution.

For MSE and both shallow and deep linear architectures the SG-based model converges to the global optimum (exact numerical results provided in Supplementary Material Table 2). However, this is not the case for logistic regression. This effect is a direct consequence of a linear SG module being unable to model $\partial L/\partial p$[3] (where $p = xW + b$ is the output of logistic regression), which often approaches the step function (when data is linearly separable), and cannot be well approximated with a linear function $\mathrm{SG}(h, y) = hA + yB + C$. Once one moves towards problems without this characteristic (*e.g.* random labeling) the problem vanishes, since now $\partial L/\partial p$ can be approximated much better. While this may not seem particularly significant, it illustrates an important characteristic of SG in the context of the log loss – it will struggle to overfit to training data, since it requires modeling step function type shapes, which is not possible with a linear model. In particular this means that for best performance one should adapt the SG module architecture to the loss function used —for MSE linear SG is a reasonable choice, however for log loss one should use architectures including a sigmoid σ applied pointwise to a linear SG, such as $\mathrm{SG}(h, y) = d\sigma(hA) + yB + C$.

As described in Section 2, using a linear SG module makes the implicit assumption that loss is a quadratic function of the activations. Furthermore, in such setting we can actually reconstruct the loss being used up to some additive constant since $\partial L/\partial h = hA + yB + C$ implies that $L(h) = \frac{1}{2}hAh^T + (yB + C)h^T + const$. If we now construct a 2-dimensional dataset, where data points are arranged in a 2D grid, we can visualise the loss implicitly predicted by the SG module and compare it with the true loss for each point.

Figure 2 shows the results of such an experiment when learning a highly non-linear model (5-hidden layer relu network). As one can see, the quality of the loss approximation has two main components to its dynamics. First, it is better in layers closer to the true loss (*i.e.* the topmost layers), which matches observations from Jaderberg et al. (2016) and the intuition that the lower layers solve a more complex problem (since they bootstrap their targets). Second, the loss is better approximated at the very beginning of the training and the quality of the approximation degrades slowly towards the end. This is a consequence of the fact that close to the end of training, the highly non-linear model has quite complex derivatives which cannot be well represented in a space of linear functions. It is worth

[2] In this case, our gradient approximation needs to be reasonable at every point through optimisation, not just the critical ones.

