On the benefits of non-linear weight updates

Paul Norridge

July 27, 2022

Abstract

Recent work has suggested that the generalisation performance of a DNN is related to the extent to which the Signal-to-Noise ratio is optimised at each of the nodes. In contrast, Gradient Descent methods do not always lead to SNR-optimal weight configurations. One way to improve SNR performance is to suppress large weight updates and amplify small weight updates. Such balancing is already implicit in some common optimizers, but we propose an approach that makes this explicit. The method applies a non-linear function to gradients prior to making DNN parameter updates. We investigate the performance with such non-linear approaches. The result is an adaptation to existing optimizers that improves performance for many problem types.

1 Introduction

Deep neural networks are being used ever more widely for machine learning applications and continue to demonstrate impressive results. Critical to this success has been the widespread use of back-propagation [1] via Gradient Descent (and, more usually, Stochastic Gradient Descent, SGD). Despite the great success of deep learning, the precise reasons why some networks generalise better than others is not always clear [2, 3].

Recently it has been proposed that a contributor to the generalisation performance of DNNs is the ‘SNR optimality’ of the weight set employed, see [4]. That is, for a given data set, do the weights act to optimise the Signal-to-Noise ratio (SNR) at the output of the nodes in the network? In particular, when a node has correlated inputs, do the weights maximise the use of all these inputs to combat noise? Empirical evidence suggests that this is indeed a contributor to performance.

This result leads to the question of application and whether it is possible to find training methods that enhance the SNR optimality of a network. In this paper, we investigate one such SNR-inspired enhancement to training. The purpose of this is twofold: First, to extend the study in [4], by developing an optimization approach inspired by the SNR optimality perspective and investigating its effectiveness; Second, to propose adaptations to standard optimizers that can be used to improve generalisation.

We emphasise that we do not intend to formally prove the validity of the optimizer adaptations proposed. The approach is to consider the weaknesses of optimisers such as SGD and Momentum from an SNR optimality point-of-view and identify ways that these might be mitigated. The benefit of the approach is shown by applying the adapted optimizers to a set of test benches.

The paper is structured as follows: We begin by giving some details on the SNR motivation for the approach and outlining the geometric picture that this viewpoint suggests. We then describe the key features of the proposed non-linear adaptation and the specific implementations considered here. These implementations are then tested against a set of benchmarks. The appendices provide some additional ‘toy model’ motivation and plots of the training evolution.

2 Related Work

The work presented here overlaps with three areas of Machine Learning study.

The main focus is the development and adaptation of optimizers for Gradient Descent training of DNNs. A significant number of optimizers have been proposed in an effort to ensure that DNN training leads to networks that perform well, while minimising training time. It is not possible to list all of the options here, but we use a number as reference points; basic Stochastic Gradient Descent [5], Momentum [6], Nesterov Accelerated Gradient (NAG) [7], Adam [8], AMSGrad [9], NADAM [10], Lookahead Momentum [11], AMS Bound [12] and RMSProp [13]. In addition, [14] is an interest comparison, since it uses non-linear functions of the gradient, but only to alter the effective learning rate. Our benchmark is Deepobs [15] and in particular the survey provided in [16]. We also note the useful overviews provided in [17] and [18].
Close in spirit to the work here is gradient clipping [19, 20]. Although this is commonly implemented as a distinct step in training calculations, it can be thought of as an adaptation applied to optimizers. (In fact, this is how it is treated in Keras [21].)

To a lesser extent, we will also touch on the question of the relationship between flat minima and generalisation performance. This relationship has been an area of investigation since the original observations in [22]. Again, it is not possible to list the full list of papers discussing this area, but we note the recent discussions in [23], [24], [25], [26] and [27]. Of particular interest here is [28], which shows that minima may only have a subset of directions that are flat. This is consistent with the expectations motivating this study. Recently, there has been interest in optimization approaches that favour flat minima, see, for example, [29], [30] and [31]. The investigation here is primarily concerned with where in a minima the model is left after optimization rather than explicitly looking for such regions.

In addition, the work here is motivated by noise robustness. Although we build primarily on the investigation in [4], we note that noise robustness has also been used in generalisation predictions, as described in [32], [33] and [34].

Tying the last two points together, in some sense, is ref [35], which demonstrates that flat minima have paths through the network that are dedicated to signals with the highest gain.

### 3 Motivation

#### 3.1 SNR optimality

The motivation for the approach we will propose is to select weight sets that minimise the impact of noise on the network performance, or conversely, optimise the preservation of useful information. We can summarise the reasoning as follows:

1. A DNN will perform best when the available (useful) information is exploited to maximum extent.

2. Optimal use of information by the entire network depends on maximising the information preservation of individual nodes.

3. The information preservation from input to output of a node can be characterised via the Signal-to-Noise Ratio (SNR)

4. It is possible to identify an optimal set of weights to maximise SNR, which motivates the way we train the network.

Here we briefly recap the results from [4].

Our building block is the usual node definition

\[ y = \sum_j w_{ij} x_j + b_i \]  

with inputs \(x_j\), weights \(w_{ij}\), bias \(b_i\) and activation function \(g()\).

We consider that every input has a contribution that is useful information – signal – together with noise. Partitioning the inputs into signal and noise components, the weighted sum can be expressed as:

\[ \sum_j w_{ij} x_j = \sum_j w_{ij} (s_j + n_j) \]  

Initially, we leave open the question of how to distinguish ‘signal’ and ‘noise’. But, critically, we assume that the signal is a component that is shared by some subset of the inputs; that is, inputs have correlated components.

The SNR for the node can be expressed as [36]

\[ SNR^{(m)}_i = \frac{\text{var} \left( \sum_j w_{ij} s_j \right)}{\sum_j w_{ij}^2 \text{var} \left( n_j \right)} \]  

where we have assumed that the noise components from different inputs are statistically independent.

Maximising this expression, we obtain a term for the optimal weight set.

\[ w_{ij} = k_i \frac{\text{cov} \left( s_j, \sum_k w_{jk} s_k \right)}{\text{var} \left( n_j \right)} \]

where \(k_i\) are contributions common to all inputs to the node.

