Neural, Parallel, and Scientific Computations 23 (2015) 343-356

A LYAPUNOV BASED ADAPTIVE AND STABLE NEURAL NETWORK WEIGHT REGULARIZATION ALGORITHM

UDHAYA S. RAVISHANKAR

ABSTRACT. In this paper, a Lyapunov based neural network weight regularization algorithm is designed based on a new insight into online weight regularization by gradient descent. Lyapunov stability theory is used for designing the adaptive learning rates for error minimization and weight regularization so that stability in learning is achieved. Furthermore, the regularization algorithm ensures that the fundamental objective of learning (i.e. error minimization) is not compromised at all. With these attributes, the regularization algorithm presented in this paper is practically applicable to real world very large dataset problems, such as computer vision. However, performance results of this algorithm are shown on 2D weight space environments and simple neural network modeling and classification examples, only to provide clear visualizations of the key ideas introduced. In effect, the paper intends to primarily establish the foundation for online weight regularization by gradient descent on future applications.

AMS (MOS) Subject Classification. 39A10.

1. INTRODUCTION

Neural Networks, over the past few decades, have become feasible tools for machine learning in the industry due to the advancements made in computing technology over the years. Nowadays, neural networks with tens of million parameters are being trained on very large scale datasets for designing state-of-the-art commercial artificial intelligent (AI) systems. In addition to advancements made in computing, other key algorithmic developments in training neural networks have recently occurred to make the design of such AI systems possible. These key algorithmic developments are: 1) deep convolutional neural networks (LeCun & Bengio, 1995; Krizhevsky et al., 2012; Clark & Storkey, 2014; LeCun et al., 2015), 2) neural language modeling (Bengio et al., 2003; Morin & Bengio, 2005; Collobert et al., 2011), 3) neural recommender systems (Wang et al., 2014), and 4) adaptive critic designs (Prokhorov & Wunsch, 1997), the details of which are beyond the scope of this paper.

In all of these algorithmic developments, gradient-descent methods are adopted because of the mathematical flexibility it provides for different application requirements. Despite this mathematical flexibility, training algorithms developed under gradient descent suffer from the need to carefully select suitable learning rates if

Received September 15, 2015

learning is to be stabilized throughout the training process. Several authors have already attempted to solve this problem in a variety of ways: some requiring additional constants that again need careful selection, while others eliminating such requirements completely (Mandic & Chambers, 2000; Behera et al., 2006; Nasr & Chtourou, 2009; Kathirvalavakumar & Subavathi, 2012; Damak et al., 2013; Schaul et al., 2013).

Also more important in training neural networks is the regularization (or minimization) of the network weights to minimize overfitting: a problem that typically occurs when weights have grown too large or when there exists excessive number of neurons in the hidden layers. Minimizing overfitting is known to improve the prediction and classification capabilities of neural networks, and therefore weight regularization (or weight decay) is always encouraged wherever possible (Hinton, 2010). There exists several methods for weight regularization. Larsen et al. (Larsen et al., 1998) introduced a gradient descent based algorithm for adapting the weight regularization parameters by including the generalization error on a validation set in the algorithm's cost function. This algorithm was then improved in the same year by adopting the conjugate gradient method (Goutte & Larsen, 1998). Since the algorithm includes the generalization error on a validation set in the cost function, it is restricted to only offline training applications. A similar but more recent algorithm adopts the idea of generalization on a validation set, but as applied to recommender systems (Rendle, 2012). Crammer (Crammer et al., 2009; Crammer & Checik, 2012), on the other hand, developed an online adaptive weight regularization algorithm that, instead of using gradient descent, uses multiple models distributed in gaussian to estimate the regularization parameters from the mean and covariances. Another different approach introduced very recently by Srivastava et al. (Srivastava et al., 2014) is to drop out the effects of certain neurons online by some probability of drop out.

In this paper, the presented regularization algorithm adopts gradient descent for adapting the weight regularization parameters, but through a new criterion in the cost function that allows the neural network to obtain a local optimum (or minimum) weight solution by online training. This new criterion is established based on the detailed understanding of the weight solution space, which will be covered in the subsequent sections of this paper. As mentioned earlier, since gradient descent algorithms suffer from the need to carefully select suitable learning rates, the paper also includes methods for adapting these rates so that the error and weight minimizations are stabilized. The adaptive learning rate methods developed in this paper adopt ideas from the work of other authors referenced earlier in this section, specifically from the work of Behera et al. (Behera et al., 2006) which uses Lyapunov stability theorems in the design of the adaptive algorithm. Furthermore, the presented algorithm also attempts to eliminate the requirement of additional constants that again need careful selection, but not fully.

