Negative Feedback System as Optimizer for Machine Learning Systems

Md Munir Hasan
Department of Electrical and Computer Engineering
University of North Carolina at Charlotte
Charlotte, NC 28223
mhasan13@uncc.edu

Jeremy Holleman
Department of Electrical and Computer Engineering
University of North Carolina at Charlotte
Charlotte, NC 28223
mhasan13@uncc.edu

Abstract

With high forward gain, a negative feedback system has the ability to perform the inverse of a linear or non-linear function that is in the feedback path. This property of negative feedback systems has been widely used in analog electronic circuits to construct precise closed-loop functions. This paper describes how the function-inverting process of a negative feedback system serves as a physical analogy of the optimization technique in machine learning. We show that this process is able to learn some non-differentiable functions in cases where a gradient descent-based method fails. We also show that the optimization process reduces to gradient descent under the constraint of squared error minimization. We derive the backpropagation technique and other known optimization techniques of deep networks from the properties of negative feedback system independently of the gradient descent method. This analysis provides a novel view of neural network optimization and may provide new insights on open problems.

1 Introduction

Gradient descent has long been the dominant method for optimizing weights in neural networks. It is constructed purely from a mathematical point of view with the goal to minimize a loss function. Many mathematical formulations are modeled after a physical process. The most relevant example is the deep neural network which is modeled after a biological process. Having a physical process behind a mathematical model has the advantage that the behavior of the physical process can provide intuition for the mathematical model. For example, the convolutional neural network [2], which now forms the backbone of image recognition, is inspired by the receptive field of the mammalian visual cortex [6]. Gradient descent with momentum is developed by analogy with stabilizing a heavy ball rolling down a hill. We believe that studying the physical process which describes the optimization should help us design a better optimizer. In this paper we present a negative feedback system as a physical analogy of optimization and show a close relationship to gradient descent. This optimization method is based on the ability of a negative feedback system to perform the inverse operation of a function. This principle is well known in the analog circuits and systems community and many useful

*Use footnote for providing further information about author (webpage, alternative address)—not for acknowledging funding agencies.

Preprint. Under review.
Figure 1: (a) A generic negative feedback system (b) An Operational amplifier with an exponential element in the feedback path realizes a logarithmic input-output function. The transistor Q has exponential voltage to current relationship. The feedback system implements inverse of the exponential i.e. logarithmic function.

analog circuits have been constructed [19] using this principle. Our contributions in this paper can be summarised as follows.

- We develop the method of using negative feedback system as an optimizer for the field of machine learning. We state the assumptions and establish necessary conditions to achieve convergent learning.
- We show that some non-differentiable functions that cannot be optimized through gradient descent can be learned using a negative feedback learning rule. We establish the conditions under which it works and provide a physical interpretation of these conditions.
- We show that the negative feedback system reduces to gradient descent under a squared error minimization constraint. We also derive the error backpropagation technique from within a negative feedback system and show that it is the same as backpropagation under gradient descent.
- We show that many optimization techniques such as weight decay, adaptive learning [10], and residual networks [5], which were previously developed independently of each other, can be described by the framework of a negative feedback system. We also offer a possible solution to the weight transport [12] problem, one of the major barriers to a biologically plausible neural network model.

2 Theoretical Background

For a negative feedback system as shown in Fig. [1a], if we define the forward function, backward function, and the error function with Eqs. [1], [2] and [3] respectively, then the input-output relationship is expressed by Eq. [4]. For a forward function of the form \( y = F(x) = Ax \) where \( A \) is the gain, the inverse of the forward function is \( x = F^{-1}(y) = y/A \). If the gain \( A \) is large then \( F^{-1} \rightarrow 0 \). For error function of the form \( y = E(x) = u x \) where \( u \) is the gain, \( E^{-1}(F^{-1}) \rightarrow 0 \) for high forward gain \( A \). Then the output of the feedback system becomes inverse of the backward function as in Eq. [7]. Effectively, the negative feedback system is implementing the inverse of the function that is in the backward path.