We note that we are essentially requiring that the weights form a matched filter (see also [37] for a matched filter approach to DNNs).

In [4], this was taken further by assuming that the signal can be defined by the output of the node. That is, that the training process causes the output to be dominated by useful information. This led to an expression for the optimal weight set that was shown to be correlated to the generalisation performance of a number of examples. Based on the results presented, there is evidence that a DNN will generalise well when it has good performance on the training dataset and the weights are close to this optimum.

---

1Not discussed in [4] or [37] is that the activation function can be characterised as a matched filter detection, with the node bias being the detection threshold.
With this in mind, a more intuitive way to interpret equation 4 is that, after convergence, for each node, we want any inputs that have good correlation with the output to have corresponding high weighting.

However, we do not need to make assumptions about the content of the signal for the current discussion; if we accept that SNR maximisation is beneficial, we can use this matched filtering expression (equation 4), to identify possible weaknesses of SGD and other optimizers.

First, observe that even if two inputs are correlated, they will typically be treated differently to each other by the training process. There is an aspect of success-breeds-success in the SGD process. That is, if there are two inputs to a node with the same SNR but different signal levels, the higher magnitude input may be enhanced the most, even though noise sensitivity would be improved if the two had equal emphasis. In a DNN, this can lead to feedback where the higher magnitude inputs get increasingly amplified throughout the net-work at the cost of performance; a ‘toy model’ demonstrating this is given in appendix A.

Second, if there are different activation rates on inputs to a node, this can cause the weighting to be applied sub-optimally. Consider the following: If a ReLU activation has been applied to the inputs, \( x_i \leftarrow \max(0, x_i) \), noise will only contribute when the \( x_i > 0 \). That is, the noise will have zero variance whenever \( x_i < 0 \). To make this explicit, we re-express the noise variance as

\[
\text{var}(n_j) = a_j \text{var}(n_j')
\]

(5)

where \( a_j \) is the rate of activations of node \( j \) of preceding layer, calculated as \( a_i = p(x_i > 0) \), and \( \text{var}(n_j') \) is the ‘raw’ noise variance of samples prior to application of the activation function.

\[
w_{ij} = k_i \frac{\text{cov}(s_j, \sum k \sum k s_k)}{a_j \text{var}(n_j')}
\]

(6)

So, the optimal SNR configuration depends on the activations of the connected input nodes.

Hence, SNR optimality requires that we compensate for the activation rate of an input when selecting optimal weights. We should select weights based on the contribution of the inputs when non-zero rather than their contribution in all cases. Gradient Descent typically does not take this into account – a node with higher activation rate will normally have larger gradients and be emphasised more. As a consequence, inputs with relatively low activation rates will typically have lower weighting than would be optimal from an SNR optimality point-of-view.

Of course, the reduction of the difference between large and small weight updates is an implicit feature of many optimizers, but typically this is a by-product of a design which is aimed at other considerations. For example, a valuable characteristic attributed to Momentum is that it allows better balance between directions with large and small gradients (see, for example, [38, 39]). Optimizers with normalisation, such as Adam, have similar benefits. Ref. [40] notes that the Adam weight updates are scale invariant (so, will not depend on the weighting of the input or the activation rate) and the magnitude of the updates are related to some extent to the noise on the inputs.

Our interest here is whether we can find an effective optimizer that takes inspiration directly from equation (4) rather than improving SNR optimality as a by-product.

3.2 Geometric intuition

It is beneficial to have a picture of the error surface that the SNR optimality perspective implies. Intuitively, if a node has correlated inputs then we expect to find it associated with a minimum that is almost flat, in the sense that some directions will have small gradients. In the ideal case, where a subset of inputs are perfectly correlated and the same magnitude, these inputs will be interchangeable. Consequently, changing the weighting over the subset will have no effect so long as the sum of the weights remains the same.

That is, if \( C \) is the set such that if \( s_j = s \forall j \in C \)

\[
\sum_j w_{ij} x_j = \sum_{j \in C} w_{ij} s_j + \sum_{j \notin C} w_{ij} x_j
\]

\[
= s \sum_{j \in C} w_{ij} + \sum_{j \notin C} w_{ij} x_j
\]

In the case where the inputs also have noise components, the situation is similar, but some points on flat region will have more noise resilience than others – to make it explicit for this example, if all inputs have IID Gaussian noise, the total noise goes like \( \sqrt{\sum_{j \in C} w_{ij}^2} \) whereas signal depends on \( \sum_{j \in C} w_{ij} \). Consequently, equal weights would be more noise resilient than other combinations with the same signal output. If the correlation is only approximate the situation is less straightforward but the principle holds.

On the other hand, we note that if correlated inputs are distributed across multiple nodes, then we may not find such a simple noise-resilient structure, since the signals
can become non-trivially linked to other unrelated features. The linking between unrelated features is also likely to cause sub-optimal SNR performance. This suggests a reason why flat minima are frequently related to good generalisation performance.

Of course, we find here a connection with the extensive studies on the relationship between flat minima and good generalisation. In particular, we note ref [28], which shows that flat minima with good generalisation may only be flat only in a subset of directions. In the perspective considered here, not all inputs will be correlated, in general.

If this picture is correct, then we can identify a critical problem: there is a specific region on the flat minimum that is optimal for SNR optimality; however, once we reach the minimum, it becomes harder to move towards that optimal point. Indeed, optimizers such as Adam will reduce the weight updates significantly when the flat region is reached since the second moments of the updates will be significantly larger than the corresponding first moments.

If our goal is to get close to an SNR-optimal location, then we would like the training process to be close at the point where it first reaches the flat region; this will minimise the need for subsequent movement in flat directions. Since, we know the criteria for SNR optimality weights in equation 6, we can identify potential strategies to meet this requirement. Whereas the motivation for other optimizers has frequently been concern over how training behaves on reaching minima or saddle points, we suggest that what happens on the way to a zero-curvature region is also significant.

