Insert Title Here

by

PARBHU Varun

Project Supervisor: Professor (Dr) BHURUTH MUDDUN OSK

Project submitted to the Department of Mathematics,
Faculty of Science, University of Mauritius,
as a partial fulfillment of the requirement
for the degree of
M.Sc. Mathematical and Scientific Computing (Part time)

December 2022

Contents

List of Figures

List of Tables

Acknowledgement

Placeholder

Abstract

Placeholder

Terms and Definitions

α	Learning Rate
η	Reduced Learning Rate
$\mathbf{I}$	Identity Matrix

Placeholder

Placeholder

Chapter 1

Introduction

- Discussion around Cryptocurrency and frequency of changes
- Discussion around Twitter and the API (amount of data)
- Discussion around the advancement Compute processing speed and Deep Learning algorithms

Chapter 2

Optimization

2.1 Introduction

The optimization problem is a computational problem in which the objective is to find the best of all possible solutions. Deep neural networks is a form of an optimization problem to find the best possible set of weights in order to reduce the error in a network.

A generic form of an optimization problem is given by

$$\begin{aligned} \underset{x}{\text{minimize/maximize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_j(x) = 0, \quad j = 1, \ldots, p \end{aligned} \tag{2.1}$$

where

$f : \mathbb{R}^n \to \mathbb{R}$ is the objective/loss function to be minimised,

$g_i(x) \leq 0$ are called inequality constraints,

$h_j(x) = 0$ are called equality constraints, and

$m \geq 0$ and $p \geq 0$ If $m = p = 0$, the problem is an unconstrained optimization problem.

Minimum/Maximum Point

Let x^* be points of the function $f(x)$ defined in (2.1) and $u \in \mathbb{N}(x^*, \delta)$ where $\mathbb{N}$ is a δ-neighbourhood of x^*. Then, x^* is said to be a

(i) local minimum if $f(x^*) < f(u)\ \forall u \in \mathbb{N}$

(ii) local maximum if $f(x^*) > f(u)\ \forall u \in \mathbb{N}$

(iii) global minimum if $f(x^*) < f(u)\ \forall u \in \mathbb{R}$

(iv) global maximum if $f(x^*) > f(u)\ \forall u \in \mathbb{R}$

Figure 2.1: !!!TO BE CHANGED!!! Min max

Determining Minimum/Maximum Point

For a function $f(\mathbf{x})$ where $\mathbf{x} = (x_1, x_2 \ldots, x_n) \in \mathbb{R}^n$, the condition for the presence of a stationary point (minimum or maximum point) is given by

$$\mathbf{G} = \nabla f \left(\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \ldots, \frac{\partial f}{\partial x_n} \right) = \mathbf{0} \tag{2.2}$$

The second derivative of $f(x)$ is given by the Hessian matrix, $\mathbf{H}$

$$\mathbf{H} = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1 \partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n \partial x_1} & \dfrac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{2.3}$$

The nature of the stationary points of $f(x)$ can be determined by studying the positive definiteness of (2.3) using its eigenvalues. If all the eigenvalues of $\mathbf{H}$ are positive, then $\mathbf{H}$ is symmetric positive definite, indicating the presence of a **local minimum**.

2.2 Optimization algorithms

The solutions to the optimization problem are vital in modern machine learning and artificial intelligence algorithms, which includes weight optimization in deep learning. There are a number of popular optimization algorithm currently developed to solve the problem. Hence, choosing the right algorithm can be challenging as well.

Network Error

Let the network error function, e, be given by the squared difference between the predicted value and the true value. The difference is squared to avoid the sum of errors of multiple input vector to be zero which can mislead the network to have perfect predictive power.

$$e = \frac{1}{2} \sum_{i}^{\text{Total Input}} (\widehat{y_i} - y_i)^2 \tag{2.4}$$

The goal of the network is to adjust the weight to reduce the network error as much as possible. The idea of reducing a function to a value is synonymous to (2.1) an optimization problem. The network error in (2.4) is a quadratic equation in this case.

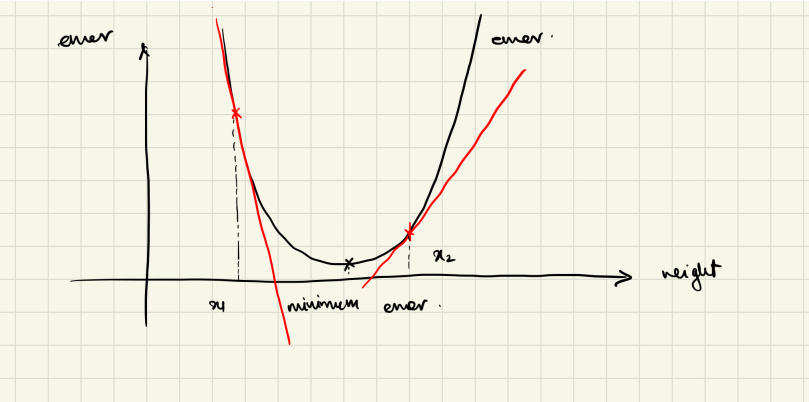

Figure 2.2: Network Error

Figure (2.2) is a graphical representation of the network error function. Our goal is to reach the minimum of the function by adjusting the weight value.