Following this section, the weight solution space is formally introduced along with the derivation of the new criterion for the algorithm's cost function. The next section (i.e. Section 3) then presents the Lyapunov based adaptive learning rate and weight regularization algorithms, followed by its demonstration on 2D weight space examples in Section 4. Section 5 applies the algorithm on neural network modeling and classification examples, before concluding in Section 6 with future work.

2. THE WEIGHT SOLUTION SPACE AND THE NEW CRITERION

Given a function $f(x, w)$, where x represents the input and w represents the parameters or weights, the general task is to find a solution in w that fits a desired function characteristic or a decision boundary that separates two or more classes. If there are equal number of minimally correlated data points to the number of parameters in w, a unique solution can be found. However, in the real world, this is impractical and therefore multiple solutions in w exist for the same problem. This region containing all the solutions to the problem is called the weight solution space and the goal is to find an optimal weight solution in this solution space. The following subsections address this by first describing the solution space of a function in general, and then discussing the required characteristics of the weight regularization algorithm so that it meets the regularization objective.

2.1. THE WEIGHT SOLUTION SPACE.

If for a given function in the weight space W, say $h(W)$, represents all the possible configurations the function can take, then the solution space in W is a subset $W^* \in W$ that satisfies the solution objective. A weight regularization algorithm, then must be capable of searching for the best weight solution within this solution space, W^*.

For the function $f(x, w) : w \in W$, the goal of weight regularization is to find a solution in the constraint W^* such that the total norm of $w \in W^*$ is minimized. One way to formulate this is as,

$$\begin{aligned} &\text{minimize} \quad w^T w \\ &\text{subject to} \quad f(x, w) = g \end{aligned} \tag{2.1}$$

where the constraint $f(x, w) = g$ represents the solution space $W^* \in W$, with g representing the supervisory signal given during the training of the function $f(x, w)$. The above optimization objective can be solved by formulating and solving a Lagrangian function L as,

$$L = w^T w + \mu^T (f(x, w) - g) \tag{2.2}$$

$$\nabla_{w,\mu} L = 0 \tag{2.3}$$

where the solution to (2.3) provides the solution in W^* that satisfies the optimization objective.

2.2. **THE NEW CRITERION.** With neural networks that are highly nonlinear and have a large number of weights, it becomes infeasible to formulate and solve such a Lagrangian function in (2.2) directly. Hence, gradient descent methods must be used instead. The cost function commonly used to design such an algorithm is,

$$J = \frac{1}{2} w^T \Gamma w \tag{2.4}$$

where Γ is a positive semi-definite matrix. Much like the work of Larsen (Larsen et al., 1998; Goutte & Larsen, 1998), wherein the weights are updated to minimize the error objective and the regularization parameters are updated to minimize the generalization objective, the new criterion for weight regularization introduced in this paper makes a similar distinction. This is to ensure that weight regularization does not compromise any part of the error minimization objective (i.e. the fundamental objective of learning) thereof. To achieve this, the Γ is computed as a function of the regularization vector λ as,

$$\Gamma = \lambda \lambda^T \tag{2.5}$$

which effectively transforms (2.4) into,

$$J = \frac{1}{2} (\lambda^T w)^T (\lambda^T w) \tag{2.6}$$

From (2.6), if the cost function reaches the minimum value of zero, then $\lambda^T w = 0$. Or in other words, the regularization vector must be orthogonal to the weight vector if the objective is to be satisfied. The trick now is to compute this orthogonal vector λ from the limited information available during online training. This information can be the weights or the per-sample (i.e., a single or a mini-batch of supervisory patterns) error-minimization gradient.

Computing the orthogonal vector λ with respect to the weights is definitely not a good idea because the cost function in (2.6) will always be zero, and this does not provide any valuable information to the weight update in order to meet the optimization objective in (2.1). However, the error-minimization gradient (or error-gradient) can provide this valuable information. The following lemma discusses this.

Lemma 2.1. *Let the error-gradient be denoted as ∇_e, and the weight update dynamics for error-minimization be given by,*

$$w(k+1) = w(k) - \eta \nabla_e(k) \tag{2.7}$$

with k as the iteration instant and η as the learning rate, then it follows from (2.7) *that if $\lambda^T \nabla_e = 0$, the optimal weight solution is obtained when $E[\nabla_e^T w]$ is maximized.*

Proof. If $\Delta w = w(k+1) - w(k)$, then from (2.7), it can be shown that $E[\Delta w^2] = \eta^2 E[\nabla_e^2]$ and $E[\Delta w^2] = E[w^2]$, and therefore $E[\nabla_e^2] = \rho E[w^2]$ for some $\rho > 0$.Then,

$E[\nabla_e^T w]$ is maximized when $\angle(\nabla_e, w) = n\pi$, $n = 0, 1, 2, \ldots$ implying that $\lambda^T w = 0$, given $\lambda^T \nabla_e = 0$. □

Remark 2.2. The optimal weight solution is obtained as per the cost function (2.6) if w and ∇_e are maximally correlated and the vector λ lies tangent to both w and ∇_e.