### 4 Non-linear weight updates

Above we identified two shortcomings of SGD from the perspective of SNR optimality. During the training process, both of these shortcomings are realised by some signal-carrying inputs being under-emphasised with relative small weight updates. This suggests that the training process would be improved if we can adjust the weight updates on the ‘useful’ inputs so that smaller updates are scaled-up relative to larger ones. Consequently, we propose an adaptation of SGD that applies weight updates of the form

\[
\Delta w \sim h \left( \frac{\partial f}{\partial w} \right)
\]

where \( h() \) is some non-linear function with

1. \( 0 \leq \frac{\partial h(x+\delta)}{\partial w} \leq \frac{\partial h(x)}{\partial w} \forall x, \delta > 0 \)
2. \( \frac{\partial h(x_1+\delta_1)}{\partial w} < \frac{\partial h(x_1)}{\partial w} \)
3. \( \text{sgn}(h(x)) = \text{sgn}(x) \)
4. \( h(x) = -h(-x) \)

Property 1 ensures that the function explicitly adjusts the SGD weight updates with an attenuation that is increased as the update magnitude increases. Hence, it effectively reduces the gap between large and small updates.

Stochastic Gradient Descent (as well as other optimizer formulations) has demonstrated that we don’t need to consistently take the steepest descent step at every training step; what is important is to continually head downhill. Property 2 ensures that this is the case.

Finally, property 3 ensures that there is no bias introduced. For example, if this was not respected, then uncorrelated inputs may grow purely from the stochastic gradients coming from mini-batches.

We note that this approach is already implemented implicitly in some of the common weight update algorithms. For example, gradient clipping [19, 20] fits this definition by applying

\[
h(x) = \begin{cases} 
  x & \text{for } x < t, \\
  t & \text{for } x > t
\end{cases}
\]

Similarly, component-wise normalisation (see, for example, [41]) corresponds to an application of the function \( h(x) = \text{sgn}(x) \).

Both of these options have weaknesses. Gradient clipping only restricts updates above a certain scale, which must be selected explicitly, prior to training; this limits the ability to balance updates. On the other hand, from an SNR optimality point-of-view, \( h(x) = \text{sgn}(x) \) risks doing too much; it treats all inputs equally, no matter how small the updates; this raises the possibility of introducing new contributions to a node ‘signal’ and coupling unrelated features. We would like some middle-ground between the extremes of pure SGD and component-wise normalisation.

In this paper, we propose an alternative function choice,

\[
h_v(x) = \text{sgn}(x)|x|^\nu
\]

That is, with weight updates of the form

\[
\Delta w_{ij} = \alpha h_v \left( \frac{\partial f}{\partial w_{ij}} \right)
\]

\[
= \alpha \text{sgn} \left( \frac{\partial f}{\partial w_{ij}} \right) \left| \frac{\partial f}{\partial w_{ij}} \right|^
u
\]
Algorithm 1 NL-SGD with L2 regularization

select \( \alpha \in \mathbb{R}^r \), \( \nu \in (0, 1] \), \( \lambda \in \mathbb{R}^r \)
repeat
\[ t \leftarrow t + 1 \]
\[ g^0_t \leftarrow \frac{1}{m} \nu \partial L (f(x^0); \theta_t), y^0) \]  
kernel gradients
\[ g^b_t \leftarrow \frac{1}{n} \nu b \partial L (f(x^0), b_t), y^0) \]  
bias gradients
\[ \theta \leftarrow \theta_t - \nu g^0_t + \lambda \theta_t \]
\[ b \leftarrow b_t - \nu g^d_t + \lambda b_t \]
until stopping criterion is met

where the exponent \( \nu \in [0, 1) \) and \( \alpha \in \mathbb{R}^r \) are both hyperparameters\(^2\). In contrast to the simpler non-linear functions considered above, this has a scale-invariant effect, but does not give too much weight to very low gradients.

Clearly, at the endpoints of the interval for \( \nu \), the weight update reduces to simple Gradient Descent and component-wise normalisation of the gradient; so, in some sense, this provides a continuous set of functions between these two extremes.

For reference, examples of the functions are given in figure 1.

Figure 1: Example curves for \( h_\nu \) functions

Henceforth in this paper, we will label optimization algorithms using one of the \( h_\nu \) functions with 'NL-'. We detail some algorithm options below.

4.1 Regularisation

When applying a NL function to the weight updates, we must consider how to introduce \( L_2 \) regularisation. It is clear that including an \( L_2 \) regularisation term directly in the loss function with our proposed update function will lead to changing regularisation behaviour depending on the magnitude of the primary loss term.

To address this issue, we follow the approach given in [44]. That work focussed on the appropriate way to apply weight decay in the context of the Adam optimiser, recommending that it is applied as a separate term in the weight update – i.e. as explicit weight decay – rather than including \( L_2 = \lambda \| \textbf{w} \|^2 \) in the loss function.

With this in mind, we propose the basic NL approach in algorithm 1.

4.2 Momentum

As well as simple NL-SGD, we will also consider Momentum and Nesterov Accelerated Gradient versions. These are applied simply by feeding our new weight update into the normal algorithm definitions. Although the adaptation is straightforward, for clarity we provide the descriptions as algorithms 2 and 3.

4.3 Behaviour near a minimum

It should be clear that as we get closer to the minimum, and the gradient becomes shallower, the amplifying ef-

\(^2\)We note that a variant on the \( \nu = 1/2 \) case was used in [42] and [43] for the same kind of scaling envisaged here, albeit for scaling reinforcement learning value functions rather than gradients.
fect of the nonlinear function will increase. It is likely that applying a NL function will see noisier behaviour at convergence than, for example, SGD. This could be a weakness of the approach; however, we know that there are already other mechanisms that produce similar effects. For example, the stochastic behaviour with small batch sizes or overly large learning rates can produce overshooting when close to convergence. This is demonstrated in the benefit of decaying learning rates or increasing of batch size [45]. So, at worst, we are exacerbating the problem rather than introducing something new.

To account for this behaviour near the minima, we can take an approach similar to that in [46] and reduce the learning rate significantly, close to the end of training – frequently referred to as an ‘annihilation’ phase. Generally, we find that such a phase does improve performance with NL algorithms; in the tests below we will identify some cases where this addition is particularly notable.

5 Benchmarking Overview

Our primary tool for assessing the performance of optimizers with fractional exponents is the DeepOBS benchmark [15]. This benchmark provides infrastructure and a set of reference cases to measure optimizer performance. A subset of the reference cases were used in [16] to assess the performance of a large set of commonly used optimizers. Consequently this provides us with a clear comparison point.