\[
\begin{align*}
x_o &= F(x_e) \quad \text{(forward function)} \quad (1) \\
x_f &= B(x_o) \quad \text{(backward function)} \quad (2) \\
x_e &= E(x_i - x_f) \quad \text{(error function)} \quad (3) \\
F^{-1}(x_o) &= E(x_i - B(x_o)) \quad (4) \\
x_i &= E^{-1}(F^{-1}(x_o)) + B(x_o) \quad (5) \\
x_i &\approx B(x_o) \quad \text{(for large \( A, \ E^{-1}(F^{-1}(x_o)) \rightarrow 0 \))} \quad (6) \\
x_o &= B^{-1}(x_i) \quad (7)
\end{align*}
\]
This property is commonly used in analog circuits in order to perform inverse operation of the transistor function \([19, 14]\). An example circuit is shown in Fig. [16] In a transistor an input voltage creates an exponential output current. However, the transistor is an uni-directional device which means that pushing a current at the output of the transistor will not produce a voltage at the input. In order to make that operation work, a negative feedback system using an operational amplifier of gain \(A\) is used which implements that inverse operation. This way an input current \(I_{in}\) into the feedback system produces the corresponding transistor voltage \(V_{out}\).

It should be noted that even if the backward function \(B\) is not completely invertible (which is the case for an uni-directional transistor), the overall system appears to be performing \(B^{-1}\). This is because the system is not using \(x_i\) (the range of \(B\)) as input to the function \(B^{-1}\) directly. Rather, as in Fig. [1a] the output of the system \(x_o\) acts as the domain of \(B\). The output of \(B\) is then compared with the target range of \(B\) i.e. \(x_i\). When the difference of \(x_i\) and \(x_f\) is zero, the overall system output \(x_o\) is approximately the output of \(B^{-1}\).

### 3 Method

#### 3.1 System Setup

To frame optimization as a negative feedback problem, we express the a layer as a function of the weights, with the inputs held constant. In a neural network, a single layer can be expressed as a function of a linear combination of \(x\) with a weight vector \(w = [w_1, w_2, \ldots, w_n]^T\) as shown in Eq. (8). There can be linear or non-linear activation function \(\sigma\) inside the function \(f\). A bias term can be easily implemented by setting an element of the \(x\) vector to 1. The variables \(x_i\) and \(y\) are training samples which are known quantities for a problem. By implementing the inverse operation of the function in Eq. (9) we can find the weights \(w_i\), which effectively implements an optimization operation.

\[
y = f(w) = \sigma(\sum_i w_i x_i) \tag{8}
\]

\[
w = f^{-1}(y) \tag{9}
\]

To implement the inverse operation using negative feedback, the function \(f\) is placed in the feedback path as shown in Fig. [2] \(x\) training samples are used in the backward function, weights are initialized randomly and \(y\) training samples are set as input to the feedback system. An initial prediction of the weight vector \(w\) is used by the backward function to produce \(y'\). Using the difference \(y - y'\) an error \(e\) is generated. The process of generating a vector \(e\) with a scalar \(y - y'\) is described in the following subsection.

#### 3.2 Stability Criteria

In order for a feedback system to be stable, the bandwidth of the system should be limited, meaning that the output should change slowly (a low frequency system). Hence, instead of changing the weight from the previous value to the new value predicted by the forward function instantly (infinite bandwidth), a small increment is made from the previous value toward the predicted value by using a
σ = identity
(a) σ = ReLU
(b) σ = tanh
(c) σ = sgn
(d)

Figure 3: Torque analogy of error. The solid and dashed lines represent y and y' respectively. The difference y - y' shown by the dotted arrows can be thought of forces acting on the x-axis. The resulting total torque \( \sum_k (y[k] - y'[k])x[k] \) is the output of the error function given by Eq. (12).

This is similar to using a small learning rate in gradient descent. The prediction labeled w' from the forward function goes into a low pass filter characterised by a time constant \( \tau \) and arbitrary constant \( \gamma \) which outputs slowly varying \( w \). This new value of \( w \) goes around the feedback loop again and with consecutive iterations around the feedback loop, the output converges to the optimum value of \( w \). The weight update method because of the low pass filter is given in Eq. (10) where the quantity \( \eta = \partial t/\tau \) acts as the learning rate. The superscript \( t \) denotes the weight at time \( t \) during iteration.

