Role of Lipschitz constant in Gradient Learning

Snehanshu Saha
PES University
Bengaluru, Karnataka
snehanshusaha@pes.edu

Rahul Yedida
PES University, Electronic City Campus
Bengaluru, Karnataka
y.rahul@outlook.com

ABSTRACT

Finding an optimal learning rate in gradient descent has typically been an empirical process. We provide elegant mathematical proofs to compute an optimal value for the learning rate by exploiting functional properties of the loss function. By making minimal assumptions about the functions and possibly using the most simple architecture, we argue that the inverse of the Lipschitz constant is this optimal value. Experimental results show that this is indeed the case.

CCS CONCEPTS

• **Mathematics of computing** → **Functional analysis**; *Convex optimization*; Continuous functions; • **Computing methodologies** → *Batch learning*.

KEYWORDS

gradient descent, Lipschitz constant, regression, classification, learning rate, machine learning

ACM Reference Format:
Snehanshu Saha and Rahul Yedida. 2018. Role of Lipschitz constant in Gradient Learning. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Gradient descent[3] is a popular optimization algorithm for finding optima for functions, and is used to find optima in loss functions in machine learning tasks. In an iterative process, it seeks to update randomly initialized weights to minimize the training error. These updates are typically small values proportional to the gradient of the loss function. The constant of proportionality is called the learning rate, and is usually manually chosen.

The gradient descent update rule is given by

$$\mathbf{w} := \mathbf{w} - \alpha \cdot \nabla_{\mathbf{w}} f$$

where f is the loss function. When the learning rate, α, is too small, then convergence takes a long time. However, when the learning rate is too large, the solution diverges.

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

For a function, the Lipschitz constant is the least positive constant L such that

$$\|f(\mathbf{w}_1) - f(\mathbf{w}_2)\| \leq L \|\mathbf{w}_1 - \mathbf{w}_2\|$$

for all $\mathbf{w}_1, \mathbf{w}_2$ in the domain of f. From the mean-value theorem for scalar fields, for any $\mathbf{w}_1, \mathbf{w}_2$, there exists $\mathbf{v}$ such that

$$\begin{aligned}\|f(\mathbf{w}_1) - f(\mathbf{w}_2)\| &= \|\nabla_{\mathbf{w}} f(\mathbf{v})\| \, \|\mathbf{w}_1 - \mathbf{w}_2\| \\ &\leq \sup_{\mathbf{v}} \|\nabla_{\mathbf{w}} f(\mathbf{v})\| \, \|\mathbf{w}_1 - \mathbf{w}_2\|\end{aligned}$$

Thus, $\sup_{\mathbf{v}} \|\nabla_{\mathbf{w}} f(\mathbf{v})\|$ is such an L. Since L is the least such constant,

$$L \leq \sup_{\mathbf{v}} \|\nabla_{\mathbf{w}} f(\mathbf{v})\|$$

In this paper, we use $\max |\nabla_{\mathbf{w}} f|$ to derive the Lipschitz constants. Our approach makes the minimal assumption that the functions are Lipschitz continuous and differentiable up to first order only [1]. Because the gradient of these loss functions is used in gradient descent, these conditions are guaranteed to be satisfied.

By setting $\alpha = \frac{1}{L}$, we have $\Delta \mathbf{w} \leq 1$, constraining the change in the weights. This makes it optimal to set the learning rate to the reciprocal of the Lipschitz constant. The following sections show the derivation for the Lipschitz constants for various loss functions. The derived constants are in terms of the data, which is a constant with respect to the weight vector.

The rest of the paper is structured as follows. Section 2 discusses related work in gradient descent and its variants. Sections 3 to 5 show the derivation of the Lipschitz constants for some commonly used loss functions. Section 6 discusses the effect of regularization terms on the Lipschitz constant. Finally, Section 7 shows experiments on public datasets empirically demonstrating the efficacy of our method.

2 RELATED WORK

The generalization ability of stochastic gradient descent (SGD) and various methods of faster optimization have quickly gained interest in machine learning and deep learning communities.

Several directions have been taken to understand these phenomena. The interest in the stability of SGD is one such direction[12][9]. Others have proven that gradient descent can find the global minima of the loss functions in over-parameterized deep neural networks[22][4].

More practical approaches in this regard have involved novel changes to the optimization procedure itself. These include adding a "momentum" term to the update rule [18], and "adaptive gradient" methods such as RMSProp[19], and Adam[11], which combines RMSProp and AdaGrad[5]. These methods have seen widespread use in deep neural networks[16][21][1], Other methods rely on an

[1]Note this is a weaker condition than assuming the gradient of the function being Lipschitz continuous. We exploit merely the boundedness of the gradient.

approximation of the Hessian. These include the Broyden-Fletcher-Goldfarb-Shanno (BFGS) [2][7][8][17] and L-BFGS[13] algorithms. However, our proposed method does not require any modification of the standard gradient descent update rule, and relies on a fixed value of the learning rate. Further, we only use the first gradient, thus requiring functions to be only once differentiable and L-Lipschitz.