The test cases used in [16] are

P1 Quadratic Deep
P2 VAE applied to MNIST
P3 Simple CNN applied to F-MNIST
P4 Simple CNN applied to CIFAR-10
P5 VAE applied to F-MNIST
P6 CNN applied to CIFAR-100
P7 Wide ResNet applied to SVHN
P8 RNN applied to War and Peace

These test cases are partitioned into ‘small’ (P1 to P4) and ‘large’ (P5 to P8). The small examples allow rapid investigations - and we will use them to identify some characteristics of the NL-SGD optimizer.

To allow exploration of the impact of different mechanisms, $L_2$ regularisation/weight decay, dropout [47] and batch normalisation [23] are applied in some cases and not in others. See text below for details in each case.

In addition, [16] aims to replicate the process of hyperparameter section with a selection of randomised search algorithms applied to all optimizers. Three algorithms with increasing search depth were considered - ‘small’, ‘medium’ and ‘large’. For the investigation here, we focus on the ‘medium’ option. In practice, we found that this gives a good indication of optimizer performance without excessive search time.

Medium randomisation takes 50 samples of the hyperparameters for each optimizer and then generates one training run for each same. A validation set is used to select the best hyperparameter set. For the purpose of evaluating the optimizer, the hyperparameter sample with the best validation accuracy or loss (depending on the test case) is selected and training runs with this setting are repeated 10 times, with different random seeds.

The distributions were as follows:

| Parameter | Tuning Distribution |
|-----------|--------------------|
| learning rate $\alpha$ | $\mathcal{L} \left( 10^{-4}, 1 \right)$ |
| momentum $\rho$ | fixed at 0.9 |
| exponent $\nu$ | selected uniformly from $\{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ |

Of course, the search could be expanded beyond this, most obviously by including additional options for the momentum parameter and a continuous range for the exponent. For this initial investigation, the decision was to avoid the search space becoming too large.

Ref [16] also considered a number of different learning rate schedules. For the purpose of this paper, we concentrate primarily on fixed learning rates. However, as already noted, where informative, we will provide results with an ‘annihilation’ phase; we implement this as a reduction in the learning rate by 10x for the last 5 epochs of training. (This is predominantly to show the benefit of counteracting the ‘noise amplification’ close to convergence rather than to imply this is of unique benefit for NL algorithms.)

For each test case, we extract five results from [16] for comparison: SGD, Momentum, NAG, Adam, plus the best performing result for the test case being considered.

For the NL-SGD results we give three variants, corresponding to the algorithms outlined above: NL-SGD, NL-Mom and NL-NAG.

Note that, for assessing performance in a consistent way with the results of [16], we want the weight decay
term in our NL training to have a comparable magnitude to the $L_2$ term used previously. For NL-SGD this is relatively straightforward, since we can simply set $\lambda = \alpha \lambda'$, where $\lambda'$ is the value reported in [16]. For NL-Mom and NL-NAG, we must recognise that the weight decay term will have an amplification due to the momentum mechanism. This complicates selecting a value for $\lambda$ for NL-Mom and NL-NAG as it is not possible to pick an exactly corresponding value. For the comparisons here, we use $\lambda = \alpha \lambda'/(1 - \rho)$. This approximates the momentum amplification as an infinite geometric series in the $\rho$ term, which would be correct if the weight magnitude did not change. Since we restrict $\rho$ to 0.9 for NL-Mom and NL-NAG in this discussion, the momentum contribution drops quickly and the comparison is considered reasonable.

We note at the outset that SNR optimality was constructed with image classification as the primary focus. These have clear features consisting of correlated inputs and will be affected by noise. Consequently, we expect the four classification test cases to be those where NL-SGD brings the most benefit. However, it is instructive to see whether there is wider applicability.

All tests were carried out using Tensorflow [48].

## 6 DeepObs – Small Test Cases

In this section, we consider the four small test problems. The ability to train these models quickly allows us to do a more thorough investigation into the relationship between performance and hyperparameter dependency.

For each test case we give a very brief description identifying key points on the construction and regularisation configuration. For further details, the reader should consult [15].

### 6.1 F-MNIST 2C2D

FMNIST 2C2D is a CNN, with 2 convolutional layers and two fully connected layers applied to the Fashion MNIST dataset [49]. No weight decay, dropout or batch normalisation is applied. The training is carried out for 100 epochs with a batch size of 128.

The results are given in table 1. The immediate observation is that each NL algorithm performs better than the standard equivalent. NL-NAG comes close to the best performing optimizer in [16].

| Optimizer   | Test Accuracy | $\nu$ | $1 - \rho$ |
|-------------|---------------|-------|------------|
| SGD         | 91.76 ± 0.19  | -     | -          |
| Momentum    | 91.91 ± 0.14  | -     | 0.01       |
| NAG         | 92.13 ± 0.14  | -     | 0.01       |
| Adam        | 91.61 ± 0.28  | -     | -          |
| Best (AMSGrad) | 92.27 ± 0.18  | -     | -          |
| NL-SGD      | 92.04 ± 0.20  | 0.5   | -          |
| NL-Mom      | 92.16 ± 0.16  | 0.7   | 0.1        |
| NL-NAG      | 92.20 ± 0.17  | 0.7   | 0.1        |

Table 1: F-MNIST 2C2D

It is instructive to plot the performance errors (validation set) for different combinations of $\nu$ and $\alpha$, see figure 2. This demonstrates that the optimal meta-parameter set is one with $\nu < 1$ and, in fact, any value for $\nu$ with $\nu \in [0.5, 1)$ will allow some improvement over conventional SGD or Momentum.

It is also notable that there is approximately a linear relationship between $\nu$ and log of the peak learning rate at that exponent.