Another important criterion for stability is that the gain around the feedback loop must be negative when the magnitude is greater than unity \[23\]. From Fig. 2, the forward gain is \( A \) and the backward gain is \( \beta = \partial y'/\partial w \). The loop gain of the system is \(-1 \times A\beta\). Hence, we have to make sure that the product of the forward and backward gain for each component of \( \beta \) is always positive. The forward gain \( A \) is typically positive. If any component of \( \beta \) is negative for a training sample then the corresponding element of the gain product becomes negative. In general, if we use a forward gain of \( A\beta \), then the element-wise product of forward and backward gain is \( A\beta \times \beta = A\beta^2 \) which is guaranteed to be positive. With \( A\beta \) as the forward gain, the forward function can now take scalar error \( y - y' \) and produce vector \( w' \) as shown below.

\[
w' = A\beta \times (y - y') = A \times \beta(y - y')
\]

In (11), we can separate \( \beta \) from the forward gain and attach it to \( y - y' \). This way we can keep using a forward gain of \( A \) and use \( e = \beta(y - y') \) as the new error. The error is now a function of scalar \( y - y' \) which is shown by an error function block \( E \) in Fig. 2. We also notice that the error function is of the form \( e = E(x) = ux \) as assumed in section 2. The error is calculated by multiplying the difference \( y - y' \) with backward gain \( \beta \). Thus the gain of the error function is \( u = \beta \).

4 Application in Machine Learning

In the following sections, we apply this method starting with simpler regression problems and then gradually develop methods for complex problems such as deep neural networks.

4.1 Regression on Noise-Free Data

In machine learning, the activation functions can be unity, ReLU, tanh etc. The backward gain of the feedback system for any activation function is \( \beta = \partial y'/\partial w = \sigma' x \) where \( \sigma' \) is the gain of the
activation function. From section \[3.2\] we know that the gain of the error function needs to be \( u = \beta \) in order to make the forward and backward gain products positive. We notice that if the activation function is monotonic and has non-negative \( \sigma' \) then the source of negative component in \( \beta \) is only \( x \). Hence, we can simplify the gain of error function as \( u = x \). For a single training sample, the error corresponding to \( i^{th} \) weight is \( \epsilon_{wi} = u_i(y - y') = x_i(y - y') \). With many training samples the error is the sum of the errors from all the samples. The error for all the weights can be expressed as matrix multiplication, as in Eq. \[\ref{eq:12}\], where \( \epsilon_{wi[k]} = x_i[k] \) is the error gain for \( i^{th} \) weight and \( k^{th} \) sample. For all the training samples the error function gain becomes a matrix \( U \).

\[
\epsilon = E(y - y') = U(y - y')^T = \begin{bmatrix}
\epsilon_{w[1]} \\
\epsilon_{w[2]} \\
\vdots \\
\epsilon_{w[n]}
\end{bmatrix} = \begin{bmatrix}
y[1] - y'[1] \\
y[2] - y'[2] \\
\vdots \\
y[m] - y'[m]
\end{bmatrix} = \begin{bmatrix}
e_{w[1]} \\
e_{w[2]} \\
\vdots \\
e_{w[n]}
\end{bmatrix}
\]

(12)

It is interesting to see that no information of the activation function is needed in Eq. \[\ref{eq:12}\] in order to determine the weights. We provide an intuition for this using the analogy of torque in Fig. \[3\]. We choose a simple target data set \( y \) which comes from a process \( y = \sigma(wx) \) where weight \( w \) is positive. Four different activation functions are shown in Fig. \[3\] with the training data \( y \) (solid blue), \( y' \) (dashed red) for randomly initialized weight. The error at each training sample is \( (y - y')x \). The difference \( y - y' \) (dotted arrow line) can be thought of as a force acting on the \( x \) axis. Thus, there is a torque on the \( x \) axis with origin as the axis of rotation. The sum of the torques represents the total error. In all four cases the resulting torque is positive. Thus, the weight will change in the positive direction. This is true regardless of the activation function being used as long as the gradient of the activation is non-negative. This makes it possible to perform regression even on the signum function \( sgn \) which is non-differentiable. The \( sgn \) function has zero gradient and also has a discontinuity at the origin. However, the gradient at the origin can be represented as a Dirac delta function which is infinite in value but is non-negative. The ReLU and tanh also have a non-negative gradient. Thus \( u = x \) can be used in these cases. If the gradient of activation is negative then \( u = sgn(\sigma')x \) is used.