3 LEAST SQUARES COST FUNCTION

We have,

$$g(\mathbf{w}) = \frac{1}{2m}\sum_{i=1}^{m}\left(\mathbf{x}^{(i)}\mathbf{w} - y^{(i)}\right)^2$$

Thus,

$$\begin{aligned} g(\mathbf{w}) - g(\mathbf{v}) &= \frac{1}{2m}\sum_{i=1}^{m}\left(\mathbf{x}^{(i)}\mathbf{w} - y^{(i)}\right)^2 - \left(\mathbf{x}^{(i)}\mathbf{v} - y^{(i)}\right)^2 \\ &= \frac{1}{2m}\sum_{i=1}^{m}\left(\mathbf{x}^{(i)}(\mathbf{w}+\mathbf{v}) - 2y^{(i)}\right)\left(\mathbf{x}^{(i)}(\mathbf{w}-\mathbf{v})\right) \\ &= \frac{1}{2m}\sum_{i=1}^{m}\left((\mathbf{w}+\mathbf{v})^T\mathbf{x}^{(i)T} - 2y^{(i)}\right)\left(\mathbf{x}^{(i)}(\mathbf{w}-\mathbf{v})\right) \\ &= \frac{1}{2m}\sum_{i=1}^{m}\left((\mathbf{w}+\mathbf{v})^T\mathbf{x}^{(i)T}\mathbf{x}^{(i)} - 2y^{(i)}\mathbf{x}^{(i)}\right)(\mathbf{w}-\mathbf{v}) \end{aligned}$$

The penultimate step is obtained by observing that $(\mathbf{w}+\mathbf{v})^T\mathbf{x}^{(i)T}$ is a real number, whose transpose is itself.

At this point, we take the norm on both sides, and then assume that $\mathbf{w}$ and $\mathbf{v}$ are bounded such that $\|\mathbf{w}\|, \|\mathbf{v}\| \leq K$. Taking norm on both sides,

$$\boxed{\frac{\|g(\mathbf{w}) - g(\mathbf{v})\|}{\|\mathbf{w}-\mathbf{v}\|} \leq \frac{K}{m}\left\|\mathbf{X}^T\mathbf{X}\right\| - \frac{1}{m}\left\|\mathbf{y}^T\mathbf{X}\right\|}$$

We are forced to use separate norms because the matrix subtraction $2K\mathbf{X}^T\mathbf{X} - 2\mathbf{y}^T\mathbf{X}$ cannot be performed. The RHS here is the Lipschitz constant. Note that the Lipschitz constant changes if the cost function is considered with a factor other than $\frac{1}{2m}$.

3.1 An alternate derivation

The least-squares system can also be formulated as

$$\begin{aligned} L(\mathbf{w}) &= \|\mathbf{X}\mathbf{w} - \mathbf{y}\| \\ &= (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) \end{aligned}$$

Then, we have

$$\begin{aligned} L(\mathbf{b}) - L(\mathbf{a}) &= (X\mathbf{b} - Y)^T(X\mathbf{b} - Y) - (X\mathbf{a} - Y)^T(X\mathbf{a} - Y) \\ &= (\mathbf{b}^TX^T - Y^T)(X\mathbf{b} - Y) - (\mathbf{a}^TX^T - Y^T)(X\mathbf{a} - Y) \\ &= \mathbf{b}^TX^TX\mathbf{b} - \mathbf{b}^TX^TY - Y^TX\mathbf{b} + Y^TY - \mathbf{a}^TX^TX\mathbf{a} + \\ &\quad \mathbf{a}^TX^TY + Y^TX\mathbf{a} - Y^TY \\ &= \mathbf{b}^TX^TX\mathbf{b} - \mathbf{b}^TX^TY - Y^TX\mathbf{b} - \mathbf{a}^TX^TX\mathbf{a} + \\ &\quad \mathbf{a}^TX^TY + Y^TX\mathbf{a} \\ &= \mathbf{b}^TX^TX\mathbf{b} - 2Y^TX\mathbf{b} - \mathbf{a}^TX^TX\mathbf{a} + 2Y^TX\mathbf{a} \\ &= \mathbf{b}^TX^TX\mathbf{b} - \mathbf{a}^TX^TX\mathbf{a} - 2Y^TX(\mathbf{b} - \mathbf{a}) \\ &= (\mathbf{b} - \mathbf{a} + \mathbf{a})^TX^TX\mathbf{b} - \mathbf{a}^TX^TX\mathbf{a} - 2Y^TX(\mathbf{b} - \mathbf{a}) \\ &= (\mathbf{b} - \mathbf{a})^TX^TX\mathbf{b} + \mathbf{a}^TX^TX\mathbf{b} - \mathbf{a}^TX^TX\mathbf{a} \\ &\quad - 2Y^TX(\mathbf{b} - \mathbf{a}) \\ &= (\mathbf{b} - \mathbf{a})^TX^TX\mathbf{b} + \mathbf{a}^TX^TX(\mathbf{b} - \mathbf{a}) - 2Y^TX(\mathbf{b} - \mathbf{a}) \\ &= (\mathbf{b} - \mathbf{a})^TX^TX(\mathbf{b} - \mathbf{a} + \mathbf{a}) + \mathbf{a}^TX^TX(\mathbf{b} - \mathbf{a}) \\ &\quad - 2Y^TX(\mathbf{b} - \mathbf{a}) \\ &= (\mathbf{b} - \mathbf{a})^TX^TX\mathbf{a} + \mathbf{a}^TX^TX(\mathbf{b} - \mathbf{a}) + \\ &\quad (\mathbf{b} - \mathbf{a})^TX^TX(\mathbf{b} - \mathbf{a}) - 2Y^TX(\mathbf{b} - \mathbf{a}) \\ &= \mathbf{a}^TX^TX(\mathbf{b} - \mathbf{a}) + \mathbf{a}^TX^TX(\mathbf{b} - \mathbf{a}) + \\ &\quad (\mathbf{b} - \mathbf{a})^TX^TX(\mathbf{b} - \mathbf{a}) - 2Y^TX(\mathbf{b} - \mathbf{a}) \\ &= 2\mathbf{a}^TX^TX(\mathbf{b} - \mathbf{a}) + (\mathbf{b} - \mathbf{a})^TX^TX(\mathbf{b} - \\ &\quad \mathbf{a}) - 2Y^TX(\mathbf{b} - \mathbf{a}) \end{aligned}$$