Hence, according to (2.6) and Lemma 2.1, the new criterion for weight regularization is the minimization of $\lambda^T w$ given that $\lambda^T \nabla_e = 0$. In this paper, the $\lambda^T w$ is termed as the regularization signal, much like an error signal.

3. THE LYAPUNOV BASED ADAPTIVE LEARNING RATE AND WEIGHT REGULARIZATION

As mentioned earlier in the introduction, gradient descent algorithms suffer from the need to carefully select suitable learning rates, specifically to ensure that learning is stabilized throughout the training process. The work of Behera (Behera et al., 2006) particularly addresses this by using Lyapunov stability theory in the design of the adaptive learning rate algorithm. Behera's algorithm uses information such as the error and the error-gradient (i.e. the arguments of the cost function and its derivative) to compute the adaptive learning rate. However, in this paper an adaptive learning rate independent of the cost function arguments is desired, mainly to ensure that the algorithm's performance is not unnecessarily degraded in the vicinity of the solution. In the following subsections, the adaptive learning rate algorithms modified from Behera's Lyapunov stability method will be derived in a manner that is free from the cost function arguments.

3.1. ADAPTIVE LEARNING RATE.

Given a neural network function $\hat{y} = W^T \phi(x, w)$ with K outputs, where $\phi(x, w)$ is a vector of sigmoid functions (or neurons) in $\Re^M$ and W is the weights of the output layer in $\Re^{M \times K}$, the function is subjected to a set of training pattern vectors, $y \in \Re^{P \times K}$. The $\phi(x, w)$ in fact represents the rest of the neural network in a compact form. For simplicity sake, if $K = 1$ (i.e. only one output), then the Lyapunov cost function for error minimization can be written as,

$$J = \frac{1}{2} e^T e, \quad e = \hat{y} - y \tag{3.1}$$

The following theorem presents the adaptive learning rate algorithm that ensures stability in error-minimization and free from the cost function arguments.

Theorem 3.1. *Given the Lyapunov cost function in* (3.1)*, the adaptive learning rate* η *in* (2.7) *is given by,*

$$\eta = E\left[\frac{\alpha_0}{2E\left[\nabla_W^2\right]_P}\right]_M \tag{3.2}$$

$$or \quad \eta = \frac{\alpha_0}{2\sum_M\left(E\left[\nabla_W^2\right]_P\right)\cdot E\left[\nabla_W^2\right]_P} \tag{3.3}$$

for some $\alpha_0 \in [0,1]$ *if stability in error-minimization is to be achieved, where* ∇_W *is the function gradient w.r.t* W*, and* $E[\cdot]_P$ *and* $E[\cdot]_M$ *are the the expected values in the pattern cardinality* P *and weight cardinality* M *respectively.*

Proof. Given the Lyapunov cost function in (3.1), the change in the cost function is derived as,

$$\Delta J = E\left[\nabla_W^T e\right]_P^T \Delta W \tag{3.4}$$

where ΔW is the desired weight update law for minimizing the cost function (3.1). If ΔW is chosen as,

$$\Delta W = -\eta E\left[\nabla_W^T e\right]_P \tag{3.5}$$

which is the popular Error-Backpropagation formula (Werbos, 1994), then ΔJ is given by,

$$\Delta J = -\eta E\left[\nabla_W^T e\right]_P^2 \tag{3.6}$$

for some positive learning η. Equation (3.6) ensures that the dynamics of the cost function (3.1) is Lyapunov stable. The goal now is to estimate the learning rate η so that the overall dynamics is exponentially stable, i.e. $\Delta J = -\alpha_0 J$, $\alpha_0 \in [0,1]$. Placing (3.6) in cost function (3.1), the dynamics of the cost function is modeled as,

$$J(k+1) = \frac{1}{2}e^T(k)e(k) - \eta E\left[\nabla_W^T(k)e(k)\right]_P^2 \tag{3.7}$$