A comparison between training with negative feedback and Stochastic Gradient Descent (SGD) is shown in Fig. \[4b\] for a non-differentiable function in Fig. \[4a\]. A value of \( \gamma = 1 \) and \( \eta = 10^{-2} \) is used in Eq. \[\ref{eq:10}\]. For SGD, the gradient of the activation is used which is mostly zero. As a result average squared error in SGD does not change. However, the error is decreasing in the negative feedback optimization. It can correctly learn the weight and the bias terms without using the gradient of the activation.

### 4.2 Regression on Noisy Data

Although the negative feedback system can learn a function properly with \( u = x \), it can only do that with clean noise-free training data. For noisy training data, \( u = x \) cannot achieve minimum squared error when a non-linear activation is present. The reason for this is shown in Fig. \[4c\] using a ReLU activation. In the figure, although \( y' \) and \( y \) matches exactly on the linear part, the noisy data points on
the saturated part of the activation (where ReLU is zero), still creates a positive torque. As a result, the negative feedback system optimizer will increase the slope further. In order to prevent that, we must isolate and ignore $y - y'$ for the saturated parts of the activation function by using a window function. The actual error function gain $u = \beta = \sigma'$ gives us that window function as $\sigma'$. Thus when the torque $(y - y')x$ is multiplied with the window function $\sigma'$, the torque contribution from the saturated parts towards the net torque will be zero. With this interpretation of $\sigma'$ as the window function that isolates saturated and non-saturated parts we can approximate a window function for the non-differentiable functions as well and minimize squared error which we show in section 6.4.

4.3 Single Layer Classifier

The regression problem can be turned into a perceptron classifier by using softmax or tanh as the activation function. Hence, Eq. (12) also represents the error function for a single layer perctorn. For a multi class classifier the error function is simply the extension of Eq. (12). The single row of $y - y'$ becomes a matrix with $y - y'$ of different classes stacked as rows.

4.4 Deep Network

To use this system in deep networks, a method for error backpropagation is needed. A network is shown in Fig. 5 with $l$ denoting layer number. The low pass filters haven been omitted in the figure for simplicity. The input from second to last layer $y^{l-2}$ generates the final output $y^l$. We treat the output $y^l$ as a result of the input $y^{l-2}$ as in Eq. (13). For $c^{th}$ class output it can be expressed as Eq. (14). The backward gain for a weight is given by Eq. (15). Multiplying the difference $d^l_c = (y^l_c - y'^l_c)$ with the backward gain, we can write the error for $e_{w^{l-1}_{j_i}}$ as Eq. (16).

$$y^l = \sigma(W^l \sigma(W^{l-1} y^{l-2}))$$  \hspace{1cm} (13)

$$y^l_c = \sigma(\sum_i w^l_{ic} (\sigma(\sum_j w^{l-1}_{ji} y^{l-2}_j)))$$ \hspace{1cm} (14)

$$\beta_{w^{l-1}_{j_i}} = \sigma'' w^l_{ic} \sigma'^{-1} y^{l-2}_j$$ \hspace{1cm} (15)

$$e_{w^{l-1}_{j_i}} = \sigma'' y^{l-2}_j \sum_c \sigma'' w^l_{ic} d^l_c$$ \hspace{1cm} (16)

$$e_{w^{l-1}_{j_i}} = \sigma'' y^{l-2}_j d^l_c = u^{l-1}_{j_i} d^l_i$$ \hspace{1cm} (17)

The sum over $c$ expresses the fact that every class output is influenced by $w^{l-1}_{j_i}$. With $d^{l-1}_i = \sum_c \sigma'' w^l_{ic} d^l_c$ in Eq. (17), $d^{l-1}_i$ can be thought of as the difference error for layer $l - 1$. Also, $u^{l-1}_{j_i}$ represents the error function gain. The outcome is shown in Fig. 5. The difference vector of the last layer is multiplied with $\sigma'' W^l$ which produces the difference vector for the previous layer. This way error is backpropagated to all the previous layers.