The triangle inequality now gives us,

$$\frac{\|L(\mathbf{b}) - L(\mathbf{a})\|}{\|\mathbf{b} - \mathbf{a}\|} \leq 2\|\mathbf{a}\|\left\|X^TX\right\| - 2\left\|Y^TX\right\| + \|\mathbf{b} - \mathbf{a}\|\left\|X^TX\right\|$$

Assuming $\|\mathbf{a}\|, \|\mathbf{b}\| \leq K$,

$$\frac{\|L(\mathbf{b}) - L(\mathbf{a})\|}{\|\mathbf{b} - \mathbf{a}\|} \leq 2K\left\|X^TX\right\| - 2\left\|Y^TX\right\|$$

and so we can set the Lipschitz constant to

$$\boxed{L = 2K\left\|X^TX\right\| - 2\left\|Y^TX\right\|}$$

4 BINARY CROSS-ENTROPY FUNCTION

We have,

$$g(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} -y^{(i)}\log h(x^{(i)}) - (1 - y^{(i)})\log\left(1 - h(x^{(i)})\right)$$

where $h(x^{(i)}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}$.

We have,

$$\begin{aligned} \frac{\partial h(\mathbf{x}^{(i)})}{\partial \mathbf{w}_j} &= -\frac{e^{-\mathbf{w}^T\mathbf{x}^{(i)}}\left(-x_j^{(i)}\right)}{1 + e^{-\mathbf{w}^T\mathbf{x}^{(i)}}} \\ &= \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}^{(i)}}} \cdot \left(1 - \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}\right) \cdot x_j^{(i)} \\ &= h(\mathbf{x}^{(i)})\left(1 - h(\mathbf{x}^{(i)})\right)x_j^{(i)} \end{aligned}$$

The gradient is then,

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{w}_j} g(\mathbf{w}) &= \frac{\partial}{\partial \mathbf{w}_j}\left(\frac{-1}{m}\sum_{i=1}^{m} y^{(i)} \log\left(h(\mathbf{x}^{(i)})\right) + (1-y^{(i)})\log\left(1-h(\mathbf{x}^{(i)})\right)\right) \\
&= \frac{-1}{m}\sum_{i=1}^{m}\left(\frac{y^{(i)}}{h(\mathbf{x}^{(i)})}h'(\mathbf{x}^{(i)}) + \frac{1-y^{(i)}}{1-h(\mathbf{x}^{(i)})}(-h'(\mathbf{x}^{(i)}))\right) \\
&= \frac{-1}{m}\sum_{i=1}^{m}\left(y^{(i)}(1-h(\mathbf{x}^{(i)}))(\mathbf{x}_j^{(i)}) - (1-y^{(i)})h(\mathbf{x}^{(i)})(x_j^{(i)})\right) \\
&= \frac{-1}{m}\sum_{i=1}^{m}\left(y^{(i)}(1-h(\mathbf{x}^{(i)})) - (1-y^{(i)})h(\mathbf{x}^{(i)})\right)x_j^{(i)} \\
&= \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - y^{(i)}h(\mathbf{x}^{(i)}) - h(\mathbf{x}^{(i)}) + y^{(i)}h(\mathbf{x}^{(i)})\right)x_j^{(i)} \\
&= \frac{1}{m}\sum_{i=1}^{m}\left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)x_j^{(i)}
\end{aligned}$$

It is trivial to show that the maximum of this occurs when $h(\mathbf{x}^{(i)}) = \frac{1}{2}$: for any k,

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{w}_k}\left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)x_j^{(i)} &= \frac{\partial}{\partial \mathbf{w}_k}h(\mathbf{x}^{(i)})x_j^{(i)} \\
&= h(\mathbf{x}^{(i)})\left(1-h(\mathbf{x}^{(i)})\right)x_k^{(i)}x_j^{(i)}
\end{aligned}$$

Within the domain $(-\infty, \infty)$, the first two terms cannot be 0; the last term is a constant, leaving $x_k^{(i)} = 0$, and thus $h(\mathbf{x}^{(i)}) = \frac{1}{2}$. Now whether $y^{(i)}$ is 0 or 1, $|h(\mathbf{x}^{(i)}) - y^{(i)}| = \frac{1}{2}$.