[3] $\partial L/\partial p = \exp(xW + b)/(1 + \exp(xW + b)) - y$

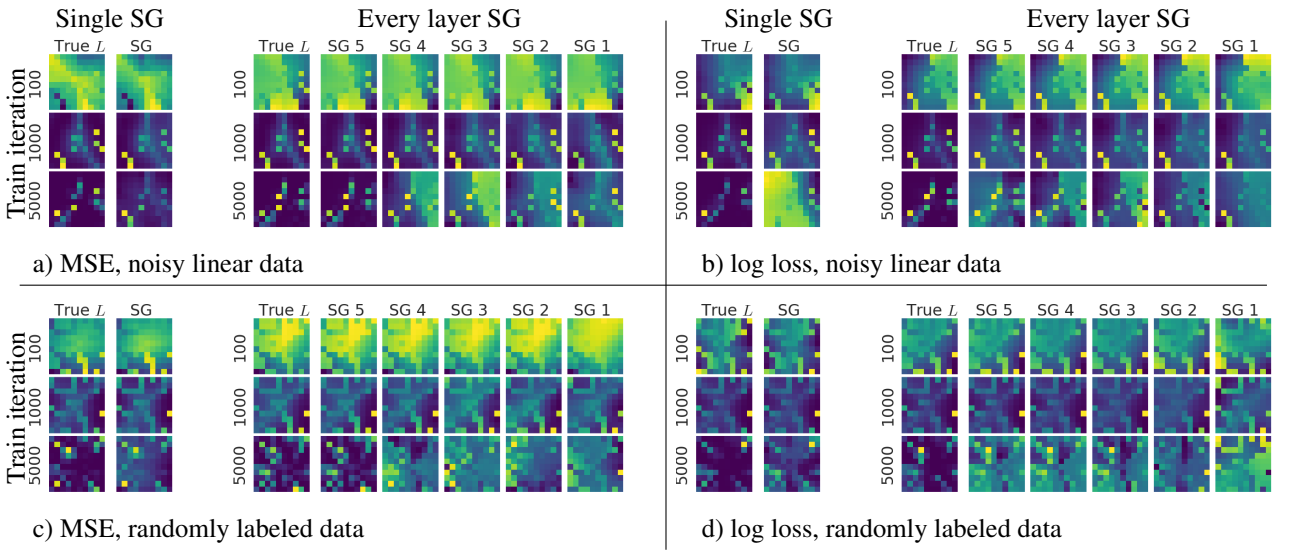


Figure 2. Visualisation of the true MSE loss and the loss approximation reconstructed from SG modules, when learning points are arranged in a 2D grid, with linearly separable 90% of points and 10% with randomly assigned labels (top row) and with completely random labels (bottom row). The model is a 6 layers deep relu network. Each image consists of visualisations for a model with a single SG (left part) and with SG between every two layers (on the right). Note, that each image has an independently scaled color range, since we are only interested in the shape of the surface, not particular values (which cannot be reconstructed from the SG). Linear SG tracks the loss well for MSE loss, while it struggles to fit to log loss towards the end of the training of nearly separable data. Furthermore, the quality of loss estimation degrades towards the bottom of the network when multiple SGs bootstrap from each other.

noting, that in these experiments, the quality of the loss approximation dropped significantly when the true loss was around 0.001, thus it created good approximations for the majority of the learning process. There is also an empirical confirmation of the previous claim, that with log loss and data that can be separated, linear SGs will have problems modeling this relation close to the end of training (Figure 2 (b) left), while there is no such problem for MSE loss (Figure 2 (a) left).

Convergence

It is trivial to note that if a SG module used is globally convergent to the true gradient, and we only use its output after it converges, then the whole model behaves like the one trained with regular backprop. However, in practice we never do this, and instead train the two models in parallel without waiting for convergence of the SG module. We now discuss some of the consequences of this, and begin by showing that as long as a synthetic gradient produced is close enough to the true one we still get convergence to the true critical points. Namely, only if the error introduced by SG, backpropagated to all the parameters, is consistently smaller than the norm of true gradient multiplied by some positive constant smaller than one, the whole system converges. Thus, we essentially need the SG error to vanish around critical points.

Proposition 2. *Let us assume that a SG module is trained in each iteration in such a way that it ϵ-tracks true gradient, i.e. that $\|\mathrm{SG}(h, y) - \partial L/\partial h\| \leq \epsilon$. If $\|\partial h/\partial \theta_{<h}\|$ is upper bounded by some K and there exists a constant $\delta \in (0, 1)$ such that in every iteration $\epsilon K \leq \|\partial L/\partial \theta_{<h}\| \frac{1-\delta}{1+\delta}$, then the whole training process converges to the solution of the original problem.*

Proof. Proof follows from showing that, under the assumptions, effectively we are training with noisy gradients, where the noise is small enough for convergence guarantees given by Zoutendijk (1970); Gratton et al. (2011) to apply. Details are provided in the Supplementary Materials Section C. □

As a consequence of Proposition 2 we can show that with specifically chosen learning rates (not merely ones that are small enough) we obtain convergence for deep linear models.

Corollary 1. *For a deep linear model minimising MSE, trained with a linear SG module attached between two of its hidden layers, there exist learning rates in each iteration such that it converges to the critical point of the original problem.*

Proof. Proof follows directly from Propositions 1 and 2. Full proof is given in Supplementary Materials Section C. □

For a shallow model we can guarantee convergence to the global solution provided we have a small enough learning rate, which is the main theoretical result of this paper.