where k is the iteration instant. If $E\left[\nabla_W^T(k)e(k)\right]_P^2$ in (3.7) is scaled down by the $2E\left[\nabla_W^2(k)\right]_P$, then it can be easily shown that $e^T(k)e(k) \geq \frac{E\left[\nabla_W^T(k)e(k)\right]_P^2}{E\left[\nabla_W^2(k)\right]_P}$, and therefore the dynamics become exponentially stable (or asymptotically stable) as desired. Since the dynamics in (3.7) are w.r.t a single weight in W, the learning rate η can be averaged over all M weights in W and be finally written as in (3.2), with $\alpha_0 \in [0,1]$ but $\alpha_0 \sim 1$. A much faster learning can be achieved if, instead of averaging over all the M weights, the learning rate is scaled by $\frac{1}{\sum_M\left(E\left[\nabla_W^2(k)\right]_P\right)}$. This is because

$$M \cdot \sum_M \left(E\left[\nabla_W^2(k)\right]_P\right) < \frac{M}{P}$$

since $\nabla_W^2 = \phi^2(x,w)$ and $|\phi(x,w)| < 1$ and therefore $\nabla_W^2 < 1$. $\frac{M}{P}$ is the value the learning rate in (3.2) gets divided by if the true expected value is to be obtained as per the dynamics in (3.7). □

Remark 3.2. Note that the adaptive learning rate derived in this section was for the output layer weights W. The same can be extended for the other weights w in $\phi(x, w)$ without further modification. But, if the adaptive learning rate in (3.3) is used for w, then the $E\left[\nabla_W^2\right]_M$ retains since ∇_w vanishes layer by layer, as this can effectively throw the learning rates out of bounds. Therefore, for the weights w, the denominator of (3.3) will be $2E\left[\nabla_W^2\right]_M \cdot E\left[\nabla_w^2\right]_P$. The adaptive learning rate in (3.2) is in fact a variant of the optimal learning rate presented by Mandic and Chambers (Mandic & Chambers, 2000), while the modification in (3.3) was inspired by the work of Jagannathan and Lewis (Jagannathan & Lewis, 1996).

3.2. **ADAPTIVE WEIGHT REGULARIZATION.** Lemma 2.1 set the criterion for adapting the parameters in λ so that the weight regularization objective is met. However, to ensure that the weight minimization is stabilized, Lyapunov stability theory will have to be applied. Firstly, λ must be computed as a vector orthogonal to the error-gradient as per Lemma 2.1. This can be done in several ways, with a more advisable approach to initially compute the error-gradient's unit vector before computing its orthogonal unit vector for λ. Then, assuming the same neural network function as in section 3.1, the vector λ can be used in the weight regularization training law given by,

$$\Delta W = -\eta E\left[\nabla_W^T e\right]_P - \gamma(\lambda^T W) \tag{3.8}$$

where η is one of the learning rates in (3.2) and (3.3), and γ is the learning rate vector for weight regularization, which will be derived by the following theorem.

Theorem 3.3. *Given the cost function in* (2.6)*, the learning rate* γ *in* (3.8) *is given by,*

$$\gamma = E\left[\frac{\beta_0}{2\lambda^T\lambda}\right]_M \lambda \tag{3.9}$$

for some $\beta_0 \in [0, 1]$*, if stability in weight regularization is to be achieved.*

Proof. Given the cost function in (2.6), the change in the cost function is derived as,

$$\Delta J = (\lambda^T W)\lambda^T \Delta W \tag{3.10}$$

where ΔW is the desired weight update law for weight regularization. If ΔW is chosen as,

$$\Delta W = -\gamma_0 \lambda(\lambda^T W) \tag{3.11}$$

for some $\gamma_0 > 0$, the dynamics of the cost function in (2.6) becomes Lyapunov stable. The goal now is to estimate γ_0 such that the cost function (2.6) is exponentially stable,

i.e. $\Delta J = -\beta_0 J$ for some $\beta_0 \in [0,1]$. Substituting (3.11) in (3.10) and then placing the ΔJ in the cost function (2.6), the dynamics of the cost function is modeled as,

$$\begin{aligned} J(k+1) &= \frac{1}{2}\left(\lambda^T(k)W(k)\right)^2 - \gamma_0\lambda^T(k)\lambda(k)\left(\lambda^T(k)W(k)\right)^2 \\ &= \frac{1}{2}\left(1 - 2\gamma_0\lambda^T(k)\lambda(k)\right)\left(\lambda^T(k)W(k)\right)^2 \end{aligned} \tag{3.12}$$

From (3.12) it is clear that if $\gamma_0 = \frac{\beta_0}{2\lambda^T(k)\lambda(k)}$ for some $\beta_0 \in [0,1]$ but $\beta_0 \sim 0$, exponential stability is achieved. Then from the weight update laws in (3.8) and (3.11), the γ in (3.9) holds true if averaged over all M weights in W. □

Remark 3.4. Just as was done in section 3.1, the learning rates derived here are for the output layer weights W. If the weights w in $\phi(x,w)$ are to be included, then λ must be computed orthogonal to the vector containing all of the weights, W and w, with the learning rate γ_0 in (3.11) averaged over the entire weight cardinality. In general and for obvious reasons, any weights in w can be included for regularization except for the bias weights. Also to be noted, the β_0 value is said to be chosen close to zero, and not one. This is highly recommended if the fundamental objective of learning (i.e. the error minimization) is not to be compromised. Higher values closer to one are encouraged only if the governing weight space allows it.