### 6.2 CIFAR-10 3C3D

This test case applies a CNN with 3 convolution layers and 3 dense layers to the CIFAR-10 dataset. The activation functions are ReLU and the training uses $L_2$ regularisation (updated to weight decay for the NL optimizers), [50]. The training is carried out for 100 epochs with a batch size of 128.

| Optimizer   | Test Accuracy | $\nu$ | $1 - \rho$ |
|-------------|---------------|-------|------------|
| SGD         | 82.37 ± 1.16  | -     | -          |
| Momentum    | 83.00 ± 0.86  | -     | 0.01       |
| NAG         | 83.43 ± 0.55  | -     | 0.14       |
| Adam        | 82.82 ± 0.91  | -     | -          |
| Best (NAG)  | 83.43 ± 0.55  | -     | 0.14       |
| NL-SGD      | 83.24 ± 0.79  | 0.7   | -          |
| NL-Mom      | 83.83 ± 0.35  | 0.7   | 0.9        |
| NL-NAG      | 84.21 ± 0.53  | 0.7   | 0.9        |

Table 2: CIFAR-10 3C3D

The results are given in table 2. Again we observe that each NL algorithm performs better than the standard equivalent. In this case, both NL-Mom and NL-NAG perform better than all other optimizers.

We should be careful to rule out the possibility this improvement has come, not from the use of the NL func-

---

*The plot also gives confirmation that, for this simple case, with very similar performance between optimizers, the improvement in the NL versions is a real effect and not simply statistical variation.*
tion, but purely from applying weight decay to Momentum instead of $L_2$ regularisation. With this in mind, we note that even if we revert to the $L_2$ approach, the mean accuracy for NL-Mom is 83.55, which still outperforms other optimizers.

Again, plotting performance with varying $\nu$ and $\alpha$ gives comparable results to the previous test case, most markedly for NL-Mom.

If we introduce an ‘annihilation’ phase at the end of training then we see some further improvement\(^4\): NL-SGD, NL-Mom and NL-NAG achieve test accuracies of 84.21, 85.56 and 85.75.

\(^4\)Note that this was not obtained with an addition hyperparameter search, but simply took the best result from the previous search and reran the test cases with the added scheduling.

### 6.3 MNIST VAE

This test case applies a VAE to the MNIST dataset [51, 52]. The convolutional layers use a leaky ReLU activation function and dropout is applied after each convolutional layer. The training is carried out for 50 epochs with a batch size of 64.

In this case, the NL algorithms have more stable results than the non-NL variants, table 3. (SGD and Momentum all have large variance, reflecting the fact that they sometimes fail to find a good minimum with the selected hyperparameters.) However, none of the NL
variants achieve performance close to Adam.

|          | Test Loss  | $\nu$ | $1 - \rho$ |
|----------|------------|-------|------------|
| SGD      | 36.18 ± 10.96 | -     | -          |
| Momentum | 35.99 ± 11.09 | 0.64  | -          |
| NAG      | 36.23 ± 10.93 | 0.64  | -          |
| Adam     | 27.77 ± 0.07  | -     | -          |
| Best (Adam) | 27.77 ± 0.07   | -     | -          |
| NL-SGD   | 28.41 ± 0.13  | 0.6   | -          |
| NL-Mom   | 28.57 ± 0.12  | 0.5   | 0.1        |
| NL-NAG   | 28.46 ± 0.19  | 0.5   | 0.1        |

Table 3: MNIST VAE

Note: SGD, Momentum and NAG all see high losses and variance due to a mix with some runs that converge well and some that converge badly.

As with previous test cases, we can look at the performance for different $\nu - \alpha$ combinations. The results for NL-SGD and NL-Mom are shown in figure 4. This suggests that the better performance of NL algorithms compared to SGD, Momentum and NAG is not simply due to the random search finding better points, but that applying the NL function allows a more stable training region to be found.

With an ‘annihilation’ phase at the end of training NL-SGD and NL-Mom achieve test accuracies of 28.08 and 28.33.

6.4 Quadratic Deep

The quadratic deep test problem generates the test data from a vector of IDD normally distributed samples and a shallow function.

$$\mathcal{L} = \frac{1}{2}(\theta - x)^T Q(\theta - x) \quad (10)$$

with Hessian $Q$, trainable parameters $\theta$ and "data" $x$ a vector if IDD samples from a zero-mean normal.

$Q$ is constructed such that 90% of the eigenvalues of the Hessian are drawn from the interval (0.0, 1.0) and the other 10% are from (30.0, 60.0). The intention is to construct an eigenspectrum comparable with that discussed in [53].

The training is carried out for 100 ‘epochs’ with a batch size of 128.

Considering that our initial motivation came from the presence of correlated inputs into the network or into the hidden layers, we would not expect application of a NL function to improve the training for this model. This is exactly what we observe. The results in table 4 show that none of the NL variants provide an advantage. In fact, they perform marginally worse than most of the other optimizers assessed in [15].

|          | Test Loss  | $\nu$ | $1 - \rho$ |
|----------|------------|-------|------------|
| SGD      | 86.29 ± 3.44 | -     | -          |
| Momentum | 87.02 ± 0.02 | -     | 0.01       |
| NAG      | 87.08 ± 0.02 | -     | 0.01       |
| Adam     | 86.58 ± 1.95 | -     | -          |
| Best (SGD) | 86.29 ± 3.44   | -     | -          |
| NL-SGD   | 87.69 ± 0.10  | 0.8   | -          |
| NL-Mom   | 87.14 ± 0.06  | 0.9   | 0.9        |
| NL-NAG   | 87.17 ± 0.06  | 0.9   | 0.9        |

Table 4: Quadratic Deep

Figure 4: Hyperparameter dependancies for MNIST VAE

Figure 5 shows the performance as the two hyperpa-
rameters are varied. Consistent with expectations, the optimal points correspond to $\nu = 1$ – that is, basic SGD and Momentum – with performance dropping off as the exponent is reduced (or, indeed, increased). Further, the optimal region is relatively small, which suggests why the random search process was unable to find a good configuration.

6.5 Discussion

In every test case, apart from Quadratic Deep, we see that applying the NL function in conjunction with SGD, Momentum or NAG leads to better performance than the usual implementation of the optimizers. In fact, the sensitivity plots indicate that even a small non-linearity will improve performance. Obviously, this could be applied without the search over $\nu$ considered here.

The exception effectively proves the rule, since Quadratic Deep does not fit the criteria that inspired the NL approach.