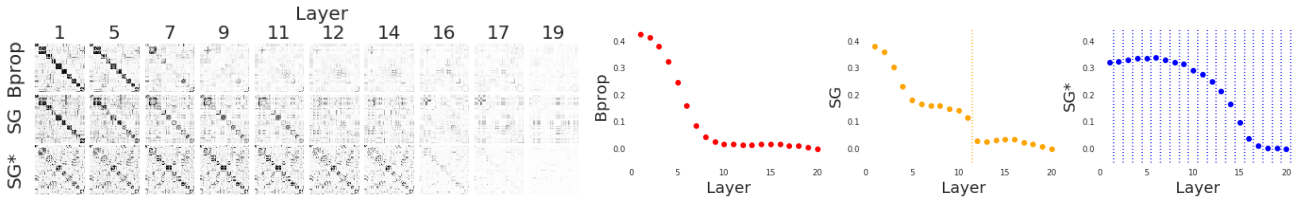


Figure 3. (left) Representation Dissimilarity Matrices for a label ordered sample from MNIST dataset pushed through 20-hidden layers deep relu networks trained with backpropagation (top row), a single SG attached between layers 11 and 12 (middle row) and SG between every pair of layers (bottom row). Notice the disappearance of dark squares on a diagonal in each learning method, which shows when a clear inner-class representation has been learned. For visual confidence off block diagonal elements are semi transparent. (right) L_2 distance between diagonal elements at a given layer and the same elements at layer 20. Dotted lines show where SGs are inserted.

Theorem 1. *Let us consider linear regression trained with a linear SG module attached between its output and the loss. If one chooses the learning rate of the SG module using line search, then in every iteration there exists small enough, positive learning rate of the main network such that it converges to the global solution.*

Proof. The general idea (full proof in the Supplementary Materials Section C) is to show that with assumed learning rates the sum of norms of network error and SG error decreases in every iteration. □

Despite covering a quite limited class of models, these are the very first convergence results for SG-based learning. Unfortunately, they do not seem to easily generalise to the non-linear cases, which we leave for future research.

5. Trained models

We now shift our attention to more realistic data. We train deep relu networks of varied depth (up to 50 hidden layers) with batch-normalisation and with two different activation functions on MNIST and compare models trained with full backpropagation to variants that employ a SG module in the middle of the hidden stack.

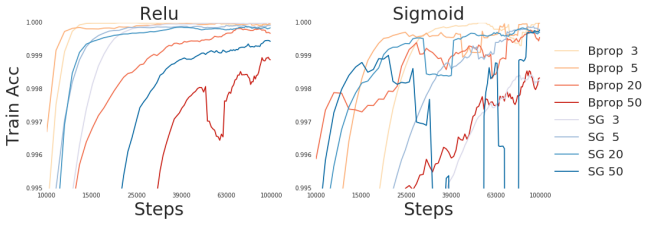


Figure 4. Learning curves for MNIST experiments with backpropagation and with a single SG in a stack of from 3 to 50 hidden layers using one of two activation functions: relu and sigmoid.

Figure 4 shows, that SG-based architectures converge well even if there are many hidden layers both below and above the module. Interestingly, SG-based models actually seem to converge faster (compare for example 20- or 50 layer deep relu network). We believe this may be due to some amount of loss function smoothing since, as described in Section 2, a linear SG module effectively models the loss function to be quadratic – thus the lower network has a simpler optimisation task and makes faster learning progress.

Obtaining similar errors on MNIST does not necessarily mean that trained models are the same or even similar. Since the use of synthetic gradients can alter learning dynamics and introduce new critical points, they might converge to different types of models. Assessing the representational similarity between different models is difficult, however. One approach is to compute and visualise Representational Dissimilarity Matrices (Kriegeskorte et al., 2008) for our data. We sample a subset of 400 points x_i from MNIST, order them by label, and then record activations on each hidden layer h when the network is presented with these points. We plot the matrix RDM for each layer, where $\text{RDM}_{ij} = 1 - \text{corr}(h(x_i), h(x_j))$, as shown in Figure 3. This representation is permutation invariant, and thus the emergence of a block-diagonal correlation matrix means that at a given layer, points from the same class already have very correlated representations.