From (3.8) and (3.9), it can note that the error-gradient and its orthogonal vector λ both expectantly play a part in the weight updates when regularization is performed. However, there isn't any condition given yet that can guarantees the minimization of weights by the choice of λ. The following theorem provides this condition on λ so that weight minimization is guaranteed.

Theorem 3.5. *For the weight update equation in* (3.8) *to ensure guaranteed weight minimization, the condition on* λ *is such that the regularization signal* $\lambda^T W$ *must always be negative, i.e.,*

$$\lambda^T W < 0 \tag{3.13}$$

Proof. If $-\mu W$ is a vector scaled in the direction of weight minimization for some $\mu > 0$, then the vector λ selected for the weight update must be positively correlated this vector $-\mu W$ so that weight minimization is guaranteed. In other words,

$$E\left[-\mu W^T\lambda\right] > 0$$

$$\Rightarrow \quad -W^T\lambda > 0$$

$$\text{or} \quad \lambda^T W < 0$$

□

4. THE REGULARIZATION ALGORITHM ON 2D WEIGHT SPACE EXAMPLES

Before the regularization algorithm can be tested on neural network examples, this section provides a visualization on its performance on 2D weight space examples. Two examples are shown in this section with weight solution spaces having the following characteristics: 1) a linear solution space with a single global optimum, and 2) a nonlinear solution space with multiple local optimums and a single global optimum. As will be shown in the examples, the regularization algorithm always drives the weights to the local or global optimums in the solution spaces, which when used without does not.

4.1. **THE LINEAR EXAMPLE.** In this example, the weight space W has a linear solution space W^* dictated by,

$$0.5W_1 - W_2 = 1 \tag{4.1}$$

The solution space is simple, and therefore a Lagrangian can be formulated and solved according to (2.2) and (2.3). Doing so, the optimal weights are,

$$W_1^* = 0.4,\ W_2^* = -0.8$$

which is the global optimum. Figure 1(a) shows the linear weight solution space of (4.1) along with its global optimum point $(0.4, -0.8)$. The circle in the figure confirms that the point $(0.4, -0.8)$ is in fact the global optimum weight solution, and that no other point in the solution space has a norm less than it.

Using the algorithm developed in section 3, the orthogonal vector λ is computed as $[2\ 1]^T$ or $[-2\ \ -1]^T$, which are both orthogonal to the function-gradient vector $[0.5\ -1]^T$ (Note: In this case, the function-gradient and the error-gradient follow the same direction). Then applying theorems 3.1 to 3.3, the regularized weight solution

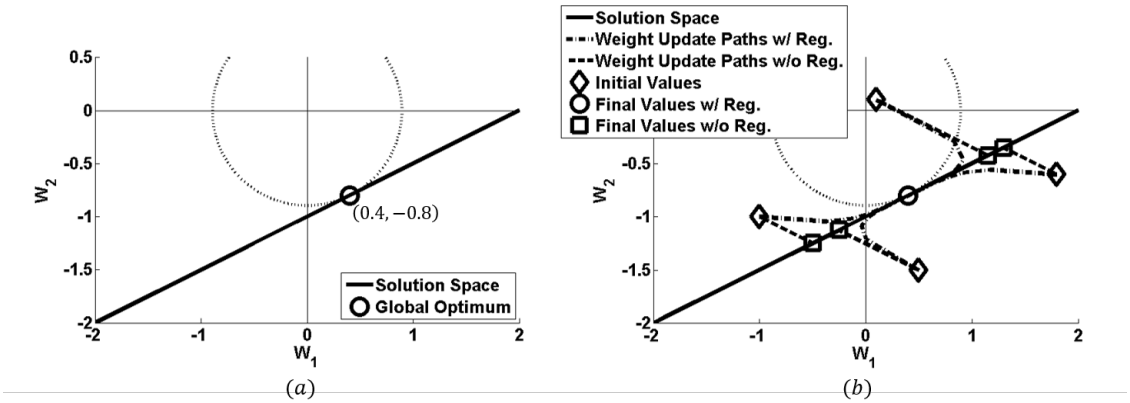


FIGURE 1. The linear weight solution space example of 4.1 with: (a) the mark of the global optimum; (b) the weight update trajectories.

always reaches the global optimum as shown in Figure 1(b). In this example, the value of β_0 is selected to be ~ 1, because it encourages faster regularization to the global optimum. This value was in fact increased manually from zero, until the probability of reaching the global optimum became equivalent to one. Otherwise, the regularization algorithm always reached a weight solution with a norm lesser than the solution obtained without regularization, but not the global optimum.