Thus, the Lipschitz constant is

$$\boxed{L = \frac{1}{2m}\|\mathbf{X}\|}$$

5 CROSS-ENTROPY LOSS FUNCTION

The cross-entropy loss (or softmax loss) is defined as[14] [2]

$$g(\mathbf{w}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k}[y^{(i)} = j]\log\left(\frac{e^{\Theta_j^T\mathbf{x}^{(i)}}}{\sum_{l=1}^{k}e^{\Theta_l^T\mathbf{x}^{(i)}}}\right)$$

Here, we are using the Iverson notation: the expression $[y^{(i)} = j]$ is 1 if the condition is true, and 0 otherwise. To find the Lipschitz constant, we require the derivative of the softmax function, which is the fraction in the log term.

$$\begin{aligned}
\frac{\partial s_j}{\partial \Theta_p} &= \frac{\partial}{\partial \Theta_p^T\mathbf{x}^{(i)}}\left(\frac{e^{\Theta_j^T\mathbf{x}^{(i)}}}{\sum_{l=1}^{k}e^{\Theta_l^T\mathbf{x}^{(i)}}}\right)\cdot\frac{\partial}{\partial \Theta_p}\left(\Theta_p^T\mathbf{x}^{(i)}\right) \\
&= \frac{[p=j]e^{\Theta_j^T\mathbf{x}^{(i)}}\sum_{l=1}^{k}e^{\Theta_l^T\mathbf{x}^{(i)}} - e^{\Theta_j^T\mathbf{x}^{(i)}}\cdot e^{\Theta_p^T\mathbf{x}^{(i)}}}{\left(\sum_{l=1}^{k}e^{\Theta_l^T\mathbf{x}^{(i)}}\right)^2}\cdot\mathbf{x}^{(i)} \\
&= \left(\frac{[p=j]e^{\Theta_j^T\mathbf{x}^{(i)}}}{\sum_{l=1}^{k}e^{\Theta_l^T\mathbf{x}^{(i)}}} - \frac{e^{\Theta_j^T\mathbf{x}^{(i)}}}{\sum_{l=1}^{k}e^{\Theta_l^T\mathbf{x}^{(i)}}}\cdot\frac{e^{\Theta_p^T\mathbf{x}^{(i)}}}{\sum_{l=1}^{k}e^{\Theta_l^T\mathbf{x}^{(i)}}}\right)\cdot\mathbf{x}^{(i)} \\
&= \left([p=j]s_j - s_j s_p\right)\cdot\mathbf{x}^{(i)} \\
&= s_j([p=j] - s_p)\mathbf{x}^{(i)}
\end{aligned}$$

We use the notation s_j to denote the softmax function. The first step uses the chain rule for derivatives. The second step is obtained by applying the quotient rule and noting that the partial derivative is taken with respect to $\Theta_p^T\mathbf{x}^{(i)}$. We then simplify, split the fraction into two, and cancel out terms. In a more concise fashion, we write the above result as:

$$\frac{\partial s_j}{\partial \Theta_p} = s_j([p=j] - s_p)\mathbf{x}^{(i)} \tag{1}$$

We can now use (1) to find the gradient of g.

$$\begin{aligned}
\nabla_{\Theta_p} g &= -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k}\nabla_{\Theta_p}[y^{(i)} = j]\log s_j \\
&= -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k}\frac{[y^{(i)}=j]}{s_j}\nabla_{\Theta_p}s_j \\
&= -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=p}[y^{(i)}=j](1-s_p)\mathbf{x}^{(i)} - \\
&\quad \frac{1}{m}\sum_{i=1}^{m}\sum_{j\neq p}[y^{(i)}=j](-s_p)\mathbf{x}^{(i)} \\
&= -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=p}[y^{(i)}=j]\mathbf{x}^{(i)} + \frac{1}{m}\sum_{i=1}^{m}\sum_{j=p}[y^{(i)}=j]s_p\mathbf{x}^{(i)} + \\
&\quad \frac{1}{m}\sum_{i=1}^{m}\sum_{j\neq p}[y^{(i)}=j]s_p\mathbf{x}^{(i)} \\
&= -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}=p]\mathbf{x}^{(i)} + \frac{1}{m}\sum_{i=1}^{m}[y^{(i)}=p]s_p\mathbf{x}^{(i)} + \\
&\quad \frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\neq p]s_p\mathbf{x}^{(i)} \\
&= -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}=p]\mathbf{x}^{(i)} + \\
&\quad \frac{1}{m}\sum_{i=1}^{m}s_p\mathbf{x}^{(i)}\left([y^{(i)}=p] + [y^{(i)}\neq p]\right) \\
&= -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}=p]\mathbf{x}^{(i)} + \frac{1}{m}\sum_{i=1}^{m}s_p\mathbf{x}^{(i)}
\end{aligned}$$

[2] This version of the softmax loss assumes the labels are distributed according to a Multinomial distribution. This is equivalent to assuming a one-hot encoded target and using $\sum_i\sum_j y_j^{(i)}\log s_j$, because the effect of the latter is to only consider the "hot" softmax value, while the former achieves that using the Iverson notation instead of explicitly one-hot encoding the target.