2.2.1 Gradient Descent

In order to reach the bottom of the function; we need to adjust the weight such that the derivative of the error function is 0. The approach of updating the weight based on the gradient of the error function is known as **gradient descent**. The derivative with respect to the weights of the network error (2.4) for a single input is given by

$$\frac{\partial e}{\partial w_{ij}} = (\widehat{y}_i - y_i)\frac{\partial \widehat{y}_i}{\partial w_{ij}} \tag{2.5}$$

From figure (2.2); we can observe that if the gradient is negative then we have underestimated the predicted value and need to increase the weight to reach the optimal value. On the other hand, if the gradient is positive then we have overestimated the predicted value and need to decrease the weight to reach the optimal value. The equation (2.5) provides us with the opposite direction and amount to adjust the weight. Hence, the update rule of the weights for gradient descent method is given by

$$w_{ij}^{t+1} = w_{ij}^{t} - \frac{de}{dw_{ij}} \tag{2.6}$$

In the gradient descent method, the network learns from the gradient of the error function and adjust the weights accordingly to reduce the error. However, the gradient alone can be quite large, causing oscillations as we go down the error function. This problem by introducing a **learning rate** α prior to adjusting the weight.

Learning Rate

The learning rate is typically denoted by the Greek letter alpha α. It helps the network to control the rate at which the weights are changing. Having a system with a high learning rate may lead to an oscillating network when trying to find the optimal weight and having a slow learning rate increases the number of iterations required when optimizing the network. The adjusted update rule of the weight is given by

$$w_{ij}^{t+1} = w_{ij}^{t} - \alpha \frac{\partial e}{\partial w_{ij}} \tag{2.7}$$

where $\alpha \in (0, 1)$

2.2.2 Stochastic Gradient Descent

Instead of adjusting the weight to the average loss function, we can approximate the gradient by only one random directional derivative of the error. Thus, the number of computation in the gradient descent method can be significantly reduced by a factor of n (where n is the number of direction). The stochastic gradient method (SGM) is given by

$$w^{t+1} = w^{t} - \alpha_t (\nabla e)_i \tag{2.8}$$

where α_t is the learning rate at the t^{th} iteration which may vary with iterations.
As we move closer to the bottom of the minimum of the function; the solution starts to oscillate since we are moving the weight in random directions. Thus, the learning rate should be reduced gradually as we iterate. Some commonly used reduction of

learning rate is given by

$$\eta(t) = \eta_i \quad \text{if } t_i \leq t \leq t_{i+1} \qquad \text{piecewise constant} \tag{2.9}$$

$$\eta(t) = \eta_0 e^{-\lambda t} \qquad \text{exponential decay} \tag{2.10}$$

$$\eta(t) = \eta_0 (\beta t + 1)^{-\alpha} \qquad \text{polynomial decay} \tag{2.11}$$

where λ, β, and α are known as hyperparameter.

Training dataset can be very large in certain cases which leads to a greater cost of compute at each iteration for the gradient descent, so stochastic gradient descent is preferred in these cases.

2.2.3 Mini-Batch Stochastic Gradient Descent

In order to make use of the features of both gradient descent and stochastic gradient descent; we introduce the mini-batch stochastic gradient descent. In this approach, instead of iterating in the direction of the full gradient or only in one direction of the full gradient, we split the dataset into small batches and compute the gradient of each batch. The formula of the stochastic gradient descent is given by

$$\begin{aligned} \mathbf{w}^{t+1} &= \mathbf{w}^t - \eta_t \mathbf{g}_t \\ \mathbf{g}_t &= \nabla_{\mathcal{B}_t} e(\mathbf{x}, \mathbf{w}) \end{aligned} \tag{2.12}$$

where $\mathcal{B}_t$ is a mini-batch of elements drawn uniformly at random from the training set (the expectation of the gradient remains unchanged).

Since we are using a mini-batch, the updates are closer to the full gradient but at a reduced cost.