4.2. **THE NONLINEAR EXAMPLE.** Applying the algorithm on a more complex 2D weight space environment, the weight space W in this example has a nonlinear solution space W^* dictated by,

$$-(W_1-3)^4+(W_2-1)^4-10\,(W_2-1)^2=-16 \tag{4.2}$$

The solution space is of the order of four, and therefore if the Lagrangian is formulated according to (2.2), the solution equations formed according to (2.3) are to the order of three and four. Effectively, one can atleast expect three optimums: two local optimums and one global optimum, in this solution space. The solution equations obtained according to (2.3) are in fact painful to solve by hand and usually not encouraged to do so. Instead, these equations are usually solved by state-of-the-art math solvers that employ gradient descent or conjugate gradient descent methods.

Since the algorithm presented in this paper is gradient descent based, it will be directly used to obtain the optimal weight solution. Figures 2(a) and 2(b) show the solution space of (4.2) along with the trajectories of the weight updates, with and without weight regularization. In the figure, it can be noted that there exists three

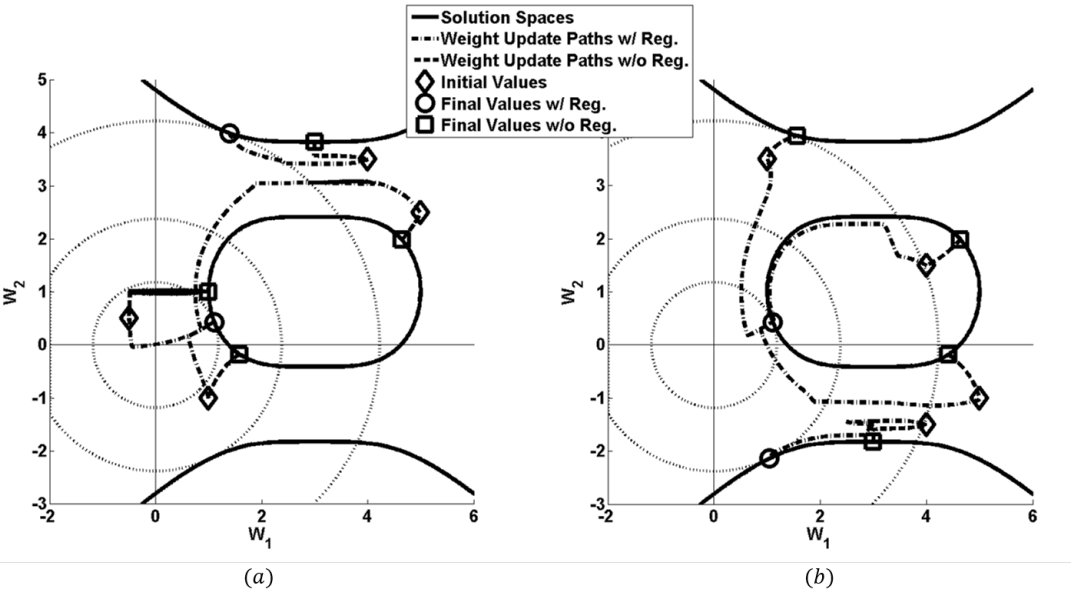


FIGURE 2. The nonlinear weight solution space example of 4.2 with weight update trajectories.

separate solution spaces: two local optimums and one global optimum, each having a global suboptimal weight solution of its own. Then depending on the initial location of the weights in the weight space, the respective global suboptimal weight solutions are always obtained by the regularization algorithm. Without regularization, the algorithm reaches a solution but not the global suboptimals. In this example, the β_0 is selected to be a value ~ 0 as recommended.

5. THE REGULARIZATION ALGORITHM ON NEURAL NETWORK EXAMPLES

The previous section demonstrated the performance of the presented regularization algorithm on 2D weight space examples, where the weight update trajectories governed by the regularization algorithm always reached the optimal weight solutions. In this section, the performance of the regularization algorithm is shown on neural network modeling and classification examples: for modeling, a highly nonlinear function, and for classification, the two-spiral problem. In both examples, a two-hidden-layer neural network with the configuration of 100-200, and weights initialized between -2 and $+2$ is used. This specific configuration was selected because the network, on an average, showed overfitting problems while being capable of meeting the training objective of a mean-square-error of 10^{-4}. In the following subsections, the performance characteristics of the regularization algorithm in mitigating the overfitting problem for this neural network is shown.