Thus,

$$\nabla_{\Theta_p} g = \frac{1}{m} \sum_{i=1}^{m} \mathbf{x}^{(i)} \left(s_p - [y^{(i)} = p] \right) \tag{2}$$

In the limiting case, all the softmax values are equal, and $y^{(i)} = p$; for k classes, $s_p = \frac{1}{k}$, yielding $|s_p - 1| = \frac{k-1}{k}$ so we get the inequality

$$|[y^{(i)} = p](s_p - 1)| \leq \frac{k-1}{k} \tag{3}$$

Using (3) in (2), we get

$$|\nabla_{\Theta_p} g| \leq \frac{k-1}{km} \sum_{i=1}^{m} \mathbf{x}^{(i)}$$

and thus we obtain the Lipschitz constant for the softmax loss function:

$$\boxed{L = \frac{k-1}{km} \|\mathbf{X}\|}$$

6 A NOTE ON REGULARIZATION

It should be noted that this framework is extensible to the case where the loss function includes a regularization term.

In particular, if an L_2 regularization term, $\frac{\lambda}{2} \|\mathbf{w}\|_2^2$ is added, it is trivial to show that the Lipschitz constant increases by λK, where K is the upper bound for $\|\mathbf{w}\|$. More generally, if a Tikhonov regularization term, $\|\Gamma \mathbf{w}\|_2^2$ term is added, then the increase in the Lipschitz constant can be computed in the same way as shown in the alternate derivation for the least-squares loss above. For the sake of continuity, we show a succinct version of the derivation below:

$$\begin{aligned} L(\mathbf{w}_1) - L(\mathbf{w}_2) &= (\Gamma \mathbf{w}_1)^T (\Gamma \mathbf{w}_1) - (\Gamma \mathbf{w}_2)^T (\Gamma \mathbf{w}_2) \\ &= \mathbf{w}_1^T \Gamma^2 \mathbf{w}_1 - \mathbf{w}_2^T \Gamma^2 \mathbf{w}_2 \\ &= 2\mathbf{w}_2^T \Gamma^2 (\mathbf{w}_1 - \mathbf{w}_2) + (\mathbf{w}_1 - \mathbf{w}_2)^T \Gamma^2 (\mathbf{w}_1 - \mathbf{w}_2) \\ \frac{\|L(\mathbf{w}_1) - L(\mathbf{w}_2)\|}{\|\mathbf{w}_1 - \mathbf{w}_2\|} &\leq 2 \|\mathbf{w}_2\| \left\|\Gamma^2\right\| + \|\mathbf{w}_1 - \mathbf{w}_2\| \left\|\Gamma^2\right\| \end{aligned}$$

Now by the assumption that $\mathbf{w}_1, \mathbf{w}_2$ are bounded by K,

$$\boxed{L = 2K \left\|\Gamma^2\right\|}$$

This additional term may be added to the Lipschitz constants derived above when gradient descent is performed on a loss function including a Tikhonov regularization term. Clearly, for an L_2-regularizer, since $\Gamma = \frac{\lambda}{2}\mathbf{I}$, $L = \lambda K$. For example, if an L_2 regularization term is included in the least-squares cost function, the Lipschitz constant now becomes $L = \frac{K}{m} \left\|X^T X\right\| - \frac{1}{m} \left\|Y^T X\right\| + \lambda K$

7 EXPERIMENTS

This section discusses experiments demonstrating the faster convergence with our approach.

In each experiment, we randomly initialize weights, and use the same initial weight vector for gradient descent with both the learning rates. In all experiments, we scale each feature to sum to 1 before running gradient descent. This scaled data is used to compute the Lipschitz constants, and consequently, the learning rates.

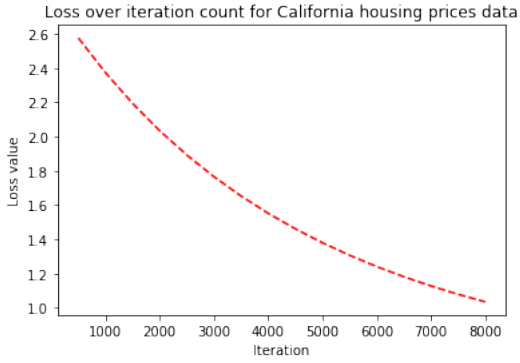

Figure 1: Loss function over iterations for California housing prices dataset

Normalizing the data is particularly important because the Lipschitz constant may get arbitrarily large, thus making the learning rate too small.

Note that the results below are obtained from simple, computationally inexpensive models. The regression experiments use a multiple linear regression model, the binary classification experiments use an ordinary logistic regression model, and the multi-class classification experiments use a softmax regression model with one-hot encoded target labels. For MNIST, however, we found it quicker to train a neural network with only an input and output layer (no hidden layers were used), a stochastic gradient descent optimizer, and softmax activations.

7.1 Regression experiments

Rather than wait for absolute convergence, which may take many iterations, we instead choose to set a threshold for the value of the loss function. When the value of the cost function goes below this threshold, we stop the gradient descent procedure. A reasonable threshold value is chosen for each dataset separately.