5 Comparison with Gradient Descent

For a negative feedback system it is important that the forward and backward gain product for each weight is positive. The gain of the error function as $u = \beta = \sigma'$ satisfies that condition. This condition is
also necessary for squared error minimization as shown in section 4.2. In fact we can use \( u = B^n \) as the gain as well where \( n \) is an odd positive integer. This way the negative feedback system represents an infinite number of optimizers. The reason for odd positive \( n \) is that it preserves the sign of \( B \).

When \( n = 1 \), the negative feedback system error implements the error gradient of the gradient descent optimization method. The gradient descent method minimizes a loss function, e.g. squared error as in Eq. (18). The weight parameters are updated by going in the opposite direction of the gradient which is given by Eq. (19) for a weight \( w_i \). Using \( u = B \) in Eq. (12), the feedback error for a weight \( w_i \) is given by Eq. (20). We see that both expressions are same except for a factor of \( 2/m \). The relationship between the two is \( e_{w_i} = (m/2)(-\nabla q_{w_i}) \).

At this stage we can see that with \( u = B \) which is the condition for squared error minimization, the negative feedback system and gradient descent method are equivalent. Also, by noticing Fig. 5 one can easily realize that the error propagation to previous layers is the same as the backpropagation technique in gradient descent method [17]. We have derived it only using the properties of the negative feedback system. Thus, the negative feedback system allows us to look at and analyze the optimization problem from a different perspective. In gradient descent the objective is to minimize a loss function. However, in negative feedback system, the objective is inverse the backward function.

### 6 Optimization Techniques derived from Negative Feedback System

In this section we will describe some commonly used optimization techniques used under gradient descent method which can be established from the properties of the negative feedback system.

#### 6.1 Weight update with Adaptive Momentum

![Figure 6: (a) Two first order low pass filters in cascade implements weight update with momentum (dark lines). Another low pass filter (gray lines) implements Filter Assisted Learning rate (FAL). (b) Comparison between Adam and FAL on CIFAR10 using ResNet20.](image)

The first order low pass filter in Fig. 2 may not be enough to control the speed in practice. In order to better control the stability, we can use a second order low pass filter which is just two first order low pass filters in cascade. This setup is shown in Fig. 6a (dark lines). The output of the high gain forward function \( w' \) goes into the first low pass filter with time constant \( \tau_m \). The output from that filter \( m \) goes into the second low pass filter with time constant \( \tau \) which updates the weight. Similar
to Eq. (10), the output of the first low pass filter can be written as follows.

\[
\begin{align*}
\hat{m}^t &= m^{t-1} + (w^t - m^{t-1})\eta_m \quad (\text{where } \eta_m = \frac{\partial t}{\tau_m}) \\
\dot{m}^t &= \beta_m m^{t-1} + (1 - \beta_m)w^t \quad (\text{where } \beta_m = 1 - \eta_m)
\end{align*}
\] (22)

The expression in Eq. (22) is exactly the same as the momentum estimation [10] as used in gradient descent. However, the filter system is capable of controlling the speed of the system only up to a certain value (given by the cut-off frequency of the filter). As a result, if the cut-off frequency is large (small \(\tau_m\) or large \(\eta_m\)) substantial oscillations from \(w^t\) can pass through. In order to stop the oscillation to pass through the second filter we need to increase \(\tau\) (consequently decrease \(\eta\)). This can be done by detecting the presence of substantial oscillation in \(w^t\) and decreasing \(\eta\) proportionally by the strength of that oscillation. The strength of oscillation is proportional to the standard deviation of \(w^t\) (the more \(w^t\) deviates from the mean, the more the oscillation). The variance of \(w^t\) is \(Var(w^t) = E[w^t] - E[w^t]^2\) where \(E[\cdot]\) is expectation or average value. A first order low pass filter is in fact a moving average calculator. Hence, we already have \(m = E[w^t]\). Additionally, we can measure \(v = E[w^t]^2\) with another low pass filter with time constant \(\tau_v\) as shown in Fig. 6a (gray lines). Thus, \(\tau\) can be proportionally scaled using the standard deviation \(\sqrt{v - m^2}\). Using Eq. (22), \(v\) is given by \(v^t = \beta_v v^{t-1} + (1 - \beta_v)w^t / 2\). At this point, the stabilization method becomes extremely similar to the Adaptive Momentum (Adam) estimation [10] method. In Adam, the subtraction by \(m^2\) is not applied and there are bias correction process. A comparison between the performance between Adam and Filter Assisted Learning rate (FAL) derived from negative feedback system is shown in Fig. 6b with CIFAR10 [11] dataset on ResNet20 [5]. A learning rate of \(10^{-3}\), \(\lambda = 1\) and a batch size of 128 is used. For Adam default settings are used [10]. For FAL we have used \(\beta_m = \beta_v = 0.99\) because both filters needs to be the same type of average calculator. Even without the bias correction in FAL, the performance is comparable.