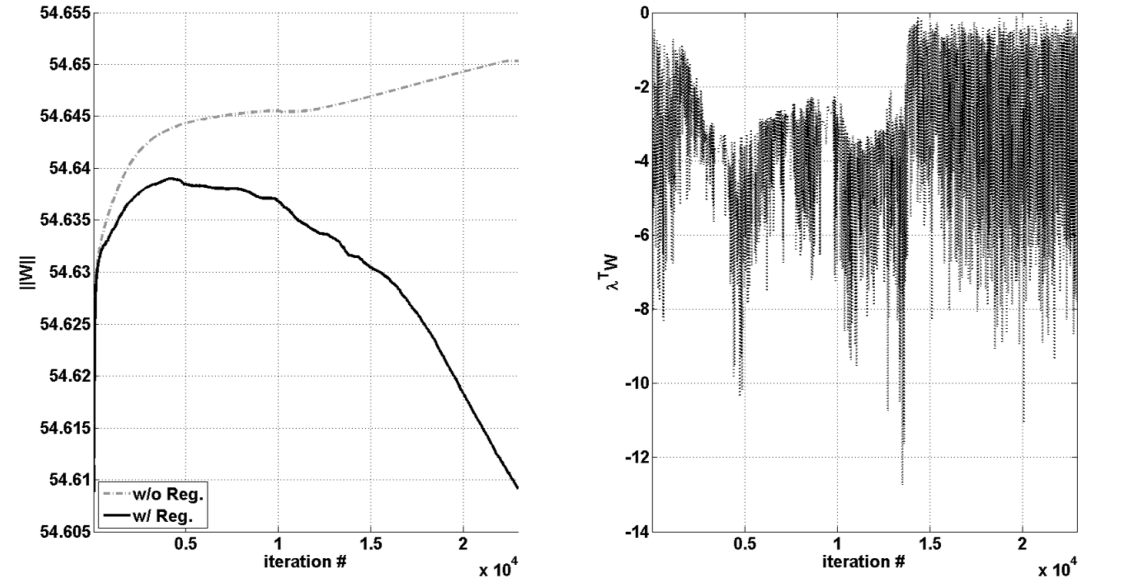
5.1. THE MODELING EXAMPLE.

In this example, the neural network is trained on a nonlinear function given by,

$$f(x) = \frac{1 + \cos x + 0.5x \sin 2x - 0.7x^2 \cos 3x}{3 + \cos x}, \quad x \in [-2\pi, 2\pi] \tag{5.1}$$

with training data sampled in intervals of $\frac{\pi}{20}$. Upon several trials without regularization, the neural network always showed overfitting problems while meeting the training objective of a MSE of 10^{-4}. However, when trained with regularization, the network achieved comparatively lesser overfitting, because of the weight minimization. Figure 3 shows a sample of the weight minimization process that mitigated the overfitting. As shown in Figure 3(b), the regularization signal $\lambda^T W$ is always maintained below zero as per theorem 3.3.

In this training problem, β_0 was selected to be ~ 1 because the weight space allowed it. But even with such a high value, the algorithm only managed to lower the norm of the weights and not obtain the optimum. This is evident in Figure 3(b) where the regularization signal did not reach zero, indicating that the solution obtained at the end of the training objective was not the optimum. This is because the training



(*a*) (*b*)

FIGURE 3. Sample performance of the regularization algorithm on the neural network modeling example (5.1): a) the weight norms; b) the regularization signal, $\lambda^T W$.

was stopped at a MSE of 10^{-4}. If the MSE was set to an order lesser than -4, the algorithm would have reached an actual local optimum as described in Section 2.

5.2. **THE CLASSIFICATION EXAMPLE.** In this example, the neural network is trained on the two-spiral dataset. This dataset consists of two spirally interlocked classes each having 97 data points. When trained without regularization, the neural network showed overfitting problems, which was mitigated to some extent when trained with regularization instead. The mitigation effectively improved the neural network's classification capabilities when tested on data points that lied in between the training data points. This improvement is summarized in Table 1. In this training problem, β_0 was selected to be ~ 1 again because the weight space allowed it. The training objective met was also a MSE of 10^{-4}. Also, just as in the modeling example, the regularization signal was always maintained below zero.

	Test Error Rate (%)
100-200 neural network: w/o Reg.	3.8083 $\pm$1.606
100-200 neural network: w/ Reg.	**3.2124 $\pm$1.1719**

TABLE 1. Improvements by regularization on the two spiral problem: averaged over 20 trials.