For the least-squares cost function, an estimate of K is required. A good estimate of K would be obtained by running gradient descent with some fixed learning rate and then taking the norm of the final weight vectors. However, because this requires actually running the algorithm for which we want to find a parameter first, we need to estimate this value instead. In our experiments, we obtain a close approximation to the value obtained above through the formula below. For the experiments in this subsection, we use this formula to compute K.

$$a = \frac{1}{m} \sum_{j=1}^{n} \sum_{i=1}^{m} x_j^{(i)}$$

$$b = \frac{1}{n} \sum_{j=1}^{n} \max_i x_j^{(i)}$$

$$K = \frac{a+b}{2}$$

In the above formulas, the notation $x_j^{(i)}$ refers to the jth column of the ith training example. Note that a is the sum of the means of each column, and b is the mean of the maximum of each column.

Table 1 shows the results of our experiments on some datasets. Clearly, our choice of α outperforms a random guess in all the datasets. Our proposed method yields a learning rate that adapts to each dataset to converge significantly faster than with a guess. In some datasets, our choice of learning rate gives over a 100x improvement in training time.

While the high learning rates may raise concerns of oscillations rather than convergence, we have checked for this in our experiments. To do this, we continued running gradient descent, monitoring the value of the loss function every 500 iterations. Figure 1 shows this plot, demonstrating that the high learning rates indeed lead to convergence.

7.2 Binary classification experiments

For classification, our experiments were conducted by running gradient descent for 1000 iterations, and then comparing accuracy scores over the two learning rates. Table 2 summarizes the results of these experiments. Note that even with high learning rates, the algorithm outperforms a small choice of learning rate. We also ran experiments in the same manner as we did for regression: we set a reasonable error threshold and compared the number of iterations required across both learning rates. Table 4 summarizes the results of these experiments.

For the iris dataset, we removed one class to make the problem a binary classification one. For the covertype data, we considered only the first two out of seven classes. This resulted in 495,141 rows. We also considered only ten features to speed up computation time.

7.3 Multi-class classification experiments

For multi-class classification, the experiments were conducted in the same manner as for binary classification. However, we did not scale the data in this case. Further, the target variable was one-hot encoded before running gradient descent. Finally, we ran gradient descent for 200 iterations, rather than 1000. Table 3 summarizes the results of these experiments.

7.3.1 MNIST. For MNIST, we handled the data differently. Because running gradient descent caused overflows even for learning rates of 10^{-4}, we standardized the data using sklearn[15] first. Further, we compared the training and validation accuracy scores across both the learning rates.

Figure 2 shows a comparative plot of the training and validation accuracy scores for both learning rates. In both the plots, the red line is for $\alpha = 0.1$, while the green line is for $\alpha = \frac{1}{L}$. Although our choice of learning rate starts off worse, it quickly (< 100 iterations) outperforms a learning rate of 0.1.

Fig 3 shows the training and validation accuracy scores for the same learning rates. In this figure, the y-axis is shared for easy comparison. With the higher learning rate of 10.24, our model shows a greater overfitting tendency. However, neither of these

[3]The learning rate obtained here is different because we restricted the data to the first 100K rows only.

Figure 2: Comparison of training and validation accuracy scores for different α on MNIST

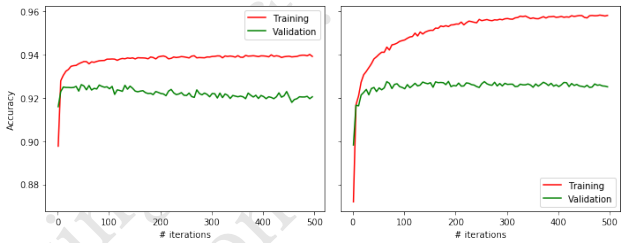

Figure 3: Comparison of training and validation accuracy scores for same α on MNIST

models used any regularization, and this tendency will reduce on including a regularization term in the loss function.

Finally, Table 5 shows the number of iterations required to reach certain accuracy thresholds with each learning rate.

8 CONCLUSION AND FUTURE WORK

Our major contribution in this paper is the presentation of a novel framework for automatically choosing a suitable learning rate in gradient descent. We then derived the formulas for some common loss functions. This is equally applicable, of course, to both batch and stochastic gradient descent. Our experiments confirmed that our choice of learning rate indeed leads to significantly faster convergence, as predicted by the theory.

Future work in this direction is two-fold. First, we need to investigate the utility of a similar approach in the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization algorithm, which uses an approximation of the Hessian matrix. In BFGS, a suitable step size is chosen via line search; however, this is computationally expensive. Identifying a more direct approach to this would be considerably beneficial.

Second, this work discussed, in section 7.1, a set of equations for computing an approximation to K. In our experiments, the approximation was quite close to the value of $\|\mathbf{w}\|$ obtained after running gradient descent. In applications where time and/or resources are limited, an approximation to the weight vector may be sufficient. Mathematically, the equation $\|\mathbf{w}\| = K$ represents an n-ball. Therefore, it may suffice to only search the surface of this n-ball, rather than the entire n-dimensional space via gradient

Table 1: Regression experiments on various datasets with $\alpha = 0.1$ and $\alpha = \frac{1}{L}$

Dataset	Loss threshold	Inverse Lipschitz constant	#it with $\alpha = 0.1$	#it with $\alpha = \frac{1}{L}$
Boston housing prices	200	9.316	46,041	555
California housing prices	2.8051	5163.5	24,582	2
Energy efficiency [20]	100	12.78	489,592	3,833
Online news popularity [6]	73,355,000	1.462	10,985	753