Under such visualisations one can notice qualitative differences between the representations developed under standard backpropagation training versus those delivered by a SG-based model. In particular, in the MNIST model with 20 hidden layers trained with standard backpropagation we see that the representation covariance after 9 layers is nearly the same as the final layer's representation. However, by contrast, if we consider the same architecture but with a SG module in the middle we see that the layers before the SG module develop a qualitatively different style of representation. Note: this does *not* mean that layers before SG do not learn anything useful. To confirm this, we also introduced linear classifier probes (Alain & Bengio, 2016) and observed that, as with the pure backpropagation trained model, such probes can achieve 100% training accuracy after the first two hidden-layers of the SG-based model, as shown in Supplementary Material's Figure 8. With 20 SG modules (one between every pair of

layers), the representation is scattered even more broadly: we see rather different learning dynamics, with each layer contributing a small amount to the final solution, and there is no longer a point in the progression of layers where the representation is more or less static in terms of correlation structure (see Figure 3).

Another way to investigate whether the trained models are qualitatively similar is to examine the norms of the weight matrices connecting consecutive hidden layers, and to assess whether the general shape of such norms are similar. While this does not definitively say anything about how much of the original classification is being solved in each hidden layer, it is a reasonable surrogate for how much computation is being performed in each layer[4]. According

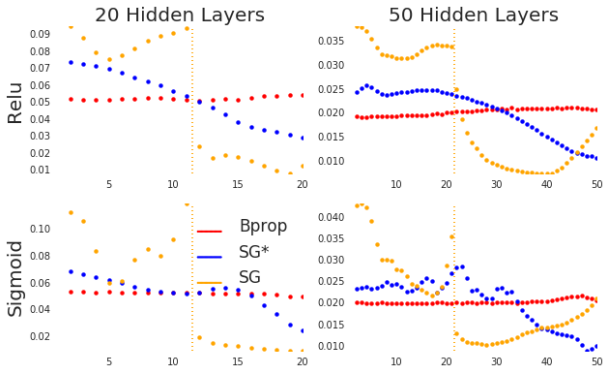


Figure 5. Visualisation of normalised squared norms of linear transformations in each hidden layer of every model considered. Dotted orange line denotes level at which a single SG is attached. SG* has a SG at every layer.

to our experiments (see Figure 5 for visualisation of one of the runs), models trained with backpropagation on MNIST tend to have norms slowly increasing towards the output of the network (with some fluctuations and differences coming from activation functions, random initialisations, etc.). If we now put a SG in between every two hidden layers, we get norms that start high, and then decrease towards the output of the network (with much more variance now). Finally, if we have a single SG module we can observe that the behaviour after the SG module resembles, at least to some degree, the distributions of norms obtained with backpropagation, while before the SG it is more chaotic, with some similarities to the distribution of weights with SGs in-between every two layers.

These observations match the results of the previous experiment and the qualitative differences observed. When synthetic gradients are used to deliver full unlocking we obtain a very basic model at the lowest layers and then see iterative corrections in deeper layers. For a one-point unlocked model with a single SG module, we have two slightly separated models where one behaves similarly to backprop, and the other supports it. Finally, a fully locked model (*i.e.* traditional backprop) solves the task relatively early on, and later just increases its confidence.

We note that the results of this section support our previous notion that we are effectively dealing with a multi-agent system, which looks for coordination/equilibrium between components, rather than a single model which simply has some small noise injected into the gradients (and this is especially true for more complex models).

6. SG and conspiring networks

We now shift our attention and consider a unified view of several different learning principles that work by replacing true gradients with surrogates. We focus on three such approaches: Feedback Alignment (FA) (Lillicrap et al., 2016), Direct Feedback Alignment (DFA) (Nøkland, 2016), and Kickback (KB) (Balduzzi et al., 2014). FA effectively uses a fixed random matrix during backpropagation, rather than the transpose of the weight matrix used in the forward pass. DFA does the same, except each layer directly uses the learning signal from the output layer rather than the subsequent local one. KB also pushes the output learning signal directly but through a predefined matrix instead of a random one. By making appropriate choices for targets, losses, and model structure we can cast all of these methods in the SG framework, and view them as comprising two networks with a SG module in between them, wherein the first module builds a representation which makes the task of the SG predictions easier.