We note that, in most cases considered, the relationship between the exponent hyperparameters and optimal learning rate for that exponent is simple. Although not addressed in detail here, this suggests that an efficient search approach could be defined to find a good combination.

If we include an annihilation phase then NL-SGD and NL-Mom reach losses of 86.29 and 85.71, respectively. For this simple case, the implication is that the optimizers are overshot due to amplification of stochastic effects.
7 DeepObs – Large Test Cases

In this section we consider results from the DeepObs 'large' test cases.

7.1 CIFAR-100 All CNN C

This test case applies a variation of the 'All Convolutional Net' [54] to the CIFAR-100 dataset [50]. The activation functions are ReLU and the training configuration includes dropout. The training is carried out for 350 epochs with a batch size of 256.

As in previous cases, we see that the NL versions of optimizers out-perform their non-NL counterparts, see table 5. In this case, NL-Mom performs better than any of the optimizers considered in [16].

| Algorithm      | Test Accuracy | $\nu$ | $1 - \rho$ |
|---------------|---------------|-------|------------|
| SGD           | 50.71 ± 1.85  | -     | -          |
| Momentum      | 55.92 ± 0.82  | 3e-4  | -          |
| NAG           | 56.94 ± 0.73  | 3e-4  | -          |
| Adam          | 53.93 ± 0.69  | -     | -          |
| Best (LA Mom) | 57.49 ± 1.16  | -     | -          |
| NL-SGD        | 55.05 ± 0.92  | 0.6   | -          |
| NL-Mom        | 58.31 ± 0.80  | 0.9   | 0.1        |
| NL-NAG        | 57.43 ± 0.64  | 0.8   | 0.1        |

Table 5: CIFAR-100 All CNN C

7.2 SVHN WRN

This test case implements a Wide ResNet 16-4 architecture, as proposed in [55] and applied to the SVHN dataset [56]. The activation functions are ReLU and the network includes batch normalisation. The training is carried out for 160 epochs with a batch size of 128.

In this case, all the NL algorithms perform better than all of the optimizers considered in [16], see table 6.

| Algorithm      | Test Accuracy | $\nu$ | $1 - \rho$ |
|---------------|---------------|-------|------------|
| SGD           | 94.88 ± 0.67  | -     | -          |
| Momentum      | 95.30 ± 0.19  | 0.01  | -          |
| NAG           | 95.45 ± 0.23  | 0.01  | -          |
| Adam          | 95.16 ± 0.32  | -     | -          |
| Best (AMS Bound) | 95.45 ± 0.14 | -     | -          |
| NL-SGD        | 96.30 ± 0.15  | 0.5   | -          |
| NL-Mom        | 96.29 ± 0.07  | 0.4   | 0.1        |
| NL-NAG        | 96.40 ± 0.08  | 0.4   | 0.1        |

Table 6: SVHN WRN

7.3 F-MNIST VAE

In this case, the VAE used above in section 6.3 is applied to the Fashion MNIST dataset. The training is carried out for 100 epochs with a batch size of 64.

As for the MNIST version, the results are better than the non-NL versions of the algorithms, but do not match the performance of Adam-like optimizers, see table 7.

| Algorithm      | Test Loss | $\nu$ | $1 - \rho$ |
|---------------|-----------|-------|------------|
| SGD           | 23.88 ± 0.31 | -     | -          |
| Momentum      | 28.36 ± 13.20 | 0.07  | -          |
| NAG           | 23.89 ± 0.14 | 0.07  | -          |
| Adam          | 23.06 ± 0.09 | -     | -          |
| Best (NADAM)  | 23.01 ± 0.07 | -     | -          |
| NL-SGD        | 23.71 ± 0.14 | 0.9   | -          |
| NL-Mom        | 28.19 ± 13.47 | 0.8  | 0.1        |
| NL-NAG        | 23.56 ± 0.18 | 0.5   | 0.1        |

Table 7: F-MNIST VAE

Note: Momentum and NL-Mom results are both affected by a single case that failed to converge. Momentum found a better solution of 23.86 ± 0.15 in the large search.

7.4 Tolstoi (War and Peace RNN)

The final test case applies an RNN for character prediction on War and Peace. It is build from the standard Tensorflow LSTMCell and MultiRNNCell, with tanh activation functions and dropout. The training is carried out for 200 epochs with a batch size of 50.

| Algorithm      | Test Accuracy | $\nu$ | $1 - \rho$ |
|---------------|---------------|-------|------------|
| SGD           | 61.23 ± 0.12  | -     | -          |
| Momentum      | 61.91 ± 0.11  | 0.17  | -          |
| NAG           | 61.93 ± 0.11  | 0.13  | -          |
| Adam          | 61.96 ± 0.11  | -     | -          |
| Best (RMSProp) | 62.24 ± 0.10 | -     | -          |
| NL-SGD        | 61.86 ± 0.08  | 0.7   | -          |
| NL-Mom        | 61.39 ± 0.13  | 0.6   | 0.1        |
| NL-NAG        | 61.57 ± 0.07  | 0.6   | 0.1        |

Table 8: Tolstoi

This is the only case, in addition to Quadratic Deep, where the NL-Mom and NL-NAG algorithms perform worse than their conventional equivalents (table 8). It is beyond the scope of this discussion to identify the reasons for this behaviour, but results suggest that the combination of the NL function, higher momentum and dropout add instabilities that impede good performance. Compared to the linear equivalents, both algorithms show a higher level of instability during training when the training rate is large. This makes it harder
to find an optimal point by random parameter selection. It is possible to improve performance by tuning the parameters by hand (following an approach similar to [57], we can reach an accuracy of 61.95 with NL-NAG) or removing dropout and re-searching (NL-NAG gives an accuracy of 61.97), but neither give the improvement seen in other cases.

8 Conclusion

We have presented an adaptation for DNN optimizers, which applies a non-linear function to gradients prior to applying them to the weights of a network. This was inspired by the work in [4] which showed that robustness to noise can be a significant contributor to generalisation performance. Our non-linear approach aims to encourage enhanced SNR-optimal performance by balancing the weights for correlated inputs to a node.