6.2 Predicting and Fixing a Dead network

The output of the error function is \(e = \beta(y - y')\). The backward function inversion or optimization is complete when \(y = y'\) and consequently \(e = 0\). However, \(e\) can become zero even for \(y \neq y'\) when \(\beta\) is zero. Hence, it is important to make sure that \(\beta \neq 0\) for negative feedback to work. Since, \(\beta = \alpha' x\), we need to make sure that either \(\alpha'\) or \(x\) (inputs to a layer) or both is not zero for all the training samples. With this criterion, it is now easy to predict a dead network. In Fig. 5 if layer inputs \(y^{l-1}\) and \(y^{l-2}\) become zero, negative feedback system stops working as an optimizer. This is the same phenomenon as seen in the dying ReLU [15] problem. A straightforward fix would be to use an activation function that does not saturate to zero. A leaky ReLU [4] or exponential ReLU [1] naturally satisfies this condition.

6.3 Fixing Vanishing Gradient

As mentioned in the previous paragraph, negative feedback stops working as an optimizer when either \(\alpha'\) or inputs to a layer becomes zero. In the case when \(\alpha'\) is zero or close to zero, a similar phenomenon happens but this time because of the gradient of the activation function called the vanishing gradient problem. From negative feedback system analysis, the solution is to make sure that \(\beta \neq 0\). A solution would be to use leaky ReLU as activation function. Another solution would be to introduce a non-zero additive gradient term \(\sigma_{nz}\) in the backward gain expression as \(\beta = (\alpha' + \sigma_{nz}) x\). The resulting backward path function takes the form as shown in Fig. 7a. If we choose \(\sigma_{nz}\) as an identity function, we get the skip connection of the residual network as described in [5]. The matrix \(W_{nz}\) matches the size of \(x\) with the size of \(y'\). When the size of \(y\) and \(x\) are same \(W_{nz}\) can be replaced by an identity mapping. In [5], the authors hypothesize that it is easier to train a deep network by skip connection. From the framework of negative feedback system, the idea of the skip connection comes naturally when we try to make the backward gain \(\beta\) non-zero.

6.4 Window Function for Non-Differentiable Activation

In section 4.2 we have shown that a window function is necessary in order to minimize squared error. The window function sets \(y - y'\) to zero where activation is saturated and accounts for \(y - y'\) where
Figure 7: (a) Fixing the vanishing gradient problem by introducing non-zero gradient term in the backward path. (b) Approximating the gradient of non-differentiable function.

activation is changing. For differentiable functions, the gradient of the activation $\sigma'$ can serve as the window function. For non-differentiable activation functions we can use an approximated window function as shown in Fig. 7b. A signum function is used as an example. A window function is taken as the approximated gradient while making sure that it captures the changing part of the activation. This can be done using the two points immediately next to the discontinuity ($z_+$ and $z_-$) as shown in Fig. 7b. For stability purpose, in order to avoid very large gradients, the resulting gradient is clipped to a value of 1. However, the process of finding the points $z_+$ and $z_-$ can be computationally intensive specially when the input data is not sorted.

6.5 Weight Transport

It is widely accepted that backpropagation is not compatible with the biological brain because there are no known mechanisms how the synaptic weights are transmitted to the backpropagation network [12, 13, 22]. However, Fig. 5 might shed some light on how the weight transport can happen. Instead of transmitting the synaptic weight from the backward path (‘backward’ in terms of the negative feedback topology) to the backpropagation path, both the paths draw the weights from the high forward gain block. The high forward gain block acts as the weight generator which supplies the weights to backward path function and backpropagation path function. The forward gain block is not a matrix multiplying block. However, together with the error function block the forward gain block can be considered as a matrix multiplying block. Thus functionally it is the same type of block as the backward path blocks and the backpropagation path blocks. Hence, it can be implemented by a neuron as well. Thus, we can hypothesize that there could be neurons in the brain that generate the weight-signals by which the other neurons adjust their respective weights.