We begin by noting that in the SG models described thus far we do not backpropagate the SG error back into the part of the main network preceding the SG module (*i.e.* we assume $\partial L_{\text{SG}}/\partial h = 0$). However, if we relax this restriction, we can use this signal (perhaps with some scaling factor α) and obtain what we will refer to as a $SG + \text{prop}$ model. Intuitively, this additional learning signal adds capacity to our SG model and forces both the main network and the SG module to "conspire" towards a common goal of making better gradient predictions. From a practical perspective, according to our experiments, this additional signal heavily stabilises learning system[5]. However, this comes at the cost of no longer being unlocked.

[4] We train with a small L_2 penalty added to weights to make norm correspond roughly to amount of computation.

[5] In fact, ignoring the gradients *predicted* by SG and only using the derivative of the SG loss, i.e. $\partial L_{\text{SG}}/\partial h$, still provides enough learning signal to converge to a solution for the original task in the simple classification problems we considered. We posit a simple rationale for this: if one can predict gradients well using a simple transformation of network activations (*e.g.* a linear mapping), this suggests that the loss itself can be predicted well too, and thus (implicitly) so can the correct outputs.

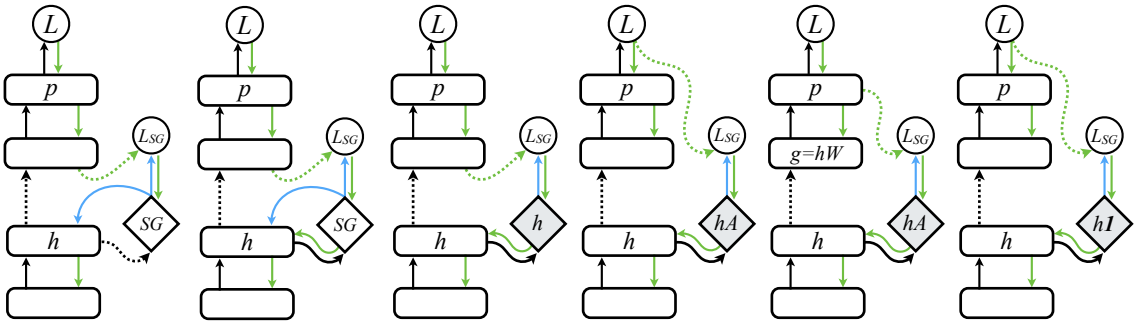
Network						
Method	SG	SG + prop	Bprop	DFA	FA	Kickback
$\widehat{\partial L/\partial h}$	$\mathrm{SG}(h, y)$	$\mathrm{SG}(h, y) + \alpha \frac{\partial L_{\mathrm{SG}}}{\partial h}$	$\partial L/\partial h$	$(\partial L/\partial p)A^T$	$(\partial L/\partial g)A^T$	$(\partial L/\partial p)\mathbf{1}^T$
$\mathrm{SG}(h, y)$	$\mathrm{SG}(h, y)$	$\mathrm{SG}(h, y)$	h	hA	hA	$h\mathbf{1}$
SG trains	yes	yes	no	no	no	no
SG target	$\partial L/\partial h$	$\partial L/\partial h$	$-\partial L/\partial h$	$-\partial L/\partial p$	$-\partial L/\partial g$	$-\partial L/\partial p$
$L_{\mathrm{SG}}(t, s)$	$\|t - s\|^2$	$\|t - s\|^2$	$-\langle t, s\rangle$	$-\langle t, s\rangle$	$-\langle t, s\rangle$	$-\langle t, s\rangle$
Update locked	no	yes*	yes	yes	yes	yes
Backw. locked	no	yes*	yes	no	yes	no
Direct error	no	no	no	yes	no	yes

Table 1. Unified view of "conspiring" gradients methods, including backpropagation, synthetic gradients are other error propagating methods. For each of them, one still trains with regular backpropagation (chain rule) however $\partial L/\partial h$ is substituted with a particular $\widehat{\partial L/\partial h}$. Black lines are forward signals, blue ones are synthetic gradients, and green ones are true gradients. Dotted lines represent non-differentiable operations. The grey modules are not trainable. A is a fixed, random matrix and $\mathbf{1}$ is a matrix of ones of an appropriate dimension. * In SG+Prop the network is locked if there is a single SG module, however if we have multiple ones, then propagating error signal only locks a module with the next one, not with the entire network. Direct error means that a model tries to solve classification problem directly at layer h.