Table 2: Binary classification experiments on various datasets with $\alpha = 0.1$ and $\alpha = \frac{1}{L}$

Dataset	Inverse Lipschitz constant	Accuracy with $\alpha = 0.1$	Accuracy with $\alpha = \frac{1}{L}$
Iris	830.22	50%	100%
Covertype	189.35M	43.05%	57.21%
Breast cancer	4280.22	43.23%	90.5%

Table 3: Softmax classification experiments on various datasets with $\alpha = 0.1$ and $\alpha = \frac{1}{L}$

Dataset	Inverse Lipschitz constant	Accuracy with $\alpha = 0.1$	Accuracy with $\alpha = \frac{1}{L}$
Iris	1.936	93.33%	97.78%
Digits	0.635	91.3%	94.63%

Table 4: Classification experiments on various datasets with an error threshold

Dataset	Loss function	Loss threshold	Inverse Lipschitz constant	#it with $\alpha = 0.1$	#it with $\alpha = \frac{1}{L}$
Breast cancer	Binary cross-entropy	0.69	4280.23	37,008	2
Covertype[3]	Binary cross-entropy	0.69314	17.48M	216,412	2
Iris	Softmax	0.2	1.902	413	49
Digits	Softmax	0.2	0.634	337	2

Table 5: Iterations required to reach accuracy thresholds in MNIST experiments

Accuracy threshold	# it with $\alpha = 0.1$	# it with $\alpha = \frac{1}{L}$
92.5%	11	33
92.6%	22	58
92.7%	22	63
92.8%	–	173

descent. The optimization objective, therefore, is

$$\min_{\mathbf{w}} J(\mathbf{w})$$
$$\text{s.t. } \|\mathbf{w}\| = K$$

where J is some loss function. Although our constraint is non-convex, this is similar to the optimization problem in Support Vector Machines (SVMs), and future work could investigate possible approaches to solving this optimization problem.

Alternatively, the surface of the n-ball could be used to initialize weights–rather than initialize the weight vector randomly in the entire n-space, one could initialize the weights at a random point on the n-ball.

REFERENCES

[1] Parnia Bahar, Tamer Alkhouli, Jan-Thorsten Peter, Christopher Jan-Steffen Brix, and Hermann Ney. 2017. Empirical investigation of optimization algorithms in neural machine translation. *The Prague Bulletin of Mathematical Linguistics* 108, 1 (2017), 13–25.

[2] Charles G Broyden. 1970. The convergence of a class of double-rank minimization algorithms: 2. The new algorithm. *IMA journal of applied mathematics* 6, 3 (1970), 222–231.

[3] Augustin Cauchy. 1847. Méthode générale pour la résolution des systemes dâĂŹéquations simultanées. *Comp. Rend. Sci. Paris* 25, 1847 (1847), 536–538.

[4] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. 2018. Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054* (2018).
[5] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.
[6] Kelwin Fernandes, Pedro Vinagre, and Paulo Cortez. 2015. A proactive intelligent decision support system for predicting the popularity of online news. In *Portuguese Conference on Artificial Intelligence*. Springer, 535–546.
[7] Roger Fletcher. 1970. A new approach to variable metric algorithms. *The computer journal* 13, 3 (1970), 317–322.
[8] Donald Goldfarb. 1970. A family of variable-metric methods derived by variational means. *Mathematics of computation* 24, 109 (1970), 23–26.
[9] Moritz Hardt, Benjamin Recht, and Yoram Singer. 2015. Train faster, generalize better: Stability of stochastic gradient descent. *arXiv preprint arXiv:1509.01240* (2015).
[10] Arthur Juliani. 2016. *A simple ipython notebook that walks through the creation of a softmax regression model using MNIST dataset.* https://gist.github.com/awjuliani/5ce098b4b76244b7a9e3#file-softmax-ipynb
[11] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[12] Ilja Kuzborskij and Christoph H Lampert. 2017. Data-dependent stability of stochastic gradient descent. *arXiv preprint arXiv:1703.01678* (2017).
[13] Dong C Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Mathematical programming* 45, 1-3 (1989), 503–528.
[14] Andrew Ng. 2000. CS229 Lecture notes. *CS229 Lecture notes* 1, 1 (2000), 1–3.
[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[16] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).
[17] David F Shanno. 1970. Conditioning of quasi-Newton methods for function minimization. *Mathematics of computation* 24, 111 (1970), 647–656.
[18] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*. 1139–1147.
[19] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4, 2 (2012), 26–31.
[20] Athanasios Tsanas and Angeliki Xifara. 2012. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings* 49 (2012), 560–567.
[21] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*. 2048–2057.
[22] Difan Zou, Yuan Cao, Dongruo Zhou, and Quanquan Gu. 2018. Stochastic gradient descent optimizes over-parameterized deep ReLU networks. *arXiv preprint arXiv:1811.08888* (2018).

A ADDITIONAL EXPERIMENT RESULTS

Figure 4a and 4b show the precision, recall, and F1-scores for different learning rates on the digits data after 200 iterations. Clearly, our choice of learning rate shows an improvement across all classes. We noticed this trend across all the datasets that we ran our experiments on.