7 Discussion

Although the negative feedback system seems to offer some solutions to dying ReLU and vanishing gradient problem, it does not seem to offer any solution to the weight initialization method. If the initial weight is negative for the particular case in Fig. 7b, the training will fail. However, deep networks can still be trained if weights are initialized properly [3, 4, 15]. Weight initialization depends on the statistical properties of the network and data. However, in the negative feedback system, the stability constraint $u = \sigma'x$ is not concerned with the statistics of the data. For the same reason, it also does not predict the batch normalization [8] process which have been proven to be useful in training deep networks. In this paper we have mostly focused our discussion on the neural network optimization. The generalization to other optimization methods such as particle swarm [9], genetic algorithm [21] or neuroevolution [20] needs further investigation.

8 Conclusion

In this paper we have presented the negative feedback system as the physical process of computation for optimization. We think it can connect learning algorithm and analog circuit design thereby paving the way for low power optimization circuits by utilizing the physics of electronic devices [7, 16, 18]. Many optimization techniques are easily derived from the properties of negative feedback system such as weight decay and adaptive momentum. The negative feedback system also predicts the problems associated with training deep network such as dying ReLU and vanishing gradient. Not
only that it also offers solution to those problems in an explainable way. We believe this novel view of neural network optimization is capable of providing valuable insights on the optimization techniques which many open problems in machine learning can benefit from.

References

[1] D. A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). International Conference on Learning Representations, 2016.

[2] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36(4):193–202, April 1980.

[3] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, volume 9, pages 249–256, 2010.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In 2015 IEEE International Conference on Computer Vision (ICCV). IEEE, December 2015.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, June 2016.

[6] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. The Journal of Physiology, 195(1):215–243, March 1968.

[7] Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynnä, Fopefolu Folowosele, Sylvain Saighi, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yinxue Wang, and Kwabena Boahen. Neuromorphic silicon neuron circuits. Frontiers in Neuroscience, 5, 2011.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32nd International Conference on Machine Learning, volume 37, pages 448–456, 07–09 Jul 2015.

[9] J. Kennedy and R. Eberhart. Particle swarm optimization. In Proceedings of ICNN’95 - International Conference on Neural Networks. IEEE.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. International Conference on Learning Representations, 2015.

[11] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[12] Qianli Liao, Joel Z Leibo, and Tomaso Poggio. How important is weight symmetry in back-propagation? In AAAI Conference on Artificial Intelligence. IEEE, 2016.

[13] Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. Nature Communications, 7(1), November 2016.

[14] Shih-Chii Liu, Jörg Kramer, Giacomo Indiveri, Tobias Delbrück, and Rodney Douglas. Analog VLSI: Circuits and Principles, pages 126–127. The MIT Press, 2002.

[15] Lu Lu. Dying ReLU and initialization: Theory and numerical examples. Communications in Computational Physics, 28(5):1671–1706, June 2020.

[16] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumslawksa, and Giacomo Indiveri. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. Frontiers in Neuroscience, 9, April 2015.
[17] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[18] Rahul Sarapeshkar. Analog versus digital: Extrapolating from electronics to neurobiology. *Neural Computation*, 10(7):1601–1638, October 1998.

[19] Rahul Sarapeshkar. *Ultra Low Power Bioelectronics*, pages 42–43. Cambridge University Press, 2009.

[20] Kenneth O. Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, January 2019.

[21] Yanan Sun, Bing Xue, Mengjie Zhang, Gary G. Yen, and Jiancheng Lv. Automatically designing CNN architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics*, 50(9):3840–3854, September 2020.

[22] Will Xiao, Honglin Chen, Qianli Liao, and Tomaso Poggio. Biologically-plausible learning algorithms can scale to large datasets, 2018.

[23] Karl Åström. *Feedback Systems : an introduction for scientists and engineers, second edition*, page 268. Princeton University Press, 2008.