2.2.4 Newton's Method

For minimizing $f(x)$, $x \in \mathbb{R}$, we need to solve $g(x) = e^{'}(x) = 0$. Newton's iteration is given by

$$x_{n+1} = x_n - \frac{g(xn)}{g^{'}(x_n)} \tag{2.13}$$

$$= x_n - \frac{f^{'}(xn)}{f^{''}(x_n)} \tag{2.14}$$

For multivariate functions we need to minimise $f(\mathbf{x})$ over $\mathbf{x} \in \mathbb{R}^n$, that it

$$\underset{\mathbf{x}\in\mathbb{R}^n}{min} f(\mathbf{x}), \quad \mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n \tag{2.15}$$

The Newton's iteration for multivariate function is given by

$$\mathbf{x_{n+1}} = \mathbf{x_n} - H(\mathbf{x_n})^{-1} \nabla f(\mathbf{x_n}) \tag{2.16}$$

where

$H(\mathbf{x_n})$ is the Hessian matrix of $f(\mathbf{x})$

We can observe from (2.3) that calculating the inverse of the Hessian matrix can be computationally very expensive for higher dimensions. Replacing $H(\mathbf{x_n})^{-1}$ by $\alpha\mathcal{I}$ where $\mathcal{I}$ is the identity matrix; we get the **method of steepest descent** given by

$$\mathbf{x_{n+1}} = \mathbf{x_n} - \alpha \mathbf{I} \nabla f(\mathbf{x_n}) \tag{2.17}$$

where $\alpha \in (0, 1)$

2.2.5 Momentum

Exponentially Weighted Moving Average

The exponentially weighed moving average (EWMA), also known as exponential moving average (EMA), of a series of data points S_t is given by

$$\begin{aligned} \mu_t &= \beta\mu_{t-1} + (1-\beta)S_t \\ \mu_0 &= c \end{aligned} \tag{2.18}$$

where $c \in \mathbb{R}$ and $\beta \in (0, 1)$ is known as the smoothing constant. β represents the weightage that is going to be assigned to the past values. The average number of previous reading is approximately given by $n = (1-\beta)^{-1}$. The higher the value of β the greater the number of points we average over.

SGD with Momentum

From the SGD, we observed that the weight update can be very noisy. In order to smoothen the search direction in the SGD, we implement an exponentially moving average on the gradient. The SGC with momentum can be written in the form

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta\boldsymbol{\mu}^t \tag{2.19}$$

where the exponential moving average of the gradient is given by

$$\begin{aligned} \boldsymbol{\mu}^t &= \beta\boldsymbol{\mu}^{t-1} + (1-\beta)\nabla(e)^t \\ \boldsymbol{\mu}_0 &= \mathbf{c} \end{aligned}$$

2.2.6 RMSProp

The Root Mean Square Propagation (RMSProp) algorithm alleviates undesirable fluctuations when adjusting the weight during optimization. RMSProp is an adaptive learning rate. The idea is to reduce the step size at very large gradient to avoid fluctuations and increase the step size at smaller gradient to move steadily in the

correct direction of the optimal solution; hence an adaptive learning rate at each iteration. The equations for the RMSProp is given by

$$\begin{aligned} S_i^t &= \gamma S_i^{t-1} + (1-\gamma)(\nabla e^t)_i^2 \\ w_i^{t+1} &= w_i^t - \frac{\eta}{\sqrt{S_i^t + \epsilon}}(\nabla e^t)_i \end{aligned} \tag{2.20}$$

where $\epsilon \approx 10^{-6}$ to ensure that we avoid dividing by zero during iterations.

2.2.7 Adam

The Adam algorithm (Adaptive Moment Estimation algorithm) is a robust update rule for the weight optimization. The algorithm combines the benefits of momentum (2.19) and RMSProp (2.20). Adam is the most popular generalized algorithm performs very well in many cases. It is considered as a state-of-the-art algorithm for deep neural network optimization. The set of equations for the Adam algorithm is given by

$$\begin{aligned} \mu_i^{(t)} &= \beta_1 \mu_i^{(t-1)} + (1-\beta_1)\nabla(e^{(t)})_i && \text{Momentum} \\ S_i^{(t)} &= \beta_2 S_i^{(t-1)} + (1-\beta_2)(\nabla e^{(t)})_i^2 && \text{RMSProp} \end{aligned}$$

To prevent S^t and μ^t from becoming zero during the initial steps, a bias correction is introduced, such that

$$\hat{\mu^{(t)}} = \frac{\mu}{1-\beta_1^t} \qquad\qquad \hat{S^{(t)}} = \frac{S^t}{1-\beta_2^t}$$

Finally, the update rule is given by

$$w_i^{t+1} = w_i^t - \frac{\eta}{\sqrt{S_i^t + \epsilon}}\mu_i \tag{2.21}$$

Bibliography

AuthorName (Year*a*), Conferencetitle, *in* 'BookTitle'.

AuthorName (Year*b*), 'Title of article', *Journal* .

BookAuthors (Year), *BookTitle*, Publisher.

- McCullough - First Neural Network Paper 1943 - F. Rosenblatt - 1958 - 1962