Our main observation in this section is that FA, DFA, and KB can be expressed in the language of "conspiring" networks (see Table 1), of two-network systems that use a SG module. The only difference between these approaches is how one parametrises SG and what target we attempt to fit it to. This comes directly from the construction of these systems, and the fact that if we treat our targets as constants (as we do in SG methods), then the backpropagated error from each SG module ($\partial L_{\mathrm{SG}}/\partial h$) matches the prescribed update rule of each of these methods ($\widehat{\partial L/\partial h}$). One direct result from this perspective is the fact that Kickback is essentially DFA with $A = \mathbf{1}$. For completeness, we note that regular backpropagation can also be expressed in this unified view – to do so, we construct a SG module such that the gradients it produces attempt to align the layer activations with the negation of the true learning signal ($-\partial L/\partial h$). In addition to unifying several different approaches, our mapping also illustrates the potential utility and diversity in the generic idea of predicting gradients.

7. Conclusions

This paper has presented new theory and analysis for the behaviour of synthetic gradients in feed forward models. Firstly, we showed that introducing SG does not necessarily change the critical points of the original problem, however at the same time it can introduce new critical points into the learning process. This is an important result showing that SG does not act like a typical regulariser despite simplifying the error signals. Secondly, we showed that (despite modifying learning dynamics) SG-based models converge to analogous solutions to the true model under some additional assumptions. We proved exact convergence for a simple class of models, and for more complex situations we demonstrated that the implicit loss model captures the characteristics of the true loss surface. It remains an open question how to characterise the learning dynamics in more general cases. Thirdly, we showed that despite these convergence properties the trained networks can be qualitatively different from the ones trained with backpropagation. While not necessarily a drawback, this is an important consequence one should be aware of when using synthetic gradients in practice. Finally, we provided a unified framework that can be used to describe alternative learning methods such as Synthetic Gradients, FA, DFA, and Kickback, as well as standard Backprop. The approach taken shows that the language of *predicting gradients* is suprisingly universal and provides additional intuitions and insights into the models.

References

Alain, Guillaume and Bengio, Yoshua. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

Baldi, Pierre, Sadowski, Peter, and Lu, Zhiqin. Learning in the machine: Random backpropagation and the learning channel. *arXiv preprint arXiv:1612.02734*, 2016.

Balduzzi, David, Vanchinathan, Hastagiri, and Buhmann, Joachim. Kickback cuts backprop's red-tape: Biologically plausible credit assignment in neural networks. *arXiv preprint arXiv:1411.6191*, 2014.

Bengio, Yoshua, Lee, Dong-Hyun, Bornschein, Jorg, Mesnard, Thomas, and Lin, Zhouhan. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*, 2015.

Gratton, Serge, Toint, Philippe L, and Tröltzsch, Anke. How much gradient noise does a gradient-based linesearch method tolerate. Technical report, Citeseer, 2011.

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

Kriegeskorte, Nikolaus, Mur, Marieke, and Bandettini, Peter A. Representational similarity analysis-connecting the branches of systems neuroscience. *Frontiers in systems neuroscience*, 2:4, 2008.

Lillicrap, Timothy P, Cownden, Daniel, Tweed, Douglas B, and Akerman, Colin J. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7, 2016.

Marblestone, Adam H, Wayne, Greg, and Kording, Konrad P. Toward an integration of deep learning and neuroscience. *Frontiers in Computational Neuroscience*, 10, 2016.

Nøkland, Arild. Direct feedback alignment provides learning in deep neural networks. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 1037–1045. Curran Associates, Inc., 2016.

Williams, DRGHR and Hinton, GE. Learning representations by back-propagating errors. *Nature*, 323(6088): 533–538, 1986.

Zoutendijk, G. Nonlinear programming, computational methods. *Integer and nonlinear programming*, 143(1): 37–86, 1970.