6. CONCLUSION

In this paper, a new gradient descent based weight regularization algorithm with Lyapunov based adaptive learning rates is presented. The algorithm is designed based on a new criterion that allows for online training by gradient descent. This particular attribute of the algorithm makes it useful for very large datasets applications such as computer vision. However, the paper only demonstrated the algorithm's performance on 2D weight space and simple neural network examples, as a means to provide clear visualizations of the key ideas introduced in this paper. For future work, the presented regularization algorithm can be tried on any adaptive parameter problems.

REFERENCES

[1] Y. LeCun, and Y. Bengio, *Convolutional Networks for Images, Speech, and Time Series*, The Handbook of Brain Theory and Neural Networks, MIT Press, 1995.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, Advances in Neural Information Processing Systems, 25 (2012).

[3] C. Clark, and A. Storkey, *Teaching Deep Convolutional Neural Networks to Play Go*, arXiv preprint arXiv:1412.3409, 2014.

[4] Y. Lecun, Y. Bengio, and G.E. Hinton, *Deep Learning*, Nature, 521 (2015) 436–444.

[5] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin *A Neural Probabilistic Language Model*, Journal of Machine Learning Research, 3 (2003) 1137–1155.

[6] F. Morin, and Y. Bengio, *Hierarchical Probabilistic Neural Network Language Model*, International Workshop on Artificial Intelligence ans Statisitcs, (2005) 246–252.

[7] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, *Natural Language Processing (Almost) from Scratch*, Journal of Machine Learning Research, 12 (2011) 2493–2537.

[8] H. Wang, N. Wang, and D-Y Yeung, *Collaborative Deep Learning for Recommender Systems*, arXiv preprint arXiv:1409.2944, 2014.

[9] D. V. Prokhorov, and D. C. Wunsch, *Adaptive Critic Designs*, IEEE Transactions on Neural Networks, 8:5 (1997) 997–1007.

[10] D. P. Mandic, and J. A. Chambers, *Towards the Optimal Learning Rate for Backpropagation*, Neural Processing Letter, 11 (2000) 1–5.

[11] L. Behera, S. Kumar, and A. Patnaik, *On Adaptive Learning Rate that Guarantees Convergence in Feedforward Networks*, IEEE Transactions on Neural Networks, 17:5 (2006) 1116–1125.

[12] M. B. Nasr, and M. Chtourou, *A Constructive based Hybrid Training Algorithm for Feedforward Neural Networks*, International Conference on Systems, Signals and Devices, (2009) 1–4.

[13] T. Kathirvalavakumar, and S. J. Subavathi, *Modified Backpropagation Algorithm with Adaptive Learning Rate based on Differential Errors and Differential Functional Constraints*, International Conference on Pattern Recognition, Informatics and Medical Engineering, (2012) 61–67.

[14] F. Damak, M. Chtourou, and M. B. Nasr, *Convergence of Hybrid Algorithm with Adaptive Learning Rate Parameter for Multilayer Neural Network*, World Congress on Computer and Information Technology, (2013) 1–4.

[15] T. Schaul, S. Zhang, and Y. LeCun, *No More Pesky Learning Rates*, International Conference on Machine Learning, (2013) 343–351.

[16] G. E. Hinton, *A Practical Guide to Training Restricted Boltzmann Machines*, Department of Computer Science, University of Toronto, 2010.

[17] J. Larsen, C. Svarer, L. N. Andersen, and L. K. Hansen, *Adaptive Regularization in Neural Network Modeling*, Neural Networks: Tricks of the Trade, Springer 1998.

[18] C. Goutte, and J. Larsen, *Adaptive Regularization of Neural Networks using Conjugate Gradient*, International Conference on Acoustics, Speech and Signal Processing, 2 (1998) 1201–1204.

[19] S. Rendle, *Learning Recommender Systems with Adaptive Regularization*, International Conference on Web Search and Data Mining, (2012) 133–142.

[20] K. Crammer, A. Kulesza and M. Drezde, *Adaptive Regularization of Weight Vectors*, Neural Information Processing Systems, 22 (2009).

[21] K. Crammer, and G. Chechik, *Adaptive Regularization for Weight Matrices*, International Conference on Machine Learning, (2012) 425–432.

[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Journal of Machine Learning Research, 15 (2014) 1929–1958.

[23] P. J. Werbos, *The Roots of Backpropagation. From Ordered Derivatives to Neural Networks and Political Forecasting*, Wiley, New York, 1994.

[24] S. Jagannathan, and F. L. Lewis, *Discrete-Time Neural Net Controller for a Class of Nonlinear Dynamical Systems*, IEEE Transactions on Automatic Control, 41:11 (1996) 1693–1699.