The results show that, with a few exceptions, the application of a NL function to weight updates does generate better performance than with the same optimizer and no NL application. For the larger image classification tasks, NL algorithms give better results than all other optimizers characterised in [16]. A benefit of this approach is that it retains a small memory footprint compared to, for example, Adam variants.

Of course, there are many possible extensions of the work presented here: we have by no means explored all the NL functions that could provide benefit, and the application of the NL adaptation could be applied to other optimizers.

The intention of this investigation was to extend the study in [4]. We have seen that the SNR-inspired approach improves performance for exactly the cases where it is most relevant, image classification, and provides no benefit in the test case where the background assumptions (correlated inputs to nodes) are not met. While this does not demonstrate the validity of SNR optimality unequivocally, it does suggest that it is a productive way of looking at optimization.

Acknowledgments Thanks to Frank Schneider for clarifications on the DeepObs benchmarking and to Max Pudney for reading the draft.

Full results will be available at http://github.com/pnorridge/nonlinear-weight-updates

References

[1] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-propagating Errors". In: Nature 323.6088 (1986), pp. 533–536.
[2] Chiyuan Zhang et al. "Understanding deep learning requires rethinking generalization". In: arXiv e-prints, arXiv:1611.03530 (Nov. 2016). arXiv: 1611.03530 [cs.LG].
[3] Chiyuan Zhang et al. “Understanding Deep Learning (Still) Requires Rethinking Generalization”. In: Commun. ACM 64.3 (Feb. 2021), pp. 107–115.
[4] Paul Norridge. “Think Global, Act Local: Relating DNN generalisation and node-level SNR”. In: CoRR abs/2002.04687 (2020). arXiv: 2002.04687.
[5] Herbert Robbins and Sutton Monro. “A Stochastic Approximation Method”. In: The Annals of Mathematical Statistics 22.3 (1951).
[6] B. T. Polyak. “Some methods of speeding up the convergence of iteration methods”. In: USSR Computational Mathematics and Mathematical Physics 4.5 (1964).
[7] Yurii Nesterov. “A method for solving the convex programming problem with convergence rate $O(1/k^2)$”. In: Soviet Mathematics Doklady 27 (1983).
[8] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: 3rd International Conference on Learning Representations, ICLR. 2015.
[9] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. "On the Convergence of Adam and Beyond". In: 6th International Conference on Learning Representations, ICLR. 2018. arXiv: 1904.09237.
[10] Timothy Dozat. "Incorporating Nesterov Momentum into Adam". In: 4th International Conference on Learning Representations, ICLR. 2016.
[11] Michael R. Zhang et al. "Lookahead Optimizer: k steps forward, 1 step back". In: Advances in Neural Information Processing Systems 32, NeurIPS (2019). arXiv: 1907.08610.
[12] Liangchen Luo et al. “Adaptive Gradient Methods with Dynamic Bound of Learning Rate”. In: 7th International Conference on Learning Representations, ICLR. 2019. arXiv: 1902.09843.
[13] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude. 2012.
[14] Irwan Bello et al. “Neural Optimizer Search with Reinforcement Learning”. In: CoRR abs/1709.07417 (2017). arXiv: 1709.07417.
Algorithms, and Applications. Cambridge University Press, 2016.

[42] Tobias Pohlen et al. "Observe and look further: Achieving consistent performance on atari". In: arXiv preprint arXiv:1805.11593 (2018).

[43] Julian Schrittwieser et al. "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model". In: CoRR abs/1911.08265 (2019). arXiv: 1911.08265.

[44] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: arXiv e-prints, arXiv:1711.05101 (Nov. 2017), arXiv:1711.05101. arXiv: 1711.05101 [cs.LG].

[45] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. "Don’t Decay the Learning Rate, Increase the Batch Size". In: CoRR abs/1711.00489 (2017). arXiv: 1711.00489.

[46] Leslie N. Smith and Nicholay Topin. "Super- Convergence: Very Fast Training of Residual Networks Using Large Learning Rates". In: CoRR abs/1708.07120 (2017). arXiv: 1708.07120.

[47] Nitish Srivastava et al. "Dropout: A simple way to prevent neural networks from overfitting". In: The Journal of Machine Learning Research 6, p. 2014.

[48] Martin Abadi et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org. 2015.

[49] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: CoRR abs/1708.07747 (2017). arXiv: 1708.07747.

[50] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Tech. rep. Department of Computer Science, University of Toronto, 2009.

[51] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: Proceedings of the IEEE 86.11 (1998), pp. 2278–2324.

[52] Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database". In: ATT Labs [Online]. Available: http://yann. lecun.com/exdb/mnist 2 (2010).

[53] Pratik Chaudhari et al. "Entropy-SGD: Biasing Gradient Descent Into Wide Valleys". In: CoRR abs/1611.01838 (2016). arXiv: 1611.01838.

[54] Jost Tobias Springenberg et al. "Striving for Simplicity: The All Convolutional Net". In: arXiv e-prints, arXiv:1412.6806 (Dec. 2014), arXiv:1412.6806. arXiv: 1412.6806 [cs.LG].

[55] Sergey Zagoruyko and Nikos Komodakis. "Wide Residual Networks". In: CoRR abs/1605.07146 (2016). arXiv: 1605.07146.

[56] Yuval Netzer et al. "Reading Digits in Natural Images with Unsupervised Feature Learning". In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011. 2011.

[57] Leslie N. Smith. "Cyclical Learning Rates for Training Neural Networks". In: CoRR abs/1506.01186 (2015). arXiv: 1506.01186.

[58] Chris M. Bishop. "Training with Noise is Equivalent to Tikhonov Regularization". In: Neural Comput. 7.1 (Jan. 1995), pp. 108–116.
A Toy Model

In this brief appendix, we consider a toy models that give some intuition related to points discussed in the main text.

A.1 Single node

First, consider a single node model (figure 6), with two inputs that have normally distribution samples $N(a, \sigma^2)$ and a constant target, $a$. The output will have the distribution $N((v_1 + v_2)a, \sigma^2(v_1^2 + v_2^2))$. Clearly, the solution is any point on the line $v_1 + v_2 = 1$. The standard deviation of the error is minimised when $v_1 = v_2 = a/2$, hence this is the weight combination that is 'SNR optimal'. (Of course, in this example, SNR-optimal is simply the MSE solution.)