B CODE SNIPPETS

B.1 Regression experiments

Our regression experiments were performed with a slightly modified gradient descent function that ran until the specified cost was below a parameter `epsilon`. The function below runs an experiment.

```
def run_experiment(x, y, alpha=0.1, K='auto'
    , epsilon=100, print_cost=1000000):
    m = x.shape[0]
    n = x.shape[1] - 1
```

	precision	recall	f1-score	support
0	0.94	0.98	0.96	46
1	0.83	0.82	0.82	60
2	0.93	0.84	0.88	49
3	0.89	0.86	0.87	56
4	0.96	0.94	0.95	51
5	0.91	0.97	0.94	62
6	0.93	0.98	0.95	51
7	0.94	0.96	0.95	51
8	0.80	0.80	0.80	60
9	0.85	0.83	0.84	54
micro avg	0.89	0.89	0.89	540
macro avg	0.90	0.90	0.90	540
weighted avg	0.89	0.89	0.89	540

(a) $\alpha = 0.1$

	precision	recall	f1-score	support
0	0.98	1.00	0.99	46
1	0.93	0.90	0.92	60
2	0.96	0.98	0.97	49
3	0.96	0.91	0.94	56
4	1.00	0.96	0.98	51
5	0.95	0.95	0.95	62
6	0.93	1.00	0.96	51
7	0.89	1.00	0.94	51
8	0.95	0.88	0.91	60
9	0.94	0.93	0.93	54
micro avg	0.95	0.95	0.95	540
macro avg	0.95	0.95	0.95	540
weighted avg	0.95	0.95	0.95	540

(b) $\alpha = 1/L$

Figure 4: Precision, recall, and F1-scores for different learning rates on the digits data

```
    theta = np.random.randn(n + 1, 1)

    x_norm = np.sum(x, axis=0)
    x = x / x_norm

    theta_final, it = batch_gradient_descent
        (theta, x, y, alpha=alpha, epsilon=
        epsilon, print_cost=print_cost)
    print('Traditional:', it, 'iterations')

    if (K == 'auto'):
        K = np.linalg.norm(theta_final)

    L = K / m * np.linalg.norm(np.dot(x.T, x
        )) - 1 / m * np.linalg.norm(np.dot(y
        .T, x))
    a = np.abs(1 / L)
    print('Custom learning rate:', a)
    _, it = batch_gradient_descent(theta, x,
         y, alpha=a, epsilon=epsilon,
        print_cost=print_cost)
    print('Custom:', it, 'iterations')
```

B.2 Classification experiments

Our binary classification experiments were performed in a similar manner as the regression experiments, only changing the computation of the Lipschitz constant and the cost. The code for implementing softmax regression used a one-hot encoding and was taken from GitHub[10]. Below is the set of helper functions to run the code.

```
def getLoss(w,x,y,lam):
    m = x.shape[0]  # number of training
        examples
    y_mat = oneHotIt(y)  # convert into a
        one-hot representation
    scores = np.dot(x,w)
    prob = softmax(scores)
    loss = (-1 / m) * np.sum(y_mat * np.log(
        prob)) + (lam/2)*np.sum(w*w)
    grad = (-1 / m) * np.dot(x.T,(y_mat -
        prob)) + lam*w
    return loss,grad

def oneHotIt(Y):
    m = Y.shape[0]
    OHX = scipy.sparse.csr_matrix((np.ones(m
        ), (Y, np.array(range(m)))))
    OHX = np.array(OHX.todense()).T
    return OHX

def softmax(z):
    z -= np.max(z)
    sm = (np.exp(z).T / np.sum(np.exp(z),
        axis=1)).T
    return sm

def getProbsAndPreds(someX):
    probs = softmax(np.dot(someX,w))
    preds = np.argmax(probs,axis=1)
    return probs,preds

def getAccuracy(someX,someY):
    prob,prede = getProbsAndPreds(someX)
    accuracy = sum(prede == someY)/(float(
        len(someY)))
    return accuracy
```

The main loop is shown below.

```
w = np.random.randn(x_train.shape[1],len(np.
    unique(y_train)))
lam = 0
iterations = 200
learningRate = 0.1
for i in tqdm(range(iterations)):
    loss,grad = getLoss(w,x_train,y_train,
        lam)
    w = w - (learningRate * grad)
print(loss)
```

B.3 MNIST

The data pre-processing for the MNIST data used the below code:

```
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

Y_train = np_utils.to_categorical(y_train,
    10)
Y_test = np_utils.to_categorical(y_test, 10)

# Compute learning rate
m = X_train.shape[0]
k = 10
L = (k - 1) / (k * m) * np.linalg.norm(
    X_train)
print(1/L)
```

The model itself was implemented using Keras:

```
model = Sequential()
model.add(Dense(10, input_shape=(784,),
          kernel_initializer='zeros',
          bias_initializer='zeros'))
model.add(Activation('softmax'))

sgd = SGD(lr=0.1)
model.compile(loss='categorical_crossentropy
    ',
              optimizer=sgd,
              metrics=['accuracy'])

history = model.fit(X_train,
                    Y_train,
                    epochs=500,
                    batch_size=128,
                    validation_data=(X_test,
                        Y_test),
                    verbose=1)
```