![Figure 6: Toy model: single node](image)

Figures 7 and 8 show typical examples of training this model with a loss $L = (v_1 x_1 + v_2 x_2 - a)^2$. In the initial stages, the model converge to a point on $v_1 + v_2 = 1$, but one that maintains where $v_1 - v_2$ is close to its initialised value. Eventually, the regularisation effect of the inherent noise (see [58]) pushes the network to an SNR-optimal solution. These are, of course, simple examples, but act as reference points for the following discussion.

![Figure 7: Results for $v_1 = 0.01, v_2 = 0.01$](image)

![Figure 8: Results for $v_1 = 0.01, v_2 = 0.0001$](image)

A.2 Three node model

We now consider a three-node model shown in figure 9. It is easy to see that as feed-forward functions there is no real difference between (a) and (b). In fact, if $v_1 = w_{11} w_{12}$ then they are essentially identical.

![Figure 9: Toy model: three nodes](image)

However, the behaviour under SGD/backprop is notably different. To see this, let’s consider the total change in the paths from each input after a single iteration. For the two-layer function, each path is characterised by the product of the two weights. (As already noted this product has an identical role to the single weights in the 1-layer model.) After the first back-
We see that even if the products of the training process, with propagation iteration the products will be:

\[ w_{11}w_{12} \leftarrow w_{11}w_{12}(1 + 2S_1)^2 - S_1 \left( w_{11}^2 + w_{12}^2 \right) \]
\[ w_{21}w_{22} \leftarrow w_{21}w_{22}(1 + 2S_2)^2 - S_2 \left( w_{11}^2 + w_{12}^2 \right) \]

where \( S_j = \sum_a x_j^{(n)} \left( y^{(n)} - a \right) \).

We see that even if the products \( w_{11}w_{22} \) are identical at the start and \( S_1 \sim S_2 \), any difference in the way the weights are distributed can cause the two products to diverge and hence the SNR optimality to degrade.

Let’s consider a case where the SNR is good at the start of the training process, with \( w_{11}w_{12} = w_{21}w_{22} \), but with

\[ w_{12} = w_{11} \]
\[ w_{21} = w_{11}\kappa \]
\[ w_{22} = w_{11}/\kappa \]

for some \( \kappa \).

When we converge, \( w_{11}w_{12} \) will no longer be close to \( w_{21}w_{22} \). Hence, the SNR of the 2-layer model will have degraded and we will see greater error in individual results (even if the mean error remains zero).

Figure 10 (a) gives an example of the behaviour in such a case. The figure shows the shows the evolution of \( w_{11}w_{12} \) and \( w_{21}w_{22} \), which is the equivalent to \( u_1 \) and \( u_2 \) above. The weights are initialised with \( \kappa = 0.5 \); that is \( w_{11} = w_{12} = 2, w_{21} = 1.6, w_{22} = 2.5 \). This, then, is identical to the initial point for 7.

We see that when the model converges the SNR is degraded; the weighting on the two input paths no longer maintains the initial distance but diverges. For this simple case, the noise regularisation will eventually push the model to an SNR-optimal solution,

Figure 10 (b) shows the training path when we apply a signed square root to the updates, that is

\[ \Delta w_{ij} = \alpha \text{sgn} \left( \frac{af}{\sigma w_{ij}} \right) \sqrt{\frac{af}{\sigma w_{ij}}} \]

As argued in the main text, this NL approach has the goal that the initial convergence reaches to a point that is better from an SNR optimality perspective.

Figure 11 shows the evolution of \( w_{11}w_{12} \) and \( w_{21}w_{22} \), when starting with weights \( w_{11} = w_{12} = 0.1, w_{21} = w_{12} = 0.01 \). This is the equivalent starting point to in figure 8. Again, the initial imbalance leads to suboptimal SNR as the model trains; in this case, with one input dominating. Again, when we apply a signed square root, the SNR at convergence is improved.

As noted, in all these cases, the regularisation effect of the inherent noise (see [58]) eventually push to an SNR-optimal solution. However, we expect that, in more complex cases, the initial convergence will become more critical to the overall performance, as demonstrated in the results above.

From both these examples, we see that SGD applied to a DNN can frequently lead to training trajectories that are not optimal from an SNR perspective. Frequently there is a ‘success-breeds-success’ effect, where strongly weighted paths through the network are emphasised even when more balance between paths would give better performance. This raises the possibility that, in more complex networks, the training never reaches an SNR-optimal solution – or does so only after a long training time.
Figure 11: Results for $w_{11} = w_{12} = 0.1$, $w_{12} = w_{22} = 0.01$
B  Training plots

In this section we give some plots of the training over time to demonstrate the dynamics of NL algorithms compared to other options. Each plot shows one NL algorithm against SGD, Momentum and Adam. The results are over the 10 repeated training runs using the parameter set selected in the random search; the plots show the mean and standard deviation of the set.

B.1  FMNIST 2C2D

Figure 12: NL-SGD, FMNIST 2C2D (accuracy)

Figure 13: NL-Momentum, FMNIST 2C2D (accuracy)
B.2 CIFAR-10 3C3D

Figure 14: NL-SGD, CIFAR-10 3C3D (accuracy)

Figure 15: NL-Momentum, CIFAR-10 3C3D (accuracy)
Figure 16: NL-SGD and NL-Momentum, CIFAR-10 3C3D (accuracy), effect of ‘annihilation’ phase

B.3 CIFAR-100 All CNNC

Figure 17: NL-SGD, CIFAR-100 All CNNC (accuracy)
B.4 SVHN WRN164

The NL optimizers show the most interesting behaviour when applied to the SVHN WRN164 case, with accuracy improvement continuing (or even accelerating) after all the optimizers appear to have converged. If we consider the loss, we see the usual overfitting indication, but that this then levels off or, in the NL-SGD case, returns to decreasing.
Figure 20: NL-Momentum, SVHN WRN164 (accuracy)

Figure 21: NL-NAG, SVHN WRN164 (accuracy)
Figure 22: NL-SGD, NL-Momentum, NL-NAG (losses)